Principles of Interaction Design (B. Tognazzini, 2003)

Anticipation

Applications should attempt to anticipate the user's wants and needs. Do not expect users to search for or gather information or evoke necessary tools. Bring to the user all the information and tools needed for each step of the process.

Autonomy

- The computer, the interface, and the task environment all "belong" to the user, but user-autonomy doesn't mean we abandon rules.

  Give users some breathing room. Users learn quickly and gain a fast sense of mastery when they are placed "in charge." Paradoxically, however, people do not feel free in the absence of all boundaries (Yallum, 1980). A little child will cry equally when held too tight or left to wander in a large and empty warehouse. Adults, too, feel most comfortable in an environment that is neither confining nor infinite, an environment explorable, but not hazardous.

- Use status mechanisms to keep users aware and informed.

  No autonomy can exist in the absence of control, and control cannot be exerted in the absence of sufficient information. Status mechanisms are vital to supplying the information necessary for workers to respond appropriately to changing conditions. As a simple example, workers, failing status information, will tend to maintain heightened pressure on themselves during slow periods, until the moment the work actually runs out. This will stress and fatigue them unnecessarily, so that when the next rush occurs, they may be lacking the physical and mental reserves to handle it.

- Keep status information up to date and within easy view

  Users should not have to seek out status information. Rather, they should be able to glance at their work environment and be able to gather at least a first approximation of state and workload. Status information can be quite subtle: the inbox icon could be switched to show an empty, somewhat full, or stuffed state. This, however, should not be overdone. The Macintosh, for years, showed an icon of a trashcan of imminent danger of explosion if a single document was placed therein. Users quickly formed the habit of emptying the trashcan as soon as the first document hit. This not only turned a single-step operation into a two-step operation (drag to the trash, then empty the trash), it negated the entire power of the trashcan, namely, undo.

  As another positive example, a search field icon can change color and appearance to indicate that the search is in progress or has been completed with too many matches, too few matches, or just enough. (Like any element of the interface, just color is not enough; 10% of males show some indication of color blindness. Even a higher percentage may have temporary alterations in perception of blue under varying conditions.)

Color Blindness

- Any time you use color to convey information in the interface, you should also use clear, secondary cues to convey the information to those who won't be experiencing any color coding today.

Most people have color displays nowadays, but they are not universal. In addition, approximately 10% of human males, along with a rare sprinkling of females, have some form of color blindness.

The cones in the eye are the source of color vision. We have cones separately sensitive to red, green, and blue. If the red ones are not functioning that is called protanopia. If the green are not functioning, that is called deuteranopia. Absence of blue, extremely rare, is called tritanopia.

Protanopia and deuteranopia are the most popular forms of color blindness, collectively called red/green blindness. (There are, in fact, significant differences in their effects, but those differences have no real effect on interaction design.) While tritanopia is far more rare, it nonetheless rules out dependence on yellow-blue differentiation without secondary cues.

Secondary cues can consist of anything from the subtlety of gray scale differentiation to having a different graphic or different text label associated with each color presented.

Disability

Consider how physical disabilities, deafness, and blindness affect user interactions. How is information gathered? Communicated? Modified? What are appropriate input/output devices and formats? Can information be accurately translated from one mode or format to another?

Consistency

The following principles, taken together, offer the interaction designer tremendous latitude in the evolution of a product without seriously disrupting those areas of consistency most important to the user.

- Levels of consistency: The importance of maintaining strict consistency varies. The following list is ordered from those interface elements demanding the most faithful consistency effort to those demanding the least. Paradoxically, many people assume that the order of items one through five should be exactly the reverse, leading to applications that look alike, but act completely different in unpredictable ways:

    1. Interpretation of user behavior, e. g., shortcut keys maintain their meanings.
    2. Invisible structures.

3. Small visible structures.
4. The overall "look" of a single application or service--splash screens, design elements.
5. A suite of products.
6. In-house consistency.
7. Platform-consistency.

"Invisible structures" refers to such invisible objects as Microsoft Word's clever little right border that has all kinds of magical properties, if you ever discover it is there. It may or may not appear in your version of Word. And if it doesn't, you'll never know for sure that it isn't really there, on account of it's invisible. Which is exactly what is wrong with invisible objects and why consistency is so important. Other objects are, strictly speaking, visible, but do not appear to be controls, so users, left to their own devices, might never discover their manipulability. The secret, if you absolutely insist on one, should be crisp and clean, for example, "you can click and drag the edges of current Macintosh windows to size them," not, "You can click and drag various things sometimes, but not other things other times."

"Small visible structures" refers to icons, size boxes, scroll arrows, etc. The appearance of such objects needs to be strictly controlled if people are not to spend half their time trying to figure out how to scroll or how to print. Location is only just slightly less important than appearance. Where it makes sense to standardize location, do so.

- Inconsistency: It is just important to be visually inconsistent when things must act differently as it is to be visually consistent when things act the same.

  Avoid uniformity. Make objects consistent with their behavior. Make objects that act differently look different.

- The most important consistency is consistency with user expectations.

  The only way to ascertain user expectations is to do user testing. No amount of study and debate will substitute.

Defaults

- Defaults should be easy to "blow away:" Fields containing defaults should come up selected, so users can replace the default contents with new material quickly and easily.
- Defaults should be "intelligent" and responsive.

- Do not use the word "default" in an application or service. Replace with "Standard," "Use Customary Settings," "Restore Initial Settings," or some other more specific terms describing what will actually happen.

Efficiency of the User

- Look at the user's productivity, not the computer's.

  People cost a lot more money than machines, and while it might appear that increasing machine productivity must result in increasing human productivity, the opposite is often true. In judging the efficiency of a system, look beyond just the efficiency of the machine.

  For example, which of the following takes less time? Heating water in a microwave for one minute and ten seconds or heating it for one minute and eleven seconds?

  From the standpoint of the microwave, one minute and ten seconds is the obviously correct answer. From the standpoint of the user of the microwave, one minute and eleven seconds is faster. Why? Because in the first case, the user must press the one key twice, then visually locate the zero key, move the finger into place over it, and press it once. In the second case, the user just presses the same key–the one key–three times. It typically takes more than one second to acquire the zero key. Hence, the water is heated faster when it is "cooked" longer.

  Other factors beyond speed make the 111 solution more efficient. Seeking out a different key not only takes time, it requires a fairly high level of cognitive processing. While the processing is underway, the main task the user was involved with–cooking their meal–must be set aside. The longer it is set aside, the longer it will take to reacquire it.

  Additionally, the user who adopts the expedient of using repeating digits for microwave cooking faces fewer decisions. They soon abandon figuring out, for example, whether bacon should be cooked for two minutes and ten seconds or two minutes and twenty-three seconds. They do a fast estimate and, given the variability of water content and bacon thickness, end up with as likely a successful result with a lot less dickering up front, again increasing human efficiency.

- Keep the user occupied.

  Since, typically, the highest expense in a business is labor cost. Any time the user must wait for the system to respond before they can proceed, money is being lost.

- To maximize the efficiency of a business or other organization you must maximize everyone’s efficiency, not just the efficiency of a single group.

  Large organizations tend to be compartmentalized, with each group looking out for its own interests, sometimes to the detriment of the organization as a whole. Information resource departments often fall into the trap of creating or adopting systems that result in increased efficiency and lowered costs for the information resources department, but only at the cost of lowered productivity for the company as a whole.

  For example, one large California corporation used floppy disks as the medium for collecting benefit enrollment information. At the beginning of open enrollment, each employee would receive a disk with the enrollment applications on which he or she would insert into their computer and run. After asking for the employee’s name, address, phone number, department name, etc., the employee would be permitted to step through

all the various benefits, ultimately returning the disk which now contained all their answers and decisions. The IR department then sucked the data off each disk and entered it into their system, all automatically. The IR department saved a great deal of money over the old system, where they had to key in the employee's decisions from a paper form.

What was the problem? Instead of the IR department bearing the burden of keying in the employees' decisions, each and every employee now bore the burden of typing in his or her name, address, phone number, department name, etc. The system was just as inefficient as before, but now the cost was borne by all departments, rather than having it concentrated in the IR department's budget.

- The great efficiency breakthroughs in software are to be found in the fundamental architecture of the system, not in the surface design of the interface.

  This simple truth is why it is so important for everyone involved in a software project to appreciate the importance of making user productivity goal one and to understand the vital difference between building an efficient system and empowering an efficient user. This truth is also key to the need for close and constant cooperation, communication, and conspiracy between engineers and human interface designers if this goal is to be achieved.

- Write help messages tightly and make them responsive to the problem: good writing pays off big in comprehension and efficiency.
- Menu and button labels should have the key word(s) first.

  Example from a fictitious word processor:

  Wrong:

  ☐ Insert page break
  ☐ Add Footnote
  ☐ Update Table of Contents

  Right:

  Insert:
  ☐ Page break
  ☐ Footnote
  ☐ Table of contents

  Here, the first example, with its leading words, is actually more informative and more accurate: one does not "insert" a footnote if it is to be placed after all the other footnotes. And one does not insert a table of contents if there is already a table of contents there. Instead, one updates it. Still, the second example will prove much more efficient in time-trials. Why? Because the extra information the first example offers does not outweigh the advantage of being able to scan only the first word in each menu item to find the specific menu item you are after.

Explorable Interfaces

- Give users well-marked roads and landmarks, then let them shift into four-wheel drive.

  Mimic the safety, smoothness, and consistency of the natural landscape. Don’t trap users into a single path through a service, but do offer them a line of least resistance. This lets the new user and the user who just wants to get the job done in the quickest way possible and "no-brainer" way through, while still enabling those who want to explore and play what-if a means to wander farther afield.

- Sometimes, however, you have to provide deep ruts.

  The closer you get to the naive end of the experience curve, the more you have to rein in your users. A single-use application for accomplishing an unknown task requires a far more directive interface than a habitual-use interface for experts.

- Offer users stable perceptual cues for a sense of "home."

  Stable visual elements not only enable people to navigate fast, they act as dependable landmarks, giving people a sense of "home."

- Make Actions reversible

  People explore in ways beyond navigation. Sometimes they want to find out what would happen if they carried out some potentially dangerous action. Sometimes they don’t want to find out, but they do anyway by accident.

  By making actions reversible, users can both explore and can "get sloppy" with their work.

- Always allow "Undo."

  The unavoidable result of not supporting undo is that you must then support a bunch of dialogs that say the equivalent of, "Are you really, really sure?" Needless to say, this slows people down.

  In the absence of such dialogs, people slow down even further. A study a few years back showed that people in a hazardous environment make no more mistakes than people in a supportive and more visually obvious environment, but they worked a lot slower and a lot more carefully to avoid making errors.

- Always allow a way out.

  Users should never feel trapped. They should have a clear path out.

- However, make it easier to stay in.

  Early software tended to make it difficult to leave. With the advent of the web, we've seen the advent of software that makes it difficult to stay. Web browsers still festoon

their windows with objects and options that have nothing to do with our applications and services running within. Our task can become akin to designing a word process which, oh, by the way, will be using Photoshop's menu bar. Having 49 options on the screen that lead directly to destruction of the user's work, along with one or two that just might help is not an explorable interface, it is the interface from hell. If you are working with complex transactions using a standard web browser, turn off the menu bar and all of the other irrelevant options, then supply our own landmarks and options.

Fitts' Law

- The time to acquire a target is a function of the distance to and size of the target.

  While at first glance, this law might seem patently obvious, it is one of the most ignored principles in design. Fitts' law (properly, but rarely, spelled "Fitts' Law") dictates the Macintosh pull-down menu acquisition should be approximately five times faster than Windows menu acquisition, and this is proven out.

  Fitts' law dictates that the windows task bar will constantly and unnecessarily get in people's way, and this is proven out. Fitts' law indicates that the most quickly accessed targets on any computer display are the four corners of the screen, because of their pinning action, and yet, for years, they seemed to be avoided at all costs by designers.

  Use large objects for important functions (Big buttons are faster).

  Use the pinning actions of the sides, bottom, top, and corners of your display: A single-row toolbar with tool icons that "bleed" into the edges of the display will be many times faster than a double row of icons with a carefully-applied one-pixel non-clickable edge between the tools and the side of the display.

Human Interface Objects

Human-interface objects are not necessarily the same as objects found in object-oriented systems. Our objects include folders, documents, and the trashcan. They appear within the user's environment and may or may not map directly to an object-oriented object. In fact, many early gui's were built entirely in non-object-oriented environments.

- Human-interface objects can be seen, heard, touched, or otherwise perceived.
- Human interface objects that can be seen are quite familiar in graphic user interfaces. Objects that play to another sense such as hearing or touch are less familiar. Good work has been done in developing auditory icons (Gaver).
- Human-interface objects have a standard way of interacting.
- Human-interface objects have standard resulting behaviors.
- Human-interface objects should be understandable, self-consistent, and stable.

Latency Reduction

- Wherever possible, use multi-threading to push latency into the background.

  Latency can often be hidden from users through multi-tasking techniques, letting them continue with their work while transmission and computation take place in the background.

- Reduce the user's experience of latency.

  - Acknowledge all button clicks by visual or aural feedback within 50 milliseconds.
  - Display an hourglass for any action that will take from 1/2 to 2 seconds.
  - Animate the hourglass so they know the system hasn't died.
  - Display a message indicating the potential length of the wait for any action that will take longer than 2 seconds.
  - Communicate the actual length through an animated progress indicator.
  - Offer engaging text messages to users informed and entertained while they are waiting for long processes, such as server saves, to be completed.
  - Make the client system beep and give a large visual indication upon return from lengthy (>10 seconds) processes, so that users know when to return to using the system.
  - Trap multiple clicks of the same button or object. Because the Internet is slow, people tend to press the same button repeatedly, causing things to be even slower.

- Make it faster

  Eliminate any element of the application that is not helping. Be ruthless.

Learnability

Ideally, products would have no learning curve: users would walk up to them for the very first time and achieve instant mastery. In practice, all applications and services, no matter how simple, will display a learning curve.

- Limit the Trade-Offs.

  Usability and learnability are not mutually exclusive. First, decide which is the most important; then attack both with vigor. Ease of learning automatically coming at the expense of ease of use is a myth.

Metaphors, Use of

- Choose metaphors well, metaphors that will enable users to instantly grasp the finest details of the conceptual model.

Good metaphors are stories, creating visible pictures in the mind.

- Bring metaphors alive by appealing to people's perceptions–sight, sound, touch, and kinesthesia–as well as triggering their memories.

  Metaphors usually evoke the familiar, but often add a new twist. For example, Windows 95 has an object called a briefcase. Like a real-world briefcase, its purpose is to help make electronic documents more portable. It does so, however, not by acting as a transport mechanism, but as a synchronizer: Documents in the desktop briefcase and the briefcase held on portable media are updated automatically when the portable media is inserted in the machine.

Protect Users' Work

- Ensure that users never lose their work as a result of error on their part, the vagaries of Internet transmission, or any other reason other than the completely unavoidable, such as sudden loss of power to the client computer.

  (Even here, it has become completely inexcusable that today's computers and operating systems do not support and encourage continuous-save. That, coupled with a small amount of power-protected memory could eliminate the embarrassment of $5000 machines offering the reliability of 10-cent toys.)

Readability

- Text that must be read should have high contrast. Favor black text on white or pale yellow backgrounds. Avoid gray backgrounds.
- Use font sizes that are large enough to be readable on standard monitors. Favor particularly large characters for the actual data you intend to display, as opposed to labels and instructions. For example, the label, "Last Name," can afford to be somewhat small. Habitual users will learn that that two-word gray blob says "Last Name." Even new users, based on the context of the form on which it appears, will have a pretty good guess that it says "Last Name." The actual last name entered/displayed, however, must be clearly readable. This becomes even more important for numbers. Human languages are highly redundant, enabling people to "heal" garbled messages. Numbers, however, unless they follow a very strict protocol, have no redundancy, so people need the ability to examine and comprehend every single character.
- Pay particular attention to the needs of older people. Presbyopia, the condition of hardened, less flexible lenses, coupled with reduced light transmission into the eye, affects most people over age 45. Do not trust your young eyes to make size and contrast decisions.

Track State

- Because many of our browser-based products exist in a stateless environment, we have the responsibility to track state as needed.

  We may need to know:

  - Whether this is the first time the user has been in the system
  - Where the user is
  - Where the user is going
  - Where the user has been during this session
  - Where the user was when they left off in the last session

  and myriad other details.

  In addition to simply knowing where they've been, we can also make good use of what they've done.

- State information should be held in a cookie on the client machine during a session with a transaction service, then stored on the server when they log off.

  Users should be able to log off at work, go home, and take up exactly where they left off.

  A private service for doctors, Physicians On Line, does an excellent job with this. Doctors can be 95% of the way through a complex transaction, log off, log in again six weeks later from another part of the world, and the service will ask them if they want to be taken right back to where they were.

Visible Navigation

- Avoid invisible navigation.

  Most users cannot and will not build elaborate mental maps and will become lost or tired if expected to do so.

  The World Wide Web, for all its pretty screens and fancy buttons, is, in effect, an invisible navigation space. True, you can always see the specific page you are on, but you cannot see anything of the vast space between pages. Once users reach our applications, we must take care to reduce navigation to a minimum and make that navigation that is left clear and natural. Present the illusion that users are always in the same place, with the work brought to them. This not only eliminates the need for maps and other navigational aids, it offers users a greater sense of mastery and autonomy.

  As with the inherent statelessness of the web (see Track State, above), our job is not to accept blindly what the architects have given us, but to add the layers of capability and protection that users want and need. That the web's navigation is inherently invisible is a challenge, not an inevitability.