Kieras (1988), however, complained that the Card, Moran, and Newell presentation "does not explain in any detail how the notation works, and it seems somewhat clumsy to use. Furthermore, the notation has only a weak connection to the underlying cognitive theory." Kieras offers a refinement, with the Natural GOMS Language (NGOMSL), and an analysis method for writing down GOMS models. He tries to clarify the situations in which the GOMS task analyst must make a judgment call, must make assumptions about how users view the system, must bypass a complex and hard-to-analyze task such as writing the wording of a sentence or finding a bug in a program, or must check for consistency.

The development of GOMS modeling has continued, with progress in both techniques and tools. A version called CPM GOMS (CPM stands for both "cognitive processing," and "critical path method") can model the overlapping (multitasking) behavior displayed by extremely skilled users. It was used to predict the efficiency of a proposed workstation for telephone operators, saving the company $2,000,000 a year in operating costs. John and Kieras (1986a, b) compare four variants of GOMS techniques and describe nine cases of their
use in practical applications. Baumeister, John, and Byrne (2000) review three
gOMS tools designed to make modeling easier.

Another avenue for progress in gOMS modeling has been to implement
gOMS models within more complex computational cognitive architectures,
such as Soar and ACT-R (Anderson and Lebiere, 1998; New and Gluck, 2000).
These architectures are embedded in software systems that model cognitive,
emotional, and perceptual processes, but these models inherit the limitations of
gOMS models, predicting only skilled user execution time for familiar tasks. A
current goal is to employ these architectures to predict a broader spectrum of
human performance, including learning time, errors, performance under stress,
and retention over time.

2.4.4 Consistency through grammars

An important goal for designers is a consistent user interface. However, the def-
nition of consistency is elusive and has multiple levels that can sometimes conflict;
it is also sometimes advantageous to be inconsistent. The argument for
consistency is that a command language or set of actions should be orderly, pre-
dictable, describable by a few rules, and therefore easy to learn and master. These
overlapping concepts are conveyed by an example that shows two kinds of
inconsistency: (A) illustrates lack of any attempt at consistency, and (B) shows consis-
tency except for a single violation:

<table>
<thead>
<tr>
<th>Consistent</th>
<th>Inconsistent A</th>
<th>Inconsistent B</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete/insert table</td>
<td>delete/insert table</td>
<td>delete/insert table</td>
</tr>
<tr>
<td>delete/insert column</td>
<td>remove/add column</td>
<td>remove/add column</td>
</tr>
<tr>
<td>delete/insert row</td>
<td>destroy/create row</td>
<td>delete/insert row</td>
</tr>
<tr>
<td>delete/insert border</td>
<td>row/draw border</td>
<td>delete/insert border</td>
</tr>
</tbody>
</table>

Each of the actions in the consistent version is the same, whereas the actions vary
for the inconsistent version. The inconsistent action verbs are all acceptable,
but their variety suggests that they will take longer to learn, cause more errors,
will slow down users, and will be harder for users to remember. Inconsistency
version B is somewhat more readable, because there is a single unpre-
dictable inconsistency that stands out and dramatically that this language is
likely to be remembered for its peculiar inconsistency.

To capture these notions, Reinharz (1982) proposed an action grammar to
describe two versions of a graphic-system interface. She demonstrated that the
version that had a simpler grammar was easier to learn. Preece and Green:
(1986) expanded her work by addressing the multiple levels of consistency
flexible, syntactic, and semantic through a notational structure they call trans-
action grammars (TAGS). They also address some aspects of completeness of a
language by trying to characterize a complete set of tasks, for example, an,
move cursor one character forward  [Direction = forward, Unit = char]
move cursor one character backward [Direction = backward, Unit = char]
move cursor one word forward    [Direction = forward, Unit = word]
move cursor one word backward   [Direction = backward, Unit = word]

The high-level rule schemas that describe the syntax of the commands would be as follows:

1. task [Direction, Unit] → symbol [Direction] + letter [Unit]
2. symbol [Direction = forward] → "CTRL"
3. symbol [Direction = backward] → "ESC"
4. letter [Unit = word] → "W"
5. letter [Unit = char] → "C"

These schemas will generate a consistent grammar:

move cursor one character forward  CTRL-C
move cursor one character backward ESC-C
move cursor one word forward      CTRL-W
move cursor one word backward     ESC-W

Payne and Crow are careful to state that their notation and approach are flexible and extendible, and they provide appealing examples in which their approach changed the thinking of designers.

Reisiger (1976) extends this work by defining consistency more formally, but Credit (1979) points out flaws in some arguments for consistency. Certainly consistency is subtle and has multiple levels; however, there are conflicting forms of consistency and sometimes inconsistency is a virtue (for example, to draw attention to a dangerous operation). Nevertheless, understanding consistency is an important goal for designers and researchers.

Consistency issues are critical in design of mobile devices. In successful products, users get accustomed to consistent patterns, such as initiating actions by a left-side button while terminating actions by a right-side button. Similarly, up and down scrolling actions should be done consistently by buttons that are var-
2.4.5 Widget-level theories

Hierarchical decomposition is often a useful tool for dealing with complexity, but many of the theories and predictive models follow an extreme reductionist approach, which may not always be valid. In some situations, it is hard to accept the low level of detail, the precise numbers that are sometimes attached to sub-tasks, and the validity of simple summations of time periods. Furthermore, models requiring numerous subjective judgments raise the question of whether several analysts would come up with the same results.

An alternative approach is to follow the simplifications made in the higher-level user interface building tools (see Chapter 5). Instead of dealing with atomic-level features, why not create a model based on the widgets interface components supported in the tool? Once a scrolling-list widget was tested to determine user performance as a function of the number of items and the size of the window, the performance of future widget users could be predicted automatically. The prediction should have to be derived from some declaration of the task frequencies, but the description of the interface would emerge from the process of designing the interface. A measure of layout appropriateness (for example, the widget should be adjacent, and the left-to-right sequence should be in harmony with the task-sequence description) would also be produced to guide the designer in a possible redesign. Estimates of the perceptual and cognitive complexity plus the motor load could be generated automatically (Sears, 1992).

As widgets become more sophisticated and more widely used, the investment in determining the complexity of each widget will be amortized over the many devices and products.

Gradually, richer patterns of interface usage are appearing, resembling what Christopher Alexander has described for architecture (1979). Familiar patterns of building, fireplaces, staircases, or tools become modular components that acquire names and are combined to form still larger patterns. Patterns are akin to guidelines, with the distinguishing feature that patterns impose an orderly structure of problem, context, solution, examples, and cross-referencing. Patterns for human-computer interaction—such as “multiple ways to navigate,” “process framed,” and “internationalized and localized content”—have been identified for desktop applications, web design (Van Damme, Landay, and Heer, 2000), and mobile devices.
2.4.6 Context-of-use theories

While the scientific methods of experimental and cognitive psychology have a profound influence on early work in human computer interaction, a growing awareness of the special needs of this new discipline led to the rise of alternative theories. The complaints against highly controlled laboratory studies of isolated phenomena grow from researchers and practitioners. Investigations of workplace and home computing identified the critical role of users' complex interactions with other people, other electronic devices, and paper resources. For example, successful users of interfaces often have nearby colleagues to ask for help or require diverse documents to complete their tasks. Unexpected interruptions are a regular part of life, and sticky notes attached to the sides of computer monitors are often consulted for vital information. In short, the physical and social environments are intrinsically intertwined with use of information and computing technologies. Design cannot be separated from patterns of use.

Sedlmair’s (1987) analysis in his book Plans and Situated Action is often credited with launching this reconsideration of human computer interaction. He argued that the cognitive model of ordinary human plans that were executed when needed was insufficient to describe the richer and livelier world of work or personal usage. He proposed that users’ actions were situated in time and space, making their behavior highly responsive to other people and to environmental contingencies. If users got stuck in using an interface, they would ask for help, depending on who was around, or cancel a task if it were available. If they were pressed for time, they might risk some shortcuts. Rather than having fixed plans, users were constantly changing their plans in response to the circumstances.

The argument of distributed cognition is that knowledge is not only in the users’ heads, but distributed in their environments—some knowledge is stored on paper documents, maintained by computers, or available from colleagues. Proponents of distributed cognition emphasize this distinction from the model of human processes described by Carl, Moran, and Newell as the basis for COILS (Brache and Rogers, 1996).

Physical space became an important notion for those who began to think more about ubiquitous, pervasive, and embedded devices. However, they sought to shift attention from place to space, implying that the social/psychological space had to be considered in addition to the physical place (Bucheli, 2003). These notions are likely to become still more important as vital sensors become more common. Sensors to art rate doers in super markets or alert drivers to hazards are first steps, but newer sensors that detect and monitor human activity seem likely to proliferate. The goals are often positive, such as safety, security, or health care. But threats to privacy, dangers of errors, and the need to preserve human control will have to be considered carefully.
Other alternative models of technology use emphasize the social environment, motivations of users, or the role of experience. Innovators believe that turbulence of actual usage, as opposed to idealized task specifications, means that users have to be more than test subjects—they have to be participants in design processes (Greenbaum and Kyng, 1991). Breakdowns are often seen as a source of insights about design, and users are encouraged to become reflective practitioners who are continuously engaged in the process of design refinement. Understanding the transition from novice to expert and the differences in skill levels has become a focus of attention, further calling into question the utility of hour-long laboratory or half-day usability testing studies as a guide to behavior of users after a month or more of experience. These methods encourage greater attention to detailed ethnographic observation, longitudinal case studies, and action research by participant observers (Nardi, 1997, Redmiles, 2002).

Context of use theories are especially relevant to mobile devices and ubiquitous computing innovations. Such devices are portable or installed in a physical space, and they are often designed specifically to provide place-specific information, such as a city guide on a portable computer or a museum guide that gives information on a nearby painting. A taxonomy of mobile device applications could guide innovators:

- Monitor blood pressure, stock prices, or air quality and give alerts when normal ranges are exceeded.
- Gather information from meeting attendees or rescue team members and send the action list or current status to all.
- Participate in a large group activity by voting and relate to specific individuals by sending private messages.
- Locate the nearest restaurant or landmark and identify the details of the current location.
- Capture information of photos left by others and share yours with future visitors.

These five pairs of actions could be fed to a variety of objects such as photos, annotations, or documents, suggesting new mobile devices and services. They also suggest that one way of thinking about user interfaces is by way of the objects that users encounter and actions that they take.

2.5 Object-Action Interface Model

The cognitive model described in this book's first edition showed the separation between task domain concepts (for example, stock market portfolio) and the
computer domain concepts that represent them (for example, folders, spreadsheets, or databases). The second edition amplified the important distinction between objects and actions, paralleling the familiar separation between nouns and verbs. In the third edition, the underlying theory of design was called the object-action interface (OAI) model. The OAI model is descriptive and explanatory, and it can also be prescriptive, in that it provides valuable guidance for designers of interfaces, online help, and training processes.

As GUIs have replaced command languages, intricate syntax has given way to relatively simple direct manipulations applied to visual representations of objects and actions. The emphasis is now on the visual display of user-task objects and actions. For example, a collection of stock market portfolios might be represented by locker folders with icons of engraved share certificates; likewise, actions might be represented by trash cans or delete buttons. These represent whole actions for portfolio copying. Of course, there are syntactic aspects of direct manipulation, such as knowing whether to drag the file to the trash can or vice versa, but the amount of syntax is small and can be thought of as being at the lowest level of interface actions. Even syntactic forms such as double-clicking, mouse-driven-and-seat, or gestures seem simple compared to the pages of grammars for early command languages.

Doing object-action design starts with understanding the task, that task includes the universe of real-world objects with which users work to accomplish their intentions and the actions that they apply to those objects (Fig. 2-4). The high-level task objects might be stock-market listing, a photo library, or a personal phone book. These objects can be decomposed into information on a single stock, for example, and finally into atomic entities, such as a share price. Task actions start from high-level intentions that are decomposed into intermediate goals and individual steps.

To accommodate the ambiguity of situated action and distributed cognition, the objects may include real-world items (such as books, maps, or other devices) and the actions may include common act values (such as speaking to colleagues, handling interruptions, or answering telephones). These may be described as part of a model of user activity, but they form a separate category since designers may have little influence on them or how they are used.

Once there is agreement on the task objects and actions and their decomposition, the designer can create the metaphoric representations of the interface objects and actions. Interface objects do not have weight or thickness, they are icons that can be moved or copied in ways that represent real-world task objects with feedback to guide users. Finally, the designer must map the interface actions visible to users, so that users can decompose their plans into a series of intermediate actions, such as opening a dialog box, all the way down to a series of detailed keystrokes and clicks.
In outline, the OAI model is a descriptive and explanatory model that focuses on task and interface objects and actions. Because the systemic details are omitted, users who know the task domain objects and actions can learn the interface relatively easily (see Chapter 56). The OAI model also reflects the higher level of design with which most designers deal when they use the widgets in user interface building tools. The standard widgets have familiar and simple syntax (click, double-click, drag, or drop) and simple forms of feedback (highlighting, scrolling, or movement), leaving designers free to focus on how these widgets create a business-oriented solution. The OAI model is in harmony with the common software engineering method of object-oriented design.

E.5.1 Task hierarchies of objects and actions

The primary way people deal with large and complex problems is to decompose them into several smaller problems, in a hierarchical manner, until each subproblem is manageable. For example, a human body is discussed in terms of neural, muscular, skeletal, reproductive, digestive, circulatory, and other subsystems, which in turn might be described by organs, tissues, and cells. Most real-world objects have similar decompositions: buildings, cities, computer programs, human genomes, and plays, for example. Some objects are
more nestly decomposable than others, and some objects are easier to understand than others.

A symphony performance has movements, measures, and notes, a baseball game has innings, outs, and pitches. A building-construction plan can be reduced to a series of steps, such as surveying the property, laying the foundation, building the frame, raising the roof, and completing the interior. Similarly, intentions can be decomposed into smaller action steps.

People generally learn the task objects and actions independently of their implementation on a computer. Likewise, people learn about buildings or books through developmental experiences in their youth, but many tasks require specialized training, such as in how to manage stock market portfolios, to design buildings, or to diagnose medical problems. It may take years to learn the technology, to acquire the decision-making skills, and to become proficient.

Designers who develop user interfaces to support professionals may have to take training courses, read textbooks, and interview users. Then, the designers can sit down with the users and generate a hierarchy of objects and actions to model the users’ tasks. This model forms a basis for designing the interface objects and actions as well as their representations in place on a screen, in physical devices, or by a voice or other audio cue.

Users who must learn to use interfaces to accomplish real-world tasks must first become proficient in the task domain. An expert interfaces user who has not studied architecture will not be able to effectively use a building design package, nor will a computer-savvy amateur be able to make reliable medical diagnoses without extensive training.

In summary, tasks include hierarchies of objects and actions at high and low levels. Hierarchies are not perfect, but they are comprehensible and useful. Most users accept a separation of their tasks into high- and low-level objects and actions.

2.5.2 Interface hierarchies of objects and actions

Like tasks, interfaces include hierarchies of objects and actions at high and low levels. For example, a central set of interface object concepts deals with storage. Users come to understand the high-level concept that computers store information. The stored information can then be refined into objects, such as a directory and the files it contains. In turn, the directory object is refined into a set of directory entries, each of which has a name, length, date of creation, owner, access, and so on. Likewise, each file is an object that has a lower-level structure consisting of lines, fields, characters, fonts, printers, and so on.

The interfaces also are decomposable into lower-level actions. The high-level action, such as backing up a data file, may require selection, duplication, and save actions. The sub-level action of saving a file is refined into the actions of selecting a destination and moving the file into a remote disk, providing a password, overwriting previous versions, assigning a name to the file, and so on.
There are also many low-level details about permissible file types or sizes, error conditions such as shortage of storage space, or responses to hardware or software errors. Finally, users carry out each low-level action by selecting a button in a dialog box or clicking on a pull-down menu item.

Designers craft interface objects and actions based on familiar examples. Furniture those objects and actions to fit the task. For example, in developing a system to manage stockbroker portfolios, the designer might consider spreadsheets, databases, work processors, or a special level graphical design that allows users to drag stock symbols to a buying or selling box.

Use can be a tutorial about interface objects and actions by seeing a demonstration, hearing an explanation of features, or conducting trial-and-error sessions. The iconic representation—abstract, concrete, or analogical—conveys the interface objects and actions. For example, to explain saving a file, an instructor might show a picture of a disk drive and a directory to show where the file goes and how the directory references the file. Alternatively, the instructor might describe how the file catalog acts as a directory for books saved in the library.

When interface objects and actions have a logical structure that can be anchored to familiar task objects and actions, we expect that structure to be relatively stable in human memory. If users remember the mid-level concept of saving a file, they will be able to conclude that the file must have a name, a size, and a storage location. The linkage to other objects and the visual demonstration can enhance the memorability of this knowledge.

These interface objects and actions were once novel, known by only a small number of scientists, engineers, and data-processing professionals. Now, these concepts are taught at the elementary school level, argued over during coffee breaks in the office, and exchanged in the aisles of corporate jets. When educators talk of computer literacy, part of their plan should include these interface concepts.

The OAI model helps us to understand the multiple levels of processing that must occur for users to be successful in using an interface to accomplish a task. For example, in writing a business letter using computer software, users must integrate smoothly their knowledge of the task objects and actions and of the interface objects and actions. They must have the high-level concept of writing (task action) and of the task object (letter). They must also recognize the mechanisms for beginning, writing, and ending a sentence. Finally, users must know the proper low-level details of spelling each word (low-level task object) and must know where the keys are for each letter (low-level interface object). The goal of minimizing interface concepts (such as the syntax of a command language) while presenting a visual representation of the task objects and actions is the heart of the direct-manipulation approach to design (see Chapter 6).
Integrating the multiple levels of task and interface concepts is a substantial challenge that requires great motivation and concentration. Educational materials that facilitate the acquisition of this knowledge are difficult to design, especially because of the diversity of background knowledge and motivation levels of typical learners. The OAI model of user knowledge can provide a guide to educational designers by highlighting the different kinds of knowledge that users need to acquire (see Chapter 13).

Designers of interactive systems can apply the OAI model to systematize their work. Where possible, the task objects should be made explicit, and the users' task actions should be laid out clearly. Then, the interface objects and actions can be identified, and appropriate representations can be created. These designs are likely to increase comprehensibility to users and independence from specific hardware. Criteria for design quality are emerging based on the fact that small numbers of objects and actions tend to be easier to learn. Designers would do well to determine how fine a granularity of objects to use and how many different actions are needed.

2.5.3 The disappearance of syntax

In the early days of computer, users had to maintain a profusion of device-dependent details in their heads. These low-level syntactic details included the knowledge of which action causes a character to appear on the screen, which character is inserted, and so on. These facts were often learned byrote, which greatly reduces the effectiveness of any attempt to learn. But memorization requires repeated rehearsals to each competence, and retention over time is poor unless the knowledge is applied frequently.

A related problem is that syntactic knowledge is in the difficulty of providing a hierarchical structure for even a moderate-sized task. For example, it may be hard for users to remember these details of using an electronic-mail system: press Return to terminate a paragraph, Ctrl-D to terminate a message, Q to quit the electronic-mail subsystem, and log out to terminate the session. The knowledge of how to use a computer is not stored in the context of the full system, but in the context of individual situations that have radically different syntactic forms.

A final difficulty is that syntactic knowledge is system-dependent. Users who switch from one machine to another may face different keyboard layouts, com-
Practitioner's Summary

Design principles and guidelines are emerging from practical experience and empirical studies. Organizations can benefit by reviewing available guidelines and documents and constructing local versions. A guideline document records organizational policies, supports consistency, and aids the application of tools for user-interface building. Facilitates training of new designers, records results of practice and experimental testing, and stimulates discussion of user-interface issues. More established principles—such as recognizing user diversity, deriving

WORLD WIDE WEB RESOURCES

Web sites include guidelines documents for desktop, web, and mobile devices. Interfaces and recommendations for universal usability strategies to accommodate users with disabilities or other special needs. Theories are proliferating, so the Web is a good place to keep up with the latest. Databases and other topics can be found in newsgroups, which are searchable from many standard services such as Yahoo! or Google.

http://www.gi.com/DTUI
for consistency, and preventing errors—have become widely accepted, but they require fresh interpretation as technology and applications evolve. Automation is increasingly present, but preserving human control is still a beneficial goal. A variety of reliable and broadly applicable theories are beginning to emerge for user interfaces.

In spite of the growing set of guidelines, principles, and theories, user interface design is a complex and highly creative process. Successful designers begin with a thorough task analysis and a detailed specification of the user communities. For expert users, with established sequences of action, predictive models that reduce the time required to perform each step are effective. For novel applications and novice users, focusing on task objects and actions (for example, flight departure airports and upgrading reservations) can lead to construction of useful indices for interface objects and actions such as form fill-in and pull-down menu selections. Still, extensive testing and iterative refinement are necessary parts of every development project.

Researcher's Agenda

The central problem for human-computer interaction researchers is to develop adequate theories and models. Traditional psychological theories must be extended and refined to accommodate the complex human learning, memory, and problem solving required in user interfaces. Useful goals include descriptive taxonomies, explanatory theories, and predictive models.

A first step might be to investigate a limited task for a single community and to develop a notation for describing task actions and objects. Then the mapping to interface actions and objects can be made. This process would lead to predictions of learning times, performance speed, error rates, subjective satisfaction, or human retention over time, for competing designs.

Next, the range of tasks and user communities could be expanded to domains of interest, such as word processing, web searching, or cell-phone data entry. Applied research problems are suggested by each of the hundreds of design principles or guidelines that have been proposed. Each validation of these principles and clarification of the breadth of applicability is a small but useful contribution to the emerging corpus of human performance with interactive systems.

References

Chapter 3: Guidelines, Principles, and Theories


Chapter 2  Guidelines, Principles, and Theories

Richard E. Friesen, Steve Dutt, and Douglas R. K. Smith

2.1 Guidelines for Graphical User Interfaces

2.2 Principles of Interaction Design

2.3 Theories of Human-Computer Interaction

References


