HUMAN-COMPUTER INTERACTION, 1990, Volume 5, pp. 221-265

The Growth of Cognitive Modeling in Human-Computer Interaction Since GOMS

Judith Reitman Olson and **Gary M. Olson**
The University of Michigan

ABSTRACT

The purpose of this article is to review where we stand with regard to modeling the kind of cognition involved in human-computer interaction. Card, Moran, and Newell's pioneering work on cognitive engineering models and explicit analyses of the knowledge people need to perform a procedure was a significant advance from the kind of modeling cognitive psychology offered at the time. Since then, coordinated bodies of research have both confirmed the basic set of parameters and advanced the number of parameters that account for the time of certain component activities. Formal modeling in grammars and production systems has provided an account for error production in some cases, as well as a basis for calculating how long a system will take to learn and how much savings there is from previous learning. Recently, we were given a new tool for modeling nonsequential component processes, adapting the "critical path analysis" from engineering to the specification of interacting processes and their consequent durations.

Though these advances have helped, there are still significant gaps in our understanding of the whole process of interacting with computers. The cumulative nature of this empirical body and its associated modeling framework has further highlighted important issues central to research in cognitive

Authors' present address: Judith Reitman Olson and Gary M. Olson, Cognitive Science and Machine Intelligence Laboratory, The University of Michigan, 701 Tappan, Ann Arbor, MI 48109-1234.

CONTENTS

psychology: how people move smoothly between skilled performance and problem solving, how people learn, how to design for consistent user interfaces, how people produce and manage errors, how we interpret visual displays for meaning, and what processes run concurrently and which depend on the completion of prior processes.

In the bigger picture, cognitive modeling is a method that is useful in both initial design (it can narrow the design space and provide early analyses of design alternatives), evaluation, and training. But it does not extend to broader aspects of the context in which people use computers, partly because

there are significant gaps in contemporary cognitive theory to inform the modeling and partly because it is the wrong form of model for certain kinds of more global questions in human-computer interaction. Notably, it fails to capture the user's fatigue, individual differences, or mental workload. And it is not the type of model that will aid the designer in designating the set of functions the software ought to contain, to assess the user's judgment of the acceptability of the software, or the change that could be expected in work life and the organization in which this work and person fits. Clearly, these kinds of considerations require modeling and tools of a different granularity and form.

1. GOMS AS COGNITIVE MODELING

The ability to predict how users will interact with proposed designs is a useful tool for the system designer. Being able to make such predictions is one of the principal goals of a class of cognitive models that has emerged following on the work of Card, Moran, and Newell (1980a, 1980b, 1983). In this article, we examine the strengths and weaknesses of such models, focusing in particular on confirmations and extensions that have emerged since the original proposals. New work has addressed some of the well-known weaknesses of the original approach, but it is still important to understand exactly the limits of these kinds of models.

In very few design fields does the process of design proceed from first principles. Rather, new designs most often arise from old designs, from analogies, or from other sources of creative thought. First principles are then used to screen these candidate designs. This is exactly the role for which cognitive models are best suited (Newell & Card, 1985, 1986). To be concrete, cognitive models are useful in:

1. Initially constraining the design space, so that one does not build an interface, for example, that requires more items to be kept in memory than will fit in working memory (WM).
2. Answering specific design decisions, so that one can decide, for example, between a dialogue that requires few keystrokes but difficult retrieval from memory or one that involves more keystrokes but is easier to remember.
3. Estimating the total time for task performance with sufficient accuracy to make decisions about how many people are needed to staff the performance of a repetitive operational task on a computer.
4. Providing the base from which both to calculate training time and to guide training documentation to help the user determine in which situations which method is most efficient.

5. Knowing which stages of activity take the longest time or produce the most errors, in directing research toward the aspects of human-computer interaction that will have strong future performance implications.

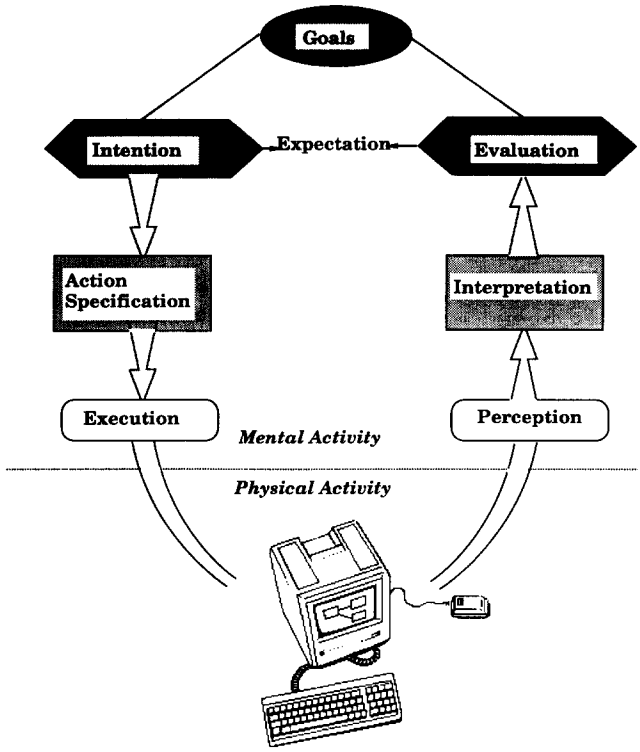
Card et al. (1980a, 1980b, 1983) proposed a framework for building such analytic models of human performance with computers. This framework represented a significant advance from modeling in cognitive psychology in that many of the processes contributing to the full cycle of perception to action were modeled together and the knowledge that is considered necessary to perform a task was described in enough detail to generate predictions about human behavior in real, naturalistic tasks.

The framework has two key components. The first is a general characterization of the human information-processing system, in terms of both a system architecture and quantitative parameters of component performance. This they called the *Model Human Processor* (MHP), and it summarized a large body of research from cognitive psychology. The second key component is a way of describing what the user needs to know in order to perform computer-based tasks, a model called *GOMS*. The GOMS model—actually a family of models—describes the knowledge necessary and the four cognitive components of skilled performance in tasks: goals, operators, methods, and selection rules.

The GOMS framework provided Card et al. with the basis for predicting the methods and operators users would follow in carrying out a particular well-known task (the goal) and, given a method, how long that task would take. Figure 1 shows the steps a user proceeds through in using a piece of software (adapted from Norman, 1986). The user perceives activity on the screen, evaluates whether it is what is expected given the goals the user is trying to accomplish, sets up an intention of the next step, retrieves the way to enact this intent on this system, and executes the appropriate motor movements. This produces new activity on the screen, and the user cycles through the process again. The original GOMS framework focused on explaining the selection from memory of methods appropriate to the situation (the goal and intention phases of Figure 1) and the time to specify and to execute the action.

The strength of this approach is its ability to predict the time it takes a skilled user to execute a task based on the composite of actions of retrieving plans from long-term memory, choosing among alternative available methods depending on features of the task at hand, keeping track of what has been done and what needs to be done, and executing the motor movements necessary for the keyboard and mouse. To make useful predictions, GOMS assumes that routine cognitive skills can be described as a serial sequence of cognitive operations and motor activities. Each of these actions has a time parameter that is independent of the particular context within a task and is

Figure 1. **The seven steps of user activities involved in the performance of a computer-based task (based on Norman, 1986).**



constant across tasks. Card et al. (1983) proposed such a theory, more constrained than general cognitive theories, in order to make engineering calculations.

Time parameters for external actions (and one internal action, that of retrieving the next unit of a plan) were estimated from empirical data derived from people using text editors, graphics systems, and some functions from the operating system of a variety of software. These numbers were generally obtained from a regression model, in which each unit task was assumed to consist of a set of keystrokes, hand movements, mental retrievals, and the like. Over a wide set of unit tasks from a variety of systems, values were obtained for these component processes. In later work, times for new parameters were derived in a similar manner except by Olson and Nilsen (1988). In their work, they recorded the time for every external act in a "keystroke capture" program, not just the total time for a task. Their regression analyses, therefore, included many more data points, each of smaller size, with each data point being a composite of a set of parameters relevant to that individual moment, not a sum over many different acts.

Card et al. (1983) found parameters that were very consistent across tasks. Of note were:

1. A keystroke, called **k,** for a midskilled typist is **280 msec.**
2. A single mental operator, called **M,** often interpreted as the time to retrieve the next chunk of information from long-term memory into WM is **1.35 s.**
3. Pointing, called **P,** to a target on a small display with a mouse takes on average **1.1 sec** (though the time is variable according to Fitts's law).
4. Moving the hands, called **H,** from the keyboard to the mouse takes **400 msec.**

An analyst or designer using these parameters to predict how long a particular task would take was given heuristics about where in the task retrieval of the next unit theoretically took place (where to put the **M**s) and when to insert the amount of time the system takes to respond. For example, retrieval (**M**) occurs at each unit boundary (a word, a single symbol or function key, a string of movements with a step key). Accordingly, if a subject were going to enter a formula in a spreadsheet by pointing with cursor-step keys to the cells that contain the desired values, a formula for adding the contents of cells from B22 to B29 would include:

@ s u m (^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ · ^ ^ ^ ^ ^ ^)<ret>
MK Mk k k Mk Mk k k k k k k k k k k Mk Mk k k k k k Mk Mk

which adds up to:

8 **M**s and 26 **k**s
8(1.35) + 26(.280) = 18.08 s.

Using the small set of parameters with associated times derived from a large set of tasks, they were able to account for 90% of the variance in predicting unit task times such as the foregoing (Card et al., 1983, p. 294).

1.1. Limitations of the GOMS Approach

From the outset, it was clear that the GOMS approach to user modeling had well-defined strengths and weaknesses. Various GOMS models, particularly the keystroke-level model, provided good quantitative fits to the performance times of skilled users during errorless performance. Indeed, the parameters were stable enough so that performance in appropriate new situations could be predicted without the need to estimate parameters from

the data. This ability to derive parameter-free estimates is part of what makes the GOMS approach useful in design, because it allows comparisons of different design alternatives. This is the key feature of the engineering approach that the GOMS models reflect.

One potential limitation of the GOMS model is that the initial work reported by Card et al. was carried out in a limited range of domains. There is always the question of whether the framework holds up when it is taken into new domains.

But there are more serious questions about the value of GOMS. Although there are many critiques of GOMS (e.g., Carroll & Campbell, 1986; Karat, 1988; Wilson, Barnard, Green, & MacLean, 1988), the most explicitly detailed list came from Card et al. (1980) themselves in their original framing of the model. A compilation of these shortcomings includes:

1. The model applied to skilled users, not to beginners or intermediates. Such *nonskilled users* spend considerable time engaged in problem-solving activities, rather than simply retrieving and executing plans, and move smoothly between problem solving and skilled behavior.
2. The model gave an account of skilled performance at asymptote but no account of either *learning* of the system or its *recall* after a period of disuse, nor how to design an easily learned consistent interface.
3. The model focused on errorless performance and, thus, gave no account of the *errors* that frequently occur even in skilled performance.
4. The model was most explicit about elementary perceptual and motor components of skilled behavior but tended to treat the *cognitive processes* in skilled behavior in a less differentiated fashion.
5. The model was developed exclusively for tasks in which the principal components that were being modeled could reasonably be assumed to be serial in nature. However, tasks have a substantial number of component processes that, at some level, must occur in *parallel.*
6. The model does not address *mental workload*—how much must be held in mind while using the system.
7. The model addresses only the usability of a task on a system and does not address *functionality,* that is, what tasks should be performed by the computer.
8. The model does not address the amount and kind of *fatigue* users experience using a system.
9. The model does not account for *individual differences* among users.
10. The model does not provide guidance in predicting whether users will judge the system to be either useful or satisfying, or whether the system will be globally *acceptable.*
11. The model stops short of addressing any aspects of how computer-supported work fits or misfits office or organizational life.

It is useful to ask which of these objectives have been addressed in subsequent work, which remain possible but unexplored, and which seem entirely beyond the scope of even an extended GOMS model.

1.2. Plan of This Article

The purpose of this article is to review these advances in cognitive modeling and to outline the significant problems that remain for both the designer in search of guidance and the cognitive psychologist interested in how cognitive processes interact to produce behavior of the sort exhibited in human-computer interaction. We do this by addressing the three main points.

First, we review the extent to which the quantitative estimates of task components described by Card et al. (1983) have held up in further research. Subsequent studies of skilled performance in several different domains have provided strong confirmation for the original work.

Second, a number of investigators have taken the Card et al. (1983) framework in new directions. We examine three in particular: the study of learning and transfer, the study of errors, and the analysis of parallel processes. In each case, the basic Card et al. framework is preserved, but additional value is added.

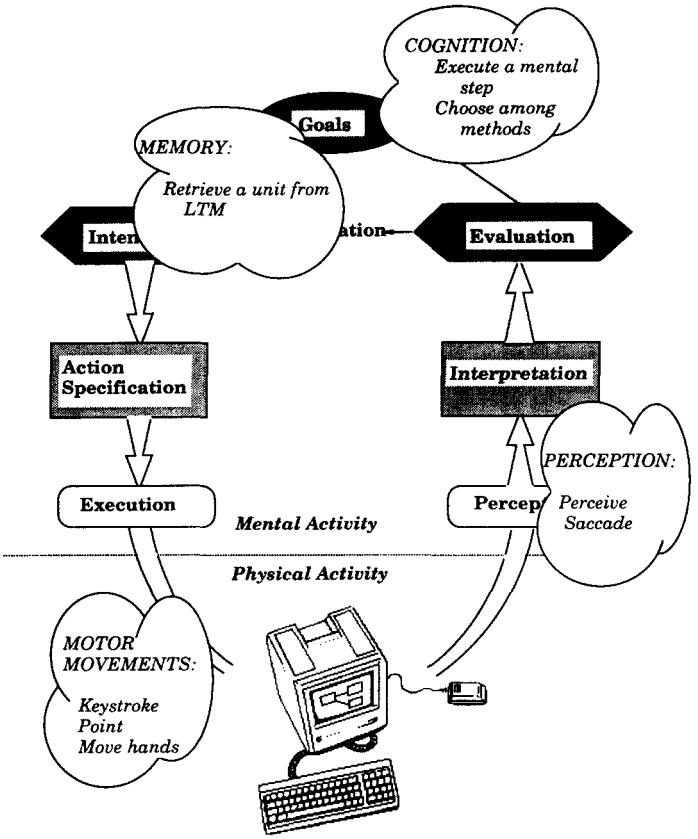
Third, a number of limitations remain, either because cognitive science currently provides no guidance to this aspect of human-computer interaction or because the questions require a wholly different modeling approach. We close with an examination of the place of cognitive modeling in the broader context of research on human-computer interaction.

2. ADVANCES IN MODELING SPECIFIC SERIAL COMPONENTS

A number of researchers have followed the spirit of GOMS and the MHP in empirical explorations. These investigations are direct tests of both the assumption of serial processing and the assumption that time parameters are constant across a wide range of tasks.

We have gathered the different parameters that various researchers have found; displayed them in Figures 3, 5, and 6; and summarized them in Figure 7. Each figure displays parameters as found in Card et al.'s (1983) empirical work and in their review of the relevant cognitive psychology literature as summarized in the MHP. Added to these parameters are values found in studies with a similar spirit: studies of entering editor commands with keystroke codes, entering formulas in spreadsheets, and so on. The new parameters are listed in italics in these three figures.

Figure 2. **The seven steps of user activities involved in the performance of a computer-based task, annotated with known cognitive processes.**



The parameters cluster into three general classes: motor movement, perception, and memory and cognition. These general classes of behavior map onto the seven steps of user activity as shown in Figure 2.

2.1. Motor Movements

Figure 3 summarizes the values found in various studies relevant to keyboard entry, using a mouse, and moving hands back and forth from keyboard to pointing devices.

Keying

A set of parameters accounts for the time to enter a keystroke in a normal typing task, the actual value depending on the skill level of the typist, the

Figure 3. **Parameters describing component processes in motor movement.**

Enter a keystroke		
Average non-secty typist	280 msec	CMN
Best typist (120 wpm)	80 msec	CMN
Good typist	120 msec	CMN
Average skilled typist (60 wpm)	200 msec	CMN
Typing random letters	500 msec	CMN
Typing complex codes	750 msec	CMN
Worst typist	1200 msec	CMN
Entering spreadsheet formulas		
Lotus	*330 msec*	*O&N*
Multiplan	*220 msec*	*O&N*
Entering column/width commands		
Lotus	*280 msec*	*O&N*
Multiplan	*230 msec*	*O&N*
Enter command abbreviations	*230 msec*	*J&N*
Expert typing cross-hand digraphs	*170 msec*	*J&N*
Expert typing same-hand digraphs	*220 msec*	*J&N*
Point with a mouse		
Average value, small screen, menu shaped target	1100 msec	CMN
Varies with distance and size of target	$1.0 + .10 \log_2(D/S+.5)$sec	CMN
Average value, small distance, menu target	*1900 msec*	*WSN*
Varies with distance and size of target	$.80 + .23 \log_2(D/S+.5)$sec	*WSN*
Move hands from keyboard to pointing device or back		
To mouse	360 msec	CMN
To joystick	260 msec	CMN
To cursor(arrow) keys	210 msec	CMN
To function keys	320 msec	CMN

CMN = Card, Moran, and Newell, 1983
O&N = Olson and Nilsen, 1988
J&N = John and Newell, 1989
WSN = Walker, Smelcer, and Nilsen, 1988

frequency with which the particular key is used, and the predictability and continuity of the text to be typed. That is, regular transcription typing is faster than entering formulas, for example, because numerals are less frequently typed and often the placement of the number and symbol keys requires perceptual search in addition to simple hand motor movements. Card et al. (1983) reported parameters ranging from 80 msec per keystroke for a typist of 135 words per minute (wpm) to 1200 msec per key for a user unfamiliar with the keyboard. An average typist is reported to take 280 msec per keystroke.

Two other researchers confirmed the basic keystroke parameters. Olson and Nilsen (1988) found two separate keystroke times from their examination of the moment by moment activity in entering and changing spreadsheets. They found a keystroke time of 330 and 220 msec for entering formulas in Lotus and Multiplan, respectively, and 280 and 230 msec for entering keystrokes in a task that changes the widths of columns in Lotus and Multiplan, respectively.

In their investigations of users entering command abbreviations, John and her colleagues (John & Newell, 1987, 1989, in press; John, Rosenbloom, & Newell, 1985) found keystrokes to be 230 msec in one, and 269 msec in the other. A more recent investigation (John & Newell, 1989) found that in tasks involving transcription typing, each keystroke was heavily dependent on skill level and task, with values ranging from 70 msec to 220 msec per key. These times correspond to the 200 msec for an average typist (60 wpm) from Card et al. (1983) if we assume a 50–50 distribution of crosshand, same-hand pairs:

$$(170 + 220)/2 = 195 \text{ msec.}$$

All these values are very close to the middle of the range of values designated by Card et al. (1983) and slightly smaller than the value designated in their work to represent typing random letters. These tasks, however, are much more predictable than typing random letters and are often performed by more skilled users.

Moving a Mouse

Pointing with a mouse at objects whose distances and target sizes covered a variety of screens requires an average 1100 msec per selection, according to Card et al. (1983). Although this value is appropriate for standard interfaces, such as choosing a ⅜-in. sized menu item on an 8½-in. diagonal screen (as in a Macintosh SE), it is known to vary with the distance of the movement and the size of the target, following Fitts's law (Fitts & Peterson, 1964). Because screens now regularly exceed the standard 8½ in., and the user more often needs to point to scroll bars, window sizers, graphic-object "handles," and

icons that are both small and widely scattered on big screens, more detail is needed to assess this movement time accurately.

Card et al. (1983) explored the relationship between target size and distance travelled in mouse movements, empirically determining the constants in Fitts's law:

$$T = 1.03 + .096\log_2(D/S + .5).$$

The form of this relationship is interesting. There is a large constant time of 1 s to begin moving, no matter what the distance. Adjustments to this base time are in increments of 100 msec each time the distance traveled is 1½ times as long as the size of the target. That is, this formula suggests that it takes about 300 msec to reach the top of the screen from the middle[1] and 100 msec to travel about ½ in. (to the second menu item) on the Macintosh to hit the ⅜-in. menu item, about 1400 msec total.

Walker, Smelcer, and Nilsen (1989) explored the variations in this relationship to predict the time to choose items from nested menus, menus that present a second array of choices after the user chooses from a first menu, usually in side-by-side strips. Their empirical evaluation determined that the time to move to a target on the menu did indeed follow Fitts's law:

$$T = .81 + .23\log_2(D/S + .5).$$

Although the initial movement time of 810 msec is close to the 1030 msec of Card et al. (1983), the increment of 230 msec per unit distance and target size is twice that found earlier. This difference is likely accounted for by the fact the distance in the nested menus they used included a composite of the user selecting the first item with a single downward motion, a turn to the right in the item's slot, and a move to the edge, where the second set of items is displayed. Thus, the distances used in the Walker et al. (1989) study include the turns, as well as horizontal and vertical distance segments. The time is slower because it includes other, more subtle component processes.

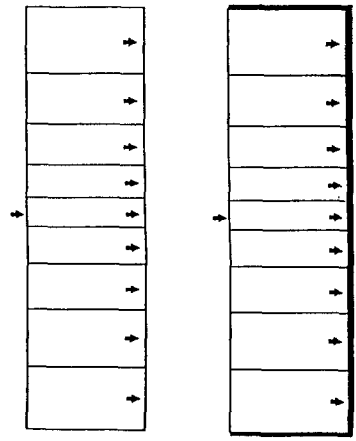
An Example of the Application of GOMS and MHP to Design Generation

Walker et al. (1989) went a step further, using these facts to drive a design process, by demonstrating how GOMS can constrain a designer's search in such a way as to lead to large improvements in performance.

With the goal of shortening this total menu-selection time, Walker et al. made three adjustments to the design of menus. One redesign shortened the total distance the user had to travel to select an item by making the menu "pop

[1] But see the discussion of borders, later in this section.

Figure 4. **Fittsized menus: Menus whose target areas grow with the distance the user has to travel to get to them. In these particular menus, the cursor begins in the middle-left of the menu. On the right panel is the menu with "borders" on the top, right, and bottom edge, boundaries beyond which the cursor cannot move.**



up" to the right of the cursor instead of below it. Though the average travel distance in this design would be halved, there was question whether the user might experience additional delays due to the uncertainty about which direction the movement had to be launched. The results showed that the time to select an item was far shorter when the menu popped up to the right of the cursor than when the menu appeared below it. The time to travel to the target was a much larger determinant of the total time than the small increment of time to decide whether to begin to move up or down.

Two new menu types followed, both intended to reduce the selection time by making the target bigger. Menus can be constructed so that the target size grows as the distance from the cursor's starting position increases. These are called "Fittsized menus" displayed on the left of Figure 4. In this menu, the further the user has to travel, the bigger the target. The mean reduction in this study, however, was only 80 msec, a small effect. Fitts's law, however, tells us to expect this small effect. It is a complicated arrangement, because as the targets grow, so does the overall distance to be traveled. But, in general, we would have to increase the size of the furthest target by a factor of nearly 3 to reduce the travel time over 100 msec, for example, making the fourth target item 1 in. tall if the first target item is the standard Macintosh ⅜ in.

In the third design on the quest for shorter selection times, Walker et al. (1989) made the target size effectively much bigger by putting a virtual border on the top, right, and bottom edges of the pop-up menu, a border beyond which the cursor could not move even though the mouse moved. This is similar to the border that the Macintosh interface has at the top of the screen.

The cursor will not move off the screen, even though the user has moved the mouse to a place that would translate beyond the screen edge if it were unconstrained. In this situation, shown on the right-hand side of Figure 4, the target size is effectively very large and much easier to hit. This kind of menu reduced the selection time by a large amount. The average selection time over all menu positions in the original design was 1.9 s. Adding the borders reduced the average time by 450 msec, a reduction of nearly 25%.

Hand Movements

Hand movement time is the time needed to move from the space bar of the keyboard until the pointing control begins to move the cursor. This is a large-muscle movement also well characterized by Fitts's law; the time to reach a target (in this case, the mouse or joystick) is a function of the distance to the target and its size. This time was calculated from behavioral data to be approximately 360 msec, a relatively small, fixed amount of time because the mouse is typically in about the same position each time, and the target itself is large (see Card et al., 1983, Figure 7.4). This is a much smaller time than pointing with the mouse to a specific intercharacter location on the screen because of the large effect of size of the target on the overall time.

Card et al. (1983) found empirical variation among parameters for different pointing devices, presumably because they were at various distances from the home position on the keyboard and because the targets are of different sizes. The user took 360 msec to find a mouse and start to move, 260 msec to find the joystick (because it is bigger), 210 msec to find the cursor or arrow keys (both nearer and smaller), and 320 msec to find the function keys (presumably because they are a few inches away but relatively small).

2.2. Perception

Summary values for the known aspects of seeing items on the screen are summarized in Figure 5. These values are primarily from the summary of the literature included in Card et al.'s (1983) MHP, although John and Newell (1989, in press) add some recent empirical support for these values.

Clearly, by recognizing features of the current task and assessing some of the parameters necessary to do a task (e.g., the letters to be edited or components of the formula to be entered), perception and scanning are involved. The perceptual processor is clocked at **100 msec** in the summary of the MHP in Card et al. (1983, Figure 2.1) and a saccade (the time to move and take in information in each jump) at **230 msec.** Although neither perception nor scanning parameters are included in the empirical work of Card et al. (1983), John and Newell (in press) used the 100 msec for the perception of lights and simple symbols in a stimulus–response compatibility

Figure 5. **Parameters describing component processes in perception.**

Recognize or perceive

Time to respond to brief light	100 msec	MHP
It varies with intensity, brighter is faster	50 -- 200 msec	MHP
Recognize a 6-letter word	*340 msec*	*J&N*
	314 msec	*J&N*

Make a saccade

Time for the eye to jump to next location	230 msec	MHP

MHP = Model Human Processor from Card, Moran, and Newell, 1983
J&N = John and Newell, in press and 1989

task. When they combined it with a cognitive operator (discussed later) and standard information theory to calculate how many bits of information had to be processed by the cognitive processor, they obtained very good predictions. This is not a direct derivation of the perceptual parameter, but indirect support for its value.

Similar in style, John and Newell (1989, in press) determined that perceiving a six-letter word took **314 msec** in a stimulus-response compatibility task that involved entering abbreviations for command names. Because perception of words includes recognition, some verbal encoding, or retrieval of meaning in addition to simple perception, it is not surprising to find this value a bit higher than the others.

Let us consider an example of application of the theory. A related "scanning" parameter was identified in the Olson and Nilsen (1988) study of experts using spreadsheet software. The Olson and Nilsen scanning parameter was identified in situations in which the user was scanning a screen for additional information, the row and column coordinates on the spreadsheet border, in known locations. This scanning took an additional 2300 msec. Why is this so much longer than the perception values discussed earlier? Obviously, from the discrepancy in not just the value but the order of magnitude of these parameters, something more than simple perception is likely to be operating.

In the Olson and Nilsen task, the user must look for the cell addresses by

following the row or column guide lines to the border where the row or cell indicators are displayed (e.g., *B,* on the column indicator and *22* on the row indicator) and then retrieve them for execution as keystrokes. The total time for this composite of scanning, storing, and retrieving was measurred at **2300 msec.** If we use the available parameters for a saccade and memory retrieval (in Figure 3), then we can calculate a reasonable storage time (one for the *B,* one for the *22*) to be 130 msec, close to the parametric value to perceive something. In more detail, if the 2300 msec is a composite of:

A saccade to the row line	230 msec
A storage of the row label	*130 msec*
A saccade to the column head	230 msec
A storage of the column label	*130 msec*
A saccade to the cell in which typing is to start	230 msec
Retrieval of the row and column labels	1350 msec
Total	2300 msec

where the 130 msec for storing a row label in working memory is the *derived* value from this calculation.

These parameter estimates, of course, warrant further empirical verification. But it is intriguing that this task can so nicely be decomposed into reasonable processes with time parameters from cognitive engineering.

This example illustrates the use of GOMS and MHP in understanding a design issue. The parameter Olson and Nilsen (1988) found was an order of magnitude too large; GOMS identifies this as an issue of grain size of task. It encourages closer examination of a linear sequence of more microscopic operations identified in previous research. Here, the result was that the 2.3 s time was not simple perception, but rather a series including search, storage, and recall, the last of which is known to have an associated long time. This might lead the designer from a solution that makes scanning per se shorter to one that eliminates the need for users to store and recall coordinate values.

2.3. Memory and Cognitive Processes

Figure 6 summarizes the values found in various studies relevant to memory retrieval, executing steps in a mental procedure, and choosing among methods.

Memory Retrieval

At the heart of the Card et al. (1983) model is *M,* called "mental," often interpreted as the time to retrieve the next unit of information. It is assumed from the listed heuristics about when this special operation occurs that this is

Figure 6. **Parameters describing component processes in memory and cognition.**

Retrieve a unit from LTM to WM

Retrieve a command name or delimiter	1350 msec	CMN
Retreive a random command abbreviation		
	1200 msec	*J&N*
	1209 msec	*J&N*
	1200 msec	*J&N*
Retrieve the next part of a formula		
Multiplan (cursor method)	*1100 msec*	*O&N*
Lotus (cursor method)	*990 msec*	*O&N*
Lotus (typing method)	*1350 msec*	*O&N*
Retrieve command part in column width task		
Multiplan	*1160 msec*	*O&N*
Lotus	*1080 msec*	*O&N*
Repeated retrieval of same command		
Lotus	*660 msec*	*O&N*

Execute a mental step

Cognitive Processor (the contents of WM initiate associatively-linked actions in LTM)	70 msec	MHP
Execute next rule in a formal model of skilled performance	*100 msec*	*BKP*
Execute next step in decoding abbreviations		
	66 msec	*J&N*
	60 msec	*J&N*
	50 msec	*J&N*

Choose among methods

"Mental" operator, choose, retrieve	620 msec	CMN
Choose whether to type or point to cells in entering a spreadsheet formula	*1760 msec*	*O&N*

MHP = Model Human Processor from Card, Moran, and Newell, 1983
CMN = Card, Moran, and Newell, 1983
BKP = Bovair, Kieras and Polson, 1985
J&N = John and Newell, 1985, 1989,1989, in press.
O&N = Olson and Nilsen, 1988

retrieval of well-known units from long-term memory (LTM) for placement in WM, ready then to be either executed by a motor processor or further decomposed by subsequent retrieval from LTM. This value is empirically determined to be **1350 msec**.

John and Newell (1987, 1989, in press) and John et al. (1985) empirically determined a parameter for retrieval from LTM to be **1200 msec**. This is described as being retrieval of a completely arbitrary association between a stimulus word and its required letter-combination response. Olson and Nilsen (1988), in their study of spreadsheet software use, found a parametric value for retrieval of syntax parts in entering a formula as **1100 msec** in Multiplan and **1350 msec** in Lotus 1-2-3. Retrieving the command parts (e.g., the keyboard equivalent for a command from a menu) in a column-width task was **1160 msec** in Multiplan and **1080 msec** in Lotus.

Olson and Nilsen found another interesting result in their study of Lotus 1-2-3. In their study, the user entered the column width command four times in a row. Lotus 1-2-3 has no feature that allows you to set the column width for a range of columns. One can set either the whole spreadsheet's default column width or one column at a time. Thus, the only way to set a range (which we asked them to do) is to repeat the same action once for each of the columns desired. Again the times for memory retrieval were calculated. The first time the user set the column width, the retrieval time was **1100 msec**. On the second, third, and fourth trials (repeated rapidly in succession), the retrieval times dropped and remained flat, calculated to be **660 msec**, half that of the first retrieval. This is not a simple practice effect; the repeated act speeded up only the memory access, not the keying times. The times for the keystrokes were calculated individually for each trial and found to be constant at 280 msec.

Executing Steps in a Task

GOMS provides an explicit representation of the mental steps involved in executing a task. It catalogues the retrieval of a goal and its subgoals, the decision to select a method to fit the particulars of the current situation, the retrieval of the motor movements necessary to execute the command, and the execution of each of those command components. Card et al. (1983) estimated from the psychological literature that the execution of each procedural step should take about **70 msec**.

Kieras and Polson (Kieras, 1988; Kieras & Bovair, 1986; Kieras & Polson, 1985; Polson & Kieras, 1985; Polson, Muncher, & Engelbeck, 1986) made the GOMS representation much more explicit by programming the procedures in production system formalism, discussed in more detail later in this article. This formalism allowed a number of predictions of behavior far wider than those covered in Card et al. (1983), many of which are mentioned in later sections. Of note here is that from empirical studies of people carrying out a

wide variety of procedures, they estimate that each production requires about **100 msec** to execute.

Similarly, John et al. (1985), John (1988), and John and Newell (1987, 1989, in press) determined from their studies of people entering command abbreviations that each mental step that translates the command word into an abbreviation takes **between 50 and 70 msec**. Even though these mental steps include a variety of processing acts, for example, storing notes in WM, calculating, and deciding, as well as very different specification languages that defined the cognitive steps, the finding across studies is remarkably consistent.

Choosing Among Methods

In the MHP, Card et al. (1983) assumed that the more choices for a response, the longer the expected response time. This is based on the work of Hick (1952) on choice reaction time tasks. Card et al. (1983) estimated that mental operators for choosing among appropriate methods is **620 msec**. Very little additional empirical validation followed this initial determination.

Olson and Nilsen (1988) found a much higher value for a similarly described process. When a system presents the user with a choice of methods, the user requires additional time to make the choice. In Lotus 1-2-3, the user has choices for indicating the elements of a formula—by typing in the cell coordinates (e.g., B22) or by pointing to the appropriate cell with the cursor, moved by step keys. In Multiplan, the contrasted piece of software in the study, there is only one way—pointing to the appropriate cell with the cursor using the step keys. Although the time to enter the formulas with the same method took exactly the same amount of time across the two packages, users of Lotus took an additional **1760 msec** before entering to decide which method to use. The time to start the formula in Lotus (which offered two methods) was 4.63 s, whereas in Multiplan (which offered only one method) it took only 2.87 s. In contrast, in another task, setting column widths, in which both Lotus and Multiplan offered only one method, the start times were identical.

Although the overall direction of this choice relationship and Hick's law are the same, the effects are a different order of magnitude. Hick's law predicts a set of simple, noncognitive reaction times in the range of 200 msec, similar in size to the MHP cognitive step. The time to choose among methods in Card et al. (1983) is 600 msec; in the study of spreadsheets, the time is on the order of 2 s. This difference suggests that a choice between methods in human-computer interaction is a more complex cognitive task, requiring several to many cognitive steps to be executed. These steps would differ from task to task. Also, some of the conditions on which methods are chosen require an estimate (e.g., of how far away certain values are), some

Figure 7. **Summary of cognitive engineering parameters derived as the median of values obtained in Figures 3, 5, and 6.**

Enter a keystroke	230 msec
Point with a mouse	1500 msec
Move hands to mouse	360 msec
Perceive	100 msec
Make a saccade	230 msec
Retrieve from memory	1200 msec
Execute a mental step	70 msec
Choose among methods	1250 msec

perception of values on the screen, or a quick examination of how immediately memorable some parameters are, all of which would add complex component times to this overall effect.

2.4. Predicting Composite Performance From These Parameters

The data from Figures 3, 5, and 6 are summarized in brief engineering form in Figure 7. For each of the components, we have taken as the single value the median of the values obtained in similar circumstances. For example, for the time to retrieve from memory, we take the median of the nine values in the top of Figure 6, resulting in a value of 1200 msec. The one exception to this is the value for moving hands is listed for the movement to a mouse, the most common motion in the applications we have analyzed.

Let us consider an example of using these parameters in a wholly new task. We use these times to analyze a very different task studied in a different laboratory to illustrate the generality of the parameters and the component processes involved in GOMS and MHP.

Young and MacLean (1988) measured the times of people entering a block of values in a spreadsheet two different ways. The user could either enter each value separately, pointing to the next cell with the mouse (called the *Mouse* method), or set up a procedure by which each Enter key would advance the cursor automatically into the cell to the right of the last one. In this second method (called the *Menu* method), the user then had to use the mouse only when the next line was started. They found that the Mouse method took 4.19 s for each cell to be entered; the Menu method took 2.81 s to start each line and 2.46 to type each cell's two-digit number with the Enter key advancing the location to the right.

Using parameters from Figure 7, we calculate that the Mouse method consists of:

Moving the hand to the mouse	360 msec
Clicking the mouse (same as a keystroke)	230 msec
Moving the hand to the keyboard	360 msec
Retrieving two digits	1200 msec
Typing two digits @230 each	460 msec
Retrieving the end action	1200 msec
Typing the <ret> key	230 msec
Total	4040 msec

This result compares well with the **4.19 s** obtained in the study, a 3% error.

We further assume that task of starting each new line in the Menu method involves:

Moving hand to mouse	360 msec
Pointing to a new line with mouse	1500 msec
Clicking the mouse	230 msec
Moving hand to keyboard	360 msec
Total	2450 msec

This result compares with the **2.81 s** found, with a 13% error.

And the task of typing each number into the cell in the Menu method involves:

Retrieving (or looking for) two digits	1200 msec
Typing two digits @230 msec each	460 msec
Retrieving the end action	1200 msec
Typing the <ret>	230 msec
Total	3090 msec

This result compares with **2.46 s** found in the empirical study, a 26% error.

These calculations could be challenged in a number of ways, each challenge focusing on the inclusion or exclusion of an operation, especially the unobservable mental ones. These challenges would be important if we were interested in the details of skilled performance in these specific tasks, and the challenges should be pursued when we are doing research on the discovery of new parameters. What is more important for cognitive engineering and its use in design, however, is that these calculated values are within an average of 14% error of the observed values, accurate enough at this level of analysis to be useful in the calculation of overall task times. They are certainly in the right order of magnitude, even within the 20% error criterion often acceptable in early design stages. These numbers are accurate enough, for example, for us to determine in a large task how many task units could be

performed each day by a dedicated clerk and, thus, how many clerks would have to be on staff to perform a work load of a target size.

In summary, although there are potential problems with the basic assumptions of GOMS and the MHP that the component processes are serial (discussed later in the Parallel Processes section) and that performance times in any one task are independent of context, these assumptions have served well in a variety of basic computer-based tasks. These tasks include remembering and entering different kinds of keyboard commands, using a text editor, manipulating files in an operating system, using graphics programs, and entering formulas and changing column widths in spreadsheet programs. All in all, within the limits imposed by the initial formulation, the model has done quite well in subsequent evaluations.

3. EXTENSIONS OF THE BASIC FRAMEWORK

Many critics have argued that the original ground rules set up for such models are overly restrictive and, thus, limit the usefulness of this approach for human–computer interaction applications. In this spirit, a number of investigators have tried to take this cognitive engineering approach beyond the scope of the original GOMS formulation. We now turn to these extensions, focusing on three areas: the analysis of learning and transfer, the analysis of errors, and the treatment of parallel processes.

The first two classes of extensions have been made possible by the explicit modeling of the user's knowledge of grammatical rules and of both knowledge and performance in terms of production systems.

Grammars. Reisner (1981, 1984) and Payne and Green (1986) used grammars to make explicit the knowledge a user must have in order to translate from goals to actions in a particular system. Grammatical rules are similar in spirit (though not form) to the goal decomposition and methods in GOMS. They are more a model of knowledge content (competence) than of a full system that can "run" to produce user performance (Green, Schiele, & Payne, 1988). In the spirit of cognitive engineering, however, these representations do provide a countable entity: the number of rules. And, in Payne and Green's (1986) Task-Action Grammar (TAG), there is an explicit aspect of the content that predicts learning: the relationship between the features encoded in the rules and the natural world associations of the user.

To illustrate what the elements of a grammar look like, we use the TAG notation for a small segment of a full grammar. TAG consists of commands, features of the goal (e.g., the direction of movement in which you want to move a cursor in text, like forward or backward), a "dictionary of tasks" (which shows the full set of actions covered and helps analysts count the coverage of each rule), and rules that translate goals into actions. Figure 8

Figure 8. **A portion of a task-action grammar for EMACS, for moving a cursor around text (adapted from Green, Schiele, & Payne, 1988).**

Commands:	
Goal	**Action**
Move cursor one character forward	cntl-C
Move cursor one character backward	meta-C
Move cursor one word forward	cntl-W
Move cursor one word backward	meta-W
Features, possible values:	
Direction forward, backward	
Unit character, word	
Rules	
Task[Direction,Unit]	→ Symbol[Direction] + Letter[unit]
Symbol[forward]	→ "cntl"
Symbol[backward]	→ "meta"
Letter[word]	→ "W"
Letter[character]	→ "C"

illustrates the commands, the features, and the rules that map a small set of goals of moving a cursor around text to the word processor EMACS. The small set of rules here says that if the task is to move in a direction a particular unit of jump, you should type a symbol for direction, where forward is <cntl> and backward is <meta>, and follow it with a typed letter for the unit, where a unit the size of a word is *W* and the size of a character is *C*. This illustrates the form of the rules, their size, and an example of a good real-world mapping in that the letter used for units the size of a character and word are their own first letters.

Production Systems. Kieras and Polson (Bovair, Kieras, & Polson, 1990; Kieras, 1988; Kieras & Bovair, 1986; Kieras & Polson, 1985; Polson & Kieras, 1985; Polson et al., 1986) have used production systems to represent the GOMS structure of task knowledge and significant aspects of the MHP (significantly, the goal stack and a WM for storing parameters). This substantially improves the original GOMS theory, because it, like the grammars, makes the underlying knowledge much more explicit. Further, although the production system may be difficult to write (Kieras, 1988), once it is specified, one can run the program to check it for completeness and accuracy. And, once programmed, one can quantify a number of features of the knowledge and processing to predict both errors and learning time behavior.

To illustrate what a production system formalism of GOMS looks like, Figure 9 presents a section of the production system for writing a database

Figure 9. **Five rules written in production system formalism. These rules help guide the behavior of a user in deciding to write a crucial join statement in a database query in the popular language, SQL.**

```
Rule 1:     (StartUp.SeeIfJoinNeeded
  IF        ((GOAL SeeIfJoinNeeded)
            (NOT(NOTE SeeingIfJoinNeeded TRUE))
  THEN      ((Add NOTE SeeingIfJoinNeeded TRUE)
            (Add STEP CountTables)))
Rule 2:     (CountTables
  IF        ((GOAL SeeIfJoinNeeded)
            (STEP CountTables))
  THEN      ((DoTask Count NumberOfTables *NumberOfTables)
            (Add NOTE Number OfTables *NumberOfTables)
            (Delete STEP CountTables)
            (Add STEP AddJoinNote)))
Rule 3:     (IfNumberOfTables=2ThenAddNOTEJoinAndCleanup
  IF        ((GOAL SeeIfJoinNeeded)
            (STEP AddJoinNote)
            (NOTE NumberOfTables 2))
  THEN      ((Add NOTE JoinNeeded TRUE)
            (Delete STEP AddJoinNote)
            (Delete NOTE NumberOfTables ?NumberOfTables)
            (Add STEP CleanUp)))
Rule 4:     (IfNumberOfTablesNot=2, ThenCleanup
  IF        ((GOAL SeeIfJoinNeeded)
            (STEP AddJoinNote)
            (NOTE NumberOfTables 1))
  THEN      ((Delete STEP AddJoinNote)
            (Delete NOTE NumberOfTables ?NumberOfTables)
            (Add STEP Cleanup)))
Rule 5:     (CleanUp.SeeIfJoinNeeded
  IF        ((GOAL SeeIfJoinNeeded)
            (STEP CleanUp))
  THEN      ((Delete NOTE SeeingIfJoinNeeded TRUE)
            (Delete STEP CleanUp)
            (Add NOTE SawIfJoinNeeded TRUE)))
```

query in SQL, a popular language that has become the de facto standard in the industry (Smelcer, 1989). In this illustration are five IF-THEN rules that guide a user in deciding if in an SQL query a closing special statement (called a "join statement") is needed. It is needed if the user has asked for data that reside in two or more tables. The join statement tells the system how to link the data elements in the two tables. In Figure 9, the first rule sets up the goal, leaving a NOTE and a STEP in WM. The second rule performs the act of counting the number of tables used, stores that value in WM, deletes the goal of counting tables, and adds the STEP to add a note to the WM about needing the join statement. Each IF part of each rule checks for a match to the

current goal (e.g., "GOAL SeeIfJoinNeeded" in the first rule) and the current notes in WM (e.g., "NOTE NumberOfTables 2" in the third rule). If there is a match, it executes other processes and adds and deletes notes and steps. This example illustrates the form and size of the productions and the putting and taking things off WM, both used in the discussions of learning and errors.

3.1. Learning and Transfer

Time to Learn

A significant shortcoming of the original GOMS framework was its restriction to skilled performance. A significant advance on this front is the work of Kieras and Polson focusing on both the time to learn new procedures and the transfer of training between procedures having various relationships to each other. For them, the explicit modeling of GOMS through production systems is the key. Kieras and Polson developed an extension of GOMS they called *Cognitive Complexity Theory* (see Polson, 1987, 1988). This theory provides a basis for making quantitative predictions about the time to learn each new piece of a task and the amount of transfer that could be expected in learning one new system or task after another.

Kieras and Polson first determined the number of steps in a procedure by counting the steps as encoded in a specialized language called NGOMSL. NGOMSL is the higher level programming language developed by Kieras (1988) to make the job of programming production system representations much easier. They then assessed the time it takes a person to learn the procedures. They studied learning under highly restrictive and controlled conditions, necessary in order to make the variance in learning times of a magnitude where one had some confidence in the quantitative estimates. They found that the time to learn each step took **30 s**, assuming a start-up or learning context time of 30 to 60 min.

This is much longer than the time estimated by Newell and Simon (1972), who calculated that learning each "chunk" of information took 5 to 10 s. The learning they referred to, however, is equivalent to learning a nonsense syllable, whereas for Kieras and Polson the learning was of steps in procedural knowledge, both the conditions under which it is appropriate and the actions associated with it. Ziegler, Vossen, and Hoppe (1986) report a **17 s** per production system step when the learning situation has some explanation associated with each piece to be learned. And Card (personal communication, January, 1989) reported **20 s** per production learning time.

These estimates vary. The authors themselves, however, claim that the variation has to do with the conditions under which the user is learning—for example, whether there is explanation associated with the steps being learned or whether opportunities for transfer are pointed out. And, of course,

learners do not acquire knowledge about new systems under the carefully controlled, artificial circumstances of Kieras and Polson's research. Thus, it is much harder to know how to quantify learning times in more naturalistic situations.

What we find more promising, however, is that the values are the same order of magnitude over widely different situations and laboratories. Though nuances of training time per production will be interesting and research on this topic is to be encouraged, for our purposes here, the current "best guess" parameter is about **25 s** per production.

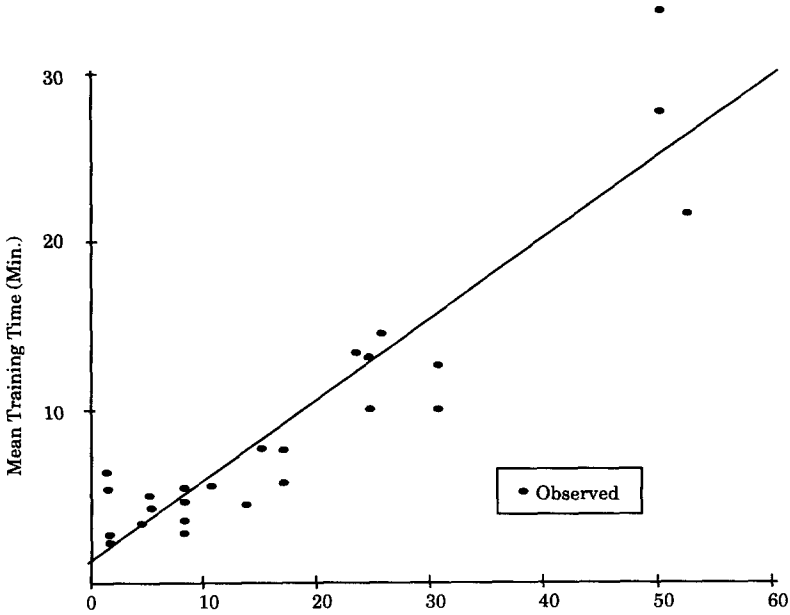
TAG (Payne & Green, 1986), because they also represent concrete aspects of the knowledge users must have to perform a task, similarly provides a basis from which to predict learning. Early rationales for using grammar representations were based on the argument that the number of rules determined the ease of learning, in that the more often a rule can be used, the more consistent a system is. Payne and Green (1989) argued that the number of rules is less critical than whether the features of those rules follow real-world features encoded in the user's memory already. Their empirical work demonstrated just this. A system described by 28 rules that had features described by well-known categories, one feature into one action, was learned nearly three times faster than one with 12 rules, but with complicated descriptions of the features of the situation to be attended. Although Payne and Green could have quantified learning time per rule, they did not because their argument made the content of the rules, not their number, the critical determinant of learning.

Transfer of Training From One System to the Other

Another, perhaps more generalizable aspect of Kieras and Polson's research has been the analysis of transfer between systems or components of systems. Their production system models make explicit exactly what it is that a person has to learn in acquiring knowledge of a new system. If productions are the units of learning, then a reasonable hypothesis is that the number of productions the two systems share provides a good metric for predictions of the amount of transfer. In other words, this analysis allows one to specify the exact effects of design consistency across systems.

In a series of studies, Kieras and Polson (Bovair et al., 1990; Kieras, 1988; Kieras & Bovair, 1986; Kieras & Polson, 1985; Polson & Kieras, 1985; Polson et al., 1986) have shown the predictive power of this approach. In a typical study, subjects learned to do various specific procedures on a computer system, and the time to master a new procedure was predicted to be a function of the number of new productions that would need to be learned. Because the procedures shared varying numbers of productions, there ought to be varying amounts of transfer or savings in learning as subjects learned a series of procedures. Figure 10 shows the results from a typical study. Here, subjects

Figure 10. **The fit of predictions of learning time to the number of new rules that have to be learned (from Polson, 1988).**



learned simple utility procedures for a floppy-disk-based microcomputer system, such as duplicating a diskette or printing a document stored on a diskette. Predicted values were based on the number of new productions to be learned, and observed values were based on learning the procedure to a criterion of learning under strict control.

These results are typical of a wide range of studies that have reviewed the learning of simple procedures. For example, Singley and Anderson (1988) found strong support for what is called the "identical elements model of transfer based on a production system representation of cognitive skill" (p. 223) in their study of transfer of text editors, both similar and dissimilar. This body of data shows the predictive power of the production system implementation of the GOMS model for characterizing transfer between procedures. In particular, it makes explicit and quantifiable the advantages of consistency in design.

Although these transfer results were obtained under the same kind of learning conditions as in the studies of learning time, there is less concern for their artificiality here. First, because there was a great deal of transfer, people were performing well; they did not often experience the somewhat harsh error-correction method. Second, in using these methods for doing compar-

ative analyses of different systems, what one most cares about is the relative transfer among system components. Because production system models can be built for systems still under design, it is possible to assess the relative costs of differing degrees of consistency among procedures.

Like production systems, TAG has the potential of aiding the design process by assessing the amount of transfer expected from one system to another, aiding, for example, the consistency in a "family of products." Although the metric of counting the number of rules the two systems have in common is possible (and mentioned in Green et al., 1988), there has been no empirical verification to date.

3.2. The Analysis of Errors: Forgetting From Working Memory

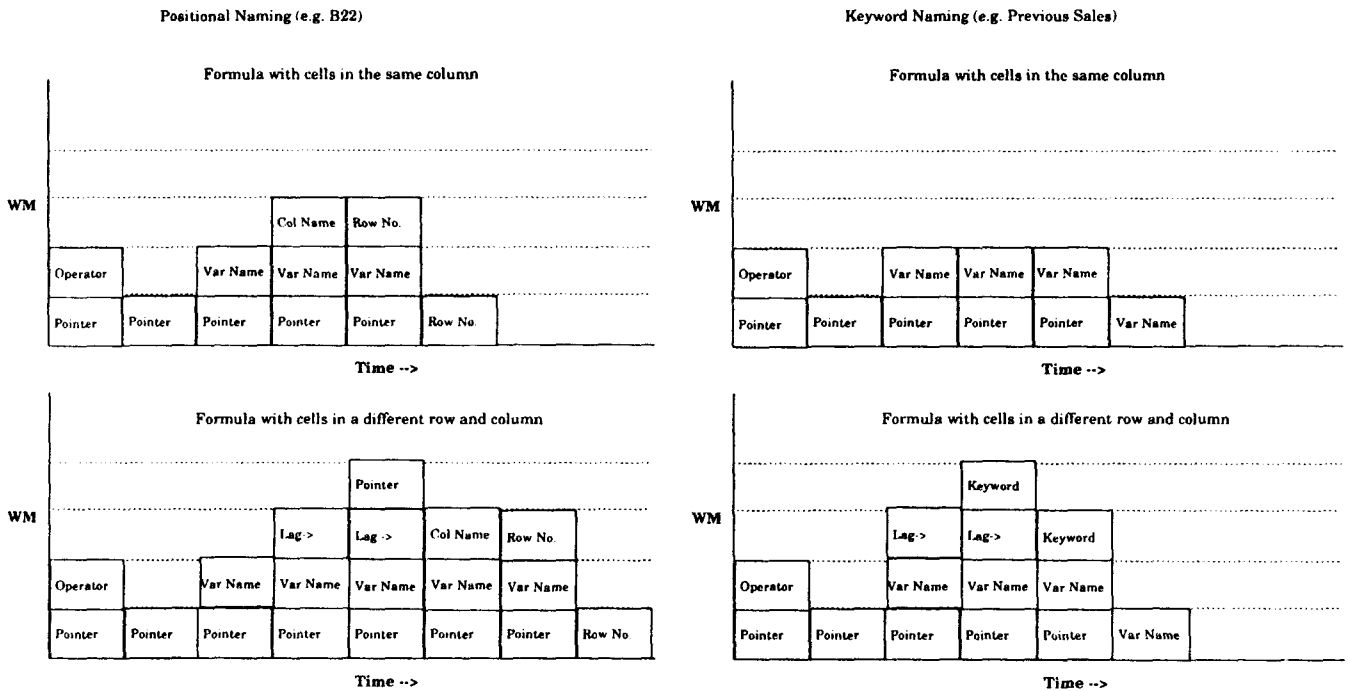
Card et al. (1983) recognized that one of the causes of errors could be overload of WM or the goal stack. They tracked (see Card et al., 1983, Figure 11.12) the contents of WM during the performance of a task. As mentioned earlier, Kieras and Polson's formalism allows one to keep track explicitly of both the contents and the resident duration of each piece of information in WM. They stated that they expect errors to increase as soon as more than five items are held in WM (Kieras, 1988). However, neither they nor Card et al. empirically tested the prediction that the more items in WM, the greater the likelihood of errors.

Two researchers, however, recently tested these predictions in laboratory studies. The first of these, Lerch (Lerch, 1988; Lerch, Mantei, & Olson, 1989), developed a GOMS representation of skilled users writing out formulas in Lotus 1-2-3 (e.g., D23-D38), and in Interactive Financial Planning System (IFPS; e.g., PROFIT = REVENUES - COSTS). These two pieces of software differ in whether the user has to find and remember the coordinates of the cells in the formula (e.g., D23) or can refer to them by name, with adjectives such as *previous* to indicate relative location.

These two systems require various levels of WM load for different types of standard formulas commonly entered on the job. For example, any formula that requires temporary storage of values from different rows and columns puts a much greater load on WM in the Lotus representation than it does on IFPS representation. Figure 11 shows the WM load for the two systems in successive units of time, determined from the GOMS notation.

The results show that WM load for the two different interface styles and different kinds of formulas (those that are simple, single-column formulas and those that require cells from different columns and different rows) predict the occurrence of errors well. The higher the WM load, the more errors. For example, for formulas with cells in different rows and columns (the most

Figure 11. **Working memory load during the specification of various formulas that refer to cells in the same row or column or different rows and columns in two different languages: a language that requires the user to specify cells by coordinate values (e.g., B22) and one that refers to cells by name, with location specified relative to the current cell (e.g., Previous Sales).**



complicated), the number of items that passed through WM for Lotus was 19[2] and for IFPS was 14. People writing in Lotus made errors on 14% of the formulas of this type, whereas people writing in IFPS made errors on only 6% of the formulas. Extrapolation of this relation suggests that when there are fewer than eight items in WM, errors would be eliminated.

Similarly, Smelcer (1989) examined the errors that users make while querying databases using SQL. In making queries, many people forget to put in an additional, necessary statement when information from two or more separate tables is used. This omission turns out to be a serious error, because unlike syntax errors that are caught by the system and corrected by the user, these "join errors" are syntactically correct and often return information that may be misleading. The difficult aspect of writing such a query is that the join statement is to be written last. The user has to remember to write it. Often there are steps in between that also require use of WM, making the likelihood of forgetting the crucial last step higher.

Smelcer calculated the WM load for different kinds of queries, those that have several steps in the middle of the query and those that have none. For example, a query that asks only for the names of all the employees of a company and their departments uses information from two tables and requires a join statement, but has no intervening "restriction" statements. On the other hand, a query that asks for the names of the employees who live in New Jersey and make over $40,000 a year has two statements between the initial part of the query and the final join statement.

Smelcer found a direct relationship between the number of intervening restriction statements (each of which requires many items to be stored in WM if calculated from the production system model) and the likelihood of people forgetting the last, crucial join statement. With no restrictions, people made 1.7% errors (20 out of 1200 opportunities); with two restrictions intervening before the join statement, they made 4.2% errors (50 out of 1200). The greater the WM load, the greater the number of omission errors.

In neither of these analyses do we know if the critical variable is the peak load in WM (in which we assume things are "bumped out") or the length of stay of each of the items (where we assume that information decays with time). Calculating exactly how heavy the load in WM has to be before items are forgotten is difficult to assess. It is likely that people adopt strategies in which they do not attempt to hold too much in WM, using clues from the external environment to remind them of things needed in the task instead.

There are, of course, many causes of errors other than WM overload. The analyses just described only open the door on the treatment of errors within

[2] This value is derived from the number of items in WM (notes and steps, not goals), summed over the number of time steps that items were held in WM.

the GOMS framework, but the work described offers a significant beginning toward addressing one of the major shortcomings of the GOMS models.

3.3. Parallel Processes

Although the MHP assumes that processes can go on in parallel, for simplicity in engineering calculations, most work carried out in the GOMS framework assumes that the elementary processes captured in the models are serial. This simplifies the quantitative modeling, because total task time is assumed to be the sum of the times of a number of subcomponents. Further, it is consistent with a long tradition of cognitive modeling that assumes that much cognitive activity, at any rate, is fundamentally serial in nature (e.g., Newell & Simon, 1972). However, even Card et al. (1983) acknowledged that there are important cases where key components of performance may operate in parallel. Certainly tasks where perceptual or motor activities dominate, such as expert transcription typing, are likely to be best characterized by processes operating in parallel. The expert typist is reading input, translating it into motor output, and executing actions that type characters, all at a speed that demands an account in terms of parallel cognitive activities.

Card et al. (1983, Figure 5.10) illustrated the family character of GOMS by presenting a number of specific models of text editing that vary in grain of analysis. At the coarsest level, the model characterizes performance in terms of unit tasks; at their finest, the model accounts for individual keystrokes as a user interacts with the system via the keyboard. Such factors as the specific task, the level of expertise of the user, and the details of system operation could introduce parallelism at any level in such a hierarchy of models. The analyst might adopt the strategy of choosing the largest grain size at which there is no parallelism, if it were deemed important to keep the analytic advantages of working with serial models. Alternatively, methods such as those we are about to describe could be used to model performance at a grain size that includes substantial parallelism. Whether the behavior should be modeled by a serial or by a parallel model, however, is in large part determined by the grain size of the behavior selected by the analyst.

An analysis of a typical situation involving a user and a computer reveals numerous opportunities for parallel or cascading processes.

1. The user is bombarded with a set of external signals that sometimes occur in parallel. For example, keypresses are "echoed" on the screen while beeps alert the user of inappropriate action.
2. Mental events perhaps occur in parallel, sending cascaded information from one to the next. For example, plans of action might be retrieved

for the next task while instructions about the immediately preceding task are executed.

3. External actions are elicited in parallel. For example, keys are pressed while eyes seek confirmation or new information to guide the next actions.

Rapid typing clearly requires a parallel model. For instance, those keystrokes that occur after one on the opposite hand are faster than if the same finger hits the key twice. This is probably because motions cascade; that is, one can be executing while another is begun. It is clear that the models have to incorporate notions of parallel processes.

This phenomenon appears in more than just use of a keyboard. Practical experience tells us that, at certain levels of analysis, skilled human–computer interaction is made up primarily of cascading internal processes. For example, in certain routine computer-supported tasks, such as imprinting bank checks with the amount (which is handwritten on the check) so that the whole check can be machine processed, the clerk is simply reading the handwritten amount from the check and keying in that same amount. How fast can the clerk key?

The keystroke model seriously overestimates the times:

> A saccade takes 230 msec; recognition takes 314 msec.
>
> Each numeral keystroke takes a minimum of 80 msec (for key entry times approaching 135 wpm), and typically there are four to five of them, plus an Enter key, which advances the display to the next check to be read, at an estimate of 1.35 s for retrieval (or less) and a 280 msec for entry.

This adds up to over 2 s per item. Real clerks are reported to complete them at less than .5 s each (J. O. Kerns, personal communication, February, 1989). Clearly, some of the processes are overlapping. Interestingly, the keying system just described allows the clerk to set a value that automatically displays the next item after a fixed number of keystrokes of the current item have been entered. Some of the best clerks are so fast that they set the system to display the next item while they are keying in the third numeral (out of five or six) of the previous item. They finish the keying of the last two numerals while they are looking at the next handwritten amount on the screen.

Furthermore, in this same situation, clerks accurately recognize that they have made a keying error after entering the whole next item. That is, the process that recognizes the miskeying finishes after the whole next item has been entered. Clerks have been known to correct check entries two checks

back, never reaccessing the one just entered. Clearly, cascading of processes is occurring.

In our own experience, we recognize this same phenomenon when trying to apply the cognitive engineering model to menu design. Our observations show that when people know the menu item's name and location, visual search (for confirmation and guidance) and motor movement take place simultaneously. When users know the word they are looking for in a menu but not its location, the search is slower, and the motor movement seems to glide along with the visual scan. When the user knows neither the precise name nor the location, the time to find the item is fully dependent on reading time. The movement traces the eyes' reading time; as soon as the item is recognized, the cursor is at the right location.

For the expert user, then, the limiting or predicting time is fully dependent on the motor movement speed; for the novice, it is dependent on the reading speed. Predicting the actual performance time requires knowledge about what processes are occurring in parallel (or cascading or altered as they work in concert), which processes depend on each other (which must complete their processing before the next can begin), and which of them has a speed that is the limiting factor in the task at hand.

3.4. Critical Path Analysis: An Approach to Parallel Processing

Schweikert (1978, 1980) first proposed an analysis of nonserial processes in timed cognitive tasks using critical path analysis. He offered a very general set of procedures for constructing the graph of task components from the patterns of reaction times across experimental conditions. Recently, John (1988) also used concepts from critical path analysis to analyze tasks in the domain of human-computer interaction. Critical path analysis is a tool from operations engineering that provides a useful framework for the modeling of cascading mental and external processes.

John (1988) focused on the task of continuous typing. She wrote:

> The parallel operation and sequential dependencies of the three processors make the processes of typing difficult to analyze and talk about. Fortunately, there is an analysis technique, borrowed from engineering project management that allows easy analysis of parallel resources (the three processors [perceptual, cognitive and motor]) working with sequential dependencies (outlined by the typing-specific assumptions). The technique is called *critical path analysis*. (italics added, p. 49)

Critical path analysis allows analysts to specify the component processes, their duration, and the dependencies among them. A program for this

analysis will calculate the path through this network that determines how long the total process can take. The program recognizes that some processes that occur simultaneously with others and take less time never enter into the calculation of the total duration of the task.

John's example analyses show how typing times can be calculated across situations in which the typist types words or random letters, with various numbers of letters displayed on the screen. Figure 12 depicts two examples of the processes involved. The top row of activity represents the perceptual processor (which is looking ahead at letters to type), the middle row is of the cognitive process (which is retrieving the individual letters of words read), and the bottom row is of the motor processor (which is enacting keypresses from either the same or different hands). The lines that connect the individual acts show the processes that theoretically must be completed before the next can start. For example, the motor processor cannot type a letter until it is retrieved, and keys on the same hand cannot be initiated until the cognitive processor intervenes.

In Figure 12, we compare the critical path for world-class and regular typists (who have motor times of 30 msec and 200 msec, respectively) in the top and bottom halves of the picture. After specifying the parameters for perceptual and cognitive processing (which are the same for both levels of skill) and calculating the critical path through the network, we find that the world-class typist is limited by the speed of the cognitive processor, whereas the regular typist is limited by the motor speed.

Critical path analysis is the kind of modeling tool that will help us in predicting times that seem to have parallel components. For example, it could help us predict the total time it takes a user to select an item from a menu if the user variously does or does not know the name of the item or where it is located. Figure 13 shows what this analysis looks like with the expert at the top, the person who knows the name but not the location in the middle, and the person who knows neither name nor location on the bottom. Simple serial GOMS modeling will not do. For these fast, simple tasks, some representation of the explicit dependencies of parallel processes as well as their component times is necessary.

In the past, we have been able to verify from observation the moment-by-moment fit of the components and their associated times. With critical path analysis, it is much harder to identify which aspects of one's assumptions about times and dependencies are right and which are wrong. Although Schweikert (1978, 1980) made some of the problem of discovery of processes and their interaction tractable, the modeling and confirmation process is still difficult. We will have to be very clever in our comparisons of performance across task demands or users' knowledge to make the inferences necessary to confirm or disconfirm the model and its parameters. However, doing the engineering predictions (of total time for the task) may have sufficient

Figure 12. Critical path analysis for two typists. The top one represents the typing speed of a world-class typist (30 msec/keystroke); the bottom one represents that of an average skilled typist (170 msec/keystroke).

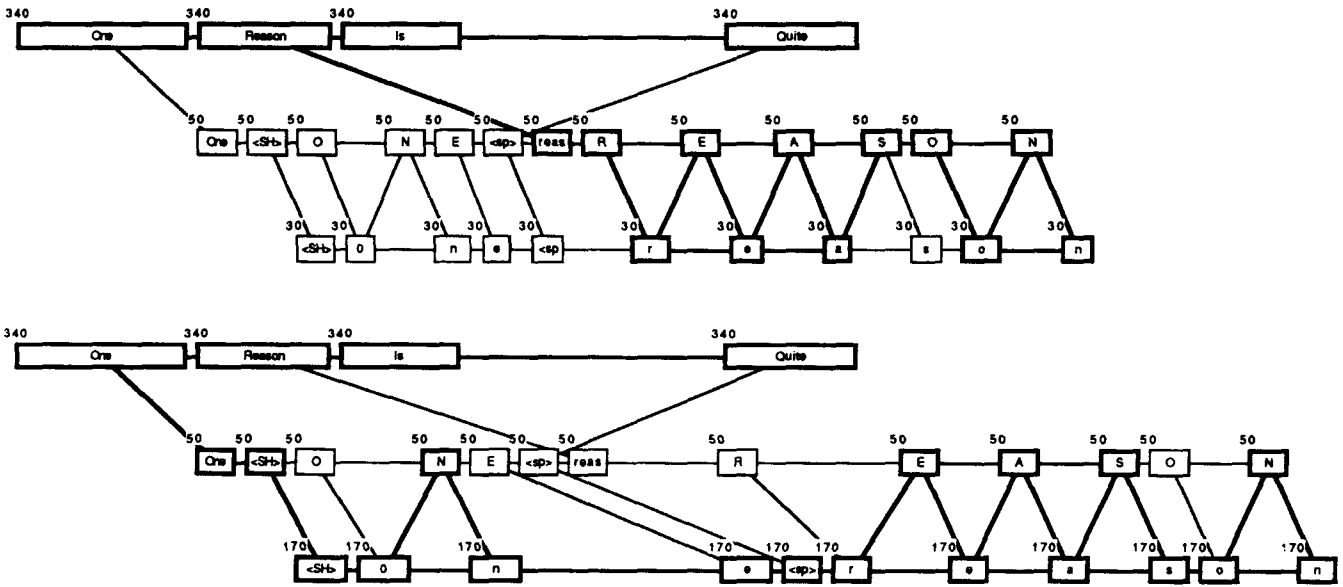
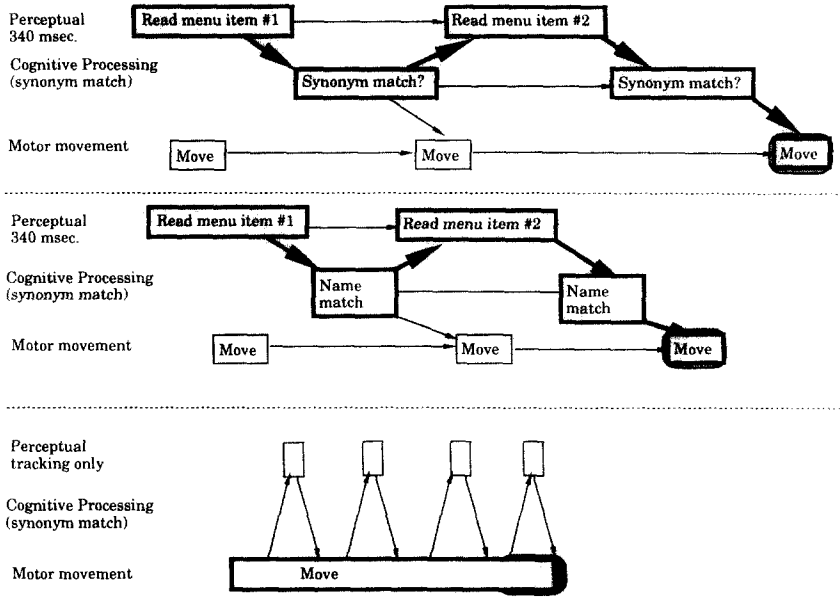


Figure 13. **Critical path analysis for people who know the name and/or the location of items on the menu.**



accuracy for design questions; what is difficult is the test of the underlying psychological model.

4. THE PLACE OF COGNITIVE MODELING IN HUMAN-COMPUTER INTERACTION

We have looked at examples of how cognitive modeling has been extended to cover learning and transfer, errors, and parallelism. In this section, we examine two issues: (a) the other kinds of useful extensions that could be made to give the framework even more scope; and (b) where, in the larger scheme of things pertaining to the study of human-computer interaction, even an extended and confirmed cognitive modeling framework would stand. These comments separate into those areas in which the framework might stand, but the kinds of information we need from cognitive theory does not yet exist to inform this extension. And there are questions about human-computer interaction that appear to require a wholly different kind of modeling. These assessments are categorized in Figure 14 and discussed next.

4.1. Additional Plausible and Useful Extensions

In this section, we follow the list of shortcomings reviewed in Section 1 of this article. The first category includes the areas that we feel could be

Figure 14. **Summary of those issues addressed by new work on GOMS, those addressed by extensions to models in the GOMS tradition, and those judged not easily accommodated in GOMS because either there is not enough known in cognitive theory to inform the effort, or the level of analysis represented in GOMS is inappropriate for such questions.**

	Some Work Has Addressed This Topic	Straightforward Extension Seems Possible	Cognitive Science Does Not Inform Us	Requires Another Kind of Modeling
Nonskilled users	–	X	–	–
Learning	X	X	–	–
Errors	X	X	–	–
Cognitive processes	X	X	X	–
Parallel processes	X	X	X	–
Mental workload	–	X	–	–
Functionality	–	–	–	X
Fatigue	–	–	X	–
Individual differences	X	–	–	–
Acceptance	–	–	–	X
Fit to organizational life	–	–	–	X

addressed within the original framework and its extensions but that have not yet been fully explored.

Nonskilled or Casual Users

Engineering models apply best to those situations that have skilled users of a particular system. Many such situations exist in industry, where clerks enter and change entries all day long. And not all of this work is simple data entry; with more advanced capabilities of systems at a price that makes them more widely available, people who do routine problem solving (e.g., debugging or error rectification) need systems with easy to use interfaces for rapid processing.

But many others of us are not skilled users of any particular system; we use a variety of systems to do our work, and we know some parts of each system better than others. Thus, some of the time in our interactions we are not simply retrieving and acting out plans. We retrieve the command appropriate to another system or inferred from its style; we often make errors of commission and figure out what must be done next given where we are now. We move smoothly from cognitive skill to problem solving and back. How people do this is of immense theoretical interest to cognitive psychologists. The article by Polson and Lewis (1990) is a step in this direction.

A second active research program that is relevant to this topic of casual user performance is the SOAR project (Laird, Newell, & Rosenbloom, 1987). SOAR is a running, growing model of the architecture of cognition. In style similar to that of cognitive engineering, SOAR could provide the detail necessary to answer the questions about how long it takes to recognize an impasse in skill, how long it takes to set up a new goal, how quickly similar productions are retrieved and analyzed, and how many steps are executed before a solution is found. All these parameters could likely be predicted in the SOAR framework (provided with a good task analysis) for situations in which a user is faced with learning a new system after knowing one that shares some relevant features.

Learning

One of the biggest, longest lasting issues in psychology is learning. People generate explanations of how a system works by watching the system respond to their inputs. We learn by asking and learn by watching. We need to know how that works; how productions grow and are integrated into methods; and what the psychological realities of methods, as opposed to chunked sequences, are.

Although we know that consistency in a system may make that system easier to learn (fewer productions to learn) and easier to operate (well-known productions can be used over and over again), we do not know much more than that. We know from the work of Kieras and Polson that a system that

shares productions with a system already known will be easier, but we do not know exactly how to design to that prescription. We do not know which system to base a new one on. Suppose the first system is internally inconsistent. We do not know if we would benefit more from reusing the old, inconsistent, confusable productions or from spending the time and effort to learn the new, consistent productions. Some phenomena from verbal learning literature seem applicable here, but as several researchers have found, not all phenomena from paired-associate learning apply (e.g., Singley & Anderson, 1988).

Errors and Mental Workload

Even in skilled performance with computers, we still make errors. As discussed earlier, some of these errors arise from an interface design that requires too much to be held in a limited-capacity WM. Users come to learn that such errors can be caught. They monitor their performance more closely by both checking concurrently for errors and breaking at the end of a task to assess whether errors have been committed. They also change their methods to include such things as writing down the address of a critical cell in a spreadsheet or printing out a document so that it can be scanned quickly for a target word in the area in which they want to work next. Understanding how people monitor their own behavior for errors and how they adapt their methods is critical to the full understanding of performance on the kinds of tasks for which analytic modeling was designed. And, as we have seen, analytic models can provide an explicit account of aspects of mental workload during tasks in terms of activity through the WM. Additional aspects of workload having to do with the use of the cognitive processor and the work involved in retrieval of procedures and facts could follow.

Cognitive Processes

Of those aspects of skilled performance in human-computer interaction that are underspecified in the models to date, interpretation of visual displays may be the most important. We know that Gestalt principles will drive some of the user's interpretation of what is on the screen, but we do not know much about how meaning is imparted on the whole display. We need to know this not only to determine how to display information for the slow, deliberate problem-solving processes involved in high-level decision making, but also for the fast, repetitive operational tasks such as those in keying handwritten characters in checks. Recent work by Tullis (1983, 1988) and Mackinlay (1986) made significant inroads in this area, but more work is called for, both in the application of what we know and in the discovery of basic facts about how people interpret what they see.

Parallel Processes

Concurrent or cascading processes seem to occur in rapid, skilled performance. We noted that clerks imprinting checks with machine-readable numerals could detect that they had made an error two checks past. This phenomenon indicates not only the existence of concurrent perception and motor movement but also that there may be some process that actively checks the accuracy of the behavior soon after that behavior has occurred. This is a natural case of simultaneous tasks. We need to know how this works and how many processes such as this background monitoring exist.

John's (1988) adoption of the critical path method of modeling concurrent, dependent processes requires us to characterize in great detail our assumptions of processes involved and their dependencies in these kinds of tasks. The literature in cognitive psychology provides even less guidance on these dependencies than it does in finding appropriate parameters. To illustrate the detail needed, John's modeling of the continuous typing task makes strong assumptions about which kinds of processes depend on the completion of other processes. For example, she stated that:

> The perceptual processor cannot perceive the next piece of information unless there is room in working memory for that information. . . . A character on the same hand cannot be initiated with a cognitive operator until the motor processor execution of the previous character is complete. (p. 47)

Though such simplifying assumptions may achieve success in some domains and at some levels of analysis, it is important to recognize that the existing psychological literature does not support such strong assumptions. Finding the kinds of links that do exist among processes is critical to this kind of modeling success.

Individual Differences

Card et al.'s (1983) original clocking of different motor operators of people with different skill levels and John's (1988) work on various levels of skill in typing exemplify how we might explore the implications of individual differences in human-computer interaction. Though these differences in motor operators clearly have an effect on predicting the total time that a task will take, there are likely large differences in perceptual and cognitive operations that similarly have large effects. People clearly differ in rate of learning, speed of retrieval, and reasoning or strategy formation. Egan (1988) reviewed a variety of ways in which individuals differ in their use of computers. This work and the more detailed parametric work in Gomez, Egan, Wheeler, Sharma, and Gruchacz (1983) and Gomez, Egan, and

Bowers (1986) could guide the further exploration of the effect of differences in the rate and form of cognitive processing on predictions of time and errors in the spirit of GOMS modeling.

4.2. Cognitive Modeling in Human-Computer Interaction

In the broader picture of providing tools for designers of human-computer interfaces, several important aspects of human-computer interaction do not seem to lend themselves to the approach described here. First, this approach does not capture the impact of fatigue on the times and errors associated with performance. Our models never tire; people do. Some extensions from research on the effect of time and mental workload may be relevant here, but the interplay of stress from the task itself from the physical environment (e.g., chairs and lighting) and the higher level aspects of one's job (e.g., work supervision, control over pacing, incentive and rewards, etc.) are beyond the scope of cognitive engineering as we see it. Some of this arises because of the little we know about the causes of fatigue on mental processing, and some arises from the possibility that when we do know, the GOMS-like modeling framework includes the wrong set of constructs.

Second, we do not see a possible extension of these models that would include assessment of people's perception of the acceptability of an interface. People's judgments of ease of use are not always in concordance with their actual productivity (e.g., Davis, Bagozzi, & Warshaw, 1989). Perceptions of whether the software's functionality is actually what the user needs and the ease with which the system can be learned all contribute to the acceptance of a piece of software and its eventual regular use. These do not appear to fit the style of analytic modeling, because they refer to an entirely different level of granularity of behavior and a different set of relevant features.

Perhaps the most important step in building a successful piece of software may be the analysis of the functions it will perform, in what order it will allow the user to do them, and how they fit into the larger picture of work and organizational roles. This point is made strongly under the rubric of "situated cognition," the understanding of work, its task and setting, and the eventual design of computers to support it (see Suchman, 1987). The topic of discovering useful functions has been explored in the management literature (e.g., Sasso, Olson, & Merten, 1987) and in the literature on traditional human factors (e.g., Sheridan, 1988), based on a broad analysis of what people do best and what computers do best. These approaches, however, do not yet converge at a level specific enough to be useful to a system designer. And the assessment of how computers change work and organizational life is as yet still at the descriptive, not theoretical, level (e.g., Markus, 1984). A kind of modeling different from that provided by the analytic models

described in this article appears to be required to answer these kinds of important questions.

Despite these limitations, large classes of applications can be characterized in just the way that GOMS and its successors require. Every day thousands of people carry out computer-based interactions that are highly repetitive and stylized. Airline reservations, telephone directory assistance, troubleshooting of discrepancies in forms such as bank deposits all are examples of tasks in which cognitive modeling can play an important role in the design of learnable and usable systems. In these kinds of tasks, the approach is very powerful. Because we now have reasonably useful quantitative estimates of a number of task components, predictions about performance can be made for designs without our having to build prototype systems and run extensive, time-consuming user tests. User tests can, of course, reveal other things such as errors, problem solving, and initial learning and representation difficulties; however, cognitive models can screen out certain classes of poor designs for these kinds of tasks.

Acknowledgments. This article benefited greatly from very careful readings and thoughtful suggestions from Peter Polson, John Karat, Clayton Lewis, Thomas Green, Richard Young, and Bonnie John. We remain responsible for any faults in the clarity, accuracy, and opinions of the result but are grateful for the help these people gave us.

Support. This work was partially funded by the Army Research Institute, MDA903-89-K-0025.

REFERENCES

Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text-editing skill: A cognitive complexity analysis. *Human-Computer Interaction, 5,* 1-48.

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, Inc.

Carroll, J. M., & Campbell, R. L. (1986). Softening up hard science: Reply to Newell and Card. *Human-Computer Interaction, 2,* 227-250.

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: A comparison of two theoretical models. *Management Science, 35,* 982-1003.

Egan, D. E. (1988). Individual differences in human-computer interaction. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 543-568). Amsterdam: North-Holland.

Fitts, P. M., & Peterson, J. R. (1964). Information capacity of discrete motor responses. *Journal of Experimental Psychology, 67,* 103-112.

Gomez, L. M., Egan, D. E., & Bowers, C. (1986). Learning to use a text editor: Some learner characteristics that predict success. *Human-Computer Interaction, 2,* 1-23.

Gomez, L. M., Egan, D. E., Wheeler, E. A., Sharma, D. K., & Gruchacz, A. M. (1983). How interface design determines who has difficulty learning to use a text editor. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems,* 176-181. New York: ACM.

Green, T. R. G., Schiele, F., & Payne, S. J. (1988). Formalisable models of user knowledge in human-computer interaction. In G. C. Van der Veer, T. R. G. Green, J.-M. Hoc, & D. M. Murray (Eds.), *Working with computers: Theory vs. outcome* (pp. 47-88). New York: Academic.

Hick, W. E. (1952). On the rate of gain of information. *Quarterly Journal of Experimental Psychology, 4,* 11-16.

John, B. E. (1988). *Contributions to engineering models of human-computer interaction.* Unpublished doctoral dissertation, Carnegie Mellon University, Pittsburgh.

John, B. E., & Newell, A. (1987). Predicting the time to recall computer command abbreviations. *Proceedings of the CHI '87 Conference on Human Factors in Computing Systems,* 33-40. New York: ACM.

John, B. E., & Newell, A. (1989). Cumulating the science of HCI: From S-R compatibility to transcription typing. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems,* 109-114. New York: ACM.

John, B. E., & Newell, A. (in press). Toward an engineering model of stimulus-response compatibility. In R. W. Gilmore & T. G. Reeve (Eds.), *Stimulus-response compatibility: An integrated approach.* Amsterdam: North-Holland.

John, B. E., Rosenbloom, P. S., & Newell, A. (1985). A theory of stimulus-response compatibility applied to human-computer interaction. *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems,* 213-219. New York: ACM.

Karat, J. (1988). Software evaluation methodologies. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 891-904). Amsterdam: North-Holland.

Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *The handbook of human-computer interaction.* (pp. 135-158). Amsterdam: North-Holland.

Kieras, D. E., & Bovair, S. (1986). The acquisition of procedures from text: A production-system analysis of transfer of training. *Journal of Memory and Language, 25,* 507-524.

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22,* 365-394.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33,* 1-64.

Lerch, F. J. (1988). *Computerized financial planning: Discovering cognitive difficulties in model building.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Lerch, F. J., Mantei, M. M., & Olson, J. R. (1989). Translating ideas into action: Cognitive analysis of errors in spreadsheet formulas. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems,* 121-126. New York: ACM.

Mackinlay, J. D. (1986). *Automatic design of graphical presentations.* Unpublished doctoral dissertation, Stanford University, Department of Computer Science, Stanford, CA.

Markus, L. (1984). *Systems in organizations: Bugs and features.* New York: Plenum.

Newell, A., & Card, S. K. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction, 1,* 209-242.

Newell, A., & Card, S. K. (1986). Straightening out softening up: Response to Carroll and Campbell. *Human-Computer Interaction, 2,* 251-267.

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

Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User centered systems design: New perspectives on human-computer interaction* (pp. 31-61). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Olson, J. R., & Nilsen, E. (1988). Analysis of the cognition involved in spreadsheet software interaction. *Human-Computer Interaction, 3,* 309-350.

Payne, S. J., & Green, T. R. G. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction, 2,* 93-133.

Payne, S. J., & Green, T. R. G. (1989). The structure of command languages: An experiment on task-action grammar. *International Journal of Man-Machine Studies, 30,* 213-234.

Polson, P. G. (1987). A quantitative theory of human-computer interaction. In J. M. Carroll (Ed.), *Interfacing thought: Cognitive aspects of human-computer interaction* (pp. 184-235). Cambridge, MA: MIT Press.

Polson, P. G. (1988). The consequences of consistent and inconsistent interfaces. In R. Guindon (Ed.), *Cognitive science and its applications for human-computer interaction* (pp. 59-108). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Polson, P. G., & Kieras, D. E. (1985). A quantitative model of the learning and performance of text editing knowledge. *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems,* 207-212. New York: ACM.

Polson, P. G., & Lewis, C. H. (1990). Theory-based design for easily learned interfaces. *Human-Computer Interaction, 5,* 191-220.

Polson, P. G., Muncher, E., & Engelbeck, G. (1986). A test of a common elements theory of transfer. *Proceedings of the CHI '86 Conference on Human Factors in Computing Systems,* 78-83. New York: ACM.

Reisner, P. (1981). Formal grammar and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering,* SE-*7,* 229-240.

Reisner, P. (1984). Formal grammar as a tool for analyzing ease of use: Some fundamental concepts. In J. Thomas & M. Schneider (Eds.), *Human factors in computer systems.* Norwood, NJ: Ablex.

Sasso, W. G., Olson, J. R., & Merten, A. (1987). The practice of office analysis: Objectives, obstacles, and opportunities. *Office Knowledge Engineering, 2,* 11-24.

Schweickert, R. (1978). A critical path generalization of the additive factor method: Analysis of a Stroop task. *Journal of Mathematical Psychology, 18,* 105-139.

Schweickert, R. (1980). Critical path scheduling of mental processes in a dual task. *Science, 209,* 704-706.

Sheridan, T. (1988). Task allocation and supervisory control. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 159-174). Amsterdam: North-Holland.

Singley, M. K., & Anderson, J. R. (1988). A keystroke analysis of learning and transfer in text editing. *Human-Computer Interaction, 3,* 223-274.

Smelcer, J. B. (1989). *Understanding user errors in database query.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Suchman, L. A. (1987). *Plans and situated actions: The problem of human-machine communication.* Cambridge, England: Cambridge University Press.

Tullis, T. S. (1983). The formatting of alphanumeric displays: A review and analysis. *Human Factors, 25,* 657-682.

Tullis, T. S. (1988). Screen design. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 377-411). Amsterdam: North-Holland.

Walker, J. N., Smelcer, J. B., & Nilsen, E. (1989). *Fitts' Law and its design implications: Attempting to optimize speed and accuracy of menu selection.* Unpublished manuscript, University of Michigan, Human Computer Interaction Laboratory, Ann Arbor.

Wilson, M. D., Barnard, P. J., Green, T. R. G., & MacLean, A. (1988). Knowledge-based task analysis for human-computer systems. In G. C. Van der Veer, T. R. G. Green, J.-M. Hoc, & D. M. Murray (Eds.), *Working with computers: Theory vs. outcome* (pp. 3-46). New York: Academic.

Young, R. M., & MacLean, A. (1988). Choosing between methods: Analysing the user's decision space in terms of schemas and linear models. *Proceedings of the CHI '88 Conference on Human Factors in Computing Systems,* 139-143. New York: ACM.

Ziegler, S. E., Vossen, P., & Hoppe, H. U. (1986). *On using production systems for cognitive task analysis and prediction of transfer of cognitive skill.* Paper presented at the Third European Conference on Cognitive Ergonomics, Paris.

HCI Editorial Record. First manuscript received April 13, 1989. Revision received June 20, 1989. Final manuscript received October 4, 1989. Accepted by Peter Polson. — *Editor*
