

# Principles of Software Driven User Interface Design for Business and Industrial Applications

*Robert Japenga*

Most of these principles were gleaned from 30 years of making software driven user interfaces for business and industry. Although some of these principles apply to other interfaces targeted to other groups (games, entertainment, children), this is my area of expertise. In addition, these principles concentrate on the utility of these interfaces as opposed to the artistic or emotional impact.

## General Principles of User Interface Design

1. **Know Your Users** - Ask yourself the following questions before you start any user interface design:
  - What assumptions will my users make about what this program does?
  - What user interface paradigms are familiar to them?
  - How will they use the system? For example, if the interface is used by sales and technical personnel, the data may need to be presented in alternative ways.
  - Will your users ever become experts? If so, you will want to provide for different levels of users. What was great when I first used a system, is now an absolute pain in the neck.
  - What are the errors that my users will make?
  - Where will my users go for help?
  - Will the users read a manual?<sup>i</sup>

A Use Case Analysis is helpful at the very beginning of a design.<sup>ii</sup>

2. **Apply as much natural mapping in your design as possible** - Most of the time, natural mappings are obvious and easily implemented. An "up arrow" next to a number will clue in most users that pressing the "up arrow" will cause the number to go up. Sometimes, what seems natural can be confusing. For example, when a list of three items has one item highlighted with a separate color, there is a natural mapping indicating that color implies selected. But what if the list has only two items? Which color indicates the selected item? This was particularly annoying when I only had a black and white screen and the designer used reverse video.

Engineers tend to like things in neat rows with everything symmetrical. But if the device you are controlling is laid out in a particular order, why not lay out the controls in the same order. Light switches in a house are laid out by engineers. How much easier would it be if the light switches were laid out in a mapping of the lights and outlets they control?

Sometimes, natural mappings are not obvious and labeling is necessary. But natural mapping will trump labeling every time.

3. **Don't let the users take an action that they cannot do** - How many times have you clicked on an item only to have the interface tell you how stupid you are for not filling in the correct data. Some are even so nice as to tell you what data you didn't fill in. Why did the interface let me move forward if it knew I didn't have all the information? This is just poor user interface design. Web interfaces are particularly bad in this area.

Or how about those interfaces that tell you not to click the button twice or else it will charge your credit card twice. Thanks for telling me - but is your design so brain-dead that it cannot detect this? You can make a distinction between your user interface and everyone else's by following this simply stated principle.

4. **Avoid sounds in user interfaces unless absolutely necessary** - One customer wanted us to provide a key click on a membrane keyboard which has no audible click or tactile feel when pressed. This is *the* one kind of exception that I agree with. Another example of this is the shutter sound on a digital camera. In both cases, the user has a particular "sound paradigm" they are expecting and it makes the addition of sound important. However, the user should know that they are doing something because of what they see - not from what they hear. Sounds can be annoying. Sounds most often get turned off.
5. **Beware of taking away control of things that the users once had control of** - Woe is me on this one. I have gotten in so much trouble when I have tried to correct something by automating it. It is an admirable goal, but here are the guidelines:
  - Make absolutely sure that your change works and covers all the real cases that your customer used. If you normally spend an hour testing this feature, spend ten.
  - Make sure that your user knows that this is being changed.
  - Let the users know what is being done under the hood. Normally this is not necessary. If you previously allowed the users to set four parameters but have simplified the process to have them choose just one parameter, at least initially, they are going to want to know this in terms of how it worked before. If your users have a history concerning how something was to be set up, they are going to want to know how it is being automatically set up.
6. **Make Liberal Use of Constraints** - I love to use cables that can only be plugged in one way. This is a good design constraint in the cable's user interface. I am somewhat adverse to cables that can be plugged in either way and still work. The first gives me a confidence that I am doing it right. The second relies on my memory that "Oh yeah - it doesn't matter how this cable plugs in."

For example, in designing user interfaces, don't let me click the down button at the end of a list. Eliminate the down button or change it in some way. Why allow the

user to travel down dead end streets. Better yet, why should the list have a constraint at all? Why not make it wrap to the top?

Also, when a user setting on one control provides a constraint for another control, indicate the constraint locally. Try to have the affected control on the same screen. Whenever possible, try to make the connections between settings and constraints obvious.

7. **The Importance of Efficiency of Operation** - User designs should take into account the number of screens and keystrokes/mouse clicks it takes to perform an operation. A good example of this is Yahoo's email address book. For years, when you added an email address, it took you back to the main address book screen where you had to click "Add a Contact." Now, after you have entered the data, it gives you the option to "Save" and to "Save and Add Another." A simple change that makes adding many names much faster.
8. **Avoid Race Conditions between the User and the Software**- Everyone who has used the Web knows of this situation:

You enter a web site like [www.nytimes.com](http://www.nytimes.com)

It takes a long time to load.

You go back to Word and continue writing your sister a letter

When the web site comes up, the focus is changed from Word to the NY Times.

Your typing gets trashed / ignored / etc.

Errors and alarms can pre-empt user input in the same way. Remember that the user may be entering data when the Alarm or Error message comes up. They may not be looking at the screen while typing. I have had situations where an alarm or error message pops up, but my typing causes a hot key to dismiss the alarm. In some cases this may just be annoying. But for some of our software, it can be downright dangerous.

9. **Restrict Type-ahead to the Current Screen Only** - Have you ever worked on an interface that allowed you to type ahead of what you see on the screen and it get you in trouble? One of the first tragic software errors in history involved the Therac-25 X-Ray machine<sup>iii</sup>. This software killed several people because the machine was taking input data and operating on it before it displayed what it was doing. In some situations it is a cute trick - but it is confusing and should be avoided at all costs.
10. **Design for Error** - Assume that your users are going to make every possible error when using your interface. Never assume "they will never do this." Through the use of constraints, keep them from as many errors as possible. When errors are detected, provided them as much information as possible about:
  - What action caused the problem
  - What remedy can be taken

If possible, allow them to correct it at the error window. The "Retry" error window in many PC applications is a great example of allowing them to correct this on the spot without having to go through the entire process again.

Make it possible for them to un-do an erroneous action. How many times have you entered a ton of data on a web page and made one small error and requiring you to enter all of the data again? Make as many operations reversible as possible. There is no need to ask "Are you sure" if the interface can un-do the operation. This makes for a more streamlined interface.

When it is absolutely necessary to make an action irreversible, make sure that the user really wants to do this. "Are you sure" is usually not enough. What is going to be irreversible? What will I have to do to get back to this state? Will this action harm me or my device?

11. **The importance of feedback** - When the users change something, they must get feedback within 100 ms that at least something is happening. While things are happening, the users should get some kind of realistic feedback that progress is being made at least every second. If a cancel button is displayed during a long process, make it actually respond when you click it!
12. **Minimize the amount your user has to remember from one action to another** - We once implemented a user interface specified for a two line display. To turn a specific control on or off, the users selected the control and then was presented with two choices - on or off. Unfortunately, this meant the users had to remember what control they were changing once it was selected. At every step in the process of working with the users, don't require them to remember what they did last (or even worse - several actions ago).
13. **The Principle of Information Hiding** - As object oriented programmers, we understand this principle as it relates to software design. Designers of user interfaces also must remember this principle. On certain older TV sets, the vertical and horizontal gains sometimes were not only in the back of the set, but recessed so as to require a special technician's screw driver--a good design. Early VCRs did practice some good user interface design principles by placing some controls behind a plastic door on the front panel. In your Use Case analysis, carefully decide who needs what and when, then design the interface accordingly.
14. **Know what your user is expecting from visual clues** - Since the advent of web pages, we have come to think of text that is underlined and colored as a link to another page with more information. If you were designing a web page, underlining and colorizing a section of text without providing a link would confuse your users. Use cases help you to address some of these issues. Not all of your users may be familiar with this, so find an additional way to provide emphasis.

15. **The Value of Comparisons** - When large amounts of data are required to set up something, a compare function is sometimes useful. For engineers, think of your BIOS settings on your PC. Imagine how valuable it would be to compare new settings to previously saved settings. Your users would probably send you fan mail and marriage proposals.
16. **The importance of the conceptual model**<sup>iv</sup> Whenever we use a product, we develop a conceptual model in our head of how the device works. I find it fascinating to ask non-technical people what they think is going on behind the scenes. As a user interface designer, you should try doing this. This is very important information, especially if your product is going to be used by non-technical folk (i.e. real people). The conceptual model is simply "How the users perceive that your device works"

When my car is cold and I want the passenger compartment to heat up fast, I turn the temperature knob furthest in the clockwise direction. When it becomes comfortable, I turn it down to mix cold air with the heat from the engine. Recently, in my son-in-law's car, the first thing I did on a cold night was to turn the identically marked control all the way up. He chided me saying that I had just set the temperature for 90° F. My idea about how the control works (my conceptual model of how the system worked) was based on 40 years of driving cars with the same control interface. The new car had the same interface but I needed a totally different conceptual model.

There are always trade-offs. In this design, keeping the interface the same helps old geezers like me to use it right away (even though with over-shoot, the temperature was hard to control). But with the display for the temperature setting 6 inches from the control knob, I may have never learned the new paradigm. I may have just complained that you just can't set the temperature in that new car.

The refrigerator temperature control is another example. Look at the controls for your freezer and fresh food compartments. What is your conceptual model of how that works based on the controls?<sup>v</sup>

**Lesson:** If you are changing the meaning of a control from a legacy system, make a radical change to the look and feel of the control. Second, always try to understand how your user is going to think the knob works.

17. **Make Sure the Users Know their options on each display** - Every presentation to the user should make it clear what actions are possible at any moment. Have you ever gotten to a screen and thought: "Now what?" Some early Windows programs came up with File, Edit, Window and Help and then a blank screen. Most have now incorporated this principle with the use of wizards initial prompts. PowerPoint is a case in point. Early versions came up with the blank screen. Newer versions start with - "Okay - what do you want to do - open an existing presentation or start a new one."

18. **A Good Checklist for any User Interface** - After you have got a good user interface design under your belt, ask yourself:

**How easily can one:**

- Determine the function of the device?
- Tell what actions are possible?
- Tell what state the device is in?
- Perform each action?
- Verify that the action is taken?<sup>vi</sup>

## Principles of Display in User Interfaces

1. **Carefully control the amount of information** - There is a delicate balance between displaying too much information, which clutters the display, and displaying too little information, which confuses or annoys the user. The ideal is to have a control always visible for each important function. Practically, this may have to be limited to every major function.

This can create problems. Take the average audio mixing board. Then try to control it during an ear drum altering concert in the dark. Hiding other controls while controlling others is one solution. Visually distinctive icons work better to reduce clutter. However, many software programs have icons that are virtually indistinguishable.

This creates the challenging problem of creating icons in an email program for "Reply" and "Reply All". But it is possible to do better than what is out there.

2. **Use Lower Case when Possible** - The brain reads UPPER CASE much more slowly than lower case. Unless the display creates very difficult to interpret lower case letters, use lower case naturally as in written text. Sometimes upper case is appropriate. For example, ASCII is more quickly recognizable than ascii.

## Good Principles of Data Entry in User Interfaces

1. Use the Palm OS model of "Saving" data entered. The user never saves and never loses data because each program remembers its state when closed and reopened. Web interfaces are typically 1980's interfaces. How often have you entered data on the web and then have to re-enter it after something goes wrong. With that said, the Palm model does not allow for abandoning edits. This should also be provided.
2. Use the "Brief"<sup>vii</sup> model of multiple un-do where needed (as modified by some Brief emulators which allows you to go back beyond the previous "Save"). Some GUI experts had dismissed un-do because they find holes in many poor implementations

of "Un-do" (Like Microsoft Word). This "multiple un-do" model used in "Brief" completely disarms arguments by GUI experts against this feature.

Un-do should also tailored to the user interface where necessary. This means that sometimes un-do is not simply a reversal of the keystrokes - but a reversal of actions (which are a group of keystrokes). Microsoft Paint does this. This should be carefully considered at design time. Category un-dos are also under-implemented and should be considered where appropriate. Thus the "Brief" un-do can be refined to become even more powerful only undoing certain categories of actions (formatting or file manipulations as opposed to text editing).

Some applications even go so far as to display a list of previous actions, allowing users to easily go back to a desired state without having to press undo over and over.

3. Each user interface must define the method of data entry - Calculator style or PC style. Calculator style fills data from the right - shifting the previously entered data to the left. PC style fills from the left - continuously concatenating new characters to the end of the stream. Understanding the user should drive this decision.
4. It is better to limit the number of characters and / or the range of the input than to tell the users that they are stupid with an out of range error. One of our testers was testing a client's Java application, which allowed 128 characters to be entered as the possible range when the valid range was from 0 to 100. If the input shouldn't take alphabetic entry, why let the user enter it?
5. Avoid Insert and Overtyping modes unless you are sure that your users are PC savvy. They are never necessary for a field that only takes a small number of characters. My wife still doesn't understand these two modes. Many respected User Interface Designer don't like modes at all. When modes are needed, use signals that make the behavior of the current mode obvious.
6. The entry field should be seeded with a previous value (for that field!) when possible. Seeded data can be accepted, modified one character at a time with the backspace key or clear hard key, or completely replaced with any new data entry. The following two sequences (using the Calculator data entry model) depict a seeded value being edited or being replaced.

__ABCD	Seeded Opening sequence
____X	User enters an X
____XY	User enters a Y
____XYZ	User enters a Z

__ABCD	Seeded Opening sequence
____ABC	User clears an entry
____ABCX	User enters an X

7. Ideally your design should be so intuitive that both the beginner and expert use the same interface. But in the real world, we must balance the interface between easy to learn to use for the beginner and powerful to use for person who lives there. Sometimes having beginner and expert levels never allows the beginner to transition. A beginner level is often a wizard which bears little resemblance to the expert level. Most of the time the interface for the wizard is completely separate from the expert level which makes the transition difficult. Strive for consistency of presentation and input between the different levels.

Make it easy for the user to go in both directions up and down the level tree.

---

<sup>i</sup> No!

<sup>ii</sup> "Use cases capture user requirements for a system by describing how a system will be used and to what ends in a way that the end user can understand"

[www.umsl.edu/~sauter/analysis/488\\_f01\\_papers/Prossman/UseCase.htm](http://www.umsl.edu/~sauter/analysis/488_f01_papers/Prossman/UseCase.htm)

There is a lot of information about use cases which I would heartily recommend.

<sup>iii</sup> There are many good articles about this tragedy and well worth every user interface designers read. Google "Therac-25" and you will get number of excellent articles. One of the best is

[www.computer.org/computer/homepage/misc/Leveson/index.htm](http://www.computer.org/computer/homepage/misc/Leveson/index.htm).

My first exposure to this problem came from Brooks, Frederick P., No Silver Bullet: Essence and Accidents of Software Engineering, *Computer*, Vol. 20, No. 4 (April 1987) pp. 10-19. This is a classic article and a must read for all software developers. It can be found at:

[www.computer.org/computer/homepage/misc/Brooks/index.htm](http://www.computer.org/computer/homepage/misc/Brooks/index.htm)

<sup>iv</sup> For more on this idea, see the [Design of Everyday Things](#) by Donald Norman. This book contains a wealth of ideas that are very applicable to designing user interfaces.

<sup>v</sup> [Ibid](#), page 14

<sup>vi</sup> [Ibid](#), page 53

<sup>vii</sup> The Brief model can be basically described as "It always un-does exactly what you want un-done." Now go and implement that!