

A Primer of Human-Computer Interaction

Affordances	3
Real vs. Perceived Affordances	3
Cues	4
Constraints	5
Example 1: login at petco.com	7
Example 2: the “wheelbarrow” link at garden.com	8
Natural Mappings	9
Metaphors	10
Don Norman's principles of good design	10
Slips	12
Mistakes	13
Preventing Errors: Instructions won't prevent errors	13
Preventing Errors: Ways to prevent errors	15
Handling errors	17



The field of human-computer interaction design has been developed to bridge the communication gap between people and computers by studying their interaction and designing systems that facilitate it. It may seem odd that a field of science should emerge to empirically study something that was artificially created. But computers introduced new possibilities for interaction and new behaviors emerged as a result.

User Interface – the parts of the device (in our case, the parts of the computer and its software) with which the user interacts by seeing, touching, hearing, talking.

It is not surprising that humans and computers are different. They were designed to be. In fact, computers were developed to compensate for human limitations. Where humans are good at reasoning, pattern recognition, developing multiple strategies, and adapting, computers are good at counting, accurate storage and recall, rapid and consistent responses, and uniform performance over time. The two are complementary. The only thing keeping this dynamic duo from reaching its full potential is communication.¹

Why is human-computer communication so hard? *Because computers can't receive and reproduce the wide range of stimuli that people create and perceive.*

When speaking, people express themselves in complex ways that usually involve a combination of talking, gesturing, making facial expressions, etc. Talking alone is complex: the variations in the pace, volume, and rhythm of speech affect the meaning of what is said. The same words can take on a new meaning in a different context. Going beyond speech, human actions and perceptions also exhibit great variability: a person never does or perceives anything exactly the same way. And no two people are exactly alike.

¹ Communication (and understanding) is the number one obstacle in human-computer interaction. While knowledge of how computer hardware and software works helps, our understanding of the inner workings of the computer will become less important as computers are “taught” to receive and produce a wider range of stimuli that are natural to humans. The move from command line, text-only interface to graphical interfaces was just the first step. Spoken language recognition was the next. And on April 28, 2003 Intel took computers another step closer to understanding humans by releasing Audio Visual Speech Recognition (AVSR) software. AVSR “enables computers to detect a speaker's face and track their mouth movements. Synchronizing video data with speech identification enables much more accurate speech recognition,” according to Intel's press release.

The only way that computers have been able to understand humans is through a translation mechanism placed between the analog world of people and the digital world of computers. That mechanism is *the interface*.

Unfortunately, due to technological limitations of input/output devices and processing capabilities of computers the interface has done more to *standardize* human inputs than to accurately translate them. It erases most input variations before they reach the computer's processor. For example, there are many different ways of double-clicking a mouse button: apply more or less pressure, move the finger quickly or slowly, with a longer or shorter interval, while moving or keeping the mouse still between the clicks etc. The device itself translates any button click into a digital signal of the same "magnitude" regardless of the pressure applied by the finger. In addition, the software ignores the variation in timing between the clicks as long as the interval is within a predefined range. The point here is that computers are not designed to be sensitive to the range of human expression or to produce stimuli with the degree of complexity that surrounds people in the physical world.

These limitations help explain why non-anthropomorphic design is the *de facto* standard in user interface design. Non-anthropomorphic design principles basically state that a computer should not be personified; the users should not be deceived to feel that they're dealing with an intelligent, human-like entity. In light of this, interface designers should utilize non-anthropomorphic language when describing what computers and users do:

- use words like *process*, *print*, *search*, or *store* rather than *know*, *think*, *understand*, or *remember* to refer to what a computer does;
- use words like *click*, *drag*, *type*, *use*, *operate* rather than *ask*, *tell*, or *touch* when referring to what a user does.

Natural language search engines like AskJeeves are an interesting example of the failure of anthropomorphic design. AskJeeves is no different than "regular" search engines like Google when it comes to interpreting the search query: they just can't handle the complexity of natural language and resort to picking out keywords from search queries. In fact, they're counterintuitive, and therefore counterproductive, in suggesting that they "understand" natural language.²

² As speech recognition becomes better and direct manipulation interfaces are introduced, the two may one day replace the conventional graphical user interfaces altogether. An intriguing example of a direct manipulation interface is digital clay – little building blocks which, when assembled to model a physical object, "know" where they are in relation to each other. (www2.parc.com/spl/projects/modrobots/).

The interface equation

Given these human-computer communication roadblocks, how does the human user determine what actions are possible and what result those actions will have? Taking advantage of the interface's affordances is a good place to start.

Affordances

An affordance is an action that a user can perform on an interface element (or an entire interface) based on its physical characteristics. For example, buttons afford pushing, knobs afford turning and so on.

In GUI design, affordances are especially important with interactive elements like buttons, sliders, lists, or checkboxes. For instance, a button affords pushing. What makes this affordance specially strong is that about the only thing you can do with a button is to push it. This is why buttons have been carried over into the graphical interface design so successfully. In fact, if you want to make something look “clickable” in GUI design, making it look like a button is one of the best ways to do it.

But what happens when actions that look possible are not? Conversely, what if a possible action is hidden? Enter real and perceived affordances.

Real vs. perceived affordances

Real affordances include all possible actions based on all properties of an interface element, visible or not. For example, the volume control knob in my car stereo not only controls the volume, but also acts as the power switch: you turn it to adjust the volume and push it to turn the power on and off. While it may not be clear from the appearance of the knob that it can be both rotated *and* pushed, both actions are its real affordances.

Perceived affordances are actions that are *perceived* to be possible. For instance, the perceived affordance of my car stereo's knob is turning only.

A design is flawed whenever real and perceived affordances do not match. There are two situations when this can occur:

- *False affordances* are present. False affordances arise from a situation where the appearance of an object suggests an action that is not possible. For example, imagine a door that has two identical U-shaped handles on either side, but opens only one way. Both handles suggest grasping and

pulling (perceived affordance), yet only one of them allows for that action while the other requires pushing (real affordance). On the web, using blue underlined text for emphasis rather than a link label is an example of giving plain text a false affordance.

- *Hidden affordances* are present. Hidden affordances are created when a possible action is not evident from the appearance of an object. The volume control knob that is also a power switch is a good example. On the web, a text link that has the same appearance and behavior as body text is an example of a control with a hidden affordance.

Here is a summary of affordances expressed in algebraic terms (“U” stands for “union” and applies to the sets of affordances on either side of it):

Real affordances = true visible affordances U true hidden affordances

Perceived affordances = true visible affordances U false visible affordances

Since users usually won’t attempt to perform an action they don’t think is possible, it is easy to see why in a well-designed system real and perceived affordances must match.

Of course, affordances alone can’t predict how users will interact with an interface. A model of how users interpret interfaces based solely on affordances doesn’t take into account two more “ingredients” in the decision-making process: the user’s goals and knowledge. Given a user’s goal and knowledge of the system, context, and the world in general, every perceived affordance gives rise to either a cue or a constraint, depending on whether the interface element looks like it will bring the user closer to accomplishing the goal or not.

Cues

A cue, in a human factors context, is a call to action derived from an interface element whose perceived affordance suggests that interacting with it will help the user reach a goal. There are three types of cues:

- *Semantic cues* are based on the user’s knowledge of the situation and how the world works in general. A simple example is the “Buy” button on eCommerce sites. The users know that to take possession of something they need to buy it, so clicking the button becomes an alternative for action, or a cue.
- *Cultural cues* are based on cultural conventions attached to the interface element and the situation. For an American customer a buy button labeled “Add to shopping cart” will be clearer than a button labeled “Add to shopping basket,” because the latter label is based on a term used

in the British culture. A cultural cue is also created by including an arrow pointing to the right in the label of a “Continue” button. This is because in Western cultures, where text is written from left to right, right is the culturally accepted forward direction.

- *Logical cues* are based on the user’s ability to use reason to determine what action to take. For example, imagine a user coming across a large, prominent button on the last page of the checkout. The user knows that placing the order is the most important thing to do on this page and that there should be a button to do it. The button’s size and prominence give the user the logical cues that clicking it will probably place the order.

Note that there are two logical cues in the last example: the button is large, and it is prominently placed. A user will often take advantage of multiple cues to make the final decision to act. In the case of a “Place Order” button, the user may first use the two logical cues to identify the button as the likely candidate and then use its label—a semantic cue—to decide to click it. The point here is that *the more cues there are, the more usable a system becomes*.

When cues are inadequate or absent, however, the user is left with relying on constraints to decide what to do. This is a much less efficient way to interact with an interface.

Constraints

The characteristics of an interface element which suggest that activating it will *not* help the user reach a goal give rise to constraints. Given a goal, one can use constraints to reject interface elements until the best option remains.

There are four types of constraints:

- *Physical constraints* are the most basic type of constraint: if something can’t be done it won’t be done. A large peg will not fit into a small hole. A GUI design example would be a disabled or restrictive form element (i.e. a pulldown instead of a text box), physically limiting the range of acceptable inputs.
- *Semantic constraints* rely on users’ ability to use the meaning of the situation to reject certain actions. For example, in a virtual dressing room a customer will not try to place a pair of pants on a manikin’s head.
- *Cultural constraints* use cultural conventions to limit the number of possible actions. For instance, a user wanting to buy something will not click a button labeled “add to wheelbarrow,” because wheelbarrows are used for things one already owns.
- *Logical constraints* rely on the user’s ability to reason their way to the correct action by eliminating illogical choices. For example, imagine that during a checkout you are presented with a screen containing two buttons labeled “Ringo” and “Paul.” In addition, the “Ringo” button is

Why are there no physical cues?
Because they are equivalent to perceived affordances. No knowledge or goal is required.

significantly smaller than the “Paul” button. If your goal is to go to the next step of the checkout, then logic will dictate that the smaller button is *not* the one you need. Your logical conclusion would be reinforced if the smaller button is also to the left of the larger one. Unfortunately, the lack of clear labeling would still leave you “cueless.”

The important point here: had the continue button been labeled properly you would not have had to rely on constraints but rather use the button’s label for cues—a much more efficient way to decide what to do.

We have covered a lot of ground in the last few pages. Let me summarize everything in an equation illustrating the relationship among affordances, goals, knowledge, cues, and constraints:³

The interface equation

$$(\text{Perceived affordances} + \text{Knowledge}) \cap \text{Goal} = \text{Cues} \cup \text{Constraints}$$

Expressed in words this equation states that *as we try to decide what interface option to choose we (a) notice the affordances of the elements in an interface, (b) use our logic and/or knowledge of the situation, culture, etc. to understand the meaning of the affordances in the context of our interaction with the system, and then (c) compare each one to our goal to get cues and constraints that help us choose the most appropriate action.*

What happens next depends on what type of a user you are. Different types of users will use cues and constraints somewhat differently to decide what to do:

- optimizing users will evaluate the entire set of interface affordances this way to arrive at a complete set of cues and constraints. If more than one cue is found they choose the one that looks most promising;
- satisficing users will not evaluate the entire set of affordances, but instead go with the first cue they see—usually a semantic cue derived from an element’s label. Most users exhibit this type of behavior when using the web.⁴

³ \cap stands for “intersect.” In our case this means that only the affordances that match the user’s goal yield cues/constraints to be used in deciding what to do.

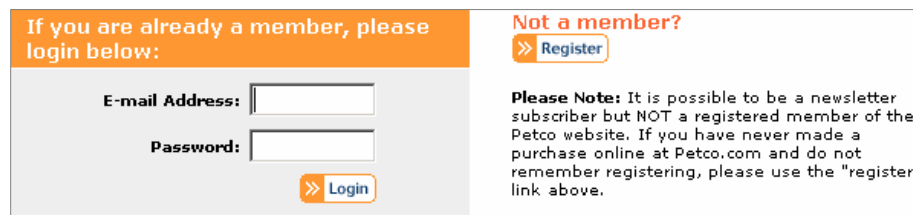
⁴ Even though optimizing and satisficing are not the only behavior types, I use these two as examples to illustrate how different types of behavior affect how people use cues and constraints to make decisions. Of course, the fact that a single user can optimize at one time and satisfice the next introduces yet another variable into the long list of factors governing human behavior, making it even more difficult to model it. All we can do is estimate.

The following two examples illustrate how cues and constraints work together to guide the user. As you go through them keep in mind that an interface element can provide both cues and constraints at the same time, depending on what knowledge and physical characteristics are used to derive them.

Example 1: Login at petco.com

Petco.com shows the following login screen to a users who are not logged in after they click “Go to the checkout” from the shopping cart page. While the options on this screen are clearly labeled, neither of them matches the user’s goal well.

The user knows what he wants to do in his own words: give his address and payment info and place the order. In this case, however, this doesn’t help: he must choose between one of the two available options: to log in or register.



If you are already a member, please login below:

E-mail Address:

Password:

[Login](#)

Not a member?

[Register](#)

Please Note: It is possible to be a newsletter subscriber but NOT a registered member of the Petco website. If you have never made a purchase online at Petco.com and do not remember registering, please use the "register" link above.

Login at petco.com.

The choice of actions is limited to:

- entering email/password and clicking “Login” in the “...already a member...” section of the screen, *or*
- clicking the “Register” button in the “Not a member?” section.

The user’s thought process to deduce the correct action may run like this:

“I am not a member so the login stuff doesn’t apply to me (semantic constraint), and the only other alternative here is the “Register” button. But I don’t want to register, I want to go to the checkout. Assuming I am on the right screen, I guess I must register to check out. Ok, I’ll click ‘Register’.”

Even though the customer was successful in choosing the proper action, using constraints along with a relatively weak logical cue was not an efficient way to decide what to do; especially if the final choice was not what the user wanted to do: the site asked the user to register while he just wanted to go to the checkout.

Example 2: The “wheelbarrow” link at garden.com

Even though garden.com is now long gone, its “wheelbarrow” link remains a great example of how poor labeling can send the customer on a wild goose chase trying to use the label for cues. Try to guess where a link labeled “wheelbarrow” can lead on a gardening site.

The “wheelbarrow” link at garden.com

While the link is placed in the general area where a shopping cart link would be found on most sites (near the top of the screen), the link name was obscure to new visitors.



Garden.com’s designers apparently were not aware of how important cultural and semantic cues are in design. Otherwise, they would not have called their shopping cart a “wheelbarrow.” The fact that you put stuff in it and you roll it around may provide a weak semantic cue for users who want to see the shopping cart. On the other hand, the cultural meaning of a wheelbarrow makes it a constraint for user wanting to see their shopping cart: you put stuff into a wheelbarrow *after* you bought it. Therefore, one could guess that the function of the wheelbarrow is to keep track of the items the customer has already bought.⁵

In user testing the label was so confusing that users who wanted to see the shopping cart resorted to going through the interface elements one by one to rule them out until the “wheelbarrow” link was the only one left. In other words, they used logical constraints—the least efficient way to make decisions. Of course, the fact that the link’s “click me” affordance was so weak (no 3-d treatment, no underline) didn’t help either.

The point of these two examples is this: whenever an interface element provides both cues and constraints, or, worse yet, no cues at all (pecto.com example) you’ve got a design flaw. As you are designing or evaluating an interface be sure to examine its physical, semantic, cultural, and logical messages for inconsistencies.

⁵ This could actually be a useful application. It could help the customers remember what they ordered come next season of planting. Because the list would be automatically maintained through order history, no effort on the part of the customer would be required to maintain it. Of course, a simpler albeit less marketable label like “Order history” would work even better here.

Two more habits of highly intuitive interfaces

Taking advantage of natural mappings

A *mapping* is a relationship between an action and its result. For example, there is a mapping between the position of a car's gas pedal and the speed with which it accelerates. It is important to understand that the user doesn't need to know how a device operates to establish mappings. In our car example, the user doesn't need to know that the movement of the gas pedal doesn't affect acceleration directly. The mapping is established simply by looking at the action and its result.

In the case of *natural mappings*, the relationship between the action and its result is clear even if it has not been empirically established. For example, there is a natural mapping between the steering wheel and the direction in which the car turns: turn the steering wheel to the left to make the car turn left and vice versa. The user only needs to know how the steering wheel operates and which behavior of the car it controls. Given this knowledge, the user can then use the natural mapping between the steering wheel's and the car's available directions of turning to figure out which way the steering wheel needs to be turned. That is, the user doesn't need direct experience using the control to predict its effect.

The relationship between the movement of a mouse and the movement of the pointer on the screen is one example of a natural mapping from the computer UI. To predict what direction the pointer will move the user must be familiar with the movements of the mouse, the movements of the pointer, and be aware that moving the mouse will move the pointer.⁶

Another example is mapping between the volume control sliders in Windows OS and the volume of the sound coming out of the speakers. The sliders don't have labels for where they would be moved to increase or decrease the volume, but the natural mapping suggests that they should be moved up to turn the volume up.

⁶ Of course, a potential problem with the mouse is that it allows actions that don't produce any results. Consider a classic example from Star Trek. When faced with a 20th Century computer, Scotty picks up the mouse and speaks into it: "Hello computer!" When I was a little kid, I tried opening a can of Coke by rotating the tab 180 degrees and tearing it off the can! The tab allowed for actions that clearly did not produce the desired result.

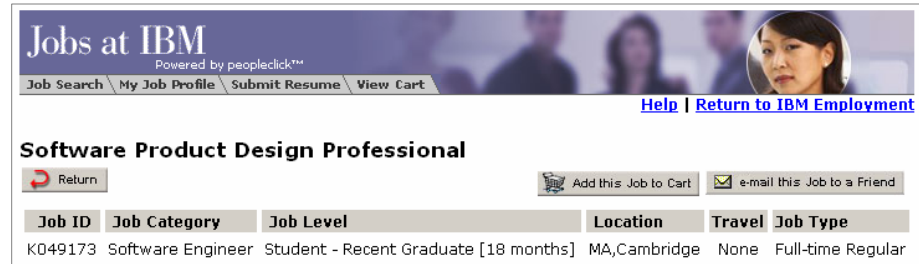
Using metaphors to build on familiar conceptual models

Interface metaphors based on physical objects make use of the conceptual models users formed by interacting with those objects. These metaphors help users get started using an interface and make immediate progress by applying what they know about the object on which the metaphor is based. The shopping cart—the metaphor of eCommerce⁷—is an example. Having learned how physical shopping carts work, users have a general model of how an online version of the same operates.

Most metaphors are powerful enough even if they take advantage of only a part of the model on which they are based. In the example of the shopping cart, both the physical cart and its online counterpart work the same at the most basic level: they let you store the items you selected for purchase until you pay for them when you leave the store. The fact that you can't push the online shopping cart around doesn't make the metaphor any less effective.

Metaphors are also flexible. The model doesn't have to be perfect and can be expanded. For example, the shopping cart metaphor still works on sites that sell services rather than products. Peopleclick pushes the metaphor even farther: a shopping cart is used to “store” *jobs*.

A shopping cart is used to store jobs at Peopleclick. A metaphor that is an interface standard is truly a powerful thing...



Don Norman's principles of good design

Don Norman's principles of good system design provide a clear and concise litmus test for usability of just about any interface. The first two principles can also be used as guidelines to help designers create better interfaces. The other two explain what happens in the users' mind when they interact with an interface.

⁷ Actually, the shopping cart has evolved from a metaphor into an interface standard. Jakob Nielsen says that most users don't think of a physical supermarket when they encounter shopping carts online. "Instead, they think of all the other websites where they have seen shopping carts."

<i>Principle</i>	<i>Description</i>	<i>Design implications</i>
<i>Visibility</i>	The available actions and system states are visible	Good visibility makes the device's affordances clear. Visibility is especially important in computer interaction design because, unlike mechanical devices, computers work invisibly.
<i>Feedback</i>	Actions produce visible results	Feedback helps the user validate the proper operation of a control and the system. For example, if the buttons on my phone merely stopped making sounds when I pushed them I would assume the entire device—the phone—was broken.
<i>Mapping</i>	Actions produce consistent results	Being able to predict the outcome of an action next time it is executed is essential in learning new interfaces. Increasing or varying the time delay between the action and its result weakens the mapping.
<i>Conceptual Model</i>	By interacting with the system the user can get an idea of how it works	The entire set of actions and results is used to form a model of how a system operates. When an undesired operation passes as normal the model becomes flawed.

I have ordered the four principles so that each principle's prerequisite is above it:

- **Visibility has no prerequisites.** Either you see what's going on or you don't.
- **Feedback requires visibility of results.** If you can't see the results of your actions you can't determine if something you did had any effect. Just think about the tree in the forest: if no one hears it is fall, did it?
- **Mapping requires consistent feedback.** If something you do produces a consistent result, you can conclude that there is a causal relationship between what you do and what happens.
- **Conceptual models require strong mappings.** The model can be formed only when the relationship between actions and consequences is firmly established.

Of course, visibility and feedback won't produce results if the object that is visible and producing feedback is not your locus of attention. This is especially important on the web, where the users' attention span is especially short and a myriad of page elements compete for attention. Jef Raskin treats locus of attention extensively in "The Humane Interface," and explains how a misplaced locus of attention leads to error.⁸ I won't try to summarize Raskin's excellent account here, but instead will turn your attention to a very common user behavior both online and off: error.

Locus of attention – an object in the physical world which has your attention or an idea about which you are thinking.⁸

⁸ Jef Raskin, "The Humane Interface" (Addison-Wesley, 2000), pp. 17 – 32.

Designing for error

People are naturally prone to error and are even tolerant of it to a degree. In fact, error tolerance plays a major role in human evolution because it encourages experimentation and learning through “trial and error.” We would still be living in the trees if we didn’t experiment. However, experience teaches us that we can be much more efficient when we don’t make errors. The purpose of this section is to give you the tools that will enable you to reduce the “error” half of “trial and error” behavior for your site’s visitors. Let’s start by first looking at the two types of errors: slips and mistakes.

Slips

Slips are errors where intending to do one thing you find yourself doing another.⁹ Slips usually occur when you get distracted, do something else at the same time, or otherwise not pay full attention to the task. It is important to understand this type of error, because the conditions for it have a lot in common with the conditions under which many people use the web.

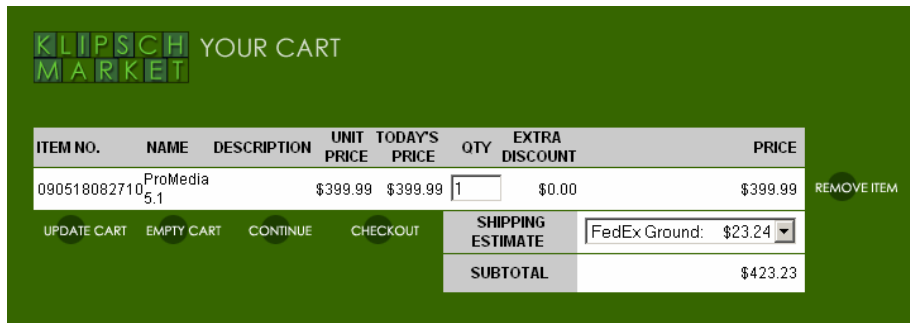
For example, have you ever typed a *search query* into a box and sent it on its merry way only to get an error message saying that what you just typed is not a valid *email address*? In this case you knew you needed to type something into an input box, but got distracted by, say, your microwave demanding attention. So you quickly typed your query into the first box you saw—which turned out to be a newsletter signup box—and left to supervise your microwave’s negotiations with a frozen dinner.

Another example is clicking the “Reset” button to submit a form because the “Reset” button was more prominent than the “Submit” button. In this case you may have relied on the logical cue that the more prominent button is the button that activates the more important action—to submit the form. So you clicked it only to lose all of your hard work.

⁹ Donald Norman, “The Design of Everyday Things” (Doubleday, 1988), p. 105.

Mistakes

Mistakes are conscious errors where, given a goal, what you believe to be a correct action turns out to be incorrect. Poor labeling is one of the major reasons for mistakes on the web. The shopping cart at klipsch.com has buttons labeled: “CONTINUE” and “CHECKOUT.” Only one of them lets the user continue with the process of purchasing a product: “CHECKOUT.” The other one is actually a “continue shopping” button. A user who wants to go to the checkout and doesn’t notice the “CHECKOUT” button is likely to click the “CONTINUE” button instead. The only thing that saves this arrangement is the fact that the buttons are placed next to each other: a user is likely to notice both alternatives and pick the best one.



Shopping cart at Klipsch.com

A row of identical buttons may look “sleek,” but users have to work harder to tell them apart. The designer should have given the buttons more differentiating characteristics than just the text labels. At least make the CHECKOUT button look different.

This is why remote controls that have buttons in various shapes are easier to use.

Preventing errors

If an error is possible someone will make it. Web sites are especially prone to user error (vis-à-vis industrial design), because design standards are still evolving and many web sites try to be different for no apparent reason. There is, however, a handful of design principles for error prevention and recovery in software and industrial design can be applied to the design of web interfaces.

Instructions won’t prevent errors

Remember the saying: if all else fails, read the instructions. Research shows that this is especially true on the web: people don’t read instructions until they’re in trouble. My own experience even suggests that some users would rather abandon a confusing page altogether than spend time reading instructions. There’s a handful of reasons for this:

- **It’s faster to figure it out by trying than by reading instructions.** If the system is responsive and the instructions are long, the users will try to figure out how to use a feature by trial and error.

- **The penalty for incorrect actions is low.** The “Back” button is the perfect antidote for clicking the wrong link. There are also very few irreversible actions to be committed on web sites. There are exceptions, of course, including stock trades, deleting emails, or placing an order on a site that doesn’t allow cancellations within a reasonable time period.
- **Reading instructions does not produce immediate results.** Reading instructions is what users *have* to do, but completing a task is what they really *want* to do. Reading instructions often seems unproductive, because it is not a core part of the task. Kids and games are a good example: kids want to be playing, not reading (or even listening to) instructions. They want the instant gratification of the game play.
- **It is easy to give up on the web.** People invest their time and money when they buy and install software. If it is difficult to use, they don’t just give up and try another product. They try to figure out how it works and maybe even read the instructions. Not so on the web. Even though there is some muddling through that inevitably occurs—the users have invested time in finding the site and their upbringing and experience teaches them not to give up easily—users are more likely to leave rather than read instructions if they can’t figure out how to use a site.

Another problem with instructions is that users often don’t look for them even when they’re stuck. I saw a good example of this during a usability test of cduniverse.com. The problem there was that the product detail page did not have a conventional buy button:

Click on price to put in cart (you can always remove it later)		  \$13.29 Add To Wish List
Status	Usually ships in 1-2 days	
List Price	\$18.98 (You save \$5.69)	
Category	Rock / Pop, Country	
Label	Dreamworks Nashville	
Orig Year	2003	
		

Unconventional way to add an item to the cart at cduniverse.com

The user struggled to find the buy button even though the instructions to place items into the cart were literally “staring her in the face.” She was not looking for the instructions!

One user scrolled up and down the page a few times looking for the button. After finally deciding that there wasn’t one she started looking for something clickable that could possibly allow her to buy the CD. Finally she decided to click the price because, as she put it, it was “near to where I expected the button to be.” She didn’t see the instructions until after she clicked the price. The moral of the story is: *If users struggle with your site, make the interface self-explanatory by carefully choosing the physical characteristics, labeling, placement, etc. of its controls and labels before adding instructions.*

The guidelines presented in the rest of this book are, at their core, designed to help web users form good conceptual models and avoid errors.

Ways to prevent errors

- › **Use physical constraints: remove and disable controls.** This one is simple: if you don't want the user to click something, hide it or deactivate it. Consider the checkout on amazon.com: the pages there don't have any links or graphics that are not essential to checking out. Even the site's logo loses its link to home.
- › **Use forcing functions.** Forcing functions are "situations in which actions are constrained so that failure at one stage prevents the next step from happening."¹⁰ For example, you can disable/enable form fields based on what is entered elsewhere on the form to prevent certain combinations of data to be entered.¹¹
- › **Provide clear labels.** Clear labels provide good semantic/cultural cues and constraints. What's more, good labels create flexibility in design. In the klipsch.com example (page 13), labeling the easily-confused "CONTINUE" and "CHECKOUT" buttons so that they could stand on their own would have enabled the designers to put them in more natural places.
- › **Use appearance of controls to differentiate their functions.** This helps prevent slips that are associated with using a control that looks too similar to the one intended. The buttons in the klipsch.com shopping cart represent the worst case scenario—all of the buttons look the same! This makes the user have to rely on labels alone for cues, taking more time to identify the correct action. Were the "CHECKOUT" button more prominent than the other buttons and the "CONTINUE" button been a link, the user would have more cues than just the button labels to go on. See the discussion of button design in the "Form Design" chapter for more on this.
- › **Reduce memory load.** A lot of decision-making on the web is based on cues and constraints supplied by the interface. A user relies extensively on short-term memory for temporary storage of elements of interest when scanning a page. Unfortunately, short-term memory has a limited size (fewer than 10 unrelated items vs. about 100 million items for long-term

¹⁰ Norman, p. 132.

¹¹ It is better to disable controls rather than hide them. Keeping the control visible helps users learn the system faster by seeing all options. When the system activates the disabled function it will be easier for the user to find again. More on this later.

memory) and is very volatile: a slightest distraction leads to loss of “data.” Here are two examples of how a good interface can reduce both long- and short-term memory load:

- don’t require the user to recall or remember choices, past actions, or data—that’s what computers are for. Instead use interface elements that prompt recognition. Put most of the knowledge required to make a decision in the world, rather than in the head.¹²
 - present manageable lists of options: keep their number low, make sure they’re easily differentiated from one another, group them logically, and sort them. A low number of differentiated options lets the user pick the best option easily. Grouping lets the user focus on the things of interest and not have to pay attention to (and therefore, remember) irrelevant options. Sorting helps locate the correct option faster. Imagine what would happen if phone book entries were not sorted!
- › **Minimize distractions.** Avoid animations and moving/blinking text. Keep irrelevant content off “mission critical” pages like the login or checkout. Doing so ensures that the user’s locus of attention is the task at hand. It also decreases the stress on short-term memory, which is volatile enough even without distractions.
- › **Reduce the overall frustration level.** For example:
- minimize delays and use progress indicators for things that will take more than a few seconds to complete. True progress indicators are difficult to implement on the web, but most sites can get away with simply displaying a static message like “searching...please wait” during long searches;
 - increase perceived page load speed by loading some elements quickly and others in the background. For example, split the page up horizontally into several tables—the browser will render each table as soon as it gets all the HTML for it. Most browsers already load images and applets in the background.

Calm users make fewer mistakes.

- › **Make sure the system itself is reliable** so that system errors don’t go uncorrected, leading to even more errors.

The point here is that instead of spending countless hours programming error-handling logic, first design an interface that *prevents* as many errors as possible. This approach is faster, cheaper, and creates a better experience for the users.

¹² For a detailed discussion of knowledge in the head and knowledge in the world see Don Norman’s “The Design of Everyday Things.”

Handling errors

It is usually impossible to prevent all errors. In some cases, it may be better to allow for more arbitrary actions/inputs (thereby creating more room for error) for the sake of efficiency of use. This is a popular consideration with systems designed for “expert” users. For example, you may choose to use a text box in favor of a dropdown for entering the two-character abbreviation of a customer’s state of residence. While this adds more error checking, the improvement in input speed is worth it.

Here are a few error handling basics that the professionals who design and develop software learned in school but sometimes ignore in the real world:

- › **Make error messages visible to let the user know that an error has occurred.** For example, make error messages stand out by using a unique color and prominent placement. Is it important for the user to realize that the behavior of the system is not normal, so that the error does not impact the user’s model of the correct system operation.
- › **Phrase the error message in terms the user can understand and never blame the user.** A message that makes no sense is useless. Tell the user what happened without being too technical. If it is a system error, clearly indicate that and take the blame for the error. If it is a user error, don’t blame the users directly: chances are the flaw is in the design. Besides, many users will blame themselves even if they are not at fault.
- › **Make the error message helpful.** The error message must be constructive. Do not just tell your users that something’s wrong. Give them enough information so that they understand why the error occurred and how to fix it.
- › **Allow users to recover from errors.** Make sure users can easily recover from errors be it correcting an invalid zip code or canceling an erroneously placed order. Confirmation messages like “Are you sure you want to do this?” are not a substitute for allowing the users to reverse their actions: most users have already made up their mind to do what they’re being asked to confirm. In such cases confirmation messages are just an irritation. For example, the “move to recycle bin” confirmation in MS Windows is unnecessary for all but first time users. Even if the user invoked a delete command by accident the deletion is reversible.

I will return to the discussion of error handling in detail at the end of the next chapter, where I can place error handling in its most common context: forms.

Ease of learning vs. efficiency of use

An interface geared toward the ease of learning will provide more information to the user (more knowledge in the world, rather than in the head), have more physical constraints and forcing functions (i.e. use more restrictive form elements), use fewer elements per screen, etc. In fact, features of easy to learn interfaces might as well be called “easy not to make a error” features.

Interfaces geared toward the efficiency of use are basically the opposite of this, allowing for maximum speed and flexibility. These interfaces “assume” that the users “know what they’re doing.” Of course this does not mean that these users will not make errors. If anything, more error checking will be required because of the added flexibility and, therefore, more room for error.