Mapping the Method Muddle: Guidance in Using Methods for User Interface Design

JUDITH S. OLSON
University of Michigan

THOMAS P. MORAN
Xerox PARC

A great deal of progress has been made in developing methods that designers can incorporate in appropriate places in the design process with the goal of producing interfaces that are easier to use and learn. This book represents another step in this line, reporting on emerging methods for designers. However, we have not yet succeeded in making user interface methods an established part of software design practice. Many designers claim they do not have enough time for usability. Even when designers want to design for usability, they find that the literature on design methods is a muddle, making it difficult to figure out if there are methods that are appropriate for their situations. They need to know what they can do, and when. The purpose of this paper is to provide that guidance.

In 1983, there was a similar effort. The National Research Council’s (NRC) Committee on Human Factors published the proceedings of a workshop entitled, “Methods for Designing Software to Fit Human Needs and Capabilities” (Anderson and Olson, 1985). Table 1 lists the methods included in that report.¹

| Questionnaires and interviews, diaries, natural observation of task performance, activity analysis, logging and metering for understanding the old situation |
| Task analysis for understanding the requirements for the new task |
| Design guidelines, user reactions, and theory-based judgments to assess the initial design, with support from toolkits to guide designers to select from some well-established design pieces, such as dialog boxes preconfigured and menu bars into which to put command names |
| Formal analyses of interfaces using structured walkthroughs, decomposition analysis, object-action analysis, metaphor analysis, mental model assessment, COMS–keystroke analysis, and grammar analysis |
| Building a prototype through facading, Wizard of Oz, or rapid prototyping |
| Using these prototypes in usability tests |
| Running the proposed final design in friendly field tests (similar to beta test sites for the system code) |

Although the major methods used today are the same as those in the NRC report (Nielsen, 1993), there are some new additions. The collection of papers in this book updates that list to include not only more methods, but also to illustrate their successful use in real design projects. Yet designers still need some help sorting through the methods in order to choose those applicable to the design situation at hand. This help is needed even more so now because the set of methods has grown, some are quite sophisticated, and they differ in applicability to the stage of design and to the kind of task being designed for.

To help orient designers, we begin by a characterization of the design process as consisting of several activities. Methods are then described in abstract form (there are too many to detail here) and associated within the appropriate activity. We then lay out a way for the designer to choose the methods appropriate to key features of the task and users in their design situation: the speed of performance expected of the user, the amount of information content in the task domain, and whether the focus is on rapid learning or expert performance. It is not expected that designers will choose to use only one of these methods, but rather that they will select a set of coordinated methods. We illustrate the use of the methods with scenarios of two contrasting design situations. We end with a discussion about the methods and recommendations for further development of design methods that carry with them the appropriate cost-benefit to the designers.
What Is a Method?

We use the term method as meaning “how to go about designing something.” A method implies a systematic, repeatable way to design. There are macro- and micromethods. A macromethod is a methodology that organizes the whole design process. Software engineering methodologies, like Jackson System Diagramming and Object-Oriented Methods, tend to be macro. Large companies usually advocate or even require formalized design methodologies to manage large-scale design projects. Micromethods are methods that address only subproblems of design. An extended design process would employ several micromethods, which would be used in combination as needed. In this paper, we discuss only micromethods.

A complete method would include:

- A statement of the problem that this method addresses
- A device (a tool, technique, model, or representation)
- A procedure for using the device
- A result

For example, a method called Cognitive Walkthrough (Lewis et al., 1990; Rieman et al., 1991) is complete. It addresses the problem of designing a system to be easy to use when the user first walks up to it. It has a set of forms for the designer to fill out (the device is a representation) that describe the user’s steps in performing a task and a procedure for “walking through” the steps, asking questions about how easily the user will discover how to do the next step. The result is a list of aspects of the user interface that are likely to give the user trouble, which guides the designer to areas of the interface that should be altered.

There are very few complete methods for user interface design. Usually, there is a technique or model or representation, but no explicit procedure for using it. Sometimes the problem addressed is vague, and sometimes the result is implicit. Often these missing aspects can be supplied by the designer in the context of a particular design situation. Therefore, we simply refer to “methods” in this paper, even if they are incomplete.

What Can Be Done to Improve Design?

The key to good systems is the commitment of time and expertise to the user-oriented aspects of the design. The designers have to know more than
computer science and technology; they need training and experience in making systems useful, usable, and acceptable. Methods help guide this effort.

A system has utility when it meets users' needs and solves problems that users have. To accomplish this goal, we need to understand the users' practices and work setting, including what their goals are and how they go about accomplishing those goals. The new system must provide appropriate functionality to meet the needs in the context of the work setting (fitting the organizational culture, the established communication patterns, and the incentive structure, among other things).

The functionality must be offered through a usable interface. The system must be understandable and learnable, relative to the skills of the users and the resources available for learning. The system must have the right performance characteristics: Users must be able to do the tasks fast enough and accurately enough to meet their needs.

Finally, the system must be subjectively acceptable to the users. They must perceive benefit from it and perhaps even enjoy using it. Such acceptance comes from both a well-designed system and a design process that successfully targets the right market and involves users in the design and deployment of the system.

Design methods can help achieve better design. System design is an art that mixes creativity and discipline. Most of the methods listed here help with the discipline, not with the creativity. Of course, regular methods that help with brainstorming ideas and structuring and evaluating them apply to the design process (for example, Adams, 1979). But the design methods reviewed in this chapter are more specific and engineeringlike. They:

- Focus attention on users' needs and capabilities
- Provide tools to represent and build designs
- Encourage thorough thinking and analysis by systematizing design activities and bringing theory and knowledge to bear where it can be useful
- Foster testing and measurement

Focus Attention on Users' Needs and Capabilities

The key feature of good design methods is that they focus on the user. One way to do this is to include users themselves in the design process. When new designs are proposed, users should be involved in testing them. Since users cannot always articulate what they know about a procedure or its goal, it is sometimes important to observe users in their current practice. Even
better, users should participate in the creation of the new system, defining requirements, constructing prototypes, and so forth. Almost all methods discussed in this chapter can be enhanced by having users themselves involved.

Probably the most important step in designing good systems is getting the functionality right. It is most important to clearly understand the needs early in the process. Design methods can help investigate the users’ task domain and current work practices. One can distill important user scenarios, making clear what the critical criteria are. Most methods rely on having a set of tasks or scenarios. Choosing the right ones is important. It is useful to have a mix of core tasks (those that are most central or frequent), critical tasks (those that illustrate the new capabilities), and benchmark tasks (those that can be used to compare the new system with other systems).

---

**Provide Tools to Represent and Build Designs**

Design methods provide useful representations for design. Good representations are important for many reasons. Representations help us see a proposed design in a particular way and come to an understanding of it from that view. Concrete representations help with communication between members of the design team and between designers, users, and other stakeholders. Some representations, called models, allow calculations of properties of the design, such as the number of new things to learn or the time the user would be expected to execute a task.

Prototyping is important in that it provides an understanding of many issues at an early stage, when something can still be done to deal with them. Some design methods provide tools that make it easier to build prototypes and even final implementations.

---

**Encourage Thorough Thinking and Analysis**

Often the greatest value of a representation comes in the effort to create it, which requires careful thinking through of particular issues. Reflecting is an integral part of design. Design methods can aid reflection both by providing representations that “talk back” (Schön, 1983) and by disciplining the designer to be concrete and complete. Methods such as checklists and walk-throughs promote good reflection and thorough analysis of aspects of the design that might cause users difficulty.
Foster Testing and Measurement

We must never assume that designs will work as planned. They need to be tried and tested. Some design methods provide ways to test and measure prototypes and implementations. Methods codify other designers' experience by prescribing particular tests and measures that have been shown to be informative for discovering difficulties and pointing to areas that need redesign.

The Activities That Designers Engage In

System design is a complex process that varies from situation to situation. For example, designing a generic new product for which there is only a description of a market is quite different from designing a custom system for a specific user in a specific setting. The process must accommodate both these large-scale differences and the specifics of the tasks and settings within them. Nevertheless, there are many recurring activities in design, for which the methods reviewed in this chapter are intended to provide support. We here consider seven common activities:

• Define the problem. As we have stressed, the most important thing is to get the problem right. Design theorists (Simon, 1981) have described design as an open problem in that the problem is not given from the start but is clarified as the solution is being formulated. Part of defining the problem is understanding the task domain and current work practices of the user community. From this, the designer must figure out what the needs are that can be met with a new system. Defining the problem is not just an analytic endeavor; it opens up possibilities.

• Generate a design. Once there is some sense of the problem and at least a general notion of the kind of system needed, a more detailed design can be created. Generating a design is very much a creative and constructive activity, involving the building of concrete representations showing what the system would be like.

• Reflect on the design. Once any part of the design is represented, it has to be assessed. This is done by reflecting on it—living with it in the imagination, analyzing it, challenging it, and so forth. This leads to new ideas. Thus, generating and reflecting are tightly coupled activities.

• Build a prototype. A prototype is a concrete representation for which some aspects "work." It could be a physical mockup, a set of pictures, or running
software that can portray the design in a way comprehensible by someone other than designers.

- **Test the prototype.** The point of a prototype is to "try it out" informally or to test it more systematically with potential users. The goal here is to discover fatal flaws, new issues, and aspects of the design that seem to work especially well.

- **Implement the design.** Building the real system involves a commitment to a lot of decisions since the flexibility to alter the design rapidly becomes less practical. Following software engineering principles can help not only with efficient implementation, but also with modularity that will preserve some flexibility.

- **Deploy the system.** As soon as the system implementation reaches a usable stage (well before it is complete), it can begin to be deployed in different ways. There are several strategies for trying it out in limited ways, both to learn if it will serve the intended needs and to get the user community prepared for its installation. One must also be conscious of the fact that once deployed, the task changes; this leads to the need to evaluate iteratively and inform the next generation of products.

We cannot stress enough that these activities are not stages in the design process. The course of design involves continual jumping back and forth between the activities. There is iteration in which feedback from activities later in the list inform those earlier.

Although these activities are common in design, they do not define a complete design and development process. We present them here to provide an organizational framework for the design methods reviewed.

## The Methods

The collection of methods for user interface design have clear relevance for different stages in the design cycle. Some methods help the analyst at the beginning of the project to understand the task and the setting in which the new system will reside. Other methods help generate the working design. Still others are intended for analysis of the proposed design, to help in suggesting improvements in learnability and usability. Methods are clustered and ordered according to the seven activities.

Each section that follows begins with a summary table listing the methods, several descriptors that should guide the designer in deciding whether the
method is useful, and references that describe the method's use in a design setting or show how to do it. The two descriptors we offer at this stage are estimates of how long it takes to do this method (effort) and how much training is required in order to make the prescribed assessments (training). The training is assumed to consist of a short course or tutorial (for example, from the conference called Computer Human Interaction, CHI) plus some exercises and apprenticeship. The person being trained is either a designer or a human factors person, typically someone with a bachelor's degree. The effort is an estimate of how long the analysis or method would take the person so trained. Both of these estimates come from the authors' experience in using these methods as well as teaching these methods and observing their use in a variety of settings. Although many contextual factors, such as background of the analyst and complexity of the application, affect the time estimates, these estimates, at a minimum, show relative values for effort and training.

### Define the Problem

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naturalistic observation (diaries, videotape, etc.)</td>
<td>2 days</td>
<td>3 months</td>
<td>Hill et al., 1993</td>
</tr>
<tr>
<td>Interviews (including focus groups, decision tree analysis, semantic nets)</td>
<td>1 day</td>
<td>1 month</td>
<td>Rudman and Engelbeck, this volume; Nielsen et al., 1986</td>
</tr>
<tr>
<td>Scenarios or use cases (including envisioning)</td>
<td>1 day</td>
<td>1 month</td>
<td>Jacobson, 1992; Carroll, in preparation for 1995</td>
</tr>
<tr>
<td>Task analysis (including operator function model)</td>
<td>2 days</td>
<td>3 months</td>
<td>Johansson, Hannum and Tessmer, 1989</td>
</tr>
</tbody>
</table>

A variety of methods center on detailing the tasks for which the system will be built. In some of the methods, the analyst watches the users do their work through naturalistic observation or analysis of work practice. Data on these activities can be collected by videotaping workers or by workers keeping diaries. In all these methods, the analyst is interested in understanding the practicalities of how the work actually gets done in the current system, the
details of the task requirements, the order in which the users do subtasks, what material or information they need, how they communicate with each other, how much time each subtask takes, and the physical, social, and organization setting in which this work takes place.

Another set of ways of gathering requisite information involve asking users directly about their needs. Focus groups or one-on-one interviews are included in this set of methods. When the task to be supported is entirely new, potential users are encouraged to imagine the new setting and capabilities and are asked what they would do with it and how it would be used, a method called envisioning.

Several innovative methods are introduced in this book. Interviews can elicit from the task performer the particular mental steps they go through, which can be represented in a decision tree. Rudman and Engelbeck (this volume), for example, attempt to reconstruct the sequential cognitive processing that users perform in solving complex problems, such as configuring telephone service for home users in the face of a myriad of offerings and billing arrangements.

For those tasks that have a nomenclature that is complex and foreign to the analyst, semantic nets can represent the organization of objects and the terms used in the task domain. This technique (used by Rudman and Engelbeck [this volume]; see also Gillan and Breedin, 1990) complements the decision tree analysis in that it uncovers complex data structures in the user's vernacular, whereas the decision tree analysis represents the processes that operate on that data. Knowledge engineering methods could potentially be applied to uncover the kinds of strategic activities and knowledge structures people use in domains that are unfamiliar to the designer, as suggested in Olson and Biolsi (1991).

Narrative descriptions of complete tasks are often called scenarios or use cases (McDaniel, Olson, and Olson, 1994)7. While gathering these, there is also opportunity to elicit and discuss the problems users have noted with the current system and their wishes for the future.

Information from these observations can be represented in terms of activity or task analyses, where subtasks are coded and summarized into flow charts, frequency tables, and state-transition diagrams (see, for example, Sasso, Olson, and Merten, 1987). These diagrams can then be used to suggest new ways of accomplishing the same goal. When measures of performance like time or quality are collected as well, these can be used to compare the old system with the new.

The operator function model8 is a specific method for displaying a detailed task analysis. It makes explicit what information the operator or user needs at each moment in the task and the sequence of subtasks that the operator goes through (and the branches, the different things that could be done). This then
can be used to suggest what information must be displayed at each moment, what steps the user should be able to go through, and what options should be presented to the user in an easy-to-access command sequence. It is a relatively small step to go from this detailed specification to an actual prototype. Furthermore, it could well serve as the input to a rapid prototyping system such as ITS (Gould, Ukelson, and Boies, this volume).

### Generate a Design

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building on previous designs</td>
<td>1 day</td>
<td>1 month</td>
<td>Perlman, 1988; Tetzlaff and Schwartz, 1991</td>
</tr>
<tr>
<td>(steal and improve, design guidelines)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Represent conceptual model</td>
<td>1 day</td>
<td>2 months</td>
<td>Moran, 1983</td>
</tr>
<tr>
<td>Represent interaction</td>
<td>2 days</td>
<td>2 months</td>
<td>Kiers and Polson, 1983</td>
</tr>
<tr>
<td>(GTN, dataflow, etc.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Represent visual displays</td>
<td>2 days</td>
<td>3 months</td>
<td></td>
</tr>
<tr>
<td>Design space analysis</td>
<td>3 days</td>
<td>1 month</td>
<td>MacLean et al., 1991</td>
</tr>
<tr>
<td>(QOC, decomposition analysis)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Design is essentially evolutionary. New designs borrow from and improve on previous designs. Even radically new designs are reactions to existing designs. This is a practical matter of the "reuse" of designs that already work and that users are familiar with. Thus, designers do not have to start from scratch, and users do not have to learn yet again.

There are two distinct "modes" of borrowing: global and local. One can start by adopting the global models of existing systems. Card (this volume) calls this steal and improve. For example, there are many common concepts among word processing systems, among database systems, and among spreadsheet systems. You would want to think hard about how and why you would want to be different if you were developing one of these kinds of systems. You can also adopt localized design components, such as user interface "widgets," many of which appear in toolkits. When the designer adopts a toolkit, other design decisions are already made as well. For example, in the Macintosh toolkit, the designer has no choice about how menus appear and
how selections are made. And embedded in the toolkit instructions are guidelines on how to make some interface features match other systems of similar type, such as the guide to make Macintosh applications all have Apple, File, and Edit as the leftmost three menu items.

Design guidelines exist for some significant portion of the components of a user interface, which are helpful to this initial design. For example, guidelines offer prescriptions about how to organize items on a set of menus, with ordering either by conceptual category (for example, editing commands) or by frequency of use (for example, Smith and Mosier, 1984). Other guidelines suggest the consistent placement of help and warning messages on the screen, and general use of gestalt principles of proximity and dissimilarity to capture and guide the user's eye to appropriate portions of the display.

Although borrowing can often provide a starting place for design, especially in domains where there are existing systems, the goal of design is to create a system that addresses the particularities of the problem at hand. Generating a design involves creating descriptions of the design, and different kinds of design representations are essential for this activity.

A conceptual model is the set of conceptual entities that the system represents, that the user needs to understand to effectively use the system. It is important that the designer be clear about this and explicitly represent the conceptual model (Newman and Sproull, 1979). For example, understanding the hierarchical nature of text objects (characters, words, sentences, paragraphs, sections, chapters, documents) separately from the features of layout (font, line, page, margin) helps construct appropriate actions and associated specifications. Moran (1983) proposed a representation of the relationship between the concepts in the users' work domain (uncovered, say, by interviews or some form of semantic analysis) and the conceptual model embodied by the system. A useful specific version of this is an object-action analysis. For example, Moran used this kind of representation to assess the differences between line-oriented and full-screen editors, showing that the conceptual model of the former (strings of things to be replaced) required a translation from the conceptual model of the way we normally think of text.

In Object–Action analysis, the nouns and verbs of the domain and task are arrayed in a table. For example, in text editing, the table would show objects such as letters, words, sentences, paragraphs, pages, and documents on one dimension, and actions such as copy and delete on the other. The table shows the complete matrix of commands that have to be offered. But the analysis shows additional features of the domain model—how characters and sentences are to be treated differently from layout features like lines and pages. These tables encourage completeness of actions on all objects and consistency where appropriate. Such an analysis was one of the key design methods used in the construction of the Xerox Star system, which resulted
in the use of various “universal command keys,” such as delete and copy (Smith et al., 1982).

Once the domain is known and the objects and actions specified, it is useful to represent dynamics of the interaction. State transition diagrams are the most common form of representation of interaction. They are networks of states (screens or modes) where all the possible actions that can be taken at each state are drawn leading to the next states. A simple form of this is the diagram of connected menu items that some training manuals show as a summary of the system’s functionality. A more sophisticated representation is the generalized transition network (Kieras and Polson, 1983), which accommodates the hierarchic nesting of contexts.

The commands and flow of the interaction are depicted in the representations described earlier. The visual display is best represented by renderings of the display on paper or other media. In PICTIVE, analysts give participants various “piecemeal” of interfaces made out of paper or plastic (for example, menus, dialog boxes, scroll bars, and windows) to arrange in an interface layout. When these pictures (full screenshots) are displayed on a board in the order in which they will appear, they are called a storyboard, similar to those used in the construction of films. The closer the representation to the final embodiment of what the user will see, the better the analyses of the arrangement, its attention-getting abilities, and the ease of finding what the user needs to know at each moment.

Design involves discovering and evaluating many different possible designs—a “design space”—and discovering the critical issues or questions that must be addressed. Often it is useful to keep track of the possibilities so that it is clear why particular design decisions are made. Formal analyses of these possibilities come from two methods: Design Space Analysis and Decomposition Analysis. Both involve systematic processes by which various issues or questions about the design are raised and recorded, and then different alternatives, options, or solutions are offered and analyzed with agreed-on criteria. Design Space Analysis takes many forms; QOC (for Questions-Options-Criteria) serves as a good example (MacLean et al., 1991). In QOC, the designer(s) systematically documents the questions to be addressed. Attached to each question are the various alternative solutions. Appropriate criteria (for example, consistency and ease of programming) are then applied to each alternative, leading to a design decision. Decomposition Analysis is similar, except less formal and at a coarser grain (Olson, 1985). Here, the major components of the interface are the representation of the underlying data structure, the command entry style, the provision of memory aids, the access to help, and so on. Each is examined in turn, alternatives are generated, and they are evaluated by any of a variety of means. In both of
these methods, it can be seen that generating a new design and reflecting on it are tightly intertwined.

**Reflect on the Design**

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checklists</td>
<td>1 day</td>
<td>1 week</td>
<td>Shneiderman, 1992</td>
</tr>
<tr>
<td>Walkthroughs</td>
<td>2 days</td>
<td>3 months</td>
<td>Lewis et al., 1990</td>
</tr>
<tr>
<td>Mapping analyses (task action, metaphor, consistency)</td>
<td>2 days</td>
<td>2 weeks</td>
<td>Douglas and Moran, 1983; Payne and Green, 1986</td>
</tr>
<tr>
<td>Methods analyses (GOMS, KLM, CPM, CCT)</td>
<td>3 days</td>
<td>1 year</td>
<td>Card, Moran, and Newell, 1983; Kieras, 1988</td>
</tr>
<tr>
<td>Display analyses</td>
<td>3 days</td>
<td>1 year</td>
<td>Lohse, 1991</td>
</tr>
</tbody>
</table>

Once we have some explicit representations of the design, there are a variety of ways to begin to assess the projected usability of this design as well as the ease of learning it. The methods listed here can also be used to some extent in generating designs. But these methods are detailed and require a fairly specific representation of a design to work on. They can be used either by analysts working alone, with users as part of the design team, or by analysts watching users trying to use a representation of the interface to perform some test task.

The quickest way to evaluate a first-cut or set of alternative components of the design involves answering a set of questions from a checklist (Shneiderman, 1992; Nielsen, 1993). Checklists and usability heuristics serve as memory aids to designers, reminding them about prescriptions for the ease of learning and ease of use for each of the major components.

Just as programmers do a code walkthrough to check how the flow of the communication proceeds in the program and to check for things missing or conflicting, user interface designers perform a walkthrough of the user interface (Weinberg and Freidman, 1984). Here, however, the flow is from the user's perspective, where the user has goals in mind and tries to perform the...
actions necessary to accomplish those goals. With a good set of core tasks as
test cases, a number of errors can be detected, especially in the flow of actions
(whether they fit the order in which the user thinks of the actions) and in the
availability of all the subfunctions needed.

Two variants of the walkthrough are the cognitive walkthrough (Lewis et al.,
1990; Riemann et al., 1991) and the claims analysis (Carroll and Rosson, this vol-
ume). Like the walkthrough, they begin with the analysts generating a core
set of common tasks and the detailed step-by-step listing of what the user has
to do to accomplish these. The analyses that follow, however, are much more
explicit than in a standard walkthrough. The methods provide sets of ques-
tions the analyst is to ask about the interface. These questions are designed to
highlight those aspects of the interface that are known to be difficult or error
inducing. Claims analysis additionally encourages an explicit discussion of
design tradeoffs, similar in style to the decomposition analysis.

Task-mapping analysis begins with a formal representation of the
goal–action mapping that the user conceives of and lays alongside it the
goal–action mapping that the system requires (Polson and Kieras, 1985). The
analysis of the side-by-side diagrams shows mismatches that can turn into
difficulty for the user to learn or execute. For example, a system that makes
you move a range of text by selecting the range to be moved first, then the tar-
get, then the action “move,” mismatches the normal English command
sequence that begins with the word move and continues by specifying the tar-
get material and the “move-to” location. Although most word processors
today follow the noun–verb format for commands, the novice in word pro-
cessing must learn that particular word order since it is not fitting the order
they have learned from spoken language.4

Similarly, Douglas and Moran (1983) suggest a careful analysis of the
metaphor chosen for learning a new piece of software. They showed that by
lining up the goal–action pairs of a target system (for example, a text editor)
and a metaphor system (for example, a typewriter), there were a number of
mismatches, some of which could significantly impede a new learner’s
understanding of the system. Since new learners will construct a metaphor or
mental model even in the absence of instruction (Halasz and Moran, 1983),
care should be taken to choose a helpful one and to teach it early.

Object–action analysis and state transition diagrams, described earlier, can
have benefit in analyzing designs as well as in generating the first design.
Analysis of the grammar of the command language, such as Moran’s Command
Language Grammar (CLG) (1981), Reisner’s formal grammar (1984), and
Payne and Green’s Task Action Grammar (TAG) (1986), first represent the
rules by which components form to produce a language of commands.
Argument is made that the smaller the number of rules in the grammar, the
easier the system will be to learn. The importance of these mapping analyses
is that they focus the designer on the relationship between the elements of the system being designed and the users' task domain and previous knowledge rather than viewing the design in elegant but unnatural isolation.

Card, Moran, and Newell (1983) generated various ways to assess the moment-by-moment cognitive/perceptual/motor resources being used when interacting with a particular device, pioneering the field of cognitive engineering. The core idea is that for certain kinds of well-learned tasks, one could model the goals the user had, the methods offered by a system to satisfy these goals, the choices people made in varying circumstances, and the operator sequences that followed. From this model, called GOMS, and a related, more detailed model called the Keystroke Level Model, one can fairly accurately predict how long a task will take (Olson and Olson, 1991). John (1988) introduced Critical Path Analysis which, in contrast to the GOMS model that assumes a sequential flow of cognitive processes (the recognize–retrieve–act cycle), recognizes that some tasks involve some cognitive processes that occur in parallel. This is often appropriate when the task to be modeled is performed repeatedly and rapidly in a high-performance situation (see Atwood, Gray, and John, this volume).

Cognitive Complexity Theory, another important extension of the original cognitive engineering modeling of Card, Moran, and Newell, represents the knowledge needed to perform these tasks. Using this theory, Kieras and Bovair (1986) were able to predict how long a task would take to learn. All these detailed analyses highlight the portions of the task that will take longer than necessary (for example, too many things to remember or an overloaded working memory that generates errors), focusing redesign efforts to concentrate on very particular interaction details.

Several methods have arisen to assist in display analysis. Tullis (1988) and Mackinlay (1986) developed programs to assess the crowding and thus readability of various aspects of a display. And, more recently, Lohse (1991) has constructed a perceptual simulation that will take a display as input and calculate how long it will take to answer a particular question about the display or to find certain critical features.

---

**Build a Prototype**

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototyping tools</td>
<td>1 month</td>
<td>3 months</td>
<td>Wilson and Rosenberg, 1988</td>
</tr>
<tr>
<td>Participatory prototyping</td>
<td>1 week</td>
<td>2 months</td>
<td>Muller, 1991</td>
</tr>
</tbody>
</table>
These methods analyze the plans or requirements of the system. They are conducted by the analyst, without direct involvement with the users of the system. To date, no one has found these analyses to be sufficient to find all the design difficulties (Nielsen, 1992; Karat, Campbell, and Fiegel, 1992). All comparative studies of methods of design of user interfaces have found value to having users actually attempt to perform a realistic task using some form of the interface.

Further, although representations of designs produced in the initial generation are sufficient for the analysis of Reflection, they are not concrete. They can be understood only through the narrow lenses of the particular analyses. A concrete working prototype is needed in order to obtain rich empirical and experiential feedback.

The system used in these evaluations need not be the final system. The prototype can be presented effectively with paper, storyboards, and other media. One can produce either sequences of screens similar to a movie production storyboard or a complex book of printed screens whose sequencing is controlled by a human analyst. These allow rapid testing for flow of control, visual clarity, and so on, without having to program a system to be fully operational. A variant of these simple prototypes is embodied in PICTIVE (Muller, 1991). When the end users put the requisite pieces of the interface together, it is called participatory prototyping (Poltrock and Grudin, this volume). Although this kind of prototyping might work at this stage, it is more likely to be effective as a first cut that can then be further refined through analysis.

Toolkits provide easy, cost-effective ways to construct a working interface for analysis and testing (Perlman, 1988; Hix and Schulman, 1991). ITS (Gould, Ukelson, and Boies, this volume) discusses ways to display the dialog design with various visual options.

## Test the Prototype

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open testing (storefront or hallway, alpha, and damage testing)</td>
<td>1 week</td>
<td>1 month</td>
<td>Gould et al., 1987</td>
</tr>
<tr>
<td>Usability testing</td>
<td>2 weeks</td>
<td>1 year</td>
<td>Gould, 1988</td>
</tr>
</tbody>
</table>

Once the system is mocked up using one of these methods, the users are asked to work through a sample realistic task while the analysts collect
various forms of data about users' performance. These can be reactions to attractiveness or appeal, ease of learning how to use it, or other characteristics of the user's ease in performing basic tasks. This method is variously called storefront or hallway testing, best exemplified in the design process used for the Olympic Messaging System (Gould et al., 1987). In alpha testing, the prototype is given to associates, who then give feedback to the designers about usability: in damage testing, users deliberately try to break the system, giving feedback to the designers about the system's robustness.

More formal analyses involve full-fledged usability tests, in which users are taught the system (which itself provides an early test of the training materials) and asked to perform a set of tasks. Early tests of the system often involve "critical tasks" that push the system and the user's capabilities so that they would be sure to see its fragile points. In other situations, when the goal is to find expected times to learn and perform, more conventional, common tasks are used, called benchmark tasks.

A whole variety of measures are possible, including the time to learn, the time to perform particular tasks, individual keystroke times (for assessment of match to predictions from the Keystroke Level Model), error types and frequencies, thinking aloud (for assessment of goals and problem-solving strategies), preference, and satisfaction. The data from usability tests are relatively easy to collect; one can tell fairly quickly whether there are major design errors. More detailed comparison of moment-by-moment keystroke times with those projected from cognitive engineering allow designers to focus on those aspects that seem to present difficulties to the user. What is not easy is fixing these difficulties, especially since every design decision involves trade-offs; each fix changes some other aspect, the overall change needing retesting or analysis.

---

**Implement the Design**

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkits (Motif, NaXTstep, Apple, etc.)</td>
<td>3 months</td>
<td>6 months</td>
<td>Perlman, 1988</td>
</tr>
</tbody>
</table>

The advantages of building the prototype in a full-scale toolkit center on the fact that the interface is not only easy to build, has style guidelines built in, and is relatively easy to change after usability testing, but also that toolkits generate production code, unlike prototypes built in some system like

17
HyperCard. With toolkits, it is not necessary to rewrite the interface into the language of implementation.

**Deploy the System**

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal testing</td>
<td>1 week</td>
<td>1 month</td>
</tr>
<tr>
<td>Beta testing (logging, metering, surveys)</td>
<td>2 weeks</td>
<td>1 month</td>
</tr>
</tbody>
</table>

Once a system is judged satisfactory, it is typically tested further, first in the local environment and then in an outside friendly environment, often a site that would like to be an early adopter of the technology in exchange for feeding back discovered bugs and mismatches in design. These tests are often called beta testing. At this point, data are often collected, but data of a less fine-grained sort. Two good sources are cataloged queries that come in on a help line and answers to questionnaires sent to customers or end users. It is also possible, in some situations, to log or meter the new system, just as one would do on an existing system. With keystroke data collected, for example, one can infer both what common tasks are being done efficiently or not and overall use of system features.

---

**What the Designer Needs to Know to Choose a Method for the Right Time and the Right Task**

The organization of the methods in the preceding list conveys that they apply to different activities of the development process. They also differ in the amount of time they require, the amount of detail uncovered, and the accuracy of the conclusions that result. For example, using a checklist on the current or proposed design takes several hours and produces general recommendations about usability and learnability. The checklist can help determine which of two competing software packages might be easier for the end user, but will not provide enough detail to determine how long the task will take or what skill or domain knowledge the end user will have to behave
accurately and with reasonable speed. In contrast, GOMS analysis and its partners, CCT and Keystroke Level Model, require a great deal of time, but provide the detail necessary to say what users will have to know to perform the tasks well, roughly how long it will take to learn, and how long representative tasks will take to perform.

We also noted that the methods differ in how much the designer needs to know about human cognition, perception, and motor movement. Task analyses require very little such knowledge, as do some forms of checklists or guidelines and the cognitive walkthrough, whereas claims analysis, Cognitive Complexity Theory, and Keystroke Level Model require a great deal.

That is, the methods differ in:

- The amount of time they take and the concomitant level of detail and risk associated with the findings
- The knowledge the analyst is required to have about basic cognitive processes of users

Some methods are particularly suited for some kinds of users and tasks and less so for others. This is probably the most difficult thing for the designer to assess. For example, tasks such as information retrieval, financial planning, piloting an airplane, and rapid transcription of text are very different in how sequentially and deliberately the user goes through the steps in the task. There have been numerous attempts to develop a task taxonomy (see, for example, Lenorovitz et al., 1984 for a short review), but in general the taxonomies are far too detailed for the use we wish to put them to here. However, the analyst does need some guidance on which of the methods suits the particular user’s tasks under consideration.

For our purposes, the following seem to be the major dimensions on which a wide set of users’ tasks differ:

1. The task is performed either as a set of sequential steps or as a rapidly overlapping series of subtasks.
2. The task involves either high information content, with consequent complex visual displays to be interpreted, or low information content, where simple signals are sufficient to alert the user that the next step is to proceed.
3. The task is intended to be performed either by a layman without much training or by a skilled practitioner in the task domain.

The first dimension has to do with whether the user’s actions are deliberate and single-minded, much like using a spreadsheet. This contrasts with tasks such as air traffic control, where attention rapidly shifts between input streams and where goals are intertwined. Air traffic control similarly is high in information content, the second dimension, whereas the task of assigning a
student a workstation in a computer lab is low in information content, much
more like reacting to a simple signal (the request of the student). The third
dimension reflects the assumed knowledge or skill level of the users. A bank
teller machine has to be recognizable by any customer, whereas a computer-
aided design (CAD) system is specific to a professional domain with its own
shared vocabulary and can be designed with the assumption that the designer
will be trained in its use.

Most of the methods are applicable to both sequential and overlapping
tasks. The one major exception is the GOMS-CCT family of models and
accompanying analyses. They fit those tasks that comprise subsets of sequen-
tially performed operators (either mental or motor). The Critical Path
Analysis grew from this set of models to explicitly accommodate rapid-fire
tasks that most likely involve cognitive and motor components that overlap
in time (see Atwood, Gray, and John, this volume).

When tasks are rich in information content, it is important both to deter-
mine the structure of the information as the user understands it and to display
it in a representation that visually maps well to that understanding. There-
fore, those methods that assess the organization of information objects and
actions, the mental model of the system, and the analysis of particulars of the
perceptibility of visual displays are particularly relevant.

Interfaces for tasks that are designed for casual use by the layperson, that
do not assume knowledge in a particular domain, should be assessed in par-
cular for their learnability and the provision of information on the screen
that suggests to the user what to do next. Several methods, such as the cogni-
tive walkthrough, storefront analysis, and claims analysis, are particularly rel-
evant for assessing this aspect of the interface.

If the task will be performed by a large work force of dedicated users, then
the more detailed methods, like GOMS, grammar analysis, and Critical Path
Analysis, will likely provide significant payoff. For example, there is a sig-
ificant work force that reconciles mismatches in customer-claimed deposits and
the accounting ledger in a bank “back room." These people work full days at
a rapid pace. It is particularly important in this task that the information that
the user needs to access to solve a problem be placed on the screen in tandem
and that the key information is readily readable. Careful analysis of eye
movements, clarity of font, and keystroke or command sequence is very
important to a good design in this task so that information is not lost out of
the user’s working memory and so that extra scans are not required to “line
up" the aspects of the accounting that mismatch. Good screen design can
shorten each reconciliation task by seconds. Although mere seconds are
saved, when multiplied by the number of tasks accomplished per day and the
number of operators doing such a task, the savings could accrue to millions of
dollars.
Some of the methods, like GOMS, CCT, Keystroke, and grammar analysis, require weeks to do for any medium-size task and system. They are very detailed, cataloging not only the action steps of the potential user, but the cognitive/perceptual/motor steps as well. They provide, however, a great amount of detail. They are therefore appropriate only when that kind of investment in time will reveal important details of the speed of interaction or complexity that might produce significant errors. They have shown value in situations where new operator workstations are being designed for high-speed work (Atwood, Gray, and John, this volume) and for situations where errors are very costly, such as wrong business decisions caused by the wrong data being retrieved from a large database because of its complex user interface (for example, Smelcer, 1989).

These time-consuming methods often also require detailed knowledge about cognition. GOMS family of models and methods require the analyst to know facts about when in a task information might reside in short-term memory and how far an eye movement will jump on a visual display of certain size. Even the claims analysis requires intuition about these processes to help discover what the appropriate and inappropriate claims are that the artifact embodies. In contrast, methods like checklists and walkthroughs often can be conducted by people without an intimate knowledge of cognition and perception, and are therefore at the same time faster to accomplish and less accurate. User testing often takes several weeks to accomplish (including building the prototype, watching the users, and analyzing the results), but can be done by careful, though not necessarily trained, observers.

Summary of Costs and Benefits

To provide guidance to the designer, we have prepared Table 2, which highlights four characteristics:

The type of the method—some collect data (empirical), some are analyses of the structure of the task and interface (analytic), and some construct various representations of the interface (constructive)

The benefits of the method in terms of what aspect of the interface the method is particularly suited to reveal—the task steps, the performance or learnability, or the user's acceptance of the system (called tasks, perform, learn, or accept in the table)

Two aspects of the costs of using the method—the effort to use it (which often correlates with amount of detail) and the training needed

Table 2 provides a rough assessment of these characteristics. It was constructed and synthesized by the authors, guided by input from the members
<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Benefits</th>
<th>Costs—Effort</th>
<th>Costs—Training</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEFINE THE PROBLEM</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naturalistic observation</td>
<td>Empirical</td>
<td>Tasks</td>
<td>2 days</td>
<td>3 months</td>
</tr>
<tr>
<td>(diaries, videotape, etc.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interviews (including focus groups, decision tree analysis, semantic nets)</td>
<td>Empirical</td>
<td>Tasks</td>
<td>1 day</td>
<td>1 month</td>
</tr>
<tr>
<td>Scenarios or use cases (including envisioning)</td>
<td>Analytic</td>
<td>Tasks</td>
<td>1 day</td>
<td>1 month</td>
</tr>
<tr>
<td>Task analysis (including operator function model)</td>
<td>Analytic</td>
<td>Tasks</td>
<td>2 days</td>
<td>3 months</td>
</tr>
<tr>
<td><strong>GENERATE A DESIGN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building on previous designs (steal and improve, design guidelines)</td>
<td>Constructive</td>
<td>Tasks, perform, learn, accept</td>
<td>1 day</td>
<td>1 month</td>
</tr>
<tr>
<td>Represent conceptual model</td>
<td>Constructive</td>
<td>Learn</td>
<td>1 day</td>
<td>2 months</td>
</tr>
<tr>
<td>Represent interaction (GTN, dataflow diagram)</td>
<td>Constructive</td>
<td>Perform, learn</td>
<td>2 days</td>
<td>2 months</td>
</tr>
<tr>
<td>Represent visual display</td>
<td>Constructive</td>
<td>Perform, learn</td>
<td>2 days</td>
<td>3 months</td>
</tr>
<tr>
<td>Design space analysis (QOC, decomposition analysis)</td>
<td>Analytic</td>
<td>Tasks, perform, learn</td>
<td>3 days</td>
<td>1 month</td>
</tr>
<tr>
<td><strong>REFLECT ON THE DESIGN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checklists</td>
<td>Analytic</td>
<td>Perform, learn</td>
<td>1 day</td>
<td>1 week</td>
</tr>
<tr>
<td>Walkthroughs</td>
<td>Analytic</td>
<td>Perform, learn</td>
<td>2 days</td>
<td>3 months</td>
</tr>
<tr>
<td>Mapping analysis (task action, metaphor, consistency)</td>
<td>Analytic</td>
<td>Perform, learn</td>
<td>2 days</td>
<td>2 weeks</td>
</tr>
<tr>
<td>Methods analysis (GOMS, KLM, CPM, CCT)</td>
<td>Analytic</td>
<td>Perform, learn</td>
<td>3 days</td>
<td>1 year</td>
</tr>
<tr>
<td>Display analyses</td>
<td>Analytic</td>
<td>Perform, learn</td>
<td>3 days</td>
<td>1 year</td>
</tr>
<tr>
<td><strong>BUILD A PROTOTYPE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping tools</td>
<td>Constructive</td>
<td>Testable system</td>
<td>1 month</td>
<td>3 months</td>
</tr>
<tr>
<td>Participatory prototyping</td>
<td>Empirical</td>
<td>Tasks, accept</td>
<td>1 week</td>
<td>2 months</td>
</tr>
<tr>
<td>Method</td>
<td>Type</td>
<td>Benefits</td>
<td>Costs—Effort</td>
<td>Costs—Training</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
<td>-------------------</td>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>TEST THE PROTOTYPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open testing (storefront or</td>
<td>Empirical</td>
<td>Perform, learn, accept</td>
<td>1 week</td>
<td>1 month</td>
</tr>
<tr>
<td>hallway, alpha, damage testing)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability testing</td>
<td>Empirical</td>
<td>Perform, learn, accept</td>
<td>2 weeks</td>
<td>1 year</td>
</tr>
<tr>
<td>IMPLEMENT THE DESIGN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toolkits (Motif, NeXTstep,</td>
<td>Constructive</td>
<td>Fully testable system</td>
<td>3 months</td>
<td>6 months</td>
</tr>
<tr>
<td>Apple, etc.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEPLOY THE SYSTEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal testing</td>
<td>Empirical</td>
<td>Perform, learn, accept</td>
<td>1 week</td>
<td>1 month</td>
</tr>
<tr>
<td>Beta testing (logging, metering,</td>
<td>Empirical</td>
<td>Tasks, perform, learn, accept</td>
<td>2 weeks</td>
<td>1 month</td>
</tr>
<tr>
<td>surveys)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

of the workshop at Boulder. The table is intended to advise when a method will or will not be useful.

Examples of Coordinated Use of the Methods

Table 2 provides some guidance to the selection of methods for the particular design task at hand. But just as good cookbooks give not only selection criteria for individual dishes but also suggest combinations of dishes to create a pleasing meal, we provide here two such “meals.” The first illustrates the use of quick methods for a simple walk-up-and-use system, such as an ATM. The second illustrates the set of design methods at the other end of the continuum, where the interface is information rich, and speedy, accurate real-time performance is critical to the operator’s success.

The literature contains several other descriptions of a coordinated set of methods. Gould’s description of the development of the Olympic Messaging System (Gould et al., 1987) demonstrates the coordinated use of several methods for walk-up-and-use interfaces, and the description by Rudman and Engelbeck (this volume), about the development of an interface for the
operator's support for configuring new telephone service, demonstrates coordinated methods for an information-rich, interactive system involving customer conversation. McDaniel, Olson, and Olson (1994) describe the use of a combination of HCI methods, those from Business Process Redesign (Hammer and Champy, 1993) and from Object-Oriented methodology (Jacobson, 1992) in the design of a system to help space physicists access remote sensors and converse about them across several continents (McDaniel, Olson, and Olson, 1994).

---

**Coordinated Methods for Quick Evaluation of a Walk-Up-and-Use System**

An ATM is an example of a system that:

- Has simple sequential task flow (that presents information on choices the user has, each of which leads to new choices)
- Is relatively low in information content (mainly verbal selections, for example, about withdrawal or deposit and how much)
- Is targeted for the layman

Because the task is performed by untrained users at their own pace, the emphasis is on the ease with which the user can learn to operate the device. Obviousness of what action to take next and error recovery are prime. Moreover, since the business objective of this system is not rapid performance of tasks, but rather widespread use leading to offloading clerical tasks to the customer, the budget for construction and evaluation is likely small. Fast methods will do, and the designer should not be expected to have a Ph.D. in cognitive psychology.

To discover the components of the task, simple questionnaires might suffice, asking the potential customers what kinds of choices they might be interested in. Often marketing has the basics of this information collected already, and use of this for the interface objects, actions, and flow will serve well. Since lots of ATMs exist already, it would also be appropriate to do some naturalistic observation, where designers observe current users at existing machines.

The initial design, guided by prescriptions from guidelines and assessed quickly with checklists, might be printed on paper and displayed as a storyboard, for analysis of flow, screen display, and so on without users. Designers can view the storyboard for aspects of ease of learning, using in particular a cognitive walkthrough. The flow of the system could be assessed with a simple generalized transition network, to assure consistency in the use of error recovery and navigation commands.

For hallway testing, a mockup of the entire display might be constructed,
with a rapid prototyping system (such as HyperCard) embodying the design. Designers can observe the test users' difficulties or get them to think out loud while they attempt to use the system.

After several such short analyses and redesigns, the system, in its penultimate form, can be installed at a friendly test site, and some basic system monitoring data can be collected for analysis of gross usability and preference characteristics.

---

**Coordinated Methods for Detailed Evaluation of a High-Performance System**

At the other end of the spectrum is a system that supports back-room workers at a bank who are reconciling the machine-read check register with what the customer wrote on the back of a deposit slip. The task:

- Requires the overlapping activation of the user's mental and physical capabilities (scanning the next set of materials while the previous tasks' corrections are keyed in)
- Is relatively high in information content (side-by-side displays of the deposit slip's handwritten entry and a list of the checks accompanying the deposit, both machine read and scanned in true copy)
- Is targeted for the dedicated, skilled user

Because the task is performed all day every day by skilled users, there is considerable payoff from having detailed, somewhat time-consuming analyses. The outcome has to be detailed enough to recommend changes to the interface that may bring about seconds of savings in each task completion. But because of the high volume of performance of each task, savings accrue rapidly. Budget for construction and evaluation of this kind of system can be quite large, given the anticipated payoff.

To understand the task, which in this case is not particularly obvious to the system designer, several discovery techniques should be employed. If there is a previous system in place (check balances are reconciled in some way before this new system is built), the designers can engage in some natural observation, plus interview the workers about aspects that are difficult or annoying. More detailed discovery of the objects and actions of the task domain and the kind of thinking that goes on during the execution of the task can come from semantic net interviews, decision tree interviews, and other techniques from the area of knowledge engineering. A detailed task analysis is performed next, showing the order of subtasks and the kinds of information that are needed at each moment so that the user can perform requisite cognitive tasks to accomplish the goal. The task analysis may take the form of an operator function...
model, with details of the knowledge necessary in the form of a GOMS model or some parts of the Task Action Grammar.

Once the task is fully understood, a series of design and evaluative iterations follow. The system can be generated and displayed as a Generalized Transition Network or a working prototype, using one of the more sophisticated toolkits like ITS. This design is then analyzed in detail for the cognitive and motor movements required to accomplish the task using the Critical Path Analysis (CPA) variant of the GOMS family of models. Since the system has high information content, detailed analysis of the visual display is also warranted. Usability tests follow, with particular emphasis on the fit of the CPA to the actual timing of the task, to hone both the model’s accuracy and to show where the system does not fit the predictions of optimal performance afforded by the model. The design iterates until the users’ performance meets preset target criteria for skilled performance.

Discussion

The preceding synthesis and Table 2 are intended to be helpful, not to provide a detailed critique of each method for its usefulness. There are methods, for example, that have the goal of being useful and usable by designers but, at present, are in a form that makes them difficult to learn or awkward to use. For example, one of the motivations for providing the Cognitive Walkthrough was to make knowledge that is gleaned from GOMS and other empirical investigations accessible in a method usable by designers.

Further, the table format masks the potential synergy between methods, those useful links that can occur between methods. For example, the Operator Function Model is a detailed analysis of the object, actions, and flow of control necessary for task performance. On inspection, we discovered that its outputs are exactly what are needed for input to ITS. Thus, whereas one method might be effective only on its own, others may have useful links between them.

It is also the case that the table makes only crude assessments of the kinds of tasks that it can be applied to and coarse-grained estimates of how much effort it takes and how much training the designer has to have in order to use the method successfully. Of particular concern is the implication that those methods that take a short time but give you broad coverage of evaluation are higher on the cost–benefit curve and therefore more valuable. Some of the methods, such as GOMS, although they take a long time to specify, have a large payoff for several different aspects of the design process. For example, once the task is specified in a GOMS terminology, not only is it possible to
estimate how long a task will take (by using parameters in the Keystroke Level Model), but you also have the basic information necessary to write effective documentation (Gong and Elkerton, 1990). The GOMS model forces the analyst to understand the major tasks and the recommended procedure to accomplish those tasks, the basic elements of writing good training material.

A listing of methods misses some of the more global design process principles that successful designers offer. For example, it has been widely recognized that an effective management procedure for assuring adequate attention to the user interface is to incorporate metrics of user acceptability into the same set of metrics that software designers are used to having to determine if the performance of the software itself is acceptable (Good et al., 1984). There are other principles for effectively using the methods in the design process. Having the software developers themselves on the team that runs a usability evaluation is recommended because they can see for themselves in real time that aspects of a current design provoke repeated difficulties across users. Summary reports do not convey the same weight for such conclusions as do real-time experiences. And it has long been recommended that users themselves sit on the design team to assure adequate input of task vocabulary, completeness of features, and flow that fits the way the user thinks about the task progression. Many of the methods listed could benefit from users being on the design or analysis team.

This overview shows that there is considerable progress in providing ways to design useful, usable, and learnable user interfaces. Many new methods have been developed since the 1983 NRC report, and recent studies have compared the cost-benefit of various methods (Nielsen, 1992, 1993). We have provided a framework for seeing the roles of different methods, but more work is needed on a detailed cost-benefit of the methods. Not only do the methods need to be assessed for their usefulness, but new methods need to be developed that are more complete and usable.

---

**Notes**

1. Of the participants in this book, many were also participants in the NRC Committee on Human Factors workshop in 1983. Stu Card, Jack Carroll, Judy Olson, and John Whiteside were at both. Others in the 1983 workshop included Nancy Anderson, Elizabeth Bailey, Alphonse Chapantis, Rex Hartson, David Lenorovitz, Marilyn Mantel, Dick Pew, Phyllis Reisner, Janet Walker, and Bob Williges.

2. Published reports (for example, Nielsen, 1992; Karat, Campbell, and Fiegel, 1992) include other numbers as estimates of the time to perform some of these analyses, but they are reporting actual times for specific, small design situations. The numbers here are intended to be more wide ranging, applying to more real-world design situations.
3. Use cases are essential components of the new Object Oriented Methods, now increasing in favor in the software design community (Jacobson, 1992).

4. Christine Mitchell presented the operator function model at the workshop but that paper is not included in this volume. See Mitchell (1987); Mitchell and Saisi (1987); Dunkler et al. (1988); Jones, Rubin, and Mitchell (1989); Smith, Govindaraj, and Mitchell (1990); and Chu, Mitchell, and Govindaraj (1989).

5. This may take longer. It requires the analyst to know the previous designs, for example, from competitive analysis, which normally take at least a week.

6. This analysis, of course, depends on what you take as “natural.” English imperatives may be verb-noun, but manually, one first grabs a thing and then does something with it, that is, noun-verb.

7. This estimate is for “delux” usability testing. “Discount” testing (Nielsen, 1992) is much faster to learn.

References


Atwood, M., Gray, W., and John, B. Project Ernestine: Analytic and empirical methods applied to a real-world CHI problem. (This volume.)

Card, S. Pioneers and settlers: Methods used in successful user interface design. (This volume.)


Carroll, J. and Rosson, M. B. Getting around the task-artifact cycle: How to make claims and design by scenario. (This volume.)


Gould, J., Ukelson, J. P., and Boies, S. Improving user interfaces and application productivity by using the ITS application development environment. (This volume.)


Poltrock, S. and Grudin, J. Organizational obstacles to interface design and development: Two participant observer studies. (This volume.)


Rudman, C. and Engelbeck, G. Lessons in choosing methods for designing complex graphical user interfaces. (This volume.)


