Designing with Details

Microinteractions

UNCORRECTED PROOF

O'REILLY®

Dan Saffer

Table of Contents

Preface

Over the last decade, designers have been encouraged to think big, to solve “wicked problems,” to use “design thinking” to tackle massive, systemic issues in business and in government. No problem is too large to not apply the tools of design to, and design engagements can involve everything from organizational restructuring to urban planning.

The results of this refocusing of design efforts are unclear. But by working at such a macro scale, an important part of design is often lost: the details that delight. Products that we love show an attention to detail: the beautiful curve, the satisfying click, the understandable mental model.

This is another way to work: not through grand, top-down design projects, but from the bottom up, by crafting—lovingly, with care—small things. This is something designers can do quite well, with immediate, tangible results. This is another way to change the world: by making seemingly inconsequential moments into instances of pleasure.

There is a joy in tiny things that are beautiful and work well. This joy is both on the part of the user and in the creator, even though it certainly takes skill, time, and thought to make it so. It’s hard work, and as admirable in its own way as tackling the Big Problems. After all, who doesn’t need more joy in their life?

San Francisco

October 2012

1

Designing Microinteractions

The furious shouting started after the conductor stopped the performance. The New York Philharmonic had reached the very end of the slow, quiet Adagio movement that finishes Mahler's Symphony No. 9. The audience, many of whom had paid hundreds of dollars for this privilege, sat attentive and rapt, listening to the still, sublime moments that resolve over an hour of music. And then it happened:

From the front row, the unmistakable sound of an iPhone's "Marimba" sound—that high-pitched xylophone tinkle—going off over and over again. An alarm. It kept going. And going. The conductor, Alan Gilbert, curtly halted the orchestra. But the alarm kept going off. By now, audience members were yelling at the phone's owner, an older executive later dubbed "Patron X" by the Philharmonic, who didn't want to further embarrass the apologetic man, a long-time symphony patron. Avery Fisher Hall, which just moments before had been unearthly calm and quiet, was awash in chaos and anger.

As the New York Times reported in January 2012[1], Patron X had just gotten the iPhone the day before; his company had replaced his Blackberry for it. Before the performance began, he had flipped the mute switch, turning silent mode on. But what he didn't know was that one of the iPhone's rules for mute was that alarms still go off even when the phone is muted. So when the alarm went off, he didn't even realize it was his phone for

[1] Daniel J. Wakin, "Ringing Finally Ended, but There's No Button to Stop Shame," *The New York Times*, January 12, 2012

an excruciatingly long time. By the time he knew it was his phone and had turned the alarm off, it was too late: the performance was ruined.

The next day, as news spread, the Internet exploded with vitriol and wisecracks. Composer Daniel Dorff tweeted, "Changed my ringtone to play #Mahler 9 just in case." Arguments and discussions spanned across blogs, with some advocating that mute should turn every sound off. Tech columnist Andy Ihnatko wrote, "My philosophy is 'It's much better to be upset with yourself for having done something stupid than to be upset with a device that made the wrong decision on its own initiative.' "[2] While others made the (in my opinion, correct) case that alarms still need to sound even when the phone is muted. As Apple pundit John Gruber pointed out, "if the mute switch silenced everything, there'd be thousands of people oversleeping every single day because they went to bed the night before unaware that the phone was still in silent mode."[3] Apple's own iOS Human Interface Guidelines gives its rationale for why mute works the way it does:

> For example, in a theater users switch their devices to silent to avoid bothering other people in the theater. In this situation, users still want to be able to use apps on their devices, but they don't want to be surprised by sounds they don't expect or explicitly request, such as ringtones or new message sounds.
>
> The Ring/Silent (or Silent) switch does not silence sounds that result from user actions that are solely and explicitly intended to produce sound.[4]

In other words, mute does not silence the sounds that users have specifically asked for, only those they have not (e.g. text messages, incoming phone calls, etc.) This is the rule. Like many rules, it's hidden, and it's compounded by the fact that other than the tiny orange mark on the switch, there is no onscreen indicator that mute is on. If Apple was to change to a different rule—that mute silences everything—other rules and feedback would have to be designed. Would the phone vibrate when an alarm went off? Would there be some persistent indicator that the phone was in Silent mode, either onscreen when you woke up the phone or a small LED indicator in the hardware? There are many different ways mute could be designed.

Mute is an example of a microinteraction. Microinteractions are contained product moments that revolve around a single use case—they have one main task. Every time you change a setting, sync your data or devices, set an alarm, pick a password, log in, set a

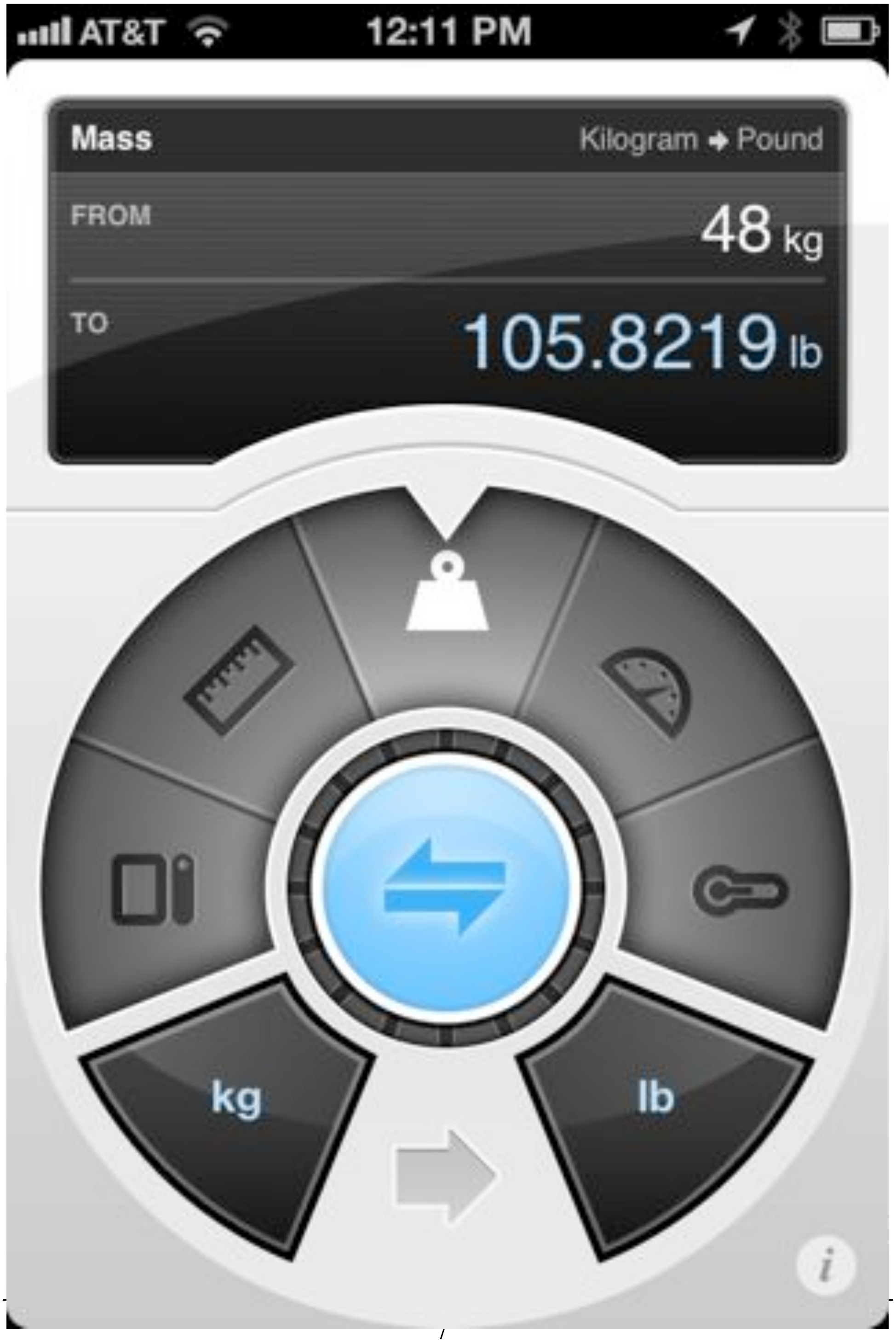
[2] "Daring Fireball: On the Behavior of the iPhone Mute Switch," January 14, 2012 http://ihnatko.com/2012/01/14/daring-fireball-on-the-behavior-of-the-iphone-mute-switch/

[3] "On the Behavior of the iPhone Mute Switch," January 13, 2012, http://daringfireball.net/2012/01/iphone_mute_switch_design

[4] Apple iOS Human Interface Guidelines, "Sound"

status message, or favorite or "like" something, you are engaging with a microinteraction. They are everywhere: in the devices we carry, the appliances in our house, the apps on our phones and desktops, even embedded in the environments we live and work in. Most appliances and some apps are built entirely around one microinteraction (see Chapter 7).

UNCORRECTED PROOF



Mass Kilogram → Pound

FROM 48 kg

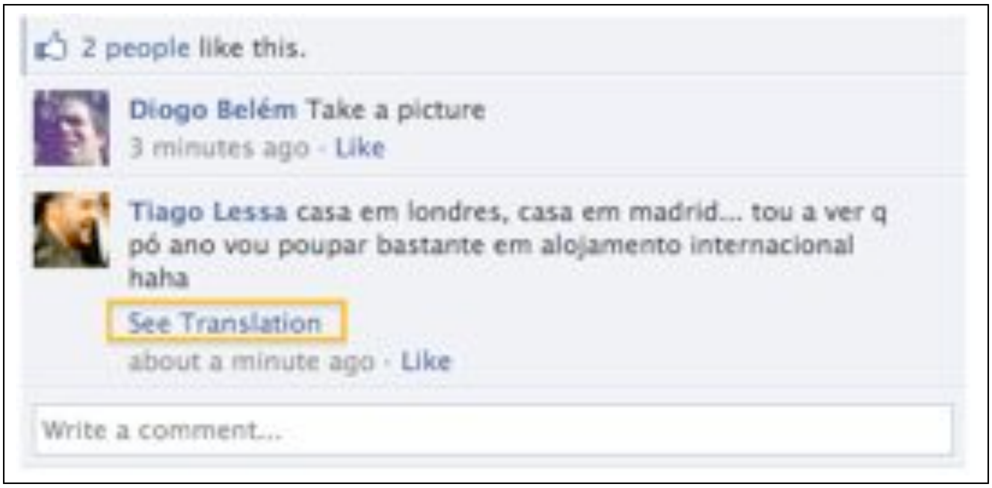
TO 105.8219 lb

Figure 1.1 Tapbot's Convertbot is an app built around a microinteraction: converting one value to another.

Microinteractions are good for:

- accomplishing a single task
- connecting devices together
- interacting with a single piece of data such as a stock price or the temperature
- controlling an ongoing process such as music volume
- adjusting a setting
- viewing or creating a small piece of content like a status message
- turning a feature or function on or off

The case of Patron X is one of the few examples of a microinteraction making news. Even though we're surrounded by microinteractions every day, we don't usually notice them until something goes horribly wrong, as it did for Patron X. But microinteractions are, despite their small size and near-invisibility, incredibly important. The difference between a product you love and a product you tolerate is often the microinteractions you have with it. They can make our lives easier, more fun, and just more interesting if done well. That's what this book is all about: how to design microinteractions well.


2 people like this.

Diogo Belém Take a picture
3 minutes ago · Like

Tiago Lessa casa em londres, casa em madrid... tou a ver q pó ano vou poupar bastante em alojamento internacional haha
See Translation
about a minute ago · Like

Write a comment...

Figure 1-1. When someone posts on your Facebook page in a language that isn't your default, Facebook offers to translate.

Microinteractions are the details of a product, and details, as Charles Eames famously said[5], aren't just the details; they are the design. Details can make make engaging with the product easier, more pleasurable—even if we don't consciously remember them. Some microinteractions are practically or literally invisible, and few are the feature that you buy a product for (although many apps and devices are created around a single microinteraction; see Chapter 7); instead, they are usually pieces of features, or the supporting or so-called "hygiene" features. For example, no one buys a mobile phone for the mute feature, but as we've seen, mute can create all sorts of experiences—for good and bad.

[5] See *100 Quotes by Charles Eames*, Eames Office, 2007

UNCORRECTED PROOF

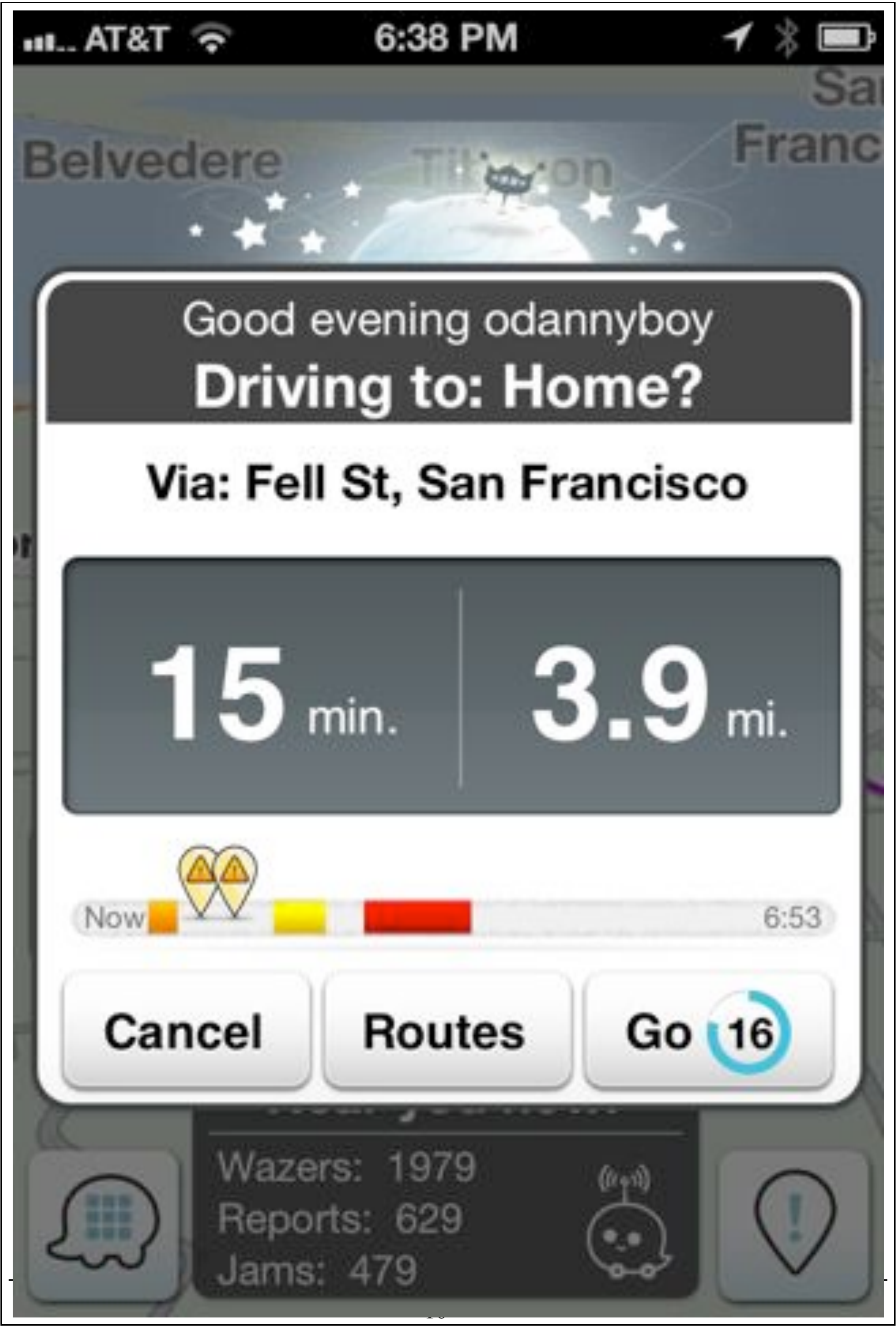


Figure 1-2. Navigation app Waze knows when I open the app in the late afternoon, I'm probably driving home and presents this as an option.

Think about it: Almost all operating systems, be they mobile or desktop, do basically the same things: install and launch applications, manage files, connect software to hardware, manage open applications/windows, etc. But the difference between operating systems—at least from a user's perspective—are the microinteractions you have with it on a daily, even hourly, basis.

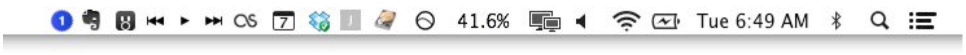


Figure 1.2 The author's Menu Bar in OS X is crammed full of icons, each of which is a Trigger for a microinteraction.

Features ("macrointeractions") are differentiated from microinteractions by their complexity and the focus it takes to execute them. Macrointeractions are longer in duration and have more use cases, and thus have more of a cognitive load on the user. In comparison, microinteractions are shallow, thin, and rapid, sometimes executed so quickly the user barely barely is aware of it as a distinct entity at all. Macrointeractions can be made up of many microinteractions put together.

UNCORRECTED PROOF

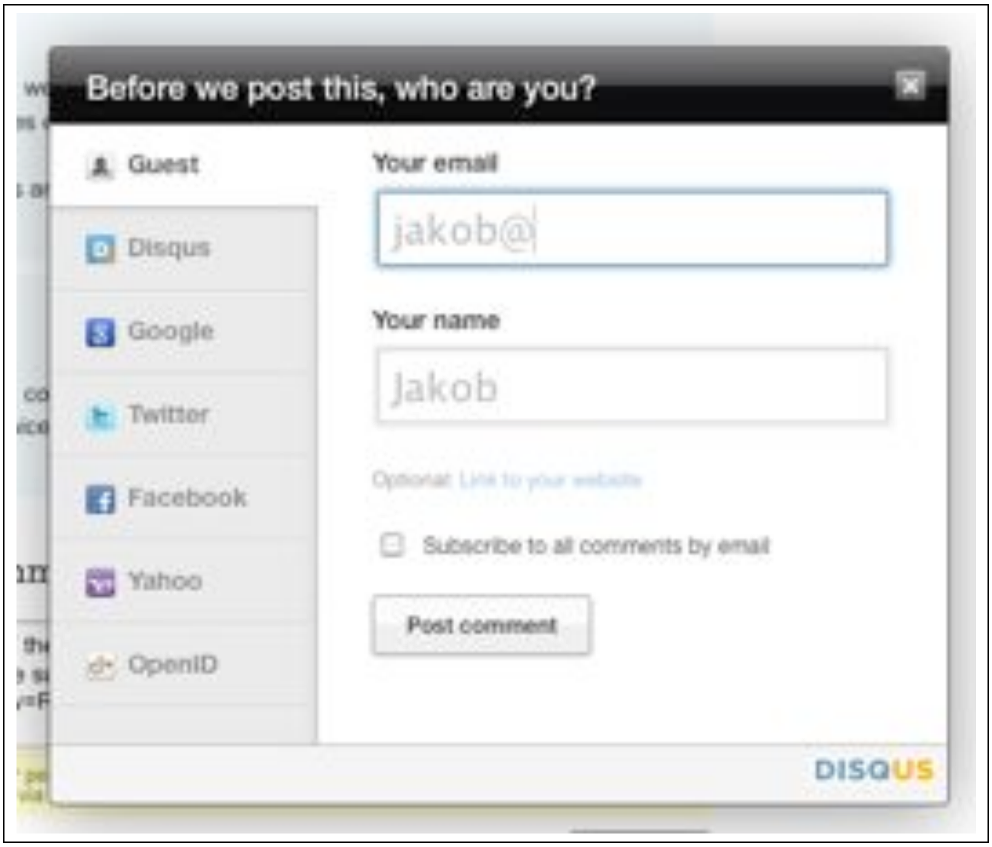

Before we post this, who are you?
Guest
Disqus
Google
Twitter
Facebook
Yahoo
OpenID
Your email
jakob@
Your name
Jakob
Optional: Link to your website
Subscribe to all comments by email
Post comment
DISQUS

Figure 1-3. The Disqus sign-up form guesses your name based on your email address.

Microinteractions are frequently the last parts of a product to be designed and developed, and as such they are often overlooked. But doing so is a mistake. The reason the original (G1) version of Android felt so unpolished was because the microinteractions were clunky, especially in comparison to the iPhone; for example, deleting items was inconsistently triggered, and in some applications pressing the search key did nothing at all. If the microinteractions are poor, the main features, no matter how nicely done, are surrounded by pain and frustration.

Of course, some features are so useful and/or powerful (or so highly-protected by IP) that the microinteractions don't matter as much. Many medical devices are examples of this, as is most early-stage technology, when it is more amazing something can be done instead of how it’s done. For instance, the first generation of the Roomba (introduced in 2002) couldn’t calculate room size or detect obstacles and dirt, but it was a novel technology nonetheless, and subsequent models (especially now that there are competitors on the market) have focused more on the human-robot microinteractions.

The combination of well-designed micro- and macro- (feature) interactions is a powerful one. This is what Experience Design truly is: paying attention to the details as well as the big picture so that users have a great experience using the product.

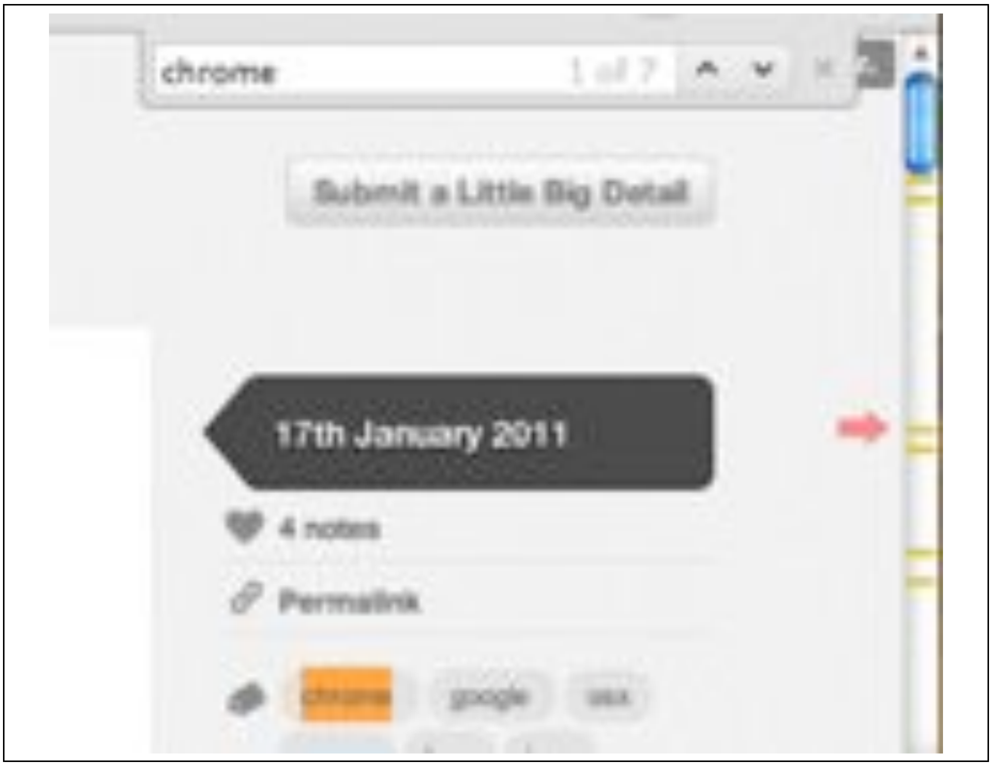


Figure 1-4. When trying to find a word on a page, Chrome indicates in the scrollbar where instances of that word appear.

In competitive markets, microinteractions are even more important. When there is feature parity, it is the experience using the product that increases adoption and brand loyalty. The overall experience of a product relies heavily on its microinteractions. They are the "feel" in look-and-feel. One reason Google Plus fell so flat against Facebook was that its microinteractions, such as sorting users into circles, became tiresome and gimmicky.

UNCORRECTED PROOF

Another reason to pay attention to microinteractions is because they fit so well into our multi-platform existence. Microinteractions are the glue that can tie together features across mobile, TV, desktop, appliances, and web. While the microinteractions could vary by platform, their small size allows for a consistency that you might not have with large features. In particular, appliances and mobile devices with their small (or no) screens seem custom-made for microinteractions. Small interactions work well on small devices.

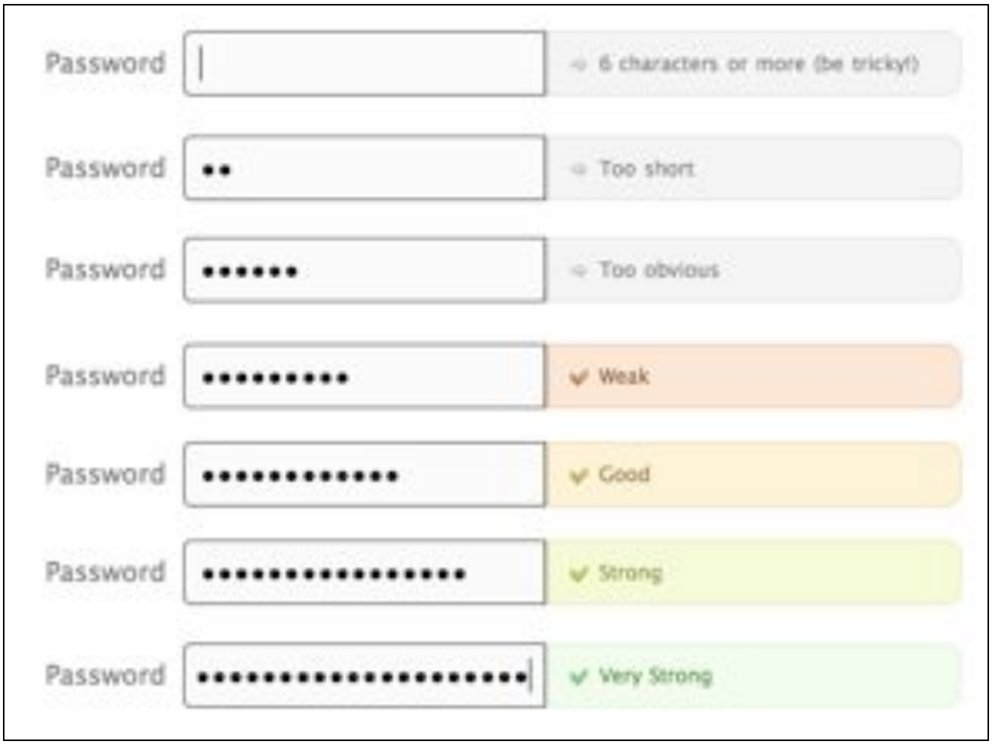
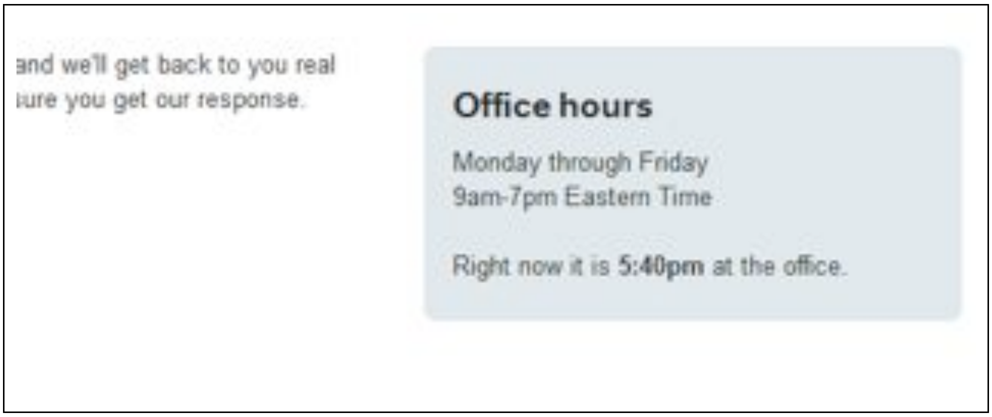


Figure 1-5. Twitter's password form is a great microinteraction, with very clear feedback.

Take Twitter for example. Twitter is built entirely around a single microinteraction: sending a <140-character message. Users can do this from practically any device, anywhere. Some objects even tweet independently, or for us. It can be used to send gossip or messages to coordinate a revolution. Well-designed microinteractions can scale well across platforms and to millions of users.



and we'll get back to you real
sure you get our response.

Office hours

Monday through Friday
9am-7pm Eastern Time

Right now it is **5:40pm** at the office.

Figure 1-6. When you go to ask for support at Harvest, it shows the time at their office alongside their office hours.Microinteractions also fit well into our already-crowded, overcomplicated, and fragmented lives. We need and even enjoy the fast glance at data, the rapid check-in at a restaurant, the casual review of messages on the subway. (The "Casual Games" category is really a set of stand-alone microinteractions for amusement.)

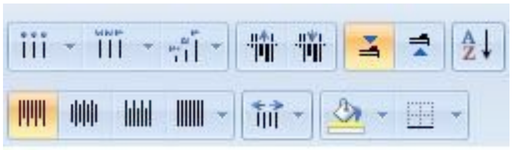


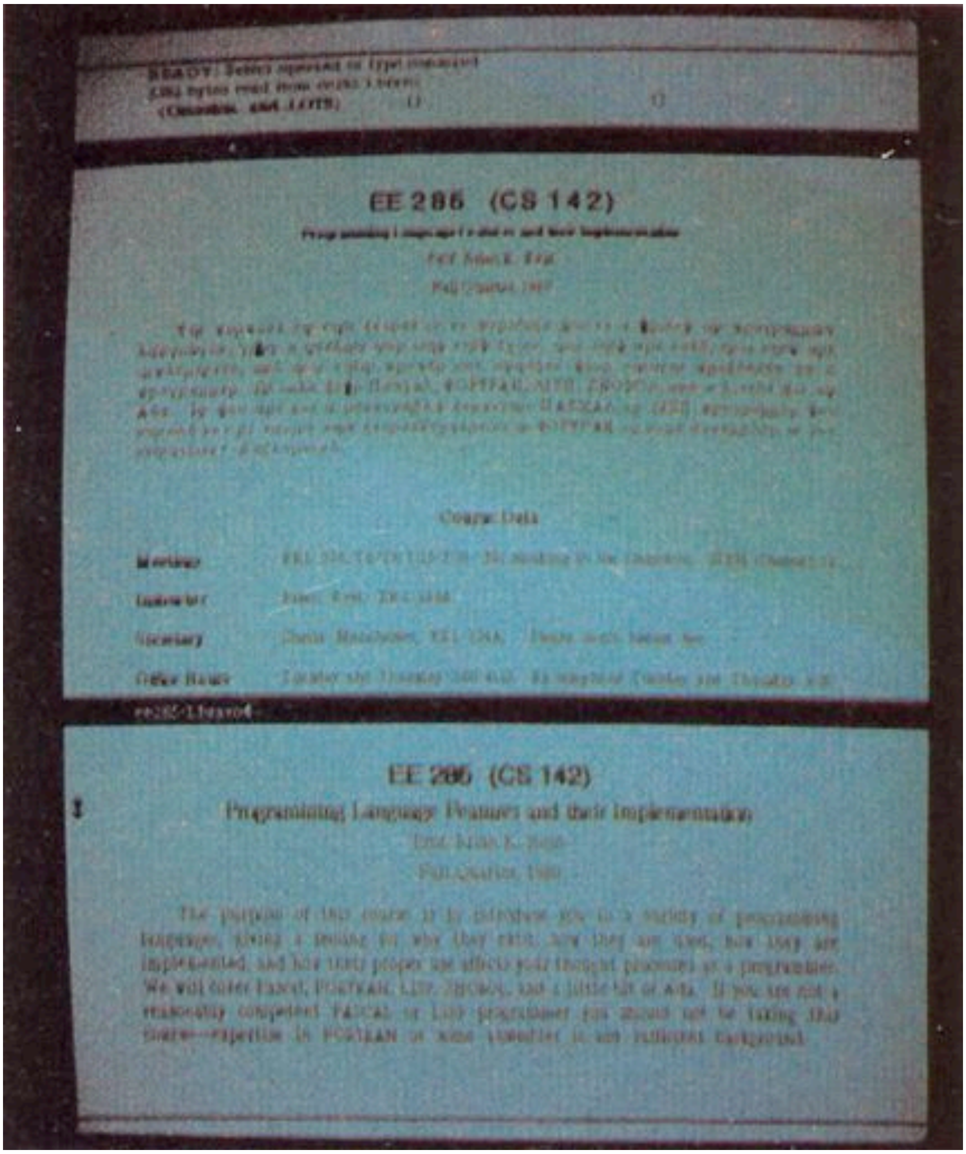
Figure 1-7. In Microsoft Office, when text is rotated, relevant styling buttons are rotated as well.

Microinteractions force designers to work simply, to focus on details. They challenge designers to see how lightweight they can design, to reduce complexity and streamline features that could otherwise be burdensome.

The Secret History of Microinteractions

In 1974, a young engineer named Larry Tesler began working on an application called Gypsy for the Xerox Alto computer. Gypsy was one of the first word-processing applications ever, and the successor to the groundbreaking Bravo, the first true WYSIWYG word-processing program and the first program to have the ability to change fonts. Even though it was still a word-processing program, Gypsy was a different kind of application altogether: it made use of a mouse and a graphical user interface (GUI). Larry's mission (and what would become his rallying cry for decades to come—his website is nomodes.com, his Twitter handle is @nomodes, and even his license plate reads NOMODES[6]) was to reduce the modality of the interface, so that users wouldn't have to switch to a separate mode to perform actions. Larry wanted users, when they typed a character key, to always have that character appear onscreen as text—not an unreasonable expectation for a word-processing application. This wasn't the case in Bravo: typing only worked in a particular mode; other times it triggered a function.

[6] http://nomodes.com/Larry_Tesler_Consulting/CV_files/NOMODES.jpg



[illegible]

EE 285 (CS 142)

[illegible]

Course Data

[illegible]

EE 285 (CS 142)

Programming Language Features and their Implementation

[illegible]

Figure 1.3 A "screenshot" (Polaroid(!)) of Bravo. The bottom window is being used to make a form in the top window.

One of those functions was moving text from one part of the document to another. In Bravo, users had to first select the destination, then press the "I" or "R" keys to enter

Insert or Replace modes, then find and select the text to move, then finally press the Escape key to execute the copy[7]. Larry knew there was a better way to perform this action, so he designed one that not only made use of the mouse, but radically simplified this microinteraction. In Gypsy, the user could select a piece of text, press the "Copy" function key, then select the destination, and finally press the "Paste" function key. No mode required. And thus, Cut and Paste was born.

The intertwined history of interaction design and human-computer interaction is really the history of microinteractions. The tiny things we unthinkingly interact with every day on desktops, laptops, and mobile devices were once novel microinteractions: everything from saving a document to organizing files into folders to connecting to a wifi network were all microinteractions that needed to be designed. Even "basics" like scrolling and opening multiple windows needed to be designed and engineered. The forward march of technology has provided a continuous need for new microinteractions. We use them unquestioningly now, and only really pay attention to them when someone designs a better way, or the technology changes and allows for or forces a new way of performing the microinteraction.

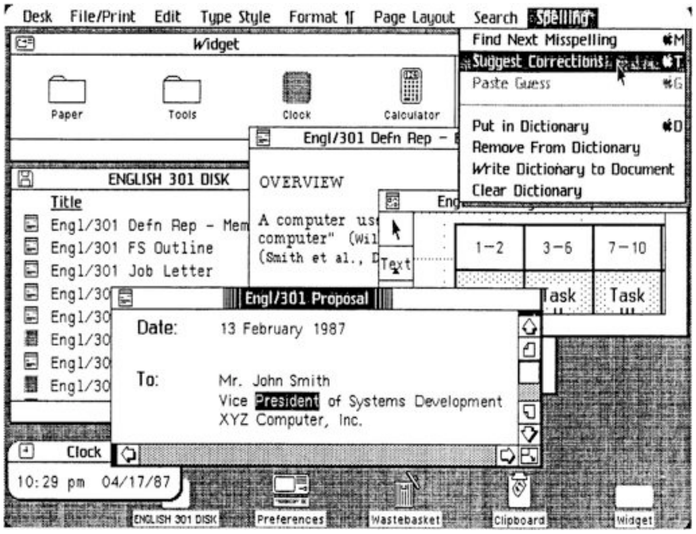
Indeed, as technologies have changed, the microinteractions that support them have also changed. Take scrolling, for instance. Bravo had a primitive version of scrolling, but scrolling really became more refined when Alan Kay, Adele Goldberg, and Dan Ingalls introduced scrollbars in another Xerox PARC product, SmallTalk, sometime between 1973-1976. SmallTalk's scrolling could be smooth, pixel-by-pixel, instead of line-by-line. (This was famously one of the UI elements demoed to Steve Jobs and his engineers, which they then built into Apple's Lisa—and subsequently the Macintosh[8].) As documents got longer, scrollbars added arrows to jump to the end without scrolling. Tooltip-style indicators would appear to indicate where you were in the document. But the real change came with touchscreen technology on trackpads and mobile devices. Do you slide up or down to scroll down? Apple famously changed directions (from down to up) in OS X Lion after the introduction of its iPhones in order to align its laptops and mobile devices to "natural scrolling.[9]" Apple has also (to the ire of many) hidden scrollbars except when scrolling is in process or the cursor nears the right edge of a scrollable window. The microinteraction keeps evolving.

---

[7] Detailed in *Bravo Course Outline* by Suzan Jerome, published by Xerox, 1976.

[8] As recounted in *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age* by Michael A. Hiltzik

[9] See for example, "Apple's Mousetrap: Why did Apple reverse the way we scroll up and down?" by Michael Agger in Slate, http://www.slate.com/articles/technology/technology/2011/09/apples_mousetrap.html



Date: 13 February 1987

To: Mr. John Smith
Vice President of Systems Development
XYZ Computer, Inc.

Figure 1-8. Apple's Lisa (1982) featured dozens of "new" (for the market) microinteractions.

But it's not just digital products that have had microinteractions; a case can be made for microinteractions to have originated with the first electric devices, such as the radio (1893), the flashlight (1986), and the washing machine (1900). As designer Bill DeRouchey points out in his talk The History of The Button, in the (pre-electric) mechanical era, users could follow their actions directly from the control to the output. You could pull a lever, watch the gears move, then finally the wheels turn. It was easy to connect the input to the output. Electricity changed all that. You could press a button on the wall and nearly instantly a light on the other side of the room turned on. Sure, the feedback was instant, but the method of execution was not. As DeRouchey says, "The button meant for the first time the result of the human motion could be completely different from the motion [it creates] itself." Action became abstracted[10].

In the digital age, particularly before the GUI, action became even more abstract. Inserting a stack of punchcards or flipping a series of switches produced output that was equally obtuse. For a time, the GUI cleared up and simplified microinteractions. But then Moore's Law (processor speed doubles every 18 months), Koomey's Law (power consumption for hardware decreases 50% every 18 months), Kryder's Law (exponential increase in storage space), and increasing bandwidth and network connectivity (LANs

[10] Bill DeRouchey, The History of The Button, http://www.slideshare.net/billder/history-of-the-button-at-sxsw

first, then wireless networks, both local and mobile) created the need for more microinteractions, and the microinteractions needed to control actions far more abstract than turning on a light. Just as one example, syncing data across devices is a conceptually-abstract idea, for which there's no readily available physical analog.

Input methods are also drastically changing microinteractions. Not only do we have physical controls like buttons, switches, keyboards, and mice, we also have touchscreens, sensors, voice, and gestural means of triggering microinteractions. In the not-too-distant past, the only way to interact with the physical environment was to adjust it manually via a physical control. This changed in 1956 when Robert Adler invented the Zenith Space Command, the first TV remote control. For the first time, users could control an object from a distance, invisibly.

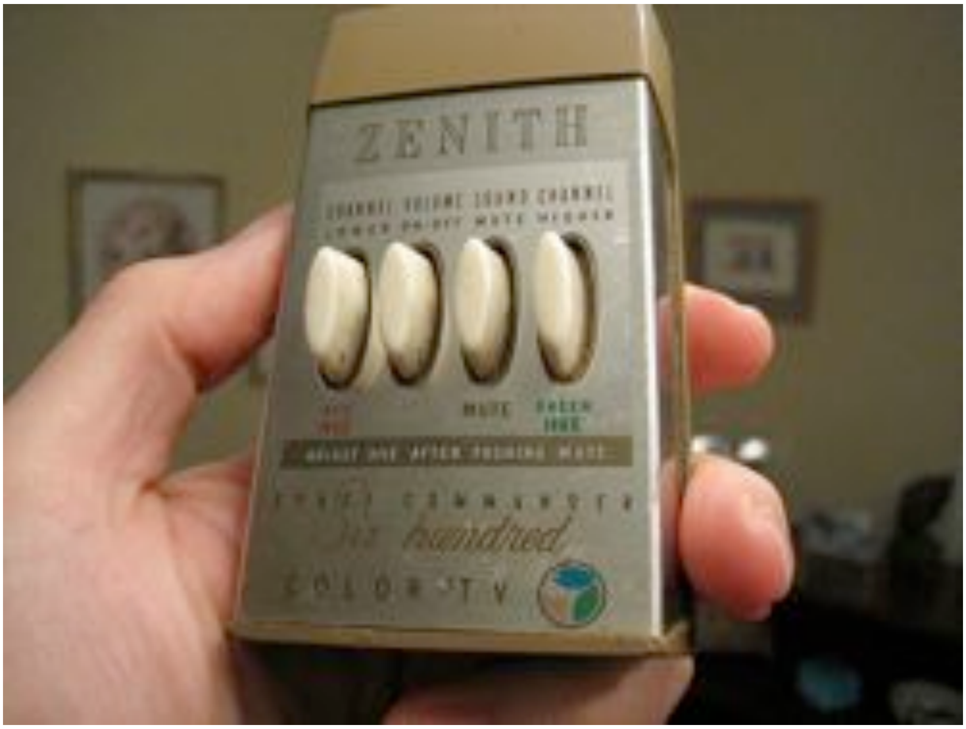


Figure 1.4 Although there had been remote-control planes and boats previously (mostly for military use), the Space Commander television remote removed proximity from control for consumers.

Today, to trigger a microinteraction, you don't even need to be in the same room. With the right equipment, you can adjust the temperature in your house from the other side of the world. Or you only need to be in the right location; just by being in a certain block, your mobile phone can remind you of a To Do item, or your GPS device can tell you where to turn left. In public restrooms, you can turn on the sinks just by putting your hands into them. You can also tell your phone to find you a nearby restaurant, or flick your finger down a touchscreen list to reveal a search bar or tap your phone on a counter to pay for your coffee.

Figure 1.5 The Nest Learning Thermostat uses proximity sensors to know when someone walks into the room, then lights up and shows the temperature in a way that's glanceable from across the room. No touching required.

The history of technology is also the secret history of the microinteractions that, like symbiotic organisms, live alongside them to frame, manage, and control them.

The Structure of Microinteractions

What makes effective microinteractions is not only their contained size, but also their form. A beautifully-crafted microinteraction pays attention to all four parts of a microinteraction.

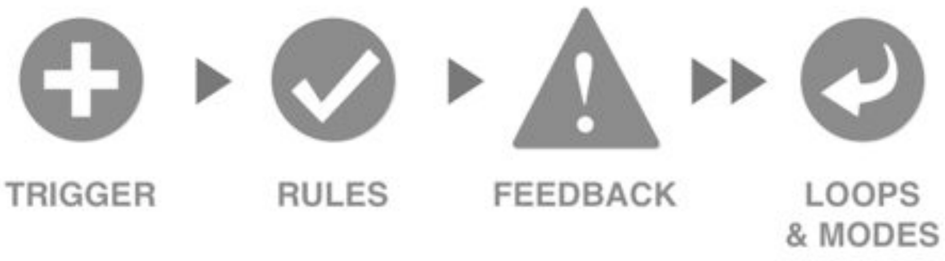


Figure 1.6 The structure of microinteractions.

The first part of any microinteraction is the **Trigger**. With mute, the trigger is user-initiated, meaning that the user has to do something—in this case flip a switch—to begin the microinteraction. Thus, many microinteractions begin with a understanding of user need: what the user wants to accomplish, when they want to do that, and how often they want to do it. This determines the affordances, accessibility, and persistence of the trigger. In our mute example, mute is a very common action that users want to perform all the time, rapidly. Thus the trigger (the mute switch) is available all the time, instantly able to be turned on and off no matter what application is running. It was so important, it's one of only five physical controls on the iPhone. Controls—digital and/or physical—are the most important part of user-initiated triggers. They provide not only the ability to engage with a microinteraction (and sometimes the ability to adjust it while in progress), but also usually the visual affordance that the microinteraction is even there. If there was no mute switch on the iPhone, you might expect the phone had that feature, but have to guess at where to find it. In many older mobile phones (and even in some phones still), the mute feature was buried under several layers of a settings menu. Even for users who knew where the feature was, it took as much as 10 seconds to turn mute on or off. It takes less than a second to flip the physical mute switch.

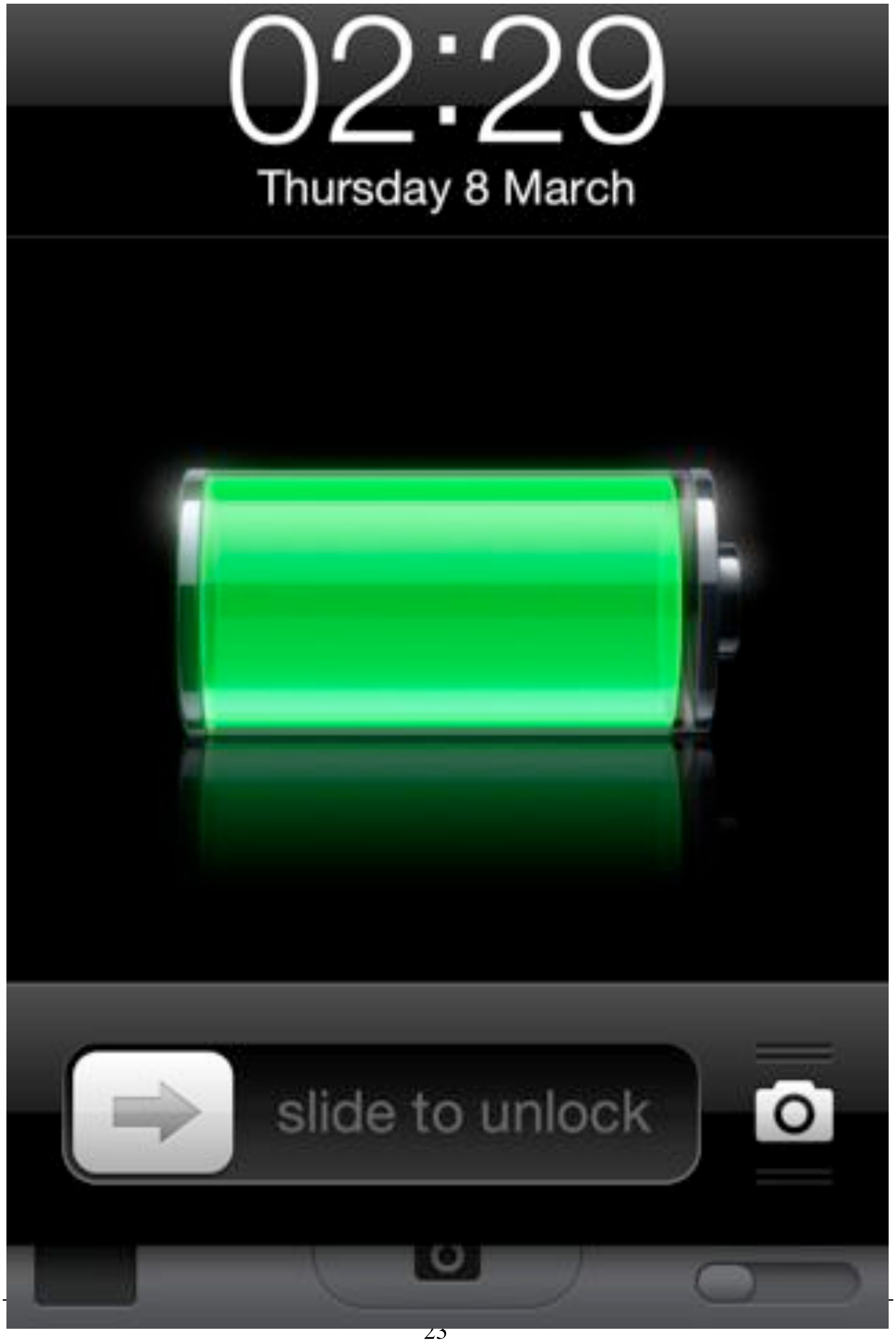

02:29
Thursday 8 March
slide to unlock

Figure 1.7 An example of a Trigger. In iOS (as in Windows Mobile), you can use the camera even on a locked phone. Pressing the camera icon bounces the bottom bar up a little, indicating that you swipe up to get the camera functionality. Of course, Slide to Unlock is its own Trigger as well.

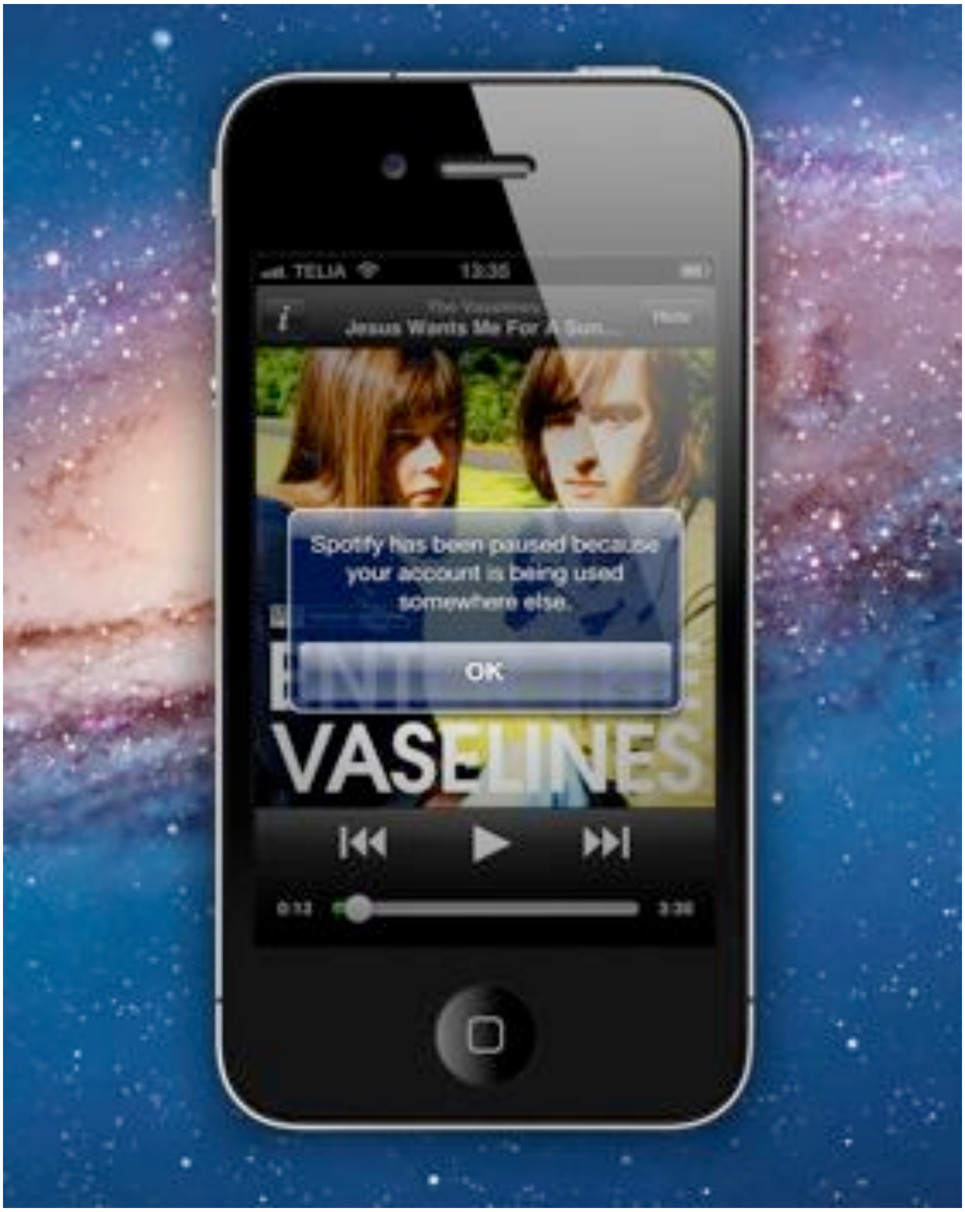
Of course, the physical control doesn't have to be a switch either. Although the best designs feel inevitable, there is almost nothing designed that could not be designed differently. On Windows Phones, the trigger is a pressable rocker button (which also controls volume) that, when pressed, presents users with a screen overlay that lets users choose ringer status as “vibrate” or “ring + vibrate."

But triggers need not be user-initiated. Increasingly, triggers are system-initiated—when the device or application detects that certain conditions have been met and itself begins a microinteraction. The triggering condition could be anything from detecting that a new email has arrived, to the time of day, to the price of a particular stock, to the location the user is in the world. For mute, one could easily imagine mute integrating with your calendar, so that it automatically mutes whenever you're in a meeting. Or by knowing your location, it automatically mutes whenever you're in a movie theater or symphony hall. As our applications and devices become more sensor-full and context-aware, the more ability they have to make decisions on their own about how they operate.

Triggers are covered in chapter 2.

Understandably, users may want—if not the ability to adjust these system-initiated triggers—then at least the understanding of how they operate, just as Patron X probably would have liked to have known how mute works. In other words, they want to know the **Rules** of the microinteraction.

Once a microinteraction has been initiated, it engages a set of rules and behavior of the microinteraction. In other words: something happens. This usually means turning a feature or a set of features on, but it might show the current state of the application or device. It might use data to guess what the user wants to do. In whatever case, it turns on (at least one) rule, and those rules can usually be defined by a designer.



Spotify has been paused because your account is being used somewhere else.

Figure 1.8 An example of a Rule. When you're using the music-streaming service Spotify and then turn it on on another platform, the first instance of Spotify pauses. If you resume playing on the first instance, the second platform will pause. This creates a very frictionless, cross-platform service.

Take what is probably the simplest microinteraction there is: turning on a light. Once you use the trigger (a light switch), the light turns on. In a basic light set up, there is a single rule: the light stays on and fully lit until the switch is turned off. You can change that rule, however, by adding a dimmer or a motion-detector that turns the light off when no motion is detected. But the basic turn on switch/turn on light rule is very simple rule, and one that becomes apparent to anyone who uses a light, even a child.

With applications or electro-digital devices, the interactions can be much, much more nuanced and hard to understand, even for small microinteractions. In the case of Patron X, it was the interaction with mute that caused the symphony incident, because unless there is a specific piece of feedback (and we'll get to that next), interactions are themselves invisible. Unlike the mechanical devices of the 19th century, users generally cannot see the activity the trigger has initiated. (This "feature" has been used to tremendous effect by hackers, whose victims launch a program that unbeknownst to them installs a virus onto their computers.)

Rules are covered in chapter 3.

Everything we see or hear while using digital devices is an abstraction. Very few of us really know what's happening when we use any kind of software or device. Just as examples, you're not really putting a "file" into a "folder" and "email" isn't really arriving into your "inbox." Those are all metaphors that allow us to understand the interactions that are going on. Anything you see, hear, or feel that helps you to understand the rules of the system is **Feedback**, the third part of microinteractions.

Feedback can take many forms: visual, aural, haptic. Sometimes the feedback can be prominent and unmistakable, like the lightbulb glowing when you flip the On switch. Sometimes it can be subtle and ambient, like the unread badges that appear on email applications and mobile apps. It can be as descriptive as a voice telling you exactly where to turn while doing turn-by-turn directions, or it can be as ambiguous as a LED light blinking in a complicated pattern. It can be as disruptive as the fart-like buzz of your phone in your pocket announcing a message, or a whisper as a digital panel opens. What is important is to match the feedback to the action, to convey information in the most appropriate channel possible.

In our iPhone mute example, mute has three pieces of feedback: a screen overlay when the mute button is turned on or off, and a tiny, visible strip of orange on the actual switch when mute is on. What doesn't appear—and what was the downfall of Patron X—is any indication that even though mute is on, set alarms will sound. There is also no onscreen indicator (other than the temporary overlay which vanishes after a few seconds) that mute is on. These are design choices.

Even more so than triggers, feedback is the place to express the personality of the product. Indeed, the feedback from microinteractions could be said, along with the overall form, to completely define its personality.

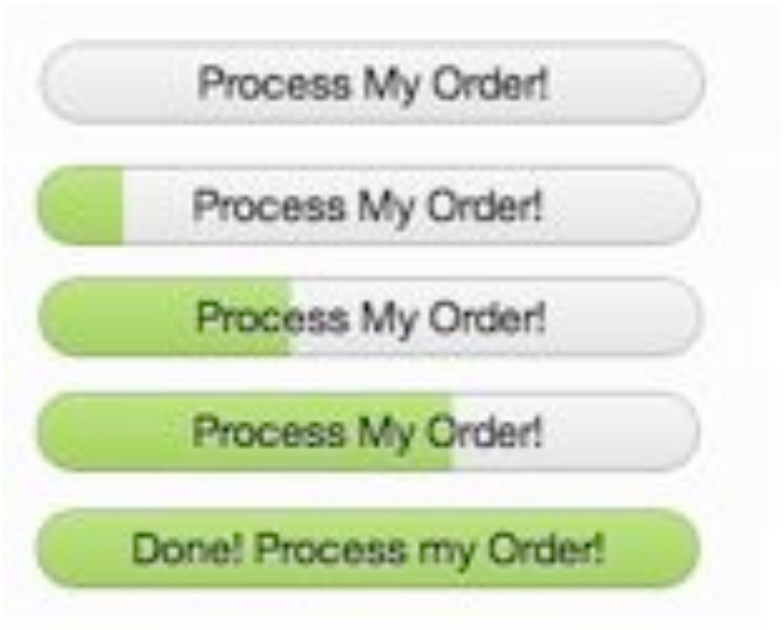


Figure 1.9 An example of Feedback. In Coda2, the Process My Order button becomes a progress bar when pressed.

Feedback is not only graphics, sounds, and vibrations; it's also animation and transitions. How does a microinteraction appear and disappear? What happens when an item moves: how fast does it go? Does the direction it moves in matter?

Like interactions (which are mostly rules) feedback can have its own rules as well, such as when to appear, how to change colors, how to rotate the screen when the user turns the tablet on its side. These rules may themselves become their own microinteractions, as users might want to adjust them manually.

Feedback is discussed in chapter 4.

The last part of microinteractions are the **Loops and Modes** that make up its meta rules. What happens over time with the microinteraction: do the interactions remain until manually turned off (as is the case with the mute switch) or do they expire after a while? What happens during an interruption or when conditions change?

Although it's often undesirable, some microinteractions have different modes. For instance, take the example of a weather app. It's main (default) mode is all about displaying the weather. But perhaps users have to enter another mode to enter the locations you'd like weather data from.

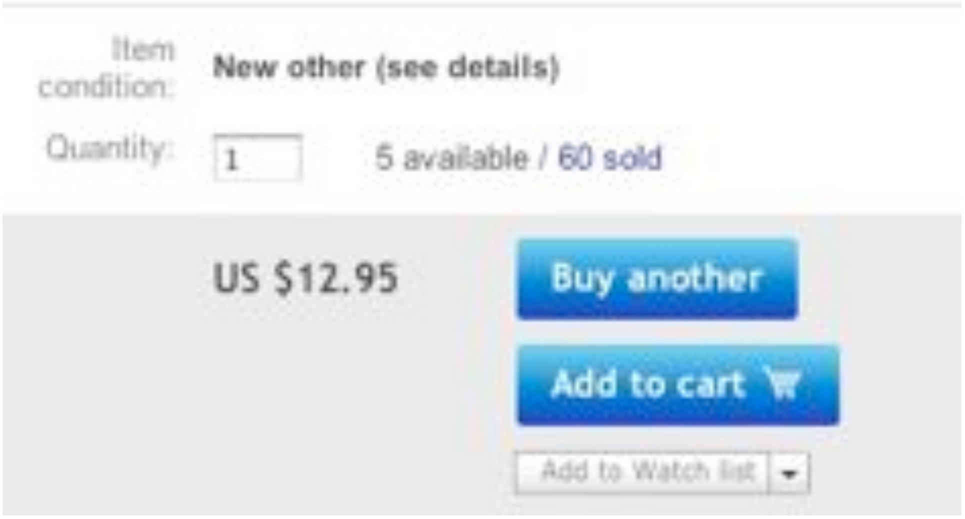


Figure 1.10 And example of a Loop. On eBay, if you've bought the same item in the past, the button changes from "Buy it now" to "Buy another."

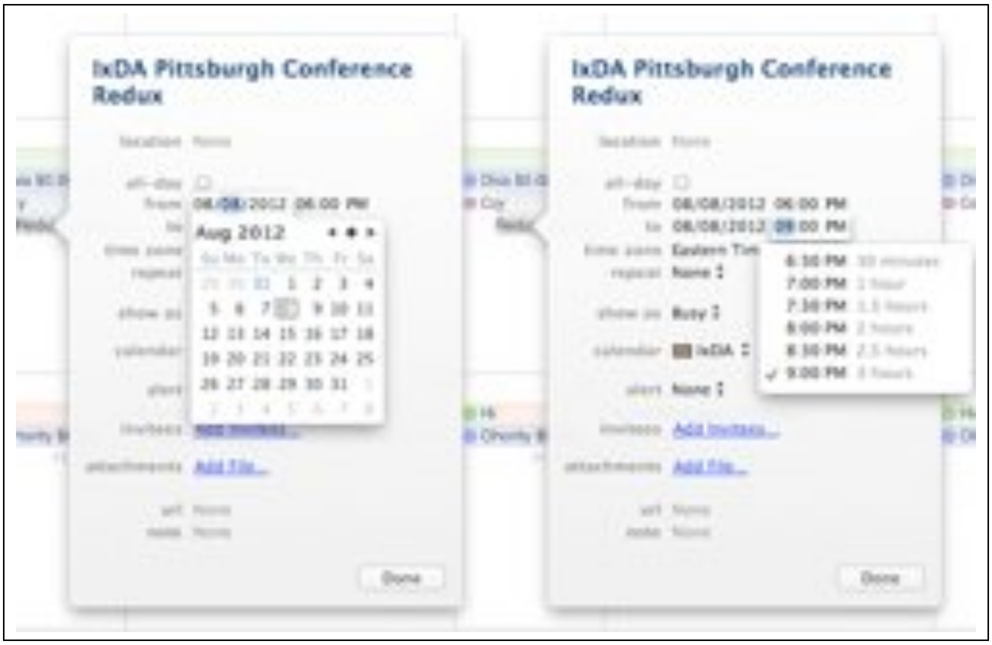
Loops and Modes are discussed in chapter 5.

UNCORRECTED PROOF

Microinteractions as a Philosophy

There are two ways of working with microinteractions. One is to think about them on a case-by-case basis. When, during the course of design project, try to identify any possible microinteractions. Make a list of them, then treat each as such. For each one, deliberately consider the structure as outlined in this book, and see if you can polish each individual component. You'll wind up with elegant microinteractions.

The challenge in working this way is keeping the scope of the microinteraction limited. The tendency is to turn them into features, because that is the way most designers are used to working. We want to tackle big problems and Solve Everything. Microinteractions are an exercise in restraint, in doing as much as possible with as little as possible. Embrace the constraints and focus your attention on doing one thing well. Mies van der Rohe's mantra of "Less is more" should be the microinteraction designer's mantra as well.


IxDA Pittsburgh Conference Redux

location None
all-day ☐
from 08/08/2012 06:00 PM
to 08/08/2012 09:00 PM
time zone Eastern Time
repeat None
show as Busy
calendar IxDA
alert None
invitees Add Invitees...
attachments Add File...
url None
note None

Figure 1-9. Even in the hated iCal, there is a nice microinteraction in the selection of a time. Rather than have you figure out how long it would be, iCal shows you event duration when selecting the end time.

A second way to think about microinteractions is to reduce more complex applications to individual products that are each built around one microinteraction. This is microinteractions as product strategy: your product does one thing and one thing well. Reduce the produce to its essence, its Buddha Nature. If you find you want to add another feature to your product, that other feature should be its own product. Many appliances, apps, and devices including the original iPod follow this model. This is how many startups work (or at least began), from Instagram to Nest. The "minimum viable product" is usually one microinteraction. Working this way justifies and provokes a radical simplicity to your product, that allows you to say no to feature requests as they arise. Of course, this is also a difficult stance to take, particularly in corporations where they want to sell one product that does everything their customers might need. Imagine breaking up Microsoft Word into individual products! And yet this is what some competitors have done. For example, apps like WriteApp are optimized just for writing, with most of the functionality of a word processing program stripped away, so that the focus is only on writing, for writers. Evernote began with a simple microinteraction: write notes that are available across platforms.

But there is a third way to think about microinteractions, and that is that many complex digital products, broken down, are likely made up of dozens, if not hundreds, of microinteractions. You can view a product as the result of all these microinteractions working in harmony. This is what Charles Eames meant when he said the details are the design: everything's a detail, everything's a microinteraction: a chance to delight, a chance to exceed users' expectations. As Dieter Rams said,

> I have always had a soft spot in my heart for the details. I consider details more important than a great draft. Nothing works without details. Details are the essentials. The standard to measure quality by[11].

In short, treat every piece of functionality—the entire product—as a set of microinteractions. The beauty of designing products this way is that it mirrors the smaller, more Agile way of working that many companies are embracing. Of course, the pitfall is that you can get lost in the microinteractions and not see the big picture, that all the details won't fit together into a coherent whole when you're finished. And working this way takes extra time and effort.

[11] Dieter Rams in conversation with Rido Busse (1980), reprinted in *Design: Dieter Rams &* (1981)

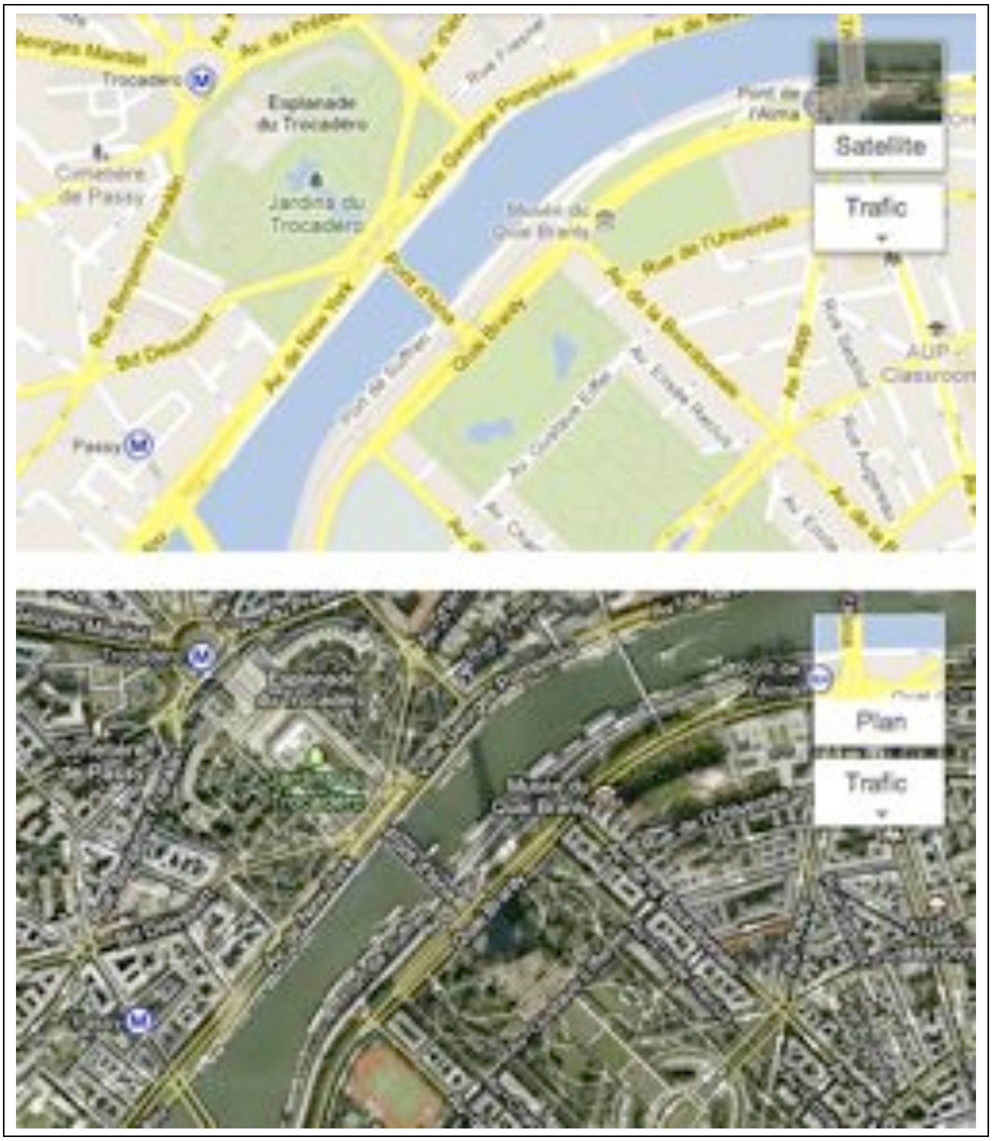


Figure 1-10. Whether in Standard ("Plan") or Satellite view, the widget for changing the view shows the map and a preview of the other view behind it.

This is also a difficult way for agencies—with their notoriously fast project schedules—to work. It's honestly a difficult way for any designer to work, as often the attention of clients and stakeholders are focused on the Big Features, not the small details that would enhance those features or improve the overall experience. Indeed, it can be difficult to get enough time to focus on microinteractions at all. Convincing business and development team members that microinteractions are worth spending time on can be a challenge. It will likely mean extra time for design and development, after all. But it's worth it.

The disastrous story of Patron X reminds us that microinteractions matter, that the designer's job is to take the tasks that could otherwise be frustrating and difficult and make them otherwise. Larry Tesler knew this when he decided there had to be a better way to move text inside a document and thus Cut and Paste was born. Microinteractions can improve the world, one tiny piece at a time. And they all start with a Trigger.

Summary

Microinteractions are the small pieces of functionality that are all around us. Focusing on them is the way to have a superior user experience.

The history of microinteractions stretches back to the first electric devices. Most of the digital standards we're used to now were once novel microinteractions.

Microinteractions are made up of four parts: a Trigger that initiates it, the Rules that determine how it functions, the feedback it generates, and the loops and modes that make up its the meta-rules.

There are two ways of working with microinteractions: either look for them and focus on each individually, or else treat every piece of functionality as a microinteraction.