Human Interface Design Principles

This section provides a theoretical base for the wealth of practical information on implementing the Aqua interface elements presented in the rest of this book.

You'll undoubtedly find that you can't design in accordance with all of the principles all the time. In those situations, you'll have to make decisions based on which principle or set of principles is most important in the context of the task you're solving. User testing is often an excellent way to decide between conflicting principles in a particular context.

In this section:

Metaphors

Take advantage of people's knowledge of the world by using metaphors to convey concepts and features of your application. Use metaphors that represent concrete, familiar ideas and make the metaphors obvious, so users can apply a set of expectations to the computer environment. For example, the Macintosh uses the metaphor of file folders for storing documents; people can organize their hard disks in a way that's analogous to the way they organize file cabinets.

Metaphors in the computer interface suggest a use for something, but that use doesn't necessarily define or limit the implementation of the metaphor. The Trash, for example, doesn't have to limit its contents to the number of items an actual wastebasket could contain. Try to strike a balance between the metaphor's suggested use and the computer's ability to support and extend the metaphor.

See-and-Point

People interact with the interface by pointing at onscreen objects with a device, typically a mouse. The Macintosh operating system works according to two fundamental paradigms, both of which assume that users can see what they're doing onscreen at all times and can point at what they see. The paradigms are based on a general form of user action: noun-then-verb.

In one paradigm, the user selects an object (the noun) and then chooses the action to be performed on the object (the verb). All actions available for a selected object are listed in the menus, so a user who is unsure of what to do next can scan through them. Users can choose an available action without having to remember a specific command.

In the second paradigm, the user directly manipulates an object (the noun) and performs an action (the verb) with it. A common example is dragging a document icon to a folder, for example. The user doesn't choose a menu command, but

it's clear what happens to an object when it's placed on another one. For this paradigm to work, the user must recognize what objects are for; the fact that the Trash looks like its real-world counterpart makes the interface easier to use.

Direct Manipulation

Direct manipulation allows people to feel that they are controlling the objects represented by the computer. According to this principle, an onscreen object should remain visible while a user performs an action on it, and the impact of the action should be immediately visible. For example, a user moves a file by dragging its icon from one location to another. With drag and drop, the most common example of direct manipulation, users can drag selected text directly into another document.

Support direct manipulation when users expect it. Avoid forcing users to use controls to manipulate data. For example, you should be able to send a facsimile by dragging a document's icon to a fax machine icon in the Dock, instead of having to open a utility program, choose a file, and click a Fax button.

User Control

Allow the user, not the computer, to initiate and control actions. Some applications attempt to take care of the user by offering only alternatives judged good for the user or that protect the user from having to make detailed decisions. This approach mistakenly puts the computer, not the user, in control.

The key is to create a balance between providing users with the capabilities they need to get their work done and helping them avoid dangerous irreversible actions. For a situation in which a user may destroy data accidentally, for example, you should always provide a warning and still allow the user to proceed if desired.

Feedback and Communication

Keep users informed about what's happening with your product. Provide feedback as they do tasks. When a user initiates an action, provide an indication that your application has received the user's input and is operating on it.

Users want to know that a command is being carried out or, if it can't be carried out, they want to know why not and what they can do instead. When used sparingly, animation is one of the best ways to show a user that a requested action is being carried out. When a user clicks an icon in the Dock, for example, the icon bounces to let the user know that the application or document is in the process of opening. In Mac OS X, the kernel environment detects when your application doesn't respond to events for 2 seconds and automatically displays a busy cursor.

For operations that don't execute immediately, use a progress indicator to provide useful information about how long the operation will take see "Progress Indicators". Users don't need to know precisely how many seconds an operation will take, but it helps to give an estimate. For example, the Mac OS uses statements such as "about a minute remains." It can also be helpful to communicate the total number of steps needed to complete a task—"Copying 30 of 850 files," for example.

Provide direct, simple feedback that people can understand. In error messages, for example, spell out exactly what situation caused the error ("There's not enough space on that disk to save the document") and possible actions the user can take to rectify it ("Try saving the document in another location"). For more information, see "Writing Good Alert Messages".

Consistency

Consistency in the interface allows people to transfer their knowledge and skills from one application to any other. Use the standard elements of the Aqua interface to ensure consistency within your application and to benefit from consistency across applications. Ask yourself the following questions when thinking about consistency in your product.

Is your product consistent:

- Within itself?
- With earlier versions of your product?
- With Mac OS standards? For example, does your application use the reserved and recommended keyboard

equivalents? (See "Keyboard Shortcuts".)
- In its use of metaphors?
- With people's expectations?

Matching everyone's expectations is the most difficult kind of consistency to achieve since your product is likely to be used by an audience with a wide range of expertise. You can address this problem by carefully weighing the consistency issues in the context of your target audience and their needs.

WYSIWYG (What You See Is What You Get)

In applications in which users can format data for printing, make sure there are no significant differences between what the user sees onscreen and what the user receives after printing. When the user makes changes to a document, display the results immediately; the user shouldn't have to wait for a printout or make mental calculations of how the document will look when printed. Use a print preview function if necessary.

WYSIWYG is not about only printing; all data experienced by users—movies, audio, and so on—should be faithfully represented in all media.

People should be able to find all the available features in your application. Don't hide features by having commands not visible in a menu. Menus present lists of commands so people can see their choices instead of having to remember command names.

Forgiveness

You can encourage people to explore your application by building in forgiveness—that is, making most actions easily reversible. People need to feel that they can try things without damaging the system; create safety nets, such as the Undo and the Revert to Saved commands, so people feel comfortable learning and using your product.

Always warn users before they initiate a task that will cause irreversible loss of data. If alerts appear frequently, however, it may mean that the product has some design flaws; when options are presented clearly and feedback is timely, using an application should be relatively error-free.

Perceived Stability

The Macintosh interface is designed to provide an understandable, familiar, and predictable environment.

To give users a visual sense of stability, the interface defines many consistent graphics elements, such as the menu bar, window controls, and so on. Users encounter a familiar environment in which they know how things behave and what to do with them.

To give users a conceptual sense of stability, the interface provides a clear, finite set of objects and a clear, finite set of actions to perform on those objects. For example, when a menu command doesn't apply to a selected object or to the object in its current state, it is shown dimmed (grayed out) rather than being omitted.

To help preserve the perception of stability, when a user sets up his or her onscreen environment in a certain layout, it should stay that way until the user changes it. Preserve user-modified settings such as window dimensions and locations.

Aesthetic Integrity

Aesthetic integrity means that information is well organized and consistent with principles of visual design. Your product should look pleasant on the screen even when viewed for a long time.

Keep graphics simple, and use them only when they truly enhance usability. Don't overload the user with icons or put dozens of buttons in windows or dialogs. Don't use arbitrary symbols to represent concepts; they may confuse or distract users.

Match a graphic element with users' expectations of its behavior. Don't change the meaning or behavior of standard items.

For example, always use checkboxes for multiple choices; don't use them sometimes for exclusive choices.

Modelessness

As much as possible, allow people to do whatever they want at all times. Avoid using modes that lock the user into one operation and don't allow the user to work on anything else until that operation is completed.

Most acceptable uses of modes fall into one of the following categories:

- Short-term modes in which the user must constantly do something to maintain the mode. Examples are holding down the mouse button to scroll text or holding down the Shift key to extend a text selection.
- Alert modes, in which the user must rectify an unusual situation before proceeding. Keep these modes to a minimum. See "Types of Dialogs and When to Use Them" for more information.

Other modes are acceptable if they do one of the following:

- They emulate a familiar real-life situation that is itself modal. For example, choosing different tools in a graphics application resembles the real-life choice of physical drawing tools.
- They change only the attributes of something, not its behavior. The boldface and underline modes of text entry are examples.
- They block most other normal operation of the system to emphasize the modality. An example is a dialog that makes all menu commands unavailable except Cut, Copy, and Paste.

If an application uses modes, there must be a clear visual indicator of the current mode, and it should be very easy for users to get in and out of the mode. For example, in many graphics applications, the pointer can look like a pencil, a cross, a paintbrush, or an eraser, depending on the function (the mode) the user selects.

Hide TOC

< Previous Page Next Page >

Last updated: 2003-10-18