Activity without intersubjectivity: A case study of side-by-side collaborative problem solving

Nick V. Flor & Seana Coulson

Collaborative problem solving is an activity which appears to require a high degree of intersubjective knowledge between participants about the sequence and content of activities prior to engaging in them. Without such intersubjectivity, collaborators would duplicate efforts and have to constantly explain their actions to one another. Conversational analysis research states that intersubjective knowledge is a precondition to engaging in activities which are content dependent. However, we show that this is not always the case. We present an analysis of two programmers working side-by-side in which intersubjective knowledge about their individual activities occurred during and not prior to activity engagement. This was possible because the media properties when working side-by-side allowed the collaborators to construct a shared working context in which they could notice opportunities to coordinate content-dependent activities. Moreover, individual-related grounding activities are the catalysts for constructing these contexts. We argue that the ability to easily shift from a personal to a shared workspace is a fundamental requirement for joint activity, as such shifts provide a media-rich context for more effective grounding and coordination of detailed activities. We end by discussing the implications of this research for theories of collaborative problem solving and for designers of remote collaboration tools. This research contributes to our understanding of the intricacies of human collaborative problem solving.

Keywords: Close-proximity collaboration, media properties, distributed cognition, media spaces

1. Introduction

A key concept in conversation is common ground: mutual knowledge, mutual beliefs, and mutual assumptions between two or more conversants (Clark & Schaefer, 1989). Without common ground every conversational detail would need to be made explicit. For example, instead of using indexicals like “it” or “that,” speakers would have to substitute the actual referents. But common ground is more than just a conversational concept. Clark & Brennan (1991) argue that common ground is necessary for effective coordination of all joint activities. Without common ground participants would be unable to coordinate the sequence and content of activities:

“Alan and Barbara, on the piano, must come to play the same Mozart duet. This is coordination of content. They must also synchronize their entrances and exits, coordi-
nate how loudly to play forte and pianissimo, and otherwise adjust to each other’s tempo and dynamics. This is coordination of process. They cannot even begin to coordinate on content without assuming a vast amount of shared information or common ground.” (p. 127)

Implicit in the definition of common ground is the notion of intersubjectivity. Not only must collaborators have common ground to work effectively, the knowledge elements which comprise the common ground have to be intersubjectively shared. A piece of knowledge, K, is intersubjectively shared between a group of collaborators if (c.f., D’Andrade, 1987):

1. Everybody knows K.
2. Everybody knows that everyone else knows K, and
3. Everybody knows that everyone knows that everyone knows K.

To understand the necessity of the third kind of “knowing” consider a person standing outside a hotel waiting for a taxi cab. As a taxi-cab approaches, the person raises their hand and the taxi-cab stops. What kind of knowledge allows the person to raise their hand and stop the taxi-cab. First, the person needs to know that in his or her culture, raising one’s hand in the face of an approaching taxi-cab is a signal for the driver to stop and pick up that person. But the person will only raise their hand if he or she knows that the taxi-cab driver also knows that the raised hand is a pick up signal (second kind of knowing). Finally, the person has to know that the taxi-cab driver knows that the person knows the hand signal (third kind of knowing), since the driver may (e.g.) assume the person is a foreigner and is raising their hand for a different reason. Thus, the third level of knowing is necessary for words, signals, and actions to be properly understood by all collaborators.

If any of these conditions are not met, then a process known as “grounding” (Clark & Schaefer, 1987) occurs to construct the intersubjective knowledge (hereinafter, we use the term intersubjective knowledge and common ground as synonymous). Clearly, intersubjective knowledge, and mechanisms for constructing such knowledge are important during conversation. However, while much is known about specific grounding techniques during conversational activity, much less is known about grounding during more specific kinds of artifact-mediated, collaborative problem solving activities like software development. In this paper, we analyze an activity in which two programmers engage in separate activities which — based on the content and sequence in which the activities occur — appears to have involved a high degree of prior intersubjective knowledge. However, a detailed analysis of the representational activity prior to the collaborators initiating their separate activities reveals no such grounding. We show that coordinated activity in terms of content and process can result serendipitously if: (a) subjects have intersubjective knowledge of the problem; (b) collaborators can communicate verbally; and (c) the work environment allows shifting from a personal to a shared workspace. The properties of the working environment allows the grounding to occur during the activities, and not prior to
as Clark and Brennan (1991) suggest. In effect, the working environment changes the distribution and kinds of grounding needed. We end by discussing the implications of this research for theories of collaborative problem solving and for designers of remote collaboration tools.

2. Background and analysis

The data analyzed below is from a different section of a problem solving situation described previously by Flor & Hutchins (1991). Briefly, two professional programmers — RC (male) and LM (female) — were brought into our laboratory and given the task of adding a “whisper” command to an existing multiplayer game. The whisper command required two pieces of information: the name of the player to send a message to, and the message itself. Using the whisper command, a player could send private messages to other players. The existing game had a “talk” command, but it broadcast messages to everyone playing the game. The instructions merely stated the whisper function along with a high level description of how the program worked. The programmers were free to implement the whisper command however they wanted, using whatever division of labor they desired. The programmers worked side by side on separate, but networked computers.

The program was written in the C-programming language and used two libraries the programmers were unfamiliar with: (a) the “curses” package for displaying character graphics; and (b) BSD sockets for networked, interprocess communication. The programmers took approximately 3 hours to add this function, which consisted of a total of twenty-three file modifications.

The method used to study my data is an observational one, guided by the theoretical framework of distributed cognition (Hutchins, 1995). Under this framework, the researcher studies a single or small number of work group situations and characterizes the movement and content of the observable representational state. This characterization is then used to help explore more specific issues. For example, in this particular study, the specific issues are whether and how two people can jointly solve a problem without intersubjective agreement on the specifics of what needs to be done.

There are several notational conventions in the analysis. Numbers in parentheses correspond to line numbers in a transcript (which is usually contained as a figure). To help determine who is speaking, the programmers’ initials sometimes precede these line numbers, e.g., LM7. Transcripts are depicted as four column tables. The columns are: (a) a reference index; (b) speaker; (c) discourse; and (d) analyst comments. Bracketed letters in the discourse column, e.g., [a], index particular comments. The comments include descriptions of what the programmers are looking at and relevant information displayed on their computer screens. In the discourse column, the pipe-symbol, |, indicates the starting point for the next utterance in the table.

2.1 Analysis: The bug discovery, an opportunity for intersubjectivity

Our analysis begins after the programmers
had already made several changes to the program. During these changes the programmers communicated discoveries made while implementing their changes and asked each other questions when they had problems with their implementations. For these changes the programmers verbally negotiated divisions of labor, agreeing on what files to edit and who would edit those files. The same kinds of verbal agreements did not occur for the situation that follows.

The programmers believed those changes successfully implemented the “whisper” command, and ran the program to test them. The program returned several error messages (refer to Figure 1):

What the program should have returned were the following messages (refer to Figure 2):

The program’s errant behavior provides a context for grounding activity, as it disagrees with their shared understanding of how the program ought to work.

2.2 Analysis: The discussion following the error messages
Indeed, the programmers entered into a discussion about the error messages (see Table 3). The discussion can be viewed as a kind of grounding activity. Interestingly, although the programmers arrive at a joint understanding (common ground) about several things the program should not do, they do not subsequently discuss what actions to take given this understanding. Yet, as the next section shows, they somehow manage to edit the correct files and make the correct changes despite the lack of joint agreement about what to do next. The grounding framework indicates that this is not possible; the programmers should not be able to coordinate on what files to edit (process) and make the correct changes (content) without prior common ground. Before the details of how they coordinated their actions without intersubjectivity, we look at the discussion in more detail.

2.2.1 Detailed analysis of the discussion about the error message
Their discussion can be divided into three sections. The first section consists of lines 1-3. In this section, RC discussed what he believed was wrong with the error message
### Table 1

RC discusses what he believes is wrong with the error message.

<table>
<thead>
<tr>
<th></th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[a] Well that's farther along. That's what I just put in. (i) But it shouldn't say 'hello world = hello world'</td>
</tr>
<tr>
<td>2</td>
<td>LM whadda you mean? (…) okay [a]</td>
</tr>
<tr>
<td>3</td>
<td>RC &quot;that* it shouldn't say</td>
</tr>
</tbody>
</table>

\[a] = LM is already looking at RC's display. Displayed is the error message (see Figure 1)\]

### Table 2

LM discusses what she believes is wrong with the error messages.

<table>
<thead>
<tr>
<th></th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>LM whisper test [a] (…) [b] why does it say [c] it shouldn't say it should only say and it shouldn't say guest says. It should say</td>
</tr>
<tr>
<td>5</td>
<td>RC Yeah, it should say guest well [i] it should change to guest whispers? [a]</td>
</tr>
<tr>
<td>6</td>
<td>LM Oh you didn't change that yet (…) you didn't save you didn't change it did you? [a]</td>
</tr>
<tr>
<td>7</td>
<td>RC Did I change what?</td>
</tr>
<tr>
<td>8</td>
<td>LM Remember how you copied? You [RC: yeah] yanked all those lines. I'll bet [RC: yeah] you didn't change it to whispers</td>
</tr>
</tbody>
</table>

\[a] = types west <end> hello \]
\[b] = the computer returns the second of the error messages (see Figure 1) \]
\[c] = RC looks at LM's computer screen \]

(Table 1, line 1). Specifically, he points out that the left hand side of the equals sign is wrong (“...it shouldn't say 'hello world = hello world'”). The left hand side should contain “guest” instead of “hello world,” i.e., ‘guest = hello world.’

Note that as RC spoke, LM looked at his screen. Her looking is a kind of grounding mechanism. RC’s screen provides a shared context for understanding indexicals in his utterances such as “that’s” and “it” (1). Thus, while within the activity of grounding the error message’s meaning, there are smaller grounding activities occurring such as LM turning to look at RC’s display.

In the second section, lines 4-8 (see Table 2), LM described what she thought was wrong with the error message (4: “...it shouldn’t say guest says”). What she points out is a different problem from RC’s. Instead of “guest says” the program should have returned “guest whispers.” LM also points out a reason why the problem exists (8): RC forgot to change says to whispers when he “yanked all those lines.”

Similar to LM looking at RC’s screen while he spoke, RC looked at LM’s screen when she spoke (4c). After he agreed with her prognosis, he looked back (5).

In the third section, lines 9-11 (see Table 3), the programmers discuss yet another problem: why a “W” was in front of the messages. RC begins by stating this misunderstanding (9). LM correctly identifies the “W” as a parsing error (10), and they both quit out of the game and work on their respective computers.

At the end of this discussion, neither programmer had a well-defined task, although both were aware of several potential problems with the code. A reasonable next step is to jointly plan out a course of action and negotiate a division of labor. Examining just the computer transcript of the files modified following this discussion suggests that this was exactly what happened.
Table 3
Discussion the “W” in the error message.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>RC</td>
</tr>
<tr>
<td>10</td>
<td>LM</td>
</tr>
<tr>
<td>11</td>
<td>RC</td>
</tr>
<tr>
<td>12</td>
<td>LM</td>
</tr>
<tr>
<td>13</td>
<td>RC</td>
</tr>
<tr>
<td>14</td>
<td>LM</td>
</tr>
</tbody>
</table>

Table 4
A summary of the programmers’ changes to their files following the bug. RC created a new subroutine in his file and LM called that subroutine in her file. The dependencies between the two changes and the sequence of activities suggests prior intersubjective agreement. However, a detailed analysis of the representational activity shows otherwise.

<table>
<thead>
<tr>
<th>RC’s CHANGES: display.c</th>
<th>LM’s CHANGES: command.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>display_whisper( )</td>
<td>case ‘w’:</td>
</tr>
<tr>
<td>{</td>
<td>display_whisper( )</td>
</tr>
<tr>
<td>...</td>
<td>break;</td>
</tr>
</tbody>
</table>

that the programmers had an intersubjective understanding of what to do prior to editing their files.

Yet no discussion what to do, or who should do it took place. The next sections examine how the programmers were able to coordinate their activities without prior intersubjective agreement. Examining a richer account of their representational activity — one that includes not just the computer transcript but also their discourse, glances, and gestures — shows how separate but related work activity can get integrated as a serendipitous side-effect of media properties and the ability of the workers to easily move from one work context to another.

2.3 Aside: The computer transcripts and the appearance of intersubjective activity

Examining just the computer transcript following the above discussion shows both coordination of process and content. Both programmers edited different files at approximately the same time; RC edited display.c and LM edited command.c. Furthermore, the changes made were dependent on one another: RC created a subroutine, display_whisper, in his file, which was called by code LM added in her file (see Table 4). These were precisely the files they had to edit and the changes they needed to make to fix the error messages. The high degree of coordination in both process and content suggests

1) The computer transcript is a record of everything the programmers typed into their computers and everything displayed on their computer screens.

2.4 Analysis: How the programmers coordinated what files to modify (process coordination)

If the programmers did not discuss which files to edit, how did they come to edit their respective files? The answer to this question can be found in the details of LM and RC’s
representational activity with their computers which followed the discussion. Both programmers were lead to their files by different individual search strategies for the cause of the error. Their search strategies could have lead them to edit different files containing irrelevant subroutines, but serendipitiously they ended up editing separate, but correct files, i.e., process coordination.

Just prior to RC editing his file some discussion takes place, but it is not clear whether there is intersubjective agreement about what file he is modifying and the nature of his modification. Prior to LM’s edit no discussion takes place and, thus, there is no intersubjective agreement about what she is going to do. Before examining how this occurred, LM and RC’s searches are analyzed in detail.

### 2.4.1 RC’s search strategy

RC’s search strategy was more conceptually-driven than LM’s. Instead of searching for a fragment of the error message, RC did two searches. A search for the string “talk” followed by a search for the string “whisper” (see Table 5, RC [85] and RC [86]). Earlier research on this particular dyad (Flor & Hutchins, 1991) showed that they followed a copy and modify strategy for implementing the whisper command. All the code related to “talk” — an existing command — was copied and modified to conform to the whisper command’s requirements. Thus, RC’s first search (“talk”) listed those files containing code that they needed to copied and modified, namely display.c and io_handler.c. His second search (“whisper”) listed those files that they already had copied and modified. The second search only returned io_handler.c, which meant they still had to copy and modify code in display.c. The verbal transcript interleaved with the commands typed into their computer support this interpretation (Table 5, lines 1–4).

RC’s discourse following his discovery can be interpreted as an attempt to create a shared understanding of what to do next. Note, however, that he does not specifically say which file, display.c or io_handler.c, needs to be modified. RC also does not specify the nature of the modification. Thus, at best we can assert that there is intersubjective agreement about what file RC is going to modify, but not about the nature of the modification. Instead of agreeing with him, LM simply turns back to her screen and does her own search.

### 2.4.2 LM’s search strategy

LM’s decision about which file to edit was driven by the error message itself. After the discussion she did a search for a fragment of the error message, i.e., a search for
"ERR" (see Table 6, LM[40]). The search returned a list of several files containing the fragment, "ERR" (see Table 6, LM[40], last column). She selected the file that contained code which actually used the fragment, command.c, over the file that simply defined the fragment, client.c. LM’s search describes a backwards-chaining strategy for determining the cause of errant program behavior. Under this strategy, programmers trace the flow of code backwards, starting by editing the code that generated the error and tracing the subroutine calling sequence backwards until they arrive at the area of code responsible for the error. Because the calling subroutines can be in different files, the programmers may edit several files before they arrive at a probable cause (Flor, 1994).

Note that unlike RC, LM does not know inform RC about the file she is editing. Thus, there is no intersubjective agreement about what file LM is modifying and the nature of her modification.

### 2.5 Analysis: How the programmers coordinated their modifications (content coordination)

Despite working individually, both programmers took advantage of working side by side; they frequently asked the other programmer for help related to their own tasks. For example, during the course of editing her file, LM asked RC several questions (6, 7, 9) which was followed by RC looking at her screen (6, 8, 10) (refer to Table 7):

At the time RC looked over, LM’s screen contained a line of code which called the subroutine display.talk(). Just before looking over, RC had copied the display.talk() subroutine and was in the midst of modifying it into display.whisper(). Thus, by chance LM was editing the file where a call to display.whisper() needed to be made and, more importantly, her screen displayed the area of code where this call needed to be added, i.e., just below the call to display.talk().

However, because LM did not know that RC was creating the display.whisper() subroutine, she was incorrectly adding code to
Table 8

<table>
<thead>
<tr>
<th></th>
<th>RC</th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>[looks back at LM’s screen] Is that where I am? Where Where are you? [!] You’re in command.c</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>In here.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Yeah. In server.</td>
<td>LM types: case ‘w’ between the case ‘t’ and case ‘y’</td>
</tr>
<tr>
<td>14</td>
<td>Wait wait wait no no no(??)</td>
<td>This is said in response to LM placing a case ‘W’ in such a way that it would call display_talk()</td>
</tr>
</tbody>
</table>

Table 9

<table>
<thead>
<tr>
<th></th>
<th>RC</th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>You don’t want to put it in there cuz it’s using talk and yell. See talk and yell do the same thing. [*]</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>We only wanted to play [*] it display it to a certain player’s screen [RC looks at LM]. But let’s just try to do [LM glances at RC’s face] it to anybody</td>
<td>After saying this, RC goes back to scrolling his file. RC looks at LM type some stuff, then goes back to his screen [*]=LM puts the case ‘w’ over the case ‘t’ and case ‘y’</td>
</tr>
<tr>
<td>20</td>
<td>I know how to do that though [RC looks back at his screen]. But let’s just try to do a certain player first.</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>So can I do display talk? [RC looks] Then whatever it is</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>No do a display [RC looks back at his screen] I’m writing a display whisper right now so do display whisper [*]</td>
<td>[*]=LM types display_whisper()</td>
</tr>
</tbody>
</table>

call display_talk() (line 13). Fortunately, however, by asking RC for help understanding the code in her file, she provided a context for RC to notice the opportunity to coordinate their activities. Indeed, he notices and then stops her from incorrectly adding the code (see Table 8, line 14).

RC described a problem with her implementation (18). He then told her to make a call to display_whisper() because he was writing the subroutine display_whisper(); LM complied (22):

Finally, we get the kind of intersubjective agreement that the grounding framework said needed to be done prior to coordinating on process and content. After summarizing the analysis, the implications of these findings will be discussed.

2.6 Analysis summary

To recap, after discussing the bug, both programmers started editing two different files at about the same time. The computer transcript of their activity shows that LM added a call to the subroutine display_whisper() in her file, while RC added the implementation of display_whisper() in his file.

Moreover, the computer record indicates that they both quit out of their files at roughly the same time. The fact that the programmers implemented two different, code-dependent, pieces of code combined with the close starting and finishing times of their file edits, strongly suggests intersubjective agreement on what to do (process coordination) and how to do it (content coordination). However, going beyond the computer record and examining a richer context which
included what the programmers looked at, the contents of their screens, and their discourse revealed that there was no such agreement about what to do until after the programmers had already started their activities. They were able to coordinate activities because they had the ability to easily notice opportunities for coordinating their separate activities and to quickly react to those opportunities. These abilities — properties of working side-by-side — are what gave the appearance of prior intersubjectivity.

3. Properties of the side-by-side working environment which facilitate opportunistic noticing and coordination

What are the properties of the working environment which facilitate opportunistically noticing situations for integrating dependent activities? In this section we show that such properties are more properly viewed as media properties. In particular, certain properties of the computer screens (an information media) allowed the programmers to look at their coworker’s screen and notice opportunities to coordinate their separate activities. Moreover, the ability to easily talk to a coworker while engaged in a private activity — a property of the verbal medium when two people work side-by-side — provided the catalyst for the coworker switching to look at the speaker’s screen. Once switched the ability to overlap talk with looking resulted in very specific content coordination. The following sections examine these media properties in more detail.

3.1 Properties of the screen: accessibility, containment, and persistence

Notice how RC determined that LM was working on the file where the call to his subroutine needed to be added. First, he could see that LM’s display contained a subroutine call to \texttt{display\_talk()}. From this line of code he could infer that she was editing the file containing display routine calls. And, since he was implementing a display routine — \texttt{display\_whisper()} — he could conclude that LM was editing the file where a call to his function had to be placed.

But RC could only see LM’s display because his proximity and orientation made her screen visible or, more generally, accessible, with a glance. Accessibility is, thus, one important property of computer screens that can lead to opportunistic noticing when programmers work side-by-side. However, accessibility was not the only screen property contributing to RC’s discovery. Recall that LM was in the process of typing a case statement, “case w:” when RC made his discovery. Her focus of attention was not on the code surrounding the case statement, but on typing the statement itself. For RC, however, it was not LM’s case statement but rather the surrounding code that allowed him to notice the opportunity to integrate their work. Thus, RC’s discovery was facilitated by the screen containing more information than what LM was currently interested in, a property we label media \textit{containment}. Finally, not only did LM’s screen contain more code than what she was currently interested in, it displayed the code long enough for RC to notice it. So, yet
another important screen property is persistence.

These screen properties are so ordinary that it is easy to take them for granted during side-by-side collaboration studies. But it is not always the case that these properties hold, particularly during remote collaboration. The other programmer’s display is not necessarily accessible at a glance, if at all. One can easily imagine a remote collaboration tool in which the designer tries to maximize the screen space available to a local worker by making the remote collaborator’s screen available via a menu selection or with a key-stroke combination. These kinds of mechanisms prevent the local worker from easily constructing the context needed to notice opportunities in a remote collaborator’s workspace.

3.2 The overlapping properties of the verbal medium

Clearly the ability to talk to one another and specify divisions of labor and to direct the content of joint activities. Indeed, RC took advantage of this ability when he told LM to add a call to _display-whisper_. The use of audio as a medium for both coordinating work and grounding is well established in collaboration research, thus we will not dwell on such instances in my data.

However, in addition to these more volitional uses of audio, the ability to easily talk facilitates both the unsolicited asking for help when questions arise (9), and also the announcing of discoveries made during one’s private work activities. In fact, it was LM’s question (9) that initially made RC glance at her screen and thereby notice the opportunity to integrate their separate activities. Thus, talk serves as a catalyst for noticing conditions for integration. The key ability exploited, is the ability for one person to talk while the coworker is looking either at his or her own screen or the speaker’s screen. This ability is possible because when working side-by-side the screens and audio can overlap.

The overlap property allows the programmers to access and propagate information in parallel and across different media (see Figure 3). In turn, this ability permits the programmers to create a variety of different kinds of collaborative work situations. For example, one programmer can tell a coworker what to type, watching that co-workers’s screen to coordinate how much and when to give more information (Flor & Hutchins, 1991; Flor, 1995). Alternatively, a programmer can simultaneously tell the other programmer what to type while they look at their own screen, e.g., RC20.

In general, the four media properties discussed — accessibility, containment, persis-
tence, and overlap — permit a collaborator to shift from his or her own workspace to a coworker’s workspace, creating a shared context for noticing opportunities to integrate dependent but non-intersubjective activities. Media properties also provide the catalyst for shifting workspaces and for coordinating content of joint activities. Designers of remote collaboration tools need to understand how these and other media properties facilitate shifting between workspaces, and designing these media properties into their tools. Although, more controlled studies are needed to determine the extent of these media properties on the quality of collaborative solution. The strength of these observations lie in providing a detailed example of how these properties can affect the solution. Not only does this kind of information help designers of remote collaboration technology, it also provides instances that theories of collaboration need to account for.

4. Discussion: Implications for grounding and cognitive theories of collaborative problem solving

The case analysis showed that intersubjective agreement about content-dependent, individual problem-solving activity does not have to be attained prior to the individuals engaging in that activity. Coordination of content can opportunistically occur during the activities if the working environment provides a context for noticing dependent solution activities. These findings have implications for grounding theories and, more generally, theories of collaborative problem solving.

Although the situation analyzed seems at odds with the grounding framework suggested by Clark & Brennan (1991), we do not see it as invalidating the framework. Clearly, grounding activity did occur for different aspects of the situation. For example, the programmers grounded the meaning of the error messages and looked at their coworker’s screen to ground indexicals in coworker utterances; moreover, grounding process and content did occur, just not prior to the programmers engaging in their separate activities. Thus, to be more applicable for studying collaboration, the grounding framework ought to be extended to explain: (a) the kinds of grounding that do occur in collaborative situations; (b) why certain activities are not grounded; and (c) the mechanisms by which non-intersubjective, albeit content-dependent, activities are grounded.

We offer the following explanation. During individual problem solving activity, the cost of maintaining intersubjective agreement is too high. In searching for a solution, collaborators engage in heuristic activity that may be difficult to justify and may not even lead to a solution. Thus, although collaborators may ground an understanding of the problem, they are not likely to ground details of their problem solving activity until they reach a point where their grounding effort is acceptable. In the situation studied, one point of acceptable effort was when both programmers were looking at LM’s screen (i.e., had a shared workspace) and RC’s implementation was dependent on LM adding code to an area displayed on her screen. In addition
to grounding an understanding of the problem, collaborators seem willing to ground indexicals in their coworkers’ utterances; part of the grounding mechanism includes turning to look at the other programmer’s display. In turn, this kind of grounding activity is a consequence of the coworkers trying to get help with their own activities, viz., coworkers trying to ground structures encountered in their individual workspaces such as LM asking RC “what are these?” (Table 7, line 6). The key ability for grounding is the ability to shift from working privately to working jointly, viz., the ability to shift from a personal to a shared workspace. Human-computer interactions researchers have noted the importance of such shifts (Ishii, Kobayashi, & Grudin, 1992; Dourish & Bellotti, 1992, Heath, Jirotka, Luff, & Hindmarsh, 1995), but have not made explicit their functional role with respect to grounding and opportunistic problem solving. The properties of the media in the working environment affects the ease with which this shifting can occur, thereby affecting when grounding occurs. Thus, not only does the medium change the kinds of grounding that take place (Clark & Brennan, 1991), *the medium changes the distribution of grounding activities*. This finding has very important implications for theories of collaborative problem solving.

Cognitive theories of collaborative problem solving have only recently started to formulate (Okada & Simon, 1995). This research has focused on describing the differences between individual and group problem solving in terms of traversing and trimming problem spaces. A key finding of this research is that explanatory activities affect the quality of the solution. The more explanatory activity between collaborators, the better the chance of arriving at a correct solution. Our observations lead us to predict that by constructing the working environment so that the collaborators also can easily shift from working individually to jointly the number of explanatory episodes will increase. Again, the ease with which shifts from personal to private workspaces occurs is a function of the media properties in the working environment. Thus, to fully explain human collaborative problem solving activities, such theories need to make explicit the role of media properties in addition to the cognitive properties of the individuals.

5. Summary

In summary, collaborators must make tradeoffs between problem solving activity and maintaining intersubjective knowledge of those activities. While performing personal problem solving activities a collaborator may ask for help, and if his or her coworker obliges this has the side-effect of grounding both process and content (who is doing what and how). In turn, if the collaborators happen to be working on dependent activities — very likely if they are working on solving the same general problem — then this side-grounding activity provides a context for integrating their work. These situations assume the ability to shift from working privately to working jointly (a shift from a personal to a shared workspace). The relative ease with which these shifts can occur is determined by the properties of the work-
ing environment. Working side by side, the collaborators can move from a personal to a private workspace with a mere turn of the head. However, this may not be true in all cases such as during remote collaboration. Thus, designers of remote collaboration tools need to understand the media properties in the working environment which permit such shifts in order to properly support them.

References


Nick V. Flor

Nick V. Flor is currently an assistant professor at Carnegie Mellon University’s Graduate School of Industrial Administration. He received his doctorate in Cognitive Science from the University of California, San Diego. His research interests are in the areas of group improvisation, cognitive theories of the firm, and remote collaboration. His theories are built up from observational studies and are tested by building technologies based on those theories.

Seana Coulson

Seana Coulson is a graduate student in the Cognitive Science department at the University of California, San Diego. She specializes in the neurological bases of language, semantics, and pragmatics of language processing.