3.8 Interaction errors

Most of analyses in this chapter are directed at the physical properties of the human-machine interface, such as degrees of freedom in 2D or 3D or spatial and temporal relationships between input controllers and output displays. Human performance, although elaborated in Chapter 2, has not entered into discussions here, except through secondary observations that certain interactions are better, worse, awkward, or unintuitive. At the end of the day, however, human performance is what counts. Physical properties, although instructive and essential, are secondary. Put another way, human performance is like food, while physical properties are like plates and bowls. It is good and nutritious food that we strive for.

Empirical research in HCI is largely about finding the physical properties and combinations that improve and enhance human performance. We conclude this chapter on interaction elements with comments on that nagging aspect of human performance that frustrates users: interaction errors. Although the time to complete a task can enhance or hinder by degree, errors only hinder. Absence of errors is, for the most part, invisible. As it turns out, errors—interaction errors—are germane to the HCI experience. The big errors are the easy ones—they get fixed. It is the small errors that are interesting.

As the field of HCI matures, a common view that emerges is that the difficult problems (in desktop computing) are solved, and now researchers should focus on new frontiers: mobility, surface computing, ubiquitous computing, online social networking, gaming, and so on. This view is partially correct. Yes, the emerging themes are exciting and fertile ground for HCI research, and many frustrating UI problems from the old days are gone. But desktop computing is still fraught with problems, lots of them. Let's examine a few of these. Although the examples below are from desktop computing, there are counterparts in mobile computing. See also student exercise 3-8 at the end of this chapter.

The four examples developed in the following discussion were chosen for a specific reason. There is a progression between them. In severity, they range from serious problems causing loss of information to innocuous problems that most users rarely think about and may not even notice. In frequency, they range from rarely, if ever, occurring any more, to occurring perhaps multiple times every minute while users engage in computing activities. The big, bad problems are well-traveled in the literature, with many excellent sources providing deep analyses on what went wrong and why (e.g., Casey, 1998, 2006; Cooper, 1999; Johnson, 2007;

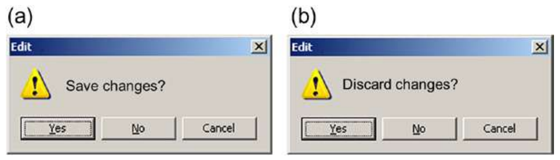


FIGURE 3.42

HCI has come a long way: (a) Today's UIs consistently use the same, predictable dialog to alert the user to a potential loss of information. (b) Legacy dialog rarely (if ever) seen today.

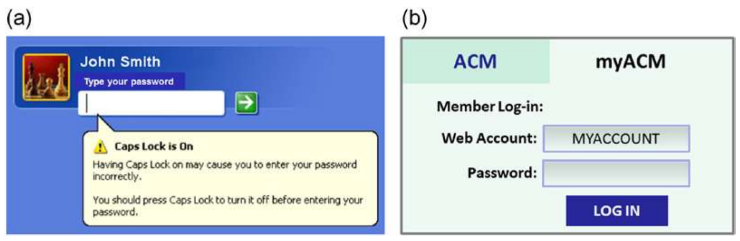
B. H. Kantowitz and Sorkin, 1983; Norman, 1988). While the big problems get lots of attention, and generally get fixed, the little ones tend to linger. We'll see the effect shortly. Let's begin with one of the big problems.

Most users have, at some point, lost information while working on their computers. Instead of saving new work, it was mistakenly discarded, overwritten, or lost in some way. Is there any user who has not experienced this? Of course, nearly everyone has a story of losing data in some silly way. Perhaps there was a distraction. Perhaps they just didn't know what happened. It doesn't matter. It happened. An example is shown in Figure 3.42. A dialog box pops up and the user responds a little too quickly. Press ENTER with the "Save changes?" dialog box (Figure 3.42a) and all is well, but the same response with the "Discard changes?" dialog box spells disaster (Figure 3.42b). The information is lost. This scenario, told by Cooper (1999, 14), is a clear and serious UI design flaw. The alert reader will quickly retort, "Yes, but if the 'Discard changes?' dialog box defaults to 'No,' the information is safe." But that misses the point. The point is that a user expectation is broken. Broken expectations sooner or later cause errors.

Today, systems and applications consistently use the "Save changes?" dialog box in Figure 3.42a. With time and experience, user expectations emerge and congeal. The "Save changes?" dialog box is expected, so we act without hesitating and all is well. But new users have no experiences, no expectations. They will develop them sure enough, but there will be some scars along the way. Fortunately, serious flaws like the "Discard changes?" dialog box are rare in desktop applications today.

The following is another error to consider. If prompted to enter a password, and CAPS_LOCK mode is in effect, logging on will fail and the password must be reentered. The user may not know that CAPS_LOCK is on. Perhaps a key-stroking error occurred. The password is reentered, slowly and correctly, with the CAPS_LOCK mode still in effect. Oops! Commit the same error a third time and further log-on attempts may be blocked. This is not as serious as losing information by pressing ENTER in response to a renegade dialog box, but still, this is an interaction error. Or is it a design flaw? It is completely unnecessary, it is a nuisance, it slows our interaction, and it is easy to correct. Today, many systems have corrected this problem (Figure 3.43a), while others have not (Figure 3.43b).

The CAPS_LOCK error is not so bad. But it's bad enough that it occasionally receives enough attention to be the beneficiary of the few extra lines of code necessary to pop up a CAPS_LOCK alert.



Caps Lock is On

Having Caps Lock on may cause you to enter your password incorrectly.

You should press Caps Lock to turn it off before entering your password.

Member Log-in:

Web Account: MYACCOUNT

Password:

FIGURE 3.43

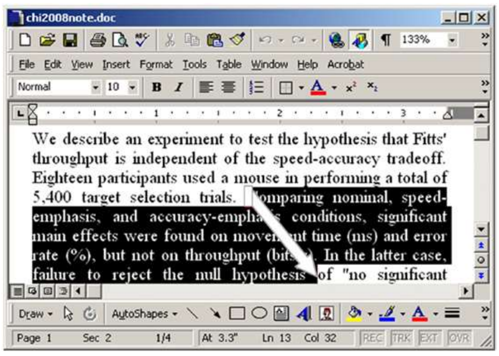
Entering a password: (a) Many systems alert the user if CAPS_LOCK is on. (b) Others do not.

Let's examine another small problem. In editing a document, suppose the user wishes move some text to another location in the document. The task is easy. With the pointer positioned at the beginning of the text, the user presses and holds the primary mouse button and begins dragging. But the text spans several lines and extends past the viewable region. As the dragging extent approaches the edge of the viewable region, the user is venturing into a difficult situation. The interaction is about to change dramatically. (See Figure 3.44.) Within the viewable region, the interaction is position-control—the displacement of the mouse pointer controls the *position* of the dragging extent. As soon as the mouse pointer moves outside the viewable region, scrolling begins and the interaction becomes velocity-control—the displacement of the mouse pointer now controls the *velocity* of the dragging extent. User beware!

Once in velocity-control mode, it is anyone's guess what will happen. This is a design flaw. A quick check of several applications while working on this example revealed dramatically different responses to the transition from position control to velocity control. In one case, scrolling was so fast that the dragging region extended to the end of the document in less time than the user could react (≈200 ms). In another case, the velocity of scrolling was controllable but frustratingly slow. Can you think of a way to improve this interaction? A two-handed approach, perhaps. Any technique that gets the job done and allows the user to develop an expectation of the interaction is an improvement. Perhaps there is some empirical research waiting in this area.

Whether the velocity-control is too sensitive or too sluggish really doesn't matter. What matters is that the user experience is broken or awkward. Any pretense to the interaction being facile, seamless, or transparent is gone. The user will recover, and no information will be lost, but the interaction has degraded to error recovery. This is a design error or, at the very least, a design-induced error. Let's move on to a very minor error.

When an application or a dialog box is active, one of the UI components has focus and receives an event from the keyboard if a key is pressed. For buttons,


chi2008note.doc
133%
File Edit View Insert Format Tools Table Window Help Acrobat
Normal
10
B I
Draw
AutoShapes
Page 1 Sec 2 1/4 At 3.3" Ln 13 Col 32 REC TRK EXT OVR

We describe an experiment to test the hypothesis that Fitts' throughput is independent of the speed-accuracy tradeoff. Eighteen participants used a mouse in performing a total of 5,400 target selection trials. omparing nominal, speed-emphasis, and accuracy-empha s conditions, significant main effects were found on moven nt time (ms) and error rate (%), but not on throughput (bit). In the latter case, failure to reject the null hypothesis of "no significant

FIGURE 3.44

On the brink of hyper-speed scrolling. As the mouse pointer is dragged toward the edge of the viewable region, the user is precipitously close to losing control over the speed of dragging.

focus is usually indicated with a dashed border (see "Yes" button in Figure 3.42). For input fields, it is usually indicated with a flashing insertion bar ("|"). "Focus advancement" refers to the progression of focus from one UI component to the next. There is wide-spread inconsistency in current applications in the way UI widgets acquire and lose focus and in the way focus advances from one component to the next. The user is in trouble most of the time. Here is a quick example. When a login dialog box pops up, can you immediately begin to enter your username and password? Sometimes yes, sometimes no. In the latter case, the entry field does not have focus. The user must click in the field with the mouse pointer or press TAB to advance the focus point to the input field. Figure 3.43 provides examples. Both are real interfaces. The username field in (a) appears with focus; the same field in (b) appears without focus. The point is simply that users don't know. This is a small problem (or is it an interaction error?), but it is entirely common. Focus uncertainty is everywhere in today's user interfaces. Here is another, more specific example:

Many online activities, such as reserving an airline ticket or booking a vacation, require a user to enter data into a form. The input fields often require very specific information, such as a two-digit month, a seven-digit account number, and so on. When the information is entered, does focus advance automatically or is a user action required? Usually, we just don't know. So we remain "on guard." Figure 3.45 gives a real example from a typical login dialog box. The user is first requested to enter an account number. Account numbers are nine digits long, in three three-digit segments. After seeing the dialog box, the user looks at the keyboard and begins entering: *9*, *8*, *0*, and then what? Chances are the user is looking at the keyboard while entering the numeric account number. Even though the user can enter the entire nine digits at once, interaction is halted after the first three-digit group because the user doesn't know if the focus will automatically advance to the next field. There are no expectations here, because this example of GUI interaction has not evolved and stabilized to a consistent pattern. Data entry fields have not reached the evolutionary status of, for example, dialog boxes for saving versus discarding changes (Figure 3.42a). The user either acts, with an approximately 50 percent likelihood of committing an error, or pauses to attend to the display (*Has the focus advanced to the next field?*).

Strictly speaking, there is no gulf of evaluation here. Although not shown in the figure, the insertion point is present. After entering *980*, the insertion point is either after the *0* in the first field, if focus did not advance, or at the beginning of the next field, if focus advanced. So the system does indeed "provide a physical

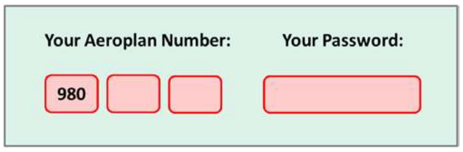


FIGURE 3.45

Inconsistent focus advancement keeps the user on guard. "What do I do next?"

representation that can be perceived and that is directly interpretable in terms of the intentions and expectations of the person" (Norman, 1988, p. 51). That's not good enough. The user's attention is on the keyboard while the physical presentation is on the system's display. The disconnect is small, but nevertheless, a shift in the user's attention is required.

The absence of expectations keeps the user on guard. The user is often never quite sure what to do or what to expect. The result is a slight increase in the attention demanded during interaction, which produces a slight decrease in transparency. Instead of engaging in the task, attention is diverted to the needs of the computer. The user is like a wood carver who sharpens tools rather than creates works of art.

Where the consequences of errors are small, such as an extra button click or a gaze shift, errors tend to linger. For the most part, these errors aren't on anyone's radar. The programmers who build the applications have bigger problems to focus on, like working on their checklist of new features to add to version 2.0 of the application before an impending deadline.[22] The little errors persist. Often, programmers' discretion rules the day (Cooper, 1999, p. 47). An interaction scenario that makes sense to the programmer is likely to percolate through to the final product, particularly if it is just a simple thing like focus advancement. Do programmers ever discuss the nuances of focus advancement in building a GUI? Perhaps. But was the discussion framed in terms of the impact on the attention or gaze shifts imposed on the user? Not likely.

Each time a user shifts his or her attention (e.g., from the keyboard to the display and back), the cost is two gaze shifts. Each gaze shift, or saccade, takes from 70 to 700 ms (Card et al., 1983, p. 28).[23] These little bits of interaction add up. They are the fine-grained details—the microstructures and microstrategies used by, or imposed on, the user. "Microstrategies focus on what designers would regard as the mundane aspects of interface design; the ways in which subtle features of interactive technology influence the ways in which users perform tasks" (W. D. Gray and Boehm-Davis, 2000, p. 322). Designers might view these fine-grained details as a mundane sidebar to the bigger goal, but the reality is different. Details are everything. User experiences exist as collections of microstrategies. Whether booking a vacation online or just hanging out with friends on a social networking site, big actions are collections of little actions. To the extent possible, user actions form the experience, our experience. It is unfortunate that they often exist simply to serve the needs of the computer or application.

[22]The reader who detects a modicum of sarcasm here is referred to Cooper (1999, 47–48 and elsewhere) for a full frontal assault on the insidious nature of feature bloat in software applications. The reference to version 2.0 of a nameless application is in deference to Johnson's second edition of his successful book where the same tone appears in the title: *GUI Bloopers 2.0*. For a more sober and academic look at software bloat, feature creep, and the like, see McGrenere and Moore (2000).

[23]An eye movement involves both a saccade and fixation. A saccade—the actual movement of the eye—is fast, about 30 ms. Fixations takes longer as they involve perceiving the new stimulus and cognitive processing of the stimulus.

Another reason little errors tend to linger is that they are often deemed *user errors*, not design, programming, or system errors. These errors, like most, are more correctly called *design-induced errors* (Casey, 2006, p. 12). They occur "when designers of products, systems, or services fail to account for the characteristics and capabilities of people and the vagaries of human behavior" (Casey, 1998, p. 11). We should all do a little better.

Figure 3.46 illustrates a tradeoff between the cost of errors and the frequency of errors. There is no solid ground here, so it's just a sketch. The four errors described above are shown. The claim is that high-cost errors occur with low frequency. They receive a lot of attention and they get dealt with. As systems mature and the big errors get fixed, designers shift their efforts to fixing less costly errors, like the CAPS_LOCK design-induced error, or consistently implementing velocity-controlled scrolling. Over time, more and more systems include reasonable and appropriate implementations of these interactions. Divergence in the implementations diminishes and, taken as a whole, there is an industry-wide coalescing toward the same consistent implementation (e.g., a popup alert for CAPS_LOCK). The ground is set for user expectation to take hold.

Of the errors noted in Figure 3.46, discard changes is ancient history (in computing terms), CAPS_LOCK is still a problem but is improving, scrolling frenzy is much more controlled in new applications, and focus uncertainty is, well, a mess. The cost is minor, but the error happens frequently.

In many ways, the little errors are the most interesting, because they slip past designers and programmers. A little self-observation and reflection goes a long way here. Observe little errors that you encounter. What were you trying to do? Did it work the first time, just as expected? Small interactions are revealing. What were your hands and eyes doing? Were your interactions quick and natural, or were there unnecessary or awkward steps? Could a slight reworking of the interaction help? Could an attention shift be averted with the judicious use of auditory or tactile feedback? Is there a "ready for input" auditory signal that could sound when an input field receives focus? Could this reduce the need for an attention shift? Would this improve user performance? Would it improve the user experience? Would users like it, or would it be annoying? The little possibilities add up. Think of them as opportunities for empirical research in HCI.

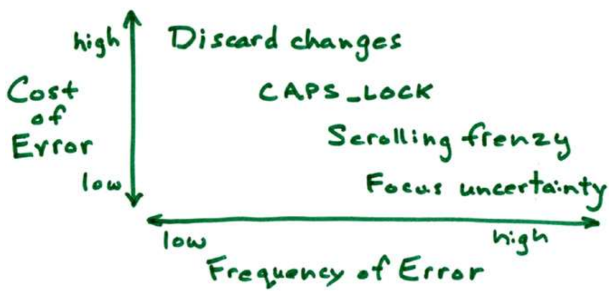
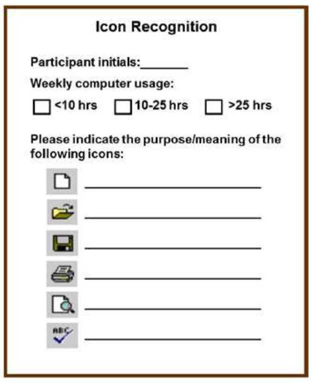


FIGURE 3.46

Trade-off between the cost of errors and the frequency of errors.

STUDENT EXERCISES

3-1. Conduct a small experiment on icon recognition, as follows. Find 15 computer users (participants) and enquire as to their computer experience. Find users with a range of computing experience (e.g., less than 10 hours per week, more than 25 hours per week. Preferably, find an equal number of participants for each experience level.) Prepare a handout sheet with the six toolbar buttons (soft controls) seen in Figure 3.3a. The general idea is shown below:



Icon Recognition

Participant initials:______

Weekly computer usage:

☐ **<10 hrs** ☐ **10-25 hrs** ☐ **>25 hrs**

Please indicate the purpose/meaning of the following icons:

Ask each participant to identify the purpose of the buttons. Record the responses. Write a brief report on the findings, showing and discussing the responses for participants overall and for participants by button and by experience level.

3-2. Conduct a small experiment on panning and zooming using Google Street View. Devise three tasks and measure users' performance using different input methods (e.g., keyboard versus mouse or touchpad versus mouse). As an example task, see Figure 3.9a. For this example, a reasonable task is to pan and zoom to the clock tower (right side of image), determine the time on the clock, then pan and zoom back to the starting position. User performance may be measured in a variety of ways, such as task completion time, accuracy in the final scene position, number of steps or corrective actions, etc. Use five participants. Write a brief report on your observations and measurements.

3-3. Conduct a small experiment to measure the CD gain of the mouse and cursor on a computer system. Build an apparatus, perhaps using wooden blocks, with an opening for a mouse, as below: