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 six 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 scant attention to any tools you use to restart the mower, just as you paid scant attention to the mower when your primary focus was the lawn. After you restart the mower and resume mowing the lawn, you probably won’t remember where you were in mowing the lawn, but the lawn itself shows you.
Other tasks—e.g., reading a document, measuring a table, counting goldfish in a fish tank—might not provide such a clear reminder of your interrupted task and your 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 Web sites 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, 2005). The title means: if your software or Web site makes me think about it, rather than what I am trying to do, you’ve lost me.

---

**WE USE EXTERNAL 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 these:

- **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.

- **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-term 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 can’t 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 applications do this by marking already-read versus unread messages, most Web sites do it by marking visited versus unvisited links, and many applications do it by marking completed steps of a multipart task (see Fig. 8.1).

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
We follow information “scent” toward our goal

Focusing our attention on 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 anything displaying the words “buy,” “flight,” “ticket,” or “reservation.” Other items that a designer or marketer might think 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 looking for bargains.

This tendency of people to notice only things on a computer display that match their goal, and the literal thinking that they exhibit when performing a task on a
computer has been called “following the scent of information toward the goal” (Chi, Pirolli, Chen, & Pitkow, 2001; Nielsen, 2003). Consider the ATM machine display shown in Figure 8.3. What is the first thing on the screen that gets your attention when you are given each of the goals listed?

You probably noticed that some of the listed goals 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 goal “Purchase traveler’s cheques” make you glance at “Request cheque book” because of the word they share?

The goal-seeking strategy of following 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 does lead users to their goals. To do that, designers need to understand the goals that users are likely to have 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.

For example, imagine that you want to cancel a reservation you made or a payment you scheduled. You tell the system to cancel it, and a confirmation dialog box appears asking if you really want to do that. How should the options be labeled? Given that people interpret words literally in following 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 dialog box from Marriott.com to one from Quicken.com, we see that Marriott.com’s labeling provides clearer scent than Quicken.com’s (see Fig. 8.4).

As a second example, imagine that you forgot that a certain document was already open, and you tried to open it again. The designers of Microsoft Excel did a better job than the designers of Microsoft Word did in anticipating this situation, understanding the goals you might have at this point, and presenting you with instructions and options that make it clear what to do (see Fig. 8.5).
We follow information “scent” toward our goal

FIGURE 8.4
Marriott’s cancellation confirmation (A) provides clearer scent than Quicken’s (B).

FIGURE 8.5
Microsoft Excel’s warning (A) when users try to open an already open file is clearer than Word’s (B).
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 is explained more fully in Chapter 10, exploring new paths is problem solving, which places a heavy load on attention and short-term memory. In contrast, taking familiar, well-learned routes can be done fairly automatically and does not consume 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, which 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 in order to think less.

This preference for familiar, relatively mindless paths has several design implications for interactive systems:

- **Sometimes mindlessness trumps keystrokes.** With software intended for casual use or infrequent use, such as bank ATM machines 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 keystrokes per task to matter much. On the other hand, in software that is used all day by highly trained users in intensive work environments, such as airline telephone reservation operators, every extra keystroke in a task adds high cumulative costs.

- **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 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 the keyboard accelerators for frequently used functions in the menu bar menus.
Our thought cycle: goal, execute, evaluate

Over many decades, scientists studying human behavior have found a cyclical pattern that seems to hold across a wide variety of activities:

- Form a **goal**, e.g., open a bank account, or eat a peach, or delete a word from a document
- Choose and **execute** actions to try to make progress toward the goal
- **Evaluate** whether the actions worked, i.e., whether the goal has been reached or is nearer than before
- Repeat until the goal is reached (or appears unreachable)

People cycle through this pattern constantly (Card, Moran, & Newell, 1983). 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 a larger task of writing a term paper, which is part of a higher-level task of passing a history course, which is part of a higher task of completing college, which is part of a higher-level goal of getting a good job, which we want 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 if they have moved the person closer to the goal.

- **Goal**: Buy airline ticket to Berlin, using your favorite travel Web site.
- **Step 1**: Go to travel Web site. You are still far from the goal.
- **Step 2**: Search for suitable flights. This is a very normal, predictable step at travel Web sites.
- **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 OK?
- **Step 7**: Print e-ticket. Goal achieved.
In the airline ticket example, to keep the example short, we didn’t get down 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 high-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 this task 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 Web site.
  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 Web site.
    Open Web browser.
    Go to Google Web search page.
    Type “flower delivery” into Google.
    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 down into its component steps.

Send flowers to friend.
  Find flower delivery Web site.
    Open Web browser.
      Click browser icon on taskbar, startup menu, or desktop.
    Go to Google Web search page.
      If Google isn’t browser’s starting page, choose Google from favorites list.
      If Google is not on favorites list, type “Google.com” into browser’s address box.
Type “flower delivery” into Google.
   Set text-insertion point in search 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 individual mouse 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 that is 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 the task rather than the 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 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 in order 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.

   An example of the “Evaluate” guideline—clear feedback about the user’s progress through a series of steps—is provided by ITN’s flight reservation system (see Fig. 8.6). By the way, does the figure seem familiar? If so, it is because you saw it in Chapter 5 (see Fig. 5.15B, page 63), and your brain recognized it.

---

**FIGURE 8.6**

ITN’s flight reservation system clearly shows users’ progress toward making a reservation.
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 what the focus of attention is 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, then back 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 information that is 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, i.e., 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, to prevent draining the battery
- Remove 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 into 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, 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 turn-signals OFF 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, which must be physically held in the non-normal state and 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 to eliminate the need for users to remember.