GUI

The Elements of User Interface Design

CHAPTER FIVE

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 a self-assurance about how they work with computers. Their interactions with computer software should be charac-

terized by "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 most important when making design trade-offs. 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 followed blindly, rather they are meant as guiding lights for sensible interface design.*

The three areas of user interface design principles are:

- Place users in control of the interface.
- Reduce users' memory load.
- 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:

- Know the user.
- Minimize memorization.
- Optimize operations.
- 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 Corporation (IBM, 1992), Microsoft Corporation (Microsoft, 1995), and UNIX OSF/Motif (Open Software Foundation, 1993). 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 (files UIGUIDE.HLP, UIGUIDE.CNT, UIGUIDE.FTS, and UIGUIDE.GID). It can also be found on the Microsoft Web site at http:///www.microsoft.com/win32dev/uiguide/. 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 among the various 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 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.

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.

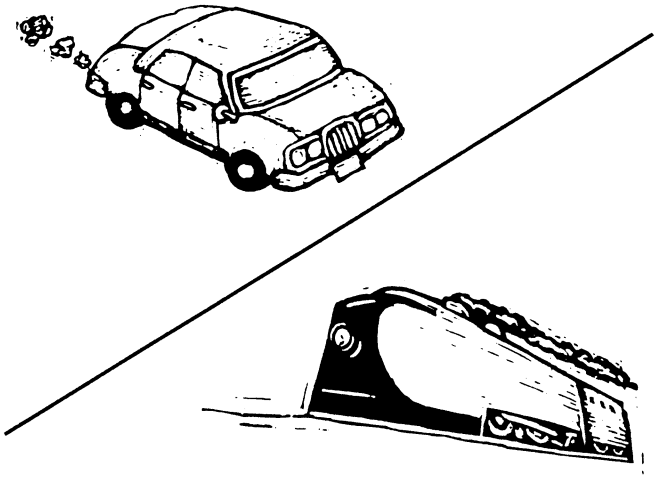

Figure 5.1 Do users want to take a train or drive a car?

The design principles that place users in control should be used appropriately to allow bank presidents systemwide 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 for all of its users.

KEY IDEA! *Wise designers let users do their work for them rather than attempting 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.

TABLE 5.1 Principles that Place Users in Control

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).

The principles that allow users to be in control are listed in Table 5.1. After each principle I've listed a key word to help fix it in your mind.

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? Because 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, when you are working with a database of information and choose the view data mode, the program will not allow you to add, delete, or modify the data record. You 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. In the case of limited access, display static text; if users 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.

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

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 about 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.*

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 Common User Access (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. As a user whose laptop mouse has been broken or disabled, or who lost the mouse pointer on the screen, or who's 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 to allow users to stay in control is to offer guidance through common tasks as an option. Casual users and novices will welcome the 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" (see Chapter 14) 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." I recently saw this system message 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! *Throughout the interface, use terms 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 be 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, is 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 used to tell 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. 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 experience 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 present 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 the DEL command, I usually use the DIR command immediately afterward to list the directory to see if 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, as well as 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 5.2 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.

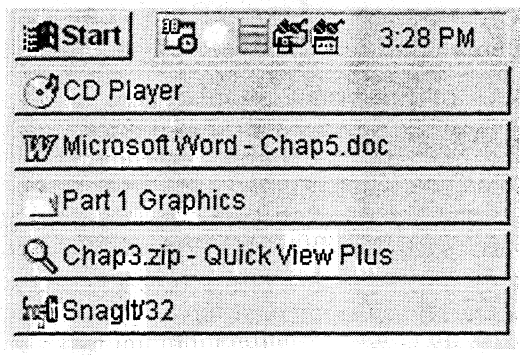

Figure 5.2 Microsoft Windows 95 taskbar and start button.

The many toolbars, launchpads, palettes, dashboards, and taskbars 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 percent 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 at different skill levels or with 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 aren't even aware of it. 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) not only lists 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. Users have no idea of what these programs are and where they came from, but they are free to choose any program from this list and, in doing so, inadvertently end that running task. This can be quite dangerous since the system is not well hidden from users.

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 but-

KEY IDEA! *The secret of a transparent interface is being in synch 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 simply that their system password must be at least six characters, and should not be concerned with how many bytes of storage that is.*

ton 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 properties 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 properties to remain consistent with other applications. However, all other aspects of the product interface, including menus and 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.

In addition to 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* (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 be allowed to access and use only 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 when 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 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 keep them occupied.

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 with it immediately. 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 keep users from having to do that work while using the computer. We aren't good at remembering things, so programs should be designed with this in mind. Table 5.2 lists the design principles in this area.

TABLE 5.2 Principles that Reduce Users' Memory Load

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. Use progressive disclosure (context).
9. Promote visual clarity (organize).

Relieve Short-Term Memory

As you may recall (and since it was discussed in Chapter 4, I hope 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 rely on external memory aids, such as sticky pads, calculators, and sheets of paper, to record what 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 than to try 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 num-

ber" 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 what they may or may not know.

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 5.3 shows what my computer screen looks like as I write this chapter. The many visual indicators 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 the characteristics of the text that is currently selected. 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.

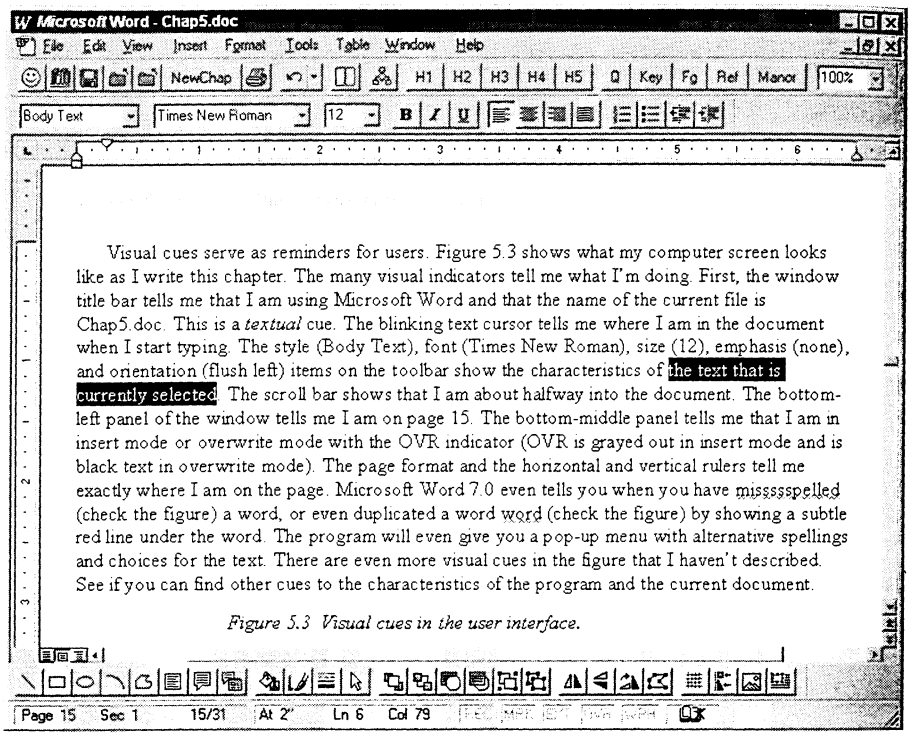

Figure 5.3 Visual cues in the user interface.

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 col-

ors, fonts, and properties so much that they may have no idea what the original properties 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 to their current work and paste the text.

KEY IDEA! *Utilize the computer's ability to store and retrieve multiple versions of users' choices and system properties. Allow users to store current properties and possibly to save and name different properties. 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 have 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 1970s. 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 benefit of the object-action syntax is 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 lists only 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 5.4 shows the interface for my computer's telephone system. Guess what? It looks like a telephone answering machine! It didn't take me long 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, 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.

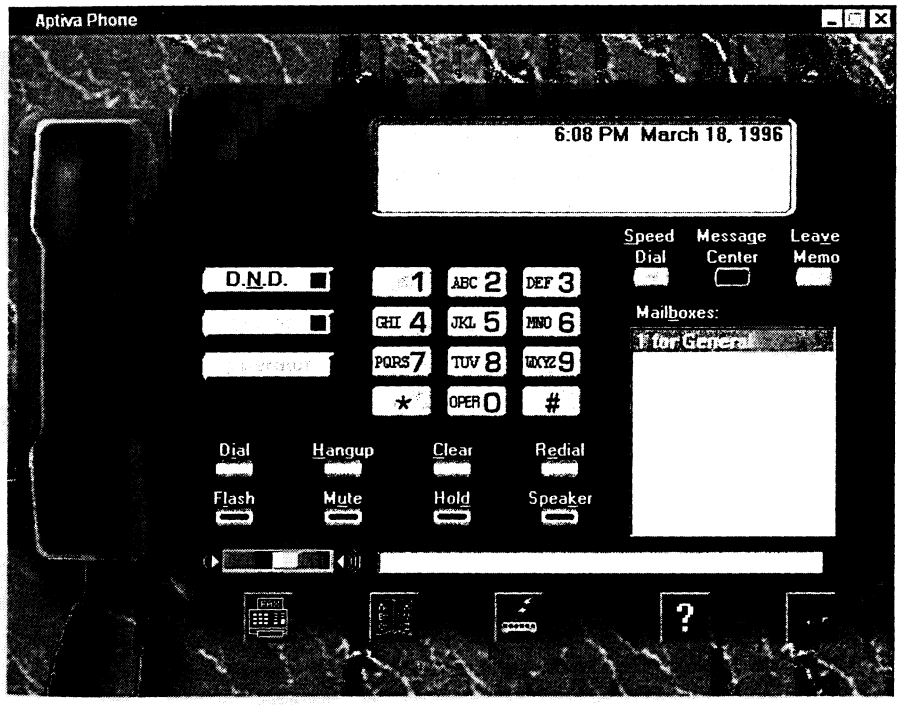

Figure 5.4 Real-world metaphors in the user interface.

Lotus Organizer version 1.0 used an icon of an anchor to represent the Create Link action (see Figure 5.5). 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 a broken chain to represent these actions (see Figure 5.6). 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. Users can choose *simple menus* that offer only common actions and functions for casual use. After they feel comfortable with the product, or if they need more sophisticated product features, they can 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

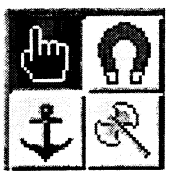

Figure 5.5 Lotus Organizer version 1.0 icon metaphors.

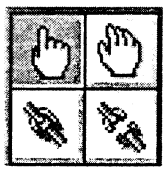

Figure 5.6 Lotus Organizer version 2.0 icon metaphors

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 (discussed in Chapter 4), such as grouping items on a menu or list, numbering items, and 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 it is organized. 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 5.7 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 5.8 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 what they are looking for, resulting in increased confidence in their ability to use the information effectively.

> **KEY IDEA!** *Graphic artists and book designers are skilled in the art of presenting appropriately designed information using the right medium. This skill should be represented on the user interface design team.*

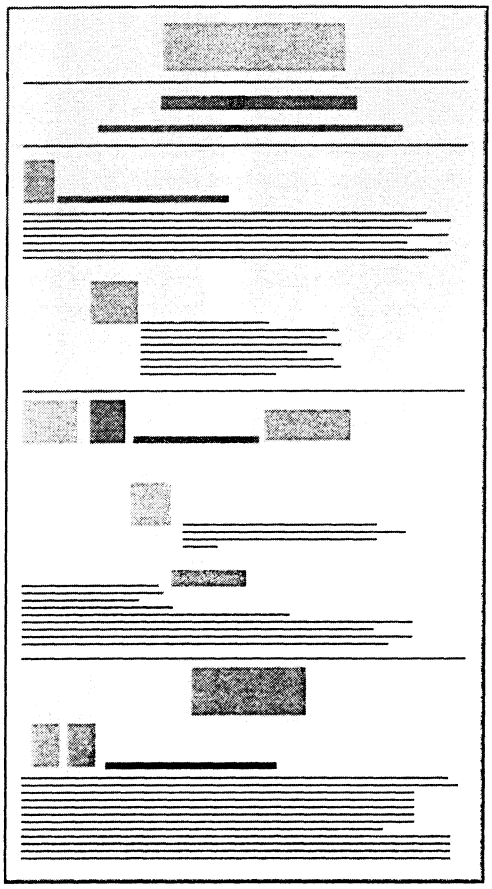

Figure 5.7 Poorly organized window, from Yale C/AIM WWW Style Manual.

Golden Rule #3: Make the Interface Consistent

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

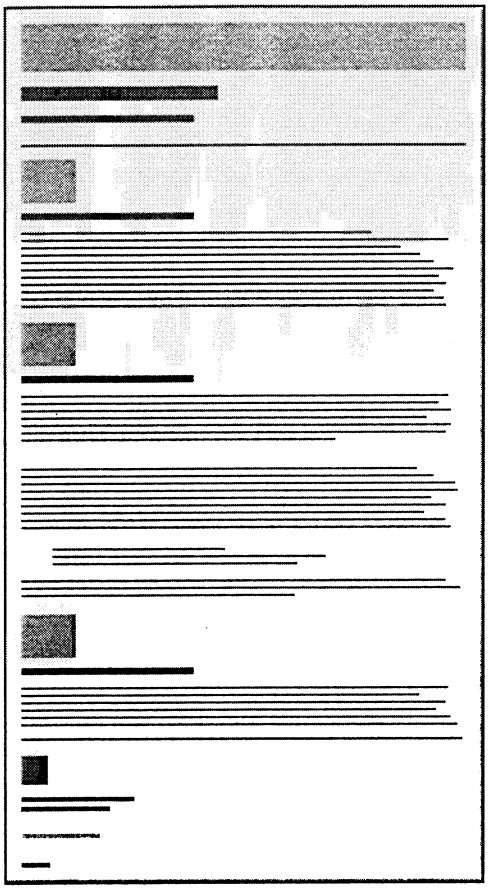

Figure 5.8 Well-organized window, from Yale C/AIM WWW Style Manual.

they can apply that learning in new situations that are consistent with their mental model of how computers work. Table 5.3 lists the design principles that make up user interface consistency.

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 in one corner. 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.

TABLE 5.3 Principles that Make the Interface Consistent

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 (predictability).

Once you create an information message, give it a message identification number. Then, everywhere 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 style of interaction.

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 be 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 the way it enables 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 an object works 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 the same in a different situation don't, users experience* negative transfer. *This can inhibit learning and prevents users from having confidence in the consistency of the interface.*

Interface Enhancements and Consistency

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 5.9 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 max-

imize 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.

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 5.10). 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 my 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.

Take a look at how the same new concept was implemented in OS/2. The OS/2 Warp version 3 operating system uses slightly different graphic icons, but the window-sizing buttons are in the same place as in 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 5.11). 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 myself, can transfer our learned behaviors between the two operating systems when common buttons retain their positional consistency across operating systems. I like this implementation of the new interface feature much better than the Windows 95 method.

Object Desktop even added a new button, the Rollup button, to the window title bar (see Figure 5.12). 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

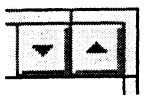

Figure 5.9 Windows 3.1 title bar buttons.

Figure 5.10 Windows 95 title bar buttons.

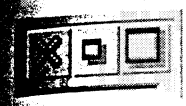

Figure 5.11 OS/2 Warp version 3 with Object Desktop Close button.

KEY IDEA! *As both a designer of products and a user 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 have to learn only 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.*

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. OS/2 Warp version 4 added the Close button to the window title bar (see Figure 5.13). Notice the graphics are 3-dimensional and embossed, and the Close button is to the left of the Minimize and Maximize buttons, in its new position.

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 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

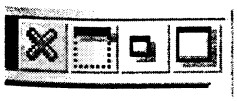

Figure 5.12 OS/2 Warp version 3 with Object Desktop Close and Rollup buttons.

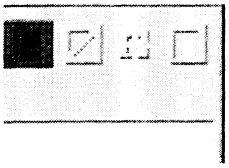

Figure 5.13 OS/2 Warp version 4 Close button.

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 results might be different from what users expect, inform them before the action is performed. Give them the option to perform the action, or cancel the operation, or perhaps perform another action.*

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.

KEY IDEA! *A pretty interface can't cover up for a lack of product functionality. Users don't just want lipstick on the bulldog, 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 nega-

tive 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 have exposed the user interface to a whole new world of computer users. It's time we moved 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.* Reading, MA: Addison-Wesley.

Hansen, W. 1971. User engineering principles for interactive systems. *AFIPS Conference Proceedings* 39, AFIPS Press, pp. 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 Access 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.* Englewood Cliffs, NJ: Prentice-Hall.

Microsoft Corporation. 1995. *The Windows Interface Guidelines for Software Design.* Redmond, WA: 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.* Englewood Cliffs, Prentice-Hall.

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

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

Continually incorporate iterative design revisions until you have met or succeeded the defined interface goals, or if you have reached a critical development deadline. Always keep users in mind when you reach critical design checkpoints. You must have a user interface designer involved who can be an advocate for the user community during the design process.

The Mandel Manor Hotels: An Interface Design Case Study

KEY IDEA! *A sample product is used in this chapter to illustrate design concepts and to show practical examples of user interface tips and techniques. This case study illustrates the steps of the interface design process. This product is fairly easy to describe and design in a book, yet it is representative of the type of software interfaces that you design and develop for your own business or products.*

Most of the projects I have worked on in the past few years have been in customer service industries, for example, banks, insurance companies, and telephone companies. The software programs designed are typically used in-house, where customer representatives collect and view information while interacting with a customer on the telephone or in person. This type of program is representative of many business software projects under development today.

As a software consultant and educator, I travel a great deal and constantly interact with hotel, airline, rental car, and travel agency representatives as I plan and conduct my activities around the world. I am a member of just about every frequent-flier and frequent-stayer program there is, so I know the *customer's viewpoint* of a reservation and customer service system very well. You may also be familiar with these systems from the traveler's viewpoint.

Welcome to a new chain of hotels called the Mandel Manor Hotels. As a new concept designed for the weary traveler, one of the goals is a program that would provide convenient ways for hotel customers to perform activities such as viewing hotel information, making reservations, and viewing their Mandel Manor Club "Honored Guest" program activity. Travel agents and Mandel Manor Hotels customer service representatives will also use this program to conduct activities related to their own particular job.

KEY IDEA! *A key design goal is to provide a similar interface for all users—customers, travel agents, and customer service representatives. With the proliferation of the Internet, this interface should run as a standalone PC program and as an Internet program accessed through the Web.*

Phase 1: Gather and Analyze User Information

Because system design is so intimately involved with how people work, designers need better approaches to gathering customer requirements. The current interest in participatory design, ethnographic techniques, and field research techniques grows out of the recognition that traditional interviewing and surveying techniques are not adequate for designing today's applications. These new approaches seek to improve requirements definition by creating new relationships between designers and customers.

Karen Holtzblatt and Hugh R. Beyer (1995)

We need to do a better job of thinking about technology from the point of view of all individuals—how they work, how they think, what they need to work and think more effectively.

Bill Gates (1990)

Here's where you must start—with your users. Before you can design and build any system, first define the problems that customers or users want solved and figure out how they do their jobs. Learn about your users' capabilities and the tasks they perform. Watch and learn from actual users of your programs. Notice the hardware and software constraints of their current computer systems and constantly remind yourself not to restrict your designs to what users currently can do with the system. Your solution must satisfy not only the current needs of users, but also their future needs.

There are key questions to ask during the user analysis phase. If you don't know the answers to these questions, don't assume that your design and development team or your marketing or sales group know the answers. The only way to find the answers is to observe users and ask them questions. A special section of *Communications of the ACM* (Holtzblatt and Beyer, 1995), entitled "Requirements Gathering: The Human Factor," devotes a number of articles to the emerging field of user analysis and requirements gathering. This is a good starting point if you are interested in learning more about this area.

Phase 1, gathering and analyzing activities, can be broken down into five steps:

1. Determine user profiles.
2. Perform user task analyses.
3. Gather user requirements.
4. Analyze user environments.
5. Match requirements to user tasks.

KEY IDEA! *There is an art to designing and asking questions, and to analyzing user feedback. Be careful what you ask users and how you analyze what they say. Borenstein (1991) says, "Listen to your users, but ignore what they say." Users tend to like whatever they are currently using, even though they will acknowledge that it could be better. It is difficult to get users to break out of their mindset concerning current systems and technology. Users are not necessarily aware of new technologies and business trends. The user analysis phase gathers information from and about users that will be factored in with parallel analyses of new technologies and business activities.*

Step 1: Determine User Profiles

User profiles answer the question, "*Who* are your users?" *User profiles* allow you to determine user demographics, skills, knowledge, background, and any other important information needed regarding *all* users of your planned system. How do you gather this information? Conduct user interviews and surveys, observe and videotape users, and also review other sources of information, such as industry reports, reviews, and press or marketing materials.

Step 2: Perform User Task Analyses

User task analyses determine *what* users want to do and *how* they accomplish their tasks. I will not cover a particular task analysis methodology here; rather, I present the questions that must be answered by any methodology. In addition to procedure-based task analysis, there are cognitive-based task analysis methodologies that focus on decisions users make while performing their work.

Whatever formal and informal task analysis techniques you use should answer these *what* and *how* questions:

- What tasks do users perform?
- What tasks are most critical?
- What steps are taken to perform tasks?
- What are users' goals for performing tasks?
- What information is needed to complete tasks?
- What tools (computer and otherwise) are used to complete tasks?
- What output is generated from user tasks?
- How do users do their work (manual, computer, telephone, etc.)?

- How do users interact with others in their tasks?
- How do tasks flow in the business processes?
- How frequently do users perform tasks?
- How would a computer or other computer software help users with tasks?

Step 3: Gather User Requirements

> *Building a system for a 2,000-pound gorilla makes it clear that user requirements are the driving force behind any project.*
>
> Jerreld Hannis (1994)

User requirements gathering and analysis answer the question, "*What* do your users expect the product and interface to do for them?" Almost all software development projects collect some user requirements. In addition to determining the *functional requirements* for a software product, user requirements should help determine the appropriate *user interface design* and what the interface should look and feel like. Here are key questions to ask:

- What core technologies do users require?
- How much are users and managers willing to pay for a product?
- Who installs the products?
- Who supports the products once they are installed?

User requirements are typically gathered through focus groups, structured interviews, and user surveys. Some common user requirements for business programs are that a new product should:

- Reduce paperwork.
- Reduce user errors.
- Automate current manual processes.
- Improve transaction processing speed.

Step 4: Analyze User Environments

User environment analysis answers the question, "*Where* do your users perform their tasks?" You must determine *environmental* characteristics that may impact how users do their work. Collect information regarding:

- Physical work environment (light, noise, space, temperature, computers, telephones, people, etc.)
- User location and mobility (office, home, on-site, mobile, etc.)
- Human factors, ergonomics, and physical considerations (vision, hearing, sitting/standing, keyboard abilities, etc.)
- Users with special needs (accessibility, and physical, cognitive, speech, and other disabilities)
- Internationalization and other cultural considerations (translation, colors, icons, text, messages, etc.)

Some environments should have more of an impact on product design than others. If you are developing a product for an office worker, the office environment may have a small impact. However, if you are developing a product for a hospital operating room or for traders on the Stock Exchange, the environment is critical to the design of the interface and also to the testing of the product interface.

There are numerous sources for guidelines, tips, and techniques in these areas of software design. Some design considerations might need to be built into your product, but many special design aids are already available in software operating systems. For example, both OS/2 and Windows 95 offer ways to configure the screen, keyboard, mouse, and other input or output devices for users with special needs. Chapter 14 of the Microsoft (1995) guide, "Special Design Considerations," is a well-written reference in this area.

Step 5: Match Requirements to User Tasks

> *We do not now (and in fact may never) understand human activities in enough detail to merely list the attributes computer systems would have to incorporate in order to meet these requirements.*
>
> John M. Carroll (1995)

Matching requirements to user tasks is a reality check. Sometimes user requirements may not be realistic for the tasks they are trying to perform. In that case, you must manage users' perceptions and requirements so they won't expect more than what they will get. Also, check to see if you may be going beyond what users really require to get their job done. If users and customers need only text information, don't go overboard and give them full multimedia capabilities at a huge cost that is "nice to have," but beyond their needs and requirements.

By reviewing user tasks and requirements, you will see where certain interface elements are required by users to perform tasks. In the case study, in addi

tion to text information, customers need to be able to view and print photographs of hotels and maps of hotel locations. This tells you to design an interface that supports more than just textual information. This matching process also tells you about product function that must be implemented to enable customer requirements.

For example, Mandel Manor Hotels customers demand up-to-date account information. This requires system databases to be frequently updated so customer and member information is current to within a few days of a customer request.

Results of Phase 1

Both customers (including Mandel Manor Club members) and customer service representatives are users of the software program. Results of phase 1 analyses are described in Table 12.1. This is a sample of analysis results, not a complete listing. For example, there are obviously more than five or six tasks that can be performed with the product. Results from each step should also have detailed backup information describing the source of the information, how the results were calculated, and how to access source data if needed.

Returning to the User Analysis Phase

The iterative method promotes returning to the user analysis stage to check whether your user profiles, tasks, environments, or requirements have changed during the design and development process. You must periodically return to phase 1 to update your user analyses. Otherwise, you might build a good product, but it won't meet current and future user needs. It would meet only their past needs.

Phase 2: Design the User Interface

> *On the order of 70 percent of product cost is spent on design of the user interface.*
>
> Bill Buxton (1991)

Designing the user interface for a software product usually requires a significant commitment of time and resources. The design phase progresses through a number of well-defined steps that should be followed *in sequence.* It is very tempting to start coding the final product now, rather than designing the interface. Follow the steps through the design process before you code the final design.

TABLE 12.1 Phase 1: User Analysis Results

Analysis Results	Customers/Program Members	Service Representatives
Step 1: User Profiles	◆ Female and male ◆ Adult ◆ Mostly USA, some International ◆ English-speaking skills ◆ Minimal computer skills ◆ No previous knowledge of program	◆ Female and male ◆ Adult ◆ Mostly USA, some International ◆ English-speaking skills ◆ Intermediate computer skills ◆ Previous training on program
Step 2: User Tasks	◆ View hotel information and rates ◆ Make/view/update reservations ◆ View/update certain account information ◆ Complete Club application ◆ Redeem hotel awards ◆ Print/save information and reservations	◆ View hotel information and rates ◆ Make/view/update reservations ◆ View/update all account information ◆ Complete Club application ◆ Redeem hotel awards ◆ Make account/reservations corrections and revisions ◆ Offer specials and discounts
Step 3: User Requirements	◆ Requires little or no training ◆ Requires little time ◆ Available 24 hours a day ◆ Able to view and print graphic and text information about hotels (pictures, maps, directions, etc.) ◆ Familiar "windows" interface style ◆ Successful task completion ◆ No other tools/programs required ◆ Up-to-date account and reservation information provided ◆ Tasks can be interrupted and canceled	◆ Requires minimal training ◆ Able to use program while customer is on telephone ◆ Similar interface style to customer's product ◆ Requires minimal technical support ◆ Up-to-date account and reservation information provide ◆ Acceptable system response time for customer on the telephone ◆ Tasks can be interrupted and canceled

Step 4: User Environments	◆ PC—Standalone program using local database	◆ Networked PCs used in a telephone-based customer service environment
	◆ PC—Internet program using system database	◆ Multiple service representatives using system simultaneously
	◆ Usable at home, office, or while traveling	◆ Multiple service representatives accessing networked database simultaneously
	◆ Minimal computer system and telephone requirements	◆ Standardized PC workstations and telephone systems
Step 5: Matching Requirements to User Tasks	◆ Hotel photos, maps, and other information must be available for viewing and printing to meet customer requirements	◆ Networked PC system must be capable of handling multiple customer service representative requests simultaneously with customers on the telephone
	◆ Customer profile and account information must be up to date to meet customer requirements	◆ Similar tasks must be performed with the same interface by service representatives and customers to ensure service representatives can support customer help requests

Phase 2, design, includes the following steps:

1. Define product usability goals and objectives.
2. Develop user scenarios and tasks.
3. Define interface objects and actions.
4. Determine object icons, views, and visual representations.
5. Design object and window menus.
6. Refine visual designs.

Step 1: Define Product Usability Goals and Objectives

Interface usability was discussed in detail in Chapter 7. Early in a product's design, you must determine just what you expect the product to do for users so those goals are clearly embedded in the minds of everyone working on the project. I was often amazed in some consulting projects that no one on the product development team could state or point to any documented product usability goals and objectives.

KEY IDEA! *How can you tell if a product works if you haven't even defined what is a usable system? It's like drawing a target around an arrow after it has been shot, rather than aiming an arrow at a target and seeing how close you got to it!*

Design goals are best expressed in terms of users' behavior and their performance terms, such as how long tasks take and how many errors are expected. Here are four areas in which goals and objectives are most appropriate (see Chapter 7 for more information):

- Usefulness
- Effectiveness
- Learnability
- Attitude

Table 12.2 lists some sample product goals and objectives for Mandel Manor Hotels system. These goals and objectives should drive the design of the product interface.

TABLE 12.2 Phase 2: Product Goals and Objectives

Product Goals and Objectives	Customers/Program Members	Service Representatives
Usefulness	Goal: ◆ Users will be able to use the program to perform their tasks Objective: ◆ 100% of the users will be able to use the system to perform a task after the first attempt	Goal: ◆ Users will be able to use the program to perform their tasks Objective: ◆ 100% of the users will be able to use the system to perform a task after training
Effectiveness	Goal: ◆ Users will be more productive using the program (compared to current manual processes) Objective: ◆ 100% of the users will complete their tasks within given time constraints	Goal: ◆ Users will be more productive using the program (compared to current manual processes) Objective: ◆ 100% of the users will complete their tasks within given time constraints
Learnability	Goal: ◆ Minimal user training will be needed Objective: ◆ Users will be able to use the product successfully after doing a tutorial	Goal: ◆ Minimal user training will be needed Objective: ◆ Users will be able to use the product successfully after completing 4 hours of training
Attitude	Goal: ◆ Users will be satisfied with the product Objective: ◆ Users will rate their satisfaction with the product at a high level	Goal: ◆ Users will be satisfied with the product Objective: ◆ Users will rate their satisfaction with the product at a high level

Step 2: Develop User Scenarios and Tasks

Computer systems and applications can, and should, be viewed as transformations of user tasks and their supporting social practices. In this sense, user interaction scenarios are a particularly pertinent medium for representing, analyzing, and planning how a computer system might impact its users' activities and experiences. They comprise a vocabulary for design and evaluation that is rich and yet evenly accessible to all the stakeholders in a development project.

John M. Carroll (1995)

It is difficult to define exactly what a scenario is, compared to a user task. A *scenario* is typically a high-level description of what the user does. I like to describe a scenario as a *sequence of user tasks* or events that make up a common transaction. "Buying lunch" is an example of a high-level scenario that would be composed of many smaller tasks such as going to a restaurant, ordering food, eating lunch, and paying for the meal. Tasks can be further decomposed into *subtasks,* if necessary.

KEY IDEA! *Remember, this is an iterative design process! The scenarios you develop now to drive the definition of the user interface will be the basis for testing the usability of the interface later. If scenarios don't drive the design of the appropriate interface, it is likely that later the interface will drive the scenarios. That is, you actually will develop scenarios that the* interface will allow, *rather than scenarios that* users want to perform.

In Step 2, develop as many user scenarios as possible. The more scenarios you develop, the less likely you will miss any key objects and actions needed in the user interface. If you have a range of users and skills, be sure to develop scenarios that cover the entire range of users and skills performing the range of tasks. Designing computer systems based on user scenarios has become an important part of user-centered system design. For a more theoretical discussion of scenarios, read John Carroll's (1995) book, *Scenario-Based Design: Envisioning Work and Technology in System Development.*

Table 12.3 presents some sample user scenarios and tasks for the Mandel Manor Hotels system.

Step 3: Define Interface Objects and Actions

This step is probably the most difficult (and most important) stage in the design process. If you can't figure out the appropriate set of objects, how can you expect users to understand, remember, or use the objects?

There are a number of activities in Step 3:

a. Derive objects, data, and actions from user scenarios and tasks.
b. Review and refine object and action lists with users.
c. Draw an object relationships diagram.
d. Complete an object direct-manipulation matrix.

TABLE 12.3 Phase 2 Output: Scenarios and Tasks

Customers/Program Members	Service Representatives
Scenario: Customer wishes to update telephone number in customer profile and see if latest hotel stay has been recorded. Customer also sends a comment card to the company. Customer Tasks: ◆ Enters Mandel Manor Club account number and password ◆ Opens customer profile ◆ Updates telephone number ◆ Opens account summary ◆ Views latest recorded hotel stay ◆ Fills out comment card ◆ Sends comment card	Scenario: Customer telephones wishing to redeem account points for free hotel stay award and reserve a hotel room Customer wishes to receive a fax of the reservation and award. Service Representative Tasks: ◆ Finds member's Mandel Manor Club account ◆ Validates member's password ◆ Opens account summary ◆ Redeems hotel award ◆ Views hotel requested by customer ◆ Reserves hotel stay for customer ◆ Sends fax of hotel award and reservation confirmation to customer
Scenario: Customer wishes to view hotels in Cancun, Mexico, print hotel information, and reserve a hotel stay for his family vacation. Customer Tasks: ◆ Enters Mandel Manor Club account number and password ◆ Looks for hotels in Cancun, Mexico ◆ Prints hotel photograph, map, and directions from airport ◆ Checks hotel availability ◆ Checks hotel rates ◆ Reserves two rooms for family vacation	Scenario: Customer telephones to correct an error in the room rate charged during a recent hotel stay and receive a credit card refund. Service Representative Tasks: ◆ Finds member's Mandel Manor Club account ◆ Validates member's password ◆ Finds hotel stay statement ◆ Verifies customer's rate correction ◆ Revises hotel stay statement ◆ Credits refund to member's credit card ◆ Sends fax of revised hotel stay statement and credit card refund receipt

First, define your initial set of user objects and actions based on the information collected in Phase 2 and from the scenarios you just developed in Step 2. It is easy to start listing objects and actions—just underline all of the nouns in your scenarios and tasks and circle (here *italicized*) all of the *verbs*. This gives you a first pass at your objects and data (nouns) and actions (*verbs*) that can be applied to those objects and data.

Table 12.4 illustrates the scenarios after you have identified objects and *actions* for the Mandel Manor Hotels system.

KEY IDEA! *Use a brainstorming approach as you first start to build your object and action lists. That is, use any materials you can find related to tasks and scenarios, write down as many objects and actions as possible, and don't worry about terminology or redundant items.*

Continue to brainstorm with your lists of objects and actions until you have a somewhat final list that eliminates duplication and contains other objects and actions that were not necessarily explicitly mentioned in the user scenarios.

Usually, any major object (for example, hotel or reservation) with (possibly) more than one instance requires a high-level container object to hold the entire collection of individual objects. In a car dealership system, for example, since there are multiple car objects, there probably should be a car lot object. These high-level container objects may end up as folders in the interface, or simply list controls. But for now, it is important not to forget that collections of similar objects typically are contained in a higher-level container object.

You will also need to define general device objects that users need to perform actions you have defined. For example, you may need printer and fax objects. You might also some kind of trash or wastebasket object.

As you refine your list of objects, start to think of what object type each object is. Remember our definition of the types of objects—data, container, and device. At this stage you may be fuzzy about objects types for some objects. As I discussed earlier, it may be difficult to figure out if some objects, such as the customer object, are data objects or container objects. Obviously there is a lot of data associated with a customer, but in the interface, users may see the customer as a container that holds *other* objects that are pieces of customer data, such as a customer profile or customer account. These issues tend to work themselves out through discussions with users as you move through this phase of interface design.

KEY IDEA! *You may think you have completed a* final *list of objects and actions after this step, but I guarantee you will return here multiple times (remember this iteration thing!) to revise and refine your lists of objects and actions.*

TABLE 12.4 Phase 2: Identifying Nouns and Verbs in Scenarios and Tasks

Customers/Program Members	Service Representatives
Scenario: Customer wishes to *update* telephone number in customer profile and see if latest hotel stay has been *recorded.* Customer also *sends* a comment card to the company.	Scenario: Customer *telephones* wishing to *redeem* account points for free hotel stay award and *reserve* a hotel room. Customer wishes to *receive* a fax of the reservation and award.
Customer Tasks: ◆ *Enters* Mandel Manor Club account number and password ◆ *Opens* customer profile ◆ *Updates* telephone number ◆ *Opens* account summary ◆ *Views* latest recorded hotel stay ◆ *Fills out* comment card ◆ *Sends* comment card	Service Representative Tasks: ◆ *Finds* member's Mandel Manor Club account ◆ *Validates* member's password ◆ *Opens* account summary ◆ *Redeems* hotel award ◆ *Views* hotel *requested* by customer ◆ *Reserves* hotel stay for customer ◆ *Sends* fax of hotel award and reservation confirmation to customer
Scenario: Customer wishes to *view* hotels in Cancun, Mexico, *print* hotel information, and *reserve* a hotel stay for his family vacation.	Scenario: Customer *telephones* to *correct* an error in the room rate *charged* during a recent hotel stay and *receive* a credit card refund.
Customer Tasks: ◆ *Enters* Mandel Manor Club account number and password ◆ *Looks for* hotels in Cancun, Mexico ◆ *Prints* hotel photograph, map, and directions from airport ◆ *Checks* hotel availability ◆ *Checks* hotel rates ◆ *Reserves* two rooms for family vacation	Service Representative Tasks: ◆ *Finds* member's Mandel Manor Club account ◆ *Validates* member's password ◆ *Finds* hotel stay statement ◆ *Verifies* customer's rate correction ◆ *Revises* hotel stay statement ◆ *Credits* refund to member's credit card ◆ *Sends* fax of revised hotel stay statement and credit card refund receipt

Table 12.5 shows the final list for the Mandel Manor Hotel example.

All objects on your list may not necessarily end up as separate interface objects in the final design. Some may become specific *views* of major objects. For example, a hotel will definitely be a key object in the final design. However, other things such as hotel information, photographs, maps, rates, and availabil-

TABLE 12.5 Phase 2: Objects, Object Types, and Actions Lists

Objects and Data	Object Type	Actions (Not Specific to Objects)
Customer list	container	
Customer	container/data	call
Mandel Manor Hotel list	container	cancel
Service representative	data	correct
Customer profile	data	credit
Account number	data	
Password	data	debit
Telephone number	data	
Account summary	data	fill out
Account points	data	find
Hotel stay	data	
Hotel stay statement	data	look up
Hotel reservation list	data	
Hotel reservation	data	open
Reservation confirmation	data	print
Hotel list	data	
Hotel	data	receive
Hotel information	data	record
Hotel rates	data	redeem
Hotel availability	data	request
Hotel photographs	data	reserve
Hotel map, directions	data	revise
Hotel stay award	data	
Hotel special	data	send
Comment card	data	update
Credit card	device	validate
Customer finder	device	verify view
Printer	device	
Fax machine	device	
Wastebasket	device	

ity may be data contained in the hotel object or separate views of a hotel object. Objects and views were defined in the previous two chapters. You will create their visual representations later in this phase.

Now that we have some idea of the objects users might expect, let's start to figure out how these objects relate to each other. One way to do this is to draw an *object relationships diagram* (see Figure 12.4).

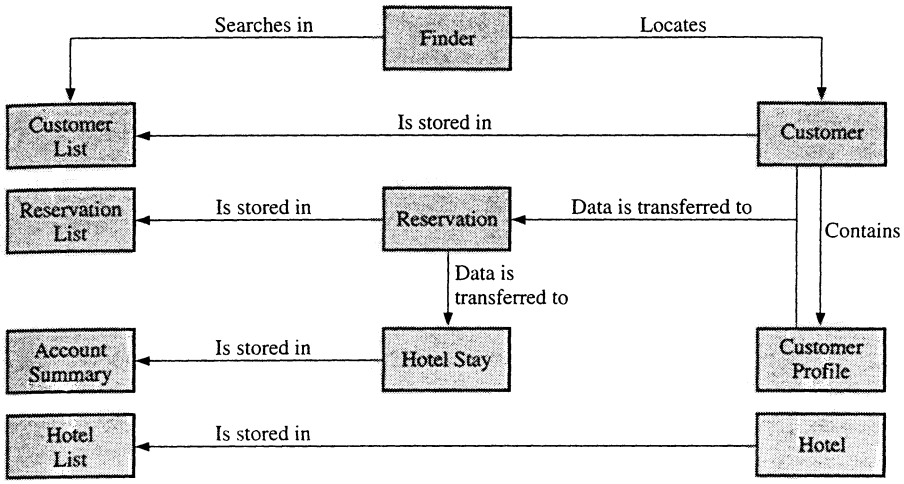

Figure 12.4 Phase 2: Object relationships diagram.

KEY IDEA! *A user interface object relationships diagram is different from traditional programming diagrams of object relationships.* User interface objects do not necessarily parallel programming or business objects.

One goal of user interface design is to hide as much of the complexity of the underlying business and programming models as possible. Users drag an object to a wastebasket—this may start a whole series of business transactions to delete or archive a database record—but all users care about is that the object was thrown away.

Look at the objects on the left side of the object relationship diagram. These high-level containers or lists represent underlying business and programming structures of the system. These relationships will show the database transactions your system will be processing on a regular basis. Bringing up a customer account on the screen may require accessing multiple databases under the covers, but users do not need to know what went on at the system level. Users see these objects only as tools for getting and storing information from the system—they don't need to know how they work. These are *system objects* rather than *interface objects.* Be careful how you represent them on the computer screen!

KEY IDEA! *Hide the functionality of "system" objects (such as customer lists, reservation lists, and hotel lists or databases) from users. They should only see these objects through a "finder" object or in lists of available items. From users' perspective, these are big containers back somewhere in the computer system that store all of the objects they work with. They don't want to know any more about these objects than is necessary to retrieve or store information.*

Next, continue Step 3 by thinking how (and, of course, if!) users would directly manipulate objects using a mouse or other input devices to perform their tasks. To do this, complete an *object direct-manipulation matrix* (Figure 12.5). Simply list all of your key objects down the left column of the matrix and across the top row. Then think about which objects might be dragged (*source objects*) and dropped on other objects (*target objects*). If it makes sense for an object to be dragged and dropped on another object, fill in that cell with the result of the direct manipulation. For example, an account summary could be dragged and dropped on a printer icon, thus printing a copy of the account summary. I have used only key objects in this chart to make it more readable.

KEY IDEA! *Notice that the matrix is not filled for most of the cells. Not all objects are candidates for drag-and-drop operations and you see that some objects are always used in certain ways as either source or target objects.*

Your devices (printer, fax, and wastebasket) will typically only be *target* objects for drag-and-drop operations. not *source* objects. As a target object, the actions performed when an object is dropped on it are listed in the column for that target object.

Also, look at the system objects (customer list, reservation list, and hotel list). Users work with objects presented in these lists, not the lists themselves. These list objects are not source objects for direct manipulation, so there should be no actions in their row of the matrix. For example, a customer in the customer list might be dragged onto a reservation to transfer the information about the customer and his hotel preferences to the reservation. Users might drag objects to these lists to complete a transaction such as a making a reservation or adding a customer to the system.

Source Object	Target Object										
	Customer	Customer List	Customer Profile	Service Rep.	Reservation	Reservation List	Hotel	Hotel List	Printer	Fax	Wastebasket
Customer		add customer to list		assign customer to service rep.	transfer data to reservation				print all customer info.	fax all customer info.	delete customer
Customer List											
Customer Profile					transfer data to reservation				print customer profile	fax customer profile	delete customer profile
Service Rep.									print service rep. info.	fax service rep. info.	
Reservation	attach reservation to customer			assign reservation to service rep.		store reservation in list			print reservation	fax reservation	delete reservation
Reservation List											
Hotel					transfer data to reservation			add hotel to list	print hotel info.	fax hotel info.	
Hotel List											
Printer											
Fax											
Wastebasket											

Figure 12.5 Phase 2, Step 3: Object direct-manipulation matrix.

KEY IDEA! *The work you've done in Step 3 is necessary to give you all of the background material on interface objects and actions. Now it is time to begin drawing representations of the objects you have identified. This is the fun part of design. All the work you have done up to this point begins to pay off here.*

In fact, it is a good idea to review your results so far from Phase 2 analysis with your users. Don't wait until you have completed the entire design phase before you go back to users for review and validation.

Step 4: Determine Object Icons, Views, and Visual Representations

Once objects have been defined, determine how to best represent them on the screen and how users will view these objects and the information they contain. When determining object views, consider the way users will interact with each object and its information.

Design Object Icons

A graphics designer should be an integral part of the interface design team and should be responsible for designing screen icons. User feedback and usability testing should also be incorporated to ensure icons are recognizable, understandable, and helpful in performing their tasks. Figure 12.6 shows the first pass at object icons—hand-drawn pictures of objects. Don't spend too much time early in the interface design on icons. Start with rough hand-drawn sketches and incrementally design and test icons through the entire design process. There are good books that cover icon design, notably William Horton's (1994) *The Icon Book: Visual Symbols for Computer Systems and Documentation.*

After hand drawing object icons, select icons from icon libraries or draw actual icon files for your demos and prototypes. Figure 12.7 shows a first attempt at computer icons for these objects.

KEY IDEA! *Developers don't view objects the same way as do users. Therefore, it makes sense that developers shouldn't design object icons. Let graphics designers work with users to design icons.*

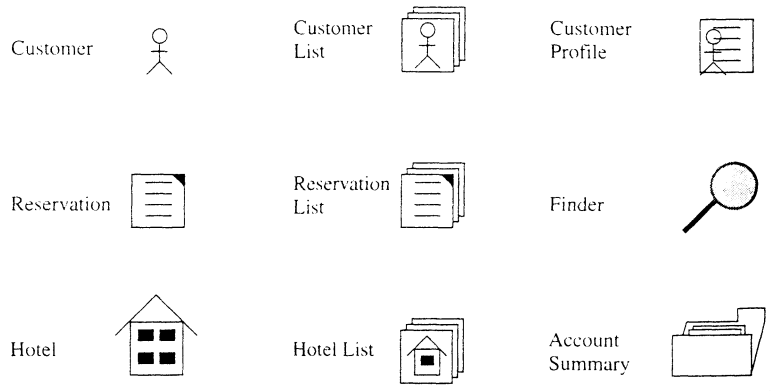

Figure 12.6 Hand-drawn object icons.

Figure 12.7 Computer-drawn object icons.

Determine Object Views

Next, determine what types of views each object will have. The type of view will help you immediately determine window elements, menus, and layouts. Table 12.5 lists object types for all objects found in Step 3. Table 12.6 lists specific views needed for some of the key objects.

Design Visual Representations

KEY IDEA! *Once you have created computer icons for objects, you can show how some objects—containers—will look when opened in a contents view, because they will basically contain some set of objects represented as icons.*

For instance, users of the Mandel Manor Hotels system first see the Desktop folder containing the objects they need to start most customer transactions—the Finder, Hotel List, and Special Promotions Folder. This folder is shown in Figure 12.8.

TABLE 12.6 Objects and Views

Objects	Object Type	Object Views
Customer	Container	Contents: Icons Details
Customer List	Container	Search Results Window
Customer Profile	Data	View/Update Information: Account number Password Name, address Telephone numbers
Account Summary	Container/Data	Hotel Stay List Account points Awards redeemed
Hotel	Data	Hotel Information: Summary Hotel rates Hotel availability Hotel photographs Hotel map, directions
Hotel List	Container	Contents: Icons Details
Reservation	Data	Create/View/Update Information: Name Hotel Arrival date Number of nights Room rate
Reservation List	Container	Contents: Icons Details
Finder	Data	Enter/Search for Information

Once a customer is found, either directly or by using the Finder search, the Customer folder is opened. A Customer folder would usually contain a Customer Profile, and Account Summary, and a Reservation List (see Figure 12.9).

Many of the other objects are data objects, and you must iteratively define and design the visual representations for each of these windows. Figure 12.10 shows a hand-drawn layout (actually drawn in Word using the graphics and text tools) of the Finder window.

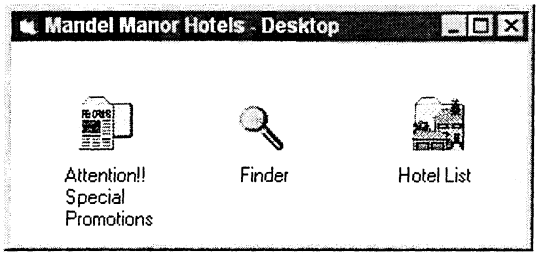

Figure 12.8 Mandel Manor Hotels desktop folder.

Step 5: Design Object and Window Menus

Once objects, their object type, and icons have been determined and designed, determine how users will interact with objects and windows using various types of menus. The following questions must be answered:

- What actions are appropriate for each object and view?
- What are the contents of object pop-up menus?
- Which windows need a menu bar?

Table 12.5 (Object and Actions list) and Figure 12.5 (Object Direct-Manipulation Matrix) from Step 3 are the starting points for Step 5. Now it's time to build on these charts and determine which actions specifically apply to objects when they are represented as icons and what actions are needed when objects are opened in view windows.

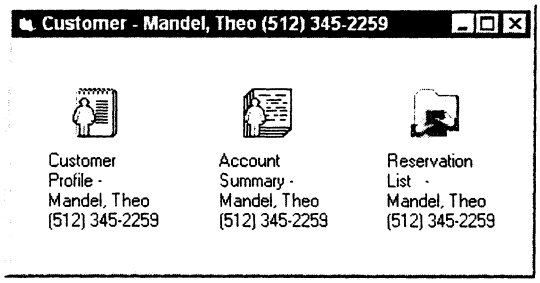

Figure 12.9 An opened Customer folder.

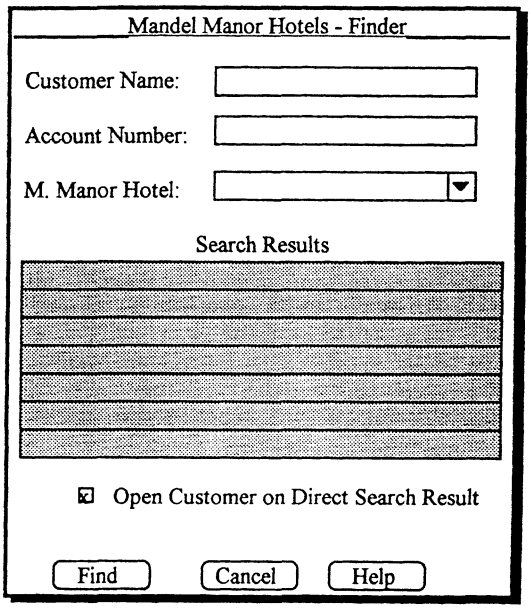

Figure 12.10 Visual representation of the Finder window.

First, determine what actions are appropriate for objects represented as icons on the screen. Figure 12.9 shows the Customer window opened, revealing three object icons. Both the Customer Profile and Account Summary are data objects and would have similar actions in a pop-up menu. Table 12.7 shows the menu choices in pop-up menus for these objects. The Reservation List is a container, and would have pop-up menu actions suitable for a container object.

Every window should be investigated to determine if a menu bar is needed. As software applications migrate toward objects and views, menu bars will give way to more direct interaction techniques like direct manipulation and pop-up menus. Single-windowed applications require complex menu bars to present all appropriate actions. As these applications are broken up into more object-oriented components, fewer actions are needed at the window level, and more actions can be done implicitly and with more dynamic pop-ups.

The Microsoft Windows guidelines offer no suggestions as to when menu bars are appropriate. The IBM (1992) guidelines state that a menu bar should be presented "when a window will provide more than six action choices or rou

TABLE 12.7 Object Pop-Up Menus

Object	Pop-Up Menu Choices	Purpose
Customer Profile,	Open	Open into data view
Account Summary	Move	Move to another location
	Create New	Create a new object
	Cut	Cut object to clipboard
	Copy	Copy object to clipboard
	Delete	Delete object
	Print	Print on printer
	Send	Send to another person
	Fax	Fax to another person
Reservation List	Open	Open into contents view
	Contents . . .	List contents without opening window
	Move	Move to another location
	Create New	Create a new object
	Cut	Cut object to clipboard
	Copy	Copy object to clipboard
	Print	Print on printer
	Send	Send to another person
	Fax	Fax to another person

ing choices." The main difference between an application-oriented menu bar and an object-oriented menu bar is the first menu bar choice, File. An object-oriented menu bar lists the object as the first menu bar choice, and the drop-down contains choices that affect the underlying object presented in that window. Figure 12.11 shows the object-oriented menu bar for the Reservation object's window.

Step 6: Refine Visual Designs

When first designing window layouts, it is much easier to use pencil and paper. As designs become more stable, use a prototyping tool or development tool to create icons and window layouts. Figure 12.12 shows a redesign of the hand-drawn layout for the Finder window (Figure 12.10). Figure 12.13 shows a first design for the Account Summary object view done using a development tool.

Based on all of the menu designs in Step 6, you will probably go back and revise Figure 12.5, the Object Direct-Manipulation Matrix.

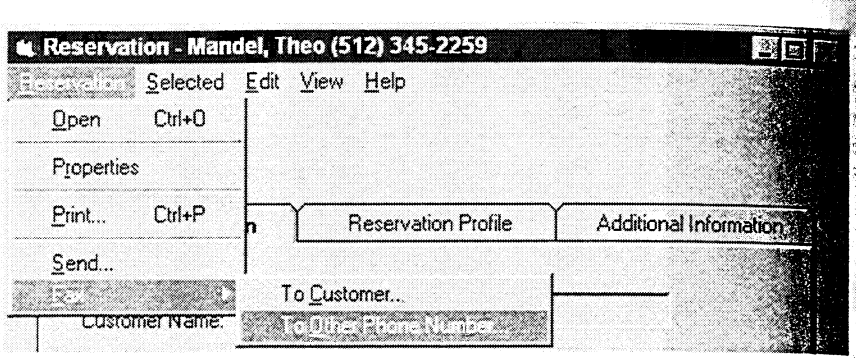

Figure 12.11 Object-oriented menu bar for Reservation object window.

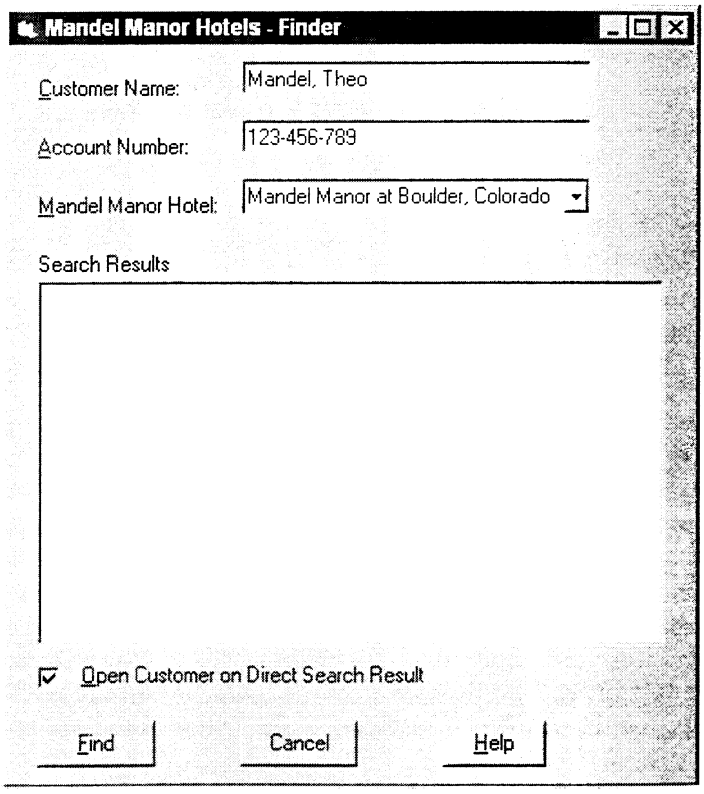

Figure 12.12 Redesign of the Finder window.

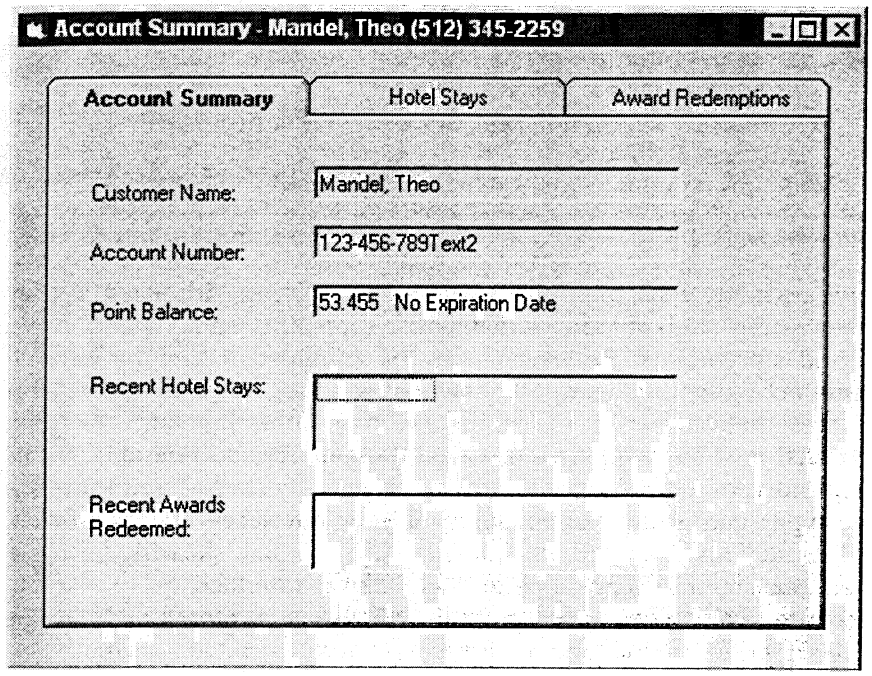

Figure 12.13 Initial online design of the Account Summary window.

Phase 3: Construct the User Interface

Prototyping as Interface Building

It is simply not possible to design a competitive user interface these days without powerful prototyping tools right from the start.

Edward Tufte (1994)

In your first pass through the iterative design process, you should *prototype,* rather than construct, the user interface. Prototyping is an extremely valuable way of building early designs and creating product demonstrations, and is necessary for early usability testing.

It is most important to remember that you must not be afraid to throw away a prototype. The purpose of prototyping is to *quickly* and *easily* visualize design alternatives and ideas, not to build code that must be used as a part of the product.

KEY IDEA! *Follow these three golden rules for using prototypes as part of the interface design process:*

- *Prototype early and iterate often.*
- *Prototype different design alternatives, not necessarily just one design.*
- *Be prepared to throw away prototype code.*

There are many ways to prototype interfaces, starting with pencil-and-p methods using chalkboards, whiteboards, flipcharts, sticky pads, overh foils, and storyboards. Prototypes may be animated, either by hand (w a person acting as the "computer" switching screens at user requests) o using presentation graphics programs (such as Lotus Freelance Graphic other tools) to show sequences of visual designs. A good reference for de oping prototypes is *The Art of Rapid Prototyping,* by Isensee and Rudd (19

Prototypes may show *visualizations* of your interface—the high-level co cepts—or they may show *functional* slices of a product, showing how spec tasks or transactions might be performed with the interface.

There are two ways to approach prototyping—use *prototyping tools* spec cally designed for the task, or use *development tools* that are normally used write product code. There are advantages and disadvantages to both type tools. Prototyping tools are easier for nonprogrammers to use and can produ prototypes and demos very quickly. However, they are often very limited their functionality and they usually use proprietary code that cannot be used your final product development. Development tools are rich in functiona but they are much more difficult for nonprogrammers to use.

KEY IDEA! *The code you build for a prototype using a software development tool can possibly be used in the final product, but be careful—this is a double-edged sword. If the prototype matches the final design, then some of the code may be salvageable, but if the prototype does not appropriately visualize the final design, the code should be thrown away. I have seen inappropriate or poorly designed prototypes become the core of the final product solely because decisions were made that the prototype code could not be thrown away—it had to be used! Remember, you must be willing to throw away prototype code. Don't use code from a poor design just because it is there!*

You should think about using a product prototype in place of (or in addition to) what is normally a lengthy written document, the *product functional specification* (PFS). If you have ever written a functional specification for a graphical user interface, you know it is very difficult to *write about* the presentation and interaction techniques involved in GUIs or OOUIs. It is much easier and much more effective to *show* a prototype, or demo, of the product interface style and behavior. Remember the old saying, "Don't tell me, show me!" You will find that a prototype can also serve as marketing and promotional aids to show managers, executives, and customers. A product demo is also a great giveaway item for business shows and promotional mailings.

KEY IDEA! *Even huge software projects follow the "prototype as specification" approach. The Windows 95 user interface design followed this approach. Sullivan (1996) noted, During the first few months of the project, the [written] spec had grown by leaps and bounds and reflected hundreds of person-hours of effort. However, due to the problems we found via user testing, the design documented in the spec was suddenly out of date. The team faced a major decision: spend weeks changing the spec to reflect the new ideas and lose valuable time for iterating or stop updating the spec and let the prototypes and code serve as a "living" spec. After some debate, the team decided to take the latter approach.*

Software prototyping is becoming a specialized area within software design and usability testing. Prototyping requires the skills and interests of both an interface designer and a software developer. Rudd and Isensee (1994) offer "The Official Prototyping Wallet Card" with twenty-two tips for a happier, healthier prototype (Table 12.8).

KEY IDEA! *Rather than developing real code as early as possible, use early design time to adequately prepare for development and create prototypes. By putting off development efforts until you have a better feel for the interface, you will actually avoid some of the rework you would do later if you had begun coding earlier. Just don't let an uninformed executive see your demo and declare, "That's not a demo, that's our product—ship it!"*

TABLE 12.8 The Official Prototyping Wallet Card, from Rudd and Isensee (1994)

1. Obtain upper-level development management support
2. Throw away your prototype
3. Make prototypes with high fidelity
4. Take every opportunity to show the prototype
5. Don't waste time prototyping add-ons
6. Start early
7. Make the prototype the functional specification
8. Disseminate to all technical leaders and developers
9. Develop idealistic instead of realistic prototypes
10. Use the best tools
11. Grab a piece of the action
12. The customer is king
13. Look outside the United States
14. Keep control of the prototype in your shop
15. Pay attention to aesthetics
16. Don't delegate the prototyping
17. Become multidisciplinary
18. Spread the word
19. Understand your corporate design guidelines
20. Research the key interface issues
21. Know the competition
22. Don't become a traditional (schedule-driven) developer

Phase 4: Validate the User Interface

One size fits all is usually ill-fitted to each person.

Mary Gorman (1996)

Usability testing is a key piece of the iterative development process. It's the way to get a product in the hands of many actual users to see if they can use it or break it! That's why I have devoted so much attention to this topic (Chapter 7). The goal of usability testing should be to measure user behavior, performance, and satisfaction. Many development processes address usability testing (if at all) near the end of the development cycle. However, this is much too late in the development cycle to make any changes based on usability test results. Even if changes are made, how can you tell if the revised product is usable unless it is tested again? Usability testing should be done early and often!

Schedule several stages of usability tests, beginning with early customer walkthroughs of initial designs. As pieces of the product and interface are built,

get them in front of users and test again. When the product is nearing completion and all of the pieces are coming together, then conduct final system tests. You shouldn't see any surprises if you have tested iteratively and incorporated user results and feedback. Table 12.9 shows how various usability activities should be conducted during the different product development stages. These are generic product development stages that parallel the iterative design phases I've discussed.

Scenarios developed in Phase 2 now come back to life in the validation phase. Did you build a product that was guided by the user scenarios you developed? How do you know? You don't, unless you validate the product against predefined usability goals and objectives using the scenarios as the vehicle of measuring user performance.

TABLE 12.9 Usability Activities Throughout the Development Process

Product Development Stage	Usability Activities
Concept Definition	◆ User requirements gathering ◆ Conceptual design definition
Concept Validation	◆ Conceptual design evaluations (paper/pencil, prototypes)
Design	◆ Evaluations of rapid prototypes ◆ Track and fix usability problems
Development	◆ Iterative tests of early designs (individual modules, key tasks) ◆ Iterative tests of final designs (integrated product, all tasks) ◆ Track and fix usability problems
Pilot Deployment	◆ Field observations of pilot users ◆ Pilot user feedback ◆ Pilot user questionnaires ◆ Track and fix usability problems
Product Implementation	◆ Field observations of product users ◆ Product user feedback ◆ Product user questionnaires ◆ Track and fix usability problems
Operations and Maintenance	◆ Long-term usability comparisons (1-month, 3-month, 6-month intervals) ◆ Test updates and changes ◆ Pilot test new product designs

KEY IDEA! *Bring users into your own usability test lab or contract your testing to a company specializing in usability testing. One word of advice from experience—don't let developers test their own products! It just doesn't work. Go to usability experts or human factors professionals who are not biased in any way about the product. Get help from outside your company. Designers end up designing the same product over and over again unless they get a fresh view from outside their environment.*

Developers should definitely observe their product being tested so they can see how users really work with their products—just don't let them design or conduct the test themselves. Developers must also be on hand to provide technical support during usability testing.

You may want to install test systems at customer locations to let users give informal feedback as they use the system. IBM developed numerous information kiosk systems that were used in the 1984 Olympics and the 1991 World Games in Barcelona, Spain. They are also developing the computer systems to be used in the 1996 Olympics in Atlanta. These systems are thoroughly tested during the entire design and development process. A kiosk was installed in the lobby of an IBM building where the product was being developed. A notebook was attached so that people could record their comments as they used the kiosk. These comments were read by the designers and were used to redesign the product and add additional features to the kiosk. The important point is to get everyone to try the product, not just selected participants. Assemble a "committee of dummies" and let them test drive the user interface!

Usability activities don't stop after a product is shipped or it is put into production. The product design team should be eagerly awaiting usability feedback from pilot participants and actual users once the product is in their hands in the real-world environment. Microsoft even asks users to send them feedback and a wish list for future versions of their products. The About Microsoft Works dialog box (Figure 12.14) asks users to send comments directly to Microsoft. It asks, "Help us make future versions even better by sending us your ideas and suggestions."

Who's Driving the Design—The System or the Interface?

Unfortunately, there are too many instances where the impact and effectiveness of the difference and power of the object-oriented paradigm are weakened when the technology is introduced into a real-world business environment.

Paul Schwork (1994)