

Principals of Good GUI Design

By James Hobart, President, Classic System Solutions

October 1995



Graphical user interfaces (GUIs) have become the user interface of choice. Yet despite the GUI's popularity, surprisingly few programs exhibit good interface design. Moreover, finding information explaining what constitutes a good and intuitive interface is exceedingly difficult. In this article, I describe the basic rules for all good interfaces -- the cardinal do's and don'ts.

However, before starting in on what constitutes good design, I need to explain the causes of bad design. This way, if you are tempted to deviate from the tried and true, you'll know where the wrong path leads -- and, I hope, get back to good design.

Design Mistake 1: Forgetting the User

Developers often design for what they know, not what the user knows. This age-old problem occurs in many other areas of software development, such as testing, documentation, and the like. It is even more pernicious in the interface because it immediately makes the user feel incapable of using the product. Avoid this error diligently.

Design Mistake 2: Failing to Give the User Control

The GUI designer's predilection for control is evident in applications that continually attempt to control user navigation by graying and blackening menu items or controls within an application. Controlling the user is completely contradictory to event-driven design in which the user, rather than the software, dictates what events occur. As a developer, if you are spending a lot of time dynamically graying and blackening controls, you need to re-examine your design approach and realize that you may be controlling the user, who may not want to be controlled. As business changes at a faster pace, flexibility in user interfaces will become a key enabler for change. Allowing the user to access the application in ways that you never dreamed can be scary, but satisfying, for you as a developer and empowering for the user.

Design Mistake 3: Providing Too Many Features at the Top Level

Examine a VCR built in 1985, and then examine one built in 1995. You will see a startling difference in the interface of the two models. The model built in 1985 will have an abundance of buttons readily available on the faceplate of the unit, many of which will remain a mystery since the manual was lost years ago. The 1995 model will have only a few buttons for the key features that people use: play, fast forward, reverse, stop, and eject. The newer model will probably have even more features than the model built a decade before, yet the features will be cleverly tucked away behind a drop-down panel or sliding door, accessible when needed but not staring the user in the face.

Likewise, you should ensure that features used frequently in an application are readily available. Avoid the temptation to put everything on the first screen or load the toolbar with rarely used buttons. Do the extra analysis necessary to find out which features can go behind the panel instead of on the faceplate.

GUI Successes

Now, let's discuss some GUI successes. Successful GUIs share many common characteristics. Most importantly, good GUIs are more intuitive than their character-based counterparts. One way to achieve this is to use real-world metaphors whenever possible. For example, an application I recently examined used bitmaps of the Visa and MasterCard logos on buttons that identified how a customer was going to pay. This graphical representation was immediately intuitive to users and helped them learn the application faster.

Another important characteristic of good GUIs is speed, or more specifically, responsiveness. Many speed issues are handled via the design of the GUI, not the hardware. Depending on the type of application, speed can be the make-or-break factor in determining an application's acceptance in the user community. For example, if your application is oriented toward online transaction processing (OLTP), slow performance will quickly result in users wanting to abandon the system.

You can give a GUI the appearance of speed in several ways. Avoid repainting the screen unless it is absolutely necessary. Another method is to have all of the field validations occur on a whole-screen basis, instead of on a field-by-field basis. Also, depending upon the skills of the user, it may be possible to design features into a GUI that give the power user the capability to enter each field of each data record rapidly. Such features include mnemonics, accelerator keys, and toolbar buttons with meaningful icons, all of which allow the speed user to control the GUI and rate of data entry.

Do's And Don'ts

Every good developer should have the goal of designing the best GUIs possible. But how does a developer make this goal a reality? By following sound, proven GUI design principles such as those listed in the following sections.

Like any good professional, YOU need some rules for repeatable successful designs. We have used the principles offered here for work with our own customers and have taught more than 20,000 GUI-design students nationally and internationally. These principles should help you as well.

Design Principle 1: Understand People

Applications must reflect the perspectives and behaviors of their users. To understand users fully, developers must first understand people because we all share common characteristics. People learn more easily by recognition than by recall. Always attempt to provide a list of data values from which the user can select, rather than having the user key in values from memory. The average person can recall about 2,000 to 3,000 words, yet can recognize more than 50,000 words.

Design Principle 2: Be Careful of Different Perspectives

Many designers unwittingly fall into the perspective trap when it comes to icon design or the overall behavior of the application. I recently saw an icon designed to signify "Rolled Up" totals for an accounting system. To show this function, the designer put a lot of artistic effort into creating an icon resembling a cinnamon roll. Unfortunately, the users of the system had no idea what metaphor the icon was supposed to represent even though it was perfectly intuitive from the designer's perspective. A reserved icons table containing standard approved icons, such as the one shown in Figure 1, will help eliminate these problems.

Figure 1 - Reserved Icons

Meaning and Behavior	Used to Identify an Application	Used to Identify a Function	Reserved Word Text Label
Information message	No	Yes (identifies an information message box)	None
Warning message	No	Yes (identifies a warning message box)	None
Question message	No	Yes (identifies a question message box)	None
Error message	No	Yes (identifies an error message box)	None

Design Principle 3: Design for Clarity

GUI applications often are not clear to end users. One effective way to increase the clarity of an application is to develop and use a list of reserved words. A common complaint among users is that certain terms are not clear or consistent. I often see developers engaging in spirited debates over the appropriate term for a button or menu item, only to see this same debate occurring in an adjacent building with a different set of developers. When the application is released, one screen may say "Item," while the next screen says "Product," and a third says "Merchandise", when all three terms denote the same thing. This lack of consistency ultimately leads to confusion and frustration for users.

Figure 2 gives an example of a list of reserved words. An application development group might complete and expand the table with additional reserved words.

Figure 2 - List of Reserved Words

Text	Meaning And Behavior	Appears on Button	Appears on Menu	Mnemonic Keystrokes	Shortcut Keystrokes
OK	Accept the data entered or acknowledge the information presented and remove the window	Yes	No	None	Return or Enter
Cancel	Do not accept the data entered and remove the window	Yes	No	None	Esc
Close	Close the current task and continue working with the application; close the view of the data	Yes	Yes	Alt+C	None
Exit	Quit the application	No	Yes	Alt+X	Alt+F4
Help	Invoke the application's help facility	Yes	Yes	Alt+H	F1
Save	Save the data entered and stay in the current window	Yes	Yes	Alt+S	Shift+F12
Save As	Save the data with a new name	No	Yes	Alt+A	F12
Undo	Undo the latest action	No	Yes	Alt+U	Ctrl+Z

Cut	Cut the highlighted information	No	Yes	Alt+T	Ctrl+X
Copy	Copy highlighted information	No	Yes	Alt+C	Ctrl+C
Paste	Paste the copied or cut information at the insertion point	No	Yes	Alt+P	Ctrl+V

Design Principle 4: Design for Consistency

Good GUIs use consistent behavior throughout the application and build upon a user's prior knowledge of other successful applications. When writing software for business applications, provide the user with as many consistent behaviors as possible. For example, both the Embassy Suites and Courtyard Marriot hotel chains are growing rapidly due to their popularity among business travelers who know that they will be provided with a familiar room and a consistent set of amenities. The bottom line is that business travelers are not looking for a new and exciting experience in each new city. Business users of your software have similar needs. Each new and exciting experience you provide in the software can become an anxiety-inducing experience or an expensive call to your help desk.

Design Principle 5: Provide Visual Feedback

If you've ever found yourself mindlessly staring at the hourglass on your terminal while waiting for an operation to finish, you know the frustration of poor visual feedback. Your users will greatly appreciate knowing how much longer a given operation will take before they can enjoy the fruits of their patience. As a general rule, most users like to have a message dialog box with a progress indicator displayed when operations are going to take longer than seven to ten seconds. This number is highly variable based on the type of user and overall characteristics of the application.

Design Principle 6: Be Careful With Audible Feedback

Last week, I had the opportunity to ride in elevators in which a pleasant voice informed riders which floor they were on. The building was fairly new, and at first, employees thought the voice was cute. After six months of traveling floor to floor, employees are sure to ignore the voice and see it as more of an annoyance than a help. The same thing can happen with your GUIs, except the annoying sounds are not contained within the walls of an elevator, but instead are available to everyone within earshot of the worker's cubicle. Put sound on a few hundred workstations, and a real cacophony emerges in the open-air cubicle environment. However, audible feedback can be useful in cases where you need to warn the user of an impending serious problem, such as one in which proceeding further could cause loss of data or software. Allow users to disable audio feedback, except in cases when an error must be addressed.

Design Principle 7: Keep Text Clear

Developers often try to make textual feedback clear by adding a lot of words. However, they ultimately make the message less clear. Concise wording of text labels, user error messages, and one-line help messages is challenging. Textual feedback can be handled most effectively by assigning these tasks to experienced technical writers.

Design Principle 8: Provide Traceable Paths

If your users ever say something akin to, "I don't know how I got to this window, and now that I'm here, I don't know how to get out," then you have not provided a traceable (or, in this case, retraceable) path. Providing a traceable path is harder than it sounds. It starts with an intuitive menu structure from which to launch your specific features.

You must also identify areas where you can flatten the menu structure and avoid more than two levels of cascading menus. Providing a descriptive title bar within each dialog box greatly helps to remind users what menu items or buttons were pressed to bring them to the window now in focus.

Design Principle 9: Provide Keyboard Support

Keyboards are a common fixture on users' desktops and provide an efficient means to enter text and data. With the introduction of GUI applications, we often assume users will embrace a mouse as the primary interactive device. Using a mouse can become time-consuming and inefficient for the touch typist or frequent users of an application.

Keyboard accelerators can provide an efficient way for users to access specific menu items or controls in a window. The accelerators used should be easy to access and limited to one or two keys (such as F3 or Ctrl-P). Keyboards have limitations in the GUI world, such as when trying to implement direct-manipulation tasks like drag and drop, pointing, and re-sizing.

In contrast, you will always find a smaller set of users who are not touch typists and hence embrace the mouse as a point-and-click nirvana. The result is that you need to provide complete and equal keyboard and mouse support for all menu and window operations.

Design Principle 10: Watch the Presentation Model

A critical aspect that ties all these facets of the interface together is the interface's look and feel. The look and feel must be consistent. On the basis of users' experiences with one screen or one dialog box, they should have some sense of how to interact with the next screen or control.

Searching the interface model for good design and continuity is very important. The model should involve careful decisions, such as whether the application has a single or multiple document interface. The model also will validate how users perform the main tasks within the application.

Identifying the appropriate presentation for the application greatly facilitates the subsequent windows being developed since they will have a common framework in which to reside. On the other hand, if you do not define the presentation model early in the design of your GUI, late changes to the look and feel of the application will be much more costly and time-consuming because nearly every window may be affected.

Design Principle 11: Use Modal vs. Modeless Dialogs Appropriately

When we need input from the user, we often use a modal dialog box. Using modal dialogs has long been shunned by many developers as too constraining on the user. However, modal dialogs do have many uses in complex applications since most people only work on one window at a time. Try to use modal dialogs when a finite task exists. For tasks with no fixed duration, modeless dialogs are normally the preferable choice with a major caveat: Try to keep the user working in no more than three modeless windows at any one time. Go beyond this magical number and the support-desk phones will start ringing, as users spend their time managing the open windows rather than concentrating on their tasks. Use the table in Figure 3 to determine the appropriate use of dialog boxes and windows.

Figure 3 - When to Use Dialog Boxes or Windows

Type	Description	Use	Example
Modal	Dialog box	Presentation of a finite task	File Open dialog box, Save As dialog box
Modeless	Dialog box	Presentation of an ongoing task	Search dialog box, History List dialog box, Task List dialog box
Application window	Window frame with document (child) windows contained within	Presentation of multiple instances of an object, Comparison of data within two or more windows	Word processor, Spreadsheet
Document window	Modeless dialog box or document window contained within and managed by the application window	Presentation of multiple parts of an application	Multiple views of data (sheets)
Secondary window	Primary window of a secondary application	Presentation of another application called from the parent window	Help window in an application

Design Principle 12: Use Controls Correctly

Controls are the visual elements that let the user interact with the application. GUI designers are faced with an unending array of controls from which to choose. Each new control brings with it expected behaviors and characteristics. Choosing the appropriate control for each user task results in higher productivity, lower error rates, and higher overall user satisfaction. Use the table in Figure 4 as a guideline for control usage in your screens.

Figure 4 - Guidelines for Using Controls

Control	Number of Choices in the Domain Shown	Type of Control
Menu bar	Maximum of 10 items	Static action
Pull-down menu	Maximum of 12 items	Static action
Cascading menu	Maximum of 5 items, 1 cascade deep	Static action
Pop-up menu	Maximum of 10 items	Static action
Push button	1 for each button, maximum of 6 per dialog box	Static action
Check box	1 for each box, maximum of 10 to 12 per group	Static set/select value
Radio button	1 for each button, maximum of 6 per group box	Static set/select value
List box	Maximum of 50 in the list, display 8 to 10 rows	Dynamic set/select value
Drop-down list box	Display 1 selection in the control at a time, up to 20 in a drop-down box	Dynamic set/select single value
Combination list box	Display 1 selection in the control at a time in standard format, up to 20 in a drop-down box	Dynamic set/select single value; add a value to the list
Spin button	Maximum of 10 values	Static set/select value
Slider	Dependent on the data displayed	Static set/select value in range

Finally, try to keep the basic behavior and placement of these controls consistent throughout your application. As soon as you change the behavior of these basic controls, your user will feel lost. Make changes thoughtfully and apply the changes consistently.

Applying Design Principles

Understanding the principles behind good GUI design and applying them to your applications can be a challenge. Let's examine an application to see how these principles can result in an improved interface.

Part 1: Exploring a GUI in Need of Redesign

The interface in Figure 5 is used by an ambulance-dispatching company to maintain customer data, provide billing information, and dispatch ambulances. The application was a port from a character-based system and contains several design errors that affect the user's performance with this mission-critical application. Keep in mind that GUI ease of use and clarity is especially important in a critical application such as this where the rapid handling of a request can make the difference between life and death. Here is what is wrong with this screen:

Figure 5

The screenshot shows a 'Patient Account' window with the following fields and controls:

- Company:** 002
- Invoice:** AK-3356
- S.S.N.:** 044-52-8995
- Trip Date:** 6/12/92
- Operator:** T. Michaels
- Date:** 6/15/92
- Last Name:** Carlson
- First:** Melissa
- Medicare:** A 74493021
- Medi-Cal:** [Empty]
- (A) Patient's Address:** 71194 El Camino Real, San Jose, CA 92344. Home: (714) 688-5433. Work: [Empty]
- (B) Birthday:** 3/5/34
- Sex:** F
- Sig/File:** [Empty]
- POE:** [Empty]
- (C) Charge To:** SHP Senior Plan, P.O. Box 3382, San Diego, CA 95445
- (B) Physician:** Marshall, 4550 Stetson Dr., (409) 448-3322
- From Location:** 445 Elm St., Sonoma
- To Location:** General Hospital, Sonoma, Admit: C
- (D) Diagnosis:** 8054 Syncope Episode, 9000 Head injury
- Billing:** 000 Private Charge 345.90, Paid 290.00, Adjust -50.00, Balance -5.90
- (E) Employer:** [Empty]
- (R) Reason... Transport:** 1
- Ambulance:** 5
- Update** button
- Caller:** Family (405) 934-5596
- Clear** button
- Buttons on the right:** Notes, Payments, Insurance, Letters, Billing, Trans
- Bottom left:** Rcvd 03:45, Disp 03:45, Arrive 03:53, Comp 04:13
- Bottom center:** R.T., (U) Auth: 9333, Inst Billing ID: [Empty]

- **Too many functions at the top level.** The users requested that the new application provide all of the information at their fingertips. This results in the screen being used for both customer maintenance and ambulance dispatching. If you enter extensive customer information and then press the Update button, the record is updated. However, if you enter minimal customer information, such as social security number, diagnosis, from-location, and to-location, and then press the Trans button, an ambulance will be dispatched. Updating and dispatching functions need to be in separate dialog boxes.
- **Too many buttons.** The buttons along the right should be on the application's parent window, possibly in a toolbar, but not on this child window.
- **Poor navigational assistance.** GUI controls should be positioned according to frequency of use. The most important field should be in the upper left; the least important field should be in the lower right. It's hard to imagine how the company and invoice number could be the most important fields when dispatching an ambulance.
- **Inappropriate use of controls.** The designer chose to use text labels rather than group boxes to identify which groups of data would be placed in the boxes. This many group boxes with text labels in these positions makes the screen appear convoluted and makes it difficult to distinguish the data from the labels. Also, the editable fields should be identified with a box around them, so that it is intuitively obvious which fields can be changed.
- **Lack of symmetry.** Just lining up fields, group boxes, and buttons will make this GUI much easier to use. Our brains like order, not chaos.

Part 2: Looking at an Improved Interface

Figures 6 and 7 show a much improved interface for this same application:
Figure 6

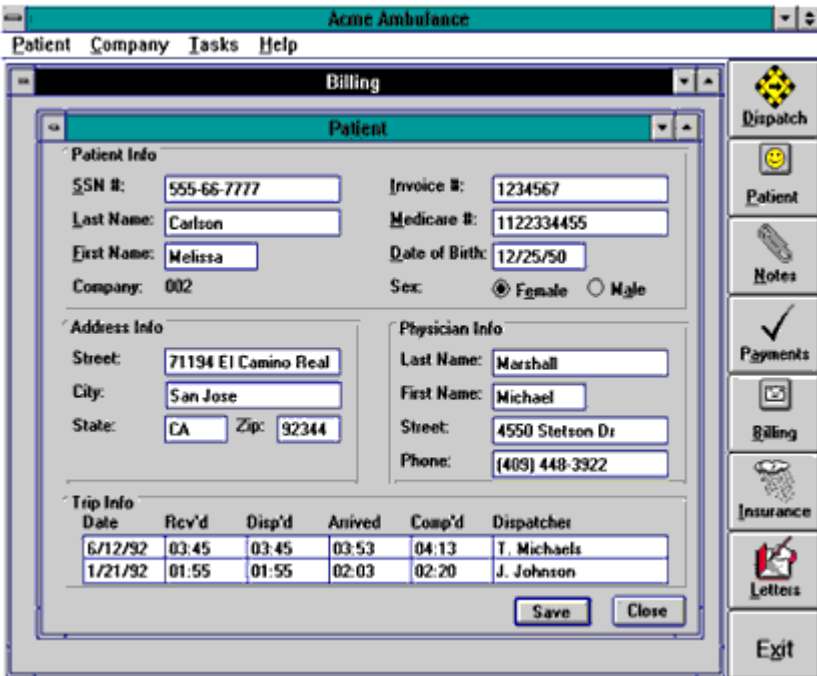
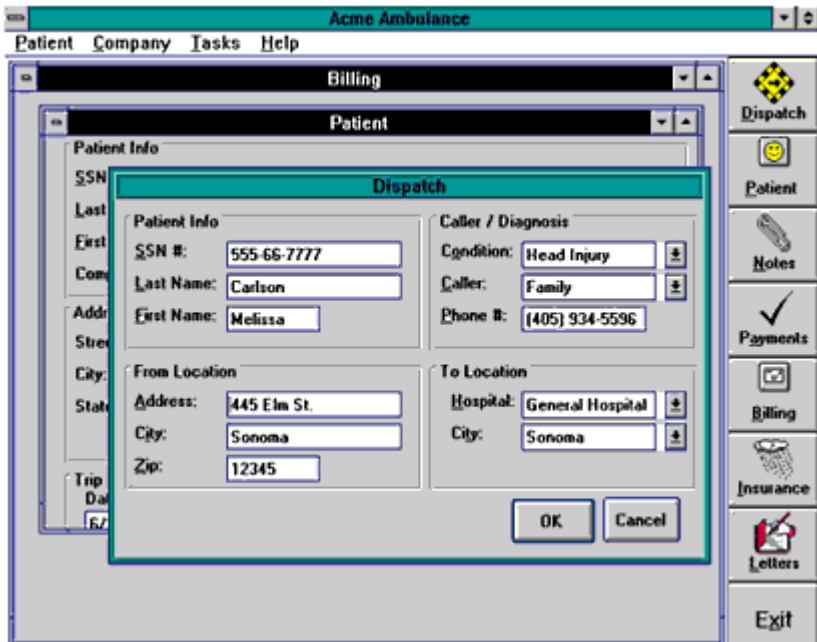


Figure 7



- **Order out of chaos.** This application should contain several child windows for the different tasks that a user might perform. These tasks can be accessed easily through the Tasks menu or by pushing a button on the vertical toolbar. The Dispatch button invokes a modal dialog box instead of a modeless child window. That way, you can require the user to acknowledge the completion of the dispatching task. If it were a modeless window, the user might overlay it without ever dispatching the ambulance.
- **Reordering input fields.** The confusing order of fields has been more logically structured based on importance and frequency of use.
- **Improved controls.** The revised interface features a consistent use of data-entry fields. Any field in which a user can enter data is surrounded by a border. Group boxes are used to group related fields together or to illustrate a domain.

These changes, suggested by the principles that we have previously discussed, make for a clean and more intuitive interface.

Implementing Effective Standards

Once you implement some good design practices into your GUI applications, how do you ensure others in your organization will do the same? The most cost-effective way to ensure consistency among your GUI applications is to implement GUI standards that are easy to use, clear, and concise. We've all experienced the "standards" manual that is energetically distributed to coworkers only to be placed immediately on the developer's shelf next to other unread manuals. To ensure that your standards do not meet this same fate, provide them in an online hypertext format. Divide your standards into rules -- which must be followed or the developer will have some explaining to do -- and recommendations. Developers like to know what is mandatory and where they have discretion.

Conclusion

Designing good GUIs is a critical skill for application developers in the 1990s, regardless of the GUI platform for which they are designing. Good GUI designs don't happen naturally. They require that the developer learn and apply some basic principles, including making the design something the user will enjoy working with every day. They also require that the developer get as much experience as possible in working on and being exposed to good GUI designs.

Remember that if you apply the principles and get some experience in good GUI design, your users will have an easier time getting their jobs accomplished using the GUIs you produce for them.

About the author:

James Hobart is an internationally recognized user interface design consultant based in California, USA. He specializes in the design and development of large-scale, high-volume client/server and web applications. He is an expert in GUI design for transaction processing systems and strategies for migration to thin-client graphical user interfaces. He can be reached at jimh@classicsys.com

Find more articles on Usability at www.classicsys.com