CHAPTER

Limits on Attention Shape Our Thought and Action

8

When people interact purposefully with the world around them, including computer systems, some aspects of their behavior follow predictable patterns, some of which result from the limited capacity of attention and short-term memory. When interactive systems are designed to recognize and support those patterns, they fit better with the way people operate. Some user-interface design rules, then, are based directly on the patterns and thus indirectly on the limits of short-term memory and attention. This chapter describes seven important patterns.

WE FOCUS ON OUR GOALS AND PAY LITTLE ATTENTION TO OUR TOOLS

As Chapter 7 explained, our attention has very limited capacity. When people are doing a task—trying to accomplish a goal—most of their attention is focused on the goals and data related to that task. Normally, people devote very little attention to the tools they are using to perform a task, whether they are using computer applications, online services, or interactive appliances. Instead, people think about their tools only superficially, and then only when necessary.

We are, of course, *capable* of attending to our tools. However, attention (i.e., short-term memory) is limited in capacity. When people refocus their attention on their tools, it is pulled away from the details of the task. This shift increases the chances of users losing track of what they were doing or exactly where they were in doing it.

For example, if your lawn mower stops running while you are mowing your lawn, you will immediately stop and focus on the mower. Restarting the mower becomes your primary task, with the mower itself as the focus. You pay little attention to any tools you use to restart the mower, just as you paid little attention to the mower when your primary focus was the lawn. After you restart the mower and resume mowing

Designing with the Mind in Mind. https://doi.org/10.1016/B978-0-12-818202-4.00008-8

the lawn, you probably won't remember where you were in mowing the lawn except that the lawn itself shows you.

Other tasks—for example, reading a document, measuring a table, counting goldfish in a fish tank—might not provide such a clear reminder of your interrupted task and position in it. You might have to start over from the beginning. You might even forget what you were doing altogether and go off to do something else.

That is why most software design guidelines state that software applications and most websites should not call attention to themselves; they should fade into the background and allow users to focus on their own goals. That design guideline is even the title of a popular Web design book: *Don't Make Me Think* (Krug, 2014). That is, if your software or website makes me think about *it* rather than what I am trying to do, you've lost me.

WE NOTICE THINGS MORE WHEN THEY ARE RELATED TO OUR GOALS

Chapter 1 described how our immediate goals filter and bias our perception. Chapter 7 discussed the connections between attention and memory. This chapter provides examples showing that perceptual filtering and biasing, attention, and memory are all closely related.

Our environment is full of perceptual details and events. Obviously, we cannot notice and keep track of everything that happens around us. Nonetheless, it is surprising to most people how little we notice of what goes on around us. Because of our extremely limited short-term memory and attention, we don't waste those resources. When an event happens, the few details about it that we notice and remember later are usually those that were important for our goals at the time of the event. This is demonstrated by two related psychological phenomena: *inattentional blindness* and *change blindness*.

Inattentional blindness

When our mind is intensely occupied with a task, goal, or emotion, we sometimes fail to notice objects and events in our environment that we otherwise would have noticed and remembered. This phenomenon has been heavily studied by psychologists and labeled *inattentional blindness* (Simons and Chabris, 1999; Simons, 2007).

A clear demonstration of inattentional blindness is an experiment in which human subjects watched a video of two basketball teams passing a ball from one player to another. Subjects were told to count the number of times the white-suited team passed the ball. While they watched the video and counted the ball passes, a person in a gorilla suit sauntered onto the basketball court, thumped his chest, and then walked out of view (see Fig. 8.1). Afterward, subjects were asked what they remembered from the video. Surprisingly, about half of them did not notice the gorilla. Their attention was fully occupied with the task (Simons and Chabris, 1999).[1]

[1] For more information and videos, search the Web or YouTube for "inattentional blindness."

FIGURE 8.1

Scene from video used in the "invisible gorilla" study. For more information about the study and to see the video, go to theinvisiblegorilla.com. (Figure provided by Daniel Simons.)

Change blindness

Another way researchers have shown that our goals strongly focus our attention and memory is by showing people a picture, then showing them an altered version of the same picture and asking them how the two pictures differ. What is surprising is that the second picture can differ from the first in fairly clear ways without people noticing. The reason is that people's attentions were captured by features in the image other than those that changed, and human attention can only keep track of a few features at a time. This is a type of inattentional blindness, but it is so common that it has its own name: *change blindness* (Angier, 2008).

Look at Fig. 8.2, then look at Fig. 8.3. Flip back and forth a few times. What is different about the two pictures?

Even if you noticed that the tree branch at the top of Fig. 8.2 is missing in Fig. 8.3, you may have missed the fact that the person on the far left in Fig. 8.2 is missing in Fig. 8.3. If so, it was probably because your attention was captured by the globe, water, clouds, building, etc.

To explore further, researchers gave people questions to answer about the first picture, which affected their goals in looking at it and therefore what features they paid attention to. The result: people tend to miss changes in features other than those their goals made them pay attention to. So for example, if I showed you Fig. 8.2 and asked you to notice any people in it, then showed you Fig. 8.3 and asked what was different, you would certainly notice the absence of the man.

A striking example of change blindness comes from experiments in which researchers holding city maps posed as lost tourists and asked local passersby for directions.

FIGURE 8.2

Look at this image, then at Fig. 8.3. What is different? (Credit: Wikipedia article on "Change Blindness.")

FIGURE 8.3

Look at this image, then at Fig. 8.2. What is different? (Credit: Wikipedia article on "Change Blindness.")

When the local person focused on the tourist's map to find the best route, two workmen—actually, more researchers—walked between the tourist and the local person carrying a large door, and a new researcher-tourist replaced the first one. After the door passed, over half the people continued helping the "tourist" without noticing any change, even when the two "tourists" differed in hair color or whether they had

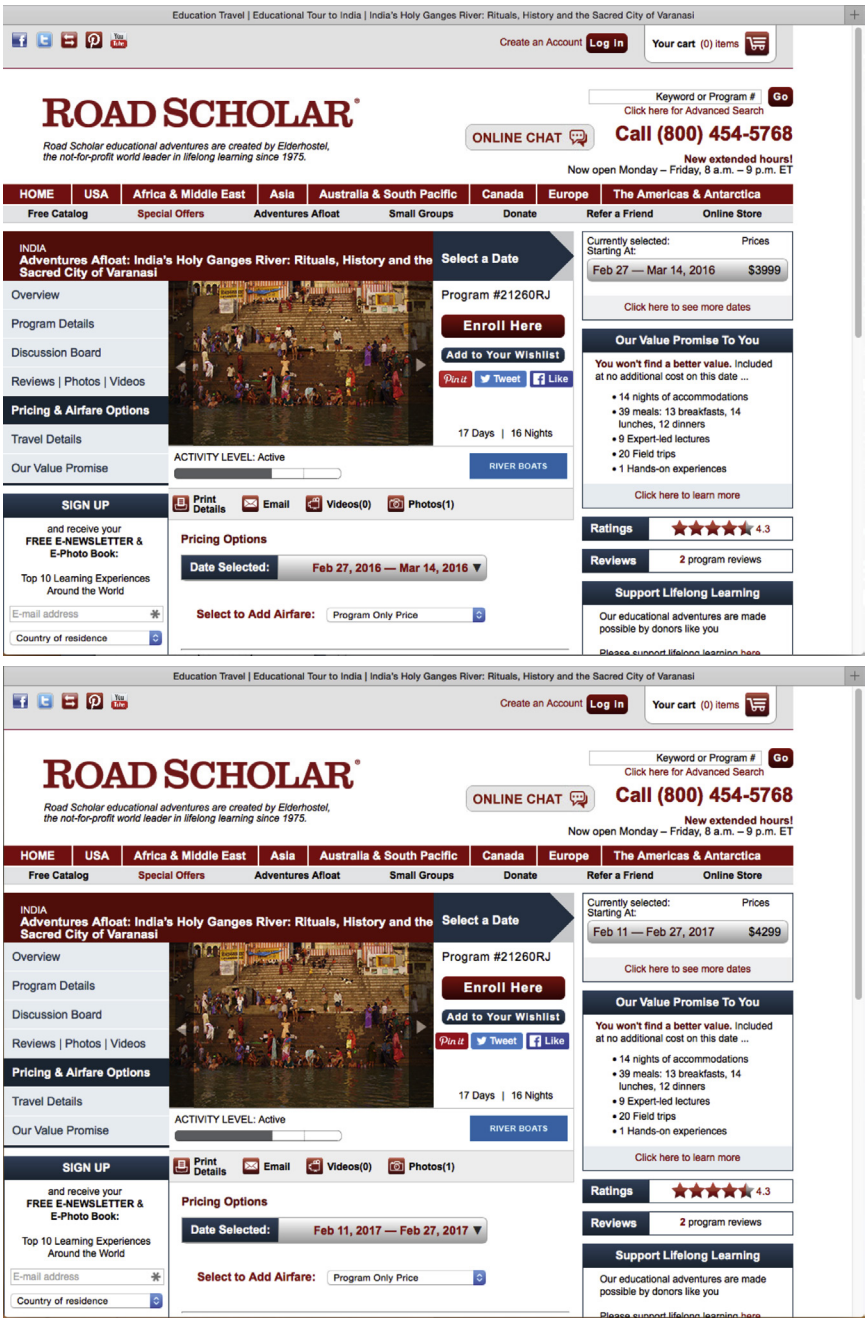


FIGURE 8.4

Users of RoadScholar.org may not notice that when they change the date of a trip (bottom center), the price changes (middle right).

a beard (Simons and Levin, 1998).[2] Some people even failed to notice changes in *gender*. The researchers' explanation: people focus attention on the tourist only enough to decide if he or she is a threat or worth helping, mentally record only that the person is a tourist needing help, and then focus on the map and the task of giving directions.

When people interact with software, electronic appliances, or online services, it is not uncommon for them to fail to notice important changes in what is displayed. For example, in a study of older adults using travel websites, some participants exhibited change blindness; when they changed trip options (e.g., departure date or city), they did not notice that the price changed (Finn and Johnson, 2013) (see Fig. 8.4).

The user-interface design guideline that follows from such findings is to make important changes highly salient—so obvious that they are hard to miss—and take steps to draw users' attention to the change. For example, a way to draw users' attention to a new error message is to vibrate it briefly when it first appears (see Chapter 6) or highlight it briefly before it reverts to a "normal" appearance.

What happens in our brains

Using functional magnetic resonance imagery and electrical encephalography, researchers have studied the effect of attention on how our brains respond to objects displayed on a computer screen.

When people passively watch a computer display with objects appearing, moving around, and disappearing, the visual cortex of their brains registers a certain activity level. When people are told to look for (i.e., pay attention to) certain objects, the activity level of their visual cortex increases significantly. When they are told to ignore certain objects, the neural activity level in their visual cortex actually drops when those objects appear. Later, their memory for which objects they saw and didn't see correlates with the degree of attention they paid to them and the level of brain activity (Gazzaley, 2009).

WE USE EXTERNAL MEMORY AIDS TO KEEP TRACK OF WHAT WE ARE DOING

Because our short-term memory and attention are so limited, we learn not to rely on them. Instead, we mark up our environment to show us where we are in a task. Examples include:

- ***Counting objects***. If possible, we move already counted objects into a different pile to indicate which objects have already been counted. If we cannot move an object, we point to the last object counted. To keep track of the number we are on, we count on our fingers, draw marks, or write numbers.

[2] For demonstrations of change blindness, search YouTube for those words, or for "door study" or "person swap."

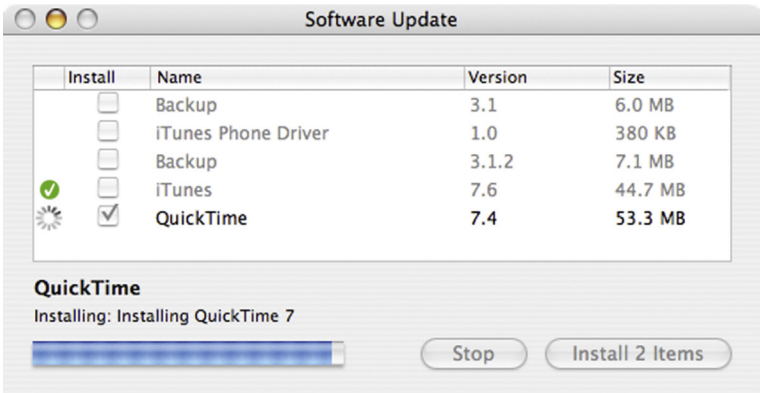


FIGURE 8.5

The macOS software update shows which updates are done (green check) versus which are in progress (rotating circle).

- ***Reading books***. When we stop reading, we insert bookmarks to show what page we were on.
- ***Arithmetic***. We learn methods of doing arithmetic on paper, or we use a calculator.
- ***Checklists***. We use checklists to aid both our long- and short-term memory. In critical or rarely performed tasks, checklists help us remember everything that needs to be done. In that way, they augment our faulty long-term memory. While doing the task, we check off items as we complete them. That is a short-term memory aid. A checklist that we cannot mark up is hard to use, so we copy it and mark the copy.
- ***Editing documents***. People often keep to-be-edited documents, documents that are currently being edited, and already edited documents in separate folders.

One implication of this pattern is that interactive systems should indicate what users have done versus what they have not yet done. Most email and texting apps do this by marking already-read versus unread messages, most websites do it by marking visited versus unvisited links, and many applications do it by marking completed steps of a multipart task (see Fig. 8.5).

A second design implication is that interactive systems should allow users to mark or move objects to indicate which ones they have worked on versus which ones they have not. MacOS lets users assign colors to files. Like moving files between folders, this technique can be used to keep track of where one is in a task (see Fig. 8.6).

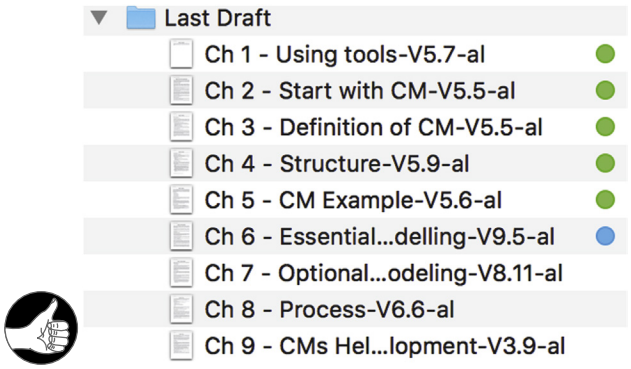
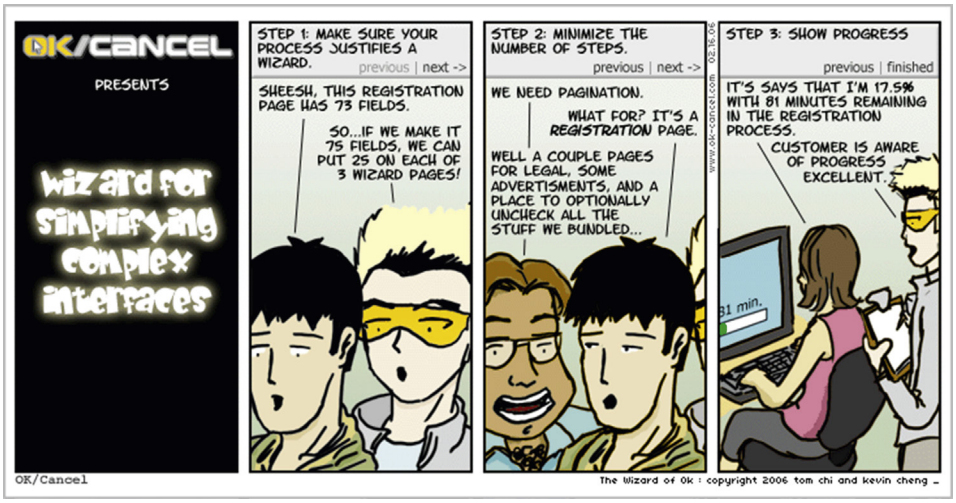


FIGURE 8.6

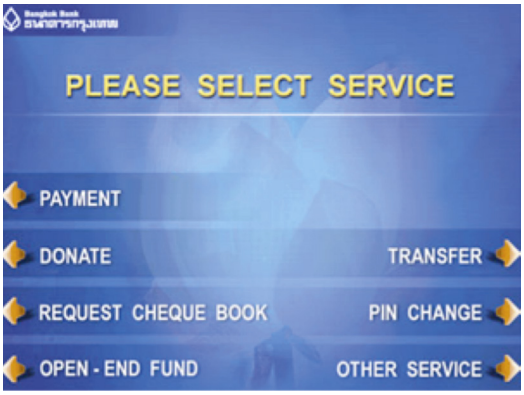
The macOS lets users assign colors to files or folders; users can use the colors to track their work.



Used by permission, OK/Cancel.com.

WE FOLLOW INFORMATION "SCENT" TOWARD OUR GOAL

Focusing our attention on the things relevant to our goals makes us interpret what we see on a display or hear in a telephone menu in a very *literal* way. People don't think deeply about instructions, command names, option labels, icons, navigation bar items, or any other aspect of the user interface of computer-based tools. If the goal in their head is to make a flight reservation, their attention will be attracted by words like "buy," "flight," "ticket," or "reservation" or symbols depicting money, tickets, or flights. Other items a designer or marketer might hope will attract customers, such as "bargain hotels," will not attract the attention of people who are trying to book a flight, although they might be noticed by people who are seeking bargains.



For each task below, what on the screen would attract your attention?

- Pay a bill
- Transfer money to your savings account
- Pay your dentist by funds transfer
- Change your PIN
- Open a new account
- Purchase travelers' cheques

FIGURE 8.7

ATM screen—our attention is drawn toward items that match our task and goal.

This tendency of people to notice only things on a computer screen that match their goal, and the literal thinking that people exhibit when performing a task on a computer, has been called "following the *scent* of information toward the goal" (Chi et al., 2001; Nielsen, 2003). Consider the ATM display shown in Fig. 8.7. What is the first thing on the screen that gets your attention when you are trying to do each of the tasks listed?

You probably noticed that some of the tasks direct your attention initially to the wrong option. Is "Pay your dentist by funds transfer" under "Payment" or "Transfer"? "Open a new account" probably sent your eyes briefly to "Open-End Fund" even though it is actually under "Other Service." Did the task "Purchase travelers' cheques" make you glance at "Request Cheque Book" because of the word they share?

The goal-seeking strategy of following the information scent, observed across a wide variety of situations and systems, suggests that interactive systems should be designed so that the scent is strong and really leads users to their goals. To do that, designers need to understand likely user goals at each decision point in a task and ensure that each choice point in the software provides options for every important user goal and clearly indicates which option leads to which goal. Analyzing software in this way is useful, and is called a *cognitive walkthrough* (Wharton et al., 1994).

For example, imagine that you want to cancel a reservation you made or payment you scheduled. You tell the system to cancel it, and a confirmation dialogue box appears asking if you really want to do that. How should the options be labeled? Given that people interpret words literally in following the information scent toward their goal, the standard confirmation button labels "OK" (for yes) and "Cancel" (for no) would give a misleading scent. If we compare a cancellation confirmation dialogue box from Marriott.com with one from Bank of New Zealand, we see that Marriott.com's labeling provides a clear scent, while Bank of New Zealand's button-labels would mislead users into clicking on the wrong buttons (see Fig. 8.8).

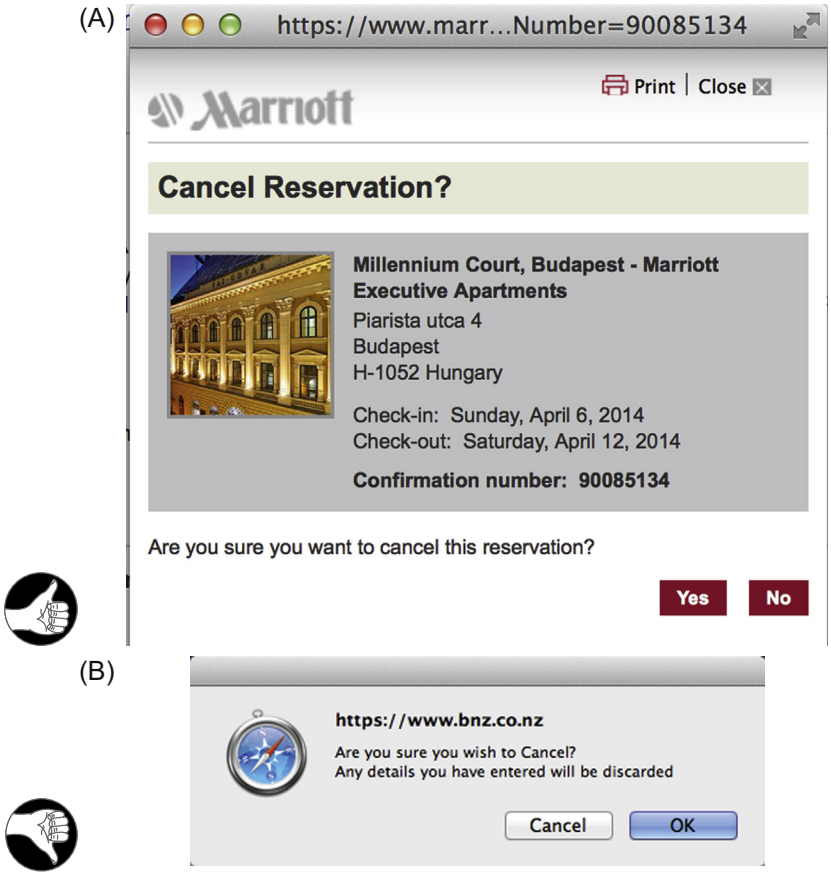


FIGURE 8.8

The button labels on Marriott's reservation cancellation confirmation dialogue (A) provide a clear information "scent" leading users toward their goal, while Bank of New Zealand's wire-transfer confirmation dialogue (B), in which "OK" cancels the transaction and "Cancel" cancels the cancellation, does not.

WE PREFER FAMILIAR PATHS

People know that their attention is limited, and they act accordingly. While pursuing a goal, they take *familiar* paths whenever possible rather than exploring *new* ones, especially when working under deadlines. As explained more fully in Chapter 10, exploring new paths is problem-solving, which severely taxes our attention and short-term memory. In contrast, taking familiar, well-learned routes can be done fairly automatically and does not consume much attention and short-term memory.

Years ago, in a usability test session, a test participant in the middle of a task said to me:

> **I'm in a hurry, so I'll do it the long way.**

He knew there probably was a more efficient way to do what he was doing, but he also knew that learning the shorter way would require time and thought that he was unwilling to spend.

Once we learn one way to perform a certain task using a software application, we may continue to do it that way and *never* discover a more efficient way. Even if we discover or are told that there is a "better" way, we may stick with the old way because it is familiar, comfortable, and most important, requires little thought. Avoiding thought when using computers is important. People are willing to type *more* to think *less*.

Why is that? Are we mentally lazy? Usually, yes. Conscious thought is slow, strains working memory, and consumes much energy. We essentially run on batteries—the food we eat—so energy conservation is an important feature of our makeup. Operating via automatic processes is fast, doesn't strain working memory, and conserves energy. So the brain tries to run on automatic as much as possible (Kahneman, 2011; Eagleman, 2012, 2015).

The human preference for familiar, mindless paths and "no-brainer" decisions has several design implications for interactive systems:

- ***Sometimes mindlessness trumps keystrokes***. With software intended for casual or infrequent use, such as bank ATMs or household accounting applications, allowing users to become productive quickly and reducing their need to problem-solve while working is more important than saving keystrokes. Such software simply isn't used enough for the number of keystrokes per task to matter much. On the other hand, in software used all day in intensive work environments by highly trained users, such as airline telephone reservations operators, unnecessary keystrokes required to perform a task are very costly.
- ***Guide users to the best paths***. From its first screen or home page, software should show users the way to their goals. This is basically the guideline that software should provide a clear information scent.
- ***Help experienced users speed up***. Make it easy for users to switch to faster paths after they have gained experience. The slower paths for newcomers should show users faster paths if there are any. This is why most applications show keyboard accelerators for frequently used functions in their menu-bar menus.

OUR THOUGHT CYCLE: GOAL, EXECUTE, EVALUATE

Over many decades, researchers studying human behavior have found a cyclical pattern that seems to hold across a wide variety of activities. Cognitive scientist Don Norman (1988) named the pattern the "human action cycle":

- Form a ***goal*** (e.g., open a bank account, make dinner, or revise a document).
- Choose and ***execute*** actions to try to make progress toward the goal.
  - Translate goal into an unordered collection of tasks (e.g., make main course, make salad, make dessert).

  - Order the tasks into an action sequence (i.e., a plan) that moves toward the goal.
  - Execute the action sequence.
- ***Evaluate*** whether the actions worked—that is, whether the goal has been reached or is nearer than before.
  - Check the results of executing the action sequence.
  - Compare the results with the expected results.
  - Determine whether at the goal or closer to the goal than before.
- ***Repeat*** until the goal is reached (or appears unreachable).

People cycle through this pattern constantly (Card et al., 1983; Norman, 1988). In fact, we run through it at many different levels simultaneously. For example, we might be trying to insert a picture into a document, which is part of the higher-level task of writing a term paper, which is part of the higher-level task of passing a history course, which is part of the higher-level task of completing college, which is part of the higher-level goal of getting a good job, in order to achieve our top-level goal of having a comfortable life.

As an example, let's run through the cycle for a typical computer task—buying an airline ticket online. The person first forms the primary goal of the task and then begins to break that down into actions that appear to lead toward the goal. Promising actions are selected for execution, executed, and then evaluated to determine whether they have moved the person closer to the goal.

- ***Goal***: Buy airline ticket to Berlin using your favorite travel website.
- ***Step 1***: Go to travel website. You are still far from the goal.
- ***Step 2***: Search for suitable flights. This is a very normal, predictable step at travel websites.
- ***Step 3***: Look at search results. Choose a flight from those listed. If no flights on the results list are suitable, return to Step 2 with new search criteria. You are not at the goal yet, but you feel confident of getting there.
- ***Step 4***: Go to checkout. Now you are getting so close to your goal that you can almost smell it.
- ***Step 5***: Confirm flight details. Check it—all correct? If no, back up; otherwise, proceed. Almost done.
- ***Step 6***: Purchase ticket with credit card. Check credit card information. Everything look okay?
- ***Step 7***: Save or print e-ticket. Goal achieved. Add departure time and date to your calendar.

To keep the airline ticket example brief, we didn't get into the details of each step. If we had, we would have seen substeps that followed the same *goal–execute–evaluate* cycle.

Let's try another example, this time examining the details of some of the higher-level steps. This time the task is sending flowers to a friend. If we simply look at the top level, we see the task like this:

Send flowers to friend.

If we want to examine the goal–execute–evaluate cycle for this task, we must break it down a bit. We must ask, *How* do we send flowers to a friend? To do that, we break the top-level task down into subtasks:

Send flowers to friend.
 Find flower delivery website.
 Order flowers to be delivered to friend.

For many purposes, the two steps we have identified are enough detail. After we execute each step, we evaluate whether we are closer to our goal. But *how* is each step executed? To see that, we have to treat each major step as a subgoal and break it down into substeps:

Send flowers to friend.
 Find flower delivery website.
  Open Web browser.
  Go to Web search page (e.g., Google).
  Type "flower delivery" into search-text box.
  Scan the first page of search results.
  Visit some of the listed links.
  Choose a flower delivery service.
 Order flowers to be delivered to friend.
  Review service's flower selection.
  Choose flowers.
  Specify delivery address and date.
  Pay for flowers and delivery.

After each substep is executed, we evaluate to see if it is getting us closer to the subgoal of which it is part. If we want to examine how a substep is executed and evaluated, we have to treat it as a sub-subgoal and break it into its component steps:

Send flowers to friend.
 Find flower delivery website.
  Open Web browser.
   • Click browser icon on taskbar, startup menu, or desktop.

Go to Web search page (e.g., Google).
- If search site isn't browser's starting page, choose it from favorites list.
- If search site is not on favorites list, type its address (e.g., "Google.com') into browser's address box.

Type "flower delivery" into search-text box.
- Set text-insertion point in search-text box.
- Type the text.
- Correct typo: "floowers" to "flowers."

Visit some of the resulting links.
- Move screen pointer to link.
- Click on link.
- Look at resulting Web page.

Choose a flower delivery service.
- Enter chosen service's URL into browser.

...

You get the idea. We could keep expanding down to the level of individual keystrokes and pointer movements, but we rarely need that level of detail to be able to understand the task well enough to design software to fit its steps and the goal–execute–evaluate cycle applied to each step.

How can software support users in carrying out the goal–execute–evaluate cycle? Any of these ways:

- ***Goal***. Provide clear paths—including initial steps—for the user goals that the software is intended to support.
- ***Execute***. Software concepts (objects and actions) should be based on task rather than implementation (see Chapter 11). Don't force users to figure out how the software's objects and actions map to those of the task. Provide a clear information scent at choice points to guide users to their goals. Don't make them choose actions that seem to take them away from their goal to achieve it.
- ***Evaluate***. Provide feedback and status information to show users their progress toward the goal. Allow users to back out of tasks that didn't take them toward their goal.

Fig. 8.9A shows a progress indicator that provides clear feedback about the user's progress through a series of steps. The indicator in Fig. 8.9B shows even more clearly where the user is in the sequence. Either design provides input for the Evaluate part of the action cycle. By the way, does Fig. 8.9A seem familiar? If so, it is because you saw it in Chapter 4 (see Fig. 4.17), and your brain recognized it.

FIGURE 8.9

Indicators showing progress through a series of steps. (A) Good; (B) best.

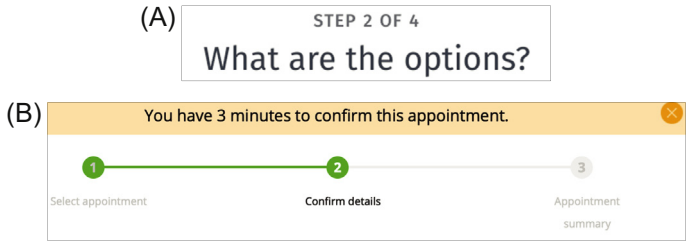


FIGURE 8.10

Progress indicators on mobile websites: (A) doodle.com; (B) kp.org.

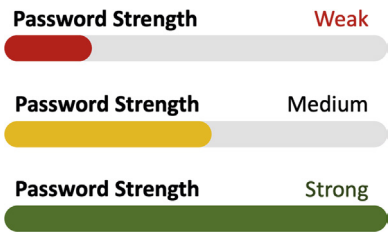


FIGURE 8.11

Password strength indicators show users how close they are to the goal.

Real examples of progress indicators can be seen on the websites of the Doodle scheduling service (doodle.com) and Kaiser Permanente (kp.org). Doodle's poll-creation function simply shows users what step number they are on out of the total number of steps (see Fig. 8.10A). That is better than no indication, but it is better to show the steps visually, as KP.org's appointment function does (see Fig. 8.10B).

Even when progress toward a goal is not measured by its location in a predefined series of steps, designers can provide feedback to help users see how close they are to the goal. Password strength indicators are a good example (see Fig. 8.11).

AFTER WE ACHIEVE A TASK'S PRIMARY GOAL, WE OFTEN FORGET CLEANUP STEPS

The goal–execute–evaluate cycle interacts strongly with short-term memory. This interaction makes perfect sense: short-term memory is really just the focus of attention at any given moment. Part of the focus of attention is our current goal. The rest of our attentional resources are directed toward obtaining the information needed to achieve our current goal. The focus shifts as tasks are executed and the current goal shifts from high-level goals to lower-level ones before returning to the next high-level goal.

Attention is a very scarce resource. Our brain does not waste it by keeping it focused on anything that is no longer important. Therefore, when we complete a task, the attentional resources focused on that task's main goal are freed to be refocused on other things that are now more important. The impression we get is that once we achieve a goal, everything related to it often immediately "falls out" of our short-term memory—that is, we forget about it.

One result is that people often forget loose ends of tasks. For example, people often forget to do these things:

- Turn car headlights off after arrival.
- Remove the last pages of documents from copiers and scanners.
- Turn stove burners and ovens off after use.
- Add closing parentheses and quotation marks after typing text passages.
- Turn off turn signals after completing turns.
- Take books they were reading on a flight with them when they exit the plane.
- Log out of public computers when finished using them.
- Set devices and software back to normal mode after putting them into a special mode.

These end-of-task short-term memory lapses are completely predictable and avoidable. When they happen to us, we call ourselves "absent-minded," but they are the result of how the brain works (or doesn't) combined with a lack of support from our devices.

To avoid such lapses, which are called "closure slips" (see Chapter 15), interactive systems can and should be designed to remind people that loose-end steps remain. In some cases, it may even be possible for the system to complete the task itself. For example:

- Cars already turn off turn signals after a turn.
- Cars should (and now do) turn off headlights automatically when the car is no longer in use or at least remind drivers that the lights are still on.
- Copiers and scanners should automatically eject all documents when tasks are finished or at least signal that a page has been left behind.

- Stoves should signal when a burner is left on with no pot present for longer than some suitable interval, and ovens should do likewise when left on with nothing in them.
- Computers should issue warnings if users try to power them down or put them to sleep before the computer has finished a background task (e.g., saving files or sending a document to a printer).
- Special software modes should revert to "normal" automatically, either by timing out—as some appliances do—or through the use of spring-loaded mode controls that must be physically held while in the nonnormal state and that revert to normal when released (Johnson, 1990).

Software designers should consider whether the tasks supported by a system they are designing have cleanup steps that users are likely to forget, and if so, they should design the system either to help users remember or eliminate the need for users to remember. See Chapter 15 for more about designing to decrease the likelihood of closure slips and other user errors.

IMPORTANT TAKEAWAYS

People do the following when using digital tools and services:

- Focus on our goals and pay little attention to our tools due to the limited capacity of working memory and attention. Design to let users do that. Design so your software fades into the background.
- Notice things more when they are related to our goals. We often miss events, changes, and objects that are right in front of us if they aren't relevant to our goals.
- Use external memory aids to keep track of what we are doing. Design software to show users what they have already done. For example, mark visited links, messages read, steps completed.
- Follow information "scent" toward our goal. When designing a digital product or service, analyze users' likely goals, and for each user goal, make sure every page or screen includes clues indicating which action will lead toward that goal.
- Prefer familiar paths. Users tend to stick to what they know because they usually prefer to run on autopilot. Design so users can, after some experience, fall into automatic habits.
- Achieve goals via a constant thought cycle: goal, execute, evaluate. Design to support this cycle.
- Often forget cleanup steps. User interfaces should remind users of anything left undone after the goal is achieved.