The GOMS Family of Analysis Techniques: Tools for Design and Evaluation

Bonnie E. John & David E. Kieras*

24 August 1994
CMU-CS-94-181

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Department of Electrical Engineering and Computer Science
University of Michigan

Also appears as Human-Computer Interaction Institute Technical Report
CMU-HCII-94-106

Work on this paper by Bonnie John was supported by the Office of Naval Research, Cognitive Science Program, Contract Number N00014-89-J-1975N158, and the Advanced Research Projects Agency, DoD, and monitored by the Office of Naval Research under contract N00014-93-1-0934. Work on this paper by David Kieras was supported by the Office of Naval Research Cognitive Science Program, under Grant Number N00014-92-J-1173 NR 4422574, and the Advanced Research Projects Agency, DoD, and monitored by the NCCOSC under contract N66001-94-C-6036. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, NCCOSC, the Advanced Research Projects Agency, or the U. S. Government.

Bonnie John is associated with the Departments of Computer Science and Psychology and the Human-Computer Interaction Institute at Carnegie Mellon University.

Keywords: GOMS, cognitive modeling, usability engineering

Abstract

Since the seminal Card, Moran, & Newell (1983) book, *The psychology of human-computer interaction*, the concept of the GOMS model has been one of the few widely known theoretical concepts in human-computer interaction. This concept has spawned much research to verify and extend the original concept and has been used in real-world design and evaluation situations. The original presentation of the GOMS concept left substantial room for interpretation and subsequent researchers and practitioners have applied the idea in a variety of ways. While this variety shows that the GOMS concept is fruitful, there is some confusion about the GOMS concept and the various approaches that share this label yet appear to be radically different. This paper synthesizes the previous work on GOMS to provide an integrated view of GOMS models and how they can be used in design. The major variants of GOMS that have matured sufficiently to be used in real-world design and evaluation situations are described and related to the original GOMS proposal and to each other. A single example is used to illustrate all of the techniques. Guidance is provided to practitioners who wish to use GOMS for their design and evaluation problems, and examples of actual applications of GOMS techniques are presented.

1. Introduction

1.1. Purpose of this Paper

Since the seminal Card, Moran, & Newell (1983) book, *The Psychology of Human-Computer Interaction*, (hereafter, CMN) the concept of the GOMS Model has been one of the few widely known theoretical concepts in human-computer interaction (HCI). This concept has spawned much research to verify and extend the original concept. In 1990, Olson and Olson reviewed the state of the art of cognitive modeling in the GOMS tradition, discussing several extensions to the basic framework in the research stage of development and pointing the way to several more plausible and useful extensions that could be explored. They also outlined several significant gaps in cognitive theory that prevent cognitive modeling in general from addressing some important aspects of HCI (e.g., fatigue) and argued that cognitive models are essentially the wrong granularity or form to address certain other aspects of computer systems design, such as user acceptance and fit to organizational life (but see Carley & Prietula, 1994, for an opposing view).

This paper, coming several years later in the history of HCI cognitive modeling, has different goals from the Olson & Olson paper. The major goal is to clarify the relationships between the different variants of GOMS models that have appeared since the CMN work. There appears to be confusion about the relationship of these seemingly different ideas to the original concept and to each other, and also about which GOMS variant is suitable for which design and evaluation tasks. Thus, the first goal of this paper is to provide a synthesis of the GOMS models in the literature to clear up these points of confusion,

A second goal of this paper is to provide guidance to practitioners wishing to select a GOMS variant for their design or evaluation task, and to briefly demonstrate the value of GOMS techniques in real-world design and evaluation tasks. Thus, this paper focuses on that subset of GOMS research that has reached sufficient maturity to be tools in the engineer's toolbox of design. The techniques we survey are all variants of the original GOMS concept that have been sufficiently tested and codified to leave the research laboratory, to be taught to practitioners, and to be used in real-world design and evaluation situations. We present some examples of applying GOMS techniques to design situations, ranging from small design issues to whole systems, and with payoffs ranging from informally positive to actual dollar figures. We hope these examples will help HCI practitioners to see value in GOMS for their own situations.

The organization of this paper is as follows. First, in the remainder of this introduction, we discuss the concept of engineering models for computer system design; GOMS models are a kind of engineering model. The second major section of the paper introduces the basic concept of GOMS models that is common to all of the techniques we discuss. The third section defines the current members of this GOMS family of models and discusses their similarities and differences using a single example task. The fourth section describes how the GOMS family can be applied in design. The fifth section provides some brief case histories of how members of the GOMS family have been applied in actual software development situations.

1.2. Engineering Models of Human-Computer Interaction

Before beginning the discussion of the GOMS concept and particular instantiations of that concept, it is useful to understand the motivation for developing such techniques. The overall philosophy behind GOMS and other HCI cognitive modeling efforts is to provide *engineering models* of human performance. Engineering models seek to optimize several criteria that distinguish them from traditional, psychologically-oriented cognitive models: (1) the ability to make *a priori* predictions; (2) the ability to be learned and used by practitioners as well as researchers; (3) coverage of relevant tasks; and (4) approximation. Card, Moran, & Newell developed the concept of GOMS models with these criteria for useful engineering models in mind. GOMS models are usefully approximate, make *a priori* predictions, cover a range of behavior involved in many HCI tasks, and have been proven to be learnable and usable for computer system designers.

A priori prediction. In a research environment, cognitive models are often evaluated by estimating their goodness of fit to empirical data, using some aspect of the data to estimate some of the parameters in the model. Appropriate statistical techniques can be applied to determine whether the model essentially gives back more information about the data than it received in the form of parameter estimates. Such parameter estimation is clearly required to develop and test scientific models. However, to be useful, engineering models must be able to make predictions in the absence of a working version, prototype, or mock-up of the system because the predictions are needed early in the design process where they can be used to shape the specifications of the system. Thus, any parameters used to make the predictions must be set during the construction of the model, not on the basis of behavioral data collected on the new system. Since a system is not yet in existence, the model cannot require new experiments to be run in the new situation to set the parameters. This does not mean that parameters have to be fixed constants for every situation, only that they must be determinable *a priori*. For example, tables of parameters covering a wide range of tasks can be created based on previous research.

Learnability and usability. It must be possible for engineering models to be learned and used by computer system designers. Since the target users of HCI engineering models are not trained psychologists or human factors experts, they cannot be expected to bring psychological expertise to the task. Thus, the basic psychology must be "built into" the models. An example is that tables of parameter values can replace detailed expertise in many situations. In addition, the procedures for constructing and applying the model must be clearly defined and representative examples must be presented to allow the techniques to be taught and learned by the intended model users. This does not mean that the procedures have to be as fixed and explicit as recipes in a cook book. However, there must be guidelines and rules about what to do in many representative situations so that a style of analysis can be developed that leads to useful predictions.

Coverage. Engineering models must address a useful range of design issues. The problem in HCI is that the range of activities performed by people interacting with computer systems is quite varied, ranging from simple perceptual-motor actions such as pointing with a mouse, to extremely complex activities such as comprehending textual or pictorial material, all the way to creative problem-solving. Covering this entire span is not possible today with engineering models, and some issues may never be addressable with engineering models (e.g. predicting creativity). However, there are three general issues for which effective coverage is possible today and extremely important. First, the lower-level perceptual-motor issues, such as the effects of layout on key stroking or mouse pointing, can be well captured by existing models, or extensions of them. Second, the complexity and efficiency of the interface procedures is addressed very well by models in the GOMS family. Since at some point the user must always acquire and execute a procedure in order to perform useful work with a computer system, it is especially valuable that engineering models can address the procedural aspects of a user interface. Third, it is essential that the whole task be considered when designing a system. That is, optimizing the various aspects or portions of an interface in isolation will not ensure that the system is more usable overall or that it will allow the whole task to be performed more effectively. GOMS models are especially useful for such analyses because one way to characterize a whole task is to describe the procedures entailed by the whole task; this allows the individual aspects of the interface to be considered in the entire task context.

Approximation. Engineering models in all disciplines of engineering are deliberately approximate. They include just the level of detail necessary to do the design job. For example, when sizing a duct for an air-conditioning unit, fine variations in volume with respect to temperature are ignored if the air can be assumed to be in a certain range of temperature. If this assumption holds, then the equations are very simple, relating the area of the duct to the rate of air passing through it. However, when the same air is traveling through the pipes in the cooling chamber of the air-conditioner, where there is a substantial temperature change from hot to cool, this assumption no longer holds, and many other factors have to be incorporated, such as the relative humidity and compressibility. In the same way, engineering models for HCI must be approximate in nature, attending to only those details necessary to analyze the design problem while keeping the modeling effort tractable.

When engineers use approximate models, they do not do so blindly, but in the context of knowledge of a more detailed theory. They know what terms exist in the detailed theory, how sensitive the predictions are to variations in those terms, and, therefore, which terms can be dropped to make calculations tractable without sacrificing the accuracy necessary for the design situation. This can be done because the theory specifies the *mechanism* of the phenomenon it describes. Similarly, engineering models in HCI must be approximations to the processes involved in human behavior, not simply approximations to the behavior. In this way, the theoretical foundations of the models allow the designer to choose the right model for the required level of detail in the design problem, and to recognize when the design problem involves issues and factors not addressed by the models.

A common engineering guideline is the "80/20 rule", which states that you get 80% of your results from 20% of your design effort, and the remaining 20% takes 80% of the effort. Engineering models in HCI should strive for that first 80% of coverage with 20% of the effort. For instance, if a computer system were designed and a prototype built, then the behavior of scores of users could be observed and analyzed in detail and the "truth" would be known about the system. Taking this as 100% knowledge and effort, engineering models in HCI should strive to for 80% accuracy with less than 20% of the effort of prototyping and testing. That is, engineering models should be able to predict various behavioral aspects, e.g., the sequence of operations, the execution time, the learning time, and the occurrence of errors, to within 80% of what would be observed if the running system could be measured. Thus, less than perfect predictions are acceptable, but engineering models must be simplified approximations of the real situation in order to obtain such predictions with a small amount of effort.

Although engineering models need not exceed the level of approximation necessary for design purposes, as long as the other criteria are met, such models may be as detailed as their traditional psychology counterparts. In fact, quite detailed and exact models could be incorporated into an easily used computer simulation of the user, or simple models could account for as much performance phenomena as more complex models. Either way, useful engineering models emphasize *a priori* quantitative predictions, usability, and coverage, usually, but not always, at the expense of accuracy. The GOMS-family of engineering models satisfy these criteria for many important HCI tasks.

2. Definition of GOMS Models

The starting point for our discussion of GOMS is to define the concept of a general GOMS model (subsequently referred to as the GOMS concept or just the GOMS model). This concept is much weaker than any other proposal, even the original CMN proposal, but it serves to capture what all GOMS models have in common. The general GOMS concept is defined as follows:

It is useful to analyze knowledge of how to do a task in terms of the components of goals, operators, methods, and selection rules.

While similar to many other task decomposition strategies (e.g., Diaper, 1989; Gilbreth & Gilbreth, 1917; Newell & Simon, 1972; Van Cott & Kinkade, 1972), this concept has spawned a family of task analysis and modeling techniques, the *GOMS family*. In this section of this paper, we define each of the components of the model (goals, operators, methods, and selection rules) in more detail, and then in the next section, we describe the different kinds of models currently in the GOMS family and their relation to underlying human information-processing architectures. This discussion is limited to approaches that have been presented in the literature in an explicitly "how-to" form, ready to use by practitioners.

2.1. Goals

Goals are what the user has to accomplish. The common-sense meaning of the term applies here; a goal is the "end towards which effort is directed" (Webster's, 1977, p. 493). The classic example presented in CMN is in the domain of text-editing, where the user is presented with a hard-copy manuscript marked-up with editing changes, and the user's goal is to make all those changes in an on-line copy of that manuscript. Goals are often broken down into sub-goals; all of the subgoals must be accomplished in order to achieve the overall goal. For instance, in a manuscript marked with the four editing changes shown in Figure 1, the top-level goal would be *EDIT-MANUSCRIPT* and the sub-goals might be *MOVE-TEXT*, *DELETE-PHRASE* and *INSERT-WORD*. All of the subgoals must be accomplished to accomplish the higher-level goal.

Goals and sub-goals are often arranged hierarchically, but a strict hierarchical goal structure is not required. In particular, some versions of GOMS models allow several goals to be active at once, and some versions encode extremely well-practiced behavior in a "flattened" structure that does not seem to require a hierarchy of subgoals.

2.2. Operators

An operator is an action performed in service of a goal. Operators can be perceptual, cognitive, or motor acts, or a composite of these. Operators can change the user's internal mental state and/or physically change the state of the external environment. The important parameters of operators, in particular execution time, are assumed to be independent of how the user or the system got into the current state (i.e., independent of the history of operators). Execution time may be approximated by a constant, by a probability distribution, or by a function of some parameter. For instance, the time to type a word might be approximated by a constant (e.g., the average time for an average word by an average typist), or a statistical distribution, or by a function involving the number of letters in the word and the time to type a single character (which could, in turn be approximated by a constant or a distribution). The accuracy of

In order to understand GOMS models that have arisen in the last decade and the relationships between them, an analyst must understand ~~each of~~ the components of the model (goals, operators, methods, and selection rules), the concept of level of detail, and the different computational forms that GOMS models take. In this section, we will define each of these concepts; in subsequent sections we will categorize existing GOMS models according to these concepts.

Figure 1. A set of simple text-editing tasks illustrating several different text editing goals.

predictions obtained from a GOMS model depends on the accuracy of this assumption and on the accuracy of the duration estimates.

2.4. Methods

Methods are sequences of operators and subgoal invocations that accomplish a goal. If the goals have a hierarchical form, then there is a corresponding hierarchy of methods. Clearly the content of the methods depends on the set of possible operators and on the nature of the tasks represented.

For instance, in our text-editing example, if *DELETE-PHRASE* was defined as a goal, and MOVE-MOUSE, CLICK-MOUSE-BUTTON, SHIFT-CLICK-MOUSE-BUTTON and HIT-DELETE-KEY were defined as operators, one method for accomplishing *DELETE-PHRASE* (in the text-editor we are using to write this paper) would be to MOVE-MOUSE to the beginning of the phrase, CLICK-MOUSE-BUTTON, MOVE-MOUSE to the end of the phrase, SHIFT-CLICK-MOUSE-BUTTON, and finally, HIT-DELETE-KEY (the *mark-and-delete* method).

2.5. Selection Rules.

There is often more than one method to accomplish a goal. Instead of the above *mark-and-delete* method just described, another method for accomplishing the *DELETE-PHRASE* goal in Figure 1 would be MOVE-MOUSE to the end of the phrase, CLICK-MOUSE-BUTTON, and HIT-DELETE-KEY 11 times (the *delete-characters* method). If there is more than one method applicable to a goal, then selection rules are necessary to represent the user's knowledge of which method should be applied. Typically such rules are based on specific properties of the task instance. Selection rules can arise through a user's personal experience with the interface or from explicit training. Continuing our text-editing example, a user may have a rule for the delete-phrase goal that says if the phrase is more than 8 characters long, then use the *mark-and-delete* method, otherwise use the *delete-characters* method.

2.3. Level of Detail

It is important to clarify a common point of confusion about goals and operators. The distinction is strictly one of the required level of detail:

The difference between a goal and an operator in a GOMS analysis is merely a matter of the level of detail chosen by the analyst; for a goal, the analyst provides a method that uses lower-level operators to specify the details of how it is to be accomplished; in contrast, operators are not broken down any further.

That is, an analyst will decide that certain user activities do not need to be "unpacked" into any more detail, and thus will represent them as operators, while other activities do need to be considered in more detail, and so will represent these in terms of goals with their associated methods. Thus, any particular GOMS analysis assumes a certain grain of analysis, a "stopping point" in the level of detail, chosen to suit the needs of the analysis. Continuing the text-editing example given above, a GOMS analysis could have only one goal (*EDIT-MANUSCRIPT*) and a few high-level operators (e.g., MOVE-TEXT, DELETE-PHRASE and INSERT-WORD). Or, if the design situation required a finer level of detail, the analysis could have four goals (*EDIT-MANUSCRIPT*, with *MOVE-TEXT*, *DELETE-PHRASE* and *INSERT-WORD* as subgoals) and finer-grained operators like MOVE-CURSOR, CLICK-MOUSE-BUTTON, DOUBLE-CLICK-MOUSE-BUTTON, SHIFT-CLICK-MOUSE-BUTTON and HIT-DELETE-KEY to accomplish these goals.

In principle, the goals and operators of a task could be described at ever-deeper levels of detail, down to muscle group twitches. However, all GOMS methodologies stop at a much higher level of detail, which is deemed adequate for the analysis problem at hand. The lowest-level operators in an analysis are termed *primitive operators* in this paper. As discussed above, at any stopping point, the analyst must be sure that it is reasonable to assume that important properties of the operators, in particular execution time, are constant (or are a constant function of some given parameter) regardless of the surrounding context. These properties can then be estimated from data, either from the current task being analyzed or from previous similar tasks, and used to predict performance on new tasks.

CMN demonstrated nine models at four levels of analysis, with the highest-level being the unit-task level, in which the operators represented a whole task unit lasting about 30 sec, and the lowest level being at the keystroke-level, in which the operators are at the level of single keystrokes, mouse moves, and so forth, with a duration of about a second or less. At a lower level, Rosenbloom (Laird, Rosenbloom & Newell,1986) proposed GOMS models in which a basic operator corresponded to the cognitive cycle time of the Model Human Processor (about 50 ms).

An analyst must make the decision about which level of detail to use. That decision hinges primarily on the demands of the design or evaluation task the model is meant to accomplish, but it also depends on the availability of data for operator time estimates. Some design and evaluation situations give the analyst opportunity to directly measure operators at a given level on existing systems or prototypes, other situations may preclude such measurement and force the analyst to choose a grain of analysis for which there are known operator estimates, such as the keystroke level. In addition, although CMN's worked examples show a uniform level of detail within the each analysis, it is not essential that all primitive operators be at the same level. Many design situations will call for some procedures to be examined in more detail than others. For instance, when designing a hypermedia system, the analyst may be especially interested in the procedures for navigating through the system, but less interested in the understanding the details of how users will search and comprehend text, graphics, and movies to find specific information. Therefore, the analyst may chose to do a keystroke-level analysis of the navigation machanisms and define higher-level primitive operators for comprehension (e.g., FIND-INFORMATION-ON-SCREEN). If different levels of detail are used, the analyst should be aware that some aspects of GOMS techniques may not apply uniformly within such an analysis. For example, predicting execution times requires that the operators have known durations, and some of the analyst-defined higher-level operators might require empirical measurement before the execution time can be predicted for the methods that use them. Likewise, as will be discussed more later, predictions of method learning time will generally not be possible if the analyst-defined high-level operators require learning by the user.

2.6. Form of a Model

The different GOMS models in the literature differ substantially, and perhaps confusingly, in the basic form and appearance of the models. GOMS models have taken two basic forms, the *program form* and the *sequence form.*

A GOMS model in program form is analogous to a parameterized computer program. The model takes any admissible set of task parameters and will execute the corresponding instance of the described task correctly. For example, the *mark-and-delete* method described above, in program form, would take as task parameters the starting and ending locations of the to-be-deleted phrase, and when executed, would cause the mouse to be moved to the corresponding locations. Thus, a GOMS model in program form describes how to accomplish a general class of tasks, with a specific instance of the class being represented by a set of values for the task parameters. Typically, such a model will explicitly contain some form of conditional branching and invocations of submethods to accomplish subgoals. The procedural knowledge represented in a program form model is fixed, but the execution pathway through the task, that is, the sequence of operators executed, will depend on the specific properties of the task instance. Thus, once the model is defined, all of the possible tasks can be covered by different execution pathways through the model. A program form model is a compact, generative description that explicitly represents the knowledge of what features of the task environment the user should attend to and how the user should operate the system to accomplish the task goals.

GOMS models in the program form can be either hand-executable or machine executable. Machine-executable models have been expressed in terms of production systems (if-then rules). The program form has the advantage that each piece of knowledge is visible to the analyst inspecting the model. However, they usually have two disadvantages. First, the only way to determine the sequence of operators used in a task is to run the program (either by hand or machine) and obtain a trace of the program's execution. Second, the machine-executable models have the disadvantage that expressing knowledge in sufficient detail to produce a runnable computer program has been historically quite time-consuming.

In contrast, the sequence form of GOMS model displays a fixed sequence of operators for accomplishing a single goal in a particular task scenario. There may be some conditionality and parameters included in the sequence model. For instance, in the text-editing example above, listing the exact operators necessary to delete the phrase indicated in Figure 1 is a GOMS model in sequence form (e.g., MOVE-MOUSE, CLICK-MOUSE-BUTTON, 11*HIT-DELETE-KEY). A more general sequence model would take the number of characters in the phrase as a parameter and contain an implicit iteration. For example, for the *delete-characters* method, there would be a MOVE-MOUSE operator, a CLICK-MOUSE-BUTTON operator, and then the HIT-DELETE-KEY operator would be repeated until there were no more characters in the phrase. The advantages and disadvantages of the sequence form are the inverse of the program form. That is, the analyst does not have the difficult job of explicitly defining the knowledge necessary for every possible task situation in program-like detail, and the sequence of operators is clearly visible to the analyst. But there may be more knowledge available about the task methods that is not explicitly written out and therefore not inspectable. Also it is time-consuming to write sequence-form models by hand for a large set of tasks.

We will discuss how an analyst might choose between program and sequence forms, depending on design or evaluation needs, in Section 4.

3. The Current GOMS Family

The concepts associated with GOMS are a mixture of several types: task analysis techniques based on different assumptions, models of human performance on specific tasks, computational models of human cognitive architecture, and loosely-defined concepts about human cognition and information processing. Figure 2 displays the relationships between these ideas. The figure is a lattice; at the top is the idea of task analysis, and at the bottom is the basic conceptual framework for human information processing, namely that of the stage model. Thus the GOMS family consists of ideas for *analyzing and representing tasks in a way that is related to the stage model of human information processing*. Perhaps this is the distinctive feature of the GOMS approach compared to the many other concepts of task analysis in the human factors and system design literature.

Reading down from the top, the top layer consists of task analysis techniques, followed by explicit computational cognitive architectures, and at the bottom, conceptual frameworks which are informal statements about how humans can be modeled. As one reads down from the top, or up from the bottom, the ideas get more explicit and detailed; the middle contains approaches whose instantiations are running computer simulation models.

Three things should be noted about the diagram: (1) Since our primary purpose is to discuss GOMS models that are currently described in "ready-to-use" form in the literature, Figure 2 emphasizes these current techniques and the concepts directly related to them. (2) There are areas in the diagram, indicated with italics, which lack ready-to-use models or techniques, or are not directly related to GOMS models. The final portion of this section discusses some current research related to these issues. Thus, the diagram is certainly not exhaustive: many more nodes and arrows could be detailed; what is shown here is only what is central to our discussion of currently documented GOMS models. (3) The diagram shows only generic ideas and approaches, not specific instances of task modeling. Examples of specific instances of using these techniques will be summarized in Section 5.

We will describe the entries in this lattice, starting with the conceptual frameworks, since they form the basis for all GOMS analyses, working through the computational cognitive architectures, up to the task-analytic approaches which are the heart of the GOMS family.

3.1. Conceptual Frameworks

The conceptual frameworks are *conceptual* in that they are informally stated assumptions about the structure of human cognition. The conceptual frameworks shown are all based on a general assertion that human cognition and behavior is usefully analyzed in terms of stages, such as the conventional notion that stimuli are first processed perceptually, the resulting information is passed to a central cognitive process, which manipulates that information and eventually initiates some motor activity.

The stage idea immediately breaks out into two more specific forms. One is that the stages are performed serially, which certainly has always seemed reasonable for many laboratory tasks. The other that the stages can be performed in parallel to some extent, since the different kinds of processing are handled by separate mechanisms, or *processors*.

The Card, Moran, & Newell (1983) Model Human Processor (MHP) is a parallel architecture. Perceptual, cognitive, and motor processing are done by separate processing mechanisms, each with their own distinctive types and timing of activities, and with associated principles of operation. CMN's important insight was that the empirical human cognition and performance literature could be used to motivate and justify an *engineering model* of human information-processing that could be used to predict performance in HCI situations.

Although the MHP is inherently parallel, only 1 of the 19 examples of reasoning from the MHP presented in CMN (1983, Example 7) depend on this fact. The parallel operation of the MHP was made clear in John's model of transcription typing (John, 1988; John & Newell, 1989)

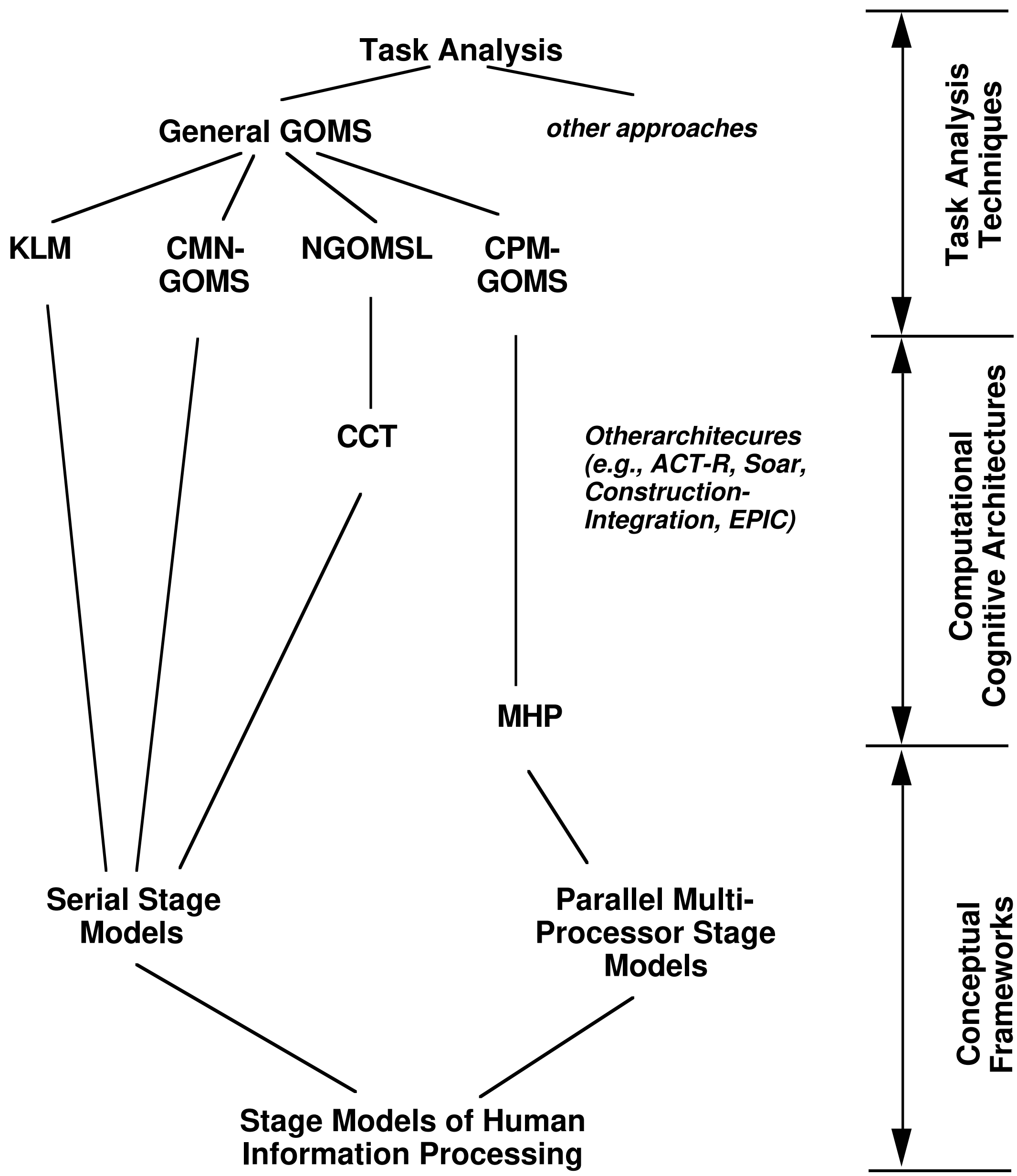


Figure 2. The GOMS family consists of task analysis techniques that are related to models of human information processing. Current research involves additional computational cognitive architectures, but only CCT is shown as a "ready-to-use" technique. See text for more discussion.

in which the processors could operate in a sort of "pipeline" mode, with information moving through perceptual, cognitive, and motor stages continuously. This model accounted for important properties of skilled typing performance and shows that parallelism can greatly influence the structure and performance of a task.

While CMN provide many examples of how the MHP can be applied to predict performance in some well-understood simple task situations similar to the experimental paradigms in the human performance literature, and simple real-world analogs of these tasks, they did not provide an explicit method for applying the MHP to complex, realistic tasks (the CPM-GOMS methodology to be discussed later provides this explication). In Figure 2, the MHP is shown at the border between conceptual architectures and computational cognitive architectures because while it is certainly more specified than the simple stage concepts, it is not as fully explicit and computationally represented as the ideas in the next level up in the diagram.

3.2. Computational Cognitive Architectures

The level of computational cognitive architectures in Figure 2 are proposals for how to represent human information processing in terms explicit enough to run as a computer program. This representation as a computer simulation is a preferred research tactic in cognitive psychology based on the assumption that a computational model has "empirical content" — that is, a properly constructed and interpreted model can make predictions of human performance, and these predictions can be empirically confirmed. There are several such architectures under development in cognitive psychology and artificial intelligence, several of which have been applied to topics in HCI, such as Cognitive Complexity Theory (CCT, Kieras & Polson, 1985; Bovair, Kieras, & Polson, 1988, 1990), ACT-R and its predecessors (Anderson, 1976, 1983, 1993), Construction-Integration (Kintsch, 1988, 1992), Soar (Newell, 1990), and EPIC (Kieras & Meyer, 1994; Meyer & Kieras, 1994). Each of these architectures makes different assumptions about how cognitive processes such as working memory management, flow of control, learning, and problem-solving are handled by the architecture, and testing the empirical content of these assumptions is an active area of psychological research. In principle, all of these architectures could be used to implement a particular GOMS task analysis in a computational model (e.g., John & Vera 1992; Peck & John, 1992; Gray & Sabnani, 1994). However, only CCT, a production-rule architecture based on the serial stage model, has been used as the basis for a specific GOMS technique, NGOMSL, which incorporates CCT's assumptions about working memory management, flow of control, and other architectural mechanisms. For brevity, CCT will not be discussed further, since its contributions are well represented by the NGOMSL technique that will be discussed in the next section.

3.3. Task Analysis Techniques

At the top of the GOMS family tree in Figure 2, under the overall node of *Task Analysis*, the node labeled *General GOMS* represents the concept stated earlier that it is useful to analyze knowledge of how to do a task in terms of goals, operators, methods, and selection rules. Thus, it is a form of task analysis that describes the procedural, "how-to-do-it" knowledge involved in a task. The result of a GOMS-based task analysis will be some form of description of the components of GOMS, the goals, operators, methods, and selection rules. As described above, GOMS admits a variety of definitions and representations of these components.

There are three critical restrictions on the kinds of task knowledge that GOMS models can be used for. The first is that the task in question must be usefully analyzed in terms of the "how to do it," or *procedural* knowledge required rather than other aspects of knowledge about the system, like mental simulations of an internalized device model, or analogical reasoning (see Kieras and Polson, 1985, for more discussion). The italicized area to the right under Task Analysis represents other existing and potential approaches to task analysis that capture other forms of task knowledge. For example, current work on electronics troubleshooting (see Gott, 1988) incorporates the person's knowledge of electronic components and the structure and function of the system under investigation, in addition to various kinds of procedural knowledge, and current work in analogical reasoning has been applied to understanding consistency in operating systems (Rieman, Lewis, Young & Polson, 1994).

The second restriction is that the GOMS family can only represent *routine cognitive skills*, which consist of procedural knowledge that originally was derived from problem-solving activity, but with practice has taken the form of a routinely invocable sequence of activities that accomplishes the goals (see CMN, 1983, Ch. 11). Of course, users often engage in problem-solving, exploration, and other non-routine activities while using a computer and other cognitive modeling approaches and task analysis techniques can be used to investigate these behaviors (e.g., the Cognitive Walkthrough technique (Wharton, Rieman, Lewis & Polson, 1994) applies to exploratory behavior by novice users). These issues, symbolized by the italicized area at the top of Figure 2, will be discussed further below, and some related research directions will be touched on very briefly. But at this point GOMS would be recommended as a form of task analysis only for tasks in which routine operating procedures are an important aspect of the interaction.

We emphasize, however, that most tasks have some element of routine cognitive skill. Composing a research paper requires the skill of text-editing, charting data requires the skill of entering information into spreadsheets, interactive programming requires text-editing, architectural design with a CAD system requires routine window manipulation, etc. Even if the primary task is not routine, those aspects of the task that are routine are amenable to analysis with GOMS techniques. Applying GOMS to improve the routine aspects of a complex task will reduce the effort necessary to master and perform those routine aspects, getting them out of the way of the primary creative task.

The third restriction is that in all GOMS analysis techniques, the designer or analyst must *start* with a list of top-level tasks or user goals. GOMS analyses and methods do not provide this list; it must come from sources external to GOMS (see also Olson and Olson, 1990, Karat & Bennett 1989). Typically, this list of goals can be obtained from other task analysis approaches (e.g., see Diaper, 1989), such as interviews with potential users, observations of users of similar or existing systems, or in the worst case, simple intuitions on the part of the analyst. Once this list is assembled, GOMS analyses can help guide the design of the system so that the user can accomplish the given tasks in an efficient and learnable way. However, except for possibly stimulating the analyst's intuitions, the subsequent analysis will not identify any new top-level user goals or tasks that the analyst overlooked, or correct a misformulation of the user goals.

The next level down in the diagram consists of specific proposals for how to carry out a task analysis within a GOMS orientation. It is at this level that the differences appear between the different versions of GOMS analysis. Note that the general GOMS concept merely asserts that it is useful to analyze a task in terms of the user's goals, methods, operators, and selection rules. It does not specify any particular technique for doing such an analysis. A particular technique requires (1) more specific definitions of the GOMS components, especially the operators, and (2) guidance and a procedure for constructing the methods in terms of these more specific definitions.

In the discussion that follows, each technique will be summarized, and examples presented, and the relative advantages and disadvantages mentioned.

KLM. The Keystroke-Level Model (KLM) is the simplest GOMS technique, and was originally described in Card, Moran, & Newell (1980a) and later in CMN (Ch. 8). The KLM makes several simplifying assumptions that make it a restricted version of GOMS. In particular, the analyst must specify the method used to accomplish the particular task of interest, which typically entails choosing specific task instances. Other GOMS techniques discussed below predict the method given the task situation and the knowledge of methods and selection rules, but the KLM does not. Furthermore, the specified method is limited to being in sequence form and containing only keystroke-level primitive operators. Given the task and the method, the KLM uses the preestablished keystroke-level primitive operators to predict the time to execute the task.

The original KLM included six types of operators: K to press a key or button, P to point with a mouse to a target on a display, H to home hands on the keyboard or other device, D to draw a line segment on a grid, M to mentally prepare to do an action or a closely-related series of primitive actions, and R to symbolize the system response time during which the user has to wait for the system. Each of these operators has an estimate of execution time, either a single value, a parameterized estimate (e.g., K is dependent on typing speed and whether a key or mouse button click, press, or release is involved), or a simple approximating function. The KLM also includes a set of five heuristic rules for placing mental operators to account for mental preparation time during a task that requires several physical operators.

Subsequent research has refined these six primitive operators, improving the time estimates or differentiating between different types of mental operations (Olson & Olson, 1990) and practitioners often tailor these operators to suit their particular user group and interface requirements (e.g., Haunold & Kuhn, 1994). In addition, the heuristics for placing mental operators have been refined for specific types of subtasks (e.g., for making a fixed series of menu choices, Lane, Napier, Batsell & Naman, 1993). In particular, since the original heuristic rules were created primarily for command-based interfaces, they need to be updated for direct manipulation interfaces. Thus, Heuristic Rule 0 should be expanded to read, "Insert **M**'s in front of all **K**'s that are not part of argument strings proper (e.g., text or numbers). Place **M'**s in front of all **P**'s that select commands (not arguments) *or that begin a sequence of direct-manipulation operations that belong to a cognitive unit.*"

Figure 3 provides a sample KLMs with computation of execution time for moving the phrase in the word processing example in Figure 1, using the operator times supplied in CMN (p. 264).

In terms of underlying architecture, KLM does not need a computational representation because the methods are supplied by the analyst and are expressed in sequence form; all the information-processing activity is assumed to be contained in the primitive operators, including

Moving text with the *MENU-METHOD*

Description	Operator	Duration (sec)
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to beginning of phrase	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
(no M by Heuristic Rule 0)		
Move cursor to end of phrase	P	1.10
(no M by Heuristic Rule 1)		
Shift-click mouse button		
(one average typing K)	K	0.28
(one mouse button click K)	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu	P	1.10
(no M by Heuristic Rule 1)		
Press mouse button	K	0.10
Move cursor to Cut menu item	P	1.10
(no M by Heuristic Rule 1)		
Release mouse button	K	0.10
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to insertion point	P	1.10
Click mouse button	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu	P	1.10
(no M by Heuristic Rule 1)		
Press mouse button	K	0.10
Move cursor to Paste menu item	P	1.10
(no M by Heuristic Rule 1)		
Release mouse button	K	0.10
TOTAL PREDICTED TIME		14.38

Figure 3. A keystroke-level model for moving the text in Figure 1.

internal actions, which are subsumed by black-box Mental operators. Thus the underlying conceptual framework is simply the serial stage model.

The primary advantage of KLM technique is that it allows a rapid estimate of execution time with an absolute minimum of theoretical and conceptual baggage. In this sense it is the most "practical" of the GOMS methodologies: it is the easiest to apply in actual interface design practice, and by far the simplest to explain and justify to computer software developers. This simple estimate of execution times can be used to compare design ideas on benchmark tasks, to do parametric evaluation to explore the space defined by important variables (e.g., the length of filenames in a command language), and to do sensitivity analyses on the assumptions made (e.g., user's typing speed) (CMN; Card, Moran, & Newell, 1980a).

CMN-GOMS. CMN-GOMS is the term we use to refer to the form of GOMS model presented in CMN (1983, Ch. 5; Card, Moran, & Newell, 1980b). CMN-GOMS is slightly more specified than general GOMS; there is a strict goal hierarchy, and methods are represented in an informal pseudo-code-like notation that can include submethods and conditionals. CMN-GOMS describes a task in terms of a hierarchical goal structure and set of methods in program form, each of which consists of a series of steps executed in a strictly sequential order.

In the context of the CMN book, it would appear that the CMN-GOMS model is based on the Model Human Processor (MHP), but in fact CMN do not make a tight linkage. In particular, in presenting the CMN-GOMS formulation, they provide no description of how the MHP would represent and execute CMN-GOMS methods. Furthermore, the GOMS concept itself cannot be derived from the MHP as presented in CMN, but is only loosely based on two of the MHP Principles of Operation, the Rationality Principle and Problem Space principle, both well developed in the problem-solving theoretical literature (e.g., Newell & Simon, 1972; see CMN Ch. 11). Thus, Figure 2 shows that the CMN-GOMS model is based only on the serial stage model.

CMN do not describe the CMN-GOMS technique with an explicit "how-to" guide, but their presentation of nine models at different levels of detail illustrates a breadth-first expansion of a goal hierarchy until the desired level of detail is attained. CMN report results in which such models predicted operator sequences and execution times for text editing tasks, operating systems tasks, and the routine aspects of computer-aided VLSI layout tasks. These examples are sufficiently detailed and extensive that researchers have been able to develop their own CMN-GOMS analyses (e.g., Lerch, Mantei, & Olson, 1989).

Figure 4 is an example of a CMN-GOMS model at the keystroke-level for the text-editing task in Figure 1, including details for the *MOVE-TEXT* goal. Moving is accomplishing by first cutting the text and then pasting it. Cutting is accomplished by first selecting the text, and then issuing the CUT command. As specified by a selection rule set, selecting can be done in two different ways, depending on the nature of the text to be selected. Finally pasting requires selecting the insertion point, and then issuing the PASTE command.

Comparing Figure 4 with Figure 3, the relationship between the CMN-GOMS technique and the KLM technique is evident. (Note that the expansion of the MOVE-TEXT goal in Figure 4 represents the same behavior as the KLM in Figure 3.) For instance, there is a one-to-one mapping between the physical operators in the CMN-GOMS model and the **K**s and **P**s in the KLM, but the CMN-GOMS model has other operators at this level: VERIFY-LOCATION and VERIFY-HIGHLIGHT, which have no observable physical counterpart (they could perhaps be observed with an eye-tracker, but this instrument is not used in any but the most detailed HCI research). The KLM has no explicit goals or choices between goals, whereas the CMN-GOMS model represents these explicitly. Roughly, the VERIFY operators, goal hierarchies and selection rules of the CMN-GOMS model are represented as the **M** operators in the KLM. That is, operators such as VERIFY and goals and selections appear in the CMN-GOMS model in groups

```
GOAL: EDIT-MANUSCRIPT
.  GOAL: EDIT-UNIT-TASK ...repeat until no more unit tasks
.  .  GOAL: ACQUIRE UNIT-TASK ...if task not remembered
.  .  .  GOAL: TURN-PAGE ...if at end of manuscript page
.  .  .  GOAL: GET-FROM-MANUSCRIPT
.  .  GOAL:EXECUTE-UNIT-TASK ...if a unit task was found
.  .  .  GOAL: MODIFY-TEXT
.  .  .  .  [select: GOAL:  MOVE-TEXT* ...if text is to be moved
.  .  .  .           GOAL:  DELETE-PHRASE ...if a phrase is to be deleted
.  .  .  .           GOAL:  INSERT-WORD] ...if a word is to be inserted
.  .  .  .    VERIFY-EDIT

*Expansion of MOVE-TEXT goal
GOAL: MOVE-TEXT
.  GOAL: CUT-TEXT
.  .  GOAL: HIGHLIGHT-TEXT
.  .  .  [select**:  GOAL: HIGHLIGHT-WORD
.  .  .             .  MOVE-CURSOR-TO-WORD
.  .  .             .  DOUBLE-CLICK-MOUSE-BUTTON
.  .  .             .  VERIFY-HIGHLIGHT
.  .  .             GOAL: HIGHLIGHT-ARBITRARY-TEXT
.  .  .             .  MOVE-CURSOR-TO-BEGINNING               1.10
.  .  .             .  CLICK-MOUSE-BUTTON                     0.20
.  .  .             .  MOVE-CURSOR-TO-END                     1.10
.  .  .             .  SHIFT-CLICK-MOUSE-BUTTON               0.48
.  .  .             .  VERIFY-HIGHLIGHT]                      1.35
.  .  GOAL: ISSUE-CUT-COMMAND
.  .  .  MOVE-CURSOR-TO-EDIT-MENU                             1.10
.  .  .  PRESS-MOUSE-BUTTON                                   0.10
.  .  .  MOVE-MOUSE-TO-CUT-ITEM                               1.10
.  .  .  VERIFY-HIGHLIGHT                                     1.35
.  .  .  RELEASE-MOUSE-BUTTON                                 0.10
.  GOAL: PASTE-TEXT
.  .  GOAL: POSITION-CURSOR-AT-INSERTION-POINT
.  .  .  MOVE-CURSOR-TO-INSERTION-POINT                       1.10
.  .  .  CLICK-MOUSE-BUTTON                                   0.20
.  .  .  VERIFY-POSITION                                      1.35
.  .  GOAL: ISSUE-PASTE-COMMAND
.  .  .  MOVE-CURSOR-TO-EDIT-MENU                             1.10
.  .  .  PRESS-MOUSE-BUTTON                                   0.10
.  .  .  MOVE-MOUSE-TO-PASTE-ITEM                             1.10
.  .  .  VERIFY-HIGHLIGHT                                     1.35
.  .  .  RELEASE-MOUSE-BUTTON                                 0.10
                                   TOTAL TIME PREDICTED (SEC) 14.38

 **Selection Rule for GOAL: HIGHLIGHT-TEXT:
   If the text to be highlighted is a single word, use the
   HIGHLIGHT-WORD method, else use the HIGHLIGHT-ARBITRARY-TEXT method.
```

Figure 4. Example of CMN-GOMS text-editing methods showing the top-level unit-task method structure and a selection rule.

that roughly correspond to the placement of **M**s in the KLM. This is only approximately the case, as the VERIFY operators sometimes occur in the middle of a group of physical operators, but the approximation is close.

A major difference between the KLM and the CMN-GOMS models is that CMN-GOMS is in program form, therefore, the analysis is general and executable. That is, any instance of the described class of tasks can be performed or simulated by following the steps in the model, which may take different paths depending on the specific task situation. Goals and method selection are predicted by the model given the task situation, and need not be dictated by the analyst as they must for the KLM.

Given the task specified by the manuscript in Figure 1, this model would predict the trace of operators shown with the estimates of operator times in the far right column. The estimates for the physical operators are identical to the ones in the KLM. The VERIFY-HIGHLIGHT and VERIFY-POSITION operators are assigned 1.35sec, the same value as the KLM's **M** operator because this is CMN's best estimate of mental time in the absence of other information.[1] Thus, the CMN-GOMS model produces the same estimate for task completion as the KLM. Notice that the CMN-GOMS technique assigns time only to operators, not to any "overhead" required to manipulate the goal hierarchy. In their results, CMN found that time predictions were as good with the simple assumption that only operators contributed time to the task as they were when goal manipulation also contributed time, but suggested that at even more detailed levels of analysis such cognitive activity might become more important. Also notice that where the KLM puts **M**s at the beginning of subprocedures, the CMN-GOMS model puts the mental time in verify operators at the end of subprocedures. Since mental time is observable only as pauses between actions, it is difficult to distinguish between these two techniques empirically, and only appeals to more detailed cognitive architectures can explain the distinction. Pragmatically, however, this difference is irrelevant in most design situations. We will discuss the issue of mental time again after presenting all the GOMS techniques.

NGOMSL. Conceptually, the NGOMSL technique (Kieras, 1988a, 1994a, b) refines the CMN-GOMS model by connecting it to a simple cognitive architecture, namely CCT. It originated from attempts to make CCT models more usable by defining higher-level notations to represent the content of a CCT model (see Bennett, Lorch, Kieras, & Polson, 1987, and Butler, Bennett, Polson, and Karat, 1989). This variation of GOMS provides a well-defined, structured natural language, NGOMSL (Natural GOMS Language) suitable for practical application and contains an explicit procedure for constructing GOMS models. NGOMSL models are in program form; they make the method structure very explicit, and can represent very general methods. Continuing the text editing example, Figure 5 shows the NGOMSL methods involved in moving text. Notice that more methods are represented than are executed in the specific task instance being used as an example.

The technique described by Kieras (1988a, 1994a) for constructing NGOMSL models consists of a top-down, breadth-first expansion of the user's top-level goals into methods, until the methods contain only the operators chosen to be primitive, typically keystroke-level operators. The analyst supplies a method for each top-level goal, in which the method steps consist first only of operators (no subgoal invocations). These operators are then rephrased as goal assertions if more detail is required, and then methods are supplied for these goals in the next pass. This heuristic of drafting a method in terms of operators helps the analyst avoid getting bogged down in details prematurely, as does the breadth-first approach, which also appears to help the analyst identify shared methods in the system.

As mentioned previously, NGOMSL is directly based on the CCT architecture. There is essentially a one-to-one relationship between statements in the NGOMSL language and the production rules for a GOMS model written in the CCT format. NGOMSL methods include *internal* operators that represent operations of the CCT architectural mechanisms, such as adding

---

[1] Some design situations may require, or provide opportunity for using better estimates of specific types of mental operators. Analysts can look at the additional empirical work of CMN in Chapter 5 where they measure many specific mental times, or other HCI empirical work (e.g. John & Newell, 1987 for estimates of time to recall command abbreviations, Olson & Olson, 1990, for mental preparation in spreadsheet use).

```
NGOMSL Statements                                                 Executions   External
                                                                               Operator
                                                                               Times
Method for goal: Move text                                              1
  Step 1.  Accomplish goal: Cut text.                                   1
  Step 2.  Accomplish goal: Paste text.                                 1
  Step 3.  Return with goal accomplished.                               1

Method for goal: Cut text                                               1
  Step 1.  Accomplish goal: Highlight text.                             1
  Step 2.  Retain that the command is CUT, and
             accomplish goal: Issue a command.                          1
  Step 3.  Return with goal accomplished.                               1

Method for goal: Paste text                                             1
  Step 1.  Accomplish goal: Position cursor at insertion point.         1
  Step 2.  Retain that the command is PASTE,
             and accomplish goal: Issue a command.                      1
  Step 3.  Return with goal accomplished.                               1

Selection rule set for goal: Highlight text                             1
  If text-is word, then accomplish goal: Highlight word.
  If text-is arbitrary, then accomplish goal: Highlight arbitrary text.  1
  Return with goal accomplished.                                        1

Method for goal: Highlight word
  Step 1.  Determine position of middle of word.
  Step 2.  Move cursor to middle of word.
  Step 3.  Double-click mouse button.
  Step 4.  Verify that correct text is selected
  Step 5.  Return with goal accomplished.

Method for goal: Highlight arbitrary text                               1
  Step 1.  Determine position of beginning of text.                     1        1.20
  Step 2.  Move cursor to beginning of text.                            1        1.10
  Step 3.  Click mouse button.                                          1        0.20
  Step 4.  Determine position of end of text. (already known)           1        0.00
  Step 5.  Move cursor to end of text.                                  1        1.10
  Step 6.  Shift-click mouse button.                                    1        0.48
  Step 7.  Verify that correct text is highlighted.                     1        1.20
  Step 8.  Return with goal accomplished.                               1

Method for goal: Position cursor at insertion point                     1
  Step 1.  Determine position of insertion point.                       1        1.20
  Step 2.  Move cursor to insertion point.                              1        1.10
  Step 3.  Click mouse button.                                          1        0.20
  Step 4.  Verify that correct point is flashing                        1        1.20
  Step 5.  Return with goal accomplished.                               1

Method for goal: Issue a command                                        1
  Step 1.  Recall command name and retrieve from LTM the menu name for it,
             and retain the menu name.                                  1
  Step 2.  Recall the menu name, and move cursor to it on Menu Bar.     1        1.10
  Step 3.  Press mouse button down.                                     1        0.10
  Step 4.  Recall command name, and move cursor to it.                  1        1.10
  Step 4.  Recall command name, and verify that it is selected.         1        1.20
  Step 5.  Release mouse button.                                        1        0.10
  Step 6.  Forget menu name, forget command name, and
             return with goal accomplished.                             1
```

Predicted Procedure Learning Time = 801 sec
Total Predicted Execution Time = 16.38 sec

Figure 5. An example of NGOMSL methods for moving text, showing a generic command-issuing method that uses items in long-term memory to associate menu names to the contained commands. Adapted from Kieras (1994a).

and removing information to working memory or asserting goals to be accomplished. At the keystroke-level, CCT has been shown to provide good predictions of both execution time (based on estimates of time per production firing) and learning time (based on estimates of time to learn a totally new or similar production) (Kieras & Bovair, 1986; Bovair, Kieras & Polson, 1988, 1990). Therefore, NGOMSL models can be used not only to estimate execution time like the KLM and CMN-GOMS models, but also learning time, with the stipulation that only the time for learning procedure steps in a specific learning situation is taken into account, as described in Kieras (1994a) and discussed more below. Although an NGOMSL analysis can provide a useful description of a task even at a high level of analysis (see Karat & Bennett 1989), quantitative predictions of learning and execution times are meaningful only if the methods use operators that the user is assumed to already know and that have known properties, such as keystroke level operators.

The basis for the learning time predictions, and some critical qualifications of them, needs some discussion. CCT and NGOMSL models have been shown to be good predictors of time to learn how to use a system, keeping in mind that what is predicted is the *pure learning time* for the *procedural knowledge* represented in the methods. Note that, as mentioned above, the user is assumed to already know how to execute the operators; the GOMS methods do not represent the procedural knowledge involved in the operators themselves, but only represent the knowledge of which operators to apply and in what order to accomplish the goal. Innovative interface technology often results in new operators, such as moving the cursor with a mouse, selecting objects with an eye-movement tracker, or manipulating 3D objects and flying about in virtual space with data-glove gestures. Clearly, the time to learn how to execute such new operators is a critical aspect of the value of new interface devices, but a GOMS model that *assumes* such operators can not predict their learning time. That is, if new operators are involved, the GOMS analysis can predict only the time required to learn the procedures that use the operators; the time for learning the new operators themselves would have to be measured, or simply not included in the analysis.

The actual total time to learn how to use a system depends not only on how much procedural knowledge is involved but on how much time it takes to complete the training curriculum itself. That is, most learning and training of computer usage takes place in the context of the new user performing tasks of some sort, and this performance would take a certain amount of time even if the user were fully trained. Thus the total learning time consists of the time to execute the training tasks plus the extra time required to learn how to perform the tasks. The *pure* learning time is the excess due to learning, that is, the difference between this total time and the time it would take to execute the tasks if the user were already trained. As pointed out by Gong (1993), these training task execution times can be estimated from GOMS model of the training tasks.

The key empirical result is that the procedure learning time is approximately linear with the number of CCT production rules or NGOMSL statements that must be learned. Thus, the pure learning time for the methods themselves can be estimated just by counting their total length and multiplying by an empirically-determined coefficient. Consistency of the methods, or transfer of training effects, can be represented by deducting the number of NGOMSL statements in methods that are identical, or highly similar, to ones already known to the learner (see Kieras, 1988a, 1994a; also Bovair, Kieras, & Polson, 1988, 1990).

An additional component of the pure learning time is the time required to memorize chunks of declarative information required by the methods, such as the menu names under which commands are found. Such items are assumed to be stored in long-term memory (LTM), and while not strictly part of the GOMS methods, are required to be in LTM for the methods to execute correctly. Including this component in the learning time estimates is thus a way to represent the learning load imposed by menu or command terms, and the heuristics suggested in CMN can be applied to estimate the time to memorize these items based on the number of chunks. However, it should be kept in mind that the heuristics for counting the number of chunks are not very well defined at this time (see Gong, 1993).

In addition, the general requirements of the learning situation must be taken into account as well. The original work by Kieras, Polson, and Bovair used a mastery learning situation, in which the users were explicitly trained on the methods and were required to each procedure fully and exactly before going to the next (Bovair, Kieras, & Polson, 1990; Kieras & Bovair, 1986; Polson, 1988). More recent work by Gong (1993) used a more typical learning situation in which users first were given a demonstration and explanation, and then had to perform a series of training tasks at their own pace, and without detailed feedback or correction. The NGOMSL operators and the number of memory chunks were excellent predictors of this more realistic training time, although the prediction coefficients were different than those given in Kieras (1988a). Furthermore, even in learning situations that are realistically unstructured, at least the ordinal predictions of learning time should hold true, as suggested by results such as Ziegler, Hoppe, & Fahnrich (1986). It seems reasonable that regardless of the learning situation, systems whose methods are longer and more complex will require more time to learn, because there is more procedural knowledge to be acquired, either by explicit study or inferential problem-solving. But clearly more work on the nature of relatively unstructured learning situations is required.

The above discussion of estimating learning time can be summarized as follows, using the values determined by Gong (1993):

Total Procedure Learning Time = *Pure Procedure Learning Time*
+ Training Procedure Execution Time.

Pure Procedure Learning Time = *NGOMSL Method Learning Time*
+ LTM Item Learning Time

NGOMSL Method Learning Time = 17 sec × *Number of NGOMSL Statements to be Learned*

LTM Item Learning Time = 7 sec × *Number of LTM Chunks to be Learned*

These formulas give a pure procedure learning time estimate for the whole set of methods shown in Figure 5 of 801 sec, in a "typical" learning situation and assuming no prior knowledge of any methods or menu terms.

A trace of this NGOMSL model performing the text moving example in Figure 1 is summarized in Figure 5. The trace includes the same sequence of physical operators as the KLM and CMN-GOMS models in Figure 3 and 4. The predicted execution time is obtained by counting 0.1 sec for each NGOMSL statement executed (corresponding to the execution of CCT production rules) and adding the total external operator time, using the values recommended in Kieras (1994). This gives a predicted execution time of 16.38 sec, which is comparable to the predictions of the other two models, which was 14.38 for both the KLM and CMN-GOMS models.

The primary difference between execution time predictions for NGOMSL, KLM and CMN-GOMS is how time is assigned to cognitive and perceptual operators. There are some stylistic differences in how many large mental operators are assumed; for example, the NGOMSL example follows the NGOMSL technique recommendations for the number and placement of DETERMINE-POSITION and VERIFY operators, and so has more of such **M**-like operators than do the CMN-GOMS and KLM models. These stylistic differences could be resolved with further research. But a more important difference is in the nature of the unobservable operators. The KLM has a single crude **M** operator that precedes each cognitive unit of action. NGOMSL, based on CCT, uniformly requires some cognitive execution time for every step, manipulating goals and working memory, and for entering and leaving methods. In contrast, CMN-GOMS assigns no time to such cognitive overhead. But all three models include **M**-like operators for substantial time-consuming mental actions such as locating information on the screen and verifying entries. Thus these methods assign roughly the same time to unobservable perceptual and cognitive activities, but do so at different places in the trace.

Because NGOMSL models specify methods in program form, they can characterize the procedural complexity of tasks, both in terms of how much must be learned, and how much has to be executed. However, NGOMSL models are based on CCT, which in turn assumes a simple serial stage model of human information processing, and so NGOMSL works only for hierarchical and sequential methods, with perceptual and motor activities represented only by external primitive operators like DETERMINE-POSITION and CLICK-MOUSE-BUTTON. NGOMSL models are thus limited in two important ways. First, there is no provision for representing methods whose steps could be executed in any order, or which could be interrupted, suspended and resumed (e.g. for purposes of error recovery). Second, since perceptual and motor activities are represented by operators embedded in the sequential methods, there is no way to represent how these might overlap with other activities. For example, there is no provision for representing a user doing perceptual processing on an icon while simultaneously homing the hand to the mouse and doing a retrieval from long-term memory. So the NGOMSL technique is unsuitable for tasks in which such perceptual-cognitive-motor overlap is important. Such cases would be ones in which the interaction was time-stressed, highly practiced, and involved displays and controls that permitted some degree of parallel activity. These limitations are not serious in many conventional situations involving desktop computing, and also it is possible to approximate overlapping operations by setting certain operator times to zero (as has been done in Figure 5, see Gong, 1993). These limitations could be overcome by extensions to NGOMSL and CCT, but such extensions would be equivalent to using the Parallel Multiple-Processor conceptual framework, and so would alter the technique and computational models in a fundamental way. However, this type of model and analysis is already represented to some extent by CPM-GOMS and some of the research approaches discussed below.

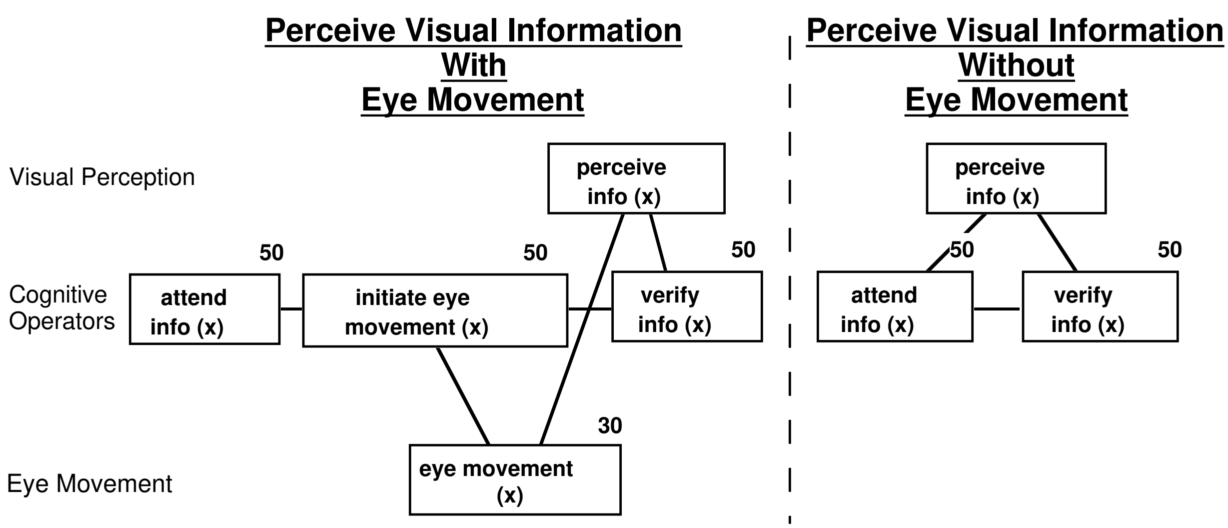
In contrast to the KLM, which we characterized as having the least conceptual baggage, NGOMSL embraces the full psychological theory of CCT. For instance, NGOMSL analyses make a commitment to deliberate goal and working memory management. As we will discuss later, the need to understand these theoretical mechanisms probably increases the time to learn how to do NGOMSL analysis compared to KLM or CMN-GOMS analyses. However, since NGOMSL is the only current GOMS variant that predicts both performance time and learning time, an analyst may be willing to master the technique's concepts to reap the benefits of its predictive power.

CPM-GOMS. CPM-GOMS is a task analysis technique based directly on the Model Human Processor (MHP), and thus, on the parallel multi-processor stage model of human information processing. It does not make the assumption that operators are performed serially, i.e., perceptual, cognitive and motor operators at the level of MHP processor cycle times can be performed in parallel as the task demands. CPM-GOMS uses a *schedule chart* (or *PERT chart*, familiar to project managers, e.g. Stires & Murphy, 1962) to represent the operators and dependencies between operators. The acronym *CPM* stands for both the *C*ognitive-*P*erceptual-*M*otor analysis of activity, and also *C*ritical *P*ath *M*ethod, since the *critical path* in a schedule chart provides a simple prediction of total task time.

To build CPM-GOMS models the analyst begins with a CMN-GOMS model of a task with operators at a level such that they are primarily perceptual (READ-SCREEN, LISTEN-TO-CUSTOMER) or motor (ENTER-COMMAND, GREET-CUSTOMER). These operators are then expressed as goals and implemented with methods of MHP-level operators. John & Gray (1992, 1994) have developed templates of the combinations of MHP-level cognitive, perceptual and motor operators that implement many different activity-level goals under different task conditions. For instance, the READ-SCREEN goal is accomplished with the operators in the first template in Figure 6 when an eye-movement is required and with the second template when it isn't because the user is already looking at the point where the highlighting will appear. Each operator in the templates is associated with a duration estimate, or a set of estimates that also depend on task conditions. For instance, visually perceiving and comprehending a 6-character word is has a duration of 290 ms, whereas visually perceiving and comprehending that a symbol is merely present or absent (e.g., the presence of highlighting) has a duration of 100 ms, as is shown in Figure 6.

These templates are first joined together serially, and then interleaved to take advantage of the parallelism of the underlying conceptual architecture. If empirical data about actual performance of observable motor operators is available from a current system that is similar to the system being designed, it is desirable to verify the model against these data. Then the verified models are modified to represent the proposed design and quantitative predictions of performance time can be determined from the critical path of the CPM-GOMS model. Qualitative analysis of what aspects of a design lead to changes in the performance time are quite easy once the models are built, as are subtask profiling, sensitivity and parametric analyses, and playing "what-if" with suggested design features (Chuah, John & Pane, 1994; Gray, John & Atwood, 1993).

Continuing the example of the MOVE-TEXT goal of Figure 1, Figure 7 shows a CPM-GOMS model in the style of Gray, John & Atwood (1993). For brevity, the model covers only the portion of the procedure involved with highlighting the text to be moved. Before discussing this



The duration of a perception of a simple binary visual signal is 100 msec.
The duration of a perception of a complex visual signal similar to a 6 letter word is 290 msec.

If the eyes have been focusing at a place other than the infomation to be perceived, or if unrelated cognitive activity precedes the attend-info operator, then an eye movement is required, with it's associated cognitive initiation; use the template "Perceive Visual Information With Eye Movement".

If the user perceived information in that exact area immediately prior to the current situation, with no intervening visual or auditory perception or cognitive activity, then it is assumed that the eyes remained fixed on the appropriate area and no eye-movement is necessary; use the template "Perceive Visual Information Without Eye Movement".

Figure 6. Example of a template for building CPM-GOMS models adapted from John & Gray, 1994.

model in detail, however, it is important to note that text-editing is not a good application of the CPM-GOMS technique and we present it here only so that similarities and differences to the other GOMS variations are clear. Text-editing is *usefully approximated* by serial processes, which is why the KLM, CMN-GOMS and NGOMSL have been so successful at predicting performance on text-editors. The CPM-GOMS technique is overly detailed for such primarily serial tasks and as will become clear, can underestimate the execution time. For examples of tasks for which a parallel-processing model is essential, and where the power of CPM-GOMS is evident, see the telephone operator task in Gray, John and Atwood (1993) and transcription typing (John, 1988; John & Newell, 1989).

Although text-editing is not the best task to display the advantages of CPM-GOMS, there are several interesting aspects of the model in Figure 7 compared to the example models of the text-moving task in the preceding sections. First, there is a direct mapping from the CMN-GOMS model to the CPM-GOMS model, because all CPM-GOMS models start with CMN-GOMS and the particular model in Figure 7 was built with reference to the one in Figure 4. As with the KLM, selection rules are not explicitly represented because CPM-GOMS models are in sequence form, and the analyst implements the selection by choosing a particular method for each task. For example, in Figure 7, the selection between *HIGHLIGHT-ARBITRARY-PHRASE* and *HIGHLIGHT-WORD* that is explicitly represented in CMN-GOMS and NGOMSL, is only implicit in the analyst's choice of the method for this model. The times for the various operators are shown on the boxes in the schedule chart, based on the durations estimated by John & Gray (1994), and the highlighted lines and boxes comprise the critical path.

Parallelism in the model is illustrated in the set of operators that accomplish the *MOVE-TO-BEGINNING-OF-PHRASE* goal. These operators are not performed strictly serially, that is, the eye-movement and perception of information occur in parallel with the cursor being moved to the new location. The information-flow dependency lines between the operators ensure that the eyes must get there first, before the new position of the cursor can be verified to be at the right location, but the movement of the mouse takes longer than the eye-movement and perception, so it defines the critical path.

Multiple active goals can be represented in CPM-GOMS models and are illustrated in Figure 7 in the sets of operators that accomplish the *MOVE-TO-END-OF-PHRASE* goal and the *SHIFT-CLICK-MOUSE-BUTTON* goal. Because the shift key is hit with the left hand (in this model of a right-handed person) and the mouse is moved with the right hand, the pressing of the shift-key can occur while the mouse is still being moved to the end of the phrase. Thus, the operators that accomplish the *SHIFT-CLICK-MOUSE-BUTTON* goal are interleaved with the operators that accomplish the *MOVE-TO-END-OF-PHRASE* goal. This interleaving represents a very high level of skill on the part of the user.

Reading the total duration on the final item of the critical path gives a total execution time through this subsequence of the task of 2.21 sec. Totaling the execution time over the same steps in the other models gives 4.23 sec for both the KLM and CMN-GOMS and 6.18 sec for the NGOMSL model.

Although the qualitative process in this example of a CPM-GOMS model is reasonable, its quantitative prediction is much shorter than the estimates from the other models. The primary source of the discrepancy between the GOMS-variants is the basic assumption in the commonly-used form of the CPM-GOMS technique that the user is extremely experienced and executes the task as rapidly as the MHP architecture permits. It should be kept in mind that this particular example task is not really suitable for CPM-GOMS, but is presented to facilitate comparison with the other techniques, and show how CPM-GOMS can represent parallel activities in the same editing task. Some discussion of why the CPM-GOMS technique predicts an execution time that is so much shorter than the others will help clarify the basic assumptions of this form of GOMS analysis.

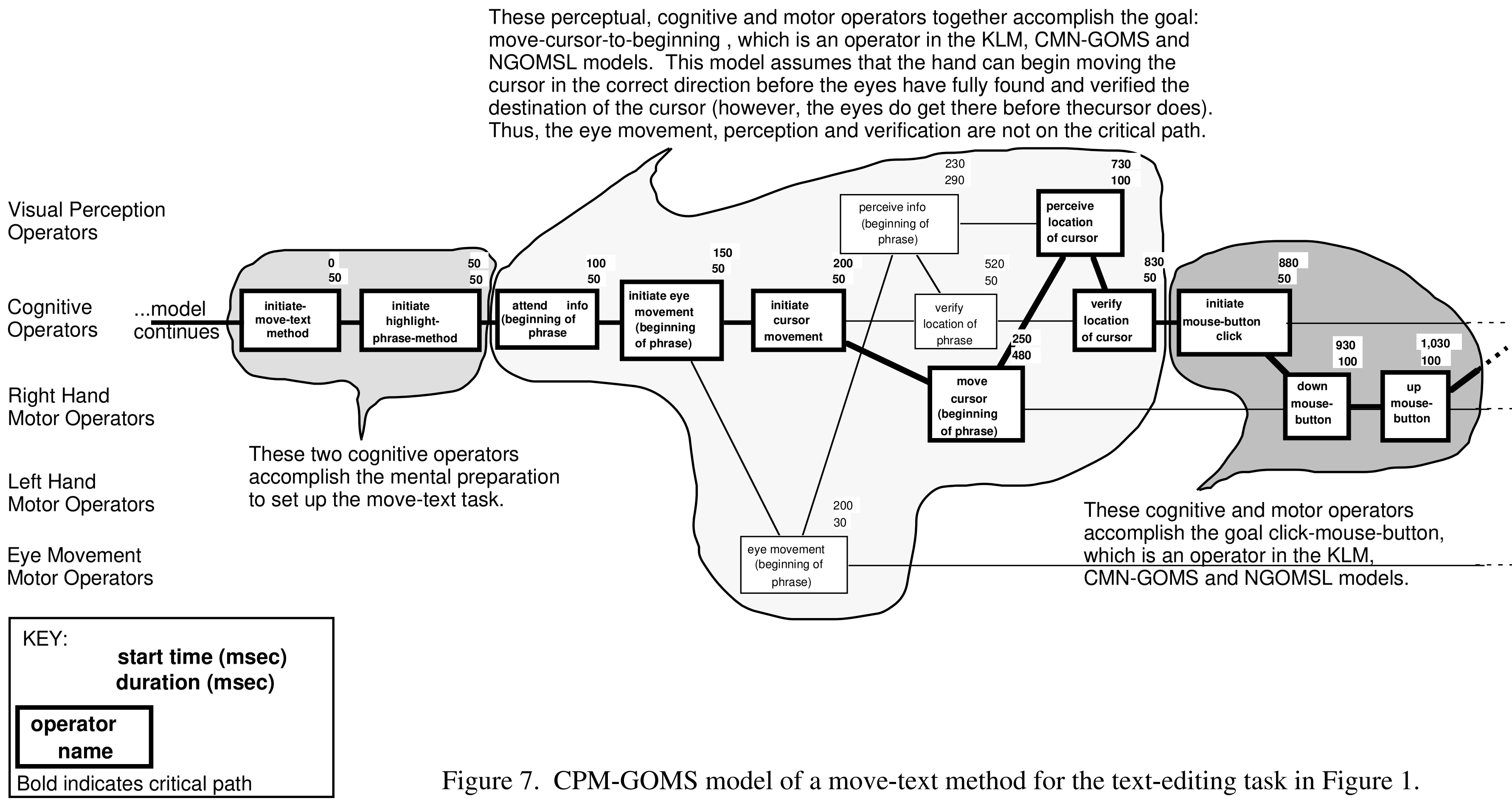
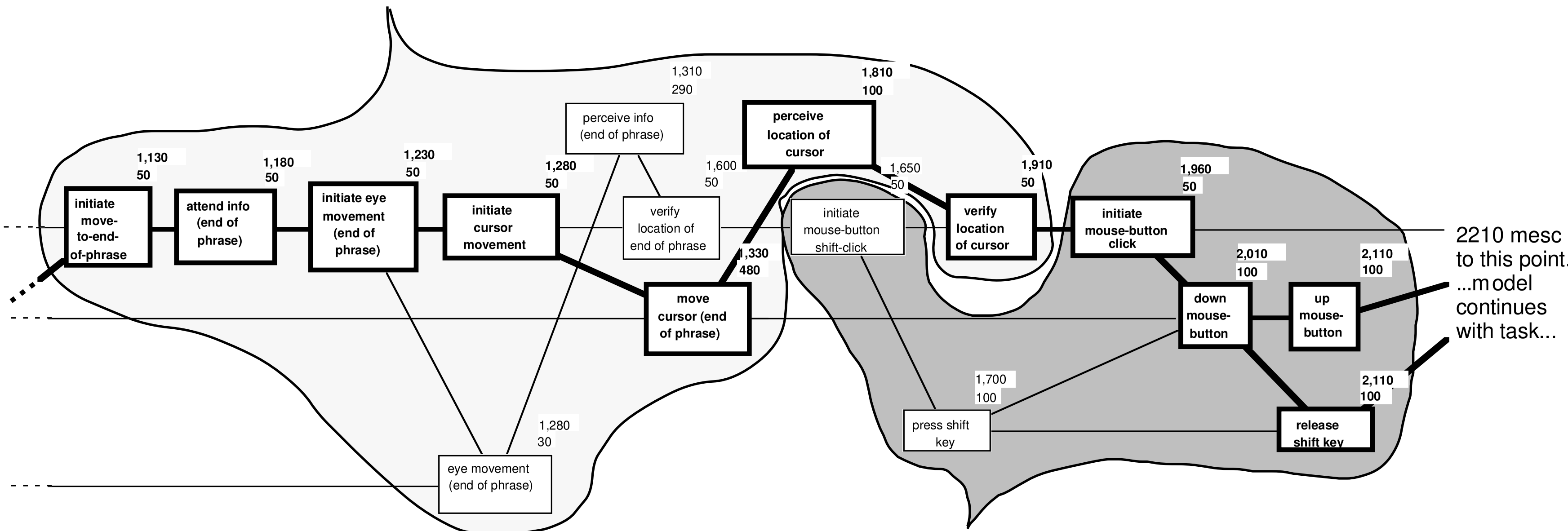


Figure 7. CPM-GOMS model of a move-text method for the text-editing task in Figure 1.

These perceptual, cognitive and motor operators accomplish the goal: move-to-end-of-phrase which is an operator in the KLM, CMN-GOMS and NGOMSL models. Notice how operators for the next goal: shift-click-mouse-button, are interleaved with the operators for this goal. This is how multiple active goals are represented in CPM-GOMS models and are an indication of extreme expertise.



These cognitive and motor operators accomplish the goal of shift-click-the-mouse-button. Notice that they interleave with the previous goal, occur in parallel with other operators, and thus only some of the six operators contribute time to the critical path.

Figure 7 (con't). CPM-GOMS model of a move-text method for the text-editing task in Figure 1.

One aspect of the extreme-expertise assumption is that the example model assumes that the user knows exactly where to look for the to-be-moved-phrase. This means that the model needs only one eye-movement to find the beginning and one to find the end of the target phrase and that the mouse movements to these points can be initiated prior to the completion of the eye movements. In some real-world tasks, like telephone operators handling calls (Gray, John, and Atwood, 1993), the required information always appears at fixed screen locations, and with experience, the user will learn where to look. But in a typical text editing task like our example, the situation changes from one task instance to the next, and so visual search would be required to locate the target phrase. The CPM-GOMS has been used to model visual search processes (Chuah, John & Pane, 1994), but for brevity, we did not include this complexity in our example.

A second aspect of the assumed extreme expertise is that the example does not include any substantial cognitive activity associated with selection of methods or complex decisions. Such cognitive activity is represented in the other GOMS variants with **M**-like operators of about a second in duration. In contrast, in Figure 7, the method selection is implicit in a single cognitive operator (INITIATE-MOVE-TEXT-METHOD) which is the minimum cognitive activity required by the MHP to recognize a situation and note it in working memory. Likewise, VERIFY-POSITION operators are included in the CPM-GOMS model, but they represent much more elementary recognitions that the cursor is indeed in the location where the model is already looking rather than complex verifications that a text modification has been done correctly required in CMN-GOMS and NGOMSL. Thus, Figure 7 represents the absolute minimum cognitive activity, which is an unreasonable assumption for a normal text-editing task. However, in an experiment by CMN (pp. 279-286), the performance time of an expert user on a novel editing task was well predicted by the KLM, but after 1100 trials on the *exact same task instance*, the performance time decreased by 35%, largely because the **M** operators became much shorter. It is this type of extreme expertise that our example CPM-GOMS model represents. A more elaborate CPM-GOMS model could represent complex decisions as a series of MHP-level operators performing minute cognitive steps serially, as in the earlier work on recalling computer command abbreviations and transcription typing in John (1988) and John & Newell (1989). However, the technique for modeling complex decisions in CPM-GOMS models is still a research issue, and so they currently should be used only for tasks in which method selection is based on obvious cues in the environment and decisions can be represented very simply.

A final contributor to the short predicted time is that the mouse movements in CPM-GOMS are calculated specifically for the particular target size and distance in this situation, yielding much shorter times than CMN's 1.10 sec estimate of average pointing time used in the other models (further discussion appears in the next section).

Thus the CPM-GOMS technique allows one to represent the overlapping and extremely efficient pattern of activity characteristic of expert performance in a task. The main contrast with the other techniques is that CPM-GOMS models constructed with the current technique do not include the time-consuming **M**-like operators that the other models do, and that would be expected to disappear with considerable practice if the system interface holds the relevant aspects constant. In fact, if the **M**-like operators are excluded from the execution time of the other models, the predicted times are much closer to the CPM-GOMS prediction, being 2.88 sec for KLM and CMN-GOMS, and 3.78 sec for NGOMSL.

Like NGOMSL, CPM-GOMS carries a substantial amount of conceptual and theoretical baggage. Since it is based on the MHP, it requires an understanding of parallel processing and information-flow dependencies. These concepts have implications for the ease of learning CPM-GOMS, and will be discussed later.

Summary comparison of GOMS techniques. We have modeled the same goal, *MOVE-TEXT*, with four different GOMS task analysis techniques. For purposes of comparison, we included a CPM-GOMS model for the same text-editing task, although the technique is not recommended

Table 1

Predicted time measures (seconds) for each technique for the MOVE-TEXT *example* .

	KLM	CMN-GOMS	NGOMSL	CPM-GOMS
<u>Overall Measures</u>				
Procedure Learning (both highlighting methods)	---	---	801.	---
Total Example Task Execution Time	14.38	14.38	16.38	not shown in this example
<u>Text Highlighting Sub-Method</u>				
Complete Method Execution Time	4.23	4.23	6.18	2.21
Method Execution Time with no long **M**-like Operators	2.88	2.88	3.78	2.21
Total Cognitive Overhead	---	---	0.90	1.10

for modeling such sequential tasks, and for brevity, it was shown only for the text-highlighting submethod. The KLM, CMN-GOMS and NGOMSL models all produce the same sequence of observable operators, as does the CPM-GOMS model (although at a more detailed level). Table 1 summarizes the quantitative predictions from the above presentation, both for the overall example task, and the subtask consisting just of highlighting the to-be-moved text.

NGOMSL is the only one of the four techniques that makes learning time predictions, and these are limited to the effects of the amount of procedural knowledge and related LTM information to be learned, and to learning situations for which the coefficients have been empirically determined.

KLM, CMN-GOMS, and NGOMSL produce execution time predictions that are roughly the same for both the overall task and the subtask, although they make different assumptions about unobservable cognitive and perceptual operators and so distribute the time in different ways (see below). An important difference is that the NGOMSL technique currently entails more **M**-like operators than the other techniques, as well as some cognitive overhead due to method step execution; thus NGOMSL will typically predict execution times that are longer than KLM or CMN-GOMS predictions.

As shown in the execution time predictions for the text-highlighting submethod, the CPM-GOMS model substantially underpredicts the execution time relative to the other models. As discussed above, this is due to the assumption of extreme expertise in the current CPM-GOMS technique: using maximum operator overlapping, finer-grain time estimates for the individual operators, and assuming the minimum of cognitive activity allowed by the MHP. An interesting similarity between NGOMSL and CPM-GOMS is the roughly similar cognitive overhead time in the example submethod; in NGOMSL this value is the statement execution time at 0.1 sec/statement; in CPM-GOMS it is the total time for which the cognitive processor is on the critical path in Figure 7.

Table 2 shows the different operator times assumed in the different techniques and used in the example. For the KLM, CMN-GOMS, and NGOMSL models, the estimates for the **M**-like operators shown (Mental Preparation, Determine Position, and Edit Verification) are those currently recommended for each technique as average values to be used in the absence of more specific measurements. They are all roughly the same at about a second duration, but are slightly different because they were determined empirically with different data sets at different historical points in the development of GOMS techniques. None of the these techniques have a theoretical commitment to any particular value. Any available empirically determined values for the operators involved in a particular analysis should be used instead of these average estimates. More significant are the differences in the distribution of mental time: the KLM tends to place mental time in the preparation for action, while CMN-GOMS mental time tends to come at the end of actions in VERIFY operators, and NGOMSL has mental time in both places. These stylistic differences could probably be resolved with further research.

On the other hand, values of those same operators in CPM-GOMS are theoretically driven, as they connect to the MHP and its cognitive cycle time (estimated at 70 ms CMN, but refined by subsequent work to be 50 ms, John & Newell 1990; Nelson, Lehman & John, 1994; Wiesmeyer, 1992) Both the duration and position of these unobservable operators are specified by the templates used to construct the model. The entry for Mental Preparation is the sum of the durations of the two cognitive operators on the critical path that set up the move-text task and highlight-phrase subtask. The entry for Determine Position is the sum of the durations of those operators that locate the beginning of the phrase on the screen that occur on the critical path. Locating this point involves 3 cognitive operators, 1 eye-movement motor operator that would be unobservable except with an eye-tracker, and 1 perceptual operator. All of these operators depend on each other and have to occur in order, thus, if this were the only activity taking place in a task, they would all be on the critical path and take 420 ms. However, since looking for the beginning of the phrase is just one part of the move-text task, other activities can occur in parallel

Table 2

Operator times (seconds) used in each technique for the MOVE-TEXT example. See text for explanation of the CPM-GOMS entries.

	KLM	CMN-GOMS	NGOMSL	CPM-GOMS
Mental Preparation	1.35	not used	not used	0.100
Determine Position	not used	not used	1.20	0.100
Edit Verification	not used	1.35	1.20	not used
Cursor movement	1.10	1.10	1.10 or Fitts' Law	0.680 by Fitts' Law
Click-mouse-button	0.20	0.20	0.20	0.250
Shift-click- mouse-button	0.48	0.48	0.48	0.250

(e.g., moving the mouse, discussed in the last section) and their operators are interleaved with these, making the critical path more complicated, so that only the first two cognitive operators appear on the critical path for this task.

The operator times for cursor movement deserves a brief note. The 1.10 sec used in the techniques is the average value suggested by CMN for large-screen text editing tasks. But Gong (1993) found that many of the mouse movements involved in using a Macintosh interface, such as making menu selections and activating windows, were much faster than 1.10 sec, and that Fitts' Law estimates (see CMN, p. 55) were much more accurate. Thus, Fitts' Law values based on the actual or typical locations of screen objects should probably be used whenever possible in all of the techniques. For CPM-GOMS, moving the cursor to point to an object is a combination of cognitive operators, motor operators and perceptual operators (see Figure 7) and only some of them occur on the critical path in any task situation. The duration of the mouse-movement motor operator itself was calculated using Fitts' Law. In this example, moving to the beginning of the phrase put 680 msec on the critical path and, coincidentally, moving to the end of the phrase also put 680 msec on the critical path.

Finally, the times for mouse button operators and using the shift key in the first three techniques are based on values from CMN. The slightly different value for a mouse click in the CPM-GOMS technique can be read from the example in Figure 7. That is, clicking the mouse button requires a 50 ms cognitive operator and two motor operators at 100 ms each. The fact that the shift-click takes the same time as the simple click is due to the shift key operation being overlapped with earlier processing, so that it is not on the critical path.

3.4. Some Research Directions

Even a cursory review of the cognitive modeling research efforts in HCI is beyond the scope of this paper, even if attention is restricted to those that are likely will lead to additional engineering models. However, it is worthwhile to briefly discuss some important research topics that are indicated by the italicized areas in Figure 2.

As mentioned before, there are properties of tasks that are not captured by GOMS models, either because more than just procedural knowledge is involved, or because the knowledge is not in the form of routine cognitive skill, but rather in a state that requires complex reasoning or problem solving. As suggested by the *other approaches* label at the top of Figure 2, research in this area could lead to additional design techniques based on identifying the critical properties of a task domain and determining whether the system design has the appropriate relationship to the task. For example, Kieras (1988b) suggests heuristics for determining what "mental model" information about the system should be conveyed to users. Other researchers have proposed models of how users learn a system by interacting with and observing it (e.g., Polson & Lewis, 1990; Howes, 1994). A different approach would be to describe the general learning processes that eventually lead to a routine cognitive skill, for example the use of a general analogy mechanism in both ACT-R and Soar (Rieman, Lewis, Young and Polson, 1994). It remains to be seen whether useful engineering models emerge from research on these highly unstructured situations, and whether the models have clear relations to the GOMS family, or take a different form.

The other italicized area concerns various other computational architectures and their application to HCI tasks. Some research architectures, such as Construction-Integration (Doane, Mannes, Kintsch, & Polson, 1992; Doane, McNamara, Kintsch, Polson, & Clawson, 1992; Kitajima & Polson, 1992) and ACT-R (Rieman, Lewis, Young, & Polson, 1994) have been applied to the analysis of HCI problems, but it is not yet clear what kind of design techniques will result. A particularly important issue is the underutilization of the parallel multiprocessor conceptual framework. The only currently documented GOMS technique based on this framework is CPM-GOMS, and as pointed out below, there is currently a lack of models and techniques for many of the related design issues. Work underway by John and colleagues (Nelson, Lehman, & John, 1994) applies the Soar cognitive architecture to the same kinds of interactions as CPM-GOMS, including using a Soar model to generate the PERT charts for a CPM-GOMS analysis. A new computational architecture occupying this space is the EPIC architecture being developed by Kieras, Meyer, and Wood (Kieras & Meyer, 1994; Wood, Kieras, & Meyer, 1994). In EPIC, a production-rule cognitive processor is embedded in a set of parallel-running perceptual and motor processors whose properties are based on the current human performance literature. Work thus far shows that human performance data in high-performance parallel tasks can be quantitatively predicted by EPIC models using GOMS methods in program form. If techniques can be articulated for constructing such models routinely, then the range of tasks covered by GOMS methodology will be substantially increased.

4. Applying GOMS to Design

When a designer approaches a design task, he or she applies the heuristic, analytic and empirical design techniques known to be useful for the task at hand. For instance, as illustrated by the presentation in Oberg, Jones, & Horton (1978), a mechanical engineer designing a flywheel may use algebraic equations to estimate the initial dimensions of the wheel (analytic technique), then make sure the design is within the maximum safe speed for that type of wheel from tables of empirical results (empirical technique), and finally modify his or her design by including a safety factor (heuristic technique). In order to apply these techniques, a designer must know what techniques are available, to what design tasks they are applicable, and whether the benefit from applying the technique outweighs the effort to apply it. In the preceding sections of this paper, we laid out the GOMS family of analytic techniques available to the computer system designer. In this section, we provide the additional information required for a designer to choose one of these techniques: which techniques are suitable for which design situations, what are the benefits of using each of techniques, and an estimate of the effort involved in using the technique.

A *design situation* has two characteristics important to selecting a GOMS analysis technique: the type of task the users will be engaged in, and the types of information gained by applying the technique. Figure 8 shows which GOMS family methods can be used for each combination of type of task and type of information. In this section of the paper, we will first discuss the task type dimension, and then for each type of information gained, we will describe the issues involved in using the different GOMS techniques that apply. The fact that some of the cells are empty points to a need for further research on GOMS family techniques. In some cases, existing techniques could be modified and adapted for these cases, but the Figure presents the techniques as currently documented.

4.1. Characterizing the User's Tasks

Although user tasks can be characterized in many different ways, three dimensions are important for deciding whether a GOMS analysis technique is applicable to the user's task, and which technique is most suitable: the degree of routinized skill involved in the user's task, the sequentiality of the user's task, and the degree to which the interaction is under the control of the user versus the computer system or other agents involved in the task.

Skill. The skill dimension of tasks runs from one extreme of problem-solving, where the user does not know how to perform a task and must search for a solution, to *routine cognitive skill*, where the user knows exactly what to do in the task situation and simply has to recognize that situation and execute the appropriate actions (see CM&N, 1983, Chapter 11). As previously discussed, the extant GOMS techniques apply only to the routine end of this dimension. GOMS has no direct way of representing the nature or difficulty of the problem-solving required to *discover* the appropriate operators, methods, or selection rules; rather, understanding and predicting such behavior is an active area of cognitive psychology research. Because of this

Information Type \ Task Type	Routine Cognitive Skill: Passive System		Routine Cognitive Skill: Active System	
	Sequential	Parallel	Sequential	Parallel
Functionality: Coverage	Any GOMS	Any GOMS	Any GOMS	Any GOMS
Functionality: Consistency	NGOMSL		NGOMSL	
Operator Sequence	CMN-GOMS NGOMSL	CPM-GOMS	see text	CPM-GOMS
Execution Time	KLM CMN-GOMS NGOMSL	CPM-GOMS	see text	CPM-GOMS
Procedure Learning Time	NGOMSL		NGOMSL	
Error Recovery Support	Any GOMS	Any GOMS	Any GOMS	Any GOMS

Figure 8. GOMS techniques available for different combinations of task type and the type of design information desired. Note that only tasks that are routine cognitive skills are included, and information types not provided by GOMS models are not shown.

limitation, Figure 8 shows that the only task type for which GOMS models apply are routine cognitive skills.

It is important to remember, however, that most computer-based tasks, even very open-ended ones, have substantial components of routine cognitive skill. First, many tasks will evolve from problem-solving to routine skill after extensive use, and predicting a fully practiced user's performance is valuable, because such performance can not be empirically measured for a system that is just being designed and not yet implemented. Second, many tasks have elements of both routine skill and problem-solving. For instance, CMN (Ch. 10) showed that the expert's task of laying out a printed circuit board with a CAD tool was about half problem-solving to figure out what to do next (i.e., acquire the next unit task) and half execution of the unit task, predictable by a GOMS technique. One detailed study of using a new programming language to create a graphing application showed that embedded in the problem-solving activities of designing the program and figuring out how to use the new language was the routine behavior of manipulating the help system and that GOMS was applicable to the analysis of this behavior (Peck & John, 1992). Other examples of GOMS analysis of routine use in otherwise complex tasks include the widely studied text-editing situation, spreadsheet use (Lerch, Mantei & Olson, 1989), digital oscilloscope use (Lee, Polson & Bailey, 1989), and even playing a video game (John & Vera, 1992; John, Vera & Newell, 1994). Thus, although a user's task may seem to be primarily a problem-solving task, there will be aspects of that task that involve routine cognitive skill. It is these aspects of the system design for which GOMS analysis can be used to improve the design to allow users to more effectively work on the non-routine, creative parts of the overall task.

Sequential vs. parallel activity. Many HCI tasks can be usefully approximated as sequential application of operators, such as text-editing. Other tasks involve so much overlapping and parallel activities that this simplification does not usefully approximate the task, as in the telephone operator tasks analyzed by Gray, John and Atwood (1992, 1993). Because currently only the very detailed CPM-GOMS is applicable to the parallel case, it is important to consider when a task involving some parallel operations can be usefully approximated by a sequential model. One such case is when the parallel operations can be represented as a simple modification to the sequential model. For example, in the text-editing example used throughout this paper it is logically necessary that users must determine visually the location of an object before they point at it with a mouse. In a sequential analysis, there would be a operator such as VISUALLY-LOCATE-OBJECT followed by a POINT-TO-OBJECT operator. But practiced users can apparently visually locate a fixed object on the screen (e.g., items on a menu bar) simultaneously with pointing at it with a mouse, meaning that these two operators can execute in parallel. This parallel execution can be approximated in a sequential model by simply setting the time for the VISUALLY-LOCATE-OBJECT operator to zero (see Gong, 1993).

The second case is when the parallel operations are taking place below the level of analysis of the design issues in question, or independently of them. For example, NGOMSL could be used to determine if a telephone operator's procedure for entering a billing number was consistent across different task contexts. As long as the configuration of parallel operators does not differ between design alternatives or task contexts, such a sequential analysis could be useful. But note that since the NGOMSL model would not accurately reflect the underlying production-rule structure for such a task, the quantitative measures of the effect of consistency on reduction of learning time would be suspect; thus NGOMSL should not be used for quantitative predictions for parallel tasks.

Locus of control. Computer system tasks can be roughly categorized into *passive-system* tasks and *active-system* tasks. In passive-system tasks, the user has control over the pace and timing of task events; the computer merely sits and waits for inputs from the user. Text editing is a typical passive-system task. In active-system tasks, the system can produce spontaneous, asynchronous events outside of the user's control. Thus the user must be prepared to react to the system, which can also include other people who are providing information or making requests. Telephone operator tasks and aircraft piloting are good examples of active system tasks. Many video games are maniacally extreme active systems. The introduction of artificial intelligence techniques into an interface to anticipate user’s needs is likely to result in active systems.

The difference between GOMS analyses for active and passive systems is somewhat subtle. The basic concept of GOMS models and analysis is indifferent to whether the system is active or passive, in that in either case, one can describe the methods and selection rules that the user must possess in order to accomplish goals with the system. The difficulty lies in the underlying human information-processing architecture. An active system may produce events that require the user to abandon the current goal and set up a new goal to accomplish. In contrast, the goals involved in using a passive system are static in the sense that once created, they endure until accomplished. Thus active systems may entail methods in which goals have to be suspended, dropped, or created on the fly. In the general case, humans interacting with an active system can be modeled only by cognitive architectures that permit dynamic goal rescheduling. Such processing involves modifying the goal stack, or perhaps rebuilding it from information in memory or on an external display. For example, a simple relaxing of CMN-GOMS' strict goal-stack allowed John and colleagues to predict the functional-level and keystroke-level operators of a nine-year-old expert playing a videogame (John & Vera, 1992; John, Vera & Newell, 1994).

For practical analysis with currently developed techniques, there are two approaches available for active systems. The first is a GOMS analysis in sequence form such as KLM or CPM-GOMS, in which a particular pattern of activity involving goal rescheduling could be represented. Here, the interruptability is handled by the analyst. This approach was used successfully by Gray, John & Atwood (1992, 1993) in modeling telephone operators in their interaction with customers.

The second approach is to construct a CMN-GOMS or NGOMSL model under the approximating assumption that the active system produces events that can be responded to with methods that either will not be interrupted, or do not conflict with each other. Typically the top-level method simply waits for an event, and then invokes whatever submethods are appropriate for responding to the event. For example, the complex operator's associate system analyzed by Endestad & Meyer (1993) had this structure. This approximation clearly fails to deal with the case in which the user must respond to simultaneous or mutually interrupting events, but the analysis can still be useful in identifying usability problems with the system.

4.2. Design Information Provided by GOMS Models

Figure 8 shows several types of design information that GOMS models can provide. Clearly there are many other kinds of information relevant to design; these are not included in the Figure because there are no GOMS family techniques for them. Examples of kinds of information not provided by GOMS are: (1) standard human factors issues such as readability of letters and words on the screen, visual quality of a display layout, recognizability of menu terms or icons, and memorability of commands; (2) the quality of the work environment, user acceptance, and affect (e.g., is the system fun to use or does it cause boredom, fatigue, and resentment); (3) the social or organizational impact of the system and the resulting influence on productivity.

An additional type of design information not shown in Figure 8 is an informal understanding of the design issues. That is, as pointed out by Bennett & Karat (1989), a GOMS analysis can have purely heuristic value. Since a GOMS model makes explicit what the system requires the user to do, constructing it is a way for a user interface designer to become more aware of the implications of a design. Since a common design error is to produce a system without careful consideration of what it imposes upon the user, any exercise that requires the designer to think carefully about the procedures entailed by the design can help in a purely intuitive way to identify usability problems and clarify the nature of the user's task.

Functionality: Coverage and consistency. The primary design question about functionality is whether the system provides some method for every user goal. As discussed above, GOMS methods cannot generate or predict the range of goals a user might bring to a system. However, once the designer generates a list of likely user goals, any member of the GOMS family can be used to check that a method exists for each one in a proposed or existing system, as shown in Figure 8.

While an analysis of functional coverage may not require a formal description of the methods, it is important to consider the content of the methods at least informally. That is, decisions about functionality are based on whether the system provides a function reasonably suitable for the task, meaning that the method involved must be reasonably simple and fast. For example, consider that the user of a word processor might have the goal of putting footnotes at the bottom of the appropriate pages. Some word processors have functionality specialized for footnoting, and so have very simple methods for accomplishing this goal. Other word processors lack this functionality, and so can put footnotes on a page only if the user places and formats them "by hand", and this work has to be redone if the length of the text changes. Despite this clumsiness, such limited word processors still provide a method for accomplishing the footnoting goal. Thus, it would be rare that functionality in a simple all-or-none sense is considered in an interface design; there are at least implicit requirements on performance or learning time for the corresponding methods. If it is important to quantify these requirements, GOMS family members can provide quantitative predictions as discussed in the next two subsections.

Information can be obtained about functional consistency by comparing methods and the knowledge necessary to perform different commands. NGOMSL is particularly suited to an analysis of consistency, because the structure and content of NGOMSL methods can be inspected, and the learning time predictions of NGOMSL and CCT take this form of consistency into account. That is, a consistent interface is one in which the same methods are used throughout for the same or similar goals, resulting in fewer methods to be learned. Furthermore, Kieras (1988, 1994a) provides simple heuristics for assessing similarity of methods for consistency analysis.

Operator sequence. Two of the GOMS family members, CMN-GOMS and NGOMSL, can predict the sequence of overt physical operators a user will perform to accomplish a task (whereas with KLM, the analyst must supply the sequence of operators). That is, the methods and selection rules specify which commands a user will enter, the menu-items they will select, and so forth, to accomplish their goals. The best technique for exploring operator sequences on sequential tasks depends on the number of benchmark tasks being considered, in a way described below for execution time predictions. The situation for parallel tasks is much less rich. At this time, only CPM-GOMS is available for studying the sequence of operators and their possible interleaving in a parallel task. As described above, currently, the CPM-GOMS technique results in models in sequence form in a PERT chart. Although each PERT chart represents only one configuration of operators that the analyst has chosen to represent, these charts are easily manipulated with project management software, so that many configuration can be explored quickly. Thus, the technique allows the analyst to investigate the effects of different selection rules, methods, and different levels of interleaving.

Execution time. Figure 8 shows that several members of the GOMS family can predict execution time, under the restrictions that the user must be well practiced and make no errors during the task. Some HCI specialists feel that these restrictions mean that GOMS models are not useful in actual design situations, because many users are novices or causal users, and errors are very common and time consuming. Errors will be discussed more below. However, we view the execution time predictions of GOMS models to be akin to EPA mileage ratings for cars. That is, although few drivers get as mileage as good as predicted by the EPA, the ratings are useful in predicting the direction and rough magnitude of differences between different makes of cars. Similarly, GOMS predictions help compare system design alternatives. If a GOMS model predicts a definite execution time difference between systems, say using the engineer's rule of thumb of more than 20%, a designer can be fairly certain that a real difference exists and is in the direction predicted by the GOMS model. Although GOMS models have been shown to be even more accurate in some cases (e.g., Gray, et al. 1993; Gong, 1993), we believe that further examination of when greater accuracy can be expected is necessary before we can make stronger recommendations for the general case. In many design situations, alternative systems do differ substantially (by more than 20%) and this level of accuracy is useful for weeding out undesirable alternatives.

Of course, if the system being designed will not be used enough by any single user to produce expert performance (e.g. a walk-up-and-use application to help home-buyers locate a house in a new city) GOMS predictions of execution time may not be useful compared to empirical results on other design issues such as the recognizability of menu terms to the first-time user. In addition, execution time is not an important variable in some systems, e.g., the success of a video game hinges more on the excitement it creates than on the speed at which a player can play the game. Likewise, the critical design target for educational software is the support it provides for learning, rather than the speed at which the user can operate it. On the other hand, interfaces for games and educational software often waste the user's time by methods that are slow and clumsy for no good reason. This mixture of the expected pattern of use, functionality, and usability concerns should be kept in mind. GOMS family models can contribute to designing software that is fast and easy to use, even if this is sometimes only a secondary design criterion.

Since sequential tasks have been the most studied in GOMS research, there are multiple techniques for predicting execution time in sequential tasks. The choice of technique depends primarily on whether the methods need to be explicitly represented for other purposes. If not, the KLM is by far the easiest technique. If explicit methods are needed for other reasons, e.g., to evaluate learning time or to design documentation, then using separate KLMs for execution time is inefficient; CMN-GOMS or NGOMSL models will provide both execution time and other types of information. For both execution time and operator sequence prediction, an important practical concern is whether the number of benchmark tasks is small. If so, then the predictions of operator sequence and execution time can be obtained by hand-simulation of the models, or simply by manually listing the operators, as for the KLM. But if the number of benchmark tasks is large, then it is probably worth creating machine-executable versions of an NGOMSL model, which can be done in a variety of ways in almost any programming language. This should become simpler in the future, as computer-based tools for GOMS models become available (e.g. Wood, 1993; Byrne, Wood, Sukaviriya, Foley, & Kieras, 1994).

Learning time. Information about learning time is provided only by NGOMSL models, and these predictions cover only the time to learn the methods in the GOMS model and any LTM information they require. These predictions have been validated in a variety of situations, and so merit serious consideration. But as mentioned above, there are clearly other aspects of a system that the user must learn, and other mechanisms involved in learning, besides those represented in the NGOMSL predictions. For example, teaching a user an appropriate mental model of a device can improve learnability and inference during subsequent use of the device (Kieras & Bovair, 1984; Kieras, 1988b), so designers may want to communicate such a model to their users. The class of models and methodologies presented here do not represent the knowledge and mechanisms required for using a mental model, and so have no basis for predicting the utility of a mental model or the time required to learn it (see Kieras, 1988b, 1990 for more discussion). Likewise, the concepts and principles discussed above as research directions (section 3.4) go beyond the simple procedure-learning situations captured by CCT and NGOMSL.

For practical situations, the recommendation is that NGOMSL learning time predictions should be used with caution, and preferably only in comparing two designs. Such comparisons should be fairly robust, since as noted above, a more complex interface should be harder to learn than a simpler one in a variety of possible learning situations. The analyst should keep in mind two important learning time issues. First, the time to learn the interface procedures may be insignificant in total training time for systems whose users must acquire substantial domain knowledge, such as a CAD/CAM system or a fighter aircraft weapons control system. Such domain knowledge may involve learning words or icons in the interface, or operators (e.g. BANK-AIRCRAFT) assumed in the analysis that the user must learn before they can execute the methods. Second, the predicted procedure learning time could be quite misleading for "walk up and use" systems or other "self evident" systems for which little or no explicit training is supposed to be required. To make the point clear, a method involving only a single step of typing an unlabelled control key would yield a very low predicted learning time, but the user may have no easy way to learn the correct keystroke in the actual usage situation. An example of just this problem is found in Karat, Boyes, Weisgerber, and Schafer (1986) who explored transfer of training between word processors (usually well predicted by CCT) and found that some experienced users of one word processor were completely stymied in trying to learn a new word processor because they could not figure out how to scroll the screen!

Error recovery support. The relationship of GOMS analyses to human error behavior deserves some discussion because of a common misunderstanding. CMN, and almost all subsequent GOMS work, presents analyses and predictions based on the assumption that the user does not make errors. Since errors in computer usage are quite frequent, it would seem that GOMS family models have little to say about actual human performance. But we would argue, along the lines of the above analogy with EPA mileage estimates, that GOMS models of error-free performance do in fact provide useful design information. For example, a poorly designed system that is difficult to learn and to use even under a no-errors assumption is almost certainly still a poor design if the user does make errors. So, optimizing learning time and execution time under the no-error assumption should result in a system that is a good design overall, given that errors do not always occur, and assuming that some reasonable error recovery is possible.

To further clarify, there are three design issues involved with errors: (1) preventing users from making errors; (2) predicting or anticipating when and what errors are likely to occur given a system design;and (3) designing the system to help the user recover from errors once they have occurred. Despite the obvious importance of the first two issues, at this time research on human errors is still far from providing more than the familiar rough guidelines concerning the prevention of user error. No prediction methodology, regardless of the theoretical approach, has yet been developed and recognized as satisfactory, and even the theoretical analysis of human error is still in its infancy (see Reason, 1990 for more discussion). At this time, GOMS models also fail to address error prediction or prevention.

However, as originally pointed out by CMN, GOMS has a direct application to the problem of error recovery. Once an error has occurred, the design question is whether the system provides a good method for the user to follow in recovering from the error. That is, is there a fast, simple, consistent method for the goal *RECOVER-FROM-ERROR*? (e.g., an ubiquitous undo command). Such a design question is no different in substance from designing the methods for the ordinary user goals. Figure 8 shows that any of the GOMS models can be used to address this question, with the specific choice depending on the specific aspect of interest, such as the time to execute the recovery procedure. Thus, once the possible frequent or important errors are determined, evaluating designs for the quality of support for error recovery can be done with ordinary GOMS approaches.

4.3. Time and Effort for Learning and Using GOMS in Design.

Using any method in design has both the cost of learning how to use the method, and also the time and effort to apply it to a specific design situation. Because of the large amount of detailed description involved, GOMS methodology has often been viewed as extremely time- and labor-intensive. This impression of the difficulty of GOMS methodology is probably a residue of the research effort involved to develop the techniques in their original form, and does not reflect the effort required to learn and to apply an already developed technique. In fact, there is now enough accumulated experience to assess the actual costs; it is clear that the GOMS methodologies shown in Figure 8 have an excellent return on investment, and the amount of the investment is much less than commonly believed. Some of the case studies in Section 5 demonstrate this effectiveness.

Based on our experience with teaching GOMS techniques to university and industrial students, the difficulty of learning a GOMS methodology depends mostly upon the complexity of the assumed underlying cognitive architecture. The KLM is based only on a simple serial stage framework, and so is conceptually the simplest. It is our experience, and the experience of other HCI instructors at several universities, that the KLM can be taught to undergraduates in a single class session with a few homework assignments and these students can construct models that produce execution time predictions accurate enough for design decisions (see Neilsen & Phillips, 1993; John, 1994). Likewise, the simple original form of GOMS, CMN-GOMS, is based on only the serial stage conceptual cognitive framework and so is relatively easy to understand and construct. A single class session seems to suffice if the student already has the basic skill of task decomposition and so can develop goal hierarchies for tasks.

Both NGOMSL and CPM-GOMS are based on more complex architectures, and are thus harder to learn and to use, but for different reasons. The difficulty in using NGOMSL models is a result of working in terms of the underlying CCT cognitive architecture, which requires working out complete and accurate methods, with deliberate goal and working memory manipulation, and a higher degree of formality and precision than CMN-GOMS. NGOMSL can be taught in a few undergraduate class sessions based on Kieras's "how-to" description (1988a, 1994a) and a couple of homework assignments with feedback. Full-day tutorials, such as at the CHI conferences (Kieras, 1994b) and in industrial short-courses appear to be adequate to get software developers started in the technique. There seems to be little point in using CCT itself, since the technical skills and facilities for production-rule modeling are much more demanding compared to using NGOMSL, which produces practically the same results.

An estimate of the overall effort in applying NGOMSL to an actual design problem is provided by the case study conducted by Gong (1993; see also Gong & Kieras, 1994) which is summarized in Section 5. In brief, he found that using NGOMSL in an application development situation to evaluate a design and compare a revision to the original required only about a fifth of the time spent on interface coding, and only about half of the time of an empirical evaluation comprising an informal survey and a single full-scale empirical comparison of the two designs.

The difficulty in using CPM-GOMS models is due to the inherent difficulty of identifying and describing in detail how perceptual, cognitive, and motor processing activities are coordinated in time. John and Gray (1992; 1994) have built a series of PERT-chart templates for a dozen or so common situations (e.g. perceiving visual information, typing, holding a conversation, etc.) and present these as building-blocks to combine into models of complex tasks. A few undergraduate class sessions is enough to allow students to manipulate existing CPM-GOMS models easily and correctly and give them a good start towards building their own models from scratch. Again full-day tutorials appear to present this material to the satisfaction of the tutorial participants (John & Gray, 1992; 1994).

In contrast to the GOMS techniques in Figure 8, the research approaches mentioned in sections 3.2 and 3.4 (ACT-R, Soar, Construction-Integration and EPIC) do not yet have articulated task analysis procedures, and so can be applied by the practitioner at this time only by emulating the ongoing research. This means that the practitioner would need to have a full set of research skills and resources to use the approach, and must be willing to accept the tentative nature of any models or results that will be produced. In order to become proficient enough to build models on their own, the learner must have an appropriate background in psychology or computer science and must undergo an apprenticeship with the experienced users of the technique that may stretch from a few weeks to a few months.

4.4. Effort Required for Answering Design Questions.

The effort required to use the different GOMS techniques to answer design and evaluation questions depends on the types of questions as well as on the techniques. Often there is a substantial start-up cost, but once the first models are built, subsequent questions require much less effort to answer. In addition, rather than fully analyzing an entire system interface, a properly selected subset of the interface can be isolated for detailed analysis, meaning that useful results can be obtained from quite modest modeling efforts. Some of the cases in Section 5 demonstrate this approach. Thus, the decision to invest effort in GOMS modeling depends on how many design issues and iterations will be involved. Here we will examine a few of the common uses of GOMS models and the corresponding required effort.

Focus for design effort. A critical practical design question is where the design effort should be focused. For example, should user procedures be streamlined to decrease the human execution time, or is it more important to improve the underlying software algorithms to decrease the response time of the system? To answer these types of questions, GOMS methods that produce quantitative predictions of system performance can be used to *profile* the overall execution times of the human-computer system to determine which portions of the interaction are taking significant or excessive amounts of time, and then priorities can be assigned to design issues in a rational manner. Some of the examples of actual GOMS design projects described in the next section used GOMS techniques in just this way. For example, CPM-GOMS was used to demonstrate that refining the screen design and keyboard layout would have relatively little effect on system performance compared to speeding up the response latency (Gray, John & Atwood, 1993). Because system profiling requires quantitative predictions of performance, it is one of the more time-consuming uses of GOMS. However, the payoff can be substantial because the analysis can be done early in the design process, at little cost compared to empirical testing, and thus can prevent resources from being poured into design efforts that have relatively little value.

Comparing alternative designs. Comparing alternative designs is the most obvious use of GOMS techniques. Since GOMS analyses do not require a running system, but can make *a priori* predictions of performance, they can be used early in the design process to evaluate different ideas before they are implemented or even prototyped. At the other extreme, existing alternative systems can be evaluated without installing them in a user organization, as will be illustrated by some of the case studies presented in the next section.

The effort involved in making comparisons between alternative systems depends on the kind of information required. Do you need only to know if important functionality is covered by both systems? Is expert execution time an important issue for the long-term use of the system? Does high turnover of personnel make training time of great importance? Answering the first question requires a rather shallow CMN-GOMS analysis, whereas the second question may require in-depth CPM-GOMS analysis, and the third question requires a full-blown NGOMSL model. Notice that models created to compare alternative designs can overlap with models created for other purposes. For example, if the design process uses GOMS to focus the design effort, the same model can be used as a basis to profile a design to identify problems, suggest solutions, and compare alternative solutions.

An additional determinant of the effort required is how many alternatives will need to be evaluated and how similar they will be. It is our experience that once a first model is constructed, it can serve as a base for similar designs, which then require only small modifications to the base model. So the effort put into modeling the initial system can be amortized over the number of alternatives evaluated. For instance, the CPM-GOMS models developed for the existing NYNEX workstation took about two staff-months, but once they were created, the potential benefits of new features could be evaluated literally in minutes (Gray, et. al. 1993).

Sensitivity and parametric analyses. In many design situations, the value of design ideas depends on assumptions about characteristics of the task domain or the users of the system. Common techniques in engineering design are to examine such dependencies with sensitivity analysis (how sensitive the predictions are to the assumptions) and parametric analysis (how the predictions vary as a function of some parameters). Again, because GOMS family members can make quantitative predictions of performance, they can be used to do such analyses. Examples can be found in the first descriptions of the KLM (Card, Moran & Newell 1980a, CMN, Ch. 8). In addition to profiling with predicted measures, such analyses also help guide empirical data collection by identifying the most sensitive issues, ensuring that the most valuable data is obtained given limited time and resources. The effort involved can be minimal if the assumptions and parameters are amenable to simple models like the KLM, and clearly more substantial if a CPM-GOMS or NGOMSL model is required. However, since these analyses typically vary only a few assumptions or parameters, they usually require only baseline models for a set of benchmark tasks and minimal manipulation of those models to discover the desired relationships. So once a base GOMS model has been constructed, exploring the sensitivity of the analysis and the effects of different parameters is inexpensive and fast.

Documentation and on-line help systems. Documentation and on-line help systems pose design questions that are very well addressed by GOMS methodology. Users normally know what goal they want to accomplish, but must turn to documentation or help because they do not know a method for accomplishing the goal, and cannot deduce one by experimenting with the system. However, most documentation and help provides only very low-level methods, at the level of command or option specification, as if the user's goal was *USE-THE-BELL-OPTION* in ftp, rather than a user-task-level goal such as *TRANSFER-FILES*. Consequently, typical documentation and help supports the rare user who already has most of the required method knowledge and needs only a few details.

In contrast, help and documentation can explicitly present the methods and selection rules users need in order to accomplish their goals. The list of user goals provides a specification of the document organization and entries for the index and table of contents. Experiments done by Elkerton and co-workers (Elkerton & Palmiter, 1991; Gong & Elkerton, 1990) using NGOMSL shows that this approach works extremely well, with results much better than typical commercial documentation and help. Thus, while it is standard advice that documentation and help should be "task oriented", it has not been very clear how one ensures that it is; GOMS provides a systematic, theory-based, and empirically-validated approach to determining the required content of procedural documentation and help.

A related application of GOMS is determining which alternative methods are the most efficient, and so should be presented in training and documentation. For example, in telephone operator call-handling, CPM-GOMS could predict execution time differences between different methods; identifying these differences would suggest the most efficient methods and selection rules to include in documentation and training materials.

5. Examples of Actual Applications of GOMS Family Members

Some of the following brief examples of the use of GOMS models have appeared as research papers because it was possible for the developers to write up their work. However, others arose in more typical development processes of real products, and are documented here through interviews with the developers.

Case 1. Mouse-driven text editor (KLM)

The first known use of the KLM for real system design was, not surprisingly, at Xerox (Card & Moran, 1988). In the early 1980s, when designing the text editor for the Xerox Star, the design team suggested several schemes for selecting text. These different schemes called for different numbers of buttons on the mouse. The goals were to use as few buttons on the mouse as possible so it would be easy to learn, while providing efficient procedures for experts. It was relatively easy to run experiments with the different schemes to test learnability for novices; everyone is a novice on a newly created system. However, it would have been substantially more difficult to run experiments with experts, because there were no experts. Experts would have to be "created" through extensive training, a prohibitive procedure both in time, workstation availability, and money. The design team therefore used a combination of experimental results on novices and KLMs of the same tasks to explore tradeoffs between learnability and expert performance. These models contributed directly to the design of the mouse for the Xerox Star.

Case 2. CAD system for mechanical design (KLM)

(Based on Monkiewicz, 1992 and an interview with Brenda Discher of Applicon, Inc.) Applicon, a leading vendor of CAD/CAM software for mechanical design, ported its BRAVO CAD package from a dedicated graphics terminal implementation into a Macintosh environment during the 1980s, but began to receive reports that the new implementation was actually slower and clumsier to use than the previous dedicated graphics terminal version. Applicon's interface design group used extensive KLM analyses to identify the source of the problems. For example, the analysis identified a major problem in the menu paradigm. In the original environment, the menu selections remained on the screen ("marching menus"), permitting multiple low-level selections without repeating the higher-level selections. The new implementation used the same menu organization, but followed the Macintosh rules that required menus to disappear once the lowest level selection was made. The resulting need to repeat the higher order selections greatly increased the task execution times. Candidate redesigns (e.g. using a dialog box) were evaluated with the KLM. Other aspects of the interface were refined with the KLM, such as reducing the depth of menu commands to only two levels to increase working speed without eliminating the many functions and options required for CAD tasks. The new design was also implemented for UNIX and VMS platforms, and this BRAVO 4.0 system is currently a successful and widely used suite of CAD applications. The quantification of execution time provided by the KLM was valuable to Applicon both internally to help justify and focus interface design efforts and set priorities, and also externally to help support competitive claims.

Case 3. Directory assistance workstation (KLM)

(Based on an interview with Judith R. Olson, University of Michigan) In 1982, some members of a human factors group at Bell Laboratories (Judith Olson, Jim Sorce, and Carla Springer) examined the task of the directory assistance telephone operators using the KLM. Directory assistance operators (DAOs) use on-line databases of telephone numbers to look up numbers for customers. The common wisdom guiding procedures for DAOs at that time in the Bell System was "key less - see more." That is, DAOs were instructed to type very few letters for the database search query (typically the first three letters of the last name and occasionally the first letter of the first name or the first letter of the street address) so that the database search would return many possible answers to the query. It was felt that it was more efficient for the DAO to visually search for the answer to the customer's request on a screen full of names than to type a longer, more restrictive, query that would produce fewer names on the screen.

The group analyzed the task and found two inefficiencies in the recommended procedures. First, the searches required unacceptably long times when the keyed-in query brought up multiple pages of names. Second, they found an unacceptably high rate of misses in the visual search. That is, the information that the customer wanted was actually on the screen, but the DAOs, trying to perform very quickly, often failed to see it in the midst of all the irrelevant information.

To arrive at a better design, the group analyzed the make-up of the database and categorized which queries would be common or rare, and whether the standard procedures would yield many names or relatively few. Based on this analysis, a set of benchmark queries was selected, and a parameterized KLM was constructed that clarified the tradeoff between query size and the number of retrieved names. The resulting recommendation was that DAOs should type many more keystrokes than had been previously thought, to restrict the search much more. This report was submitted at about the time of the breakup of the Bell System and the direct results of this particular report are impossible to track. However, current DAO training for NYNEX employees no longer advocates "key less - see more." Instead, DAOs are taught to key as much as needed to get the number of responses down to less than one screen's worth and to add more letters and redo the search rather than visually search through more than one screen. Currently, about 40% of NYNEX DAO searches result in only one answer returning from the search (Wayne Gray, personal communication).

Case 4. Bank deposit reconciliation system (KLM)

(Based on an interview with Judith R. Olson, University of Michigan) Olson, acting as a consultant to a software vendor, used the KLM in 1985 to redesign the interface for a system to be used by banks for deposit reconciliation. The system would allow a bank employee to compare the teller's keyed-in transaction information with scanned-in images of a customer's deposited checks and handwritten deposit slip in order to look for discrepancies such as keying errors by the teller, or duplications by the customer. Any discrepancy would be need to be detected, resolved, and then reported to the customer or to another department that would make the correction. The focus of the design work was on the layout of the display, to ensure that the required information could be rapidly obtained and compared.

Olson's analysis assumed that the system operators would search for the possible discrepancies in order of decreasing frequency, which was assumed to be known to the operators. This specified which items would be examined in which order, thus determining the basic task method. Olson developed KLM operator times for display activities such as visual scanning, matching handwritten digits to computer display digits, and then constructed KLMs for different types of discrepancies and display designs, and was able to rapidly evaluate different configurations and layouts for the display to arrive at an optimal design. Unfortunately, in the end, the software vendor did not adopt Olson's proposed redesign for reasons not involving the execution time of the task.

Case 5. Space operations database system (KLM)

(Overmeyer, personal communication). In 1983, the KLM was used in the design of a large command and control system for space operations. The system was to be used to monitor and maintain a catalog of existing orbital objects and debris. A new version of the system to replace the existing text-based database system was intended to have a graphical user interface. The software design of the new system was to be analyzed using simulation techniques to determine whether the system architecture and algorithms would provide adequate performance before the system was implemented. In order to quickly construct this simulation of the complete system, the KLM was used to represent the human operator's time with a preliminary interface design. With a couple of person-months work, about 50 benchmark tasks were selected that represented the basic interaction techniques, such a obtaining information about an orbiting object by using a joystick to select it and open an information window about it. With an additional person-month of work, KLMs were constructed for the preliminary design to give the execution time for each of the benchmark tasks. The system simulation was then run, and the software architecture modified to produce the required performance.

The new interface was eventually prototyped and used in experiments to get actual human performance data for later simulations, and to obtain data on tasks that involved processes such as complex decision-making that were beyond the scope of the KLM. The empirical results showed that the earlier estimates provided by the KLMs were reasonably accurate.

Thus the availability of usefully accurate estimates of user execution time early during the design process was critical in allowing the overall system performance to be assessed using simulation. The system was built, installed and in operation in the late 1980s and a descendant of the original system is still in operation today.

Case 6. Television control system (NGOMSL)

Elkerton (1993) summarizes a design problem involving designing an on-screen menu interface for a high-end consumer electronics television set. In the current technology of such systems, the television set is actually under computer control, and the user must perform setup and adjustment tasks by navigating a menu structure and selecting options for setting and adjustments. With some of the more complex consumer electronics products now available, the resulting interface can be fairly complex, and has considerable potential for being misdesigned. Needless to say, ease of learning and use are both extremely important in such a product.

According to Elkerton, currently most on-screen menu interfaces for complex televisions have obscure menu labels, deep menus for frequently performed tasks, and an arbitrary organization based on the product features rather than the user's tasks. In the product development situation described by Elkerton, there was not adequate time for extensive user testing and iteration of prototypes, and so an NGOMSL analysis was applied in an effort to help arrive at an improved interface quickly. The actual candidate designs were generated in the usual ways, and then analyzed with GOMS.

An early result of the NGOMSL task analysis was determining that there was a key distinction between the infrequent but critical tasks required to set up the television (e.g. configure it for a cable system), and the occasional tasks of adjusting the set during viewing (e.g. changing the brightness), and the major "task", that of actually watching the programming on a TV or VCR. The actual starting point for the NGOMSL analysis was an initial proposal for an improved interface design whose main virtue was simplicity, in which a single function key would cycle through each of the possible control functions of the set, resulting in very simple navigation methods and on-screen displays. This design preserved the setup/adjustment distinction, and was confirmed by some user testing as superior to the original interface. However, the NGOMSL analysis also verified that using the interface was quite slow, thus interfering with the viewing task.

The response was another proposed interface, following a more conventional menu structure, which the analysis showed, and user testing confirmed, interfered less with the user's main task. However, the NGOMSL analysis showed that the new prototype had inconsistent methods for navigating the menu structure; the setup and adjustment methods were different, which would lead to increased learning times and user frustration, and there were inconsistent methods for moving from one low-level function to another. Correcting these problems identified by the NGOMSL analysis produced a simpler, easier-to-learn interface. A interface based on some of these analyses and revised designs appeared in a television product line and is being considered for wider adoption by the manufacturer.

Case 7. Nuclear power plant operator's associate (NGOMSL)

Following a brief NGOMSL training workshop, Endestad and Meyer (1993) performed a fairly complete analysis of the interface for an experimental prototype of an intelligent associate for nuclear power plant operators. The system combined the outputs of several separate expert systems and other operator support systems, thus providing an integrated surveillance function. The total prototype system involved multiple networked computers, each with their own display monitors, and included a full simulation of a nuclear power plant. The basic concept of the system was that the information provided by the separate expert and support systems would be integrated by a single coordinating agent which would be responsible for informing the operator of an alarm event, making a recommendation, and referring the operator to the appropriate subsystem for supporting detail. NGOMSL methodology was a good choice to apply because of the difficulty of performing usability tests with highly trained users of limited availability and with such a complex system. For example, only a few types of emergency scenarios were fully supported in the prototype system.

The top-level NGOMSL method was written to show the basic structure of the task with and without the associate, thus clarifying the relative roles of the human operator and the system. The top-level method for the conventional situation, in which the operator's associate system was not present, simply had the operator working on his or her own in dealing with the various possible alarms and interacting with the separate expert and support systems. Thus the operator was required to engage in fairly elaborate reasoning, information searching, and ill-structured problem-solving right from the beginning of an alarm event. In contrast, the associate system would present an alarm event, a reference to the relevant subsystem, and a recommendation for action. It then requested the human operator to explicitly state agreement or disagreement with the recommendation. But any subsequent interactions concerning the alarm event were strictly at the operator's initiative; the operator was free to ignore the alarm, disregard the recommendation, or deal with it on their own. Thus the system potentially considerably simplified the initial reasoning and problem-solving required to handle the alarm event, and did not complicate the operator's task significantly. Of course, whether the associate was accurate enough to be trusted, or whether operators would come to rely on it unduly, could not addressed by a GOMS analysis.

At lower levels of the interaction, the NGOMSL model identified some specific problems and suggested solutions. For example, the operator designated which alarm event should be displayed using a calculator-like palette of buttons to enter in the number, but the required method was clearly more convoluted than necessary. Another example is that the lack of certain information on many of the displays resulted in methods that required excess looking from one display to another, in same cases requiring large physical movements. A final example is that a newer design for a support system that provided on-line operating procedures was predicted to be faster than a previous design, but could be further improved by more generic methods.

Case 8. Intelligent tutoring system (NGOMSL)

Steinberg & Gitomer (1993) describe how an NGOMSL analysis was used to revise the interface for an intelligent tutoring system. The tutoring system concerned training Air Force maintenance personnel in troubleshooting aircraft hydraulic systems such as the flight control systems. The basic content and structure of the tutoring was based on a cognitive analysis of the task domain and troubleshooting skills required. The tutoring system provided a full multimedia environment in which the trainee could "move" around the aircraft by selecting areas of the aircraft, manipulate cockpit controls, observe external components in motion, and open inspection panels and examine internal components.

The user's basic method for troubleshooting was to think of a troubleshooting operation and carry it out. The original interface assumed that the troubleshooting operations were local in the sense the user would think of a single component to observe or manipulate, carry out this action, and then would think of another component-action combination to perform. However, the NGOMSL analysis showed that many troubleshooting activities had a larger scope spanning several components or locations on the aircraft. A typical activity was an input-output test, in which inputs would be supplied to one set of components, and then several other components, often in an entirely different location, would be observed. For example, the troubleshooter would enter the cockpit, set several switches, and then start moving the control stick, and then observe the rudders to see if they moved.

In the original interface, there was no support for such multiple-component input-output tests, and so the user had to traverse the component hierarchy of the simulated aircraft several times and perform the component actions or observations individually. The revised interface suggested by the NGOMSL analysis allowed the user to easily view and act on multiple parts of the aircraft, with rapid access to and from the aircraft cockpit. This reflected the basic structure of the troubleshooter's task in a more realistic fashion, as well as making it faster and simpler to carry out the testing activity in the context of the tutoring system.

Case 9. Industrial scheduling system (NGOMSL)

Nesbitt (in preparation) reports a use of an NGOMSL model to deal with a common situation in which an existing interface is to be extended. The system is question is a partly automated scheduling system for managing equipment maintenance activities in a steel-making plant. Since shutting down equipment in steel plants can have serious effects on production scheduling, accurate management of downtimes is critical. The original version of the system included automatically generated downtimes, and a interface for viewing the downtimes. The required extension was to allow users to enter downtimes directly into the scheduling system.

The steel plant has a natural hierarchical structure, about five levels deep, in which either a terminal or non-terminal location (a set of machines) could be shut down for maintenance. If a non-terminal location of the plant is shut down for a downtime period, then all sublocations are also deemed to be shut down. Given the natural hierarchical structure, the choice for the display of downtime information was based on a combination of plant hierarchy and date, in which the user viewed a grid showing locations as rows and days as columns, with each cell containing the number of schedule downtime hours. By selecting a row, the user can move down a hierarchical level to view the downtimes broken out into more detail for the sublocations. By clicking on a cell, the user can view a list of the individual downtime schedule items comprising the listed total hours.

A new set of requirements was to allow the user to enter new downtimes or modify existing ones, under the assumption that the list of downtimes was unordered. The first solution was to simply add a dialog box to the grid-based display interface, so that once the relevant location and date had been selected by traversing the hierarchy, the user could simply specify the downtime start time, duration and other information, for that location on that date. The implementation effort would be minimal. However, the GOMS analysis showed that the resulting interface would be extremely inefficient. Adding a new downtime requires first traversing the hierarchy to the affected location and date, while modifying the location or downtime date requires deletion and reentry. A side effect is that there is no method to allow the user to create a new downtime entry by simply selecting and modifying an existing downtime, since the dates and location of a downtime were unmodifiable.

A redesigned interface alleviated these problems. The solution was a form-based screen in which the user could specify all of the downtime attributes by selecting from context-sensitive lists or by editing the attribute fields. All of the downtime attributes, such as all five location levels, were simultaneously displayed. While selecting a location still required traversing the plant structure hierarchy, only the locations were involved, not the date, so the selection consisted of simply filling in a set of fields using selection from lists whose contents were determined by the higher-level selection. In addition, a new downtime entry could be created by selecting an existing entry and then modifying its fields as needed.

The GOMS analysis made the difficulties of the original interface clear, and over a set of actual downtime scheduling tasks predicted that the redesigned interface would require overall only half the execution time as the original, with a substantial improvement for modifying existing schedule times.

Case 10. CAD system for ergonomic design (NGOMSL)

Gong (1993, see also Gong & Kieras, 1994) provides a detailed case study of the application of GOMS methodology to an actual software product. The program was a CAD system for the ergonomic analysis of workplace design, with an emphasis on biomechanical analyses to detect problems of occupational safety in situations such as assembly line jobs requiring handling of awkward or heavy objects. The user, typically an industrial engineer, would describe a work situation by specifying the user's physical posture while carrying out a step in the job and other parameters such as the weight of a lifted object, and the program would generate information on stress factors, such as the likelihood of lower back injury. This program was being sold on a commercial basis in a PC DOS version; Gong's task was to develop a Macintosh version of the program for commercial distribution. Applying GOMS analysis to refine the design was suggested by the fact that too few domain experts were available to serve as subjects in a formal conventional user test, and an attempt to collect informal feedback produced mostly information about functionality or user expectations rather than ease of use.

Gong constructed a GOMS model of the initial version of the software, which adhered to the Macintosh interface guidelines, and then examined the model for problems. An example of such a problem was that the interface assumed a default method for specifying posture that users would probably always override in favor of a far simpler and easier method. Another example is that the methods had many RETRIEVE-FROM-LTM operators; the user had to memorize many associations between commands and the menu names that they appeared under. A final example is that certain methods involved time-consuming interaction with "modal" dialogs, which are dialog boxes that have to be explicitly dismissed before the user can continue. Gong (1993) lists many such specific identified problems and addressed them in specific interface design solutions. The revised interface was predicted to be about 46% faster to learn and also about 40% faster to use than the original interface. A subsequent empirical test confirmed these predictions.

Gong reported that the time spent developing and working with the GOMS model was only about 15 days, compared to about 80 days spent on software development and programming, and 34 days spent on both the informal user feedback collection and the formal evaluation study.

Thus the NGOMSL methodology was usefully accurate in its predictions, helped identify specific usability problems, and provided a basis for design solutions. In addition, despite the widespread opinion that GOMS analysis is too time-consuming to be practical, the actual effort required was quite reasonable, especially given that a single design iteration using the methodology produced a substantial improvement in learning and execution time.

Case 11. Telephone operator workstation (CPM-GOMS)

The details of this application of GOMS, both technical and managerial, can be found in Gray, John and Atwood, 1993, and Atwood, Gray and John, in press). In 1988, the telephone company serving New York and New England (NYNEX) considered replacing the workstations then used by toll and assistance operators (TAOs), who handle calls such as collect calls, and person-to-person calls, with a new workstation claimed to be superior by the manufacturer. A major factor in making the purchase decision was how quickly the expected decrease in average work time per call would offset the capital cost of making the purchase. Since an average decrease of one second in work time per call would save an estimated $3 million per year, the decision was economically significant.

To evaluate the new workstations, NYNEX conducted a large-scale field trial. At the same time, a research group at NYNEX worked with Bonnie John to use CPM-GOMS models in an effort to predict the outcome of the field trial. First, models were constructed for the current workstation for a set of benchmark tasks. They then modified these models to reflect the differences in design between the two workstations, which included different keyboard and screen layout, keying procedures, and system response time. This modeling effort took about two person-months, but this time included making extensions to the CPM-GOMS modeling technique to handle this type of task and teaching NYNEX personnel how to use CPM-GOMS. The models produced quantitative predictions of expert call-handling time for each benchmark task on both workstations, which when combined with the frequency of each call type, predicted that the new workstation would be an average of 0.63 seconds slower than the old workstation. Thus the new workstation would not save money, but would cost NYNEX about 2 million dollars a year.

This was a counter-intuitive prediction. The new workstation had many technically superior features. The workstation used more advanced technology to communicate with the switch at a much higher speed. The new keyboard placed the most frequently used keys closer together. The new display had a graphic user interface with recognizable icons instead of obscure alphanumeric codes. The procedures were streamlined, sometimes combining previously separate keystrokes into one keystroke, sometimes using defaults to eliminate keystrokes from most call types, with a net decrease of about one keystroke per call. Both the manufacturer and NYNEX believed that the new workstation would be substantially faster than the old one, by one estimate, as much as 4 seconds faster per call. Despite the intuition to the contrary, when the empirical field-trial data were analyzed, they supported the CPM-GOMS predictions. The new workstation was 0.65 seconds slower than the old workstation.

In addition to predicting the quantitative outcome of the field trail, the CPM-GOMS models explained *why* the new workstation was slower than the old workstation, something which empirical trials typically cannot do. The simple estimate that the new workstation would be faster was based on the greater speed of the new features considered in isolation. But the execution time for the whole task depends on how all of the components of the interaction fit together, and this is captured by the critical path in the CPM-GOMS model. Because of the structure of the whole task, the faster features of the new workstation failed to shorten the critical path.

Thus, examination of the critical paths revealed situations in which the new keyboard design slowed down the call, why the new screen design did not change the time of the call, why the new keying procedures with fewer keystrokes actually increased the time of some calls, and why the more advanced technology communication technology often slowed down a call. The complex interaction of all these features with the task of the TAO was captured and displayed by CPM-GOMS in a way that no other analysis technique or empirical trial had been able to accomplish.

NYNEX decided not to buy the new workstations. The initial investment in adopting the CPM-GOMS technique paid off both in this one purchase decision, and by allowing NYNEX to make some future design evaluations in as little as a few hours of analysis work.

6. Summary and Conclusions

The several specific GOMS modeling techniques are all related to a set of general concepts, both conceptual frameworks for human information processing, and a general approach to the analysis of tasks. This general approach emphasizes the importance of the procedures that a user must learn and follow to perform well with the system to accomplish goals. By using descriptions of user procedures, the techniques can provide quantitative predictions of procedure learning and execution time. While other aspects of system design are undoubtedly important, the ability of GOMS models to address this critical aspect makes them a key part of the scientific theory of human-computer interaction and also useful tools for practical design. The different GOMS techniques correspond to different set of assumptions about the underlying information-processing architecture and the simplifications required to economically address design issues. These assumptions are clear-cut enough for practical advice to be stated on which technique should be used for what purpose. Finally, when the effort required to use GOMS analysis is considered, both in terms of the logic of analysis and design, and also in terms of the application examples presented in this paper, it is clear that GOMS models can make a worthwhile contribution to developing more usable computer systems.

Acknowledgments

The authors contributed equally to this article; the order of their names reflects alphabetical order and not seniority of authorship. We thank Wayne Gray for his comments on early drafts of this paper.

References

Anderson, J. R. (1976). *Language, memory, and thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1987). Skill acquisition: compilation of weak-method problem solutions. *Psychological Review*, 94, 192-210.

Anderson, J. R. (1993). *Rules of the Mind.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Atwood, M. E., Gray, W. D., & John, B. E. (in press) Project Ernestine: Analytic and empirical methods applied to a real-world CHI problem. To appear in *Human-Computer Interface Design: Success Cases, Emerging Methods and Real-World Context*, L. Gugerty & P. Polson (eds.), San Mateo, CA: Morgan-Kaufmann.

Bovair, S., Kieras, D. E., & Polson, P. G. (1988). *The acquisition and performance of text-editing skill: A production–system analysis.* (Tech. Rep. No. 28). Ann Arbor: University of Michigan, Technical Communication Program.

Bennett, J. L., Lorch, D. J., Kieras, D. E., & Polson, P. G. (1987). Developing a user interface technology for use in industry. In Bullinger, H. J., & Shackel, B. (Eds.), *Proceedings of the Second IFIP Conference on Human-Computer Interaction, Human-Computer Interaction – INTERACT '87*. (Stuttgart, Federal Republic of Germany, Sept. 1–4). Elsevier Science Publishers B.V., North-Holland, 21-26.

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

Butler, K. A., Bennett, J., Polson, P., and Karat, J. (1989). Report on the workshop on analytical models: Predicting the complexity of human-computer interaction. *SIGCHI Bulletin*, 20(4), pp. 63-79.

Byrne, M.D., Wood, S.D, Sukaviriya, P., Foley, J.D, and Kieras, D.E. (1994). Automating Interface Evaluation. In *Proceedings of CHI*'94 (Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 232-237.

Card, S., & Moran, T. (1988). User technology: From pointing to pondering. In Goldberg, A. (Ed.) *A history of personal workstations*. New York, ACM. [pp. 489-522]

Card, S.K., Moran, T.P., & Newell, A. (1980a). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM* , 23(7), 396-410.

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

Carley, K., Kjaer-Hansen, J., Prietula, M., & Newell, A. (1990) "Plural-Soar: A Prolegomenon on artificial agents and organizational behavior" to appear in M. Masuch and G. Massimo (Eds.) *Distributed Intelligence: Applications in Human Organizations*, Elsevier.

Carley, K. M., & Prietula, M. J. (1994b). *Computational Organization Theory.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Chuah, M. C., John, B. E., & Pane, J. (1994). Analyzing graphic and textual layouts with GOMS: Results of a preliminary analysis. In *Proceedings Companion of CHI*, 1994, Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 323-324.

Diaper, D. (Ed.) (1989). *Task analysis for human-computer interaction.* Chicester, U.K.: Ellis Horwood.

Doane, S. M., Mannes, S. M., Kintsch W., and Polson, P. G.(1992). Modeling user action planning: A comprehension based approach. *User Modeling and User-Adapted Interaction* 2, 249-285

Doane, S. M., McNamara, D. S., Kintsch, W., Polson, P. G. and Clawson, D. M. (1992). Prompt comprehension in UNIX command production. *Memory and Cognition,* 20,4, 327-343.

Elkerton, J. (1993). *Using GOMS models to design documentation and user interfaces: An uneasy courtship.* Position paper for workshop on Human-Computer Interaction Advances Derived from Real World Experiences, INTERCHI'93 Conference, Amsterdam, April 24-25.

Elkerton, J., & Palmiter, S. L. (1991). Designing help using a GOMS model: An information retrieval evaluation. *Human Factors*, 33, 2, pp. 185-204.

Endestad, T., and Meyer, P. (1993). *GOMS analysis as an evaluation tool in process control: An evaluation of the ISACS-1 prototype and the COPMA system.* Technical Report HWR-349, OECD Halden Reactor Project, Instituut for Energiteknikk, Halden, Norway.

Gilbreth, F. B. & Gilbreth, L. M. *Applied Motion Study*. New York: The Macmillian Company, 1917.

Gong, R. & Elkerton, J. (1990). Designing minimal documentation using a GOMS model: A usability evaluation of an engineering approach. In *Proceedings of CHI*, 1990 (Seattle, Washington, April 30-May 4, 1990) ACM, New York, 99-106.

Gong, R., & Kieras, D. (1994). A Validation of the GOMS Model Methodology in the Development of a Specialized, Commercial Software Application. In *Proceedings of CHI*, 1994, Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 351-357.

Gong, R. (1993). *Validating and refining the GOMS model methodology for software user interface design and evaluation.* PhD dissertation, University of Michigan, 1993.

Gott, S. P., (1988). Apprenticeship instruction for real-world tasks: The coordination of procedures, mental models, and strategies. In Ernst Z. Rothkopf (Ed.), *Review of Research in Education.* Washington, D.C.: AERA.

Gray, W. D., John, B. E., & Atwood, M. E. (1992). The precis of Project Ernestine or an overview of a validation of GOMS. *Proceedings of CHI* (Monterey, May 3-7, 1992) ACM, New York.

Gray, W. D., John, B. E., & Atwood, M. E. (1993) "Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance." *Human-Computer Interaction*, **8**, 3, pp. 237-209.

Gray, W. D., & Sabnani, H. (1994). "Why you can't program your VCR, or, predicting errors and performance with production system models of display-based action" *Proceedings Companion of CHI '94* (Boston, April 24-28, 1994) ACM, New York. pp. 79-80.

Haunold, P. & Kuhn, W. (1994). A keystroke level analysis of a graphics application: Manual map digitizing. In *Proceedings of CHI*, 1994, Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 337-343.

Howes, A. (1994). A model of the acquisition of menu knowledge by exploration. In *Proceedings of CHI*, 1994, Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 445-451.

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

John, B. E. (1990) Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. In *Proceedings of CHI*, 1990 (Seattle, Washington, April 30-May 4, 1990) ACM, New York, 107-115.

John, B. E. (1994) Toward a deeper comparison of methods: A reaction to Nielsen & Phillips and new data. In the *Proceedings Companion of CHI* 1994 (Boston, April 24-28, 1994) ACM, New York. pp. 285-286.

John, B. E. (in press) Task matters. In D. M. Steier and T. Mitchell (Eds.), *Mind matters.* Hillsdale, NJ: Lawrence Erlbaum.

John, B. E. & Newell, A. (1989) Cumulating the science of HCI: From S-R compatibility to transcription typing. In *Proceedings of CHI*, 1989 (Austin, Texas, April 30-May 4, 1989) ACM, New York, 109-114.

John, B. E. & Vera, A. H. (1992). A GOMS analysis for a graphic, machine-paced, highly interactive task. In *Proceedings of CHI* (Monterey, May 3-7, 1992) ACM, New York. pp. 251-258.

John, B. E., Vera, A. H. & Newell, A. (1994) Toward real-time GOMS: A model of expert behavior in a highly interactive task. *Behavior and Information Technology*, vol 13, no. 4, pp. 255-267.

Karat, J., and Bennett, J. (1989). *Modeling the user interaction methods imposed by designs.* Technical report RC 14649. IBM T. J. Watson Research Center, Yorktown Hts., NY.

Karat, J., Boyes, L., Weisgerber, S., & Schafer, C. Transfer between word processing systems. In *Proceedings of CHI'86*, (Boston, April 13-17, 1986), New York: ACM, pp. 67-71.

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

Kieras, D.E. (1988b). What mental model should be taught: Choosing instructional content for complex engineered systems. In J. Psotka, L.D. Massey, and S. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned* (pp. 85-111). Hillsdale, NJ: Erlbaum.

Kieras, D.E. (1990). The role of cognitive simulation models in the development of advanced training and testing systems. In N. Frederiksen, R. Glaser, A. Lesgold, & M. Shafto (Eds.), *Diagnostic monitoring of skill and knowledge acquisition*. Hillsdale, NJ: Erlbaum.

Kieras, D. E. (1994a). *A guide to GOMS task analysis (Spring, 1994)*. Unpublished manuscript, University of Michigan.

Kieras, D. (1994b). *GOMS Modeling of User Interfaces using NGOMSL.* Tutorial Notes, CHI'94 Conference on Human Factors in Computer Systems, Boston, MA, April 24-28, 1994.

Kieras, D.E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science*, 8, 255-273.

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

Kieras, D.E., & Meyer, D.E. (1994). *The EPIC architecture for modeling human information-processing: A brief introduction.* (EPIC Tech. Rep. No. 1, TR-94/ONR-EPIC-1). Ann Arbor, University of Michigan, Department of Electrical Engineering and Computer Science.

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

Kintsch, W. (1988). The role of knowledge in discourse comprehension: a construction-integration model. *Psychological Review,* 95, 2, 163-182.

Kintsch, W. (1992). A cognitive architecture for comprehension. In H. L. Pick, Jr., P. van den Broek, and D. C. Knill, (Eds.), *Cognition: Conceptual and methodological issues.* Washington, D. C..: American Psychological Association, 143-164.

Kitajima, M., & Polson, P. G. (1992) A computational model of skilled use oif a graphical user interface. In *Proceedings of CHI* (Monterey, May 3-7, 1992) ACM, New York. pp. 241-249

Laird, J., Rosenbloom, P, & Newell, A. (1986) *Universal subgoaling and chunking*., Kluwer Academic Publishers: Boston.

Lane, D. M., Napier, H. A., Batsell, R. R., & Naman, J. L. (1993). Predicting the skilled use of hierarchical menus with the Keystroke-Level Model. *Human-Computer Interaction*, **8**, 2, pp. 185-192.

Lee, A. Y., Polson, P. G., & Bailey, W. A. (1989) Learning and transfer of measurement tasks. In *Proceedings of CHI*, 1989 (Austin, Texas, April 30-May 4, 1989) ACM, New York, 1989, pp.115-120.

Lerch, F. J., Mantei, M. M., & Olson, J. R. (1989). Translating ideas into action: Cognitive analysis of errors in spreadsheet formulas. in *Proceedings of CHI*, 1989, 121-126. New York: ACM.

Meyer, D.E., & Kieras, D.E. (1994). *EPIC computational models of psychological refractory-period effects in human multiple-task performance.* (EPIC Tech. Rep. No. 2, TR-94/ONR-EPIC-2). Ann Arbor, University of Michigan, Department of Psychology.

Monkiewicz, J. (1992). CAD's next-generation user interface. *Computer-Aided Engineering*, November, 1992, 55-56.

Nesbitt, Keith. (1994, unpublished). *A case study of GOMS analysis: Extension of user interfaces.* Unpublished report, BHP Research - Newcastle Laboratories, Wallsend, NSW, Australia.

Nelson, G. H., Lehman, J. F., & John. B. E. (1994) Integrating cognitive capabilities in a real-time task. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society* (Atlanta, Georgia, August 13-16, 1994).

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

Nielsen, J. and Phillips, V. L. (1993) Estimating the relative usability of two interfaces: Heuristic, formal, and empirical methods compared. In *Proceedings of INTERCHI*, 1993 Amsterdam, The Netherlands, pp. 214-221, New York: ACM.

Oberg, E., Jones, F. D., and Horton, H. L. (1978) *Machinery's Handbook: A reference book for the mechanical engineer, draftsman, toolmaker and machinist*. Twentieth edition, fourth printing, Industrial Press Inc: New York.

Olson, J. R., & Olson, G. M. (1990). The growth of cognitive modeling in human-computer interaction since GOMS. *Human-Computer Interaction*, 5, 221-265.

Peck, V. A. & John, B. E. (1992). Browser-Soar: A cognitive model of a highly interactive task. In *Proceedings of CHI*, 1992 (Monterey, California, May 3- May 7, 1992) ACM, New York, 1992. pp. 165-172.

Polson, P.G. (1988). Transfer and retention. In R. Guindon (Ed.), *Cognitive science and its application for human-computer interaction*. Hillsdale, N.J: Erlbaum. Pp. 59-162.

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

Reason, J. (1990). *Human Error*. Cambridge: Cambridge University Press.

Rieman, J., Lewis, C. Young, R. M., & Polson, P. G. "'Why is a raven like a writing desk?' Lessons in interface consistency and analogical reasoning from two cognitive architectures" In *Proceedings of CHI*, 1994, (Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 438-444.

Steinberg, L. S., & Gitomer, D. H. (1993). Cognitive task analysis, interface design, and technical troubleshooting. In W. D. Gray, W. E. Hefley, & D. Murray (Eds.),*Proceedings of the 1993 International Workshop on Intelligent User Interfaces*. New York: ACM. (pp. 185-191).

Stires, D. M. & Murphy, M. M. (1962). *PERT (Program Evaluation and Review Technique) CPM (Critical Path Method).* Boston: Materials Management Institute.

Webster's New Collegiate Dictionary. (1977). Springfield, MA.: G. & C. Merriam Company.

Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994) The Cognitive Walkthrough Method: A practitioner's guide. In Jakob Nielsen and Robert L. Mack (eds.) *Usability Inspection Methods*. New York: John Wiley & Sons, Inc.

Wiesmeyer, M. D. (1992). *An operator-based model of human covert visual attention.* PhD thesis, University of Michigan.

Wood, S. (1993). *Issues in the implementation of a GOMS-model design tool.* Unpublished report, University of Michigan.

Wood, S., Kieras, D., & Meyer, D. (1994). An EPIC model for a high performance HCI task. *CHI'94 Conference Companion*, ACM. (pp. 24-28).

Van Cott, H. P. & Kinkade, R. G. (1972) *Human engineering guide to equipment design (revised edition)*. Washington, D. C.: American Institutes for Research.

Ziegler, J. E., Hoppe, H. U., & Fahnrich, K. P. (1986). Learning and transfer for text and graphics editing with a direct manipulation interface. In *Proceeding of CHI*, 1986. New York: ACM.