Events in the world take time to play out. Perceiving objects and events also takes time. So does remembering perceived events, thinking about past and future events, learning from events, acting on plans, and reacting to perceived and remembered events. How much time? And how does knowing the duration of perceptual and cognitive processes help us design interactive systems?

This chapter provides answers to those questions. It presents the durations of perceptual and cognitive processes, and based on those, provides some real-time deadlines that interactive systems must meet in order to synchronize well with human users. Systems that don’t synchronize well with users’ time requirements are less effective tools and they are perceived as unresponsive.

The second issue, perceived responsiveness, may seem less important than effectiveness, but in fact it is more important. Over the past four decades, researchers have found consistently that an interactive system’s responsiveness—its ability to keep up with users, keep them informed about its status, and not make them wait unexpectedly—is the most important factor in determining user satisfaction. It is not just one of the most important factors; it is the most important factor.\(^1\) It is more important than ease of learning. It is more important than ease of use. Study after study has confirmed this finding (Barber & Lucas, 1983; Carroll & Rosson, 1984; Lambert, 1984; Miller, 1968; Rushinek & Rushinek, 1986; Shneiderman, 1984; Thadhani, 1981).

This chapter first defines responsiveness. It then enumerates some important time constants of human perception and cognition. It ends with real-time guidelines for interactive system design, including examples.

---

\(^1\)Some researchers have suggested that for users’ perception of the loading speed of Web sites, the causality may go the other way: the more success people have at a site, the faster they think it is, even when their ratings don’t correlate with the sites’ actual speed (Perfetti & Landesman, 2001).
RESPONSIVENESS DEFINED

Responsiveness is related to performance, but it is different. Performance is measured in terms of computations per unit of time. Responsiveness is measured in terms of compliance with human time requirements and, as described above, user satisfaction.

Interactive systems can be responsive despite low performance. If you call someone to ask a question, he can be responsive even if he can’t answer your question immediately: he can acknowledge the question and promise to call back. He can be even more responsive by saying when he will call back.

Responsive systems keep a user informed even if they cannot fulfill the user’s requests immediately. They provide feedback about what the user has done and what is happening, and they prioritize the feedback based on human perceptual, motor, and cognitive deadlines (Duis & Johnson, 1990). Specifically, they do the following:

- Let you know immediately that your input was received
- Provide some indication of how long operations will take (see Fig. 12.1)
- Free you to do other things while waiting
- Manage queued events intelligently
- Perform housekeeping and low-priority tasks in the background
- Anticipate your most common requests

Software can have poor responsiveness even if it is fast. Even if a watch repairman is very fast at fixing watches, he is unresponsive if you walk into his shop and he ignores you until he finishes working on another watch. He is unresponsive if you hand him your watch and he silently walks away without saying whether he is going to fix it now or go to lunch. Even if he starts working on your watch immediately, he is unresponsive if he doesn’t tell you whether fixing it will take five minutes or five hours.

FIGURE 12.1

MacOS X file transfer: good progress indicator, useful time estimate, cancel button (circled X).

... Systems that display poor responsiveness do not meet human time deadlines. They don’t keep up with users. They don’t provide timely feedback for user actions, so users are unsure of what they have done or what the system is doing. They...
make users wait at unpredictable times and for unpredictable periods. They limit users’ work pace—sometimes severely. Here are some specific examples of poor responsiveness:

- Delayed feedback for button presses, scrollbar movement, or object manipulations
- Time-consuming operations that block other activity and cannot be aborted (see Fig. 12.2)
- Providing no clue how long lengthy operations will take (see Fig. 12.2)
- Jerky, hard-to-follow animations
- Ignoring user input while performing “housekeeping” tasks users did not request

These problems impede users’ productivity and frustrate and annoy them. Unfortunately, despite all of the research showing that responsiveness is critical to user satisfaction and productivity, a lot of today’s interactive systems have poor responsiveness (Johnson, 2007).

THE MANY TIME CONSTANTS OF THE HUMAN BRAIN

To understand the time requirements of human users of interactive systems, let’s start with neurophysiology.

The human brain and nervous system are not really a single organ; rather, they are made up of a collection of neuron-based organs that appeared at vastly different points in the evolutionary chain from worms to people. This collection provides a large variety of sensory, regulatory, motor, and cognitive functions. Not surprisingly, these functions operate at different speeds. Some work very fast, executing functions in small fractions of a second, while others are many, many times slower, executing over many seconds, minutes, hours, or even longer time spans.

For example, Chapter 10 explained that automatic processing, such as playing a memorized musical piece, operates on a “clock” that is at least 10 times faster than the one governing highly monitored, controlled processing, such as composing a
Listed below are measured durations for perceptual and cognitive brain functions that affect our perceptions of system responsiveness. The times are listed from shortest to longest (Card et al., 1991; Johnson, 2007; Sousa, 2005; Stafford & Webb, 2005):

<table>
<thead>
<tr>
<th>Perceptual and Cognitive Functions</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest gap of silence that we can detect in a sound</td>
<td>1 millisecond (0.001 second)</td>
</tr>
<tr>
<td>Minimum time between spikes in auditory neurons, the fastest neurons in the brain</td>
<td>milliseconds (0.002 second)</td>
</tr>
<tr>
<td>Shortest time a visual stimulus can be shown and still affect us (perhaps unconsciously)</td>
<td>5 milliseconds (0.005 second)</td>
</tr>
<tr>
<td>Minimum noticeable lag in ink as someone draws with a stylus</td>
<td>10 milliseconds (0.01 second)</td>
</tr>
<tr>
<td>Maximum interval for auditory fusion of successive sound pulses into a pitched tone</td>
<td>20 milliseconds (0.02 second)</td>
</tr>
<tr>
<td>Maximum interval for visual fusion of successive images</td>
<td>50 milliseconds (0.05 second)</td>
</tr>
<tr>
<td>Speed of flinch reflex (involuntary motor response to possible danger)</td>
<td>80 milliseconds (0.08 second)</td>
</tr>
<tr>
<td>Time lag between a visual event and our full perception of it (or perceptual cycle time)</td>
<td>100 milliseconds (0.1 second)</td>
</tr>
<tr>
<td>Duration of <em>saccade</em> (involuntary eye movement), during which vision is suppressed</td>
<td>100 milliseconds (0.1 second)</td>
</tr>
<tr>
<td>Maximum interval between events for perception that one event caused another event</td>
<td>140 milliseconds (0.14 second)</td>
</tr>
<tr>
<td>Time required for a skilled reader's brain to comprehend a printed word</td>
<td>150 milliseconds (0.15 second)</td>
</tr>
</tbody>
</table>
The many time constants of the human brain

<table>
<thead>
<tr>
<th>Function</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to <em>subitize</em> (determine the number of) up to four to five items in our visual field</td>
<td>200 milliseconds (0.2 second; 50 milliseconds/item)</td>
</tr>
<tr>
<td>Editorial “window” for events that reach consciousness</td>
<td>200 milliseconds (0.2 second)</td>
</tr>
<tr>
<td>Time to identify (i.e., name) a visually presented object</td>
<td>250 milliseconds (0.25 second)</td>
</tr>
<tr>
<td>Time required to mentally count <em>each</em> item in a scene when there are <em>more than four items</em></td>
<td>300 milliseconds (0.3 second)</td>
</tr>
<tr>
<td>Attentional “blink” (inattentiveness to other input) following recognition of an object</td>
<td>500 milliseconds (0.5 second)</td>
</tr>
<tr>
<td>Visual-motor reaction time (intentional response to unexpected event)</td>
<td>700 milliseconds (0.7 second)</td>
</tr>
<tr>
<td>Maximum duration of silent gap between turns in person-to-person conversation</td>
<td>About 1 second</td>
</tr>
<tr>
<td>Duration of unbroken attention to a single task (“unit task”)</td>
<td>6–30 second</td>
</tr>
<tr>
<td>Time to make critical decisions in emergency situations, e.g., medical triage</td>
<td>1–5 minutes</td>
</tr>
<tr>
<td>Duration of important purchase decision, e.g., buying a car</td>
<td>1–10 days</td>
</tr>
<tr>
<td>Time to choose a lifetime career</td>
<td>20 years</td>
</tr>
</tbody>
</table>

musical piece. Another example is the flinch reflex: a region of the brain called the superior colliculus—part of the evolutionarily ancient old brain—can “see” a rapidly approaching object and can make you flinch or raise your arms long before your cortex has perceived and identified the object.

The sidebar gives the durations of some important perceptual and cognitive brain functions. Most are self-explanatory, but a few require additional explanation.

**Shortest gap of silence that we can detect in a sound: 1 millisecond (0.001 second)**

Our hearing is much more sensitive to short events and small differences than our vision is. Our ears operate using mechanical sound transducers, not electrochemical
neural circuitry. The eardrum transmits vibration to the ossicles (middle-ear bones), which in turn transmit vibration to the cochlea’s hair cells, which, when vibrated, trigger electrical pulses that go to the brain. Because the connection is mechanical, our ears respond to sound much faster than the rods and cones in our retina respond to light. This speed allows our auditory system to detect very small differences in the time when a sound arrives at our two different ears, from which our brain calculates the direction of the sound’s source.

**Shortest time a visual stimulus can be shown and still affect us (perhaps unconsciously): 5 milliseconds (0.005 second)**

This is the basis of so-called subliminal perception. If you are shown an image for 5–10 milliseconds, you won’t be aware of seeing it, but low-level parts of your visual system will register it. One effect of such exposure to an image is that your familiarity with it will increase: if you see it again later, it will seem familiar. Brief exposure to an image or a looming object can also trigger responses from your old brain and midbrain—avoidance, fear, anger, sadness, joy—even if the image disappears before the conscious mind identifies it. However, contrary to popular myth, subliminal perception is not a strong determinant of behavior. It cannot make you do things you wouldn’t otherwise do or want things you wouldn’t otherwise want (Stafford & Web, 2005).

**Speed of flinch reflex (involuntary motor response to possible danger): 80 milliseconds (0.08 second)**

When an object—even a shadow—approaches you rapidly, or if you hear a loud sound nearby, or if something suddenly pushes, jabs, or grabs you, your reflex is to flinch: pull away, close your eyes, throw up your hands in defense, etc. This is the flinch reflex. It is very fast compared to intentional reaction to a perceived event: about 10 times faster. Evidence of the speed of the flinch reflex has been seen not only experimentally but also in examining the injuries that occur when people are attacked or involved in vehicle accidents: often their arms and hands are injured in ways that indicate that they managed to get their hands up in a split second (Blauer, 2007).

**Time lag between a visual event and our full perception of it: 100 milliseconds (0.1 seconds)**

From the time that light from an external event hits your retina to the time that neural impulses from that event reach your cerebral cortex, about 0.1 second elapses. Suppose our conscious awareness of the world lagged behind the real world by a tenth of a second. That lag would not be conducive to our survival: a tenth of a second is a long time when a rabbit you are hunting darts across a meadow. Our brain compensates by extrapolating the position of moving objects by 0.1 second.
Therefore, as a rabbit runs across your visual field, you see it where your brain estimates it is *now*, not where it was 0.1 second ago (Stafford & Web, 2005).

**Maximum interval between events for perception that one event caused another event: 140 milliseconds (0.14 second)**

This interval is the deadline for perception of cause and effect. If an interactive system takes longer than 0.14 second to respond to your action, you won’t perceive your action as having caused the response. For example, if the echoing of characters you type lags more than 140 milliseconds behind your typing, then you will lose the perception that you are typing those characters. Your attention will be diverted away from the meaning of the text and toward the act of typing, which slows you down, pulls typing out of automatic processing and into conscious processing, and increases your chances of making an error.

**Time to subitize (determine the number of) up to four to five items in our visual field: 200 milliseconds (0.2 second; 50 milliseconds/item)**

If someone tosses two coins onto the table and asks how many coins there are, it takes only a glance for you to see that there are two. You don’t have to explicitly count them. You can do the same with three coins, or four. Some people can do it with five. This function is called *subitizing*. Beyond four or five, it gets harder: now you are starting to have to count, or, if the coins happen to fall into separate groups on the table, you can subitize each subgroup and add the results. This phenomenon is why when we count objects using tick-marks, we write the tick-marks in groups of four, then draw the fifth tick-mark across the group, like this: $\ldots \quad \ldots \quad \ldots \quad \mid$. Subitizing feels instantaneous, but it isn’t. It takes about 50 milliseconds per item (Card *et al.*, 1983; Stafford & Webb, 2005). However, that’s much less time per item than explicit counting, which takes about 300 milliseconds per item.

**Editorial “window” for events that reach consciousness: 200 milliseconds (0.2 second)**

The order in which we perceive events is not necessarily the order in which they occur. The brain apparently has a moving “editorial window” of about 200 milliseconds, during which perceived and recalled items vie for promotion to consciousness. Within that time window, events and objects that might otherwise have made it to consciousness may be superseded by others—even ones that occurred later in time (within the window). Within the window, events can also be re-sequenced on the way to consciousness. An example: we see a dot that disappears and immediately reappears in a new position as moving. Why? Our brain certainly does not do it by “guessing” the second object’s position and making us see “phantom” motion in that direction, because we see motion in the correct direction *regardless* of where the new object appears. Answer: We don’t actually perceive motion until the dot
appears in the new position. The second dot must appear within 0.2 second of the disappearance of the first dot in order for the brain to resequence the events.

**Attentional “blink” (inattentiveness to other objects) following recognition of an object: 500 milliseconds (0.5 second)**

Imagine you are riding a subway, gliding slowly through a station. You pass dozens of strangers who are standing on the platform, but you pay little attention to them. Then you spot a friend on the platform, but the train keeps moving and the friend goes out of view. Your attention snaps to thinking about that friend: all sorts of thoughts and feelings about her are triggered. In that moment, your window passes another friend on the platform. Chances are that you would miss the second friend, because your mind was still on the first. That’s the attentional blink (Stafford & Webb, 2005). With a colleague’s help, you can demonstrate it. Choose two target words. Tell the colleague the two words. Then explain that you will read a list of words and at the end you want to know if either of the two target words was in the list. Quickly read a long list of words at a rate of three words per second. Somewhere in the list, include one target word. If the second target word is presented right after the first—within one or two items—your colleague probably won’t hear it.

**Visual-motor reaction time (intentional response to unexpected event): 700 milliseconds (0.7 second)**

This interval is the combined time for your visual system to notice something in the environment and initiate a conscious motor action, and for the motor system to execute the action. If you are driving your car toward an intersection and the light turns red, this is the time required for you to notice the red light, decide to stop, and put your foot on the brake pedal. How long it takes your car to actually stop is not included in the 700 milliseconds. The vehicle stopping time depends on how fast the car is going, the condition of the pavement under the wheels, etc.

This reaction time is *not* the flinch reflex—the old brain responding to rapidly approaching objects, making you automatically close your eyes, dodge, or throw your hands up to protect yourself. That reflex operates about 10 times faster (see above).

The visual-motor intentional reaction time is approximate. It varies a bit among people. It also increases with distractions, drowsiness, blood-alcohol level, and possibly age.

**Maximum duration of silent gap between turns in person-to-person conversation: ~1 second**

This is the approximate normal length of gaps in a conversation. When gaps exceed this limit, participants—either speakers or listeners—often say something to keep the conversation going: they interject “uh” or “uh-huh,” or take over as speaker. Listeners respond to such pauses by turning their attention to the speaker to see what caused it. The precise length of such gaps varies by culture, but it is always in the range of 0.5–2 seconds.
Duration of unbroken attention to a single task (“unit task”):
~10 seconds

When people perform a task, they break it down into little pieces: subtasks. For example, buying airline tickets online consists of: (1) going to a travel or airline Web site, (2) entering the trip details, (3) scanning the results, (4) selecting a set of flights, (5) providing credit card information, (6) reviewing the purchase, and (7) finalizing the purchase. Some subtasks are broken down further, for example entering the trip details consists of entering the trip origin, destination, dates, time, etc., piece by piece. This breaking down of tasks into subtasks ends with small subtasks that can be completed without a break in concentration, with the subgoal and all necessary information either held in working memory or directly perceivable in the environment. These bottom-level subtasks are called “unit tasks” (Card et al., 1983). Between unit tasks, people typically look up from their work, check to see if anything else requires attention, perhaps look out the window or take a sip of coffee, etc. Unit tasks have been observed in activities as diverse as editing documents, entering checkbook transactions, designing electronic circuits, and maneuvering fighter jet planes in dogfights, and they always last somewhere in the range of 6–30 seconds.

ENGINEERING APPROXIMATIONS OF TIME CONSTANTS: ORDERS OF MAGNITUDE

Interactive systems should be designed to meet the temporal requirements of their human users. However, trying to design interactive systems for the wide variety of perceptual and cognitive time constants would be nearly impossible.

But people who design interactive systems are engineers, not scientists. We don’t have to account for the full variety of brain-related time constants and clock-cycle times. We just have to design interactive systems that work for human beings. This more approximate requirement gives us the freedom to consolidate the many perceptual and cognitive time constants into a smaller set that is easier to teach, remember, and use in design.

Examining the list of critical durations presented above yields some useful groupings. Times related to sound perception are all on the order of a millisecond, so we can consolidate them all into that value. Whether they are really 1 millisecond or 2, or 3—we don’t care. We only care about factors of 10.

Similarly, there are groups of durations around 10 milliseconds, 100 milliseconds, 1 second, 10 seconds, and 100 seconds. Above 100 seconds, we are beyond durations that most interaction designers care about. Thus, for designing interactive systems, these consolidated deadlines provide the required accuracy:

- 0.001 second (1 millisecond): Shortest detectable silent audio gap
- 0.01 second (10 milliseconds): Preconscious (“subliminal”) visual perception, shortest noticeable pen-ink lag, auditory fusion
Notice that these deadlines form a convenient series: each successive deadline is 10 times—one order of magnitude—greater than the previous one. That makes the series fairly easy for designers to remember, although remembering what each deadline is for may be challenging.

**DESIGNING TO MEET REAL-TIME HUMAN INTERACTION DEADLINES**

To be perceived by users as responsive, interactive software must follow these guidelines:

- Acknowledge user actions instantly, even if returning the answer will take time; preserve users’ perception of cause and effect
- Let users know when the software is busy and when it isn’t
- Free users to do other things while waiting for a function to finish
- Animate movement smoothly and clearly
- Allow users to abort (cancel) lengthy operations they don’t want
- Allow users to judge how much time lengthy operations will take
- Do its best to let users set their own work pace

In the above guidelines, “instantly” means within about 0.1 second. Much longer than that, and the user interface will have moved out of the realm of cause and effect, reflexes, perceptual-motor feedback, and automatic behavior, into the realm of conversational gaps and intentional behavior see sidebar: “How long does our brain take to ....” After two seconds, a system has exceeded the expected time for turn taking in dialog and has moved into the time range of unit tasks, decision making, and planning.

Now that we have listed time-constants of human perception and cognition, and consolidated them into a simplified set, we can quantify terms such as “instantly,” “take time,” “smoothly,” and “lengthy” in the above guidelines (see also Table 12.1).

**0.001 second (1 millisecond)**

As described above, the human auditory system is sensitive to very brief intervals between sounds. If an interactive system provides audible feedback or content, its
The Time Deadlines for Human Computer Interaction

<table>
<thead>
<tr>
<th>Deadline</th>
<th>Perceptual and Cognitive Functions</th>
<th>Deadlines for Interactive System Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001 second</td>
<td>• Minimum detectable silent audio gap</td>
<td>• Maximum tolerable delay or drop-out time for audio feedback (e.g., tones, “earcons,” music)</td>
</tr>
<tr>
<td>0.01 second</td>
<td>• Preconscious perception • Shortest noticeable pen-ink lag</td>
<td>• Inducing unconscious familiarity of images or symbols • Generating tones of various pitches • Electronic ink maximum lag time</td>
</tr>
<tr>
<td>0.1 second</td>
<td>• Subitizing 1–4 items • Involuntary eye movement (saccade) • Flinch reflex • Perception of cause-effect • Perceptual-motor feedback • Visual fusion • Object identification • Editorial window of consciousness • The perceptual “moment”</td>
<td>• Assume users can “count” 1–4 screen items in ~100 milliseconds, but more than four take 300 milliseconds/item • Feedback for successful hand-eye coordination, e.g., pointer movement, object movement or resizing, scrolling, drawing with mouse • Feedback for click on button or link • Displaying “busy” indicators • Allowable overlap between speech utterances • Maximum interval between animation frames</td>
</tr>
<tr>
<td>1 second</td>
<td>• Max conversational gaps • Visual-motor reaction time for unexpected events • Attentional “blink”</td>
<td>• Displaying progress indicators for long operations • Finishing user-requested operations, e.g., open window • Finishing unrequested operations, e.g., auto-save • Time after info presentation that can be used for other computation, e.g., to make inactive objects active • Required wait time after presenting important info before presenting more</td>
</tr>
<tr>
<td>10 seconds</td>
<td>• Unbroken concentration on a task • Unit task: one part of a larger task</td>
<td>• Completing one step of a multistep task, e.g., one edit in a text editor • Completing user input to an operation • Completing one step in a wizard (multipage dialog box)</td>
</tr>
<tr>
<td>100 seconds</td>
<td>• Critical decision in emergency situation</td>
<td>• Assure that all info required for decision is provided or can be found within this time</td>
</tr>
</tbody>
</table>
audio-generation software should be engineered to avoid network bottlenecks, getting swapped out, deadlocks, and other interruptions. Otherwise, it may produce noticeable gaps, clicks, or lack of synchrony between audio tracks. Audio feedback and content should be provided by well-timed processes running with high priority and sufficient resources.

**0.01 second (10 milliseconds)**

“Subliminal” perception is rarely, if ever, used in interactive systems, so we needn’t concern ourselves with that issue. Suffice it to say that if designers wanted to boost the familiarity of certain visual symbols or images without the users’ awareness, the designers could do so by presenting the images or symbols repeatedly for 10 milliseconds at a time. It is also worth mentioning that while extremely brief exposure to an image can increase its familiarity, the effect is weak—certainly not strong enough to make people like or dislike specific products.

One way for software to generate tones is by sounding clicks at various rates. If the clicks are less than 10 milliseconds apart, they will be heard as a single sustained buzz, in which the pitch is determined in part by the click frequency. Users will hear clicks as distinct if the clicks are separated by intervals of more than 10 milliseconds.

Systems that use stylus-based input should ensure that electronic “ink” does not lag behind the stylus by more than 10 milliseconds; otherwise users will notice the lag and be annoyed.

**0.1 second (100 milliseconds)**

If software waits longer than 0.1 second to show a response to a user’s action, the perception of cause and effect is broken: the software’s reaction will not seem to be a result of the user’s action. Therefore, onscreen buttons have 0.1 second to show they’ve been clicked; otherwise users will assume they missed and click again. This does not mean that buttons have to complete their function in 0.1 second—only that buttons must show that they have been pressed by that deadline.

The main design point about the flinch reflex is that interactive systems should not startle users and cause flinching. Other than that, the flinch reflex and its duration don’t seem very relevant to interactive system design. It is difficult to imagine beneficial uses of the flinch reflex in human-computer interaction, but one can imagine games with loud noises, joysticks with sudden tactile jolts, or three-dimensional virtual environments that cause their users to flinch under some circumstances, perhaps purposefully. For example, if a vehicle detects a pending collision, it could do something to make riders flinch in order to help protect them upon impact.

If an object the user is dragging or resizing lags more than 0.1 second behind the user’s pointer movements, users will have trouble placing or resizing the object as desired. Therefore, interactive systems should prioritize feedback for hand-eye coordination tasks so that the feedback never lags past this deadline. If meeting that deadline is unachievable, then the system should be designed so that the task does not require close hand-eye coordination.
If an operation will take more than a perceptual “moment” (0.1 second) to complete, it should display a busy indicator. If a busy indicator can be displayed within 0.1 second, it can double as the action acknowledgment. If not, the software’s response should come in two parts: a quick acknowledgment within 0.1 second, followed by a busy (or progress) indicator within 1 second. More guidance for displaying busy indicators is given below.

The brain can reorder events within this approximate time window before the events reach consciousness. Human speech is highly prone to such reordering if it occurs out of order. If you listen to several people talking and some people start talking just before the person before them has finished talking (within the time window), your brain automatically “untangles” the utterances so that you seem to hear them sequentially, without perceived overlap. Television and movies sometimes take advantage of this phenomenon to speed up conversations that normally would take too long.

We also regard 10 per second as an approximate minimum frame rate for perception of smooth animation, even though smooth animation really requires a rate more like 20 frames per second.

1.0 second

Because 1 second is the maximum gap expected in conversation, and because operating an interactive system is a form of conversation, interactive systems should avoid lengthy gaps in on their side of the conversation. Otherwise, the human user will wonder what is happening. Systems have about 1 second to either do what the user asked or indicate how long it will take. Otherwise, users get impatient.

If an operation will take more than a few seconds, a progress indicator is needed. Progress indicators are an interactive system’s way of keeping its side of the expected conversational protocol: “I’m working on the problem. Here’s how much progress I’ve made and an indication of how much more time it will take.” More guidelines for progress indicators are provided below.

One second is also the approximate minimum time a user needs to respond intentionally to an unanticipated event. Therefore, when information suddenly appears on the screen, designers can assume that users will take at least a second to react to it (unless it causes a flinch response; see above). That lag time can be useful in cases when the system needs to display an interactive object but cannot both render the object and make it interactive within 0.1 second. Instead, the system can display a “fake,” inactive version of the object, and then take its time (1 second) to fill in details and make the object fully interactive. Today’s computers can do a lot in 1 second.

10 seconds

Ten seconds is the approximate unit of time into which people usually break down their planning and execution of larger tasks. Examples of unit tasks: completing a single edit in a text editing application, entering a transaction into a bank account program, and executing a maneuver in an airplane dogfight. Software should support segmentation of tasks into these 10-second pieces.
Ten seconds is also roughly the amount of time users are willing to spend setting up “heavyweight” operations like file transfers or searches—if it takes any longer, users start to lose patience. Computing the result can then take longer if the system provides progress feedback.

Similarly, each step in a multipage “wizard” dialog box should have at most about 10 seconds of work for a user to do. If a step of a wizard takes significantly longer than 10 seconds to complete, it probably should be broken up into multiple smaller steps.

100 seconds (∼1.5 minutes)
Interactive systems that support rapid critical decision making should be designed so that all the necessary information is either already in front of the decision maker or can be easily obtained with minimal browsing or searching. The best user interface for this type of situation is one in which users can obtain all crucial information simply by moving their eyes to where it is displayed2 (Isaacs & Walendowski, 2001).

ADDITIONAL GUIDELINES FOR ACHIEVING RESPONSIVE INTERACTIVE SYSTEMS
In addition to design guidelines specific to each of the consolidated human-computer interaction deadlines, there are general guidelines for achieving responsiveness in interactive systems.

Use busy indicators
Busy indicators vary in sophistication. At the low end, we have simple, static wait-cursors (e.g., an hourglass). They provide very little information: only that the software is temporarily occupied and unavailable to the user for other operations.

Next, we have wait-animations. Some of these are animated wait-cursors, such as the MacOS rotating color wheel. Some wait-animations are not cursors but, rather, larger graphics elsewhere on the screen, such as the “downloading data” animations displayed by some Web browsers. Wait animations are more “friendly” to users than static wait-cursors because they show that the system is working, not crashed or hung up waiting for a network connection to open or data to unlock. Of course, busy animations should cycle in response to the actual computations they represent. Busy animations that are simply started by a function but run independently of it are not really busy animations: they keep running even when the process they represent has hung or crashed and thereby potentially misleading users.

A common excuse for not displaying a busy indicator is that the function is supposed to execute quickly and so doesn’t need to display one. But how quickly is “quickly”? What if the function doesn’t always execute quickly? What if the user has a slower computer than the developer, or one that is not optimally configured?

2 Sometimes called “no-click” user interfaces.
What if the function tries to access data that is temporarily locked? What if the function uses network services and the network is hung or overloaded?

Software should display a busy indicator for any function that blocks further user actions while it is executing, even if the function normally executes quickly (e.g., in less than 0.1 second). This indicator can be very helpful to a user if for some reason the function gets bogged down or hung. Furthermore, it harms nothing: when the function executes at the normal speed, the busy indicator appears and disappears so quickly that users barely see it.

**Use progress indicators**

Progress indicators are better than busy indicators because they let users see how much time remains. Again: the deadline for displaying a progress indicator is 1 second.

Progress indicators can be graphical (e.g., a progress bar), textual (e.g., a count of files yet to be copied), or a combination of graphical and textual. They greatly increase the perceived responsiveness of an application, even though they don’t shorten the time to complete operations.

Progress indicators should be displayed for any operation that will take longer than a few seconds. The longer the operation, the more important they are. Many noncomputer devices provide progress indicators, so we often take them for granted. Elevators that don’t show the elevator’s progress toward your floor are annoying. Most people wouldn’t like a microwave oven that didn’t show the remaining cooking time.

Here are some guidelines for designing effective progress indicators (McInerney & Li, 2002):

- Show work remaining, not work completed. Bad: “3 files copied.” Good: “3 of 4 files copied.”
- Show total progress, not progress on the current step. Bad: “5 seconds left on this step.” Good: “15 seconds left.”
- To show the percentage of an operation that is complete, start at 1%, not 0%. Users worry if the bar stays at 0% for more than a second or two.
- Similarly, display 100% only very briefly at the end of an operation. If the progress bar stays at 100% for more than a second or two, users assume it’s wrong.
- Show smooth, linear progress, not erratic bursts of progress.

**Delays between unit tasks are less bothersome than delays within unit tasks**

Unit tasks are useful not only as a way of understanding how (and why) users break down large tasks. They also provide insight into when system response delays are most and least harmful or annoying.
During execution of a unit task, users keep their goal and necessary information in working memory or within their perceptual field. After they complete one unit task, before moving onto the next one, they relax a bit, and then pull the information needed for the next unit task into memory or into view.

Because unit tasks are intervals during which the content of working memory and the perceptual field must remain fairly stable, unexpected system delays during a unit task are particularly harmful and annoying. They can cause users to lose track of some or all of what they were doing. By contrast, system delays between unit tasks are not as harmful or annoying, even though they may slow the user's overall work rate.

This difference between the impact of system response delays during and between unit tasks is sometimes expressed in user interface design guidelines in terms of task “closure,” as in the classic user interface design handbook *Human-Computer Interface Design Guidelines* (Brown, 1988):

A key factor determining acceptable response delays is level of closure. … A delay after completing a major unit of work may not bother a user or adversely affect performance. Delays between minor steps in a larger unit of work, however, may cause the user to forget the next planned steps. In general, actions with high levels of closure, such as saving a completed document to a file, are less sensitive to response time delays. Actions at the lowest levels of closure, such as typing a character and seeing it echoed on the display, are most sensitive to response time delays.

Bottom line: If a system has to impose delays, it should do so between unit tasks, not during tasks.

**Display important information first**

Interactive systems can appear to be operating fast by displaying important information first, then details and auxiliary information later. Don’t wait until a display is fully rendered before letting users see it. Give users something to think about and act upon as soon as possible.

This approach has several benefits. It distracts users from the absence of the rest of the information and it fools them into believing that the computer did what they asked quickly. Research indicates that users prefer progressive results to progress indicators (Geelhoed, Toft, Roberts, & Hyland, 1995). Displaying results progressively lets users start planning their next unit task. Finally, because of the aforementioned minimum reaction time for users to respond intentionally to what they see, this approach buys at least one more second for the system to catch up before the user tries to do anything. Here are some examples:

- **Document editing software**: When you open a document, the software shows the first page as soon as it has it, rather than waiting until it has loaded the entire document.
• **Web or Database search function:** When you do a search, the application displays items as soon as it finds them, while continuing to search for more.

High-resolution images sometimes render slowly, especially in Web browsers. To decrease the perceived time for an image to render, the system can display the image quickly at low resolution and then re-render it at a higher resolution. Because the visual system processes images holistically, this appears faster than revealing a full-resolution image slowly from top to bottom (see Fig. 12.3). Exception: For text, rendering a page at low resolution first and then substituting a higher resolution version is *not* recommended: it annoys users (Geelhoed *et al*., 1995).

**FIGURE 12.3**
If displaying images takes more than two seconds, display the whole image first at low-resolution (A), not at full resolution from the top down (B).
Fake heavyweight computations during hand-eye coordination tasks

In interactive systems, some user actions require rapid successive adjustments—with hand-eye coordination—until the goal is achieved. Examples include scrolling through a document, moving a game character through a landscape, resizing a window, or dragging an object to a new position. If feedback lags behind user actions by more than 0.1 second, users will have trouble hitting their goal. When your system cannot update its display fast enough to meet this hand-eye-coordination deadline, provide lightweight simulated feedback until the goal is clear and then apply the real operation.

Graphics editors fake feedback when they provide rubberband outlines of objects that a user is trying to move or resize. Some document editing applications make quick-and-dirty changes to internal document data structures to represent the effect of user actions, and then straighten things out later.

Work ahead

Work ahead of users when possible. Software can use periods of low load to precompute responses to high-probability requests. There will be periods of low load because the users are human. Interactive software typically spends a lot of time waiting for input from the user. Don’t waste that time! Use it to prepare something the user will probably want. If the user never wants it, so what? The software did it in “free” time; it didn’t take time away from anything else. Here are some examples of using background processing to work ahead of users:

- A text search function looks for the next occurrence of the target word while you look at the current one. When you tell the function to find the next occurrence of the word, it already has it and so it seems very fast.

- A document viewer renders the next page while you view the current page. When you ask to see the next page, it is already ready.

Process user input according to priority, not the order in which it was received

The order in which tasks are done often matters. Blindly doing tasks in the order in which they were requested may waste time and resources or even create extra work. Interactive systems should look for opportunities to reorder tasks in their queue. Sometimes reordering tasks can make completing the entire set more efficient.

Airline personnel use nonsequential input processing when they walk up and down long check-in lines looking for people whose flights are leaving very soon so they can pull them out of line and get them checked in. In Web browsers, clicking the Back or Stop buttons or on a link immediately aborts the process of loading and displaying the current page. Given how long it can take to load and display a Web page, the ability to abort a page load is critical to user acceptance.
Monitor time compliance; decrease the quality of work to keep up

An interactive system can measure how well it is meeting the real-time deadlines. If it is missing deadlines or determines that it is at risk of missing a pending deadline, it can adopt simpler, faster methods, usually resulting in a temporary reduction in the quality of its output. This approach must be based on real time, not on processor cycles, so that it yields the same responsiveness on different computers.

Some interactive animation uses this technique. As described above, animation requires a frame rate of about 20 frames per second to be seen as smooth. In the late 1980s, researchers at Xerox Palo Alto Research Center (PARC) developed a software engine for presenting interactive animations that treats the frame rate as the most important aspect of the animation (Robertson, Card, & Mackinlay, 1989, 1993). If the graphics engine has trouble maintaining the minimum frame rate because the images are complex or the user is interacting with them, it simplifies its rendering, sacrificing details such as text labels, three-dimensional effects, highlighting and shading, and color. The idea is that it is better to reduce an animated three-dimensional image temporarily to a line drawing than it is to let the frame rate drop below the limit.

The Cone Tree, developed at PARC, is based upon this graphics engine. It is an interactive display of a hierarchical data structure, such as file directories and subdirectories (Fig. 12.4). Users can grab any part of the tree and rotate it. While the tree

![Cone Tree](image)

**FIGURE 12.4**

Cone-tree (A) renders folder labels as blobs while a user rotates the tree (B).
rotates, the software might not have time to render all details of each frame while maintaining smooth animation. In that case, it might, for example, save time by rendering the filename labels on each folder as black blobs instead of as text. When the user stops rotating the tree, it is again rendered in full detail. Most users don’t even notice a degradation of the image during the movement, because they attribute their inability to read the labels to motion blur.

**Provide timely feedback even on the Web**

Developers of Web applications may have dismissed the time deadlines presented above as pure fantasy.

It is true that meeting those deadlines on the Web is difficult—often impossible. However, it is also true that those deadlines are psychological time constants, wired into us by millions of years of evolution, governing our perception of responsiveness. They are not arbitrary targets that we can adjust at will to match the limitations of the Web or of any technology platform. If an interactive system does not meet those deadlines, even if it is a Web application, users will consider its responsiveness to be poor. That means most Web software has poor responsiveness. The question is: how can designers and developers maximize responsiveness on the Web? Here are some approaches:

- Minimize the size and number of images
- Provide quick-to-display thumbnail images or overviews, with ways to show details only as needed
- When the amount of data is too large or time-consuming to display all at once, design the system to give an overview of all the data, and allow users to drill down into specific parts of the data to the level of detail they need
- Style and lay out pages using Cascading Style Sheets (CSS) instead of presentational HTML, frames, or tables
- Use built-in browser components—e.g., error dialog boxes—instead of constructing them in HTML
- Download applets and scripts to the browser; use AJAX methods

**ACHIEVING RESPONSIVENESS IS IMPORTANT**

By following the guidelines described in this chapter and additional responsiveness guidelines given in Johnson (2007), interaction designers and developers can create systems that meet human real-time deadlines, and that users therefore perceive as responsive.

However, the software industry must first recognize these facts about responsiveness:

- It is of great importance to users
• It is different from performance; responsiveness problems are not solvable merely by tuning performance or making hardware faster
• It is a design issue, not just an implementation issue

History shows that faster processors will not solve the problem. Today’s personal computers are as fast as supercomputers were 30 years ago, yet people still wait for their computers and grumble about a lack of responsiveness. Ten years from now, when personal computers and electronic appliances are as powerful as today’s most powerful supercomputers, responsiveness will still be an issue because the software of that day will demand much more from the machines and the networks connecting them. For example, whereas today’s text and document editing applications do spell-checking in the background, future versions may well do Internet-based fact-checking in the background. Additionally, applications 10 years from now will probably be based upon these capabilities and technologies:

• Deductive reasoning
• Image recognition
• Real-time speech generation and recognition
• Downloading terabyte-sized files
• Wireless communication among dozens of household appliances
• Collation of data from thousands of remote databases
• Complex searches of the entire Web

The result will be systems that place a much heavier load on the computer than today’s systems do. As computers grow more powerful, history shows that much of that power is eaten up by applications that demand ever more processing power. Therefore, despite ever-increasing performance, responsiveness will never disappear as an issue.

For design flaws (bloopers) that hurt responsiveness, principles for designing responsive systems, and more techniques for achieving responsiveness, see Johnson (2007).