IDI – Course Notes | P

IDI – Principles of Interaction and UI Design

1. About the document

This is a set of excerpts from the original work by Bruce Tognazzini, former Apple worker, where he led the development of the UI guidelines for 14 years. You can check the original document here:

http://asktog.com/atc/principles-of-interaction-design/

Here you have some excerpted parts to reduce the amount of reading.

2. Introduction

Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and can settle down to do their work. Effective interfaces do not concern the user with the inner workings of the system. Work is carefully and continuously saved, with full option for the user to undo any activity at any time. Effective applications and services perform a maximum of work, while requiring a minimum of information from users.

Because an application or service appears on the web or mobile device, the principles do not change. If anything, applying these principles—all these principles—becomes even more important.

3. Principles

As stated previously this document will highlight only a subset of the principles shown in the original document.

3.1 Aesthetics. Fashion should never trump usability

Obviously, Aesthetic design should be left to those schooled and skilled in its application: Graphic/visual designers.

Generating artificial obsolescence through fashion is a time-honored and effective way to sell everything from clothing to cars. A new fashion should not and need not detract from user-performance: Enormous visual and even behavioral changes can be carried out that either do not hurt productivity or markedly increase it.

In any case, user test after aesthetic changes have been made, benchmarking, where applicable, the new design against the old. Ensure that learnability, satisfaction, and productivity have been improved or at least have stayed the same. If not, newly-added aesthetics that are causing a problem need to be rethought.

3.2 Anticipation. Bring to the user all the information and tools needed for each step of the process

Software and hardware systems should attempt to anticipate the user's wants and needs. Do not expect users to leave the current screen to search for and collect necessary information. Information must be in place and necessary tools present and visible.

Anticipation requires that designers have a deep understanding of both the task domain and the users in order to predict what will be needed. It also requires sufficient usability testing to ensure the goal has been met: If a tool or source for information is there on the screen, but users can't find it, it may as well not even be present.

The penalty for failing to anticipate is often swift and permanent, particularly if you do not have a captive user, as is the case with public websites and apps, for example.

3.3 Autonomy. The computer, interface, and 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 confined in too small a space 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.

We should enable the users make their own decisions, otherwise they may feel constrained and frustrated. However, allowing users latitude does not mean developers should abandon all control. On the contrary, developers must exercise necessary control. Users should not be given so much rope they hang themselves.

Adding some thresholds to determine a link has been pressed, or a scroll has been started may turn to be inadequate for some users. Such thresholds should be modifiable by the user.

3.4 Keep the user informed. 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 users to respond appropriately to changing conditions.

Status information must be kept up to date and within easy view, and the status information accurate.

Status information can be up to date, yet inaccurate. At the time of this writing, when a user updated an iPhone or iPad to a new generation of system software, a progress indicator would appear showing that it will take approximately five minutes to complete the task. Actually, it typically takes an hour or more. (The new system itself would update in five minutes, but then all the other tens or hundreds of megabytes of information on the phone had to be re-uploaded.) The user, having been lied to, was left with no way to predict when she might actually get her device back. Such a user is not feeling autonomous.

3.5 Color. 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 cannot see the colors presented.

Most people have color displays nowadays. However, approximately 10% of human males, along with fewer than 1% of females, have some form of color blindness. As a general advice, it is good to test each site or app to see what color-blind individuals see.

You can use websites such as http://enably.com/chrometric/. For images, http://www.colblindor.com/coblis-color-blindness-simulator/.

Do not avoid color in the interface just because not every user can see every color. Color is a vital dimension of our limited communication abilities. Stripping away colors that a person who is color blind can't see does no more for that person than turning off the entire picture does for a person who is completely without sight. It's the presence of an alternate set of cues for that person that is important.

As said previously with other aesthetics of design, do not strip away or overwhelm color cues in the interface because of a passing graphic-design fad.

3.6 Consistency

There are different types and levels of consistency, so consistency is a concept that has to be analyzed in different ways.

3.6.1 Levels of consistency: The importance of maintaining strict consistency varies per level.

The following list is ordered from those interface elements demanding the *least* faithful consistency effort to those demanding the *most*.

1. **Top level consistency:** *Platform consistency:* Be generally consistent with ***de jure*** (as dictated by guidelines and standards) and ***de facto*** (The unwritten rules to which the community adheres) standards. *In-house consistency:* Maintain a general look & feel across your products/services, it communicates brand and makes adoption of your other products and services easier and faster.
2. **Consistency across a suite of products:** General look & feel communicates family.
3. **The overall look & feel of a single app, application or service–splash screens, design elements, etc.:** A visual designer should establish a purposeful & well thought-through visual language, shaped by usability testing. User behaviors should be fully transferable throughout the product.
4. **Small visible structures, such as icons, symbols, buttons, scroll bars, 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 print. Their location is only just slightly less important than their appearance.
5. **Invisible structures:** Invisible structures refers to such invisible objects as Microsoft Word's clever little left 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. That is exactly what is wrong with invisible objects and why, if you insist on using them, rigid consistency becomes so important.

6. **Interpretation of user behavior:** Changing your interpretation of a user's habitual action is one of the worst things you can do to a user. Shortcut keys must maintain their meanings. A learned gesture must be interpreted in the standard way. If the button that carries the user to the next page or screen has been located at the bottom right for the last 30 years, don't move it to the top right. Changes that require a user to unlearn a subconscious action and learn a new one are extremely frustrating to users.

Apple apparently thought this was a good idea and started copying Microsoft by adding invisible controls from scroll bars to buttons everywhere. The situation on the Mac got so bad that, by the early 2010s, the only way a user could discover how to use many of the most fundamental features of the computer was to use Google to search for help.

Some objects while, strictly speaking, visible, do not appear to be controls, so users, left to their own devices, might never discover their ability to be manipulated. If you absolutely insist on disguising a control, the secret rule should be crisp and clean, for example, "you can click and drag the edges of current Macintosh windows to resize them," not, "You can click and drag various things sometimes, but not other things other times, so just try a lot of stuff and see what happens."

Objects that convey information, rather than being used to generate information, should rarely, if ever, be made invisible. Apple has violated this in making the scroll bars on the Macintosh invisible until a user passes over them.

If you want to attract existing users of someone else's product to your product, you should try to interpret your new user's commands in the same way by, for example, allowing them to reuse the same shortcut keys they've grown used to.

3.6.2 Induced Inconsistency: It is just important to be visually inconsistent when things act differently as it is to be visually consistent when things act the same

Make objects that act differently look different. For example, a trash can is an object into which a user may place trash and later pull it back out. If you want to skip the "and pull it back out" functionality, that's fine. Just make it look like an incinerator or shredder or anything other than a trash can.

Make pages that have changed look changed. If someone encounters an unfamiliar page on an updated website or in a revised app, they know to look around and figure out what's different. In the absence of such a cue, they will attempt to use the page exactly as they have always done, and it won't work.

3.6.3 Induced Continuity: Over time, strive for continuity, not consistency

If you come out with a completely re-worked area of your product or even a completely new product, it is important that people instantly recognize that something big has changed. Otherwise, they will jump into trying to use it exactly the way they always have and it just isn't going to work. "Uniformity" would mean that your next product would be identical to your last, clearly wrong, but "consistency" is little better in a field where so much growth will continue to take place. Our goal is continuity, where there is a thread that weaves through our various products and releases, guiding our users, but not tying us to the past.

3.6.4 Consistency with User Expectation

It doesn't matter how fine a logical argument you can put together for how something should work. If users expect it to work a different way, you will be facing an uphill and often unwinnable battle to change those expectations. If your way offers no clear advantage, go with what your users expect.

3.7 Default values

There are many aspects of default values and the way they should act.

For a field: default values should be easy to "blow away": When a user activates a field, the current entry should be auto-selected so that pressing Backspace/Delete or starting to type will eliminate the current entry. Avoid the cursor appear in any unpredictable location.

Not everything should have a default. If there isn't a predictable winner, consider not offering any default.

3.8 Discoverability: Any attempt to hide complexity will serve to increase it

Functional software does not have to look like a tractor; it can look like a Porsche. It cannot, however, look like a Porsche that's missing its steering wheel, brake, and accelerator pedal. Yet many tech companies in the late 1990s began purposely hiding their most basic controls, often to the serious detriment of their users. Why? Because they found it more important to generate the Illusion of Simplicity for potential buyers than to reveal the extent of complexity to their actual users.

A notable example is the *invisible Mac scroll bars*: Scroll bars are used to generate information, as a user clicks or drags within them to inform the software that the user wishes to move to a different position within the page or document. However, just as often, users will glance at the scroll bar just to see where they are in a page. Making a complex control like the scroll bar invisible likewise slows the user attempting to actually scroll.

A designer can easily create a system that will fully support both buyer and user, switching appearance depending on current need. You can design the software for an operating system, for example, that will present itself in a very simple form in the store only to slowly open up like a flower, offering the user more and more accessibility and functionality as the user becomes progressively more skilled and more comfortable. Crippling an interface might help make the initial sale, but in the long run, it can lead to having your most important "sales force," your existing customer base, not only leave you, but tell your potential buyers to stay away as well.

An important principle is **if the user cannot find it, it does not exist**. Not all buyers are naive. Even those that are don't stay that way long. Only the most persistent buyers/users will travel the web searching for a treasure map to features that you choose to hide from them. Most will simply turn to your competitors, taking you at your word that you just don't offer whatever they were after.

You can use **Active Discovery** to guide people to more advanced features. With Active Discovery, you cease waiting for people to find something and, instead, offer it to them. In its ideal form, your system "realizes" they now need it and offers it to them. In most instances, we are far from being able to do that. A workable compromise:

1. Mention to a user that a feature exists about the earliest he might need it
2. Repeat the message at intelligently spaced intervals. Not over and over again.
3. Stop mentioning it once either explored or adopted

An important consequence of all this is that **Controls and other objects necessary for the successful use of software should be visibly accessible at all times:** The object itself should either be view or enclosed by an object or series of objects (documents within folders, menu items within a menu, for example) that, in turn, are visibly accessible at all times.

Exceptions can be made for systems that are used habitually, such as a mobile browser or reader, where the screen size is so limited that it is impractical to display items not currently needed, or when it would be difficult or impossible for the user to fail to trigger the appearance of the controls by accident, thus ensuring the user will discover their existence.

Take into account that **there is no "elegance" exception to discoverability:** A few designers, having fallen in love with the clean lines of smartphone apps, thought it would be great to visit those same clean lines on giant-screen computers. Wrong! Hiding functionality to create the Illusion of Simplicity is an approach that saps user-efficiency and makes products an easy target for competitors.

Smartphone and tablet controls are sometimes forced into the content area because the screens are so small, that's the only area there is. Even there, you need to provide a standard trigger, such as a tap in the middle of the content area, that will simultaneously expose all the icons and buttons representing all the hidden controls so that users don't have to carry out a treasure hunt. But don't do this in desktop.

Apple, as of the early 2010s, began migrating controls from the area surrounding the content region of their Macintosh applications into the content region itself. The controls popped up in locations that would obscure the very content the user was attempting to affect. The user was often unable to move the controls far enough out of the way to be able to see what they were doing. Even when a control panel could be moved beyond the edges of the content window, it would revert to its original obscuring position the next time it was invoked.

Other tips:

- Communicate your gestural vocabulary with visual diagrams.

- Strive for balance: Don't put an info icon next to every single item on the page. You can have an initial overlay image to show everything (e.g. Google's Snapseed) and then remove it.
- User test for discoverability: Ensure that what you added works properly.

3.9 Efficiency: Look at the user's productivity, not the computer's

In judging the efficiency of a system, look beyond just the efficiency of the machine. 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. As a single example, forcing customers to enter telephone numbers without normal spacing or punctuation saves a single line of code and a handful of machine cycles. It also results in a lot of incorrectly captured phone numbers as people cannot scan clusters of ten or more digits to discover errors. (That's exactly why phone numbers are broken up into smaller pieces.)

Four other important principles regarding efficiency are:

- Keep the user occupied: Typically the highest expense by far in a business is labor cost.
- To maximize the efficiency of a business or other organization, you must maximize everyone's efficiency, not just the efficiency of the IT department or a similar group. Information technology 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 at the cost of lowered productivity for the company as a whole.
- 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.
- Error messages should actually help.

Error messages must be written by a skilled writer to:

1. Explain what's wrong
2. Tell the user specifically what to do about it
3. Leave open the possibility the message is improperly being generated by a deeper system malfunction

"Error -1264" doesn't do any of these. Rare is the error message that covers even Point One well. Yours should cover all three. Your Quality Assurance group should be charged with the responsibility for reporting back to you any message that does not fulfill the criteria.

3.10 Explorable interfaces: Give users well-marked roads and landmarks, then let them shift into four-wheel drive

Mimic the safety, consistency, visibility, and predictability of the natural landscape we've evolved to navigate successfully. 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 a "no-brainer" way through, while still enabling those who want to explore and play what-if a means to wander farther afield.

The earlier your users are on the experience curve, the more you need to guide them.

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."

Two important principles that facilitate exploration are:

- **Make Actions reversible:** By making actions reversible, users can both explore and can "get sloppy" with their work. A perfect user is a slow user.
- **Always allow "Undo":** The unavoidable result of not supporting undo is that you must then support a bunch of confirmation dialogs that say the equivalent of, "Are you really, really sure?" Needless to say, this slows people down.

We usually think of the absence of Undo as being the sign of lazy programming, but sometimes people do it on purpose. For example, some ecommerce sites want to make it hard for you to take things back out of your shopping cart once you've put them in there.

You should always provide a way out. Users should never feel trapped inside a maze. They should have a clear path out.

Make it easy and attractive to stay in: A clear, visible workflow that enables people to understand where they are and move either backward or forward in a process will encourage people to stick with a task.

3.11 Fitts's Law: The time to acquire a target is a function of the distance to and size of the target

Use large objects for important functions (Big buttons are faster). Use small objects for functions you would prefer users not perform.

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 significantly faster than a double row of icons with a carefully-applied one-pixel non-clickable edge between the closer tools and the side of the display. (Even a one-pixel boundary can result in a 20% to 30% slow-down for tools along the edge.)

Multiple Fitts: **The time to acquire multiple targets is the sum of the time to acquire each**. In attempting to "Fittsize" a design, look to not only reduce distances and increase target sizes, but to reduce the total number of targets that must be acquired to carry out a given task. Fitts's Law is in effect regardless of the kind of pointing device or the nature of the target

3.12 Human Interface Objects can be seen, heard, felt, or otherwise perceived

Human-interface objects are separate and distinct from the objects found in object-oriented systems. Our objects include folders, documents, buttons, menus, and the trashcan.

Human interface objects that can be seen are quite familiar in graphic user interfaces. Objects that are perceived by another sense such as hearing or touch are less familiar or are not necessarily recognized by us as being objects.

An important principle is that **human-interface objects have a standard way of being manipulated** and as a consequence **they should have standard resulting behaviors**.

3.13 Reduce the user's experience of latency

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

Acknowledge all button clicks by visual or aural feedback within 50 milliseconds

In any case, an important principle is to **keep users informed when they face delay**. From providing proper feedback to indicate that the task is being carried out to give an accurate prediction on the remaining time or entertain the user meanwhile, dealing with delays is very important to make the user comfortable.

It is important **to make it faster to begin with**. Eliminate any element of the application that is not helping. Be ruthless.

Mobile, which has an architecture more in keeping with traditional GUI applications than web browsing, has been reminding people that computers can be fast, and they are even more impatient with slow-downs. Wearables will come with an even higher level of expectation: No one waits to see what time it is, and they will not wait to see who is calling, what the temperature is outside, or any other information to be displayed.

3.14 Limit the Trade-Offs

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

How do you decide whether learnability or usability is most important? The first thing you must do is identify **frequency of use**: Are you working on a product or service that will be used only once or infrequently, or is it one that will be used habitually? If it's single-use, the answer is clear: Learnability. If someone will use this every day, eight hours a day for the rest of his or her life, the answer is equally clear: Usability.

Next, **who is the buyer?** If the person who will use it habitually will also make the buying decision, a product's reputation for learnability may be a key factor in making the sale. That's why you want to identify the most important of the two, then attack both.

3.15 Choose metaphors that will enable users to instantly grasp the finest details of the conceptual model

Good metaphors generate in the users' minds a strong series of connections to past experiences from the real world or from a previous cyberspace encounter, enabling users to form a fast and accurate sense of your system's capabilities and limitations.

Having said that, you can safely **expand beyond literal interpretation of real-world counterparts**. Not only is there no need to slavishly copy a real-world object (skeuomorphism), but unnecessarily limiting the functionality of a software counterpart just to "perfect" the imitation is most often bad design.

The inverse of skeuomorphism is abstraction, a prominent feature in so-called flat design, a fashion that took hold in 2013, turning once well-understood icons and other elements into meaningless abstractions and even false symbols. (For example, the icon for the browser on the iPhone became a compass, only connected to the concept of the web through the vaguest of abstractions. The iPhone has an actual compass, so they turned its icon into… another compass! Two compass icons: One tells you which way is north and the other connects you to your bank account. The Settings icon had originally looked like the inner workings of a clock, clearly carrying the message that this is an app that will let you see and affect the inside workings of the iPhone. That was abstracted to the point that it looks exactly like a large industrial fan.)

If a metaphor is holding you back, abandon it. A metaphor is intended to empower your user. However, there are times when it can also hold back your design.

3.15 Ensure that users never lose their work

This principle is all but absolute. Users should not 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 a client computer without proper power protection. We've gotten so used to being the victim of data loss that we often don't even notice it. So consider if what happens routinely on the web happened in real life:

3.16 Readability

There are many easy tips that should be obvious:

- **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 displays**.
- **Favor particularly large characters for the actual data you intend to display, as opposed to labels and instructions.**
- **Menu and button labels should have the key word(s) first, forming unique labels**

There are some other advices, such as **testing the designs on the oldest expected user population** or the principle that **there's often an inverse relationship between the "prettiness" of a font and its readability**.

3.16 Simplicity

Balance ease of installation vs. ease of use: As designers, we need to strive to simplify users' lives. That often requires a delicate balance between our effort to make installation of a product easier and making subsequent use of that product easier or better.

It is also important to **avoid the “illusion of simplicity”** In the early years of this century, Apple became so focused on generating the illusion of simplicity for the potential buyer that they began seriously eroding the productivity of their products. They thought they had a good reason: They wanted new products to look bright, shiny, and simple to potential users. That’s an excellent goal, but actual simplicity is achieved by simplifying things, not by hiding complexity.

You can use **Progressive Revelation to flatten the learning curve**: It is OK to make the user’s environment simpler when they are learning by hiding more advanced pathways and capabilities, revealing them when users come to need them and know how to handle them. This is distinct from the illusion of simplicity, where necessary controls are made invisible or hidden in obscure and unusual places so that users have to go on treasure hunts to find the tools they need to use right now.

And, as an aftermath, **do not simplify by eliminating necessary capabilities:** This became another Apple problem after the release of their mobile devices. In 2014 on a Mac, you can set an alarm that will trigger 90 minutes before a calendar event. On the iPad, you can set a trigger for either one hour or two, but not 90 minutes. If a person needs a warning 90 minutes before the event, that’s when they need the warning. Apple has “simplified” the interface by giving the user no way to set an arbitrary time. No weakness or defect in the underlying interface would prevent Apple from giving users this capability. It is a conscious decision to limit what people can do with the product.

3.17 Make navigation visible

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 sites or web-based applications, we must take care to reduce navigation to a minimum and make sure the remaining navigation is clear and natural. Ideally, present the illusion that users are always in the same place, with the work brought to them, as is done with the desktop metaphor. This not only eliminates the need for maps and other navigational aids, it offers users a greater sense of mastery and autonomy.