Designing Intuitive Icons And Toolbars

Intuitive icons and well-designed toolbars provide an ease of use that belies the difficulty of their design.

Most of us do not give much thought to the icons and toolbars used in our favorite applications. Generally, icons are self-explanatory, and we use them automatically. Even application developers seldom spend much time considering which icons to employ for various functions. Developers often use a stock library of icons, and most automatically assign a pair of scissors, for example, to represent the cut function. However, good icon selection and design requires careful and sophisticated thought.

Developing universally recognizable icons involves many design considerations. Two such considerations are symbolic representation and color.

Consider the accepted icon for mail in the United States: the common mailbox. Few of us actually have a mailbox posted outside our front gate, yet we understand what the icon represents. However, in countries such as Hong Kong, England, or Germany where U.S. mailboxes have never been commonplace, users may not understand the icon's meaning.

Like symbols, colors carry different meanings in different countries. Moreover, although most users prefer color icons, icons that depend on color can cause difficulties for color-blind individuals or when used on gray-scale monitors.

Successful icon design is further complicated because iconic symbols are not consistent among windowing systems. Even icons that
represent fundamental concepts and objects such as directory files and folders vary across different windowing systems. For example, an experienced Macintosh user will find creating a simple folder on a PC windowing system an arduous task.

Furthermore, icon-rich, seemingly user-friendly interfaces often pose considerable challenges for experienced users. Data-entry clerks, for example, are burdened by applications that require them to verify visually that they have chosen the appropriate icons.

Icon design and toolbar layout have more than an ancillary effect on users: they are the interface between the user and the application and must be designed to provide simple, obvious, inoffensive, and universal meanings. A successful icon is easy to recognize and understand, and it is unambiguous.

**Icon Design**

Most application developers have only two potential ways to design an icon. One option is to use a graphical user interface (GUI) design package, such as Microsoft’s Image Editor or Microsoft’s IconWorks, and either choose from predefined Microsoft or Visual Basic icons or design icons from scratch with the drawing facilities provided by those packages. A second option is to utilize the GUI capabilities that are part of a larger application-development system.

Once icons are designed, they can be integrated into an application one of two ways. One way is for programmers to refer to icons by name and hard-code the names into the application. With this method, users cannot easily customize or change their icons. Any changes to such graphical elements must be accomplished through the source code of the application. Alternatively, programmers can keep the GUI separate from the application logic so that GUI elements become modularized resources and therefore accessible to users for modification.

The ability to customize icons and other GUI elements is critical when designing a distributed, widely used application. For example, since the U.S. mailbox icon will not work well in Hong Kong, users there can design a different mail icon and easily replace the original icon without touching the source code of the application.

The icon itself has significant size limitations. Icons built into MS Windows and Motif are typically limited to a square area of 32 pixels per side. Each icon border is 2 pixels wide; consequently, the remaining working area for the image is reduced to 28 pixels per side.

The semantics associated with icons are also important. Icons should typically represent a single function. In some cases, additional choices can be displayed for the user when the icon is selected. Consider the situation when a user cannot determine an icon’s purpose because the representation is not obvious. Getting information about what functions icons represent is often difficult. Techniques such as Windows’ word balloons are effective enough for most users, but word balloons cannot be implemented uniformly in all GUIs.

When choosing an icon, some basic rules should be followed to ensure that the icon is useful. Choose icons that will be familiar across all cultures. For example, arrows and other control buttons that resemble VCR controls will be easily recognized by most users.

They may not be exotic, but they will not cause confusion. Unfortunately, this approach leads to a limited development set. One way to circumvent this issue is to allow customization of icon bitmaps.

An icon should be designed to match its function. When crafting an icon, first develop several icon designs that appropriately illustrate the icon’s function. Then choose the design that best suits the icon’s purpose. Marketing and advertising may want to make the icon cute or unique to the company, but keep in mind that the icon must be simple and intuitive. For applications already employed or when one application is integrated with another, stick with icon symbols that are already familiar to users, even if the symbols are not ideal.

**Color And Depth**

Two important icon design elements are color and depth. To achieve the latter, a technique called antialiasing may be used to smooth an icon’s jagged edges. To use antialiasing, developers design edges that gradually lighten from black to white to avoid false registration and to develop sharper images. Curves, for example, can be softened by using tones of gray to smooth rough edges. Most GUIs model shadows on the lower right of the icon so that light appears to be shining over the user’s left
shoulder. (See Figure 1.) Changing this design can elicit an unsettled response from the user, although few users will recognize the source of their disturbance.

When choosing icon color, use colors that will map reasonably well into gray scales. Choose a basic palette of colors. Most GUIs contain a 16-color palette. Try to work with no more than 16 colors because exceeding that number can cause problems with different video drivers. Some developers create both color and gray-scale versions of their icons. However, it is much easier to program and maintain a single icon.

Choose a dominant color for an icon, bearing in mind human perception of color. Blue is difficult to recognize for those with vision problems and as one grows older. Red connotes danger or quick action. Yellow is often associated with caution. Across cultures, certain colors are associated with religious or societal events. Generally, the best colors for icons are greens, dark blues, and similar dominant colors.

Through the phenomenon of chromostereopsis, color also may be used to establish hierarchies and apparent dimension in and among icons. Chromostereopsis refers to how the retina focuses on color; it affects how objects are perceived. For example, when presented side by side, a red object will appear to be closer than a blue object. When stacking colors in an icon, always put the darkest shade (for example, blue) in back, the medium shade (green) in the middle, and the brightest shade (red) on top. This method will create perspective.

Icon background-color choice is important as well, although gray backgrounds now seem to be the de facto standard. With a black background, pastel-colored text is most readable. When a light background is used, text can be a more saturated color. Background color should be the same color as the GUI background and should remain consistent across all icons unless overwhelming reasons force a change. Avoid color dithering (mixing light shades with dark shades to create a textured medium shade) because dithered colors translate poorly across resolution changes.

Other Icon Aspects

When determining fonts used for icon lettering, readability is paramount. Some argue in favor of eliminating icon lettering completely because lettering renders the application ineffective for those who cannot read the language used. If text is utilized, fonts should be simple and legible.

After an icon is developed, test the icon by showing it to other users. If they can quickly and easily determine its purpose, the icon is appropriate. If the icon elicits confusion or is perceived as ambiguous, it must be redesigned.

Toolbars

Inevitably, icons are assembled into toolbars. Toolbars were originally designed to act as accelerators for frequently used menu options. In most GUIs, the toolbar is positioned horizontally across the top of the window. Some applications let the user move the toolbar to a vertical position on the left or right side of the window. An effective alternative that is gaining popularity is the floating toolbar. Floating toolbars can be reduced from long, horizontal boxes to smaller rectangles that the user can move about the window freely. This provides a cleaner-looking, more spacious window.

Over time, the use of toolbars has increased, and currently, numerous applications overuse toolbars. To develop effective toolbars, designers must follow two rules:

- Only the most frequently used menu options should be put on toolbar icons.
Only button-press or toggle-switch operations should be placed on toolbars. For instance, menu options that save a file or toggle boldface are good candidates for toolbar icons.

Avoid creating long toolbars cluttered with icons. Toolbars should extend only the length of the typical window. Avoid toolbar scrolling and using too many toolbars on one window. Because of toolbar overuse, users often spend too much time hunting for the icons they need. Typically, one toolbar should suffice; if more are needed, prioritize menu options and place lesser-used choices in pull-down or pull-up menus.

Choose icon order on the toolbar carefully. Consistency is important. Icons that appear in more than one toolbar should be placed in the same position on each toolbar. The most commonly used icons should appear on the left side of the toolbar; those that are used least often should appear on the right. Display functional groups (such as copy, cut, and paste) together.

Finally, the color of the toolbar should blend with the rest of the window and should be muted so it does not distract the user. Gray is a good color for toolbars because it neutralizes the toolbar background and complements icons that also have gray backgrounds. Adjacent icons should contain background colors that blend so they do not jar the overall user interface.

**Menus And Toolbars**

It is important to duplicate all iconic functions as menu options. Consequently, in a well-designed user interface, the total functionality available in its menus will be a superset of the functionality available on its toolbars. Design menus thoughtfully. Pull-down and pull-up menus may have subsidiary menus expanding from them, but limit the number of expansions to three. If more than three expanding menus are required, create a new menu option. After selecting a heading, limit the number of corresponding choices to ten.

Control the number of pull-down and pull-up menus. Many applications must balance the number of menu titles against the number of items on each menu, but a busy window menubar can be daunting to users. As with icon toolbars, menus should be arranged from left to right in order of priority or functionality.

**Think And Test**

While they may seem obvious, these design techniques are rarely employed by developers today. Creating a functional user interface requires much thought. A reasonable amount of testing prior to making the user interface widely available will help smooth out many design issues. Although particularly important to commercial or multinational corporate applications, thoughtful user-interface design is a critical success factor for designing good applications.

Lil Mohan is director of product marketing management at Neuron Data (Palo Alto, CA). He is an expert in object-oriented technology and visual programming. Neuron Data produces portable, object-oriented application-development software for building enterprise-wide client-server applications. He can be reached at lmohan@neuronda.com.

John Byrne is the director of technologies at Comptron (Rutherford, NJ) and has 17 years’ experience developing software applications and designing graphical user interfaces. Comptron is a supplier of international financial, workflow, and COLD software products. He can be reached at jcb@ctronsoft.com.