User Technology: From Pointing to Pondering

Stuart K. Card and Thomas P. Moran
Xerox Palo Alto Research Center

From its beginning, the technology of personal workstations has been driven by visions of a future in which people would work in intimate partnership with computer systems on significant intellectual tasks. These visions have been expressed in various forms: Memex (Bush, 1945), Man-Machine Symbiosis (Licklider, 1960), NLS (Engelbart, 1963), Dynabook (Kay, 1977), and others.

The tight coupling between human and computer required by these visions necessitated advances in the ways humans and computers interact. These advances have slowly begun to accumulate into what might be called a *user technology*. This user technology includes hardware and software techniques for building effective user interfaces: bitmapped displays, menus, pointing devices, "modeless" command languages, animation, and interface metaphors. But it must include a technical understanding of the user himself and of the nature of human-computer interaction. This latter part, the scientific base of user technology, is necessary in order to understand why interaction techniques are (or are not) successful, to help us invent new techniques, and to pave the way for machines that aid humans in performing significant intellectual tasks.

In this paper, we trace some of the history of our understanding of users and their interaction with workstations—the personal part of personal workstations. In keeping with the spirit of other papers at this conference, we have centered this review around our own experiences, perspectives, and work and have not attempted a complete history of the field. In concentrating on our own work, we do not wish to mimimize the importance of others' work; we simply want to tell our own story. Our focus is on what we have learned about users in our years of studying them and how we see our findings relating to the original visions of the personal workstation.

© 1986 ACM-0-89791-176-8-1/86-0183 $00.75

1. The Vision of an Applied User Psychology

The opportunity to tackle a science of the user brought us to PARC in 1974 (collaborating with Allen Newell, as consultant). As other PARC researchers were beginning to pursue the vision of highly graphic, interactive, network-based personal workstations, we were following a vision of our own. The idea was to draw concepts from cognitive psychology and artificial intelligence to create an applied cognitive science of the user. We called our project the Applied Information-processing Psychology Project (AIP). A 1971 memo by Allen Newell proposing this project to PARC stated the basic argument this way:

> (1) There is emerging a psychology of cognitive behavior that will permit calculation of behavior in new situations and with new humans. . . .
>
> (2) Several of the tasks that are central to the activities of computing—programming, debugging, etc.—are tasks that appear to be within the early scope of this emerging theory.
>
> (3) Computer science in general is extremely one-sided (for understandable reasons) in the treatment of its phenomena: almost no effort goes into understanding the nature of the human user. . . .
>
> (4) There is a substantial payoff (in dollars) to be had by really designing systems with detailed understanding of the way the human must process the information attendant thereto.

In 1974, we were in the position of having to create a new field. Psychological theories and methodologies held the promise of being able to represent and manage complex cognitive tasks, but the only body of research pertaining to human-computer interaction was in the field of human factors, where studies were largely empirical and evaluative, concentrating mostly on sensory-motor questions like the best shape for a switch. Human-computer interaction, involving two active agents each capable of initiating an exchange of information, inherently involves human cognitive processing. Our vision was to create a science of the user rooted in cognitive theory. But we also wanted the science to be practical, providing the system designer with the conceptual tools to think about the key characteristics of the user and the calculational tools to take account of the user's behavior.

To simplify a rather complex history for the purposes of this paper, we will narrate just a few strands of what unfolded. Each strand is organized around a particular aspect of the interface between the user, the system, and the task. The strands are organized into a sequence of ever-broader interfaces, from the physical interaction of the user with devices and displays to the symbiotic interaction of the user with the system while grappling with complex intellectual tasks—that is, from pointing to pondering:

> *The physical interface:* The user interacts with a system by means of physical input devices, such as a keyboard and a mouse, and output devices, such as a high-quality graphical display.
>
> *The cognitive interface:* The user has certain characteristics as an information processor, such as a limited working memory, that together with the goals he is trying to achieve determine his behavior.
>
> *The conceptual interface:* The computer system is also a complex information processing mechanism, and the the user needs to have some kind of mental model of it in order to effectively interact with it.
>
> *The task interface:* Systems are designed to help their users do tasks, not only small, routine tasks, but also larger, difficult intellectual tasks that are the object of the grand visions of personal workstations.

The story within each strand of our work is chronological; but the first three strands, representing our previous work, overlap in time. The last strand represents our current work.

2. The Physical Interface: Pointing

Bill English, who had among other thing engineered and built the mouse for Doug Engelbart (English, Engelbart, and Berman, 1967), was one of the people who migrated to PARC when it was founded. We worked for English when we first came to PARC. Engelbart and English had always considered the mouse to be an interim device, and English wanted to see if it was possible to invent other devices that would improve on its speed. In particular, he was interested in whether one could put a key on the keyboard that would sense force and direction without requiring the user to remove his hands from the keyboard. He had a number of devices built, and we helped design an experiment to test them. The results (Figure 1) showed that none of the devices tested improved on the mouse either for speed or for error rate (Card, English, & Burr, 1978).

This was a typical human factors experiment: we compared a set of systems to determine which one performed best. But this direct empirical comparison between devices was just the sort of methodology of human factors testing that we wanted to improve: we wanted to understand the reasons why the results came out the way they did. We therefore made mathematical models of each device and tested them against the data until we found models that fit. The model for the mouse was particularly instructive. The mouse was best modeled by a version of Fitts's Law:

$$Movement\ Time = Constant + .1 \log_2(D/S + .05) \text{ sec},$$

Device	Triats	Movement Time for Non-Error Trials						Error Rate	
		Homing Time		Positioning Time		Total Time			
	N	*M* (sec)	*SD* (sec)	*M* (sec)	*SD* (sec)	*M* (sec)	*SD* (sec)	*M*	*SD*
Mouse	1973	.36	.13	1.29	.42	1.66	.48	5%	22%
Joystick	1869	.26	.11	1.57	.54	1.83	.57	11%	31%
Step Keys	1813	.21	.30	2.31	1.52	2.51	1.64	13%	33%
Text Keys	1877	.32	.61	1.95	1.30	2.26	1.70	9%	28%

Figure 1. Overall pointing times for all devices. Homing Time is the time to move the hand from the keyboard to the device, Positioning Time is the time to move the cursor to the target. Times are based on a standardized set of distances and target widths. Averages are computed on the basis of four users x 600 trials/users. *Reprinted from Card, Moran, & Newell (1983, Fig. 7.4) with the permission of Lawrence Erlbaum Associates, Inc.*

where D is the distance the hand moves to the target and S is the target width (see Figure 2). The significance of this result is that this is the same law that describes movement time for the hand alone, with about the same constant of proportionality. The limiting factor in moving the mouse, therefore, is not in the mouse, but in the eye-hand coordinate system itself. That, in turn, means the mouse is nearly optimal, at least with respect to the set of muscles used. Therefore, designing a device that is faster than the mouse would be difficult.

Here was a prototypical example of the kind of theory we wanted to build—a model precise enough to enable designers to perform back-of-the-envelope calculations, a model that identified key constraints in the design space for pointing devices. With this model we could be sure that the data giving comparisons among devices would probably generalize to new situations, because we knew the main factors that governed the results. Furthermore, the model provided guidance for interface designers: make distant buttons large, for example. These studies were heavily used in the debate within Xerox that led to the decision to depart from tradition by including a mouse with the new Star product.

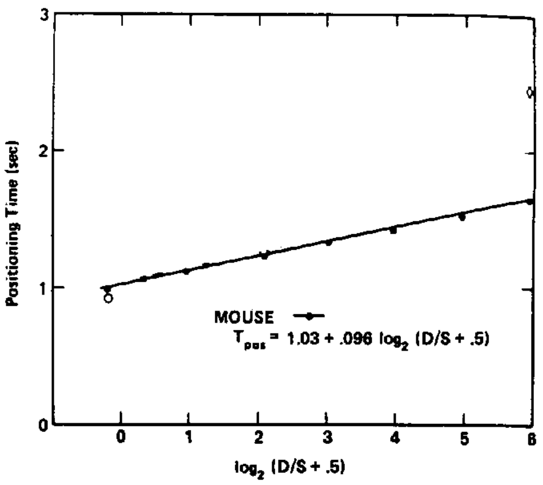

Figure 2. Positioning time for the mouse as a function of Fitts's Index of Difficulty. The fit of positioning time data to the straight line in the figure shows that time for the mouse is governed by Fitts's Law (is proportional to Fitts's Index of Difficulty). The constant of proportionality, .096 sec/bit (= 10.4 bits/sec), is approximately the same value as for pointing with the hand alone. *Reprinted from Card, Moran, & Newell (1983, Fig. 7.8) with the permission of Lawrence Erlbaum Associates, Inc.*

3. The Cognitive Interface: Cognitive Skill

The Model Human Processor

An interesting result of the mouse study was the way an evaluation of pointing devices led to a consideration of human information processing characteristics. This experience pointed to the need for an engineering model of users that would summarize such characteristics. We knew of many phenomena scattered in the literature of psychology, such as Fitts's law, that would be helpful for system design.

To someone who is not a specialist, such as a designer, this literature appears disorganized and contradictory. Psychologists love to split hairs and find small contradictions in published models. The robust but approximate generalizations that might be made to work for engineering tend to get trampled in the debates. Although we had the notion of such a model from about 1974, it wasn't until 1982, when we were nearing the completion of our book, that we were able to formulate it. The model, called the *Model Human Processor*, was inspired by the processors, memories, and switches (PMS) notation of Bell and Newell (1971) for describing the architecture of computing systems. It was a simplified architecture of the user, described in terms of three processors, four memories, 19 parameters of these, and 10 principles of operation (Figure 3).

As an example of the sort of calculation that can be done with the Model Human Processor, consider the case where a programmer is programming a video game version of billiards. He needs to know how long he has after the collision of two balls to compute the balls new trajectory before the illusion of causality breaks down. The Model Human Processor tells us that every event that occurs within 100 msec will be perceived as a single event, so a rough estimate is 100 msec. But it also recognizes that second-order phenomena can change this number, so it also supplies a range of uncertainty for this number (in this instance, 50 to 200 msec).

In this case, we can say that if the programmer can make the balls move within the 50 msec lower bound for this parameter, then it is pretty certain that users will perceive the collision as the cause of the balls change in direction, regardless of secondary effects such as brightness or contrast ratio of the screen.

The Model Human Processor can be used to compute predictions about human performance: how fast people can read, how fast they can scribble, the effect of different abbreviation schemes on memory error, and so on. Derivations based on the Model Human Processor were used to set the maximum velocity of the mouse on the workstation for the Xerox Star.

Text Editing and Cognitive Skill

The Model Human Processor also contains, as one of its principles of operation, Herbert Simon's bounded rationality principle. Our version of this principle may be stated:

A person acts so as to attain his goals through rational action, given the structure of the task, his *inputs of information, and bounded by limitations upon his knowledge and processing ability.* (Card, Moran, & Newell, 1983).

That is, in order to predict a person's behavior, one needs to analyze the task he is trying to do, because the person will simply do what is rational to accomplish the task, constrained by limitations in his knowledge and ability to process information. Studies of how people solve problems had shown (Newell & Simon, 1972) that their behavior could be modeled as a search through a space of states of the problem, a *problem space.* We expected that in studies of human-computer interaction we would find users searching through problem spaces to accomplish goals, trying various solutions, backing off and taking other tacks when they ran into trouble.

Two early decisions we made led us to results which were contrary to our expectations. Although we had come to PARC initially with the intention of studying computer programming, we decided once we arrived that there were strategic advantages to studying text editing. A second, tactical, decision was to work with expert subjects rather than novices, in order to have more stable behavior to analyze. Further, to make sure that all our subjects had the

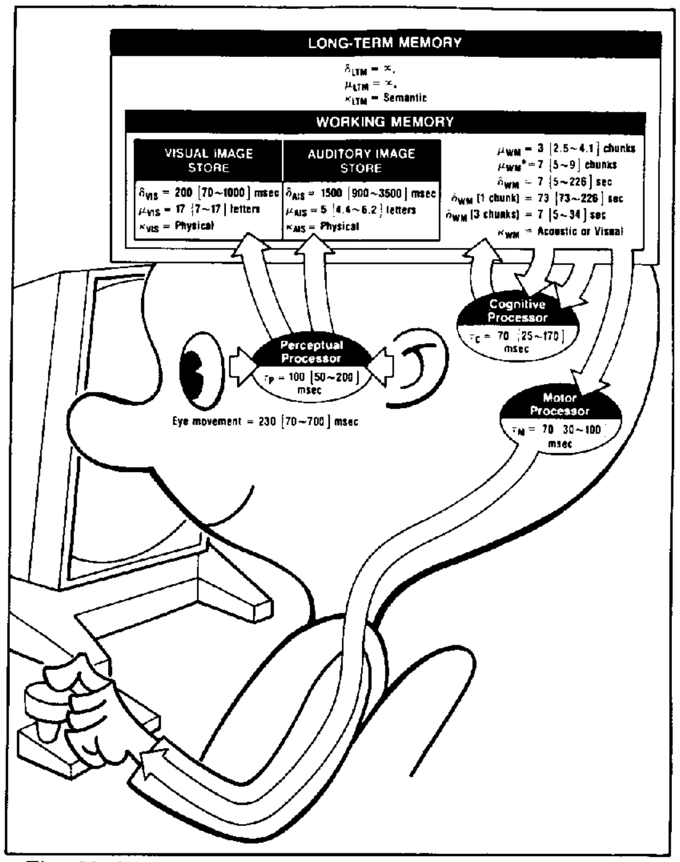
LONG-TERM MEMORY
$\delta_{LTM} = \infty$,
$\mu_{LTM} = \infty$,
κ_{LTM} = Semantic
WORKING MEMORY
VISUAL IMAGE STORE
δ_{VIS} = 200 [70~1000] msec
μ_{VIS} = 17 [7~17] letters
κ_{VIS} = Physical
AUDITORY IMAGE STORE
δ_{AIS} = 1500 [900~3500] msec
μ_{AIS} = 5 [4.4~6.2] letters
κ_{AIS} = Physical
μ_{WM} = 3 [2.5~4.1] chunks
μ_{WM}^* = 7 [5~9] chunks
δ_{WM} = 7 [5~226] sec
δ_{WM} (1 chunk) = 73 [73~226] sec
δ_{WM} (3 chunks) = 7 [5~34] sec
κ_{WM} = Acoustic or Visual
Cognitive Processor
τ_C = 70 [25~170] msec
Perceptual Processor
τ_P = 100 [50~200] msec
Eye movement = 230 [70~700] msec
Motor Processor
τ_M = 70 [30~100] msec

Figure 3. The Model Human Processor memories and processors. Depicted schematically in the figure are the memories, processors, and constants used for making simple computations. The basic architecture of the model can be summarized thus: Sensory information flows into Working Memory through the Perceptual Processor. Working Memory consists of activated chunks in Long-Term Memory. The basic principle of operation of the Model Human Processor is the Recognize-Act Cycle of the Cognitive Processor: On each cycle of the Cognitive Processor, the contents of Working Memory initiate actions associatively linked to them in Long-Term Memory; these actions in turn modify the contents of Working Memory. The Motor Processor is set in motion through activiation of chunks in Working Memory.

Predictions are made using time constants from the figure and a set of associated Principles of Operation: (P0) The Recognize-Act cycle of the Cognitive Processor; (P1) The Variable Perceptual Processor Rate Principle; (P2) The Encoding Specificity Principle; (P3) The Discrimination-Principle; (P4) The Variable Cognitive Processor Rate Principle; (P5) Fitts's Law; (P6) The Power Law of Practice; (P7) The Psychological Uncertainty Prinicple; (P8) The Rationality Principle; and (P9) The Problem Space Principle. *Reprinted from Card, Moran, & Newell (1983, Fig. 2.1.) with the permission of Lawrence Erlbaum Associates, Inc.*

same goals in mind, we presented them with a fairly explicit task—the "manuscript editing task"—which required them to work from a marked-up manuscript, making the changes explicitly indicated.

When we analyzed videotaped protocols of our subjects doing this task, we were surprised to find little of the search behavior of problem solving that we had expected. Subjects simply looked at the tasks and did them. The tasks were not problematic. The subjects had done similar tasks many times before, and had built up a large repertoire of methods that could be applied to the tasks. This wasn't problem solving; we came to call it *cognitive skill* and set out to build models to characterize and predict this mode of behavior.

We applied our theories of human information processing to the kind of specific skills necessary for the text-editing task. The result was a class of models in which the user's cognitive structure consists of four components: (1) a set of familiar Goals that the user would recognize when faced with a specific task; (2) a set of primitive Operators (actions) that the user was skilled at performing and could deploy whenever necessary; (3) a set of Methods, consisting of "compiled" sequences of subgoals and operators, that the user could use to attain his goals; and (4) a set of Selection rules that enable the user to choose among competing methods for goals. We call a model specified by these components a *GOMS model* (Figure 4 shows an example of one). Together, these components constitute the user's cognitive skills for performing tasks. If a user has enough knowledge of this kind, it isn't necessary to use problem-solving strategies. All that is needed is to examine the task, characterize it in terms of a specific goal, select the appropriate method, and then execute it.

In order to test our GOMS models, we ran a set of experiments to determine whether we could explicitly specify this kind of GOMS knowledge and thus predict what users would actually do. We also wanted to learn how the degree of resolution—the grain of specified detail of such a model—affects the degree of accuracy of predictions based on the model. We expected fine-grained models to yield increased accuracy. Knowing that it takes a lot more work to develop a fine-grained model, a practical question related to the applicability of our models was whether the additional work of constructing a fine-grained model was worth the effort.

To find out, we built a family of models in the GOMS framework that characterized the behavior of users of a specific computer text-editing system and ran experiments to test our predictions from these models. We were surprised to discover that fine-grained models did not yield a worthwhile or even a significant increase in prediction accuracy (Card, Moran, & Newell, 1976, 1980a). More felicitous, practically speaking, was our discovery that even the crude models seemed to capture and predict behavior fairly well. These properties suggested that the GOMS model could be turned into the kind of engineering tool that a designer could use to model and predict skilled user behavior in computer-mediated tasks. That was what we proceeded to try next.

KEYSTROKE LEVEL

Model K2:

```
GOAL: EDIT-MANUSCRIPT
.   GOAL: EDIT-UNIT-TASK                                   . repeat until no more unit tasks
.   .   GOAL: ACQUIRE-UNIT-TASK                            . . if task not remembered
.   .   .   GOAL: TURN-PAGE* (see below)                   . . . if at end of manuscript page
.   .   .   GOAL: GET-FROM-MANUSCRIPT*
.   .   GOAL: EXECUTE-UNIT-TASK                            . . if an edit task was found
.   .   .   GOAL: LOCATE-LINE                              . . . if task not on current line
.   .   .   .   CHOOSE-COMMAND
.   .   .   .   [select GOAL: USE-QS-METHOD
.   .   .   .             .   GOAL: SPECIFY-COMMAND*
.   .   .   .             .   GOAL: SPECIFY-ARG*
.   .   .   .           GOAL: USE-LF-METHOD
.   .   .   .             .   GOAL: SPECIFY-COMMAND*]  . . . . . repeat until at line
.   .   .   .   GOAL: VERIFY-LOC*
.   .   .   GOAL: MODIFY-TEXT
.   .   .   .   CHOOSE-COMMAND
.   .   .   .   [select GOAL: USE-S-COMMAND
.   .   .   .             .   GOAL: SPECIFY-COMMAND*
.   .   .   .             .   GOAL: SPECIFY-ARG*
.   .   .   .             .   GOAL: SPECIFY-ARG*
.   .   .   .           GOAL: USE-M-COMMAND
.   .   .   .             .   GOAL: SPECIFY-COMMAND*
.   .   .   .             .   GOAL: SPECIFY-COMMAND*   . . . . . repeat until at text
.   .   .   .             .   GOAL: SPECIFY-ARG*
.   .   .   .             .   GOAL: SPECIFY-COMMAND*]
.   .   .   .   GOAL: VERIFY-EDIT*
```

* Expansion of goals appearing several times:

```
GOAL: TURN-PAGE
. LOOK-AT-MANUSCRIPT                    . repeat twice
. ACTION
. MOVE-HAND                              . repeat twice
. TURN-PAGE
GOAL: GET-FROM-MANUSCRIPT
. LOOK-AT-MANUSCRIPT
. SEARCH-FOR
. LOOK-AT-DISPLAY                       . optional
GOAL: SPECIFY-COMMAND
. GOAL: GET-FROM-MANUSCRIPT*            . if not already selected
. CHOOSE-COMMAND                        . if not already selected
. GOAL: TYPE-STRING*
GOAL: SPECIFY-ARG
. GOAL: GET-FROM-MANUSCRIPT*            . optional
. CHOOSE-ARG
. GOAL: TYPE-STRING*
GOAL: VERIFY
. LOOK-AT-DISPLAY
. GOAL: GET-FROM-MANUSCRIPT*            . optional
. COMPARE
GOAL: TYPE-STRING
. HOME                                  . optional
. LOOK-AT-KEYBOARD                      . optional
. LOOK-AT-DISPLAY                       . optional
. TYPE-STRING
```

Figure 4. GOMS Model K2 for the text editor POET. This model has a grain of about 0.5 sec/operator. The user is using a line-oriented editor to make changes to a computer file previously marked on a paper manuscript. *Reprinted from Card, Moran, & Newell (1983, Fig. 5.12) with the permission of Lawrence Erlbaum Associates, publishers.*

The Keystroke-Level Model

In simplifying the GOMS models into an engineering model that we could hand to a designer, we constructed an idealized prediction problem:

> *Given a task (possibly involving several subtasks), the command language of a system, the motor skill parameters of the user, the response time parameters of the system, and the method used for the task, predict how long an expert user will take to execute the task using the system, providing he uses the method without error.* (Card, Moran, & Newell, 1980b)

The Keystroke-Level Model enables a system designer to make such predictions with a "back-of-the-envelope" style of calculation.

To do the calculation, the designer codes the method a user employs to do a task in terms of a set of operations derived from one of the fine grain GOMS models (Figure 5). In this simplified model, all keystrokes are assumed to take a constant amount of time; and pointing with the mouse is also assumed to take a constant amount of time. Mental activity by the user is reduced to a single generic mental preparation operation, and rules governing when it will occur are provided.

A typical use of the model to analyze a method is given in Figure 6. This method is one way in which a hypothetical text-editor could be used to replace a word recently typed by the user. Each action of the method is described in terms of the operators of the model, then a time for the method is computed. In this case, the method is expected to take the same amount of time regardless of how many words back the word to be replaced is located. Figure 7 shows the expected time for performing this methods and two other methods available in the editor as a function of the number of words back the to-be-changed word is located. It can be seen that the time profiles of the three methods are quite different and that each is fastest at different times.

Operator	Description and Remarks	Time (sec)
K	PRESS KEY OR BUTTON. Pressing the SHIFT or CONTROL key counts as a separate K operation. Time varies with the typing skill of the user; the following shows the range of typical values:	
	Best typist (135 wpm)	.08
	Good typist (90 wpm)	.12
	Average skilled typist (55 wpm)	.20
	Average non-secretary typist (40 wpm)	.28
	Typing random letters	.50
	Typing complex codes	.75
	Worst typist (unfamiliar with keyboard)	1.20
P	POINT WITH MOUSE TO TARGET ON A DISPLAY. The time to point varies with distance and target size according to Fitts's Law, ranging from .8 to 1.5 sec, with 1.1 being an average. This operator does *not* include the (.2 sec) button press that often follows. Mouse pointing time is also a good estimate for other efficient analogue pointing devices, such as joysticks (see Chapter 7).	1.10
H	HOME HAND(S) ON KEYBOARD OR OTHER DEVICE.	.40
$D(n_D, l_D)$	DRAW n_D STRAIGHT-LINE SEGMENTS OF TOTAL LENGTH l_D CM. This is a very restricted operator; it assumes that drawing is done with the mouse on a system that constrains all lines to fall on a square .56 cm grid. Users vary in their drawing skill; the time given is an average value.	$.9n_D + .16l_D$
M	MENTALLY PREPARE.	1.35
$R(t)$	RESPONSE BY SYSTEM. Different commands require different response times. The response time is counted only if it causes the user to wait.	t

Figure 5. The operators of the Keystroke-Level Model. The figure lists the operators needed to analyze user interface methods and to make calculations of user performance with these methods. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.1) with the permission of Lawrence Erlbaum Associates, Inc.*

Even though our GOMS study suggested that simple models could be effective, and even though the Keystroke-Level Model was a careful simplification of one of the GOMS models, we felt that it was necessary to rigorously text the explicit performance assumptions of the model. To validate the Keystroke-Level Model, we ran a large set of experiments in which people performed tasks with text editors, graphics editors, and executive

Method R (Replace):

Terminate type-in mode	MK[ESC]
Point to target word and select it	H[mouse] P[word] K[YELLOW]
Call Replace command	H[keyboard] MK[R]
Type new word	4.5K[word]
Terminate Replace command	MK[ESC]
Point to last input word and select it	H[mouse] P[word] K[YELLOW]
Re-enter type-in mode	H[keyboard] MK[I]

$$T_{execute} = 4t_M + 10.5t_K + 4t_H + 2t_P$$
$$= 12.1 \text{ sec}.$$

Figure 6. Encoding of Method R. Use of the Keystroke-Level Model to describe one possible method in a mouse-based display editor for replacing a word previously mistyped. It is assumed the word is still visible on the screen. *Reprinted from Card, Moran, & Newell (1983, p. 289) with the permission of Lawrence Erlbaum Associates, Inc.*

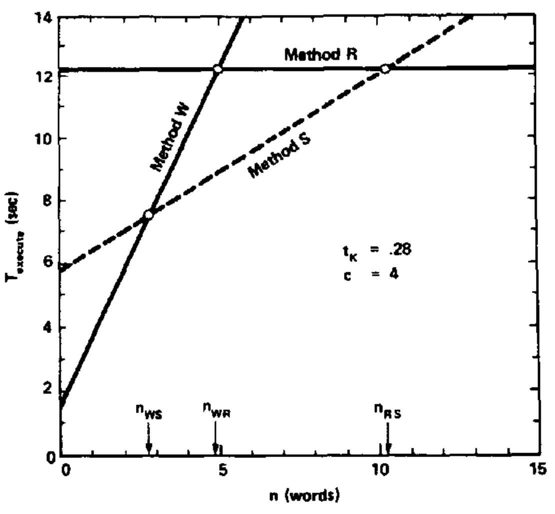

Figure 7. Execution time of three methods for the misspelled-word task as a function of *n*. This figure shows the use of the Keystroke-Level Model for the parametric comparison of three different methods for accomplishing the same goal. In this case each method appears to be superior in a certain range of *n*. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.12a) with the permission of Lawrence Erlbaum Associates, Inc.*

subsystems. The results (Figure 8) showed a good fit between predictions derived from the model and observed times required for the tasks. Therefore, our simple approximative model would be of some use in making practical time estimates.

The Keystroke-Level Model has actually proved useful in real system design. One application was the determination of the number of buttons on the mouse for the Xerox Star product. Several schemes for selecting text in the Star text editor were proposed. These schemes required different numbers of buttons. The goal was to make a mouse with the smallest number of buttons possible, so that it was easy to learn to operate. Experiments to test the schemes were reasonably easy to run with novice subjects. Everyone is a novice subject on a new system, and being a novice doesn't require training. But understanding how well the schemes would work for expert users (which most users would be for most of their time on the system) was expensive, because a long time would have to be spent training the users. The solution was to run experiments for novices and to use the Keystroke-Level Model for predicting expert performance.

The Keystroke-Level Model allowed us to carry through, at least in regard to a very specific kind of behavior, part of our original vision of packaging psychological knowledge into a model that designers can use to calculate user performance with a variety of interactive computer systems.

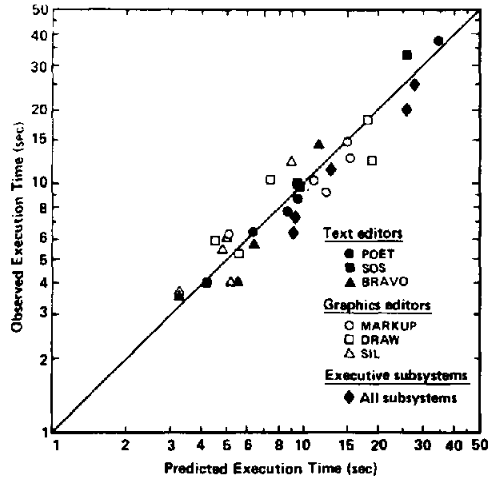

Figure 8. Predicted vs. observed execution times in tests of the Keystroke-Level Model. Predicted execution times were calculated from the Keystroke-Level Model. Observed execution times come from empirical observation. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.6) with the permission of Lawrence Erlbaum Associates, Inc.*

The Unit Task

Not all skill characteristics can be reduced to simple counting, however. In studying text editing, we observed a characteristic of cognitive skill that is fundamentally cognitive, namely the organization of user behavior into short, quasi-independent tasks, which we call *unit-tasks.* Figure 9 shows a typical timeline of user behavior, in this case from a protocol of an electrical engineer using a graphic CAD system to design a VLSI circuit. The figure shows that the behavior is divided into chunks, each consisting of a a few seconds of pause (to formulate a task to do) followed by a few seconds of activity (to execute the task). Each think-execute chunk of behavior is a unit task.

The unit task structure of cognitive skill is interesting because the performance limitations of the user show through the purely rational organization of his behavior. The most significant reason why the unit task breakdown of behavior arises is because of limitations in the user's working memory. If the user can manage input and output streams in his working memory, then the user's behavior will have a continuous structure, as in touch typing from a manuscript. But, when conditions on the inputs and outputs do not allow this, then behavior must be structured into a series of unit tasks.

```
Clock |         10        20        30        40        50        60 sec
 Time |....:....|....:....|....:....|....:....|....:....|....:....|

Transcription Phase

 0:18 |-------DDDD=DDD=DDDDDDDDD=DDDD=DDDDDD
 0:55 |------D=D=D
 1:06 |-----D===DDDDDDDDDD
 1:25 |-----DD==DD
 1:36 |-------------------------AAAAAAAAAAAAAAAA
 2:17 |----------DDDDDD===DDDD=DDDDDDDDD
 2:50 |--------DDD=DDDDDDDDDD==DDD=D==DDDDD
 3:30 |----DDDD==DDDDD
 3:41 |--------D---------DD
 4:01 |-------------AAAAAAAA
 4:22 |------A=A
 4:31 |-------DDDDDD=DDD==DD
 4:52 |---------------------------D=DD
 5:23 |-----DD
 5:30 |-----A=A
 5:38 |-------DD====DDDDD=DDD
 6:00 |---------------D==DD
 6:20 |-----DDDD=DDDDDDDDDD
 6:44 |---D-D------D==DD
 6:57 |------------DDDDDDDDD=DDDD
 7:23 |------------D---DD----D
 7:46 |----------AAAAAAAA==AAAAA
 8:11 |-------A=A=AAA=AAAAAAA
 8:33 |---------D===DDDD=DDDD==DDDD
 9:01 |---------D=DD
 9:14 |-----------DDDD==DDD=DDD==DDDDD
 9:45 |--------------D==D=DD=DDDDDDDDDDD
10:18 |-----DDDD==DDD===DDD
10:38 |--------DD=DDDDD
10:54 |-----------D==D=DD===DDD
11:18 |-----D=DD==DDDD==DDDDD
11:44 |----MM
11:46 |----------D-DDDD----DDDD==DDDDDDD
12:21 |C------------------------------------------
13:05 |A=AA
13:09 |-----------DDDDD=DDDDD=DDDDDDD====DDDD
13:47 |-------DDDDD=DDDDDDD
14:07 |-------------------
```

<u>**Symbol Code**</u>:

-	Task acquisition
D	Draw task execution
A	Alter task execution
M	Dimension task execution
C	Check task execution
=	Intra-execution pause
+	Pause within Move command

Figure 9. Time line representing the user's behavior sequence in a VLSI design session. Each single-character symbol represents one second of behavior. The symbol sequence begins on a new line at the beginning of each unit task, and the clock time is the time at the beginning of the unit task. *Reprinted from Card, Moran, & Newell (1983, Fig. 10.7) with the permission of Lawrence Erlbaum Associates, Inc.*

Figure 10 portrays a trace of the working memory load of the user, based on logical considerations of when information must be available in memory in order to do a task. As we can see, memory requirements build from a low point at the start of a unit task to a high within the unit task, where information for the task has been assembled, and back to a low point at the end of the unit task, where the information is no longer needed. If the information in working memory should reach a level higher than the working memory capacity, then user performance will suffer, usually manifested by the user committing errors. To avoid these errors, the user must break down the overall task into smaller (unit) units, each of which can be managed in the available working memory.

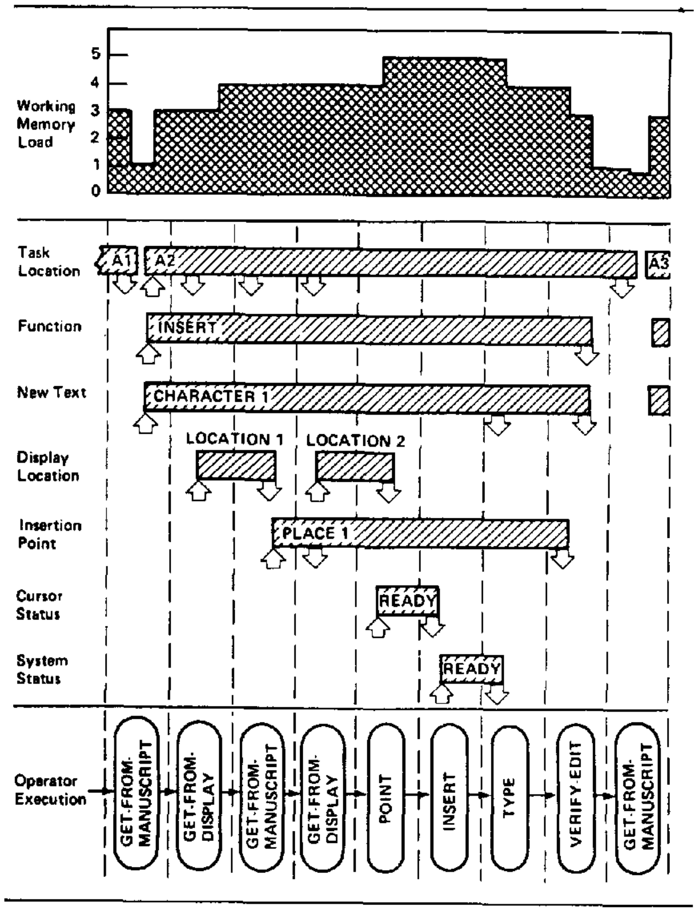

Figure 10. Data in Working Memory during a unit task. This figure is an hypothetical trace of the performance of one unit task. Time runs to the right on the horizontal axis. The bars indicate the time during which each piece of information about the task is needed. The arrows indicate the initial time data is available and the subsequent times it is needed. The histogram on the top plots the total Working Memory load over time, showing how the load peaks within a unit task and dips between unit tasks. *Reprinted from Card, Moran, & Newell (1983, Fig. 11.12) with the permission of Lawrence Erlbaum Associates, Inc.*

Unit-task behavior is at present a theoretical notion based on empirical observation (see Card, Moran, Newell, 1983, for further discussion). But it represents an important feature of user behavior—a sort of "cognitive rhythm"—that should be taken into account in designing user-computer interfaces that are sensitively tuned to the user's capabilities.

The Psychology of Human-Computer Interaction

In 1978, one of us had completed a doctoral thesis (Card, 1978) that consolidated much of the work we had been doing. The thesis helped convince us that the time was appropriate for writing a book presenting our vision of an applied science of the user. Work on the book gave us the opportunity to focus on the larger vision instead of just the pieces, and we became aware of some missing unifying theoretical work that we felt we had to do.

Although we had several calculational models (Fitts's law for the mouse, GOMS models, the Keystroke-Level Model, plus other minor models), there were questions of relating these to the classical literature in cognitive psychology and human factors. We had long sought a unifying framework for tying together the relevant psychological knowledge about users (one of our 1974 working papers called this conception the "Handbook of Cognitive Man"), but had made little progress. In preparing the book we revived this goal and came to a much more satisfying conception, the Model Human Processor, which captured the relevant psychological literature in the terms of a unified, approximative model. This model also provided a foundation for our other models.

The other big missing piece for us was to understand the relationship between the cognitive skill we had discovered and the classical notion of problem solving in cognitive psychology. Here we built a theory of the behavioral continuum between problem solving and cognitive skill and showed how practice on a task would gradually change problem solving behavior into skilled behavior. This is an active area of research in cognitive psychology today (e.g., Anderson, 1981).

The result of these efforts was the book, *The Psychology of Human Computer Interaction* (Card, Moran, & Newell, 1983). However, the book represented only the main line of our research efforts that fit together into a tightly knit view. There were many other areas of our work that we decided not to put into the book, such as the issues of learning and of users' mental models.

4. The Conceptual Interface: Mental Models

The early effort of the AIP Project was focused on understanding expert user performance. As we explained in the previous section, skilled performance is characterized by methods that users know and quickly execute to accomplish tasks. That is, we were focused on what is typically called procedural knowledge ("how-to-do-it" knowledge). However, from the beginning, we were aware that methods are not sequences of meaningless actions, but that expert users also have an understanding of what the procedures cause the machine to do. That is, the expert users have some sort of *mental model* of what is happening inside the computer ("how-it-works" knowledge).

The first AIP memo in 1974 proposed the notion of the "user's model," which refers to the conceptual model that the user can have of the system. A user's conceptual model is distinct from (but related to) the designer's implementation model. It is an abstraction of the system's architecture and software structures—the conceptual entities that the architecture and software implements—that is simple enough for non-technical users to grasp. (For example, a user might not know how the mechanism of the delete buffer of a text-editor works, but would simply know that the deleted text goes into a "clipboard.") A user's model would typically include knowledge of where information is stored (core memory, local disk files, remote file servers). It is important for the user to have an understanding of these kinds of features, for they are often not visible to the user.

The user's model provides an integrated package of knowledge that allows the user to predict what the system will do if certain commands are executed, to predict the state of the system after the commands have been executed, to plan methods for novel tasks, and to deal with odd error situations (by characterizing the system's state according to the model, then choosing operations necessary to leave that state).

Early Encounters with Conceptual Models

Early in our research we tried to write down rudimentary descriptions of the elements of conceptual models, which included the conceptual objects, their properties, how they related to each other (the characters in a text editor, for example, including the notion of a blank space as a character), and the conceptual operations that could be performed on those objects (inserting, moving, or deleting characters, including blank spaces).

Given the obvious importance of this kind of knowledge for the user, we were surprised to find that almost no system documentation ever clearly laid out a conceptual model of the system for the user. We were also surprised to discover what a difficult inductive task it is to specify such a model, even for a seemingly simple text editor. It was a detective game in which we were forced to hypothesize and test elements of possible models in order to find a succinct conceptual characterization of how the system worked. It was a game that took us days (not minutes) to do, hence not a game in which busy users were likely to engage.

It is clear that users attempt to make sense—by building mental models—of the behavior of a system as they use it. If a simple model is not explicitly or implicitly provided, users formulate their own myths about how the system works. These user-invented models may be inaccurate or misleading outside the very limited situation from which they emerged. Therefore, we believed that if the user is to understand the system, the system has to be designed with an *explicit* conceptual model that is easy enough for the user to learn. We call this the *intended user's model,* because it is the model the designer intends the user to learn. Just what mental model the user actually forms is another issue, which depends on how clearly the intended user's model is designed, how well it is implemented, and how well it is documented.

Although we were very concerned about the mental model issue, we didn't pursue explicit studies for several reasons: we didn't have a satisfactory methodology for studying it, we didn't have satisfactory representations of it, and we were busy pursuing the performance issues we have discussed. Still, we felt that the intended user's model was an important consideration in the user-interface design process.

User Interface Design Methodology

The practical application of our concerns came when Xerox began arranging the technology transfer between PARC and the Systems Development Division (SDD), which was created to develop office system products based on the research at PARC. In the Spring of 1976, a joint PARC/SDD committee (which included one of us) was formed to advise SDD on the design of the user interface of the office systems.

The committee decided not to try to design an actual interface, but to propose a methodology for SDD designers to follow in designing their interfaces. The methodology (Irby et al., 1977) included four parts: (1) analyze what tasks the user will want to do and the steps they go through to accomplish the tasks; (2) design an intended user's model in terms of which the tasks may be cast; (3) design a command language to make that model work; and (4) design an information display to reflect the operations of the system in terms of the conceptual model.

Thus, we recommended that the designer should lay out an intended user's model before designing the command language and the information display. The whole design effort should be oriented toward keeping this model "under control," i.e., keeping it simple, consistent, and clear enough for users to grasp.

The original designers of the Xerox Star interface, the workstation product SDD built, used this methodology. The conceptual model was clearly laid out in the system's functional specifications, and the designers worked hard to keep the model consistent. Although this model was represented informally, the fact that the designers focused on it contributed heavily to the widely recognized success of Star's user interface. The conceptual model is an under-appreciated aspect of Star's interface, but Star's more widely-touted icons and desktop metaphor only make sense with respect to its underlying conceptual model.

William Newman, another member of the committee, presented this methodology in the second edition of his book with Bob Sproull (Newman & Sproull, 1979).

Empirical Studies

By 1979, we were ready to tackle some empirical studies in order to understand the role of the user's model. The first study was an attempt to elicit the knowledge that real users of real systems have about the systems they use every day. The system we chose to study was the Alto Executive, a system which was in wide use at PARC for several years by nontechnical support people as well as programmers.

We wanted to find out what Alto users knew about the Executive. The goal was to see whether we could find some kind of mental models buried in the user's knowledge. The methodology we used was to transcribe interviews with several users into logical propositions and then classify them into categories.

The surprising result was that many nontechnical expert users (e.g., secretaries who used the system effectively every day) gave no evidence of having anything but very shallow models of how the system worked. (Perhaps we shouldn't have been surprised, because no conceptual model was documented and training was informal.) This led us to consider more carefully the role of the user's model.

In order to characterize the role a user's mental model would play in the use of a system, we performed experiments with the simplest kind of system we could devise, a simple stack-based calculator (Halasz & Moran, 1983; Halasz, 1984). We thought that a model of a stack might well be useful to help rationalize what for many users is a nonintuitive postfix command language. The formal experiment compared one group of users who were taught an explicit conceptual model of the stack, and a group of users who were carefully shielded from the stack model. The model group was trained in relation to a specific model and were told how that model related to the methods for solving arithmetic problems. The no-model group was only taught specific methods for performing the same tasks. Then we gave the two groups sets of problems that were categorized as simple routine tasks, slightly more complex tasks, and very difficult "invention tasks" (which required the user to invent new methods to solve).

The results (see Figure 11) revealed no difference between the two groups in both the routine and complex problems, but the model group performed much better on the invention tasks. The most surprising result was that even some of the no-model group were able to perform some of the invention tasks. We wanted to account for these results according to the cognitive theories we understood, so we analyzed the protocols gathered from the users as they performed the tasks.

Problem Type	No-Model Users	Model Users
Routine	98	95
Complex	87	88
Invention	25	67

Figure 11. Percentage of problems correctly done in the calculator experiments. There were two groups of users; one was taught an explicit conceptual model of the calculator's stack and the other group was prevented from having such a model. There were three types of problems, each of a different difficulty relative to what the users were taught.

Problem Type	No-Model Users				Model Users			
	Skilled Method Execution	Problem Solving			Skilled Method Execution	Problem Solving		
		Model Space	Methods Space	Task Space		Model Space	Methods Space	Task Space
Routine	90	0	8	2	89	11	0	0
Complex	94	0	6	0	91	7	0	2
Invention	2	0	84	14	7	71	19	3

Figure 12. Partitioning of the users' behavior in the calculator experiments. The users' behavior was devided into four behavioral modes: skilled method execution plus three problem solving modes, which are distinguished according to which problem space they were working in. The partitioning is based on an analysis of verbal protocol records; each line of protocol was categorized into the mode it manifested. The numbers in the table are the percentages of lines of protocol in each behavior mode. We believe that this measure underestimates the amount of work in the task space.

We divided the users' behavior into skilled method execution, as we had modeled in our earlier studies, and problem-solving. We found (see Figure 12) that in the routine and complex problems, the behavior was almost all skilled method execution: the subjects had been taught what to do, and they did it; even for the complex problems it wasn't difficult to knit together the methods they had learned for solving the problems.

Our hypothesis going into the study was that the conceptual model we taught the model group of users would provide them with a problem space through which they could search in order to find solutions. According to this theory, the user would characterize a difficult "invention" task in terms of this problem space: the state of the system when they started, the state of the system they would like to achieve, and a set of operations to move them through that space; they would solve the problem using generic problem-solving strategies.

However, when we analyzed the users protocols, we found two other kinds of problem spaces in which the users worked when they were problem solving. One was a *task space* within which they manipulated the given arithmetic task in various ways, such as dividing it into subtasks that could be solved by known methods. Another space in which they worked was a *methods space*, where they took known solution methods as strings of steps, manipulated those steps in various ways to produce new methods, and tried them to see how they worked.

We found that the most critical (although not very time consuming) problem solving was in the task space, where the user analyzed the given task into subtasks and delegated them to the model space or the methods space. The main difference between the users who had a model and those who didn't was that they had different problem spaces in which to work (see Figure 12). The model space was an effective problem space, within which the solution to the invention problems could be found; the method space was not particularly effective, but it was sufficient to allow some non-model users to stumble onto solutions to some of the invention problems (often, much to their surprise).

We concluded that mental models can be useful for novel task situations, but we found that users only use their models in specific subtasks; there was a lot of switching between the task space and other problem spaces. Users were cautious about going into a mode of behavior that involved thinking through a mental model. Model-based problem solving appears to be very mentally intensive, so users avoid it if they can apply cognitive skills. But, if users don't have appropriate methods available, they will retreat to some kind of problem solving. In these cases, a good conceptual model provides an effective problem space in which to work.

Thus, system designers should think of a conceptual model of a system as not just a simple view of a complex system, but as a problem space through which users can search for solutions to a variety of novel problems. The conceptual entities and operators in the intended user's model should be closely related to the kinds of tasks the users are likely to do, and the users should be provided with heuristics for moving through the model space.

Theoretical Studies: Task Mapping

We also worked on a theoretical analysis of conceptual models to show where they fit into the overall structure of the user interface. The Command Language Grammar formalism (Moran, 1978, 1981) shows how models relate to the task domain, the command language, and the detailed user-computer interactions. According to this theory, the conceptual model provides the user with a link between his task domain and the syntax of the interactive dialogue. That is, on the one hand the conceptual model serves as the semantics of the dialogue actions, while on the other hand it serves as a base into which the task can be mapped into the system.

Richard Young called this kind of mechanistic conceptual model a *surrogate model* (Young, 1983). Interactions on these issues with Young, who was exploring the domain of simple calculators, led to the discovery of a new kind of mental model—*task-action mappings* (Young, 1981). The properties of radically different calculator designs, such as algebraic versus stack calculators, could be best understood by an analysis of how well calculation tasks could be directly mapped into the actions available on the calculators. Surrogate models were completely bypassed in this analysis, which helps explain why people sometimes seem to get along with systems without having surrogate models of them.

In further work along this line, we have proposed a calculus, called *ETIT analysis*, for task mapping (Moran, 1983). The "fit" of a system to a task domain can be assessed by enumerating rules for reformulating system-independent task descriptions ("external tasks") into system-specific task descriptions ("internal tasks"). Rule-based system description techniques, such as this or Payne's (1984) *task-action grammar*, are beginning to provide a way to help us formalize the fuzzy notion of the *consistency* of a system (both internal consistency and consistency with respect to a task domain). Such techniques look promising as a way provide system designers with calculational techniques for predicting the learnability and "guessability" of systems.

5. The Task Interface: Pondering Ideas

Let us now turn to the role of a science of the user in the future development of the personal workstation. For us, the real challenges for user technology are now at the larger task level in which users are grappling with complex intellectual tasks. This is, of course, a return to working directly on a modern version of the original vision of augmentation workstations.

We now have workstations powerful enough to give each user a personal system equivalent to one that only a decade ago would have been shared with a hundred other users, and we are on the verge of major upgrades to even greater computing and communication power. We now have a much more developed base for user technology—not only techniques for designing user interfaces, but also models for understanding users. Thus, we are now in a better position than ever before to explore systems that can really augment human intellectual endeavors.

The challenge is to create systems that, through intimate cognitive interaction with users, aid them in structuring and manipulating their ideas. With such systems to help them, people will then be able to cope with more and more complex intellectual tasks of all sorts—authoring books and multi-media presentations, designing products and programs, composing music, analyzing experiments, the arguing points of law and policy, reasoning about scientific and social issues, and on and on.

The key to building such systems is to find ways by which a user can act on his ideas as objects, just as current text editors allow him to act on words as objects. This is difficult, since ideas are often tacit and ill-formed. A means is needed to *externalize* the ideas, to get them out of the user's head and into a form that can be organized and shaped. Two problems require solution:

(1) The user needs new ways to *represent* his ideas—to get the user's mind around the ideas, as it were. For this we can exploit advances in artificial intelligence and cognitive science.

(2) The user needs new ways to *manipulate* his ideas—to get the user's hands on the ideas. For this we can exploit advances in interactive computer graphics.

These problems are part of our current research agenda. In this endeavor we must not only build on our current base of understanding of the user, but also advance that understanding. For one thing, we have to launch studies into the nature of the complex intellectual tasks we wish to augment. This calls for a shift in our research strategy from studying users of existing systems to studying users of new systems that we ourselves build, which enables us to understand the nature of the tasks and the limits of users and systems for dealing with them. We give illustrations of this strategy from some of our own current projects.

Representation: Idea Structuring

Word processors and text editors, even powerful ones, are tailored for the final stages of writing a paper—crafting the text and graphics of the final product. Outline processors help with the previous stage of outlining. Although the latter are sometimes called "idea processors," it is clear that idea processing begins well before the stage of outlining. A genuine idea processor should allow the user to deal with ideas that are vague and ill-structured and help him gradually add structure as it is discovered.

We use the term *authoring* to refer to the larger intellectual task of gathering information, extracting and discovering ideas, structuring them, and finally composing them into a readable product. Authoring in this sense is a highly general task composed of generic subtasks, as illustrated in Figure 13. One begins by collecting sources of information; from these a set of notes (idea-sized units) are created to represent potentially relevant facts and ideas. As notes accumulate, they need to be filed in a structure suitable for retrieval, such as a topic hierarchy. But these notes also need structures that organize the ideas into meaningful, coherent themes. Such structures must be discovered, elaborated, supported by evidence in the notes and other sources. One then communicates these ideas by composing them into an interpretable product—a document, a slide presentation, or a browsable network of ideas.

The key research issue here is to help the users develop explicit mental models of idea structures, so they can see them, play with them, and evaluate them. This requires the invention of representations for *externalizing* ideas and idea structures. Cognitively, these representations serve the

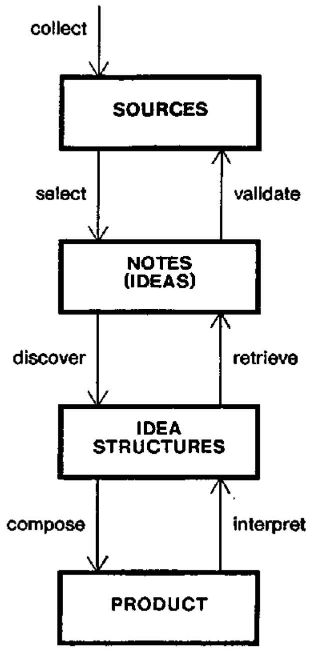

Figure 13. Schematic diagram of the generic authoring task. The boxes are the types of information to be managed in the authoring process, and the arrows are some of the processes for generating and transforming the information.

users as problem spaces; and new representations can fundamentally alter the way they are able to deal with ideas. Some representations should be "weak" in the sense that they can easily capture a large variety of even vague ideas; other representations should be "strong" in the sense that they can precisely capture and process particular classes of idea structures; and facilities must be provided to help users transform weak representations into strong ones.

We are developing a system, called *NoteCards*, to explore the broad nature of the authoring task. NoteCards supports the orienting metaphor of the notecard as a medium for capturing an idea. Notecards can be stored in fileboxes and linked into complex structures, which can then be viewed in graphic representations (such as the one shown in Figure 14). NoteCards is designed to be an open, flexible environment, so that we can use it as a "laboratory" to explore new representations and tools to support authoring tasks. Because it is an open system, NoteCards users are faced with the problem of devising appropriate ways to use the system, that is, mapping their particular tasks onto the basic elements of the system. This leads to a further interface issue of making the environment tailorable by users themselves to the wide variety of situations that we encourage them to bring to NoteCards (paper and documentation writing, scientific and legal argumentation, instruction authoring, design analysis). Thus, in contrast to our earlier methodology of observing a controlled, skilled user population, we are now observing a population of idiosyncratic, exploratory users.

Manipulation: Idea Browsing

The representations in an idea-structuring environment must be assimilated and manipulated with facility and speed. Idea-processing tasks are difficult for users, because they involve the retrieval, tracking, and manipulation of a large number of ideas (as is being attempted in Figure 14). Authoring a paper, for example, may require hundreds of notes and scores of references; programming may require hundreds of routines. But, as the Model Human Processor

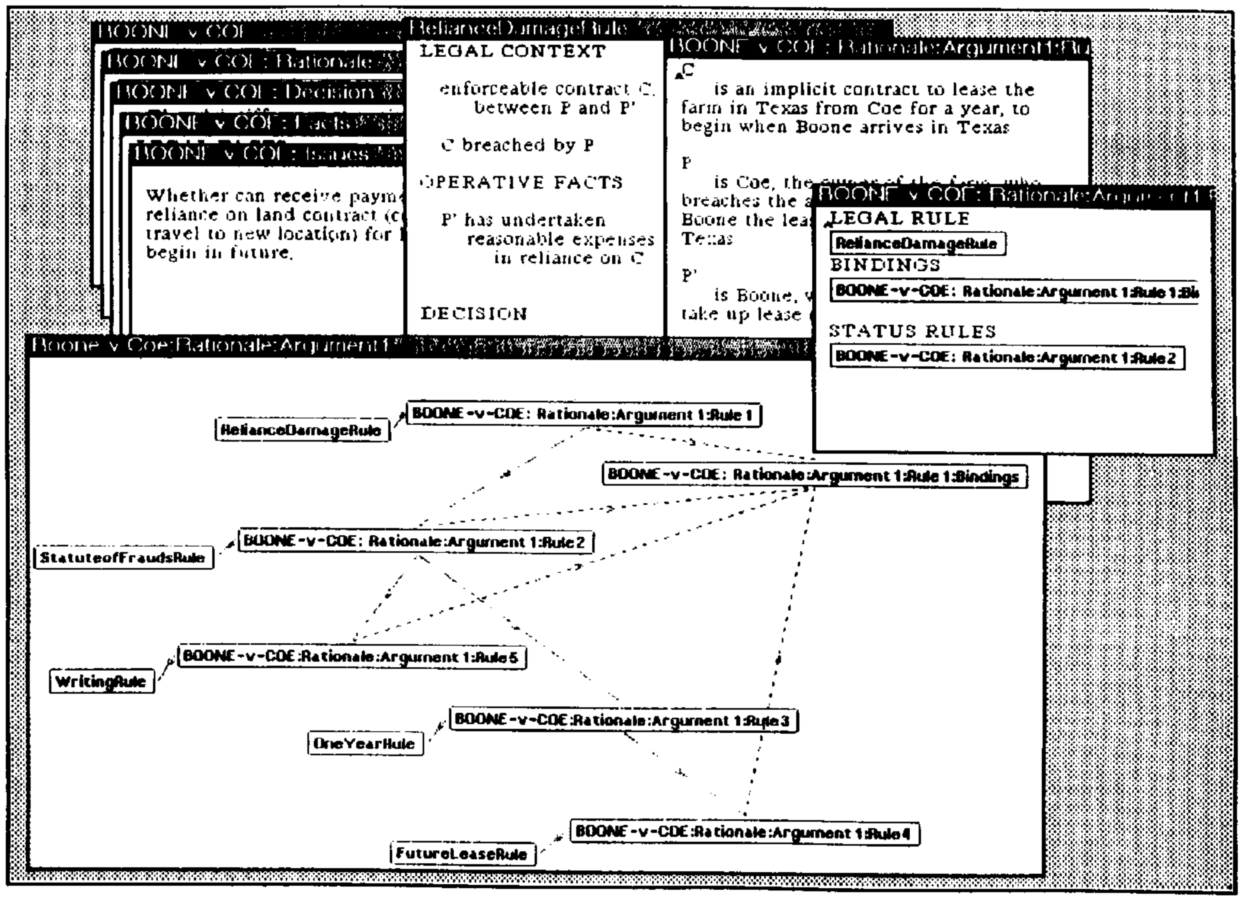
BOONE v COE
BOONE v COE : Rationale
BOONE v COE : Decision
BOONE v COE : Facts
BOONE v COE : Issues
Whether can receive paym
reliance on land contract (c
travel to new locati(on) for
begin in future.
RelianceDamageRule
LEGAL CONTEXT
enforceable contract C between P and P'
C breached by P
OPERATIVE FACTS
P' has undertaken reasonable expenses in reliance on C
DECISION
BOONE v COE : Rationale:Argument1:Ru
C
is an implicit contract to lease the farm in Texas from Coe for a year, to begin when Boone arrives in Texas
P
is Coe, the
breaches the
Boone the lea
Texas
P'
is Boone,
take up lease
BOONE v COE : Rationale:Argu
LEGAL RULE
RelianceDamageRule
BINDINGS
BOONE-v-COE: Rationale:Argument 1:Rule 1:Bi
STATUS RULES
BOONE-v-COE: Rationale:Argument 1:Rule 2
Boone v Coe:Rationale:Argument1
RelianceDamageRule
BOONE-v-COE: Rationale:Argument 1:Rule 1
BOONE-v-COE: Rationale:Argument 1:Rule 1:Bindings
StatuteofFraudsRule
BOONE-v-COE: Rationale:Argument 1:Rule 2
WritingRule
BOONE-v-COE:Rationale:Argument 1:Rule 5
OneYearRule
BOONE-v-COE:Rationale:Argument 1:Rule 3
FutureLeaseRule
BOONE-v-COE:Rationale:Argument 1:Rule 4

Figure 14. A screen image of the NoteCards system. The application shown is the analysis of a legal argument. A variety of legal rules must be brought to bear in arguing a particular case. The graph shows how the rules are related by applicability conditions and by bindings to the particulars of the case.

shows, only limited amounts of information can be handled in the user's working memory. These limitations lead the user to structure his behavior into unit-task bursts.

These considerations lead us to ask how a computer system can compensate for human cognitive limitations. The display can be used as an external memory to augment the user's internal working memory. This partly explains what has made the "desk-top" metaphor successful: users can use windows and icons to keep track of more documents, notes, and messages from other users than with the previous style of command language interfaces. By having these things visible as reminders, the user's effective working memory is increased, enabling him to do more complex tasks.

But the problem is that these techniques, which work well with with a few dozen objects, do not scale well up to the several hundred or even thousands of objects necessary for the idea-processing tasks we wish to do. Here, it helps to begin with a better understanding of the nature of the problem. Again, our method for gaining that understanding is through approximative models of user behavior. In this case, we have sketched a model called the Window Working Set Model (Card, Pavel, & Farrell, 1984) that analyzes access to screen objects in a manner analogous to the analysis of virtual memory operating systems. Informally, this model suggests that screen space itself is the key constraint and that at some point as the number of, say, overlapped windows required increases, user performance will decrease in a sudden and non-linear way, sending the user into the window version of thrashing.

Thus there are limitations on the uses of current graphical interface techniques for browsing large sets of ideas. Advanced graphics systems, however, open up a new set of possibilities. Figure 15 shows a fragment of an

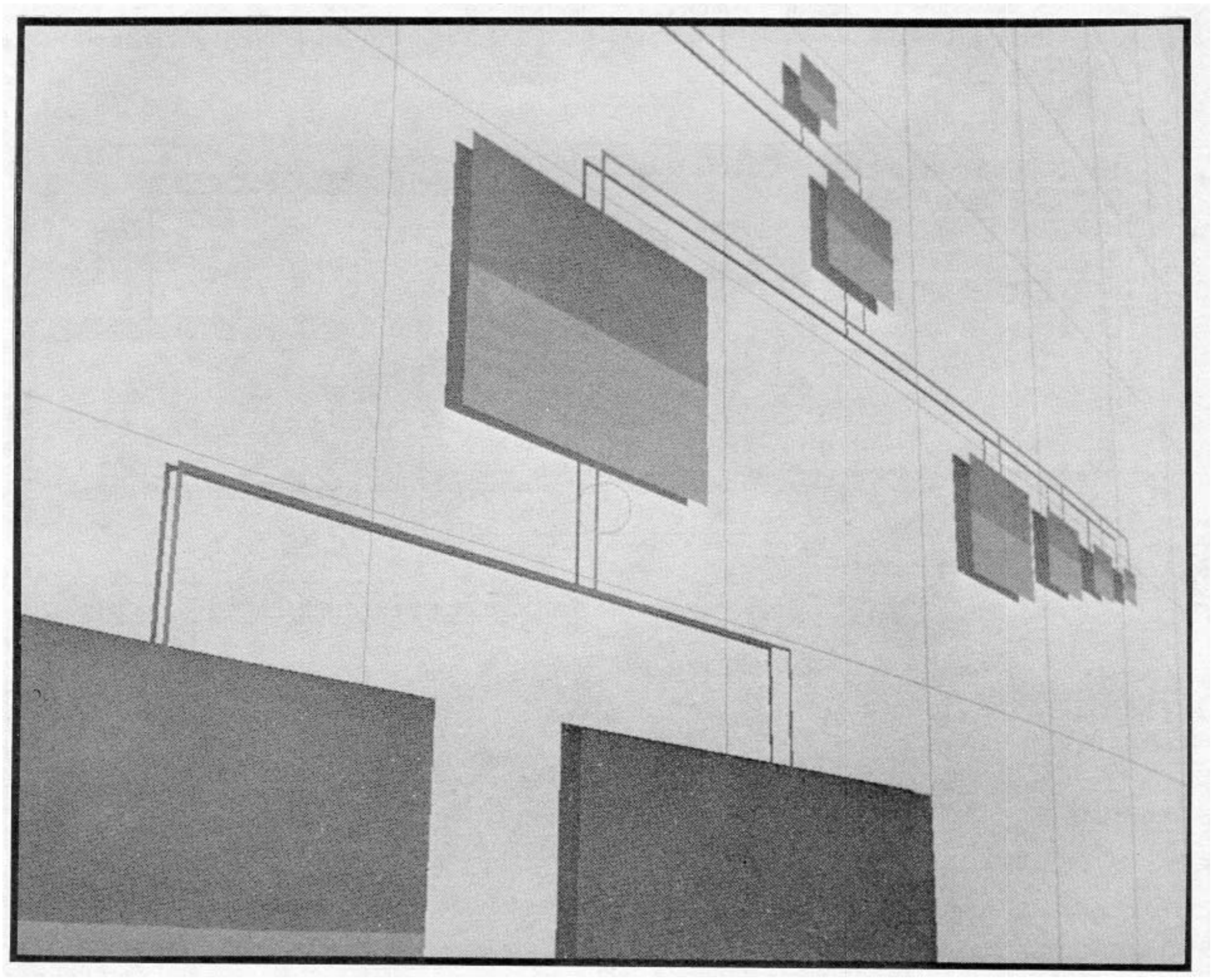

Figure 15. Experimental DandelrIs browser for organization charts. The photograph shows a view of a fragment of an organization chart as seen by a user flying around this chart. The chart is laid out on a simulated 2-dimensional whiteboard. The user can change his location in simulated 3-dimensional space, his direction of gaze, and by pointing with the mouse to indicate focus, the relative sizes of different organization chart boxes. (The experimental program was written by S. Card, A. Henderson, L. Lovstrand, and B. Verner.)

organization chart from an experimental browsing system we are implementing (the DandeIris, combining a Xerox DandeTiger Lisp machine and a Silicon Graphics Iris graphics processor). The node that is the user's focus is largest and is readable in the most detail. Nodes become smaller with distance from the focus. The user can fly around this chart to see it from different points of view and change his direction of gaze; if he points at a node, indicating a new focus of attention, it grows larger. The transitions are all animated in real-time, color, three-dimensional representations to help the user keep track of the transformed identities of various objects.

Such a display is designed to help the user navigate among a large number of objects; only a few of these objects would be visible in detail at any time, but many more would be visible as orienting cues and retrieval keys. The visual movement techniques is coupled with other retrieval techniques for allowing the user to focus on a limited number of items at a time (as required by our understanding of user's processing capacity) while retaining rapid access to a large number of items (as required by our understanding of the requirements of complex intellectual tasks). By coupling experimental programming with the analysis of user behavior, we hope to find theory-based techniques to aid the user in keeping track of and maintaining effective browsing and retrieval capability for large information structures.

6. Conclusions

What has been learned about users and how does this relate to the original visions of personal workstations? The short answer is that, unlike the early 1970's when little was known scientifically about computer users, we now have a vision of the form of an applied science of the user and a few areas of knowledge where that vision has been realized. In this paper we have narrated some of our efforts at building this applied science:

At the physical interface level, we have discovered that user performance with pointing devices is constrained by the information-processing capacity of the user. We have learned the quantitative law describing this constraint and have determined that certain devices, such as the mouse, are at the performance limits allowed by this law.

At the cognitive level, we have learned that routine human-computer interaction, such as text-editing, does not involve problem solving, but rather cognitive skills based on the execution of known methods. We can see the information-processing constraints of the user show through this skilled performance in the unit task of users. We have characterized cognitive skill to the extent of developing an engineering model for use by user-interface designers.

At the conceptual level, we have learned that users often have mental models of the systems they use, and that such models enable performance of novel tasks. System conceptual models provide the basis for users to acquire mental models, and thus are an important basis for system design. But mental models are cognitively intensive, and users will avoid them by attempting to map directly from their tasks to the actions required in a system. Theories of task mapping are just beginning to emerge.

At each of the above levels, it has been possible to base an applied science on a theory of the cognitive mechanisms underlying user behavior. At each level, the applied science was shown to be practical, in particular by being influential in the design of the Xerox Star.

The challenge for us today is to use our understanding of users to discover new ways to augment users in complex intellectual endeavors. Thus, we are concentrating our efforts in those areas where users' cognitive limitations need to be overcome and where users' cognitive abilities can be aided with computer-based tools. We believe that the most interesting problems are at the task level: understanding the nature of complex intellectual tasks and finding ways to build idea-structuring tools, both representation tools for structuring ideas and display tools for browsing ideas.

Whereas we once studied users empirically using existing systems in order to understand the nature of human-computer interaction, we now use what we have learned about users to help drive the creation of new experimental systems.

REFERENCES

Anderson, J. R. (1981). *Cognitive skills and their acquisition.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Bell, C. G., & Newell, A. (1971). *Computer structures: readings and examples.* New York: McGraw-Hill.

Bush, V. (1945). As we may think. *The Atlantic Monthly,* August.

Card, S. K. (1978). Studies in the psychology of computer text editing systems. Unpublished doctoral dissertation, Carnegie-Mellon University, Department of Psychology.

Card, S. K., English, W. K., & Burr, B. J. (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics, 21,* 601-613.

Card, S. K., Moran, T. P., & Newell, A. (1976). *The manuscript editing task: A routine cognitive skill* (Technical Report SSL-76-8). Palo Alto, CA: Xerox Palo Alto Research Center.

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

Card, S. K., Moran, T. P., & Newell, A. (1980b). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM 23,* 396-410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Card, S. K., Pavel, M., & Farrell, J. E.. (1984). Window-based computer dialogues. *Proceedings of IFIP Interact '84,* 355-359. London: IFIP.

Engelbart, D. (1963). A conceptual framework for the augmentation of man's intellect. In P. W. Howerton and D. C. Weeks (Eds.), *Vistas in Information Handling,* Vol. 1. Washington, D.C.: Spartan Books.

English, W. K., Engelbart, D. C., & Berman, M. A. (1967). Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics, HFE-8,* 5-15.

Halasz, F. G. (1984). *Mental models and problem solving in using a calculator.* Unpublished Ph.D. Dissertation. Stanford, CA: Stanford University.

Halasz, F. G., & Moran, T. P. (1983). Mental models and problem solving in using a calculator. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM.

Irby, C., Bergsteinsson, L., Moran, T. P., Newman, W., & Tesler, L. (1977). *A methodology for user interface design.* Palo Alto, CA: Xerox Palo Alto Research Center.

Kay, A. (1977). Microelectronics and the personal computer. *Scientific American,* September, 230-244.

Licklider, J. C. R. (1960). Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics, HFE-1* (March), 4-11.

Moran, T. P. (1978). *Introduction to the Command Language Grammar* (Technical Report SSL-78-3). Palo Alto, CA: Xerox Palo Alto Research Center.

Moran, T. P. (1981). The command language grammar: A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies, 15,* 3-50.

Moran, T. P. (1983). Getting into a system: external-internal task mapping analysis. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Newman, W., & Sproull, R. (1979). *Principles of interactive computer graphics,* 2nd ed. New York: McGraw-Hill.

Payne, S. J. (1984). Task-action grammars. *Proceedings of Interact '84: First IFIP Conference on Human-Computer Interaction,* London. Amsterdam: Elsevier.

Young, R. M. (1981). The machine inside the machine: users' models of pocket calculators. *International Journal of Man-Machine Studies, 15,* 51-86.

Young, R. M. (1983). Surrogates and mappings: two kinds of conceptual models for interactive devices. In D. Gentner & A. L. Stevens (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum Associates.