

# IDI – Common Mistakes, Principles of design

Pere-Pau Vázquez, Dep. Computer Science - UPC



<b>1. COMMON DESIGN MISTAKES.....</b>	<b>2</b>
1.1 INTRODUCTION .....	2
1.2 THE 10+ COMMON DESIGN ERRORS.....	2
<b>2 PRINCIPLES ON INTERACTION AND DESIGN .....</b>	<b>11</b>
2.1. ABOUT THE DOCUMENT .....	11
2.2. INTRODUCTION .....	11
2.3. PRINCIPLES.....	12
<b>3 OTHER ORGANIZATIONAL PROBLEMS RELATED TO UX .....</b>	<b>25</b>



### 1. Common design mistakes

#### 1.1 Introduction

The usability of applications improves when the users know how to operate the interface and it guides them through the working flow. Failing to accomplish the common rules usually prevents from both objectives.

Although the worst application design mistakes may be domain-specific, there are some common mistakes that plague lots of applications. The three main reasons for an application to fail are:

1. They do not solve the right problem.
2. They have the wrong features for the right problem.
3. They make the right features too complicated for users to understand.

Any of these mistakes may be critical for your application. In order to evaluate the application, one safe advice is to conduct user studies with representative or real users. Early prototyping and testing in order to get user feedback may avoid resources waste. Designing should be an iterative task, with several rounds of user testing in between.

The most usual way to get a wrong design is to listen to users. The best advice is to watch them rather than listening to them. Analyzing how they work may be a better tool for a successful application design.

Despite this, there are a lot of general guidelines for user interface design that should be accomplished, or, at least, taken into account. Some guidelines are important enough that breaking them may doom your application or web site.

#### 1.2 The 10+ common design errors

Next we review a list of common design mistakes.

##### 1.2.1 Non-Standard GUI Controls

We have to take into account that the user will spend more time on other applications than our application of interest. Therefore, the user has a higher amount of *exposure* to a certain type of GUI controls. If those are common to many applications, we should not change them.

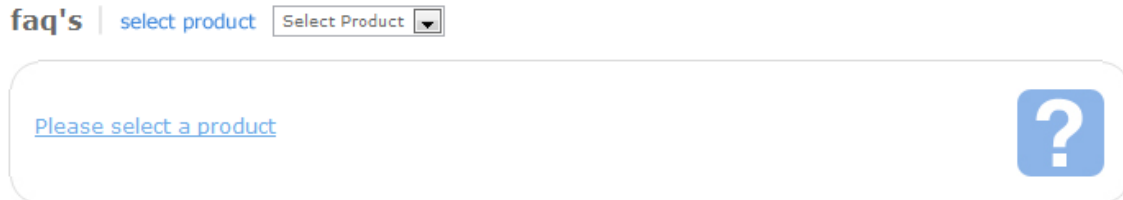
This is the main reason behind the consistency principles. Changing the appearance or behavior of those controls will confuse readers because it breaks several consistency principles, and makes learnability difficult.

The best interaction designers have refined the standard look-and-feel of GUI controls over 30 years, supported by thousands of user-testing hours. It's unlikely that you'll invent a better button over the weekend. Even if your design is better, it will be only present in a single application. When seen in context, it will be an anomalous example: Users will always have thousand times more *exposure* to a different design. Users' cognitive resources are better spent understanding how your application's features can help them achieve their goals.

### 1.2.2 Elements that look a GUI Control without being one

It is the opposite problem to the previous one. Again, when breaking user's expectations because you do not maintain consistency, a problem occurs. This is commonly a consequence of not choosing the adequate metaphors for the UI design.

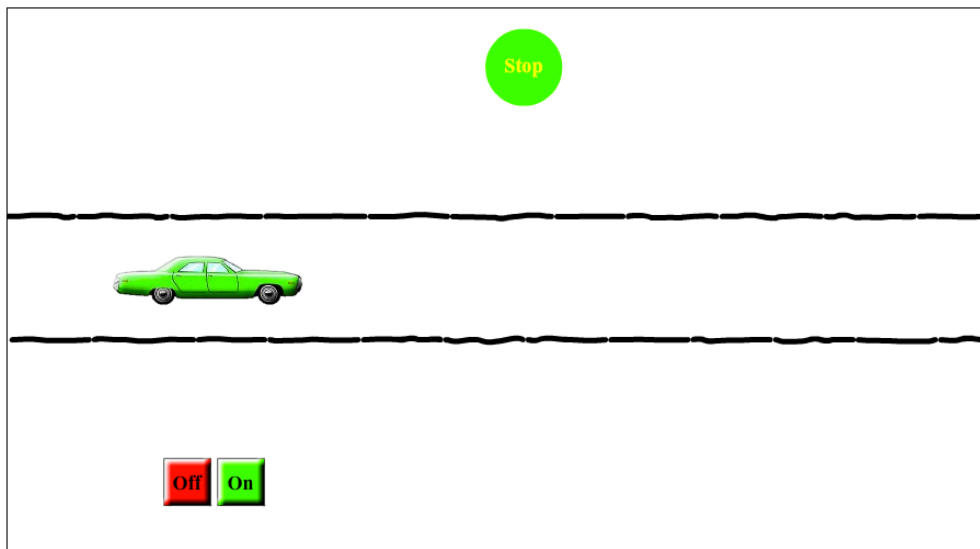
For instance, seeing text that looks like links or buttons will induce mistakes and confusion to the users. This has been present in several tools or webpages. See Figure 10. The "Please select a product" control looks like a link but it does not start any action.



**Figure 10:** Text that looks like a link and it is not.

### 1.2.3 Inconsistency

All other kinds of mistakes affect the consistency principles directly, and should be avoided. The same words, buttons, or commands should be used throughout the same application for the same task. For instance, using "Save Model", "Save File", "Save Design" on different parts of the application for the same task will confuse the user and make him or her to feel insecure. Not only commands should be consistent, but designers should also make sure they the conventional rules of colors, shapes and forms to maintain identity of the interface or product. Striving for consistency will reduce errors, avoid confusion and



improve efficiency.

**Figure 11.** Experiment that demonstrates that inconsistencies in the interface leads to a higher number of errors or longer reaction times.

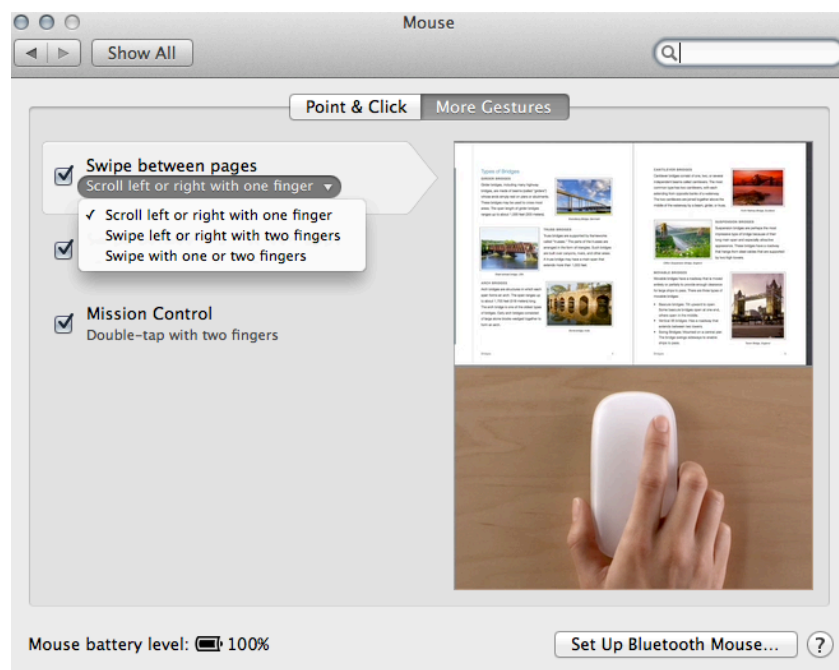
There is an interesting experiment you can test by yourselves designed by Sakshat Virtual Labs in <http://iitg.vlab.co.in/index.php?sub=72&brch=170&sim=862&cnt=1604>. The

objective of the experiment is to understand use of consistency of presentation in user interface design. You can see it in Figure 11. There is a car that starts moving when we press the “On” button, and stops when we press the “Off” button. We have to follow the instructions that appear on top of the screen, sometimes with inconsistency of colors, for example. The results of the experiment show that, even if you do not make any error, the reaction time is higher during inconsistency notifications.

#### 1.2.4 No Perceived Affordance – Lack of signifiers

“Affordance” means what you can do to an object. Don Norman commonly uses the word “signifier”. Sometimes you visit a website or open an app and you do not know what to do there or where to go from there. The lack of familiar UI elements, signals on the screen that help us understand which objects are actionable, may produce this sensation. Signifiers are especially important in UI design, because all screen pixels afford clicking — even though nothing usually happens if you click. Therefore, one must avoid that the user requires clicking here and there in order to get an action. Action starters must be clear and comprehensible.

Lacking affordances breaks the rules of discoverability. A typical example comes from today’s Mac OSX mouse modes (shown in Figure 12). They are not consistent with all devices (trackpad works differently than Magic Mouse and so on), and they can only be learnt by checking the manual. It is not clear how many fingers will do what, because no signifiers are provided. So you must go the manual and learn there the different features. To make things worse, some of these features can be configured so that the user gets more confused.



**Figure 12.** The configuration menu for the mouse gestures.

One example of element that may conflict with perceived affordance is drag-and-drop. Although in some applications (such as graphic design) the users may be familiar with this interaction mode, drag-and-drop elements in other UIs often do not provide any clue to

the user that something can be done there, or what the final effect will be. In contrast, buttons and checkboxes are UI elements that are obvious to use. Sometimes this problem is solved with text, but, if the amount of required text is high, or it disappears after the first use, we should reconsider redesigning.

This was a common problem in old computers where the resolution of screens induced the programmers to create drop-down menus that were invisible unless pulled. As a consequence, sometimes the user felt in this situation of “not knowing what to do”. Note that this is something that is appearing again, especially in mobile devices applications. Again, the reason is the same, the lack of space. Instead of solving the problem smartly, such as with a small semi-transparent button that affords to click but, at the same time, allows to see through, some programmers make the access to these features difficult such a certain key click or gesture is necessary.

Currently, there is a trend of adding too much functionality that is hidden, no signifiers are provided, in the OS of mobile devices, most notably in Apple’s iOS. In the absence of such signifiers, how is a person to know whether to swipe up or down, touch with a harder pressure than normal, or use more than one finger to a certain operation? The only way is to ask users to memorize these gestures, which completely breaks the discoverability principle. The users must be helped to discover features by signals, signifiers on screen, not by accident or reading some manual or some article that points to “X secret features” of iOS. “Secret”? Who would like to make any fancy feature secret anyway?

#### 1.2.5 Too small click targets

Click targets may be big enough for users to click properly and not be missed out. Sometimes buttons or checkboxes are clearly perceived, but they are too small or do not appear in the visible area and therefore are difficult to reach.

Like in the previous case, these sorts of unreachable UI elements appear again and again in mobile devices. They usually come in two different flavors:

- User interfaces not properly designed for the screen resolution or size: Designers of applications for mobile devices may take into account the resolution and size of the display been used. Sometimes the application has been optimized for a certain device and does not work adequately on a different, smaller one.
- Web pages that contain UI elements that are displayed very small. With the new browsers present in smartphones, users do want to visit the desktop versions of the webpages instead of the versions designed for mobile devices, which are often poorly designed, or are updated infrequently. Webpages that contain buttons to be clicked, often present those too small because the mobile browser does not automatically resize them, or facilitates their use.

Of course, these elements are particularly problematic for elder people or for people with motor skill disabilities.

#### 1.2.6 Lack of feedback

One of the most basic guidelines for improving a dialog’s usability is to provide feedback. The user must always know the system’s current state, what response has been given to their commands, and what is happening.





If you do not provide visual or audible feedback on what is happening, the user will guess, and maybe he will guess wrong.

This breaks the principle of keeping the user informed, thus making the user less autonomous.

### 1.2.7 Lack of progress indicator

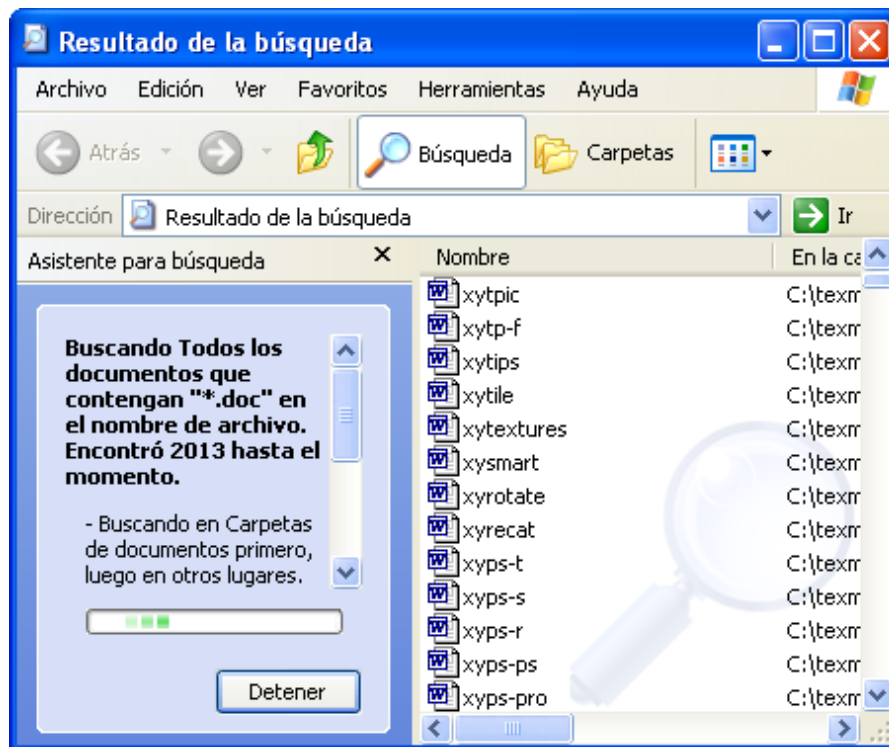
The lack of progress indicator is a variant of the previous one. If no progress indicator is provided, or the progress indicator does not work as expected (see Figure 25), then the user does not know if the system is effectively doing what he or she asked it to do. When a system fails to notify users that it's taking a long time to complete an action, users often think that the application is broken, or they start clicking on new actions.

If you can't meet the recommended response time limits, say so, and keep users informed about what's going on. In the document by Bruce Tognazzini, you have the delay times and the actions that should be taken depicted as in Figure 13.

Expected Delay	Indication
1/2 to 2 seconds	Use animated mouse cursor or other "busy" indicator 
> 2 seconds	Tell them potential length of wait
> 5 seconds	Use an animated progress indicator  Process must end by the time indicator is full!
> 10 seconds	Keep users a) informed & b) entertained
> 15 seconds	Same as >10 plus add at end a noticeable sound & strong visual indication so users know to return

**Figure 13.** Delays and actions to be taken.

Failing to provide the adequate feedbacks breaks the rule of keeping the user informed, and also breaks a second principle, that of reduce the user's experience of latency. When we do not inform properly our users, their experience worsens and their expectations get lower.



**Figure 14:** Searching window with a progress indicator that does not indicate progress.

The search window in Windows XP (see Figure 14), like the one in Windows 7, shows a progress indicator that does not provide any clue on the remaining time for the search to finish. In Mac OSX, there is only an animation that indicates that something is running, but, since the search can last several seconds, it should be replaced by a progress indicator.

Concerning the state of the application, you must never forget:

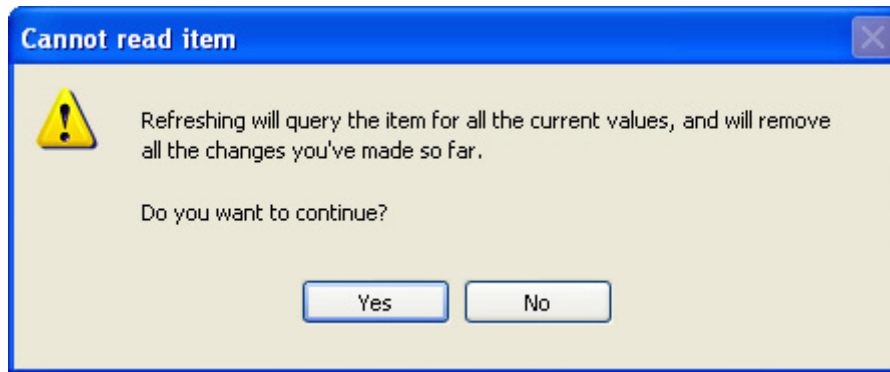
- Show users the system's current state.
- Tell users how their commands have been interpreted.
- Tell users what's happening.

### 1.2.8 Bad Error Messages

Error messages are a special form of feedback: they tell users that something has gone wrong. The most common guideline violation is when an error message simply says something is wrong, without explaining why and how the user can fix the problem. Such messages leave users stranded.

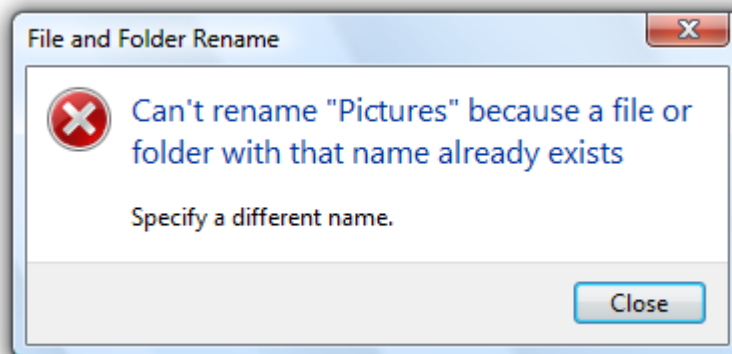
Informative error messages not only help users fix their current problems, they can also serve as a teachable moment. Typically, users won't invest time in reading and learning about features, but they will spend the time to understand an error situation if you explain it clearly, because they want to overcome the error.

An important error has to remain on screen for one or more seconds: If the message is too fast, and comes after an action that required several seconds, you might be looking somewhere else and miss the message. Take for instance the error message shown in Figure 15: Can the user grasp what is going to happen here?



**Figure 15:** Error issued when triggering a refresh. No clue of the error is provided although somebody took a time to write a long error message.

Another example, also from Windows appears in Figure 16. Although the problem seems clearly stated (though more information concerning the paths could be given), there is no clue on how to solve it, because no option to solving the problem is provided.



**Figure 16:** Error issued when triggering a refresh. No clue of the error is provided

On the Web, there's a second common problem with error messages: people overlook them on most Web pages because they're buried in masses of junk. When we try to register for a service such as a blog, a shop or so on, we eventually have to fill a large form that does not fit in a single screen. Since the registration process may require some information as compulsory, and some other fields might be checked for correction, we may end up with one or more errors that may come in different flavors. Sometimes, we see a new red colored label accompanying the wrong field. If we have multiple mistakes, we often change a single file and try again, and fail again. Having simpler pages may alleviate this problem, but providing a better error presentation (often discarded by the developers because it is more difficult to implement) may improve our experience.

These two examples show how to break the rule that error messages should actually help.

#### 1.2.9 Asking for the Same Info Twice

Most of us have suffered this on hot line services: typically, automated response machines from communications companies ask us several things such as the ID number, our telephone number, our name... And then, a human also ask us for the same data... And he eventually passes us with another person that asks us the same information!!!

This is a bothering practice that sometimes applications reproduce. Users shouldn't have to enter the same information more than once. If there is something computer excel at is at managing data. Therefore, the application should gather the information and make it available for the different parts that might require it.

Some booking applications (theatre or other events that last for several days) do recurrently commit this mistake. An example of booking flow that commits this mistake in a real application was:

- 1 Fill in the number of people
- 2 Select the day of the booking
- 3 Select the timing
- 4 Try to book seats

When the user was not able to book correlative seats for all the family, he wanted to cancel and select a different timing. However, the application turned back to first step making the process cumbersome and prone to mistakes. As a consequence, unless you found the right seats the first time, more and more time was consumed, and more and more frustration was generated.

#### 1.2.10 Lack of default values

Default values may help users in various ways. They are especially important in form filling tasks. Several uses of default values are:

- Accelerate interaction: Several tasks may have acceptable default values, such as the length of tasks to be added to a calendar. Having default values in this case may free the user from specifying a value and therefore its task is achieved faster.
- Teaching how the formatting of a certain input is or a certain answer that can be appropriate for a question. Non-American users have had a date formatting problem many times. When entering dates in apps, having an example date may reduce wrong entries in a simple manner.
- Sometimes a default value may be perfectly safe or a common value, for instance in software installation processes. This may help novice users if they do not know about the details of the task. For example, port identifiers for different security protocols in mail servers are usually the same, and therefore, the configuration utility for mail may put them as default.

#### 1.2.11 Fail to provide any usage notion on the application

Today, with the proliferation web-based applications and app stores, it is very common to encounter many new applications even without a deliberate search for an application to solve a problem. Application recommendations, for instance, that rely on groups of applications downloaded by users, that might be unlinked by theme or content is one of the sources of what could be called *arbitrary* app testing. As a consequence, users often approach the app without any conceptual model of how it works or which problem it solves. They don't know the expected outcome, and they don't know the basic concepts that they'll be manipulating. For well-known applications, such as word processors, this is less of a problem, but for new applications this may be a problem.

When you throw a user into an app and do not have any tutorial introduction or properly guide him through the steps he has to perform, the user will probably feel frustrated. For recommendation sites, such as Apple's App Store, this might turn into a bad evaluation that might throw the app into the back seat of low quality applications and never be able to go back. Note that most downloaded applications are in the top 25 or top 50 because the model of search of applications implemented in iPhones and iPads was built this way (i. e. easily see the Top 25 or Top 50 tiers but a nightmare to dig into the remaining apps). This is a problem for the newcomers: A couple of bad evaluations may sentence the app to lie into the almost unreachable jungle of mediocre applications.

You may see this from time to time when you read evaluations of apps you have used: You may find a user complaining about a missing feature you know it exists, though it is difficult to find because it appears in the custom Preferences of the operating system, instead of the application itself.

Applications should guide the users into the tasks by taking into account both the type of application and the type of user. When designing mission-critical applications may often assume that the users will have been trained and/or tried the app many times before. For other cases, we cannot assume the user will read the upfront instructions.

Again, this problem is a different flavor of how the rules of discoverability can be broken.

#### 1.2.12 Not Indicating How Information Will Be Used

The information that an application asks to the user should be clear. The user needs to know how the application will use each bit of information. A classical mistake by inexperienced users arises when they are asked for a nickname in a blog or forum: Often, the user does not know that the nickname will be used for identifying them in their postings. As a result, many times users end up typing something inappropriate.

Other pages may require entering a postcode prior to showing any product in order to show the users either the shipping costs or the availability of a certain product in the shop closer to the user. If the application fails to communicate the reason, many people may enter a fake code or simply leave the page if they are worried about privacy concerns. A simple solution is to tell the reason of this question, or to delay the calculation of the shipping costs until a certain product has been selected, provided that the user is informed that the shipping costs may vary depending on the location.

#### 1.2.13 Organizing the data according to internal application design

Any application should not reflect the system's internal view of data rather than users' understanding of the problem space.

For instance a Research Tracking system that stores the information relative to the publications of the researchers will require an option to enter a new research track (i. e. a new published paper). Obviously, this will be the most often used option by a researcher. Probably, the options to generate the users' CVs will be second to those, and the rest of features will be of infrequent use. A normal user would expect from the system, after selecting a button for entering a new research track, to enter the title of the research paper. What the user found was a view **to search** by title the supposed research element. Obviously, this was an option added to avoid double copies of the elements, which can be

entered by any of the authors. Clearly, this means moving the programmers' effort to detect duplicates to the user! This is wrong because it breaks some of the golden rules of design that state that the system should pardon the user if he errs, and should be able to recover from users' errors. Moreover, it incorrectly exposes the inability of the system to detect duplicates. A simpler way should be to allow the user to introduce the title of the work, and then check in background whether there is a similar title in the database, and then ask the user if it is the same.

#### 1.2.14 Reset buttons on Web Forms

It is almost always wrong to have a Reset button on a Web form. Reset buttons clear all the fields, thus forgetting the entire user's input. Unless you are working on a system tailored to entering data, which almost never happens on a Web Form, this is an undesirable behavior. It violates one of the most basic usability principles: to respect and protect the user's work at almost any cost. Therefore, destructive actions should have a confirmation dialog, the most destructive the most prominent.

However, designers should avoid falling into the opposite problem: too many confirmation dialogs. Asking the user to act too many times will lead to users that do not read the warnings, and therefore answering wrongly some times.

## 2 Principles on Interaction and Design

### 2.1. 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/>

I excerpted some parts to reduce the amount of reading.

### 2.2. 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.

### 2.3. Principles

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

#### 2.3.1 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.

#### 2.3.2 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.

#### 2.3.3 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.

#### **2.3.4 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.

#### **2.3.5 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.

#### **2.3.6 Consistency**

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



### 2.3.6.1 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.

#### **2.3.6.2 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.

#### **2.3.6.3 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.

#### **2.3.6.4 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.

#### **2.3.7 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.

#### **2.3.8 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.

### 2.3.9 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.

### 2.3.10 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.

### 2.3.11 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.

### 2.3.12 Human Interface Objects must be properly perceived and act as their real counterparts.

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. They appear within the user’s environment and may or may not map directly to an object-oriented program’s object. In fact, many early GUI’s were built entirely in non-object-oriented environments.

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. Ring tones are auditory objects, for example, but we tend to just think of them as ring tones, without assigning any higher-level category to them.

Human-interface objects have a standard way of being manipulated: Buttons are pressed, sliders are dragged, etc. Moreover, they have a standard resulting behavior (e.g. dropping a document on a trash can does not delete it, it stores it in the trash can. Selecting “Empty Trash” is necessary to actually delete it.). Thus, Human interface objects must look and behave as their real counterparts.



Therefore, if a new behavior is to be implemented, **use a new object** instead of a known one. If dropping a document on your delete-document icon will destroy it instantly and permanently, do not make it look like a trash can. People come with expectations about previously encountered objects. It’s important not to confuse or water down such expectations. For example, if you use a trash can icon, but instantly destroy documents dropped into it, it broadens the rule for trash cans. Instead of the rule remaining: “Dropping a document on a trash can does not delete it. Rather, it stores it in the trash can. Selecting ‘Empty Trash’ is necessary to actually delete it,” it will shift to, “Dropping a document on a trash can will destroy it either right now or sometime in the next six months to a year.” That is not only confusing for your users, it is damaging to every other developer that uses the trash can icon in the proper way.

### 2.3.13 Reduce the user’s experience of latency

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. Modern web browsers can pre-fetch data, reducing the dead time when the user reaches the end of a task and must wait for the next page to appear.



## 2.3.14 Keep users informed when they face delay

Expected Delay	Indication
1/2 to 2 seconds	Use animated mouse cursor or other “busy” indicator 
> 2 seconds	Tell them potential length of wait
> 5 seconds	Use an animated progress indicator  Process must end by the time indicator is full!
> 10 seconds	Keep users a) informed & b) entertained
> 15 seconds	Same as >10 plus add at end a noticeable sound & strong visual indication so users know to return

Make it faster to begin with. Eliminate any element of the application that is not helping. Be ruthless.

The sluggish speed of the early web set users’s expectations extremely low. (It also burst the Internet bubble when people realized they could get in their car and drive round-trip to the shopping center in less time than it took to “trick” the website into selling them something.) They have become less forgiving as time has passed.

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.

Automotive applications today are oftentimes sluggish, suffering from a fatal cocktail of weak hardware, poor design/coding practices, and high latency. Consider the car hurtling down the road at 88 feet per second (27 meters per second) while the user, eyes fixed on the flat panel display, waits to learn which of ACDC’s many fine works he’s currently enjoying. Imagine the rich irony when the accident report reveals it was “Highway to Hell.”

## 2.3.15 Limit the trade-offs between usability and 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, however, all applications and services, no matter how simple, will display a learning curve.

Learnability and usability 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.

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.

Note that testing can be done for learnability or usability. If you are working on an application that will be used habitually, it might be useful to have long tests such that you can improve efficiency for long-time workers, not just first-time users.

### 2.3.16 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.

Try making your concepts visually apparent in the software itself. If that proves impractical, make it visual apparent through an illustration. The illustration should be compact and meaningful. Test it to see if it works, then embed it in such a way that every user that needs to see it will see it.

### 2.3.17 Expand beyond literal interpretation of real-world counterparts

Most metaphors evoke the familiar, but can and usually should add a new twist. For example, an electronic newspaper might bear a strong resemblance to a traditional paper, but with hyperlinks that enable users to quickly dive as far into articles as their interest drives them, something quite impossible with their paper 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.)

The aftermath is that **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.

### 2.3.18 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.

Travel sites, in general, think nothing of repeatedly tossing all the information the user has entered about cities, times, days of travel, frequent flyer numbers, anything that takes time and trouble to type in. The user may attempt nothing more radical than leaving a half hour later, but that is apparently grounds to destroy their choice of departure city and date as well as arrival city and date. If the user is impolite enough to go to the bathroom, well, that sort of activity poses a significant security risk, so of course their entire evening's work must be destroyed, with a message explaining its been done for their own good.

Travel sites may be the tip of the iceberg, but websites in general, are notorious for their cavalier attitude when it comes to their user's hard work, and it doesn't stop there: Traditional applications continue to crash and burn, and the excuses for entire computer systems crashing and burning are at an end. Small portables can survive a power outage. It's no longer acceptable that many of today's high-end desktop computers and operating systems still 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 less reliability than 10-cent toys.

### 2.3.19 Text that must be read should have high contrast

Favor black text on white or pale yellow backgrounds. Avoid gray backgrounds. Together with this, you should also **use font sizes that are large enough to be readable on standard displays**.

Note that the current trend is not this (check <http://contrastrebellion.com/>).

It is also useful to take into account other's potential problems by **testing all designs on your oldest expected user population**.

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. You cannot.

Note that there is often an inverse relationship between the “prettiness” of a font and its readability.

Specifically, anti-aliasing softens the edges of a font, giving it a much smoother appearance on the digital page. The problem is that the human vision system responds to sharp edges, so, in smaller font sizes, an anti-aliased font, while often appearing more attractive, can be quite difficult to comprehend. There are anti-alias techniques that specifically increase the sharpness of the edges the eye is seeking, so this is not strictly a black and white issue (so to speak), but it is definitely something of which you should be aware and not something in which every graphic designer has been schooled. You will want to run some reading speed and comprehension tests on proposed font changes.

### 2.3.20 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. (See Visibility.)

It’s just fine to make your showroom products look simple, but, to the extent you want to hide complexity to avoid scaring away buyers, do so in the showroom, not in the home or office of the purchaser now trying to accomplish real work. I started putting a special Dealer Mode into Apple software in 1978, so that the product would look and act differently in the showroom than in the buyer’s home. Computers allow that. Somewhere along the line, people forgot.

One way to ease the learning curve is to use **progressive revelation**.

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.

Progressive revelation can cut down on support costs by eliminating calls from users trying to understand advanced capabilities before they have learned enough about the task domain to need them. It can also raise costs if advanced features are not introduced when they are needed or are too well hidden.

At the same time, take into account **not to 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.

The way you set a 90 minute warning on an iPad is to create a second event, 30 minutes before the real event, and set a 60 minute warning.

How is that simpler?

Likewise, they have a very simple interface for finding photographs in your collection: You look through all your folders of photographs, one at a time, until you find the picture you're looking for. Apple will neither display nor allow you to sort/search on the title, caption, or keywords you've carefully associated with your images. One can argue that the interface is simple: If you want to find a specific photo among that 20,000 on your iDevice, just look through your 73 folders you've created in iPhoto or Aperture to hold them until you find it. You don't have to learn about searching, you don't have to remember the name, you just have to have 10 to 20 minutes on your hands to spend the time looking.

How is that simple?

Fortunately, after many years, help is at hand: Apps like [Photo Shack HD](#) (the HD is important) enable you to search on all the criteria that Apple is importing but refusing to show you. That's OK for really advanced features. However a remarkably high percentage (100% to be exact) know how to search.

### 2.3.21 Track the state

Because most browser-based products exist in a stateless environment, this does not mean we cannot track the state. Tracking the state may greatly help us and our customers in several tasks.

Our systems should “know”:

- Whether this is the first time the user has been in the system
- Where the user was when they left off in the last session
- What the user has found of interest based on time spent with a pointing device moving, objects being touched, etc., in different areas
- Where the user has been during this session
- Where the user is right now and what they are doing

and myriad other details. In addition to simply knowing where our users have been, we can also make good use of what they've done.

One site with which you are familiar is so involved in and good at tracking state that it could be described as a state-tracking system that happens to do other stuff. That site is amazon.com. Their uncanny ability to make suggestions on what we might want to explore and buy is the result of their understanding our full history on their site. They know what expensive items we've come back to repeatedly in the past, what we've lingered over recently, and what would go well with what we just or recently purchased based on like-minded individuals.

### 3 Other organizational problems related to UX

As we have commented elsewhere, designing and implementing an application is something complex, and user acceptance of the application may be a key issue in the future results. UX is often neglected during the design of an application, but it is of key importance for consumers, since it will determine its technical and emotional connection with the applications or web pages. Moreover, studies say that every dollar invested in UX yields a return between 2 and 100 dollars.

Therefore, it may become very important to be aware of the users' needs and expectations. Throughout the whole development process, many actions can be taken to make sure that the user experience does not suffer or that we detect UX problems early enough to have time to correct them.

Despite that, it is quite common to *forget* the user in early stages of the project, or to deal with UX as if it was a separate part of the project. Some of the common problems are listed below.

#### 3.1 Leave the UX for too late

As we have said, a very common problem is leaving the UX research for the final stages of a project. Since usability testing uncover important problems that might require changes in the architecture of the application, letting the analysis for the latter stages (which may be commonly pressed by the schedule), may be a big mistake.

Significant changes in the architecture of the application or the overall design may be easily addressed when the application is in its early stages, but very difficult the more time has been invested into it. Several studies have shown that the maximum benefits are achieved when UX introduced early in the development process.

In many cases user feedback can be obtained with simple, unfinished versions of the applications. Even with prototypes, a lot of information can be gathered on the user interface design. Therefore, it is highly advisable to evaluate the design in its early stages, leaving more room for changes or improvements in case the results guide you to do so.

#### 3.2 Improper distribution of UX efforts

Usability testing may discover many elements that can be improved in an application. Commonly, the result of a user study will be a list of problems with some level of priority that rate the importance of the problems. When dealing with problems, it is important to keep the focus. Since the resources are limited, it is always important to address the most important problems first, and the less important later.

In order to do so, it is useful to keep in mind the 80/20 principle: Not all the features will be accessed with the same frequency. Devoting more efforts to the 20% of the features that are more commonly used or are more important to our application may be a good way to invest efforts.

#### 3.3 Lack of communication

In big companies/projects, the people working in UX may form a medium-sized team. Since its work may be quite different from the rest of engineers, this might result into an

isolated group of people that seems to be working in a very different project than the others, with just a finite number of common points of interest.

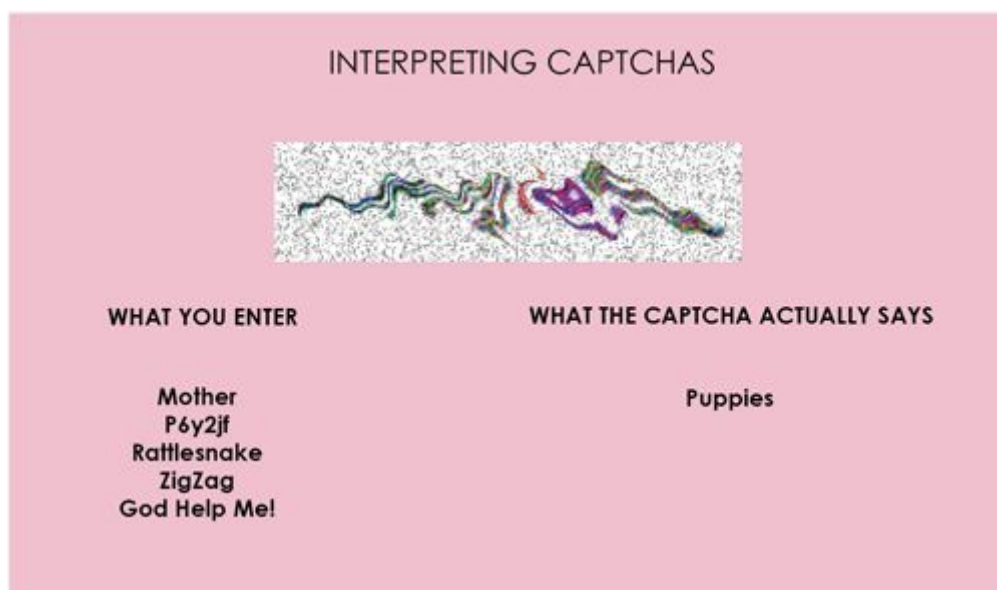
Working together with developers and analysts, and spreading the information throughout the company/project members may make the work carried out less mysterious. Note that UX team are not there to *correct* or find the mistakes of anybody else in the project, but to help the other project members to make a better product for the user.

Thus, a good policy is to communicate frequently with other groups, both formally and informally, since this will facilitate collaboration.

### 3.4 Poor UX examples

Captchas by themselves are problematic because they transfer a problem to the user. The maintainers of a computer system, or a server may encounter the problem that any automatic piece of program could register to a website, mail service, and then make probably not adequate use.

To avoid this, somebody (Mark D. Lillibridge, Martin Abadi, Krishna Bharat, and Andrei Z. Broder) invented captchas, that are a form of text deformation that it is difficult to read by machines. This moves the responsibility to the users, that have to interpret sometimes gibberish and happens what appears in Figure 17.



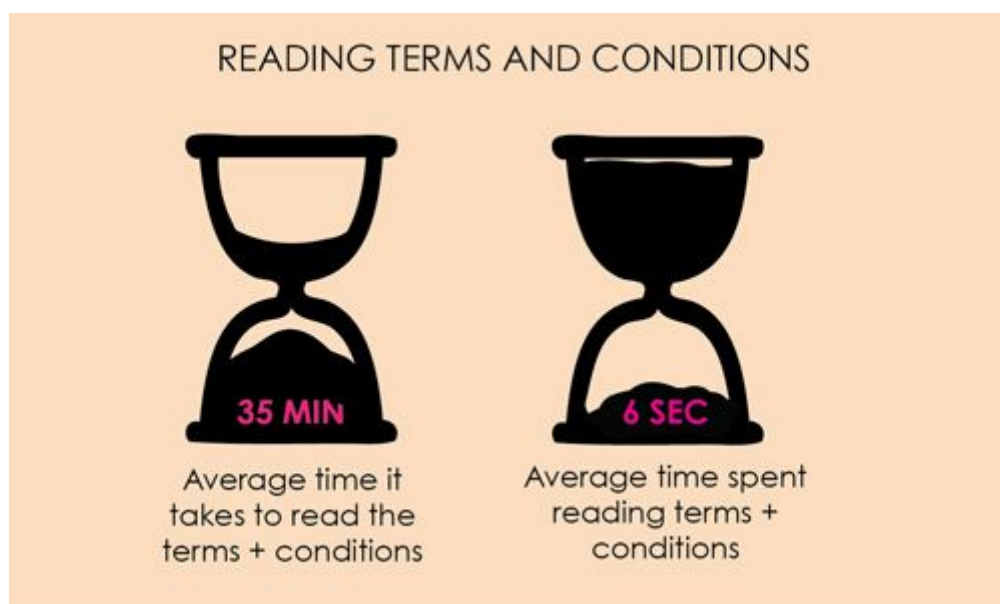
**Figure 17:** Captchas: Properly designed are a way to move a problem from the application owner to the user, but poorly designed they are even worse, because users cannot figure out what is written.

The difficulties in software also make users waste tons of time in things that should be simple, such as videoconferencing systems, as depicted in Figure 18.



**Figure 18:** Time spent in virtual meetings.

Many software packages, webpages, services, and so on, are accompanied by a set of terms and conditions that everybody should read, because they are defining the relationship that we will have with the company that provides the service. However, in most cases, nobody reads them, and the companies know that the users did not read them, as illustrated in Figure 19.



**Figure 19:** Readability problems are typically exaggerated in Terms and Conditions documents, something everybody should read and nobody does.