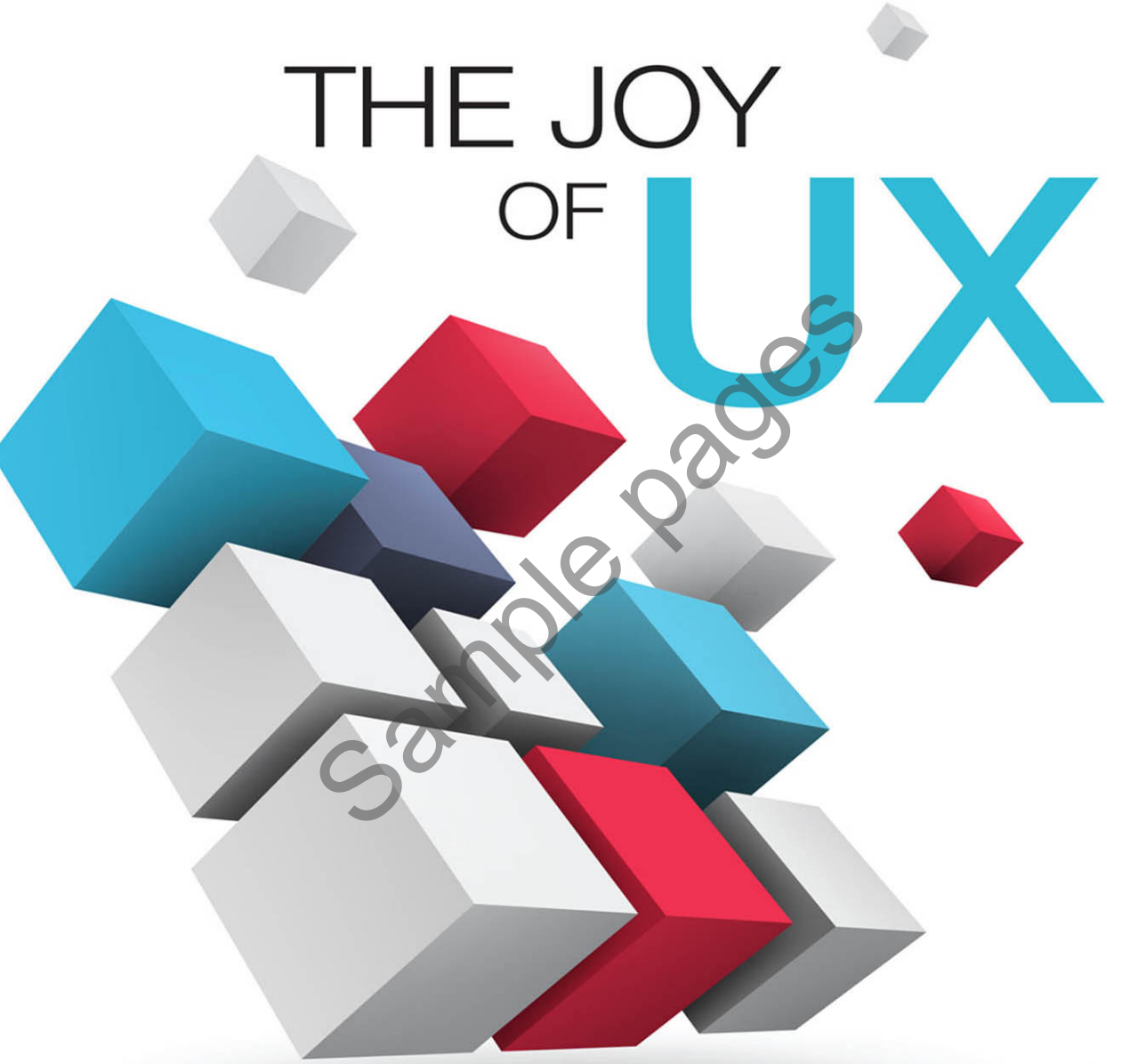
USER EXPERIENCE AND INTERACTIVE DESIGN FOR DEVELOPERS

# THE JOY OF UX





David **PLATT**

# Contents

INTRODUCTION

# UX RULES THE ROOST

User experience (UX) is the primary driver of competitive advantage in software today. Programs with bad UX just won't sell, nor will the hardware or services they are supposed to enable.

Good UX is not that hard, but it requires you to think in new ways. This book shows you how, step by step, with examples along the way.

## Your Biggest Advantage

UX is the primary driver of competitive advantage in the software industry today. Whether you design and sell software as a product (Microsoft), or use it to sell hardware (Apple) or services (UPS), the user experience of your software is absolutely critical.

Just as smoking in public places was once common, it was once common to force users to contort themselves into five-dimensional hyper-pretzels to match their software—to become "computer literate," in the term of that day. UX guru Alan Cooper wrote that a computer-literate user is one who "has been hurt so often that the scar tissue is so thick that he no longer feels the pain." Users once accepted this as the price of getting their computing jobs done. They won't do that anymore.

Remember when Apple was left for dead in 1997, kept alive solely by a cash infusion from Microsoft so that Microsoft could claim it wasn't a monopoly? How did Apple become the most valuable company ever seen on the face of the planet? By turning out great UX, for which its customers pay premium prices. That's how important UX has become.

UX is critical to the enterprise sector as well. In December of 2014, Avon had to kill a new version of its order management software. The *Wall Street Journal* reported that the company's sales force of independent reps "found the new system so burdensome and disruptive to their daily routine that many left Avon."

Even IBM, the stodgiest of stodgy companies, recently announced that it was spending $100 million on its UX consulting business, opening ten new labs worldwide and hiring 1,000 new workers.

Whatever you are doing, and whomever you are doing it for, you need an excellent UX. It's not optional anymore.



## UX Is *Not* Fonts and Colors

Too many developers and managers think that UX design is selecting colors and fonts and button radii. Nothing could be further from the truth. The rounded window corners and cutesy animations are the last and least important pieces of the UX. My fellow Software Legend Billy Hollis calls that stuff "decoration, not design."

What's the difference between user experience (UX) and user interface (UI)? As is always the case when specific meanings are forced onto generic words, it is difficult to find any two writers who agree on what they mean. Throughout this book, I will use UI to mean the decoration function that is the very last thing you do to a piece of software. I will use UX in the meaning published by Jakob Nielsen and Donald Norman, who wrote that "user experience encompasses all aspects of the end-user's interaction with the company, its services, and its products." That

means that anything the user ever sees, hears, touches, or thinks about is the UX: a program's workflows, its feature sets, its required inputs, the form of its outputs.

Figure 0.1 illustrates the differences. Figure 0.1a shows a product. Consider this to be the computing job that you need done. Figure 0.1b shows the UI, the tool with which you interact with that product. Figure 0.1c shows the full UX, the totality of your interaction with that product.

The battle for good UX is usually won or lost long before the program reaches the decorators. Think of your program as a piece of furniture—say, a table. The decoration is the surface finish on that table. Certainly tables should be finished well rather than poorly, and so should your programs. But if you build the table out of the wrong material, one that doesn't satisfy the user's needs, even the best finish in the world can't help. If your user wants a decorative table for a nature-inspired living room, choosing wood will probably make him happy. On the other hand, if your user needs a working table for a cafeteria kitchen that undergoes daily sanitizing, metal would be far better. And backing up a step, does your user really need a table, or would a chair solve his problem better?



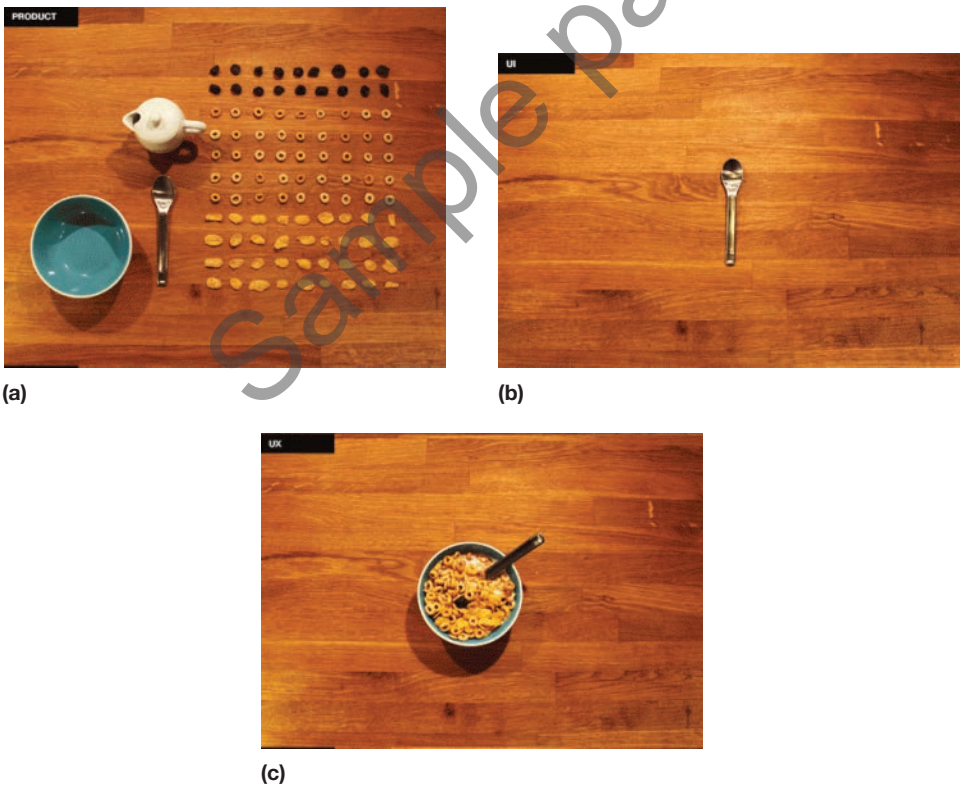



**Figure 0.1** (a) Product, (b) UI, and (c) UX. (Ed Lea, Product Design)

# Fundamental Example

Let's look at an example of the fundamental development decisions that make or break a UX. I was teaching in Sweden some time ago and opened up Google by using its base address, www.google.com. Its servers detected the country in which I was located and automatically replied with the Swedish version of the page (Figure 0.2). This decision is correct for most users most of the time and requires just one click (lower center) to fix it permanently (persistent cookie) if it's ever wrong.

On the other hand, consider UPS.com, home of the package delivery company (Figure 0.3). UPS.com requires users to select their country before they can do anything at all. That takes 30 clicks if you're in Sweden. You also have to explicitly tell the site to remember you (see the check box) or it'll make you do it again next time. That's no way to treat customers.

What happened here? Were the Google programmers so much smarter than the UPS programmers that Google could detect the incoming country of a Web request while UPS couldn't?

No way. According to UPS.com, its site handled over 100 million tracking requests on its peak day in 2014. The UPS programmers have to be pretty good to build a site that successfully handles such a large volume. There's no way that such skillful programmers wouldn't know how to find the IP address of an incoming request, and hence determine its probable country of origin. (It's not difficult: simple static table lookup, cache it in RAM for speed, update the table once per day. Easy.) UPS is therefore choosing to make users explicitly enter their country, instead of automatically detecting it.




**Figure 0.2** Google home page accessed from Sweden.

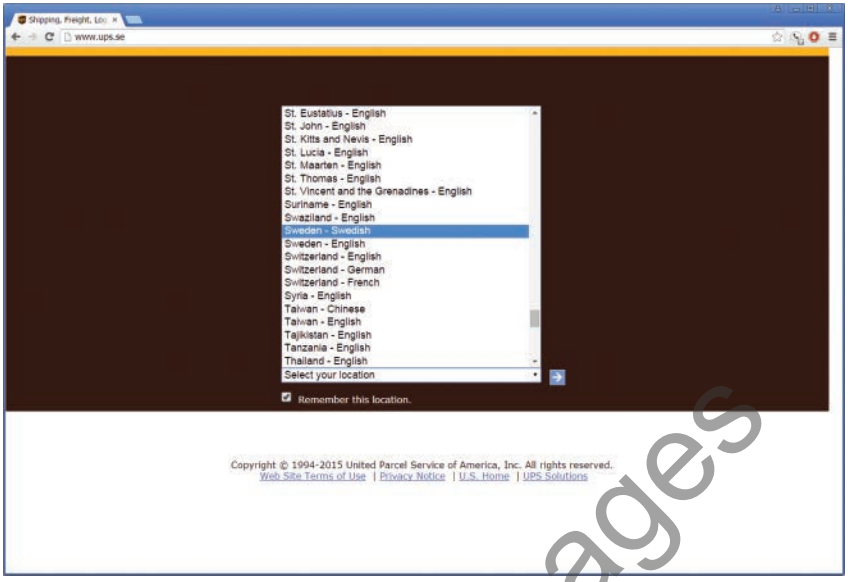



**Figure 0.3** UPS home page accessed from Sweden.



In my opinion, UPS is committing *the* cardinal sin of all UX work: failing to put itself in its users' shoes. The technologists who made this choice are behaving like the geeks they undoubtedly are (and I am, and you are too). We are trained mathematically, logically. We get it hammered into us from middle-school algebra onward: a theorem that's true in 99 cases but false in the 100th case is a false theorem. Bad geek. Throw it away; go find a true one. UPS won't make a guess because it might be wrong.

That's acceptable for mathematical theorems, but not for human users. Unlike a theorem, if your program makes 99 out of 100 users happy, you are probably having a pretty good day. And it's probably more important to make those 99 users happy again tomorrow than it is to figure out how to please that 100th user—especially if what that user wants would annoy the other 99. Clearly there are cases when that's not good enough—air traffic control springs to mind. But for most business and consumer situations, the world is a better place when you handle the main case seamlessly and fix unusual cases only as they arise, rather than annoying all users by making them do work that the site could be and should be doing on their behalf.

Google's language selection algorithm doesn't always guess right; maybe the request isn't actually coming from Sweden, maybe it is but the user doesn't speak Swedish (me), or maybe it is and the user does speak Swedish but doesn't want to right now (for example, a Swedish college student practicing English). But making its best guess, and having the user correct any resulting errors, is a large net profit for the overall user population. Which company's approach makes you feel that it values your time and effort, and makes you want to come back? In fact,

Google has thought so long and hard about its users that it has figured out how to recognize a UPS tracking number. If you type one directly into Google search, Google will offer to track it for you (Figure 0.4). If you click that link, Google will jump you straight to the UPS tracking page (Figure 0.5) with the correct language already selected. That's why I always use Google to track my UPS packages, instead of hassling with UPS.com. And I can't suppress an ironic chuckle as I do so.

You can see that this isn't a *graphical* design problem, not at all. Both sites have their logos, their corporate color palettes and fonts, everything. But one site makes all users do extraneous work—overhead, excise, distraction—before users can even begin to do what they went to the site for: their business logic, in this case tracking packages. The other site jumps right in and starts banging away as best it can, taking care of most users seamlessly and allowing corrections as needed.

The difference between these sites is one of *interaction design*, sometimes known as behavioral design, and occasionally as information architecture. That's what this book is about. We won't be discussing graphical design. We won't be discussing how to program these designs, either. There are lots of books on both of these topics. We'll be studying *how to decide what should get programmed*. And we will always, *always*, be hammering on the side of the user.



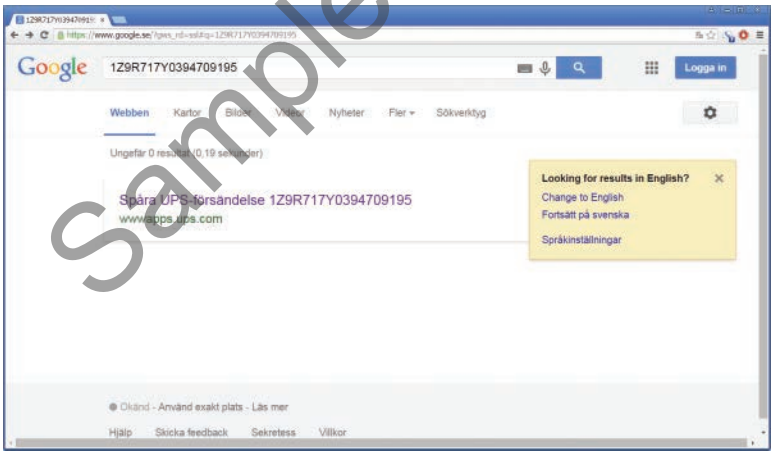



**Figure 0.4** Google automatically recognizes a UPS tracking number and offers to track the package.

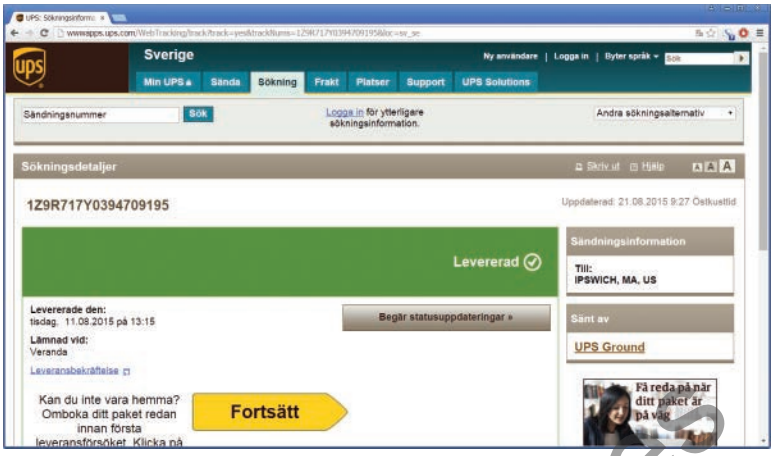



**Figure 0.5** Google automatically displays the UPS tracking page in the language of the user's browser—much easier than going through UPS.com.



### PLATT'S FIRST, LAST, AND ONLY LAW OF UX DESIGN

The Talmud speaks of the impatient man who came to the famous rabbi Hillel, saying, "Teach me the Torah [the first five books of the Hebrew Bible] while I stand on one foot." Hillel replied, "What you do not want done to yourself, do not do to others. The rest is commentary, go and study."

Do you want to know everything there is to know about UX? I can answer in one sentence, Platt's First, Last, and Only Law of User Experience:

KNOW THY USER, FOR HE IS NOT THEE.

The rest is commentary, my friends. Come and study along with me.

## The Three Fundamental Corollaries

I majored in physics as an undergraduate, and there's still enough of the physicist in me to insist on setting forth the fundamental principles from which I derive my judgments as to good and bad usage. Starting from the FLaO Law mentioned in the sidebar, we can derive the following corollaries:

- **First Corollary:** The software that you write has zero value in and of itself. The only value that your software ever has or ever will have is the degree to which it makes your users happier or more productive.

- **Second Corollary:** The software increases the happiness or productivity of users in one of two ways. First, it could help a user solve a specific problem—write an article, pay a bill, navigate a car. Or it might put the user into a state that she finds pleasurable—listening to music, playing a game, video phoning her grandchildren. Those are the only two cases that ever can exist, though sometimes they blend into a hybrid.
- **Third Corollary:** In neither of these cases do users want to think about the programs they are using. At all. Ever. In the former case, they want to think about the problem they are solving: the wording of the document they are writing, or whether they have enough money to pay their bills, and which unpaid creditor would hurt them the most. In the latter case, the users want to get into that pleasurable state as quickly as they can and stay there as long as they can. Anything that delays the start of their fix, or distracts them from it while they're enjoying it, is even less welcome than the interruption of a work task.

To summarize these three corollaries: Users don't care about your program, or you either. Never have, never will. Your mother might, because you wrote it and she loves you, and then again she might not; but no one else does. Users care only about their own productivity or their own pleasure. Every single user of every single program wants what Donald Norman calls the "invisible computer" in his landmark book of that title.

You can see in that earlier simple example that Google does its best to be as invisible as it can, while UPS does not. With the FLaO Law and its corollaries in mind, let's examine another common case.



## Example: Save Me?

Here's a situation you see every day. It's probably become second nature, to the point where you don't think about it anymore. But we're going to think about it now.

You open a document in Microsoft Word. You add or edit some text, and then you go to close Word. What does Word do? It pops up a dialog box asking, "Do you want to save changes?" (Figure 0.6). The beret-wearing, incense-burning graphical designer can decorate the "Save Changes?" dialog box with nifty fonts and color gradients and nicely rounded corners. But he doesn't, can't, address the question of whether the program should prompt the user on exit, as it does, or whether it should automatically save changes as they are made, as does Microsoft OneNote. Which would make the user happier and more productive? It falls to us, the UX interaction designers, to make that determination. What should we choose?

Start by doing the arithmetic. Word requires a choice and a mouse click each time the user closes a document. If the user has 100 editing sessions, he makes 100 choices, 100 clicks. If we switch to automatic saving, we have to put the capability of a complete rollback somewhere

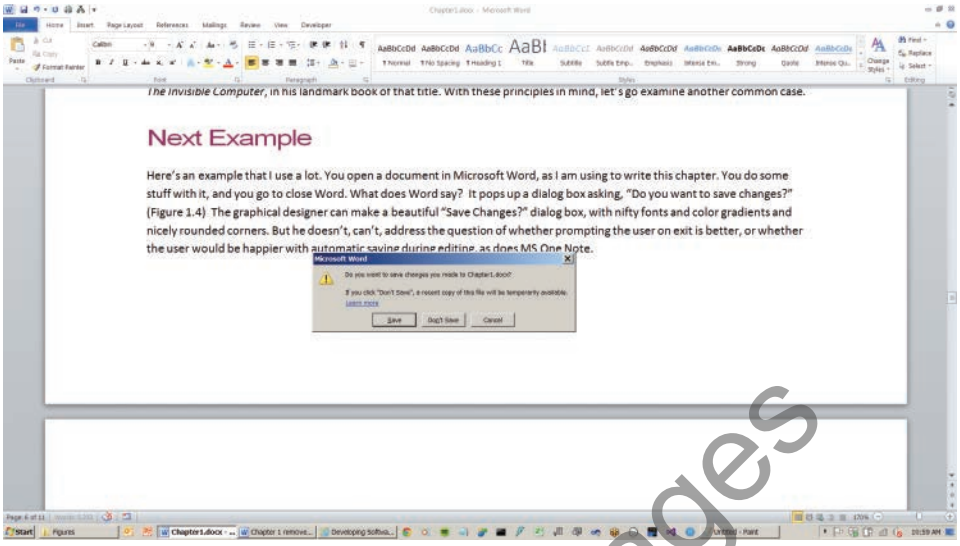



**Figure 0.6** Microsoft Word, prompting the user to save changes. Why not save automatically, as does Microsoft OneNote?



else—possibly a "Discard Entire Session" item on the Edit menu, which would take perhaps five clicks to access. If users save their changes 99% of the time, automatic saving would eliminate 95 clicks out of every 100 in the saving process, a huge reduction of overall user effort.[1] If users save their changes only 50% of the time, automatic saving would actually increase the overall user effort by a lot; every 100 clicks in the saving process now mushroom to 250 clicks.[2] Which choice is right for your program?

As usual, it depends. How often do you save your changes in Word? Or look at it the other way: how often do you screw up your document so badly that you discard the changes by selecting "Don't Save"?

Wrong question. Are you developing this app for yourself? Almost certainly not. Then what does it matter what you yourself do? It doesn't. Not in the slightest. Read the FLaO Law again: Thy user is not thee.

How often do *your users* save their changes, versus how often do *they* discard their changes? Entirely different question. Still, don't you feel yourself wanting to say, "Well, I hardly ever see them discarding"? Again, those are your own preferences talking. It is surprisingly difficult to

1. The 99 users who save changes go from one click to zero. The one guy who wants to discard all his changes now goes from one click to five clicks. One hundred clicks now shrink to only five.
2. The 50 users who save changes go from one click to zero. The other 50 guys who discard all their changes go from one click to five clicks. One hundred clicks now mushroom to 250.

remove yourself from this equation. You subconsciously resist the notion that your users are different from yourself. So what do you do?

You could try asking the actual users, if you can find them. If you work on an in-house development team, building programs for use inside your company, this could work well. You go to the floor where the users are and ask them. However, there are snags with this approach. Are the users willing to talk to you? Can they remember accurately? Are they afraid of looking stupid? Will their bosses allow them to take the time? Talking to actual users is a very good start. Chapters 1 and 3 discuss ways of obtaining information from this channel. But you can't always get it.

If you don't work on an in-house development team, that is, if you build products for external customers, the problem gets harder. Suppose you asked people in your office. Your coworkers, by definition, spend all day, every day, developing software for sale. Do they resemble your user population? Unless you are in the business of building software development tools, probably not. Whatever they would tell you is probably misleading for your user population. Microsoft has stumbled over this problem more than once.

So how do you find out what percentage of users save their changes? Not by some mystical telepathic intuition, known only to crystal gazers who burn incense and eat sprouts and wear berets, but by collecting hard engineering data via telemetry, over many more users than you could afford to test in the lab. Chapter 5 explains telemetry in more detail. You could also do some early usability lab testing, as described in Chapter 4.



## Bake UX In from the Beginning

The biggest single mistake that I see companies making is not starting their UX planning at the beginning of a project. "We need to get the services in place first; then we'll think about what it looks like." That's crazy. That's like saying to an architect who is designing a house, "We won't ask who's going to live here until we get the heating and the plumbing in place." Are you building a house for a downsizing older couple? You'll want a full bath on the ground floor with a wide door and the potential for grab bars. For a younger couple with two kids and planning four more? Entirely different problem. The last thing you want to do is to spend your development budget before you know even the most basic things about what you're building. And the UX determines that.

As you can see from the previous examples, UX design decisions determine the code that needs to be written, not just in the top layers that handle the user interactions, but reaching down to the lowest levels of the application. In the case of Word, the structure of the entire Undo mechanism depends on UX design decisions about when and how files get saved. And in the Google versus UPS case earlier in this chapter, the developers who build the home page need to know if the information about the user's country will be available to them when their code runs (Google), or if they need to get that information from the users and put it somewhere for rest of the site's code to use (UPS).

Good UX design starts at the very beginning of a project. It's not a superficial layer. It permeates all levels of an application, as character and honor (or lack thereof) do a human personality. And it needs attention through all stages of program development, nay, throughout all stages of the program life cycle, as character and honor need attention throughout all stages of the human life cycle.

Clients sometimes ask me to critique their UXs just before they ship. That's way too late to change anything. The architecture is set, the budgets spent, the attitudes hardened. Consider yourself warned.

# Why Developers Don't Consider UX

A computer that users can't figure out how to use, or that requires expensive training to use, or that makes expensive mistakes because the program misleads users, is a very expensive paperweight. Yet many developers or architects think they don't need to understand UX. Here's why they say that, and why they're wrong.

## Our Projects Are Low-Level, So UX Doesn't Matter

Nonsense. Every project has some connection to a human user somewhere. Even a programmatic Web service needs error reporting, installation and configuration, status- and performance-monitoring dashboards, and so on. If a project has only small amounts of UX, that's all the more reason that those pieces need to work well. Twenty years ago, you might have gotten away with a dialog box saying, "Web Service Failure, Error 20. Consult Documentation." Today you would get laughed out of the arena for shipping something like that.

## Marketing Decides Our UX

It's wise to have a good relationship with your marketing department. They certainly feel pain points from the customer and bring them back to you. At the same time, marketeers are not interaction designers. They might give you the first clue as to what would make the customer happier and more productive—"Customers complain that they're losing work when our app crashes"—but it's up to you to take it from there. How often does it happen? How do you detect and measure it? To fix it: Auto-save? Rollback? How often? Configurable? Imagine asking these questions of your marketing department, and tell me if they're really driving the UX. They're not; you are, even if they sound the initial alarm.

You should also be talking to your tech support department. They feel your customers' pain far more immediately and brutally than the glad-handing marketeers.

## We Have a UX Group That Does That Stuff

Some large companies have a UX design group that approves every user interaction. If you have one of these, you've already found that their time is in tight supply, as it is for other ultra-specialists such as ophthalmologists. You can get a 15-minute appointment in six months if you beg. They can't follow up or help you iterate. You need to understand what they tell you in your far-too-limited interactions and implement those principles day to day to make the best use of the thimblefuls of their concentrated wisdom that you actually get.

Also, their time is (quite rationally) dedicated to your company's primary, outward-facing programs. They don't have time for internal or second-tier apps. A company that has this type of group values good UX. The apps you work on are held to a higher standard. But your bosses don't give you the skill set or resources to meet these demands, now do they? You have to be ready to do the day-to-day work at a project team level.

## UX Is for the Beret-Heads

Also known as graphical designers, more accurately they are decorators. As we've seen, the UX game is almost always won or lost before it reaches them. Be nice to them. But the main battle isn't theirs, it's ours.



# Where to Get the Skills

Precisely because your UX needs attention throughout the development process, you need someone with UX skills assigned directly to your design team. This person will know that the correct choice in the document-saving example comes not from arguing personal tastes and philosophies, but from hard user data—knowing what percentage of documents are discarded rather than saved. And she will know how to obtain that data, ideally by instrumenting the program, but through skillful interviews and observations if that can't be done. Where are we going to get such people, and how are we going to manage them?

Regular programmer geeks don't know how to do it; they mistakenly think that their users resemble their geeky selves, and their UX designs come out looking like Visual Studio. Sometimes marketing people want to get into the act, figuring that they interact with customers so they know what they need. That's like saying that you have teeth in your mouth, so you know how to do a root canal. Graphic designers sometimes try to get into the picture, but as you can see, this interaction design isn't fundamentally a graphical problem. How do we get the people we need?

Consider the US Army, specifically its most basic unit of operations, the infantry platoon. It consists of a green second lieutenant in nominal command, an experienced first sergeant who's really running it, and about 40 fighters. And each platoon has a medic. The medic is not a fully qualified doctor, although the platoon soldiers customarily address him as "Doc." The army

can't afford to create enough full-fledged doctors to place one in each platoon. The medic is trained in battlefield first intervention to stabilize the wounded soldier—stopping severe bleeding, starting IVs, opening airways, and so on. Having this intervention immediately available is the first link in the amazing chain of battlefield casualty survival today.

What we need in the UX business is the equivalent of a medic. We need someone who knows the basic concepts of UX design and their most common applications—for example, knowing that data is the key to most UX questions, and knowing how to start obtaining it. Someone who knows how to generate a user persona quickly and accurately, to help the design team grasp the slippery concept of "the user." Someone who knows how to do a usability test quickly and cheaply so it doesn't hold up the project, or get skipped to keep it from holding up the project. The key point is getting UX questions answered quickly. As in trauma medicine, getting treated in the golden hour is key.

Sometimes you get pushback on the medic concept in companies that recognize the importance of UX and have a central UX team that wants to control everything. Continuing the army medic analogy, these are the highly skilled surgeons in the base hospital. If you frame it right, these people will be the biggest beneficiaries of having UX medics on the project teams, for example, having the first round of usability tests already done when they get called in to evaluate the puzzling data.



## You Can Do This

My readers and students tell me that the hardest part of producing a good UX is knowing where to start. It is so very tempting to jump right into the development—OK, we've got this project, the schedule is tight (it's always tight), let's get going. No, don't waste time on that persona nonsense, we have to get going. Stories? What are they? Never mind, fire up Visual Studio (or Expression Blend if you're less geeky). Jump right in and drag and drop. Should we have check boxes here? Radio buttons? How about a set of tabs instead?

I recall the opening of *The Joy of Cooking*, one of the *Joy* books that inspired my title choice for this volume. Written for the person who knew nothing about cooking, it began with the instruction "Stand facing the stove." That's how I've written this book, starting you from zero.

After this introduction, each of the first seven chapters represents one step to a great UX. (That's five steps fewer than it takes to kick the booze.) Each chapter introduces one specific UX design technique. I've placed them in the order in which I generally use them in my practice, though as you'll see, there are certain loopbacks and iterations. If you follow these steps, without skipping any, you will come up with something good, or at least a whole lot better than if you had just jumped in and flailed away. With my usual modesty, I call them the Platt UX Protocol.

Chapter 8 and Chapter 9 each presents a case study, working the seven steps from beginning to end on a specific project. I present personas, stories, sketches, testing, telemetry, security, and final simplification. My students tell me that this, the end-to-end discussion about how all the pieces fit together, is their favorite part of the class.

Here's what each chapter deals with:

- **Chapter 1: Personas**—We learn and understand who the user actually is. Is the user male or female? Old or young? High or low disposable income? Education type and level? What do users hope for and what do they fear? We write up this data in the form of a persona, an artificial person who represents our user population.
- **Chapter 2: What Do Users Want? (And Where, and When, and Why?)**—We work on understanding a user's motives and activities in using our software. What problem is the user trying to solve, or what pleasurable state does the user want to maintain? What would the user consider to be the characteristics of a good solution or pleasurable state? We represent this information through stories, narratives written from the user's point of view. (If you're familiar with stories as part of agile development, you'll see that these are different.)
- **Chapter 3: Sketching and Prototyping**—We know who the users are and what they need. Now, and only now, do we start sketching out some possible solutions. Using a low-fidelity editor (in this book, Balsamiq), we generate mockups quickly, so that we can begin the iteration process of testing them and refining them.
- **Chapter 4: Testing on Live Users**—We have some mockups illustrating possible solutions. Now we test them, ideally on actual users but on user surrogates or representatives if that's not possible. The degree of fidelity that we show to the users depends on the progress of our project. We will generally iterate steps 3 and 4 several times during the course of the project.
- **Chapter 5: Telemetry and Analytics**—We plan for our applications to have some sort of telemetry, so that we can understand what users are actually doing with it. We will see which features they are using, and in what order, as well as information about their hardware. Failing to provide telemetry in today's environment would be like practicing medicine without X-rays or lab tests.
- **Chapter 6: Security and Privacy**—Security and usability are often seen as polar opposites. In this chapter, we carefully examine the interaction between these two and understand what happens when it breaks. We work out a plan for securing our application as tightly as needed, while still making it as usable as possible.
- **Chapter 7: Making It Just Work**—As we get closer to release, we start looking not to add features, but for ways to remove user effort from the features we have. This is the level of final polish that we give our program.

- **Chapter 8: Case Study: Commuter Rail Mobile App**—We work through all of the steps as we design a new mobile app for Boston's commuter rail system.
- **Chapter 9: Case Study: Medical Patient Portal**—We work through all of the steps as we design a new Web portal for a Boston-area hospital system.

## This Book's Web Site

This book, like everything else in the world, has its own Web site, JoyOfUX.com. Figure 0.7 shows a screenshot.

You can find on it all of the resources to which I refer in this book, such as persona templates. I will also be adding case studies, similar to the ones at the end of this book. So come back every month or so and have a look at them. And if you have a case study that you'd like to contribute, I'd be happy to see it.

## And Here We Go . . .

To paraphrase Arlo Guthrie's song about resisting the Vietnam War draft: If one guy does it, they'll think he's crazy. And if three guys do it, they'll think it's an organization. And if fifty people do it, they'll think it's a movement.

And that's what I hope our revolt against bad UX will become. So let's get to it.




**Figure 0.7** JoyOfUx.com Web site.