CHAPTER 2
Triggers

In the 1990s, New York City Transit began converting its seven million daily bus-and-subway passengers from paying fares with tokens—which had been in use since 1904—to paying with a MetroCard, a thin, paper-like plastic card. One of the key pieces of the city's conversion plan was the installation of hundreds of vending machines all over the five boroughs for riders to purchase and fund these new MetroCards. This was no easy task. New York City is home to over eight million people, and tens of millions more live in the surrounding tristate area. According to a report by the Department of City Planning, in 2000, 36% of New York City residents were foreign born; there were enough people speaking a language other than English in 2002 to support 40 magazines and newspapers in another language.[1] Tens of thousands of residents are visually impaired, physically disabled, have little or no schooling, or are illiterate—or some combination

1. "Ethnic Press Booms In New York City." *Editor & Publisher*. July 10, 2002.

thereof. The official guide to New York City reports that over 35 million tourists visit every year (in some years as many as 50 million), many of whom will ride the subway, but few of whom are familiar with it or know how to buy a MetroCard. In fact, the Metropolitan Transit Authority (MTA) had done studies of early MetroCard vending machine prototypes and had found that users were intimidated by the physical form and found the user interface to be incomprehensible.

Stepping into this challenge were designers Masamichi Udagawa, Sigi Moeslinger, and their team at Antenna Design, who were tasked with designing the MetroCard Vending Machine.

As Moeslinger recounts,[2] one assumption they had to dispel for themselves was that their users had experience using touchscreen-style kiosks. In the mid-1990s, few people outside of the service industry (where touchscreens were behind bars and fast-food restaurant counters) had much interaction with touchscreens, with one exception: automatic teller machines (ATMs). The designers assumed that even for the lowest common denominator, they would have at least some experience using an ATM. This turned out not to be the case—at the time, anecdotally up to 50% of the MTA riders didn't have a bank account, and thus didn't own an ATM card. They'd likely never used a machine like the MetroCard dispenser. "The concept of a touchscreen was really alien to them," said Moeslinger. Just getting these users—millions of them—to approach and start using the new, unfamiliar machines was a real issue.

Antenna decided to make each screen of the machine only do one task. "It simulates a dialog and asks one question per screen," said Moeslinger. (In other words, they made every screen a microinteraction.) There was some concern by the MTA that by doing so, it would make the transaction too slow. With millions of people using the machines, additional seconds in the transaction could cause lines and rider complaints. But the opposite proved to be the case. "Having quickly graspable bits of information made the transaction much faster than trying to save screens in the steps of the process."

Antenna explored two interaction models: one in which you put your money in first, then you select what you want (like a soda machine) and a second in which you select what you want first, then pay. Users much preferred the second model, but there was still the problem of getting them to start using the new machines in the first place.

Their solution: turn the entire touchscreen into one huge trigger (see Figure 2-1). As discussed in Chapter 1, a trigger is the physical or digital control or condition(s) that begins a microinteraction. In this case the idle screen—the screen that appears after a transaction is completed or when a machine is sitting idle—became a giant call to action: TOUCH ME. As you can see in Figure 2-1, Antenna did everything short of lighting off signal flares to attract users to the trigger. The word "start" appears three times and

2. The full story is told in her 2008 talk "Intervention-Interaction" at Interaction08.

"touch" twice. The hand animates, pointing towards the Start button. But here's the thing: the whole screen is the trigger. You can touch anywhere to begin using the machine. The Start button is just a visual cue—a faux affordance—so that people know to "push" (when they will actually just tap) it to start. Although it seems like the button is the trigger, really it's the whole screen. It's a great solution to a very hard challenge—and one that is still in use over a decade later.

Figure 2-1. The idle screen from the MetroCard Vending Machine. Antenna Design deliberately overemphasized the trigger, which was not, as one might suspect, the button in the top right. It's actually the whole screen. (Courtesy Antenna Design.)

The MetroCard Vending Machine introduces the first principle of triggers: *make the trigger something the target users will recognize as a trigger in context*. This might mean a physical (or seemingly physical, as with the fake Start button on the MetroCard Vending Machine) control like a button or a switch, or it could be an icon in the task or menu bar. Make it look like you can do something, and make it engaging. And while having a large, animated glowing finger pointing up to a Start button isn't the right affordance for most microinteractions, it was appropriate—and wildly successful—for this context.

Manual Triggers

Where do microinteractions begin? Often they are the very first thing a user encounters as they turn a device on or launch an app. The on/off switch (or its digital equivalent) is the first trigger they encounter. On/off switches are, like the Start screen on the MetroCard, examples of manual triggers. (Automatic, system-initiated triggers are covered later.)

Manual triggers usually spring from a user want or need: "I want to turn the TV on." "I want to turn the ringer off on this phone." "I need to move this text from one place to another." "I want to buy a MetroCard." From a strategic point of view, it is critically important to understand what a user wants (or needs) to do, when they want to do it, and in what context(s) they want to do it. This determines when and where your manual trigger should instantiate. It might need to be globally available, like an on/off switch, or it might be very contextual, only appearing when certain conditions are met, such as being in a particular mode or when the user is in a particular functional area of the app. For example, Microsoft Office's "minibar" formatting menu only appears when text has been highlighted. You can find out these user needs the usual ways: either through design research (observations, interviews, exercises) or through intuition and understanding of the subject area. Or you find out the hard way: in product testing or when the product is launched or out in the field. The point is to match the user need (when and where) with the location of the trigger. (See "Making manual triggers discoverable" on page 29.)

The second principle of triggers, although it seems incredible to even have to say this, is *have the trigger initiate the same action every time*. This is so users can create an accurate mental model of how the microinteraction works. This is violated more frequently than one might imagine. Tech reviewer David Pogue on the Samsung S Note:

> Some of the icons in S Note actually display a different menu every other time you tap them. I'm not making this up.[3]

Another example is the Home button on iPhone and iPad, which either takes you to the home screen or, if you're on the home screen, to Search. (Not to mention all the other functions that it does when you press it twice or press and hold. See "Spring-Loaded and One-off Modes" on page 113 in Chapter 5.) While bundling functionality under the home button is a great way to reuse limited hardware, the single press that takes you to Search instead of doing nothing (or giving some kind of "Hey! You're already there!" feedback) if you're on the home screen is probably a step too far.

Possibly the least effective visible triggers are those that are only items in a drop-down menu. As a menu item, the trigger is effectively invisible; if the microinteraction isn't frequently used, having it buried in a menu requires users to do a lot of searching to find it. Of course, the alternative is to have a visible trigger onscreen for a

3. "A Tablet Straining to Do It All", *The New York Times*, August 15, 2012.

microinteraction that is infrequently used, which might not be the best solution either. Settings are a perfect example of this; users only use them infrequently, yet they can be essential for certain apps, so it can be a design challenge to figure out how visible the trigger for them needs to be.

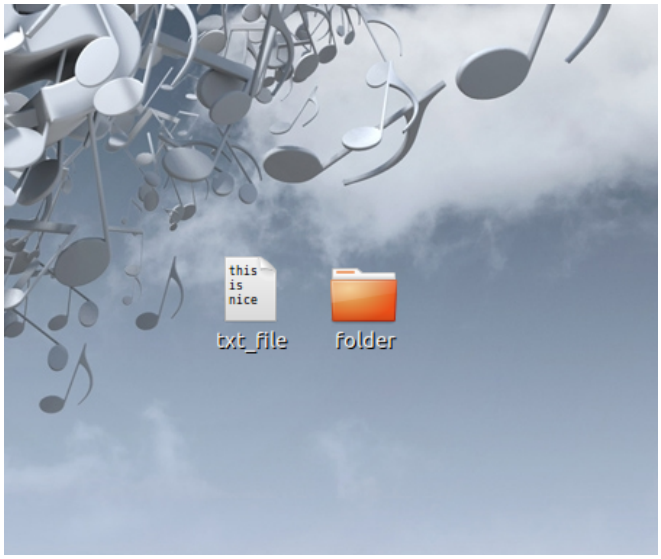

Figure 2-2. On the Gnome desktop, rather than a static text file icon, the icon shows the first three rows of text. (Courtesy Drazen Peric and Little Big Details.)

Bring the Data Forward

The third principle of manual triggers is to *bring the data forward*. The trigger itself can reflect the data contained inside the microinteraction. Ask yourself, what can I show about the internal state of the microinteraction before it is even engaged or while a process is ongoing? What are the most valuable pieces of information I can show? This requires knowing what most people will use the microinteraction for, but you should know that key piece of information before you even begin. A simple example is a stock market app. Perhaps it indicates (via color or an arrow) the current state of the market or a stock portfolio, which could prompt the user to launch the microinteraction—or not. The trigger becomes a piece of ambient information available at a glance that might lead to using the trigger.

The trigger can also indicate where in a process a product is (see Figure 2-3 for an example). The button you use to start a process (making toast, for example) could indicate how long it is until the toast is ready.

Figure 2-3. Google's Chrome browser icon (the trigger to launch it) also indicates active downloads and the download's progress.

The Components of a Trigger

Manual triggers can have three components: the control itself, the states of the control, and any text or iconographic label.

Controls

For manual triggers, the least you can have is a control (see Figure 2-4). The kind of control you choose can be determined by how much control you want to give:

- For a single action (e.g., fast-forward), a button or a simple gesture is a good choice. The "button" in some cases could be an icon or menu item, while the gesture could be a movement like a tap, swipe, or wave. A button could also be (or be paired with) a key command or a gesture.
- For an action with two states (e.g., on or off), a toggle switch makes sense. Alternatively, a toggle button could be used, although it is often hard to tell at a glance what state the button is in—or even that it might have another state. A third (and perhaps worst) choice is that of a regular button where a single press changes the state. If you choose this method, the state the button controls should be absolutely clear. A lamp is clearly on or off, so a regular (nontoggle) button could be used to turn it on and off.
- For an action with several defined states, a dial is a good choice. Aside from having detents, dials can have a push/pull toggle state as well. Alternatively, a set of buttons could be used, one for each choice.

- For an action along a continuum (e.g., adjusting volume) with a defined range, a slide or dial (particularly a jog dial, which can spin quickly) are the best choices. Alternatively, and particularly if there is no defined range, a pair of buttons could be used to change the value up/down or high/low.
- Some manual triggers are made up of multiple controls or elements such as form fields (radio buttons, checkboxes, text-entry fields, etc.). For example, a microinteraction such as logging in might have text-entry fields to put in a username and password. These should be used sparingly and, whenever possible, prepopulated with either previously entered values or smart defaults.

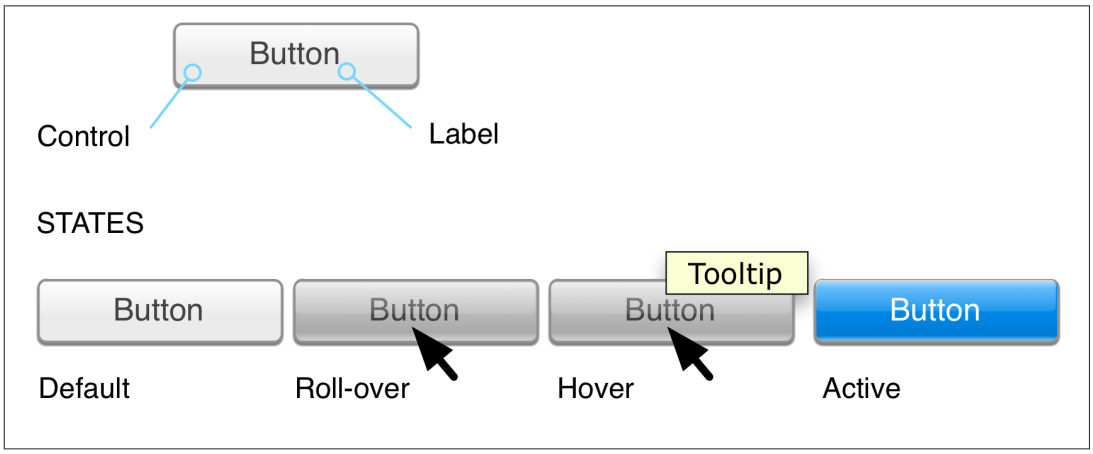

Figure 2-4. The parts of a control.

There are also custom controls that fall outside the traditional buttons, switches, and dials—an example being the scroll wheel from the original (nontouch) iPods. Custom controls will bring a distinct emphasis to your microinteraction, perhaps even making it a Signature Moment. Custom controls can also be gestures or touches (see "Invisible triggers" on page 32).

The goal for microinteractions is to minimize choice and instead provide a smart default and a very limited number of choices. The control you select for the trigger should reflect this philosophy.

Controls are tightly coupled with visual affordances—what users expect can be done, based on sight. The fourth principle of triggers is *don't break the visual affordance*: if your trigger looks like a button, it should work like a button and be able to be pushed.

Making manual triggers discoverable. An important first question to ask is: how noticeable should this trigger be? The fifth principle of triggers is that *the more frequently the microinteraction is used, the more visible it should be*. Author Scott Berkun has a golden rule for discoverability that I've adapted for microinteractions. It's this:

> Microinteractions that most people do, most often, should be highly discoverable. Microinteractions that some people do, somewhat often, should be easily discoverable. Microinteractions that few people do, infrequently, should take some searching to find.[4]

This golden rule will serve you well when determining how discoverable your trigger should be.

But how do we discover anything?

There are two ways we as humans become aware of anything in our environment. The first is that the item, either through movement or sound, causes our attention to involuntarily attune to it. This stimulus-driven attention is what kept our ancestors alive, drawing their attention to charging rhinos and other dangers in the environment. Designers can use this same device to draw attention to a trigger by having it move or make noise. Doing this, particularly on a desktop or web environment, can be incredibly obnoxious. Because we involuntarily focus our attention on movement and sound, having a trigger move or make a sound should be reserved for high-priority microinteractions—and to have it repetitively do so should be reserved for the highest priority microinteractions, such as errors and alerts.

The second way we pay attention to anything is when we're actively seeking to find something—when we're goal-based. We actively turn our attention on items/areas to see if we can find something that meets our current needs. This attention, unless we are impaired or blind, is mostly visual. We turn our bodies, heads, or just eyes to visually search for what we're looking for.

However, it should be noted that our reaction time to sound is faster than visual; auditory stimulus takes 8–10 milliseconds to reach the brain but visual stimulus takes 20–40 milliseconds.[5] Reaction time to sound is also faster: 140–160 milliseconds for sound versus 180–200 milliseconds for visual.[6] Again, this makes evolutionary sense. The human eye is limited to about 180 degrees horizontal and 100 degrees vertical, while hearing is 360 degrees. A predator coming up from behind wouldn't be seen, but could be heard. (Some reptiles and birds actually have 360-degree vision.) But while you could (in theory) use sound as a kind of sonar to find a trigger, in nearly every instance this is impractical.

4. Adapted from Scott Berkun, "The Myth of Discoverability".
5. Marshall, W. H., S. A. Talbot, and H. W. Ades. "Cortical response of the anaesthesized cat to gross photic and electrical afferent stimulation." *Journal of Neurophysiology* 6: 1–15. (1943).
6. Welford, A. T. "Choice reaction time: Basic concepts." In A. T. Welford (Ed.), *Reaction Times*. Academic Press, New York, pp. 73–128. (1980).

When we're searching for something, our field of vision can narrow to as little as 1 degree[7] or less than 1% of what we typically see. This narrowing of our field of vision has been compared to a spotlight[8] or zoom-in lens.[9] We engage in a process of object recognition, wherein we identify and categorize items in the environment.

When we're engaged in object recognition, our eyes are looking for familiar shapes, known as geons. Geons are simple shapes such as squares, triangles, cubes, and cylinders that our brains combine together to figure out what an object is.[10]

Because of geons, it's especially good practice to make triggers, particularly iconic ones, geometric. In general, it's easier to find a target when we're looking for a single characteristic rather than a combination of characteristics,[11] so it's best to keep your triggers visually simple—especially if they are going to live in a crowded environment such as among other icons.

Once we identify an item ("That's a button"), we can associate an affordance to it ("I can push a button"), unless there is another visual cue such as it being grayed out or having a big red X over it that negates the affordance. The sixth principle of manual triggers is *don't make a false affordance*. If an item looks like a button, it should act like a button. With microinteractions, the least amount of cognitive effort is the goal. Don't make users guess how a trigger works. Use standard controls as much as possible. As Charles Eames said, "Innovate as a last resort."

The most discoverable triggers are (from most discoverable to least):

- An object that is moving, like a pulsing icon
- An object with an affordance and a label, such as a labeled button
- An object with a label, such as a labeled icon
- An object alone, such as an icon
- A label only, such as a menu item
- Nothing: an invisible trigger

7. Eriksen, C; Hoffman, J. "Temporal and spatial characteristics of selective encoding from visual displays". *Perception & Psychophysics* 12 (2B): 201–204. (1972).
8. Ibid.
9. Eriksen, C; St James, J. "Visual attention within and around the field of focal attention: A zoom lens model." *Perception & Psychophysics* 40 (4): 225–240. (1986).
10. Geons were first espoused in "Recognition-by-components: A theory of human image understanding" by Irving Biederman in *Psychological Review* 94 (2): 115–47. (1987).
11. Treisman, A. "Features and objects in visual processing." *Scientific American*, 255, 114B–125. (1986).

Invisible triggers. Manual triggers can also be invisible—there might be no label or affordance to let the user know there's a microinteraction to be triggered. Invisible triggers are often sensor-based, made possible via touchscreens, cameras, microphones, and other sensors such as accelerometers (as in Figure 2-5). However, you could also have an invisible trigger that is only a command key (Figure 2-6) or a mouse movement (to the corner of the screen, for example).

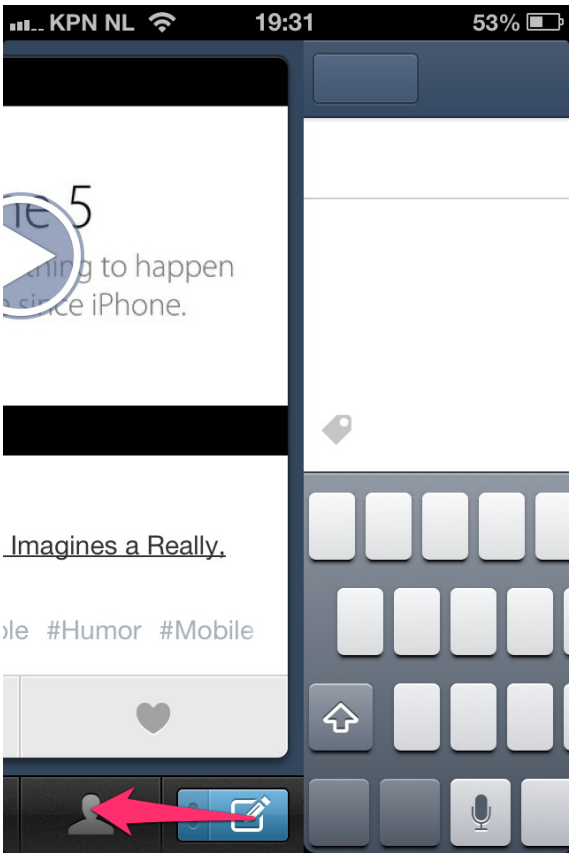

Figure 2-5. Swiping the button to the left on the Tumblr iPhone app (instead of pressing it) is an invisible trigger for creating a new text blog post. You can also swipe upwards to make a new photo post. (Courtesy Robin van't Slot and Little Big Details.)

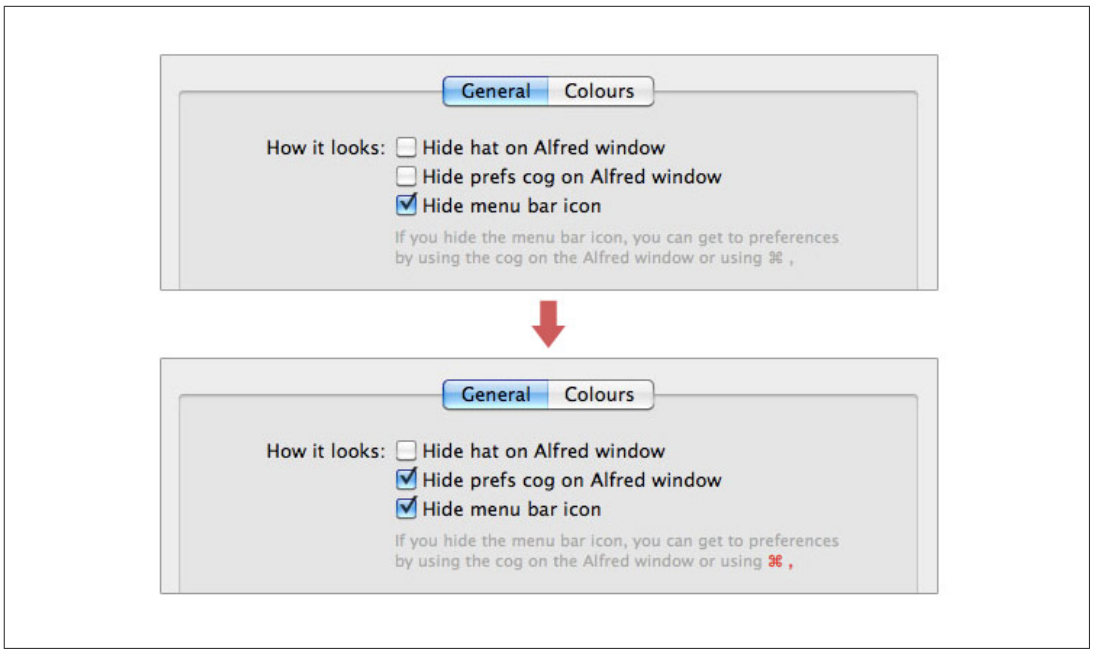

Figure 2-6. In Alfred's settings, if you disable the visible triggers, the invisible one becomes highlighted. (Courtesy Hans Petter Eikemo and Little Big Details.)

Touchscreen UIs currently contain the most common invisible controls. Many multitouch gestures have no visual affordance to indicate their presence, and custom gestures beyond the usual taps and swipes are often found through a process of trial and error (see Figure 2-7).

Voice input is another example of an invisible control. There are three kinds of voice controls:

Always listening
: The product's microphone is always on and users only need to address it (usually by name) to issue a command. Microsoft's Kinect for Xbox works in this manner. "Xbox, play!" is an example of this kind of control.

Dialogue
: The product's microphone turns on at specific times to listen for a response to a prompt. ("Say 'yes' to continue in English.") Most automated customer call interfaces work thus.

Combined with a control
: In order to initiate a voice command, a physical control has to be engaged first. Apple's Siri works like this: users press and hold the Home button in order to issue voice commands.

Gestural controls such as hand waves to turn something on, or a shake to shuffle are also often invisible. Like voice controls, sometimes there is an initial action (like a wave)

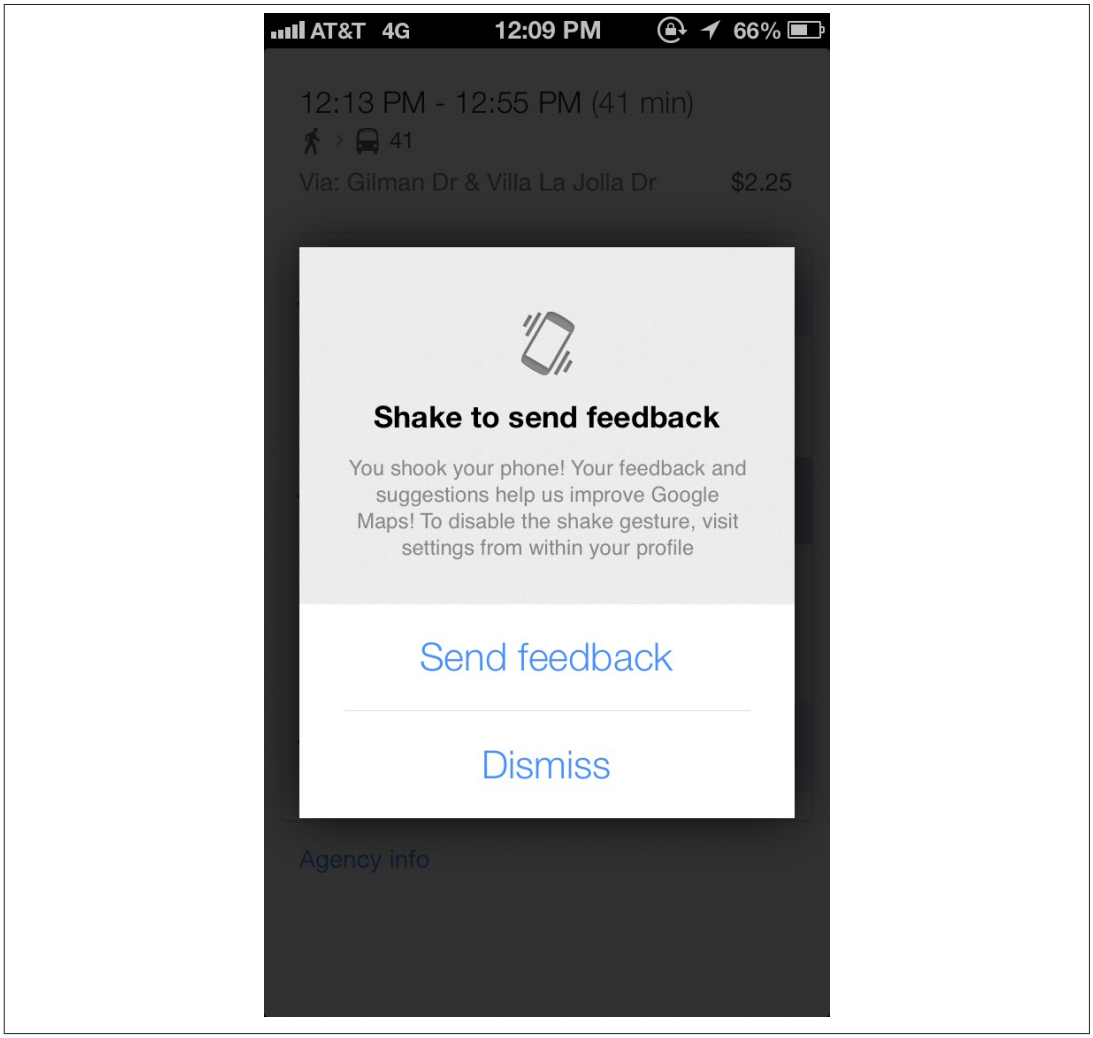

Figure 2-7. In Google Maps for iOS, shaking is an invisible trigger for sending feedback. (Courtesy Little Big Details.)

or a physical control to get the device ready for other gestural commands. With Google Glass, tilting your head upwards or touching the side of the frame turns on the screen. Touching or being close to a device can be an invisible trigger, such as turning on a bathroom sink when hands are put under the faucet. Similarly, moving away from an object can be a trigger as well, such as automatically flushing a toilet when the person has moved away.

Why ever have an invisible trigger? The truth is, no matter what the interface, not every item is going to be immediately discoverable. Making everything visible and discoverable will often mean an incredibly cluttered, complicated, and not easily scannable screen. Hiding items makes the screen or object visually simpler, while not jettisoning

functionality (Figure 2-8). Invisible controls allow for an emphasis on what *is* visible, and creates a hierarchy of what's important. But it is important to note that invisibility should not be an explicit goal for microinteraction (or any kind of interaction) design; rather it should be a byproduct of context and technology: what makes sense to hide, given this environment? Or what must we hide because there is no place to display a visible control with this technology? The best microinteractions have just enough interface, but no more.

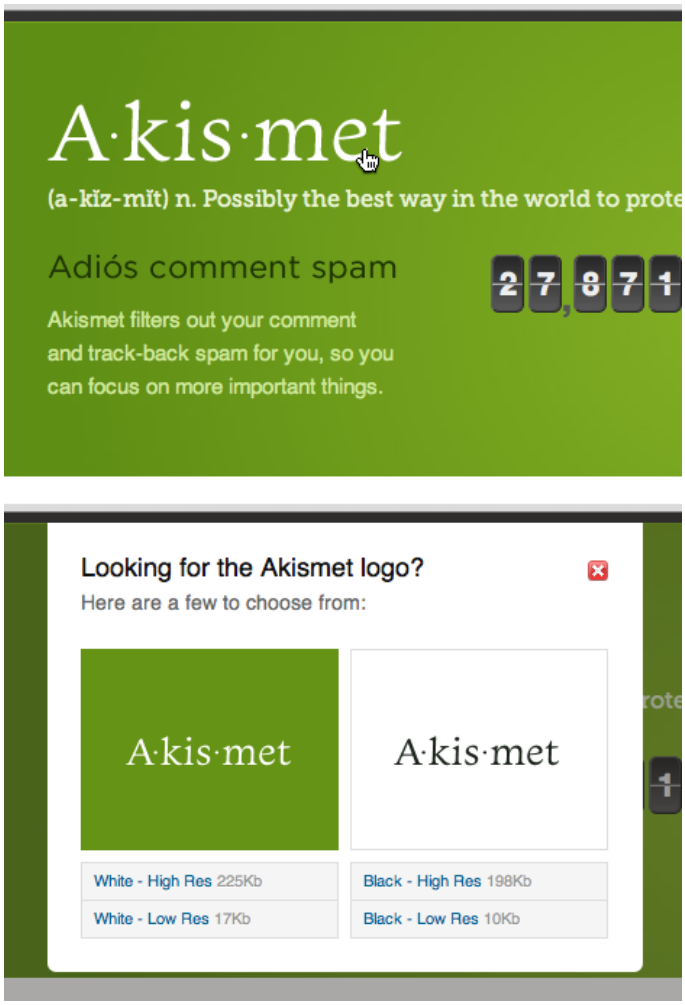

Figure 2-8. Akismet has a clever invisible trigger. When someone right-clicks the logo (presumably to save it), Akismet shows a window with several different resolutions. (Courtesy Fabian Beiner.)

Invisible triggers should be learnable. Once discovered (either through accident, word-of-mouth, or help), users often only have their (faulty) memories to rely on to initiate the microinteraction again. Being learnable means the invisible trigger should be nearly

universally available, or alternatively, only available under particular conditions. Invisible triggers should be guessable, or, ideally, stumbled upon as the user performs other actions. For example, scrolling up past the top of a list reveals a reload microinteraction.

Figure 2-9. KanaSwirl's settings allow for disabling what would otherwise be an invisible trigger (Shake to Pause). (Courtesy Shawn M. Moore and Little Big Details.)

Unless it's impossible—there is no screen or place to put a physical control, such as with Google Glass—never make an invisible trigger for a high-priority microinteraction. Try to, at least, create a visible trigger for the microinteraction. For example, a command key and menu items.

Control states

Some manual triggers have multiple states. Although in most cases you won't have all of these states, when designing a trigger, you should consider them:

Default
: The idle state when there is no activity.

Active
: If there is an activity working in the background—for example, downloading an update or syncing—the trigger could be used to indicate that.

Hover
: Can be used to bring up a tool-tip-style description, expand the size of the trigger to reveal more controls or form fields, or simply indicate that an item is clickable. Even more useful, a hover can display a piece of data that is contained within the microinteraction (see Figure 2-10). For example, hovering over an icon that launches a weather app could show you today's weather without ever having to launch the app. Bring the data forward.

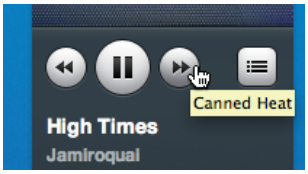

Figure 2-10. In the Rdio player, hovering over the fast-forward and rewind buttons display the upcoming or previous track. (Courtesy Nicholas Kreidberg and Little Big Details.)

Rollover
: Often used to indicate presence or activity, or just an added indicator that the cursor is positioned correctly to engage (see Figure 2-11).

On click/tap/in process
: What happens when the trigger is clicked, tapped, or begun. This can mean the trigger disappears, opens, changes color, or becomes a progress indicator as the microinteraction loads (see Figures 2-12 and 2-14). One variation is that the trigger does not launch the microinteraction immediately, but expands the trigger to reveal more controls. For example, a Save button could open up a panel that asks whether to Overwrite or Save As.

Toggle
: Switches and buttons can indicate their current setting (left/right, up/down, or pressed/unpressed, respectively). On physical devices, switches often make this

Figure 2-11. If you aren't logged in and roll over the Comment field, YouTube prompts you to sign in or sign up. (Courtesy Marian Buhnici and Little Big Details.)

Figure 2-12. Path's Sign Up button smiles when clicked. (Courtesy Little Big Details.)

easier to determine this at a glance, unless the button has some accompanying indicator, such as an LED that glows when in a pressed state.

Setting
: Dials, switches, and sliders can show what setting or stage the microinteraction is currently at (see Figure 2-13).

These indicators of state are usually the trigger itself—the trigger changes its appearance or animates—but it can also be an indicator light such as an LED positioned near the trigger. For example, a glowing red LED near an on/off switch could indicate its off setting. It's good practice to keep any state indicator that isn't attached to the trigger near the trigger. The same applies for any "expanded" version of the trigger: don't open up a window elsewhere. Keep the focus on the trigger itself.

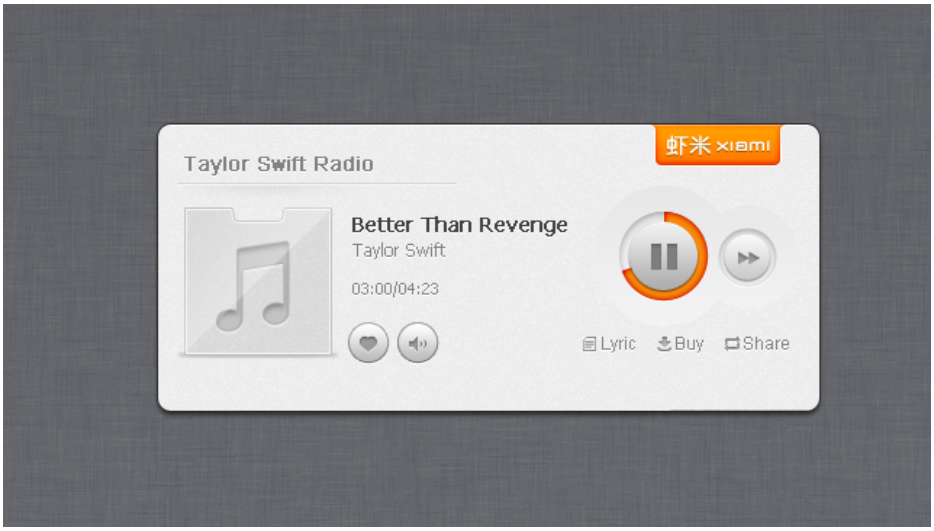

Figure 2-13. The play/pause control on Xiami.com indicates the playing time of a song. (Courtesy Little Big Details.)

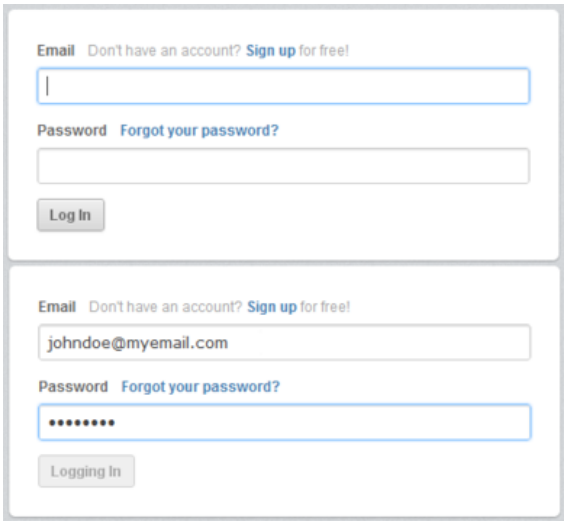

Figure 2-14. In CloudApp, the Log In button changes state after being clicked to let users know an action is happening in the background. (Courtesy Little Big Details.)

Labels

An important part of some triggers are their labels. Labels can name the whole micro-interaction (e.g., the menu item or Microsoft Ribbon item name) or they can be indicators of state, such as a name at each detent on a dial. Labels are interface.

The purpose of a label is clarity: is what I'm about to do the thing I want to be doing? Labels put a name on an action and create understanding where there could otherwise be ambiguity. But because a label becomes one more item to scan and parse, only provide a label if there could be ambiguity. The better practice is to design the control so it has no inherent ambiguity (Figure 2-15).

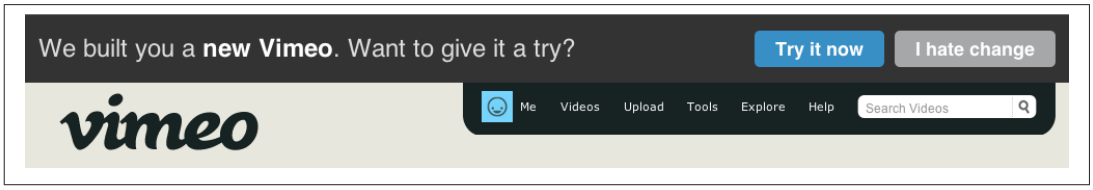

Figure 2-15. Vimeo's cancel/dismiss/not now button is humorously labeled "I hate change." (Courtesy Joe Ortenzi and Little Big Details.)

The seventh principle of manual triggers is to *add a label only if it provides information that the trigger itself cannot*. Consider how you could represent the label visually instead of by adding text. For instance, imagine a rating system of 1–5 stars. You could design a slider with numeric labels of 1–5 or you could have the trigger be just the five stars that light up one by one on hover.

This is obviously not possible or desirable in some cases. A missing label on a button can mean that that button is indistinguishable from every other button around it and thus is never pushed.

Unlike other kinds of product copy (i.e., instructional, marketing), microinteraction labels are not typically the place for brand creativity; they are utilitarian, to create clarity (see Figures 2-16 and 2-17). This is not to say to ignore whimsy or personality, but to do so only when the label remains clear. Google's "I'm Feeling Lucky" button label might be amusing, but tells you absolutely nothing about what is going to happen when you press the button. There is no feedforward—an understanding of what is going to happen before it happens.[12]

12. For more on feedforward, see "But how, Donald, tell us how?: On the creation of meaning in interaction design through feedforward and inherent feedback," by Tom Djajadiningrat, Kees Overbeeke, and Stephan Wensveen, *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*, ACM, New York, NY, USA (2002).

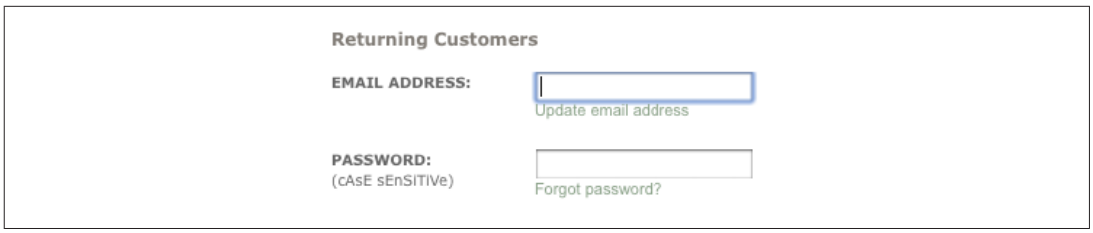

Figure 2-16. Barnes & Noble's website has a label that visually indicates case sensitivity. (Courtesy Paul Clip and Little Big Details.)

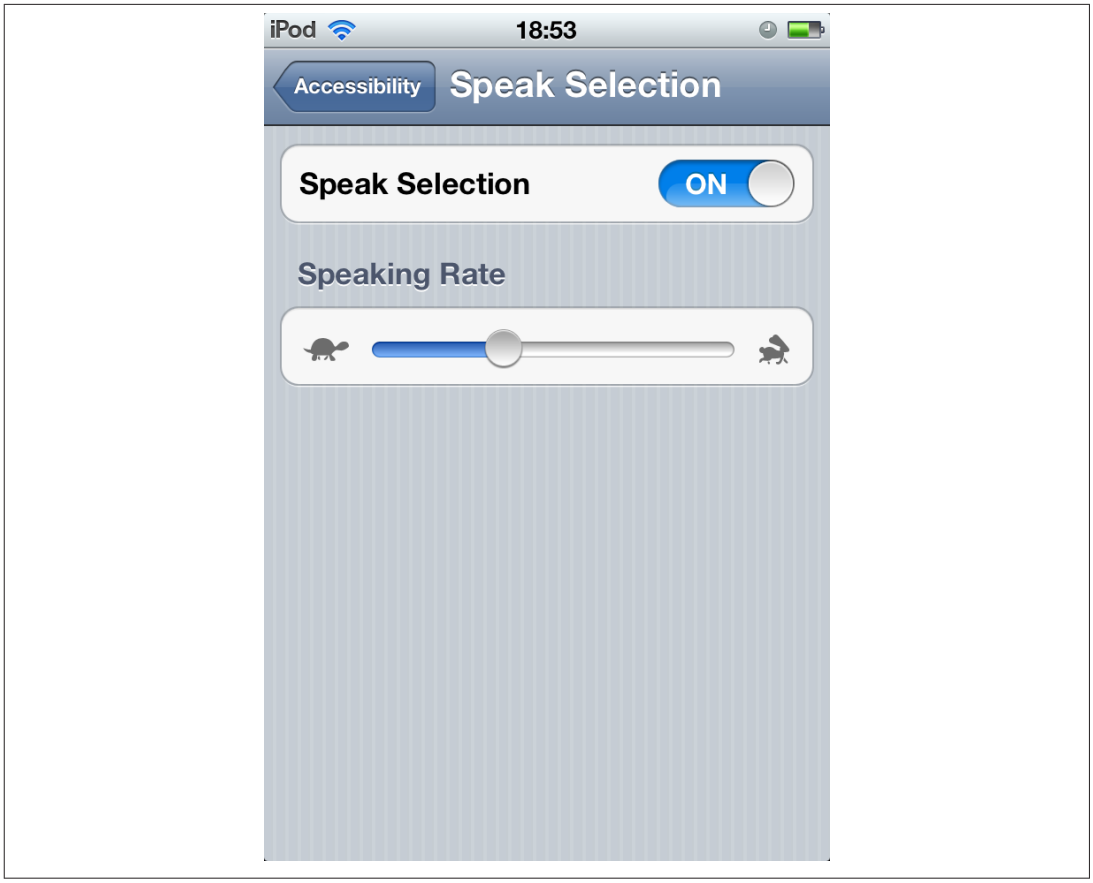

Figure 2-17. Apple's iOS Speak Selection setting has an example of a whimsical but clear iconic label, using the fable of "The Tortoise and the Hare." Although, in cultures where this analogy is unknown, this would certainly be puzzling. (Courtesy Victor Boaretto and Little Big Details.)

In general, labels need to be short yet descriptive and in clear language. "Submit" as a button label may be short, but it doesn't clearly indicate in nontechnical language what action the user is about to take. In microinteractions, specificity matters. Being vague

is the enemy of a good label. Be specific. (For more on this topic, see Microcopy in Chapter 3.)

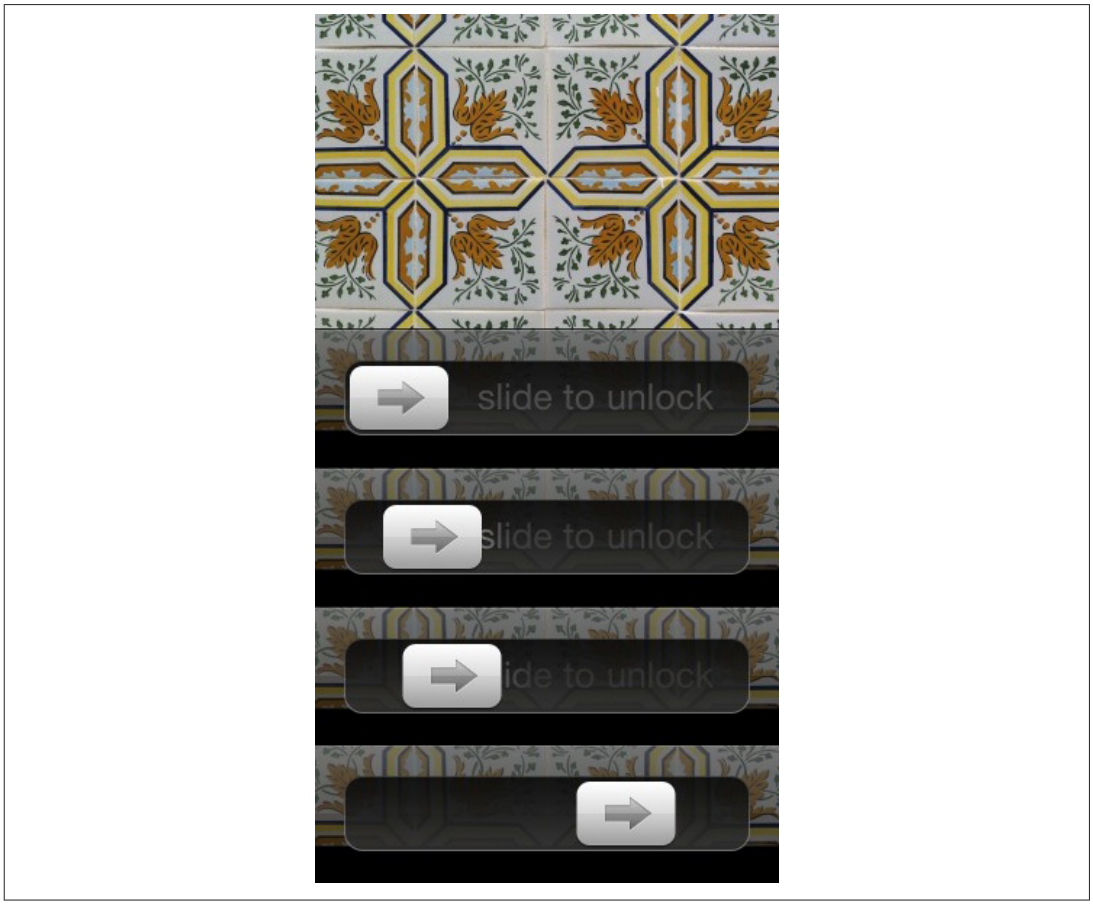

Figure 2-18. The label on the iPhone's Slide to Unlock Trigger vanishes as you slide. (Courtesy Little Big Details.)

Consistency is also important. Since labels can be names, be sure you title anything you're labeling (the microinteraction, a state, a setting, a piece of data) the same name throughout the microinteraction. Don't call it an "alert" in one part of the microinteraction and a "warning" in another part.

The best way to ensure that your labels are successful is to write them in the language of those who will use it. If you're using technical terms, your audience had best be technical as well; otherwise, use casual, plain language. Secondly, test the labels with the target users (see Appendix A). It's not an exaggeration that a majority of usability problems are caused by poor (or no) labeling.

System Triggers

Not all triggers are manual. In fact, we're likely in the era when most triggers aren't human initiated at all, but instead are system initiated. System triggers are those that engage when certain condition(s) are met without any conscious intervention by the user, as in Figures 2-19 and 2-20.

Figure 2-19. The deliveries app checks if there is a tracking number in the clipboard on launch, and if so, a system trigger launches this microinteraction. It's also smart enough to indicate from which courier the number is from. (Courtesy Patrick Patience and Little Big Details.)

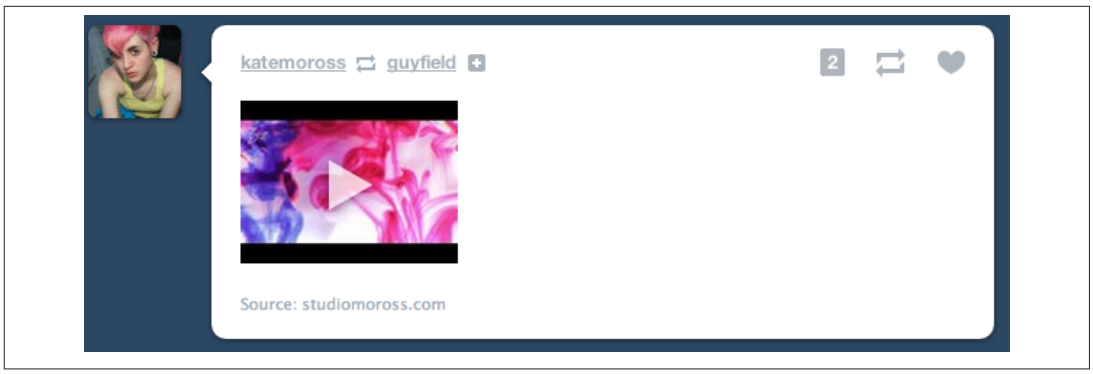

Figure 2-20. An example of a system trigger caused by another person. When someone you follow re-blogs someone you don't on Tumblr, a follow button appears. (Courtesy Brian Jacobs and Little Big Details.)

These common conditions that can initiate a trigger:

Errors
: When a system encounters an error, it often addresses the problem via a microinteraction, such as asking what to do or simply indicating something untoward has happened (see Figure 2-21).

Location
: Location can be on many scales: from within a country, to a particular city or neighborhood, to a particular part of a room. A user in any of these settings can cause a microinteraction to fire.

Incoming data
: Email, status messages, software updates, weather, brightness, and a host of other data that enter networked devices and apps can be triggers for microinteractions such as "You've Got Mail!" alerts.

Internal data
: Likewise internal data such as time and system resources can be triggers (see Figure 2-22). An example is dimming the screen after a set amount of time.

Other microinteractions
: One particular kind of system trigger is when one microinteraction triggers another. A simple example of this is a wizard-style interface. The end of step one (a microinteraction) is the trigger for step two (another microinteraction), and so on. (See "Orchestrating Microinteractions" on page 137 in Chapter 6)

Other people
: In many social interactions, what another person does (e.g., reply to a chat, post a picture or message, send a friend request) can be the basis for a trigger.

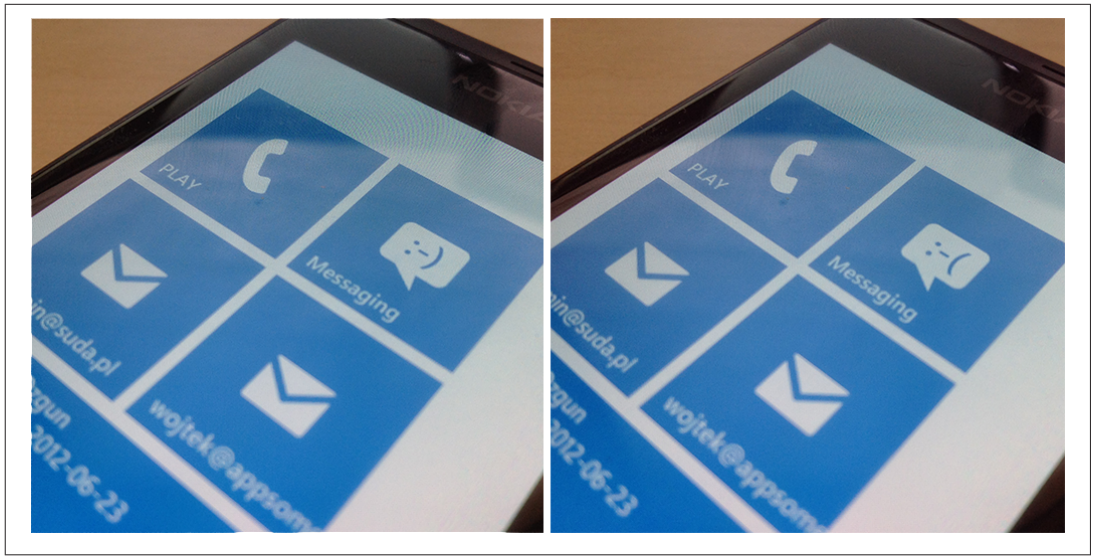

Figure 2-21. In Windows Phone, the messaging icon (a trigger) changes to a sad face if there was an error sending a message. (Courtesy Wojtek Siudzinski and Little Big Details.)

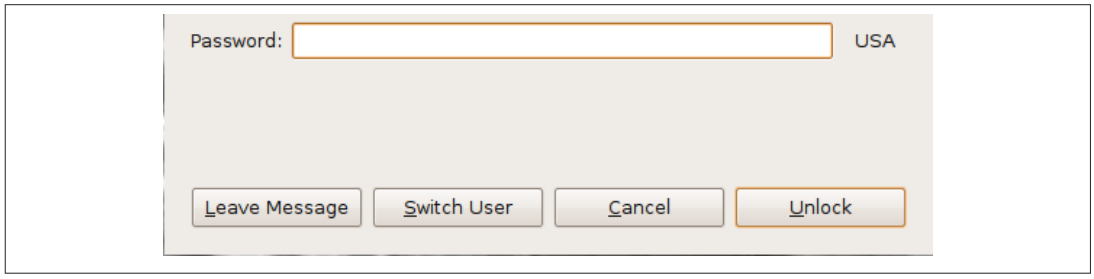

Figure 2-22. In Ubuntu, if the screen has timed out and locked, another trigger appears that lets a visitor leave a message for the device's owner. (Courtesy Herman Koos Scheele and Little Big Details.)

Users might not manually initiate these triggers, but it is good practice to provide some means (e.g., a setting) of adjusting them. Every system-initiated trigger should have some manual means of managing or disabling it. Ideally, this is at the point of instantiation, when the microinteraction has been triggered ("Stop showing me these alerts"), but at a minimum in a settings area.

Additionally, users may want a manual control even when there is a system trigger (See Figure 2-23). For example, a user might want to manually sync a document instead of waiting for it to automatically happen. A manual control can provide assurance, as well as the ability to trigger the microinteraction in case there is something wrong with the system (e.g., the network connection is down, or the sensor didn't register).

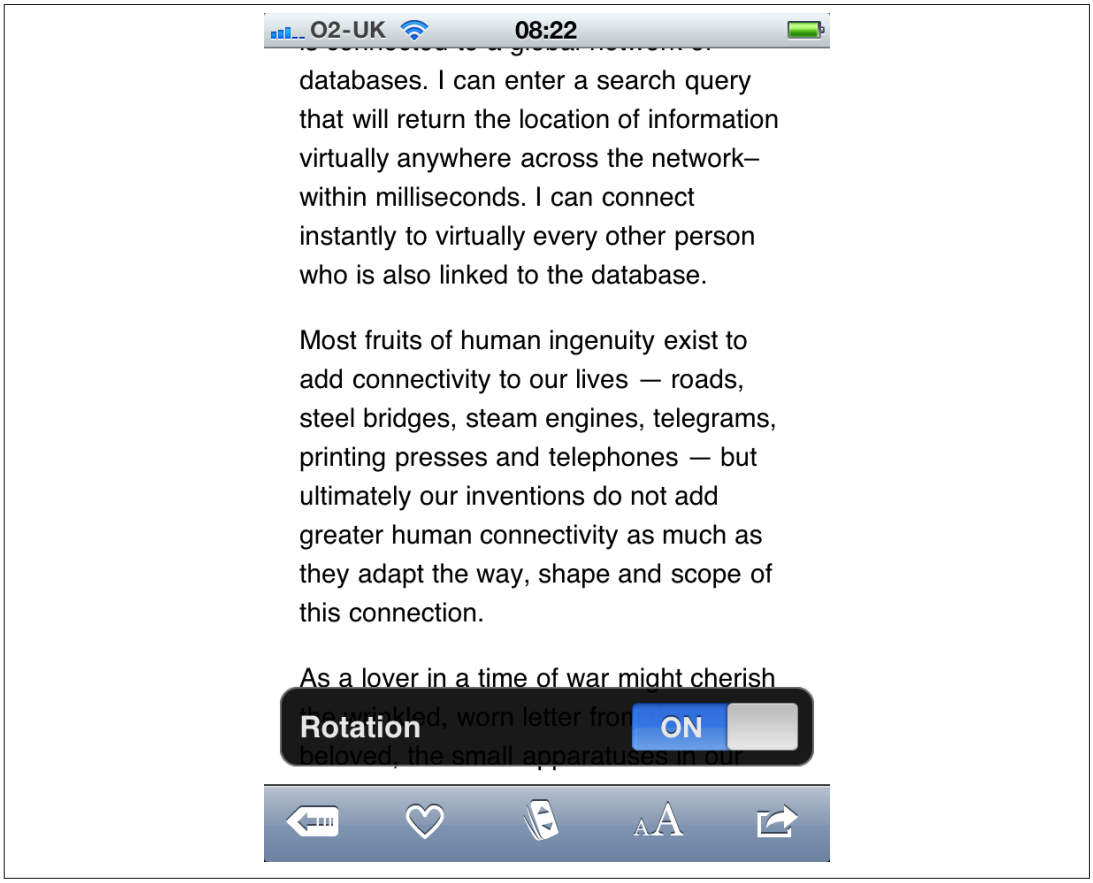

Figure 2-23. In the Instapaper iPhone app, if you accidentally rotate the phone between portrait and landscape mode and then quickly rotate it back, the Rotation lock setting appears. (Courtesy Richard Harrison and Little Big Details.)

System Trigger Rules

Some system triggers themselves need their own rules, the most common of which are when and how often to initiate (Figure 2-24). It can be system-resource intensive—draining battery life, or using bandwidth or processing power—for a product to be constantly pinging remote servers or reading data from sensors.

System trigger rules should answer the following questions:

- How frequently should this trigger initiate?
- What data about the user is already known? How could that be used to make this trigger more effective, more pleasurable, or more customized? For example, knowing it is the middle of the night could reduce the number of times the system trigger initiates. (See "Don't Start from Zero" on page 64 in Chapter 3 for more.)

- Is there any indicator the trigger has initiated? Is there a visible state change while this is happening? After it's happened? When it is about to happen?
- What happens when there is a system error (e.g., no network connection, no data available)? Stop trying, or try again? If the latter, what is the delay until trying again? (Loops are covered more thoroughly in Chapter 5.)

System trigger rules are closely related to the overall rules, which are covered next in Chapter 3.

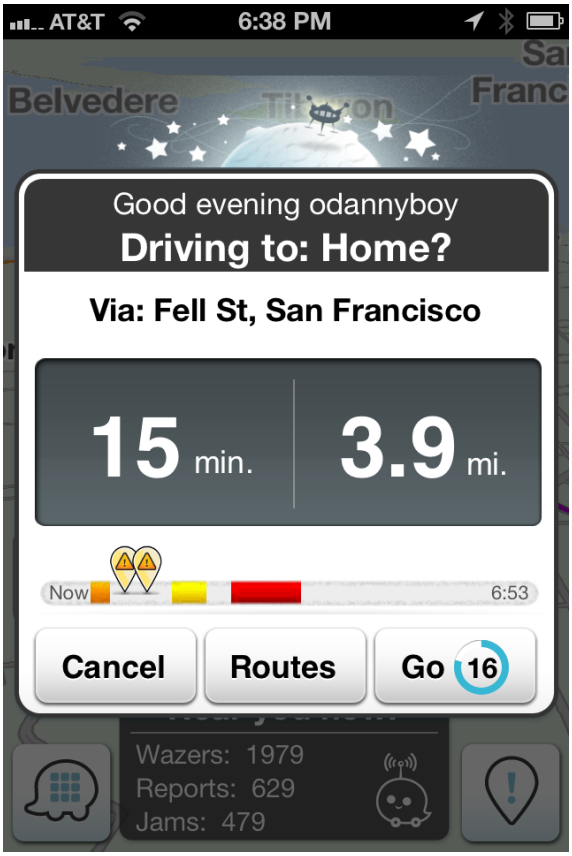

Figure 2-24. Navigation app Waze knows when I open the app in the late afternoon, I'm probably driving home and presents this as an option.

The best triggers are those that, like the Start screen on the MetroCard Vending Machine, fit the context of use and the people who'll use it. The trigger's control matches the states it has to communicate and is appropriately discoverable for how often it will be used. Its labels are clear and written in casual language. And most importantly, it launches users into the actual interaction—the rules.

Summary

A trigger is whatever initiates a microinteraction. Manual triggers are user initiated, and can be a control, an icon, a form, or a voice, touch, or gestural command. System-initiated triggers happen when a certain set of conditions are met.

Make the trigger something the user will recognize as a trigger in context. Have the trigger perform the same action every time.

Bring the data forward. Show essential information from inside the microinteraction on the trigger when possible, such as unread messages or ongoing processes.

If the trigger looks like a button, it should act like a button. Don't break visual affordances.

The more used a microinteraction is, the more visible the trigger should be. Inside a menu is the least visible place for a trigger.

Add labels when there is a need for clarity, when the trigger alone cannot convey all the necessary information. Labels should be brief and in clear language.

System triggers need rules for defining when and how often they appear.

CHAPTER 3
Rules

In October of 2010 at Apple's "Back to the Mac" event, Apple announced the then-latest version of its desktop operating system, Mac OS X Lion (version 10.7), which was released nine months later in July 2011. It sold one million copies on its first day, and over six million copies afterwards. In it, Apple unveiled new versions of Calendar, Mail, and Address Book apps. But there was one microinteraction that garnered a lot of attention, mostly because Apple deemed it unnecessary and removed it. That microinteraction? Save As.

In the early 1980s, Save used to be Save and Put Away (Xerox Star), or Save and Continue alongside Save and Put Away (Apple Lisa). (Put Away meaning close.) Save and Continue eventually just became Save, while Save and Put Away vanished, probably once more RAM allowed for multiple documents to be open at the same time without processor issues. Save As seems to have begun in the 1980s as Save a Copy as, which let users save a version as a new file without renaming. Eventually some applications had all three: Save, Save As, and Save a Copy as. Over time, as people understood the Save As paradigm, and with the broad adoption of the Undo command, Save a Copy as has mostly vanished.

At the time Apple decided to get rid of Save As, the rules of the microinteraction had been fairly stable for about 30 years:

- Make changes to a file.
- Save the file with a new name.
- Subsequent changes happen to the newly created file. The previous file remains as it was the last time it was saved.

With Lion, Apple seemed to feel that Autosave, which allows users to return to previous versions, would obviate the need for Save As. Lion's rules for saving go something like this:

- Make changes to a file.
- Those changes are autosaved every five minutes.
- Subsequent changes happen to the latest version of the file.
- You can rewind to earlier version of the file using the Revert to Last command.
- You can also Browse All Versions, which triggers another microinteraction: the versions browser.
- After two weeks, the file becomes locked and no changes can be made to it without first unlocking it or duplicating it.

If you want to create a separate file, you have to access Duplicate, an entirely different microinteraction:

- Use the Duplicate command to make another (cloned) file.
- The new file appears alongside the current file.

- Rename the new (duplicated) file.
- Subsequent changes happen to the newly created file. The previous file remains as it was the last time it was (auto)saved.

The new rules were practically the inverse of the previous rules: users had to decide *before they made changes* if they wanted the changes to be in a different file. Unfortunately, this is not how most people work (or, more precisely, not how we've been trained to work over the last 30 years). This change severely broke an established mental model and replaced it, not with a better microinteraction but with two microinteractions that together were difficult to understand and misaligned with how most users work. Most people don't need the previous version of their document open at the same time as the altered version. Versioning is what programmers do, not what most people do. When users (infrequently) need an earlier version of a document, they'll manually open it.

Response to the change ranged from puzzlement to outright anger: "The elimination of the Save As... command in applications such as Pages '09 and TextEdit is, in my view, a downright stupid move. It completely breaks a very common workflow for creating a new file, which consists of opening an existing file and saving it under a new name," fumed Macintosh blogger Pierre Igot in "Mac OS X 10.7 (Lion): Why ditch the 'Save As' command?". "I really tried to make myself believe that was an OK decision, but after several months, it was clear that it wasn't," wrote web developer Chris Shiflett in his article "Apple botches 'Save As' ".

Apple responded by quietly returning Save As in the 10.8 version of their OS, Mountain Lion, in 2012—although not to the menu, it should be noted, but as a hidden command—an invisible trigger. But it still didn't work as before: the rules changed again. Lloyd Chambers, author of the *Mac Performance Guide*, summed up the changes and problems in "OS X Mountain Lion: Data Loss via 'Save As' ":

> If one edits a document, then chooses Save As, then BOTH the edited original document and the copy are saved, thus not only saving a new copy, but silently saving the original with the same changes, thus overwriting the original. If you notice this auto-whack, you can "Revert To" the older version (manually), but if you don't notice, then at some later date you'll be in for a confusing surprise. And maybe an OMG-what-happened (consider a customer invoice that was overwritten).

So in Mountain Lion, the rules for Save As work like this:

- Make changes to a file.
- Save the file with a new name.

- Subsequent changes happen to the newly created file. Any changes made to the original file are also saved.
- You can rewind to an earlier version of the original file using the "Revert to Last" command.

This is in addition to the rules for Saving and Duplicating above. So a simple, well-understood microinteraction was replaced by three difficult-to-understand microinteractions, with no feedback as to what the rules are doing in the background. Finally, in an update to Mountain Lion, Apple added a "Keep changes in original document" checkbox in the Save dialog. What a mess.

There are some lessons to be learned. If you can't easily write out or diagram the rules of a microinteraction, users are going to have difficulty figuring out the mental model of the microinteraction, unless you provide feedback to create a "false" model that nonetheless allows users to figure out what is going on. Secondly, unless it's radically new, users likely come to a microinteraction with a set of expectations about how it will work. You can violate those expectations (and in fact the best microinteractions do so by offering an unexpected moment of delight, often by subverting those very expectations), but only if the microinteraction is offering something *significantly better*, where the value to the user is apparent—and, ideally, instantly apparent. Apple is often amazing at this: just as one example, changing the iOS keyboard based on context, so that @ symbols are available on the main keyboard when filling in an email address field. But if the value isn't instantly apparent, your microinteraction could come off as needlessly different, a gimmick. "Things which are different in order simply to be different are seldom better, but that which is made to be better is almost always different," said Dieter Rams.[1]

Designing Rules

At the heart of every microinteraction—just as at the center of every game—are a set of rules that govern how the microinteraction can be used ("played"). What you're trying to create with rules is a simplified, nontechnical model of how the microinteraction operates.

Perhaps the most important part of the rules is the goal. Before designing the rules, you need to determine in the simplest, clearest terms what the goal of the microinteraction is. The best goals are those that are understandable (I know why I'm doing this) and

1. Supposedly said in 1993, and quoted by Klaus Kemp in *Dieter Rams: As Little Design as Possible*, Phaidon Press, 2011. Rams may have unknowingly been paraphrasing 18th century German philosopher Georg Christoph Lichtenberg, who said, "Ich weiss nicht, ob es besser wird, wenn es anders wird. Aber es muss anders werden, wenn es besser werden soll." ("I do not know if it is better if it is different. But it has to be different if it is to be better.")

achievable (I know I can do this). Make sure the goal you're defining isn't just a step in the process; it's the end state. For example, the goal of a login microinteraction isn't to get users to enter their password; the goal is to get them logged in and into the application. The more the microinteraction is focused on the goal rather than the steps, the more successful the microinteraction is likely to be. The goal is the engine of the rules; everything must be in service toward it (Figure 3-1).

Figure 3-1. The goal of this microinteraction on Amazon is to prevent users from buying something off their wish list that someone may have purchased already—to prevent a situation...without spoiling the surprise (sort of). (Courtesy Artur Pokusin and Little Big Details.)

While the purpose of rules is to limit user actions, it's important that the rules not feel like, well, rules. Users shouldn't feel like they have to follow—or worse, memorize—a strict set of instructions to achieve the goal. Instead, what you're striving for is a feeling of naturalness, an inevitability, a flow. The rules should gently guide users through the "interaction" of the microinteraction.

The rules determine:

- *How the microinteraction responds to the trigger being activated.* What happens when the icon is clicked? (See "Don't Start from Zero" on page 64 later in the chapter.)
- *What control the user has (if any) over a microinteraction in process.* Can the user cancel a download, change the volume, or manually initiate what is usually an automatic process like checking for email?
- *The sequence in which actions take place and the timing thereof.* For example, before the Search button becomes active, users have to enter text into the search field.
- *What data is being used and from where.* Does the microinteraction rely on geolocation? The weather? The time of day? A stock price? And if so, where is this information coming from?

Figure 3-2. In Apple's Mountain Lion OS, when you turn on Speech and Dictation, the fans in the machine slow down so the background noise doesn't interfere. (Courtesy Artur Pokusin and Little Big Details.)

- *The configuration and parameters of any algorithms*. While the rules in their entirety can be thought of algorithmically, often certain parts of a microinteraction are driven by algorithms. (See the section on "Algorithms" on page 78 later in the chapter.)
- *What feedback is delivered and when*. The rules could indicate which "steps" should get feedback and which operate behind the scenes.
- *What mode the microinteraction is in*. A mode is a fork in the rules that, when possible, should be avoided. But sometimes it's necessary. For example, in many weather apps, entering the cities you want to know the weather for is a separate entry mode from the default mode of viewing the weather. See Chapter 5 for more on modes.
- *If the microinteraction repeats and how often*. Is the microinteraction a one-time activity, or does it loop? See Chapter 5 for more on loops.
- *What happens when the microinteraction ends*. Does the microinteraction switch to another microinteraction? Does it vanish? Or does it never end?

The set of rules may or may not be entirely known to the user, and they reveal themselves in two ways: by what can be done and by what cannot (see Figure 3-3). Both of these can be an occasion for feedback (see Chapter 4), although as the story of Patron X in

Chapter 1 demonstrates, sometimes the user's mental model does not match up with the conceptual model that the rules create.

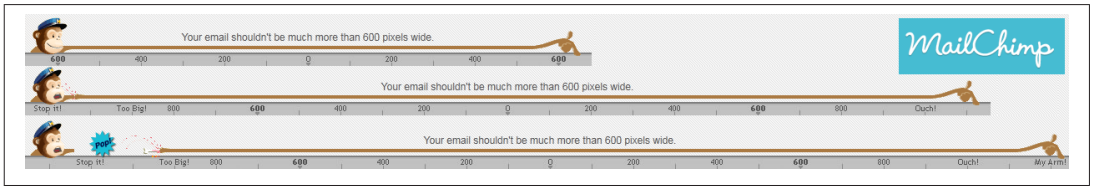

Figure 3-3. MailChimp shows you what can't be done, by having the poor chimp's arm stretch so far that it pops off when you try to make an email too wide. (Courtesy Little Big Details.)

Let's take perhaps the simplest microinteraction there is: turning on a light. The rules are these:

- When the switch is thrown, the light turns on and stays on.
- If the switch is thrown again, turn the light off.

Very simple.[2] But if we put a motion sensor on that light, the rules become a lot more complicated:

- Check for motion every three seconds.
- If anything is moving, is it human sized? (You don't want the light to go on because a cat ran by.)
- If so, turn on the light.
- Check for motion every three seconds.
- Is anything moving?
- If no, wait for 10 seconds, then turn off the lights.

Of course, all of these rules are debatable. Is three seconds too long to check? Or too much: will it use too much power checking that often? Maybe you want the light to turn on when a cat runs by. And I think many of us have a story about being in a bathroom stall and having the lights go out because the sensor didn't detect any motion—maybe 10 seconds is too brief. Needless to say, the rules affect user experience by determining what happens and in what order.

2. Of course, this isn't exactly physically how a light switch works. Flipping the switch completes an electric circuit—a circular path—which allows electrons to flow to the lightbulb. Flipping the switch again breaks the circuit. But users don't need to know this; they only need to understand the rule.

Generating Rules

The easiest way to get started with rules is to simply write down all the general rules you know. These are usually the main actions the microinteraction has to perform, in order. For example for adding an item to a shopping cart, the initial rules might be:

1. On an item page, user clicks Add to Cart button.
2. The item is added to the Shopping Cart.

Very straightforward. But as you continue designing, nuance gets added to the rules. For example:

1. On an item page, check to see if the user has purchased this item before. If so, change the button label from Add to Cart to Add Again to Cart.
2. Does the user already have this item in the cart? If so, change Add to Cart to Add Another to Cart.
3. The user clicks button.
4. The item is added to the Shopping Cart.

And so on. And that's just for a button like the one shown in Figure 3-4. There could be many more rules here.

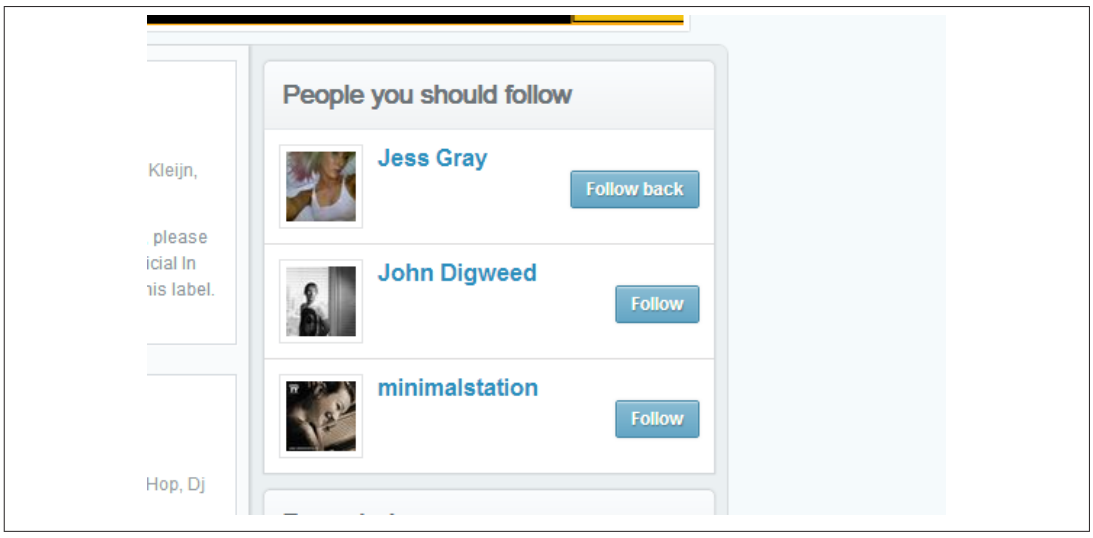

Figure 3-4. A simple button rule. If someone is already following you in Mixcloud, the Follow button becomes Follow back. (Courtesy Murat Mutlu and Little Big Details.)

Of course, rules can also benefit from being visualized. Sometimes a logic diagram can be useful (see Figure 3-5).

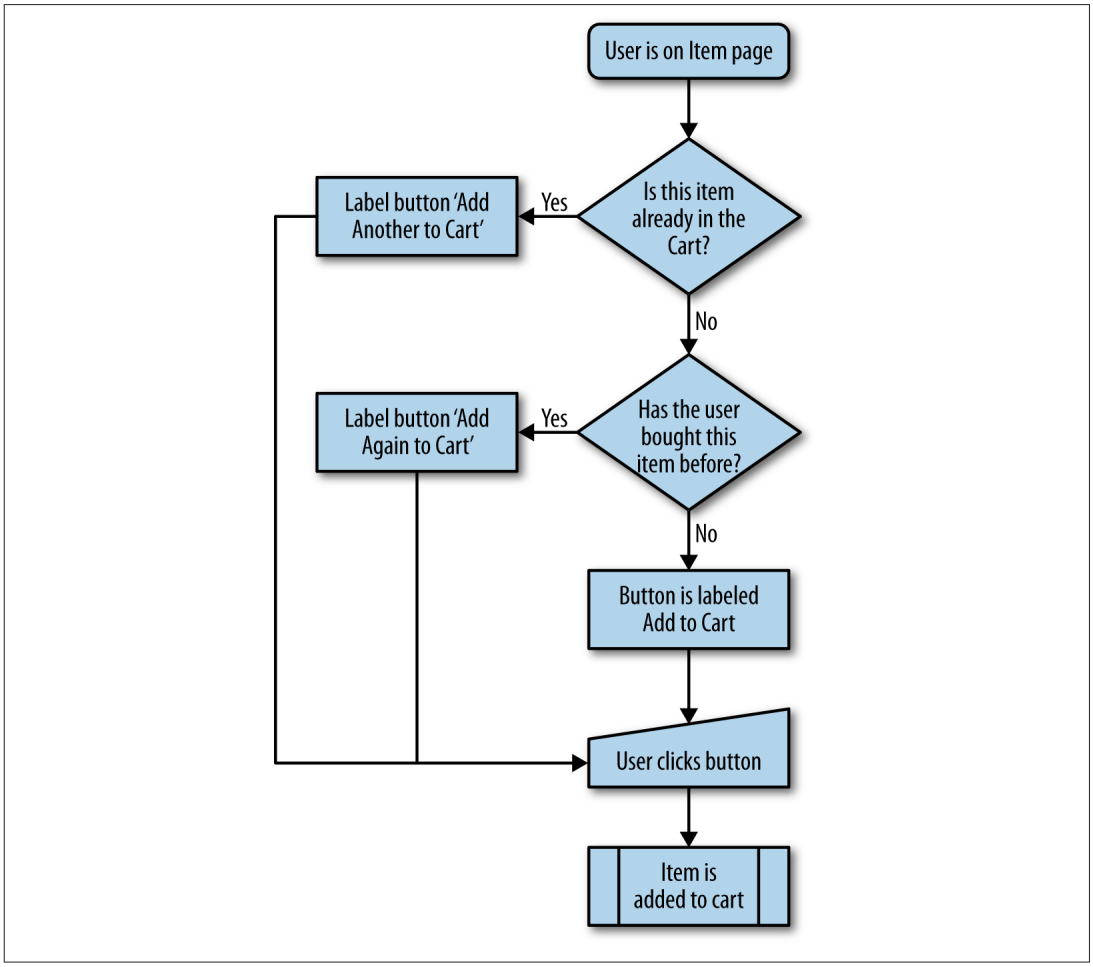

Figure 3-5. An example of a rules logic diagram.

A rules diagram can help you see the rules in a visual way, which can allow you to notice where actions get (overly) complex. It can also show errors in logic that might be hidden by text alone. You can see the effect of nuanced rules in Figure 3-6.

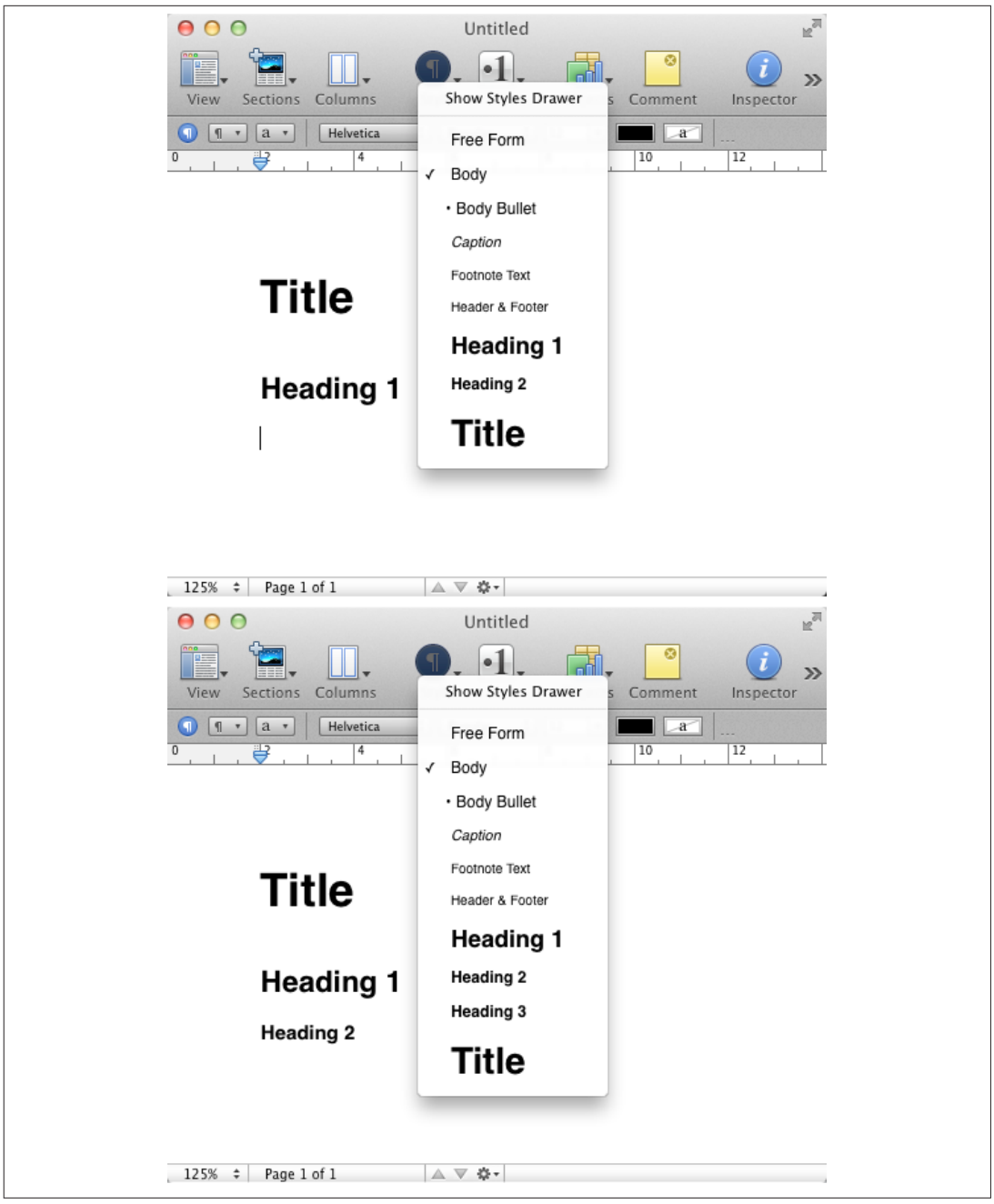

Figure 3-6. Apple's Pages will automatically add smaller heading styles, but only after you've used the smallest displayed style. Heading 3 will only appear as an option once you've used Heading 2. (Courtesy Little Big Details.)

Verbs and Nouns

It can be helpful to think of your entire microinteraction as a sentence. The verbs are the actions that a user can engage in, while the nouns are the objects that enable those actions. For example, a slider enables the raising or lowering of volume. Verbs are what the users can do (raise or lower the volume), and nouns are what they do them with (the slider).

Figure 3-7. When friends Like your run on Facebook, you hear cheers in your headphones while using the Nike+ app. (Courtesy Little Big Details.)

Every object in your microinteraction—every piece of UI chrome, every form element, every control, every LED—is a noun with characteristics and states. The rules define what those characteristics and states are. Take a simple drop-down menu. It generally has two states: open and closed. When open, it reveals its options, which are some of its characteristics. It could have other characteristics, such as the maximum number of options and the maximum length of any option label. It could also have other states, such as opened with hovers, wherein tool tips appear when a user hovers over options. All of these details should be defined by the rules. (Verbs, too, have characteristics; for example, how fast something is accomplished and how long an action takes. These too should be defined in rules.)

Every noun in your microinteraction should be unique. If you have two of the same nouns, consider combining them. Also make sure that any two (or more) nouns that look the same also behave the same. Don't have two similar buttons that act completely different. Objects that behave differently should look differently. Likewise, don't have the same noun work differently in different places. The Back button in Android is famous for being seemingly arbitrary about where it takes the user back to: sometimes

previous modes, sometimes entirely different applications [see Ron Amadeo's article, "Stock Android Isn't Perfect"].

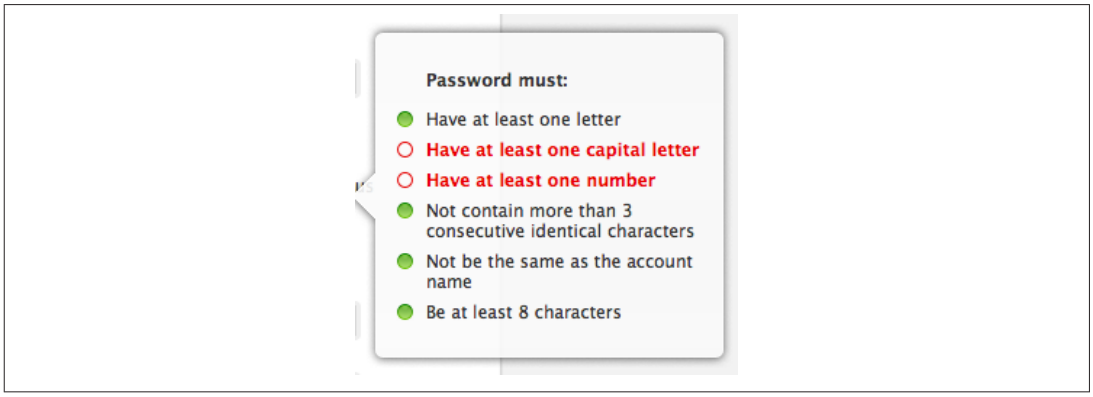

Figure 3-8. When changing your Apple ID password, must-have items are checked off as the user enters them. It reveals the constraints of the microinteraction in a very literal way. (Courtesy Stephen Lewis and Little Big Details.)

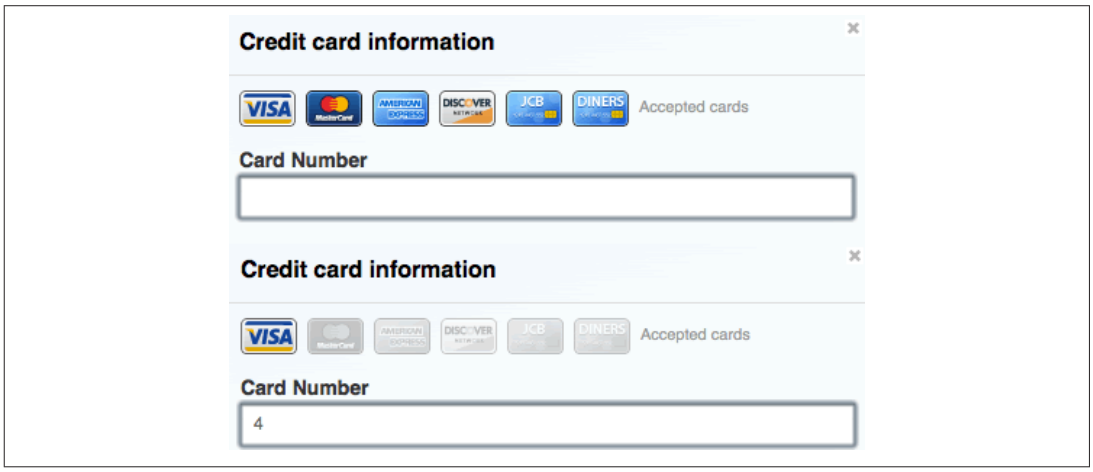

Figure 3-9. GitHub doesn't make users select a credit card. Instead it automatically selects it for them by using the number they type into the field to detect what card type it is. (Courtesy of Little Big Details.)

The best, most elegant microinteractions are often those that allow users a variety of verbs with the fewest possible nouns.

Screens and States

It might be tempting to turn each step of the rules into its own screen; that is, to turn every microinteraction into a wizard-like UI. This works for specific kinds of microinteractions—namely those with defined, discrete steps that are not done often, or are done only once. But for most microinteractions, this would be disruptive and unnecessarily break up the flow of the activity. It's much better to make use of state changes instead. In this way, we use progressive disclosure to reveal only what is necessary at that moment to make a decision or manipulate a control without loading an entirely new screen (see Figure 3-10 for an example).

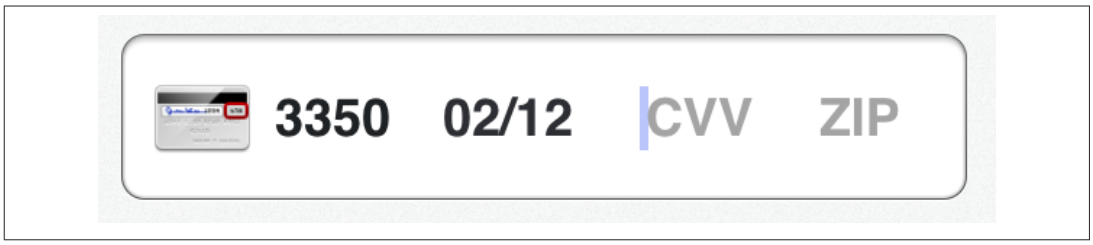

Figure 3-10. When it comes time to enter the CVV number on the Square iOS app, the image of the credit card flips over so that you can immediately see where the number would be. (Courtesy Dion Almaer.)

As the user steps through the rules, the objects (nouns) inside the microinteraction can (and likely will) change to reflect those changes in time. Each of these is a state that should be designed.

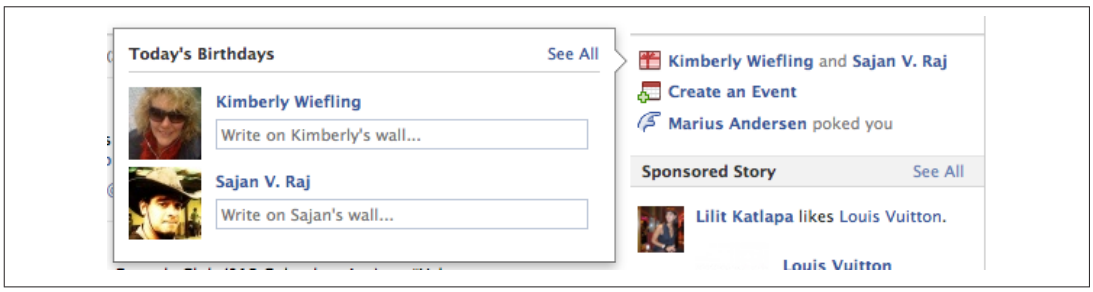

Figure 3-11. If multiple friends have their birthdays on the same day, Facebook's birthday microinteraction lets you write on both of their walls at the same time. (Courtesy Marina Janeiko and Little Big Details.)

Any objects the user can interact with can have (at least) three states:

An invitation/default state
: This is when the user first finds the object. This is also where prepopulated data can be deployed.

Activated state
: What is the object doing while the user is interacting with it?

Updated state
: What happens when the user stops interacting with the object?

Let's take a simple drag-and-drop as an example. An object's initial/default state should look draggable. Or, barring that, the object (and/or the cursor) should have a hover state that indicates the object can be dragged. Then the object should likely have another state while being dragged. (It's also possible the screen itself [another noun] at this point has a different state, indicating where the object could be dropped.) And finally, a state when it is at last dropped, which might be simply to return to the default state.

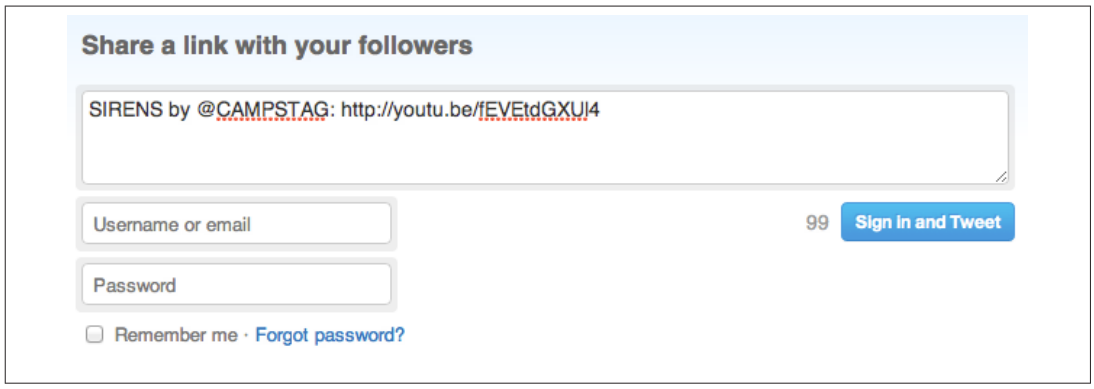

Figure 3-12. On Twitter, the button to share a link has two idle states: signed in and not signed in. If not signed in, the button allows users to do both at once. (Courtesy Rich Dooley and Little Big Details.)

A designer of microinteractions pays attention to each state, namely because each state can convey information to the user about what is happening—even if what is happening is nothing.

Constraints

The rules have to take into account business, environmental, and technical constraints. These can include, but certainly aren't limited to:

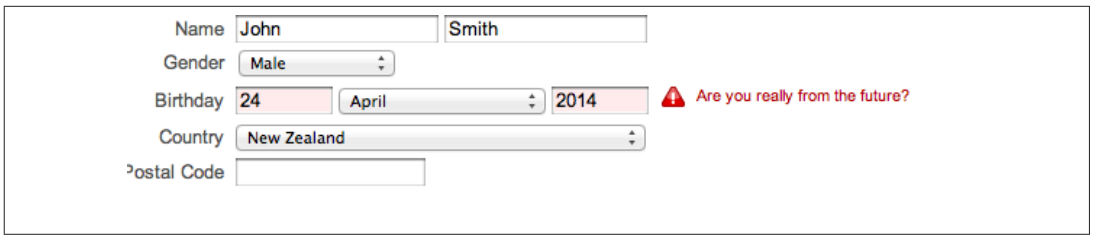

Figure 3-13. Yahoo! has a sign up microinteraction that won't let you put in a future date. Making that field a drop-down with only acceptable years would prevent this error entirely. (Courtesy Little Big Details.)

- *What input and output methods are available*. Is there a keyboard? A speaker?
- *What is the type or range of any input*. For example, the number of characters allowed in a password, or the maximum volume a user can turn the sound up to.
- *What is expensive*. Not just what costs money (such as access to certain data services, as in Figure 3-14), but also what is expensive from a resources standpoint. Perhaps doing a call to the server every 10 seconds would be a massive hit to the server load and drain the device battery too quickly.
- *What kind of data is available*. What can be collected from sensors? What services/APIs can we access to get information about location, news, weather, time, etc.
- *What kind of data can be collected*. What personal (behavioral) data can be collected and used?

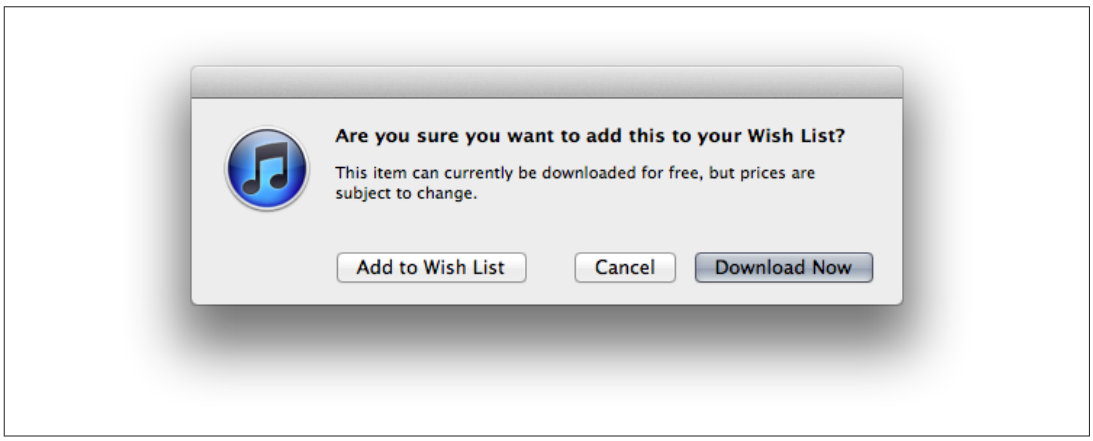

Figure 3-14. When trying to add a free item to a Wish List, iTunes lets you know you can just download it for free instead. (Courtesy Little Big Details.)

These last two constraints allow you to not start from zero.

Don't Start from Zero

After the trigger has been initiated, the first question for any microinteraction should be: what do I know about the user and the context? You almost always know something, and that something can be used to improve the microinteraction (Figure 3-15).

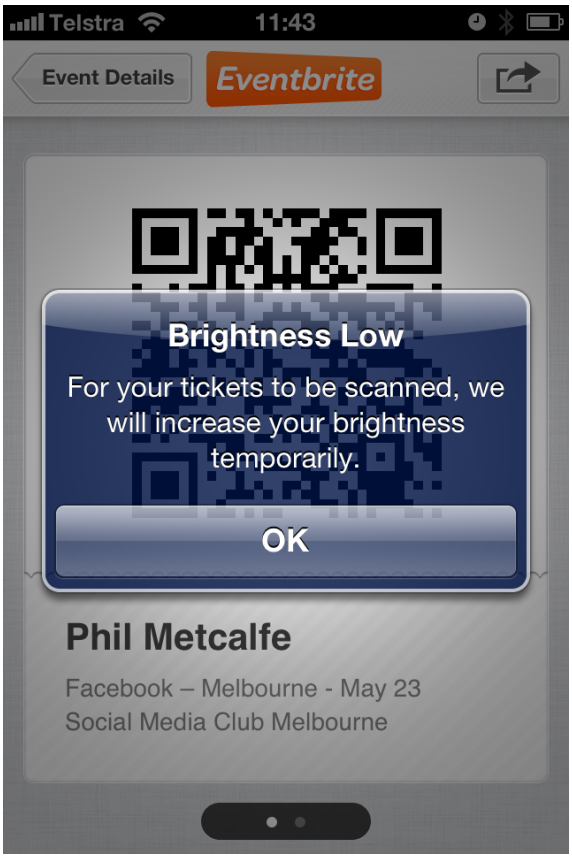

Figure 3-15. The Eventbrite iOS app increases the brightness of the Mobile Ticket screen for easier scanning of the QR code. Useful for the context. The alert is probably unnecessary, however. (Courtesy Phil Metcalfe and Little Big Details.)

Some examples of data that could be used:

- What platform/device is being used
- The time of day
- The noise in the room
- How long since the microinteraction was last used
- Is the user in a meeting

- Is the user alone
- The battery life
- The location and/or direction
- What the user has done in the past

Data can even be useful when it doesn't come directly from the user (Figure 3-16).

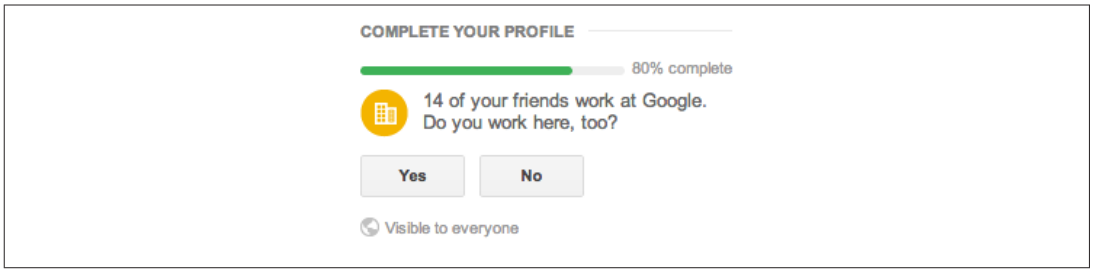

Figure 3-16. Google+ guesses where you work based on your friends' employment. (Courtesy Artem Gassan and Little Big Details.)

That last piece of data—which may be the most important one—relies on collecting information about user behavior, but we're long since past the point where this should be an issue from a system resources point of view; even low-powered appliances have enough memory and processing power to do it. It's just whether or not human resources (developers) can be convinced it's worthwhile. (It is.) Of course, designers should be cognizant of privacy; if the microinteraction deals with sensitive subject matter such as medical information, you might reconsider collecting personal behavior. Ask: could the information that the microinteraction collects be used to embarrass, shame, or endanger users? If so, don't collect it. It's better to have a depersonalized experience than one that is fraught with fear of exposure.

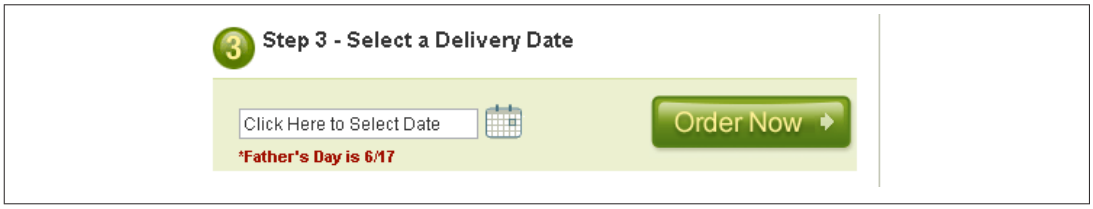

Figure 3-17. Pro Flowers uses the date to show you the next big holiday when selecting a delivery date. (Courtesy Gabriel Henrique and Little Big Details.)

Many of these pieces of data can be used in combination: at 10:00 every day, the user does X, so perhaps when the microinteraction is triggered at that time, offer her X. Or every time the user is in a particular location that he hasn't been to in a while, he does

X. Or every time the user logs in from her mobile device, she's interested in seeing Y. You can see an example of this in Figure 3-18.

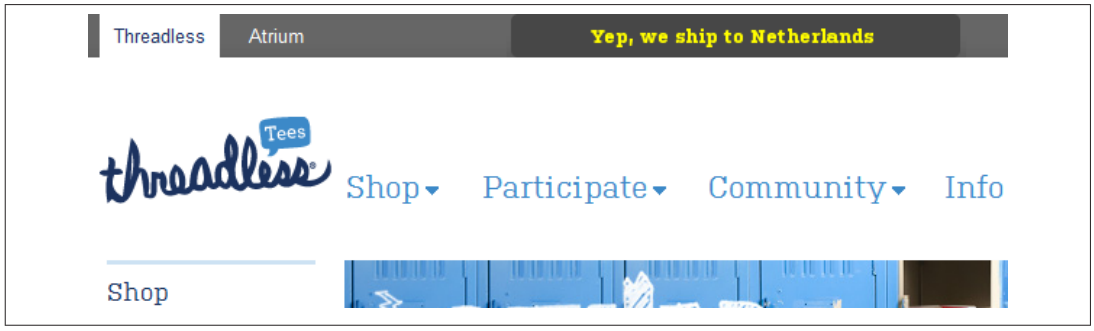

Figure 3-18. Threadless lets you know when you first land on the site whether it can ship to the country you're in or not. (Courtesy Little Big Details.)

The point is to use the context and previous behavior (if any) to predict or enhance the microinteraction (Figure 3-19). This data collection can be thought of as ongoing user research; with some analysis you can see how people are using the microinteraction and adjust accordingly. For example, by collecting behavioral data, you might discover that power users could employ an invisible trigger to get them to a certain point in the rules. Navigation app Waze lets power users slide (instead of push) a button to get directly to Navigation, saving two taps.

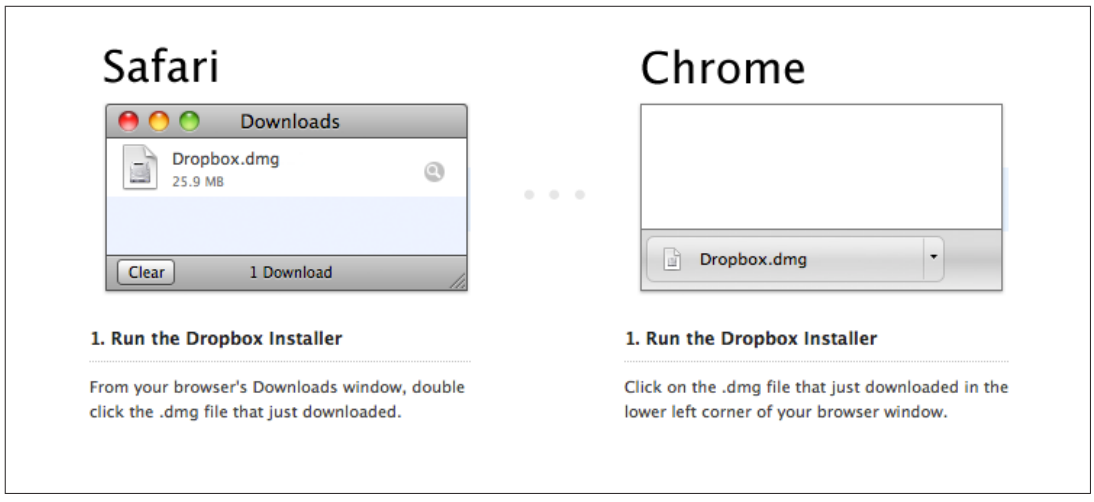

Figure 3-19. Dropbox changes the download instructions based on which browser you're using. (Courtesy Mikko Leino and Little Big Details.)

Absorb Complexity

Larry Tesler, the inventor of cut and paste whom we met back in Chapter 1, came up with an axiom that is important to keep in mind when designing rules: Tesler's Law of the Conservation of Complexity. Tesler's Law, briefly stated, says that all activities have an inherent complexity; there is a point beyond which you cannot simplify a process any further. The only question then becomes what to do with that complexity. Either the system handles it and thus removes control from the user, or else the user handles it, pushing more decisions—yet more control—onto the user.

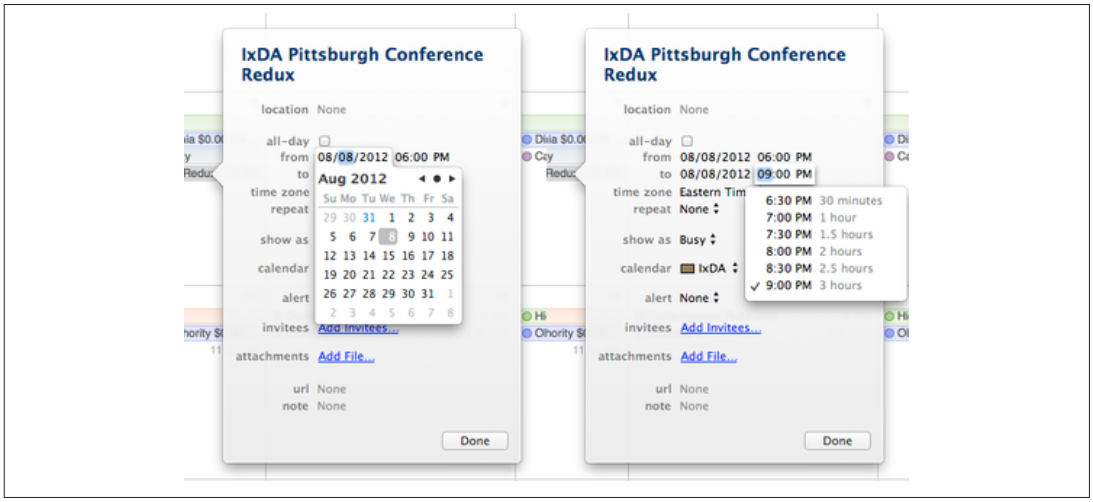

Figure 3-20. Even in the clunky iCal, there is a nice rule in the selection of a time microinteraction. Rather than have you do the math to figure out how long an event would be, iCal shows you event duration when selecting the end time. It's an effective use of microcopy. (Courtesy Jack Moffett.)

For microinteractions, you're going to want to err on the side of removing control and having the microinteraction handle most of the decision making. One caveat to this is that some microinteractions are completely about giving control to the user, but even then there is likely to be complexity that the system should handle (Figure 3-21).

Start by figuring out where the core complexity lies, then decide which parts of that the user might like to have, and when in the overall process. Then, if control is absolutely necessary, provide it at that time (Figure 3-22).

Computers are simply much better at handling some kinds of complexity than humans. If any of these are in your microinteraction, have the system handle it:

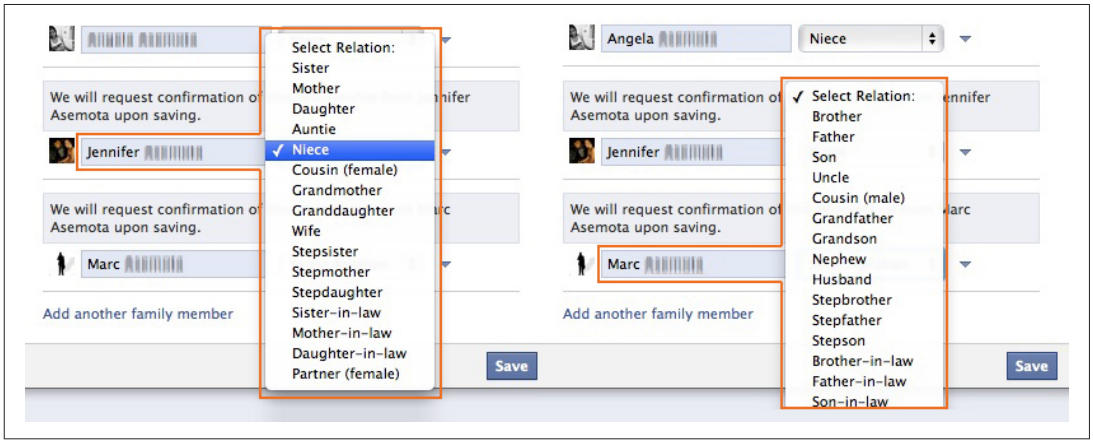

Figure 3-21. When you add a new family member on Facebook, Facebook automatically recognizes the chosen family member's gender and adjusts the list of possible familial relationships in the list box accordingly. (Courtesy Stefan Asemota and Little Big Details.)

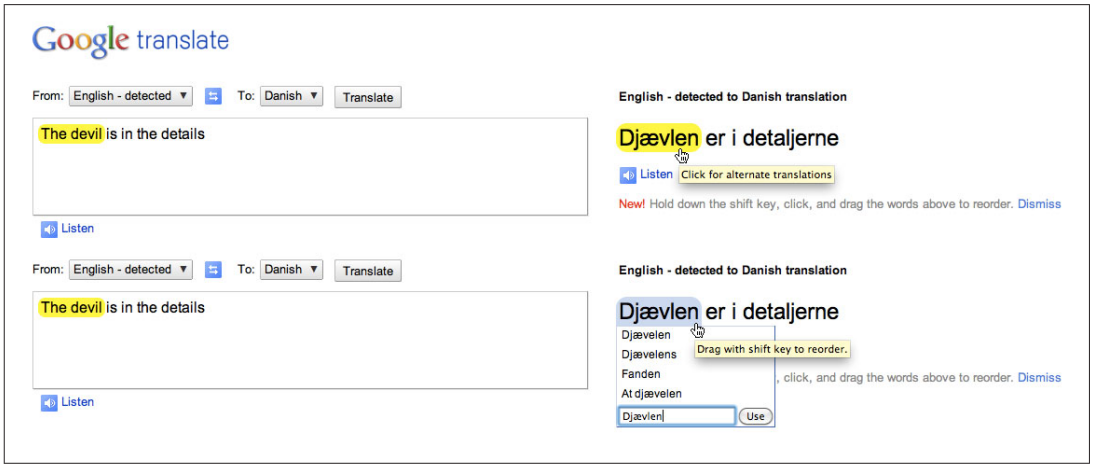

Figure 3-22. When hovering over the translation in Google Translate, it highlights the translated phrase in the original text. You can get alternate translations, but only by clicking on the translated text. (Courtesy Shruti Ramiah and Little Big Details.)

- Rapidly performing computation and calculations
- Doing multiple tasks simultaneously
- Unfailingly remembering things
- Detecting complicated patterns
- Searching through large datasets for particular item(s)

Of course, removing complexity means you must be smart about the choices you do offer and the defaults you have.

Limited Options and Smart Defaults

The more options that you give a user, the more rules a microinteraction has to have, and in general, fewer rules make for better, more understandable microinteractions. This means limiting the choices you give to the user and instead presenting smart defaults.

With microinteractions, a good practice is to emphasize (or perform automatically) the next action the user is most likely to take. This emphasis can be can be done by removing any other options, or just by visual means (making the button large, for instance). As game designer Jesse Schell put it in his book *The Art of Game Design* (CRC Press), "If you can control where someone is going to look, you can control where they are going to go."

Knowing the next likely step is also valuable in that you can perform or present that step automatically, without the user having to do anything else (see Figures 3-23 and 3-24). This is one way to link microinteractions together (see the section "Orchestrating Microinteractions" on page 137 in Chapter 6).

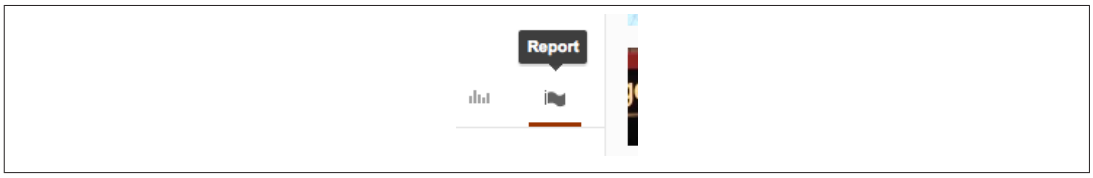

Figure 3-23. Clicking the Report button on YouTube automatically stops the video you're about to report. It performs the next likely action for you. (Courtesy Aaron Laibson and Little Big Details.)

Every option a user has is at least another rule, so *the best way to keep your rules to a minimum is to limit options*. In short, be ruthless in eliminating options. Microinteractions do one thing well, so ideally the user would have no options, just smart defaults throughout the entire microinteraction. Everyone does one action, and that action plays out: from Rule 1 to Rule 2 to Rule 3. This is what made Google's search box the most effective (or at least the most used online) microinteraction of the early 21st century. Everyone followed the same rules:

- Enter text and press (the emphasized) search button.
- Show search results.

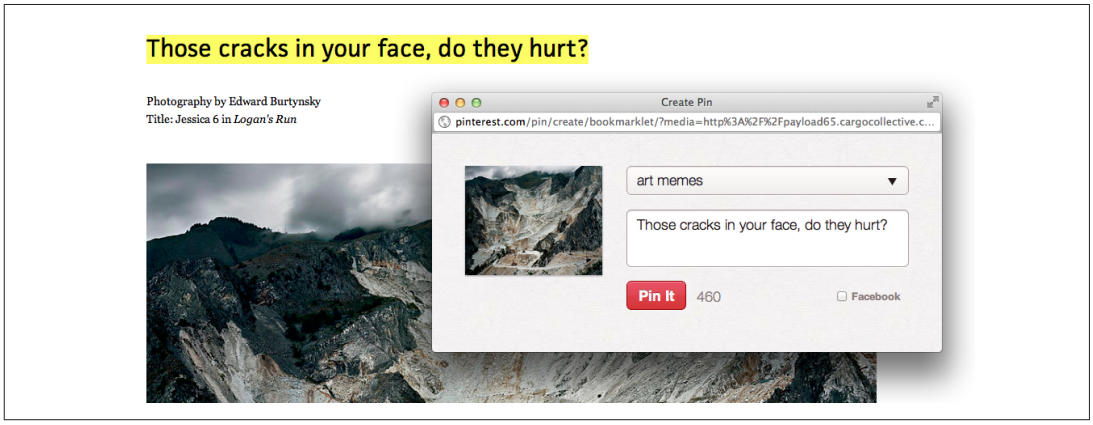

Figure 3-24. Any selected text on a page will prepopulate the caption field when adding it to Pinterest. (Courtesy Louisa Fosco and Little Big Details.)

Of course, even here Google added an option: the I'm Feeling Lucky button, which took you directly to the top search result. I'm Feeling Lucky was only used by 1% of users...and reportedly cost Google $100 million a year in lost ad revenues. In 2010, Google effectively killed I'm Feeling Lucky when it introduced Google Instant, which immediately started showing search results as you type, so there is no chance to press the I'm Feeling Lucky button.[3] Now the rules look like this:

- Enter text.
- Show search results.

It literally cannot get any simpler, unless at some point in the future Google is able to guess what you want to search on and immediately shows you results.

For microinteractions, more than one major option is probably too many. This is not to say you cannot have choices, such as a temperature setting (hot, warm, cold), but rather more than one option that radically changes the rules is ill advised. It's likely that this kind of change puts the microinteraction into a different mode (see Chapter 5). One common example of this is the Forgot Your Password? Mode that many login microinteractions have. Clicking that link takes the user into a different mode that hopefully, eventually takes the user back to the main mode to enter the remembered password.

If you are going to make a default decision for a user, in some instances there should be some indication of what that decision is. One example is Apple's Calendar notifications. When a calendar notification appears (e.g., "Meeting in 15 minutes") there is a Snooze button the user can press. However, there is no indication of the duration of that snooze (as it turns out, it's, in my opinion, an overly long 15-minute snooze) and there's no way

3. Nicholas Carlson, "Google Just Killed The 'I'm Feeling Lucky Button,'" *Business Insider*, September 8, 2010.

to change this default. "Snooze 15 Minutes" would be a better button label: one that indicates what the rule is.

The most prominent default should be the action that most people do most of the time. Even if you decide that this shouldn't be automatically done for the user, it should be visually prominent. The most common example of this are OK/Cancel buttons. Cancel is likely pressed considerably less often than OK, so OK should be more easily seen (larger and/or colored). And don't forget the Return key (if there is one). Pressing Return should perform the default action.

If you have to present a choice to the user, remember that how you present that choice can affect what is chosen. Items at the top and bottom of a list are better recalled than those in the middle. A highlighted option is more often selected than one that is not. And if the user has to make a series of decisions, start with simpler, broader decisions, and move toward more detailed options. Colleen Roller, Vice President of Usability for Bank of America Merrill Lynch, rightly says that, "People feel most confident in their decisions when they understand the available options and can comfortably compare and evaluate each one. It's easiest to evaluate the options when there are only a few of them, and they are easily distinguishable from each other."[4]

Since every option means (at least) one other rule (and remember we're trying to keep rules to as few as possible), the options you present to a user have to be *meaningful*. Meaningful choices affect how the user achieves the goal of the microinteraction—or even what the goal is. An example of a meaningful choice might be to sign in via Facebook or to enter a username/password. Nonmeaningful choices are those that don't affect the outcome no matter what is chosen. Amazon's Kindle app makes users select what color highlight they want to highlight passages in, even though you can't search or export by highlight color; it's only marginally meaningful and should probably have been left out of the default microinteraction of highlighting. Ask: is giving this choice to a user going to make the experience more interesting, valuable, or pleasurable? If the answer is no, leave it out.

The elimination of choice should have one beneficial side effect: the removal of many possible edge cases. Edges cases are those challenging-to-resolve problems that occur only occasionally, typically for a small minority of (power) users. Edge cases can cause your microinteraction to warp so that you are designing to accommodate unusual use cases, not the most common. Edge cases are kryptonite for microinteractions, and everything possible should be done to avoid them, including revising rules to make them impossible. For example, if a Year of Birth form field is a text box, it's easy to put in invalid dates, such as those in the future. Remove this edge case by making the field a drop-down menu.

4. "Abundance of Choice and Its Effect on Decision Making," *UX Matters*, December 6, 2010.

Controls and User Input

Most microinteractions have some place for manual user input. What has to be decided is which controls, and how they manifest. Take something as simple as a volume microinteraction. Volume can have three states: louder, quieter, and muted. These could appear as three buttons, a slider, a dial, two buttons, a scroll wheel, a slider and a button, and probably several other variations as well.

With controls, the choice is between operational simplicity and perceived simplicity. Operational simplicity gives every command its own control. In our volume example, this is the three-button solution: one button for Make Louder, one button to Make Quieter, one button for Mute. With perceived simplicity, a single control does multiple actions. For volume, this would mean selecting the slider or scroll-wheel options.

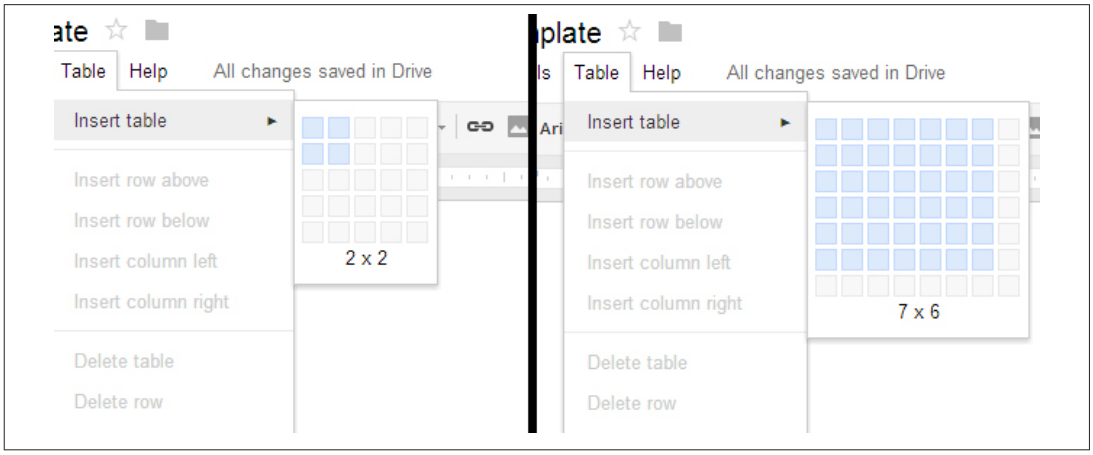

Figure 3-25. Google Drive's Insert Table microinteraction has an expanding hover window that lets users visually determine the size of the table. (Courtesy Kjetil Holmefjord and Little Big Details.)

For microinteractions that will be done repeatedly, err on the side of perceived simplicity, unless it is an action that needs to be done quickly and with no chance of error—for example, the Mute button on a conference phone; combining it with the Make Quieter action would probably be a disaster. For microinteractions that will only be done once or occasionally, err on the side of operational simplicity; display all the options so that little to no foreknowledge is required.

Text fields should be forgiving of what is placed in them and assume that the text could be coming from any number of places, particularly from the clipboard or the user's memory. For example, a form for a telephone number should support users putting in any of the following: (415) 555-1212, 4155551212, or 415-555-1212. Text fields in particular need what system designers call requisite variety—the ability to survive under varied conditions. Often this means "fixing" input behind the scenes in code so that all

the varied inputs conform to the format that the code/database needs (see Figure 3-26 for a poor example and Figure 3-27 for a positive one).

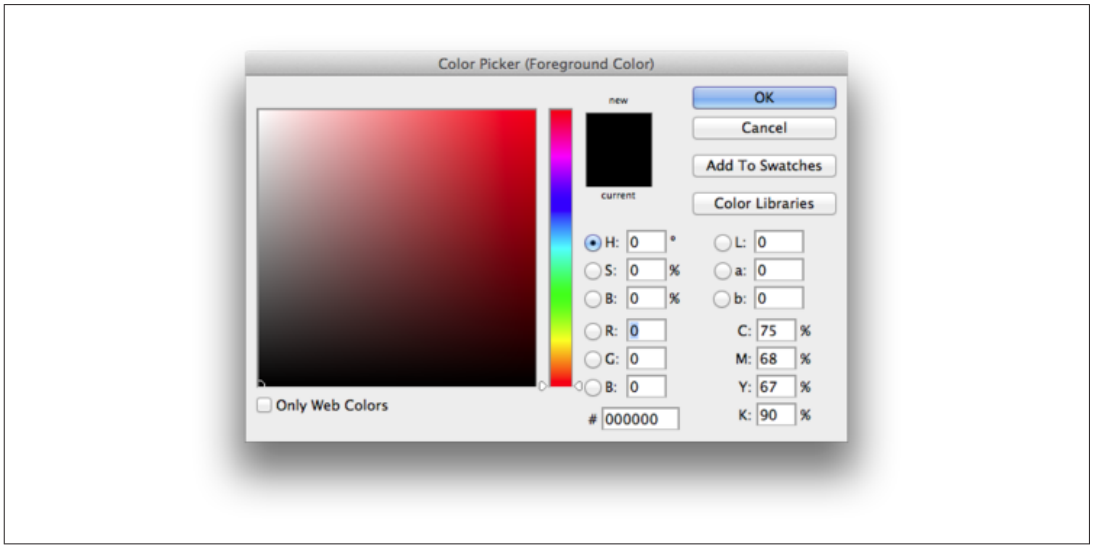

Figure 3-26. Adobe Photoshop's Color Picker microinteraction has a place to enter a hex value. However, it's not smart enough to strip out the # if one is pasted into it. (Courtesy Jack Moffett.)

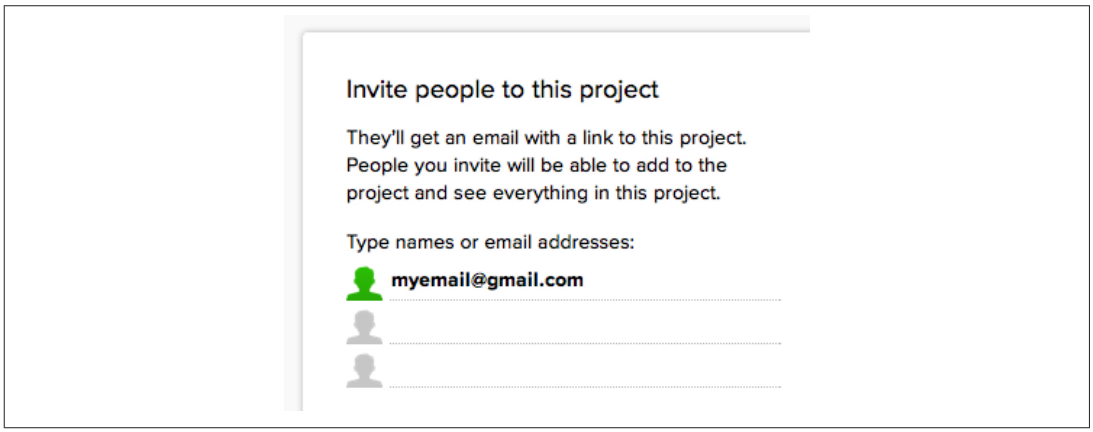

Figure 3-27. 37signals' Basecamp gets it right. When you paste an email ID like "Jane Smith <myemail@gmail.com>," it automatically strips out everything extraneous and leaves just the email address. (Courtesy Harpal Singe and Little Big Details.)

Ordering of lists, such as in a drop-down menu, should be carefully thought out. Sometimes it makes sense to have a predetermined scheme, such as alphabetical or last used. Other times, it might make more sense to be seemingly illogical. For example, if most of your users come from the United States, it makes no sense to have them scroll through the previous 20 letters of the alphabet to reach the U countries—be seemingly irrational and put it at the top of the list or else just make it the default.

Sometimes it makes sense to have redundant controls. Particularly if your microinteraction is going to be used frequently by the same user, it may be wise to design in shortcuts. In desktop software, these have traditionally been keyboard shortcuts such as Command-Q for Quit, while on touchscreen devices and trackpads they have been a gesture (usually multitouch). Just make sure that no significant (to the activity flow) control is buried under a shortcut. For any important action, there should be a visible, manual way to engage with it.

Preventing Errors

One of the main tasks for rules should be error prevention (see Figures 3-28 and 3-29). Microinteractions should follow the Poka-Yoke ("mistake proofing") Principle, which was created in the 1960s by Toyota's legendary industrial engineer Shigeo Shingo. Poka-Yoke says that products and processes should be designed so that it's impossible for users to commit an error because the product/process simply won't allow one. One quick example of Poka-Yoke in action is Apple's Lightning cable. Unlike their previous 30-pin connector (and every USB cord), the Lightning cable can be plugged into the iPhone's or iPad's port facing up or down. Unlike with a USB cable, you can't try to put it in upside down (where it won't fit) because it fits either way.

Figure 3-28. Gmail gives you a notification before sending the mail to see if you've forgotten to attach a file. (Courtesy Little Big Details.)

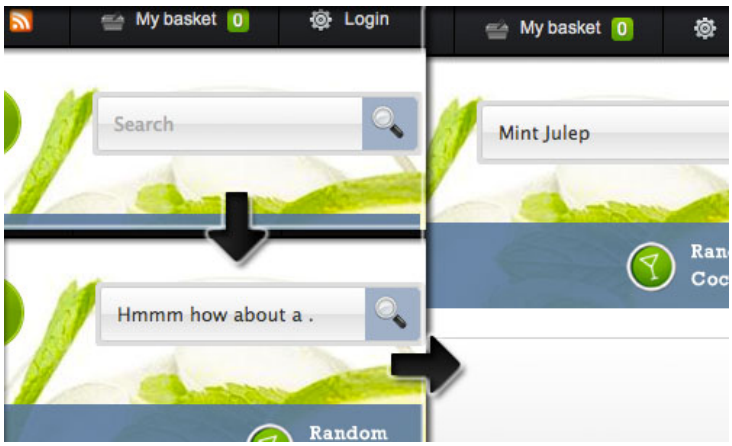

Figure 3-29. If you press the search button on Make Me a Cocktail with nothing in the search field, instead of displaying an error message or nothing, it shows a random cocktail. (Courtesy Nick Wilkins and Little Big Details.)

Similarly, you want to design your microinteraction so that the rules don't allow for mistakes to be made (Figure 3-30). This may mean reducing user control and input, but for microinteractions reducing choice is seldom a bad practice.

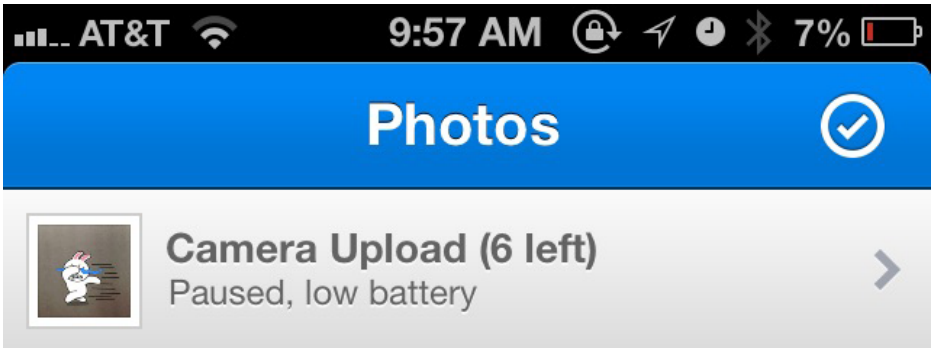

Figure 3-30. Dropbox for iOS pauses uploads when there is a low battery. (Courtesy Little Big Details.)

Ideally, your microinteraction should be designed so that it does not present an error message when the user has done everything right (because the user shouldn't be able to do anything wrong), and only presents an error message when the system itself cannot respond properly. Pop-up error alerts are the tool of the lazy. If an error does occur, the microinteraction should do everything in its power to fix it first (see Figure 3-31).

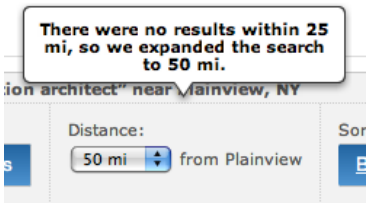

Figure 3-31. Meetup.com adjusts your search results to attempt to correct the error of no found results. (Courtesy Michael J. Morgan and Little Big Details.)

Using the rules, you can also prevent people from using your microinteraction in ways it wasn't intended to be used (see Figures 3-32 and 3-33). For example, you could disallow expletives in comments.

Figure 3-32. What do you love? won't let you enter expletives. It just changes the word to "kittens" and shows those results instead. (Courtesy Zachary Reese.)

Microcopy

Microcopy—labels, instructions, and other tiny pieces of text—is part of understanding the rules. Microcopy is a kind of fixed feedback or feedforward. The entirety of a microinteraction can be a single piece of microcopy: look at Facebook's Like "button," which is based entirely on the word Like in blue text.

A system trigger could cause an essential piece of microcopy to appear when it would be most helpful. For example, on a store's Contact page, a "Sorry, we're closed" message

Figure 3-33. Twitter won't let you tweet the same message twice, mostly to protect its service from abuse. Detecting a duplicate before the user presses Send would be better, although more system-resource intense. (Courtesy Sindre Sorhus and Little Big Details.)

could appear beside the phone number during off hours. And that would be the entire microinteraction right there!

With almost all microinteractions, you want to first make sure any text is absolutely necessary for understanding; instructional copy for microinteractions often isn't. You don't usually have to put "Please log in" at the top of a login form for users to understand that is what they should do. If you do need to include text, make sure it is as short as possible. As Winston Churchill so aptly put it, "The short words are the best, and the old words best of all."

Never use instructional copy when a label will suffice. Tap Next to Continue is unnecessary if there is a button labeled Next or Continue. If a label, such as the name of an album, has to be truncated (for space), there should be a way to see the full title on hover or rollover (desktop/web apps) or tap/click (mobile). Sometimes, particularly with physical buttons, there isn't enough space for a word and manufacturers try to put part of the word on the control, ending up with a letter jumble that resembles a customized license plate. This is not recommended. If a word doesn't fit, consider an icon instead.

Avoid labels that could be misinterpreted. On photo-sharing service Flickr, for instance, the two choices to navigate photos are ← Previous and Next →. However, Previous takes you to the next newer photo, while Next takes you to the next older photo (Figure 3-34).

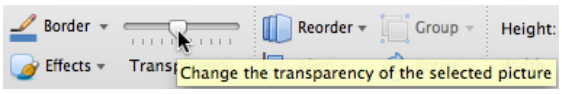

Figure 3-34. Microsoft's Powerpoint transparency slider in the Ribbon. There is no label to indicate if you're making it more or less transparent, and the change doesn't occur until after you release the slider. (Courtesy Jack Moffett.)

The best place for most labels is above what is going to be manipulated. The second best place is on or in the object to be manipulated, as Luke Wroblewski notes in "Top, Right, or Left-Aligned Form Labels" and "Web Form Design: Labels Within Inputs" . This is because it only requires a single-eye fixation to take in both the label and the object. In other words, the eye doesn't have to spend time moving between two objects, which the mind then has to connect.[5] However, the tradition with icons is the label goes below the icon.

Be careful putting a label inside a text form field. When it disappears (as it must because the user clicks into it to put text there), the user can forget what the field is for, and there is no easy way of going back short of clicking out of the text field. It's better in some cases to put the label above (Toy Search) or on a button (Search for Toys) alongside, with examples (e.g., "board games, Lego, or dolls") in the text form field itself.

Be sure that any instructional copy matches the control exactly. For example, don't have the instructions read, "Add items to your shopping cart," then have the button say, Purchase Objects instead of Add Items.

When possible, make text relational instead of exact, particularly dates and times. "Three hours ago" is much easier to understand than showing a date and time stamp, which causes users to make translations and calculations in their head as to when that was. (Of course, sometimes an exact date or time is necessary and shouldn't be obscured.)

Avoid double (or more!) negatives, unless your intention is to confuse or deliberately mislead people. "If you don't want to unsubscribe to our email newsletter, don't uncheck this box."

Algorithms

In 1832, a 17-year-old self-taught son of a shoemaker had a vision of how "a mind most readily accumulates knowledge ... that man's mind works by means of some mechanism." Twenty-two years later, as a university professor, this former child prodigy published his masterpiece: *An Investigation of the Laws of Thought, On Which Are Founded the Mathematical Theories of Logic and Probability*. (Like many masterpieces, it was criticized, dismissed, or simply ignored when it was first published.) That professor's name was George Boole, and he was the father of what we now know of as Boolean logic.

5. For more on eye fixations, see J. Edward Russo, "Eye Fixations Can Save the World: A Critical Evaluation" and "A Comparison Between Eye Fixations and Other Information Processing Methodologies," in *Advances in Consumer Research* Volume 05. 561–570 (1978).

Figure 3-35. Budge's setting screen for To Do Reminders uses clear copy and choices to make what could have been a boring form interesting. (Courtesy Paula Te and Little Big Details.)

Boole devised a kind of linguistic algebra, in which the three basic operations are AND, OR, and NOT. These operations form the basis for generating algorithms. Algorithms are, in the words of Christopher Steiner in *Automate This: How Algorithms Came to Rule Our World* (Portfolio Hardcover):

> Giant decision trees composed of one binary decision after another. Almost everything we do, from driving a car to trading a stock to picking a spouse, can be broken down to a string of binary decisions based on binary input.
>
> [...]
>
> At its core, an algorithm is a set of instructions to be carried out perfunctorily to achieve an ideal result. Information goes into a given algorithm, answers come out.

Although the rules could, in a meta fashion, be thought of algorithmically, some microinteractions depend on algorithms to run. For example, take search. What appears in autofill—not to mention the order of the results themselves—is all generated by an algorithm (Figure 3-36). Recommendations, driving directions, and most emailed/read

are all generated algorithmically. Some branded elements, such as Nike FuelBand's NikeFuel points, are based on an algorithm, as is the custom color picker in FiftyThree's outstanding iPad app, Paper.[6]

Traditionally, these algorithms have all been generated by engineers, but as more and more products come to rely on algorithms, it behooves designers to get involved in their design. After all, a beautiful search microinteraction is meaningless without valuable search results.

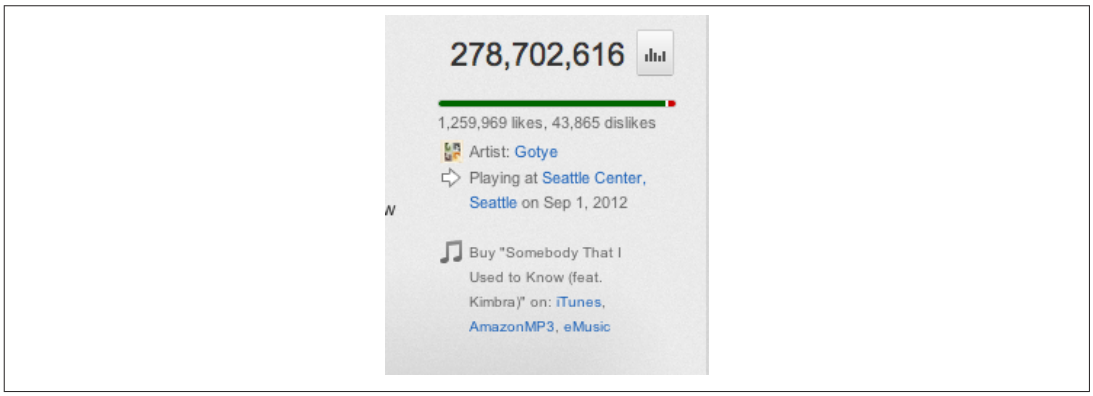

Figure 3-36. If you're watching a music video on YouTube, it algorithmically matches your location to the artist's touring schedule. (Courtesy of Nanakoe and Little Big Details.)

While the code behind algorithms is far too complex to get into here, defining the algorithm is. There are four major parts to any algorithm:

Sequence
: What are the steps in the process? What item comes before what? Are there any conditionals, where an action is dependent on a particular condition? For a device like the Nike FuelBand, this might be something like: for every two steps (as measured by an accelerometer in the hardware), add one to NikeFuel.

Decisions
: These are usually in the form of if ... then statements. For example, if the time is 00:00, then reset.

Repetitions
: How does the algorithm loop? This can be the whole algorithm, or just a particular sequence. For example, while the user is typing in the search field, update search results every time there is a new letter.

6. See "The Magical Tech Behind Paper For iPad's Color-Mixing Perfection," by Chris Dannen in *Fast Company*, November 8, 2012.

Variables
: Variables are containers for the data that powers algorithms. Defining these will allow you to tweak the algorithm without having to rewrite it entirely. Number of Search Results could be a variable, as could Number of Steps Taken. Variables are numeric, alphabetic (text), or logical (true/false).

To put this all together, let's say a microinteraction involves displaying music recommendations. The steps in the sequence are the kinds of music you want to show, and in what order. Are they all from one genre? Does new music take priority over old? Decisions might include: has the user ever listened to this artist before? If so, do not recommend. The algorithm might loop until all the recommendations are filled. And variables could be genre, artist, album, listened to, similar to, tempo, and a whole host of possible characteristics one could use to match music. Variables could also include values such as the percentage of new music to old, and the total number of recommendations to show.

It can be helpful for users to know what data/variables are being acted upon in an algorithm, so that they can manually adjust them if possible. For example, knowing how your FuelBand adds FuelPoints would be valuable so that users could increase their activity appropriately. As it is now, it's a bit of a mystery. Of course, some algorithms, such as Google's search algorithm, are deeply complex and could not be easily explained, especially in microcopy.

What is important to keep in mind from a microinteraction design standpoint is what the user is intending to do, and what data/content is going to be the most valuable, then ensure that those human values get baked into the algorithm. Too often, and too easily, algorithms can be designed solely for efficiency, not for value.

The trouble with rules is that, in the end, they are invisible. Users can only figure them out when something drastic happens, like Apple's change to Save As, or from the feedback the system provides, which is the subject of Chapter 4.

Summary

Rules create a nontechnical model of the microinteraction. They define what can and cannot be done, and in what order.

Rules must reflect constraints. Business, contextual, and technical constraints must be handled.

Don't start from zero. Use what you know about the user, the platform, or the environment to improve the microinteraction.

Remove complexity. Reduce controls to a minimum.

Reduce options and make smart defaults. More options means more rules.

Define states for each object. How do the items change over time or with interactivity?

Err on the side of perceived simplicity. Do more with less.

Use the rules to prevent errors. Make human errors impossible.

Keep copy short. Never use instructional text where a label will suffice.

Help define algorithms. Keep human values in coded decision making.

CHAPTER 4
Feedback

A 56-year-old man punched his fist through the glass and into the electronics of the machine. "Yes, I broke the machine and I'd do it again," he told the security guards. (He was sentenced to 90 days in jail.) Another man, 59-year-old Douglas Batiste, was also arrested for assaulting a machine—by urinating on it. A woman caused $1,800 in damages to another machine by slapping it three times.[1] And 67-year-old Albert Lee Clark, after complaining to an employee and getting no satisfaction, went to his car and got his gun. He came back inside and shot the machine several times.[2]

What device is causing so much rage? Slot machines.

Slot machines are a multi-billion-dollar business. Slot machines take in $7 out of every $10 spent on gambling. Collectively, the money they generate is in the tens of billions,

1. Nir, Sarah Maslin, "Failing to Hit Jackpot, and Hitting Machine Instead," The *New York Times*, July 13, 2012.
2. "Man charged with shooting slot machine," Associated Press, February 13, 2012.

far surpassing the revenue of other forms of entertainment, such as movies, video games, and even pornography.[3] The reason that slot machines—microinteraction devices for sure—work so well at taking money from people is because of the feedback they provide. Most (read: all) of this feedback is insidious, designed specifically to keep people playing for as long as possible.

If you are the statistical anomaly who has never seen or played a slot machine, they work like this: you put coins, bills, or (in newer machines) paper tickets with barcodes into the machine. Pushing a button, tapping the touchscreen, or pulling a lever (the trigger) causes three (or more) seemingly independent "tumblers" to spin. When they stop spinning after a few seconds, if they are aligned in particular ways (if the symbols are the same on all three tumblers, for example), the player is a winner and money drops out of the slot machine. A committed player can do a few hundred (!) spins in an hour.

What really happens is that the rules are rigged in the slot machine's favor; statistically, the slot machine will never pay out more than 90%, so the tumblers never "randomly" do anything, although the feedback makes it seem that way. If the tumblers actually worked the way they appear to work, the payback percentage would be 185% to 297%—obviously an undesirable outcome for casino owners. The outcome is "random but weighted." Blank spaces and low-paying symbols appear more frequently than jackpot symbols—that is, less frequently than they would if the tumbler were actually (instead of just seemingly) random. Thanks to the feedback they get, players have no idea what the actual weighting is; an identical model can be weighted differently than the machine next to it. Since modern slot machines are networked devices, the weighting can even be adjusted from afar, on the fly.[4]

No matter how players trigger the tumblers—by pulling the lever harder, for example—players cannot influence or change the outcome. Some slot machines also have a stop button to stop the tumbler "manually" while they spin. This too doesn't affect the outcome; it only provides an illusion of control.[5]

Not only are the tumblers weighted to prevent winning, but they are designed to incite what gambling researcher Kevin Harrigan calls the Aww Shucks Effect by frequently halting on a "near win," or a failure that's close to a success (see Figure 4-1). For example, the first two tumblers show the same symbol, but the third is blank. These near wins occur 12 times more often than they would by chance alone. Research has shown that

3. Rivlin, Gary, "The Tug of the Newfangled Slot Machines," The *New York Times*, May 9, 2004.

4. Richtel, Matt, "From the Back Office, a Casino Can Change the Slot Machine in Seconds," The *New York Times*, April 12, 2006.

5. All from Kevin Harrigan's "The Design of Slot Machine Games," 2009.

near wins make people want to gamble more by activating the parts of the brain that are associated by wins—even though they didn't win![6]

Figure 4-1. An example of a "near win." (Courtesy Marco Verch.)

When a player does win, the win is usually small, although the feedback is disproportionate to the winning, so that players think they've won big. Lights flash, sounds play. And the sounds! In the *New York Times* profile of slot machine designer Joe Kaminkow, it notes:

> Before Kaminkow's arrival, [slot machine manufacturer] I.G.T.'s games weren't quiet—hardly—but they didn't take full advantage of the power of special effects like "smart sounds"—bright bursts of music. So Kaminkow decreed that every action, every spin of the wheel, every outcome, would have its own unique sound. The typical slot machine featured maybe 15 "sound events" when Kaminkow first arrived at I.G.T. [in 1999]; now that average is closer to 400. And the deeper a player gets into a game, the quicker and usually louder the music.[7]

6. Clark, L, Laurence, A., Astley-Jones, F., Gray, N., "Gambling near-misses enhance motivation to gamble and recruit brain-related circuitry," *Neuron* 61, 2009.
7. Rivlin, Gary, "The Tug of the Newfangled Slot Machines." The *New York Times*.

The slot machine microinteraction is so addictive because it provides, via feedback, *intermittent reinforcement of behavior*. Slot machine players keep performing the same behavior until they are eventually rewarded. With slot machines, if payout was predictable—if the player won every other time, for example—players would quickly get bored or annoyed. What keeps people playing is the very unpredictability of the payouts, plus the promise that very rarely there will be a big jackpot. In general, this is *not* the kind of reinforcement you want for most microinteractions, where you want consistent feedback with positive reinforcement (via feedback) of desirable behavior. Predictability is desirable.

Slot machines teach us that feedback is extremely powerful and can make or break a microinteraction. Visuals and sound combine to make an engaging experience out of what could be a repetitive, dull activity of pulling a lever over and over. Obviously, they do this to their mind-blowingly lucrative benefit and you certainly don't want every microinteraction being like a flashing, noisy slot machine, but the lesson is the same: feedback provides the character, the personality, of the microinteraction.

Feedback Illuminates the Rules

Unlike slot machines, which are designed to deliberately obscure the rules, with microinteractions the true purpose of feedback is to help users understand how the rules of the microinteraction work. If a user pushes a button, something should happen that indicates two things: that the button has been pushed, and what has happened as a result of that button being pushed. Slot machines will certainly tell you the first half (that the lever was pulled), just not the second half (what is happening behind the scenes) because if they did, people probably wouldn't play—or at least not as much. But since feedback doesn't have to tell users how the microinteraction *actually* works—what the rules actually are—the feedback should be just enough for users to make a working mental model of the microinteraction. Along with the affordances of the trigger, feedback should let users know what they can and cannot do with the microinteraction.

One caveat: you can have legitimate, nondeceitful reasons for not wanting users to know how the rules work; for example, users may not need to know every time a sensor is triggered or every time the device goes out to fetch data, only if something significant changes. For example, you don't often need to know when there is no new email message, only when there is a new one. *The first principle of feedback for microinteractions is to not overburden users with feedback*. Ask: what is the least amount of feedback that can be delivered to convey what is going on (Figures 4-2 and 4-3)?

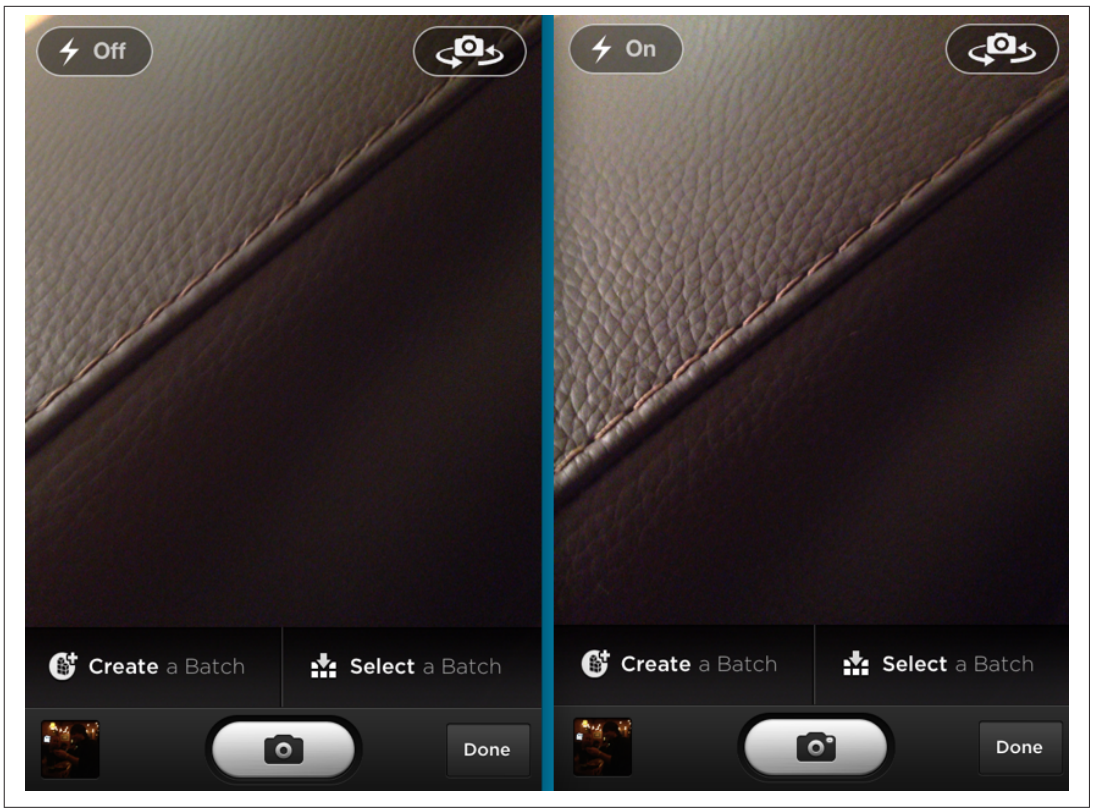

Figure 4-2. In Batch, when the flash is on, the camera icon on the shutter button gets a white flash indicator. (Courtesy Little Big Details.)

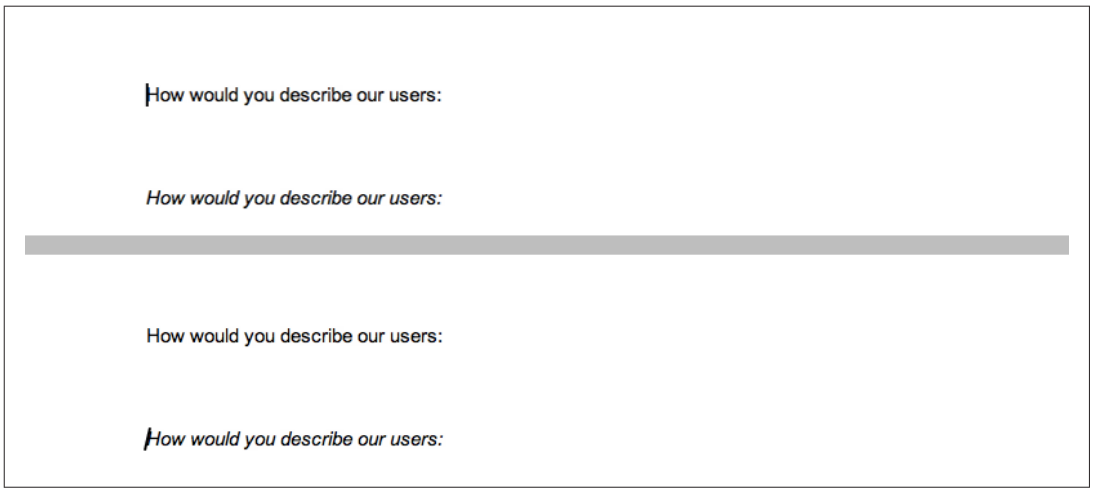

Figure 4-3. Google Docs slants the cursor when you're typing in italics. Microsoft Word does this as well. (Courtesy Gregg Bernstein and Little Big Details.)

Feedback should be driven by need: what does the user need to know and when (how often)? Then it is up to the designer to determine what format that feedback should take: visual, audible, or haptic, or some combination thereof.

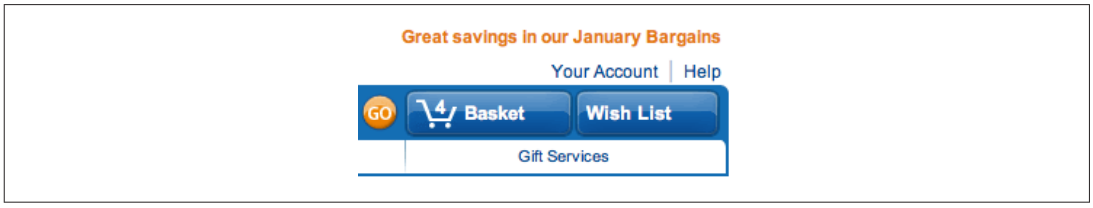

Figure 4-4. Amazon puts the item counter inside the shopping cart button. (Courtesy Matthew Solle and Little Big Details.)

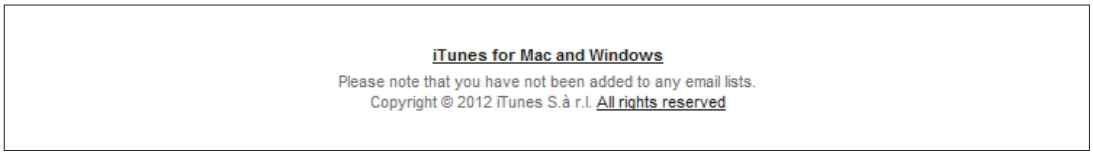

Figure 4-5. Sometimes it's important to indicate what didn't happen. When recommending an app via email, Apple's App Store tells you that you haven't been added to any email lists. (Courtesy Little Big Details.)

Feedback should occur:

- *Immediately after a manual trigger or following/during a manual adjustment of a rule*. All user-initiated actions should be accompanied by a system acknowledgment (see Figure 4-6). Pushing a button should indicate what happened.
- *On any system-initiated triggers in which the state of the microinteraction (or the surrounding feature) has changed significantly*. The significance will vary by context and will have to be determined on a case-by-case basis by the designer. Some microinteractions will (and should) run in the background. An example is an email client checking to see if there are new messages. Users might not need to know every time it checks, but will want to know when there are new messages.
- *Whenever a user reaches the edge (or beyond) of a rule*. This would be the case of an error about to occur. Ideally, this state would never occur, but it's sometimes necessary, such as when a user enters a wrong value (e.g., a password) into a field. Another example is reaching the bottom of a scrolling list when there are no more items to display.
- *Whenever the system cannot execute a command*. For instance, if the microinteraction cannot send a message because the device is offline. One caveat to this is that multiple attempts to execute the command could occur before the feedback that

something is amiss. It might take several tries to connect to a network, for example, and knowing this, you might wait to show an error message until after several attempts have been made.

- *Showing progress on any critical process, particularly if that process will take a long time*. If your microinteraction is about uploading or downloading, for example, it would be appropriate to estimate duration of the process (see Figure 4-7).

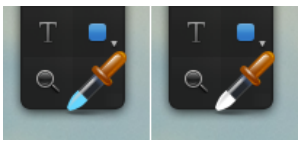

Figure 4-6. Pixelmator's eyedropper tool shows you the color you've chosen inside the pipette. (Courtesy Little Big Details.)

Feedback could occur:

- *At the beginning or end of a process*. For example, after an item has finished downloading.
- *At the beginning or end of a mode or when switching between modes*.

Figure 4-7. Transmit 4 shows in one progress bar both the total transfer and individual transfers. (Courtesy Stef van der Feen and Little Big Details.)

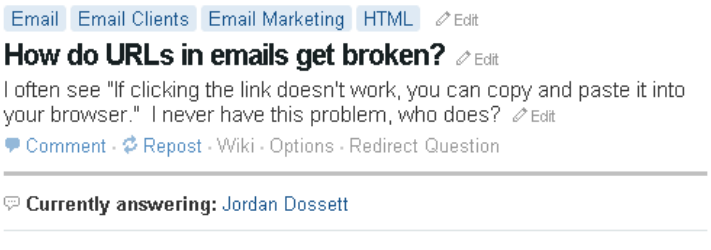

Figure 4-8. On Quora, you can see if someone is answering the question you're looking at. (Courtesy Allison Ko and Little Big Details.)

Always look for moments where the feedback can demystify what the microinteraction is doing; without feedback, the user will never understand the rules.

Feedback Is for Humans

While there is certainly machine-to-machine feedback, the feedback we're most concerned with is communicating to the human beings using the product. For microinteractions, that message is usually one of the following:

- Something has happened
- You did something
- A process has started
- A process has ended
- A process is ongoing
- You can't do that

Once you know what message you want to send, the only decisions remaining are how these messages manifest. The kind of feedback you can provide depends entirely upon the type of hardware the microinteraction is on. On a mobile phone, you might have visual, audible, and haptic feedback possible. On a piece of consumer electronics, feedback could only be visual, in the form of LEDs.

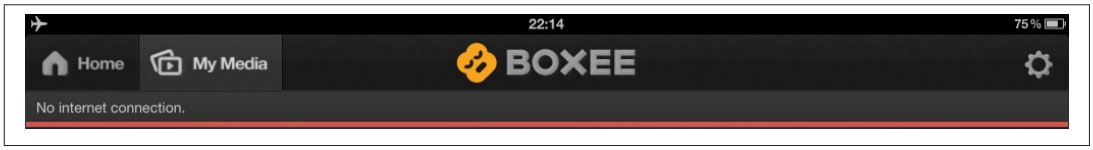

Figure 4-9. Humans respond to faces. The Boxee logo turns orange and "goes to sleep" when there is no Internet connection. (Courtesy Emil Tullstedt and Little Big Details.)

Figure 4-10. The Threadless shopping cart face turns from frowning to happy when you put items in it. (Courtesy Ahmed Alley and Little Big Details.)

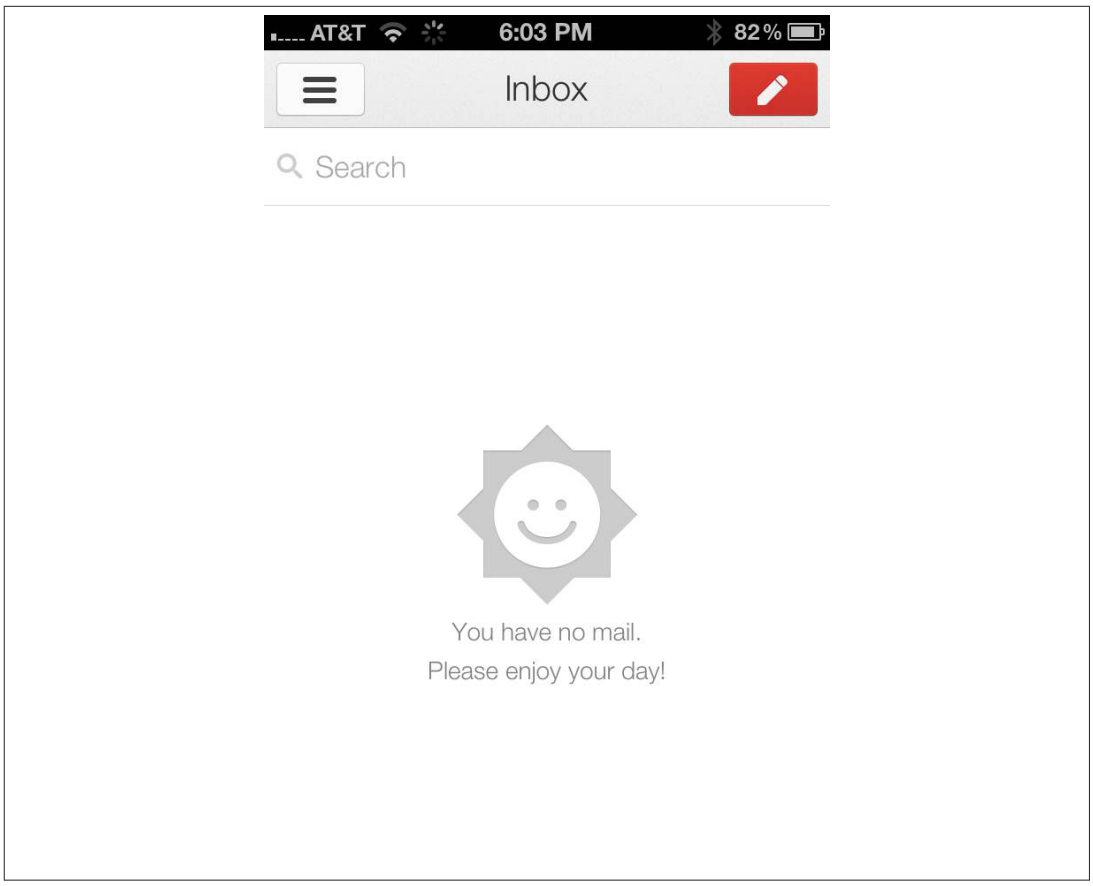

Figure 4-11. The Gmail iPhone app shows what not to do: randomly put a smiley face for a message that isn't necessarily a happy one. (Courtesy Steve Portigal.)

Let's take a microinteraction appliance like a dishwasher as an example. The dishwasher process goes something like this: a user selects a setting, turns the dishwasher on, the dishwasher washes the dishes and stops. If someone opens the dishwasher midprocess, it complains. Now, if the dishwasher has a screen, each of these actions could be accompanied by a message on the screen ("Washing Dishes. 20 minutes until complete."). If there is no screen, there might be only LEDs and sounds to convey these messages. One option might be that an LED blinks while the dishwasher is running, and a chime sounds when the washing cycle is completed.

Text (written) feedback is not always an option (for example, if there is no screen or simply no screen real estate). Once we move past actual words—and let's not forget that a substantial portion of the planet's population is illiterate: 793 million adults, according to the Central Intelligence Agency—we have to convey messages via other means: sound, iconography, images, light, and haptics. Since they are not text (and even words can be vague and slippery), they can be open to interpretation. What does that blinking LED mean? When the icon changes color, what is it trying to convey? Some feedback is clearly

learned over time: when that icon lights up and I click it, I see there is a new message. The "penalty" for clicking (or acting on) any feedback that might be misinterpreted should be none. If I can't guess that the blinking LED means the dishwasher is in use, opening the dishwasher shouldn't spray me with scalding hot water. In fact, neurologically, errors improve performance; how humans learn is when our expectation doesn't match the outcome.

The second principle of feedback is that the best feedback is never arbitrary: it always exists to convey a message that helps users, and there is a deep connection between the action causing the feedback and the feedback itself. Pressing a button to turn on a device and hearing a beep is practically meaningless, as there is no relationship between the trigger (pressing the button) or the resulting action (the device turning on) and the resulting sound. It would be much better to either have a click (the sound of a button being pushed) or some visual/sound cue of the device powering up, such as a note that increases in pitch. Arbitrary feedback makes it harder to connect actions to results, and thus harder for users to understand what is happening. The best microinteractions couple the trigger to the rule to the feedback, so that each feels like a "natural" extension of the other.

Less Is More

The more methods of feedback you use, the more intrusive the feedback is. An animation accompanied by a sound and a haptic buzz is far more attention getting than any of those alone. *The third principle for microinteractions feedback is to convey the most with the least*. Decide what message you wish to convey ("Downloading has begun") then determine what is the least amount of feedback you could provide to convey that message. The more important the feedback is, the more prominent (and multichannel) it should be.

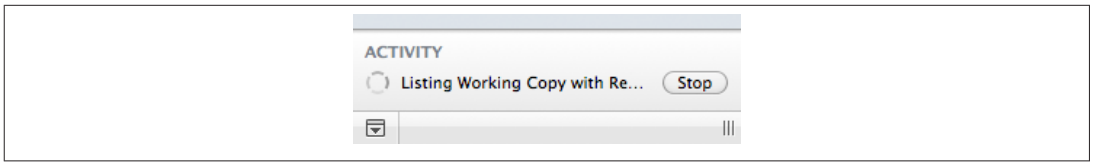

Figure 4-12. In Cornerstone, the number of segments in the spinning activity wheel are equal to the number of processes happening in the background. (Courtesy Yusuf Miles and Little Big Details.)

The fourth principle of feedback is to use the overlooked as a means of message delivery. Many microinteractions contain conventional parts of any interface—as they should. These overlooked parts of the UI—scrollbars, cursors, progress bars, tooltips/hovers, etc.—can be used for feedback delivery. This way, nothing that isn't already there will get added to the screen, but it can communicate slightly more than is usual

(Figure 4-13). For example, a cursor could change color to gray if the user is rolling over an inactive button.

Figure 4-13. OS X Lion's cursor changes to tell you when you can't resize a window in a particular direction. (Courtesy Little Big Details.)

Feedback as a Personality-Delivery Mechanism

Unlike the more utilitarian trigger and any controls for the rules, feedback can be a means of creating a personality for your microinteraction—and for your product as a whole. Feedback can be the moment to inject a little edge or a touch of humor into your microinteraction (see Figures 4-14 and 4-15).

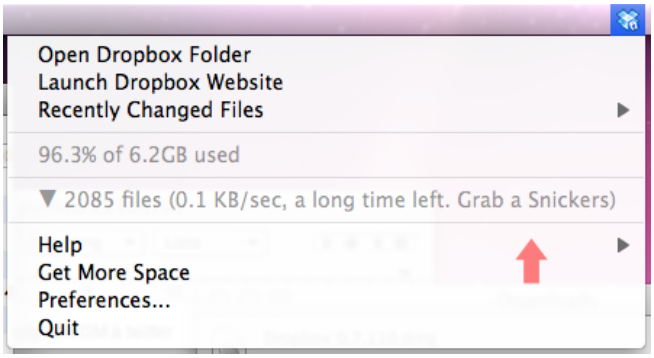

Figure 4-14. When there is a long upload time, Dropbox suggests you eat a candy bar while waiting. (Courtesy John Darke and Little Big Details.)

The reason you'd want to do that is that, as pointed out previously, feedback is for humans. We respond well to human responses to situations, even from machines. Humans anthropomorphize our products already, attributing them motivations and characteristics that they do not possess. Your laptop didn't deliberately crash and your phone isn't mad at you. Designers can use this human tendency to our advantage by deliberately adding personality to products. This works particularly well for microinteractions;

because of their brevity, moments of personality are more likely to be endearing, not intrusive or annoying.

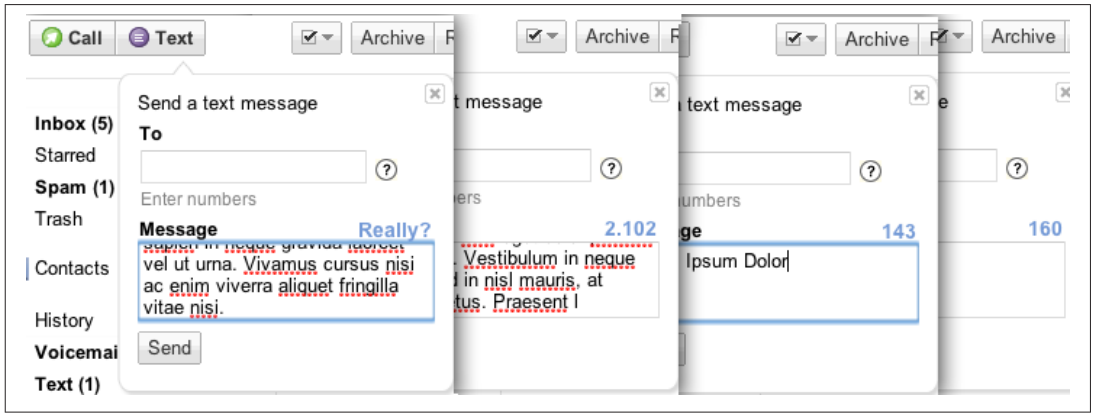

Figure 4-15. If your SMS gets too long, Google Voice stops counting characters and says, "Really?" (Courtesy Zoli Honig and Little Big Details.)

Take Apple's natural-language software agent Siri, for example. Siri easily could have been extremely utilitarian, and indeed, for most answers, "she"—it—is. But for questions with ambiguous or no possible factual responses like "What is the meaning of life?" Siri offers up responses such as "I don't know. But I think there's an app for that." In other words, what could have been potentially an error message ("I'm sorry. I can't answer that.") became something humorous and engaging. Indeed, errors or moments that could be frustrating for users such as a long download are the perfect place to show personality to relieve tension (see Figure 4-16).

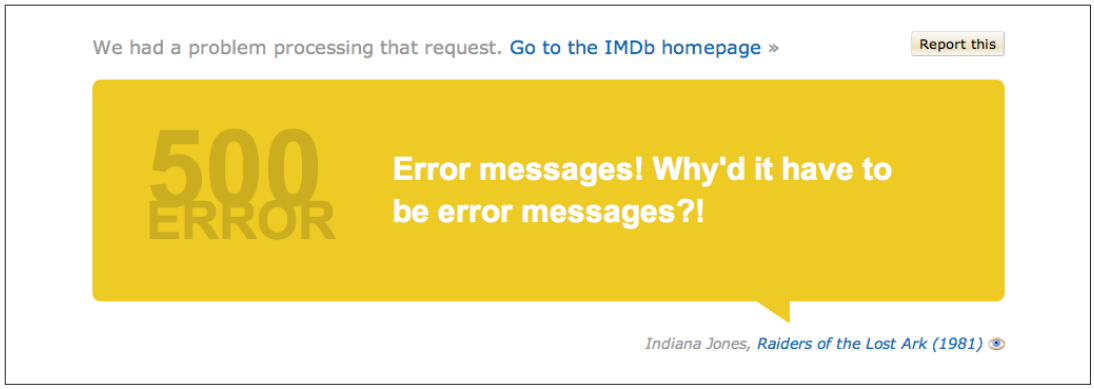

Figure 4-16. For the Internet Movie Database (IMDb), the 500 error message is based on a movie quote. (Courtesy Factor.us and Little Big Details.)

Feedback with personality can, of course, be annoying if not done well or overdone. You probably don't want the login microinteraction giving you attitude every time you

want to log in. And you might not want an app to chastise you if you forget your password: "Forgot your password again? FAIL!" What you should strive for is a *veneer of personality*. In the same way that being too human is creepy for robots—the so-called "uncanny valley"[8]—so too is too much personality detrimental for microinteractions. A little personality goes a long way (Figure 4-17). Making them too human-like not only sets expectations high—users will assume the microinteraction is smarter than it probably is—but can also come across as tone-deaf, creepy, or obnoxious.

Figure 4-17. Twitter for mobile acknowledges typing on a phone is difficult and may cause errors when logging in. (Courtesy Joris Bruijnzeels and Little Big Details.)

8. For a more complete definition and analysis of the uncanny valley, see "The Truth About Robotic's Uncanny Valley: Human-Like Robots and the Uncanny Valley," *Popular Mechanics*. January 20, 2010.

Speaking of creepy, while you want to collect and use behavioral data (and be transparent about what data you're collecting) to improve (or create) the microinteraction over time, being obvious (providing feedback) about collecting that data is a fast way to appear intrusive and predatory. You want people to be delighted with the personalization that data collection can provide, without being disgusted that data collection is going on.

Feedback Methods

We experience feedback through our five senses, but mostly through the three main ways we'll examine here: sight, hearing, and touch.

Visual

Let's face it: most feedback is visual. There's a reason for that, of course, in that we're often looking directly at what we're interacting with, so it is logical to make the feedback visual. Visual feedback can take many forms, from the blinking cursor that indicates where text should go, to text on a screen, to a glowing LED, to a transition between screens.

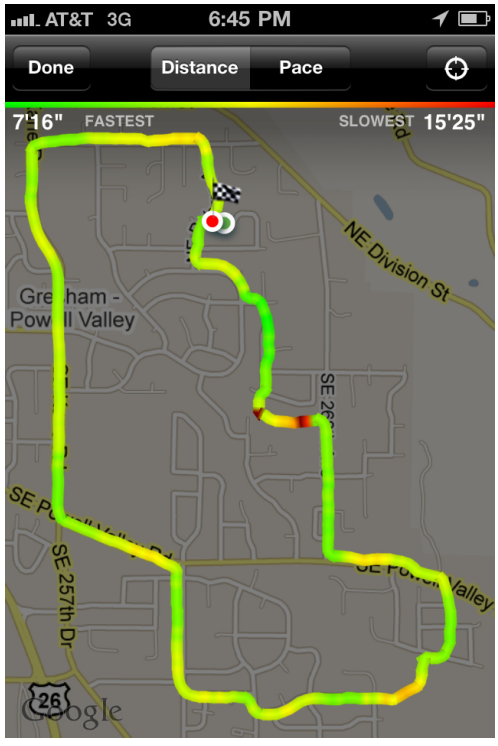

Figure 4-18. Nike+ App shows your slowest and fastest pace on its route map. (Courtesy David Knepprath and Little Big Details.)

Unless no screen or LED is available, assume that your default feedback is visual. Almost every user-initiated action (with the exception of actions users cannot do, such as clicking where there is no target) should be accompanied by visual feedback. With system-initiated triggers and rules, only some should have accompanying visual feedback—namely those that would require human intervention (e.g., an error indicator) or those that provide information a user may want to act upon (e.g., a badge indicating a new voicemail has arrived). Ask what the user needs to see to make a decision, then show that in as subtle a way as possible. Often what the user needs to be aware of is resources: time, effort, unread messages, etc.

Figure 4-19. Navigon app changes its background when you go into a tunnel, as well as indicating how long before you reach the tunnel's end. (Courtesy Little Big Details.)

Don't show redundant visual feedback. For instance, never have a tooltip that mirrors the button label. Any visual feedback must add to clarity, not to clutter. Similarly, don't overdo a visual effect; the more frequent the feedback is, the less intrusive it should be. Don't make something blink unless it *must* be paid attention to.

Figure 4-20. Github uses a tooltip to show the absolute timestamp. (Courtesy Scott W. Bradley and Little Big Details.)

Visual feedback should also ideally occur near or at the point of user input. Don't have an error message appear at the top of the screen when the Submit button is on the bottom. As noted in Chapter 2, when we're attentive to something, our field of vision narrows. Anything outside of that field of vision can be overlooked. If you need to place visual feedback away from the locus of attention, adding movement to it (e.g., having it fade in) can draw attention to it.

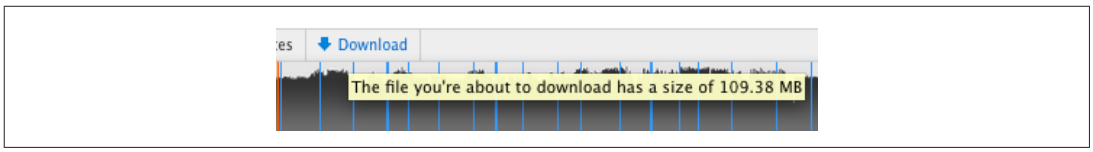

Figure 4-21. SoundCloud places the download size into a tooltip. This only works if the majority of users don't care about download size. (Courtesy David L. Miller and Little Big Details.)

Animation

Our brains respond powerfully to movement, so use animation sparingly. If you can do without animation, avoid it. Your microinteraction will be faster and less cognitively challenging without it. That being said, tiny, brief animations can add interest and convey meaning if done well (Figure 4-22).

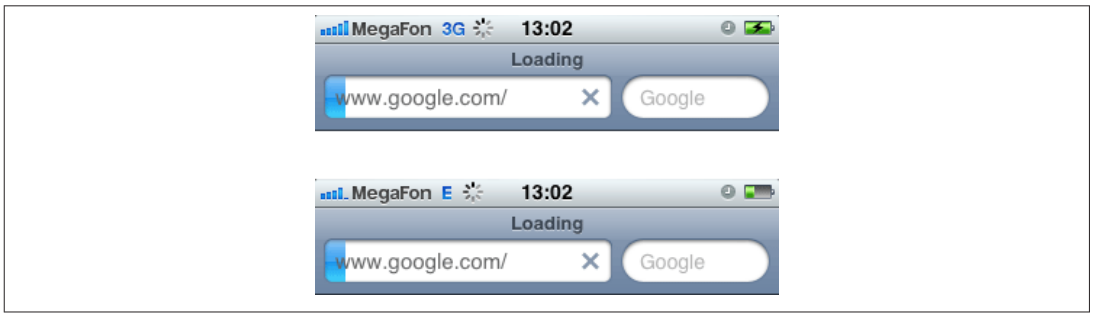

Figure 4-22. On iPhones, the spinner next to the network speed spins faster or slower depending on the network speed. For instance, Edge networks spin slower than 3G ones. (Courtesy Little Big Details.)

The most important part of animation in microinteractions is that it indicates—accurately—a behavioral model of how the microinteraction works. Don't have a panel slide in from the left if it isn't going to slide out to the right, or if the user accesses it by swiping down. Animation for animation's sake is deadly. *The best animations communicate something to the user*: about the structure of the microinteraction, what to look at, what process is happening, etc.

Google's Android engineers Chet Haase and Romain Guy have devised a set of UI characteristics for animation. Animations should be:

Fast
: Do not delay the activity

Smooth
: Stuttering or choppy movements ruin the effect and make the microinteraction seem broken

Natural
: They seemingly obey natural laws, such as gravity and inertia

Simple
: Meaningful, understandable

Purposeful
: Not just as eye candy

On this last point, designer and engineer Bill Scott outlines the reasons for using animation:[9]

- *Maintaining context while changing views*. Scrolling a list or a flipping through a carousel allows you see the previous and next items.
- *Explaining what just happened*. That poof of smoke means the item was deleted.
- *Showing relationships between objects*. For example, animating one item going into another at the end of a drag-and-drop.
- *Focusing attention*. As an item changes value, an animation can make that change more obvious.
- *Improving perceived performance*. Progress bars don't decrease the time needed for a download to happen, but they do make the time seem less grating.

9. "Anti-Pattern: Animation Gone Wild - Borders.com," July 16, 2008.

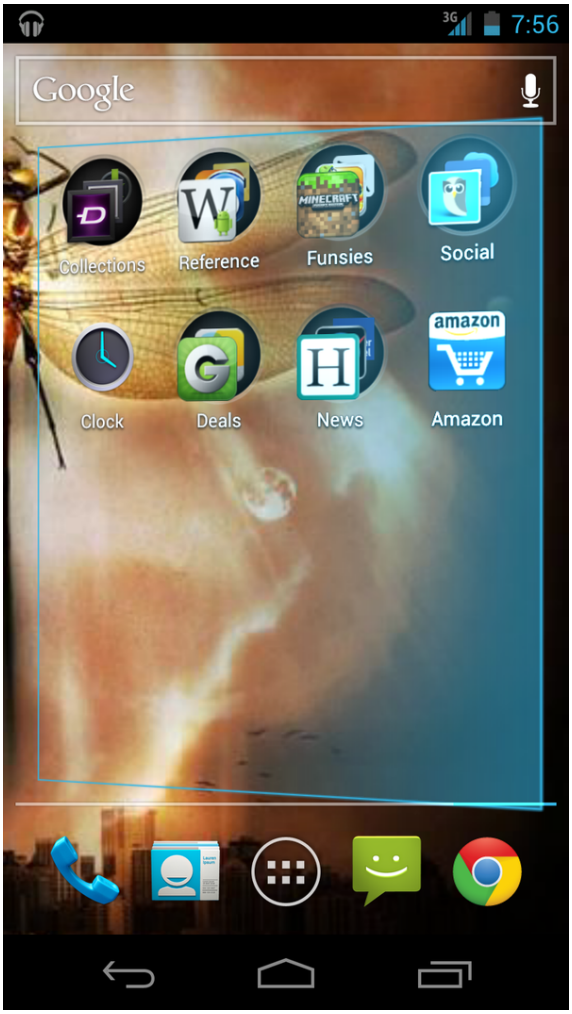

Figure 4-23. In Android (Ice Cream Sandwich versions), the screen skews if you try to scroll past where there are items. (Courtesy Tony Mooch and Little Big Details.)

- *Creating an illusion of virtual space*. How (and where) panels slide in and out, for example. *Transitions* can be an important part of microinteraction animations as users move from one state to another, or from one mode to another. Transitions help give a sense of location and navigation, letting users know where they are and where they are going to.
- *Encouraging deeper engagement*. Interesting animations invite interaction.

Scott has a valuable rule for animation timing: make any animation half as long as you think it should be. And then possibly halve the timing again (as detailed in *Designing Web Interfaces*, O'Reilly). Animation should make the microinteraction more efficient

(by illuminating the mental model or providing a means of directing attention) or at least *seem* more efficient, not less.

Messages

Designer Catriona Cornett tells of her experience updating the in-car Ford SYNC system. After putting the update on a USB drive to plug in to the car, she read these instructions:

> "Follow your printed out instructions exactly with your vehicle running. Approximately 60 seconds after you begin the installation, you will hear an 'Installation Complete' message. DO NOT REMOVE your USB drive or turn off your vehicle. You must wait an additional 4–18 minutes until you hear a second 'Installation Complete' message before you can remove your USB drive."
>
> OK, so, even though it will give me a message saying it's complete, it's really not, and if I didn't read this little note about the process, it makes it sound like I could cause some form of irreversible damage. Great.[10]

"Installation Complete" is clear enough as a message, in the above described case, unfortunately it's misleading. Any messages delivered as feedback to an action should—at a minimum—be accurate. As with instructional copy, any text as feedback should be short and clear. Avoid words like "error" and "warning" that provide no information and serve to only increase anxiety. Feedback text for any error messages should not only indicate what the error was, but also how to correct it. Ideally, it would even provide a mechanism for correcting the error alongside the message. For example, don't tell a user only that an entered password is wrong, provide the form field to re-enter it and/or a means of retrieving it.

While any text should be direct (and human), it's best to avoid using personal pronouns such as "you." "You entered the wrong password" is far more accusatory and off-putting than "Password incorrect." Likewise, avoid using "I," "me," or "my," as these are the uncanny valley of feedback copy. Although they can have human-like responses, microinteractions aren't human. Some voice interfaces like Siri can get away with using first-person pronouns, but in written form it can be jarring.

The ideal microinteraction text is measured in words, not lines, and certainly not paragraphs or even a single paragraph (Figure 4-25). Keep copy short and choose verbs carefully, focusing on actions that could or need to be taken: "Re-enter your password."

Audio

As noted in Chapter 2, sound can be a powerful cue that arrives quickly in our brains—more quickly than visual feedback. We're wired to respond to sound (and, as noted above, movement). Since it provides such a strong reaction, audio should be used sparingly. However, audio can be particularly useful on devices with no screens, or as

10. "UX principles in action: Feedback systems and Ford SYNC", July 11, 2011.

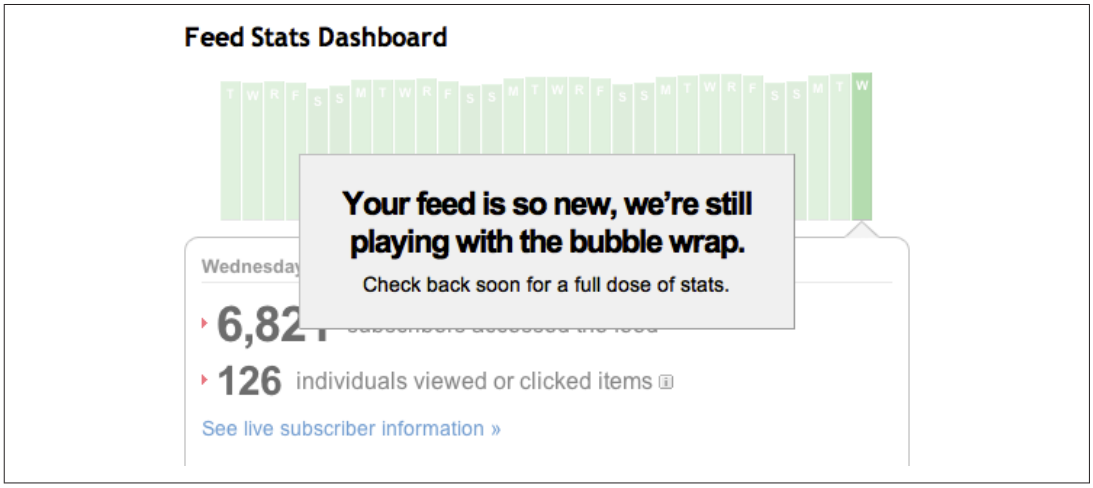

Figure 4-24. When you try to view your stats right away on Feedburner, you get this message. It would be better if it gave a more definite time to return.

Figure 4-25. Banking service Simple explains in one pithy line exactly what is going to happen in the future. (Courtesy Little Big Details.)

part of microinteractions that work in the background when the user isn't fully paying attention to them. It can also be useful in situations where looking at a screen can be unsafe, such as while driving.

In general, there are two ways to use audible feedback: for emphasis and for alerts. Audio for emphasis is typically for reinforcing a user-initiated action, as a confirmation that what the user thought happened actually did. Clicking a button and hearing a click is an example. These are often combined with visual feedback, and audio combined with visuals has been shown to be more effective than visuals alone.[11] The other kind of audio feedback—alerts—are typically indicators of system-initiated actions: a process has ended, a condition has changed, or something is wrong. A voice telling you to turn left in a navigation app is an example of an audio alert.

Any audio cue for a microinteraction should pass the *Foghorn Test*: is this action important enough that users would want to become aware of it when they cannot see it?

11. Brown, Newsome, and Glinert, "An experiment into the use of auditory cues to reduce visual workload," 1989.

Even if you think the answer is yes, you should possibly provide a mechanism to turn the sound off.

Like other feedback, audio can be adjusted if there is an understanding of the context of use. Some HTC phones buzz and ring loudly in their user's pockets or purses (the phone knows it is there via sensor data) and diminish in volume as the user removes them. Some automobiles increase the volume of music to compensate as the engine gets louder. Similarly, if the user isn't in the room with a device (detected via a proximity sensor) or the noise in the room is loud (detected by a microphone), volume and pitch could increase. And sound cues could also turn on (or increase in volume) if the device knows you are in a situation where visual cues are compromised, such as while driving (detected via GPS).

Sound designer Karen Kaushansky also cautions designers to consider the "non-use-case" when designing audio: when does audio not make sense? Broadcasting a sound—particularly voices—into an empty room in the middle of the night can be both startling and annoying.[12]

Earcons

There are two kinds of audio feedback: earcons and words. Earcons—a play on the word "icons" ("eye-cons")—are short, distinct sounds meant to convey information.[13] The amount of information that earcons can convey is limited, however, and sometimes words are necessary. Words are recorded (spoken) or computer-generated text. Words are particularly useful for instructions or directions, although if your product has to be in many languages, localization of the text could be nontrivial. Speech is also much slower than earcons; what can be conveyed in a fraction of a second with an earcon could take several seconds in speech—a ping versus "You've got mail!"

Earcons are, by their very nature, abstract, so care should be taken to select a sound that could indicate the message being conveyed. For microinteractions, the best earcons are those that users (consciously or unconsciously) can relate to other sounds they have heard and make associations. For example, the click of a latch closing can be the earcon for the microinteraction ending, or an upward whoosh can accompany an item moving to the top of a list. Avoid earcons that are too shrill (except for critical warnings) or too soft ("Did I just hear that?"). As with animation, the best earcons are brief: under one second in duration, and usually a fraction of a second. One exception is an ambient sound to indicate an ongoing process, such as a drone to indicate a file being synced.

12. See "Guidelines for Designing with Audio," *Smashing Magazine*.

13. Blattner, Meera M., Sumikawa, Denise A., and Greenberg, Robert M., "Earcons and Icons: Their Structure and Common Design Principles," Journal of Human-Computer Interaction, Volume 4, Issue 1, 1989.

Any earcon should also match the emotional content being conveyed. Is the feedback urgent or just utilitarian? A warning or an announcement? The qualities of the earcon (timbre, pitch, duration, volume) should match what is being communicated.

If you want your earcon to be iconic and memorable (a Signature Sound), it should contain two to four pitches (notes) played in succession.[14] As you don't necessarily want your microinteraction to be memorable, this trick should be used only once per microinteraction, if at all. Most microinteraction earcons should be a single-pitch sound, played once. Beware of playing any earcon in a loop, as even the softest, gentlest sound can be irritating played over and over and over.

Earcons should be unique to an action. Just as you want to avoid using the same visual feedback for different actions, you shouldn't use the same—or even similar-sounding—earcons for dissimilar events. This is especially true for alert sounds that could be triggered independent of user actions. If the user is looking away from (or isn't close to) the device, the user won't be sure of which action just happened.

Speech

If you're going to use words as audio feedback, keep the spoken message brief and clear. If there is a prompt for a response, make the choices clear, short, and few. Ideally with microinteractions, any voice responses would be "Yes" or "No," or at worst a single word. As noted in Chapter 3, microinteractions are the place for smart defaults, not multiple choice. Any word prompt should be at the end of the message. Use "To turn sound off, say yes" instead of "Say yes to turn sound off." Always end with the action.

With speech, your choice is to use actors to record the messages, or to use text-to-speech (TTS). Recorded messages have the advantage of feeling more human and can have more texture and nuance, although care has to be taken to make sure the actors convey the right message and tone via their inflections and pauses. The minus is that any time you change the message, it has to be rerecorded.

If the messages are dynamic (for example, turn-by-turn directions), TTS is probably your only option, as you would unlikely be able to record every street name. Although TTS has improved in recent years, it can still feel inhuman and impersonal, and some people actively dislike it, so use with care.

Haptics

Haptics, or as they are technically known, "vibrotactile feedback." Haptics are vibrations, usually generated by tiny motors, which can create a strong, tactile buzz or more delicate tremors that can simulate texture on flat surfaces. Compared to the decades of visual

14. See previous footnote, and Kerman, Joseph, *Listen*, Bedford/St. Martin's, 1980.

and audio feedback, haptics is relatively new, with the majority of people only having experienced it with the advent of pagers and mobile phones.

Haptics, since they are mostly felt (although the vibration can make noise against a hard surface like a tabletop), are best utilized on devices the user will be in close proximity to (by holding, touching, wearing, or carrying), although they can also be embedded in objects like furniture to enhance entertainment like movies and games. Faces and hands (particularly fingertips) are the most sensitive to haptics, while legs and torso are much less so.

Even more than vision and hearing, our sense of touch (technically our cutaneous sense) is limited. Not by our skin, which is extensive (although of varying sensitivity to touch), but by our brains. There are four kinds of fibers known as mechanoreceptors that convey cutaneous sense, each of which can detect different frequencies. The different mechanoreceptors engage in crosstalk with each other, the result of which determines what we can feel—which, as it turns out, isn't much. One researcher claims the amount of information we can get from touch is 1% that of hearing.[15] Most people can only easily detect three or four levels of vibration.[16] Thus, complex messages are not readily conveyed with haptics.

Luckily, complex messages are usually unnecessary with microinteractions. Haptics have three main uses for microinteraction feedback. The first is to enhance a physical action, such as by simulating the press of a button on a touchscreen, or by giving an added jolt when the ringer of your phone is turned off. The second (and currently most common) use of haptics is as an alert when audio isn't available or is undesirable. The vehicle-initiated vibration of the steering wheel to wake a sleepy driver is an example of this use. The third (and thus far rarest) use is to create an artificial texture or friction on surfaces such as touchscreens. This can be used to slow down scrolling, for instance.

Because of humans' limited ability to detect differences in haptics, they are currently best used in microinteractions for either subtle, ambient communication or for a disruptive alert. There's very little middle ground, except perhaps in specialty devices, like those for musicians and surgeons, where varying levels of haptics can provide more physical feedback while doing an action like making music or performing surgery.

15. R. T. Verrillo, A. J. Fraioli, and R. L. Smith, "Sensation magnitude of vibrotactile stimuli," *Perception & Psychophysics*, vol. 6, pp. 366–372, 1969.

16. Gill, John, "Guidelines for the design of accessible information and communication technology systems". Royal Institute of the Blind, 2004, and F. A. Geldard and C. E. Sherrick, "Princeton cutaneous research project, report no. 38," Princeton University, Princeton, NJ, 1974.

Feedback Rules

Feedback can also have its own set of rules that dictate its instantiation. Feedback rules define:

Contextual Changes
: Does the feedback change based on the known context? For instance, if it is night, does the volume increase? Decrease?

Duration
: How long does the feedback last? What dismisses it?

Intensity
: How bright/fast/loud/vibrating is the effect? Is it ambient or noticeable? Does the intensity grow in time, or remain constant?

Repetition
: Does the feedback repeat? How often? Does the effect remain forever, or just for a few seconds?

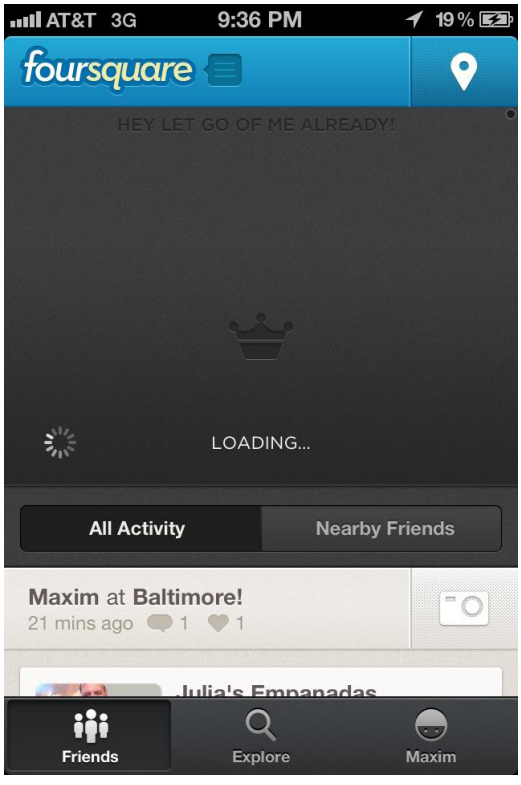

Figure 4-26. Foursquare makes a plea for help when you pull down too far to refresh. (Courtesy Tory Briggs.)

These rules can determine much of the character of the feedback.

If you don't want your users feeling cheated and putting their fists through the screen as they do with slot machines, look to your feedback. Make the rules understandable, and inform them of changes in state when appropriate. Make the feedback consistent, rewarding positive behavior.

Sometimes it's not just a piece of feedback that repeats, it's the whole microinteraction. In Chapter 5, next, we'll discuss how to use loops and modes to extend your microinteraction.

Summary

Understand what information the user needs to know and when. All feedback relies on this understanding.

Feedback is for understanding the rules of the microinteraction. Figure out which rules deserve feedback.

Determine what message you want to convey with feedback, then select the correct channel(s) for that message.

Look at context and see if the feedback can (or should) be altered by it.

Be human. Feedback can use a veneer of humanity to provide personality to the microinteraction.

Use preexisting UI elements to convey feedback messages. Add to what is already there if you can before adding another element.

Don't make feedback arbitrary. Link the feedback to the control and/or the resulting behavior.

Whenever possible, have visual feedback for every user-initiated action. Add sound and haptics for emphasis and alerts.

CHAPTER 5

Loops and Modes

On January 4, 2004, a 400-pound, six-wheeled, solar-powered robot landed on Mars, in the massive impact crater Gusev. The robot was the $400-million-dollar *Spirit* rover that had taken over a year to build. As Passport to Knowledge reported, *Spirit* had just survived the six-month journey to the Red Planet and a perilous landing, including bouncing as high as a four-story building on first impact with the surface. The Jet Propulsion Laboratory (JPL) team that built and commanded the rover thought the worst was behind them. They were wrong.

Once the (literal) dust—red—had settled, *Spirit* began its mission of taking pictures and performing scientific experiments, rolling toward a nearby destination ("Sleepy Hollow"). But then on January 21, less than three weeks into the mission, something happened. NASA's Deep Space Network lost contact with *Spirit*.

At first, the rover's disappearance was blamed on a thunderstorm in Australia disrupting the network, but no, there was something wrong with *Spirit* itself. The next day, a