Chapter 7. User Interface Design

> Good design is also an act of communication between the designer and the user, except that all the communication has to come about by the appearance of the device itself. The device must explain itself.
>
> Donald A. Norman, author of *The Design of Everyday Things*

In far too many instances, user interface (UI) design often boiled down to putting input fields wherever they would fit, irrespective of all else. While that may have been the most expeditious approach for the developers, it rarely resulted in the most usable application. Remember, the UI is the *what* of user experience (UX) and to nearly all users, the UI *is* the application. Good UI design is based on accomplishing the goals set forth by UX.

UI design is more than just a polished look and feel; it's about making your application easy to use for your intended audience. Ease of use encompasses a number of topics, including accessibility and inclusivity of UI design. Of course, few software engineers are ever taught the basics of UI design. Although this chapter won't make you an expert, you'll be a step ahead of most by the time you're done reading!

Designing for Everyone

Accessibility, usability, and inclusion are distinct yet interconnected concepts in design that aim to make products usable by people of different backgrounds and abilities. You'll often see them referred to by their numeronyms: a11y for accessibility, L10N for localization, and I18N for internationalization.

Note

A *numeronym* is a word that is partially or wholly composed of numerals. Although it can describe many constructs, you'll often see it as a contraction with all of the letters between the first and last replaced by the number of letters that were omitted. For those who like regular expressions, numeronyms of this type would follow this pattern: `[a-zA-Z]\d{1,2}[a-zA-Z]`.

Challenges can arise in balancing these aspects, ensuring that they are integrated throughout the design process, and addressing different user needs effectively. Each of these can make or break your application and in some instances are subject to laws and regulations such as the Americans with Disabilities Act (ADA), which requires accessible design; Section 508 of the Rehabilitation Act, which stipulates US federal agencies must make information technology accessible; and the European Accessibility Act (EAA), which mandates products be designed to maximize usability by those with disabilities, among other things.

The Retrofit Challenge

Nate here. Years ago, I worked on the UI team at a software company. We had a mature web application used extensively by professional engineers. Our product managers decided accessibility would be a focus for an upcoming release and set our team to retrofit the application to be more hospitable to screen readers and direct keyboard interaction. To say the app was built without these concepts in mind would be an understatement; the first time we walked through the page with a screen reader, the enormity of the effort was clear.

Fields lacked clear labels, and there were almost no keyboard shortcuts to speak of. Moving around the application without a mouse was laborious. I can't speak to decisions that were made well before my tenure, but suffice it to say, it is far simpler to build with accessibility, localization, and internationalization in mind from the start. I will say that the app did an excellent job of supporting multiple languages, though! And yes, with a massive amount of effort, we managed to (mostly) make the app usable from the keyboard, and at least it wasn't overly hostile to screen readers.

As we go through usability, accessibility, and localization in the coming sections, keep in mind that unless you are building software for other software engineers, you may not be representative of your user base. Consider users with different abilities; access to and quality of hardware, software, and internet connectivity; and geographic locations. Your vision may be comparable to an eagle's, but that isn't a universal ability.

What Is Usability?

Usability is the software equivalent of flossing your teeth: everyone knows it's important, but it's usually one of the first things to get cut from a project.[1] You may be lucky enough to have UI experts at your disposal, but odds are you'll shoulder some (or all) of the work. Despite what some people think, usability matters, at least if you'd rather not cause users to run screaming from your application. While most software engineers are well schooled in the practices of algorithms and compiler theory, they often lack even a basic grounding in the art and science of UI design. Add in the long-standing myth that usability is best left to those with PhDs and graphic designers, and you've got an "ility" that has been ignored for too long.

Despite what many software engineers have been led to think, building usable applications is not an impossible task. Over the next few pages, you'll explore the basics of what it takes to make an application usable and dispel the myths that surround this misunderstood ility. So what is usability?

Usability boils down to one simple question: how easy is your application to use for its intended audience?[2] Of course, that's a fairly broad concept and can be interpreted in different ways. Usability is often defined by various quality components:

Learnability
: How easy is it to learn your application's interface? Can a new user pick it up in minutes, or does it take years to master? Can your application be used by both novices and experts? Can "training wheels" be added or removed as necessary? In other words, can the application grow with your users? Does the application include adequate contextual help and documentation that is easy to find and access?

Efficiency
: How efficient are users of your application? Are clicks minimized, or does even the simplest task require several screens?

Memorability
: How memorable is your application's interface? Is your application like riding a bike and easy to pick up after some time away, or do users feel like they're learning to walk all over again every time they start it up?

Discoverability
: How easy is it to fathom the features of your system? Is the right thing always obvious to your users, or are even the core functions hiding in obscure places?

Error handling
: How does your application handle and recover from errors? Does it gracefully handle any omissions by a user, or does a mistake cause a blue screen of death? Does your application protect your users from data loss when error conditions occur? Does your application *prevent* errors through hints, validations, field masks, and visual cues?

User satisfaction
: How happy are your users after working with your application? Are they sitting on cloud nine or cursing those responsible for the hideous mess they're forced to work with day in and day out?

Accessibility
: Is the interface accessible? Does the interface enable assistive technologies?

As you can see, multiple factors ultimately affect the overall usability of your application. It may seem overwhelming, but with the right approach and thoughtful design, you can ensure that your application will be one that delights users instead of being overtly hostile to their needs.

What Is Accessibility?

While usability is about the ease of use and satisfaction of all users, *accessibility* is about making your software usable for people with different abilities. Users of all abilities should be able to navigate and interact with your interface. Depending on the laws or regulations that apply to your situation, you may need to consider specific aspects, but in general, you should be mindful of the following challenges:

Visual challenges
: Low vision to blindness to color blindness[3]

Auditory challenges
: Deaf or hard of hearing

Dexterity challenges
: Repetitive strain injuries or motor impairments that may make precise mouse movements difficult

Cognitive challenges
: Conditions such as autism or dyslexia or poor reading or communication skills

Making software accessible doesn't just help those with disabilities; it helps *all* your users. Though vital for those in wheelchairs, a curb cut or ramp also helps the traveler dragging their rollerboard through an airport. An interface with larger tap or pointer targets doesn't just help those using alternative input devices, and keyboard shortcuts benefit power users as much as those who can't use a mouse or trackpad. Software that doesn't rely on a high-bandwidth internet connection helps those without access to broadband as well as those on a mobile device as they roam in and out of coverage areas.

When you are building a UI, think in terms of inclusivity, ensuring that your software can be used by as many people as possible. Many modern platforms and operating systems support APIs and features you can leverage to make your application more accessible. Apple operating systems include multiple assistive technologies, from color filters to live captions to eye tracking support. Microsoft includes many of the same in its products as well as a collection of customizable adaptive buttons and mice.

What Are Localization and Internationalization?

Though often used interchangeably, localization (L10N) and internationalization (I18N) are distinct but related concepts. *I18N* is the process of *designing and building* your application to be adaptable to different languages and cultural expectations. For example, instead of hardcoding labels, you build your UI such that the label text is loaded from a properties file. It's also important to design for label expansion; some translations can add a significant amount of text for the same words or phrases.[4] Not all cultures are left-to-right (LTR) oriented; several languages such as Arabic and Hebrew are read right-to-left (RTL), while Japanese can also be oriented top to bottom!

L10N, on the other hand, is the *process of adapting* your interface to the cultural and linguistics of your target markets. In other words, L10N is translating labels and fields to match the local language and using the proper currency symbols as well as expected date formats and units of measurement. Internationalization is the foundation that allows you to adapt to different languages and cultures, while localization implements those necessary customizations.

Unless you can guarantee that your application will never be used outside of a specific locale, you should design with I18N and L10N in mind from the start of your project. If there is even the possibility your application will expand beyond a single market, build with a global audience in mind. Your future self will thank you!

Know Your User

You cannot build a usable application without an understanding of your audience and the environment within which they will utilize your interface. It can be dangerous to overly generalize about your users (see the following sidebar "Don't Assume"); be sure to challenge assumptions. While you certainly could spend months in the field analyzing users, asking some basic questions about how they'll use your application can go a long way toward building a usable application. For instance:

- Are your users experienced with computers? What operating systems?
- Do any of your users have a functional limitation? How does your application respond to screen readers? What happens if a user increases the font or changes the resolution on the screen?
- Will your users access your application with a phone? A tablet? A laptop? A headset? A television? All of the above?
- What is the expected education level of the user population?
- Is the user group fairly stable, or is there significant turnover?

- How frequently is the application used?
- Where will the application be used? An office? At someone’s dining room table? On the couch? In the field?
- What is the environment like? Noisy? Quiet? Stressful?
- Will your users receive any training on your application?

Don’t Assume

It can be tempting to make assumptions about your users as a shortcut to performing user research, but doing so can lead to unforced errors. Years ago, a retirement community reached out to a local company to see whether it would donate its older computers. The office manager laughingly agreed, assuming that older people weren’t tech savvy enough to utilize them and felt they’d just gather dust. When the retirement community reached out again a few months later, the office manager thought they’d want to return the devices. In fact, they wanted more keyboards as the residents were wearing them out. It turns out residents loved staying in touch with family and friends online. Take the time it takes to learn about your audience.

Asking basic questions about your intended users is invaluable. The responses will tell you how to make your app more usable and valuable to them. If your application is used in a loud environment, users will probably miss notification sounds. Plan to have alternative notifications. While some designs can assume a lengthy training period, odds are users won’t, in fact, read the manual. In that case, ensure that the easy thing to do is the right thing to do—for example, don’t let a user enter characters when only numbers are permitted. Applications designed for a ruggedized tablet used by hydrologists measuring groundwater in the field have different constraints than those targeting office workers with large monitors and trackpads.

A more usable and valuable app will lead to better user satisfaction. Many developers scoff at the idea of user satisfaction; some go so far as to say, “My users don’t have a choice.” Some of you may develop for internal customers who are required to use your application, but having a captive audience isn’t a license to poke people in the eye with a stick. Your users deserve better! This factor is more obvious when you’re competing on the open market with healthy competitors—customers will often navigate to the more usable solution. Whichever situation you find yourself in, don’t underestimate the importance of happy users.

If at all possible, shadow or observe a representative set of your customers while they use your application; you’ll be amazed at what you learn. Again, this kind of research does not require months in the field. Spending a few days understanding actual users in the space the application will be used can mean the difference between an app that delights and one that disgusts.

Secondary Users

While your focus should be on the primary users of your software, don’t forget about the *secondary* users! In many cases, the people using your application are themselves working on behalf of another person, the secondary user. Anytime you’ve interacted with a customer service representative or made an appointment to get your car serviced, you’ve been a secondary user.

Think about the last time you were a secondary user: did the customer service representative say some variant of “Give me a minute, I’m looking up your order, my system is really slow to respond.” Did you enjoy that experience? Nate once watched a scheduler take several minutes to book an appointment; she apologized, saying the new software takes “11 clicks, we counted,” and they already had all of his information on file!

Real costs are associated with poor usability. Not only does it directly impact the end users and their work, but it also bleeds into your reputation in the marketplace.

You Are Not Your User

It should go without saying, but (unless you’re designing an application for other developers) you are not representative of your actual users. Just because something is obvious to you doesn’t mean it is to someone with a different background. Your comfort level typing cryptic commands into a terminal may not translate to a broader audience. And just because something looks great on a massive curved-screen monitor doesn’t mean it will work on a smaller screen.

Users are also likely to do some things that you won’t. For example, a few years back, a very reputable polling organization was approached by a political campaign because its candidate wasn’t one of the options given to several campaign volunteers who had been randomly contacted. The polling company assured the campaign that its candidate was included in the survey and promised to get to the bottom of the issue. When the polling company actually went to one of its call centers, it quickly discovered the problem: some of the call center staff had increased the font to make the questions more readable.[5]

While not a surprising action from an end user, it turns out no one had tested the consequences of doing so. To remove bias, candidates’ names were displayed in a changing random order from poll to poll; however, with the font increased, some names were no longer visible to the call center workers! In the end, the poll had to be thrown out, and the polling organization’s reputation suffered.

The Tyranny of Defaults

Never underestimate the power of defaults. Quick check, on your phone, what application do you use for mail? For browsing the web? For turn-by-turn directions? Most likely, the defaults of your mobile operating system. Being the default can be worth a lot of money: it is estimated Google pays Apple several billion dollars to be the default search engine across its devices.

Odds are your defaults won’t be chosen by a multibillion-dollar arrangement, but that doesn’t mean you shouldn’t actively consider what they should be. One of Nate’s students was demoing a feature to a customer, and the very first thing the customer did was change everything the app had defaulted. The student was surprised and asked about it, to which the customer replied, “We always change those.” His student went back to his desk, updated the default configuration, and then returned to his customer who was thrilled with the update. The lesson? Don’t assume you know what the proper defaults are; don’t be shy to ask your customers. Better yet, test it!

While you might have ample usage data for an existing application, you will need more insights when you’re building something new. In other words, it is not always possible to know a priori what the right thing to do actually is. In these

instances, embedding telemetry features into the user interface can help gain a fuller understanding of how the application is used, allowing you to react to data instead of hunches.[6] You can then evolve the interface accordingly.

Also, if your users change the settings, your application should remember them moving forward. How many times have you changed a setting to your preference only to have an application decide it knew better the next time you launched it?

Lastly, defaults should be "safe," especially when users are stressed or distracted. If an action is destructive and cannot be undone, adding friction to the process is actually a good thing. The default shouldn't be "delete account," especially on a dialog that users have been effectively trained to click through automatically.

Impact of Culture

It is also important to understand the impact of culture. Users' mental models will differ, something any tourist in London can see firsthand while crossing the street (see Figure 7-1). There are obvious differences; for example, languages that are read right to left will require a different UI layout than those read left to right. Things that may be normal to *you* may in fact be offensive to someone living on a different continent. Color can even have different connotations as well; for example, red represents luck and good fortune in many Eastern cultures, but in South Africa it is the color of mourning.

Figure 7-1. London crosswalks remind visitors that vehicles approach from the right

Cultural differences can even extend to the visual design of icons. Some cultures may prefer more concrete icons, while others may prefer more abstract representations. And that's before you consider historical symbols like a diskette to represent Save, considering an ever-growing share of people have never used a computer with a disk drive![7] Localization could be an entire chapter, and while it may be relatively straightforward to translate your interface elements from one

language to another, don’t be surprised if interface elements need to be resized or relocated afterward.

Taking the time to investigate who your users are and how they’ll use your software will help you avoid choices that can seem overly hostile to them. It may seem like a waste of precious time, but basic user research pays for itself again and again. You never get a second chance to make a first impression: never underestimate the goodwill you can earn by simply listening to those on the other side of your code.

Maximizing Usability

At this point, you might be tempted to ask which aspects of usability (learnability, efficiency, memorability, discoverability, error handling, user satisfaction, and accessibility) you should focus on. Before you run off and try to maximize each and every one of these qualities, you have to think about the most important question: how will your application be used? Some applications are used all day, every day (making efficiency, memorability, and user satisfaction important) or are inherently complex (think CAD software); in cases such as these, users will typically get a side of training with their application or they’ll learn the interface through sheer force of repetition.[8] If your application is an attempt to be the next social networking unicorn, you can’t expect users to tolerate the need to learn much of anything (meaning learnability and discoverability are key); the app better be intuitive, or they’ll leave your site for one that is. In other words, you have to consider the purpose and context of your application in relation to your user population.

Efficiency follows a similar rule. If your users will spend most of their day in your application, then saving even a couple of clicks can add up in a hurry. Shaving even a few seconds off an interaction adds up when it is performed thousands of times a day. Never forget, a small number times a big number is a big number!

In some cases, your users are judged on how many “tasks” they finish in an hour or a day, so making their workflow as streamlined as possible could have a significant impact on the bottom line. By the same token, a rarely used application can suffer a bit on the efficiency scale, especially if it improves another aspect of usability such as memorability or learnability.

The importance of memorability, learnability, and discoverability are closely related. Ideally, an application is both memorable and learnable, but again, context matters. For instance, an application that is used every day will probably be memorable (hopefully, for positive reasons). In these cases, you can include some interactions that might not be as easy to learn or as discoverable. If your application is used sporadically, you can’t expect your users to invest hours of time learning your approach; in cases like these, learnability and discoverability are vital but memorability isn’t. Puzzle games are deliberately designed so that discoverability is low; figuring things out is the whole point. Again, context is your guide.

Principles of Design

It takes time to develop design skills; however, by following well-worn principles, even the novice can create a compelling design. In her highly regarded book *The Non-Designer’s Design Book* (Peachpit Press), Robin Williams introduces the

apprentice designer to the importance of contrast, repetition, alignment, and proximity.

Though primarily aimed at creating attractive and effective documents, newsletters, and business cards, the concepts apply equally well to application interface design. The concepts are simple and easy to apply, but ignoring them has serious consequences for the usability of your application.

As you advance in your career, you'll discover the power in bending or breaking these rules; however, you have to consider what you're designing. While a jagged alignment might make for a memorable advertising campaign, it may not work out as well for a corporate time-tracking application.

Contrast

Contrast is one of the most effective tools you have at your disposal. You typically think of contrast as a difference between two colors, but any two things that are different gives you contrast. Pitting a large font against a small font, a thick line versus a thin line, or a small image counter a large one all create visual contrast. Different colors, shapes, and even different alignments can help make your page pop.

Effective use of contrast provides visual interest to your design, but to be effective, use things that are *really* different. Be bold, or as Williams says, "Don't be a wimp." Contrasting two shades of gray isn't nearly as effective as using two distinct colors such as red and black. Don't be afraid to really push the envelope; it's better to go a bit overboard than to have elements that are too similar.

Contrast also acts as an organizational tool. Glancing at a page, a user should quickly grasp the flow of the design; applied poorly, contrast can confuse the user, creating visual groups where none are meant to exist. Headings, subheadings, and body text should be distinct and vary enough that users can clearly tell them apart.

Repetition

Repeating key visual elements across a design provides a sense of familiarity and cohesiveness to an application. Though most often seen in a persistent navigational element or header image, the effect can be subtler. Using the same font, color scheme, screen layout, a type of bullet, italics—anything that helps tie the design together can be repeated.

You may think of this as just being consistent, but visually it is key to creating good designs. At the most basic level, repetition lets users know they're still on the same site; imagine how disconcerting it would be to use an application whose visual design constantly shifts.

Once you've made sure all of your headings are the same font and weight, start thinking about other items you can intentionally repeat; a persistent line or bullet point can serve as that little extra element that takes a design to the next level. On one application Nate worked on, the bottom of the page had a single thin line separating the main body content from the boilerplate privacy/copyright section. This footer was repeated, but to further emphasize the repetition, we added a double thin line. It may not seem like an important distinction, but it made the section stand apart from the rest of the application.

Like contrast, repetition creates visual units that help your users work with your application. Users will quickly achieve a comfort level with a layout and learn

what they can safely “ignore” as they go about their daily work. The repeated shapes help the user quickly parse the page—and make differences stand out.

It is easy to go overboard with a repeated element, especially color. Red might be highly identified with your brand, but that doesn’t give you license to inundate your application with it. There’s a fine line between a unifying element and gaudy excess: proceed with caution.

Alignment

Alignment is one of the easiest principles to put into practice. Place every element with care; don’t just drop an element somewhere because there happens to be some space on the page. Aligning elements creates a sharp, cohesive look and ties visual elements together. Using alignment, items that aren’t located near one another still have a visual connection.

Aligning left or right creates sharp vertical edges; sharp edges lead to a polished, professional look. Avoid centering as this leads to jagged edges that aren’t as pleasing to the eye and also results in harder-to-read text. Alignment isn’t just for text, though. Images or icons should be aligned with other visual elements on the page; whenever you place an element, find something to align it with.

Pick an alignment approach and repeat it throughout your design. Using the same alignment throughout a design is reassuring to your users and makes the organization more evident.

Proximity

Proximity plays a vital role in how people assess what they see. Items that are grouped together are perceived to be related even if they aren’t. Take this list:

- Shorts
- Sunscreen
- Socks
- Sunglasses
- Sandals
- Shirts

Since the items are grouped together, you instinctively see them as one visual unit. How these items are related isn’t relevant, and even if you didn’t suspect someone was preparing for a trip—and had a thing for stuff that starts with *s*—you sense some similarities in the items. Let’s tweak the list just a bit:

- Shorts
- Sunscreen
- Socks
- Sunglasses
    - Sandals

- Shirts

At a glance, sandals and shirts appear to have a special association within the overall list. It isn't evident what that relationship is, and some users will (consciously or not) try to determine just what that connection is.

As another example, Apple's Fitness app keeps tabs of the various awards you earn. At one point, all the various badges were listed together, making it rather challenging to navigate. An update changed the interface to group like awards together under category cards (see Figure 7-2). The main Awards screen shows you the status of your latest challenges, while tapping into the category will show the other badges you were awarded for that category.

3:18

Summary

Awards

Workouts

7-Workout Week
Saturday
397

+22 more
Show All

Monthly Challenges

February Challenge
2/17/24

+57 more
Show All

Limited Edition Challenges

Heart Month Challenge
2/14/24

+43 more
Show All

Fitness+ Workouts

25 Fitness+ Mindful Cooldown Workouts
11/28/23

+7 more
Show All

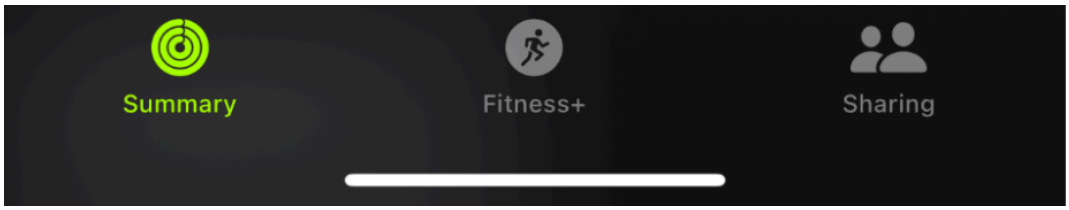

Figure 7-2. Proximity in action on the Awards section of Apple's Fitness application

Conversely, items that aren't related shouldn't be grouped together; if you don't want to confuse your users, keep unlike things apart.

As you create these visual units, be sure to leave some whitespace between them. Separating groups of items with a buffer further enhances the bond between the items and helps your users understand the page. Proximity creates visual groups that your eye will naturally follow. To avoid confusion, minimize the number of visual groups on the page. If you have too many (more than five or so), try to alter the placement to form new groups.

Applying the Principles of Design

Let's analyze these principles further by looking at the O'Reilly learning platform. Throughout its design, you see the principles of design at work, resulting in an easy-to-use site that is pleasing to the eye.

Take a look at Figure 7-3: we see contrast in action. The large cover art stands out, and the contrasting color draws the eye while providing a nod to the brand identity. Note too the contrasting color indicating the format type of a given resource. The current topic area is prominently identified by utilizing a larger font than anything else on the page.

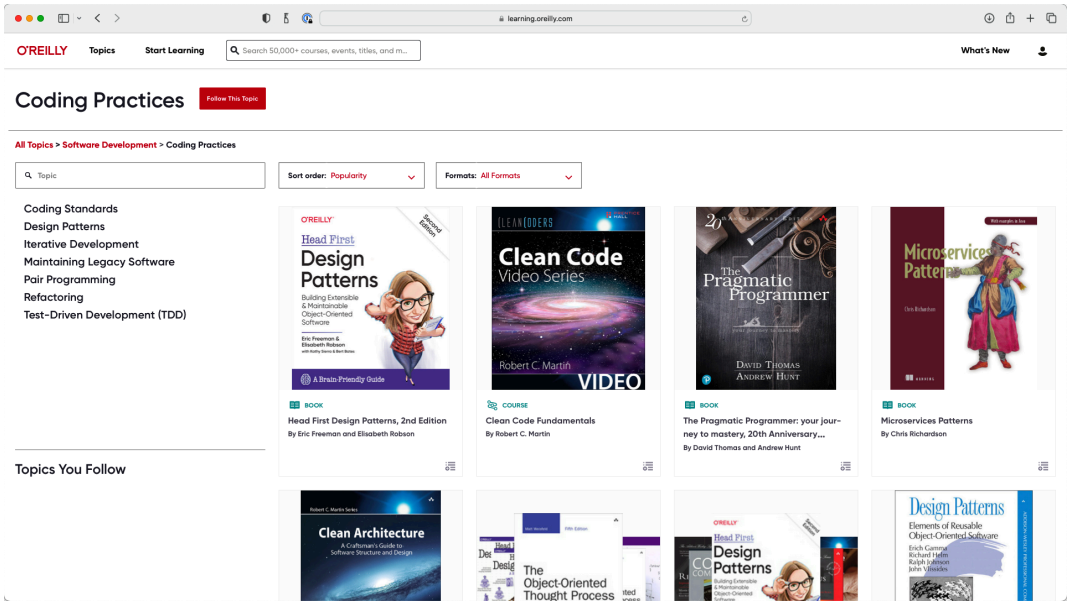

Figure 7-3. Design principles illustrated by the O'Reilly learning platform

In Figure 7-4, you can see some of the various blocks that make up the result page. The Search box clearly links to the topic area, while the two drop-downs will further refine the results area. Note the contrasting color of the Topics You Follow button, which clearly stands out on the page, inviting your gaze.

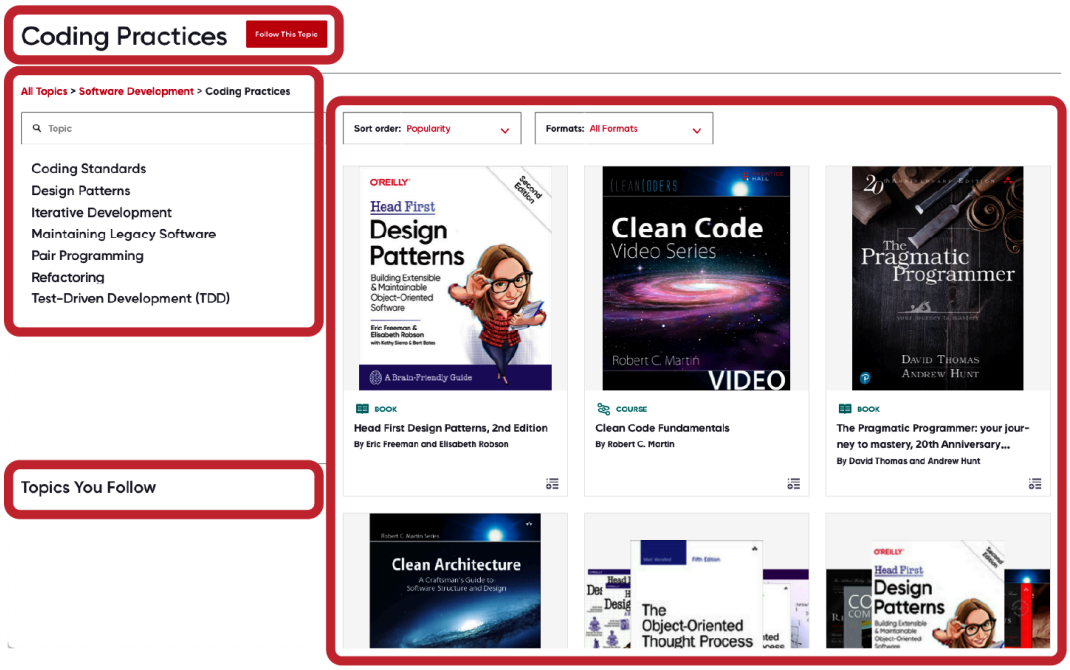

Figure 7-4. Notice the various blocks of related material

These building blocks continue through to the details page of a specific piece of content (see Figure 7-5). Notice again the familiar contrasting color—just enough to draw the eye, not too much to overwhelm. The same font family provides reassurance you're still on the same site, and the main menu at the top of the page provides consistency. Again, you have one button set in a constrasting color, inviting you to start this material.

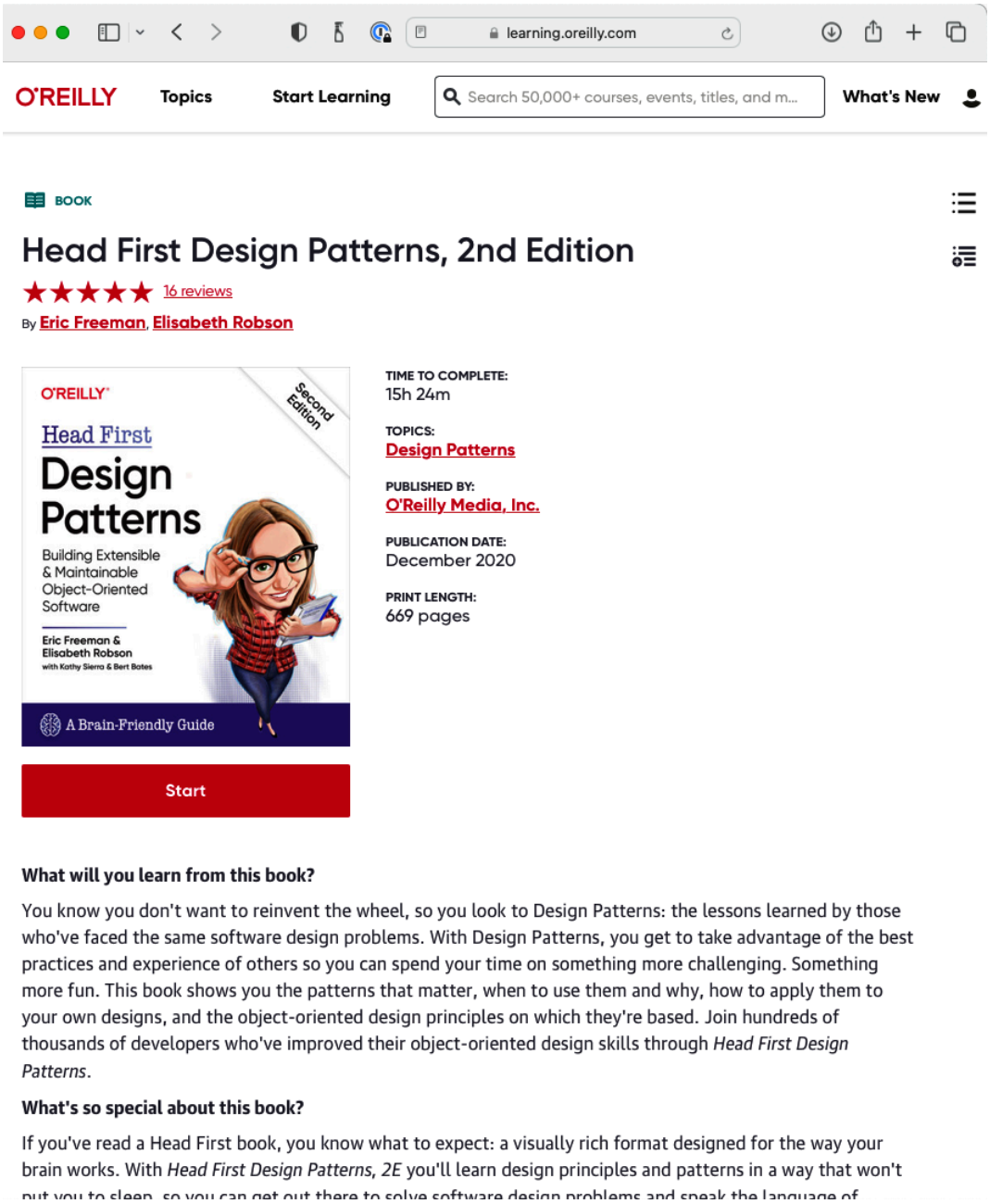

Figure 7-5. Repetition of design elements

These principles aren't the sole purview of web applications, though. Take the sidebars from Apple's Music, Books, and App Store applications in Figure 7-6. While all three are clearly customized for their given domain, each contains a search bar, consistent fonts, and related icons. A subtle color change further reinforces the specific application. Had they used the exact same colors across apps, users might have been confused about *which* application they were actually in!

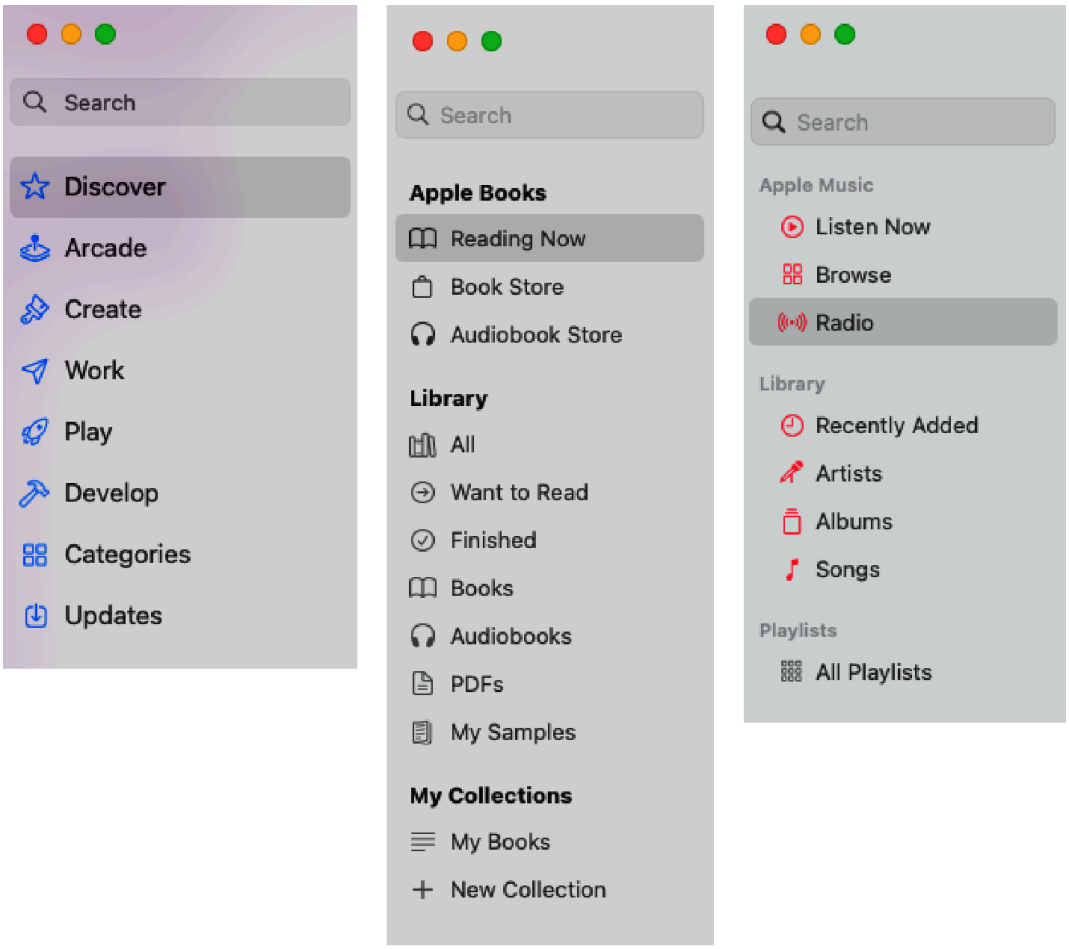

Figure 7-6. Apple sidebar similarities

Make the Right Thing the Obvious Thing

Discoverability is key to learning an application. If every feature is hiding where people aren't expecting, learning the system will take them much longer.[9] While a less than obvious approach *might* work if it's memorable or the system is used daily, it's best to stick with obvious approaches unless your interaction is radically better than the alternatives.

For example, let's look at some software one of your authors suffered through nearly daily many moons ago. Given the dialog in Figure 7-7, how would you record the audio? Go ahead, ponder away.

Figure 7-7. An example of a less than discoverable feature: how do you record the audio?

First-time users of this software usually couldn't figure out how to record the audio without either the help of a frequent user or the detailed instruction manual. This dialog is a perfect example of bad discoverability. Oh, the answer lies (again obscurely) in Figure 7-8. Give up? You have to click Test Audio (after setting up the conference number) and don't forget to check "Remain connected to meeting audio after testing." See, piece of cake.

Figure 7-8. This is where you record audio: can you figure out how?

Now, you could argue that this is memorable in the sense that it's so bad, most people will vaguely recall that there's some "odd" thing you need to do to make it work. However, what would happen if you didn't record a session for a few months? Would you still remember the trick? Don't count on it. Research indicates that discoverability contributes to memorability; when in doubt, go with the more obvious approach.

How would you fix this example? First and foremost, why do you even need to "test" the audio? The audio level meter could simply be part of the initial dialog, negating the need for this challenging second dialog entirely. If that approach isn't feasible, then selecting "Include audio with meeting recording" should be sufficient to, well, include audio. A user shouldn't have to select "Remain connected to meeting audio after testing" at all: that should be removed.

Wherever possible, you should also prevent the user from doing the wrong thing. The simple act of choosing the right field type on a form can eliminate a host of errors from ever occurring. Use hints, field masks, visual cues, and formatted examples to make the right choice obvious to your users.

Does Every Feature Need to Be Discoverable?

In some instances, a perfect interaction isn't discoverable. Some applications have so many features, they can't all take center stage: some have to move to secondary areas. With more and more software taking flight as mobile first or mobile only, you may have even less space to work with than during the desktop era of software.

Take, for example, the iPhone's pinch-to-zoom feature. Starting from scratch, a new user might not even realize you could zoom on a web page or a picture. Looking at the interface, it certainly isn't obvious that pinching will cause anything to happen; however, it takes only two seconds to teach someone that pinch equals zoom. This approach is an eminently learnable, very memorable feature, and once discovered, it is obvious and seems perfectly natural. It's so natural, you've actually broken your users' mental model if your interface doesn't support it!

Pinch to zoom is a special case, though. The interaction is really well designed and clear: the benefits of this "hidden" feature greatly outweigh the possible downsides. In this case, Apple made zooming a central part of its initial ad campaign, so it's likely that most users are aware of the feature. Apple also provided a variety of instructional videos on its iPhone website. And, the interface was usable even without pinching to zoom. When the situation dictates it, discoverability can suffer, but the benefits of the interaction should be readily apparent before you hide a key feature.

Sometimes interfaces are very discoverable but not all that memorable. Keyboard shortcuts are a must for user efficiency and provide much needed support for expert users, but they aren't always memorable. Used often enough, shortcuts become ingrained: most users can rattle off Save, Copy, and Paste without much effort. But when it comes to more infrequent combinations, it pays for them to be discoverable. Many IDEs do a fantastic job of doing exactly this. For example, IntelliJ includes a universal Find Action option (see Figure 7-9) that allows you to type in the name of an action to find something you otherwise couldn't recall.[10]

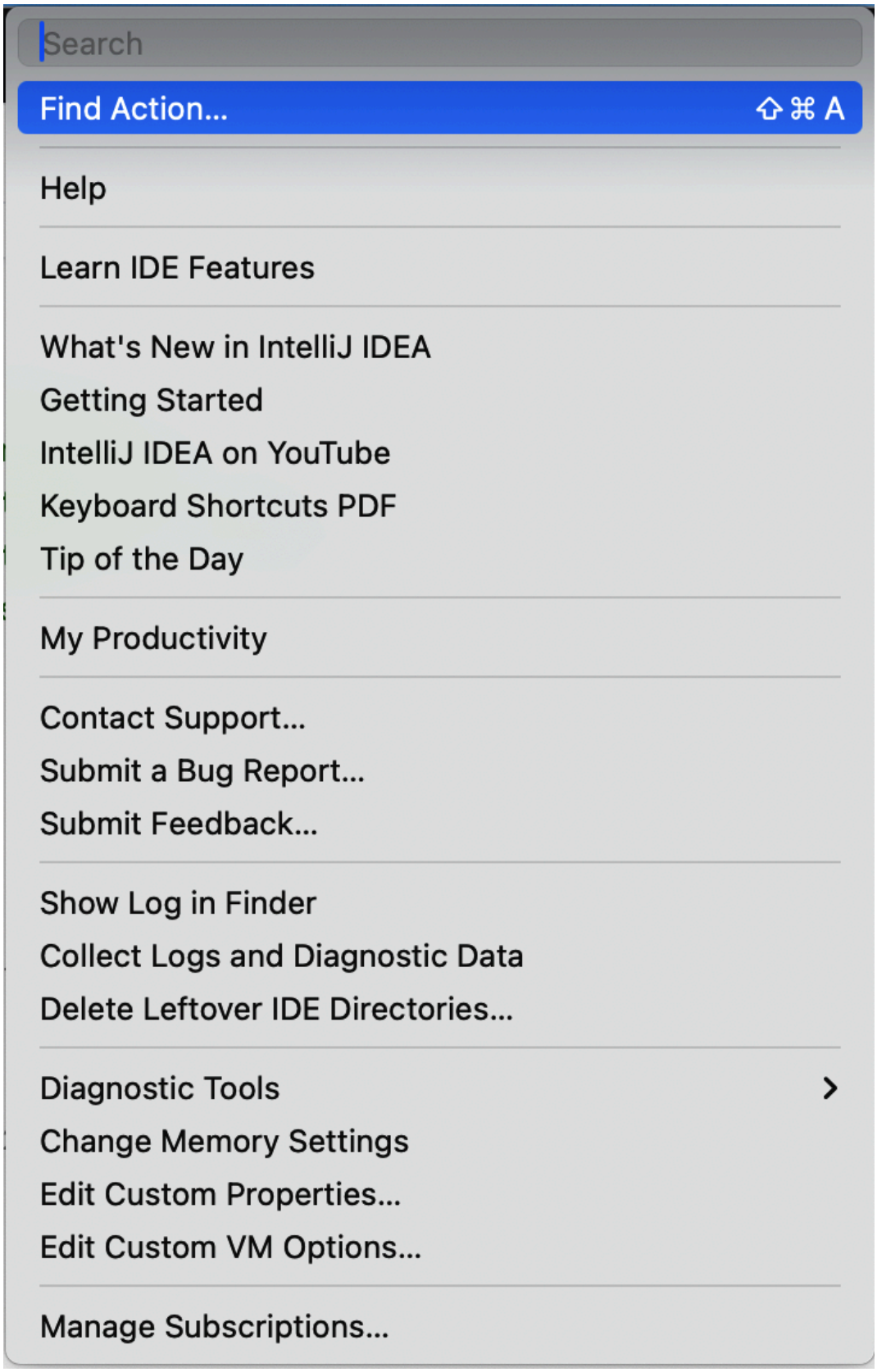

Figure 7-9. IntelliJ IDEA's Find Action option

It's also wise to reinforce the shortcuts by including them in the menu hierarchy; this way, as people use the menus, they'll pick up on the shortcuts. For example, modern development tools offer quick ways to navigate from file to file, but they may not be the same from one editor to another. For example, see VS Code's Go menu in Figure 7-10 and IntelliJ IDEA's Navigate menu in Figure 7-11.

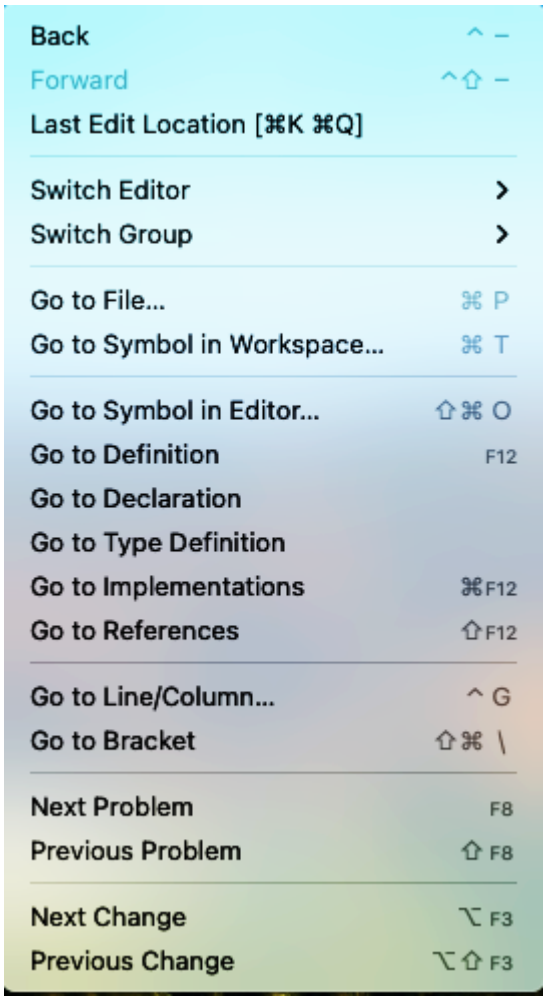

Figure 7-10. VS Code's Go menu

Some operating systems and tools support powerful search functionality enabling you to quickly find and learn menu items. For example, Figure 7-12 shows searching for "file" in IntelliJ IDEA.

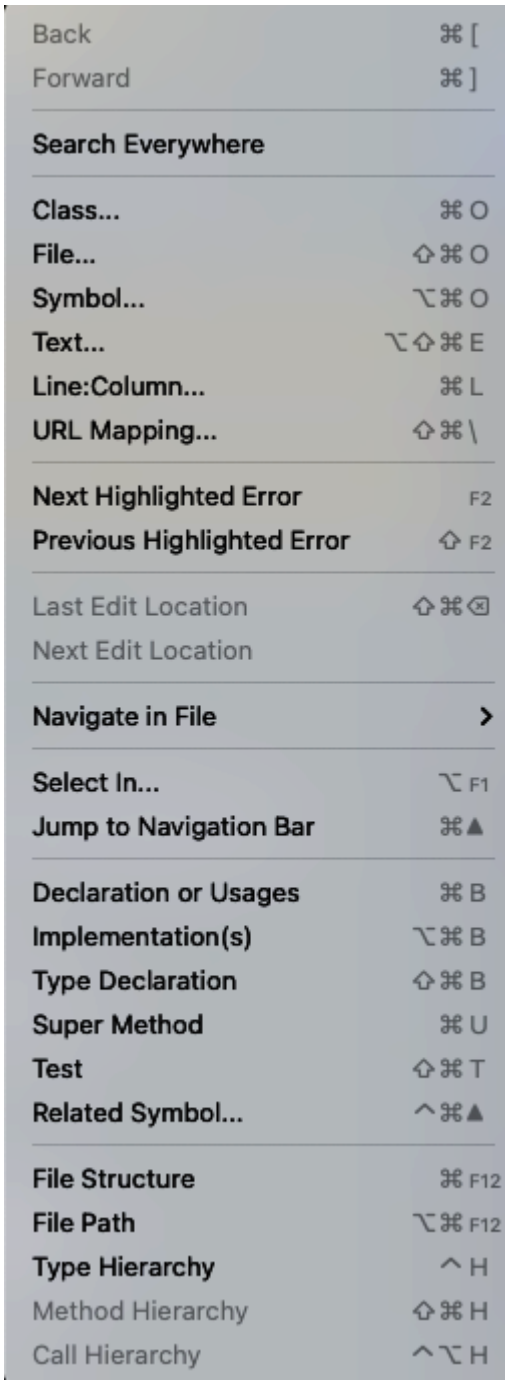

Figure 7-11. IntelliJ IDEA's Navigate menu

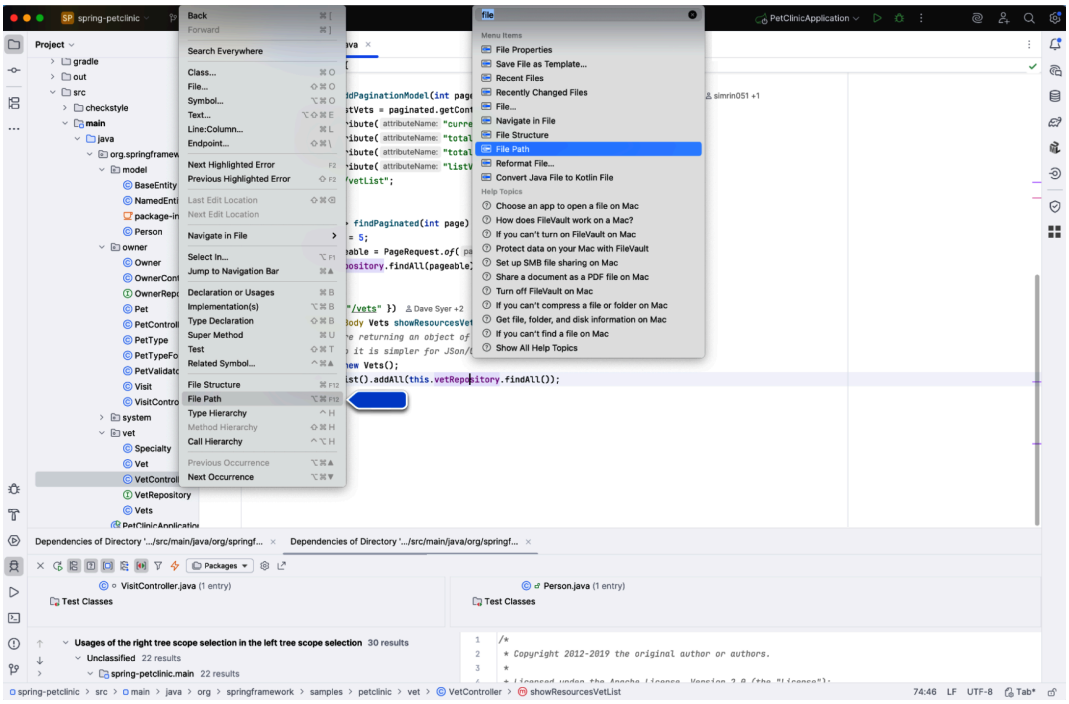

Figure 7-12. Searching Help for “file” in IntelliJ IDEA

The Importance of Good Error Messages

As much as you might like to think otherwise, your users aren’t perfect and will make mistakes. Responding appropriately to errors is often a determining factor in the overall usability of an application. As much as you can, you want to prevent errors from happening in the first place, but when your users inevitably step off the happy path, they should be greeted with a meaningful message (something more meaningful than “Interface not registered” please, as shown in Figure 7-13). Good error messages also help *you* debug and fix issues your users encounter.

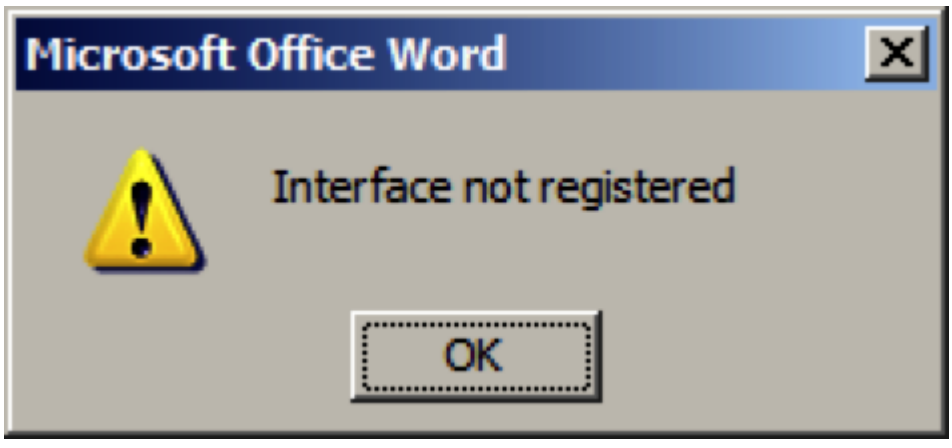

Figure 7-13. A less than helpful message

In addition to giving the users some sense of context, your application should allow your users to recover; don’t reformat their hard drive because they entered the wrong zip code. Providing support for undo or revert is a key factor in allowing users to explore. They won’t try things if they can’t recover from them.

Good error messages do more than just tell the developer what line of code ran into problems; they should help the user understand what went wrong and suggest

alternatives. In Figure 7-14, Google's Gmail ran into an issue. Rather than just giving us a cryptic message, you quickly see you have an internet connectivity issue, and the app will attempt to connect automatically while also offering you an option to "Try now." The message is clear and concise: the application defaults to retrying on the user's behalf and allows the user to take an action if they wish.

Figure 7-14. An excellent error message: the user has an option

Destructive Actions

The edict of making tasks easy for your users has one major exception, and that revolves around any action that is destructive. You rarely want to make it harder for customers to edit data. But when it comes to irreversible decisions—like deleting an account—adding a step or three to the process is often the right thing to do.

Let's take deleting a repository from GitHub as an example. First and foremost, the designers have put the destructive settings at the bottom of the page (also an example of proximity); forcing you to scroll down to them ensures that you won't accidentally click one of these actions. The clear Danger Zone title explicitly indicates that these settings involve danger (see Figure 7-15). Notice as well the text of the buttons include a pop of color to further draw the eye (an example of contrast).

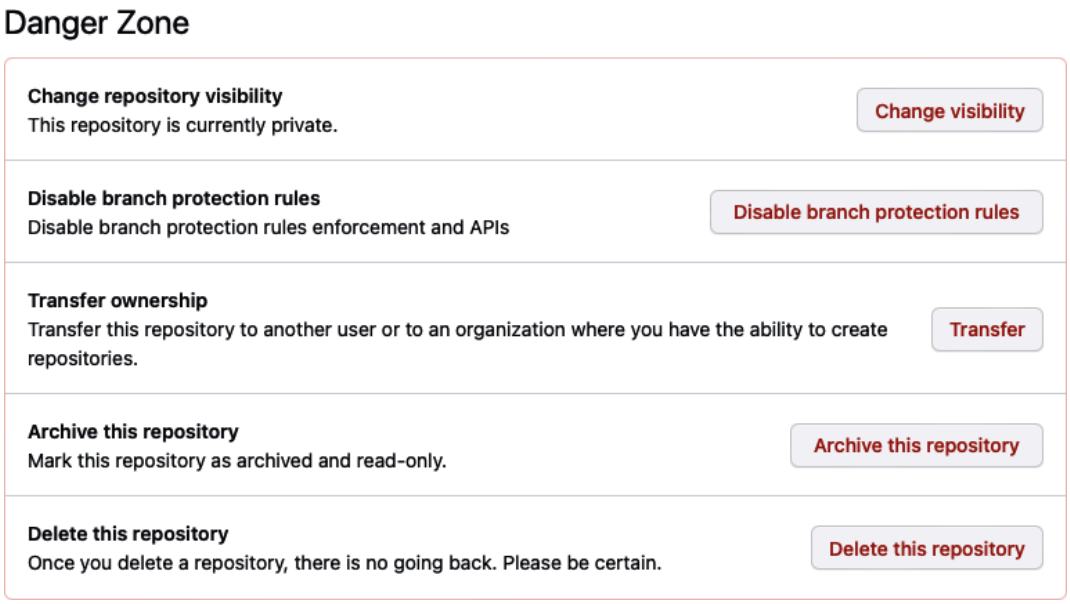

Figure 7-15. Dangerous settings are set apart visually, labeled accordingly, and use a contrasting color (in this example, red) to further emphasize caution

Clicking the "Delete this repository" button does not, in fact, immediately delete the repository! You must click another button that explicitly describes the action (Figure 7-16), and then you receive a warning that further explains what is about to happen, stating the action is irreversible, with *another* button that describes the action (Figure 7-17).

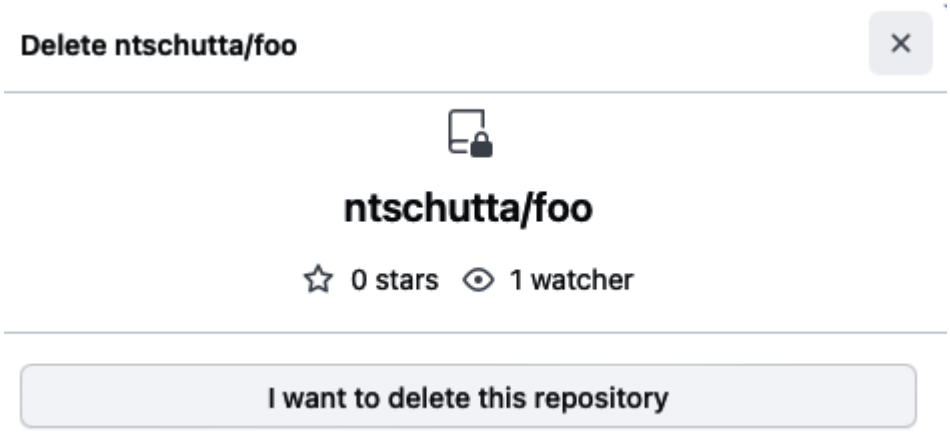

Figure 7-16. Step 1 in the process of deleting a repository: note the button's label isn't merely Delete, but fully spells out the action the user will take by clicking this button

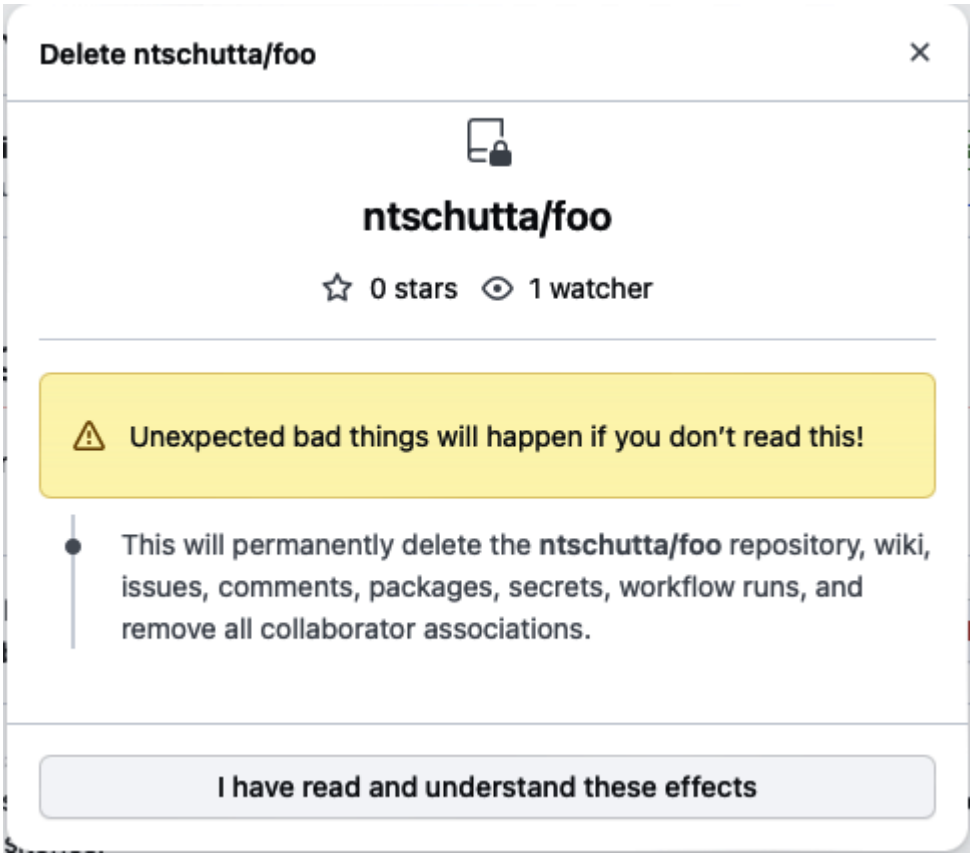

Figure 7-17. Step 2 in the process of deleting a repository includes a warning about the result of the action, and again the label fully spells out the action the user will take by clicking

But you're still not done! Because deleting a repository isn't something you want to do by accident, GitHub asks you to fully spell out the repository name, activating the final, explicitly labeled "Delete this repository" button only if you correctly enter the repository name (Figure 7-18). It may seem like overkill, but the alternative is worse. And these highly destructive actions should be relatively rare use cases for your application.

Figure 7-18. The final step in the process of deleting a repository: the fully labeled button isn't active until the user correctly spells the repository name

Wrapping Up

Usability may not be the first thing you think about on your projects, but it's important to never lose sight of the end user. Do not underestimate the cost of poor usability and haphazard design. While it is helpful to have design expertise on your application team, a bit of knowledge goes a long way! Contrast, repetition, alignment, and proximity may seem simple and basic, but combined with thought, they mean the difference between an average application and a top-notch user experience. In this chapter, you've seen these principles applied to real-world applications, providing you with inspiration for using them in your own systems.

Designing good user interfaces also requires you to consider accessibility, localization, and internationalization. Building a usable application means making it work for everyone, not just those fluent in English or with perfect vision.

Putting It into Practice

On a given day you likely interact with several applications, some with well-designed user interfaces, others…not so much. Now that you've read this chapter, you should have a better understanding of what separates an application that is a joy to use from one that is a chore. Looking at one of those applications now, identify examples of contrast, repetition, alignment, and proximity.

Examine the user interface for an application you've worked on: are there any obvious usability issues? Armed with what you know after reading this chapter, what changes would you make to your application? Take an hour to redesign the interface. Spend an hour or two analyzing the UI design of an application you use regularly: what principles can you find from this chapter? What would you change? Give yourself an hour and see how many violations of these principles you can find in the applications on your laptop or phone. Again, what would you do differently?

Lastly, shadow actual users of your application in their environment. Take notes on what works, what doesn't, and what can be improved. Ask your customers

what they like and don't like about the interface. What would they change? Are the defaults correct? Are interactions as efficient as possible? Don't be surprised if your customers have opinions!

Additional Resources

- *The Design of Everyday Things*, Revised and Expanded Edition, by Donald A. Norman (Basic Books, 2013)
- *The Non-Designer's Design Book*, 4th Edition, by Robin Williams (Peachpit Press, 2014)
- *About Face: The Essentials of Interaction Design*, 4th Edition, by Alan Cooper et al. (John Wiley & Sons, 2014)
- *Designing Interfaces*, 3rd Edition, by Jennifer Tidwell et al. (O'Reilly, 2019)
- The Work of Edward Tufte

[1] Your dentist already praises your flossing regiment? Replace flossing in this analogy with regular exercise, sufficient sleep, or proper eating.

[2] Never forget, you are not the audience for your application (unless you're building developer tools). Just because something is obvious to you does not mean it will be obvious to your customers.

[3] About 8% of men and 0.4% of women have some form of color blindness.

[4] For example, English-to-German translation can expand the text by 35%.

[5] Raise your hand if you've ever increased the font to make something more readable!

[6] Or even giving your users the ability to submit feedback, even something as simple as a thumbs up or down, can provide valuable insights.

[7] Of course, many applications autosave.

[8] Don't think that training is an excuse for producing an unusable mess, though. Training is another component of software that is quick to get the ax.

[9] If they learn it at all—hide the feature well enough, and they will assume your application can't do it!

[10] There's even a plug-in called Key Promoter that will (nicely) tell you when you've missed using a shortcut.