Knowledge Structures Involved in Comprehending Computer Documentation

Darlene Clement

Institute of Human Learning
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

A model of computer-manual comprehension is proposed in which four processes operate simultaneously: task-mapping of the structure of regular procedures onto the structure of computer commands, constructing a mental model of the computer system, inducing the command language grammar, and learning the structure of computer procedures. Findings from a study of five novices' comprehension problems with UNIX documentation are analyzed in terms of these four processes. Two of the four processes—task-mapping and procedure learning—are described in this paper. The analysis focuses on the knowledge structures involved in comprehending a technical text.

The solution to the problem of computer manual comprehension has been narrowly viewed as simply a matter of eliminating jargon, using "good" sentence structure, and so on. That is, the emphasis has been on low-level linguistic aspects instead of the fundamental cognitive ones involving the knowledge structures tapped by a technical text. The comprehension problem may be considered from the perspective of each of the three factors that give rise to it, viz., the system, the reader, and the writer.

First, a computer manual gives directions for operating a device that is unlike any other machine. Computer systems are difficult to learn both because they are operated symbolically, and because they "operate on invisible objects with consequences that are not readily apparent" (Nakatani and Rohrlich, 1983). In particular, the reader must understand both the system's conceptual model and interface.

Second, the reader's knowledge base must be taken into account. If the reader is a non-programmer there are only two things that he or she can bring to the text: a model of how conventional editing is done, and some expectations about text structure. Beyond this, non-programmers are at a loss. They have no understanding of computer programming which some have argued is the basis for a generative model of the system (e.g., Sheil, 1981).

Third, the technical writer's ability to convey the new information must be analyzed. The writer's role is one in which he or she must compensate for both the novice's naiveté, and any unnaturalness in the system's design. To date, writers have only been given very general suggestions for accomplishing this feat (e.g., "write clearly"). The gap between the suggestions and their implementation must be filled entirely by intuition. Cognitive psychology, and in particular text comprehension research, can provide a means of bridging this gap by elucidating the specific schemata tapped by a technical text.

In order to investigate users' problems learning a system with a manual, an in-depth qualitative study was carried out in which 5 novices attempted to learn UNIX[1] with only a manual to guide them. The subjects were asked to read a section of two locally produced tutorials in advance of meeting with the researcher. The tutorials covered file manipulation and text editing with a line-oriented editor called "Edit." During the meetings subjects used the computer to follow the instructions in the tutorial. The 5

[1] UNIX is a trademark of Bell Laboratories.

sessions lasted two hours on average and were tape recorded, yielding approximately 10 hours of tape from each subject.

The model derived from the analysis of the data partitions the information contained in computer manuals into four classes, each with a corresponding comprehension task. **Functional** information describes the purpose of each command and triggers a **task-mapping** comprehension process. In this process, users map the new functional information given in the manual onto their tacit models of regular text-editing and general office procedures. Examples of such preexisting models are the familiar procedures of cutting and pasting text in documents, creating new files, and typewriter editing. **Structural** information describes the underlying structures and processes of the computer system itself. A description of a device triggers a **model-building** process in which the reader attempts to construct a mental model of how the device functions, for example, how the editor buffer and disk (important entities which the user never sees) are related. **Command** information describes the way in which commands are issued. It triggers the **command learning** process in which users attempt to learn the syntax and semantics of the command language. **Procedural** information provides directions for navigating through the system, i.e., knowing which command to issue in which context. The corresponding **procedure learning** process entails recognizing these different program contexts, and learning the order in which commands must be issued.

Though each of these processes taps radically different knowledge structures, they are fundamentally the same: in each case it is necessary for the new information to connect with the reader's knowledge base. It is apparent from the problems novices had that the documentation they used had serious shortcomings in each of these areas. Because of space limitations, only two of the four comprehension processes will be described here—the task-mapping process and the procedure-learning process.

TASK-MAPPING

The task-mapping process was initially described in Clement (1983) as a global process of mapping the structure of the regular editing task onto the corresponding computer version of the task. So, for example, the regular editing procedure of changing a word by crossing it out and writing another word above it, gets mapped to the text editor's substitute command. Recently, this same process has been described in more detail by Moran (1983). Moran states that the user's knowledge of editing procedures consists of at least eight editing functions (add, remove, change, transpose, move, copy, split, join) which operate on five text entities (character, word, sentence, line, paragraph). The 37 tasks that result from the combination of editing functions and text entities constitute the core knowledge the user possesses. This knowledge comprises the "external task space." The computer system also has entities and operations defined within it, but these may be very different from the ones the user knows. The entities and operations internal to the computer constitute the "internal task space." Moran gives the example of a system that defines only one entity (a character string), and only three editing operations. With this system users must learn to conflate the five separate text entities they are familiar with onto this one system entity, and the eight editing functions they are used to must be collapsed onto three: cut, paste, and insert. In other words, the task-mapping process requires that the user learn to carry out familiar tasks by means of unfamiliar functions which operate on unfamiliar entities.

The operation of this process was especially evident when subjects attempted to learn the UNIX **read** command. This command allows a file to be inserted into the file currently being revised, that is, it allows the user to cut and paste. How is it similar to conventional cutting and pasting? In both the computer version and the regular version of the cutting and pasting procedure the point at which the new information is to be inserted must be located. Then the pasting action can be carried out. In the computer procedure the "read" command is issued; in the regular procedure the material is actually pasted in. There are two ways in which the procedures differ. First, in regular cutting and pasting the material pasted in typically no longer exists in its original location. In contrast, in the computer version of the task, the file pasted in still exists as a separate file. Second, in regular cutting and pasting usually only *part* of a remote document is spliced into the document under revision. In contrast, in computer cutting and pasting the *entire* remote file is pasted in, not just a section of it.

The manual described the command as follows.

Reading additional files (r)

The **read (r)** command allows you to add the contents of a file to the buffer at a specified location, essentially copying new lines between two existing lines. To use it, specify the line after which the new text will be placed, the **read (r)** command, and then the name of the file. If you have a file named "example", the command

:$r example
"example" 18 lines, 473 characters

reads the file "example" and adds it to the buffer after the last line. The current filename is not changed by the read command. (*Edit: A Tutorial* p. 22)

In general, the subjects had difficulty understanding this paragraph. After they were told that it referred to cutting and pasting further discussion revealed their attempts to map the structure of the regular procedure onto the computer procedure. Two subjects thought that the file pasted in disappears from its original location. Notice that this is what would be predicted from a model of regular editing. Another subject wondered if only part of the remote file is pasted in, or the whole file.

From a text comprehension standpoint this paragraph from the manual is reminiscent of passages used in text comprehension studies in the early 70's (e.g., Dooling and Lachman, 1972) where subjects were presented with texts that were incomprehensible without a title. Once a title was provided the texts were easily comprehended because *the title triggered the schema that the text was about.* Similarly, this text would have been easier for the subjects to assimilate had the cutting and pasting schema been activated at the outset, say, in the heading. This is a point that can be of use to document developers. Once the appropriate regular-editing schema is activated then the task-mapping process can be carried out more easily. The document developer can further facilitate the task-mapping process by explicitly comparing the similarities and differences between the regular editing procedure and the computer procedure. This would reduce the amount of inferencing the reader would have to engage in, and would simultaneously answer the reader's questions.

PROCEDURE LEARNING

According to the Card, Moran, and Newell (1980) model of the manuscript editing task, an expert's knowledge structure consists of goals, operators, methods, and selection rules. That is, experts have pre-stored information about the sequence of operations and alternative methods available for performing an editing task. It is this knowledge that the novice must acquire from the manual and from interactions with the system.

It is clear from the data that novices come to the procedure learning task with a rudimentary procedure schema containing slots for goals, steps, and methods. However, the process of filling these slots is not easy to do if the role of each piece of information is not clearly marked in the text. For example, one section of the tutorial described how to correct typographical errors with a line-oriented editor. To carry out this task the user must understand three things: 1) how the editor functions (the need to position it on the relevant line); 2) the sequence of steps necessary for carrying out the task; and 3) the various methods that can be used to carry out the task. The structure of the task is as follows:

Goal: Correct typographical error in text.

Step 1: Position editor on relevant line.

Method 1: Search for pattern on relevant line.

Method 2: Type number of relevant line.

Step 2: Issue substitute command.

After performing only the first step in the procedure, two subjects assumed that the task had been completed, i.e., that the correction had been made. This indicates that the manual did not make clear the two-step nature of the task. One subject read about the two methods for carrying out a step and assumed that each method was a necessary part of the sequence. This indicates that the various methods were not clearly marked in the manual as alternatives. After reading the two pages describing the procedure, one subject, after much thought, managed to induce the two-step structure of the task. Together these examples show how much inferencing the subjects were forced to do, and

how difficult it was for them.

This analysis describes novices' attempts to induce the structure of a particular editing task. Yet we know from the research previously described that the expert's knowledge is a finely articulated goal structure in which the steps and methods are clearly differentiated. The writer could facilitate the construction of this structure by simply making it explicit. If the goals, steps, and methods of the procedure were explicitly marked in the text, then the novice would be able to assimilate each piece of information, as it is read, to the appropriate slot in the schema. Like the suggestion put forth in the task-mapping section, this suggestion also reduces the amount of inferencing the reader would have to do.

Research on the schemata novices bring to the text, as well as the schemata ultimately constructed by the expert can lend more precision to the task of *packaging* information in a technical text. In particular, this kind of analysis gives rise to psychologically-based heuristics for document development which address the important conceptual aspects of comprehending computer documentation.

REFERENCES

Card, S. K., Moran, T. P., & Newell, A. Computer text-editing: An information-processing analysis of a routine cognitive skill. *Cognitive Psychology,* 1980 *12,* 32-74.

Clement, D. Comprehending Computer Documentation, unpublished manuscript, (March, 1983).

Dooling, D. J., & Lachman, R. Effects of comprehension on retention of prose. *Journal of Experimental Psychology,* 1971, *88,* 216-222.

Edit: A Tutorial. Documentation produced by the U.C. Berkeley Computer Center.

Moran, T. P. Getting into a System: External-Internal Task Mapping Analysis. *Proceedings of the Conference on Human-Computer Interaction.* Boston, MA., 1983.

Nakatani L. H., & Rohrlich, J. A. Soft Machines: A Philosophy of User-Computer Interface Design. *Proceedings of the Conference on Human-Computer Interaction.* Boston, MA., 1983.

Sheil, B. A. Coping with complexity, (Tech. Rep. CIS-15). Xerox Palo Alto Research Center, 1981.