THOUGHTLESS

THE DESIGN OF EVERYDAY THINGS

Karl Wiegers, PhD

TESTIMONIALS

"Karl Wiegers has written a delightful and practical book that challenges how we think about designing products. Karl's past experiences and publications identify him as a true expert in how to define software efficiently and effectively. His eye for details and thinking about how users will use products naturally extends to physical products too. Karl is absolutely an expert in how to write good software requirements; it only makes sense that he would equally be able to teach us about good design."

"As with Karl's other books, *The Thoughtless Design of Everyday Things* is lighthearted, entertaining, and thought provoking. He offers many tips regarding how to do thoughtful design and design mistakes to avoid, with loads of relatable real-world examples. You'll learn something about future product design and probably be more annoyed by those who haven't."

—Joy Beatty, Vice President at Seilevel

"Karl Wiegers's gift is his ability to look at an important topic and write a comprehensive study that gives practical advice, illustrated by real-world examples. And he does it in such a compelling manner! The densely packed wisdom, mined from Karl's years of design experiences, is interesting and insightful. I found it hard to put this book down; I just wanted to turn the page and see what came next. Anyone who designs items for real-world use—software, machines, medical instruments, homes, appliances, and the like—will find this book immensely valuable."

"Now, before I say my design work is done, I will review my design decisions against Karl's 70 design lessons. Most likely, I will find something I want to rethink, to make my work just that little bit better."

—Norman L. Kerth, software engineer
and author of *Project Retrospectives*

"Have you ever been irritated by a product that's way harder to use than you expected? Or outraged when something you've bought turns out to be useless for your needs? Why are such travesties so common? Karl Wiegers, well known in the software world for his excellent work on system requirements, explains. In *The Thoughtless Design of Everyday Things*, he calls out many of the egregious design errors that we've all encountered, drawing on examples from the kitchen, the car, the computer, and many other sources. Then he takes it further, using these "thoughtless design" examples to illuminate a specific design principle that the product designer should have followed."

"I enjoyed this book on several levels. Like all of Karl's writing, it is clearly written and easy to read. It's an entertaining diatribe about products that have made our lives hell at some point in time. I found myself saying, "Yeah, that exact thing happened to me." But, at a deeper level, I learned *why* designers tend to make such crazy errors. As a customer, I learned how to clarify my exact needs when choosing among rival products. And as a designer, I learned some valuable rules and processes for *thoughtful* design, including understanding the requirements for—and the contexts of—usage."

—Meilir Page-Jones, author of *The Practical Guide to Structured Systems Design* and several other software-design books

"This might be the most depressing book you'll read if, like me, you appreciate it when the products you use are well designed. On the other hand, it just might be the most entertaining read when you see the examples of calamitously bad products. Karl has put together here the most alarming, yet amusing chamber of horrors of poorly, thoughtlessly designed products. But it is not all on the dark side. Karl also brings you wonderful insights on design, design principles, how good design comes about, and how it makes so much difference to everybody's equilibrium."

—James Robertson, requirements guru and author of *Mastering the Requirements Process*

THE THOUGHTLESS DESIGN OF EVERYDAY THINGS

Karl Wiegers, PhD

J.ROSS PUBLISHING

ABOUT THE AUTHOR

Since 1997, Karl Wiegers has been Principal Consultant with Process Impact, a software development consulting and training company in Happy Valley, Oregon. Previously, he spent 18 years at Kodak, where he held positions as a photographic research scientist, software developer, software manager, and software process and quality improvement leader. Karl received a PhD in organic chemistry from the University of Illinois. During the past 50 years Karl has designed chemistry experiments, software applications and user interfaces, software development processes, books, songs, games, and training courses.

Karl is the author of the books *Successful Business Analysis Consulting*, *Software Requirements*, *More About Software Requirements*, *Practical Project Initiation*, *Peer Reviews in Software*, and *Creating a Software Engineering Culture*. He has written more than 230 articles on many aspects of software development and management, consulting, chemistry, military history, and self-help. Karl is also the author of a forensic mystery novel called *The Reconstruction* and a memoir of life lessons titled *Pearls from Sand*. Several of Karl's books have won awards, most recently the Society for Technical Communication's Award of Excellence for *Software Requirements, 3rd Edition* (co-authored with Joy Beatty). Karl has served on the Editorial Board for *IEEE Software* magazine and as a contributing editor for *Software Development* magazine.

When he's not at the keyboard, Karl enjoys volunteering at the public library, delivering Meals on Wheels, playing guitar, writing and recording songs, traveling, and wine tasting. You can reach him through www.processimpact.com or www.karlwiegers.com.

CHAPTER 1

THOUGHTLESS DESIGN

In the early 1960s, my father was an officer in the United States Air Force, stationed in southern Italy. His job there was launching nuclear-armed Jupiter missiles at Russia. Fortunately, he didn't have much work.

Each missile site had both an American and an Italian launch control officer. The launch control panel was designed such that two well-separated people must insert their keys into locks and turn them simultaneously to launch a missile. This separation was to prevent a lone maniac from initiating Armageddon. It seems like an effective design.

One day, my father's Italian counterpart opened the launch control panel from the back and showed Dad how it was possible to circumvent the need for two keys to initiate the launch sequence. He didn't offer a full-cycle demonstration, fortunately. All it required was a short piece of wire with alligator clips on both ends to bypass one of the most critical safety mechanisms in the world at that time. Maybe that launch control panel design wasn't quite ready for prime time.

The launch control panel reflects an extreme design shortcoming—the inability to thwart a malicious actor with potentially devastating consequences—but we are all surrounded by countless products with design problems that annoy us and waste our time. Some of these design issues are minor irritations; others can render products useless or even harmful in certain situations. I've often encountered a product that appears to have been designed by someone who never used a product of that kind before. Otherwise, they surely would have made some different design choices.

THOUGHTFUL AND THOUGHTLESS DESIGN

Don Norman published his classic book *The Design of Everyday Things* in 1988, with a revised edition in 2013 (Norman, 2013). Norman took a

psychological approach to understanding design thinking and how people interact with products. *The Design of Everyday Things* explored why people make mistakes when using products and recommended ways to design better products, prevent errors, and help users get the most out of the product.

It appears that not all of today's designers have taken Norman's wisdom to heart. Too many products still exhibit silly design failings. And since *The Design of Everyday Things* first appeared, we now have a vast assortment of new design issues with software applications, websites, and mobile devices.

Designers do put thought into creating their products, of course, even if the results sometimes fall short of our expectations. A thoughtfully designed product is a joy to use. It lets you accomplish a task efficiently, without aggravation, and without having to think too hard about it. As Norman (2013) points out, "Good design is actually a lot harder to notice than poor design, in part because good designs fit our needs so well that the design is invisible." In my own experience, I do find it far easier to think of thoughtless design examples than to remember those products that are designed really well.

When I say *thoughtless design*, I'm referring to products that could have been designed better without a great deal of additional effort. These products—both physical items and software applications—range from being unusable for their intended purpose to being acceptable but with need for improvement. Thoughtless design can lead to products that:

- Are not easy or obvious to use;
- Do not work well in realistic usage scenarios;
- Do not function properly in certain environments;
- Make it too easy for users to make a mistake;
- Do not provide the user with meaningful feedback;
- Waste the user's time;
- Are not designed for the user's convenience;
- Fail to accommodate the range of human variation; or
- Place excessive mental burdens on users.

Chapters 3 through 11 address each of these nine categories of design deficiencies in turn, illustrating the problems with many real examples. The examples are all drawn from my personal experiences or those that others have shared with me. I'll describe why these product designs are less than optimal, how many of them could be improved, and what designers can learn from them.

You'll also find numerous examples of especially well-designed products in this book. You can probably think of others from your own experience. They stand out because they're so much better than too many of the products we routinely encounter.

But shouldn't good design be the norm? Aren't consumers entitled to products that are obvious and easy to use, respect our time, work properly, and don't aggravate us? Yes, we are. I would feel differently if it took considerably more effort to create superior designs or if the resulting products would cost a lot more, but often that's not the case.

Here's a simple example. Everyone wears clothes, so it's surprising to encounter an item of apparel that makes me wonder, "What were they thinking?" I have a nice medium-weight blue jacket. It's well made and comfortable, with a cozy zip-out lining, but that lining has a strange inside pocket. The pocket is five inches wide, a good size to hold a phone, a museum brochure, or sunglasses. For some reason, though, the manufacturer sewed a vertical seam down the middle of that pocket, which splits it into two pockets that are each just under 2.5 inches wide. You can't put much into a pocket that narrow, maybe a nutrition bar or a couple of pens.

This pocket subdivision epitomizes thoughtless design. It cripples a useful feature while adding no value. If the designer had asked for input from typical jacket-wearers, I can't imagine anyone would have said, "That five-inch pocket is way too big. How about if you split it in two?" It's hard to envision the thought process that led to that design choice.

You might not agree with all of the examples I consider to be thoughtlessly designed. Some properties I find puzzling or annoying might suit your preferences just fine. There are lots of consumers, with different characteristics, needs, expectations, and tolerances. Their needs often diverge or conflict, such that no single solution will delight everyone.

It's worth considering why a designer made the choices that resulted in a product that doesn't seem quite right to you or to me. Perhaps the nature of the product constrains the design such that there's only one possible solution, which might not be ideal. The designer could be working under constraints we consumers aren't aware of: regulatory restrictions, cost or competitive pressures, compatibility issues, and the like. Complex products are developed by multiple teams that may not communicate, collaborate, and compromise effectively, leading to parts of a product that are great and others that fall short. In too many cases, though, the product's design is simply flawed. Chapter 2 explores why

design is difficult; the rest of the book looks at designs that could be better and how to improve them.

ABOUT THIS BOOK

This book is for people who design or use anything: software systems, websites, automobiles, consumer products, bathrooms, appliances, user manuals—anything. Designers can apply the design principles and lessons I present to make a wide variety of products better suit customer needs. Consumers can learn to think more clearly about what they're really looking for before they buy a product. Managers will learn about the inescapable trade-off decisions they must make to reach the optimal balance of product attributes that will satisfy a diverse group of stakeholders.

There are many cases of large-scale design failures that did enormous damage, cost huge sums of money, and sometimes killed people (Shariat and Saucier, 2017). Television series like "Deadly Engineering" and "Engineering Disasters" tell the tragic tales of collapsing bridges and buildings, rupturing dams, and nuclear reactor accidents. I've watched episodes about new skyscrapers shedding windowpanes or tiles that plummeted to the streets below, endangering passersby. Airplanes have crashed because of hardware or software design flaws. Rockets and spacecraft have exploded on the launch pad or in flight. Huge ships have gone to the bottom on their maiden voyages. Engineers study such failures to learn how to avoid them in the future.

I don't discuss such large-scale catastrophes here. Nor do I focus on irritations with products or processes that don't specifically relate to design problems. Yes, there are way too many of those: poorly made products, buggy software with features that don't work right, processes that make no sense. A friend told me about a relative of his who could pass the vision test for renewing his driver's license only by wearing his reading glasses, even though he can't safely drive with those glasses. I find such things as ridiculous as you do, but that's not what this book is about.

My focus here is on perplexing design shortcomings in the consumer products we all encounter. Many products contain small defects or lack some functions you might like. However, I'm most concerned with silly issues that suggest the designer didn't consider real users, their usage scenarios, their environments, and their expectations.

I'll present dozens of examples of what I consider thoughtless design of both physical devices and software products. These examples violate the nine guiding design principles explored in Chapters 3 through 11. As we can gain powerful insights from problems, I call out 70 specific design lessons from the examples I describe. These lessons are collected in the Appendix for easy reference.

Many designers are well schooled in design principles and create useful, clever, and even brilliantly conceived products. I have also included examples of particularly thoughtful designs to show how well it can be done. Less experienced and less educated designers must accumulate expertise through trial and error, self-education, and learning from proficient mentors. Therefore, the final chapter describes numerous specific practices that can help any designer create pleasing products.

In most cases, I don't identify the manufacturer of the product I'm using to illustrate what I consider thoughtless design. I do identify the producer for some examples where it's particularly relevant or unavoidable, such as specific apps or devices. My purpose here is not to criticize particular companies, but rather to draw general lessons that can help designers do a better job on their next product. Software systems, websites, smartphones, and other products are modified over time, so some of my examples might look or function differently in the future. All of the examples presented are accurate as of the date of writing, though.

A THOUGHTLESS DESIGN CASE STUDY: THE SPONGE MOP

My wife once bought a new sponge mop. It seemed to have the necessary mop features and properties, including a handle for squeezing out excess water (Figure 1.1). We've found that the squeeze handle was useful on other mops we've owned.

However, we quickly discovered a design shortcoming in this squeeze handle. When you place the mop in the mopping position with the sponge on the floor, the squeeze handle also drops to the floor (Figure 1.2). There ought to be some way to clip the squeeze handle onto the mop handle when you aren't using it, as I've seen with other mops. We just assumed any reasonable designer would incorporate a clipping mechanism, because it's a nuisance to have the squeeze handle drag on the floor. I wondered if the designer ever used this mop before sending the specifications off to manufacturing.

Figure 1.1 This is an apparently reasonable sponge mop.

On closer examination, I discovered that the mop is indeed designed to clip the squeeze handle in place, but the clip doesn't work right. The latching mechanism is too weak to hold the squeeze handle—it pops right out again. The design is still flawed, but it's the latching mechanism itself that is poorly designed. That's a different problem than neglecting to include a latching mechanism at all. Someone who tested a prototype of this mop should have noticed immediately that the latching mechanism's current design was inadequate.

Beyond this specific design failing, there are three messages here that are worth exploring further:

1. The difference between quality and crap is often small.
2. Consumers don't always think carefully about their requirements before buying a product.
3. Product problems are not always design failures.

Figure 1.2 In reality, experience showed it to be a thoughtlessly designed sponge mop.

THE CRAP GAP

Hold your hand up with your thumb and index finger about one inch apart, like in Figure 1.3. In many situations, that short distance represents the difference between quality and crap (Wiegers, 2019c). Often, all it takes to close the crap gap is to do some more questioning, listening, thinking, measuring, designing, prototyping, or testing before you deliver a product or declare a job complete.

Bridging the crap gap takes little effort in many cases. It's a matter of measuring twice, checking one more time, and then making the cut. It's thinking about how people will use a product, not just its features, and using checklists to avoid overlooking something important. It's asking one more person for input, confirming a decision before proceeding, and validating your assumptions. It's testing a product under realistic conditions instead of merely assuming the design will work right. It's not cutting corners to make a bit more profit on an item that will quickly disappoint consumers and drive them to buy a replacement.

Many of the examples of thoughtless design that we'll encounter will exemplify the crap gap. You'll see that it wouldn't have been much harder

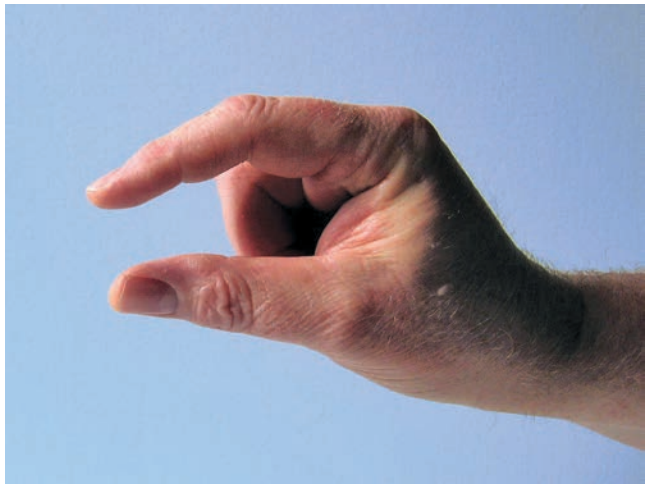

Figure 1.3 Beware the crap gap!

to design the product so as to avoid the problems that I describe. And you might wonder why a designer or manufacturer didn't get it right in the first place.

CONSUMERS AND THEIR REQUIREMENTS

Buying products with design deficiencies is partly our own fault. We consumers often do not think carefully about exactly what our requirements are before we go shopping. I need a sponge mop, so I go to a store like Target and look through their mop offerings. Ah, I like the kind with the squeeze handle, so I'll buy this one. I use the mop for the first time, perhaps a couple of weeks later. Only then do I discover the problem. "I can't believe they're selling a mop with a squeeze handle that drags on the floor!" I say, exasperated. By then, I've probably lost the receipt and can't return the annoying mop.

Designers, engineers, and developers base their work on specified product requirements. I've spent much of my career as a software engineer and consultant exploring better ways to define the requirements for software systems. The terms *assumed requirements* and *implied requirements* make my blood run cold. You cannot expect someone else to read your mind to learn just what capabilities and characteristics you expect in the product. As systems analyst Meilir Page-Jones says,

"Requirements are *always* made explicit eventually, preferably before the product is built, but too often afterwards. The difference in cost and frustration is enormous."

But we do assume certain requirements all the time with everyday products. We have a vague notion of what we're looking for, we acquire an item that seems reasonable, and then sometimes we're disappointed. Rarely do we carefully specify our requirements in advance and then assess each candidate product against them. We expect products both to be designed well and to work correctly when used as intended.

Perhaps you need a new kitchen spatula, so you simply buy one. Maybe you differentiate between a metal spatula and a plastic one that won't scratch your nonstick pans. You expect it to be pretty much like all the other spatulas you've ever used and that it will work just fine.

However, you might discover a problem the first time you make pancakes. Figure 1.4 shows a pair of spatulas that I own. I love the one in the foreground. It's comfortable and effective. The one in the background, however, is awkward to use. Since I'm not seven feet tall, that sharp angle between the handle and the spatula head forces my arm into an uncomfortable position when I slip the spatula beneath something in a frying pan. It makes me wonder whether anyone ever tried to flip a pancake with the second spatula before manufacturing thousands of them.

Figure 1.4 I prefer the comfortable and effective spatula (near) over the awkward spatula (far).

Perhaps this spatula wasn't intended for pancakes at all. Maybe it was designed to work well with certain kinds of steep-sided pans that I don't own, but it's still awkward to use. I could have thought more carefully about exactly what I needed before I headed to World of Spatulas.

As consumers, it's not our fault when basic items don't do a reasonable job, but we do have some responsibility to think through our requirements before pulling out the credit card. I've tried to train myself to be a better-prepared consumer so I know which product characteristics to look for and which to avoid. For large and infrequent purchases like a car or a house, I use detailed checklists and spreadsheets to help me assess the options and choose the right one. And as a consumer who cares about quality and products that last, I'm willing to pay a bit more if necessary, rather than settling for junk. However, even careful analysis won't help if a product—like the sponge mop—seems to possess the desired feature but fails in ordinary usage.

IS IT REALLY A DESIGN PROBLEM?

What might at first appear to be thoughtless design could have other explanations. There's a difference between conception and execution. Sometimes it's hard to tell if a product is poorly designed, is well-designed but poorly manufactured, is made with low-quality materials, or is simply defective. A friend of mine has had some problems with his dishwasher:

> *You would think both racks would glide in and out easily, a basic mechanical function you expect of a dishwasher. However, the top rack jams every time I push it in, and I have to lift or wiggle it back into place.*

This sounds like a design problem. But maybe there's some other root cause, like a poorly manufactured rack or even a missing part. Frankly, consumers don't care what the explanation is. They just want a new product to work right.

My Honda has a speech recognition system that lets the driver control navigation functions, climate control settings, and the audio system by voice. However, it responds to spoken commands s-l-o-w-l-y. Using the dashboard controls is much quicker than wading through the voice menus. The advantage is safety; I can make changes by voice with less distraction than if I have to take my eyes off the road to locate the correct

dashboard button. Unfortunately, this advantage is negated if I must glance at the display screen to remember the keywords to use for making adjustments. Also, the recognition accuracy isn't very good. Once when I told the speech recognition system I wanted to exit from the function, it turned on the air conditioner. Maybe "exit" sounded like "A/C" or something.

The mediocre speech recognition accuracy renders the system practically useless for navigation. It does get some things right, and it's a real convenience when it works. However, when I ask the system to "Find Place," it finds the location I'm seeking only about 20 percent of the time. It's vastly easier, faster, and more flexible to use the excellent speech recognition on my iPhone and display the map on the car's navigation screen via Apple CarPlay.

Now, is this speech recognition failure a design problem, a component problem, an implementation problem, or user error? Perhaps the microphone isn't sensitive enough or is too sensitive for accurate recognition, or maybe it's positioned poorly in the car. Maybe the software needs to be trained to better recognize each driver's voice, but the car provides no such option. Maybe I don't speak clearly or loudly enough, or I could be talking too loud or too fast. I don't know what the cause is. The result is that I rarely use the car's voice control system because it's so slow and error-prone, although it seemed like a cool feature when I bought the car.

Nontechnical considerations also can lead to subpar designs. An existing product's superior design might be protected by patents, constraining competing manufacturers either to license that better design or to invent an alternative approach. That's what happened when Kodak—where I worked for 18 years—entered the instant camera market in the 1970s. Because of certain features that Polaroid had patented previously, Kodak had to use completely different film technology and camera design approaches. The resulting camera was bulkier and more awkward to use than Polaroid's.

So, while there are many reasons why products can be deficient besides design flaws, thoughtless design is all around us. Designers can do better; consumers are entitled to it.

SOME TERMINOLOGY CONVENTIONS

One person might design both the software and the user interface for a small mobile app, but teams design complex products. In this book, I will

often use the term *designer* as a singular term, though I recognize that many people could have contributed to the overall product design.

The *customer* who acquires a product or specifies its requirements isn't always the ultimate, or end, user. A *user* directly interacts with the product, benefiting from the delights of thoughtful design and suffering the frustrations of thoughtless design. Some software systems also have indirect users who receive outputs from it but don't interact directly with the system themselves. A *consumer* could be an acquiring customer, a user, or both.

I should define some other terms that will recur throughout the book. A dictionary definition of *obvious* will suffice: easily seen, discovered, recognized, or understood. A product feature is obvious if its presence and usage are readily apparent without further explanation to anyone who engages with the product.

A product is *intuitive* if you can make a good guess at how to use it based on its similarity to other products from your experience or from other cultural or environmental cues. An intuitively designed product is one a user can understand and employ immediately, without consciously thinking about it or referring to a manual (Interaction Design Foundation, n.d.).

A feature or product is *easy to use* if the user can successfully perform a task with minimal effort and assistance. To make a product easy to use, a designer must understand who its users will be, their goals and expectations, their backgrounds, and the anticipated contexts of usage. Performing even a complex task with a software system could be regarded as easy by an expert who has done it many times before, whereas someone encountering the product for the first time could find it highly difficult to use.

A KEY ASSUMPTION

I make the assumption that product designers and engineers do the best job they can with the information and skills they have available at the time. Perhaps that's not always true, but I like to think it is. I'm not trying to be overly harsh on designers here, and I'm not accusing them of malice (except perhaps for a few cases described in Chapter 9). My examples are opinions about specific products that suffer from what some consumers consider to be thoughtless design. Sometimes things that are not obvious initially do turn out to be clear on a second look or to someone

else's eyes. We each interpret the world in the context of our own experiences and mental models. And we can't always put the whole onus on the designer. No matter how well a product is designed, someone somewhere will fail to make it work or will manage to break it.

A colleague of mine had previously worked in a call center providing technical support to software users. His group had an unofficial problem classification category of "user brain damage." We all make silly mistakes; we all misunderstand, overlook, and forget things. A woman I know related this experience:

> *One time a friend had me drive a hundred miles because her computer would not talk to her printer no matter what setting she selected. I got there. Using the power switch on the printer, I turned the printer on. Problem solved. We have all suffered brain blackouts.*

Many people like to say, "The customer is always right." That simply isn't true. Sometimes the customer is unreasonable, misinformed, or in a bad mood, which can lead to complaints that don't reflect design problems at all. Sometimes, though, a nugget of wisdom in the complaint can help us make the next design even more obvious, intuitive, and easy to use.

I used to carpool with a man who complained about the restricted view his new car offered through the rear window. I asked if he had noticed the visibility problem on his test drive. George sheepishly confessed that he hadn't taken the car on a test drive. If he had, he might have spotted the rear window problem before he bought it. Maybe the design really was deficient, but George should have explored the car more carefully first.

People sometimes use products incorrectly because they don't understand them well. Some hotel bathrooms have heat lamps in the ceiling light fixture. To save energy, many hotels have switched their incandescent light bulbs to more efficient LED bulbs. I've stayed in hotels that also had replaced their bathroom heat lamps with LEDs. LEDs make the bathroom brighter, but because they put out very little heat, they don't warm up the room. This confusion about the bulbs isn't a design inadequacy, but rather a matter of the hotel staff not understanding the product.

Consumers must take some responsibility for assessing their needs, evaluating candidate products, and becoming familiar with how to use them before they complain about product deficiencies. Even so, we're

still surrounded by products that disappoint and irritate us. Things should just work properly.

LOOKING AHEAD

In Chapter 2, I'll explore why it truly is hard to design products that perform in a way that makes consumers happy. I provide some background about requirements as the foundation for design, designing for usage rather than simply cramming in more product features, and the tensions that designers face as they attempt to satisfy multiple stakeholders who have conflicting expectations and constraints. If you don't need that technical background, feel free to skip ahead to Chapter 3.

Each of the subsequent nine chapters addresses a basic principle of effective design:

Chapter 3: Make the product easy and obvious to use
Chapter 4: Consider realistic usage scenarios
Chapter 5: Consider a wide range of usage environments
Chapter 6: Make it hard to make a mistake
Chapter 7: Provide meaningful feedback
Chapter 8: Don't waste the user's time
Chapter 9: Design for the user's convenience
Chapter 10: Accommodate the range of human variation
Chapter 11. Place the minimum mental burden on the user

Each chapter describes numerous examples of products that violate the design principle. Other cases of thoughtfully designed products exemplify the intent of each principle. We'll also see many lessons for designers to keep in mind as they strive to create better products. You can read these chapters in any sequence you like.

Finally, Chapter 12 recommends several techniques for creating more thoughtful designs. I'll discuss the importance of usage-centric design; ways to engage customers in the design process through product champions, personas, prototypes, and usability testing; and learning how to improve our future designs from project retrospectives and customer feedback. But first, let's see why creating good designs is so challenging.

CHAPTER 6

MAKE IT HARD TO MAKE A MISTAKE

One summer, I visited an art museum in the Netherlands. It had many glass doors between rooms. The doors all had identical graspable handles on both sides, but all the doors operated as push-to-open in one direction and pull-to-open in the other. I tried the wrong opening action just about half the time I approached a door—the 50-50 statistical outcome of simply guessing which way the door works.

Anytime the correct way to use an object isn't apparent and results in errors—such as pushing on the wrong side of a door or pushing the handle when you should pull it—the designer should have included signifiers to make usage clear (Norman, 2013). One way to indicate correct usage is with written signs. If I see a panel labeled "push" on one side of a door, I'm pretty sure I can open it successfully on the first try. But the very need for signs indicates a design failure (Norman, 2013). Explanatory signs also pose problems for users who can't read the language used.

An alternative, text-free design uses actuators that are shaped differently on the two sides of the door: a flat panel that indicates "push me" on one side and a handle inviting "pull me" on the other. Such unambiguous and universal signifiers can prevent people from feeling silly as they struggle to open a door.

Product designs that unintentionally facilitate user mistakes can have far-reaching consequences. In the year 2000, the presidential election in the United States was marred by confusing ballots in some locations. A significant number of citizens voted for unintended—or even multiple—candidates. Ballot design problems led to recounts, delayed the election's resolution, triggered a legal battle that reached the Supreme Court of the United States, and very probably determined the election's outcome (Mestel, 2019). A field guide titled "Designing Usable Ballots" is now available that—if heeded—should greatly reduce such problems in future elections (Center for Civic Design, 2020).

One of the most important principles of designing for usability is to:

DESIGN LESSON #27:
Make products easy to use correctly and hard to use incorrectly.

Designers should contemplate how both physical products and software user interfaces could be used incorrectly and then strive to prevent or to mitigate such problems. Allowing users to do what they want to do with the product is only half of the design solution. Protecting them against making mistakes or hurting themselves is the other half.

TO ERR IS WAY TOO HUMAN

Effective design marries a user with a product through a user interface, removing distance and friction between the two. When there's a gap, there are two ways to bring user and product closer together:

- Move the product toward the user through a better-designed user interface.
- Move the user toward the product through training and guidance.

When an operational failure occurs—especially in safety-critical systems where the stakes are high and someone or something must be held accountable—too often the user is quickly blamed. Rather than simply declaring "user error," people should recognize that the design marriage just didn't work. There is likely fault to some degree attributable to both user and system. We don't want errors of any sort to prevent users from successfully completing a task, so we must design products toward that end.

Requirements explorations naturally focus on the product's expected behavior when everything takes place normally. However, experienced requirements analysts, designers, engineers, and software developers know they also must consider the many things that could go wrong and how to prevent them if possible, to detect them if they occur, and to recover from them if necessary. Error conditions in software are called *exceptions*. If you don't identify exceptions and determine how to handle them during requirements development, the designer might overlook some possible failure conditions or unexpected user actions. Those oversights can result in unpleasant surprises when a user later encounters

the unhandled exception. Then you get to fix the product and try again, which isn't much fun.

Robustness was one of the design-for quality attributes listed in Table 2.1. Robustness has to do with how well a product handles unanticipated inputs and unexpected conditions that arise during operation. Robust designs respond to exceptions in sensible ways. They make it hard for a user to make a mistake that results in a failure. Robust products recover from internal system or communications problems smoothly. What we call "user error" sometimes results from an insufficiently robust product design.

When I was working as a business analyst on software systems, users would sometimes tell me, "The system should handle errors gracefully." As stated, that's a poor requirement: it's vague, subjective, and unverifiable. Nonetheless, the request informed me that robustness was an important user expectation. I knew it was a topic we needed to explore further so I could specify just what the users meant by "gracefully" and guide designers toward that goal.

Designers must recognize that users aren't always precise in their thoughts and inerrant in their actions. Users will sometimes enter invalid data, start an operation by mistake, misunderstand how to do something, or take longer than expected to complete a transaction. A few might even deliberately try to break the system, preferably during testing, but sometimes for fun or out of malice.

Products also can experience problems that have nothing to do with user actions, arising from occurrences in the usage environment or interactions with other components. Software systems and electronic devices will drop network connections, fail to interface properly to other systems or devices, hang, and present users with confusing or incomplete options. Data can be missing or corrupted. A function that worked fine yesterday can go crazy or die today (this just happened to me; I haven't been able to fix it yet).

It's the designer's responsibility to anticipate less-than-ideal behavior—from people, devices, and software—and to handle situations that could otherwise result in a failure. As Don Norman (2013) pointed out:

> *It is easy to design devices that work well when everything goes as planned. The hard and necessary part of design is to make things work well even when things do not go as planned.*

Your problem analysis should consider what could go wrong, how severe each incident could be, and how to prevent or mitigate it. Once you've

identified possible error conditions, use the scale of options in Figure 6.1 to decide how to deal with them.

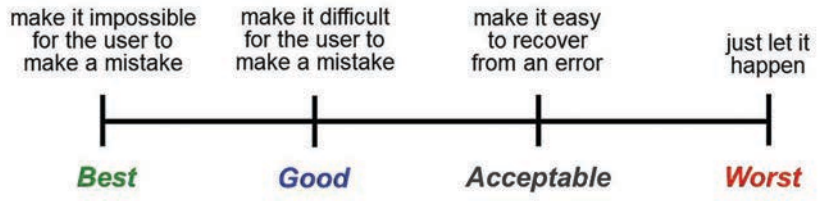

Figure 6.1 There are four ways to handle user and system errors, some more desirable than others.

MAKE MISTAKES IMPOSSIBLE

Error prevention is always preferable to error recovery or correction. Usability experts Larry Constantine and Lucy Lockwood (1999) said, "Good software can be thought of as both flexible and forgiving." Designers do their users a great service if they anticipate possible error conditions and build products that protect users from mistakes. Constantine and Lockwood's Tolerance Principle states this nicely:

> *Be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonably.*

With Software

It's exasperating when I'm trying to perform a task with a website or a software application, only to have the system inform me after I've entered all my data that it can't complete the task because of some error condition. My reaction might be, "Hey, stupid computer, you knew two screens ago that you wouldn't be able to finish my task. Why did you waste my time and let me keep going?"

An important component of a use case analysis is to identify specific prerequisites—called *preconditions*—that must be satisfied to potentially enable success. If those preconditions aren't fulfilled, the system shouldn't even allow the user to initiate the use case. The system should communicate what the problem is and, if possible, help the user take

actions to satisfy the preconditions so they can continue with the task. The message here is:

DESIGN LESSON #28:
Detect unsatisfied preconditions and erroneous inputs early so the user doesn't waste time on a task they cannot complete.

Suppose I'm hungry, so I decide to place an order for takeout at a restaurant's website. I browse through menus (actual food menus, not computer menus), select several tasty-sounding items, and add them to the cart. The website calculates the price and I add a tip. After clicking on a button to pay, a message appears saying that I must first set up an account with the restaurant, possibly losing my cart contents in the process.

A considerate designer would not have allowed me to add items to the cart without telling me about this prerequisite condition. An even more considerate designer would give me the option to create an account just prior to the payment step without losing the cart contents. Customers won't be happy if they must to go to a separate page to create an account and then begin the shopping process all over.

I can't tell you how many times I've typed text into a web form to report a problem, request assistance, or respond to a survey question, only to have the site complain about my input when I tried to submit it. Sometimes a form field has a character limit, but the web page doesn't tell me about it. I don't know what the maximum number of characters is until I've typed too many. It might truncate and discard the excess characters I entered above the limit. Or, it might report that my input is too long, but I still have to guess at the maximum length. It's especially irritating when the input field still contains plenty of space after I've reached the invisible character limit. Such thoughtless design wastes my time and diminishes the site's appeal.

The best solution to the field length problem is to eliminate arbitrary size restrictions. Assuming we do have a limit, though, Figure 6.2 illustrates a friendlier approach. The counter below the input field shows the user the maximum number of characters and the number of additional characters they can enter at any time. The field stops accepting text when the user reaches the limit, rather than taking whatever the user types and then making him whittle it back to fit. These considerations reduce the surprise factor the user could otherwise experience after inputting all of their data.

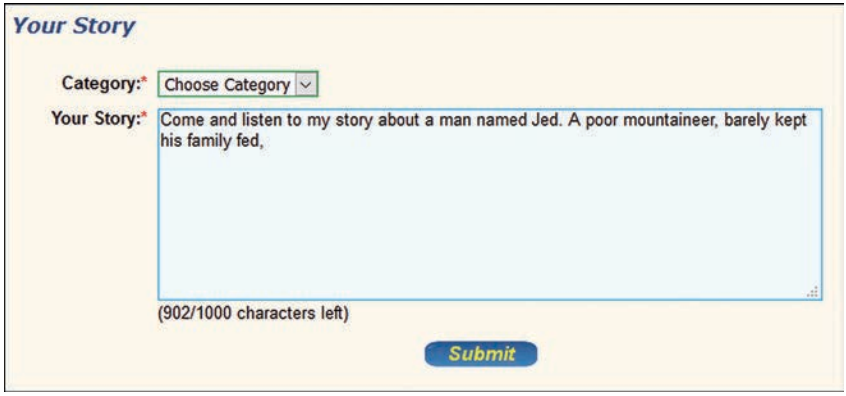

Figure 6.2 This input form tells the user exactly how many characters they can enter into a field.

If certain characters aren't permitted in a field for legitimate reasons, inform the user as soon as they enter one or even earlier, right up front. Block or highlight the invalid character, and provide a meaningful message so the user can correct it right away. Preventing erroneous input is much better than waiting until the user has completed their entry and then reporting that there's a problem. I've seen messages that complained about an invalid character in my input but didn't tell me which one(s) it was, so I had to guess. The computer knows exactly what's wrong, or my input would not have failed validation; it should just tell me. There's another lesson here:

DESIGN LESSON #29:
Notify users of invalid input immediately, clearly describe the problem, and guide them to correct it.

With Physical Objects

Designers of physical objects have an effective way to prevent mistakes: they exploit asymmetry to permit items to be connected in just one way. Polarized North American electrical plugs can be inserted into a socket in only a single orientation because one plug prong is wider than the other (Figure 6.3). A smartphone's SIM card tray constrains the SIM card into the correct orientation. Strategies that exploit asymmetry to

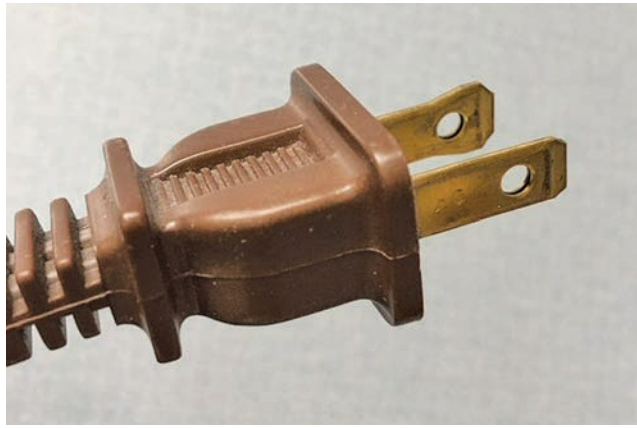

Figure 6.3 Polarized electrical plugs can be inserted in only one way because the prongs are different sizes.

make sure objects can only be connected properly demonstrate a useful design lesson:

DESIGN LESSON #30:
Employ physical or logical constraints to make it impossible for the user to make a mistake.

In principle, using physical constraints is an excellent strategy. It allows people to assemble kits properly or to correctly reassemble some object they had dismantled for repair. In practice, though, using asymmetry can fail in several ways.

One problem arises if the two components are not sufficiently asymmetrical and properly mated to prevent the user from connecting them incorrectly. I have an electric toothbrush that mounts on a charger base. A peg on the base fits into a hole in the bottom of the toothbrush handle to constrain it to the correct position (Figure 6.4). The peg and its corresponding hole are roughly trapezoidal in shape. This shape permits attachment in only one way, unlike a rectangle or an oval (two ways), a square (four ways), or a circle (infinite ways).

However, I have sometimes noticed that the toothbrush handle is positioned backward on its stand. The hole in the bottom of the toothbrush handle is larger than the matching peg in the handle's base. This loose fit

Figure 6.4 An asymmetrical peg forces the electric toothbrush handle to mount on its base in just one way. Or does it?

makes it too easy to inadvertently replace the handle rotated 180 degrees from how it's supposed to be, which somewhat negates the design. The concept was sound, but the execution could have been better.

Older USB 1, 2, and 3 connectors provide another asymmetry example: they can be joined in only one way. However, it's difficult to tell what that way is if you cannot see both the cable plug and the receptacle. The receptacle could be located in an awkward spot on your car's dashboard, on the back of a computer, in an electrical outlet near the floor, or on the back of a wall-mounted television. Hidden USB connectors present the same lack of signifiers you sometimes see with doors, where you have just a 50-50 probability of getting it right on the first try.

It's not always possible to devise a perfect solution to such problems, but tactile methods can help. Many USB cables have the universal USB symbol printed on the open side of the flat connector. More useful are cables with the USB symbol molded into the plastic plug so you can feel the cable's orientation even if you can't see it. Still, neither of these helps if you can't see the receptacle.

The best solution for electrical connectivity challenges is a symmetrical plug that will work in either orientation. For instance, the newer USB-C standard and Apple Lightning connectors will work whether they're attached face up or face down. No constraint or signifier is needed—you can't get it wrong.

Thoughtful Design: 3D Motion Model

I recently encountered a superb example of how to use asymmetry to ensure correct assembly. I received as a gift a kit containing several sheets of laser-cut wooden parts that you assemble into the three-dimensional motion model shown in Figure 6.5. As you turn a crank, gears and lifts carry steel marbles up to the top, where they run through a series of troughs and collect back down at the bottom so you can do it again. It's entertaining and an especially clever design.

This kit contained nearly 300 pieces and connectors. It was a bit intimidating. The pieces are held together entirely by friction: slots mate together, tiny wooden pegs join pairs of pieces, and tabs fit into holes

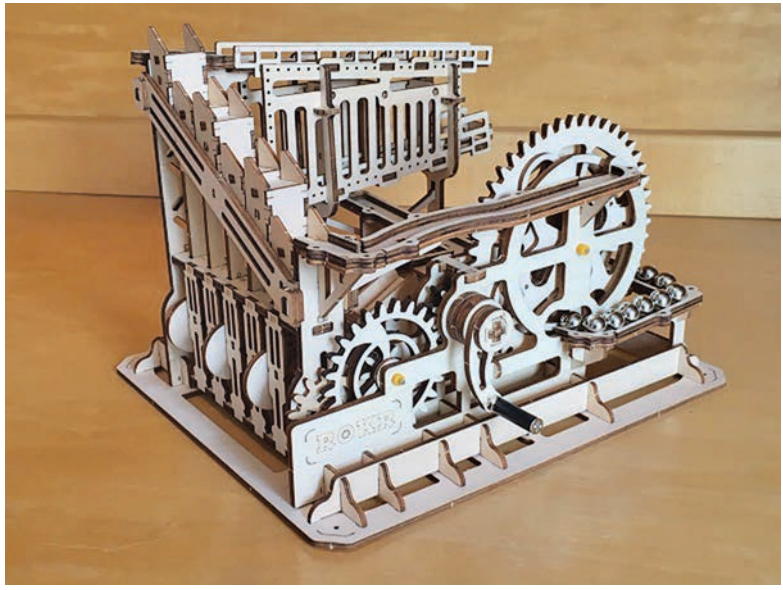

Figure 6.5 The kit for this 3D motion model was easy to assemble because of the clever ways the designer exploited asymmetry in the parts.

cut in certain pieces. The brilliant design exploits asymmetry to ensure that even someone who isn't mechanically inclined, like me, could assemble it correctly. To connect two pieces that could potentially attach in more than one way, I just had to press the rectangular tab into the rectangular hole and the square tab into the square hole. I couldn't go wrong. It helped that the assembly instructions—almost entirely visual, not textual—were clearly drawn, so I could easily see how to fit the pieces together. It was a fun project, and now I have an amusing new toy.

Thoughtful Design: Garden Hose Sprayer

Sometimes even a gadget that's been around for decades can be rethought for improved usability. Consider the venerable garden hose sprayer. Figure 6.6 shows a typical inexpensive sprayer like those I used for decades. If you ever dropped one on the ground, you could bet it would land handle-down and spray water everywhere.

Now, compare that design with my new one, shown in Figure 6.7. The trigger handle is on the *inside* of the pistol grip. If you drop it, not to

Figure 6.6 If you drop this traditional hose sprayer, you might get soaked.

Figure 6.7 A better hose sprayer design prevents accidental soaking.

worry: the ground can't press the trigger and soak you down. The plastic body doesn't get as uncomfortably cold as the metal body on the traditional model, and the textured surface provides a secure and comfortable grip. The nozzle offers seven different spray patterns. I like this hose sprayer much better than the old kind.

MAKE MISTAKES DIFFICULT

Sometimes when I'm driving my car, I hear a beeping sound, indicating that the speech recognition function is active, even though I didn't consciously press the button to activate it. The steering wheels of modern cars include many buttons and switches, often crowded close together. Figure 6.8 shows the 19 controls on my car's steering wheel; speech recognition is on the bottom left. It's too easy for my fingers to hit a button by accident. A friend who complained about a similar design problem told me, "At least half of the times that I make a sharp turn, I change the station on the radio. The button is just too close to the perimeter of the wheel."

It's also easy to press an unintended control by mistake. My cruise control and audio system use the identically shaped circular controllers

Figure 6.8 My car's steering wheel is packed full of controls.

on either side of the Honda logo in Figure 6.8. I have occasionally pressed one of the cruise control buttons on the right side of the steering wheel in a futile attempt to change a setting on the audio system, which uses the left controller. My brain just did a symmetry switch by mistake.

Modern cars contain many systems and functions. The driver can control the entertainment system, cruise controls, climate controls, lights, navigation, traction controls, vehicle information displays, cameras, and perhaps varying degrees of driving automation. The proliferation of buttons, switches, and knobs that these controls require is part of what led BMW to introduce the iDrive controller that I described in Chapter 3.

There's no ideal solution to this surfeit of controls, but it's something designers need to be very aware of. It's not enough to provide the user with logically placed, symmetrically arranged, aesthetically designed buttons and switches. Designers also must think carefully about possible mistakes the user could make while manipulating the controls, consider how much risk each such error could pose, and then modify the design if necessary.

Because it is so much better to prevent errors than to report and correct them, software designers also must consider how to make it hard for users to make damaging mistakes. It's easy to hastily click on the wrong button by mistake in a dialog box, sometimes with an

unpleasant and irreversible outcome. Some simple techniques can help reduce such errors:

- Label the options unambiguously, so the user understands the consequence of choosing each one.
- Follow standard color conventions to make it clear that certain options are riskier than others.
- When choosing the default action in a dialog, pick the one that poses the least risk of a harmful outcome.
- Make those default "proceed" buttons larger and easier to hit deliberately than options that could lead to a more damaging outcome.

Figure 6.9 illustrates a vivid way to make the options clear when an action has a potentially destructive outcome.

Constantine and Lockwood (1999) suggest numerous ways that designers can reduce user entry errors. For instance, if you provide selection lists of allowed values for a field, the software need not validate free-form input. Rather than complaining if the user enters a date format in an unexpected way, reformat it to the style the software wants. Don't force the user either to add or omit a leading zero at the beginning of a single-digit month number. Dealing with leading zeros was one of the first things I learned when I took a COBOL programming course decades ago.

And why should the program complain if I use hyphens in a DD-MM-YYYY date format instead of the slashes the program wants to see: DD/MM/YYYY? The software can readily parse those inputs and reformat them. The same is true with other structured numeric entries like

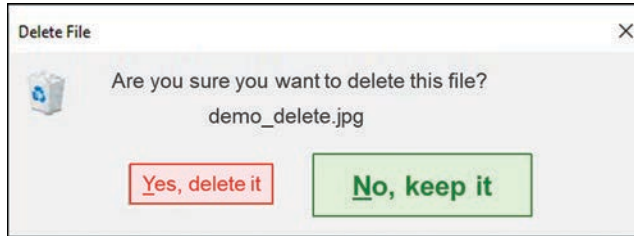

Figure 6.9 This file deletion confirmation dialog reduces the chance of making a mistake.

phone numbers, credit card numbers, and Social Security numbers. As well as preventing errors, a design that automatically reformats your input saves users time and minimizes the effort needed to complete a form. Friendly designs will format your input as you go along, adding spaces and punctuation as needed. They follow this advice:

DESIGN LESSON #31:
Provide input controls and processing to minimize the need to report errors.

The welcome screen for one website I use demonstrates a mix of thoughtful and thoughtless design (Figure 6.10). The form nicely blocks the entry of non-numeric characters in the fields for customer and invoice number, which helps to prevent user-entry errors and simplifies validation. Buttons to perform certain operations, such as to create an account or to make a one-time payment, are shown in green. But I don't understand why the log on button is in orange, a color often used to signify caution. Logging on is a perfectly reasonable thing to do at the screen, so what is the orange telling me? Some of the pages on this site use orange buttons to signify Confirm or Proceed, but other action buttons—even Cancel—are in green. The color coding used doesn't communicate significance clearly to the user.

Notice that the orange button in Figure 6.10 says "Log On," although the section heading says "Sign In." Consistency facilitates error-free communication. Also, I'm only going to create an account once, but I might sign in many times. I'd prefer to have the sign-in fields placed leftmost on

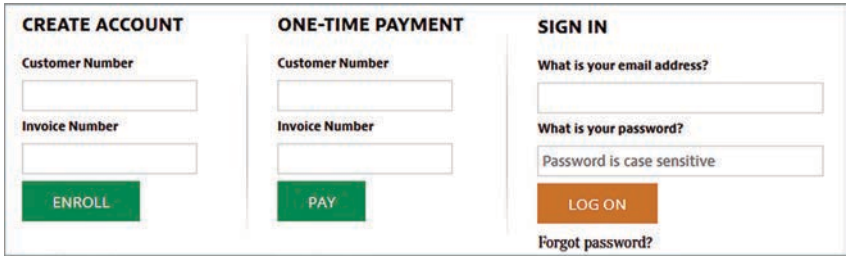

Figure 6.10 This website's welcome screen is a mix of thoughtful and thoughtless design.

the screen with the cursor starting in the email address field. Whenever you can, design to save the user keystrokes and navigation actions.

The password field cautions me that the "Password is case sensitive." That's good: it helps me to avoid the mistake of entering a password with the caps lock key accidentally engaged. Even better would be an indicator that my caps lock key is on, right next to the password field. The question "Forgot password?" is a link that leads to a password reset page. However, the only subtle clue that this text might be a link is that the font differs from all the other text on the screen.

People have been designing computer user interfaces for many decades now. It's unfortunate that we still encounter so many instances that don't apply the kinds of established best practices described by Alan Cooper et al. (2014), Steve Krug (2014), and others. Designers can do better with a little more education, a little more thought, and a little more customer engagement.

Thoughtful design applies to processes as well as to products, such as guidance to help you work with some piece of equipment. My friend John was pleased with the way an app on his phone helped him to install some Nest Protect combination smoke and carbon monoxide detectors, followed by a Nest Hello video doorbell. The installation process made it difficult to make a mistake. Here's John's experience with the Nest app:

> *Set-up was amazingly easy. First, the app asked a set of compatibility questions to understand how our current doorbell was wired. (You can access those prior to purchase to make sure you'll be okay.) Based on my answers, it walked me through a detailed, illustrated, step-by-step installation using my iPhone, down to the level of separate steps for disconnecting and then reconnecting each existing wire. The final step was to press the test button on the nearest Nest Protect. It and the Nest Hello then completed all the connection details without intervention from me.*

The combination of an effective process and well-designed documentation, packaged together in an easy-to-use app, made John's Nest setup easy and efficient.

The company that hosts my websites, Tiger Technologies, is the best technology company I've ever interacted with. Though not fancy, their website is functional, easy to navigate, and easy to use—things that are

a lot more important to me than cosmetics. Tiger Technologies heeded this important usability tip:

DESIGN LESSON #32:
Make it very hard for a user to take a destructive action by mistake.

Suppose I need to delete an email address (mailbox) from my Tiger Technologies account. As Figure 6.11 shows, the website cautions me about the implications of the deletion and asks if I'm sure I want to proceed. I must check a box confirming the deletion, which activates the button to delete the mailbox. Even then, when I return to the email management page, a message reminds me that I deleted the mailbox and tells me how I can restore the mailbox and its contents if I wish to (Figure 6.12). They

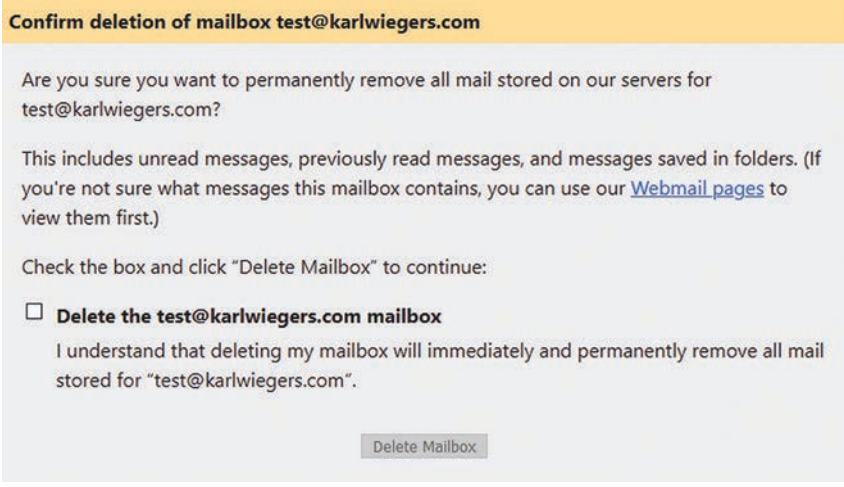

Figure 6.11 The dialog to confirm deletion of a mailbox at my hosting provider makes it hard for me to do something destructive by mistake.

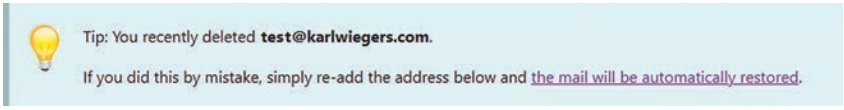

Figure 6.12 Even if I do delete a mailbox by mistake, I can easily restore it.

generously give me 30 days to realize that I made a mistake and recover the deleted mail. I very much appreciate the designer who helped to protect me from myself in this way.

As a counterexample of Design Lesson #32, my PC backup software permits me to restore a file from a backup even if the destination folder contains a newer file with the same name. It overwrites the newer file with no warning or protection, a truly thoughtless design that could trash a lot of work. A better design would detect the risk and then offer several options:

- Cancel the file restore operation.
- Restore the old version of the file to a different folder or with a different name, perhaps suggesting a new name that distinguishes the two files.
- Let me rename the current version of the file and then restore the old version with its original name.
- Proceed with the overwrite, checking again to make sure I really, REALLY want to do that.

Sometimes I want to look at an old version of a file I'm still working on. Because of this dangerous overwriting behavior, I must remember to first rename my current version to protect it. Then I can restore the old file from the backup, and finally, I must put everything back the way it was. The way this backup program handles file restoration unreasonably burdens the user to protect himself from damage the software can cause.

MAKE RECOVERY EASY

Users are going to make mistakes. They enter data formatted in some different way than the system expects, or they start down a path but then change their minds. Systems are going to experience failures also. Failures could be transient, such as a flaky network connection, or they could result from some deficient design approach that paints users into a corner. It's not possible to prevent every error, but considerate designers make it as easy as possible for the user to recover from one.

The Case of the Mysterious Error Message

My friend Scott recently was providing some feedback about a seller from whom he had bought a product on eBay. Scott wanted to compliment the

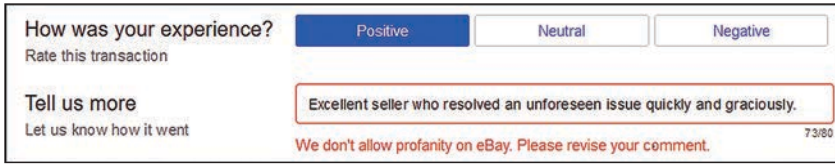

Figure 6.13 This error message from eBay's feedback form was puzzling. (Screenshot courtesy of Scott Meyers.)

seller for being particularly helpful in resolving an issue. When he tried to submit his comment, Scott saw a perplexing error message, shown in red in Figure 6.13.

Scott's message didn't contain any profanity, so eBay's error message baffled him. He discovered that eBay accepted his comment if he changed it to "Excellent seller **that** resolved an unforeseen issue quickly and graciously." Ah, there's a clue. I'll leave it to you to discern precisely what might have bothered eBay's input-checking software. Scott and I surmised that it strips blanks from the user-supplied text before checking for words on eBay's naughty list. That makes no sense. If you remove blanks between words before performing some lexical analysis, then what you have left is, well, not words. That was the best interpretation we could come up with, though eBay could be doing something different under the hood.

There are two instances of thoughtless design here. The first lies in the algorithm apparently being used to scan comments for eBay's notion of "profanity." That algorithm generated a false positive, rejecting a perfectly fine user comment. The second design shortcoming is to report an error without telling the user precisely what the problem is. This is the equivalent of making a rude buzzer noise during a game show to flag an incorrect answer. How can you correct an error if you don't know what's wrong? Had eBay highlighted what it interpreted as offending characters, Scott would have figured out the problem immediately. Instead, it took him some effort. It wasn't Scott's error at all.

The Case of the Forced Completion

I have occasionally used a website or software application that forced me to complete a task I began, even if I changed my mind partway through and didn't want to proceed. Maybe I discovered that I lacked a piece of

necessary information, or I realized I didn't need to perform that task after all. For example, perhaps while completing a survey online I'd like to change a previous response, but there's no way to do that. Sure, I can exit from the application, close the website, or use the browser's back button to return to a previous page and start all over. But that's not the same as using designed-in logical ways for me to back up, cancel, or undo what I've done. I wish those site designers had heeded this lesson:

Design Lesson #33:
Give the user options to exit from a task or to return to a previous step.

Even more considerate designs will remember data that the user had entered previously or allow users to save an incomplete task for possible future completion. Sometimes I have had to reenter information three or four times on a form because the input fields were blank when I returned to a previous page. Yes, it takes more design, programming, and testing effort for developers to provide those user conveniences, but users cumulatively waste a lot more time because of thoughtlessly designed applications.

A diagram called a *dialog map* can help ensure that you include appropriate reverse navigation options (Wiegers and Beatty, 2013). A dialog map depicts the architecture of a proposed user interface design. Figure 6.14 illustrates a dialog map for an online survey application. Each rectangle represents an individual dialog element, such as a screen, web page, menu, or dialog box—any place where the user can interact with the application. At this stage of design, we don't care at all what those screens might look like. The arrows show the allowed navigation pathways between one dialog element and another. Labels on the arrows indicate the combinations of user actions and system conditions that cause each navigation.

Of all the many visual analysis models I've used in my software career, the dialog map is my favorite. You can trace through a sequence of arrows in a dialog map to visualize how the user will interact with the system to accomplish the task and also to see how the system will handle various alternative scenarios. Figure 6.14 shows that the user can cancel out of the survey at any point and return to the welcome page. Choosing to restart the survey always takes the user back to Survey Page 1.

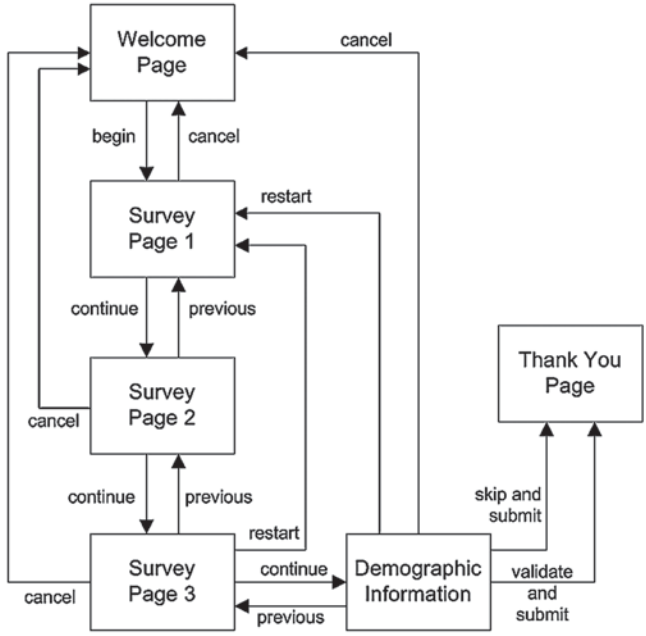

Figure 6.14 Drawing a dialog map for an online survey application ensures a user interface design that lets the user back out of an operation smoothly.

A dialog map also can reveal missing requirements, actions a user might want to take but cannot perform with the present design concept. I encountered a missing requirement while driving my Meals on Wheels route recently. One of my clients lives in an apartment building with a locked outer door. I must call her phone using an intercom panel so she can buzz me in through the door. Yesterday she was on the phone when the intercom called her, so my call rolled into her voicemail. However, the intercom's control panel did not allow me to hang up so I could try to reach her again in a couple of minutes. That seems like a significant missing function from the intercom system. Drawing a dialog map to think through all the issues involved with making an intercom call might have revealed that omission.

Another common functional absence is the inability to correct a mistakenly pressed elevator button. To summon the elevator, maybe I accidentally press the "down" button instead of "up," but I can't cancel that

action. Once I'm in the elevator, perhaps I inadvertently press a button for the wrong floor. If I'm alone in the elevator and the next passenger gets in and sees multiple buttons pressed, they'll know I made a silly mistake. Without an undo function for elevator buttons, people must take side trips to the wrong destinations. I wonder how many cumulative hours people waste each year on those unplanned excursions.

The Case of the Inactive Function

A system must be in a particular state to allow the user to perform certain actions. Does it make sense to let a user print an empty document? Should undo be active if the user hasn't done anything yet in a file? Graying out menu options that aren't currently available is a common user interface convention. The gray lets the user know the system offers certain functions, but it offers no clue about how to get the system into a state that will activate each function. The user-experience experts at the Nielsen Norman group suggest the designer include a balloon help message that appears if users hover over a grayed-out option, explaining why that function is disabled and how to make it active (Li, 2017).

I find it frustrating whenever I make some mistake that prevents the system from letting me use a function that I expect to have available. Forward-thinking design might anticipate some of these errors—you certainly cannot anticipate all of them—and help the user get back on track. Let me give you an example.

I like to record songs with my guitars—both original songs that I have written and covers of other people's songs. Figure 6.15 shows a portion of the screen that I see when I'm working with my digital audio workstation software. The software is amazing, but it can't quite read my mind yet. I wanted to increase the gain (volume) of one track at the beginning of the guitar solo on "Hotel California" by the Eagles. The green bars in Figure 6.15 show my recorded guitar tracks. I selected the section I wanted to adjust (circled at the bottom) and then tried to use the Audio/Process/Gain menu function. But as the figure shows, that function is grayed out, as are all the other audio process operations. I was stumped, since I've performed this same operation countless times before.

I had made a mistake. I had inadvertently selected the *volume* information about that guitar track, shown in gray immediately below the green audio track itself, which is labeled "Guitar 7 LP-07." The data shown for both elements—the music and its volume data—are nearly

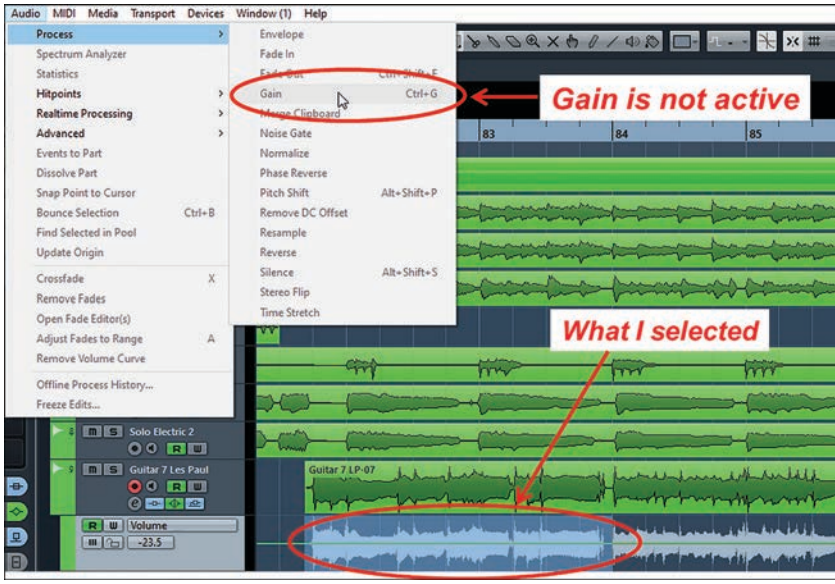

Figure 6.15 I thought I had selected part of a music track, but the audio processing options were all grayed out.

identical; only the colors differ. Figure 6.16 shows what I had intended to do. I meant to select a piece of the green audio track instead. When I selected the proper object, all the audio process functions became active.

This selection error is an easy mistake to make; I've done it more than once. It always takes me a few seconds to determine what's wrong. Wouldn't it be nice if the designer had anticipated that mistake and had the software tell me why I can't perform the audio process functions? Graying out the menu options tells me something's wrong, but it doesn't help me fix my error. Instead of simply showing the grayed-out options, feedback like "no audio selected" would give me a clue.

Providing users with easy error recovery requires that designers think carefully about what kinds of mistakes users might make and then imagine how to get them back on track. It's the difference between a feature-centric mindset (look at all the things you can do to your recording!) and a usage-centric mindset (I wonder what Karl's trying to do here?). Here's the main message to take away:

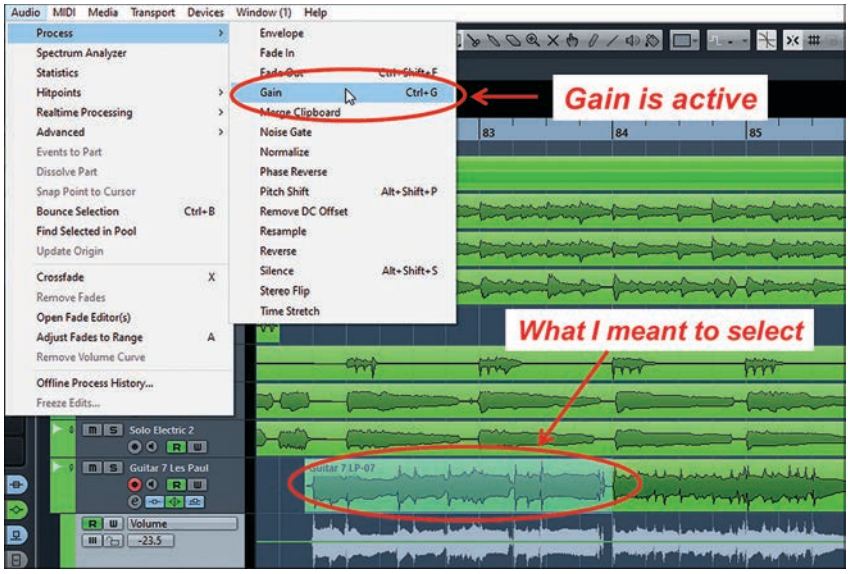

Figure 6.16 If I select what I really intended to, the audio process functions will all be active.

DESIGN LESSON #34:
Anticipate errors that users might commonly make, assess the user's intent when possible, and help users recover from the errors.

JUST LET IT HAPPEN

The least considerate design approach is to simply allow mistakes to happen and expect the user to handle any resulting problems. At best, this is annoying. At worst, it's dangerous.

Of Course, Cars Again

My brother, Bruce, who is a mechanical engineer, pointed out a design problem with some controls in his pickup truck:

> *The headlight switch is a 1.5-inch diameter rotary knob that you twist clockwise to turn on the lights [Figure 6.17]. Three inches to*

Figure 6.17 The knobs to control headlights (left) and four-wheel drive (right) are easily confused. (Photo courtesy of Bruce Wiegers.)

> *the right is an identical rotary knob for engaging the four-wheel drive. It is very easy to accidentally turn on four-wheel drive when you intend to turn on the lights. Very poor design.*

This control panel's symmetry causes user confusion. Bruce has driven many trucks; he's very familiar with the concepts of both headlights and four-wheel drive. A less error-prone design would make it easier for the driver to distinguish these two unrelated controls either by location or by touch. That could prevent the driver from making an inadvertent change, rather than forcing him to undo the mistake. I can imagine a driver who reaches for the headlight knob, only to be startled when he feels the truck shift into four-wheel drive. He quickly reaches for the four-wheel-drive knob to go back into two-wheel mode, but instead he turns off the headlights by accident. Driving should not be a slapstick routine. Here's a good guideline to keep in mind:

DESIGN LESSON #35:
Design device controls that the user can easily distinguish by both sight and touch.

Less risky but equally startling are my experiences with the smart key fob for one of my two cars. Its design causes misbehavior that I cannot easily prevent. Figure 6.18 illustrates the key fobs for both cars, a 2009 Nissan and a 2016 Honda. The Nissan's fob on the left has four buttons: from

Figure 6.18 It's too easy to press the alarm button on the Nissan key fob by accident.

top to bottom they are lock, unlock, open trunk, and alarm (circled). The buttons are all about the same size and equally easy to press. If I have my keys in my front pants pocket and I lean against something, I'll sometimes hear the car alarm go off because whatever I leaned against pressed the exposed alarm button. I've even triggered the alarm simply by bending over with my keys in my pocket. Other Nissan owners have reported the same problem in online discussion forums.

The Honda key fob on the right in Figure 6.18 contains buttons for the same four functions, as well as a remote engine start button ("Hold" with the circular arrow). This fob is better designed than the Nissan's. The two prominent buttons are to lock and unlock the doors, precisely the two functions I use most often. The alarm button (circled) is smaller, separated from the other buttons, and slightly recessed. I engage the alarm by pressing on that narrow, recessed button with my thumbnail for a second or two. I've never triggered the Honda's alarm button by accident.

One essential aspect of being a designer is continuous learning. You need to collect many experience reports from a variety of users in a wide

range of situations. You learn what works and what's problematic. You accumulate lessons that feed forward into every subsequent project, thereby improving each future product. I enjoy seeing new product versions become more usable and better designed over time.

But maybe not every designer or every company takes that approach. Figure 6.19 shows the key fob for a friend's 2019 Nissan. Other than the addition of the remote start function at the top, this fob design is the same as mine from ten years earlier, shown on the left in Figure 6.18. The most frequently used buttons, to lock and unlock the doors, are now in the middle, even though it's easiest for your fingers to locate the remote start and alarm buttons at the two ends of the button sequence. Maybe not enough car owners reported the alarm button issue to motivate a fob redesign.

Actually, the new Nissan fob might be even more problematic than my older one. With the old one, leaning against a kitchen counter when the keys are in my pocket could trigger the car alarm. With the new key fob, I wonder if the car would start if I accidentally pressed the remote

Figure 6.19 It's still too easy to accidentally press the alarm button on the newer Nissan key fob. (Photo courtesy of Scott Meyers.)

start button at the other end of the fob, the top button in Figure 6.19. I hope not.

In the Optometrist's Exam Room

Thoughtless design isn't limited to household products: it extends even to products used by the medical profession. Here's what my optometrist said about the remote control shown in Figure 6.20:

> *The remote control for my new visual acuity chart has buttons on* ***both sides****, which is ridiculously difficult to get used to. If I don't hold it carefully on the outer edges, I end up pushing front and back buttons at the same time and scrambling the programming.*

As Figure 6.20 shows, the front part of the control (left image) contains just a few buttons, but the back has a full alphabetic keyboard and other special-purpose buttons (right image). It's easy to see how the user could accidentally press random buttons on the back panel while using mainly the front panel during an eye examination. To make matters even more

Figure 6.20 An optometrist's remote control contains many buttons on both the front (left) and the back (right). (Photo courtesy of Nancy Buset.)

inconvenient, the on-off switch for the device this remote controls is on a completely separate remote.

Products that exasperate the intended user make you wonder whether the designer had any representative users evaluate the design before committing to it. Did an actual optometrist try out a prototype of the remote in Figure 6.20 in a realistic setting? We don't know. It brings us back to the all-important Design Lesson #3 from Chapter 2: "Involve real users." No matter how good a product seems in the design lab, something's wrong if people who must use it daily find it problematic.

People are going to make mistakes. Good product designers will strive to minimize the consequences of those mistakes. Their users will be grateful for the consideration.