USER INTERFACES

For articles on related subjects, see HUMAN FACTORS IN COMPUTING SYSTEMS; INTERACTIVE INPUT DEVICES; INTERACTIVE SYSTEM; and WINDOW ENVIRONMENTS.

A user interface is that portion of an interactive computer system that communicates with the user. Design of the user interface includes any aspect of the system that is visible to the user. Once, all computer user were specialists in computing, and interfaces consisted of jumper wires in patch boards, punched cards prepared offline, and batch printouts. Today a wide range of nonspecialists use computers, and keyboards, mice, and graphical displays are the most common interface hardware. The user interface is becoming a larger and larger portion of the software in a computer system—and a more important portion, as broader groups of people use computers. As computers become more powerful, the critical bottleneck in applying computer-based systems to solve problems is now more often in the user interface, rather than the computer hardware or software.

Because the design of the user interface includes anything that is visible to the user, interface design extends deep into the design of the interactive system as a whole. A good user interface cannot be applied to a system after it is built but must be part of the design process from the beginning. Proper design of a user interface can make a substantial difference in training time, performance speed, error rates, user satisfaction, and the user's retention of knowledge of operations over time. The poor designs of the past are giving way to elegant systems. Descriptive taxonomies of users and tasks, predictive models of performance, and explanatory theories are being developed to guide designers and evaluators. Haphazard and intuitive development strategies with claims of ‘‘user friendliness’’ are yielding to a more scientific approach. Measurement of learning time, performance, errors, and subjective satisfaction is now a part of the design process.

Design of a User Interface

Design of a user interface begins with *task analysis*—an understanding of the user's underlying tasks and the problem domain. The user interface should be designed in terms of the user's terminology and conception of his or her job, rather than the programmer's. A good understanding of the cognitive and behavioral characteristics of people in general as well as the particular user population is thus important. Good user interface design works from the user's capabilities and limitations, not the machine's; this applies to generic interfaces for large groups of people as well as to designing special interfaces for users with physical or other disabilities. Knowledge of the nature of the user's work and environment is also critical. The task to be performed can then be divided and portions assigned to the user or machine, based on knowledge of the capabilities and limitations of each.

Levels of Design

It is useful to consider the user interface at several distinct levels of abstraction and to develop a design and implementation for each. This simplifies the developer's task by allowing it to be divided into several smaller problems. The design of a user interface is often divided into the *conceptual, semantic, syntactic,* and *lexical* levels. The conceptual level describes the basic entities underlying the user's view of the system and the actions possible upon them. The semantic level describes the functions performed by the system. This corresponds to a description of the functional requirements of the system, but it does not address how the user will invoke the functions. The syntactic level describes the sequences of inputs and outputs necessary to invoke the functions described. The lexical level determines how the inputs and outputs are actually formed from primitive hardware operations.

The *syntactic-semantic object-action model* is a related approach; it, too, separates the task and computer concepts (i.e., the semantics in the previous paragraph) from the syntax for carrying out the task. For example, the task of writing a scientific journal article can be decomposed into the sub-tasks for writing the title page, the body, and the references. Similarly, the title page might be decomposed into a unique title, one or more authors, an abstract, and several keywords. To write a scientific article, the user must understand these task semantics. To use a word processor, the user must learn about computer semantics, such as directories, filenames, files, and the structure of a file. Finally, the user must learn the syntax of the commands for opening a file, inserting text, editing, and saving or printing the file. Novices often struggle to learn how to carry out their tasks on the computer and to remember the syntactic details. Once learned, the task and computer semantics are relatively stable in human memory, but the syntactic details must be frequently rehearsed. A knowledgeable user of one word processor who wishes to learn a second one only needs to learn the new syntactic details.

User Interface Management Systems

A *user interface management system (UIMS)* is a software component that is separate from the application program that performs the underlying task. The UIMS conducts the interaction with the user, implementing the syntactic and lexical levels, while the rest of the system implements the semantic level. Like an operating system or graphics library, a UIMS separates functions used by many applications and moves them to a shared subsystem. It centralizes implementation of the user interface and permits some of the effort of designing tools for user interfaces to be amortized over many applications and shared by them. It also encourages consistent ‘‘look and feel’’ in user interfaces to different systems, since they share the user interface component. A UIMS also supports the concept of *dialogue independence*, where changes can be made to the interface design (the user-computer dialogue) without affecting the application code. This supports the development of alternative user interfaces for the same application (semantics), which facilitates both iterative refinement of the interface through prototyping and testing and, in the future, alternative interfaces for users of different physical or other disabilities. A UIMS requires a language or method for specifying user interfaces precisely; this also allows the interface designer to describe and study a variety of possible user interfaces before building any. UIMSs are emerging as powerful tools that not only reduce development effort, but also encourage exploratory prototyping.

Syntactic Level Design: Interaction Styles

The principal classes of user interfaces currently in use are command languages, menus, forms, natural language, direct manipulation, virtual reality, and combinations of these. Each interaction style has its merits for particular user communities or sets of tasks. Choosing a style or a combination of styles is a key step, but within each there are numerous minute decisions that determine the efficacy of the resulting system.

Command Language

Command language user interfaces use artificial languages, much like programming languages. They are concise and unambiguous, but they are often difficult for a novice to learn and remember. However, since they usually permit a user to combine constructs in new and complex ways, they can be more powerful for advanced users. For them, command languages provide a strong feeling that they are in charge and that they are taking the initiative rather than responding to the computer. Command language users must learn the syntax, but they can often express complex possibilities rapidly, without having to read distracting prompts. However, error rates are typically high, training is necessary, and retention may be poor. Error messages and on-line assistance are difficult to provide because of the diversity of possibilities and the complexity of relating tasks to computer concepts and syntax. Command languages and lengthier query or programming languages are the domain of the expert frequent users (*power* *users*), who often derive satisfaction from mastering a complex set of concepts and syntax. Command language interfaces are also the style most amenable to programming, that is, writing programs or scripts of user input commands.

Menu

Menu-based user interfaces explicitly present the options available to a user at each point in a dialogue. Users read a list of items, select the one most appropriate to their task, type or point to indicate their selection, verify that the selection is correct, initiate the action, and observe the effect. If the terminology and meaning of the items are understandable and distinct, users can accomplish their tasks with little learning or memorization and few keystrokes. The menu requires only that the user be able to recognize the desired entry from a list rather than recall it, placing a smaller load on long-term memory. The greatest benefit may be that there is a clear structure to decision making, since only a few choices are presented at a time. This interaction style is appropriate for novice and intermittent users. It can also be appealing to frequent users if the display and selection mechanisms are very rapid. A principal disadvantage is that they can be annoying for experienced users who already know the choices they want to make and do not need to see them listed. Well-designed menu systems, however, can provide bypasses for expert users. Menus are also difficult to apply to ‘‘shallow’’ languages, which have large numbers of choices at a few points, because the option display becomes too big. For designers, menu selection systems require careful task analysis to ensure that all functions are supported conveniently and that terminology is chosen carefully and used consistently. Software tools to support menu selection help in ensuring consistent screen design, validating completeness, and supporting maintenance.

Form Fill-in

Menu selection usually becomes cumbersome when data entry is required; *form fill-in* (also called fill-in-the-blanks) is useful here. Users see a display of related fields, move a cursor among the fields, and enter data where desired, much as they would with a paper form for an invoice, personnel data sheet, or order form. Seeing the full set of related fields on the screen at one time in a familiar format is often very helpful. Form fill-in interaction does require that users understand the field labels, know the permissible values, be familiar with typing and editing fields, and be capable of responding to error messages. These demands imply that users must have some training or experience.

Natural Language

The principal benefit of *natural language* user interfaces is, of course, that the user already knows the language. The hope that computers will respond properly to arbitrary natural language sentences or phrases has engaged many researchers and system developers, but with limited success thus far. Natural language interaction usually provides little context for issuing the next command, frequently requires ‘‘clarification dialog,’’ and may be slower and more cumbersome than the alternatives. Therefore, given the state of the art, such an interface must be restricted to some subset of natural language, and the subset must be chosen carefully—both in vocabulary and range of syntactic constructs. Such systems often behave poorly when the user veers even slightly away from the subset. Since they begin by presenting the illusion that the computer really can ‘‘speak English,’’ the systems can trap or frustrate novice users. For this reason, the techniques of human factors engineering can help. A human factors study of the task and the terms and constructs people normally use to describe it can be used to restrict the subset of natural language in an appropriate way, based on empirical observation. Human factors study can also identify tasks for which natural language input is good or bad. Although future research in natural language offers the hope of human-computer communication that is so natural it is ‘‘just like talking to a person,’’ such conversation may not always be the most effective way of commanding a machine. It is often more verbose and less precise than computer languages. In settings such as surgery, air traffic control, and emergency vehicle dispatching, people have evolved terse, highly formatted languages, similar to computer languages, for communicating with other people. For a frequent user, the effort of learning such an artificial language is outweighed by its conciseness and precision, and it is often preferable to natural language.

Direct Manipulation

In a *graphical* or *direct manipulation* style of user interface (GUI), a set of objects is presented on a screen, and the user has a repertoire of manipulations that can be performed on any of them. This means that the user has no command language to remember beyond the standard set of manipulations, few cognitive changes of mode, and a reminder of the available objects and their states shown continuously on the display. Examples of this approach include painting programs, spreadsheets, manufacturing or process control systems that show a schematic diagram of the plant, air traffic control systems, some educational and flight simulations, video games, and the Xerox Star desktop and its descendants (Macintosh, Windows, and various X Window file managers). By pointing at objects and actions, users can rapidly carry out tasks, immediately observe the results, and, if necessary, reverse the action. Keyboard entry of commands or menu choices is replaced by cursor motion devices, such as a lightpen, joystick, touchscreen, trackball, or mouse, to select from a visible set of objects and actions. Direct manipulation is appealing to novices, is easy to remember for intermittent users, encourages exploration, and, with careful design, can be rapid for power users. The key difficulty in designing such interfaces is to find suitable manipulable graphical representations or visual metaphors for the objects of the problem domain, such as the desktop and filing cabinet. A principal drawback of direct manipulation is that it is often difficult to create scripts or parameterized programs in such an inherently dynamic and ephemeral language.

In a well-designed direct manipulation interface, the user’s input actions should be as close as possible to the user’s thoughts that motivated those actions; the gap between the user’s intentions and the actions necessary to input them into the computer should be reduced. The goal is to build on the equipment and skills humans have acquired through evolution and experience and exploit these for communicating with the computer. Direct manipulation interfaces have enjoyed great success, particularly with new users, largely because they draw on analogies to existing human skills (pointing, grabbing, moving objects in space), rather than trained behaviors.

Virtual Reality

Virtual reality environments carry the user’s illusion of manipulating real objects and the benefit of natural interaction still further. By coupling the motion of the user’s head to changes in the images presented on a head-mounted display, the illusion of being surrounded by a world of computer-generated images, or a virtual environment, is created. Hand-mounted sensors allow the user to interact with these images as if they were real objects located in space surrounding him or her. *Augmented reality* interfaces blend the virtual world with a view of the real world through a half-silvered mirror or a TV camera, allowing virtual images to be superimposed on real objects and annotations or other computer data to be attached to real objects. The state of the art in virtual reality requires expensive and cumbersome equipment and provides very low resoution display, so such interfaces are currently used mainly where a feeling of ‘‘presence’’ in the virtual world is of paramount importance, such as training of fire fighters or treatment of phobias. As the technology improves they will likely find wider use.

Virtual reality interfaces, like direct manipulation interfaces, gain their strength by exploiting the user’s pre-existing abilities and expectations. Navigating through a conventional computer system requires a set of learned, unnatural commands, such as keywords to be typed in, or function keys to be pressed. Navigating through a virtual reality system exploits the user’s existing, natural ‘‘navigational commands,’’ such as positioning his or her head and eyes, turning his or her body, or walking toward something of interest. The result is a more natural user interface, because interacting with it is more like interacting with the rest of the world.

Other Issues

Blending several styles may be appropriate when the required tasks and users are diverse. Commands may lead the user to a form fill-in where data entry is required or pop-up (or pull-down) menus may be used to control a direct manipulation environment when a suitable visualization of operations cannot be found. The area of *computer-supported cooperative work* extends the notion of a single user-computer interface to an interface that supports the collaboration of a group of users.

Although interfaces using modern techniques such as direct manipulation are often easier to learn and use than conventional ones, they are considerably more difficult to build, since they are currently typically programmed in a low-level, ad-hoc manner. Appropriate higher-level software engineering concepts and abstractions for dealing with these new interaction techniques are still needed. Direct manipulation techniques for the actual design and building of direct manipulation interfaces are one solution. Specifying the graphical appearance of the user interface (the ‘‘look’’ via direct manipulation is relatively straightforward and provided by many current tools, such as Visual Basic, but describing the behavior of the dialogue (the ‘‘feel’’) is more difficult and not yet well supported; predefined or ‘‘canned’’ controls and widgets represent the current state of the art.

Lexical Level Design: Interaction Tasks, Devices, and Techniques

Lexical design begins with the *interaction tasks* necessary for a particular application. These are low-level primitive inputs required from the user, such as entering a text string or choosing a command. For each interaction task, the designer chooses an appropriate *interaction device* and *interaction technique* (a way of using a physical device to perform an interaction task). There may be several different ways of using the same device to perform the same task. For example, one could use a mouse to select a command by using a pop-up menu, a fixed menu (palette or toolbox), multiple clicking, circling the desired command, or even writing the name of the command with the mouse.

Input Devices

Input operations range from open-ended word processing or painting programs to simple repeated ENTER key presses for page turning in an electronic document. While keyboards and mice have been the standard computer input device, there are increasingly attractive alternatives for many tasks. High-precision touchscreens have made this durable device more attractive for public access, home control, process control, and other applications. Joysticks, trackballs, and data tablets with styluses with numerous variations are also useful for various pointing and manipulation tasks. Speech input for voice mail and speech recognition for commands are effective, especially over the telephone and for the physically disabled. Other techniques for input include keys that can be dynamically labeled, speech, 3D pointing, hand gesture, whole body motion, and visual line of gaze.

Output Devices

Output mechanisms must be successful in conveying to the user the current state of the system and what actions are currently available. The CRT display has become the standard approach, but flat panel (LED, LCD, plasma, electroluminescent, and others) and hardcopy devices are alternatives. Current high-resolution screens provide approximately 1000 x 1000 pixels; but their resolution (in dots per inch) is still far cruder than a typical paper printout or photograph, and their size, far smaller than a typical user's desk, bulletin board, or other work surface. High-resolution displays can improve the readability of textual displays so that performance can match that of typewritten documents. Synthesized or digitized voice output is effective and economical, especially in telephone applications and for the physically disabled. Voice mail systems that store and forward digitized voice messages continue to grow in popularity. Other output media include animated graphics, windows, icons, active value displays, manipulable objects, hypertext and hypermedia, head-coupled displays, and non-speech audio.

Conclusions

Human engineering, once seen as the paint put on at the end of a project, is now more often becoming the steel frame on which the structure is built. Academic and industrial researchers are exploiting the power of empirical observation and traditional scientific method in human-computer interaction. The classic experimental methods of psychology are being applied to deal with the complex cognitive tasks of human performance with information and computer systems. A reductionist approach required for controlled experimentation yields small but reliable results. Through multiple replications with similar tasks, subjects, and experimental conditions, generality and validity can be enhanced. Each small experimental result becomes a tile in the mosaic of human performance with computerized information systems.

At the same time, holistic approaches of participant observation, also known as *usability testing* or *formative evaluation*, are contributing insights that can be immediately applied to designs and hypotheses that lead to controlled experiments. The simple notion of asking users to ‘‘think aloud’’ as they use a computer system yields great benefits. Videotapes of users struggling with a system make a strong impact on designers and identify at least some of the flaws in a design. Empirical techniques are also beneficially applied for informal evaluations of early prototypes and rigorous acceptance tests before delivery.

A good vision for successful user interface design is that computer-related idiosyncrasies vanish, and users are free to concentrate on their tasks. Designers are seeking to reduce the burden of complex syntax and awkward computer concepts, but much work remains. Empirical studies produce many important small results, provide the basis for refining the emerging theories, and contribute to practical guidelines. Commercial system designers apply the guidelines and their intuition to create elegant systems, but must thoroughly test their designs with real users.

Bibliography

J.D. Foley, ‘‘Interfaces for Advanced Computing,’’ *Scientific American*, vol. 257, no. 4, pp. 127-135, October 1987.

J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley, Reading, Mass., 1990.

H.R. Hartson and D. Hix, ‘‘Human-computer Interface Development: Concepts and Systems for its Management,’’ *Computing Surveys*, vol. 21, no. 1, pp. 5-92, 1989.

E.L. Hutchins, J.D. Hollan, and D.A. Norman, ‘‘Direct Manipulation Interfaces,’’ in *User Centered System Design: New Perspectives on Human-computer Interaction*, ed. by D.A. Norman and S.W. Draper, pp. 87-124, Lawrence Erlbaum, Hillsdale, N.J., 1986.

R.J.K. Jacob, ‘‘A Specification Language for Direct Manipulation User Interfaces,’’ *ACM Transactions on Graphics*, vol. 5, no. 4, pp. 283-317, 1986.

J. Johnson and others, ‘‘The Xerox Star: A Retrospective,’’ *IEEE Computer*, vol. 22, no. 9, pp. 11-29, 1989.

B.A. Myers, ‘‘User Interface Software Tools,’’ *ACM Transactions on Computer-Human Interaction*, vol. 2, no. 1, pp. 64-103, March 1995.

D.R. Olsen, *User Interface Management Systems: Models and Algorithms,* Morgan Kaufmann, San Francisco, 1992.

B. Shneiderman, ‘‘Direct Manipulation: A Step Beyond Programming Languages,’’ *IEEE Computer*, vol. 16, no. 8, pp. 57-69, 1983.

B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction, Second Edition*, Addison-Wesley, Reading, Mass., 1992.

ROBERT J. K. JACOB (REVISION OF 3RD EDITION ARTICLE BY BEN SHNEIDERMAN)