A formal description of users' knowledge of how to operate a device and user complexity

PETER G. POLSON

University of Colorado, Boulder, Colorado

and

DAVID E. KIERAS

University of Arizona, Tucson, Arizona

This paper presents a production-system-based formalism for the description of a user's knowledge of how to operate a device. This formalism permits us to make quantitative predictions for processing loads, learning times, and execution times for operating procedures for a device. Results that suggest that there is a linear relationship between number of productions and the time taken to learn a new operating procedure are summarized.

The terms "user friendliness," user complexity, and their numerous synonyms appear frequently in the popular press and the technical literature on human-computer interaction. The claim that a particular system is user friendly makes nice advertising copy, but this term has not been adequately defined in terms of more basic psychological constructs (Dehning, Essig, & Maass, 1981). Kieras and Polson (in press) developed a formal theory of user complexity that is the complement of user friendliness. They assumed that user complexity is a function of the amount and structure of the new knowledge that a subject must acquire in order to learn to operate a new system or device, and they argued that an adequate theory of user complexity should incorporate a formal description of this knowledge. Furthermore, an investigator should be able to derive predictions of observed learning times and performance from the theory.

This paper summarizes the theory of user complexity developed by Kieras and Polson (in press) and presents new experimental results that support the theory's predictions of learning times. The paper begins with a summary of previous relevant work. The next section describes the various components of a user's knowledge of how to operate a device. Two components of a user's knowledge, formalized as a production system, are described in the next three sections. Finally, the results of a study that shows that learning time is a linear func-

tion of the number of productions in the theoretical description of a method are summarized.

A major goal of this research is to develop a rigorous quantitative analysis of user complexity. Achieving this goal requires abstract descriptions of both the user and the device. The device-description formalism is presented in Kieras and Polson (1983, in press), who assumed that users' knowledge includes job-situation knowledge, how-to-do-it knowledge, and how-it-works knowledge. This theoretical paradigm is illustrated in Figure 1. A user's job-situation knowledge is knowledge of the various tasks that the device can perform. How-to-do-it knowledge is knowledge of how to carry out tasks using the device. How-it-works knowledge is knowledge of a device's functions and structure. A user's job-situation and how-to-do-it knowledge is defined as the user's job-task representation.

Here we describe a formal representation of a user's job-task knowledge that is implemented as a production system that is a special-purpose programming language.

Figure 1. A diagram showing the various kinds of knowledge a subject has in order to successfully operate a device.

This paper is the product of a collaborative project conducted by the authors at the University of Arizona and the University of Colorado with support from the International Business Machines Corporation. The opinions and conclusions contained within this research are those of the authors and not necessarily those of IBM. Requests for reprints should be sent to Peter G. Polson, Department of Psychology, Campus Box 345, University of Colorado, Boulder, CO 80309.

This implementation enables an investigator to simulate the interaction of a user and a device, making it possible to validate the adequacy of a description of the user's job-task representation and to investigate user-complexity issues without having the actual device available. The use of the simulation technology also permits the derivation of quantitative performance predictions.

The basic elements of a user's job-task representation are assumed to be described by a model developed by Card, Moran, and Newell (1980, 1983), the GOMS model. This representation comprises the following components: goals, operations, methods, and selection rules.

Goals have two functions: (1) They represent a user's intention to carry out a given task, and (2) they define the decomposition of a given task into more elementary subtasks that can be accomplished employing the device. Goals and subgoals have a hierarchical and sequential structure that defines the interrelationships of the various components that make up a complete task, and these interrelationships are referred to as the goal structure.

Operations are mental representations of cognitive functions performed by the user during the execution of a task and of the manipulations executed by the device.

Methods are procedures for accomplishing goals and subgoals. Methods have a wide range of complexity, with the most complex consisting of a hierarchy of goals and subgoals with the operations necessary to accomplish each component of the method. The simplest methods can consist of only a single keystroke.

Selection rules assert goals and specify what method should be used to accomplish a goal or a subgoal under different conditions. Job-selection rules specify what aspects of the job situation determine the assertion of a goal to accomplish a specific task. Other kinds of selection rules specify a particular method that is most appropriate to accomplish a given goal in a specific context.

The Job-Task Representation

The two major components of a user's job-task representation within the Kieras and Polson (in press) framework can be described in terms of the GOMS model. The job-situation knowledge is described by only two elements of the GOMS model, goals and selection rules. How-to-do-it knowledge for each of the individual tasks is represented by all four components of the GOMS model: goals, operations, methods, and selection rules.

The job-situation representation consists of job goals and job-selection rules. Job goals are representations of intentions to perform tasks utilizing the device. There are also more abstract goals characterizing relationships between various tasks. Job goals describe the functionality of the device. The job-selection rules specify which task is to be performed by asserting the appropriate goal in circumstances under which the task should be carried

out. They enable a user to choose correctly an appropriate operating procedure, a subset of his or her how-to-do-it knowledge, in order to accomplish a specified job goal.

The second major component of the user's job-task representation is the how-to-do-it knowledge for each task; the how-to-do-it knowledge is made up of all four components of the GOMS model. To review, goals define the overall structure of a task and properly sequence the execution of each of a task's major methods. Methods are procedures for accomplishing the major subgoals that make up a task. Selection rules select appropriate methods as defined by a given context for each goal. Operations are the actions carried out by the device and the user.

The next two sections present a formal notation to describe the user's job-task knowledge. This notation is based on the production-system concept. We first describe the general characteristics of a production system and then the formal representation implemented in LISP. The production-system formalism permits very simple derivations of quantitative performance predictions.

Basic Principles of Production Systems

A production system is composed of a collection of production rules and a working memory. The working memory contains representations of the system's current goals, other information about the status of current and past actions, and descriptions of inputs from the environment. Production systems have been used to formalize theories of problem-solving processes (e.g., Karat, 1983; Newell & Simon, 1972; Simon, 1981), text comprehension (Kieras, 1982), learning (Anderson, 1982), and other cognitive processes (Anderson, 1983).

A production rule, or production for short, is a condition-action pair of the schematic form:

IF (condition) THEN (action)

The condition of a production is a series of tests ascertaining the state of the current contents of working memory, such as the presence or absence of specified goals or particular inputs from the environment. If the results of all tests are true, then the production is said to fire, and the action component of the production is executed. The action consists of one or more elementary actions that insert and delete information from working memory or perform operations on the external environment.

A set of production rules is a program that is executed by the system cycling through recognize and act modes of operation. During the recognize mode, the conditions of all productions are tested against the contents of working memory. The system then goes into the act mode, during which the action components of all productions that fired are executed. The system then returns to the recognize mode with the contents of its working memory having been modified, leading to the

opportunity for a different production to fire. Writing a program in production rules amounts to manipulating the contents of working memory to cause the pattern of production firings that will generate the desired sequence of overt actions on the part of the system. These are operator actions that cause the device to execute various methods.

Description of Notation

The notation presented here has been developed in the analysis of word processing tasks and the writing of descriptions of the how-to-do-it knowledge for editing functions on the IBM Displaywriter. Other types of tasks in the operations of different devices may require additional constructs to be incorporated into the formalism. The examples in Tables 1 and 2 should be studied carefully while the following descriptions are read. The examples are from a running simulation program written in LISP. Labels, identifiers, and function names may be of arbitrary lengths and can contain hyphens.

Each production begins with a label. It is not functional but is used for purposes of tracing and debugging. The condition of each production is prefaced by IF and is enclosed in parentheses. Following the IF is the logical operator AND, which specifies that the condition of the production is true only if each of the elementary conditions is true. The elementary condition is a test for the presence and absence of a specific piece of information in working memory, or in the environment. Each elementary condition, enclosed in parentheses, consists of an identifier that specifies the type of information involved in the test, followed by a specific pattern defining the information being tested for.

TEST-GOAL and TEST-NOTE test for patterns of strings of characters in working memory representing goals and notes, respectively. Goals are the system's representations of GOMS-model goals. Notes provide information about current or past actions. TEST-MSS and TEST-CURSOR interrogate the external environment, reading information from the manuscript and evaluating the position of the cursor on the screen, respectively. The simulation program incorporates representations of the screen, the manuscript, and information in the manuscript that specifies the editing actions to be carried out. Finally, there are conditions that test for the negation of a specified elementary condition.

Variables indicated by the prefix % can appear in a pattern defining an elementary condition. When the condition is evaluated, if the variable is bound to a string, that string substitutes for the variable in the pattern. If the variable has a null value, it will match any part of a pattern, and then it will be assigned a new value during the match process. For example, if there was a pattern "A B C" in working memory, and "A %X C" was an elementary condition pattern, and %X equaled null, then the pattern would match and %X would be bound to "B." Variables are unbound—set to null—by

an explicit action incorporated into the action portion of a production rule. Variables could also be assigned values from tests for information on the screen or in the manuscript and are most often used to deal with cursor-positioning operations. The symbol ??? in a pattern is a place holder; ??? will always match a corresponding substring, but without binding the value of the substring to a variable.

The action portion of a production rule is a sequence of elementary actions. The list of actions is prefaced by THEN and is enclosed in parentheses. Elementary actions can add and delete goals and notes, operate keys and controls on the device, and search for information on the screen or in the manuscript.

The actions ADD-GOAL, ADD-NOTE, DELETE-GOAL, and DELETE-NOTE add or delete the specified patterns, either goals or notes, from working memory. LOOK-MSS scans the manuscript for specified information. DO-KEYSTROKE causes the specified key to be pressed on the device. TYPE-IN causes a character string to be entered into the device. PRINT-MSG outputs the associated message and is used as a place holder for unimplemented functions in the simulator.

The formalism makes no attempt to represent more fundamental cognitive processes, such as those that would comprehend the manuscript or editing instructions, and the system has no knowledge of the semantics of its various goals, notes, and other operations that appear in elementary conditions or actions. This information is provided directly to the simulation in the form of internal representations of the manuscript in the screen. Current cognitive theory provides us with some answers to these issues. However, describing these basic cognitive processes would needlessly increase the complexity of the simulation without contributing anything to our understanding of the psychological processes involved in operating the device under study.

The relationship between the GOMS model and the production-system formalism is direct. Goals are directly represented in the production system for job-task knowledge. They appear in the conditions of almost all production rules, and are manipulated in a majority of production's actions. Methods are defined by collections of production rules that fire in a sequence whose first member is triggered by the assertion of the goal associated with that method. The structure of the subgoals incorporated into the method is defined by the sequence of ADD-GOAL and DELETE-GOAL actions that occur during the execution of the productions that define the method. Selection rules are productions that assert goals and thus control the execution of methods. A selection rule can also be a collection of productions that evaluates the current context in order to select the most appropriate method. Operations are scattered throughout the whole system and consist primarily of the different elementary actions, as well as of environment testing conditions, such as TEST-MSS.

Examples of How-to-Do-It Knowledge

In this section, the production-system formalism described in the preceding section is illustrated by an analysis of some of the components of a user's job-task knowledge for the IBM Displaywriter for both expert and novice users. These examples are taken from a running simulation of the Displaywriter's DELETE functions.

Production systems are arbitrary programming languages; thus, constraints have to be incorporated into the theory to make reasonable predictions. These constraints are expressed in the form of assumptions about the external determinants of a user's internal representation, the productions, and a set of style rules that constrain the possible programs for a method (Kieras & Bovair, 1983). There are different style rules for expert and novice programs for the same method. It is assumed that transition from novice to expert production systems involves the process of composition and other mechanisms described by Anderson (1982, 1983).

Kieras and Bovair (1983) developed novice-style rules that assume that the description of the procedure contained in the instructions, the contents of various displays that appear during the execution of the procedure, and the control actions determine the production system describing a procedure. Further, novice-style rules add additional constraints. It is assumed that novice goal structures are derived directly from the instructions and are highly redundant. Novices explicitly test all goals that are in working memory. Novices explicitly attend to and evaluate feedback information appearing on a display and explicitly verify the successful execution of each step in a procedure. Finally, novices execute only one control action in each production, although typing in a string of characters is perceived to be a single control action.

Expert productions incorporate efficient goal structures. They do not attempt to explicitly confirm the successful completion of each step and do not attempt to evaluate feedback provided on displays. Multiple operator actions that occur in a fixed sequence can be generated by a single production. There is a 3- to 5-to-1 difference in the number of productions in novice versus expert representations of methods.

The first example shown in Table 1 is the selection rule and the expert version of a method for deleting a single word from a manuscript. The second example shown in Table 2 is the novice version of the same method. These examples are based directly on the description of the DELETE command in *Displaywriter System Operator Reference Guide* (IBM, 1981). The examples do not include the higher level control structure of the overall editing task or the complex cursor-positioning rules and methods that these examples use.

Card, Moran, and Newell (1980, 1983) have assumed that the task of editing a manuscript is decomposed into a sequence of unit tasks. Each unit task is an independent activity and involves two stages. The first is acquiring

the unit task, and the second is performing the manuscript changes specified in the manuscript. Omitted from the examples are the productions that assert the goal of executing a next unit task, which causes the simulation to search the representation of the manuscript and to discover the next modification to be made to the manuscript and its location.

The productions for the expert version of a method for deleting a single word are given in Table 1. There are four separate productions. The first, labeled SELECT-WORD-DELETION, is the selection rule for the method. The second, WORD-DELETION-DONE, is the completion production that deletes the note from working memory asserting that the system is in the process of deleting a word and deletes the goal to perform unit task. The third production positions the cursor at the beginning of the word to be deleted if it is not already there. The action of that production, adding the goal to move the cursor, triggers the cursor-movement productions, which analyze the distance between the current position of the cursor and the beginning of the location of the word to be deleted and then cause the appropriate cursor-movement method to be used to perform the movement. The fourth production generates a sequence of three keystrokes that actually deletes the word from the manuscript and then deletes the goal of deleting the word.

Table 1
Expert Method for Deleting a Single Word
Terminated by a Space

```
(SELECT-WORD-DELETION
  IF (AND (TEST-GOAL PERFORM UNIT TASK)
    (TEST-MSS FUNCTION IS DELETE)
    (TEST-MSS ENTITY IS WORD)
    (NOT (TEST-GOAL DELETE WORD))
    (NOT (TEST-WM EXECUTING WORD DELETION))) )

  THEN ((ADD-GOAL DELETE WORD)
    (ADD-NOTE EXECUTING WORD DELETION)
    (LOOK-MSS TASK IS AT %UT-HP %UT-VP)) )

(WORD-DELETION-DONE
  IF (AND (TEST-GOAL PERFORM UNIT TASK)
    (TEST-WM EXECUTING WORD DELETION)
    (NOT (TEST-GOAL DELETE WORD))) )

  THEN ((DELETE-WM EXECUTING WORD DELETION)
    (DELETE-GOAL PERFORM UNIT TASK) ))

(PDELW1
  IF (AND (TEST-GOAL DELETE WORD)
    (NOT (TEST-GOAL MOVE CURSOR TO %UT-HP %UT-VP))
    (NOT (TEST-CURSOR %UT-HP %UT-VP))) )

  THEN ((ADD-GOAL MOVE CURSOR TO %UT-HP %UT-VP)) )

(PDELW2
  IF (AND (TEST-GOAL DELETE WORD)
    (TEST-CURSOR %UT-HP %UT-VP))

  THEN ((DO-KEYSTROKE DEL)
    (DO-KEYSTROKE )
    (DO-KEYSTROKE ENTER)
    (DELETE-GOAL DELETE WORD)
    (UNBIND %UT-HP %UT-VP) ))
```

The novice version of a method to delete a single word is presented in Table 2. Observe that the first two productions are identical to the expert version. They are the selection rule and the production that terminates

Table 2
Novice Version of Method to Delete a Single Word

```
(SELECT-WORD-DELETION
  IF (AND (TEST-GOAL PERFORM UNIT TASK)
    (TEST-MSS FUNCTION IS DELETE)
    (TEST-MSS ENTITY IS WORD)
    (NOT (TEST-GOAL DELETE WORD))
    (NOT (TEST-NOTE EXECUTING WORD DELETION))) )
  THEN ((ADD-GOAL DELETE WORD)
    (ADD-NOTE EXECUTING WORD DELETION)) )

(WORD-DELETION-DONE
  IF (AND (TEST-GOAL PERFORM UNIT TASK)
    (TEST-NOTE EXECUTING WORD DELETION)
    (NOT (TEST-GOAL DELETE WORD))) )
  THEN ((DELETE-NOTE EXECUTING WORD DELETION)
    (DELETE-GOAL PERFORM UNIT TASK) ))

(NDW-START
  IF (AND (TEST-GOAL DELETE WORD)
    (NOT (TEST-GOAL DO WORD DELETION STEP ???)))
  THEN ((ADD-GOAL DO WORD DELETION STEP 1)) )

(NDW1
  IF (TEST-GOAL DO WORD DELETION STEP 1)
  THEN ((LOOK-MSS TASK IS AT %UT-HP %UT-VP)
    (DELETE-GOAL DO WORD DELETION STEP 1)
    (ADD-GOAL DO WORD DELETION STEP 2)) )

(NDW2
  IF (AND (TEST-GOAL DO WORD DELETION STEP 2)
    (NOT (TEST-GOAL CHECK CURSOR POSITION)))
  THEN ((ADD-GOAL CHECK CURSOR POSITION)) )

(NDW2A
  IF (AND (TEST-GOAL DO WORD DELETION STEP 2)
    (TEST-GOAL CHECK CURSOR POSITION)
    (TEST-CURSOR %UT-HP %UT-VP))
  THEN ((DELETE-GOAL CHECK CURSOR POSITION)
    (DELETE-GOAL DO WORD DELETION STEP 2)
    (ADD-GOAL DO WORD DELETION STEP 3)) )

(NDW2B
  IF (AND (TEST-GOAL DO WORD DELETION STEP 2)
    (TEST-GOAL CHECK CURSOR POSITION)
    (NOT (TEST-CURSOR %UT-HP %UT-VP))
    (NOT (TEST-GOAL MOVE CURSOR TO ??? ????)))
  THEN ((ADD-GOAL MOVE CURSOR TO %UT-HP %UT-VP)) )

(NDW3
  IF (TEST-GOAL DO WORD DELETION STEP 3)
  THEN ((DO-KEYSTROKE DEL)
    (DELETE-GOAL DO WORD DELETION STEP 3)
    (ADD-GOAL DO WORD DELETION STEP 4)) )

(NDW4
  IF (AND (TEST-GOAL DO WORD DELETION STEP 4)
    (EQUAL *USER-MESSAGE 'DELETE-WHAT'))
  THEN ((DELETE-GOAL DO WORD DELETION STEP 4)
    (ADD-GOAL DO WORD DELETION STEP 5)) )

(NDW5
  IF (TEST-GOAL DO WORD DELETION STEP 5)
  THEN ((DO-KEYSTROKE --)
    (DELETE-GOAL DO WORD DELETION STEP 5)
    (ADD-GOAL DO WORD DELETION STEP 6)) )

(NDW6
  IF (TEST-GOAL DO WORD DELETION STEP 6)
  THEN ((PRINT-MSG '"CHECKING FOR HIGHLIGHTED CORRECT"')
    (DELETE-GOAL DO WORD DELETION STEP 6)
    (ADD-GOAL DO WORD DELETION STEP 7)) )
```

```
(NDW7
  IF (TEST-GOAL DO WORD DELETION STEP 7)
  THEN ((DO-KEYSTROKE ENTER)
    (DELETE-GOAL DO WORD DELETION STEP 7)
    (ADD-GOAL DO WORD DELETION STEP 8)) )

(NDW8
  IF (TEST-GOAL DO WORD DELETION STEP 8)
  THEN ((PRINT-MSG '"VERIFY THAT MATERIAL DELETED"')
    (DELETE-GOAL DO WORD DELETION STEP 8)
    (ADD-GOAL DO WORD DELETION STEP 9)) )

(NDW-STOP
  IF (TEST-GOAL DO WORD DELETION STEP 9)
  THEN ((DELETE-GOAL DO WORD DELETION STEP 9)
    (DELETE-GOAL DELETE WORD)
    (UNBIND %UT-HP %UT-VP)) )
```

execution of the procedure by deleting various control information from working memory. The remaining 11 productions generate the same sequence of key-strokes as the last two productions in Table 1, a 5-to-1 difference in length. The procedure is broken up into a series of explicit steps that are contained in the description of this procedure in the reference manual (IBM, 1981). In addition, novices explicitly check the consequences of each step and all feedback provided by the system. Finally, only one control action is executed in a single production.

Evaluation of the Theory

The evaluation process involves three stages. The first is to write production systems like those shown in Tables 1 and 2 for tasks that subjects will learn and execute in an experiment. The second is to run the simulation collecting various kinds of statistics that are the basis for predictions of observed performance. The third step is to demonstrate that there are lawful relationships between attributes of each production system, the statistics accumulated during the execution of the simulator, and performance measures such as time to learn a task and time to execute it correctly.

Quantitative predictions of various aspects of subjects' performance are derived from production systems for each task and the output of simulation runs of the execution of each of the tasks (Kieras & Bovair, 1983). The characteristics of the simulation programs and their execution-time behavior are assumed to be directly related to different aspects of subjects' performance. The following assumptions follow from the basic characteristics of the production-system architecture described earlier. Time to learn a procedure is assumed to be a function of the number of new productions in its description that must be learned by a user. The number of recognize-act cycles executed during the simulated performance of a procedure predicts the amount of time for a subject to correctly execute the procedure. Processing effort or working-memory load for a user is assumed to be a direct function of the number of items in the working memory of the simulation during various steps of the procedure. There are relationships between simu-

lated working-memory load and the probability of a subject's making an error. Learning and other performance parameters may also be affected by high working-memory loads during execution of an operating procedure.

EXPERIMENTAL EVIDENCE

The theory's predictions for time to learn operating procedures for a menu-driven word processor were evaluated in an experiment described below. Kieras and Bovair (1983) carried out more detailed tests using a simple control-panel device.

Method

A FORTRAN program was written that simulates an IBM Displaywriter, which is a stand-alone, floppy-disk-based word processor. The simulation also incorporates a simple computer-assisted instruction subsystem. A group of 43 subjects recruited from the Boulder community participated in an experiment in which they learned seven utility procedures for the Displaywriter. The procedures included printing a document, copying a diskette, deleting a document, setting various default format parameters, and checking spelling. The tasks were learned in the same order for all subjects.

Each task involved two components. First, subjects carried out a series of actions that loaded a simulated diskette containing documents into a simulated diskette drive. The simulation involved a series of graphic displays on a standard CRT terminal. The subjects then learned to perform one of the seven utility tasks. Each utility task involved traversal through a series of menus to a menu that specified the utility task to be performed. The task was performed by filling in a series of parameters on that menu and then exiting the menu, which caused the specified operations to actually be carried out.

Each task was learned by a serial anticipation method. On each step, a subject was presented with a high-fidelity emulation of the IBM Displaywriter menu for that step in a procedure. If a subject made the correct response, the program simulated the corresponding action of the Displaywriter. A subject received error messages generated by the actual Displaywriter if he or she made an error. The training procedure confined a subject to the correct path through the menu. If two consecutive errors were made, a feedback screen that precisely described the next step to be carried out appeared. After studying the feedback screen, a subject again had to attempt to anticipate the correct step. Training continued until a subject had completed three consecutive errorless trials. The data discussed here are the average times to criterion for each of the seven tasks.

Results and Discussion

Predicted learning times were derived from two sets of assumptions: the various assumptions and novice-style rules that describe how the production system for each of the seven methods is to be written, and assumptions about transfer from one procedure to another. The production systems for each method ranged from 28 to 49 productions in length. The first 12 productions for each task were identical; these productions simulated the loading of a diskette into the diskette unit. The remaining productions performed the actual utility task.

The number of productions in the method is not a good predictor of learning times, and should not be. A large amount of transfer is possible from one method to another. The actions in loading the diskette unit were

Figure 2. Observed and predicted mean learning times for each of seven experimental tasks.

identical for all seven tasks. Several of the tasks had common initial paths through the sequence of menus. Predictions were derived from a strict, common-elements theory of transfer (Kieras & Bovair, 1983). It was assumed that the number of new productions describing a task was the correct predictor of time to learn. It was also assumed that once learned, common productions would transfer to a later task even if one or more different tasks not containing these productions intervened. The observed and predicted learning times are shown in Figure 2. The total number of productions, the number of new productions, and the number of productions transferred from previously acquired tasks are shown in Table 3.

The predicted values for the mean learning times are calculated from the best-fitting linear function relating number of new productions given in Table 3 to learning time. This function has an intercept of 94.7 sec and a

Table 3
Numbers of Productions in Theoretical Descriptions of Seven Training Tasks

	Task						
	1	2	3	4	5	6	7
Total Productions	31	31	41	38	31	47	27
New Productions	31	19	23	17	11	27	9
Productions Transferred from Previously Learned Task	0	12	18	21	20	20	18

slope of 31.1 sec per new production. As can be seen from Figure 2, this equation does a relatively credible job of accounting for the mean learning times, predicting 61% of the variance in the original data points. Kieras and Bovair (1983) found very similar results for their control-panel device.

In summary, the results provide strong support for the learning-time predictions of the Kieras and Polson (in press) formalism. The next step in this research program will be to validate the predictions of the formalism for execution times and the effects of working-memory load on learning and performance. The most important general implication is that these results support the contention that computer-simulation techniques combined with a well-integrated empirical program can lead to a rigorous and valid definition of user complexity. Such results could be the basis for a rational design technology for computer-based systems. The designer would attempt to develop systems that were richly functional and at the same time structured in such a fashion as to minimize the amount and complexity of the knowledge necessary to operate a system.

REFERENCES

- ANDERSON, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, **89**, 369-406.
- ANDERSON, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.
- CARD, S. K., MORAN, T. P., & NEWELL, A. (1980). 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. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum.
- DEHNING, W., ESSIG, H., & MAASS, S. (1981). The adaptation of virtual man-computer interfaces to user requirements in dialogues. In G. Goos & J. Hartmanis (Eds.), *Lecture notes in computer science* (Vol. 110). Berlin: Springer-Verlag.
- IBM. (1981). *Displaywriter system operator reference guide, Text-pack 1* (pp. 3-36-3-40). Austin, TX: Author.
- KARAT, J. (1983). A model of problem solving with incomplete constraint knowledge. *Cognitive Psychology*, **14**, 538-559.
- KIERAS, D. E. (1982). A model of reader strategy for abstracting main ideas from simple technical prose. *Text*, **2**, 47-82.
- KIERAS, D. E., & BOVAIR, S. (1983). *The acquisition of procedures from text* (Tech. Rep. in preparation). Tucson: University of Arizona, Department of Psychology.
- KIERAS, D. E., & POLSON, P. G. (1983). A generalized transition network representation of interactive systems. In A. Janda (Ed.), *Proceedings of the CHI 1983 Conference on Human Factors in Computing* (pp. 103-106). Boston: New York: The Association for Computing Machinery.
- KIERAS, D. E., & POLSON, P. G. (in press). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*.
- NEWELL, A., & SIMON, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.
- SIMON, H. A. (1981). *The sciences of the artificial*. Cambridge, MA: MIT Press.