Task Analysis and the Design of Functionality

David Kieras, University of Michigan

Introduction

To appear in A. Tucker (Ed.) The Computer Science and Engineering Handbook (2nd Ed). Boca Raton, CRC Inc.

Task analysis is the process of understanding the user's task thoroughly enough to help design a computer system that will effectively support users in doing the task. By **task** is meant the user's job or work activities, what the user is attempting to accomplish. By *analysis* is meant a relatively systematic approach to understanding the user's task that goes beyond unaided intuitions or speculations, and attempts to document and describe exactly what the task involves. The design of **functionality** is a stage of the design of computer systems in which the user-accessible functions of the computer system are chosen and specified. The basic thesis of this article is that the successful design of functionality requires a task analysis early enough in the system design to enable the developers to create a system that effectively supports the user's task. Thus the proper goal of the design of functionality is to choose functions that are both *useful* in the user's task, and which together with a good **user interface**, results in a system that is *usable*, being easy to learn and easy to use.

The user's task is not just to interact with the computer, but to get a job done. Thus understanding the user's task involves understanding the user's **task domain** and the user's larger job goals. Many systems are designed for ordinary people, who presumably lack specialized knowledge, and so the designers might believe that they understand the user's task adequately well without any further consideration. This belief is often not correct; the tasks of even ordinary people are often complex and poorly understood by developers. But in contrast, many economically significant systems are intended for expert users, and understanding their tasks is absolutely critical. For example, a system to assist a petroleum geologist must be based on an understanding of the knowledge and goals of the petroleum geologist. To be useful, such a system will require functions that produce information useful for the geologist; to be usable, the system will have to provide these functions in a way that the frequent and most important activities of the geologist are well supported. Thus, for success, the developer must design not just the user interface, but also the functionality behind the interface.

The purpose of this article is to provide some background and beginning "how to" information on how to conduct a task analysis and how to approach the design of functionality. The next portion of this article discusses why task analysis and the design of functionality are critical stages in software development, and how typical development processes interfere with these stages. Then will be presented some background on methods for human-machine system design that have developed in the field of **Human Factors** over the last few decades, including newer methods that attempt to identify the more cognitive components of tasks. The final section provides some how-to-do-it information in the form of a summary of existing methods, and a newly developing method that is especially suitable for computer system design. A general overview of the user interface design process, usability, and other specific aspects are provided in other chapters in this section.

Principles

The Critical Role of Task Analysis and Design of Functionality

In many software development organizations, some group, such as a marketing department or government procurement agents, prepares a list of requirements for the system to be developed. Such requirements specify the system functions at least in part. The designers and developers then further specify the functions, possibly adding or deleting functions from the list, and then begin to design an implementation for them. Typically, only at this point, or later, is the user interface design begun. Ideally, the design of the user interface will use appropriate techniques to arrive at a usable design, but these techniques normally are based on whatever conception of the user and the user's tasks have already been determined, and the interface is designed in terms of the functions that have already been specified. Thus, even when a usability process is followed, it might well arrive at only a local optimum defined by an inadequate characterization of the user's needs and the corresponding functions.

That is, the focus of usability methods tends to be on relatively low-level questions (such as menu structure) and on how to conduct usability tests to identify usability problems; the problem is posed as developing a usable interface to functions that have already been chosen. Typically, the system requirements have been prepared by a group which then "throws it over the wall" to a development group who arrives at the overall design and functional implementation, and then throws it over the wall to a usability group to "put a good interface on it."

The problem is that if the initial requirements and system functions are poorly chosen, the rest of the development will probably fail to produce a usable product. It is a truism in human-computer interaction that if customers need the functionality, they will buy and use even a clumsy, hard-to-use product, and if the functionality is poorly chosen, no amount of effort spent on user interface design will result in a usable system, and it might not even be useful at all. This is by no means a rare occurrence; it is easy to find cases of poorly chosen functionality that undermine whatever usability properties the system otherwise possesses. Some examples:

The interface is often not the problem. An important article with this title by [Goransson, Lind, Pettersson, Sandblad, & Schwalbe, 1987] presents several brief case studies that involve failures of functionality design masquerading as usability problems. The most painful is a database system in a business organization that was considered to be too difficult to use. The interface was improved to make the system reasonably easy to use, but then it became clear that nobody in the organization needed the data provided by the system! Apparently the original system development did not include an analysis of the needs of the organization or the system users. The best way to improve the usability of the system would have been to simply remove it.

Half a loaf is worse than none. The second major version of an otherwise easy-to-use basic word processing application included a multiple-column feature; however, it was not possible to mix the number of columns on a page. Note that documents having a uniform layout of 2 or 3 columns throughout do not exist in the real world; rather, real multicolumn documents always mix the number of columns on at least one page. For example, a common pattern is a title page with a single column for the title that spans the page, followed by the body of the document in two-column format. The application could produce such a document only if two separate documents were prepared, printed, and then physically cut-and-pasted together! In other words, the multiple-column feature of this second version was essentially useless for preparing real documents. A proper task analysis would have determined the kinds and structures of multiple-column documents that users would be likely to prepare. Using this information during the product development would have led to either more useful functionality (like the basic page-layout features in the third major release of the product), or a decision not to waste resources on a premature implementation of incomplete functionality.

Why doesn't it do that? A first-generation hand-held "digital diary" device provided calendar and date book functions equivalent to paper calendar books, but included no clock, no alarms, and no awareness of the current date, although such functions would have been minor additions to the hardware. In addition, there was no facility for scheduling repeating meetings, making such scheduling remarkably tedious. The only short cut was to use a rather clumsy copy-paste function, but it did not work for the meeting time field in the meeting information. A task analysis of typical user's needs would have identified all of these as highly desirable functions. Including them would have made the first generation of these devices much more viable.

Progress is not necessarily monotonic. The second version of a Personal Digital Assistant also had a problem with recurring meetings. In the first version, a single interface dialog was used for specifying recurring meetings, and it was possible to select multiple days per week for the repeating meeting. Thus, the user could easily specify a weekly repeating meeting schedule of the sort common in academics, e.g. scheduling a class that meets at the same time every Monday, Wednesday, and Friday for a semester. However, in the second version, which attempted many interface improvements, this facility moved down a couple of menu levels and became both invisible in the interface (unless a certain option was selected) and undocumented in the user manual. If any task analysis was done in connection with either the original or second interface design, it did not take into account the type of repeating meeting patterns needed in academic settings, a major segment of the user population that includes many valuable "early adopter" customers.

The role of task analysis in development

Problems with misdefined functionality arise because first, there is a tendency to assume that the requirements specifications for a piece of software can and should contain all that is necessary to design and implement the software, and second, the common processes for preparing these specifications often fail to include a real task analysis. Usually the requirements are simply a list of desirable features or functions, chosen haphazardly, and without critical examination on how they will fit together to support the user. Simply mentioning a user need in the requirements does not mean that the final system will include the right functions to make the system either useful or usable for meeting that need. The result is often a serious waste: pointless implementation effort and unused functionality. Thus, understanding the user's task is the most important step in system and interface design. The results of task analyses can be used in several different phases in the development process:

Development of requirements. This stage of development is emphasized in this article. A task analysis should be conducted before developing the system requirements to guide the choice and design of the system functionality; the ultimate usability of the product is actually determined at this stage. The goal of task analysis at this point is to find out what the user needs to be able to accomplish, so that the functionality of the system can be designed so that the user can accomplish the required tasks easily. While some later revision is likely to be required, these critical choices can be made before the system implementation or user interface is designed.

User interface design and evaluation. Task analysis results are needed during interface design to effectively design and evaluate the user interface. The usability process itself, and user testing, requires information about the user's tasks. Task analysis results can be used to choose benchmark tasks for user testing that will represent important uses of the system. Usage scenarios valuable during interface design can be chosen that are properly representative of user activities. A task analysis will help identify the portions of the interface that are most important for the user's tasks. Once an interface is designed and is undergoing evaluation, the original task analysis can be supplemented with an additional analysis of how the task would be done with the proposed interface. This can suggest usability improvements either by modifying the interface or improving the fit of the functionality to the more specific form of the user's task entailed by a proposed interface. In fact, some task analysis methods are very similar to user testing; the difference is that in user testing, one seeks to identify problems that the users have with an interface while performing selected tasks; in task analysis, one tries to understand how users will perform their task given a specific interface. Thus a task analysis might identify usability problems, but task analysis does not necessarily require user testing.

Follow-up after installation. Task analysis can be conducted on fielded or in-place systems to compare systems or identify potential problems or improvements. When a fully implemented system is in place, it is possible to conduct a fully detailed task analysis; these results could be used to compare the demands of different systems, identify problems that should be corrected in a new system, or to determine properties of the task that should be preserved in a new system.

Research and Application Background

The contribution of Human Factors to task analysis

Task analysis developed in the discipline of Human Factors, which has a long history of concern with the design process and how human-centered issues should be incorporated into system design. Much of Human Factors has been concerned with human participation in extremely large and complex systems, usually military systems, but in fact task analysis methods have been applied to a broad range of systems, ranging from hand-held radios, to radar consoles, to whole aircraft, chemical process plant control rooms, and very complex multiperson systems such as a warship combat information center. The breadth of this experience is no accident. Historically, Human Factors is the only discipline with an extended record of involvement and concern with human-system design, long predating the newer field of Human-Computer Interaction. These methods have been presented in comprehensive collections of task analysis techniques (e.g. [Kirwan & Ainsworth, 1992], [Beevis, Bost, Doering, Nordo, Oberman, Papin, Schuffel, & Streets, 1992], [Diaper & Stanton, in press-a]), and it remains a active area of research within Human Factors and allied fields ([Annett & Stanton, 2000a], [Schraagen, Chipman, & Shalin, 2000]). One purpose of this chapter is to summarize some of these techniques and concepts. But there are some key differences between the interfaces of the traditional kinds of systems treated in the Human Factors experience and the interfaces to computer-based systems.

Task properties of both types of systems. At a general level, the task analysis of both traditional and computer-based interfaces involve collecting and representing the answers to the following primary overall questions:

1. What does the system do as a whole?
2. Where does the human operator fit into the system? What role will the operator play?
3. What specific tasks must the operator perform in order to play that role?

A large-scale general difference between the two kinds of systems shows up in response to these first three questions. Namely, the function or "mission" of the system as a whole often goes well beyond the immediate goals of the human operators, especially for high-complexity systems. For example, the mission of a system such as AEGIS is to defend a naval task force against all forms of airborne attack. While each human operator of course has an interest in the success of this mission, the individual operator has a much more constrained set of goals, such as monitoring and identifying radar targets. HCI has tended to emphasize the design problem only at the single-operator level, as if only the third question was important. This myopic focus is another manifestation of the problem discussed above, that the choice and design of functionality is often not considered to be part of the process of designing for usability.

Task properties of traditional systems. At the level of the detailed design of the interface, much of Human Factors expertise involves the detailed design of traditional interfaces constructed using conventional pre-computer technology, often termed "knobs and dials" interfaces. A key design convention of traditional interfaces is that a single display device (e.g. a pressure gauge) displays a single system parameter or sensor value, and a single control device (e.g. a pump power switch) controls a single parameter or component of the system. Such interfaces require many individual physical devices, and so tend to be distributed in space, resulting in control rooms in which every surface is covered with displays and controls. Thus, many of the interface design issues involve the visual and spatial properties of the devices and their layout. For example, some contributors to problems with nuclear power plant control rooms are gauges that cannot be seen by the operator, and misleadingly symmetric switch layouts that have mirror-imaged on and off positions [Woods, O'Brien, & Hanes, 1987]. On the other hand, a useful design principle is to place related controls and displays together, and organize them so that the spatial arrangement corresponds to the order in which they are used (e.g. left-to-right). The entire array of controls and displays are supposed to be always or readily visible and accessible simply by looking or moving, and thus constitute a large external memory for the operator. But the contents of this external memory is usually at the level of individual system parameters and components. This constraint is a product of the available interface and control technology used in traditional systems.

Thus, in addition to the first three task analysis questions, a task analysis of a traditional system seeks answers to the following fourth question:

4. What system parameters and components must be accessible to allow the operator to perform the tasks?

Once these four preliminary questions are answered, an interface can be designed or evaluated. If an interface design does not already exist, then Human Factors guidelines and principles can be used to guide the creation of a design. Once an interface design is available, a further, more detailed task analysis can then determine how the operator would access the controls and displays involved in order to carry out the tasks. Due to the spatiality of such interfaces, visual and spatial issues receive considerable emphasis. The sequential or temporal properties of the procedures followed by the operators are thus closely related to the spatial layout of the interface.

Task properties of computer-based systems. In contrast to the myriad controls and displays of traditional interfaces, computer-based systems often have few "in-place objects" [see Beevis et al., 1992], meaning that the interface for even an extremely complex system can consist of only a single screen, keyboard, and mouse. The operator must thus control the system by sequentially-organized activities with this small number of mechanical or visual objects that tend to stay in the same place and compete for the limited keyboard and display space. Time has been traded for space; temporal layout of activity has been traded for spatial layout. The operator often must remember more, because the limited display space means that only a little information is visible at any one time, and complicated procedures are often needed to bring up other information. The user's procedures have relatively little spatial content, and instead become relatively arbitrary sequences of actions on the small set of objects, such as typing different command strings on a keyboard. These procedures can become quite elaborate sequential, repeating, and hierarchical patterns.

On the other hand, the power of the computer behind the interface permits the displayed information to be potentially more useful and more relevant to the user's task than the single-sensor/single-display traditional approach. For example, the software can combine several sensor readings and show a derived value that is what the user really wants to know. Likewise, the software can control several system components in response to a user command, radically simplifying the operator's procedures. Thus the available computer technology means that the interface displays and controls can be chosen much more flexibly than the traditional knob and dial technology. In conjunction with the greater complexity of such systems, it is now both possible and critically important to choose the system display and control functionality on the basis of what will work well for the user, rather than simply sorting through the system components and parameters to determine the relevant ones.

Thus a task analysis for a computer-based system must articulate what services, or functions, the computer should provide the operator, rather than on what fixed components and parameters the operator must have access to, leading to the following additional question for computer-based systems:

5. What display and control functions should the computer provide to support the operator in performing the tasks?

In other words, the critical step in computer-based system design is the *choice of functionality*. Once the functions are chosen, the constraints of computer interfaces mean that the procedural requirements of the interface are especially prominent; if the functions are well chosen, the procedures that the operator must follow will be simple and consistent. Thus, the focus on spatial layout in traditional systems is replaced by a concern with the choice of functionality and interface procedures in computer-based systems. The importance of this combination of task analysis, choice of functionality, and the predominance of the procedural aspects of the interface are the basis for the recommendations later in this chapter.

Contributions of Human-Computer Interaction to task analysis

The field of Human-Computer Interaction (HCI) is a relatively new and highly interdisciplinary field and still lacks a consensus on scientific, practical, and philosophical foundations. Consequently, a variety of ideas have been discussed concerning how developers should approach the problem of understanding what a new computer system needs to do for its users. While many original researchers and practitioners in HCI had their roots in Human Factors, several other disciplines have had a strong influence on HCI theory and practice.

There are roughly two groups of these disciplines. The first is **Cognitive Psychology**, which is a branch of scientific psychology that is concerned with human cognitive abilities such as comprehension, problem-solving, and learning. The second is a mixture of ideas from the social sciences, such as social-organizational psychology, ethnography, and anthropology. While the contribution of these fields has been important in developing the scientific basis of HCI, they either have little experience with humans in a work context, as is the case with cognitive psychology, or no experience with practical system design problems, as with the social sciences. On the other hand, Human Factors is almost completely oriented toward solving practical design problems in an ad-hoc manner, and is almost completely atheoretic in content. Thus the disciplines with the broad and theoretical science base lack experience in solving design problems, and the discipline with this practical knowledge lacks a comprehensive scientific foundation.

The current state of task analysis in HCI is thus rather confused [Diaper & Stanton, in press-b]; there has been an unfortunate tendency to *reinvent* task analysis under a variety of guises, as each theoretical approach presents its own insights about how to understand a work situation and design a system to support it. Moreover, many designers and developers have apparently simply started over with experience-based approaches. One example is *Contextual Design* [Holtzblatt, 2003] which is a collection of task-analytic techniques and activities which its proponents claim will lead one through the stages of understanding what users do, what they need, and what system and interface will help them do it; despite few explicit connections with previous work, many of the suggestions are familiar task-analytic techniques.

As each scientific community spawned its own ways of analyzing tasks, and as many development groups invented their own experienced-based approaches, the resulting hodgepodge of newly-minted ideas has become bewildering enough to HCI specialists, but is impenetrably obscure to the software developer who merely wants to develop a better system. For this reason, this chapter focuses on the "tried and true" pragmatic methodologies from Human Factors and a closely-related methodology based on the most clearly articulated of the newer theoretical approaches, the GOMS model.

New social-science approaches to task analysis. Since much computer usage takes place in organizations in which individual users must cooperate and interact as part of their work, view computer usage as a social activity is a way to try to capture the larger context of a computer user’s task. Some relevant social-science concepts can be summarized (see [Baecker, Grudin, Buxton, and Greenberg, 1995] for a sampling and overview). A general methodological approach is *ethnography*, which is the set of methods used by anthropologists to immerse oneself in a culture and document its structure (see [Blomberg, Burrell, & Guest, 2003]). An approach based on anthropology called *situated cognition* emphasizes understanding human activity in its larger and social context (see [Nardi, 1995] for an overview). Another theoretical approach is *activity theory* ([Nardi, 1995], [Turner & McEwan, in press]) which originated in the former Soviet Union as a comprehensive psychological theory, with some interesting differences from conventional western or American psychology. The proponents of all of these social-science approaches have had some successes, apparently due to their insistence on observing and documenting what people are actually doing in a specific situation and in their work context. In the context of common computer-industry practice in system design, this insistence might seem novel and noteworthy, but such attention to the user's context is characteristic of all competent task analyses; the contribution of these approaches is an emphasis on levels of the user's context that can be easily overlooked if one's focus is too narrowly on how the user interacts with the technological artifacts.

Contributions from cognitive psychology. The contribution of cognitive psychology to HCI is both more limited and more successful within its limited scope. Cognitive psychology treats an individual human as an information-processor who is considered to acquire information from the environment, transform it, store it, retrieve it, and act on it. This *information-processing* approach, also called the *computer metaphor*, has an obvious application to how humans would interact with computer systems. In a cognitive approach to HCI, the interaction between human and computer is viewed as two interacting information-processing systems with different capabilities, and in which one, the human, has goals to accomplish, and the other, the computer, is an artificial system which should be designed to facilitate the human's efforts. The relevance of cognitive psychology research is that it directly addresses two important aspects of usability: how difficult it is for the human to learn how to interact successfully with the computer, and how long it takes the human to conduct the interaction. The underlying topics of human learning, problem-solving, and skilled behavior, have been intensively researched for decades in cognitive psychology.

The application of cognitive psychology research results to human-computer interaction was first systematically presented by Card, Moran and Newell (1983) at two levels of analysis. The lower-level analysis is the Model Human Processor, a summary of about a century's worth of research on basic human perceptual, cognitive, and motor abilities in the form of an engineering model that could be applied to produce quantitative analysis and prediction of task execution times. The higher-level analysis was the **GOMS model**, which is a description of the *procedural knowledge* involved in doing a task. The acronym GOMS stands for the following: The user has **G**oals that can be accomplished with the system. **O**perators are the basic actions such as keystrokes performed in the task. **M**ethods are the procedures, consisting of sequences of operators, that will accomplish the goals. **S**election rules determine which method is appropriate for accomplishing a goal in a specific situation. In the Card, et al. formulation, the new user of a computer system will use various problem-solving and learning strategies to figure out how to accomplish tasks using the computer system, and then with additional practice, these results of problem-solving will become procedures that the user can routinely invoke to accomplish tasks in a smooth, skilled manner. The properties of the procedures will thus govern both the ease of learning and ease of use of the computer system. In the research program stemming from the original proposal, approaches to representing GOMS models based on cognitive psychology theory have been developed and validated empirically, along with the corresponding techniques and computer-based tools for representing, analyzing, and predicting human performance in human-computer interaction situations (see [John & Kieras, 1996a, b] for reviews).

The significance of the GOMS model for task analysis is that it provides a way to describe the task procedures in a way that has a theoretically rigorous and empirically validated scientific relationship to human cognition and performance. Space limitations preclude any further presentation of how GOMS can be used to express and evaluate a detailed interface design (see [John & Kieras, 1996a, b] and [Kieras, 1997, in press]). Later in this chapter a technique based on GOMS will be used couple task analysis with the design of functionality.

Best Practices: How to Do a Task Analysis

The basic idea of conducting a task analysis is to understand the user's activity in the context of the whole system, either an existing or a future system. While understanding human activity is the subject of scientific study in psychology and the social sciences, the conditions under which systems must be designed usually preclude the kind of extended and intensive research necessary to document and account for human behavior in a scientific mode. Thus a task analysis for system design must be rather more informal, and primarily heuristic in flavor compared to scientific research. The task analyst must do his or her best to understand the user's task situation well enough to influence the system design given the limited time and resources available. This does not mean that a task analysis is an easy job; large amounts of detailed information must be collected and interpreted, and experience in task analysis is valuable even in the most-structured methodologies (e.g. see Annett, in press).

The role of formalized methods for task analysis. Despite the fundamentally informal character of task analysis, many formal and quasi-formal systems for task analysis have been proposed, and have been widely recommended. Several will be summarized below. It is critical to understand that these systems do not in themselves analyze the task or produce an understanding of the task. Rather, they are ways to structure the task analysis process and notations for representing the results of task analysis. They have the important benefit of helping the analyst observe and think carefully about the user's actual task activity, both specifying what kinds of task information are likely to be useful to analyze, and providing a heuristic test for whether the task has actually been understood. That is, a good test for understanding something is whether one can represent it or document it, and constructing such a representation can be a good approach to trying to understand it. A formal representation of a task helps by representing the results of the task analysis is a form that can help document the analysis, so that it can be inspected, criticized, and revised. Finally, some of the more formal representations can be used as the basis for computer simulations or mathematical analyses to obtain quantitative predictions of task performance, but it must be understood that such results are no more correct than the original, and informally-obtained, task analysis underlying the representation.

An informal task analysis is better than none. Most of the task analysis methods to be surveyed require significant time and effort; spending these resources would usually be justified, given the near-certain failure of a system that fails to meet the actual needs of users. However, the current reality of software development is that developers often will not have adequate time and support to conduct a full-fledged task analysis. Under these conditions, what can be recommended? As pointed out by sources such as [Gould, 1988] and [Grudin, 1991], perhaps the most serious problem to remedy is that the developers often have no contact with actual users. Thus, if nothing more systematic is possible, the developers should spend some time in informal observation of real users actually doing real work. The developers should observe unobtrusively, but ask for explanation or clarification as needed, perhaps trying to learn the job themselves, but they do not make any recommendations, and do not discuss the system design. The goal of this activity is simply to try to gain some experience-based intuitions about the nature of the user's job, and what real users do and why. See [Gould, 1988] for additional discussion. Such informal, intuition-building contact with users will provide tremendous benefits at relatively little cost. Approaches such as Contextual Design [Holtzblatt, 2003] and the more elaborate methods presented here provide more detail and more systematic documentation, and will permit more careful and exact design and evaluation than casual observation, but some informal observation of users is infinitely better than no attempt at task analysis at all.

Collecting Task Data

Task analysis requires information about the user's situation and activities, but simply collecting data about the user's task is not necessarily a task analysis. In a task analysis, the goal is to understand the properties of the user task that can be used to specify the design of a system; this requires synthesis and interpretation beyond the data. The data collection methods summarized here are those that have been found to produce useful information about tasks (see [Kirwan & Ainsworth, 1992], [Gould, 1988]); the task analytic methods summarized in the next section are approaches that help analysts perform the synthesis and interpretation.

Observation of user behavior. In this fundamental family of methods, the analyst observes actual user behavior, usually with minimal intrusion or interference, and describes what has been observed in a thorough, systematic, and documented way. This type of task data collection is most similar to user testing, except that, as discussed above, the goal of task analysis is to understand the user's task, not just to identify problems that the user might have with a specific system design.

The setting for the user's activity can be the actual situation (e.g. in the field) or a laboratory simulation of the actual situation. Either all of the user's behavior can be recorded, or it could be sampled periodically to cover more time while reducing the data collection effort. The user's activities can be categorized, counted and analyzed in various ways. For example, the frequency of different activities and be tabulated, or the total time spent in different activities could be determined. Both such measures contribute valuable information on which task activities are most frequent or time-consuming, and thus important to address in the system design. Finer grain recording and analysis can provide information on the exact timing and sequence of task activities, which can be important in the detailed design of the interface. Videotaping users is a simple recording approach that supports both very general and very detailed analysis at low cost; consumer-grade equipment is often adequate.

A more intrusive method of observation is to have users "*think aloud*" about a task while performing it, or to have two users discuss and explain to each other how to do the task while performing it. The verbalization can disrupt normal task performance, but such *verbal protocols* are believed to be a rich source of information about the user's mental processes such as inferences and decision-making. The pitfall for the inexperienced is that the protocols can be extremely labor-intensive to analyze, especially if the goal is to reconstruct the user's cognitive processes. The most fruitful path is to transcribe the protocols, isolate segments of content, and attempt to classify them into an informative set of categories.

A final technique in this class is *walkthroughs* and *talkthroughs*, in which the users or designers carry out a task and describe it as they do so. The results are similar to a think-aloud protocol, but with more emphasis on the procedural steps involved. An important feature is that the interface or system need not exist; the users or designers can describe how the task would or should be carried out.

Critical incidents and major episodes. Instead of attempting to observe or understand the full variety of activity in the task, the analyst chooses incidents or episodes which are especially informative about the task and the system, and attempts to understand what happens in these; this is basically a case-study approach. Often the critical incidents are accidents, failures, or errors, and the analysis is based on retrospective reports from the people involved and any records produced during the incident. An important extension on this approach is the *critical decision method* (Wong, in press) which focuses on understanding the knowledge involved in making expert-level decisions in difficult situations. However, the critical "incident" might be a major episode of otherwise routine activity that serves especially well to reveal the problems in a system. For example, observation of a highly skilled operator performing a very specialized task revealed that most of the time was spent doing ordinary file maintenance; understanding why led to major improvements in the system [Brooks, personal communication].

Questionnaires. Questionnaires are a fixed set of questions that can be used to collect some types of user and task information on a large scale quite economically. The main problem is that the accuracy of the data is unknown compared to observation, and can be susceptible to memory errors and social influences. Despite the apparent simplicity of a questionnaire, designing and implementing a successful one is not easy, and can require an effort comparable to interviews or workplace observation. The newcomer should consult sources on questionnaire design before proceeding.

Structured interviews. Interviews involve talking to users or domain experts about the task. Typically some unstructured interviews might be done first, in which the analyst simply seeks any and all kinds of comments about the task. Structured interviews can then be planned; a series of predetermined questions for the interview is prepared to ensure a more systematic, complete, and consistent collection of information.

Interface surveys. An interface survey collects information about an existing, in-place or designed interface. Several examples are: Control and Display surveys determine what system parameters are shown to the user, and what components can be controlled. Labeling and Coding surveys can determine whether there are confusing labels or inconsistent color codes present in the interface. Operator Modifications surveys assess changes made to the interface by the users, such as added notes or markings, that can indicate problems in the interface. Finally, sightline surveys determine what parts of the interface can be seen from the operator's position; such surveys have found critical problems in nuclear power plant control rooms. Sightlines would not seem important for computer interfaces, but a common interface design problem is that the information required during a task is not on the screen at the time it is required; an analog to a sightline survey would identify these problems.

Representing Systems and Tasks

Once the task data is collected, the problem for the analyst is to determine how to represent the task data, which requires a decision about what aspects of the task are important, and how much detail to represent. The key function of a representation is to make the task structure visible or apparent in some way that supports the analyst's understanding of the task. By examining a task representation, an analyst hopes to identify problems in the task flow, such as critical bottlenecks, inconsistencies in procedures, excessive workloads, and activities that could be better supported by the system. Traditionally, a graphical representation, such as a flowchart or diagram, has been preferred, but as the complexity of the system and the operating procedures increase, diagrammatic representations lose their advantage.

Task decomposition. One general form of task analysis is often termed *task decomposition.* This is not a well-defined method at all, but merely reflects a philosophy that tasks usually have a complex structure, and a major problem for the analyst will be to decompose the whole task situation into sub-parts for further analysis, some of which will be critical to the system design, and others possibly less important. For example, one powerful approach is to consider how a task might be decomposed into a hierarchy of subtasks and the procedures for executing them, leading to a popular form of analysis called (somewhat too broadly) Hierarchical Task Analysis. However, another approach would be to decompose the task situation into considerations of how the controls are labeled, how they are arranged, and how the displays are coded. This is also a task decomposition, and might also have a hierarchical structure, but the emphasis is on describing aspects of the displays in the task situation. Obviously, depending on the specific system and its interface, some aspects of the user's task situation may be far more important to analyze than others. Developing an initial task decomposition can help identify what is involved overall in the user's task, and thus allow the analyst to choose what aspects of the task merit intensive analysis.

Level of detail. The question of how much detail to represent in a task analysis is difficult to answer. At the level of whole tasks, [Kirwan and Ainsworth, 1992] suggest a probability ¥ cost rule: if the probability of inadequate performance multiplied by the cost of inadequate performance is low, then the task is probably not worthwhile to analyze. But even if a task has been chosen as important, the level of detail at which to describe the particular task still must be chosen. Some terminology must be clarified at this point: task decompositions can be viewed as a standard inverted tree structure, with a single item, the overall task, at the top, and the individual actions (such as keystrokes or manipulating valves) or interface objects (switches, gauges) at the bottom. A *high-level* analysis deals only with the low-detail top parts of the tree, while a *low-level* analysis includes all of the tree from the top to the high-detail bottom.

The cost of task analysis rises quickly as more detail is represented and examined, but on the other hand, many critical design issues appear only at a detailed level. For example, at a high enough level of abstraction, the Unix operating system interface is essentially just like the Macintosh operating system interface; both interfaces provide the functionality for invoking application programs and copying, moving, and deleting files and directories. The notorious usability problems of Unix relative to other systems only appear at a level of detail that the cryptic, inconsistent, and clumsy command structure and generally poor feedback come to the surface. "The devil is in the details." Thus a task analysis capable of identifying usability problems in an interface design typically involves working at a low, fully detailed, level that involves individual commands and mouse selections. The opposite consideration holds for the design of functionality, as will be discussed more below. When choosing functionality, the question is how the user will carry out tasks using a set of system functions, and it is important to avoid being distracted by the details of the interface.

Task analysis at the whole-system level

When large systems are being designed, an important component of task analysis is to consider how the system, consisting of all the machines, and all the humans, is supposed to work as a whole in order to accomplish the overall system goal. This kind of very high-level analysis can be done even with very large systems, such as military systems involving multiple machines and humans. The purpose of the analysis is to determine what role in the whole system the individual human operators will play. Various methods for whole system analysis have been in routine use for some time. Briefly, these are as follows (see [Beevis et al., 1992], [Kirwan & Ainsworth, 1992]).

Mission and scenario analysis. Mission and scenario analysis is an approach to starting the system design from a description of what the system has to do (the mission), especially using specific concrete examples, or scenarios. See Brooks (this volume) for related discussion.

Function-flow diagrams. Function-flow diagrams are constructed to show the sequential or information-flow relationships of the functions performed in the system. [Beevis et al., 1992] provides a sets of large scale examples such as naval vessels.

Petri nets. Petri nets also represent the causal and sequential relationships between the functions performed in a system, but in a rigorous formalism. Various methodologies for modeling and simulating the system performance can also include timing information.

Function allocation. Function allocation is a set of fairly informal techniques for deciding which system functions should be performed by machines, and which by people. Usually mentioned in this context is the *Fitts list* that describes what kinds of activities can be best performed by humans versus machines. However, according to surveys described in [Beevis et al., 1992], this classic technique is rarely used in real design problems since it is simply not specific enough to drive design decisions. Rather, functions are typically allocated in an *ad-hoc* manner, often simply maintaining whatever allocation was used in the predecessor system, or following the rule that whatever can be automated should be, even though it is known that automation often produces safety or vigilance problems for human operators.

In the military systems analyzed heavily in Human Factors, the overall system goal is normally rather larger-scale and well above the level of concerns of the human operators. For example, in designing a new naval fighter aircraft, the system goals might be stated in terms such as "enable air superiority in naval operations under any conditions of weather and all possible combat theaters through the year 2000." At this level, the "users" of the system as a whole are military strategist and commanders, not the pilot, and the system as a whole will involve not just the pilot, but other people in the cockpit (such as a radar intercept operator), and also maintenance and ground operations personnel. Thus the humans involved in the system will have a variety of goals and tasks, depending on their role in the whole system.

At first glance, this level of analysis would appear to have little to do with computer systems; we often think of computer users as isolated individuals carrying out their tasks by interacting with their individual computers. However, when considering the needs of an organization, the mission level of analysis is clearly important; the system is supposed to accomplish something as a whole, and the individual humans all play roles defined by their relationship with each other and with the machines in the system. HCI has begun to consider higher levels of analysis, as in the field of computer-supported collaborative work, but perhaps the main reason why the mission level of analysis is not common parlance in HCI is that HCI has a cultural bias that organizations revolve around the humans, with the computers playing only a supporting role. Such a bias would explain the movement mentioned earlier towards incorporating more social-science methodology into system design. In contrast, in military systems, the human operators are often viewed as "parts" in the overall system, whose ultimate "user" is the commanding officer, leading to a concern with how the humans and machines fit together. Regardless of the perspective taken to the whole system, at some point in the analysis, the activities of the individual humans that actually interact directly with the equipment begin to appear, and it is then both possible and essential to identify the goals that they, as individual operators, will need to accomplish. At this point, task analysis methodology begins to overlap with the concerns of computer user interface design.

Representing the User's Task

Once the whole system and the roles of the individual users and operators has been characterized, the main focus of task analytic work is identify more specific properties of the situation and activities of the human operator or user. These can be summarized as what the user needs to know, what the user must do, what the user sees and interacts with, or what the user might do wrong.

Representing what the user needs to know

The goal of this type of task analysis is to represent what knowledge the human needs to have in order to effectively operate the system. Clearly, the human needs to know how to operate the equipment; such procedural knowledge is treated under its own heading below. But the operator might need additional procedural knowledge that is not directly related to the equipment, and additional non-procedural conceptual background knowledge. For example, a successful fighter aircraft pilot must know more than just the procedures for operating the aircraft and the on-board equipment; he or she must have additional procedural skills such as combat tactics, navigation, and communication protocols, and an understanding of the aircraft mechanisms and overall military situation is valuable in dealing with unanticipated and novel situations.

Information on what the user needs to know is clearly useful for specifying the content of operator training and operator qualifications. It can also be useful in choosing system functionality in that large benefits can be obtained by implementing system functions that make it unnecessary for users to know concepts or skills that are difficult to learn; such simplifications typically are accompanied by simplifications in the operating procedures as well. Aircraft computer systems that automate navigation and fuel conservation tasks are an obvious example.

In some cases where the user knowledge is mostly procedural in content, it can be represented in a straightforward way, such as decision-action tables that describe what interpretation should be made of a specific situation, as in an equipment trouble-shooting guide. However, the required knowledge can be extremely hard to identify if it does not have a direct and overt relationship to "what to do" operating procedures. An example is the petroleum geologist, who after staring at a display of complex data for some time, comes up with a decision about where to drill, and probably cannot provide a rigorous explanation for how the decision was made. Understanding how and why users make such decisions is difficult because there is very little or no observable behavior prior to producing the result; it is all "in the head," a *purely cognitive task.*

Analyzing a purely cognitive task in complete detail is essentially a cognitive psychology research project, and so is not usually practical in a system design context. Furthermore, to the extent that a cognitive task can be completely characterized, it becomes a candidate for automatization, making the design of the user interface moot. Expert systems technology when successfully applied using "knowledge acquisition" methods (see [Boose, 1992]) is an example of task analyses carried out in enough detail that the need for a human performer of the task is eliminated.

However, most cognitive tasks seem neither possible nor practical to analyze this thoroughly, but there is still a need to support the human performer with system functions and interfaces and training programs and materials that improve performance in the task even if it is not completely understood. For example, [Gott, 1988] surveys cases in which an intensive effort to identify the knowledge required for tasks can produce large improvements in training programs for highly demanding cognitive tasks such as electronics troubleshooting.

During the 1990s there were many efforts to develop methods for **cognitive task analysis** (CTA) in which the emphasis is on describing the knowledge required for a task, and how it is used in decision-making, situation-recognition, or problem-solving rather than just the procedures involved, even though most task analysis procedures developed prior to that time included cognitive activities such as decision-making and problem-solving and were often conducted in order to identify the required background knowledge that needed to included in operator training. For example, [Shepherd, 2000] summarizes how the the first systematic task analysis method, Hierarchical Task Analysis (see below) normally includes various cognitive information. Furthermore, as argued by [Chipman, Schraagen, and Shalin, 2000], it makes little sense to attempt to divide cognitive processes such as decision-making from the actions and procedures involved in actual task performance. There are currently a wide variety of CTA techniques proposed or under development; see [Chipman, Schraagen, and Shalin, 2000] for an overview, and [Dubois & Shalin, 2000] and [Seamster, Redding, and Kaempf, 2000] for useful summaries of some important methods. Most current CTA methods involve some form of interview techniques for eliciting a subset of the most critical knowledge involved in a task, often focussing on critical incidents (e.g. [Militello & Hutton, 2000], see [Wong, in press]).

There are many cases where CTA resulted in dramatically improved training materials and programs (e.g. [O'Hare, Wiggins, Williams, & Wong, 2000], [Schaafstal & Schraagen, 2000] . However, an earlier review by [Essens, Fallesen, McCann, Cannon-Bowers, and Dorfel, 1994] reported that while cognitive task analysis techniques have demonstrated successes in developing training programs, there had been few demonstrations of successful design of computer systems for decision aiding. [Landauer, 1995] notes that decision-support systems have yet to produce a convincing track record of improving human productivity; this result would be expected given the general lack of task analysis in system design, and the great difficulty of task analysis in the case of heavily cognitive tasks.

Given that support for cognitively demanding tasks is touted as being one of the main contributions of computers, it is critical that more progress be made in the future on how to conduct task analysis and system design for purely cognitive tasks.

Representing what the user has to do

A major form of task analysis is describing the actions or activities carried out by the human operator while tasks are being executed. Such analyses have many uses; the description of how a task is currently conducted, or would be conducted with a proposed design, can be used for prescribing training, assisting in the identification of design problems in the interface, or as a basis for quantitative or simulation modeling to obtain predictions of system performance. Depending on the level of detail chosen for the analysis, the description might be very high-level, or might be fully detailed, describing the individual valve operations or keystrokes needed to carry out a task. The following are the major methods for representing procedures:

Operational sequence diagrams. Operational sequence diagrams and related techniques show the sequence of the operations (actions) carried out by the user (or the machine) to perform a task, represented graphically as a flowchart using standardized symbols for the types of operations. Such diagrams are often partitioned, showing the user's actions on one side, and machine's on the other, to show the pattern of operation between the user and the machine.

Timeline analysis. Timeline analyses simply display activities, or some characteristic of them, as a function of time during task execution. For example, a workload profile for an airliner cockpit would show a large variety and intensity of activities during landing and takeoff, but not during cruising. After constructing a timeline display, the analyst looks for workload peaks (such as the operator having to remember too many things), or conflicts, such as the operator having to use two widely separated controls at the same time.

Hierarchical task analysis. Hierarchical task analysis (HTA) involves describing a task as a hierarchy of tasks and subtasks, emphasizing the procedures that operators will carry out, using several specific forms of description. The term "hierarchical" is somewhat misleading, since many forms of task analysis produce hierarchical descriptions; a better term might be "Procedure hierarchy task analysis." The results of an HTA are typically represented either graphically, as a sort of annotated tree diagram of the task structure similar to the conventional diagram of a function-call hierarchy in programming, or in a more compact tabular form. This is the original form of systematic task analysis [Annett, Duncan, Stammers, and Gray, 1971] and still the single most heavily-used form of task analysis. See [Annett, in press] for additional background and procedural guide.

HTA descriptions involve *goals*, *tasks*, *operations*, and *plans*. A goal is a desired state of affairs (e.g. a chemical process proceeding at a certain rate). A task is a combination of a goal and a context (e.g. get a chemical process going at a certain rate given the initial conditions in the reactor). Operations are activities for attaining a goal (e.g. procedures for introducing reagents into the reactor, increasing the temperature, and so forth). Plans specify which operations should be applied under what conditions (e.g. which procedure to follow if the reactor is already hot). Plans usually appear as annotations to the tree-structure diagram that explain which portions of the tree will be executed under what conditions. Each operation in turn might be decomposed into subtasks, leading to a hierarchical structure. The analysis can be carried out to a any desired level of detail, depending on the requirements on the analysis.

GOMS models. GOMS models, introduced above, are closely related to Hierarchical task analysis; and in fact [Kirwan and Ainsworth, 1992] include GOMS as a form of HTA. GOMS models describe a task in terms of a hierarchy of goals and subgoals, methods which are sequence of operators (actions) that when executed will accomplish the goals, and selection rules that choose which method should be applied to accomplish a particular goal in a specific situation. However, both in theory and in practice, GOMS models are different from HTA. The concept of GOMS models grew out of research on human problem solving and cognitive skill, whereas HTA appears to have originated out of the pragmatic common-sense observation that tasks often involve subtasks, and eventually involve carrying out sequences of actions. Because of its more principled origins, GOMS models are more disciplined than HTA descriptions. The contrast is perhaps most clear in the difficulty HTA descriptions have in expressing the flow of control: the procedural structure of goals and subgoals must be deduced from the plans, which appear only as annotations to the sequence of operations. In contrast, GOMS models represent plans and operations in a uniform format using only methods and selection rules. An HTA plan would be represented as simply a higher-order method that carries out lower-level methods or actions in the appropriate sequence, along with a selection rule for when the higher-order method should be applied.

Representing what the user sees and interacts with

The set of objects that the user interacts with during task execution are clearly closely related to the procedures that the user must follow, in that a full procedural description of the user's task will (or should) refer to all objects in the task situation that the user must observe or manipulated. However, it can be useful to attempt to identify and describe the relevant objects and event independently of the procedures in which they are used. Such a task analysis can identify some potential serious problems or design issues quite rapidly. For example, studies of nuclear power plant control rooms [Woods et al., 1987] found that important displays were located in positions such that they could not be read by the operator. A task decomposition can be applied to break the overall task situation down into smaller portions, and the interface survey technique mentioned above can then determine the various objects in the task situation. Collecting additional information, e.g. from interviews or walk-through, can then lead to an assessment of whether and under what conditions the individual controls or displays are required for task execution.

There are then a variety of guidelines in human factors for determining whether the controls and displays are adequately accessible.

A related form of task analysis is concerned with the layout in space of the displays, controls, or other people that the operator must interact with. *Link analysis* is a straightforward methodology for tabulating the pairwise accessibility relationships between people and objects in the task environment, and weighting them by the frequency and difficulty of access (e.g. greater distance). Alternative arrangements can be easily explored in order to minimize the difficulty of the most frequent access paths. For example, a link analysis revealed that a combat information center on a warship was laid out in such a way that the movement and communication patterns involved frequent crossing of paths, sightlines, and so forth. A simple rearrangement of workstation positions greatly reduced the amount of interference. An analog for computer interfaces would be analyzing the transitions between different portions of the interface such as dialogs or screen objects. A design could be improved by making the most frequent transitions short and direct.

Representing what the user might do wrong

Human Factors practitioners and researchers have developed a variety of techniques for analyzing situations in which errors have happened, or might happen. The goal is to determine whether human errors will have serious consequences, and to try to identify where they might occur and how likely they are to occur. The design of the system or the interface can then be modified to try to reduce the likelihood of human errors, or mitigate the consequences of them. Some key techniques can be summarized:

Event trees. In an event tree, the possible paths, or sequences of behaviors, through the task are shown as a tree diagram; each behavior outcome is represented either as success/failure, or a multi-way branch, e.g. for the type of diagnosis made by an operation in response to a system alarm display. An event tree can be used to determine the consequences of human errors, such as misunderstanding an alarm. Each path can be given a predicted probability of occurrence based on estimates of the reliability of human operators at performing each step in the sequence (these estimates are controversial; see [Reason, 1990] for discussion).

Failure modes and effects analysis. The analysis of human failure modes and their effects is modeled after a common hardware reliability assessment process. The analyst considers each step in a procedure, and attempts to list all the possible failures an operator might commit, such as to omit the action, perform it too early, too late, too forcefully, and so forth. The consequences of each such failure "mode" can then be worked out, and again a probability of failure could be predicted.

Fault trees. In a fault tree analysis, the analyst starts with a possible system failure, and then documents the logical combination of human and machine failures that could lead to it. The probability of the fault occurring can then be estimated, and possible ways to reduce the probability can be determined.

Until recently, these techniques had not been applied in computer user interface design to any visible extent. At most, user interface design guides contained a few general suggestions for how interfaces could be designed to reduce the chance of human error. Recent work such as [Stanton, 2003] and [Wood, 1999] shows promise in using a task analysis as a basis for systematically examining how errors might be made and how they can be detected and recovered from. Future work along these line will be an extraordinarily important contribution to system and interface design.

Using GOMS Task Analysis in Functionality and Interface Design

The task analysis techniques described above work well enough to have been developed and applied in actual system design contexts. However, they have mainly developed in the analysis of traditional interface technologies rather than computer user interfaces. While as discussed above, the general concepts of task analysis hold for computer interfaces, there are some key differences and a clear need to address computer user interfaces more directly. In summary, the problems with traditional analysis methods are:

Representing a mass of procedural detail. Computer interface procedures tend to be complicated, repetitious, and hierarchical. The primarily graphical and tabular representations traditionally used for procedures become unwieldy when the amount of detail is large.

Representing procedures that differ in level of analysis and type. For example, in Hierarchical Task Analysis, a plan is represented differently from a procedure, even though a plan is simply a kind of higher-order procedure.

Moving from a task analysis to a functional design to an interface design. To a great extent, Human Factors practice uses different representations for different stages of the design process (see [Kirwan & Ainsworth, 1992], [Beevis et al., 1992]). It would be desirable to have a single representation that spans these stages even if it only covers part of the task analysis and design issues.

This section describes how GOMS models could be used to represent a high-level task analysis that can be used to help choose the desirable functionality for a system. Because GOMS models have a programming-language-like form, they can represent large quantities of procedural detail in a uniform notation that works from a very high level down to the lowest level of the interface design.

High-level GOMS analysis

Using high-level GOMS models is an alternative to the conventional requirements development and interface design process discussed in the introduction to this chapter. The approach is to drive the choice of functionality from the high-level procedures for doing the tasks, choosing functions that will produce simple procedures for the user. By considering the task at a high level, these decisions can be made independently of, and prior to, the interface design, thereby improving the chances that the chosen functionality will enable a highly useful and usable product once a good interface is developed. Key interface design decisions, such as whether a color display is needed, can be made explicit and given a well-founded basis, such as how color-coding could be used to make the task easier.

The methodology involves choosing the system functionality based on high-level GOMS analysis of how the task would be done using a proposed set of functions. The analyst can then begin to elaborate the design by making some interface design decisions and writing the corresponding lower-level methods. If the design of the functionality is sound, it should be possible to expand the high-level model into a more detailed GOMS model that also has simple and efficient methods. If desired, the GOMS model can be fully elaborated down to the keystroke level of detail that can produce usability predictions (see [Kieras, 1997], [John & Kieras, 1996a, b]).

GOMS models involve goals and operators at all levels of analysis, with the lowest level being the so-called keystroke level, of individual keystrokes or mouse movements. The lowest level goals will have methods consisting of keystroke-level operators, and might be basic procedures such as moving an object on the screen, or selecting a piece of text. However, in a high-level GOMS model, the goals may refer only to parts of the users task that are independent of the specific interface, and may not specify operations in the interface. For example, a possible high-level goal would be Add a Footnote, but not Select INSERT FOOTNOTE from EDIT menu. Likewise, the operators must be well above the keystroke level of detail, and not be specific interface actions. The lowest level of detail a operator may have is to invoke a system function, or perform a mental decision or action such as choosing which files to delete or thinking of a file name. For example, an allowable operator would be Invoke the database update function, but not Click on the UPDATE button.

The methods in a high-level GOMS model describe the order in which mental actions or decisions, submethods, and invocations of system functions are executed. The methods should document what information the user needs to acquire in order to make any required decisions and to invoke the system functions, and also should represent where the user might detect errors and how they might be corrected with additional system functions. All too often, support for error detection and correction by the user is either missing, or is a clumsy add-on to a system design; by including it in the high-level model for the task, the designer may be able to identify ways in which errors can be prevented, detected, and corrected, early and easily.

An example of high-level GOMS analysis

The domain for this example is electronic circuit design and computer-aided design systems for electronic design (ECAD). A task analysis of this domain would reveal many very complex activities on the part of electronic circuit designers. Of special interest are several tasks for which computer support is feasible: After a circuit is designed, its correct functioning must be verified, and then its manufacturing cost estimated, power and cooling requirements determined, the layout of the printed circuit boards designed, and automated assembly operations specified, and so fourth.

This example involves computer support for the task of verifying the circuit design by using computer simulation to replace the traditional slow and costly "breadboard" prototyping. For many years now, computer-based tools for this process have been available and undergoing development, based on techniques for simulating the behavior of the circuit using an abstract mathematical representation of the circuit. For purposes of this example, attention will be limited to the somewhat simpler domain of digital circuit design. Here the components are black-box modules (i.e. integrated circuits) with known behaviors, and the circuit consists simply of these components with their terminals interconnected with wires.

The high-level functionality design of such a system will be illustrated with an ideal task-driven design example, describing how such systems *should* have been designed. Then the design of a typical actual system will be presented in terms of the high-level analysis and compared with the task-driven design.

Task-driven design example. Given the basic functionality concept of using a simulation of a circuit, the top-level method to accomplish this goal is the first method shown in Figure 1, which accomplishes the goal Verify circuit with ECAD system. This first method needs some explanation of the notation, which is the "Natural" GOMS Language (NGOMSL) described in [Kieras, 1997] for representing GOMS models in a readable format. The first line introduces a method for accomplishing the top-level user goal. It will be executed whenever the goal is asserted, and terminates when the return with goal accomplished operator in Step 4 is executed. Step 1 represents the user's "black-box" mental activity of thinking up the original idea for the circuit design; this think-of operator is just a place holder; no attempt is made to represent the extraordinarily complex cognitive processes involved. Step 2 asserts a subgoal to specify the circuit for the ECAD system; the method for this subgoal appears next in Figure 1. Step 3 of the top-level method is a high-level operator for invoking the functionality of the circuit simulator and getting the results; no commitment is made at this point to how this will be done in the interface. Step 4 documents that at this point the user will decide whether the job is complete or not based on the output of the simulator. Step 5 is another placeholder for a complex cognitive process of deciding what modification to make to the circuit. Step 6 invokes the functionality to modify the circuit, which would probably be much like that involved in Step 2, but which will not be further elaborated in this example. Finally, the loop in Step 7 shows that the top-level task is iterative.

The next step in the example is to consider the method for entering a circuit into the ECAD system. In this domain, schematic circuit drawings are the conventional representations for a circuit, so the basic functionality that needs to be provided is a tool for drawing circuit diagrams. This is reflected in the method for the goal Enter circuit into ECAD system.

This method starts with invoking the tool, and then has a simple iteration consisting of thinking of something to draw, accomplishing the goal of drawing it, and repeating until done. The method gets more interesting when the goal of drawing an object is considered, because in this domain there are some fundamentally different kinds of objects, and the information requirements for drawing them are different. Only two kinds of objects will be considered here.

A selection rule is needed in a GOMS model to choose what method to apply depending on the kind of object, and then a separate method is needed for each kind of object; the selection rule set in Figure 1 thus accomplishes a general goal by asserting a more specific goal, which then triggers the corresponding method. The method for drawing a component requires a decision about the type of component (e.g. what specific multiplexer chip should be used) and where in the diagram the component should be placed to produce a visually clear diagram. Drawing a connecting wire requires deciding which two points in the circuit the wire should connect, and also how the wire should be routed to produce a clear appearance.

At this point, the analysis has documented some possibly difficult and time-consuming activities that the user will have to do; candidates for additional system functions to simplify these activities could be considered. For example, the step of thinking of an appropriate component might be quite difficult, due to the huge number of integrated circuits available, and likewise thinking of the starting and ending points for the wire involves knowing which input or output function goes with each of the many pins on the chips. Some kind of on-line documentation or database to provide this information in a form that meshes well with the task might be valuable. Likewise, a welcome function might be some automation to choose a good routing for wires.

```
Method for goal: Verify circuit with ECAD system
       Step 1. Think-of circuit idea.
       Step 2. Accomplish Goal: Enter circuit into ECAD system.
       Step 3. Run simulation of circuit with ECAD system.
       Step 4. Decide: If circuit performs correct function,
                    then return with goal accomplished.
       Step 5. Think-of modification to circuit.
       Step 6. Make modification with ECAD system.
       Step 7. Go to 3.

Method for goal: Enter circuit into ECAD system
       Step 1. Invoke drawing tool.
       Step 2. Think-of object to draw next.
       Step 3. If no more objects, then Return with goal accomplished.
       Step 4. Accomplish Goal: draw the next object.
       Step 5. Go to 2.

Selection rule set for goal: Drawing an object
       If object is a component, then accomplish Goal: draw a component.
       If object is a wire, then accomplish Goal: draw a wire.
       ...
       Return with goal accomplished

Method for goal: Draw a component
       Step 1. Think-of component type.
       Step 2. Think-of component placement.
       Step 3. Invoke component-drawing function with type and placement.
       Step 4. Return with goal accomplished.

Method for goal: Draw a wire
       Step 1. Think-of starting and ending points for wire.
       Step 2. Think-of route for wire.
       Step 3. Invoke wire drawing function with starting point,
                    ending point, and route.
       Step 4. Return with goal accomplished.
```

Figure 1. Preliminary high-level methods for ECAD system.

Although the analysis of the desirable functions has just begun, it is worthwhile to consider what errors the user might make and how the system functionality will support identifying and correcting them.

In this domain, the errors the user might make can be divided into *semantic* errors, in which the specified circuit does not do what it is supposed to do, and *syntactic* errors, in which the specified circuit is invalid, regardless of the function. The semantic errors can only be detected by the user evaluating the behavior of the circuit, and discovering that the idea for the circuit was incorrect, or incorrectly specified. Notice that the iteration in the top-level method provides for detecting and correcting semantic errors. Syntactic errors arise in the digital circuit domain because there are certain connection patterns that are incorrect just in terms of what is permissible and meaningful with digital logic circuits. Two common cases are disallowed connections (e.g. shorting an output terminal to ground) or missing connections (e.g. leaving an input terminal unconnected).

It is important to correct syntactic errors prior to running the simulator because their presence will cause the simulator to fail or produce invalid results. Functions to detect syntactic errors can be implemented fairly easily; the design question is exactly how this functionality should be defined so that it most helps the user. Note that disallowed connections appear at the level of individual wires, while missing connections only show up at the level of the entire drawing. Thus the functions to assist in detecting and correcting disallowed connections should be operating while the user is drawing wires, and those for missing connections while the user is working on the whole drawing. Figure 2 shows the corresponding revisions and additions to some of the methods in Figure 1.

The method for entering the circuit now incorporates a method invoked in Step 6 to proofread the drawing, which is done by finding the missing connections in the drawing and adding a wire for each one. The method for drawing a wire now has Step 4 that checks for the wire being disallowed immediately after it is drawn; if there is a problem, the wire will be corrected before proceeding.

```
Method for goal: Enter circuit into ECAD system
       Step 1. Invoke drawing tool.
       Step 2. Think-of object to draw next.
       Step 3. Decide: If no more objects, then go to 6.
       Step 4. Accomplish Goal: draw the next object.
       Step 5. Go to 2.
       Step 6. Accomplish Goal: Proofread drawing.
       Step 7. Return with goal accomplished.

Method for goal: Proofread drawing
       Step 1. Find missing connection in drawing.
       Step 2. Decide: If no missing connection, return with goal accomplished.
       Step 3. Accomplish Goal: Draw wire for connection.
       Step 4. Go to 1.

Method for goal: Draw a wire
       Step 1. Think-of starting and ending points for wire.
       Step 2. Think-of route for wire.
       Step 3. Invoke wire drawing function with starting point,
                    ending point, and route.
       Step 4. Decide: If wire is not disallowed, return with goal accomplished.
       Step 5. Correct the wire.
       Step 6. Return with goal accomplished.
```

Figure 2. Revised methods incorporating error detection and correction steps.

At this point in the design of the functionality, the analysis has documented what information the user needs to get about syntactic errors in the circuit diagram, and where this information can be used to detect and correct the error immediately. Some thought can now be given to what functionality might be useful to support the user in finding syntactically missing and disallowed connections. One obvious candidate that comes to mind is to use color coding; for example, perhaps unconnected input terminals could start out as red in color on the display, and then turn green when validly connected. Likewise, perhaps as soon as a wire is connected at both ends, it could turn red if it is a disallowed connection, and green if it is legal. This use of color should work very well, since properly-designed color-coding is known to be an extremely effective way to aid visual search. In addition, the color-coding calls the user's attention to the problems but without forcibly interrupting the user. This design rationale for using color is an interesting contrast to actual ECAD displays, which generally make profligate use of color in ways that lack any obvious value in performing the task. Figure 3 presents a revision of the methods to incorporate this preliminary interface design decision.

```
Method for goal: Proofread drawing
       Step 1. Find a red terminal in drawing.
       Step 2. Decide: If no red terminals, return with goal accomplished.
       Step 3. Accomplish Goal: Draw wire at red terminal.
       Step 4. Go to 1.

Method for goal: Draw a wire
       Step 1. Think-of starting and ending points for wire.
       Step 2. Think-of route for wire.
       Step 3. Invoke wire drawing function with starting point,
                    ending point, and route.
       Step 4. Decide: If wire is now green, return with goal accomplished.
       Step 5. Decide: If wire is red, think-of problem with wire.
       Step 6. Go to 1.
```

Figure 3. Methods incorporating color codes for syntactic drawing errors.

At this point, the functionality design also has clear implications for how the system implementation needs to be designed, in that the system must be able to perform the required syntax-checking computations on the diagram quickly enough update the display while the drawing is in progress. Thus, performing the task analysis for the design of the functionality has not only helped guide the design to a fundamentally more usable approach, but also produces some critical implementation specifications very early in the design.

An actual design example. The above example of how the design of functionality can be aided by working out a high-level GOMS model of the task seems straightforward and unremarkable. A good design is usually intuitively "right," and once presented, seems obvious. However, at least the first few generations of ECAD tools did not implement such an "intuitively obvious" design at all, probably because nothing was done that resembled the kind of task and functionality analysis just presented. Rather, a first version of the system was probably designed and implemented whose methods were the obvious ones shown in Figure 1: the user will draw a schematic diagram in the obvious way, and then run the simulator on it. However, once the system was in use, it became obvious that errors could be made in the schematic diagram that would cause the simulation to fail or produce misleading results. The solution was to simply provide a set of functions to check the diagram for errors, but to do so in an opportunistic, ad-hoc fashion, involving minimum implementation effort, that failed to take into account the impact on the user's task. Figure 4 shows the resulting method, which was actually implemented in some popular ECAD systems.

The top level is the same as in the previous example, except for Step 3, which checks and corrects the circuit after the entire drawing is completed. The method for checking and correcting the circuit first involves invoking a checking function, which was designed to produce a series of error messages that the user would process one at a time. For ease in implementation, the checking function does not work in terms of the drawing, but in terms of the abstract circuit representation, the "netlist", and so reports the site of the syntactically illegal circuit feature in terms of the name of the "node" in the netlist. However, the only way the user has to examine and modify the circuit is in terms of the schematic diagram. So the method for processing each error message then first involves locating the corresponding point in the circuit diagram, and then making a modification to the diagram. To locate the site of the problem on the circuit diagram, the user invokes an identification function and provides the netlist node name; the function then highlights the corresponding part of the circuit diagram, which the user can then locate on the screen. In other words, to check the diagram for errors, the user must wait until the entire diagram is completely drawn, and then invoke a function whose output must be manually transferred into another function that finally identifies the location of the error!

Obviously, the design of the functionality in this version of the system will inevitably result in a far less usable system than in the task-driven design; instead of getting immediate feedback at the time and place of an error, the user must finish drawing the circuit and then engage in a convoluted procedure to identify the errors in the drawing. Although the interface has not yet been specified, the inferior usability of the actual design relative to the task-driven design is clearly indicated by the additional number of methods and method steps, and the time-consuming nature of many of the additional steps. In contrast to the task-driven design, this actual design seems preposterous, and could be dismissed as a silly example except for the fact that at least one major vendor of ECAD software used exactly this design.

```
Method for goal: Verify circuit with ECAD system
       Step 1. Think-of circuit idea.
       Step 2. Accomplish Goal: Enter circuit into ECAD tool.
       Step 3. Accomplish Goal: Check and correct circuit.
       Step 4. Run simulation of circuit with ECAD tool.
       Step 5. Decide: If circuit performs correct function,
                     then return with goal accomplished.
       Step 6. Think-of modification to circuit.
       Step 7. Make modification in ECAD tool.
       Step 8. Go to 3.

Method for goal: Check and correct circuit
       Step 1. Invoke checking function.
       Step 2. Look at next error message.
       Step 3. If no more error messages, Return with goal accomplished.
       Step 4. Accomplish Goal: Process error message.
       Step 5. Go to 2.

Method for goal: Process error message
       Step 1. Accomplish Goal: Locate erroneous point in circuit.
       Step 2. Think-of modification to erroneous point.
       Step 3. Make modification to circuit.
       Step 4. Return with goal accomplished.

Method for goal: Locate erroneous point in circuit
       Step 1. Read type of error, netlist node name from error message.
       Step 2. Invoke identification function.
       Step 3. Enter netlist node name into identification function.
       Step 4. Locate highlighted portion of circuit.
       Step 5. Return with goal accomplished.
```

Figure 4. Methods for an actual ECAD system.

In summary, the task-driven design was based on an analysis of how the user would do the task, and what functionality would help the user do it easily. The result was that users could detect and correct errors in the diagram while drawing the diagram, and so could always work directly with the natural display of the circuit structure. In addition, good use was made of color display capabilities, which often go to waste. The actual design probably arose because user errors were not considered until very late in the development process, and the response was minimal add-ons of functionality, leaving the initial functionality decisions intact. The high-level GOMS model clarifies the difference between the two designs by showing the overall structure of the interaction. Even at a very high level of abstraction, a poor design of functionality can result in task methods that are obviously inefficient and clumsy. Thus, high-level GOMS models can capture critical insights from a task analysis to help guide the initial design of a system and its functionality.

Research Issues and Concluding Summary

The major research problem in task analysis is attempting to bring some coherence and theoretical structure to the field. While psychology as a whole rather seriously lacks a single theoretical structure, the subfields most relevant to human-system interaction, are potentially unified by work in cognitive psychology on cognitive architectures, which are computational modeling systems that attempt to provide a framework for explaining human cognition and performance (see [Byrne, 2003] for an overview). These architectures are directly useful in system design in two ways: First, because the architecture must be "programmed" to perform the task with task-specific knowledge, the resulting model contains the content of a full-fledged task analysis, both the procedural and cognitive components. Thus, constructing a cognitive-architectural model is a way to represent the results of a task analysis and verify its completeness and accuracy. Second, because the architecture represents the constants and constraints on human activity (such as the speed of mouse pointing movements and short-term memory capacity) the model for a task is able to predict performance on the task, and so can be used to evaluate a design very early in the development process (see [Kieras, 2003] for an overview).

The promise is that these comprehensive cognitive architectures will encourage the development of coherent theory in the science base for HCI, and also provide a high-fidelity way to represent how humans would perform a task. While some would consider such predictive models as the ultimate form of task analysis [Annett and Stanton, 2000b], there is currently a gap between the level of detail required the construct such models and the information available from a task analysis both in principle and in practice [Kieras & Meyer, 2000]; there is not a clear pathway for moving from one of the well-established task analysis methods and a fully-detailed cognitive-architectural model. It should be possible to bridge this gap because GOMS models, which can be viewed as a highly-simplified form of cognitive architectural model [John & Kieras, 1996a, b], are similar enough to HTA that it is easy to move from this most popular task analysis to a GOMS model. Future work in this area should result in better methods for developing high-fidelity predictive models in the context of more sophisticated task analysis methods.

Another area of research concerns the analysis of team activities and team tasks. A serious failing of conventional psychology and the social sciences in general is a gap between the theory of humans as individual intellects and actors and the theory of humans as members of a social group or organization. This leaves HCI as an applied science without an articulated scientific basis for moving between designing a system that works well for an individual user and designing a system that meets the needs of an group. Despite this theoretical weakness, task analysis can be done for whole teams with some success, as shown by [Zachary, Ryder, & Hicinbotham, 2000], [Essens, Post, & Rasker, 2000] and [Klein, 2000]. What is less convincing at this point is how such analyses can be used to identify an optimum team structure or interfaces to optimally support team performance. One approach will be to use computational modeling approaches that take individual human cognition and performance into account as the fundamental determiner on the performance of a team, such as in preliminary work by [Santoro & Kieras, 2001].

The claim that a task analysis is a critical step in system design is well illustrated by the introductory examples, in which entire systems were seriously weakened by failure to consider what users actually need to do and what functionality is needed to support them, and also by the final example, which shows how rather than the usual ad-hoc design of functionality, a task analysis can directly support a choice of functions that results in a useful and usable system. While there are serious practical problems in performing task analysis, the experience of Human Factors shows that these problems can be overcome, even for rather large and complex systems. The numerous methods developed by Human Factors for collecting and representing task data are ready to be used and adapted to the problems of computer interface design. The additional contributions of cognitive psychology have resulted in procedural task analyses that can help evaluate designs rapidly and efficiently. System developers thus have a powerful set of concepts and tools already available, and can anticipate even more comprehensive task analysis methods in the future.

Defining Terms

Cognitive Psychology: A branch of psychology which is concerned with rigorous empirical and theoretical study of human cognition, the intellectual processes having to do with knowledge acquisition, representation, and application.

Cognitive Task Analysis: A task analysis that emphasizes the knowledge required for a task and its application, such as decision-making and its background knowledge.

Functionality: The set of user-accessible functions performed by a computer system; the kinds of services or computations performed that the user can invoke, control, or observe the results of.

GOMS model: A theoretical description of human procedural knowledge in terms of a set of Goals, Operators (basic actions), Methods, which are sequences of operators that accomplish goals, and Selection rules that select methods appropriate for goals. The goals and methods typically have a hierarchical structure. GOMS models can be thought of as programs that the user learns and then executes in the course of accomplishing task goals.

Human Factors: Originating when psychologists were asked to tackle serious equipment design problems during World War II, this discipline is concerned with designing systems and devices so that they can be effectively used by humans. Much of Human Factors is concerned with psychological factors, but important other areas are biomechanics, anthropometrics, work physiology, and safety.

Task: This term is not very well defined, and is used differently in different contexts, even within Human Factors and Human-Computer Interaction. Here it refers to purposeful activities performed by users, either a general class of activities, or a specific case or type of activity.

Task domain: The set of knowledge, skills, and goals possessed by users that is specific to a kind of job or task.

Usability: The extent to which a system can be used effectively to accomplish tasks. A multidimensional attribute of a system, covering ease of learning, speed of use, resistance to user errors, intelligibility of displays, and so forth.

User interface: The portion of a computer system that the user directly interacts with, consisting not just of physical input and output devices, but also the contents of the displays, the observable behavior of the system, and the rules and procedures for controlling the system.

References

Annett, J. (in press). Hierarchical task analysis. In D.Diaper & N.A. Stanton (Eds.), *The handbook of task analysis for human-computer interaction.* Mahwah, New Jersey: Lawrence Erlbaum Associates.

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

Annett, J. & Stanton, N.A. (Eds) (2000a). *Task analysis*. London: Taylor & Francis.

Annett, J. & Stanton, N.A. (2000b). Research and development in task analysis. In J. Annett & N.A. Stanton (Eds), *Task analysis*. London: Taylor & Francis, 3-8.

Baber, C., & Stanton, N.A. (in press). Task analysis for error identification. In D.Diaper & N.A. Stanton (Eds.), *The handbook of task analysis for human-computer interaction.* Mahwah, New Jersey: Lawrence Erlbaum Associates.

Baecker, R. M., Grudin, J., Buxton, W. A. S., and Greenberg, S. (Eds.) 1995. *Readings in human-computer interaction. Toward the year 2000*. San Francisco: Morgan Kaufmann.

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.

Blomberg, J., Burrell, M., & Guest, G. (2003). An ethnographic approach to design. In Jacko, J.A. & Sears, A. (Eds) *The human-computer interaction handbook*. Mahwah, New Jersey: 964-986.

Boose, J. H. 1992. Knowledge acquisition. In *Encyclopedia of Artificial Intelligence*, 2nd edition. New York: Wiley. 719-742.

Byrne, M.D. (2003). Cognitive architecture. In Jacko, J.A. & Sears, A. (Eds) *The human-computer interaction handbook*. Mahwah, New Jersey: 97-117.

Chipman, S.F., Schraagen, J.M., & Shalin, V.L. (2000). Introduction to cognitive task analysis. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 3-24.

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

Diaper, D. & Stanton, N.A. (in press-b) Wishing on a sTAr: The future of task analysis. In D.Diaper & N.A. Stanton (Eds.), *The handbook of task analysis for human-computer interaction*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Dubois, D., & Shalin, V.L. (2000). Describing job expertise using cognitively oriented task analysis. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 41-56.

Essens, P .J. M. D., Fallesen, J. J., McCann, C. A., Cannon-Bowers, J., and Dorfel, G. 1994. COADE: A framework for cognitive analysis, design, and evaluation. Technical Report, TNO Human Factors Research Institute, Soesterberg, Netherlands.

Essens, P.J.M.D., Post, W.M., & Rasker, P.C. (2000). Modeling a command center. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 385-400.

Goransson, B., Lind, M., Pettersson, E., Sandblad, B., & Schwalbe, P. 1987. The interface is often not the problem. In *Proceedings of CHI+GI 1987*. New York: ACM.

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

Gould, J. D. 1988. How to design usable systems. In M. Helander (Ed.), *Handbook of human-computer interaction*. Amsterdam: North-Holland. 757-789.

Grudin, J. 1991. Systematic sources of suboptimal interface design in large product development organizations. *Human-Computer Interaction*, **6**, 147-196.

Holtzblatt, K. Contextual design. (2003). In Jacko, J.A. & Sears, A. (Eds) *The human-computer interaction handbook*. Mahwah, New Jersey: 941-963.

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

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). Amsterdam: North-Holland. 733-766.

Kieras, D.E. (2003). Model-based evaluation. In Jacko, J.A. & Sears, A. (Eds) *The human-computer interaction handbook.* Mahwah, New Jersey: 1139-1151.

Kieras, D.E. (in press). GOMS models and task analysis. In D.Diaper & N.A. Stanton (Eds.), *The handbook of task analysis for human-computer interaction.* Mahwah, New Jersey: Lawrence Erlbaum Associates.

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.

Klein, G. (2000). Cognitive task analysis of teams. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 417-430.

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.

Militello, L.G., & Hutton, R.J.B. (2000). Applied cognitive task analysis (ACTA): A practioner's toolkiet for understanding cognitive task demands. In J. Annett & N.A. Stanton (Eds), *Task analysis*. London: Taylor & Francis, 90-113.

Nardi, B. (Ed.) 1995. *Context and consciousness: Activity theory and human-computer interaction.* Cambridge, MA: MIT Press.

O'Hare, D., Wiggins, M., Williams, A., & Wong, W. (2000). Cognitive task analysis for decision centered design and training. In J. Annett & N.A. Stanton (Eds), *Task analysis.* London: Taylor & Francis, 170-190.

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

Santoro, T., & Kieras, D. (2001). GOMS models for team performance. In J.Pharmer and J. Freeman (Organizers), Complementary methods of modeling team performance. Panel presented at The 45th Annual Meeting of the Human Factors and Ergonomics Society, Minneapolis/St. Paul.

Schaafstal, A., & Schraagen, J.M. Training of troubleshooting: A structured task analytical approach. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 57-70.

Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Seamster, T.L., Redding, R.E., & Kaempf, G.L. (2000). A skill-based cognitive task analysis framework. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 135-146.

Shepherd, A. (2000). HTA as a framework for task analysis. In J. Annett & N.A. Stanton (Eds), *Task analysis*. London: Taylor & Francis, 9-23.

Stanton, N.A. (2003). Human error identification in human-computer interaction. In Jacko, J.A. & Sears, A. (Eds) *The human-computer interaction handbook.* Mahwah, New Jersey: 371-383.

Turner, P., & McEwan, T. (in press). Activity theory: Another perspective on task analysis. In D.Diaper & N.A. Stanton (Eds.), *The handbook of task analysis for human-computer interaction*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Wong, W. (in press). Data analysis for the critical decision method. In D.Diaper & N.A. Stanton (Eds.), *The handbook of task analysis for human-computer interaction*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Wood, S. D. (1999). The Application of GOMS to Error-Tolerant Design. Paper presented at the 17th International System Safety Conference, Orlando, FL.

Woods, D.D, O'Brien, J.F., & Hanes, L.F. 1987. Human factors challenges in process control: The case of nuclear power plants. In G. Salvendy (Ed.), *Handbook of Human Factors*, New York: Wiley.

Zachary, W.W., Ryder, J.M., & Hicinbotham, J.H. (2000). Building cognitive task analyses and models of a decision-making team in a complex real-time environment. In Schraagen, J.M., Chipman, S.F., & Shalin, V.L. (Eds) (2000). *Cognitive task analysis*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 365-384.

For Further Information

The reference list contains useful sources for following up this chapter. Landauer's book provides excellent economic arguments on how many systems fail to be useful and usable. The most useful sources on task analysis are the books by Kirwan & Ainsworth and Diaper & Stanton, and the report by Beevis, Bost, Doering, Nordo, Oberman, Papin, Schuffel, & Streets. A readable introduction to GOMS modeling is B. John's article, "Why GOMS?" in *Interactions* magazine, 2(4), 1995. The John and Kieras, and Kieras references provide detailed overviews and methods.

Acknowledgement

The concept of high-level GOMS analysis was developed in conjunction with Ruven Brooks, of Rockwell Automation, who also provided helpful comments on the first version of this chapter.