Using GOMS for User Interface Design and Evaluation:
Which Technique?

Bonnie E. John
Human-Computer Interaction Institute
and Departments of Computer Science and Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
Phone: (412) 268-7182
Email: bonnie.john@cs.cmu.edu

&

David E. Kieras
Department of Electrical Engineering and Computer Science
University of Michigan
Advanced Technology Laboratory Building
1101 Beal Avenue
Ann Arbor, MI 48109-2110
Phone: (313) 763-6739
Email: kieras@eecs.umich.edu

10 June 1996

ABSTRACT

Since the seminal Card, Moran, & Newell (1983) book, *The psychology of human-computer interaction*, 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 work and has been used in real-world design and evaluation situations. This paper synthesizes the previous work on GOMS to provide an integrated view of GOMS models and how they can be used in design. We briefly describe the major variants of GOMS that have matured sufficiently to be used in actual design. We then provide guidance to practitioners about which GOMS variant to use for different design situations. Finally, we present examples of the application of GOMS to practical design problems and summarize the lessons learned.

1. INTRODUCTION

1.1. Purposes of this Paper

Since the seminal Card, Moran, & Newell (1983) book, *The Psychology of Human-Computer Interaction*, (hereafter, CMN) 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 goals of this paper are 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 without further empirical validation. 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.

This paper is *not* a tutorial in how to use any version of GOMS; that information is elsewhere in textbooks, handbooks and tutorial notes (Card, Moran & Newell, 1983; John & Gray, 1995; Kieras, 1988, in press-a). We expect this paper to be useful in deciding whether GOMS is appropriate for a design problem, and if so, which variant of GOMS is most appropriate. The details of how to actually conduct an analysis should then be obtained from these other sources. This paper is also not a comparison of the techniques. John & Kieras (in press) provides a detailed comparison of the four GOMS variants surveyed here, how they make their predictions, comparing and contrasting the underlying mechanisms.

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 and the GOMS family of techniques as instantiations of this engineering approach. The second section of the paper describes how GOMS can be applied in design. The third 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

1.2.1. *Why engineering models?*

The overall motivation for GOMS and other HCI cognitive modeling efforts is to provide *engineering models* of human performance. In the ideal, such models produce *a priori* quantitative predictions of performance at an earlier stage in the development process than prototyping and user testing. That is, they predict execution time, learning time, errors, and identify those parts of an interface that lead to these predictions, thereby focusing the designer on what to fix. They allow analysis at different levels of approximation so predictions appropriate to the design situation can be obtained with minimum effort. They are straight-forward enough for computer designers to use without extensive training in psychology, and these models are integrated enough to cover total tasks. Although HCI research has not yet reached this ideal, GOMS is currently the most mature of engineering models, has many of these properties as discussed below, and can be truly useful in real-world system development.

In psychology, researchers fit the parameters of their models to data they've collected on the task they're studying. But in interface design, system developers need quantitative *a priori* predictions for systems that have not yet been built. Thus, HCI researchers have done extensive theoretical and empirical work to estimate parameters that are robust and reliable across tasks and can be used *without further empirical validation* to make predictions. This does not mean that parameters have to be fixed constants for every situation, only that they must be determinable *a priori*. For GOMS models, tables of parameters covering a wide range of tasks have been created based on previous research (see Card, Moran & Newell, 1983; Gray, John & Atwood, 1993; Olson & Olson, 1990; Kieras, 1988; in press-a).

Many interactive systems today are developed without trained psychologists or human factors experts (Dillon, Sweeney, & Maguire, 1993). Thus, computer system developers cannot be expected to bring psychological expertise to the task, so the basic psychology must be "built into" the models. The tables of parameter values mentioned above is an example of incorporating basic psychological knowledge in a form that engineers can use it. 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. Handbook chapters and tutorials in GOMS address this issue (John & Gray, 1995; Kieras, 1988, 1994, in press-a).

Engineering models must address a useful range of design issues. The 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. Second, the complexity and efficiency of the interface procedures is addressed very well by current GOMS models. 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

all these activities, and how they are performed together, be considered for design. That is, optimizing the various aspects or portions of an interface in isolation (e.g., the physical layout of the keyboard or the visual layout of the screen) 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 the procedural aspects of a task exercise all parts of the system design.

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. GOMS models range from the "quick and dirty" Keystroke-Level Model, to more detailed accounts of knowledge and interleaving tasks. In Section 2, we will discuss which approximations are viable in different design situations.

Card, Moran, & Newell developed the concept of GOMS with these criteria for useful engineering models in mind. Thus, 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. As we will show, the evaluation results produced by GOMS engineering models are both quite useful and quite limited. Thus, rather than replacing user testing, the current engineering models are best viewed as *reducing* the amount of user testing required to develop a highly usable system. The iterative design process should then use engineering models, and other non-user testing techniques (e.g., see Nielsen & Mack, 1994), where applicable early in the design process, to evaluate candidate designs and resolve design issues as much as possible before investing in actual user testing. Such a multiple-technique approach will make the best use of available scientific and practical knowledge about human-computer interaction (see Olson & Moran, 1996, for a discussion of coordinated use of methods).

1.3. Overview of the GOMS Concept

1.3.1. *The general GOMS concept.*

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*). 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 the knowledge of how to do a task in terms of goals, operators, methods, and selection rules.*

The acronym *GOMS* thus stands for the components of a GOMS model, namely the **g**oals, **o**perators, **m**ethods and **s**election rules. Briefly, *goals* are simply the user's goals, as defined in layman's language. What does he or she want to accomplish by using the software? In the next day, the next few minutes, the next few seconds? Goals are often divided into subgoals: to accomplish the goal of writing a paper about Alfred Hitchcock, a student might set subgoals to find information on his life from an on-line encyclopedia, find the names of all his films from a CD-ROM about movies, outline the paper with a word processor, then fill in the outline. *Operators* are the actions that the software allows the user to take. With the original command-line interfaces, an operator was often a command and its parameters, typed on a keyboard. Today, with graphic user interfaces, operators are just as likely to be menu selections, button presses, or direct-manipulation actions. In the future, operators will be gestures, spoken commands, or even eye movements. Operators can actually be defined at many different levels of abstraction but most GOMS models define them at a concrete level, like button presses and menu selections. *Methods* are well-learned sequences of subgoals and operators that can accomplish a goal. If there is more than one method to accomplish the same goal, then selection rules, the last component of the GOMS model, are required. *Selection rules* are the personal rules that users follow in deciding what method to use in a particular circumstance. Together, goals, operators, methods and selection rules describe the procedural or "how to do it" knowledge a user requires to perform a task. A more complete discussion of the GOMS components of a task and how to identify them is included in John & Kieras (in press).

1.3.2. *The GOMS family of techniques.*

While the general GOMS concept is similar to many other task decomposition concepts (e.g., Diaper, 1989; Gilbreth & Gilbreth, 1917; Kirwan & Ainsworth, 1992; Newell & Simon, 1972; Van Cott & Kinkade, 1972, Kieras, in press-b), this concept has spawned a family of task analysis and modeling techniques, the GOMS family. There are four different versions of GOMS in use today, all based on the same GOMS concept:

- CMN-GOMS. The original formulation proposed by Card, Moran and Newell was a loosely defined demonstration of how to express a goal and subgoals in a hierarchy, methods and operators, and how to formulate selection rules.

- KLM. A simplified version CMN called the Keystroke-Level Model uses only keystroke-level operators, no goals, methods or selection rules. The analyst simply lists the keystrokes and mouse-movements a user must perform to accomplish a task and then uses a few simple heuristics to place "mental operators."

- NGOMSL. A more rigorously defined version called NGOMSL (Kieras, 1988; in press-a) presents a procedure for identifying all the GOMS components, expressed in a form similar to an ordinary computer programming language. NGOMSL includes rules-of-thumb about how many steps can be in a method, how goals are set and terminated, and what information needs to be remembered by the user while doing the task.

- CPM-GOMS. A parallel-activity version called CPM-GOMS (John, 1990) uses **c**ognitive, **p**erceptual and **m**otor operators in a **c**ritical **p**ath **m**ethod schedule chart (PERT chart) to show how activities can be performed in parallel.

All GOMS techniques produce quantitative and qualitative predictions of how people will use a proposed system, though the different versions have different emphases. All of the techniques can

speak to the coverage of the functionality of a system and all provide estimates of task performance time. CMN-GOMS and NGOMSL provide predictions of the sequence of operators people will employ in a specific task instance. NGOMSL provides predictions of learning time, the benefits of consistency, and some indication of the likelihood of some errors. A detailed account of how and why these techniques produce these predictions can be found in John & Kieras (in press).

2.   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 this section of this paper, we provide the information required for a designer to choose one of the four existing GOMS techniques: which techniques are suitable for which design situations, what are the benefits of using each technique, and an estimate of the effort involved in using the technique. In the subsequent section, we present examples of how the different techniques have been used in actual system design and evaluation.

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 1 lists which GOMS family methods can be used for each combination of type of task and type of information. 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 predictive user-modeling techniques. In some cases, existing GOMS techniques could be modified and adapted for these cases, but the figure presents the techniques as currently documented.

2.1.   Characterizing the User's Tasks

Although user tasks can be characterized in many different ways, four dimensions are important for deciding whether a GOMS analysis technique is applicable to the user's task, and which technique is most suitable: (1) the degree of goal-directedness in the user's purposes for using the system, (2) the degree of routinized skill involved in the user's task, (3) the degree to which the interaction is under the control of the user versus the computer system or other agents involved in the task, and (4) the sequentiality of the user's task.

Design Information \ Task Type	Goal-Directed, Routine Cognitive Skill with Passive or Active Systems	
	Sequential	Parallel
Functionality: Coverage	Any GOMS	Any GOMS
Functionality: Consistency	NGOMSL	
Operator Sequence	CMN-GOMS NGOMSL	CPM-GOMS (see text)
Execution Time	KLM CMN-GOMS NGOMSL	CPM-GOMS
Procedure Learning Time	NGOMSL	
Error Recovery	Any GOMS	Any GOMS

Figure 1. GOMS techniques available for different combinations of task type and the type of design information desired. Note that only tasks that are goal-directed, routine cognitive skills are included, and information types not provided by GOMS models are not shown.

2.1.1. *Goal-directness.*

Many computer applications today are to support work-related goals: find information, make calculations, do analyses, produce reports, and control apparatus. In these instances, the user has a task goal and the application should support that goal as effectively as possible, both in terms of learning how to accomplish the goal and in actually accomplishing the goal. However, some applications are less goal-directed, like an electronic magazine through which a user would browse primarily for relaxation (as opposed to trying to find a particular article). GOMS is appropriate only for analyzing the goal-directed portions of a user's interaction, thus Figure 1 indicates that only a goal-directed task type is amenable to GOMS analysis. However, even in task situations that seem less goal-directed, there are often subgoals that need effective support; how relaxing would a magazine be if you couldn't figure out how to turn the electronic page?

In all GOMS analysis techniques, the designer or analyst must *start* with a list of high-level user goals. GOMS analyses and techniques do not provide this list; it must come from sources external to GOMS. Typically, this list of goals can be obtained from other task analysis approaches (e.g., see Diaper, 1989, Kirwan & Ainsworth, 1992, and Kieras, in press-b), including 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. Clearly such high-level task analysis is a critical step in any successful design approach, including using GOMS. Once this list is assembled, GOMS analyses can help identify the lower-level user goals quickly, and guide the design of the system so that the user can accomplish the given tasks in an efficient and learnable way. However, the GOMS analysis will not identify any new high-level goals or tasks that the analyst overlooked, or correct a misformulation of the user goals. At best, the analysis may stimulate the analyst's intuitions, and thus lead to a correction in the list of users' goals.

2.1.2. *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*,[1] 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 CMN, Chapter 11). 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* appropriate operators, methods, or selection rules. Rather, understanding and predicting such behavior is an active area of cognitive psychology research and may be addressed by other HCI analysis techniques (e.g., the Cognitive Walkthrough technique (Wharton, Rieman, Lewis & Polson, 1994) applies to exploratory behavior by novice users). Because of this limitation, Figure 1 indicates 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 and creative ones, have substantial components of routine cognitive skill. First, many tasks will evolve from problem-solving to routine skill after extensive use. 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 and half execution of routine procedures. 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 an analyst can use GOMS to improve the design to allow users to more effectively work on the non-routine, creative parts of the overall task.

---

[1] These skills usually involve some aspects of perception and motor actions, as well as cognitive processing. However, the psychology and HCI literature has typically used "cognitive skill" as a shorthand for "perceptual/cognitive/motor skill" and we continue that tradition.

2.1.3. *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 users' needs is likely to result in active systems.

GOMS analyses can be useful for either passive or active systems, using either of two approaches. The first is that the interruptability can be handled by the analyst. That is, the analyst provides a particular pattern of activity that has interruptions and goal rescheduling represented in a KLM or CPM-GOMS model. This approach was used successfully by Gray, John & Atwood (1993) in modeling telephone operators in their interaction with customers. Likewise, John and colleagues used CMN-GOMS with a relaxed goal-stack to predict the actions of a nine-year-old expert playing a video game (John & Vera, 1992; John, Vera & Newell, 1994).

The second approach is to assume 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. Endestad & Meyer (1993) used this approach to analyze a power plant operator's associate system. 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.

2.1.4. *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 (1993). Currently only CPM-GOMS is applicable to the truly parallel case.

However, it is important to consider when a task involving some parallel operations can be usefully approximated by a sequential model. Sometimes parallel operations can be represented as a simple modification to the sequential model. For example, it is logically necessary that users must visually locate 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 locate a fixed object on the screen (e.g., items on a menu bar) while pointing at it with a mouse, meaning that these two operators can execute in parallel. Because pointing takes longer than visually locating, 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).

Alternatively the parallel operations might take place below the level of analysis of the design issues in question. For example, although a telephone operator performs call-completion tasks by typing, listening, and visually searching in parallel, and predictions of execution time must be done with CPM-GOMS, a NGOMSL model could still be used to determine if the 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.[2]

In sum, GOMS analysis is applicable to many instances of goal-directed routine cognitive skill, even when it is embedded in problem-solving activities, creative interactions, or unstructured exploration. The tasks can be on both passive systems and active systems, and can either be usefully approximated as sequential operators or truly parallel. The next section describes the type of information obtainable from GOMS models of such tasks.

2.2.   Design Information Provided by GOMS Models

2.2.1.   *Functionality: Coverage and consistency.*

The primary design question about functionality is whether the system provides some method for every user goal. GOMS models 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 listed in Figure 1.

GOMS goes beyond other methods of analyzing functionality by looking at the procedures to accomplish the function. For instance, Object-Action analysis lists only the nouns and verbs (e.g., "move file") and is a useful early step in many designs (Olson & Moran, 1996). However, the system should provide functionality reasonably suited to the task, meaning that the method involved must be reasonably simple and fast. For example, consider that the user of a word processor might want to put 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. In other word processors, the user can only put footnotes on a page "by hand", and this work has to be redone if the length of the text changes. Despite this clumsiness, such limited word processors do 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. Working out a high-level GOMS model can reveal whether the system functionality has been chosen well in terms of meeting such requirements (see Kieras, in press-b). If it is important to quantify the 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 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.

2.2.2.   *Operator sequence.*

CMN-GOMS and NGOMSL can *predict* the sequence of overt physical operators a user will perform to accomplish a task (whereas with KLM and CPM-GOMS, the analyst must supply the

---

[2] But note that since the NGOMSL model would not accurately reflect the underlying knowledge 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 of learning for parallel tasks.

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. This prediction can be used to decide whether to add a new method to a system (for an example, see Card, Moran & Newell, 1980), or how best to train the use of methods. For sequential tasks, choose CMN-GOMS or NGOMSL depending on how many benchmark tasks are necessary (see further discussion in the next section). For parallel tasks, there is currently no technique for predicting the operator sequence. However, CPM-GOMS can be used to explore possible effects of design changes that could affect operator sequence.

2.2.3. *Execution time.*

Figure 1 indicates 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. These predictions hold for tasks that involve typing commands, point-and-click, and any other tasks for which good estimates of primitive operators can be obtained. Even visual-search tasks can be modeled with GOMS, allowing different screen layouts to be evaluated (Chuah, John & Pane, 1994; Lohse, 1993). These predictions have been validated in many laboratory studies and in actual real-world use of systems in the field. Therefore, using a well-constructed GOMS analysis, designers can use these predictions without further empirical validation.

Some HCI specialists feel that the restrictions of skilled error-free performance 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. However, the utility of GOMS execution time predictions should not be dismissed so casually. First, note that even empirical measurements of execution speed are questionable if users make many errors; this evaluation problem is not unique to GOMS. Second, there are many task domains where users are in fact well-practiced, and execution speed is critical. Such domains include not only clerical work, but also systems that support domain experts who perform highly skilled tasks for days at a time. The time of such domain experts is too valuable to waste on a slow and clumsy interface, so designing for execution speed is critical, and GOMS can help do so. Third, even if users do make errors, it is likely that a system that is predicted to be slow under a no-error assumption will also be slow when real users are involved and make errors, which generally result in much longer execution times. Thus GOMS predictions of execution time give essentially a best-case basis for comparing two designs on this dimension. As an analogy, note that EPA automobile gas mileage ratings allow useful comparison between cars during a purchase decision, even though the actual mileage obtained may vary. Likewise, comparing execution time predictions can be useful in choosing designs even if the occurrence of errors inflates the actual execution time. The status of GOMS with regard to user errors will be discussed more below.

Although no system should be designed with unnecessarily slow action sequences, execution time may not be a critical design issue. For example, if the system will not be used enough by any single user to produce highly-practiced performance (e.g. a walk-up-and-use application to help home-buyers locate a house in a new city), then GOMS predictions of execution time may not be as useful to the design as evaluation techniques that focus on other design issues such as the recognizability of menu terms to the first-time user. Also, execution time is not an important variable in some domains, 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. But it is important to realize that current interfaces for walk-up-and-use, entertainment, and educational software often waste the user's time with methods that are slow, inconsistent, and clumsy for no good reason. 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 (discussed below), then CMN-GOMS or NGOMSL models will provide both execution time and this other information. CPM-GOMS is the technique for making execution time predictions in truly parallel task situations.

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; Kieras, Wood, Abotel, & Hornof, 1995).

2.2.4. *Learning time.*

Information about learning time is provided only by NGOMSL models, and these predictions cover 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. However, making absolute predictions of learning time involves many issues and complications discussed in John & Kieras (in press). The simplest advice for practical situations is to limit the use of NGOMSL learning time predictions to *relative* comparisons of alternative designs. Such comparisons between designs are fairly robust, since a more complex interface will normally be harder to learn than a simpler one in a variety of possible learning situations.

When applying the predictions, the analyst should keep in mind two important qualifications. 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, constructing a mental model of the system, or learning new operators (e.g. BANK-AIRCRAFT). GOMS models do not represent the knowledge and mechanisms required for learning substantial domain knowledge.

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 unlabeled 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 NGOMSL) 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! Other analysis techniques, e.g. the Cognitive Walkthrough technique (Wharton, Rieman, Lewis & Polson, 1994), are better suited to uncover such problems.

2.2.5. *Error recovery support.*

There are three design goals involved with user 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 goals, at this time research on human errors is still far from providing more than the familiar rough guidelines concerning the prevention of user error (e.g. Norman, 1983). No methodology for predicting when and what errors users will make as a function of interface design has yet been sufficiently validated to be an engineering tool, regardless of the theoretical approach, and even the theoretical analysis of human error is still in its infancy (see Reason, 1990 for more discussion). Several doctoral theses have demonstrated a relationship between observed errors and properties of GOMS models (e.g. Lerch, Mantei, & Olson, 1989; Smelcer, 1989; Jong, 1991), but none have yet made the transition to practical use in real-world design situations.

However, as originally pointed out by CMN, GOMS has a direct application to the third design goal listed above, that of helping users recover from an error once it has occurred. In this case, 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, such as a ubiquitous UNDO command, for the goal *RECOVER-FROM-ERROR*? Such a design question is no different in substance from designing the methods for other user goals. Figure 1 indicates that any of the GOMS models can be used to address this question, with the specific choice depending on the specific aspect of interest. Thus, once the possible frequent or important errors are determined, evaluating designs for the quality of support for error recovery can be done with extant GOMS family members.

2.2.6. *Informal understanding of a design.*

An additional type of design information, not listed in Figure 1, is an informal understanding of the design issues. That is, as pointed out by Karat & Bennett (1991), a GOMS model can have purely heuristic value since it makes explicit what the system requires the user to do. So constructing a GOMS model is a way for a user interface designer to become more aware of the implications of a design. This can be important because it appears that a common design error is to focus on non-procedural issues such as screen layout or graphic design, while burdening the user with a clumsy interface. Thus, 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 (see Kieras, in press-b, for further discussion).

2.2.7. *Information not provided by GOMS.*

There are many other kinds of information relevant to design that are not included in Figure 1 because there are no GOMS family techniques for them. Other aspects of the user's knowledge of the system may be important to some design situations, like being able to perform mental simulations of an internalized device model, or analogical reasoning to infer operating procedures (see Kieras and Polson, 1985, for more discussion). Other existing and potential approaches to task analysis are required to capture these other forms of task knowledge. For example, work on electronics troubleshooting (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). In addition, information other than the users' knowledge may also be relevant to design. Examples of such information 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 command words; (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. Since these types of design issues are generally independent of the procedural quality of the interface, designers must use other techniques besides GOMS to explore their effect.

2.3.   Using GOMS in Design: Uses, effort, and payoff.

In this section we will examine a few of the common uses of the information provided by GOMS models. First, however, we will discuss general issues about the time to learn and use GOMS models. For specific design and evaluation questions, the effort required to use the different GOMS techniques depends on the types of questions as well as on the techniques themselves. There is a start-up cost commensurate with the complexity of the system being modeled, but once the first models for a design or evaluation problem are built, answers to subsequent questions require much less effort. 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 3 demonstrate this approach. Thus, the decision to invest effort in GOMS modeling depends on how many design issues and iterations will be involved.

2.7.4. *General issues about the time to learn and use GOMS*

Using any method in design entails 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, as a sort of "Cadillac" of design techniques to be used only when cost is no object (e.g. Lewis & Rieman, 1994; Landauer, 1995). This impression has been so pervasive and misleading that it requires some discussion to correct.

First, the impression is probably a residue of the well-known difficulty of applying the techniques in their research form, which does not reflect the effort required to learn and to apply an already developed technique. In fact, there is now enough accumulated experience (as in the cases described in Section 3) to assess the actual costs. It is clear that the GOMS techniques discussed here have an excellent return on investment, and the amount of the investment is much less than commonly believed. For example, an estimate of the overall effort in applying NGOMSL to an actual design problem is provided in Gong (1993; summarized in Section 3.2, Case 9). In brief, he found that using NGOMSL in an application development situation required a fraction of the software programming time, and much less time than empirical evaluation.

Second, the impression is unfounded, because at this time, there is an inadequate basis for comparing the effectiveness of GOMS techniques with other techniques. The amount of technique-comparison research is very scanty, and is rife with methodological problems (Gray & Salzman, 1996). However, at least some of these results support the argument that GOMS is easy and efficient; for example, in the *raw* results of the Nielsen & Phillips (1993) comparison, KLM done by undergraduates in their first HCI class was the most accurate technique. Also, the analytic work required by GOMS techniques relative to other techniques has often been mischaracterized. For example, the Cognitive Walkthrough technique (Wharton, Rieman, Lewis & Polson, 1994) is often considered one of the "discount techniques". Cognitive Walkthrough requires first defining the user's actions at keystroke level, for a set of benchmark cases that represent important user tasks. Developing this information is in fact most of the work required to apply the KLM, and is a substantial part of the preliminary work in defining a CMN-GOMS or NGOMSL model. So, if Cognitive Walkthrough with its level of effort qualifies as a fast and efficient "discount" evaluation technique, then GOMS modeling should also be considered a fast and efficient technique.

Based on our experience with teaching GOMS techniques to university and industrial students, and the experience of other HCI instructors at several universities, 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 Nielsen & Phillips, 1993; John, 1994). A single class session seems to suffice for CMN-GOMS

as well, 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 harder to learn and to use, but for different reasons. The difficulty in using NGOMSL models is in 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 (1988, in press-a) and a couple of homework assignments with feedback. Full-day tutorials, such as at the CHI conferences (Kieras, 1994) and in industrial short-courses appear to be adequate to get software developers started in the technique. Also, an NGOMSL interpreter is under development which should facilitate precise model-building (Wood, 1993; Byrne, Wood, Sukaviriya, Foley, & Kieras, 1994; Kieras, Wood, Abotel, & Hornof, 1995).

CPM-GOMS is more difficult to use than the KLM or CMN-GOMS because the analyst must identify and describe in detail how perceptual, cognitive, and motor processing activities are coordinated in time. However, John and Gray (1995) 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, 1995).

Thus, GOMS modeling is not an unconquerable skill attained by only a few researchers and their apprentices after years of effort. In fact, there have been numerous applications of GOMS in the real world, many by people who attended only a tutorial or read about the technique in the how-to references. Section 3 details several case studies that demonstrate this point.

2.3.1. *Profiling: a means to focus design effort.*

A critical practical design problem is where the design effort should be focused and GOMS can make substantial contributions to this problem. For example, a developer might need to know if user procedures should be streamlined to decrease the human execution time, or if it is more important to improve the underlying software algorithms to decrease the response time of the system. To answer such questions, GOMS 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. That is, since GOMS models describe what the user actually does, keystroke by keystroke, sometimes even eye-movement by eye-movement, it is easy to identify which procedures are contributing the most time, be they procedures performed by the user or system responses. Then priorities can be assigned to design issues in a rational manner.

Some of the examples of actual GOMS design projects described in Section 3 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 system's response latency (Gray, John & Atwood, 1993). Because system profiling requires quantitative predictions of execution time, 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 building prototypes that respond in real-time and empirical testing, and thus can prevent resources from being poured into design efforts that have relatively little value.

2.3.2. *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 Section 3.

Comparisons can be made with summative predictions, i.e., System A takes half the time of System B to perform a task, or System C requires one third the learning time of System D. Perhaps more importantly, as discussed under profiling above, it is easy to compare the procedures of one system with those another system to discover the merits and deficits of each.

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? The first situation requires a rather shallow CMN-GOMS analysis, whereas the second may require in-depth CPM-GOMS analysis, and the third 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 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, John & Atwood, 1993).

2.3.3. *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 1980; 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.

2.3.4. *Documentation and on-line help systems.*

Documentation and on-line help systems pose design questions that are very well addressed by the 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, often documentation and help provide 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, should be included in the design, and presented in training and documentation (e.g., Nilsen, Jong, Olson & Polson, 1992). For example, in telephone operator call-handling, CPM-GOMS could predict execution time differences between different methods; identifying these differences suggested the most efficient methods and selection rules to include in documentation and training materials.

Thus, GOMS models can be used to inform many parts of the design process, both qualitatively and quantitatively. The next section demonstrates a diverse set of these uses in design and evaluation through ten real-world examples.

3. EXAMPLES OF ACTUAL APPLICATIONS OF GOMS ANALYSES IN REAL-WORLD SYSTEM DESIGNS

This section presents examples of the application of GOMS to the design and evaluation of the interfaces for a variety of different kinds of systems, all of which were real systems under development. These cases illustrate both the wide scope of applicability of GOMS models, and also their effectiveness at answering design questions quickly and efficiently.[3] Because most of these cases concern actual development projects, they have not been described in the open scientific literature, but rather have been documented in industrial technical reports, or as noted below, only as a result of interviews by the authors. The descriptions are brief both because our goal is to illustrate a wide variety of applications in a limited space, and also because in several cases, proprietary concerns limit how much information is available. Note also that these are not the only cases of GOMS in real-world use; for example, additional cases have appeared (e.g., Beard, Smith & Denelsbeck, in press; Nielsen's sidebar in John, 1995; Vera & Rosenblatt, 1995).

While considering these cases, the reader should beware of trivializing the presented design

---

[3] The purpose of this section is to illustrate how GOMS has been applied to real problems. We do not argue that the design problems in these cases could not have been dealt with by other techniques.

problems with the benefit of hindsight — almost any interface problem will look simple to identify and fix *after* it has been described! Rather, every example presented concerns design issues that arose in the course of a real-world professional software development effort.

3.1.   KLM Applications

Case 1.   Mouse-driven text editor. 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 goal was 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.   Directory assistance workstation. (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 retrieved 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 3. Space operations database system. *(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, the results analyzed, 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 4. CAD system for mechanical design. *(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. They 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.

3.2.   NGOMSL Applications

Case 5. Television control system. 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, and inconsistent navigation and selection methods. 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 6. Nuclear power plant operator's associate. 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 complete prototype system involved multiple networked computers with a total of 17 display monitors, and included a full simulation of an actual 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 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 7. Intelligent tutoring system. Steinberg & Gitomer (1993) describe how a 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 8. Industrial scheduling system. Nesbitt, Gorton, and Rantanen (1994) report 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, or to delete or modify downtimes already scheduled.

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 scheduled 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 delete or modify existing ones, under the assumption that the list of downtimes or modifications 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 of the interface designs required only about two days total, and was described as being neither difficult nor time consuming. In return, the 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, mostly due to a substantial improvement in modifying existing schedule times.

Case 9. CAD system for ergonomic design. 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.

3.3.   CPM-GOMS Application

Case 10.   Telephone operator workstation. The details of this application of CPM-GOMS, both technical and managerial, can be found in Gray, John, and Atwood (1993) and Atwood, Gray, and John (1996). 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.

4.   SUMMARY AND CONCLUSIONS

The GOMS family of analysis techniques provide many different types of information for system design and evaluation, as discussed in Section 2. The real-world cases in Section 3 demonstrate many of these uses and provide the following lessons.

- The systems involved were actual systems under development, showing that GOMS is not just a research approach, but is applicable in the real world of software and system development.

- The analyses "worked" — they generated design guidance, showing that GOMS modeling does indeed produce information that can be effectively used in the design process.

- The systems were quite varied in type, being a mix of multimedia tutoring systems, specialized clerical applications, process control systems, consumer electronics, and systems for domain experts, as well as general-purpose desktop applications. Thus, GOMS is applicable across types of applications and levels of domain expertise.

- Most of the systems were quite complex and elaborate, refuting the stereotype that GOMS is limited to desktop tasks such as text-editing; rather GOMS can be applied on a large scale to fully-complicated real-world systems.

- The GOMS modeling provided usability information when the system was in the earliest of design phases, undergoing a redesign, or was being evaluated after implementation. Thus GOMS is useful across all phases of system development.

- In some cases, the GOMS evaluation provided usability information when empirical evaluation would be extremely difficult, showing that GOMS can be useful when other evaluation techniques might not be practical.

- When documented, the effort required to conduct the analyses was reasonable in light of the value of the results obtained, showing by actual experience that GOMS modeling is cost-effective.

- In several cases, the analyses were done by groups following a brief workshop or tutorial introduction, showing that GOMS can be learned quickly.

Thus, the current family of GOMS techniques are truly usable and useful as a designers analytic tool. Readers interested in details of the theoretical differences between GOMS family-members and pointers to ongoing cognitive modeling research for HCI, are directed to the following paper (John & Kieras, in press).

ACKNOWLEDGMENTS

REFERENCES

Atwood, M. E., Gray, W. D., & John, B. E. (1996).Project Ernestine: Analytic and empirical methods applied to a real-world CHI problem. In M. Rudisill, C. Lewis, P. B., Polson, & T. D. McKay (eds.) *Human-Computer Interface Design: Success Stories, Emerging Methods and Real-World Context*. San Francisco: Morgan Kaufmann Publishers.

Beard, D. V., Smith, D. K., & Denelsbeck, K. M. (in press). Quick and dirty GOMS: A case study of computed tomography interpretation. *Human-Computer Interaction*.

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. (1980). 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. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Carley, K. M., & Prietula, M. J. (1994). *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, UK.: Ellis Horwood.

Dillon, A., Sweeney, M., & Maguire, M. (1993). A survey of usability engineering within the European IT industry -- Current practice and needs. In J. L. Alty, D. Diaper, and S. Guest (eds.) *People and Computers VIII*. Cambridge: Cambridge University Press. pp. 81-94.

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 MacMillan Company, 1917.

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

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.

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. (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. & Salzman, M. C. (1996) *Damaged merchandise? A review of experiments that compare usability evaluation methods*. Manuscript submitted for publication.

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. (1995) Why GOMS? *interactions*, *vol. 2*, no. 4. pp. 80-89.

John, B. E. & Gray, W. D. (1995) *GOMS Analyses for Parallel Activities*. Tutorial Notes, CHI, 1995 (Denver, Colorado, May 7-11, 1995), ACM, New York.

John, B. E., & Kieras, D. E. (in press). The GOMS family of analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction* .

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*, *13*, 4, pp. 255-267.

Jong, H-S. (1991) *The subgoal structure as a cognitive control mechanism in a human-computer interaction framework*. Ph.D. dissertation, The University of Michigan.

Karat, J., and Bennett, J. (1991). Modeling the user interaction methods imposed by designs. In M. Tauber and D. Ackermann (Eds.), *Mental models and Human-Computer Interaction 2*. Amsterdam: Elsevier.

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

Kieras, D. E. (1994). *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. (in press-a). A Guide to GOMS Model Usability Evaluation using NGOMSL. In M. Helander & T. Landauer (Eds.), *The handbook of human-computer interaction*. (Second Edition). Amsterdam: North-Holland.

Kieras, D. E. (in press-b). Task analysis and the design of functionality. In T. Allen (Ed), *Handbook of computer science and engineering*, CRC Press, Inc.

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.

Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs. *UIST'95 Proceedings*.

Kirwan, B., & Ainsworth, L. K. (1992). *A Guide to Task Analysis*. London: Taylor and Francis.

Landauer, T. K. (1995). *The trouble with computers: Usefulness, usability, and productivity*. Cambridge: MIT Press.

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.

Lewis, C. & Rieman, J. (1994). *Task-centered user interface design: A practical introduction*. Shareware book available at ftp.cs.colorado.edu/pub/cs/distribs/clewis/HCI-Design-Book.

Lohse, G. L. (1993). A cognitive model for understanding graphical perception. *Human-Computer Interaction, 8*(4), 353-388.

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

Nesbitt, K. Gorton, D., & Rantanen, J. (1994). *A case study of GOMS analysis: Extension of user interfaces*. Technical Report BHPR/ETR/R/94/048, BHP Research - Newcastle Laboratories, Wallsend, NSW, Australia 2287.

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

Nielsen, J. & Mack, R. L. (Eds.) (1994). *Usability Inspection Methods*. New York: John Wiley & Sons, Inc.

Nielsen, J. & 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.

Nilsen, E., Jong, H, Olson, J. S., & Polson, P. G. Method engineering: From data to model to practice. In *Proceedings of CHI*, 1992 (Monterey, California, May 3- May 7, 1992) ACM, New York, 1992. pp. 313-319.

Norman, D. A. (1983). Design Rules based on analyses of human error. *Communications of the ACM*, 26(4), 254-258.

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. S. & Moran, T. P. (1996) Mapping the method muddle: Guidance in using methods for user interface design. In M. Rudisill, C. Lewis, P. B., Polson, & T. D. McKay (eds.) *Human-Computer Interface Design: Success Stories, Emerging Methods and Real-World Context.* San Francisco: Morgan Kaufmann Publishers.

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.

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.

Smelcer, J. B. (1989). *Understanding user errors in database query*. Ph.D. dissertation, The University of Michigan.

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.

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

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

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

Vera, A. H. & Rosenblatt, J. K. (1995). Developing user model-based intelligent agents. In J. D. Moore and J. F. Lehman (eds.) *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*. Mahwah, NJ: Lawrence Erlbaum. pp. 500-505.