Model-based Evaluation

David Kieras

University of Michigan

a preprint of
Kieras, D.E. Model-based evaluation (in press). In J. Jacko & A. Sears (Eds.), *The Human-Computer Interaction Handbook* (2nd Ed). Mahwah, New Jersey: Lawrence Erlbaum Associates.

Introduction

What is model-based evaluation?

Model-based evaluation is using a *model* of how a human would use a proposed system to obtain predicted usability measures by calculation or simulation. These predictions can replace or supplement empirical measurements obtained by user testing. In addition, the content of the model itself conveys useful information about the relationship between the user's task and the system design.

Organization of this chapter

This chapter will first argue that model-based evaluation is a valuable supplement to conventional usability evaluation, and then survey the current approaches for performing model-based evaluation. Because of the considerable technical detail involved in applying model-based evaluation techniques, this chapter cannot include "how to" guides on the specific modeling methods, but they are all well documented elsewhere. Instead, this chapter will present several high-level issues in constructing and using models for interface evaluation, and comment on the current approaches in the context of those issues. This will assist the reader in deciding whether to apply a model-based technique, which one to use, what problems to avoid, and what benefits to expect. Somewhat more detail will be presented about one form of model-based evaluation, GOMS models, which is a well-developed, relatively simple and "ready to use" methodology applicable to many interface design problems. A set of concluding recommendations will summarize the practical advice.

Why use model-based evaluation?

Model-based evaluation can be best viewed as an alternative way to implement an iterative process for developing a usable system. This section will summarize the standard usability process, and contrast it with a process using model-based evaluation.

Standard usability design process. In simplified and idealized form, the standard process for developing a usable system centers on user testing of prototypes that seeks to compare user performance to a specification or identify problems that impair learning or performance. After performing a task analysis and choosing a set of benchmark tasks, an interface design is specified based on intuition and guidelines both for the platform/application style and usability. A prototype of some sort is implemented, and then a sample of representative users attempts to complete the benchmark tasks with the prototype. Usability problems are noted, such as excessive task completion time or errors, being unable to complete a task, or confusion over what to do next. If the problems are serious enough, the prototype is revised, and a new user test conducted. At some point the process is terminated and the product completed, either because no more serious problems have been detected, or there is not enough time or money for further development. See Dumas (this volume) for a complete presentation.

The standard process is a straightforward, well-documented methodology with a proven record of success (Landauer, 1995). The guidelines for user interface design, together with knowledge possessed by those experienced in interface design and user testing, adds up to a substantial accumulation of wisdom on developing usable systems. There is no doubt that if this process were applied more widely and thoroughly, the result would be a tremendous improvement in software quality. User testing has always been considered the "gold standard" for usability assessment. However, it has some serious limitations - some practical and others theoretical.

Practical limitations of user testing. A major practical problem is that user testing can be too slow and expensive to be compatible with current software development schedules, so a focus of HCI research for many years has been ways to tighten the iterative design loop. For example, better prototyping tools allow prototypes to be developed and modified more rapidly. Clever use of paper mockups or other early user input techniques allows important issues to be addressed before making the substantial investment in programming a prototype. So-called inspection evaluation methods seek to replace user testing with other forms of evaluation, such as expert surveys of the design, or techniques such as cognitive walkthroughs (see Cockton, et al, this volume).

If user testing is really the best method for usability assessment, then it is necessary to come to terms with the unavoidable time and cost demands of collecting behavioral data and analyzing it, even in the rather informal manner that normally suffices for user testing. For example, if the system design were substantially altered on an iteration, it would be necessary to retest the design with a new set of test users. While it is hoped that the testing process finds fewer important problems with each iteration, the process does not get any faster with each iteration - the same adequate number of test users must perform the same adequate number of representative tasks, and their performance assessed.

The cost of user testing is especially pronounced in expert-use domains, where the user is somebody like a physician, a petroleum geologist, or an engineer. Such users are few, and their time is valuable. This may make relying on user testing too costly to adequately refine an interface. A related problem is evaluating software that is intended to serve experienced users especially well. Assessing the quality of the interface requires a very complete prototype that can be used in a realistic way for an extended period of time so that the test users can become experienced. This drives up the cost of each iteration, because the new version of the highly-functional prototype must be developed and the lengthy training process has to be repeated. Other design goals can also make user testing problematic: Consider developing a pair of products for which skill is supposed to transfer from one to the other. Assessing such transfer requires prototyping both products fully enough to train users on the first, and then training them on the second, to see

if the savings in training time are adequate. Any design change in either of the products might affect the transfer, and thus require a repeat test of the two systems. This double-dose of development and testing effort is probably impractical except in critical domains, where the additional problem of testing with expert users will probably appear.

Theoretical limitations of user testing. From the perspective of scientific psychology, the user testing approach takes very little advantage of what is known about human psychology, and thus lacks grounding in psychological theory. Although scientific psychology has been underway since the late 1800s, the only concepts relied on by user testing are a few basic concepts of how to collect behavioral data. Surely more is known about human psychology than this! The fact is that user testing methodology would work even if there was no systematic scientific knowledge of human psychology at all - as long as the designer's intuition leads in a reasonable direction on each iteration, it suffices merely to revise and retest until no more problems are found. While this is undoubtedly an advantage, it does suggest that user testing may be a relatively inefficient way to develop a good interface.

This lack of grounding in psychological principles is related to the most profound limitation of user testing: it lacks a systematic and explicit representation of the knowledge developed during the design experience; such a representation could allow design knowledge to be accumulated, documented, and systematically reused. After a successful user testing process, there is no representation of how the design "works" psychologically to ensure usability - there is only the final design itself, as described in specifications or in the implementation code. These descriptions normally have no theoretical relationship to the user's task or the psychological characteristics of the user. Any change to the design, or to the user's tasks, might produce a new and different usability situation, but there is no way to tell what aspects of the design are still relevant or valid. The information on why the design is good, or how it works for users, resides only in the intuitions of the designers. While designers often have outstanding intuitions, we know from the history of creations such as the medieval cathedrals that intuitive design is capable of producing magnificent results, but is also routinely guilty of costly over-engineering or disastrous failures.

The model-based approach. The goal of model-based evaluation is to get some usability results before implementing a prototype or testing with human subjects. The approach uses a *model* of the human-computer interaction situation to represent the interface design and produce predicted measurements of the usability of the interface. Such models are also termed *engineering models* or *analytic models* for usability. The model is based on a detailed description of the proposed design and a detailed task analysis; it explains how the users will accomplish the tasks by interacting with the proposed interface, and uses psychological theory and parametric data to generate the predicted usability metrics. Once the model is built, the usability predictions can be quickly and easily obtained by calculation or by running a simulation. Moreover, the implications of variations on the design can be quickly explored by making the corresponding changes in the model. Since most variations are relatively small, a circuit around the revise/evaluate iterative design loop is typically quite fast once the initial model-building investment is made. Thus unlike user testing, iterations generally get faster and easier as the design is refined.

In addition, the model itself summarizes the design, and can be inspected for insight into how the design supports (or fails to support) the user in performing the tasks. Depending on the type of model, components of it may be reusable not in just different versions of the system under development, but in other systems as well. Such a reusable model component captures a stable feature of human performance, task structures, or interaction techniques; characterizing them contributes to our scientific understanding of human-computer interaction.

The basic scheme for using model-based evaluation in the overall design process is that iterative design is done first using the model, and then by user testing. In this way, many design decisions can be worked out before investing in prototype construction or user testing. The final user testing process is required for two reasons: First, the available modeling methods only cover certain aspects of usability; at this time, they are limited to predicting the sequence of actions, the time required to execute the task, and certain aspects of the time required to learn how to use the system. Thus user testing is required to cover the remaining aspects. Second, since the modeling process is necessarily imperfect, user testing is required to ensure that some critical issue has not been overlooked. If the user testing reveals major problems along the lines of a fundamental error in the basic concept of the interface, it will be necessary to go back and reconsider the entire design; again model-based iterations can help address some of the issues quickly. Thus, the purpose of the model-based evaluation is to perform some of the design iterations in a lower-cost, higher-speed mode before the relatively slow and expensive user testing.

What "interface engineering" should be. Model-based evaluation is not the dominant approach to user interface development; most practitioners and academics seem to favor some combination of user testing and inspection methods. Some have tagged this majority approach as a form of "engineering." However, even a cursory comparison to established engineering disciplines makes it clear that conventional approaches to user interface design and evaluation has little resemblance to an engineering discipline. In fact, model-based evaluation is a deliberate attempt to develop and apply true engineering methods for user interface design. The following somewhat extended analogy will help clarify the distinction, as well as explain the need for further research in modeling techniques.

If civil engineering were done with iterative empirical testing, bridges would be built by erecting a bridge according to an intuitively appealing design, and then driving heavy trucks over it to see if it cracks or collapses. If it does, it would be rebuilt in a new version (e.g. with thicker columns) and the trial repeated; the iterative process continues with additional guesses until a satisfactory result is obtained. Over time, experienced bridge-builders would develop an intuitive feel for good designs and how strong the structural members need to be, and so will often guess right. However, time and cost pressures will probably lead to cutting the process short by favoring conservative designs that are likely to work, even though they might be unnecessarily clumsy and costly.

Although early bridge-building undoubtedly proceeded in this fashion, modern civil engineers do not build bridges by iterative testing of trial structures. Rather, under the stimulus of design failures (Petrosky, 1985), they developed a body of scientific theory on the behaviors of structures and forces, and a body of principles and parametric data on the strengths and limitations of bridge-building materials. From this theory and data, they can quickly construct models in the form of equations or computer simulations that allow them to evaluate the quality of a proposed design without having to physically construct a bridge. Thus an investment in theory development and measurement enables engineers to replace an empirical iterative process with a theoretical iterative process that is much faster and cheaper per iteration. The bridge is not built until the design has been tested and evaluated based on the models, and the new bridge almost always performs correctly. Of course, the modeling process is fallible, so the completed bridge is tested before it is opened to the public, and occasionally the model for a new design is found to be seriously inaccurate and a spectacular and deadly design failure is the result. The claim is not that using engineering models is perfect or infallible, only that it saves time and money, and thus allows designs to be more highly refined. In short, more design iterations results in better designs, and better designs are possible if some of the iterations can be done very cheaply using models.

Moreover, the theory and the model summarize the design and explain why the design works well or poorly. The theoretical analysis identifies the weak and strong points of the design, giving guidance to the designer where intuition can be applied to improve the design; a new analysis can then test whether the design has actually been improved. Engineering analysis does not result in simply static repetition of proven ideas. Rather, it enables more creativity because it is now possible to cheaply and quickly determine whether a new concept will work. Thus novel and creative concepts for bridge structures have steadily appeared once the engineering models were developed.

Correspondingly, model-based evaluation of user interfaces is simply the rigorous and science-based techniques for how to evaluate user interfaces without user testing; it likewise relies on a body of theory and parametric data to generate predictions of the performance of an engineered artifact, and explain why the artifact behaves as it does. While true interface engineering is nowhere as advanced as bridge engineering, useful techniques have been available for some time, and should be more widely used. As model-based evaluation becomes more developed, it will become possible to rely on true engineering methods to handle most of the routine problems in user interface design, with considerable savings in cost and time, and with reliably higher quality. As has happened in other branches of engineering, the availability of powerful analysis tools means that the designer's energy and creativity can be unleashed to explore fundamentally new applications and design concepts.

Three Current Approaches

Research in HCI and allied fields has resulted in many models of human-computer interaction at many levels of analysis. This chapter restricts attention to approaches that have developed to the point that they have some claim, either practical or scientific, to being suitable for actual application in design problems. This section identifies three current approaches to modeling human performance that are the most relevant to model-based evaluation for system and interface design. These are task network models, cognitive architecture models, and GOMS models.

Task network models. In task network models, task performance is modeled in terms of a PERT-chart-like network of processes. Each process starts when its prerequisite processes have been completed, and has an assumed distribution of completion times, This basic model can be augmented with arbitrary computations to determine the completion time, and what its symbolic or numeric inputs and outputs should be. Note that the processes are usually termed "tasks," but they need not be human-performed at all, but can be machine processes instead. In addition, other information, such as workload or resource parameters can be attached to each process. Performance predictions are obtained by running a Monte-Carlo simulation of the model activity, in which the triggering input events are generated either by random variables or by task scenarios. A variety of statistical results, including aggregations of workload or resource usage, values can be readily produced. The classic SAINT (Chubb, 1981) and the commercial MicroSaint tool (Laughery, 1989) are prime examples. These systems originated in applied human factors and systems engineering, and are heavily used in system design, especially for military systems.

Cognitive architecture models. Cognitive architecture systems are surveyed by Byrne (this volume). These systems consist of a set of hypothetical interacting perceptual, cognitive, and motor components assumed to be present in the human, and whose properties are based on empirical and theoretical results from scientific research in psychology and allied fields. The functioning of the components and their interactions are typically simulated with a computer program, which in

effect produces a *simulated human* performing in a *simulated task environment* that supplies inputs (stimuli) to the simulated human, and reacts to the outputs (responses) produced by the simulated human. Tasks are modeled primarily by programming the cognitive component according to a task analysis, and then performance predictions are obtained by running the simulation using selected scenarios to generate the input events in the task. Because these systems are serious attempts to represent a theory of human psychological functions, they tend to be rather complex, and are primarily used in basic research projects; there has been very limited experience in using them in actual design settings.

GOMS models. GOMS models are the original approach to model-based evaluation in the computer user interface field; both the model-based evaluation approach and GOMS models were presented as methods for user interface design in the seminal Card, Moran, and Newell (1983) presentation of the psychology of human-computer interaction. They based the GOMS concept on the theory of human problem-solving and skill acquisition. In brief, GOMS models describe the knowledge of procedures that a user must have in order to operate a system, The acronym and the approach can be summarized as follows: The user can accomplish certain Goals (G) with the system; Operators (O) are the basic actions that can be performed on the system such as striking a key or finding an icon on the screen; Methods (M) are sequences of Operators that when executed, accomplish a Goal; Selection Rules (S) describe which Method should be used in which situation to accomplish a Goal, if there is more than one available. Constructing a GOMS model involves writing out the methods for accomplishing the task goals of interest, and then calculating predicted usability metrics from the method representation.

There are different forms of GOMS models, systematized by John & Kieras (1996a, b), which represent the methods at different levels of detail, and whose calculations can range in complexity from simple hand calculations to full-fledged simulations. John and Kieras pointed out that the different forms can be viewed as being based on different simplified cognitive architectures that make the models easy to apply to typical interface design problems and insulate the model-builder from many difficult theoretical issues. More so than any other model-based approach, GOMS models have a long and well-established track record of success in user interface design, although they are not used as widely as their simplicity and record would justify. Although still under development by researchers, GOMS models are emphasized in this chapter because in some forms they are a “ready to use” modeling methodology. A later section will describe their rationale more completely, but the reader is referred to John & Kieras (1996a, b) for a thorough discussion.

Theoretical Basis for Choosing a Model-based Evaluation Technique

This section presents several key issues concerning the theoretical foundations of model-based evaluation, concerning the basic sources of information and applicability of the modeling approach. When choosing or evaluating a technique for model-based evaluation, the potential user should consider these issues; the techniques differ widely in how well they handle certain fundamental questions. The next section will focus on the practical problems of applying a modeling technique once it has been chosen. In both sections, the three basic approaches to model-based evaluation are commented on as appropriate. Advice is given to both the user of model-based evaluation and the developer of model-based techniques.

Psychological constraints are essential

The concept of model-based evaluation in system design has a long history and many different proposed methods (for early surveys see Pew, Baron, Feehrer, & Miller, 1977; MacMillan, Beevis, Salas, Strub, Sutton, & Van Breda, 1989; Elkind, Card, Hochberg, & Huey, 1989). However, the necessary scientific basis for genuinely powerful models has been slow to develop. The key requirement for model-based evaluation is that building a model to evaluate a design must be a routine, production, or engineering activity, and not a piece of basic scientific research on how human psychological factors are involved in a particular computer usage situation. This means that the relevant psychological science must not only be developed first, but also then systematized and encapsulated in the modeling methodology itself. That is, a modeling methodology must provide *constraints* on the content and form of the model, and these constraints must provide the psychological validity of the model as a predictor of human performance. In other words, if the model builder can do essentially anything in the modeling system, then the only way the resulting model can be psychologically valid is if the model builder does all of the work to construct a valid psychological theory of human cognition and behavior in the task, and then ensure that the constructed model accurately reflects this theory.

Of course it takes tremendous time, effort, and training to construct original psychological theory, far more than should be necessary for most interface design situations. Although the decisions in truly novel or critical design situations might require some fundamental psychological research, most interface design situations are rather routine: the problem is to match a computer system to the user's tasks using known interface design concepts and techniques. It should not be necessary to be an expert researcher in human cognition and performance to carry this out.

Thus the key role of a modeling system is to provide constraints based on the psychological science, so that a model constructed within the system has a useful degree of predictive validity. In essence, simply by using the modeling system according to its rules, the designer must be able to construct a scientifically plausible and usefully accurate model "automatically."

A simple series of examples will help make the point: Computer user interfaces involve typing of arbitrary strings of text on the keyboard and pointing with a mouse. The time required to type on the keyboard and to point with a mouse are fairly well documented. If task execution times are of interest, an acceptable modeling system should include these human performance parameters so that the interface designer does not have to collect them or guess them.

Furthermore, because both hands are involved in typing strings of text, users cannot type at the same time as they move the mouse cursor; these operations must be performed sequentially, taking rather more time than if they could be done simultaneously. A modeling system should make it impossible to construct a model of an interface that overzealously optimizes execution speed by assuming that the user could type strings and point simultaneously; the sequential constraint should be enforced automatically. A high-quality modeling system would not only enforce this constraint, but also automatically include the time costs of switching between typing and pointing, such as the time to move the hand between the mouse and the keyboard. There are many such constraints on human performance, some of them quite obvious, as in these examples, and some very subtle. A good modeling system will represent these constraints in such a way that they are automatically taken into account in how the model can be constructed and used. Because of the subtleties involved, computational tools are especially valuable for constructing and using models because they can help enforce the psychological constraints and make it easier for the model-builder to work within them.

A brief history of constraints in modern psychological theory

Theoretical constraints are not easy to represent or incorporate; a coherent and rigorous theoretical foundation is required to serve as the substrate for the network of constraints, and suitable foundations were not constructed until fairly recently. Through most of second half of the 20th century, psychological theory was mired in a rather crude form of information-processing theory, in which human activity was divided into information-processing stages, such as perception, memory, decision-making, and action, usually depicted as a flowchart with a box for each stage, various connections between the boxes, and perhaps with some fairly simple equations that described the time required for each stage or the accuracy of its processing. However, there was little constraint on the possible data contained in each box, or the operations performed there; a box could be of arbitrary complexity, and no actual explicit mechanism had to be provided for any of them. Such models were little more than a "visual aid" for theories posed in the dominant forms of informal verbal statements or rather abstract mathematical equations. Later many researchers began to construct computer simulations of these "box models," which provided more flexibility than traditional mathematical models and also contributed more explicitness and rigor than traditional verbal models. But still the operations performed in each box were generally unstructured and arbitrary.

An early effort at model-based evaluation in this theoretical mode appears in the famous Human Operator Simulator (HOS) system (see Wherry, 1976; Pew, et al., 1977; Strieb & Wherry, 1979; Lane, Strieb, Glenn, & Wherry, 1981; Glenn, Zaklad, & Wherry, 1982; and Harris, Iavecchia, & Bittner, 1988; Harris, Iavecchia, & Dick, 1989). HOS contained a set of *micromodels* for low-level perceptual, cognitive, and motor activities, invoked by task-specific programs written in a special-purpose procedural programming language called HOPROC (Human Operator Procedures language). The micromodels included such things as Hick's and Fitts' Law, formulas for visual recognition time, a model of short-term memory retention, and formulas for calculating the time required for various motor actions such as pushing buttons and walking. The effort was ambitious and the results impressive, but in a real sense, HOS was ahead of its time. The problem was that psychological theory was not well enough developed at the time to provide a sound foundation for such a tool; the developers were basically trying to invent a cognitive architecture good enough for practical application before the concept had been developed in the scientific community. Interestingly, the spirit of the HOPROC language lives on in the independently-developed notations for some forms of GOMS models. In addition, the scientific base for the micromodels was in fact very sparse at the time, and many of them are currently out of date empirically and theoretically. HOS appears to have been subsumed into some commercial modeling systems; for example, a task network version of HOS is available from Micro Analysis and Design, Inc. (http://www.maad.com/), and its micromodels are used in their Integrated Performance Modeling Environment (IPME), as well as CHI System's COGNET/IGEN produced by CHI Systems (http://www.chiinc.com/).

The task network models also originated in this box-model mode of psychology theory, and show it in their lack of psychological constraints; their very generality means they contribute little built-in psychological validity. Even if the HOS micromodels are used, the flexibility of the modeling system means that model-builders themselves must identify the psychological processes and constraints involved in the task being modeled and program them into the model explicitly.

Led by Anderson (1983) and Newell (1990) researchers in human cognition and performance began to construct models using a *cognitive architecture* (see Byrne, this volume). Cognitive architecture parallels the concept of computer architecture: a cognitive architecture specifies a set of

fixed mechanisms, the “hardware”, that comprise the human mind. To construct a model for a specific task, the researcher “programs” the architecture by specifying a psychological strategy for doing the task, the “software”. (Parameter value settings and other auxiliary information might be involved as well.) The architecture provides the coherent theoretical framework within which processes and constraints can be proposed and given an explicit and rigorous definition. Several proposed cognitive architectures exist in the form of computer simulation packages in which programming the architecture is done in the form of production systems, collections of modular if-then rules, that have proved to be an especially good theoretical model of human procedural knowledge. Developing these architectures, and demonstrating their utility, is a continuing research activity (see Byrne, this volume). Not surprisingly, they all have a long way to go before they accurately incorporate even a subset of the human abilities and limitations that appear in an HCI design context.

The psychological validity of a model constructed with a cognitive architecture depends on the validity of both the architecture and the task-specific programming, so it can be difficult to assign credit or blame for success or failure in modeling an individual task. However, the fixed architecture and its associated parameters are supposed to be based on fundamental psychological mechanisms that are required to be invariant across all tasks, while the task-specific programming is free to vary with a particular modeled task. To the extent that the architecture is correct, one should be able to model any task simply by programming the architecture using only task-analytic information and supplying a few task-specific parameters. The value of such architectures lies in this clear division between universal and task-specific features of human cognition; the model builder should be free to focus solely on the specific task and system under design, and let the architecture handle the psychology.

Achieving this goal in psychological research is a daunting challenge. What about the practical sphere? In fact, the role of architectural constraints in some of the extant commercial modeling systems is problematic. The task network models basically have such an abstract representation that there is no straightforward way for architectural assumptions to constrain the modeling system. Once one has opted for representing human activity as a set of arbitrary interconnected task processes, there is no easy way to somehow impose more constrained structure and mechanism on the system. Attempting to do so simply creates more complexity in the modeling problem - the modeler must figure out how to *underuse* the *over-general* capabilities of the system in just the right way.

Another commercial system, COGNET/IGEN (see Zachary, Santarelli, Ryder, & Stokes, 2000, for a recent and relatively complete description), is in the form of a cognitive architecture, but is a very complex one that incorporates a multitude of ideas about human cognition and performance, so many that it appears to be rather hard to understand how it works. However, the essence of a cognitive architecture is the insistence on a small number of fundamental mechanisms that provide a comprehensive and coherent system. For example, several of the scientifically successful cognitive architectures require that all cognitive processing must be expressed in the form of production rules that can include only certain things in their conditions and actions. These rules control all of the other components in the architecture, which in turn have strictly defined and highly limited capabilities. These highly constrained systems have been successful in a wide range of modeling problems, so it is difficult to see why a very complex architecture is a better starting point. Again, to be useful in both scientific and practical prediction, the possible models must be constrained - too many possibilities are not helpful, but harmful.

From the point of view of cognitive architectures and the constraints supplied by the architecture, the modeling approaches described in this chapter, as currently implemented, span the range from little or no architectural content or constraints (the task network systems), to considerable architectural complexity and constraints (the cognitive-architecture systems). GOMS models occupy an intermediate position: they assume a simplified, but definitely constraining, cognitive architecture that allows them to be applied easily by interface designers and still produce usefully accurate results. But at the same time, they are less flexible than the modeling systems at the other extremes.

Modeling cognitive versus perceptual-motor aspects of a design

As pointed out by Byrne (this volume), cognitive architectures have lately begun to incorporate not just proposed cognitive mechanisms, but also proposals for perceptual and motor mechanisms that act as additional sources of constraint on performance. Calling these a "cognitive" architecture is something of a misnomer, since perceptual and motor mechanisms are normally distinguished from cognitive ones. However, including perceptual and motor constraints is actually a critical requirement for modeling user interfaces; this follows from the traditional characterization of human-computer interaction in terms of the interactive cycle (Norman, 1986). The user sees something on the screen if they are looking in the right place and can sense and recognize it, involving the perceptual system and associated motor processes such as eye movements. The user decides what to do, an exclusively cognitive activity, and then carries out the decision by performing motor actions that are determined by the physical interaction devices that are present, and may also involve the perceptual system, such as visual guidance for mouse pointing.

Occasionally, the cognitive processes of deciding what to do next can dominate the perceptual and motor activities. For example, one mouse click might bring up a screen containing a single number, such as a stock price, and the user might think about it for many minutes before simply clicking on a "buy" or "sell" button. But much of the time, users engage in a stream of routine activities that require only relatively simple cognitive processing, and so the perceptual and motor actions take up most of the time and determine most of the task structure. Two implications follow from this thumbnail analysis.

Modeling purely cognitive tasks is generally impractical. Trying to model purely cognitive tasks such as human problem-solving, reasoning, or decision-making processes is extremely difficult because they are so open-ended and unconstrained (see also Landauer, 1995). For example, there are a myriad possible ways in which people could decide to buy or sell a stock, and the nature of the task does not set any substantial or observable constraints on how people might make such decisions - stock decisions are based on everything from gut feel, to transient financial situations, to detailed long-term analysis of market trends and individual corporate strategies. Trying to identify the strategy that a user population will follow in such tasks is not a routine interface design problem, but a scientific research problem, or at least a very difficult task analysis problem. Fortunately, a routine task analysis may produce enough information to allow the designer to *finesse* the problem, that is, side-step it or avoid having to confront it. For example, if one could determine what information the stock-trader needs to make the decisions, and then make that information available in an effective and usable manner, the result will be a highly useful and usable system without having to understand exactly how users make their decisions.

Modeling perceptual-motor activities is critical. A good modeling approach at a minimum must explicitly represent the perceptual and motor operations involved in a task. For most sys-

tems, the perceptual and motor activities involved in interacting with a computer take relatively well-defined amounts of time, are heavily determined by the system design, and frequently dominate the user's activity; leaving them out of the picture means that the resulting model is likely to be seriously inaccurate. For example, if two interface designs differ in how many visual searches or mouse points they logically require to complete a task, the one requiring fewer is almost certainly going to be faster to execute, and will probably have a simpler task structure as well, meaning it will probably be easier to learn and less error-prone. This means that any modeling approach that represents the basic timing and the structure of perceptual and motor activity entailed by an interface is likely to provide a good approximation to the basic usability characteristics of the interface.

This conclusion is both good news and bad news. The good news is that since perceptual-motor activities are relatively easy to model, it can be easy to get fairly reliable and robust model-based evaluation information in many cases. One reason why GOMS models work so well is that they allow the modeler to easily represent perceptual-motor activity fairly completely, with a minimum of complications to represent the cognitive activity. The bad news is that different modeling approaches that include perceptual-motor operations are likely to produce similar results in many situations, making it difficult to tell which approach is the most accurate.

This does not mean continuing the effort to develop modeling systems is futile; rather the point is that trying to verify or compare models in complex tasks is quite difficult due to practical difficulties in both applied and basic research. Despite the considerable effort and expense to collect it, data on actual real-world task performance is often lacking in detail, coverage, and adequate sample sizes, Even in the laboratory, collecting highly precise, detailed, and complete data about task performance is quite difficult, and researchers are typically trapped into using tasks that are artificial, performed by non-expert subjects, or trivial relative to actual tasks. There is no easy or affordable resolution to this dilemma, so the practitioner who seeks to use models must be cautious about claims made by rival modeling approaches, and look first at how they handle perceptual-motor activities. The theorist seeking to improve modeling approaches must be constantly iterating and integrating over both laboratory and actual applications of modeling methods.

The science base must be visible

Even though the modeling methodology encapsulates the constraints provided by psychological theory, it is critical that the psychological assumptions be accessible, justified, and intelligible. An architecture is the best way to do this, because the psychological assumptions are either hard-wired into the modeling system architecture, or are explicitly stated in the task-specific programming supplied by the modeler. The basis for the task-specific programming is the task analysis obtained during the overall design process, and the basis for the architecture is a documented synthesis of the scientific literature.

The importance of the documented synthesis of the scientific literature cannot be overstated. The science of human cognition and performance that is relevant to system design is not at all "finished"; important new results are constantly appearing, and many long-documented phenomena are incompletely understood. Thus any modeling system will have to be updated repeatedly as these theoretical and empirical issues are thrashed out, and it will have to be kept clear which results it incorporates and which it does not.

The commercial modeling tools have seriously lagged behind the scientific literature; while some conservatism would be desirable to damp out some of the volatility in scientific work, the

problem is not just conservatism, but rather obsolescence, as in the case of the micromodels inherited from HOS. Perhaps these systems would still be adequate for practical work but, unfortunately, it is very difficult to get a scientific perspective on their adequacy because they have been neither described nor tested in forums and under ground rules similar to those used for mainstream scientific work in human cognition and performance. Thus they have not been subject to the full presentation, strict review, criticism, and evolution that are characteristic of the cognitive architecture and GOMS model work. The practitioner should therefore greet the claims of commercial modeling system with healthy skepticism, and developers of modeling systems should participate more completely in the open scientific process.

The value of generativity

It is useful if a modeling method is *generative*, meaning that a single model can *generate* predicted human behavior for a whole class of scenarios, where a scenario is defined solely in terms of the sequence of input events or the specifications for a task situation, neither of which specifies the behavior the user is expected to produce. Many familiar modeling methods, including the Keystroke-Level type of GOMS model, are non-generative, in that they start with a specific scenario in which the model builder has specified, usually manually, what the user's actions are supposed to be for the specified inputs. A non-generative model predicts metrics defined only over this particular input-output sequence. To see what the results would be for a different scenario, a whole new model must be constructed (though parts might be duplicated). Since non-generative modeling methods are typically labor intensive, involving a manual assignment of user actions to each input-output event, they tend to sharply limit how many scenarios are considered, which can be very risky in complex or critical design problems.

An example of a sophisticated non-generative modeling method is the CPM-GOMS models developed by Gray, John, & Atwood (1993) to model telephone operator tasks. These models decomposed each task scenario into a set of operations performed by perceptual, cognitive, and motor processors likes those proposed in the Card, Moran, and Newell (1983) Model Human Processor. The sequential dependencies and time durations of these operations were represented with a PERT chart, which then specified the total task time, and whose critical path revealed the processing bottlenecks in task performance. Such models are non-generative in that a different scenario with a different pattern of events requires a different PERT chart to represent the different set of process dependencies. Since there is a chart for each scenario, predicting the time for a new scenario, or different interface design, requires creating a new chart to fit the new sequence of events. However, a new chart can often be assembled from templates or portions of previous charts, saving considerable effort (see John & Kieras, 1996a,b for more detail). In addition, computational tools for modeling based on directly representing the sequential constraints implied by a cognitive architecture are under development and may substantially simplify the construction of such models (e.g. John, Vera, Matessa, Freed, & Remington, 2002; Vera, Howes, McCurdy, & Lewis, 2004).

However, if a model is generative, a single model can produce predicted usability results for any relevant scenario, just like a computer program for calculating the mean of a set of numbers can be applied to any specific set of values. A typical Hierarchical Task Analysis (HTA), (see Annett, Duncan, Stammers, and Gray, 1971; Kirwan & Ainsworth, 1992) results in a generative representation, in that the HTA chart can be followed to perform the task in any subsumed situation. The forms of GOMS models that explicitly represent methods (see John & Kieras, 1996a,b) are also generative. The typical cognitive-architecture model is generative in that it is programmed

to perform the cognitive processes necessary to decide how to respond appropriately to any possible input that might occur in the task. In essence, the model programming expresses the general procedural knowledge required to perform the task, and the architecture, when executing this procedural knowledge, supplies all of the details; the result is that the model responds with a different specific time sequence of actions to different specific situations.

For example, Kieras, Wood, & Meyer (1997) used a cognitive architecture to construct a production-rule model of some of the telephone operator tasks studied by Gray, John, & Atwood (1993). Because the model consisted of a general "program" for doing the tasks, it would behave differently depending on the details of the input events; for example, greeting a customer differently depending on information on the display, and punching function keys and entering data depending on what the customer says and requires. Thus the specific behavior and its time course of the model depend on the specific inputs, in a way expressed by a single set of general procedures.

A generative model is typically more difficult to construct initially, but because it is not bound to a specific scenario, it can be directly applied to a large selection of scenarios to provide a comprehensive analysis of complex tasks. The technique is especially powerful if the model runs as a computer simulation in which there is a *simulated device* that represents how the scenario data results in specific display events and governs how the system will respond to the user, and the *simulated human*, which is the model of how the user will perform the task. The different scenarios are just the input data for the simulation, which produces the predicted behavior for each one. Furthermore, because generative models represent the procedural knowledge of the user explicitly, they readily satisfy the desirable property of models described above: the content of a generative model can be inspected to see how a design "works" and what procedures the user must know and execute.

The role of detail

In the initial presentation above, the reader may have noticed the emphasis on the role of detailed description, both of the user's task and the proposed interface design. Modeling has sometimes been criticized because it appears to be unduly labor intensive. Building a model and using it to obtain predictions may indeed involve substantial detail work. However, working out the details about the user's task and the interface design is, or should be, a necessary part of any interface design approach; the usability lies in the details, not the generalities (Whiteside, Jones, Levy, & Wixon, 1985). If the user's task has not been described in detail, chances are that the task analysis is inadequate and a successful interface will be more difficult to achieve; extra design iterations may be required to discover and correct deficiencies in the original understanding of the user's needs. If the interface designer has not worked out the interface design in detail, the prospects of success are especially poor. The final form of an interface reflects a mass of detailed design decisions; these should have been explicitly made by an interface designer whose focus is on the user, rather than the programmers who happen to write the interface code. So the designer has to develop this detail as part of any successful design effort. In short, using model-based evaluation does not require any more detail than should be available anyway; it just requires that this detail be developed more explicitly and perhaps earlier than is often the case.

Cognitive architectures are committed to detail. Cognitive architecture systems are primarily research systems dedicated to synthesizing and testing basic psychological theory. Because they have a heavy commitment to characterizing the human cognitive architecture in detail, they naturally work at an extremely detailed level. The current cognitive architecture systems differ widely

in the extent to which they incorporate the most potent source of practical constraints, namely perceptual-motor constraints, but at the same time, they are committed to enabling the representation of a comprehensive range of very complex cognitive processes, ranging from multitask performance to problem-solving and learning. Thus these systems are generally very flexible in what cognitive processes they can represent within their otherwise very constrained architectures.

However, the detail has a downside. Cognitive architectures are typically difficult to program, even for simple tasks, and have the further drawback that, as a consequence of their detail, currently unresolved psychological issues can become exposed to the modeler for resolution. For example, the nature of visual short-term memory is rather poorly understood at this time, and no current architecture has an empirically sound representation of it. Using one of the current architectures to model a task in which visual short-term memory appears to be prominent might require many detailed assumptions about how it works and is used in the task, and these assumptions typically cannot be tested within the modeling project itself. One reason why is the difficulty discussed above of getting high-precision data for complex tasks. But the more serious reason is that in a design context, data to test the model is normally not available because there is not yet a system to collect the data with! Less detailed modeling approaches such as GOMS may not be any more accurate, but they at least have the virtue of not side-tracking the modeler into time-consuming detailed guesswork or speculation about fundamental issues. See Kieras (2005b) for more discussion.

Task networks can be used before detailed design. Although model-based evaluation works best for detailed designs, the task network modeling techniques were developed to assist in design stages before detailed design, especially for complex military systems. For example, task network modeling was used to determine how many human operators would be required to properly man a new combat helicopter. Too many operators drastically increases the cost and size of the aircraft; too few means the helicopter could not be operated successfully or safely. Thus questions at these stages of design are what capacity (in terms of the number of people or machines) is needed to handle the workload, and what kinds of work needs to be performed by the each person or machine.

In outline, these early design stages involve first selecting a *mission profile*, essentially a high-level scenario that describes what the system and its operators must accomplish in a typical mission, then developing a basic *functional analysis* that determines the functions (large-scale operations) that must be performed to accomplish the mission, and what their interactions and dependencies are. Then the candidate high-level design consists of a tentative *function allocation* to determine which human operator or machine will perform each function (see Beevis, Bost, Doering, Nordo, Oberman, Papin, Schuffel, & Streets, 1992). The task network model can then be set up to include the tasks and their dependencies, and simulations run to determine execution times and compute workload metrics based on the workload characteristics of each task.

Clearly, entertaining detailed designs for each operator's controls or workstation is pointless until such a high-level analysis determines how many operators there will be and what tasks they are responsible for. Note that the cognitive-architecture and GOMS models are inherently limited to predicting performance in detailed designs, because their basic logic is to use the exact sequence of activities required in a task to determine the sequence of primitive operations. However, as will be discussed below, recent work with *high-level GOMS models* suggests an alternative approach in which a GOMS model using abstract or high-level operators to interact with the device can be developed first, and then elaborated into a model for a specific interface as the design takes

shape. But at this time, for high-level design modeling, the task-network models appear to be the best, or only, choice.

However, there are limitations that must be clearly understood. The ability of the task network models to represent a design at these earliest stages is a direct consequence of the fact that these modeling methods do not have any detailed mechanisms or constraints for representing human cognition and performance. Recall that the tasks in the network can consist of any arbitrary process whose execution characteristics can follow any desired distribution. Thus the tasks and their parameters can be freely chosen without any regard to how a human will be actually do them in the final version of the system. Hence this early-design capability is a result of a lack of theoretical content in the modeling system itself.

While the choice of tasks in a network model is based on a task analysis, the time distribution parameters are more problematic - how does one estimate the time required for a human to perform a process specified only in the most general terms? One way is to rely on empirical measurements of similar tasks performed in similar systems, but this requires that the new system must be similar to a previous system not only at the task-function level, but at least roughly at the level of design details.

Given the difficulty of arriving at task parameter estimates rigorously, a commonly applied technique is to ask a subject matter expert to supply *subjective estimates* of task time means and standard deviations and workload parameters. When used in this way, a task-network model is essentially a mathematically straightforward way to start with estimates of individual subtask performance, with no restrictions on the origin or quality of these estimates, and then to combine them to arrive at performance estimates for the entire task and system.

Clearly, basing major design decisions on an aggregation of mere subjective estimates is hardly ideal, but as long as a detailed design or preexisting system is not available, there is really no alternative to guide early design. In the absence of such analyses, system developers would have to choose an early design based on “gut feel” about the entire design, which is surely more dangerous.

Note that if there is a detailed design available, the task-network modeler could decompose the task structure down to a fine enough level to make use of basic human performance parameters, similar to those used in the cognitive-architecture and GOMS models. For example, some commercial tools supply the HOS micromodels. However, it is hard to see the advantage in using task network models for detailed design. The networks and their supplementary executable code do not seem to be a superior way to represent task procedures compared to the computer-program-like format of some GOMS models or the highly flexible and modular structure of production systems.

Another option would be to construct GOMS or cognitive architecture models to produce time estimates for the individual tasks, and use these in the network model instead of subjective estimates. This might be useful if only part of the design has been detailed, but otherwise, staying with a single modeling approach would surely be simpler. If one believes that interface usability is mostly a matter of getting the details right, along the lines originally argued by Whiteside, Jones, Levy, & Wixon (1985) and verified by many experiences in user testing, modeling approaches that naturally and conveniently work at a detailed design level will be especially valuable.

Practical Issues in Applying a Model-based Evaluation Technique

Once a model-based evaluation technique is chosen, there are some practical issues that arise in seeking to apply the technique to a particular user interface design situation. This section presents several of these issues.

Creating the Simulated Device

As mentioned above, the basic structure of a model used for evaluation is that a simulated human representing the user is interacting with a simulated device that represents the system under design. In a parallel with Norman's interactive cycle, the simulated human receives simulated visual and auditory input from the simulated device, and responds with simulated actions that provide input to the simulated device, which can then respond with different visual and auditory inputs to the human. Depending on the level of generativity and fidelity of the model, the simulated device can range from being a dummy device that does nothing in response to the simulated human interaction, to a highly detailed simulation of the device interface and functionality. An example of a dummy device is the device that is assumed in the Keystroke-Level Model, which is not at all explicitly defined; the modeler simply assumes that a specific sequence of user actions will result in the device doing the correct thing. At the other extreme are models such as the ones used by Kieras & Santoro (2004), which actually implemented significant portions of the logical functionality of a complex radar workstation and the domain of moving aircraft and ships. In modeling situations where a generative model is called for, namely a complex task domain with multiple or lengthy detailed scenarios, a fully simulated device is the most convenient way to ensure that a simulated human is in fact performing the task correctly, and to easily work with more than one scenario for the task situation.

It is important to realize that the simulated device is not required to implement the actual interface whose design is being evaluated. Rather, it suffices to produce abstract psychological inputs to the simulated human. For example, if a red circle is supposed to appear on the screen, the simulated device can merely signal the simulated human with an abstract description that an object has appeared at certain (x, y) coordinates that has a "shape" of "circle", and a "color" of "red." It is not at all necessary for the simulated device to actually produce a human-viewable graphical display containing a circular red area at a certain position.

A lesson learned by Kieras and Santoro (2004) was that the effort required to construct even such an abstract simulated device in a complex domain is a major part of the modeling effort, and is more difficult in some ways than constructing the models of the simulated users! Clearly, to some extent, this effort is redundant with the effort required to develop the actual system, and so can undermine the rationale for modeling early the design and development process.

A common response to this problem is to seek to connect the cognitive architecture directly to an intact application or system prototype that plays the role of the simulated device, in short replacing the simulated device with an actual device. Work such as St. Amant and Reidl (2001) provides a pathway for interfacing to an intact application: the technique is basically to use the existing API of the application platform (e.g. Windows) to capture the screen bitmap and run visual object recognition algorithms on it to produce the description of the visual inputs to the simulated human, and outputs from the simulated human can be directly supplied to the platform to

produce keyboard input or to control the cursor position. Even for the limited domain of Windows applications using the standard GUI objects, this is technically challenging, but quite feasible.

A less ambitious approach is instead of connecting to an intact application program, to instrument a prototype version of the interface, so that for example, when the prototype causes a certain object to appear on the screen, the simulated human is supplied with the visual input description. Given the considerable variety in how GUIs are implemented, this solution is not very general, but does have interesting solutions if the application prototype is programmed in Java, html, or similar cross-platform languages or general-purpose tools that can be used for prototyping. A good example is the html-based modeling tool described in John, Prevas, Salvucci, and Koedinger (2004).

However, both of these methods of coupling a user model to an application suffer from an easily overlooked limitation: The time when modeling is most useful is *early* in design, *before* the system has been prototyped. Thus because coupling to a prototype or an application can only happen late in the development process or after development, these approaches come too late to provide the most benefit of model-based evaluation.

Thus multiple approaches for creating the simulated device are both possible and needed: If the design questions can be answered with evaluation techniques such as the Keystroke-Level Model, then no simulated device is needed at all. If the model is for an existing application, coupling to the intact application is clearly the best solution. If a prototype is going to be constructed at this point in the design process anyway, using it as the simulated device is the best solution. But in the potentially most useful case, the simulated device must be created before any prototype or final application, making the fewest possible commitments to prototyping or coding effort; this requires constructing a simulated device from scratch, stripped down to the bare minimum necessary to allow a candidate design to interact with the simulated user. The next sections provide some advice on this process.

How to simplify the simulated device. Distinguish between device behavior that is relevant to the modeling effort and that which is not. Basically, if the simulated human will not use or respond to certain behaviors of the simulated device, then the simulated device does not need to produce those behaviors. A similar argument applies to the amount of detail in the behavior. Of course, as the interface design is elaborated, the simulated device may need to cover more aspects of the task. Good programming techniques will make it easy to extend the simulated device as needed; a good programmer on the project is a definite asset.

Distinguish between what the device has to provide to the simulated human, and what would be a convenience to the modeler. That is, while the device can be supplied with abstract descriptions, an actual graphical display of what the simulated device is displaying can be a very useful tool for the modeler in monitoring, debugging, or demonstrating the model. A very crude general-purpose display module that shows what the simulated human "sees" will suffice for these purposes, and can be reasonably easy to provide in a form that is re-usable for a variety of simulation projects. However, developing this handy display should be recognized as an optional convenience, rather than an essential part of the simulated device.

Since programming the simulated device can be a significant programming effort, an attractive simplification would be a programming language that is specialized for describing abstract device behavior. Clearly, using such a language could be valuable if the modeling system already provides it and it is adequate for the purpose, especially if the device programming language can generates a prototype for the interface that can be directly coupled to the simulated human, moving

the whole process along rapidly. An extensive project involving many different but similar interface designs would profit especially if the language matches the problem domain well.

However, in less than ideal situations, a specialized language is unlikely to be an advantage. The reason is that to cover the full span of devices that might need to be simulated, the device programming language will have to include a full set of general programming language facilities. For example, to handle the Kieras & Santoro (2004) domain, trigonometric functions are needed to calculate courses and trajectories, and containers of complex objects are required to keep track of the separate aircraft and their properties. Thus specialized languages will inevitably have to include most of the same feature set as general-purpose programming languages, meaning that the developers of modeling systems will have to develop, document, maintain, and support with editors and debuggers a full-fledged programming language. This takes effort away from the functions that are unique to human performance modeling systems, such as ensuring that the psychology is correctly represented. In addition, the modeler will also have to expend the time and effort necessary to learn a specialized language whose complexity is similar to a general-purpose programming language, also taking effort away from unique aspects of the modeling effort.

A better choice would be to provide for the device to be programmed easily in a standard general-purpose programming language that modelers can (or should) know anyway, allowing re-use of not just the modeler's skills, but existing programming tools and education resources as well. A well-designed modeling system can ensure that a minimum of system-specific knowledge must be acquired before coding can begin.

Identifying the Task Strategy

A task analysis does not necessarily specify a task strategy. Human performance in a task is determined by: (1) the logical requirements of the task - what the human is supposed to accomplish, as determined by a task analysis; (2) the human cognitive architecture - the basic mechanisms available to produce behavior; and (3) a specific strategy for doing the task - given the task requirements, and the architecture, what should be done in what order and at what time to complete the task. Thus to construct a model for doing the task, one must first understand the task, then choose an architecture, and then choose a strategy to specify how the architecture will be used to accomplish the task. Identifying this strategy is the critical prerequisite for constructing a model.

Normal task analysis methods, such as those described in sources such as (Kirwan & Ainsworth, 1992; Beevis, et al, 1992; Diaper & Stanton, 2004), do not necessarily identify the exact sequence of actions to perform with the interface under design, and they rarely specify the timing of actions. For example, anyone who has made coffee with a home coffee maker knows that there are certain constraints that must be met, but there is still considerable variation in the sequence and time in which the individual required steps could be performed. In fact, there can be variation on how the activity is organized; for example, one strategy is to use the visible state of the coffee maker as an external memory to determine which actions should be performed next (Larkin, 1989). A normal task analysis will not identify these variations. But even further, task analysis will not necessarily identify how any trade-offs should be decided, even as basic as speed vs. accuracy, much less more global problems such as managing workload, dealing with multiple task priorities, and so forth.

Thus to model a human performing in such situations, some additional information beyond a normal task analysis has to be added, namely the specific task strategies that are used to accom-

plish the tasks. Conversely, the performance that a human can produce in a task can vary over a wide range depending on the specific strategy that is used, and this is true over the range of tasks from elementary psychology laboratory tasks to highly complex real-world tasks (Kieras & Meyer, 2000). This raises a general problem: Given that we have a model that predicts performance on an interface design, how do we distinguish the effects of the interface design from the effects of the particular strategy for using the interface? Not only does this apply to the model performance, but also to the human's performance. It has always been clear that clever and experienced users could get a lot out of a poorly designed system, and even a reasonably well-designed powerful system could be seriously under-used (Bhavnani & John, 1996). How can we predict performance without knowing the actual task strategy, and how does our model's task strategy relate to the actual user's task strategy?

Difficulties in identifying task strategy. The state of the art in cognitive modeling research for identifying a task strategy is to choose a candidate intuitively, build the model using the strategy, evaluate the goodness of fit to data, and then choose a better strategy and repeat until a satisfactory fit is obtained. If there is adequate detail in the data, such as the sequence of activities, it might be possible to make good initial guesses at the task strategy and then revise these through the modeling process. This iterative refinement process is known to be very slow, but more seriously, in system design we normally do not have data to fit a model to - this is what the modeling is supposed to replace. The task strategy has to be chosen in the absence of such data.

Another approach is to get the task strategy by knowledge engineering techniques with existing task performers or other sources such as training materials. As will be argued below, it is especially important to identify the best (or at least a good) strategy for doing the task. But a good strategy for doing a task is often not obvious, even to experts. Even highly experienced people do not always know or use the best procedures; even the trainers may not know them, and it is common to discover that procedural training materials present suboptimal methods. Finally, and again most importantly, if the system is new, there are no experts or training materials to consult to see how it is used.

A heuristic: Model what users* should *do. Given the obstacles to identifying task strategies, how do we find out what strategies users will follow in using the system under development? The short answer is that it is too hard to find out within the constraints of a design process. Instead, start from the design goals that the system is supposed to meet, and assume that *users will be using the system like it is supposed to be used*. For example, if the system provides a feature that is supposed to allow the user to perform a certain task easily, assume that the simulated user will use that feature in the intended fashion. This is essentially a best-case analysis of the ability of the user to make use of the interface design. If the usability under these conditions is too low, then it would certainly be inadequate when used by actual users! It is a separate issue whether the users can or will use the system in this intended way, and the failures can be dramatic and serious (Bhavnani & John, 1996). Whether users use a system in the intended way depends on several factors: problems in the learnability of the design, which some models (see John & Kieras, 1996a,b) can predict; the quality of the training materials, which also can be improved with modeling (see John & Kieras, 1996a,b); and perhaps most importantly, matters beyond the scope of model-based evaluation, such as whether users have the opportunity or incentive to take full advantage of the system. Finally, there is no point to trying to improve a system under the assumption that users will ignore the capabilities that it provides. Thus, in terms of choosing a task strategy to model for design evaluation, the most effective approach is to assume that *the design will be used as intended*. Not only will this strategy be the easiest to identify and implement, but it is also most directly relevant to evaluating the design!

Within this basic strategy, there is another range of variation, which is whether the user attempts to perform at the highest possible level, or simply satisfices by performing at some adequate level. That is, people can use clever low-level strategies (what Gray & Boehm-Davis (2000) termed microstrategies), to greatly improve performance. For example, if the task is to classify "blips" on a radar display, the user can speed up considerably by looking for the next blip to inspect while still hitting the keys to respond to the previous one (Kieras, Meyer, & Ballas, 2001). On the other hand, the user is performing reasonably if they finish each blip before going on to the next one. Kieras & Meyer (2000) pointed out that in a variety of even elementary laboratory tasks, subjects do not always adopt high-performance strategies, even when large improvements would result; they are optional, not mandatory. So even if we are willing to assume that the design will be used as intended, how do we know whether the actual users will be going "all out" with a clever strategy versus just "getting the job done?" Again, the short answer is that it is too difficult to find out, especially in a design process where the system does not yet exist.

In response to this quandary, Kieras and Meyer (2000) proposed the *bracketing heuristic*. Construct a base model in which the user performs the task in a straightforward way using the interface as designed, but without any special strategic optimizations, a *slowest-reasonable* model. Derive from this a *fastest-possible* model that performs the task at the maximum performance level allowed by the cognitive architecture used for the model. The two models should bracket the actual user's future performance. If both models produce adequate performance, then the design should be adequate; if both produce inadequate performance, then the design needs to be improved. If the slowest-reasonable model is inadequate, but the fastest-possible model is acceptable, boosting the level of training or perhaps motivation of the user might result in satisfactory performance, although clearly improving the design would be a more robust solution.

Concerns over Model Validity

Can You Believe the Model? Suppose a model implies critical design choices. Should you follow them? A poor response is to build and test prototypes just as if no modeling had been done. It could be argued that the modeling might have clarified the situation, but the purpose of model-based evaluation is to reduce the amount of prototyping and user testing required to refine a design. So this response under-utilizes the approach.

A better response to the situation is to understand how the model implies the design choices - what aspects of the model are contributing to the outcome? This can be done by profiling the model processing and analyzing the model structure. If the critical aspects of the model are known to be valid and appear to be properly represented, then the model results should be accepted. For example, perhaps one design is slower than the other simply because it turns out that more navigation through menus is required; the model processes involved are relatively simple and adequately validated in the literature. However, if the relevant aspects of the model are problematic, the result needs further study. For example, suppose the model for the better of two designs differs from the poorer design in assuming that the user can remember all of the information about previously inspected screen objects, and so does not need to search the screen again when the information is needed later. Because the bounds on visual memory are unclear, as discussed above, the modeling architecture might not enforce any bounds. Thus the modeling result is suspicious for this reason, and the bounds might be much smaller than the model assumes. The modeler could then perform a sensitivity analysis to reveal how much the design choice might be affected by the problematic assumption. For example, the model could be modified to vary the number of previous inspected objects that could be remembered. At what value does this parameter change which design is pre-

dicted to be better? If the decision is not very sensitive to this parameter (for example, because the effects are minor compared to the total time involved, or there are other improvements in the better design), then choosing the better design is a reasonably safe decision.

If the decision turns out to be sensitive to the problematic model assumption, then the situation becomes quite difficult. One possible solution is to remove the problematic aspect of the model by changing it in the direction of less human capability; this is both a conservative strategy and might be appropriate if the user will be under stress as well. But if data or theory is available that resolves the issue, the modeler can go beyond the normal model-based process and modify the model or architecture to incorporate the more accurate psychological information.

Should you validate the model? Remember that testing with real users must be done sometime in the development process, because the models do not cover all of the design issues, and are only approximate where they do apply and, like any analytic method, can be misapplied. The model can thus be validated after use by comparing the final user test results to the model predictions; this will reveal problems in the accuracy of the model and its application to the design; any modeling mistakes and design errors can then be corrected for the future.

However, should special data to validate the model be collected *prior* to using it to guide the design? While it would seem to be a good idea to validate a model before using it, the answer really should be *no* because validation is not supposed to be a normal part of using a predictive model of human performance. The whole idea of model-based evaluation is to avoid data collection during design. Only the developers of the modeling methodology are supposed to be concerned about the validity, not the user of the methodology.

There are a couple of special cases about data collection that need discussion. One is data collection to provide basic parameter values for modeling, such as how long it takes the user to input characters with a novel device. If the parameters concern low-level processes, the data collection is independent of a specific design and will be generally useful for many different modeling applications. The second special case is data collection to support modeling how an *existing* system is *actually* being used. Such a model cannot be constructed *a priori,* but rather must be based on data about how actual users interact with the actual system. Due to the uncertainties involved in constructing a model based on human behavior, the model will have to be validated with a suitably complete and reliable data set before it can be taken as a usefully accurate model. This purpose of modeling is very different from the model-based evaluation approach presented in this chapter: instead of serving as a guide for designing a new system and a surrogate for user testing, the model is an explanation and characterization of observed behavior; it might serve as a guide for a new design, but only in the sense of characterizing the current situation that we want to improve upon. The model itself will not directly apply to the new design. In short, modeling the actual use of an existing system has very different methods, goals, and applications from modeling the usage of a system being designed.

But if in spite of all of the above considerations, the validity of the model is in question, then it needs to be realized that a data set adequate for a scientifically sound test of validity must be much more controlled and detailed than normal user testing data, and can be very difficult to collect in the context of a development project. Furthermore, if the resources are available to do such data collection and analysis before the final stages of development, what function is served by modeling? If the model validity would be in doubt, couldn't the data collection resources be better devoted to user testing?

To elaborate on the difficulty of collecting adequate validation data, while all would agree that a model is almost certainly incorrect at some level, it is often mistakenly assumed that collecting empirical data on human performance in complex tasks is a royal road to certainty. Rather, as pointed out in Kieras & Santoro (2004), complex real-world tasks involve subtle user strategies, team interactions, influences of background knowledge, and the specifics of the scenarios. Such experiments are extremely slow and expensive to conduct, even with small samples, where the reliability of the results then comes into question. Clearly, it is not practical to run experiments using many scenarios, every reasonable design variation, every candidate team organization, and ample numbers of subjects. Furthermore, even for a well-chosen subset of these possibilities, it may be difficult to understand why people did what they did in the tasks - asking them is usually ambiguous at best, and their strategies might be idiosyncratic. Thus, the reliability, generalizability, and even the meaning of the data can be difficult to determine. In fact, it can be difficult to ensure that the model and the experiment are even relevant to each other. For example (see Kieras & Santoro, 2004), if the model is based on what users *should* do in the task and the test users don't follow the strategy that the model follows, then the failure of the model to behave the same way as the test users is actually irrelevant - the data is "wrong", not the model! Thus, even if deemed appropriate, attempting to validate a model of a complex task is likely to be impractically difficult for a normal development process.

Summary: Assessing model validity. Instead of collecting data to validate the model, assess its validity in terms of whether it meets the following basic requirements: (1) Is the model strategy based on an analysis of what users *should* do? If not, it is a poor choice to inform the design of a new system. (2) Is it likely that users can or will follow the same strategy as the model? If not, then the model is irrelevant - either the model was misconstructed or the design is fundamentally wrong. (3) Are the assumptions about human abilities in the model plausible? If not, see the earlier suggestions. If the answer to all three questions is yes, then the model results can be accepted as useful guidance for the design decisions without special validation efforts. Of course, it needs to be kept in mind that the model might be seriously incorrect, but modeling is not supposed to be perfect; it suffices merely that it help a design process.

GOMS Models: A Ready-to-Use Approach

As summarized above, GOMS is an approach to describing the knowledge of procedures that a user must have in order to operate a system. The different types of GOMS models differ in the specifics of how the methods and sequences of operators are represented. The aforementioned CPM-GOMS model represents a specific sequence of activity in terms of the cognitive, perceptual, and motor operators performed in the context of a simple model of human information processing. At the other extreme of detail, the Keystroke-Level Model (Card, Moran, & Newell, 1980) is likewise based on a specific sequence of activities, but these are limited to the overt *keystroke-level* operators (i.e. easily observable actions at the level of keystrokes, mouse moves, finding something on the screen, turning a page, and so forth). The task execution time can be predicted by simply looking up a standardized time estimate for each operator and then summing the times. The Keystroke-Level Model has a long string of successes to its credit (see John & Kieras, 1996a). Without a doubt, if the design question involves which alternative design is faster in fairly simple situations, there is no excuse for measuring or guessing when a few simple calculations will produce a usefully accurate answer.

It is easy to generalize the Keystroke-Level Model somewhat to apply to more than one specific sequence of operators. For example, if the scenario calls for typing in some variable number of strings of text, the model can be parameterized by the number of strings and their length. However, if the situation calls for complex branching or iteration, and clearly involves some kind of hierarchy of task procedures, such sequence-based models become quite awkward and a more generative form of model is required.

The generative forms of GOMS models are those in which the procedural knowledge is represented in a form resembling an ordinary computer programming language, and are written in a fairly general sort of way. This form of GOMS model can be applied to many conventional desktop computing interface design situations. It was originally presented in Card, Moran, and Newell (1983, Ch. 5), and further developed by Kieras, Polson, and Bovair (Kieras & Polson, 1985; Polson, 1987; Bovair, Kieras, & Polson, 1990), who provided a translation between GOMS models and the production-rule representations popular in several cognitive architectures and demonstrated how these models could be used to predict learning and execution times. Kieras (1988, 1997) proposed a structured-natural-language notation, NGOMSL ("Natural" GOMS Language), which preserved the empirical content of the production-rule representation, but resembled a conventional procedural programming language. This notation was later formalized into a fully executable form, GOMSL (GOMS Language), for use in computer simulation tools that implement a simplified cognitive architecture that incorporates a simple hierarchical-sequential flow of control (Kieras, Wood, Abotel, & Hornof, 1995; Kieras, 2005a). This tool has been applied to modeling team tasks (e.g. Kieras & Santoro, 2004), and extended to provide analysis of error recovery methods supported by error-source heuristics (Wood, 2000). See Baumeister, John, and Byrne (2000) for a survey of computer tools for GOMS modeling.

Continuing the analogy with conventional computer programming languages, in generative GOMS models, the operators are like the primitive operations in a programming language; methods are like functions or subroutines that are called to accomplish a particular goal, with individual steps or statements containing the operators, which are executed one at a time, as in a conventional programming language. Methods can assert a sub-goal, which amounts to a call of a sub-method, in a conventional hierarchical flow of control. When procedural knowledge is represented explicitly in this way, and in a format that enforces a uniform "grain size" of the operators and steps in a method, then there are characteristics of the representation that relate to usability metrics in straightforward ways.

For example, the collection of methods represents "how to use the system" to accomplish goals. If a system requires a large number of lengthy methods, then it will be hard to learn; there is literally more required knowledge than for a system with a smaller number or simpler methods. If the methods for similar goals are similar, or in fact the same method can be used to accomplish different, but similar, goals, then the system is "consistent" in a certain, easily characterized sense: in a procedurally consistent system, fewer methods, or unique steps in methods, must be learned to cover a set of goals compared to an inconsistent system, and so it is easier to learn. One can literally count the amount of overlap between the methods to measure procedural consistency.

Finally, by starting with a goal and the information about the specific situation, one can follow the sequence of operators specified by the methods and sub-methods to accomplish the goal. This generates the sequence of operators required to accomplish the goal under that specific situation; if the methods were written to be adequately general, they should suffice to generate the correct sequence of operators for any relevant task situation. The times for the operators in the trace can be

summed, as in the Keystroke-Level Model, to obtain a predicted execution time. Details of the timing can be examined, or "profiled" to see where the processing bottlenecks are.

Why GOMS Models Work

The reasons why GOMS Models have useful predictive and heuristic power in interface design can be summarized under three principles: The *rationality principle* (cf. Card, Moran, & Newell, 1983) asserts that humans attempt to be efficient given the constraints on their knowledge, ability, and the task situation. Generally, when people attempt to accomplish a goal with a computer system, they do not engage in behavior that they know is irrelevant or superfluous - they are focused on getting the job done. Although they might perform suboptimally due to poor training (see Bhavnani & John, 1996) they generally try to work as efficiently as they know how, given the system they are working with. How they accomplish a goal depends on the design of the system and its interface - for example, in a word-processing system, there are only a certain number of sensible ways to delete a word, and the user has some basis for choosing between these that minimizes effort along some dimension. Between these two sets of constraints - the user's desire to get the job done easily and efficiently, and the computer system's design - there is considerable constraint on the possible user actions. This means that we can predict user behavior and performance at a useful level of accuracy *just from the design of the system and an analysis of the user's task goals and situation*. A GOMS model is one way of combining this information to produce predicted performance.

Procedural primacy is the claim that regardless of what else is involved in using a system, at some level the user must infer, learn, and execute procedures in order to accomplish goals using the system. That is, computers are not used purely passively - the user has to do something with them, and this activity takes the form of a procedure that the user must acquire and execute. Note that even display-only systems still require some procedural knowledge for visual search - for example, making use of the flight status displays at an airport requires choosing and following some procedure for finding one's flight and extracting the desired information - different airlines use different display organizations, some of which are probably more usable than others. Because the user must always acquire and follow procedures, the complexity of the procedures entailed by an interface design is therefore related to the difficulty of using the interface. While other aspects of usability are important, the procedural aspect is always present. Therefore, analyzing the procedural requirements of an interface design with a technique such as GOMS will provide critical information on the usability of the design.

Explicit representation refers to the fact that any attempt to assess something benefits from being explicit and clear and relying on some form of written formalized expression. Thus all task analysis techniques (Kirwan & Ainsworth, 1992; Beevis et al., 1992; Diaper & Stanton, 2004) involve some way to express aspects of a user's task. Likewise, capturing the procedural implications of an interface design will benefit from representing the procedures explicitly in a form that allows them to be inspected and manipulated. Hence GOMS models involve writing out user procedures in a complete, accurate, and detailed form. By doing so in a specified format, it becomes possible to define metrics over the representation (e.g. counting the number of statements) that can be calibrated against empirical measurements to provide predictions of usability. Moreover, by making user procedures explicit, the designer can then apply the same kinds of intuition and heuristics used in the design of software: clumsy, convoluted, inconsistent, and "ugly" user procedures can often be spotted and corrected just like poorly written computer code. Thus by writing out user procedures in a notation like GOMS, the designer can often detect and correct usability

problems without even performing the calculations. This approach can be applied immediately after a task analysis to help choose the functionality behind the interface, as well as to help in the initial design decisions (Kieras, 2004). Work in progress consists of adding high-level operators to a GOMSL and GLEAN (see Kieras, 2005b) to directly support a "seamless" transition of computational modeling from the task and functionality level of analysis down to detailed design.

Limitations of GOMS Models

GOMS models address only the procedural aspects of a computer interface design. This means that they do not address a variety of non-procedural aspects of usability, such as the readability of displayed text, the discriminability of color codes, or memorability of command strings. Fortunately, these properties of usability are directly addressed by standard methods in human factors.

Within the procedural aspect, user activity can be divided into the open-ended "creative" parts of the task, such as composing the content of a document, or thinking of the concept for the design of an electronic circuit on the one hand, and the routine parts of the task on the other, which consist of simply manipulating the computer to accept the information that the user has created, and then to supply new information that the user needs. For example, the creator of a document has to input specific strings of words in the computer, rearrange them, format them, spell check them, and then print them out. The creator of an electronic device design has to specify the circuit and its components to a CAD system and then obtain measures of its performance. If the user is reasonably skilled, these activities take the form of executing routine procedures involving little or no creativity.

The bulk of time spent working with a computer is in this routine activity, and the goal of computer system design should be to minimize the difficulty and time cost of this routine activity so as to free up time and energy for the creative activity. GOMS models are easy to construct for the routine parts of a task, because, as described above, the user's procedures are constrained by the task requirements and the design of the system, and these models can then be used to improve the ability of the system to support the user. However, the creative parts of task activity are purely cognitive tasks and, as discussed above, attempting to formulate a GOMS model for them is highly speculative at best, and would generally be impractical. Applying GOMS thus takes some task analysis skill to identify and separate the creative and routine procedural portions of the user's overall task situation.

Finally, it is important to recognize that while a GOMS model is often a useful way to express the results of a task analysis, similar to the popular Hierarchical Task Analysis technique (Annett, Duncan, Stammers, & Gray, 1971; Kirwan & Ainsworth, 1992), building a GOMS model does not "do" a task analysis. The designer must first engage in task analysis work to understand the user's task before a GOMS model for the task can be constructed. In particular, identifying the top-level goals of the user and selecting relevant task scenarios are all logically prior to constructing a GOMS model.

Concluding Recommendations

• If you need to predict the performance of a system prior to detailed design when overall system structure and functions are being considered, use a task network model.

• If you are developing a detailed design and want immediate intuitive feedback on how well it supports the user's tasks, write out and inspect a high-level or informal GOMS model for the user procedures while you are making the design decisions.

• If your design criterion is the execution speed for a discrete selected task, use a Keystroke-Level model.

• If your design criteria include the learnability, consistency, or execution speed of a whole set of task procedures, use a generative GOMS model such as CMN-GOMS or NGOMSL. If numerous or complex task scenarios are involved, use a GOMS model simulation system.

• If the design issues hinge on understanding detailed or subtle interactions of human cognitive, perceptual, and motor processing and their effect on execution speed, and only a few scenarios need to be analyzed, use a CPM-GOMS model.

• If the resources for a research-level activity are available, and a detailed analysis is needed of the cognitive, perceptual, and motor interactions for a complex task or many task scenarios, use a model built with the simplest cognitive architecture that incorporates the relevant scientific phenomena.

References

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

Annett, J., Duncan, K.D., Stammers, R.B., & Gray, M.J. (1971). *Task analysis*. London: Her Majesty's Stationery Office.

Baumeister, L.K, John, B.E., Byrne, M.D. (2000). A comparison of tools for building GOMS models. In *Proceedings of CHI 2000*. New York: ACM.

Bhavnani, S. K., & John, B. E. (1996). Exploring the unrealized potential of computer-aided drafting. In *Proceedings of the CHI'96 Conference on Human Factors in Computing Systems,* ACM, New York, 1996.

Beevis, D., Bost, R., Doering, B., Nordo, E., Oberman, F., Papin, J-P., I., H. Schuffel, & Streets, D. 1992. Analysis techniques for man-machine system design. (Report AC/243(P8)TR/7). Brussels, Belgium: Defense Research Group, NATO HQ.

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

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

Card, S., Moran, T. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, New Jersey: Erlbaum.

Chubb, G. P. (1981). SAINT, a digital simulation language for the study of manned systems. In J. Moraal, & K. F. Kraas, (Eds.), *Manned system design* (pp. 153-179). New York: Plenum.

Diaper, D. & Stanton, N.A. (Eds.) (2004), T*he handbook of task analysis for human-computer interaction*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Elkind, J. I., Card, S. K., Hochberg, J., & Huey, B. M. (Eds.) (1989). *Human performance models for computer-aided engineering*. Committee on Human Factors, National Research Council. Washington: National Academy Press.

Glenn, F.A., Zaklad, A.L., & Wherry, R.J. (1982). Human operator simulation in the cognitive domain. In *Proceedings of the Human Factors Society*, 964-969.

Gray, W. D., & Boehm-Davis, D. A. (2000). Milliseconds Matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experiment Psychology: Applied*, 6(4), 322-335.

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, 237-209.

Harris, R.M., Iavecchia, H.P., & Bittner, A.C. (1988). Everything you always wanted to know about HOS micromodels but were afraid to ask. In *Proceedings of the Human Factors Society*. 1051- 1055.

Harris, R., Iavecchia, H.P, & Dick, A.O. (1989). The Human Operator Simulator (HOS-IV). In G.R. McMillan, D. Beevis, E. Salas, M.H. Strub, R. Sutton, & L. Van Breda (Eds.). *Applications of human performance models to system design* (pp. 275-280). New York: Plenum Press.

John, B. E., & Kieras, D. E. (1996a). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction*, **3**, 287-319.

John, B. E., & Kieras, D. E. (1996b). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, **3**, 320-351.

John, B., Prevas, K., Salvucci, D., & Koedinger, K. (2004). Predictive Human Performance Modeling Made Easy. In *Proceedings of CHI 2004*. New York: ACM.

John, B., Vera, A., Matessa, M., Freed, M., & Remington, R. (2002). Automating CPM-GOMS. In *Proceedings of CHI 2002*. New York: ACM.

Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of Human–Computer Interaction* (pp. 135–158). Amsterdam: North–Holland Elsevier.

Kieras, D. E. (1997). A Guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, and P. Prabhu (Eds.), *Handbook of human-computer interaction*. (Second Edition) (pp. 733-766). Amsterdam: North-Holland.

Kieras, D. E. (2004). Task analysis and the design of functionality. In A. Tucker (Ed.) *The Computer Science and Engineering Handbook (2nd Ed.)* (pp. 46-1 - 46-25). Boca Raton, CRC Inc.

Kieras, D.E. (2005a). *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN4*. Document available at ftp://www.eecs.umich.edu/people/kieras

Kieras, D. (2005b). Fidelity issues in cognitive architectures for HCI modeling: Be careful what you wish for. Paper presented at the 11th International Conference on Human Computer Interaction (HCII 2005). Las Vegas, July 22-27. Proceedings published as CD-ROM.

Kieras, D. E., & Meyer, D. E. (2000). The role of cognitive task analysis in the application of predictive models of human performance. In J. M. C. Schraagen, S. E. Chipman, & V. L. Shalin (Eds.), Cognitive task analysis. Mahwah, NJ: Lawrence Erlbaum, 2000. 237-260.

Kieras, D., Meyer, D., & Ballas, J. (2001). Towards demystification of direct manipulation: Cognitive modeling charts the gulf of execution. In *Proceedings of CHI 2001*. New York: ACM. Pp. 128 – 135.

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. & Santoro, T.P. (2004). Computational GOMS Modeling of a Complex Team Task: Lessons Learned. In Proceedings of CHI 2004: Human Factors in Computing Systems. New York: ACM, Inc.

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. In Proceeding of UIST, 1995, Pittsburgh, PA, USA, November 14-17, 1995. New York: ACM. pp. 91-100.

Kieras, D.E., Wood, S.D., & Meyer, D.E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. ACM *Transactions on Computer-Human Interaction. 4*, 230-275.

Kirwan, B., & Ainsworth, L. K. (1992). *A guide to task analysis*. London: Taylor and Francis.

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

Lane, N. E., Strieb, M. I., Glenn, F. A., & Wherry, R. J. (1981). The human operator simulator: An overview. In J. Moraal, & K. F. Kraas, (Eds.). *Manned system design* (pp. 121-152). New York: Plenum.

Larkin, J.H. (1989) Display Based Problem Solving. In D. Klahr & K. Kotovsky (Eds.), *Complex Information Processing: The Impact of Herbert A. Simon*. Hillsdale, NJ: Erlbaum.

Laughery, K. R. (1989). Micro SAINT - A tool for modeling human performance in systems. In G.R. McMillan, D. Beevis, E. Salas, M.H. Strub, R. Sutton, & L. Van Breda (Eds.), *Applications of human performance models to system design*. New York: Plenum Press. 219-230. See also the web site of Micro Analysis and Design, Inc., http://www.maad.com.

McMillan, G. R., Beevis, D., Salas, E., Strub, M. H., Sutton, R., & Van Breda, L. (1989). *Applications of human performance models to system design*. New York: Plenum Press.

Newell, A. *Unified theories of cognition*. (1990). Cambridge, MA: Harvard University Press.

Norman, D. A. (1986). Cognitive engineering. In D.A. Norman, & Draper, S. W. (Eds.), *User centered system design*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Petrosky, H. (1985). *To engineer is human: The role of failure in successful design*. New York: St. Martin's Press.

Pew, R.W., Baron, S., Feehrer, C.E., & Miller, D.C. (1977). Critical review and analysis of performance models applicable to man-machine systems operation. Technical Report No. 3446, Bolt, Beranek and Newman, Inc., Cambridge MA.

Polson, P.G. (1987). A quantitative model of human-computer interaction. In J.M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. Cambridge, MA: Bradford, MIT Press.

St. Amant, R., and Riedl, M.O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies*, 55(1), 15-39.

Strieb, M.I., & Wherry, R.J. (1979). An introduction to the human operator simulator. Technical Report 1400.02-D, Analytics Inc., Willow Grove, PA.

Vera, A.H., Howes, A., McCurdy, M., Lewis, R.L. (2004). A constraint satisfaction approach to predicting skilled interactive cognition. In Proceedings of the CHI 2004, New York: ACM. Pp. 121 - 128

Wherry, R.J. (1976). The human operator simulator - HOS. In T.B. Sheridan, & G. Johannsen (Eds.), *Monitoring behavior and supervisory control*. New York: Plenum Press.

Whiteside, J., Jones, S., Levy, P. S., & Wixon, D. (1985). User performance with command, menu, and iconic interfaces. In *Proceedings of CHI '85*. New York: ACM.

Wood, S. D. (2000). Extending GOMS to Human Error and Applying it to Error-Tolerant Design. Doctoral dissertation, University of Michigan.

Zachary, W., Santarelli, T., Ryder, J., & Stokes, J. (2000). Developing a multi-tasking cognitive agent using the COGNET/IGEN integrative architecture. Technical Report No. 001004.9915, CHI Systems, Inc., Lower Gwynedd, PA. See also the web site for CHI Systems, Inc., http://www.chiinc.com/