Chapter 5
The Golden Rules of User Interface Design

"Make it simple, but no simpler."
Albert Einstein

"Before you buy software, make sure it believes in the same things you do. Whether you realize it or not, software comes with a set of beliefs built in. Before you choose software, make sure it shares yours."

PeopleSoft Advertisement (1996)

User Interface Design Principles

"The golden rule of design: Don't do to others what others have done to you. Remember the things you don't like in software interfaces you use. Then make sure you don't do the same things to users of interfaces you design and develop."

Tracy Leonard (1996)

Why should you need to follow user interface principles? In the past, computer software was designed with little regard for the user, *so the user had to somehow adapt to the system*. This approach to system design is not at all appropriate today—*the system must adapt to the user*. This is why design principles are so important.

Computer users should have successful experiences that allow them to build confidence in themselves and establish self-assurance about how they work with computers. Their interactions with computer software should be "success begets success." Each positive experience with a software program allows users to explore outside their area of familiarity and encourages them to expand their knowledge of the interface. Well-designed software interfaces, like good educators and instructional materials, should build a "teacher-student" relationship that guides users to learn and enjoy what they are doing. Good interfaces can even challenge users to explore beyond their normal boundaries and stretch their understanding of the user interface and the computer. When you see this happen, it is a beautiful experience.

You should have an understanding and awareness of the user's mental model and the physical, physiological, and psychological abilities of users. This information (discussed in Chapters 3 and 4) has been distilled into general *principles* of user interface design, which are agreed upon by most experts in the field. User interface design principles address each of the key components of the "Look and Feel" iceberg (see Chapter 3): presentation, interaction, and object relationships.

Interface design principles represent high-level concepts and beliefs that should be used to guide software design. You should determine which principles are most important and most applicable for your systems and then use these principles to establish guidelines and determine design decisions.

> Key Idea! The trick to using interface design principles is knowing which ones are more important when making design tradeoffs. For certain products and specific design situations, these design principles may be in conflict with each other or at odds with product design goals and objectives. Principles are not meant to be follow blindly, rather they are meant as guiding lights for sensible interface design.

The three areas of user interface design principles are:

1. Place users in control of the interface
2. Reduce users' memory load
3. Make the user interface consistent.

Where to Find Interface Design Principles

User interface design principles are not just relevant to today's graphical user interfaces. In fact, they have been around for quite some time. Hansen (1971) proposed the first (and perhaps the shortest) list of design principles in his paper, "User Engineering Principles for Interactive Systems." Hansen's principles were:

1. Know the user
2. Minimize memorization
3. Optimize operations
4. Engineer for errors.

A more recent and encompassing list of design principles can be found in *The Human Factor* by Rubenstein and Hersch (1984). This classic book on human-computer interaction presents 93 design principles, ranging from "1. Designers make myths; users make conceptual models." to "93. Videotape real users." I've listed some key books in the reference section at the end of this chapter. These books fall into two categories: books on interface design and software design guides. Some good interface design books (in addition to Rubenstein and Hersch) are Heckel (1984), Mayhew (1992), and Shneiderman (1992).

The major software operating system vendors have all either published or republished their design guidelines and reference materials in the past few years as they introduce new operating systems. These guidelines exemplify and encapsulate their interface design approaches. It is critical to keep up to date with these guides for your design and development environment. These software design guides include Apple Computer, Inc. (Apple, 1992), IBM Corp. (IBM, 1992), Microsoft Corp. (Microsoft, 1995), and UNIX OSF/Motif (Open Software Foundation, 1992). All of these design guides address, at a minimum, the importance of user interface design principles.

Key Idea! The most recent industry guide is *The Windows® Interface Guidelines for Software Design* from Microsoft, published in conjunction with the release of the Windows 95 operating system. Most people don't know that the document is also available online as a Windows 95 Help file. It can be found on the Windows 95 Developer's Toolkit CD-ROM. The file is called **UIGUIDE.HLP**. There are three other files associated with the help file that provide the contents, index, and search views for the document. These files are **UIGUIDE.CNT**, **UIGUIDE.FTS**, and **UIGUIDE.GID**. Those designing Windows applications should have this document close by, both hardcopy and online!

Readers should use the publications that best address their learning and working environment (including hardware, operating system, and key software products). The interface terminology may differ slightly between books, but they all address, at some level, the user interface principles that make up the major categories described here.

Why Should You Care about Interface Design Principles?

"The conclusion: interface inconsistency can cost a big company millions of dollars in lost productivity and increased support costs."

Jesse Berst (1993)

These principles are generally thought to be common across all computer hardware and software environments. They also apply across all interface types and styles. They have evolved over time through many years of interface design efforts, research, testing, and user feedback in computing environments from mainframes to the Macintosh and PCs.

These principles should even endure as new user interface technologies emerge. Jakob Nielsen (1990) noted, "The principles are so basic that even futuristic dialogue designs such as three-dimensional interfaces with DataGlove input devices, gesture recognition, and live video images will always have to take them into account as long as they are based on the basic paradigm of dialogues and user commands."

Key Idea! The actual implementation of these principles will, of course, vary according to the hardware environment, operating system, user interface capabilities of the system, and interface design goals. Often, business decisions influence the designer's use of the principles and the priorities that may be assigned to them. The user's and designer's models should also determine how the principles are applied to user interface design. At critical points in the interface design process you'll be asking the question, "What happens next?" The answer should be, "Whatever users want to do!" Remember—"Know thy users, for they are not you."

An interface design team, in conjunction with managers, team leaders, and workers, should figure out together which principles are the most appropriate for their environment and work tasks. They should then focus on purchasing or developing software products that offer usable and productive interfaces that exemplify those key principles.

Golden Rule #1: Place Users in Control

The first set of principles addresses placing users in control of the interface. A simple analogy is whether to let people drive a car or force them to take a train (Figure 5-1). In a car, users control their own direction, navigation, and final destination. One problem is that drivers need a certain amount of skill and knowledge before they are able to successfully drive a car. Drivers also need to know where they are going! A train forces users to become *passengers* rather than *drivers*. People used to driving their own car to their destination may not enjoy the train ride, where they can't control the schedule or the path a train will take to reach the destination. However, novice or casual users may enjoy the train if they don't know exactly where they are going and they don't mind relying on the train to guide and direct them on their journey. The ultimate decision to drive a car or take the train should be the user's, not someone else's. Users also deserve the right to change their mind and take the car one day and the train the next.

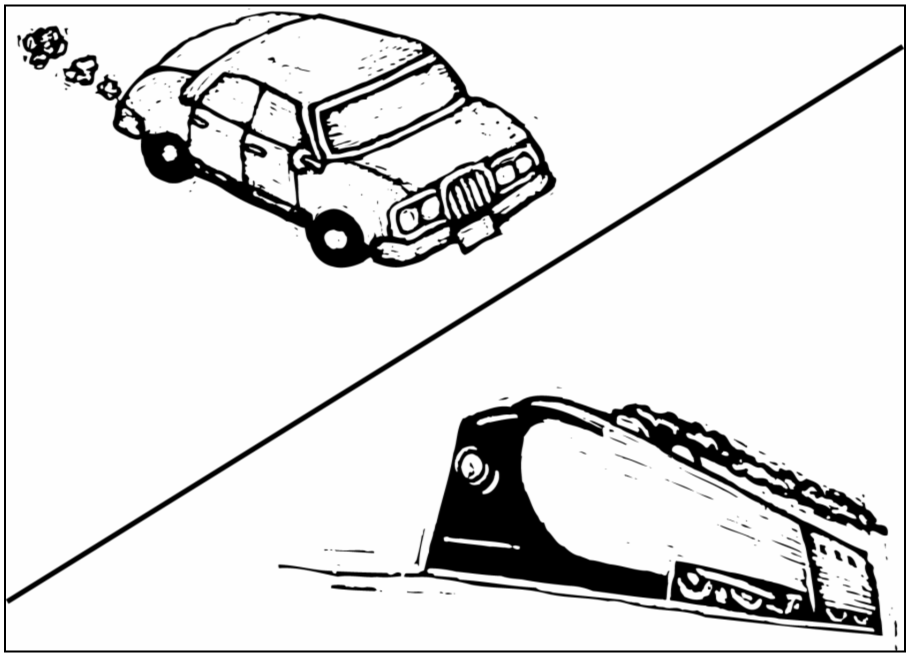


Figure 5-1. Do Users want to take a Train or drive a Car?

Let's first look at the banking environment, where computer users range from bank presidents to bank tellers. Presidents have more flexibility and authority in the tasks they can do with the computer system. Meanwhile, bank tellers have a much more limited set of tasks they perform. Their tasks are customer-directed and are repeated very often throughout the work day. Tellers should also not be allowed to access the tasks and information that bank presidents work with.

The design principles that place users in control should be used appropriately to allow bank presidents system-wide access and control. The tellers, however, should be given an interface that allows them to work within their limited set of tasks. The interface should also give them some degree of control and flexibility to do their tasks quickly, comfortably, and efficiently. In this environment, the interface designer must determine which of these principles are most important when designing the bank's computer system to be used by the all of the users.

Key Idea! A wise designer lets users do his work for him rather than attempt to figure out what they want. After designing a complex of buildings, an architect was supposed to design the walkways between the buildings. He did not assume that he knew how users would really use the walkways between buildings. So he didn't design the walkways or build them at the same time as the buildings. Rather, he had fields of grass planted between the buildings. It is rumored that he even posted signs saying, "Please walk on the grass." A few months after the buildings were completed; he came back and saw where the most worn paths were where people walked across the grass between buildings. Then he knew where he should put the walkways.

This is a wonderful, real-life example of a designer letting *users* be in control, observing their behavior, and then building an interface that allows them to go where they want to go and how they want to get there.

The principles that allow users to be in control are listed in Figure 5-2. After each principle I've listed a key word to help remember these principles.

1. Use modes judiciously (modeless)
2. Allow users to use either the keyboard or mouse (flexible)
3. Allow users to change focus (interruptible)
4. Display descriptive messages and text(Helpful)
5. Provide immediate and reversible actions, and feedback (forgiving)
6. Provide meaningful paths and exits (navigable)
7. Accommodate users with different skill levels (accessible)
8. Make the user interface transparent (facilitative)
9. Allow users to customize the interface (preferences)
10. Allow users to directly manipulate interface objects (interactive).

Figure 5-2. Principles that Place Users in Control

Use Modes Judiciously

Here's a very familiar example of modes from the "real world" of VCRs. When you press the **fast forward** or **rewind** button on your VCR or on the remote control, you don't always get the same response from the VCR. Why? Well, it depends! The system's response depends on which mode the VCR is in—either the **stop** mode or the **play** mode. If the VCR is *stopped*, then **fast forward** or **rewind** buttons fast forward or rewind the tape very quickly. However, if the VCR is *playing*, then these buttons *search* forward or backward, showing the picture on the TV. The search functions don't move the tape as quickly as the fast forward and rewind functions.

Say you've just finished watching a rented videotape. You want to rewind the tape so you won't be charged a rewinding fee when you return the tape to the video store. You press the **rewind** button while you're watching the film credits on the screen and turn off the television. You won't even know that the VCR is really searching backward in the **play** mode. This could take a very long time and since the television is turned off, you can't even see that the VCR is still in the **play** mode. What you probably wanted to do was to press the **stop** button first and *then* press **rewind**. This would rewind the tape quickly, taking much less time and placing less strain on your VCR.

This whole episode I just described recently happened at home to my wife. I had bought a new VCR and she wasn't very familiar with its operation yet. She was rewinding a tape with the television off and she commented that it seemed to be taking a long time to rewind the whole tape. I looked at the display on the front of the VCR and sure enough, there was an indicator arrow (➢) showing that the VCR was still in the **play** mode. After I explained this to my wife, she said, "That's a stupid way to design a VCR! Why don't you use that as an example in your book?"

Have you ever used a graphical drawing program on your computer? The palette of drawing tools you use is an example of the use of modes. When you select the **draw** tool, you are in the *draw* mode. Your mouse movement, mouse button presses, and keyboard keystrokes will all produce some type of drawing actions. Then select the **text** tool, and the same mouse and keyboard actions produce text and text functions. Modes are a necessary part of many software interfaces. You probably can't avoid using modes altogether, but use them only when needed. A common example of a familiar and unavoidable mode can be found in any word processor. When you are typing text, you are *always* in a mode—either *insert* mode or *replace* mode.

It's easy to find interfaces that put users in modes unnecessarily. Any time a message pops up on the computer screen and users can't do anything else in the program or even anywhere else on the screen, they are imprisoned by a *modal* dialog! There are two types of interface modes, and although they may be necessary in some cases, they are not needed or are unnecessarily restrictive most of the time.

The first type is *application* modal. When in an application mode, users are not allowed to work elsewhere in the program, or it places them in a certain mode for the whole program. For example, if working with a database of information and the **view data** mode is chosen, the program will not allow users to add, delete, or modify the data record. Users would have to change to the **update data** mode to perform these actions. This may be appropriate for users who are only allowed to browse the database. What if users are constantly viewing data and wish to change data when they want to? Why should they be forced to change modes all the time? Why are users forced to either be in **view** or **update** mode in the first place? A more user- and task-oriented approach is to let users access the data without being forced to choose a mode beforehand. If they choose to modify data, they should be able to save the data and update the database without being in a particular mode. If they don't make any changes, they can access a new data record or exit the record, without having made any decision about program modes. Perhaps the best method is to display data in a format that is consistent with a user's access. If limited access, display static text; if uses have update access, provide entry fields that are updateable.

The second type of mode is *system* modal. This mode should rarely be forced on users. While in a system mode, users are not allowed to work anywhere else on the computer until the mode is ended, or it places them in a certain mode no matter what program they are using. Let's say a document is printing, and a message window pops up stating the printer is out of paper. Users should not have to get up and put paper in the printer immediately, or even remove the message from the screen. They should be able to continue working with a word processing program or do anything else on the computer. Users might even want to keep the message window on the screen as a reminder to add paper later. Programs sometimes take control of the entire system when they present messages on the screen. There is no reason why the "Printer out of paper" message should be a system modal dialog. Watch out for this, especially when designing and programming messages and help information. You can see how frustrating system modes can be for users.

> Key Idea! Modes are not always bad things. Let users choose when they want to go into a particular mode, rather than forcing them into a mode based on where they are in the program or in the interface. The true test of interface modes is if users don't think of being in a mode or if the modes are so natural to them that they feel comfortable using them. Users don't even think about being in *insert* or *replace* (overwrite) mode while using a word-processor—it is natural for them to switch between modes whenever they wish.

When using modes, it is important to follow the principle of immediate visual feedback. Every time users choose a mode there should be some form of visual feedback while they are in that mode. Many programs change the mouse pointer or the text selection cursor to show the current mode. This is an example of the interaction between principles.

Allow Users to Use Either the Keyboard or Mouse

Don't assume that since users have a mouse attached to their computer, they will use it all of the time. Although you may design the interface to be optimized for mouse users, provide a way to do most actions and tasks using the keyboard. One of the key CUA design principles is that users must be able to do any action or task using either the keyboard or the mouse.

> Key Idea! Keyboard *access* means users can perform an action using the keyboard rather than the mouse. It does not mean that it will be easier for users to use the keyboard, just that they don't have to use the mouse if they don't want to, or can't. Toolbars, for example, are fast-path buttons for mouse users. However, users can't get to the toolbar from the keyboard—they must be able to use the menu bar drop-downs to navigate to the action they want to perform.

Users have very different habits when using keyboards and mice, and they often switch between them during any one task or while using one program. With the push toward mouse-driven, direct-manipulation interfaces, not all of the major design guides follow this philosophy of implementing both a keyboard and mouse interface. There is not a total consensus of agreement on this principle. Many Macintosh products do not provide complete keyboard access.

However, designers may want to follow this principle for the sake of users as they migrate to graphical interfaces and for consistency with other programs that may only have keyboard input. Users with special needs usually require an interface with keyboard access. Some new interface techniques also may need keyboard support to ensure user productivity. Also, as a user whose laptop mouse has been broken or disabled, or lost the mouse pointer on the screen, or been on an airplane with no room to use a mouse, I appreciate being able to access all important actions from the keyboard. Special-purpose software may choose not to follow this principle, but I recommend that all general-purpose software programs offer keyboard access unless there are compelling reasons to do otherwise.

Allow Users to Change Focus

People are always being interrupted—by a telephone, a colleague, a manager, or other things they have to do. Software interfaces should be designed so users are able to interrupt their current actions or tasks and either continue later or save them in the current state. It's easy to forget that users may not want to complete what they themselves started!

A way for users to stay in control is to offer guidance through common tasks as an option. Casual users and novices will welcome guidance, while frequent users will likely go off on their own without guidance.

> Key Idea! Don't force users to complete predefined sequences. Give them options—to cancel or to save and return to where they left off. "Wizards" are used more and more to lead users through common tasks, but don't lead with an iron hand, let users stay in control while the interface *guides* them rather than *forces* them through steps in a task.

Display Descriptive Messages and Text

"The password is too short. It must be at least 26908 bytes long. Type the password again." This is a system message I recently saw on one of my client's computer screens! Although it may be *descriptive* and accurate (how are we to know?), it certainly isn't *helpful* or appropriate. Do users know how many characters are needed to be at least 26908 bytes long? I don't think so! Maybe the message's creator can translate bytes to number of characters, but users shouldn't have to. The message also violates the principle of making the interface transparent. Users don't need to know that a password is stored as a certain number of bytes (users may not even know what bytes are!), only that they must remember it when they logon to the system. Here's a more helpful version of the message—"Your password must contain 6 to 16 characters. Please type the password again."

> Key Idea! Use terms throughout the interface that users can understand, rather than system or developer terms. Users don't necessarily know about bits and bytes, and they shouldn't have to!

This principle applies not only to messages, but to all text on the screen, whether it is prompts, instructions, headings, or labels on buttons. Yes, screen space is valuable, but it is important to use language that is easy to read and understand. Messages are key to a program's dialog with users. All textual aspects of the interface should be designed by those with writing skills. All writing is an art, including writing system and program documentation and messages. In many projects I've seen, all text on the screen, including messages, prompts, labels, titles, codes, and all help information, are the responsibility of information developers or technical writers on the design and development team.

Key Idea! It is critical to establish the proper tone of voice in messages and prompts. It is important to assign no blame for errors or problems. Poor message terminology and tone encourages users to blame themselves for problems that occur.

Provide Immediate and Reversible Actions, and Feedback

Airline crews rarely told passengers when they were experiencing difficulties with an aircraft on the ground or in the air. There were usually no announcements as to what the problem was or how long it might take to fix the problem. Passengers got very restless and impatient without any feedback. Studies found that people are much more forgiving if they are told the truth about what is going on and are given periodic feedback about the status of the situation. Now, airline pilots and crews make a point to periodically announce exactly what the situation is and how much time they expect to take to resolve it.

I recently went to the MGM Studios theme park in Orlando, Florida. The most popular ride is the "Back to the Future" adventure and there are always long lines of people waiting to get in. Signs are posted in front of each ride telling people how long the estimated waiting time is to get into the ride. There was a 45 minute wait for this popular ride when we were there. However, the entry areas for all the rides are designed like a maze, with walkways winding around and around in a very small space, so you are constantly moving and turning in different directions as you make gradual progress toward the ride entrance. Television monitors preview the ride at stations along the way so that everyone standing in line can see and hear what they are about to see in the ride.

The entry areas and the television monitors were designed specifically to keep people moving at all times, to distract them, and keep them entertained. It is very important that an "illusion of progress" be felt by users, whether it is people standing in line for an amusement park ride or computer users waiting for a program to complete an action or process. The use of feedback and progress indicators is one of the subtle aspects of a user interface that is of tremendous value to the comfort and enjoyment of users.

The lack of feedback in most software products forces users to double-check to see if their actions have been performed. In a command-line interface, whenever I delete a file using **DEL**, I usually use the **DIR** command immediately afterward to list the directory to ensure the file was actually deleted. There is no feedback after you type the **DEL** command! This forces users to perform *superstitious behaviors* to comfort themselves since there is little or no feedback from the system interface.

Key Idea! Every product should provide undo actions for users, and hopefully, also redo actions. Inform users if an action cannot be undone and allow them to choose alternative actions, if possible. Provide users with some indication that an action has been performed, either by showing them the results of the action, or acknowledging that the action has taken place successfully.

Provide Meaningful Paths and Exits

Allow users to navigate easily through the interface. Provide ways for them to get to any part of the product they want to. Allow them to move forward or backward, upward or downward through the interface structure. Make them comfortable by providing some *context* of where they are, where they've been, and where they can go next. Figure 5-3 shows the Microsoft Windows 95 taskbar, with the (by now) famous Start button. The main reason for these interface elements is to show users what programs are opened, and to allow quick access to all programs and data via the Start button.

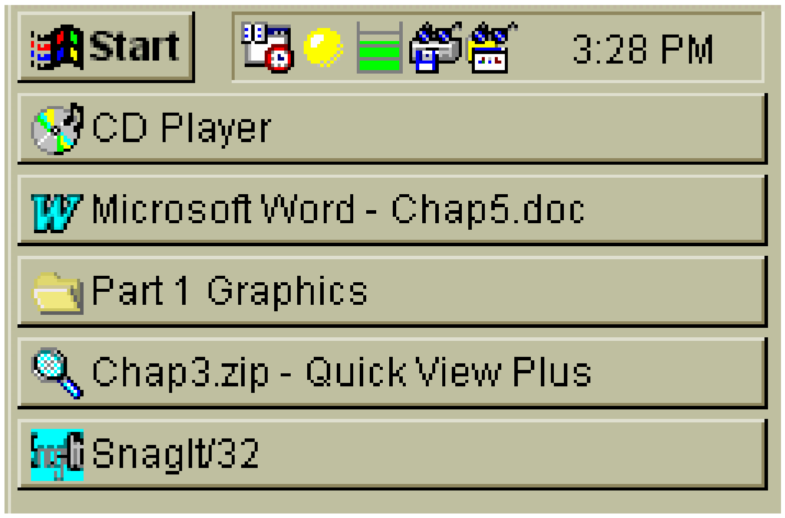


Figure 5-3. Microsoft® Windows® 95 Taskbar and Start button

The many toolbars, launch pads, button bars, dashboards, and task bars you see in today's operating systems, product suites, and utilities all are designed to help users navigate through the operating system and their hard disk, in search of programs and data. Users want fast paths to files, folders, programs and common actions, and that's what these interface utilities offer.

System and program wizards and assistants also offer guidance for navigating through a program's functions or tasks. These new interface elements are discussed in **Part 4**.

Key Idea! Users should be able to relax and enjoy exploring the interface of any software product. Even industrial-strength products shouldn't intimidate users so that they are afraid to press a button or navigate to another screen. The Internet explosion shows that navigation is a simple process to learn—basically, navigation is the main interaction technique on the Internet. If users can figure out how to get to pages on the World Wide Web, they have 80% of the interface figured out. People become experienced "browsers" very quickly.

Accommodate Users with Different Skill Levels

Users of different skill levels should be able to interact with a program at different levels. Many programs offer customizable interfaces that allow users to choose their interaction level. For example, the menu bar and pull-downs of a program can be set up as "standard" or "advanced", depending on user preferences and the types of tasks being performed.

Providing both keyboard and mouse interfaces offers users flexibility and allows users of different skill levels or physical handicaps to use input devices in whatever ways they feel comfortable.

> Key Idea! Don't sacrifice expert users for an easy-to-use interface for casual users. You must provide fast paths for experienced users. Nothing drives experienced users crazy like having to go through too many steps to perform an action they use all the time and would like to perform using one step or a macro command.

Make the User Interface Transparent

The user interface is the mystical, mythical part of a software product. If done well, users don't even feel that it is there. If done poorly, users can't get past it to effectively use the product. A goal of the interface is to help users feel like they are reaching right through the computer and directly manipulating the objects they are working with. Now, that's a transparent interface!

The interface can be made transparent by giving users work objects rather than system objects. Trash cans, waste baskets, shredders, and in- and out-baskets all let users focus on the tasks they want to do using these objects, rather than the underlying system functions actually performed by these objects. Make sure these objects work like they do in the real world, rather than in some other way in the computer. Microsoft's Windows 95 interface provides a Recycle Bin, rather than a waste basket or shredder, to remind users that things are not necessarily thrown away immediately.

Other aspects of Windows 95 are not so transparent, however. The **Close Program** dialog (displayed by keying **Ctrl+Alt+Del**) lists not only the programs users started and are currently running, but it also displays a long list of other system programs, with names like **Explorer**, **Rscrmtr**, **Symapudo**, **Qvp32**, and **Runner**, where users have no idea of what these programs are and where they came from. But users can choose any program from this list and end that running task. This can be quite dangerous and does not hide the system well from users.

> Key Idea! The secret of a transparent interface is synching up with the user's mental model. Users should be free to focus on the work they are trying to perform, rather than translating their tasks into the functions that the software program provides. Users should understand that their system password must be at least 6 characters, not how many bytes of storage a password must be.

Allow Users to Customize the Interface

Allow users to customize *information presentation* (colors, fonts, location, arrangement, view types), *interface behavior* (default actions, macros, buttons), and *interaction techniques* (keystrokes, shortcut keys, mnemonics, mouse button mappings). The rich visual and sensory environment of graphical and multimedia user interfaces requires users to be able to customize the interface. Users feel more comfortable and in control of the interface if they can personalize it with their favorite colors, patterns, fonts, and background graphics for their desktop.

> Key Idea! Today's operating systems offer a great deal of customization for interface elements. OS/2's settings views and Windows 95's properties dialogs allow users to set preferences for many operating system features and objects. Windows 95 developers even created an add-on utility called *Tweak UI*. Your products should use operating system settings to remain consistent with other applications. However, all other aspects of the product interface, including menus, buttons, can be customizable at the product level.

Allow Users to Directly Manipulate Interface Objects

Wherever possible, encourage users to directly interact with things on the screen, rather than using indirect methods, such as typing commands or selecting from menus. While you still must allow for both keyboard and mouse navigation and selection, you should optimize the interface toward users' most natural interaction style.

Coupled with the principle of making the interface transparent, users should feel like the interface isn't even there. When determining direct manipulation relationships, work within the interface metaphors and user models. The popular personal information manager, or PIM, Lotus Organizer, has its own waste basket. Drag an appointment or address book entry to the waste basket and watch what happens—the item bursts into flames! Is this the behavior you expect from a waste basket? I don't think so.

> Key Idea! Users begin to question their own beliefs if the results of direct manipulations don't match their own mental model of how things interact in the real world. A simple rule is: extend a metaphor, but don't break it. Sometimes direct manipulation interfaces fail because users don't know what they can pick up and where they can put it. Your interface objects should shout out to users, "Drag me, drop me, treat me like the object that I am!" If they don't, users won't know what to do. The one problem with direct manipulation is that it isn't visually obvious that things can be picked up and dragged around the screen. Users should feel comfortable picking up objects and exploring dragging and dropping them in the interface to see what might happen. The interface must be explorable.

At least let Users *Think* They're in Control

Users should be given some control of the user interface. In some situations, users should have total control of what they can do within a product or throughout the operating system. In other environments, users should only be allowed to access and use objects and programs needed for their work tasks. It is human nature for people to be frustrated when they want to go somewhere and they can't get there quickly, or they can't take the route they would normally take. Well, there are times where a computer system or program takes a certain amount of time to do some action or process and there is nothing that can speed it up. What do you do to keep users informed of the progress of the action and keep them from getting upset? Is there any way to keep users busy so they won't think that the computer is slow or not working?

Let's learn from the real world how to solve this problem. A maintenance supervisor for a large office building was getting complaints from hurried office workers in the building that the elevators were too slow. The supervisor knew that there was very little he could do to improve the speed of the elevators. He said to himself, "What can I do to take their minds off how slow the elevators are?" In his infinite wisdom, he figured out that if he installed mirrors inside the elevators, that just might keep passengers occupied while the elevators traveled at their normal speed. Guess what? He never got another complaint about the slowness of the elevators after the mirrors were installed. The problem with the speed of the elevators could not be solved, but *users' perception of the problem* could be influenced by the designer. Elevator floor information lights and sounds, both inside elevators and in elevator lobbies are also designed as visual and auditory progress indicators. Mechanical and electrical engineers know it is a good idea to show the elevator's status or progress, both to inform users and to let them pass the time quickly.

> Key Idea! A well-designed interface can comfort and entertain users while the computer system is completing a process. Users don't like to be left just sitting there doing nothing and seeing nothing on the computer screen while the computer is supposed to be doing something. Even if you can't let users be in control, let them think they are! At least entertain and teach them!

Many product developers (or probably their marketing staff) have realized they have a captive audience while users are installing a new program from numerous diskettes or a CD-ROM. I've seen a number of installation programs that advertise their products on the screen while users are just sitting there waiting for the program installation. Users can't help but read what is presented on the screen. Some products even use this time to teach users about the product so they can become productive sooner when they start using the product. That's a good use of the user's time!

Golden Rule #2: Reduce Users' Memory Load

The capabilities and limitations of the human memory system were discussed in Chapter 4. Based on what we know about how people store and remember information, the power of the computer interface should help users from having to remember information while using the computer. We aren't good at remembering things, so programs should be designed with this in mind. Figure 5-4 lists the design principles in this area.

1. Relieve short-term memory (remember)
2. Rely on recognition, not recall (recognition)
3. Provide visual cues (inform)
4. Provide defaults, undo, and redo (forgiving)
5. Provide interface shortcuts (frequency)
6. Promote an object-action syntax (intuitive)
7. Use real-world metaphors (transfer)
8. User progressive disclosure (context)
9. Promote visual clarity (organize)

Figure 5-4. Principles that Reduce Users' Memory Load

Relieve Short-Term Memory

If you remember (since it was discussed in Chapter 4, hopefully it's in your long-term memory by now!), short-term memory helps keep information available so you can retrieve it in a very short period of time. Users usually do many things at once, so computer interfaces shouldn't force them to try to keep information in their own short-term memory while they are switching tasks. This is a design principle that is often violated, causing users to use external memory aids, such as sticky pads, calculators, and sheets of paper, to jot things down they know they will need later in a customer transaction. It is such a simple interface principle, but one that is often neglected.

Program elements such as **undo** and **redo**, and clipboard actions like **cut**, **copy**, and **paste**, allow users to manipulate pieces of information needed in multiple places and within a particular task. Even better, programs should automatically save and transfer data when needed at different times and in different places during user tasks.

Key Idea! Don't force users to have to remember and repeat what the computer could (and should) be doing for them. For example, when filling in online forms, customer names, addresses, and telephone numbers should be remembered by the system once a user has entered them, or once a customer record has been opened. I've talked with countless airline, hotel, and car rental phone agents who know they need the same information a few screens later. They have to try to remember the information until they get to the later screen, or they have to write down the information while talking to the customer so they have it when they get to the other screen. The system should be able to retrieve the previous information so users don't have to remember and retype the information again.

Rely on Recognition, Not Recall

User interfaces support long-term memory retrieval by providing users with items for them to *recognize* rather than having to *recall* information. It is easier to browse a list to select an item rather than trying to remember the correct item to type into a blank entry field.

Take a look at common tasks performed in a popular program such as Quicken©. When entering information for a check written from a checking account, a calendar helps users select the appropriate date, a "next check number" selection enters the correct check number, and a list of memorized transactions can fill in the payee and the amount to be paid. Users can complete the task without even typing a word!

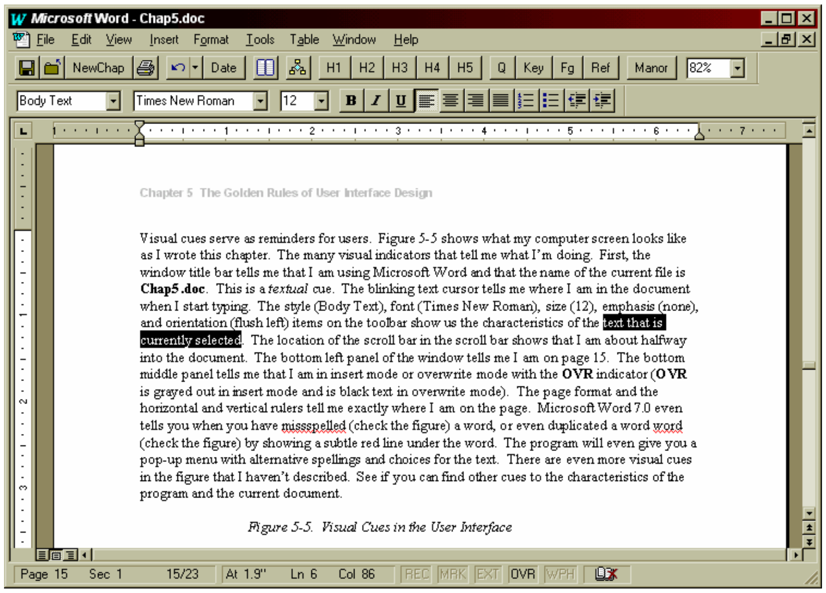
Online aids such as messages, tooltips, and context-sensitive help are interface elements that support users in recognizing information, rather than trying to remember information they may or may not know or have learned.

> Key Idea! Provide lists and menus containing selectable items instead of fields where users must type in information without support from the system. Why should users have to remember the two-character abbreviations for each state of the United States when they are filling out an online form? Don't force them to memorize codes for future use. Provide lists of frequently chosen items for selection rather than just giving them a blank entry field.

Provide Visual Cues

A necessary aspect of any *graphical* user interface (and of course, an object-oriented user interface) is that users must know *where* they are, *what* they are doing, and *what* they can do next.

Visual cues serve as reminders for users. Figure 5-5 shows what my computer screen looks like as I wrote this chapter. The many visual indicators that tell me what I'm doing. First, the window title bar tells me that I am using Microsoft Word and that the name of the current file is **Chap5.doc**. This is a *textual* cue. The blinking text cursor tells me where I am in the document when I start typing. The style (Body Text), font (Times New Roman), size (12), emphasis (none), and orientation (flush left) items on the toolbar show us the characteristics of the text that is currently selected. The location of the scroll bar in the scroll bar shows that I am about halfway into the document. The bottom left panel of the window tells me I am on page 15. The bottom middle panel tells me that I am in insert mode or overwrite mode with the **OVR** indicator (**OVR** is grayed out in insert mode and is black text in overwrite mode). The page format and the horizontal and vertical rulers tell me exactly where I am on the page. Microsoft Word 7.0 even tells you when you have misspelled (check the figure) a word, or even duplicated a word (check the figure) by showing a subtle red line under the word. The program will even give you a pop-up menu with alternative spellings and choices for the text. There are even more visual cues in the figure that I haven't described. See if you can find other cues to the characteristics of the program and the current document.


Microsoft Word - Chap5.doc
File Edit View Insert Format Tools Table Window Help
NewChap Date H1 H2 H3 H4 H5 Q Key Fg Ref Manor 82%
Body Text Times New Roman 12
Page 15 Sec 1 15/23 At 1.9" Ln 6 Col 86 REC MRK EXT OVR WPH

Chapter 5 The Golden Rules of User Interface Design

Visual cues serve as reminders for users. Figure 5-5 shows what my computer screen looks like as I wrote this chapter. The many visual indicators that tell me what I'm doing. First, the window title bar tells me that I am using Microsoft Word and that the name of the current file is **Chap5.doc**. This is a *textual* cue. The blinking text cursor tells me where I am in the document when I start typing. The style (Body Text), font (Times New Roman), size (12), emphasis (none), and orientation (flush left) items on the toolbar show us the characteristics of the text that is currently selected. The location of the scroll bar in the scroll bar shows that I am about halfway into the document. The bottom left panel of the window tells me I am on page 15. The bottom middle panel tells me that I am in insert mode or overwrite mode with the **OVR** indicator (**OVR** is grayed out in insert mode and is black text in overwrite mode). The page format and the horizontal and vertical rulers tell me exactly where I am on the page. Microsoft Word 7.0 even tells you when you have missspelled (check the figure) a word, or even duplicated a word word (check the figure) by showing a subtle red line under the word. The program will even give you a pop-up menu with alternative spellings and choices for the text. There are even more visual cues in the figure that I haven't described. See if you can find other cues to the characteristics of the program and the current document.

Figure 5-5. Visual Cues in the User Interface

Figure 5-5. Visual Cues in the User Interface

Key Idea! Whenever users are in a mode, or are performing actions with the mouse, there should be some visual indication somewhere on the screen that they are in that mode. The mouse pointer may change to show the mode or the current action, or an indicator might toggle on or off. Test a product's visual cues—walk away from the computer in the middle of a task and come back sometime later. Look for cues in the interface that tell you what you are working with, where you are, and what you are doing.

Provide Defaults, Undo, and Redo

Following the design principles for placing users in control, interfaces allow a wide variety of customizing features. With the power to change the interface, you must also give users the ability to reset choices and preferences to system or program defaults. Otherwise, users will be able to change their system colors, fonts, and settings so much that they may have no idea what the original settings were and how they can get them back.

While editing and manipulating data, such as writing text or creating graphics, undo and redo are very important to users. Undo lets users make changes and go back step by step through the changes that were made. Redo lets users undo the undo, which means that once they go back a few steps, they should be able to move forward through their changes again, step by step. Most programs allow users to undo and redo their last action, or maybe even the last few actions. A few programs can offer what is called "infinite undo." Many word processors actually save every keystroke and action during an entire working session. Users can move forward and backward, step by step or in larger increments, to restore a document to any state it was in during the session. Users can access material from hours before, copy some text that had been deleted, and then return their current work to paste the text.

Key Idea! Utilize the computer's ability to store and retrieve multiple versions of users' choices and system settings. Allow users to store current settings and possibly to save and name different settings. Provide multiple levels of undo and redo to encourage users to explore programs without fear.

Provide Interface Shortcuts

In addition to defining both keyboard and mouse techniques for interface actions, determine ways to shorten the number of keystrokes or mouse actions users need to perform common actions. Shortcut key sequences reduce users' memory load and quickly become automatic.

There are two basic ways to provide keyboard shortcuts—mnemonics and accelerator keys. A *mnemonic* (also called an access key) is a single, easy-to-remember alphanumeric character that moves the cursor to a choice and selects the choice. Mnemonics are used in menus (menu bars, pull-down menus, pop-up menus) and in lists to navigate and select an item in the list. Mnemonics must be unique to the current menu or list. A typical window menu bar configuration shows standard mnemonics—**F** for **File**, **E** for **Edit**, **V** for **View**, and **H** for **Help**. The next level of menus, pull-downs, each has their own set of mnemonics for items in the menu. For example, the **File** pull-down has **N** for **New**, **O** for **Open**, **C** for **Close**, and **S** for **Save**. Mnemonics speed up navigation and selection using menus and lists. To close the current window, users can key **Alt** (an accelerator key to navigate to the menu bar), then **F** (**File** pull-down), and **C** (**Close** action).

An *accelerator* (also called a shortcut key) is a key or combination of keys that users can press to perform an action. In the above example, **Alt** is a keyboard accelerator to move from within a window to the menu bar. Other common actions have standard accelerators, for example **Ctrl+P** for **Print**.

Key Idea! Once users are familiar with a product, they will look for shortcuts to speed up commonly used actions. Don't overlook the benefit you can provide by defining shortcuts and by following industry standards where they apply.

Promote an Object-Action Syntax

You don't need to build a fully object-oriented interface to benefit from using the object-oriented interaction syntax. Even an application-oriented program like a word processor follows this syntax. Select a word or some text (an *object*), then browse the menu bar pull-downs or click on the right mouse button to bring up a pop-up menu showing actions that can be performed (valid *actions*).

The object-action syntax was specified by Xerox PARC developers when they built the Star user interface in the late 1970's. The Xerox Star was introduced in 1981. Johnson et al (1989) described how this worked: "Applications and system features were to be described in terms of the *objects* that users would manipulate with the software and the *actions* that the software provided for manipulating objects. This 'objects and actions' analysis was supposed to occur at a fairly high level, without regard to how the objects would actually be presented or how the actions would actually be invoked by users. A full specification was then written from the 'objects and actions' version."

The benefits of the object-action syntax are that users do not have to remember what actions are valid at what time for which objects. First, select an object. Then only those actions that can be performed with that object are available. Unavailable menu bar actions are grayed out if they are not available for the selected object. A pop-up menu only lists available actions for the object.

> Key Idea! Consistent implementation of object-action syntax allows users to learn the relationships between objects and actions in the product. Users can explore and browse the interface by selecting objects and seeing what actions are available.

Use Real-World Metaphors

Real-world metaphors allow users to transfer knowledge about how things should look and work. Today's home computer comes equipped with a fully functional telephone, answering machine, and fax machine. How do users interact with these programs? They shouldn't have to learn anything new, since most users already know how to use these devices.

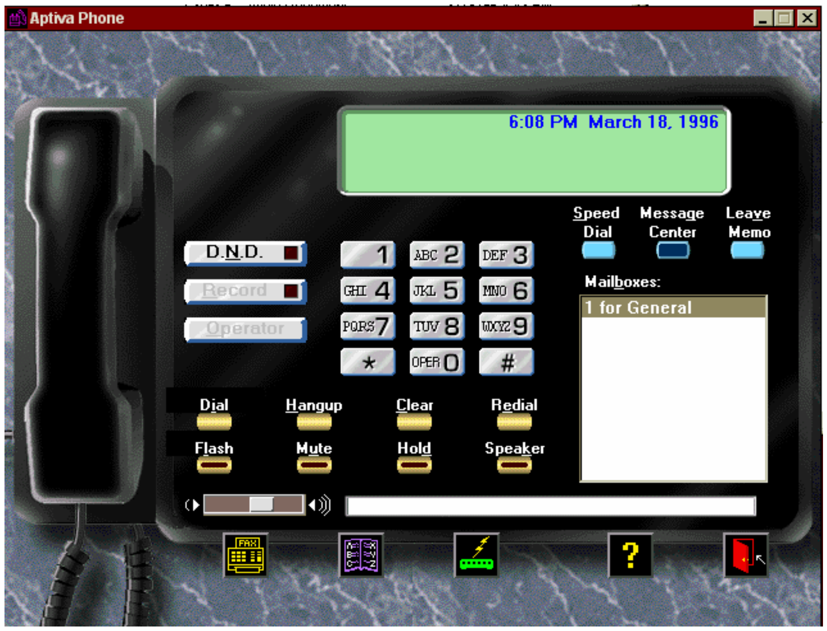


Figure 5-6. Real-World Metaphors in the User Interface

Figure 5-6 shows the interface for my computer's telephone system. Guess what? It looks like a telephone answering machine! It didn't take me very long at all to figure out how to use the telephone or answering system. I didn't even have to look at the brief documentation that came with the product. The same thing happens when users first see a Personal Information Manager, or PIM, such as Lotus Organizer. People know how to use organizers and Day-Timers™ already, so they have the experience and also have certain expectations about how an appointment organizer, address, and phone book should work.

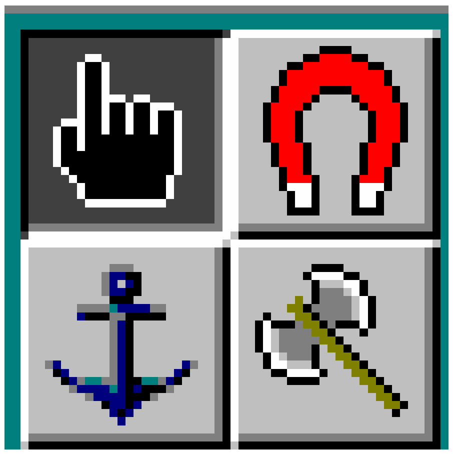


Figure 5-7. Lotus Organizer Version 1.0 Icon Metaphors

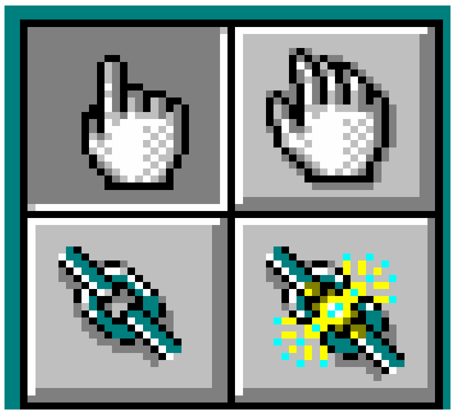


Figure 5-8. Lotus Organizer Version 2.0 Icon Metaphors

Lotus Organizer Version 1.0 used an icon of an anchor to represent the **Create Link** action (see Figure 5-7). This was not a very intuitive icon to use. Next to the anchor icon was an icon of an axe. What action did this button perform? You wouldn't guess—it represented breaking a link! How do the visual icons of an anchor and an axe represent these two related actions? Not very well. For the past few years I have used this example of inconsistent metaphors and poor icon designs. None of my students could figure out what these icons meant. Well, Organizer version 2 fixed this metaphor faux pas by using icons of a chain of links and the chain being broken to represent these actions (see Figure 5-8). I'm glad Lotus finally listened to me and (I'm sure) other designers and users about these obtuse icons!

Key Idea! Be careful how you choose and use interface metaphors. Once you have chosen a metaphor, stick with it and follow it consistently. If you find a metaphor doesn't work throughout the interface, go back and reevaluate your choice of metaphors. Remember—extend a metaphor, but don't break it.

Use Progressive Disclosure

Users should not be overwhelmed by what they can do in a product. You don't need to show users all of the functions the product offers. The best way to teach and guide users is to show them *what* they need, *when* they need it, and *where* they want it. This is the concept of *progressive disclosure*.

Some software programs offer graduated menus for users to choose from. User can choose *simple menus* that offer only common actions and function for casual use. After they feel comfortable with the product, or if they have a need for more sophisticated product features, they can choose to use the *advanced menus*. The key is that users are in control and they choose how much of the program and interface they see and work with.

New interface technologies such as wizards and assistants use progressive disclosure to guide users through common tasks. Wizards step users through tasks in a progressive manner where each step is simple and meaningful for even casual users.

Key Idea! Always provide easy access to common features and frequently used actions. Hide less common features and actions and allow users to navigate to them. Don't try to put every piece of information in one main window. Use secondary windows for information that is not key information.

Promote Visual Clarity

Apply visual design principles of human perception I discussed in Chapter 4, such as grouping items on a menu or list, numbering items, and by using headings and prompt text. Think of information on the screen in the same way as information you would present in any other medium.

The general principles of organization, continuity, gestalt, and so on should be followed. Most programs present too much information at one time on the screen. This results in visual clutter and users don't know where on the screen to look for information. Information should be presented with some priority and order so users can understand how information is organized on the screen. Remember the old adage, "form follows function." Some of these graphic principles are discussed in more detail in Chapter 13.

Avoid arbitrary groupings, distinctions, and other elements that seem to provide organizational information, but really don't. Figure 5-9 shows a window layout with haphazardly organized graphic objects and text, resulting in a "clown's pants" effect (from the Yale C/AIM WWW Style Manual, **http://info.med.yale.edu/caim/stylemanual**). This visual disorganization impedes usability and legibility, and users cannot browse or search for information in an orderly fashion.

Figure 5-10 shows a similar window layout using a carefully organized grid containing both graphic objects and text. The visual organization improves usability and legibility, allowing users to quickly find information they are looking for, resulting in increased confidence in their ability to use the information effectively.

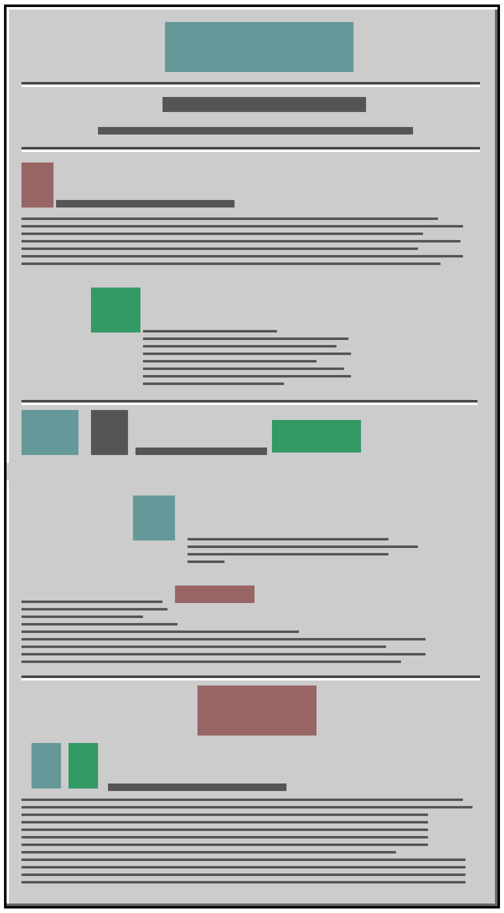


Figure 5-9. Poorly Organized Window, from Yale C/AIM WWW Style Manual

Key Idea! Graphic artists and book designers are skilled in the art of presenting information using the right medium and then designing the presentation of that information appropriately. These skills should be represented on the user interface design team.

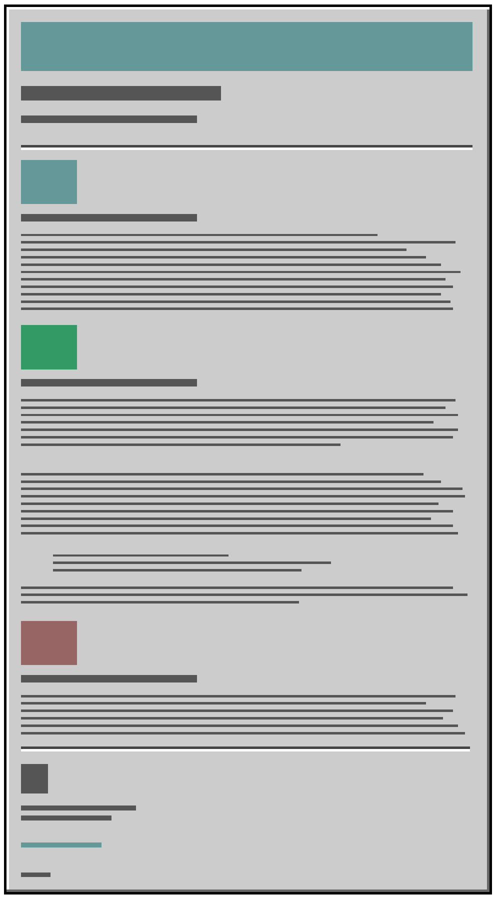


Figure 5-10. Well Organized Window, from Yale C/AIM WWW Style Manual

Golden Rule #3: Make the Interface Consistent

User interface consistency is a key aspect of usable interfaces. It's also a major area of debate. However, just like all principles, consistency might be a lower priority than other factors, so don't follow consistency principles and guidelines if they don't make sense in your environment. One of the major benefits of consistency is that users can transfer their knowledge and learning to a new program if it is consistent with other programs they already use. This is the "brass ring" for computer trainers and educators—train users how to do something once, then they can apply that learning in new situations that are consistent with their mental model of how computers work. Figure 5-11 lists the design principles that make up user interface consistency.

1. Sustain the context of users' tasks (continuity)
2. Maintain consistency within and across products (experience)
3. Keep interaction results the same (expectations)
4. Provide aesthetic appeal and integrity (attitude)
5. Encourage exploration (predictable)

Figure 5-11. Principles that Make the Interface Consistent

Next time you're on a commercial airplane, notice all of the little signs on the walls and doors of the toilets. Each of the signs has an identification number on one corner of the sign. This is done to ensure consistency in the signs you see on every airplane. It also simplifies the process of tracking and installing signs for airline workers. This is a good idea for help and message information you may develop for computer software programs.

Once you create an information message, give it a message identification number. Then, everywhere that the message is appropriate, use the same message number instead of writing the message again or writing a similar message. This will ensure that users will see the very same messages every time they are in the same situation no matter where they are in the program or in the system. This layer of consistency is very comforting and friendly to users.

Sustain the Context of Users' Tasks

Users should be provided points of reference as they navigate through a product interface. Window titles, navigation maps and trees, and other visual aids give users an immediate, dynamic view of where they are and where they've been. Users should also be able to complete tasks without having to change context or switch between input styles. If users start a task using the keyboard, they should be able to complete the task using the keyboard as the main interaction style.

Users should also be provided with cues that help them predict the result of an action. When an object is dragged over another object, some visual indicator should be given to users that tells them if the target object can accept the dropped object and what the action might be. Users should then be able to cancel the drag-and-drop action if they wish.

Context-specific aids such as help and tips on individual fields, menu items, and buttons, also help users maintain the flow of the tasks they are performing. They shouldn't have to leave a window to find supplemental information needed to complete a task.

Maintain Consistency Within and Across Products

One of the most important aspects of an interface is to enable users to learn general concepts about systems and products and then apply what they've learned to new situations in different programs or different parts of the system. This consistency applies at three levels: presentation, behavior, and interaction techniques. Consistency is one of the key issues behind user interface guidelines and standards discussed in the next chapter.

Consistency in *presentation* means that users should see information and objects in the same logical, visual, or physical way throughout the product. If information users can't change (*static text*) is in blue on one screen, then static text on all other screens should also be presented in blue. If a certain type of information is entered using one type of control, then use that same control to capture the same information throughout the product. Don't change presentation styles within your product for no apparent reason.

Consistency in *behavior* means that the way an object works is the same everywhere. The behavior of interface controls such as buttons, lists, and menu items should not change within or between programs. I've seen programs where the menu bar choices immediately performed actions, instead of displaying pull-down menus, as everyone expects. Users should not be surprised by object behaviors in the interface.

Interaction technique consistency is also important. The same shortcut keys should work in similar programs. Mouse techniques should produce the same results anywhere in the interface. Keyboard mnemonics should not change for the same menus from program to program. Users expect the same results when they interact the same way with different objects.

> Key Idea! Learning how to use one program should provide *positive transfer* when learning how to use another similar program interface. When things that look like they should work in a different situation don't, users experience *negative transfer*. This can inhibit learning and does not let users feel confident about the consistency in the interface.

Interface Enhancements and Consistency

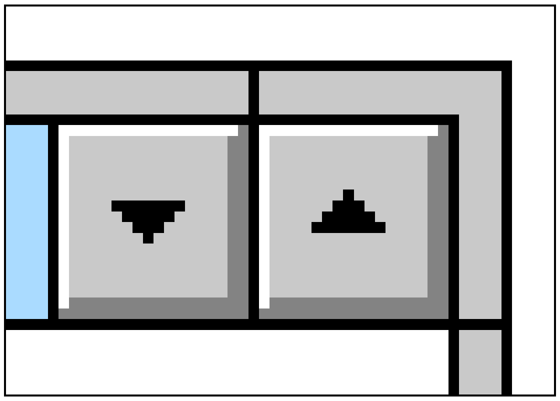


Figure 5-12. Windows 3.1 Title Bar Buttons

Windows 95 and OS/2 Warp are the current popular PC operating systems. Numerous visual "enhancements" were made to both interfaces. This example points out the power of consistency and the problems of inconsistency. Figure 5-12 shows the window title bar button configuration used in older versions of Windows—the familiar down and up arrows. The rightmost button (⮝) is the **Maximize/Restore** button and to its left is the **Minimize** button (⮟). Users can perform these actions by clicking on the system menu in the left corner of the title bar, and then choosing the window action.

This common task involves two mouse clicks, so the window sizing buttons on the right of the title bar represent quicker one-click ways to size windows. This title bar button configuration should look familiar to PC users, since over 50 million users (according to Microsoft) see this button configuration in their version of Windows. Users have learned to move the mouse to the top right button on the title bar to maximize a window or to restore it to its previous size and location. This is visual and positional consistency found in *every window* on their screen.

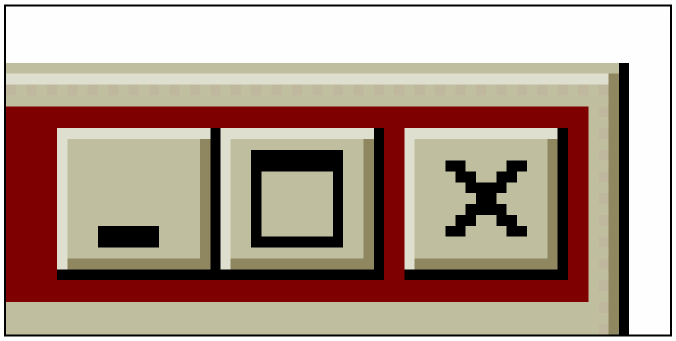


Figure 5-13. Windows 95 Title Bar Buttons

To close an opened window, users must click on the system menu and then select **Close**. A shortcut technique is to double-click on the system menu. Either technique is a two-click process. Windows 95 offered a usability "enhancement" by providing a one-click button as a way to close a window rather than the two-click methods. That's fine by me, but I don't like the way they did it (see Figure 5-13). The new **Close** button (with an **X** as the button icon) is now the rightmost button on the title bar and the size buttons are moved to the right. Instead of adding a *new* button and providing a *new* technique or button location to access that action, Microsoft added a *new* button and *changed* the way users interact with the traditional window size buttons. It may seem like a minor thing and an inconsequential change, but every time I use the mouse with Windows 95 I have to unlearn behaviors ingrained in me by previous Windows operating systems. I keep going to the rightmost button to maximize the window, and instead the window is closed. In my opinion, the benefit of the new Close button does not outweigh the inconsistent behavior users have to perform in unlearning years of mouse usage.

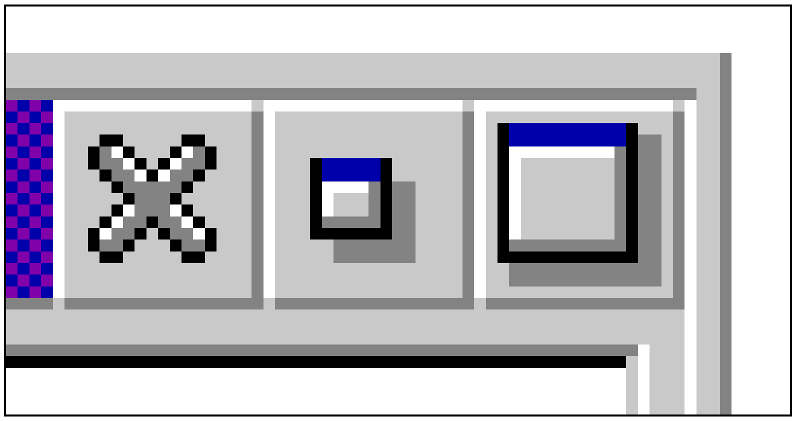


*Figure 5-14. OS/2 Warp with Object Desktop **Close** Buttons*

Take a look at how the same new concept was implemented in OS/2. The OS/2 Warp operating system uses slightly different graphic icons, but the window sizing buttons are in the same place as earlier versions of Windows. A new product, Object Desktop, developed by Kurt Westerfeld at Stardock Systems, provides the new **Close** button, but it is placed to the left of the window sizing buttons (see Figure 5-14). If users want to use this new button, they can quickly learn to move the mouse to the new position on the title bar, and *they don't have to change their already learned behavior* when using the window sizing buttons! Users of both operating systems, such as me, can transfer our learned behaviors between the two operating systems when common buttons retain their positional consistency across operating systems. I like this implement of the new interface feature much better than the Windows 95 method.

Key Idea! As both designers of products and also users of products, be aware of how you use learned behaviors and be careful how you introduce new behaviors. When interface enhancements are made, users should only have to learn a few new behaviors or techniques. They should not be forced to unlearn behaviors they have been using for years. Unlearning trained behavior is much more difficult than learning new behavior.

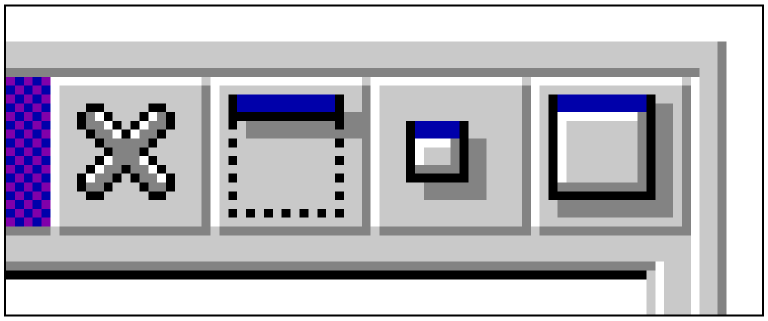


*Figure 5-15. OS/2 Warp with Object Desktop **Close** Buttons*

Object Desktop even added a new button, the **Rollup** button, to the window title bar (see Figure 5-15). Notice where it is placed on the title bar. The window size buttons are not changed, and the **Rollup** button is placed between the **Close** button and the **Minimize** button. This arrangement provides both *location* consistency and *position* consistency. The sizing buttons are always in the same location, providing consistency for common mouse actions. The **Close** button provides *position* consistency, that is, it is always in the same position with respect to the other buttons in the group, at the left of the group. Position consistency is also important in determining menu bar pull-down choice location, where certain menu bar pull-down choices will always appear in the same position, regardless of the other items in the pull-down list.

Keep Interaction Results the Same

As mentioned above, consistency in interface behavior is very important. If users experience different results from the same action, they tend to question their own behavior rather than the product's behavior. This leads to users developing *superstitious behavior*, that is, they think they must do things in just exactly a certain way for the desired result to happen; otherwise they are not sure of the results.

Sequences of steps and actions should also be consistent throughout a product. I've seen products where users had to logon multiple times to access different parts of the program. This was bad enough, but it was made worse because the logon process was different each time. Navigation sequences must also be consistent—don't use **Esc** to back up one step in one window and then use **Exit** in another window to do the same action.

Standard interface elements must behave the same way. For example, menu bar choices must always display a drop-down menu when selected. Don't surprise users by performing actions directly from the menu bar. Don't incinerate a discarded object when it is dropped in a waste basket!

> Key Idea! If by design, different results might happen than users expect, inform users before the action is performed. Give them a choice to perform the action, cancel the operation, or perhaps choose another action to perform.

Provide Aesthetic Appeal and Integrity

Many of today's products look like they were designed and developed by different people or even different divisions who never talked with each other. Users question the integrity of a product if inconsistent colors, fonts, icons, and window layouts are present throughout the product.

Just as a printed book has a predefined page layout, font, title, and color scheme, users should be able to quickly learn how product interfaces visually fit together. Again, utilize the skills of graphic designers on the design team effort.

> Key Idea! A pretty interface can't cover up for a lack of product functionality. Users don't just want "lipstick on the pig," they want a visually pleasing interface that allows them to get the job done.

Encourage Exploration

A goal for most user interface designers has been to produce user-friendly interfaces. A friendly interface encourages users to explore the interface to find out where things are and what happens when they do things, without fear of negative consequences. We are slowly achieving this goal, but users now expect even more from a product interface. They expect guidance, direction, information, and even entertainment while they use a product.

> Key Idea! Interfaces today and in the future must be more intuitive, enticing, predictable, and forgiving than the interfaces we've designed to date. The explosion of CD-ROM products and Internet browsers, home pages, and applets has exposed the user interface to a whole new world of computer users. It's time we moved onward past user-friendly interfaces to user-seductive and fun-to-use product interfaces, even in the business environment.

References

Apple Computer, Inc. 1992. *Macintosh Human Interface Guidelines*. Massachusetts: Addison-Wesley.

Hansen, W. 1971. User Engineering Principles for Interactive Systems. *AFIPS Conference Proceedings 39*. AFIPS Press: 523-532.

Heckel, Paul. 1984. *The Elements of Friendly Software Design*. New York: Warner Books.

IBM Corporation. 1992. Object-Oriented Interface Design: IBM Common User Acess Guidelines. New York: QUE.

Johnson, Jeff, Teresa Roberts, William Verplank, David Smith, Charles Irby, Marian Beard, and Kevin Mackey. 1989. The Xerox Star: A Retrospective. *IEEE Computer* 22(9): 11-29.

Mayhew, Deborah. 1992. Principles and Guidelines in Software User Interface Design. New Jersey: Prentice-Hall.

Microsoft Corporation. 1995. *The Windows® Interface Guidelines for Software Design*. Washington: Microsoft Press.

Nielsen, Jakob. 1990. Traditional Dialogue Design Applied to Modern User Interfaces. *Communications of the ACM* 33(10): 109-118.

Open Software Foundation. 1993. *OSF/Motif Style Guide, Revision 1.2*. New Jersey: Prentice Hall.

Rubenstein, R. and H. Hersch. 1984. *The Human Factor: Designing Computer Systems for People*. Massachusetts: Digital Press.

Shneiderman, Ben. 1992. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Massachusetts: Addison-Wesley.