One way of conceptualizing many of the human factors issues in interactive computing is as issues in communication about computers. Presented are a framework for this conceptualization and a review of research addressed to several levels of the communication process. Communication as an ill-structured design process is analyzed and contrasted with a process of algorithmic encoding and decoding. The design framework is then applied to examinations of how people name and refer to entities, how people understand and express relations (quantifiers and other predicates) between entities, how more complex communications (business letters) are created, and how preprinted forms reflect previous knowledge.

Human factors in communication

by J. C. Thomas and J. M. Carroll

Human factors is now and may be expected to continue as one of the key elements of success in the data processing industry.[1] This is particularly true in end-user application areas. In this paper we sketch a prototheory of communication processes, based largely on our earlier work on communication and other types of design. We have come to believe that communication (between people and systems and between people via systems) stands at center stage among human factors issues. Scott and Simmons[2] use a technique known as Delphi to canvass programming managers about the most important factors influencing programmer productivity. The opinions of these managers focus on communication. Walston and Felix,[3] in a more quantitative approach, use multiple regression to predict programming productivity in terms of lines of code. They also show that communication variables are most

important among the general set of variables. For example, they show that the nature of the interface with the customer is about three times as important as whether structured programming is used. These are all matters of human factors in communications.

Human factors problems of communication are often signaled by two groups of people voicing "Why-can't-they . . .?" question pairs. The designers of a computer system may ask: "Why can't they (the users) understand a simple computer system?" This question has its corresponding opposite question by an end user: "Why can't they (the designers) explain the system without using all that jargon?" Another example pair is: "Why can't they be more formal in stating what they want the program to do?" and "Why didn't they tell me the program would do this? This isn't what I want." Another example is: "Why can't the computer tell me what to do next?" and "DHZ301J." A message that one cannot understand is not communication.

Because we find such pairs of questions far less likely to lead to a solution than "How-can-we?" questions, we focus in this paper on the following thoughts. How can we use what we already know about communication to improve (not perfect or determine) our computer systems from the standpoint of human factors? How can one explain a computer system to another? How can someone who wants a computer application program explain the application to a data processing expert? How should one design the communication process between person and machine? How can the computer system phrase error messages and menu selections so that users may understand them?

We limit the scope of our discussion by focusing on software issues, and by concentrating on reviewing our own work (though we believe that the weight of related research only strengthens our analysis). For a more general overview of relevant human factors issues, the reader is referred to Meister,[4] Miller and Thomas,[5] or Shneiderman.[6] Our emphasis in this paper on general principles is in no way meant to serve as a substitute for application-specific studies, some excellent examples of which are found elsewhere in this issue. Rather, principles are set forth to help focus development and testing efforts on reasonable alternatives. General principles cannot specify what to do, but they can aid us in doing design work better. Thus our paper is not a human factors case study; rather it expresses some human factors principles derived from psychology.

Communication as design

A common view of communication is roughly the following. A sender has an idea and encodes it into symbols that are then

transmitted to a receiver. The receiver decodes these symbols into an *internal code* (or idea), and communication is good to the extent that there is an isomorphism between the internal idea states of both sender and receiver.

We have, however, found the following view more adequate in providing a reasonable basis for understanding human factors issues in communication. There is a sender or *designer* who wants to communicate for some purpose. The sender designs a message that he or she believes will result in a desired effect when translated by an *interpreter*. The rationale for this view is more fully expressed in References 7 and 8. Although the original rationale for viewing communication as design was based purely on an analysis of communication, we believe that viewing communication in this way has an unanticipated pragmatic benefit for the system designer in that user-computer design considerations may be viewed in the same general framework as other design considerations. Human factors need not be added to the system. The communication purposes of the system and its documentation are integral and crucial to the designer's overall creative effort.

design process

Under this view, a fuller understanding of person-computer interaction depends on an understanding of how one designs communications. But what is the design process? Our initial studies of the design process consisted of having real designers and real clients discuss such a real design problem as that of attempting to redesign output devices for use in a research library. We videotaped these interactions and later analyzed their dialogues.[9,10] (For further case studies see References 11–13.) From these studies we deduced that the design process is cyclic.

design is cyclic

A design problem is neither solved at once nor decomposed into subproblems that are solved independently. Rather, problem solving begins by studying one part of the problem (the first cycle). Then the solution or partial solution of that part may result in another way of partitioning the problem or of defining the problem. Subproblems evolve dynamically, depending on the design process. The cyclic nature of design in general also seems true for the special case of designing an adequate communication.

design cycles progress through phases

Within each cycle, we find some regularity in the dialogues in that each problem progresses through a regular sequence of phases. The client states the problem, and then the client and designer together elaborate the problem. The designer offers a (partial) solution. Then the client and designer together elaborate the outlined solution. Next, the designer and especially the client mentally test the elaborated solution against the concrete realities of the situation. Finally, the solution for that portion of the problem is either accepted or rejected. If it is accepted, a new

cycle begins. If the solution is rejected, the client and designer may return to an earlier phase. We discuss later in this paper how in letter-writing, the designing of a communication also seems to progress through these phases.

goals are often implicit

The client's goal statements are typically focused on current symptoms of current difficulties. In fact, many of the requirements are implicit. These may already be met by a current system and thus may not be explicitly thought about until the designer proposes a solution that does not meet those requirements (or, worse, delivers a system that fails to meet those unstated requirements).

It is only through interaction with the designer that many of the important but unstated goals of the client are brought forth. Rather than ask the client to attempt to state all goals precisely and explicitly at the very beginning of the design process, it is probably wiser to concentrate on attempts to optimize the client-designer interactions to ensure that all the client's unstated requirements are made explicit. Our work in letter writing, dialogues, and naming confirms this principle for the case of communication.

goal structure aids can accelerate the convergence of design cycles

To achieve further experimental control of the design process, we have simulated the client part of the dialogue process by giving subjects requirement statements. In one experiment,[14] participants were given a description of a complex library procedure and told to design a schedule for these library procedures to maximize efficiency. When higher-level goal information was explicitly structured, this was reflected in more convergent problem solving activity. In this case, the trajectory of successive design cycles ballistically converged on the final design solution. Ultimately, this goal orientation resulted in better-structured schedules for the library procedures.

the designer selects a metaphor to represent the problem

In further studies of design,[15] we found that the particular metaphorical device used to explain the nature of a design problem affected the goodness of solution as well as the people's understanding of the problem. We presented participants with logically identical problems couched in either a spatial or temporal metaphor. A spatial-temporal problem pair might be the design of an office layout for a group of people who share various personal and functional relations versus the design of a manufacturing process involving a set of steps that share various priority and sequencing relations. There were two key findings. Persons in the spatial problem condition who spontaneously generated graphic representations of the problem produced more successful solutions in shorter times. When persons in the temporal problem condition were provided with a graphic representation, solution time and performance differences were reduced. The application

of this principle to providing effective communication is explored later in this paper under the heading of existent knowledge and communicative exchange.

metaphors differ in their behavioral utility

We have been using two separate senses of the concept of representation. The first is that of the cognitive representation of the problem information that the subject uses to deal with the design requirements. We call this the *metaphor* that is invoked by the problem.[16] The second sense of representation is the graphic format used to work with the problem information, and is a consequence of the metaphor. Thus, some metaphors suggest ways of thinking about a problem that are more useful behaviorally than others, as for example the spatial problem version.

communication levels

There are several levels at which communication is examined in this paper. Perhaps the most primitive act of communication is to refer to a single entity. Studies of this process, called *naming*, are reviewed first. Next most primitive are the ways that people communicate about such simple relationships as *quantifier relationships* and *conditional expressions*. We next review studies of more complex communication structures such as complex data processing procedures and business letters. We then present empirical data concerning *metacomments*—communication about communication. And last we address the effects of the *social context* in which communication takes place.

Our premise is that at each communicative level, communication is a design activity. We find that the generalizations about design previously stated are valid for the special case of designing communications. When viewed as design, communication is richer and more complex than when viewed as transmission. We have come to believe that the more adequate view of communication as design entrains more adequate person-machine communication facilities. An explanation of that extension of our communication-as-design model is the main theme of this paper.

Naming: how people name and refer to objects

Naming and reference are at the most basic level of language, and so that is the place we begin a consideration of communication. At the very least we need things to talk about, and yet the way we refer to the things we talk about is not simple at all. Naming is a genuine design problem. This is especially true in relatively complicated naming domains like naming computer files, system commands, new products, and program variables.

Despite the pervasiveness and importance of naming, there has been almost no experimental study of it to date. We have conducted a series of studies to begin to characterize naming with

the objective of suggesting more behaviorally effective naming strategies or naming aids. For convenience, we divide naming into consideration of its inputs, the process of generation, and the usability of names—although there is considerable overlap.

inputs to naming

There appear to be the following four principal inputs to the creation of a name: the defining properties of the referent, the functional context, the goals of the namer, and the needs of the expected users. As it turns out, these inputs are listed in order of the amount we know about each one, although probably in reverse order of their importance. Indeed, extensive discussion in the philosophy of language from the time of Frege to the present has focused upon the suggestion (due to Frege) that names have meaning as well as reference. A typical example is the co-reference of two names like *Evening Star* and *Morning Star*. Both names refer to the planet Venus, but the names are not the same. Thus, reasoned Frege, both names must have distinct meanings.

The meanings, following the Frege school of thought, are conceived of as propositions or structured sets of propositions such as the following:

MORNING STAR = (X: (SECOND PLANET) AND (VISIBLE IN THE MORNING))
EVENING STAR = (X: (SECOND PLANET) AND (VISIBLE IN THE EVENING))

Names stand for, or abbreviate, their descriptions. A similar type of situation exists when entities are referred to by verbal names and number names. For example, a serial number relates a given typewriter to others, whereas the character string EXECUTIVE refers to members of a typewriter model class. Another example is when the values of variables are referred to indirectly or directly in the flow of program control. Our knowledge grows through the resolution of such paradoxes and confusion in meaning.

A series of experimental studies required participants to name a series of things. These studies revealed that a variety of entities such as geometric designs, people's everyday roles (such as their occupations), computer files, and system commands all conform to this framework. Descriptive material that logically underlies the intended referent is collapsed into a compact designator, i.e., a name, as discussed in References 17–22 and illustrated by the following examples:

ADVANCE = COMMANDS THE ROBOT TO MOVE FORWARD OR ADVANCE ONE STEP
ARM DOWN = COMMANDS THE ROBOT TO LOWER ITS ARM, TO MOVE IT DOWN

Since people spontaneously refer to properties of referents in the names they create, we might presume that this would be a desirable property of name schemes that are designed for people to use. A traditional justification for not having descriptive names in many application domains, say data bases, is that a large number of unique designators is needed, and people have assumed that descriptive natural language names could not provide the required level of differentiation. O'Dierno[23] has shown that this is false for an extensive inventory data base, and all of our own research also suggests that descriptive names are not only adequate but preferable.

We have found that the *functional context* of the naming situation is a strong determinant of the specific descriptive material that becomes the basis for a name. Thus, in the context of a general-purpose computation package, a routine that takes two arguments and produces their product might be called the *multiplication routine*. The same routine, however, in a payroll application package might be called the *gross pay routine* (the two arguments being wage rate and hours worked). People who were asked to create names for novel referents spontaneously incorporated elements of the context into their names.[19] This suggests that when names are designed for people to use, important aspects about the likely contexts of use should be built into the names.

The goals people have when they create and use names are also important input to naming. Carroll[24] and Olson[25] have examined a goal termed *minimal distinguishing* in which names are frequently designed to distinguish their referents from other similar entities. Thus, in minimal distinguishing, only enough information to make the distinction is designed into the name. If a program has only one output routine, it might be called OUTPUT. But if there are three output routines, say for binary, octal, and decimal numbers, a different naming strategy is required just to distinguish the routines at all. To the extent that minimal distinguishing is a relevant goal, the word "output" might be omitted altogether, leaving the names "binary," "octal," "decimal," with "output" understood. Indeed, Carroll and Olson observed this type of naming pattern.

A real example of this can be found in the naming scheme for data types in PL/I and FORTRAN. In FORTRAN there are two data types, integer and real. In PL/I these labels fail to distinguish all data types. Therefore, there are four types in PL/I: fixed binary, floating binary, fixed decimal, and floating decimal.

That other people use a name and the nature of such other persons comprise a fourth design consideration. For example, it is known that a novel compound noun, like system communication, is difficult to comprehend unequivocally. (Is it a type of

communication, a communication from a system, a communication addressed to a system, or a communication about a system?) Subjects who are asked to create names to be used by others tend to create far fewer names based on such neologisms than do subjects who are asked to create names in nonsocial situations.[18,21]

We have only indicated the kinds of considerations that interact as inputs to naming. Notice that differing goals, recipients, and contexts might interact in very complex ways to suggest behaviorally adequate names.

the process of naming

The process of name generation is frequently a progression from descriptive epithets to proper names. The descriptive phrase "digital computing machine" (in the 1940s) becomes the briefer descriptive name "digital computer" (in the 1950s), and today simply "computer." Several studies of this process have suggested a range of specific shortening strategies for name creation.[18,26]

Structural redundancies are built into the form of a name to make the content elements of the name (the referent's description, the context, goals, and the expected recipients of the name) more obvious. However, name patterns extend to other names by relating patterns of structural redundancy to patterns of referent similarity. The naming strategies are called *rule schemes*.[21,27,28]

Examples of rule schemes that involve the explicit repetition of character substrings come from a study of file naming.[22] In this study, it was found that over ninety percent of the files whose contents could be recalled had file names organized into literal rule schemes. In the example of this that follows, the term on the left is the file name, and the term on the right is the file type, a field probably originally intended by the language designers to differentiate among executable modules, source programs, etc.:

```
CONTENT PHOTO
CONTENT DRAWING
CONTENT TERMTEXT
```

Here all file names are identical; files are differentiated only by file type. The following less trivial example involves indexing within the file name:

NLBODY	NLCOUPON
NLCVRSHT	NLDOC5
NL11.10	NL12.2
NL12.4	NL12.5

The repetition of a common character substring (NL) in these eight file names creates a partial rule basis for the creation of new file names as well as a structured aid for recalling file names and file contents.

A smaller percentage of file names were organized into rule schemes without literal repetitions. The chief nonliteral scheme is the *person's name scheme*, that is, text files, often letters and memos from, to, for, about, etc., some person Smith. The rule scheme predicts the file name SMITH. A nonliteral rule scheme called *congruence* has been studied in the context of designing command paradigms.[29] Congruent command paradigms explicitly represent the semantic oppositions in the definitions of the commands to which they refer. The following two commands comprise a congruent rule scheme;

RELEASE = COMMANDS THE ROBOT TO RELEASE OR UNHOOK AN OBJECT BY OPENING ITS CLAW
TAKE = COMMANDS THE ROBOT TO TAKE OR GRAB ONTO AN OBJECT BY CLOSING ITS CLAW

The semantic opposition of the command descriptions is paralleled in the opposition of command words selected to name these commands. Deciding on one command name largely determines the other, although not by virtue of an explicit repetition of character substrings, but rather because of a nonliteral relation between the names.

As a counter-example, the following two command names are noncongruent:

UNHOOK = COMMANDS THE ROBOT TO RELEASE OR UNHOOK AN OBJECT BY OPENING ITS CLAW
GRAB = COMMANDS THE ROBOT TO TAKE OR GRAB ONTO AN OBJECT BY CLOSING ITS CLAW

People who were asked to design a command language for a simulated robot spontaneously used congruence with remarkable consistency.

More abstract rule schemes have been examined primarily in the task context of designing system command paradigms for the robot system. Several rule schemes for structuring command languages have been explored, among them hierarchicalness and hierarchical consistency. Hierarchical command languages have multiple structural elements that are combined in fixed ways. The following is an example of a pair of hierarchical commands for the robot system: CHANGE ARM OPEN and CHANGE ARM CLOSE. These commands are hierarchical in the sense that CHANGE defines a large class; ARM specifies which part of the robot is to be changed, and OPEN and CLOSE specify the exact action. A corresponding example of nonhierarchical commands might be RELEASE and TAKE. Both pairs define congruences, although the properties of congruence and hierarchicalness can be combined orthogonally.[29]

Hierarchical consistency is the property of maintaining a given level of hierarchicalness throughout a command language. Thus, in a hierarchically consistent language, if hierarchical commands with three elements appear in some commands, they appear analogously in all commands. Conversely, if any commands like RELEASE occur, then there are no hierarchical commands.

Hierarchicalness and hierarchical consistency do not necessarily involve literal repetition or the predictability of structural elements, given the occurrence of others; they are abstract conditions on the syntactic form of names. As was shown in the study reported by Carroll,[29] neither rule scheme was incorporated in the command paradigms generated by experimental participants. Thus two features found to make a language more usable are not spontaneously thought of during initial design, at least not by inexperienced designers.

Names that people create are not arbitrary with respect to other names they create, an observation that has significant implications for the design of names in other application areas. For example, it is misleading to have a text editing command U (=up) and another command D (=delete). People presume that the congruence implied by the U-D pair of commands is up-down, and otherwise are led to error (see next section). It is also important to note that the types of structural redundancy people spontaneously design into names they create can be theoretically analyzed into abstract cognitive principles like congruence. Thus there is neither need nor desirability to proceed on a strictly case-by-case basis in designing command languages. Obviously, this is not meant to deny the utility of testing languages, even when they are designed in a principled fashion.

usability of names

To do productive research on the usability of names, one must have a sufficient theory of what names can be like to design critical experimental comparisons. Accordingly, it is not surprising that the least amount of research we have completed to date concerns usability directly. The studies we have completed on the usability of names involve a robot system, in which a simple robot is ordered to change configuration and location and to perform simple tasks.

name schemes should be congruent

The most clear-cut finding on usability in this domain is that congruent command paradigms are rated as being better, are learned more quickly and more completely, and are used with greater success in solving problems. Thus rule schemes that people spontaneously build into command languages they design are powerful aids when those people are asked to learn and use command languages incorporating those rule schemes.

The results for hierarchicalness and hierarchical consistency are less comforting. People rate command languages as better and learn them more quickly when they are hierarchically consistent. The frequencies of certain error types are reduced when subjects are using hierarchical command languages. Thus the evidence suggests that these two abstract rule schemes can be used at least to some extent by people in learning and using command languages. However, recall that people tended not to design in these properties spontaneously when they were asked to generate command languages. There is a *prima facie* misfit in that what is useful for the user is not natural for the designer.

name schemes should generally be hierarchically consistent

As in the general case of design, a variety of sometimes competing requirements constitute naming. People who create names try to incorporate material descriptive of the referent and the functional context; they try to satisfy communicative and social goals. Clearly, some of these inputs can be suppressed in favor of others, and trade-offs can be codified in rule scheme strategies for generating names and for building in patterns of redundancy that can be recognized and used by communicative recipients. Recipients, conversely, seem to presume that patterns exist and that a rational and behaviorally efficacious basis for names can be identified and used. The simplistic view that names are arbitrary labels for things, in spite of its long and continuing tradition, is demonstrably wrong, and leads to bad communication systems.

Before leaving the topic of the usability of names the following rather obvious but frequently overlooked guideline to person-computer systems should be mentioned: When possible, use the terminology of the user, not the designer.

For example, many people without data processing backgrounds are more familiar with the terms "choice" or "variable" than they are with the word "parameter." Whereas people with technical background in education may use the word "module" to refer to a unit of learning, a general reader is probably more familiar with the term "lesson." Why require a student using a manual on a subject other than education to learn the term "module"? These examples could be multiplied, but the point is clear: As soon as possible, but certainly before making final decisions on manuals, prompts, and menus, at least one or two representatives of the intended user population should read through the materials and circle every word they do not understand. Then, whenever possible, words or phrases from the intended user's vocabulary should be substituted.

Relations: quantifiers, conditionals, and sentences

quantifiers

In addition to referring to things (objects, actions, and attributes), even the simplest programming, command, and editing languages

require the user to specify relationships of various kinds between objects, actions, and attributes. We now discuss studies on ways in which these relationships can be effectively communicated. The simplest relationships are quantifiers (all, some, none), connectives (and, or, not), and conditionals (if-then, if-and-only-if). These are now discussed in turn.

The following are the five basic set relationships that may obtain between the two sets A and B: (1) A is a proper subset of B; (2) A and B are identical sets; (3) B is a proper subset of A; (4) A and B are partially overlapping sets; and (5) A and B are disjoint sets. There are also more restrictive, more quantified relationships that may exist between two sets. For instance, every element of set A is the ancestor of an element in set B; or, if A and B are subsets of positive real numbers, we may define a relationship such that every element of A is the square of a corresponding element of set B. The basic set relationships, however, can be applied to any type of object and are, therefore, most fundamental in some sense. Nearly every query system uses some type of quantified relationship. For this reason, initial work in studying people's understanding of simple relationships has focused on quantifiers.

referring to quantified relationships

The first question we may ask is: How do people spontaneously refer to quantified relationships?[30] This question was investigated by having subjects without formal training in logic describe the relationship depicted in Venn diagrams after the form of representation was briefly explained. The study showed a wide variety of expressions used to describe each relationship, and some set relations were described more accurately than others. Disjoint relationships were always described unambiguously, whereas partially overlapping sets were described least accurately. Although subjects tended to give a fair proportion of ambiguous descriptions, not a single description was inconsistent with the Venn diagram pictured. These findings emphasize several points. The Venn diagram is a fairly good way of describing set relationships for nonprogrammers. People tend to err in not telling everything about a relationship rather than something untrue. These observations suggest several recommendations:

- Do not expect natural language descriptions of set relations to eliminate confusion.
- Allow people to communicate in terms of set equivalence and set disjunction when that is feasible and accurate.
- If a system requires the use of quantified relationships, give the user feedback and provide easy recovery from errors in such relationships.

interpreting quantified statements

Besides knowing how people express quantified statements in natural language, a second and related issue is that of how people

interpret quantified statements. Three experiments were performed that required people to interpret such statements.[30]

The interpretation tasks were as follows: (1) Having participants judge the equivalence of several related statements; (2) Presenting the participants with a series of English sentences and asking them to draw Venn diagrams to show all possible interpretations of the sentences; and (3) Presenting them with a relational data base and quantified questions, and having them manually find the answers in the data base.

Results were consistent across the three interpretation tasks. For example, in each task, the participants understood most easily and more or less perfectly the following set relation: No A are B. Notions of proper subset and equivalence were less well understood. The least understood of the simple two-set relationships was that A and B partially overlapped. Finally, statements admitting of several distinct possibilities were dealt with least well of all. For example, few subjects realized all the possible set relations consistent with the statement "Some A are B." These results are compatible with earlier published results,[31-34] and indicate quite clearly that English statements are not typically generated or interpreted in a strict rule-like fashion that relates them to actual set relationships.

A further quantifier experiment based explicitly on the communication-as-design model of communication studied the way that people use ambiguity in motivated situations.[7] In this experiment, pairs of participants communicated about quantified set relationships from either a cooperative or a competitive situation. In both situations, the interpreter of messages began with some (possibly incorrect) statement concerning the relationship between two sets, and then received a message from the message designer. The job of the interpreter was to draw a Venn diagram that illustrated the actual relationship between the two sets. The message designer had knowledge of the actual relationship between two sets and also had knowledge of what the interpreter already knew. The designer was always to send a relevant, true message about the two sets. In the cooperative case, it was to the designer's advantage to make sure that the message was also complete and unambiguous; and in the competitive case, it was to the designer's advantage to mislead the interpreter.

The main results of this experiment were that message designers, given the same actual relationship between two sets, constructed quite different messages depending upon whether they were in a cooperative or competitive situation, and upon what they knew to be the previous state of knowledge of the interpreter. Interpreters, for their part, interpreted identical messages differently depending upon whether they were in a cooperative or competitive situation, and upon their own previous state of knowledge.

Thus, the view that a statement refers to a quantified relationship was shown to be false. Consistent with and predictive of the results was the view that a designer's statement represented an attempt to change an interpreter's state of mind from a given state to a desired state. It was the given and the desired states that predicted the designer's message, not the true state of the world. In microcosm, this illustrates the futility of attempting to design a computer system without understanding the end-user.[35]

quantifiers in query languages

Query-by-Example[35] provides an easy-to-use interface for a relational data base. An early version of the language was tested with pencil and paper[36,37] and found generally to be quite easy to use. One source of difficulty for subjects in these experiments was that of translating questions stated in English into the query language syntax. The greater difficulty with quantifiers was apparently not due to the query syntax; rather, it was due to a misunderstanding of the English questions.

In a later phase of the experiment, subjects were each given a tabular data base and five problem situations. For each problem situation, subjects were asked to write an English question that could be translated into Query-by-Example and whose answer would help them solve the problem. In no case did any subject write an English question that involved an explicit universal quantifier (including subjects who made zero errors on quantifier syntax). These results suggest that although universal quantification is a very basic concept in thinking, people seldom use it explicitly in natural language.

quantifiers in natural language

To shed further light on the issue of the use of quantifiers in natural language, several observations are given here. In a pilot experiment, John Gould gave five students a relational data base and some problems. The students were to ask questions the answers to which were in the relational data base and might be helpful in solving the problem. Only seven of the 185 questions contained quantifiers. Two of these students also transcribed every question they heard during the course of one day. None of the 100 questions so recorded involved quantification in the logician's sense.

An examination of other dialogues collected by Thomas[30] and Carroll, Thomas, and Malhotra[10] reveals that the logician's use of quantifiers is rare or absent. Where a token "all" appears, it seldom literally refers to universal quantification. More accurately, it often seems to signal high emotions rather than universal quantification. On the other hand, some notion of "for all or nearly all normal cases" is quite often implicit in the dialogues.

A number of recommendations for averting some of the difficulties people typically encounter with quantifiers are listed in

Reference 30. One recommendation is to try to limit the user's task to one of choosing something that is consistent with the correct relationship rather than unambiguously specifying it. Another, in answer to a quasi-natural-language query, is to provide more data than are requested. For example, if a user asks for gross sales of GM and Ford for 1979 and 1980, provide a two-by-two table with the four annual gross sales figures and the subtotals and grand total rather than attempting to distinguish which of four possible questions is intended.

simple logical connectives

Simple propositions and quantified relations can be connected into more complex relationships by the use of logical *connectives*—AND, OR, and NOT. Yet there is evidence that people have problems of ambiguity even when using these simple relationships. Consider the ambiguities in the following statement: Print items that are red and in stock and not large or square or size seven.

OR relations typically give more difficulty than AND relations, but any logical connective seems to increase the psychological complexity considerably. The logical connectives AND and OR can be confusing because they do not map directly into the English words "and" and "or." Such a statement as "Put the blocks that are red and the green ones into the bin" may map into the programming-like command PUT BLOCKS (RED OR GREEN) INTO BIN. System users without data processing experience often attempt a word-by-word match between command language and problem statement that can result in incorrect statements.[36-38] A review of many of the difficulties with connectives can be found in References 39-41.

conditionals

Aside from quantified expressions and AND/OR/NOT, other simple logical relationships are the *conditional* (material implication) and the *biconditional*. Experiments by Miller and Becker[42] as well as the focus of much literature in computer science point to the importance of *transfer-of-control* statements in the writing and comprehension of programs. Several other studies have directly compared various ways of specifying transfer of control in order to determine which methods are least error prone.[6,43]

For certain classes of problems, a procedure table[43] is apparently the easiest method of specifying transfer of control. Other results[43] indicate that when subjects write sorting programs, *disjunctive* sets are slightly more difficult to use in transferring control, particularly when negatives are introduced into the specification.

Miller[43] points out that in natural language, people generally avoid conditional statements and prefer qualificational statements. Consider the statement, "If we have this item, ship it; if

we do not, produce a back order." People are more likely to say, "Ship the items we have, and back-order the rest." The preferred sequence, at least in English, seems to be ACTION followed by QUALIFIERS rather than CONDITIONAL followed by ACTION.

Thomas[44] studied dialogues about an invoicing system in which the subjects were to accomplish one of three tasks: (1) Specify a particular type of invoicing system; (2) Understand a particular invoicing system; or (3) Diagnose what was wrong with a particular invoicing system.

Two results of interest from these studies concern the use of conditionals and communication in a natural language system. Regarding conditionals, the dialogues give further support to the notions of Miller and Becker that people are spontaneously more likely to use qualificational rather than conditional statements. As in the case of quantifiers, a rather astonishing deficit of simple relationships on the part of people in laboratory studies seldom appears in dialogues because people also use nonverbal mechanisms to express themselves. It is, however, doubtful that the experimental participants had communicated with sufficient precision for a computer program to be written based on their understanding.

Regarding the attempts to build a computerized natural language dialogue system, the experiments reported show that people discussing a common topic use different syntactic and logical constructions depending upon their task. A computer system that took into account the user's task would be better able to parse and interpret the user's comments than one that relied solely on a general knowledge of the topic of conversation. The implication is that a system designer must have an appreciation for the user's application of the system; a general knowledge that the system is for bankers, for example, is inadequate.

sentences

In understanding or creating a sentence, one is dealing with a complex structure that can be analyzed at a number of different levels, e.g., syllables, words, phrases, and so on. Successful comprehension or production involves the development of an integrated description addressing each of these different levels almost in real time. The structure of the activity seems to be that as language input or output is partitioned into more manageable segments, or cycles, these units are processed more or less independently. Finally, they are integrated at higher levels of comprehension and behavior.

Several purely hypothetical structural views about the nature of sentence processing units have been advanced, a review of which is given in Reference 45. Typically, these views adopt some particular level of linguistic representation as the level of sen-

tence processing organization. Further research has shown, however, that the partitioning of sentences into behavioral units is a process of balancing competing goals. For example, it is well-known that human information processing is limited, as shown for example in Reference 46. Hence one goal of sentence processing is to design segments of optimal length and complexity to fully utilize but not overtax available processing capacity. Another goal, however, is accuracy. To the extent that the available processing units are logically complete and explicit, the communicative recipient is more successful in understanding the communication.

The consequence of this is a trading relation. Proper linguistic constituent boundaries only sometimes partition sentence strings into optimally long and complex segments. When considerations of length and complexity lead to a mismatch, the partitioning of the sentence is a compromise, possibly corresponding neither to the linguistic parsing nor to the optimal processing parsing. Various heuristics are used by communicators and recipients to design workable interchanges.[47-51]

To the extent that this model of communication processing at the sentence level is valid, we may develop some suggestions for designing communications that are easy to process. People understand sentences by analyzing linguistic sequences into discrete behavioral units. This processing may be facilitated therefore by organizing linguistic material so as to allow the ready identification and segmentation of such units. Consider the following examples in which nominalized clauses have explicit subject nouns:

After winning the poker hand, Harry decided to cash in his chips.
After Harry won the poker hand, he decided to cash in his chips.

The initial clause of the second sentence is more readily understood. Indeed, Daiute[51] found that one of the characteristics of poor writers is that they fail to reliably make sentential relations explicit for the reader.

Higher-level communication structures

We began by discussing how people design messages about things by naming. We then discussed communication about simple logical relationships (quantifiers, logical connectives, and conditionals), and the relationships expressed in sentences. The way in which relationships are expressed in sentences is governed by a complex but orderly set of syntactic rules that we call grammar. We now turn our attention to the ways in which people design and interpret messages at still higher levels of organization by metacomments, business letters, and metaphors.

metacomments

Although there are grammatical rules that constrain the ways in which people relate ideas within a sentence, it is also important for two communicators to understand how sentences are related in a higher-level organization. One method of achieving this kind of understanding is by using *metacomments*, that is, messages about the communication itself.

In three studies previously described, people variously attempted to describe, understand, or diagnose an order-handling and invoicing system by typing messages simulating a natural language computer system over an IBM 3270 Visual Display Terminal to another human being. The resulting dialogues were recorded and analyzed. It was clear that metacomments were crucial in effective communication.

In one example, the user and the system discussed discounts. Without being aware of it, however, the two people were referring to two quite different types of discounts. After that discussion was over and apparent agreement reached, the person simulating the system said, "O.K. Do you also want a discount applied to the invoice total?"

The user had thought that a discount applied to the invoice total was exactly what they had just been discussing. The system's question informed the person that they had not been talking about the same thing. Notice though that the superfluous words, "O.K." and "also" of the system were crucial in the user's perception of the miscommunication. The words "O.K." and ". . . also . . ." are clues that one topic had ended and that another was about to start. Without these clues, the question would simply have been, "Do you want a discount applied to the invoice total?"

The user could easily have thought that this was merely a summary of what they had been discussing, rather than the introduction of a new topic. The "O.K." and ". . . also . . ." were metacomments because they signaled the other person that a new topic was being brought up and, in this case, were crucial to the detection of a misunderstanding. It appears, based on this and other examples in the dialogues, that a "natural language" computer system that ignores metacomments is ineffective. In human natural language communication, comments made about the communication process are very important.

metaphor and metacomments

To help visualize the effects of metacomments, the following metaphor is sometimes useful. Communication is like two runners on either side of an opaque wall. They are running over rough terrain, which causes them to vary their speeds. The payoff is tied to their staying close together. In communication, staying close together is like communicating well. Obviously, the two

runners need to communicate with each other, such as by shouting or tapping on the wall. This is analogous to using metacomments.

How close together can the runners stay if they have no way to communicate with each other? They do fairly well until they come to some rocky terrain or until one of them falls. Without being able to communicate, they have no method of coordinating their positions. Similarly, in communication, without metacommentary, the two communicators may grow hopelessly divergent. This is why system designers must provide for metacommunication in an overall system design.[7]

types of metacomments

More specifically, we can ask what kinds of signals the two runners would like to be able to give each other. Presumably, the runners would like to communicate their speeds, direction, internal state, and something about the terrain they are experiencing. These four types of comments are precisely the kinds of metacomments people were observed to make in the dialogues. They communicated about the speed of conversation with terms like, "Hey, slow down," or "Wait a moment," or "Yes, yes, I understand. Go on."

People communicate about direction with comments that are *keepers* and *turners*. Turners might, for example, request that the conversation turn to something more specific, as in "Well, give me an example.", or more general, as in "Yes, but what about it in general?" Or there might be a request to move to an analogous case or move to another case of the same type. A keeper is more or less analogous to the communication in a draw poker game, "I'll stand pat."

People also attempt to describe or control their internal state, the corresponding state of the other, or the state of the dialogue itself. Consider the following examples: "I'm tired."; "I'm going to understand this if it kills me."; "I know you're tired, but let's try to keep going 'til midnight and then knock off."; or "We just don't seem to be on the same wavelength."

People also describe the conversational terrain they are traversing. A teacher may introduce a topic by saying, "Look, I know this material on the analysis of variance is difficult . . ."; "We always seem to argue when we discuss money."

In the interest of economy, error messages, prompts, menus, feedback messages, and descriptions in manuals are sometimes devoid of the metacommentary. This lack may be a false economy. Thus, the user may see a message appear on a screen such as: Display Device Number. Apart from the syntactic ambiguity, there is an absence of the appropriate metacomments that tell the

user how to take this message. Is it a prompt to enter the device number? Is the system about to display the device number? Has the user just input something that the computer has now classified as a device number? Has the user inadvertently chosen the menu item Display Device Number?

Thus we have arrived at the following admonition: Make clear to the user not only the content of the message, but also how it is to be taken. In other words—by spatial convention or otherwise—clarify whether a message is meant to inform of error, inform of state, prompt for action, or give feedback. The following are appropriate metacomments to clarify the output "Display Device Number": "Please key in the device number of your display and press ENTER."; "The device number of your display is as follows:"; "Is that the device number of your display? Please answer 'Yes' or 'No.' "; "You chose item 3. 'Display Device Number.' "

pre-existing formats for information

When the same kinds of information are communicated over and over, people often use special formats for presenting the information. The use of a pre-existing format serves much the same function as metacomments; it implicitly tells the interpreter of a message how to use the information. It saves the designer from having to specify over and over that information which is common to a series of messages. It also focuses the interpreter's attention on those portions of the information that are different. Checks, invoices, requisitions, and income tax forms are common examples of information presented within a fixed format. If these formats are well-designed with the users of the information in mind, they can be excellent means of communicating. To the extent that this knowledge can be made explicit and then embedded in a computer system that deals with these structures, the computer system becomes easier to use and more useful.

Perhaps the most common error in designing forms is that the designer uses terms whose referent is known to the designer but not to the person filling out the form. Thus, in a particular organization, the term "originator" may have an obvious referent to people in accounting who designed the form, but it may not be obvious at all to the various originators.

business letters as a structure

There are other classes of documents whose format is less fixed than a check or invoice but which definitely do have some higher-level form. These include reports, memos, and business letters. A particular example that has been studied in some depth is the business letter.[52,53]

There are at least two kinds of knowledge people have about business letters that can be used in a computer system. First, a business letter has a certain structure. For a given organization,

the formatting may be fairly standard. In such cases, the user ideally has to specify this information only once, and subsequent requests for business letters assume that formatting information, which may be overridden depending on the circumstances.

For example, the system may automatically enter the current date, which the user can override and begin with the next item, the address of the intended recipient. Again, if an address already exists in a previous letter to that recipient the system may automatically include that address, which again can be overridden by the user.

procedures for producing business letters

In addition to the structure of letters, most people have a more or less fixed procedure for producing a business letter. A word-processing system can incorporate that procedure but allow other options. For instance, good communication typically begins with an assessment of the audience. A system can prompt the user to consider his audience by presenting that item first on a menu of choices but allow the author to skip directly to composition.

When composing a document, even a letter, many books on writing recommend beginning by making an outline. The system might also present this option first, while allowing the user to choose another option, such as entering the address.

After a letter is composed, a spelling checker may be engaged unless the user decides not to. In general, the system may order menu choices at each point in the production to put first on the list necessary options that have not yet been done or options that are typically done next in sequence. A normative (ideal) model of the letter-writing process has been designed and compared to actual procedures used by a small number of subjects.[52]

Although people tend to engage in a particular sequence of tasks in producing a letter, they often interrupt the process and address a previously unstated goal. In this way, the process of composing a (nonroutine) business letter illustrates the same cyclical problem solving process observed in the design dialogues. The following excerpt from the protocol of someone talking aloud while writing a letter in the service of getting a job illustrates this process:

(The writer expressed the goal of getting a job and began by writing the first sentence of the letter). "Umm. O.K. So, now what I'm thinking about is well that's not bad. . . . Maybe that should be the second paragraph. Maybe it can stay as the first paragraph. Does it meet my criteria? Let's see. Ummm. For a beginning. Which I suppose would be to give whoever's reading the letter an idea of my particular need . . . my . . . and the thrust, I guess, of the interview, which is to . . . for them to tell

me about the things I don't know about since I say that I'm changing my career. Umm. And, I'm obviously also interested in what's there. As far as job possibilities. So that's really not a bad beginning, I guess. O.K. O.K. Now. So our first paragraph ends. . . . Now, what else do I want to say and what else do they want to know . . .?''

The letter-writing composition process appears to be analyzable into cycles in a manner similar to other design problems.

Letter-writing presents just one common example of semistructured documents. One can also find procedural and content regularities in composing such other documents as technical reports. For those formats that require references to be sequentially numbered, one can save time and effort by automatically numbering references and by renumbering them after the insertion of additional material.

The general conclusion is that the study of the process of designing and interpreting various forms can help determine how systems such as office communication systems can aid people to communicate more effectively and more efficiently.

Existent knowledge and communicative exchange

So far in our discussion of higher-level communication structures, we have been focused on the form of the linguistic vehicle for communication. We now turn to issues of the communicative content. Consider a communication problem in which one is trying to find out about a computer text editing system. The person has several partners in this communication exercise, including the system documentation, the system itself, perhaps a training manual, and perhaps even a teacher. Information—basically linguistic information—is exchanged and the person either learns enough about the system to use it to some extent or fails in this.

a defining metaphor

The traditional learning process is as follows: Understand a function (such as advancing the cursor), pair off a command word with that function, and finally be able to remember and employ one when you become aware of the other. Thus if a person becomes aware of wanting to advance the cursor, the immediate response is to type in NEXT. This conception of learning is inadequate to deal with any but the simplest and most uncharacteristic kinds of learning that naturally occur. We discuss this in more detail elsewhere.[16] For present purposes, we restrict ourselves to an adequate formulation of learning and its considerable implications for communication.

People almost always try to learn about new things by making use of past learning. New concepts are typically expressed in the terms of old concepts—at least initially. One way that this occurs we call metaphorical extension or simply metaphor. (See References 54 and 55.) An existent knowledge structure is loaded into memory and used as a structural template for further learning. The entities and relations of this *source knowledge structure* are transformed into a new domain by metaphor, and with the default assumption that the mapping can be an exhaustive isomorphism. Source domains can be consolidated and/or partitioned, but they are rejected only rarely. An immediate consequence of this is that the metaphors selected in learning or implicitly or explicitly suggested to the learner should be carefully chosen.

Consider the text editor example. Many users of text editing systems have unreasonable metaphorical models of such systems. Generally, this is because no attention has been paid to directing the selection and development of metaphors in the early stages of system learning. Almost all editors have cursor commands like UP, DOWN, NEXT, etc. but it seems that almost half of all new users confuse these directions. If, for example, the cursor moves up, the text window moves down. These users have not been given a concrete metaphor for cursor movement. They develop their own metaphor that often corresponds, only by pure chance, with the system designer's idea.

suitable metaphors

Zloof's Query-by-Example[35] may be an example of a system that capitalizes on an understanding of the way in which people think about a relational data base. We believe this query system is easy to learn because it uses the natural metaphor of a printed table for representing a computerized query system to the user.[36,37] (For further demonstrations of the utility of appropriate metaphors see References 15, 56, and 57.)

Cases of inappropriate metaphors are of course abundant. Bott[58] finds an interesting example in the word "command." About seventy percent of new users learning a text editing system misinterpreted the word as being something the machine tells them to do. Clearly, people do not literally misunderstand the word itself. Rather, they apply the wrong metaphor, thereby placing the computer in control of the editing session. Misperceptions occur when interfaces and instructional materials fail to direct people to useful metaphors.

What constitutes a suitable metaphor, however, is impossible to prescribe in advance, at least given our current understanding. We can, however, suggest general guidelines for choosing a metaphor for a computer system or some aspect of the computer system,[16] but these guidelines are tools, not rules.

The expected users of the system must be defined and characterized as a first step in creating metaphors. What metaphors are they spontaneously likely to adopt? How do people represent the knowledge that is to serve as the source for the metaphor? For example, someone who is designing a word processor for secretaries might assume that many intended users may initially try to understand the word processor as though it were a supertypewriter. What sorts of things does one expect a supertypewriter to do? How does one conceive a typewriter to work? Thus we may use appropriate metaphors as learning aids.

A system with different subparts may lend itself best to a composite metaphor in which fairly distinct parts of the system are related to different things. In such cases, the different parts of the metaphor should probably not be mutually incompatible. It is probably not wise, for example, to compare one part of a system to a tape recorder and another part to a dictating machine. On the other hand, the parts of the metaphor should not be taken from very disparate domains because it is probably too confusing to compare a system to a composite of a schoolroom, a robot cook, and a mosquito net. Composite metaphors should probably be of moderate diversity.

The differences between the source (typewriter) and the metaphor target (word processor) must also be examined. They must be pointed out to the learner, perhaps not initially but eventually. It is important to note that these are all empirical matters that for the present must be resolved on a case-by-case basis by systematic empirical studies of user populations and application environments.

Concluding remarks

Perhaps it would have been more satisfying to discover that communication can be adequately characterized as transmission across a channel from an encoding station to a decoding station. But if research work in cognitive and social psychology teaches us anything, it teaches us that mechanical simplicities are distinctly the exception. Thinking, behaving, and communicating are so much a part of the purposive, social, and design contexts in which they occur that it hardly makes sense to examine them outside these contexts.

Examining human capacities and propensities within these rich task environments, however, can be doubly rewarding. Research work that addresses the human condition in all of its inherent complexity can produce usable insights into the structure of human psychology. And the very activity of pursuing such research questions induces empathy for the user-end of a man-machine interaction. It helps to remind us that the user-end is indeed unique.

CITED REFERENCES

1. B. W. Boehm, "Software and its impact: a quantitative assessment," *Datamation* **19,** No. 5, 48–59 (May 1973).
2. R. Scott and D. Simmons, "Programmer productivity and the Delphi technique," *Datamation* **20,** No. 5, 71–73 (1974).
3. C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Systems Journal* **16,** No. 1, 54–73 (1977).
4. D. Meister, *Behavioral Foundations of System Development*, John Wiley & Sons, Inc., New York (1976).
5. L. A. Miller and J. C. Thomas, "Behavioral issues in the use of interactive systems: Part I. General issues," *International Journal of Man-Machine Studies* **9,** No. 5, 509–536 (1977).
6. B. Shneiderman, *Software Psychology*, Winthrop Publishers, Cambridge, MA (1979).
7. J. C. Thomas, "A design-interpretation analysis of natural English," *International Journal of Man-Machine Studies* **10,** 651–668 (1978).
8. T. Winograd, "What does it mean to understand language?", *Cognitive Science* **4,** 209–241 (1980).
9. A. Malhotra, J. C. Thomas, J. M. Carroll, and L. A. Miller, "Cognitive processes in design," *International Journal of Man-Machine Studies* **12,** 119–140 (1980).
10. J. M. Carroll, J. C. Thomas, and A. Malhotra, "A clinical-experimental analysis of design problem solving," *Design Studies* **1,** 84–92 (1979).
11. G. Glegg, *The Science of Design*, Cambridge University Press, Cambridge, England (1973).
12. E. deBono, *The Use of Lateral Thinking*, Cape, Ltd., London, England (1967).
13. V. Papanek, *Designing for the Real World*, Pantheon Books, New York (1971).
14. J. M. Carroll, J. C. Thomas, L. A. Miller, and H. P. Friedman, "Aspects of solution structure in design problem solving," *American Journal of Psychology* **93,** No. 2, 269–284 (1980).
15. J. M. Carroll, J. C. Thomas, and A. Malhotra, "Presentation and representation in design problem solving," *British Journal of Psychology* **71,** 143–153 (1980); reprinted in S. K. Reed (Editor), *Cognition: Theory and Applications*, Brooks/Cole Publishing Company, Monterey, CA (1981).
16. J. M. Carroll and J. C. Thomas, "Metaphor and the cognitive representation of computing systems," to be published by *IEEE Transactions on Systems, Man, and Cybernetics*.
17. J. M. Carroll, *Names and Naming: An Interdisciplinary Review*, Research Report RC 7370, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1978).
18. J. M. Carroll, *Natural Strategies in Naming*, Research Report RC 7533, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1979).
19. J. M. Carroll, "The role of context in creating names," *Discourse Processes* **1,** No. 3, 1–24 (1980).
20. J. M. Carroll, "Naming and describing in social communication," *Language and Speech*, to be published 1981.
21. J. M. Carroll, "Creating names for things," *Journal of Psycholinguistic Research*, to be published 1981.
22. J. M. Carroll, *Naming Personal Files in an Interactive Computing Environment*, Research Report RC 8356, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1980).
23. E. O'Dierno, "Designing computer systems for people," *Proceedings of the Symposium on The Role of Human Factors in Computers*, Baruch College, New York, November 1976.
24. J. M. Carroll, "Purpose in cognitive theory of reference," *Bulletin of the Psychonomic Society* **16,** 37–40 (1980).
25. D. R. Olson, "Language and thought: aspects of cognitive theory of seman-

tics," *Psychological Review* **77,** 257–273 (1970).
26. R. M. Krauss and S. Weinheimer, "Confirmation and the encoding of referents in verbal communication," *Journal of Personality and Social Psychology* **4,** 343–346 (1966).
27. J. M. Carroll, "Creative analogy and language evolution," *Journal of Psychological Research* **9,** 595–617 (1980).
28. J. M. Carroll, "Complex compounds: Phrasal embedding in lexical structures," *Linguistics* **17,** 863–877 (1980).
29. J. M. Carroll, *Learning, Using, and Designing Command Paradigms*, Research Report RC 8141, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1980).
30. J. C. Thomas, *Quantifiers and Question-Asking*, Research Report RC 5866, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1976).
31. J. Roberge, "A reexamination of the interpretation of errors in formal syllogistic reasoning," *Psychonomic Science* **19,** No. 6, 331–333 (1970).
32. J. Ceraso and A. Provitera, "Sources of error in syllogistic reasoning," *Cognitive Psychology* **2,** 400–410 (1971).
33. R. Revlis, "Syllogistic reasoning: Logical decisions from a complex data base," R. J. Falmagne (Editor), *Reasoning: Representation and Process in Children and Adults*, John Wiley & Sons, Inc., New York (1975).
34. E. Niemark and J. Santa, "Thinking and concept attainment," M. R. Rosenzweig and L. W. Porter (Editors), *Annual Review of Psychology* **26,** Annual Reviews, Palo Alto, CA (1975).
35. M. M. Zloof, "Query by Example," *AFIPS Conference Proceedings, 1975 National Computer Conference* **44,** 431–438 (1975).
36. J. C. Thomas and J. D. Gould, "A psychological study of Query by Example," *AFIPS Conference Proceedings, 1975 National Computer Conference* **44,** 439–445 (1975).
37. D. Greenblatt and J. Waxman, "A study of three data base query languages," B. Shneiderman (Editor), *Databases: Improving Usability and Response*, Academic Press, Inc., New York (1978).
38. P. Reisner, "Use of psychological experimentation as an aid to development of a query language," *IEEE Transactions on Software Engineering* **SE-3,** 218–229 (1976).
39. P. Wason and P. Johnson-Laird, *Psychology of Reasoning*, Harvard University Press, Cambridge, MA (1972).
40. J. Scandura, *Problem Solving*, Academic Press, Inc., New York (1977).
41. J. Erikson and M. Jones, "Thinking," M. Rosenzweig and L. Porter (Editors), *Annual Review of Psychology*, Annual Reviews, Palo Alto, CA (1978).
42. L. A. Miller and C. Becker, *Programming in Natural English*, Research Report RC 5137, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1974).
43. L. A. Miller, *Behavioral Studies of the Programming Process*, Research Report RC 7367, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1978).
44. J. C. Thomas, *A Method for Studying Natural Language Dialogue*, Research Report RC 5882, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1976).
45. J. M. Carroll and T. G. Bever, "Sentence comprehension: A case study in the relation of knowledge to perception," E. Carterrette and M. Friedman (Editors), *The Handbook of Perception, Volume 7, Speech and Language*, Academic Press, Inc., New York (1976), pp. 299–344.
46. G. A. Miller, "The magical number 7 plus or minus 2: Some limits on our capacity for processing information," *Psychological Review* **63,** 81–97 (1956).
47. T. G. Bever and J. M. Carroll, "On some continuous properties of languages," T. Meyers, J. Laver, and J. Anderson (Editors), *The Cognitive Representation of Speech*, North-Holland Publishing Company, The Hague, Netherlands (1980).

48. J. M. Carroll, "Sentence perception units and levels of syntactic structure," *Perception and Psychophysics* **23,** 506-514 (1978).
49. J. M. Carroll, "Functional completeness as a determinant of processing load during sentence comprehension," *Language and Speech* **22,** 347-369 (1979).
50. J. M. Carroll, M. K. Tanenhaus, and T. G. Bever, "The perception of relations: The interaction of structural, functional, and contextual factors in the segmentation of sentences," W. J. M. Levelt and G. B. Flores d'Arcais (Editors), *Studies in the Perception of Language*, John Wiley & Sons, Ltd., London, England (1978), pp. 187-218.
51. C. Daiute, "Psycholinguistic foundations of writing," *Research in the Teaching of English* (February 1981).
52. J. C. Thomas, "A cognitive model of letter writing procedures," Paper presented at the American Psychological Association Meeting, Toronto, Canada 1978; may be obtained from the author.
53. J. D. Gould and S. J. Boies, "Writing, dictating, and speaking letters," *Science* **201,** 1145-1147 (1978).
54. D. Norman, D. Gentner, and A. Stevens, "Comments on learning: Schemata and memory representation," D. Klahr (Editor), *Cognition and Instruction*, Erlbaum Associates, Hillsdale, NJ (1976).
55. D. Rumelhart and D. Norman, *Analogical Processes in Learning*, Office of Naval Research Report 8005, University of California, San Diego (1980).
56. R. Mayer, "Different problem-solving competencies established in learning computer programming with and without meaningful models," *Journal of Educational Psychology* **1,** No. 67 (6), 725-734 (1975).
57. R. Mayer, "Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order," *Journal of Educational Psychology* **1,** No. 68 (2), 143-150 (1976).
58. R. A. Bott, *A Study of Complex Learning: Theory and Methodologies*, Ph.D. thesis, University of California, San Diego, 1978.

John C. Thomas is located at the IBM Corporate Headquarters, Armonk, NY 10504, and John M. Carroll is located at the IBM Thomas J. Watson Research Center, P.O. Box 218, Route 134, Yorktown Heights, NY 10598.

Reprint Order No. G321-5148.