CHAPTER

Many Factors Affect Learning 11

Chapter 10 contrasted system one, the automatic processes our brain uses to carry out well-learned activities and make quick judgments, with system two, the conscious, highly monitored, controlled processes we use to solve novel problems, make rational choices, and perform calculations. Automatic processes (system one) consume little or no short-term memory (attention) resources and can operate in parallel with each other, while controlled processes (system two) place high demands on short-term memory and operate one at a time (Schneider and Shiffrin, 1977; Kahneman, 2011; Eagleman, 2012, 2015).

SIDEBAR: OUR BRAIN REWIRES ITSELF CONSTANTLY

How does the brain learn? Recent brain research has found that the brain adapts to new situations and environmental requirements constantly, mainly by rewiring itself: neurons that formerly fired independently become connected and fire in concert or in opposition, and neurons that formerly participated in one perception or behavior are reused for others. This is known as brain plasticity (Doidge, 2007).

It has been known for over 40 years that infants' brains are highly plastic: within months of birth, somewhat random collections of neurons develop into highly organized neural networks. However, the degree to which brains remain plastic (i.e., reorganizable) into adulthood was unknown until nuclear magnetic resonance imagery and similar brain observation methods became available.

One dramatic example of brain plasticity is the fact that blind people can be taught to "see" by connecting a video camera to an array of tactile stimulators resting on the person's back. The stimulators touch the person's back in patterns corresponding to images captured by the camera, vibrating for dark areas of the image but not for light areas. With training, participants in these studies can read, perceive scenes in three dimensions, and recognize objects. Initially, they perceive the stimulation as tactile patterns on their back, but after a while they report it as actually "seeing."

Brain plasticity is also exemplified by a new approach to stroke rehabilitation. People who have strokes sometimes lose the use of an arm or leg on one side of their body. Recovering use of the limb has traditionally been difficult—often impossible—and stroke victims often learn to compensate by ignoring their bad limbs and relying on their good ones. However, some stroke doctors have recently started placing patients' good limbs in casts to immobilize them, literally forcing patients to use their bad limbs. Results have been positive. Apparently, the brain reassigns different neurons to the bad arm, for example, allowing it to function again (Doidge, 2007).

The first time or even the first several times we perform an activity, we do it in a highly controlled and conscious manner, but with practice it becomes more automatic. Examples include peeling an apple, driving a car, juggling balls, riding a bicycle, reading, playing a musical instrument, and using an app on your mobile phone. Even an activity that might seem to require our attention, such as sorting good cherries from bad ones, can become automated to the point that we can do it as a background task with plenty of cognitive resources left over for having a conversation, watching the news on television, etc.

This progression from controlled to automatic raises an obvious question for designers of interactive applications, online services, and electronic appliances: How can we design them so that using them becomes automatic within a reasonable amount of time?

This chapter explains and demonstrates factors that affect how quickly people can learn to use interactive systems. To preview the factors, we learn faster when:

- practice is frequent, regular, and precise;
- operation is task focused, simple, and consistent;
- vocabulary is task focused, familiar, and consistent;
- risk is low.

WE LEARN FASTER WHEN WE PRACTICE FREQUENTLY, REGULARLY, AND PRECISELY

Obviously, practice facilitates learning. Here are some details about practice.

Frequency of practice

If people use an interactive system only rarely (e.g., once every few weeks or less), it is hard for them to remember from one time to the next the details of how to use it. However, if they use an interactive system frequently, familiarity develops quickly. Most user-interface designers are well aware of this, and therefore design applications, appliances, and online services differently depending on whether people will use them casually and rarely or intensively and often.

For example, automated teller machines (ATMs) for banks are designed under the assumption that people will not remember much from one usage to the next. They are designed to be simple and remind people what they do and how they work. ATMs present a short list of anticipated user goals (e.g., withdraw cash, deposit funds, transfer funds), then guide users through the steps of the selected task. Airline and hotel booking websites are similarly task oriented: "Tell me your goal and I'll guide you to it." In contrast, document editing applications, electronic calendars, smartphone texting apps, air traffic control systems, and online accounting services are designed with the understanding that people will be using them daily—perhaps even minute-by-minute—and will quickly learn and remember details about how to use them.

Regularity of practice

How long does it take for an activity to become an automatic habit? Lally and her colleagues conducted a study to measure this (Lally et al., 2010). They asked about 100 volunteer participants to choose a new eating, drinking, or physical activity to do every day for at least 2 months. They monitored the participants and measured how long it took for the new behavior to become automatic—that is, to be executed without conscious thought or effort.

They found that forming automatic habits takes from 18 to 254 days, with more complex activities taking more time. They also found that habits formed faster if practiced regularly (e.g., daily). Skipping a practice session now and then didn't make much difference, but skipping many practice sessions significantly slowed participants' progress toward habit formation.

Bottom line: If you want the use of your software to become habitual and automatic for users, design it so as to encourage people to use it regularly.

Precision of practice

Unorganized collections of neurons are noisy; they fire randomly, not in an organized way. When people practice an activity repeatedly, the brain organizes itself to support and control that activity: networks of neurons get "trained up" to fire in concert. Their firing becomes more systematic and less "noisy." This is true regardless of whether the activity is perceptual like identifying a word, motor like skiing, cognitive like counting numbers, or a combination like singing a song.

The more carefully and precisely a person practices an activity, the more systematic and predictable the activation of the corresponding neural network. If a person practices an activity carelessly and sloppily, the supporting neural networks remain somewhat disorganized (i.e., noisy), and the execution of the activity will remain sloppy and imprecise (Doidge, 2007).

Stated simply: Practicing the same activity imprecisely just strengthens the imprecision, because the neural networks controlling it remain noisy. To train the neural networks to make the activity precise, one must practice precisely and carefully, even if that requires practicing slowly at first or breaking down the activity into parts.

If efficiency and precision are important for a task, design the supporting software and its documentation to (1) help people be precise (e.g., by providing guides and grids) and (2) encourage people to use it purposefully and carefully rather than absentmindedly and sloppily. The following section explains how providing users with a clear conceptual model can support (2). Consistency of keystrokes and gestures, discussed later in this chapter, is also important.

WE LEARN FASTER WHEN OPERATION IS TASK FOCUSED, SIMPLE, CONSISTENT, AND PREDICTABLE

When we use a tool—whether it is computer based or not—to do a task, we have to translate what we want to do into the operations provided by the tool. Some examples:

- You are an astronomer. You want to point your telescope at the star Alpha Centauri. Most telescopes don't let you specify a star to observe. Instead, you have to translate that goal into how the telescope's positioning controls operate—in terms of a vertical angle (azimuth) and horizontal angle, or perhaps even the *difference* between where the telescope is pointing now and where you want it to point.
- You want to call someone who isn't in your phone's contact list. To call this person, you have to get the person's telephone number and enter it into the phone.

- You want to create an organizational chart for your company using a generic drawing program. To indicate organizations, suborganizations, and their managers, you have to draw boxes, label them with organization and manager names, and connect them with lines.
- You want to make a two-sided copy of a two-sided document, but the copier only makes one-sided copies. To make your copy, you must first copy one side of each document sheet, take those copies and put them back into the copier's paper tray upside down, and then copy the other side of each document sheet.
- You want to ask Alexa a complicated question, such as *What is the earliest known evidence of dental work in humans*? You have to spend a minute thinking about how to best ask the question, because you don't want to waste time sifting through irrelevant results—e.g., the first dental school (1828) or first use of dental bridges (~700 BCE). It isn't easy, because you don't know what Alexa knows; that is, you have a weak mental model of Alexa's capabilities.

Cognitive psychologists call the gap between what a tool user wants and the operations the tool provides "the gulf of execution" (Norman and Draper, 1986). A person using a tool must expend cognitive effort to translate what he or she wants into a plan based on the tool's available operations and vice versa. That cognitive effort pulls the person's attention away from the task and refocuses it on the requirements of the tool. The smaller the gulf between the operations that a tool provides and what its users want to do, the less the users need to think about the tool and the more they can concentrate on their task. As a result, the tool becomes automatic more quickly.

The way to reduce the gulf is to design the tool to provide operations that match what users are trying to do. To build on the preceding examples:

- A telescope's control system could have a database of celestial objects so users could simply indicate which object they want to observe, perhaps by pointing to it on a display.
- Telephones with contact lists allow users to simply specify the person or organization they want to call rather than having to translate that to a number first.
- A special-purpose organizational chart–editing application would let users simply enter the names of organizations and managers, freeing users from having to create boxes and connect them.
- A copier that can make double-sided copies allows users who want such copies to simply choose that option on the copier's control panel.
- An AI-powered digital assistant (Alexa, Siri, Google Assistant, Cortana, etc.) could initiate a conversation with you, asking you follow-up questions to tease out what you really want to know, and gradually refine the answers it gives. To do this it would have to maintain the context of the conversation over several minutes.

To design software, services, and appliances to provide operations matching users' goals and tasks, designers must thoroughly understand the user goals and tasks the tool is intended to support. Gaining that understanding requires three steps:

1. Perform a task analysis.
2. Design a task-focused conceptual model consisting mainly of an objects/actions analysis.
3. Design a user interface based strictly on the task analysis and conceptual model.

Task analysis

Describing in detail how to analyze users' goals and tasks is beyond the scope of this book. Entire chapters—even whole books—have been written about it (Beyer and Holtzblatt, 1997; Hackos and Redish, 1998; Johnson, 2007). For now, it is enough to say that a good task analysis answers these questions:

- What goals do users want to achieve by using the application?
- What set of human tasks is the application intended to support?
- Which tasks are common, and which are rare?
- Which tasks are most important, and which are least important?
- What are the steps of each task?
- What is the result and output of each task?
- Where does the information for each task come from?
- How is the information that results from each task used?
- Which people do which tasks?
- What tools are used to do each task?
- What problems do people have performing each task? What sorts of mistakes are common? What causes them? How damaging are mistakes?
- What terminology is used by people who do these tasks?
- What communication with other people is required to do the tasks?
- How are different tasks related?

Conceptual model

Once these questions are answered (by observing and/or interviewing people who do the tasks that the tool supports), the next step is *not* to start sketching possible user interfaces. Rather, the next step is to design a conceptual model for the tool that focuses on the users' tasks and goals (Johnson and Henderson, 2002, 2011, 2013).

A conceptual model of an app or online service is the one that the designers want users to understand. By using the system, talking with other users, and reading the documentation, users construct a model in their minds—a *mental model*—of how to use it. Hopefully, the mental model that users build is close to the one the designers intended. That is more likely when designers explicitly design a clear *conceptual model* as a key part of the development process and then base the user-interface design on that.

A conceptual model describes abstractly what tasks a user can perform with the system and what concepts they must know about to complete the tasks. The concepts should be those that came out of a task analysis so the conceptual model is focused on the task domain. It should include few—ideally no—concepts for users to master outside of the target task domain. The more direct the mapping between the application's concepts and those of the tasks it is intended to support, the less translating users will have to do and the easier the tool will be to learn.

In addition to being focused on users' tasks, a conceptual model should be as simple as possible. Simple means few concepts. The fewer concepts a model has for users to master, the better, as long as it provides the required functionality. Less is more, as long as what is there fits with users' goals and tasks. For example:

- In a to-do list application, do users need to be able to assign priorities of 1–10 to items, or are two priority levels, low and high, enough?
- Does a ticket machine in a train station need to be able to offer tickets for train routes other than the routes this station is on?
- Does an online shopping site need to give customers both a *wish list* and a *shopping cart*? If not, combine them into one concept.
- Does a voice-controlled personal assistant (e.g., Siri, Alexa, Google Assistant, Cortana) present a conceptual model that is predictable and easily understood, or is its operation mysterious to users—e.g., it tries but fails to be like a person—preventing users from predicting effectively what it can and cannot do? Unfortunately, today's voice-controlled personal assistants are mostly like the latter (Budiu, 2018; Budiu and Laubheimer, 2018; Pearl, 2018a, 2018b).

In most development efforts, there is pressure to add extra functionality in case a user might want it. Resist such pressure unless there is considerable evidence that a significant number of potential customers and users would use the extra functionality. Why? Because every extra concept increases the complexity of the software. It is one more thing users have to learn. But in fact it is not just *one* more thing. Each concept in an application interacts with most of the other concepts, and those interactions result in more complexity. Therefore, as concepts are added to an application, the application's complexity grows not just linearly but multiplicatively.

For a more comprehensive discussion of conceptual models, including some of the difficulties that arise in trying to keep them simple and task focused while providing the required functionality and flexibility, see Johnson and Henderson (2002, 2011, 2013).

After you have designed a conceptual model that is task focused, as simple as possible, and as consistent as possible, you can design a user interface for it that minimizes the time and experience required for use of the application to become automatic.

SIDEBAR: EXCESS COMPLEXITY DUE TO CONCEPTS BEING TOO SIMILAR

Some software applications are too complex because they have concepts with overlapping functionality. Here are two real examples:

- Example 1: MacOS has three different notes apps: a Stickies note app for the desktop, a Notes app for the desktop, and a sticky notes app in the Dashboard (see Fig. 11.1). Mac users can easily lose track of which one they wrote a note into. I have even seen Mac users who didn't realize there are multiple notes apps and could not find a note they wrote because they were looking in the wrong app.
- Example 2: A real estate company developed a website for people seeking to buy a home. As a first step in searching for a home, users had to indicate how they wanted to search:

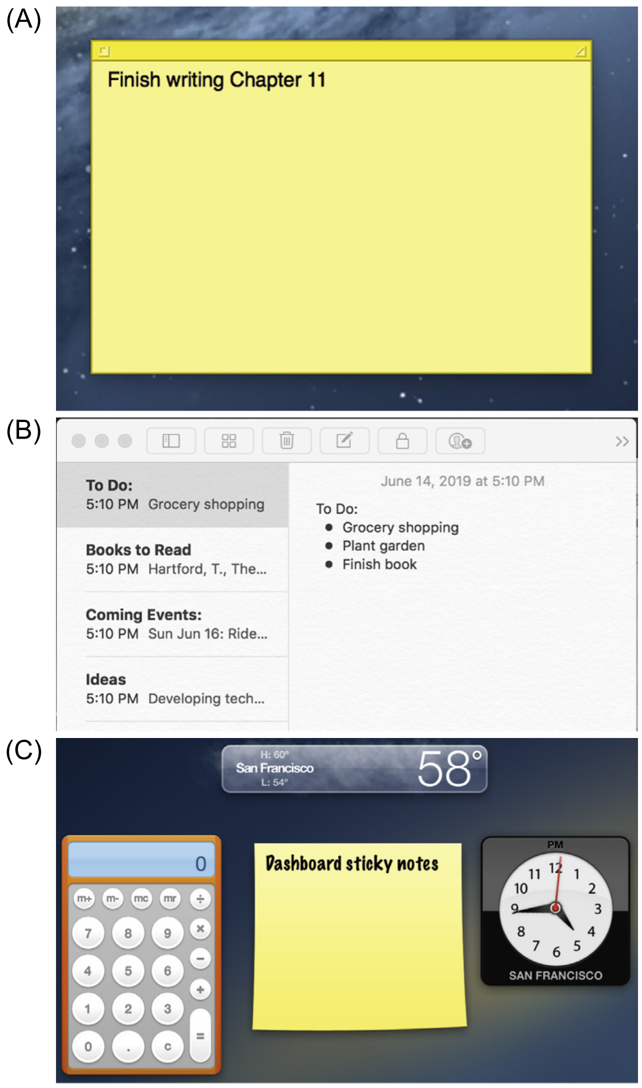
  - "by location": enter the desired postal zip code, or
  - "by map": point to a location on a map.

  A usability test found that many users did not think of those as different ways of finding a home. To them, both methods were by location; they only differed in how the location was specified. After the test, the two options were merged into one, with both a map and a zip code entry field.

Consistency

The *consistency* of an interactive system strongly affects how quickly its users progress from controlled, consciously monitored, slow operation to automatic, unmonitored, faster operation (Schneider and Shiffrin, 1977). The more predictable the operation of a system's different functions, the more consistent it is. In a highly consistent system, the operation of a particular function is predictable from its *type*, so people quickly learn how everything in the system works, and its use quickly becomes habitual. In an inconsistent system, users cannot predict how its different functions work, so they must learn each one anew, which slows their learning of the overall system and forces them to keep using it in a nonautomatic (i.e., controlled), attention-consuming way. In psychological terms, inconsistent systems place a high *cognitive load* on users.

The designer's goal is to devise a conceptual model that is task focused, as simple as possible, and as consistent as possible (Johnson and Henderson, 2002, 2011, 2013).



Finish writing Chapter 11

To Do:
5:10 PM Grocery shopping

Books to Read
5:10 PM Hartford, T., The...

Coming Events:
5:10 PM Sun Jun 16: Ride...

Ideas
5:10 PM Developing tech...

June 14, 2019 at 5:10 PM

To Do:
- Grocery shopping
- Plant garden
- Finish book

Dashboard sticky notes

FIGURE 11.1

Three notes apps in MacOS. (A) Desktop stickies, (B) Desktop notes, (C) Dashboard sticky notes.

From such a model, one can design a user interface for it that minimizes the time and experience required for use of the application to become automatic.

Interactive systems can be consistent or inconsistent on at least two different levels: conceptual and keystroke.

Consistency at the conceptual level is determined by the mapping between the objects, actions, and attributes of the conceptual model (see earlier). Do most objects in the system have the same actions and attributes, or not?

Consistency at the keystroke level is determined by the mapping between the conceptual actions and the physical movements or vocal utterances required to execute them. Are all conceptual actions of each type initiated and controlled by the same physical actions, or not?

Conceptual consistency

As explained above, the *number of concepts* an application, website, or digital device requires users to understand affects its complexity—how easy it is to learn. In general, the more concepts a system has, the harder it will be to learn.

Another factor that influences how easy a digital system will be to learn and remember is the *consistency* of its conceptual model. If every conceptual object has different things you can do with it (actions) and different settings (attributes), its users will find the system hard to learn and remember. On the other hand, if all types of conceptual objects in the system have mostly the same actions and attributes, then once you learn how to operate one type of object, you already know how to operate other types.

For example, imagine that your company is developing a drawing app. The app provides several objects (shapes) that users can add to drawings. Each object-type was implemented by a different programmer, who did not communicate with the others. The only actions all object-types share is that they can all be created and deleted (see Table 11.1). Users can set the line weight of rectangles, squares, lines, and text boxes but not of triangles, circles, and ovals. Once placed on the screen, all shapes can be

Table 11.1 Object-Action Matrix for a Drawing App With an Inconsistent Conceptual Model

Object	Create	Delete	Set Line Weight	Set Line Color	Set Fill Color	Move	Rotate
Rectangle	X	X	X	X	X	X	
Square	X	X	X			X	
Triangle	X	X		X		X	X
Circle	X	X		X		X	
Oval	X	X			X	X	
Line	X	X	X	X		X	X
Text box	X	X	X				

moved, but text boxes cannot. Only triangles and lines can be rotated. Only rectangles and ovals have a user-settable fill color. Et cetera.

It is pretty clear that the inconsistency makes this drawing app hard to learn. Users would have trouble remembering what actions apply to each type of object. Even if there were a document showing what can be done with each object, users would have trouble remembering. Stated in psychological terms, the cognitive load for this app can be approximated as $N_{objects}$ x $N_{actions}$, a multiplicative function, meaning that the app's complexity increases very quickly as functionality (more objects and actions) is added (Rosenberg, 2020).

A totally consistent design would be one in which all actions apply to all objects (see Table 11.2). With that app, once users learn how to work with one type of object—e.g., rectangles—they already know how to work with other types of objects. The cognitive load for the consistent app can be approximated as $N_{objects} + N_{actions}$, an additive function, meaning that the app's complexity increases much more slowly as more functionality is added (Rosenberg, 2020).

Table 11.2 Object-Action Matrix for a Drawing App with a Highly Consistent Conceptual Model

Object	Create	Delete	Set Line Weight	Set Line Color	Set Fill Color	Move	Rotate
Rectangle	X	X	X	X	X	X	X
Square	X	X	X	X	X	X	X
Triangle	X	X	X	X	X	X	X
Circle	X	X	X	X	X	X	X
Oval	X	X	X	X	X	X	X
Line	X	X	X	X	X	X	X
Text box	X	X	X	X	X	X	X

The designers could simplify this app's conceptual model even further by removing the separate text box object and adding an action on all shapes: "add a text container."

The drawing-app example may seem contrived, but many real apps and websites have inconsistent conceptual models. Such inconsistencies are common in large enterprise software systems that contain legacy subsystems—components and functionality retained from prior systems. For example, if a personnel system originally managed only full-time and part-time employees but later was extended to also manage contractors and interns, there would probably be significant differences in what users could do with the different types of employees, at least until the legacy software was scrapped and rewritten.

As an example of past conceptual inconsistency that has largely been reduced, early airline reservation websites provided two types of flights: those paid for with money and those paid for with frequent-flier miles. The main flight-search page could book only flights paid for with money. To book a flight with miles, you had to sign in and go to the frequent-flier section of the site. Nowadays, most flight-booking websites let you search for all flights the same way, and whether you will pay with money or miles is just a setting.

Keystroke consistency

After a designer has developed a conceptual model for a digital product or service, it is time to design the details of the user interface. At that point, a second type of consistency becomes important: keystroke-level consistency.

Keystroke-level consistency is *at least* as important as conceptual consistency in determining how quickly the operation of an interactive system becomes automatic. The goal is to foster the growth of what is often called "muscle memory," meaning motor habits. A system inconsistent at the keystroke level does not let people quickly fall into muscle-memory motor habits. Rather, it forces them to remain consciously aware of, and guess about, which keystrokes to use in each context even when those gestures in different contexts differ only slightly. In addition, it makes it likely that people will make errors—accidently do something they did not intend (see Chapter 15).

Achieving keystroke-level consistency requires standardizing the physical actions for all activities of the same type. An example of a type of activity is editing text. Keystroke-level consistency for text editing requires keystrokes, pointer movements, and gestures to be the same regardless of the context in which text is being edited (documents, form fields, filenames, etc.). Keystroke-level consistency is also desirable for other types of activities, such as opening documents, following links, choosing from a menu, choosing from a displayed set of options, and clicking buttons.

Consider three alternative designs for the keyboard shortcuts for Cut and Paste in a hypothetical multimedia document editor. The document editor supports the creation of documents containing text, sketches, tables, images, and videos. In design A, Cut and Paste have the same two keyboard shortcuts regardless of what type of content is being edited. In design B, the keyboard shortcuts for Cut and Paste are different for every type of content. In design C, all types of content *except* videos have the same Cut and Paste keyboard shortcuts (see Table 11.3).

The first question is: Which of these designs is easiest to learn? It is pretty clear that design A is the easiest.

The second question is: Which design is hardest to learn? That is a tougher question. It is tempting to say design B because it seems to be the least consistent of the three. However, the answer really depends on what we mean by "hardest to learn." If we mean "the design for which users will require the most time to become productive," that is certainly design (B). It will take most users a long time to learn all the different Cut and Paste keyboard shortcuts for the different types of content. But people are remarkably adaptable if sufficiently motivated—they can learn amazingly arbitrary things if, for example, using the software is required for their job. Eventually—maybe in a month—users would be comfortable and even quick with design (B). In contrast, users of design C would begin to be productive in about the same short time as users of design (A), probably a matter of minutes.

Table 11.3 Which Keyboard-Shortcut Design Will Be Easiest/Hardest to Learn and Remember?

Object	Document Editor Keyboard Shortcuts: Alternative Designs					
	Design A		Design B		Design C	
	Cut	Paste	Cut	Paste	Cut	Paste
Text	CTRL-X	CTRL-V	CTRL-X	CTRL-V	CTRL-X	CTRL-V
Sketch	CTRL-X	CTRL-V	CTRL-C	CTRL-P	CTRL-X	CTRL-V
Table	CTRL-X	CTRL-V	CTRL-Z	CTRL-Y	CTRL-X	CTRL-V
Image	CTRL-X	CTRL-V	CTRL-M	CTRL-N	CTRL-X	CTRL-V
Video	CTRL-X	CTRL-V	CTRL-Q	CTRL-R	CTRL-E	CTRL-R

However, if we interpret "hardest to learn" as meaning "the design for which users will take the longest to be error-free," that is design C. The shortcut keys for Cut and Paste are the same for every type of document content except videos. Although users of design C will be productive quickly, they will continue to make the error of trying to use CTRL-X and CTRL-V with videos for at least several months—perhaps forever.

Consistency is extremely important for learning hand–eye coordination activities such as scrolling, panning, and zooming a display, especially on touch-controlled screens. If those actions require users to make different gestures in different contexts (e.g., for different apps), the corresponding neural networks in users' brains will remain noisy, preventing users from ever being able to pan and scroll automatically (i.e., without conscious thought).

For example, on Apple Macintosh computers running MacOS X,[1] panning and scrolling is usually accomplished by dragging two fingers across the trackpad in the desired direction, and zooming is controlled by spreading or pinching two fingers. But what if a Mac user is using Google Maps? In the column on the left that lists search results (see Fig. 11.2), scrolling/panning the list uses the standard MacOS X gesture of sliding two fingers up or down, and text is zoomed by spreading or pinching two fingers. But over the map itself—oops, sorry: dragging two fingers there doesn't *pan* it, it *zooms* it. Panning the map requires clicking the trackpad down with one finger and dragging it. And spreading or pinching two fingers over the map doesn't zoom it, it zooms the entire contents of the browser window. Needless to say, such inconsistencies effectively block scrolling, panning, and zooming from becoming automatic for users.

A common way that developers promote keystroke-level consistency is to follow look-and-feel standards. Such standards can be presented in style guides or be built into common user-interface construction tools and component sets. Style guides exist for the entire industry and separately for desktop software (Apple Computer, 2020; Microsoft Corporation, 2018) and web design (Koyani et al., 2006). Ideally, companies

[1]As this was being written, the current version of MacOS X was 10.15 (Catalina).

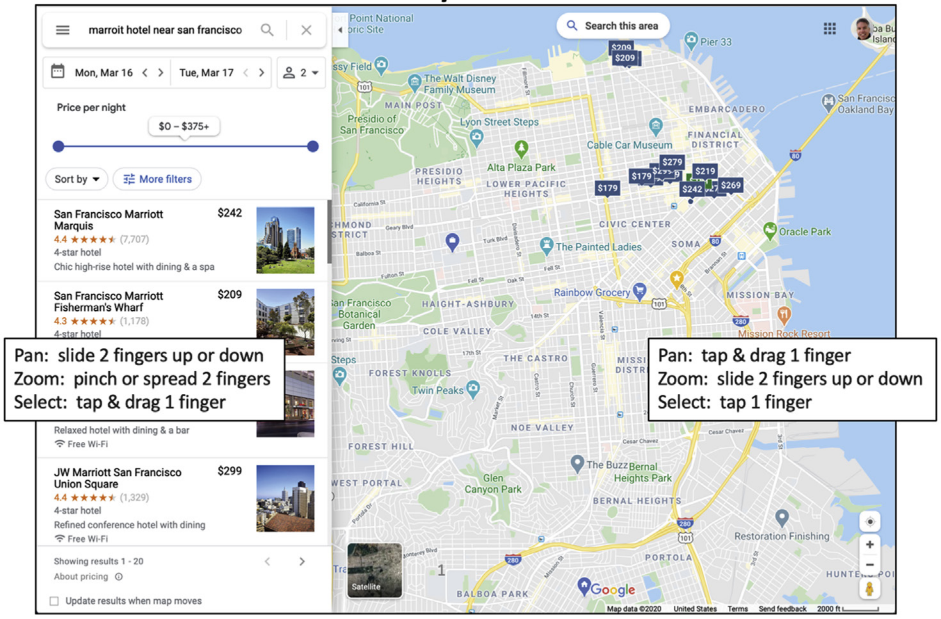

marroit hotel near san francisco
Mon, Mar 16
Tue, Mar 17
Price per night
$0 – $375+
Sort by
More filters
San Francisco Marriott Marquis
$242
San Francisco Marriott Fisherman's Wharf
$209
JW Marriott San Francisco Union Square
$299
Showing results 1 - 20
Update results when map moves
Search this area
Pan: slide 2 fingers up or down
Zoom: pinch or spread 2 fingers
Select: tap & drag 1 finger
Pan: tap & drag 1 finger
Zoom: slide 2 fingers up or down
Select: tap 1 finger

FIGURE 11.2

Google Maps (2020): Inconsistent gestures block scrolling, panning, and zooming from becoming automatic.

also have internal style guides that augment industry style guides and define a look and feel for their own products.

However conventions are encapsulated, designers should stick to conventions at the keystroke level while perhaps innovating at the conceptual and task levels. We as designers really don't want our software's users to have to keep thinking about their keystroke-level actions as they work, and users don't want to think about them either.

Predictability

If a digital system is predictable, people can learn to use it more quickly. For example, suppose you ask a map app for "Italian restaurants" and it shows you Italian restaurants within a short distance of you. An hour later you ask it the same thing and it shows Italian restaurants throughout your city. Because you can't predict how this map app will interpret your query and scope its results, it will probably take you a fair amount of time to learn to use the app effectively. In psychological terms, the app's unpredictability makes it hard for you to build a predictive *mental model* of how it works. You might even abandon the app and look for a better one. In contrast, if the app consistently showed Italian restaurants near you, or consistently showed Italian restaurants throughout your city, you could more easily build a mental model predicting how it would respond to your queries so you could more quickly learn to use it.

Designers of an app, online service, or digital assistant should help users develop a predictive *mental* model by basing their user-interface design on a *conceptual* model that is not only task focused, simple, and consistent but also *predictable*. Users should be able to predict what the app will do; e.g., "If I do ***X***, the app will do ***Z***." If they cannot, the app will seem mysterious and will be hard for people to learn to use effectively. If they have a choice, they might choose not to use it.

Current-day apps and online services based on artificial intelligence technology—e.g., digital assistants, smart speakers, recommendation systems—can do amazing and useful things, but too often they are unpredictable and mysterious (Budiu, 2018). As a result, many people don't trust them, or they even avoid them. This will have to change if AI technology is to become widely accepted. AI researchers must figure out how to make AI-based systems more predictable rather than functioning as inscrutable "black boxes."

WE LEARN FASTER WHEN VOCABULARY IS TASK FOCUSED, FAMILIAR, AND CONSISTENT

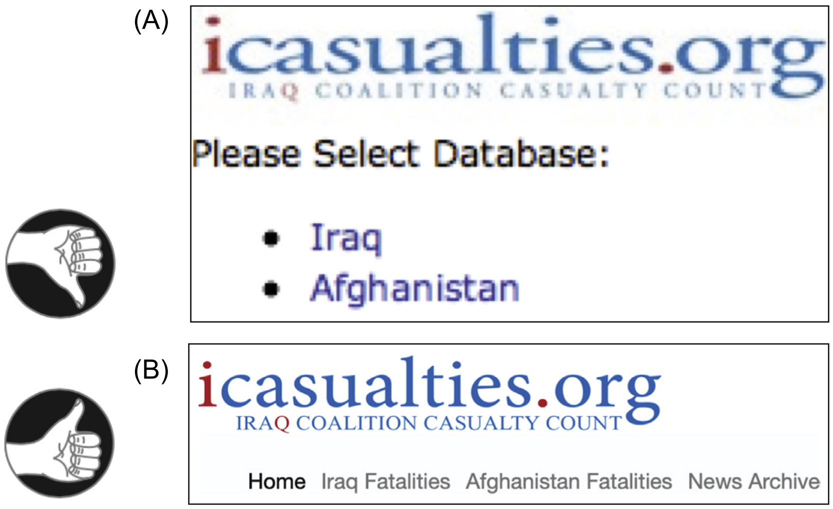
Ensuring that an application, web service, or appliance exposes a small, consistent, and task-appropriate set of concepts to its users is a big first step, but it is not enough to minimize the time it takes for people to learn an interactive system. You also have to make sure that the *vocabulary*—what concepts are called—fits the task, is familiar, and is consistent.

Terminology should be task focused

Just as the user-visible concepts in an interactive system should be task focused, so should the *names* for the concepts. Usually, task-focused terms for concepts emerge from the interviews and observations of users that designers conduct as part of the task analysis. Occasionally, software needs to expose a concept that is new to users; the challenge for a designer is keeping such concepts and their names focused on the task, not on the technology.

Here are examples of interactive software systems using terminology that is not task focused:

- A company developed a desktop software application for performing investment transactions. The application let users create and save templates for common transactions. It gave users the option of saving templates either on their own PC or on a network server. Templates stored on the PC were private; templates stored on the server were accessible to other people. The developers used "database" for templates on the server because they were kept in a database. They used "local" for templates on the users' own PCs because that is what "local" meant to them. Terms that would be more task focused are "shared" or "public" instead of "database," and "private" instead of "local."
- iCasualties.org provides up-to-date tallies of the number of coalition military personnel killed or injured in the Iraq and Afghanistan wars. Formerly its homepage started by asking site visitors to select a "database" (see Fig. 11.3A). However, visitors to this


(A)

icasualties.org
IRAQ COALITION CASUALTY COUNT

Please Select Database:

- Iraq
- Afghanistan

(B)

icasualties.org
IRAQ COALITION CASUALTY COUNT

Home Iraq Fatalities Afghanistan Fatalities News Archive

FIGURE 11.3

iCasualties.org in 2009 (A) used a non-task-focused term: "database." In 2020 (B), that was fixed.

site don't care or need to know that the website's data is stored in multiple databases. Task-focused instructions would ask them to select a country in which there is an ongoing conflict, not a "database." Their 2020 site corrected the error (see Fig. 11.3B).

Terminology should be familiar

To reduce the time it takes for people to master your application, website, or appliance so that using it becomes automatic or nearly so; don't force them to learn a whole new vocabulary. Chapter 4 explained that familiar words are easier to read and understand because they can be recognized automatically. Unfamiliar words cause people to use more conscious decoding methods, which consumes scarce short-term memory resources and thereby lowers comprehension.

Unfortunately, many computer-based products and services present users with unfamiliar terms from computer engineering—often called "geek speak"—and require them to master those terms (see Fig. 11.4). Why? Operating a stove does not require us to master terminology about the pressure and chemical composition of natural gas or terminology about the production and delivery of electricity. Why should shopping on the Web, sharing photographs, or checking email require us to learn "geek speak" such as "USB," "TIFF," or "broadband"? But in many cases, it does.

The following are examples of interactive software systems using unfamiliar terminology:

- A development team was designing a video-on-demand system for schoolteachers to use in classrooms. The purpose of the system was to allow teachers to

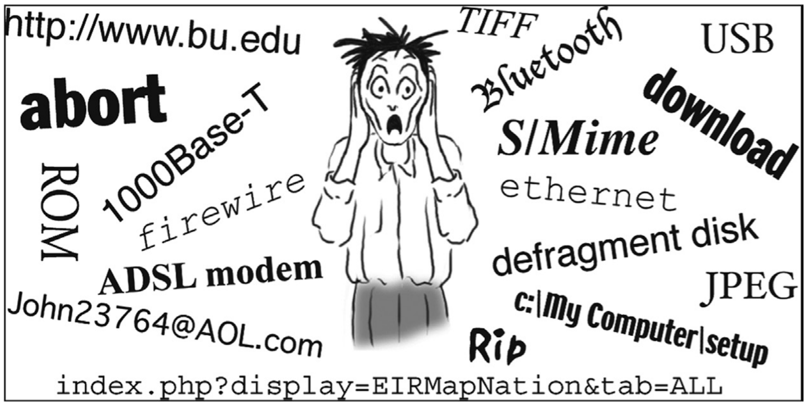
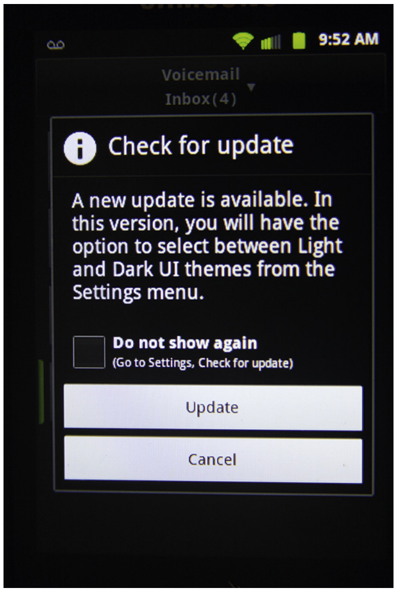


FIGURE 11.4

Unfamiliar computer jargon (aka "geek speak") slows learning and frustrates users.

find videos offered by their school district, download them, and show them in their classrooms. The developers' initial plan was to organize the videos into a hierarchy of "categories" and "subcategories." Interviews with teachers showed, however, that they use the terms "subject" and "unit" to organize instructional content, including videos. If the system used the developers' terminology, teachers who used it would have to *learn* that "category" meant "subject" and "subcategory" meant "unit," making the system harder to master.

- SPRINT, a mobile-phone service provider, sends announcements of software updates to customers' phones. These announcements often indicate new features that the update includes. One SPRINT update announcement offered "the option to select between Light and Dark UI themes from the Settings menu" (see Fig. 11.5). Most consumers won't know that UI is an abbreviation for "user interface." Even if they know that, they probably won't know what a user-interface *theme* is, as that is a technical term used mainly by software designers and developers.
- Google's Android operating system for mobile devices sometimes displays warnings using obscure technical jargon or acronyms, such as the one shown in Fig. 11.6. Very few people who have Android phones would understand that message.
- An example from American Airlines' AAdvantage website shows that unless designers carefully review and approve *all* text displayed by consumer-facing apps and websites, technical jargon can slip in. Each product category has a pop-up tool tip that explains what clicking on the category does (see Fig. 11.7). But "dialogue" is a software design jargon term. To most English speakers, it means a conversation between two people.



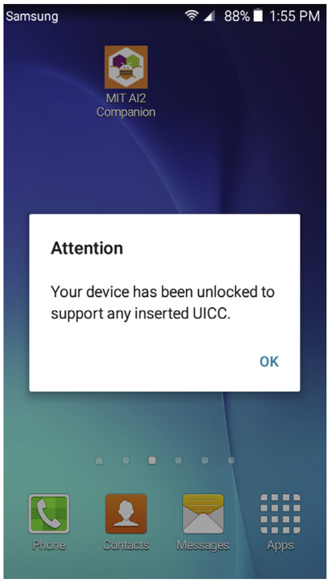
Check for update

A new update is available. In this version, you will have the option to select between Light and Dark UI themes from the Settings menu.

Do not show again
(Go to Settings, Check for update)

FIGURE 11.5

An update message from the SPRINT mobile-phone service uses the computer-jargon term “UI themes.”



Attention

Your device has been unlocked to support any inserted UICC.

FIGURE 11.6

An informational message that sometimes appears on Android mobile phones will not be understood by the majority of phone users.

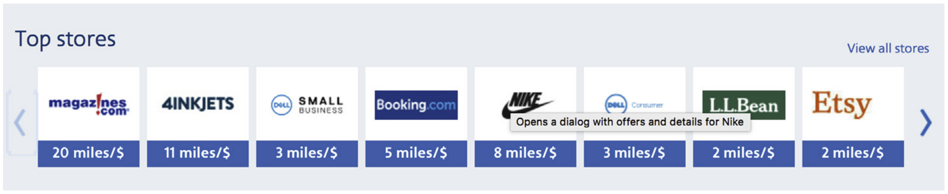


FIGURE 11.7

AdvantageShopping.com displays a pop-up tool-tip message that includes the technical jargon term “dialogue.”

Terminology should be consistent

People want to focus their cognitive resources on their own goals and tasks, not on the software they are using. They just want to accomplish their goal, whatever it is. They are not interested in the software. They interpret what the system presents only superficially and very literally. Their limited attentional resources are so focused on their goal that if they are looking for a Search function but it is labeled “Query” on the current screen or page, they may miss it. Therefore, the terminology in an interactive system should be designed for maximum consistency.

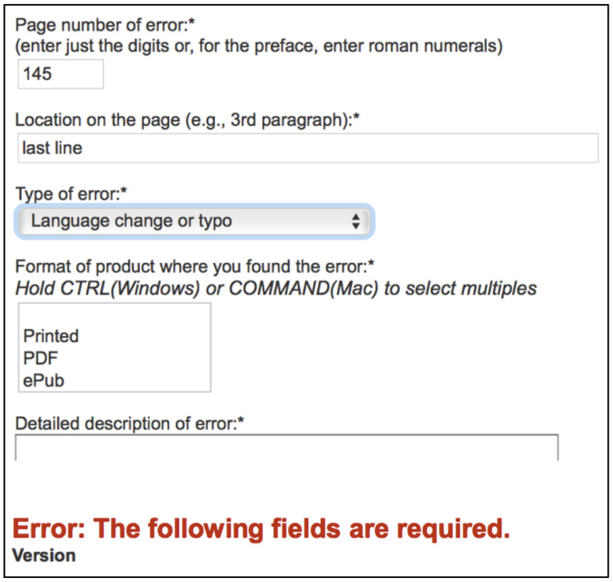
The terminology used in an interactive system is consistent when each concept has *one and only one* name. Caroline Jarrett, an authority on user-interface and forms design, provides this rule:

Same name, same thing; different name, different thing—FormsThatWork.com.

This means that terms and concepts should map strictly 1:1. Never use *different* terms for the *same* concept, or the *same* term for *different* concepts. Even terms that are ambiguous in the real world should mean only one thing in the system. Otherwise, the system will be harder to learn and remember.

An example of different terms for the same concepts is provided by O’Reilly’s online form for reporting errata in books O’Reilly has published. If a reader submits the form without selecting a product format (“Printed,” “PDF,” or “ePub”), an error message appears telling the user that the “Version” field is required. The form labels the field “Format of product where you found the error,” but the error message labels it “Version” (see Fig. 11.8). This will surely confuse users.

WordPress.com provides an example of the opposite sort of ambiguity: the *same* term for *different* concepts—also called *overloading* a term—and how to fix it. WordPress.com’s 2009 site for administering a blog had a Dashboard consisting of monitoring and administrative functions organized into several pages. The problem was that the first administrative function page in the Dashboard was also called the “Dashboard,” so the same name referred to both the whole Dashboard and one page of it (see Fig. 11.9A). This surely confused many new bloggers just learning to use WordPress.com; they had



Page number of error:*
(enter just the digits or, for the preface, enter roman numerals)
145

Location on the page (e.g., 3rd paragraph):*
last line

Type of error:*
Language change or typo

Format of product where you found the error:*
Hold CTRL(Windows) or COMMAND(Mac) to select multiples
Printed
PDF
ePub

Detailed description of error:*

Error: The following fields are required.
Version

FIGURE 11.8

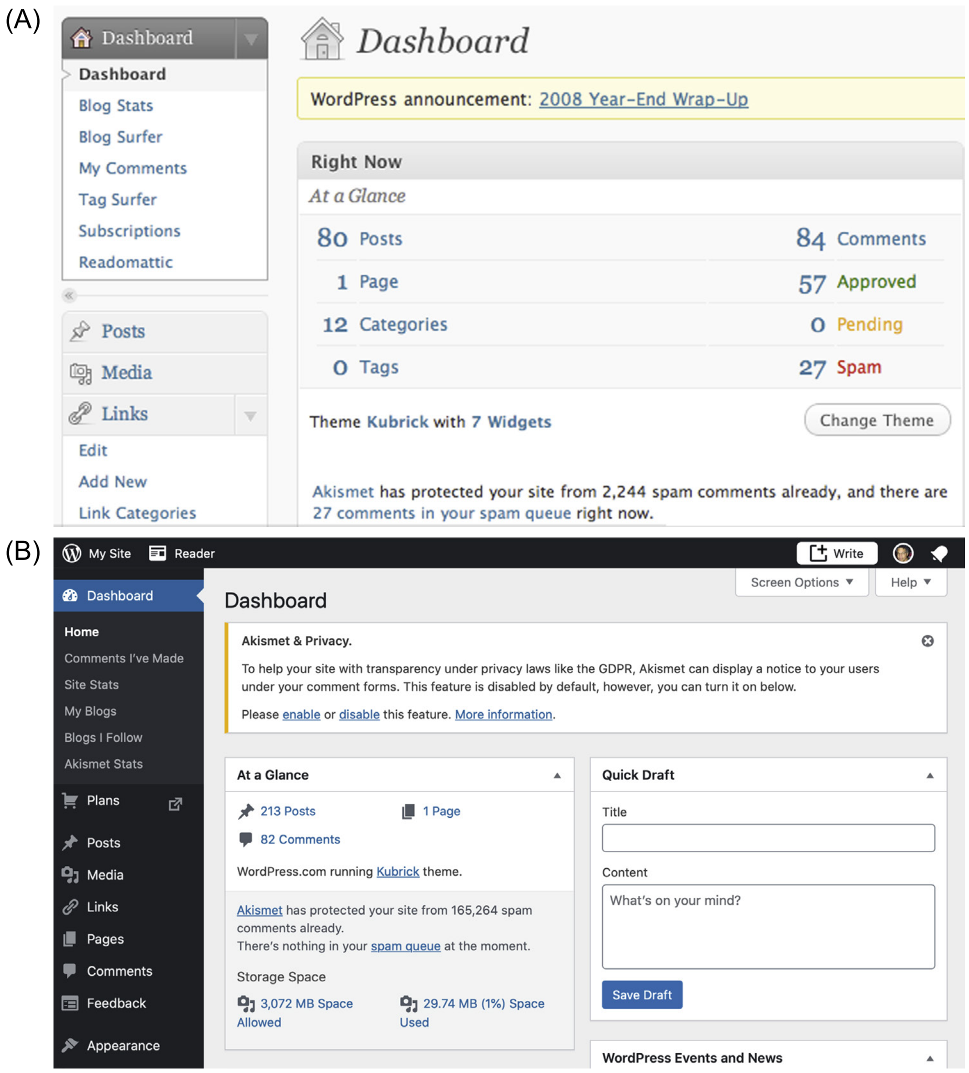
Online form at OReilly.com for reporting errors in books uses two different names for the same field: on the form it is "Format …," but in the error message it is "Version."

to discover and remember that sometimes "Dashboard" meant the entire administrative area, and sometimes it meant the Dashboard *page* of the administrative area. This overloading of "Dashboard" no doubt happened as follows—early versions of WordPress had only one Dashboard page containing a few features; later versions added Dashboard features until there were too many for one page, so it was split into several and the whole set of pages was called the "Dashboard." Their design mistake was keeping the same name for the first page—the original Dashboard. To avoid the overloaded term, all they had to do was devise a new name for the first page in the new multipage Dashboard. In WordPress's 2020 blog-admin site, they did exactly that—they named the first page "Home" (see Fig. 11.9B).

Developing task-focused, familiar, consistent terminology is easier with a good conceptual model

The good news is that when you perform a task analysis and develop a task-focused conceptual model, you also get the vocabulary your target user population uses to talk about the tasks. You don't have to make up new terms for the user-visible concepts in your application—you can use the terms that people who do the task *already* use. In fact, you *shouldn't* assign new names for those concepts, because any names you assign will likely be computer technology concepts foreign to the task domain.[2]

[2] Unless you are designing software development tools.


(A)

Dashboard

Dashboard

Blog Stats

Blog Surfer

My Comments

Tag Surfer

Subscriptions

Readomattic

Posts

Media

Links

Edit

Add New

Link Categories

Dashboard

WordPress announcement: 2008 Year-End Wrap-Up

Right Now

At a Glance

80	Posts	84	Comments
1	Page	57	Approved
12	Categories	0	Pending
0	Tags	27	Spam

Theme Kubrick with 7 Widgets

Change Theme

Akismet has protected your site from 2,244 spam comments already, and there are 27 comments in your spam queue right now.

(B)

My Site Reader Write

Dashboard

Home

Comments I've Made

Site Stats

My Blogs

Blogs I Follow

Akismet Stats

Plans

Posts

Media

Links

Pages

Comments

Feedback

Appearance

Screen Options ▼ Help ▼

Dashboard

Akismet & Privacy.

To help your site with transparency under privacy laws like the GDPR, Akismet can display a notice to your users under your comment forms. This feature is disabled by default, however, you can turn it on below.

Please enable or disable this feature. More information.

At a Glance

213 Posts 1 Page

82 Comments

WordPress.com running Kubrick theme.

Akismet has protected your site from 165,264 spam comments already.

There's nothing in your spam queue at the moment.

Storage Space

3,072 MB Space Allowed 29.74 MB (1%) Space Used

Quick Draft

Title

Content

What's on your mind?

Save Draft

WordPress Events and News

FIGURE 11.9

WordPress.com. (A) 2009 site uses "Dashboard" for all the blogging controls and the first page of them. (B) 2020 site avoids overloading by naming the first page "Home."

From the conceptual model, a software development team should create a product *lexicon*. The lexicon gives a name and definition for each object, action, and attribute that the product—including its documentation—exposes to users. The lexicon should map terms onto concepts 1:1. It should not assign multiple terms to a single concept or a single term to multiple concepts.

Terms in the lexicon should come from the software's supported *tasks*, not its implementation. Terms should fit well into the users' normal task vocabulary, even those that are new. Typically, technical writers, user-interface designers, developers, managers, and users all help create the lexicon.

Certain concepts in GUIs have industry-standard names. These are the GUI equivalents of "reserved words" in programming languages. If you rename such concepts or assign new meanings to the standard names, you will confuse users. One such reserved term is "select." It means clicking on an object to highlight it, marking it as the object for future actions. The word "select" should not be used for any other purpose in a GUI (e.g., adding an item to a list or a collection). Other reserved GUI terms are "click," "press," "drag," "button," and "link."

Follow the product lexicon consistently throughout the software, user manuals, and marketing literature. Treat it as a living document: as the product evolves, the lexicon changes based on new design insights, changes in functionality, usability test results, and market feedback.

WHEN RISK IS LOW, WE EXPLORE MORE AND LEARN MORE

Imagine you are visiting a foreign city on business for a week or two. You have spare time after your work duties are finished in the evenings and on weekends. Compare two possible cities:

- You have been told that this city is easy to get around in. It is laid out in a consistent grid of streets and avenues with clear street and subway signs written in a language you understand, and the residents and police speak your language and are friendly and eager to help tourists.
- You have been warned that this city has a convoluted, confusing layout with winding, poorly marked streets; the few street and subway signs are in a language you cannot read, and residents don't speak your language and are generally contemptuous of tourists.

In which city are you more likely to go out exploring?

Most interactive systems—desktop software, web services, electronic appliances—have far more functionality than most of their users ever try. Often people don't even know about most of the functionality provided by software or gadgets they use every day. One reason for this is fear of being "burned."

People make mistakes (see Chapter 15). Many interactive systems make it too easy for users to make mistakes, do not allow users to correct mistakes, or make it costly or time-consuming to correct mistakes. People won't be very productive in using such systems: they will waste too much time correcting or recovering from mistakes.

Even more important than the impact on *time* is the impact on *practice and exploration*. A high-risk system, in which mistakes are easy to make and costly, discourages both: people who are anxious and afraid of making mistakes will avoid using the system, and when they do use it, they will tend to stick to familiar, safe paths and functions.

Imagine a violin or trumpet that gave mild electric shocks to its players when they made a mistake. Musicians would avoid practicing with it and would never use it to play new, unfamiliar tunes.

When practice and exploration are discouraged, learning suffers.[3] In contrast, a low-risk system—in which mistakes are hard to make, low in cost, and easy to correct—reduces stress and encourages practice and exploration, and therefore supports learning. With such systems, users are more willing to try new paths: "Hmmm, I wonder what *that* does?" Creating a low-risk environment means doing the following:

- Prevent errors where possible.
- Deactivate invalid commands.
- Make errors easy to detect by showing users clearly what they have done (e.g., deleting a paragraph by mistake).
- Allow users to undo, reverse, or correct errors easily.

For more on the different types of errors people make and how to help people avoid them and recover from them, please see Chapter 15.

PROGRESSIVE DISCLOSURE AND METAPHOR CAN SUPPORT LEARNING

Progressive disclosure is a design technique that fosters learning by hiding features until the user knows enough to need them (Johnson, 2007). It is a bit like putting training wheels on a bicycle while a child is learning to keep balanced.

In some cases, hidden controls and settings appear when users click a "Details" or "More" button, as in Microsoft Word's print controls (see Fig. 11.10).

In other cases, controls and settings stay hidden or inactive (grayed out) until user actions or choices make them relevant, as in the fictional map app shown in Fig. 11.11.

Another design technique that supports learning is the use of *metaphor*: designing a user interface to be similar to something users already know how to use, so they can transfer what they know from the other thing to the new user interface (Johnson, 2007). Examples of metaphors used in digital systems are the desktop, the trash can for deleting, file folders, and on-screen copies of handheld calculators. Interactive robots and voice-controlled assistants employ another sort of metaphor: by looking or sounding similar to a person, they prompt users to interact with them similarly to how they would interact with other people.

[3]The benefits of practice were described earlier in this chapter.

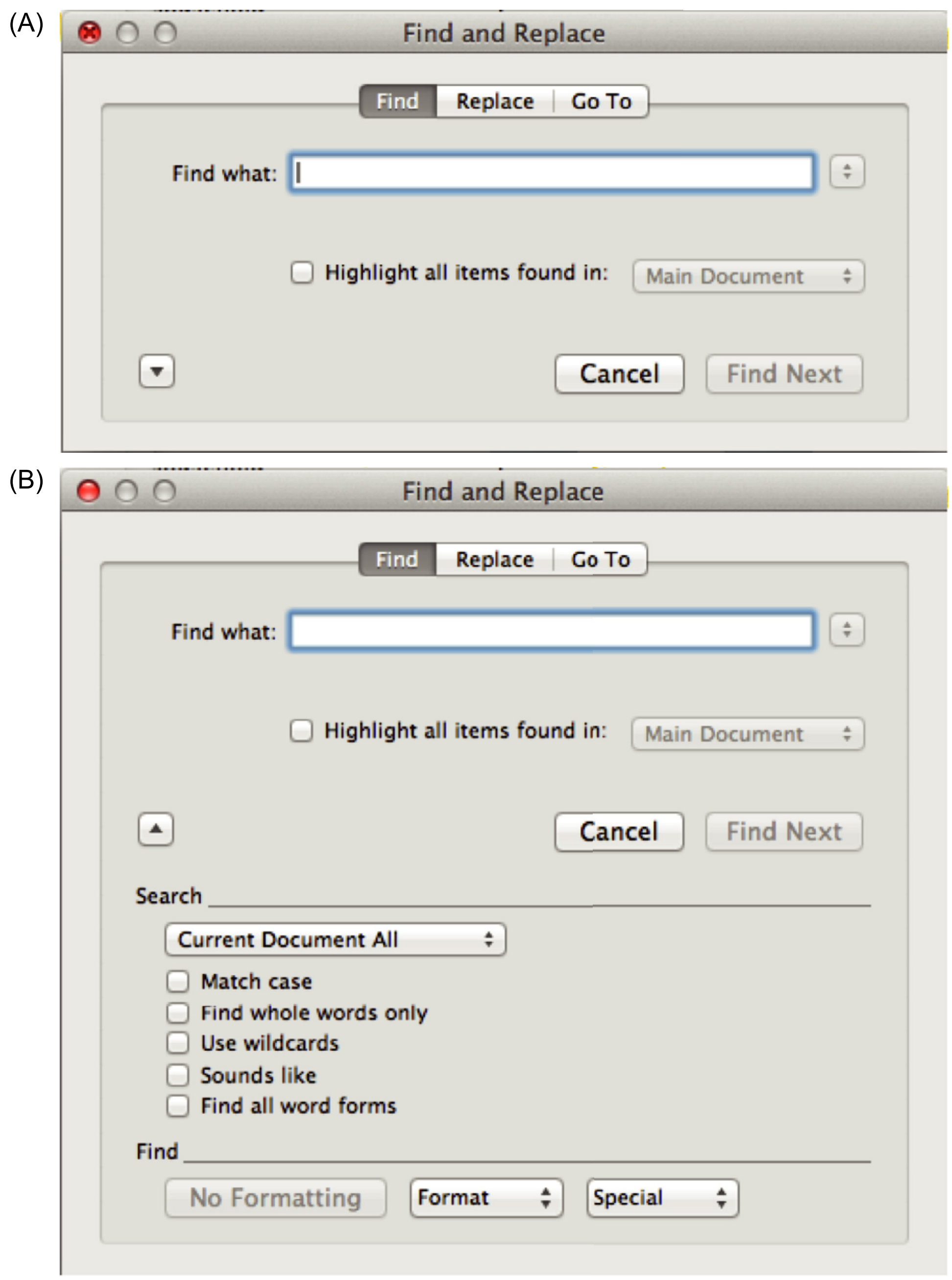
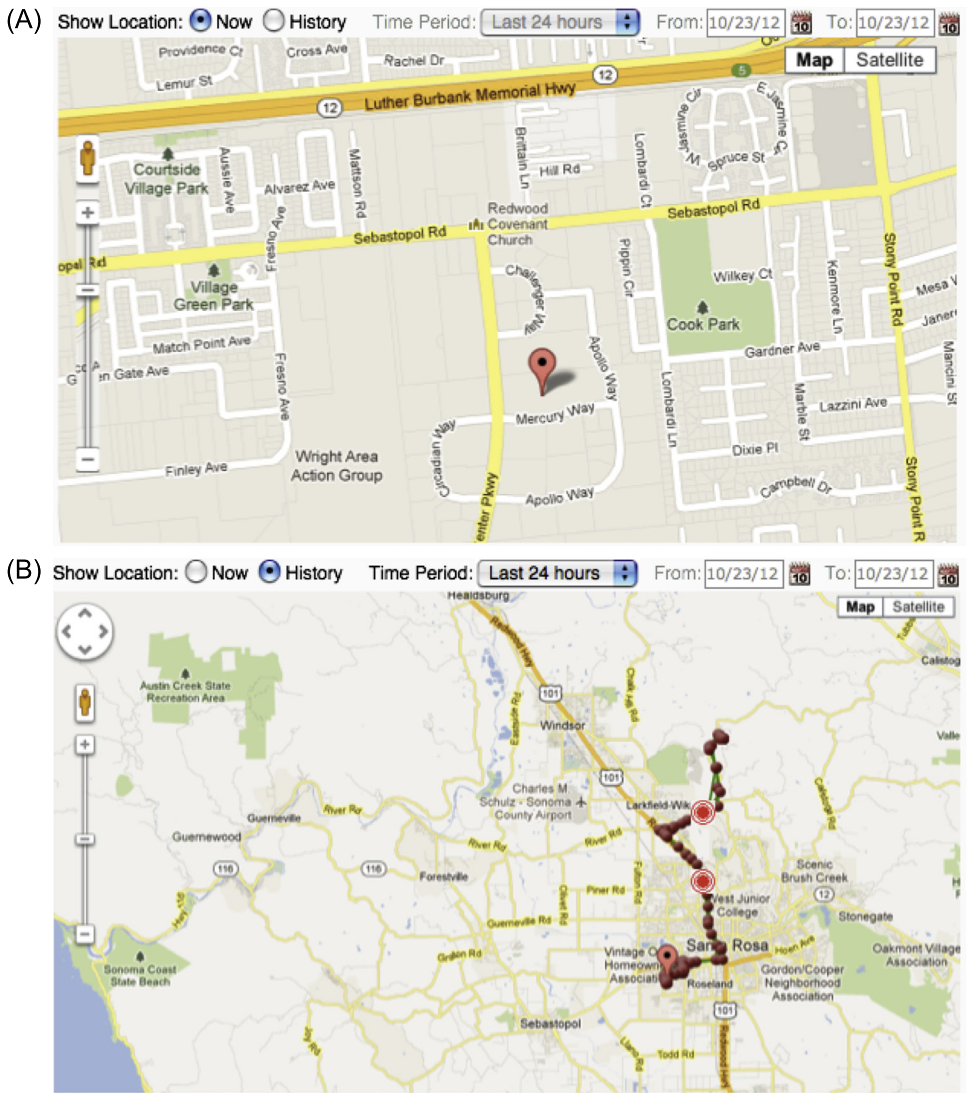


FIGURE 11.10

Microsoft Word print controls use progressive disclosure. (A) Basic controls. (B) Detailed controls.



(A) Show Location: [✓ Now] History Time Period: Last 24 hours From: 10/23/12 To: 10/23/12

(B) Show Location: Now [✓ History] Time Period: Last 24 hours From: 10/23/12 To: 10/23/12

FIGURE 11.11

Map app keeps History controls inactive until relevant. (A) Showing current location. (B) Showing location history.

IMPORTANT TAKEAWAYS

- Learning mainly consists of shifting activities from controlled to automatic processing. The brain encodes new knowledge and experience by constantly rewiring its neural circuits.
- People learn faster when they practice frequently, regularly, and precisely.
- People learn faster when the operation of a system is task focused and simple. That requires minimizing the gap between users' goals and the actions required to achieve them. Task analysis and conceptual modeling can help reduce that gap.
- Consistency and predictability in digital products and services also help people learn faster. Consistency is important both at a conceptual level and at the level of keystrokes. Predictability allows users to develop a mental model of how to operate an app or website. In order to be learnable, products and services based on AI technology must be predictable.
- People learn faster when vocabulary is task focused, familiar, and consistent. Designers should avoid exposing users to tech jargon terminology. The vocabulary of an app, a website, or an appliance should match the task domain, not the digital domain. Terms should map 1:1 to concepts. Follow this slogan: "Same word, same thing. Different word, different thing."
- People learn faster in low-risk environments. Help users avoid errors, and help them recover from errors.
- Design to support new users by using progressive disclosure and metaphor. Using progressive disclosure means hiding or deactivating advanced functionality until users need it. Using metaphor is designing a UI to be similar to something users already know how to use.