Graphics and Image Processing
J.D. Foley Editor

The Keystroke-Level Model for User Performance Time with Interactive Systems

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

Allen Newell
Carnegie-Mellon University

There are several aspects of user-computer performance that system designers should systematically consider. This article proposes a simple model, the Keystroke-Level Model, for predicting one aspect of performance: the time it takes an expert user to perform a given task on a given computer system. The model is based on counting keystrokes and other low-level operations, including the user's mental preparations and the system's responses. Performance is coded in terms of these operations and operator times summed to give predictions. Heuristic rules are given for predicting where mental preparations occur. When tested against data on 10 different systems, the model's prediction error is 21 percent for individual tasks. An example is given to illustrate how the model can be used to produce parametric predictions and how sensitivity analysis can be used to redeem conclusions in the face of uncertain assumptions. Finally, the model is compared to several simpler versions. The potential role for the Keystroke-Level Model in system design is discussed.

Key Words and Phrases: human-computer interface, human-computer interaction, user model, user performance, cognitive psychology, ergonomics, human factors, systems design

CR Categories: 3.36, 4.6, 8.1

Authors' present addresses: S.K. Card and T.P. Moran, Xerox Corporation, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304; A. Newell, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

1. Introduction

The design and evaluation of interactive computer systems should take into account the total performance of the combined user-computer system. Such an account would reflect the psychological characteristics of users and their interaction with the task and the computer. This rarely occurs in any systematic and explicit way. The causes of this failure may lie partly in attitudes toward the possibility of dealing successfully with psychological factors, such as the belief that intuition, subjective experience, and anecdote form the only possible bases for dealing with them. Whatever may be true of these more global issues, one major cause is the absence of good analysis tools for assessing combined user-computer performance.

There exists quite a bit of research relevant to the area of user-computer performance, but most of it is preliminary in nature. Pew et al. [14], in a review of 40 potentially relevant human-system performance models, conclude "that integrative models of human performance compatible with the requirements for representing command and control system performance do not exist at the present time." Ramsey and Atwood [15], after reviewing the human factors literature pertinent to computer systems, conclude that while there exists enough material to develop a qualitative "human factors design guide," there is insufficient material for a "quantitative reference handbook."

This paper presents one specific quantitative analysis tool: a simple model for the time it takes a user to perform a task with a given method on an interactive computer system. This model appears to us to be simple enough, accurate enough, and flexible enough to be applied in practical design and evaluation situations.

The model addresses only a single aspect of performance. To put this aspect into perspective, note that there are many different dimensions to the performance of a user-computer system:

- *Time.* How long does it take a user to accomplish a given set of tasks using the system?
- *Errors.* How many errors does a user make and how serious are they?
- *Learning.* How long does it take a novice user to learn how to use the system to do a given set of tasks?
- *Functionality.* What range of tasks can a user do in practice with the system?
- *Recall.* How easy is it for a user to recall how to use the system on a task that he has not done for some time?

The authors of this report are listed in alphabetical order. A. Newell is a consultant to Xerox PARC. This paper is a revised version of [3]. For a view of the larger research program of which the study described in this paper is a part, see [5].

— *Concentration.* How many things does a user have to keep in mind while using the system?
— *Fatigue.* How tired do users get when they use the system for extended periods?
— *Acceptability.* How do users subjectively evaluate the system?

Next, note that there is *no single kind of user.* Users vary along many dimensions:

— Their *extent of knowledge* of the different tasks.
— Their *knowledge of other systems*, which may have positive or negative effects on the performance in the system of interest.
— Their *motor skills* on various input devices (e.g., typing speed).
— Their general *technical ability* in using systems (e.g., programmers vs. nonprogrammers).
— Their *experience* with the system, i.e., whether they are *novice users*, who know little about the system; *casual users*, who know a moderate amount about the system and use it at irregular intervals; or *expert users*, who know the system intimately and use it frequently.

Finally, note that there is *no single kind of task.* This is especially true in interactive systems, which are expressly built around a command language to permit a wide diversity of tasks to be accomplished. The number of qualitatively different tasks performable by a modern text editor, for instance, runs to the hundreds.

All aspects of performance, all types of users, and all kinds of tasks are important. However, no uniform approach to modeling the entire range of factors in a simple way appears possible at this time. Thus, of necessity, the model to be presented is specific to one aspect of the total user-computer system: *How long it takes expert users to perform routine tasks.*

The model we present here is simple, yet effective. The central idea behind the model is that the time for an expert to do a task on an interactive system is determined by the time it takes to do the keystrokes. Therefore, just write down the method for the task, count the number of keystrokes required, and multiply by the time per keystroke to get the total time. This idea is a little too simplistic. Operations other than keystrokes must be added to the model. Since these other operations are at about the same level (time grain) as keystrokes, we dub it the "Keystroke-Level Model." (The only other similar proposal we know of is that of Embley et al. [9], which we discuss in Section 6.1.)

The structure of this paper is as follows: Section 2 formulates the time prediction problem more precisely. Section 3 lays out the Keystroke-Level Model. Section 4 provides some empirical validation for the model. Section 5 illustrates how the model can be applied in practice. And Section 6 analyzes some simpler versions of the model.

2. The Time Prediction Problem

The prediction problem that we will address is as follows:

Given: A task (possibly involving several subtasks); the command language of a system; the motor skill parameters of the user; the response time parameters of the system; the method used for the task.

Predict: The time an expert user will take to execute the task using the system, providing he uses the method without error.

Several aspects of this formulation need explication, especially the stipulations about execution, methods, and the absence of error.

2.1 Unit Tasks and Execution Time

Given a large task, such as editing a large document, a user will break it into a series of small, cognitively manageable, quasi-independent tasks, which we call *unit tasks* [4; 5, ch. 11]. The task and the interactive system influence the structure of these unit tasks, but unit tasks appear to owe their existence primarily to the memory limits on human cognition. The importance of unit tasks for our analysis is that they permit the time to do a large task to be decomposed into the sum of the times to do its constituent unit tasks. Note that not all tasks have a unit-task substructure. For example, inputting an entire manuscript by typing permits a continuous throughput organization.

For our purposes here, a unit task has two parts: (1) *acquisition* of the task and (2) *execution* of the task acquired. During acquisition the user builds a mental representation of the task, and during execution the user calls on the system facilities to accomplish the task. The total time to do a unit task is the sum of the time for these two parts:

$$T_{task} = T_{acquire} + T_{execute}$$

The acquisition time for a unit task depends on the characteristics of the larger task situation in which it occurs. In a manuscript interpretation situation, in which unit tasks are read from a marked-up page or from written instructions, it takes about 2 to 3 seconds to acquire each unit task. In a routine design situation, in which unit tasks are generated in the user's mind, it takes about 5 to 30 seconds to acquire each unit task. In a creative composition situation, it can take even longer.

The execution of a unit task involves calling the appropriate system commands. This rarely takes over 20 seconds (assuming the system has a reasonably efficient command syntax). If a task requires a longer execution time, the user will likely break it into smaller unit tasks.

We have formulated the prediction problem to predict only the execution time of unit tasks, not the acquisition time. This is the part of the task over which the system designer has most direct control (i.e., by manipulating the system's command language), so its prediction suffices for many practical purposes. Task acquisi-

tion times are highly variable, except in special situations (such as the manuscript interpretation situation); and we can say little yet about predicting them.

Two important assumptions underlie our treatment of execution time. First, execution time is the same no matter how a task is acquired. Second, acquisition time and execution time are independent (e.g., reducing execution time by making the command language more efficient does not affect acquisition time). These assumptions are no doubt false at a fine level of detail, but the error they produce is probably well below the threshold of concern in practical work.

2.2 Methods

A *method* is a sequence of system commands for executing a unit task that forms a well-integrated ("compiled") segment of a user's behavior. It is characteristic of an expert user that he has one or more methods for each type of unit task that he encounters and that he can quickly (in about a second) choose the appropriate method in any instance. This is what makes expert user behavior routine, as opposed to novice user behavior, which is distinctly nonroutine.

Methods can be specified at several levels. A user actually knows a method at all its levels, from a general system-independent functional specification, down through the commands in the language of the computer system, to the keystrokes and device manipulations that actually communicate the method to the system. Models can deal with methods defined at any of these levels [4, 11]. The Keystroke-Level Model adopts one specific level—the keystroke level—to formalize the notion of a method, leaving all the other levels to be treated informally.

Many methods that achieve a given task can exist. In general such methods bear no systematic relationship to each other (except that of attaining the same end). Each can take a different amount of time to execute, and the differences can be large. Thus, in general, if the method is unknown, reasonable predictions of execution time are not possible. For this reason, the proper prediction problem is the one posed at the beginning of the section: Predict the time given the method.

2.3 Error-Free Execution

The Keystroke-Level Model assumes that the user faithfully executes the given method. The user deviates from a postulated method when he makes an error. Up to a fourth of an expert's time can be spent correcting errors, though users vary in their trade-off between speed and errors. We are simply ignoring the tasks containing errors and only predicting the error-free tasks, for we do not know how to predict where and how often errors occur. But, if the method for correcting an error is given, the model can be used to predict how long it will take to make the correction. Indeed, experts handle most errors in routine ways, i.e., according to fixed, available methods.

3. The Keystroke-Level Model

We lay out the primitive operators for the Keystroke-Level Model and give a set of heuristics for coding methods in terms of these operators. Then we present a few examples of method encoding.

3.1 Operators

The Keystroke-Level Model asserts that the execution part of a task can be described in terms of four different physical-motor operators: **K** (keystroking), **P** (pointing), **H** (homing), and **D** (drawing), and one mental operator, **M**, by the user, plus a response operator, **R**, by the system. These operators are listed in Figure 1. Execution time is simply the sum of the time for each of the operators.

$$T_{execute} = T_K + T_P + T_H + T_D + T_M + T_R. \quad (1)$$

Most operators are assumed to take a constant time for each occurrence, e.g., $T_K = n_K t_K$, where n_K is the number of keystrokes and t_K is the time per keystroke. (Operators **D** and **R** are treated somewhat differently.)

The most frequently used operator is **K**, which represents a keystroke or a button push (on a typewriter keyboard or any other button device). **K** refers to keys, not characters (e.g., hitting the SHIFT key counts as a separate **K**). The average time for **K**, t_K, will be taken to be the standard typing rate, as determined by standard one-minute typing tests. This is an approximation in two respects. First, keying time is different for different keys and key devices. Second, the time for immediately caught typing errors (involving BACKSPACE and rekeying) should be folded into t_K. Thus, the preferred way to calculate t_K from a typing test is to divide the total time taken in the test by the total number of nonerror keystrokes, which gives the *effective* keying time. We accept both these approximations in the interest of simplicity.

Users can differ in their typing rates by as much as a factor of 15. The range of typing speeds is given in Figure 1. Given a population of users, an appropriate t_K can be selected from this range. If a user population has users with large t_K differences, then the population should be partitioned and analyzed separately, since the different classes of users will be likely to use different methods.

The operator **P** represents pointing to a target on a display with a "mouse," a wheeled device that is rolled around on a table to guide the display's cursor. Pointing time for the mouse varies as a function of the distance to the target, d, and the size of the target, s, according to Fitts's Law [2]:

$$t_P = .8 + .1 \log_2 (d/s + .5) \text{ sec.}$$

The fastest time according to this equation is .8 sec, and the longest likely time ($d/s = 128$) is 1.5 sec. Again, to keep the model simple, we will use a constant time of 1.1 sec for t_P. Often, pointing with the mouse is followed by pressing one of the buttons on the mouse. This key press is not part of **P**; it is represented by a **K** following the **P**.

Fig. 1. The Operators of the Keystroke Model.

Operator	Description and Remarks	Time (sec)
K	Keystroke or button press. Pressing the SHIFT or CONTROL key counts as a separate K operation. Time varies with the typing skill of the user; the following shows the range of typical values:	
	Best typist (135 wpm)	.08[a]
	Good typist (90 wpm)	.12[a]
	Average skilled typist (55 wpm)	.20[a]
	Average non-secretary typist (40 wpm)	.28[b]
	Typing random letters	.50[a]
	Typing complex codes	.75[a]
	Worst typist (unfamiliar with keyboard)	1.20[a]
P	Pointing to a target on a display with a mouse. The time to point varies with distance and target size according to Fitts's Law. The time ranges from .8 to 1.5 sec, with 1.1 being an average time. This operator does *not* include the button press that often follows (.2 sec).	1.10[c]
H	Homing the hand(s) on the keyboard or other device.	.40[d]
D(n_D,l_D)	Drawing (manually) n_D straight-line segments having a total length of l_D cm. This is a very restricted operator; it assumes that drawing is done with the mouse on a system that constrains all lines to fall on a square .56 cm grid. Users vary in their drawing skill; the time given is an average value.	$.9n_D + .16l_D$[e]
M	Mentally preparing for executing physical actions.	1.35[f]
R(t)	Response of t sec *by the system.* This takes different times for different commands in the system. These times must be input to the model. The response time counts only if it causes the user to wait.	t

[a] See [8].
[b] This is the average typing rate of the nonsecretary subjects in the experiment described in Section 4.1.
[c] See [2].
[d] See [2, 4].
[e] The drawing time function and the coefficients were derived from least squares fits on the drawing test data from the four MARKUP subjects. See Sections 3.1 and 4.1.
[f] The time for **M** was estimated from the data from experiment described in Section 4.1. See Section 4.2.1.

The mouse is an optimal pointing device as far as time is concerned; but the t_P is about the same for other analog pointing devices, such as lightpens and some joysticks [2].

When there are different physical devices for the user to operate, he will move his hands between them as needed. This hand movement, including the fine positioning adjustment of the hand on the device, is represented by the **H** ("homing") operator. From previous studies [2, 4], we assume a constant t_H of .4 sec for movement between any two devices.

The **D** operator represents manually drawing a set of straight-line segments using the mouse. **D** takes two parameters, the number of segments (n_D) and the total length of all segments (l_D). $t_D(n_D, l_D)$ is a linear function of these two parameters. The coefficients of this function are different for different users; Figure 1 gives an average value for them. Note that this is a very specialized operator. Not only is it restricted to the mouse, but also it assumes that the drawing system constrains the cursor to lie on a .56 cm grid. This allows the user to draw straight lines fairly easily, but we would expect t_D to be different for different grid sizes. We make no claim for the generality of these times or for the form of the drawing time function. However, inclusion of one instance of a drawing operator serves to indicate the wide scope of the model.

The user spends some time "mentally preparing" to execute many of the physical operators just described; e.g., he decides which command to call or whether to terminate an argument string. These mental preparations are represented by the **M** operator, which we estimate to take 1.35 sec on the average (see Section 4.2.1). The use of a single mental operator is, again, a deliberate simplification.

Finally, the Keystroke-Level Model represents the system response time by the **R** operator. This operator has one parameter, t, which is just the response time in seconds. Response times are different for different systems, for different commands within a system, and for different contexts of a given command. The Keystroke-Level Model does not embody a theory of system re-

sponse time. The response times must be input to the model by giving specific values for the parameter t, which is a placeholder for these input times.

The **R** times are counted only when they require the user to *wait for the system.* For example, a system response counts as an **R** when it is followed by a **K** and the system does *not* allow type-ahead, and the user must wait until the response is complete. However, when an **M** operation follows a response, the response time is not counted unless it is over 1.35 sec, since the expert user can completely overlap the **M** operation with the response time. Response times can also overlap with task acquisition. When a response is counted as an **R**, only the nonoverlapping portion of the response time is given as the parameter to **R**.

3.2 Encoding Methods

Methods are represented as sequences of Keystroke-Level operations. We will introduce the notation with examples. Suppose that there is a command named PUT in some system and that the method for calling it is to type its name followed by the RETURN key. This method is coded by simply listing the operations in sequence: **MK**[P] **K**[U] **K**[T] **K**[RETURN], which we abbreviate as **M 4K**[P U T RETURN]. In this notation we allow descriptive notes (such as key names) in square brackets. If, on the other hand, the method to call the PUT command is to point to its name in a menu and press the RED mouse button, we have: **H**[mouse] **MP**[PUT] **K**[RED] **H**[keyboard].

As another example, consider the text editing task (called T1) of replacing a 5-letter word with another 5-letter word, where this replacement takes place one line below the previous modification. The method for executing task T1 in a line-oriented editor called POET (see Section 4) can be described as follows:

Method for Task T1-Poet:

Jump to next line	**MK**[LINEFEED]
Call Substitute command	**MK**[S]
Specify new 5-digit word	**5K**[word]
Terminate argument	**MK**[RETURN]
Specify old 5-digit word	**5K**[word]
Terminate argument	**MK**[RETURN]
Terminate command	**K**[RETURN]

Using the operator times from Figure 1 and assuming the user is an average skilled typist (i.e., $t_K = .2$ sec), we can predict the time it will take to execute this method:

$$T_{execute} = 4t_M + 15t_K = 8.4 \text{ sec.}$$

This method can be compared to the method for executing task T1 on another editor, a display-based system called DISPED (see Section 4):

Method for Task T1-Disped:

Reach for mouse	**H**[mouse]
Point to word	**P**[word]
Select word	**K**[YELLOW]
Home on keyboard	**H**[keyboard]
Call Replace command	**MK**[R]
Type new 5-digit word	**5K**[word]
Terminate type-in	**MK**[ESC]

$$T_{execute} = 2t_M + 8t_K + 2t_H + t_P = 6.2 \text{ sec.}$$

Fig. 2. Heuristic rules for placing the M operations.

Begin with a method encoding that includes all physical operations and response operations. Use Rule 0 to place candidate Ms, and then cycle through Rules 1 to 4 for each M to see whether it should be deleted.

Rule 0. Insert Ms in front of all Ks that are not part of argument strings proper (e.g., text strings or numbers). Place Ms in front of all Ps that select commands (not arguments).

Rule 1. If an operator following an M is *fully anticipated* in the operator just previous to M, then delete the M (e.g., PMK → PK).

Rule 2. If a string of MKs *belong to a cognitive unit* (e.g., the name of a command), then delete all Ms but the first.

Rule 3. If a K is a *redundant terminator* (e.g., the terminator of a command immediately following the terminator of its argument), then delete the M in front of the K.

Rule 4. If a K *terminates a constant string* (e.g., a command name), then delete the M in front of the K; but if the K terminates a variable string (e.g., an argument string), then keep the M.

Thus, we predict that the task will take about two seconds longer on POET than on DISPED. The accuracy of such predictions is discussed in Section 4.

The methods above are simple unconditional sequences. More complex or more general tasks are likely to have multiple methods and/or conditionalities within methods for accomplishing different versions of the task. For example, in a DISPED-like system the user often has to "scroll" the text on the display before being able to point to the desired target. We can represent this method as follows:

.4(**MP**[SCROLL-ICON] **K**[RED] **R**(.5)) **P**[word] **K**[YELLOW].

Here we assume a specific situation where the average number of scroll jumps per selection is .4 and that the average system response time for a scroll jump is .5 sec. From this we can predict the average selection time:

$$T_{execute} = .4t_M + 1.4t_K + 1.4t_P + .4(.5) = 2.6 \text{ sec.}$$

For more complex contingencies, we can put the operations on a flowchart and label the paths with their frequencies.

When there are alternative methods for doing a specific task in a given system, we have found [4] that expert users will, in general, use the most efficient method, i.e., the method taking the least time. Thus, in making predictions we can use the model to compute the times for the alternative methods and predict that the fastest method will be used. (If the alternatives take about the same time, it does not matter which method we predict.) The optimality assumption holds, of course, only if the users are familiar with the alternatives, which is usually true of experts (excepting the more esoteric alternatives). This assumption is helped by the tendency of optimal methods to be the simplest.

3.3 Heuristics for the M Operator

M operations represent acts of mental preparation for applying subsequent physical operations. Their occurrence does not follow directly from the method as

defined by the command language of the system, but from the specific knowledge and skill of the user. The Keystroke-Level Model provides a set of rules (Figure 2) for placing M's in the method encodings. These rules embody psychological assumptions about the user and are necessarily heuristic, especially given the simplicity of the model. They should be viewed simply as guidelines.

The rules in Figure 2 define a procedure. The procedure begins with an encoding that contains only the physical operations (**K, P, H,** and **D**). First, all candidate **M**'s are inserted into the encoding according to Rule 0, which is a heuristic for identifying all possible decision points in the method. Rules 1 to 4 are then applied to each candidate **M** to see if it should be deleted.

There is a single psychological principle behind all the deletion heuristics. Methods are composed of highly integrated submethods ("subroutines") that show up over and over again in different methods. We will call them *method chunks* or just *chunks*, a term common in cognitive psychology [17]. The user cognitively organizes his methods according to chunks, which usually reflect syntactic constituents of the system's command language. Hence, the user mentally prepares for the next chunk, not just the next operation. It follows that in executing methods the user is more likely to pause between chunks than within chunks. The rules attempt to identify method chunks.

Rule 1 asserts that when an operation is fully anticipated in another operation, they belong in a chunk. A common example is pointing with the mouse and then pressing the mouse button to indicate a selection. The button press is fully anticipated during the pointing operation, and there is no pause between them (i.e., **PMK** becomes **PK** according to Rule 1). This anticipation holds even if the selection indication is done on another device (e.g., the keyboard or a foot pedal). Rule 2 asserts that an obvious syntactic unit, such as a command name, constitutes a chunk when it must be typed out in full.

The last two heuristics deal with syntactic terminators. Rule 3 asserts that the user will bundle up redundant terminators into a single chunk. For example, in the POET example in Section 3.2, a RETURN is required to terminate the second argument and then another RETURN to terminate the command; but any user will quickly learn to simply hit a double RETURN after the second argument (i.e., **MKMK** becomes **MKK** according to Rule 3). Rule 4 asserts that a terminator of a constant-string chunk will be assimilated to that chunk. The most common example of this is in systems that require a terminator, such as RETURN, after each command name; the user learns to immediately follow the command name with RETURN.

It is clear that these heuristics do not capture the notion of method chunks precisely, but are only rough approximations. Further, their application is ambiguous in many situations, e.g., whether something is "fully anticipated" or is a "cognitive unit." What can we do about this ambiguity? Better general heuristics will help in reducing this ambiguity. However, some of the variability in what are chunks stems from a corresponding variability in expertness. Individuals differ widely in their behavior; their categorization into *novice, casual,* and *expert* users provides only a crude separation and leaves wide variation within each category. One way that experts differ is in what chunks they have (see [6] for related evidence). Thus, some of the difficulties in placing **M**'s is unavoidable because not enough is known (or can be known in practical work) about the experts involved. Part of the variability in expertness can be represented by the Keystroke-Level Model as encodings with different placements of **M** operations.

4. Empirical Validation of the Keystroke-Level Model

To determine how well the Keystroke-Level Model actually predicts performance times, we ran an experiment in which calculations from the model were compared against measured times for a number of different tasks, systems, and users.

4.1 Description of the Experiment

A total of 1,280 user-system-task interactions were observed, comprised of various combinations of 28 users, 10 systems, and 14 tasks.

Systems. The systems were all typical application programs available locally (at Xerox PARC) and widely used by both technical and nontechnical users. Some of the systems are also widely used nationally. Three of the systems were text editors, three were graphics editors, and five were executive subsystems. The systems are briefly described in Figure 3.

Together, these systems display a considerable diversity of user interface techniques. For example, POET, one of the text editors, is a typical line-oriented system, which uses first-letter mnemonics to specify commands and search strings to locate lines. In contrast, DRAW, one of the graphics systems, displays a menu of graphic icons on the CRT display to represent the commands, which the user selects by pointing with the mouse.

Tasks. The 14 tasks performed by the users (see Figure 4) were also diverse, but typical. Users of the editing systems were given tasks ranging from a simple word substitution to the more difficult task of moving a sentence from the middle to the end of a paragraph. Users of the graphics systems were given tasks such as adding a box to a diagram or deleting a box (but keeping a line which overlapped it). Users of the executive subsystems were given tasks such as transferring a file between computers or examining part of a file directory.

Task-system methods. In all there were 32 task-system combinations: $4 \times 3 = 12$ for the text editors, $5 \times 3 = 15$ for the graphics systems, and one task each for the five

executive subsystems. For each task-system combination, the most efficient "natural" method was determined (by consulting experts) and then coded in Keystroke-Level Model operations. For example, the methods for T1-POET and T1-DISPED are given in Section 3.2. (A complete listing of all the methods can be found in [3].)

Experimental design. The basic design of the experiment was to have ten versions of each task on each system done by four different users, giving 40 observed instances per task-system. No user was observed on more than one system to avoid transfer effects. Four tasks were observed for each of the text-editing systems, five tasks for each of the graphics systems, and one task for the executive subsystems.

Subjects. There were in all 28 different users (some technical, some secretarial): 12 for the editing systems, 12 for the graphics systems, and 4 for the executive subsystems. All were experts in that they had used the systems for months in their regular work and had used them recently.

Experimental procedure. Each user was first given five one-minute typing tests to determine his keystroke time, t_K. In addition, users of MARKUP (the only system requiring manual drawing) were given a series of drawing tasks to determine the parameters of their drawing rate (as discussed in Section 3.1).

After the preliminary tests, the user was given a small number of practice problems of the sort to be tested and was told the method to use (see above). In most cases, the methods presented were those users claimed they would have used anyway; in other cases, the method was easily adopted. Users practiced tasks until they were judged to be at ease with using the correct method; this was usually accomplished in three or four practice trials on each task type.

After practicing, the user proceeded to the main part of the experiment. The user was given a notebook containing several manuscript pages with the tasks to be done marked in red ink. Text-editing and graphics tasks appeared in randomized order. Executive subsystem tasks were always in the order T11, T12, T13, T14. All ten instances of task T10 were done in succession.

Each experimental session, lasting approximately 40 minutes, was videotaped and the user's keystrokes recorded automatically. Time stamps on the videotaped record and on each keystroke allowed protocols to be constructed in which the time of each event was known to within .033 sec. These protocols are the basic data from which the results below are derived.

4.2 Results of the Experiment

Each task instance in the protocols was divided into acquisition time and execution time (see Section 2.1) according to the following definitions. Acquisition time began when the user first looked over to the manuscript to get instructions for the next task and ended when the user started to perform the first operator of the method. Execution time began at that point and ended when the

Fig. 3. Systems measured in the experiment.

System	Description
Text Editors	
POET[a]	Line-oriented with relative line numbers.
SOS[b]	Line-oriented with "sticky" line-numbers.
DISPED[e]	Display-oriented; full-page; uses mouse for pointing.
Graphics Systems	
MARKUP[c]	Uses mouse to draw and erase lines on a bitmap display; commands selected from a hidden menu, which must be re-displayed each time.
DRAW[e]	Lines defined by pointing with mouse to end points; commands selected with mouse from a menu.
SIL[e]	Lines defined by pointing with mouse to end points; boxes defined by pointing to opposite vertices; commands selected by combinations of mouse buttons.
Executive Subsystems	
LOGIN[d]	TENEX command for logging in.
FTP[e]	Program for transferring files between computers.
CHAT[e]	Program for establishing a "teletype" connection between two computers.
DIR[d]	TENEX command for printing a file directory; has a subcommand mode.
DELVER[d]	TENEX command for deleting old versions of a file.

[a] POET is a dialect of the QED editor [7].
[b] See [16].
[c] See [13, ch. 17].
[d] See [12].
[e] Experimental systems local to Xerox PARC, designed and implemented by many individuals, including: Roger Bates, Patrick Baudelaire, David Boggs, Butler Lampson, Charles Simonyi, Robert Sproull, Edward Taft, and Chuck Thacker.

Fig. 4. Tasks for the experiment.

Editing Tasks (used for POET, SOS, DISPED)

- T1. Replace one 5-letter word with another (one line from previous task).
- T2. Add a 5th character to a 4-letter word (one line from previous task).
- T3. Delete a line, all on one line (eight lines from previous task).
- T4. Move a 50-character sentence, spread over two lines, to the end of its paragraph (eight lines from previous task).

Graphics Tasks (used for MARKUP, DRAW, SIL)

- T5. Add a box to a diagram.
- T6. Add a 5-character label to a box.
- T7. Reconnect a 2-stroke line to a different box.
- T8. Delete a box, but keep an overlapped line.
- T9. Copy a box.

Executive Tasks

- T10. Phone computer and log in (4 char name, 6 char password).
- T11. Transfer a file to another computer, renaming it.
- T12. Connect to another computer.
- T13. Display a subset of the file directory with file lengths.
- T14. Delete old versions of a file.

user looked over to the notebook for the next task. (On the protocol the first measured time at the beginning of an execution is always the end of the first **K** of the method. Thus, operationally, the beginning of execution time was estimated by subtracting from this first **K** time the operator times for this first **K** plus all the operators that preceded it.)

Those tasks on which there were significant errors (i.e., other than typing errors) or in which the user did not use the prescribed method were excluded from further consideration. After this exclusion, 855 (69 percent) of the task instances remained as observations to be matched against the predictions. No analysis was made of the excluded tasks.

The resulting observed times for task acquisition and execution were stable over repetition. There was no statistical evidence for task times decreasing (learning) or increasing (fatigue) with repetition.

4.2.1 Calculation of execution time. Execution time was calculated using the method analysis for each task-system combination together with estimates of the times required for each operator. All times, except for the mental preparation time, were taken from sources outside of the experiment. Pointing time, t_P, and homing time, t_H, were taken from Figure 1. Typing time, t_K, and drawing time, $t_D(n_D, l_D)$, were estimated from the typing and drawing tests by averaging the times of the four users involved in each task-system. System response time, T_R, for each task-system was estimated from independent measurements of the response times for the various commands required in each method. For task T10, logging in to a computer, a telephone button-press was assumed to take time t_K. Moving the telephone receiver to the computer terminal modem was estimated to take .7 sec, using the MTM system of times for industrial operations [10].

Mental preparation time, t_M, was estimated from the experimental data itself. First, the total mental time for each method was estimated by removing the predicted time for all physical operations from the observed execution time. Then t_M was estimated by a least-squares fit of the estimated mental times as a function of the predicted number of **M** operations. The result was $t_M = 1.35$ sec ($R^2 = .84$, standard error of estimate = .11 sec, standard error about the regression line = 2.48 sec). A rough estimate of the SD of t_M is 1.1 sec, which indicates that the **M** operator has the characteristic variability of mental operators [4].

Execution times for each task-system combination were calculated by formula (1) in Section 3. The calculations of the execution times are summarized in Figure 5, which also gives the observed execution times from the experiment for comparison.

4.2.2 Execution time. The predicted execution times are quite accurate. This can be seen in Figure 6, which plots the predicted versus the observed data from Figure 5. The scales are logarithmic, since prediction error appears to be roughly proportional to duration. The root-mean-square (RMS) error is 21 percent of the average predicted execution time. This accuracy is about the best that can be expected from the Keystroke-Level Model, since the methods used by the subjects were controlled by the experimental procedure. The 21 percent RMS error is comparable to the 20–30 percent we have obtained in other studies on text-editing with more elaborate models that also predict the method [4].

The distribution of percentage prediction errors is fairly evenly spread, as an analysis of Figure 6 will show. No particular systems or tasks make excessively large contributions. Predictions are not consistently positive or negative for systems or tasks, except that all the executive subsystem tasks were overpredicted. Examination of the individual observations does not reveal any small set of outliers or particular users that inflate the prediction error.

Prediction accuracy is related to the duration of the attempted prediction. The results above are for individual unit tasks. Since unit tasks are essentially independent, prediction of the time to do a sequence of tasks will tend to be more accurate. This can be seen directly in the present data, since each user ran all the tasks for a given system. For example, consider predicting by the model how long it took to do *all four* editing tasks. The average RMS error is only 5 percent. The corresponding RMS error for the graphics editors over the five tasks is only 6 percent.

Ideally, all of the parameters of the model should be determined independently of the experimental situation. This was achieved for all the physical operation times, but not for the mental operation time, t_M. We did not have available an appropriate independent source of data from which to determine t_M. The accuracy of the model is somewhat inflated by the determination of one of its parameters from the data itself. The substantial variability of t_M indicates that this inflation is probably not too serious, which is to say that small changes in the value of t_M do not make much difference. For example, if a t_M as small as 1.2 sec or as large as 2.0 sec were used in the predictions, the RMS error for the Keystroke-Level Model would only increase from 21 to 23 percent. It should be noted that the t_M estimated from this experiment is now available as an independent estimate for use by others.

The variability of the observed task times is of interest per se, since user behavior is inherently variable. In our data the average coefficient of variation (CV = SD/Mean) of the individual observations over each task is .31, which is the normal variability for behavior of this duration [4]. In comparing the predictions of the model against any actual behavior, the prediction error will always be confounded with some error from the process of sampling the behavior. The sampling error for each of our observed task times is indicated in the SE column of Figure 5. The average standard error is 9 percent.

Fig. 5. Calculated and Observed Execution Times in the Experiment.

Task-System	Calculated[a]								Observed		Pred. Error[d]
	t_K[b]	n_M	n_K	n_H	n_P	n_D	l_D	T_R	$T_{execute}$	$T_{execute}$ $M \pm SE(N)$[c]	
	(sec)						(cm)	(sec)	(sec)	(sec) (sec)	
T1-POET	.23	4	15	..	..	..	..	..	8.8[e]	7.8 ± 0.9(27)	11%
T1-SOS	.22	4	19	..	..	..	..	..	9.6	9.6 ± 0.8(31)	1%
T1-DISPED	.23	2	8	2	1	..	..	..	6.4[e]	5.7 ± 0.3(31)	11%
T2-POET	.28	4	14	..	..	..	..	..	9.4	8.9 ± 0.7(17)	5%
T2-SOS	.23	4	18	..	..	..	..	..	9.5	9.7 ± 0.8(32)	- 3%
T2-DISPED	.24	2	4	2	1	..	..	..	5.6	4.1 ± 0.3(32)	26%
T3-POET	.19	3	12	..	..	..	..	..	6.3	6.3 ± 0.4(24)	0%
T3-SOS	.23	2	7	..	..	..	..	..	4.3	4.0 ± 0.3(37)	8%
T3-DISPED	.23	1	2	1	1	..	..	..	3.3	3.5 ± 0.2(38)	- 7%
T4-POET	.19	13	92	..	..	..	..	..	35.3	37.1 ± 4.3(20)	- 6%
T4-SOS	.23	12	47	..	..	..	..	..	26.8	32.7 ± 1.8(16)	-22%
T4-DISPED	.24	2	6	1	3	..	..	3.8	11.6	14.3 ± 1.1(33)	-23%
T5-MARKUP	.25	..	3.2	..	2.5	4	24.9	..	11.1	10.5 ± 1.1(27)	6%
T5-DRAW	.25	7.6	12.6	..	5	..	..	..	18.9	12.5 ± 3.0(22)	34%
T5-SIL	.27	1	4	0.4	2	..	..	..	4.8	5.4 ± 0.7(32)	-12%
T6-MARKUP	.26	1	7	2	1	..	..	..	5.0	6.2 ± 0.4(34)	-23%
T6-DRAW	.25	1	7	1	1	..	..	..	4.6	5.9 ± 0.4(34)	-29%
T6-SIL	.27	..	6	1.4	1	..	..	..	3.3	3.6 ± 0.3(19)	- 9%
T7-MARKUP	.24	..	8.6	..	4.8	6	13.6	..	15.1	15.0 ± 2.1(29)	2%
T7-DRAW	.19	5	13	..	8	..	..	..	18.0	18.2 ± 1.9(9)	- 1%
T7-SIL	.28	1	8	..	5	..	..	..	9.1	12.3 ± 2.1(23)	-36%
T8-MARKUP	.26	..	8	..	8	1	4.0	..	12.3	9.3 ± 0.4(22)	24%
T8-DRAW	.21	1	5	..	3	..	..	..	5.7	5.3 ± 0.3(25)	7%
T8-SIL	.27	1	5	0.7	2	..	..	..	5.2	4.1 ± 0.2(33)	20%
T9-MARKUP	.25	2	8	..	6.5	..	..	3.5	15.4	13.0 ± 2.5(26)	15%
T9-DRAW	.22	..	5.7	..	5.7	..	..	..	7.5	10.5 ± 1.0(25)	-40%
T9-SIL	.28	..	5	0.3	3	..	..	..	4.8	6.0 ± 1.0(28)	-24%
T10-LOGIN	.29	2	28	..	..	..	..	15.9	27.4[f]	25.1 ± 0.7(29)	9%
T11-FTP	.30	5	31	..	..	..	..	10.1	26.1	19.7 ± 0.7(29)	24%
T12-CHAT	.31	1	11	..	..	..	..	8.3	13.1	11.5 ± 0.6(36)	12%
T13-DIR	.30	2	20	..	..	..	..	0.5	9.2	6.6 ± 0.3(32)	28%
T14-DELVER	.32	2	20	..	..	..	..	0.4	9.4	7.5 ± 0.4(33)	20%

[a] The calculations are done according to formula (1) using the operator times in Figure 1, except for t_K.

[b] t_K is the average time from the typing tests for the subjects on a given system. Each subject's time is weighted by the correct number of instances for that subject on a given task (see Section 4.2.1).

[c] SE is the standard error of estimation of the population mean for samples of size N.

[d] The prediction error is given as a percentage of the calculated time, $T_{execute}$.

[e] The calculated times for these tasks are different from the calculated times in the examples in Section 3.2, because different t_K are used.

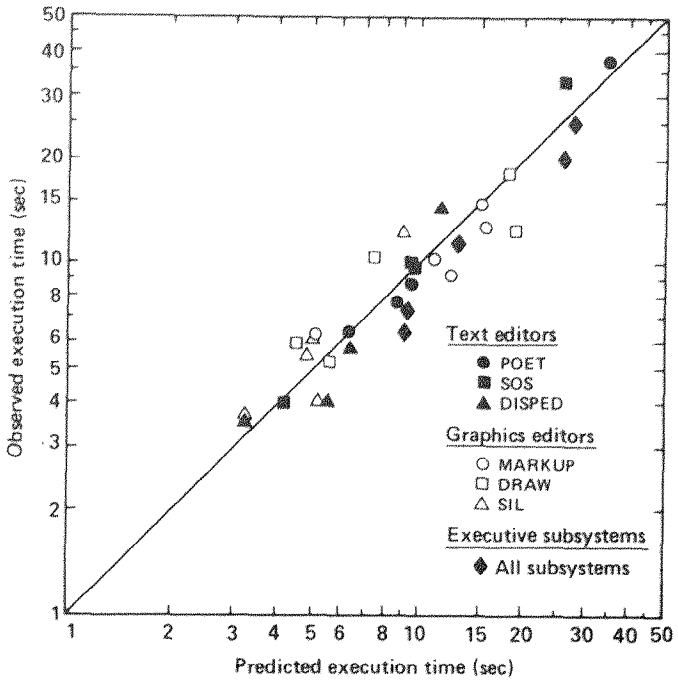
[f] The execute time for this task also includes .7 sec for the operation of moving the telephone receiver (see Section 4.2.1).

That the prediction error of the Keystroke-Level Model is over two times larger than this indicates that most of the prediction error is due to the inaccuracy of the model and not just unreliable observations.

4.2.3 Acquisition time. Turning from the execution part of the task to the acquisition part, the data shows that it took users 2 sec, on the average, to acquire a task from the manuscript. This number may be refined by breaking the tasks into three types: (a) those tasks that the user already had in memory (the executive subsystem tasks that were done each time in the same order); (b) those tasks for which the user had to look at the manuscript each time (all the graphics tasks, the POET and SOS tasks, and task T11); and (c) those tasks for which the user had to look at the manuscript, then scan text on the CRT to locate the task. The times for these three types of acquisition are given in Figure 7. Users took only .5 sec when the task was in memory, 1.8 sec to get the task from the manuscript, and 4.0 sec to get the task from the manuscript and scan the CRT. These times are similar to results obtained in previous experiments [4]. It is interesting to note that, although display editors are generally faster to use, they impose a 2 sec penalty by requiring the user to visually scan the text on the display.

We can use the acquisition times in Figure 7, along with the predicted execution times in Figure 5, to predict the total task times. The RMS error of these predictions is 21 percent, which is just as accurate as predicting the execution times alone.

Fig. 6. Predicted vs. observed execution times in the experiment.



5. Sample Applications of the Keystroke-Level Model

The experiment has provided evidence for the Keystroke-Level Model in a wide range of user-computer interactions. Given the method used, the time required for experts to perform a unit task can be predicted to within about 20 percent by a linear function of a small set of operators. This result is powerful in permitting prediction without having to do any measurements of the actual situation and in expressing the prediction as a simple algebraic expression. Its limitation lies in requiring that the method be completely specified at the level of keystrokes and in being limited to error-free expert behavior.

In this section we illustrate how the Keystroke-Level Model can be used, both to exploit its power and to work within its restrictions. The basic application—to predict a time for a specific situation by writing down a method and computing the value—has been sufficiently illustrated in the course of the experiment, where such point predictions were made for 32 different tasks involving 10 highly diverse systems. We now show three further uses: (1) calculated benchmarks for systems; (2) parametric analysis, where predictions are expressed as functions of task variables; and (3) sensitivity analysis, where changes in the predictions are examined as a function of changes in task or model parameters.

5.1 Calculated Benchmarks

Given the ability to predict tasks, it is possible to calculate the equivalent of a benchmark for a system and hence to compare systems. This has obvious cost advantages over obtaining actual measurements. More importantly, it permits benchmarking at design time, before the system exists in a form that permits measurement. The analysis for the experimental data lets us illustrate this easily.

Consider the three text editors, POET, SOS, and DISPED. Let the benchmark be the four tasks T1 to T4. We can use the Keystroke-Level Model to compute the total time to do the benchmark for each system. The answer comes directly from Figure 5 by summing the calculated $T_{execute}$ for T1–T4 for each editor. This gives 59.8 sec, 50.2 sec, and 26.9 sec as the predicted execution times, respectively, for POET, SOS, and DISPED. Taking the POET time (the slowest) as 100, we get ratios of 100:84:45. Thus, as we might have expected, the two line-oriented editors are relatively close to each other and the display editor is substantially faster. Since we have also done the experiment, we can compare these calculated benchmarks with the observed benchmarks (by summing the observed $T_{execute}$ from Figure 5). We get 60.1 sec, 56.0 sec, and 27.6 sec, respectively. This gives experimentally determined ratios 100:93:46, which is essentially the same result. This agreement between the calculated and observed benchmark provides confidence only in using the calculated benchmark in place of a measured one. It does not provide evidence for the validity of the particular benchmark (tasks T1–T4) or whether benchmarks are generally a valid way to compare editors.

A similar analysis can be performed for the three graphics systems, using tasks T5–T9 as the benchmark. This yields predicted ratios of 100:93:46 for MARKUP, DRAW, and SIL, respectively, with observed ratios of 100:97:58. MARKUP and DRAW are close enough to raise the question of whether the predicted difference between them is too small to be reliable. The calculated difference between MARKUP and DRAW on the benchmark is 59.0 −54.7 = 4.3 sec or 7 percent. The model has an RMS prediction error of 21 percent for a single unit task. Since this benchmark is essentially an independent sum of five unit tasks, the RMS error should theoretically be 21 percent/SQRT(5) = 9 percent. Thus, the predictions for the two systems are within the RMS error of the model, and so the predicted difference between them can hardly

Fig. 7. Observed acquisition times in the experiment.

Task Type	Task numbers	Acquisition Time $M \pm SE^a$ (N) (sec) (sec)
All tasks	T1–T14	2.0 ± 2.0 (885)
Repeated task, recalled from memory	T10, T12, T13, T14	0.5 ± 0.3 (130)
Task acquired by looking at manuscript	T1–T4 (POET, SOS), T5–T9, T11	1.8 ± 1.9 (621)
Task acquired by looking at manuscript, then scanning for task on display	T1–T4 (DISPED)	4.0 ± 1.9 (134)

[a] SE is the standard error of estimation of the population mean for samples of size N.

be reliable. The fact that the model correctly predicted that DRAW was slightly faster than MARKUP was lucky—there is no reason to expect the Keystroke-Level Model to make such close calls.

5.2 Parametric Analysis

We illustrate the notions of parametric analysis and sensitivity analysis with a new example. Consider the following task: A user is typing text into an editor and detects a misspelled word n words back from the word he is currently typing. How long will it take to correct the misspelled word and resume typing?

In DISPED there are two methods for making the correction, which we wish to compare. Since the methods may behave quite differently depending on how far back the misspelled word is, we need to determine how long each method takes as a function of n. The first method makes use of the Backword command (called by hitting the CTRL key and then W), which erases the last typed in word:

Method W (Backword):

Set up Backword command	**MK**[CTRL]
Execute Backword n times	$n((1/c)$**MK**[W])
Type new word	5.5**K**[word]
Retype destroyed text	$5.5(n-1)$**K**

$$T_{execute} = (1 + n/c)t_M + (1 + 6.5n)t_K = 1.6 + 2.16n \text{ sec.} \quad (2)$$

The execution time is a function not only of n, but also of another parameter, c. When a user has to repeat a single-keystroke command several times, such as the Backword command in the above method, he will tend to break the sequence into small bursts or *chunks*, separated by pauses, which are represented as **M** operations, according to Rule 2 in Figure 2. Thus, we postulate a chunk size, c, which is the average number of Backword commands in a burst. This is used in the second step in the above method, where we count $1/c$ **M** operations for each call of the Backword command. An exact value for c is unknown, but we use a "reasonable" value, $c = 4$, in our calculations (we will return to this decision in Section 5.3). In the calculations we also assume an average nonsecretary typist ($t_K = .28$ sec) and an average word length of 5.5 characters (including punctuation and spaces).

The second method is to get out of type-in mode, use the Replace command to correct the word, and then get back into type-in mode, so that input can be resumed:

Method R (Replace):

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

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

The predicted time for each method as a function of n is plotted as the solid lines in Figure 8(a). As the figure shows, it is faster to use the Backword method up until a certain crossover point, n_{WR}, after which it becomes faster to use the Replace method. Under the above assumptions, the crossover from the Backword method to the Replace method is found to be at 4.9 words.

Suppose a designer wants to add a feature to DISPED to improve performance on this task. We wish to determine, *before* implementation, whether the proposed feature is likely to be much of an improvement.

The designer proposes two new commands. The first is a Backskip command (CRTL S), which moves the insertion point back one word without erasing any text. The second is a Resume command (CTRL R), which moves the insertion point back to the end of the current type-in (where Backskip was first called). These commands allow:

Method S (Backskip):

Set up Backskip command	**MK**[CTRL]
Execute Backskip $n-1$ times	$(n-1)((1/c)$**MK**[S])
Call Backword command	**MK**[W]
Type new word	4.5**K**[word]
Call Resume command	M2**K**[CTRL R]

$$T_{execute} = (3 + (n-1)/c)t_M + (n + 7.5)t_K = 5.8 + .62n \text{ sec.} \quad (3)$$

The predicted time for the Backskip method is plotted as the dashed line in Figure 8(a). With the addition of this method there are two additional crossover points, n_{WS} and n_{RS}, between it and the other two methods. As can be seen, the Backskip method is faster than both of the other methods between n_{WS} and n_{RS}, i.e., from 2.7 to 10.2 words. Thus, a brief analysis provides evidence that the proposed new feature probably will be useful, in the sense that it will be the fastest method over a region of the task space.

5.3 Sensitivity Analysis

How sensitive to variations in the parameters of the methods are the aforementioned calculations? The question of interest is whether, over such variations, there is still a region in the task space in which the Backskip method is the fastest. An important parameter is the user's typing speed, t_K. How much does the crossover between the Backword method and the Backskip method change as a function of typing speed? Setting eq. (2) equal to (3) and solving for n as a function of t_K gives $n = 1.2 + .43/t_K$. The crossover increases with faster typists (decreasing t_K), going up to $n = 6.6$ words for the fastest typist ($t_K = .08$ sec). That is to say, faster typists should prefer the old Backword method (which involves more typing) for larger n before switching to the new Backskip method (which involves less typing, but more mental overhead).

We can plot the crossover boundary between the two methods in the space of the two parameters: n (characterizing different tasks) and t_K (characterizing different users). The two boundaries of the new Backskip method are plotted in Figure 8(b). These boundaries define the

Fig. 8(a). Execution time for three methods as a function of n.

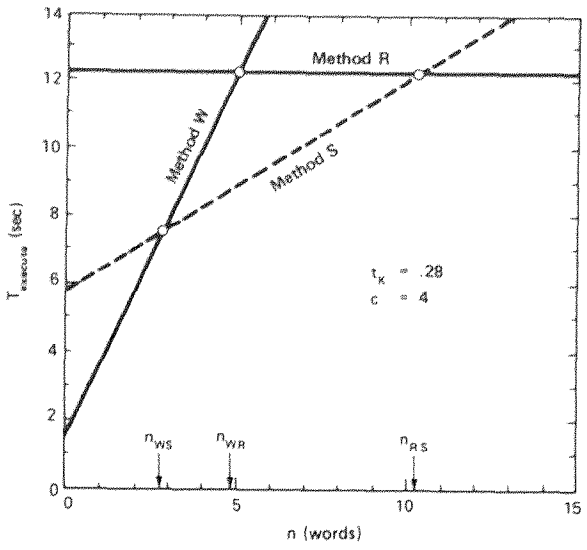


Fig. 8(b). Phase diagram for the fastest method.

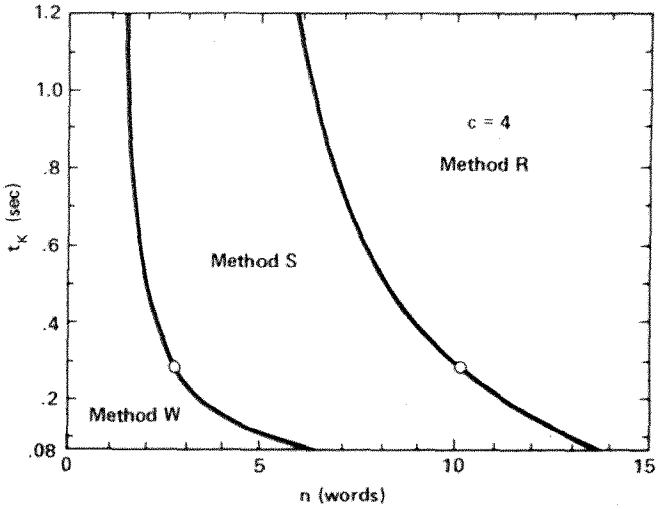
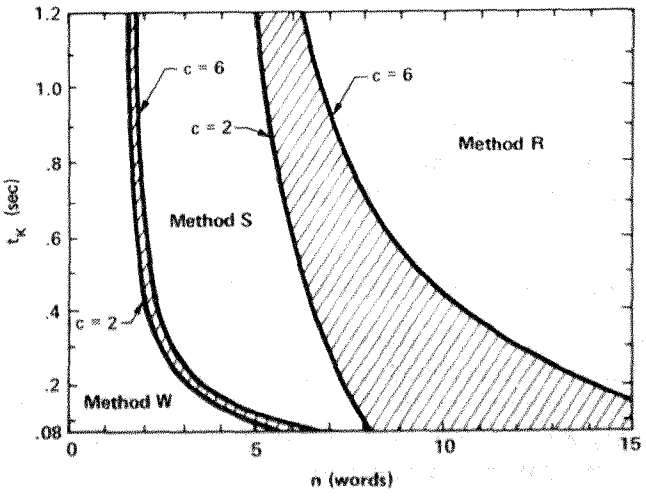


Fig. 8(c). Phase diagram adjusted for different chunk sizes.



regions in the parameter space where each method is fastest. The circles mark the crossover points corresponding to the ones in Figure 8(a) (i.e., at $t_K = .28$ sec). This diagram clearly shows the shift of crossovers for fast typists. It also shows that, for any speed of typist, there are some tasks for which the Backskip method is the fastest.

We are not sure of the exact chunk size, c, and so we must check whether our conclusions about the usefulness of the Backskip method are sensitive to the choice of a value for c. To do this, we rederive the crossover between the Backword and Backskip methods by setting eq. (2) equal to (3) and solving for n as a function of both c and t_K; this gives $n = 1.2 + .49/t_K - .24/ct_K$. Although we do not know an exact value for c, we can be reasonably confident that it will be between 2 and 6. With $t_K = .28$ sec, for example, the crossover varies between 2.5 and 2.8 words as c varies between 2 and 6; so the value of c does not seem to have a great effect at this point.

The best way to see the overall affect of the value of c is to replot Figure 8(b) using the reasonable extreme values of c. The two crossover boundaries for the Backskip method are plotted in Figure 8(c) as "fat" lines defined by setting c to 2 and 6 in the crossover equations. This diagram clearly shows that the value of c affects one boundary more than the other. The boundary between the Backword and Backskip methods is not affected much by c, because the chunk size is involved in both methods in exactly the same way. But the boundary between the Backskip and Replace methods is greatly affected by the value of c, since c is not involved in the Replace method at all. Small chunk sizes, especially, penalize the Backskip method. Overall, however, varying c does not squeeze out the region for the Backskip method; and our basic conclusion—that the new method is a useful addition—still holds.

There are other aspects of the above methods for which we could do a sensitivity analysis. (For example, if the last two **M** operations of the Backskip method were eliminated according to Rule 1, how much would the value of the Backskip method increase?) However, the sensitivity analyses above illustrate how the Keystroke-Level Model can be used to evaluate design choices—even when many aspects of the calculation are uncertain—for the principal conclusions are often insensitive to many of the uncertainties.

6. Simplifications of the Keystroke-Level Model

The question naturally arises as to whether further simplifications of the Keystroke-Level Model might do reasonably well at predicting execution time. One could (a) count only the number of keystrokes, (b) count just the physical operators and prorate the time for mental activity, or (c) use a single constant time for all operators. We show below that such simplifications substantially degrade accuracy. However, they provide useful approximations where the lowered accuracy can be tolerated.

6.1 Keystrokes Only

With this simplification, execution time is proportional to the number of keystrokes:

$$T_{execute} = \kappa n_K + T_R.$$

We separate out the system response times, T_R, so as not to confound the comparison. The constant of proportionality, κ, should be distinguished from t_K, the typing speed, which is determined from standard typing tests. Estimating the value of κ from a least-squares fit of the values of n_K and the observed $T_{execute}$ in Figure 5 gives $\kappa = .49$ sec/keystroke. The correlation between the times predicted by this model and the observed times is .87, and the RMS error is 49 percent. The statistics for comparing all models are presented in Figure 9. As can be seen, using keystrokes only is substantially less accurate than the full Keystroke-Level Model. This simplification is inappropriate for tasks that are not dominated by keystroking. For example, it only predicts about a third of the observed time for the MARKUP tasks, which are dominated by pointing and drawing operations.

The above estimate of κ is held down by one outlying point in the data, T4-POET ($n_K = 92$). Estimating κ with this one point removed gives $\kappa = .60$ sec, a value close to another estimate obtained in an earlier benchmark study [1; 5, ch. 3]. T4-POET is the only task that requires any input-typing of text. One obvious refinement of the keystrokes-only model would be to distinguish two kinds of keystrokes: mass input-typing (at t_K sec/keystroke) versus command-language keying (at κ sec/keystroke). For this purpose, a κ of .60 sec is the more reasonable value.

The model of Embley et al. [9] is formally similar to our keystrokes-only version. However, their model is conceptually different from ours. The Keystroke-Level Model is based on the notion of a unit task structure; Embley et al. use commands instead. Our model is restricted to skilled expert behavior, whereas they attempt to model all kinds of users (essentially, by varying their versions of the parameters $T_{acquire}$ and κ). Unfortunately, they did not compare their model against any empirical performance data, so we cannot compare our results to theirs. The keystrokes-only model can, perhaps, be taken as an indicator of the accuracy of their model for expert behavior.

6.2 Prorated Mental Time

According to this simplification, execution time is the time required for the physical operations multiplied by a factor to account for the mental time:

$$T_{execute} = \mu(T_K + T_H + T_P + T_D) + T_R.$$

The idea is that the physical operations will require a certain average overhead of mental activity. Thus, instead of trying to predict exactly how many mental operations there are, we can do fairly well by just using a multiplicative mental overhead constant, μ.

Using a least-squares analysis to determine μ from the sum of the calculated times for the physical operations and the observed values of $T_{execute}$ in Figure 5 gives $\mu = 1.67$; i.e., there is a 67 percent overhead for mental activity. The correlation between predicted and observed times is .81, and the RMS error is 45 percent.

This simplification is also less accurate than the Keystroke-Level Model, as can be seen in Figure 9. This suggests that the extra detail in the Keystroke-Level Model, involving the placements of the mental preparedness operator, **M**, is effective. It is this operator that qualifies the Keystroke-Level Model as a genuine psychological model and not simply as an analysis of the physical operations.

There is an interesting relation between these two simpler models and the rules for placing occurrences of **M** in the Keystroke-Level Model (Figure 2). The initial placement of **M**'s, by Rule 0, with certain **K**'s and **P**'s is essentially an assumption that mental time is proportional to a subset of the physical operators. If Rule 0 had specified all physical operators, Rule 0 by itself would have been equivalent to prorating mental time. If the other physical operators (**P**, **H**, and **D**) had been ignored, this would have been equivalent to counting keystrokes only. Therefore, the deletion of the **M**'s according to Rules 1 to 4 constitutes the ways in which the Keystroke-Level Model departs from these simpler models. The evidence for the superiority of the Keystroke-Level Model presented in Figure 9 is also evidence that Rules 1 to 4 had a significant effect. In fact, each of the rules individually makes a significant contribution, in the sense that its removal leads to a decrease in the accuracy of the Keystroke-Level Model.

6.3 Constant Operator Time

According to this simplification, execution time is proportional to the *number* of Keystroke-Level operations:

$$T_{execute} = \tau(n_M + n_K + n_P + n_H + n_D) + T_R.$$

The idea here is the statistical observation [18] that the accuracy of linear models is not sensitive to the differential weighting of the factors—equal weighting does nearly as well as any other weighting. Thus, we disregard the different operator times and use a single time, τ, for

Fig. 9. Comparison of the keystroke model with simpler variations.

Model Variation	Parameters	Correlation (r)[a]	RMS Error[b]
Keystrokes Only	κ = .49 sec/keystroke[c]	.87	49%
Prorated Mental Time	μ = 1.67	.81	45%
Constant Operator Time	τ = .43 sec/operator[c]	.92	34%
Keystroke Model	(See Figure 1)	.95	22%

[a] The correlations are between the execution times predicted by each of the models and the observed execution times from Figure 5.

[b] The RMS error is given as a percentage of the observed execution time, 11.0 sec.

[c] More useful parameter values are κ = .60 sec and τ = .49 sec (see Sections 6.1 and 6.3).

all operators. Note that the constant-operator-time model is formally similar to the keystrokes-only model; the latter can be viewed as using n_K as a crude estimate of the total number of operators.

Estimating τ by a least-squares fit of the data in Figure 5 gives τ = .43 sec/operator. The correlation between predicted and observed times is .92, and the RMS error is 34 percent. (For the reason discussed in Section 6.1, it is useful to estimate τ with the T4-POET task removed, getting τ = .49 sec/operator.)

The constant-time model is quite a bit more accurate than the keystrokes-only model, which tells us that taking into account operators other than **K** is useful. In fact, most of the action in the constant-time model (over the set of data in Figure 5, at least) comes from counting only the **K**, **P**, and **M** operators. In any particular task, of course, any of the operators can be dominant. On the other hand, the constant-time model is still less accurate than the Keystroke-Level Model, showing that taking into account accurate estimates of each operator's time yields another increment of accuracy.

In summary, all of the simplifications presented in this section are less accurate than the Keystroke-Level Model. However, these simplified models are probably good enough for many practical applications, especially for "back-of-the-envelope" calculations, where it is too much trouble to worry about the subtleties of counting the **M**'s that the full Keystroke-Level Model requires.

7. Conclusion

We have presented the Keystroke-Level Model for predicting the time it will take a user to perform a task using a system. We view this model as a *system design tool.* We have shaped it with two main concerns in mind. First, the tool must be quick and easy to use, if it is to be useful *during the design* of interactive systems. The existing strengths of psychology and human factors methods are primarily in the design and analysis of experiments; but experiments are too slow and cumbersome to be incorporated into practice. Ease of use implies that the tool be analytical—that it permit calculation in the style familiar to all engineers. Second, the tool must be useful to practicing computer system designers, who are not psychologists. This implies that the entire tool must be packaged to avoid requiring specialized psychological knowledge. We think that the Keystroke-Level Model satisfies these concerns, along with the primary consideration of being accurate enough to make design decisions. We believe that the Keystroke-Level Model belongs in the system designer's tool-kit.

It is possible to formulate more complicated and refined models than the Keystroke-Level Model by increasing its accuracy or by relaxing some of its serious restrictions (e.g., models that predict methods or that predict errors). One of the great virtues of the Keystroke-Level Model, from our own perspective as scientists trying to understand how humans interact with computer systems, is that it puts a lower bound on the effectiveness of new proposals. Any new proposal must do better than the Keystroke-Level Model (improve on its accuracy or lessen its restrictions) to merit serious consideration.

The Keystroke-Level Model has several restrictions: The user must be an expert; the task must be a routine unit task; the method must be specified in detail; and the performance must be error-free. These restrictions are important and must be carefully considered when using the model. Yet, we believe that the Keystroke-Level Model model represents an appropriate idealization of this aspect of performance and that it is a flexible tool allowing the system designer to deal systematically with this aspect of behavior.

The Keystroke-Level Model predicts only one aspect of the total user-computer interaction, namely, the time to perform a task. As we discussed at the beginning of this paper, there are many other important aspects of performance, there are nonexpert users, and there are nonroutine tasks. All of these must be considered by the system designer. Designing for expert, error-free performance time on routine tasks will not satisfy these other aspects. We would like to see appropriate models developed for these other aspects. However, even with a collection of such models, the designer still must make the inevitable trade-offs. Scientific models do not eliminate the design problem, but only help the designer control the different aspects.

Acknowledgments. We thank J. Farness, who ran the experiments described in this report, and T. Roberts, who helped in some of our early explorations of the keystroke-counting idea and provided extensive comments on earlier drafts of this paper.

Received 3/79; revised 2/80; accepted 3/80

References

1. Card, S.K. Studies in the psychology of computer text-editing systems. Ph.D. Th., Dept. of Psychol., Carnegie-Mellon Univ., Pittsburgh, Pa., May 1978.
2. Card, S.K., English, W.K., and Burr, B.J. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics 21* (1978), 601–613.
3. Card, S.K., Moran, T.P., and Newell, A. The keystroke-level model of user performance time with interactive systems. Rep. SSL-79-1, Xerox, Palo Alto Res. Ctr., Palo Alto, Ca., March 1979.
4. Card, S.K., Moran, T.P., and Newell, A. Computer text-editing: An information-processing analysis of a routine cognitive skill. *Cognitive Psychol. 12* (1980), 32–74.
5. Card, S.K., Moran, T.P., and Newell, A. *Applied Information Processing Psychology: The Human-Computer Interface.* Erlbaum, Hillsdale, N.J. (in preparation).
6. Chase, W.G., and Simon, H.A. Perception in chess. *Cognitive Psychol. 4* (1973), 55–81.
7. Deutsch, P.L., and Lampson, B.W. An online editor. *Comm. ACM 10*, 12 (Dec. 1967), 793–799.

8. Devoe, D.B. Alternatives to handprinting in the manual entry of data. *IEEE Trans. HFE-8* (1967), 21–32.
9. Embley, D.W., Lan, M.T., Leinbaugh, D.W., and Nagy, G. A procedure for predicting program editor performance from the user's point of view. *Internat. J. Man-Machine Studies 10* (1978), 639–650.
10. Maynard, H.B. *Industrial Engineering Handbook*. McGraw-Hill, New York, 3rd ed., 1971.
11. Moran, T.P. The command language grammar: A representation for the user interface of interactive computer systems. *Internat. J. Man-Machine Studies* (in press).
12. Myer, T. H., and Barnaby, J.R. Tenex executive language manual for users. Bolt, Beranek, and Newman, Inc., Cambridge, Mass., 1973.
13. Newman, W., and Sproull, R. *Principles of Interactive Computer Graphics*. McGraw-Hill, New York, 2nd ed., 1979.
14. Pew, R.W., Baron, S., Feehrer, C.E., and Miller, D.C. Critical review and analysis of performance models applicable to man-machine systems evaluation. Bolt, Beranek, and Newman, Inc., Cambridge, Mass., 1977.
15. Ramsey, H.R., and Atwood, M.E. Human factors in computer systems: A review of the literature. Science Applications, Inc., Englewood, Colorado, 1979.
16. Savitsky, S. Son of STOPGAP. Rep. SAILON 50.1, Stanford Artif. Intell. Lab., Stanford, Ca., 1969.
17. Simon, H.A. How big is a chunk? *Science 183* (1974), 482–488.
18. Wainer, H. Estimating coefficients in linear models: It don't make no nevermind. *Psychol. Bull. 83* (1976), 213–217.

Professional Activities Calendar of Events

ACM's calendar policy is to list open computer science meetings that are held on a not-for-profit basis. Not included in the calendar are educational seminars, institutes, and courses. Submittals should be substantiated with name of the sponsoring organization, fee schedule, and chairman's name and full address.

One telephone number contact for those interested in attending a meeting will be given when a number is specified for this purpose.

All requests for ACM sponsorship or cooperation should be addressed to Chairman, Conferences and Symposia Committee, Seymour J. Wolfson, 643 MacKenzie Hall, Wayne State University, Detroit, MI 48202, with a copy to Louis Fiora, Conference Coordinator, ACM Headquarters, 1133 Avenue of the Americas, New York, NY 10036; 212 265-6300. For European events, a copy of the request should also be sent to the European Representative. Technical Meeting Request Forms for this purpose can be obtained from ACM Headquarters or from the European Regional Representative. Lead time should include 2 months (3 months if for Europe) for processing of the request, plus the necessary months (minimum 3) for any publicity to appear in *Communications*.

■ This symbol indicates that the Conferences and Symposia Committee has given its approval for ACM sponsorship or cooperation.

In this issue the calendar is given to November 1981. New Listings are shown first; they will appear next month as Previous Listings.

NEW LISTINGS

19–22 July 1980
National Conference on Computer Related Crime, Washington, D.C. Sponsor: U.S. Dept. of Justice. Contact: Koba Associates, Inc., 2000 Florida Ave., NW, Washington, DC 20009; 202 328–5735.

3–5 September 1980
Conference and Workshop on Model Acceptance, Washington, D.C. Sponsors: SCS, GMD. Contact: Society for Computer Simulation, Box 2228, La Jolla, CA 92038; 714 459–3888.

6–7 October 1980
Fifth Conference on Local Computer Networks, Minneapolis, Minn. Sponsor: IEEE-CS. Contact: Abe Franck, USS: University of Minnesota, 227 Experimental Engr., 208 Union St. SE, Minneapolis, MN 55455.

9–10 October 1980
1980 Annual Meeting and Conference of the Museum Computer Network, New York State Museum, Albany. Sponsor: Museum Computer Network, Inc. Contact: David Vance, Center for Contemporary Arts and Letters, Library E-2340, State University of New York, Stony Brook, NY 11794; 516 246–6077.

15–18 October 1980
Symposium on Optimization Methods—Applied Aspects, Varna, Bulgaria. Sponsors: IFAC, IFORS. Contact: National Centre for Cybernetics & Computer Techniques, Committee for Science, 8 Slavyanska St., Sofia, Bulgaria.

20–22 October 1980
Texas Association for Educational Data Systems 1980 Annual Convention, Austin, Tex. Sponsor: TAEDS. Contact: Phil Gensler, Dept. of CIS, West Texas State University, Canyon, TX 79016.

4–6 November 1980
Federal Office Automation Conference, Washington, D.C. Sponsor: Office Automation Council. Contact: Federal Office Automation Conference, Box E, Wayland, MA 01778; 617 358–5119.

9–13 November 1980
Computer Related Crime: Training Workshop for Prevention, Detection, Investigation, and Prosecution, Charleston, S.C. Sponsor: U.S. Dept. of Justice. Contact: Koba Associates, Inc., 2000 Florida Ave., NW, Washington, DC 20009; 202 328–5780.

10–12 November 1980
ORSA/TIMS National Meeting, Houston, Tex. Sponsors: ORSA, TIMS. Contact: James McFarland, School of Business, University of Houston, Houston, TX 77004.

7–10 December 1980
1980 CAUSE National Conference, Phoenix, Ariz. Sponsor: CAUSE. Contact: CAUSE, 737 Twenty-Ninth Street, Boulder, CO 80303; 303 449–4430.

3–8 January 1981
Annual Meeting of AAAS, Toronto, Canada. Sponsor: American Association for the Advancement of Science. Contact: AAAS, 1515 Massachusetts Ave., NW, Washington, DC 20005

7–9 January 1981
Second Western Conference and Exhibition, Anaheim, Calif. Sponsor: Armed Forces Communications and Electronics Association. Contact: AFCEA, Skyline Center, Suite 300, 5205 Leesburg Pike, Falls Church, VA 22041.

12–14 January 1981
PTC 81, Honolulu, Hawaii. Sponsor: Pacific Telecommunications Council. Contact: Richard J. Barber, 2424 Maile Way, #704, Honolulu, Hawaii.

26–28 January 1981
■ **Eighth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages**, Williamsburg, Va. Sponsors: ACM SIGACT, SIGPLAN. Contact: Robert E. Noonan, Dept. of Mathematics and Computer Science, College of William & Mary, Williamsburg, VA 23185.

12–13 March 1981
Computer Science and Statistics: 13th Symposium on the Interface, Pittsburgh, Pa. Sponsor: Carnegie-Mellon University. Symp. chm: William F. Eddy, Dept. of Statistics, Carnegie-Mellon University, Pittsburgh, PA 15213.

6–10 April 1981
International Congress on Logic, Informatics, Law, Florence, Italy. Sponsor: Istituto per la documentazione giuridica of Consiglio Nazionale delle Richerche. Contact: Istituto per la documentazione giuridica, Via Panciatichi, 56/16–50127 Florence, Italy.

30 April–1 May 1981
Twelfth Annual Pittsburgh Conference on Modeling and Simulation, Pittsburgh, Pa. Sponsor: University of Pittsburgh in cooperation with Pittsburgh sections of IEEE, Systems, Man and Cybernetics Society, ISA, SCS, International Association for Mathematics and Computers in Simulation. Conf. co-chm: William G. Vogt and Marlin H. Mickle, 348 Benedum Engineering Hall, University of Pittsburgh, Pittsburgh, PA 15261.

5–8 May 1981
AEDS Annual Convention, Minneapolis, Minn. Sponsor: Association for Educational Data Systems. Contact: Shirley Easterwood, AEDS, 1201 Sixteenth St., NW, Washington, DC 20036.

10–12 June 1981
Seventh Conference of the Canadian Man-Computer Communications Society, Waterloo, Ontario (in conjunction with CIPS National Conference). Sponsor: Canadian Man-Computer Communications Society. Prog. chm: Marceli Wein, Computer Graphics Section, Division of EE, National Research Council, Ottawa, Ontario, Canada K1A 0R8.

3–7 August 1981
■ **ACM SIGGRAPH 81**, Dallas, Tex. Sponsor: ACM Special Interest Group on Computer Graphics. Conf. chm: Anthony Lucido, Industrial Engineering Dept.—CS Services Division, Texas A&M University, College Station, TX 77843; 713 845-5531.

24–28 August 1981
CO 81, Conference on Combinatorial Optimization, Stirling, Scotland. Sponsor: Stirling University. Contact: L. Wilson (CO 81), Dept. of Computing, Stirling University, Stirling, Scotland.

9–11 September 1981
Eurographics 81, Annual Conference of the Eurographics Society, Technische Hochschule, Darmstadt, FRG. Sponsor: German Computer Society. Prog. chm: J. Encarnacao, Eurographics 81, Technische Hochschule Darmstadt, FG Graphisch-Interaktive Systeme, Steubenplatz 12, D–6100 Darmstadt, Federal Republic of Germany.

14–16 September 1981
■ **ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems**, Las Vegas, Nev. Sponsor: ACM Special Interest Group on Measurement and Evaluation. Conference Chm: Herbert Schwetman, Dept. of Computer Science, Purdue University, West Lafayette, IN 47907; 317 494-8566.

PREVIOUS LISTINGS

21–23 July 1980
Sixth South African Symposium on Numerical Mathematics, Durban, Rep. of South Africa. Sponsor: University of Natal. Contact: Chairman, Computer Science Dept., University of Natal, King George V Ave., Durban 4001, Republic of South Africa.

6–8 August 1980
Conference on Human Aided Optimization, Wharton School, Philadelphia, Pa. Sponsors: Office of Naval Research, Wharton School. Conf. chm: Gerald Hurst, Decision Sciences, W-83 Dietrich Hall/CC, University of Pennsylvania, Philadelphia, PA 19104; 215 243–7730.

17–21 August 1980
1980 Urban and Regional Information Systems Association Conference, Toronto, Ont., Canada. Sponsor: URISA. Contact: URISA, 180 North Michigan Ave., Suite 800, Chicago, IL 60601.

18–22 August 1980
4th Symposium on Computational Statistics, Edinburgh University, Scotland. Sponsor: International Association for Statistical Computing. Contact: COMPSTAT 1980, c/o Director, Program Library Unit, Edinburgh University, 18 Buccleuch Place, Edinburgh EH8 9LN, Scotland.

19–21 August 1980
■ **National Artificial Intelligence Conference**, Palo Alto, Calif. Sponsor: American Association for Artificial Intelligence in cooperation with ACM SIGART. Conf. chm: J. Marty Tenenbaum, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025; 415 326–6200 x4167.

20–22 August 1980
8th SIMULA Users' Conference, California State University, Northridge, Calif. Sponsor: Asso-

(*Calendar continued on p. 414*)