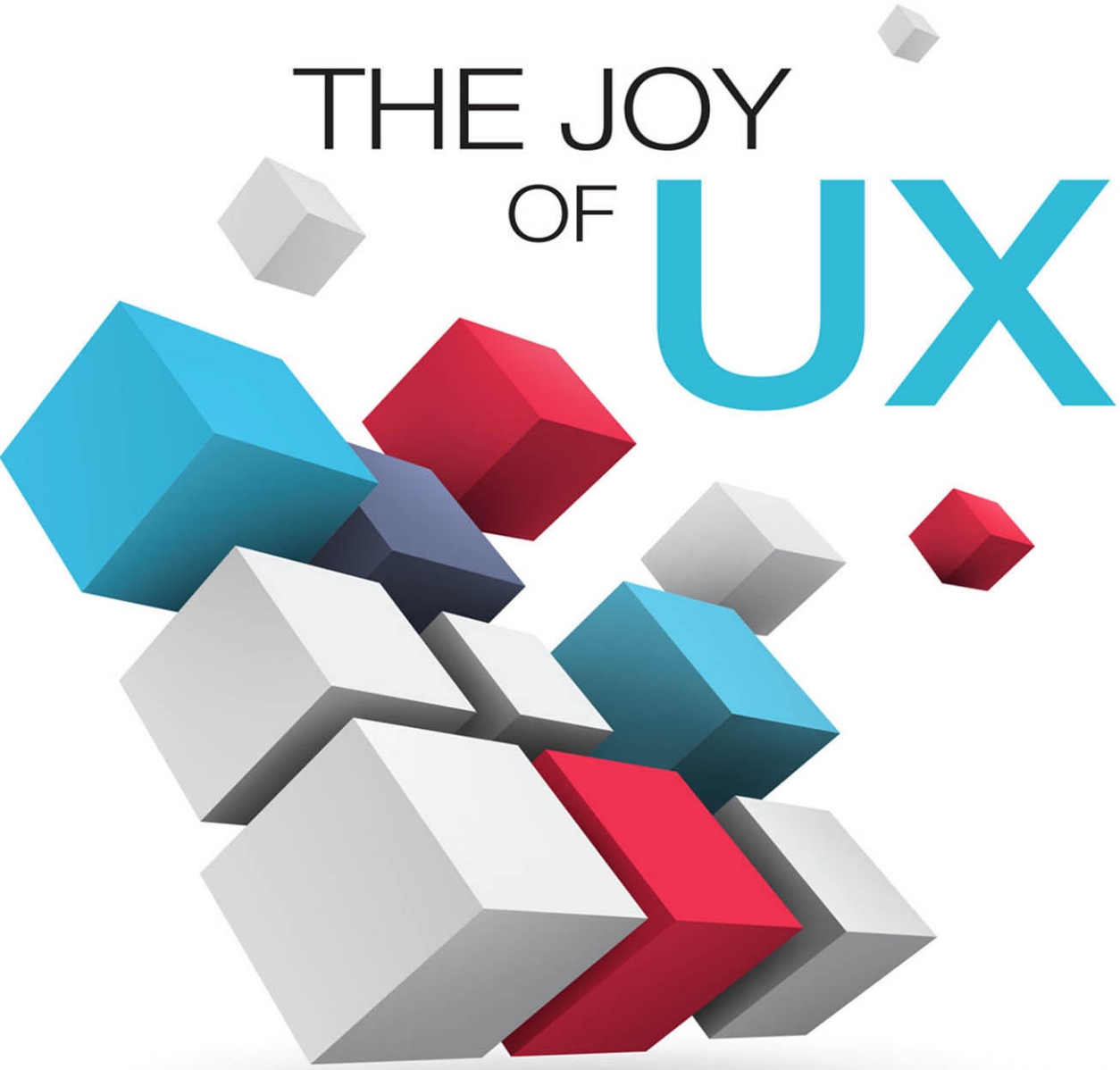
USER EXPERIENCE AND INTERACTIVE DESIGN FOR DEVELOPERS

THE JOY OF UX



David **PLATT**

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

The Joy of UX

The Joy of UX

User Experience and Interactive Design for Developers

David Platt

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Names: Platt, David S., author.
Title: The joy of UX : User Experience and interactive design for developers / David Platt.
Description: Boston : Addison-Wesley, [2016] | Includes index.
Identifiers: LCCN 2016009039| ISBN 9780134276717 (pbk. : alk. paper) | ISBN 013427671X (pbk. : alk. paper)
Subjects: LCSH: User interfaces (Computer systems) | Human-computer interaction. | Computer software—Development.
Classification: LCC QA76.9.U83 P54 2016 | DDC 005.4/37—dc23
LC record available at https://lccn.loc.gov/2016009039

Copyright © 2016 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-427671-7
ISBN-10: 0-13-427671-X
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, May 2016

Publisher
Mark L. Taub

Executive Editor
Laura Lewin

Development Editor
Michael Thurston

Managing Editor
Sandra Schroeder

Full-Service Production Manager
Julie B. Nahil

Copy Editor
Barbara Wood

Indexer
Infodex Indexing Services

Proofreader
Linda Begley

Technical Reviewers
Lars Athle Larsen
Gregg Tompkins
Moshe Raab

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
Shepherd, Inc.

To all my students, from whom I learn so much.

This page intentionally left blank

Contents

This page intentionally left blank

Foreword

Occasionally you meet a teacher who's so enthusiastic about what he teaches that the enthusiasm rubs off almost contagiously. David is that rare kind of teacher. I've seen it live in his classes and with his students. I see that same love and enthusiasm in *The Joy of UX*, and I know you will too.

I have to confess: I'm the wrong person to write this foreword. I may have the world's worst natural sense of user interface design. But over the years, and especially with the assistance of David Platt, I've produced some pretty great software. The software produced by the teams I've led has generated billions in revenue and has been used by millions of people.

Incredibly, I've also learned to enjoy creating breathtaking UX. I once thought building a great user experience was a tedious process, primarily about colors and fonts and something to do with the golden ratio. Far more critical are the tools and techniques David teaches: empathy for your user, hypothesis testing, and iterating on those hypotheses by watching how users interact with the software. Put another way: fall in love with your users and prioritize their happiness.

When I finally embraced the techniques and processes David writes about in this book, I didn't just end up creating delightful user experiences. It also had a profound effect on how I develop software. At the heart of a great user experience is empathy for the user. Developing this empathy requires a true understanding of who your users are. This changed my entire orientation from thinking about *what* (What code should I write? What language should I use? What is the technology?) to thinking in terms of *who* and *why*. This requires hard work and hard thinking. You must leave your keyboard behind and adventure out into the world. You must meet the people who are using (or will use) your software and talk to them. They will surprise you. They will defy expectations. After you build, you must go back to them and continuously develop your relationship and understanding of how they think, what motivates them, and discover the "why" behind what drives them.

I also learned to take a much more iterative approach to software development. "I should build feature 'X'" turns into a hypothesis that needs to be validated—by the customer. Applying a user-centric process to all feature decisions made my process leaner than just using a Kanban board or Scrum. To determine what to build next, we must determine what will have the highest impact on the user's experience.

These techniques are vital today, especially when building mobile applications. End users have tremendous choices. They are fickle. If they don't grok your application immediately, they'll never touch it again.

You can delight your users by incorporating David's advice into your workflow. Get inside your users' heads, develop a deep understanding and empathy for their lives, test your assumptions, and discover "why."

—Keith Ballinger
Vice President of Product at Xamarin

About the Author

David Platt teaches User Experience Engineering at Harvard University Extension School and at companies all over the world. He's the author of 12 programming books, including *Why Software Sucks* (Addison-Wesley, 2006) and *Introducing Microsoft .NET* (Microsoft Press, 2003). When he finishes working, he spends his free time working some more. When readers ask, "Did you really tape down two of your daughter's fingers so she'd learn how to count in octal?", he just smiles. Microsoft named him a Software Legend in 2002. Dave lives in Ipswich, Massachusetts. You can contact him at www.joyofux.com.

This page intentionally left blank

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?

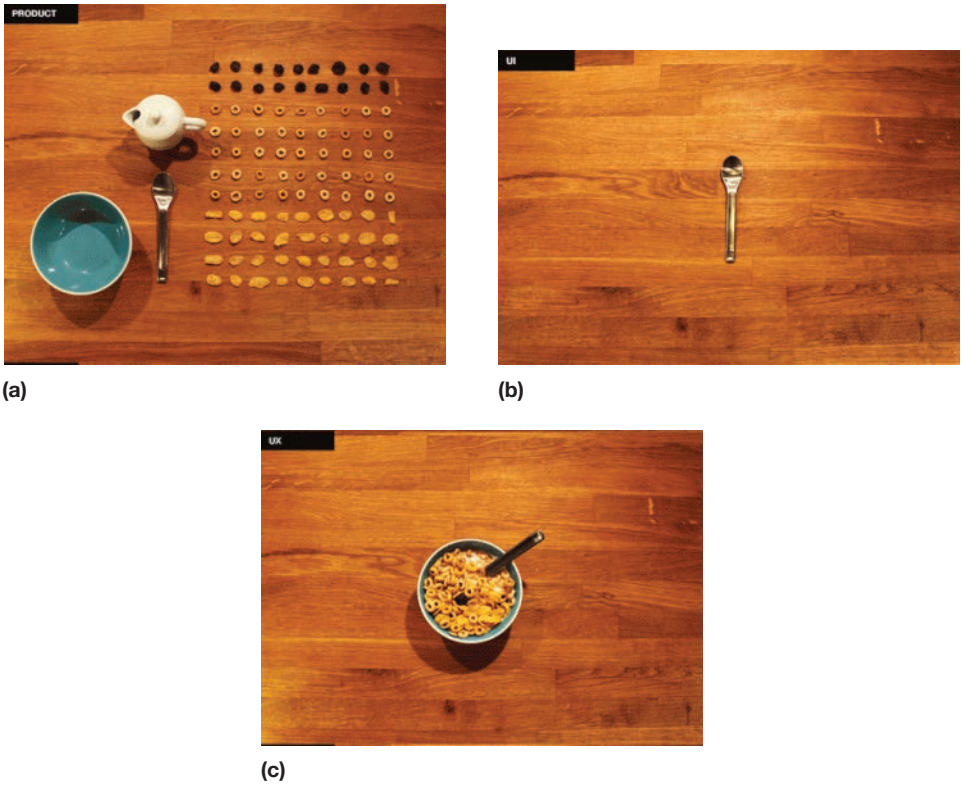


Figure 0.1 (a) Product, (b) UI, and (c) UX. (Ed Lea, Product Design)

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.

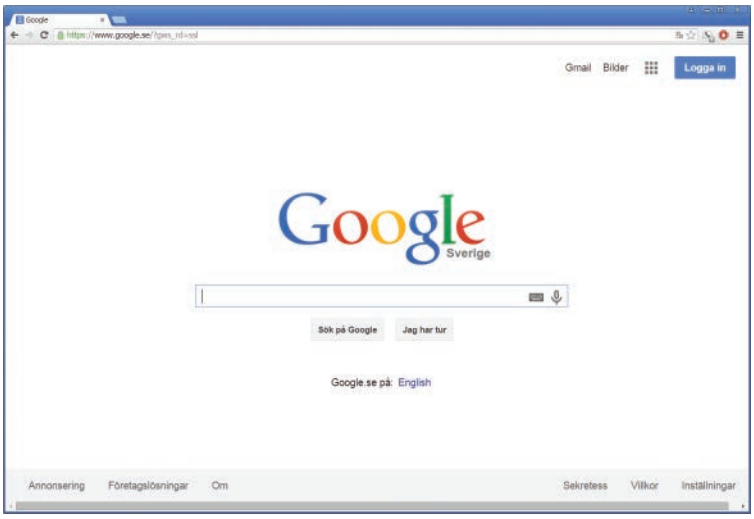


Figure 0.2 Google home page accessed from Sweden.

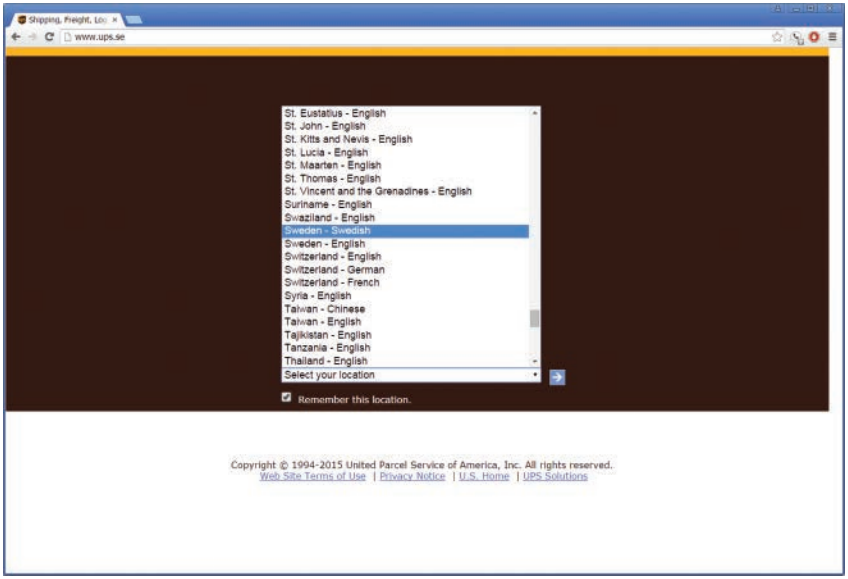


Figure 0.3 UPS home page accessed from Sweden.

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.

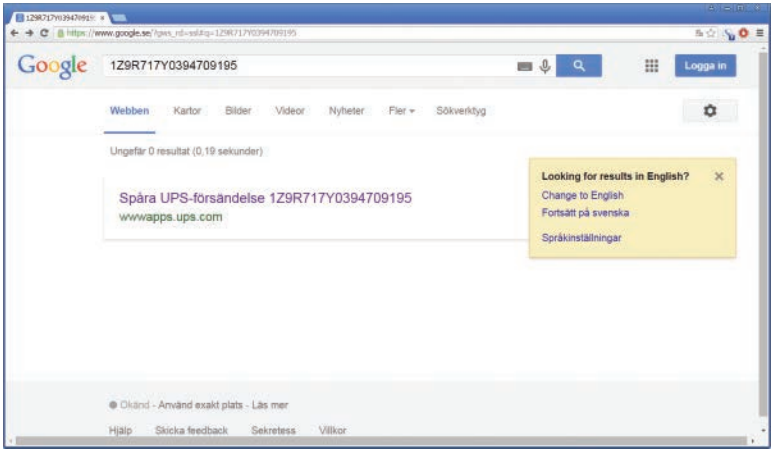


Figure 0.4 Google automatically recognizes a UPS tracking number and offers to track the package.

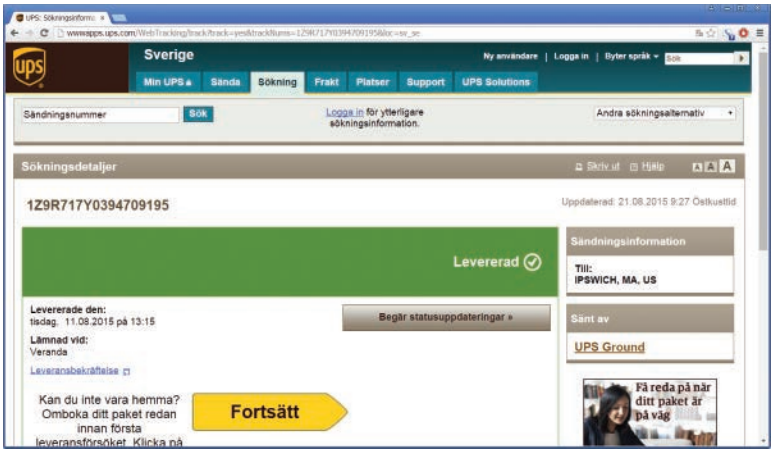


Figure 0.5 Google automatically displays the UPS tracking page in the language of the user's browser—much easier than going through UPS.com.

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

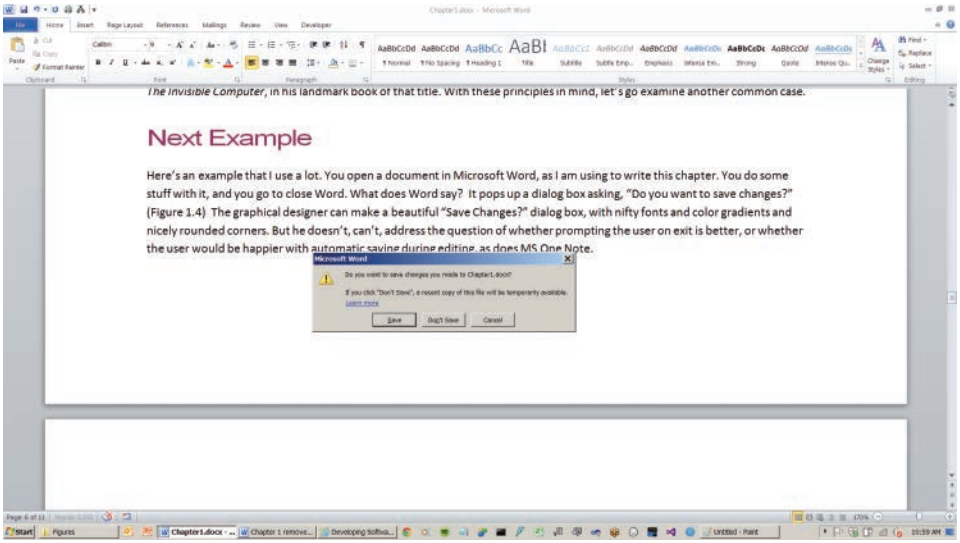


Figure 0.6 Microsoft Word, prompting the user to save changes. Why not save automatically, as does Microsoft OneNote?

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.

This page intentionally left blank

CHAPTER 8

CASE STUDY: COMMUTER RAIL MOBILE APP

Now that you've seen each UX design step individually, let's see how they all work together. In this chapter and the next, we'll do a case study on a real-world problem, applying our new skills and techniques end to end.

We'll start with a mobile phone app aimed at easing the daily grind for commuter rail riders. We'll focus on Boston's system, because we can easily find specific details and riders to work with. When we start applying what we've learned, you'll see that we can make things a whole lot better than they currently are.

Pity the Poor Commuter

The commuter rail system in the Boston area is owned by a state agency officially named the Massachusetts Bay Transportation Authority (MBTA), universally called "the T." It serves approximately 130,000 riders each weekday over ten lines, 394 miles of track, and 127 stations. It's the third-largest such system in the United States, behind New York City and Chicago, tied with Philadelphia.

The T got absolutely hammered by record snowfall in the winter of 2015. Trains were canceled, rescheduled, and canceled again, while cold riders shivered on windswept station platforms, fuming, "Where's Mussolini when we need him?" The director of the system resigned under fire, "for personal reasons." (I say she jumped while being pushed.)[1]

We can't make the trains run on time. But we can tell the riders when they actually are running—the true up-to-the-minute performance, not the wishful thinking of a paper schedule printed months before. We can smooth out our riders' lives. They'll know when to leave their homes or workplaces for the station, they won't waste time trying to catch a train that isn't running, and they'll be able to schedule their lives again.

The second need of commuter rail passengers is help with buying their tickets. They have to wait in line at the very few staffed ticket windows (fewer when the weather is bad, as the government employees stay home) or use the few available vending machines, which are always broken anyway. This extends their already-annoying commute and causes them stress. It would be great if we could make that go away too.

Can we make use of our new skills and knowledge, the steps we've seen in this book, to make their lives easier with a well-designed mobile app?

Current State of the Art

The MBTA already has a mobile app for buying tickets. The T made a great fuss over its introduction in late 2012, as the first such app in the nation. When we start examining it, we see that it doesn't help *our* users solve their own problems anywhere near as well as it could and should.

The home screen (Figure 8.1) is terrible. Most of its area is wasted. The top third shows what some graphic designer probably considered a pretty picture. The designers probably think of it as "our branding." The bottom third is completely blank.

1. The memory of that rail fiasco still burns in the region, even as I write these words a year later. As Howie Carr wrote in the February 24, 2016, *Boston Herald,* "The only way to stop the [Donald] Trump train now may be to turn it over to the MBTA."

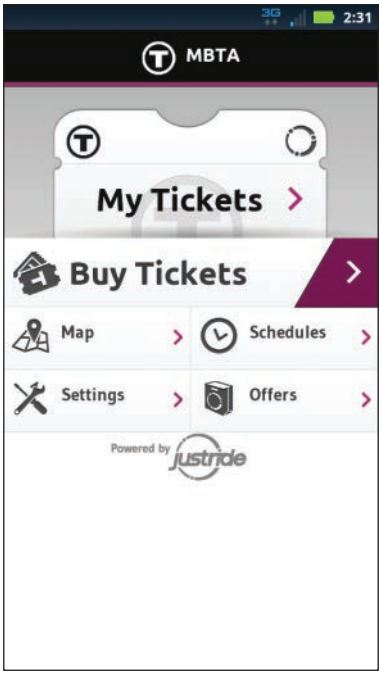


Figure 8.1 MBTA mobile app home screen with wasted space and no functionality at all.

We can't *do* anything at all on the home screen. We have to leave it to accomplish anything—view a schedule, buy a ticket, see alerts that might affect our commute. They're squandering the most precious resource in any mobile app, a resource that could have helped us accomplish something that we actually cared about. Instead, they've given us a picture, a blank space that sort of balances it visually, and no functionality whatsoever. I suspect it's the art major's revenge for all those jokes ending, "Would you like fries with that?"

The purchasing and displaying of a ticket works not too badly, once you navigate to it from the home screen. Figure 8.2 shows the process. We select the stations by typing in the first few letters, and auto-complete (good) narrows the list. The app retains our most recent selection at the top of the list (also good), because almost everyone on commuter rail uses the same stations repeatedly (Figure 8.2a). We type in our credit card number, which it also remembers for the next time (also good), and the transaction is consummated (Figure 8.2b). When we're ready to ride, we tap a button to activate the ticket. It then flashes the color code of the day so the conductor knows it's valid. It also has a button that shows a bar code for readers that conductors might someday start carrying (Figure 8.2c).

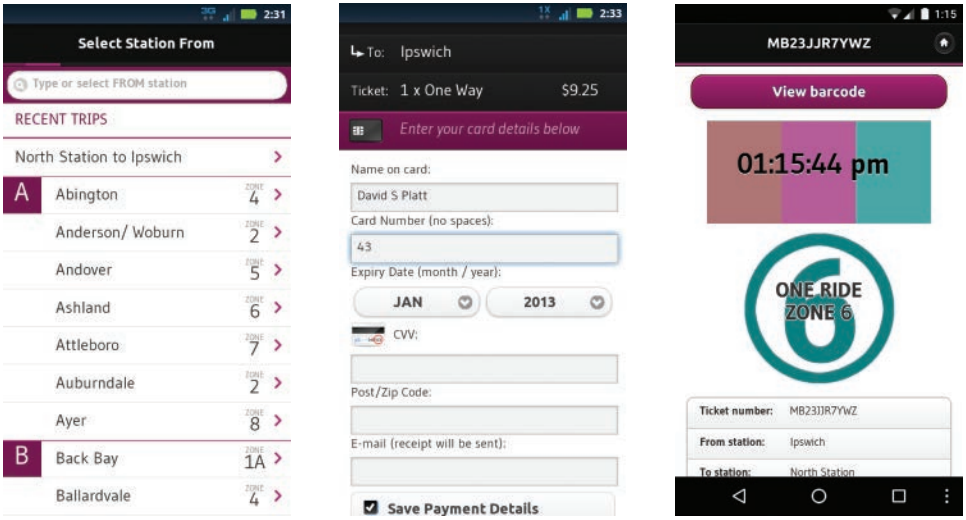


Figure 8.2 MBTA mobile app ticket purchase—not too bad.

Because it works not too badly once you get to it, we won't discuss the ticket purchase portion of this app very much. However, even with all the data it remembers, we still have to type in our credit card CVV number every time. This app is most commonly used by occasional riders, who will not have that memorized—monthly pass buyers with auto-renewal don't need it. Occasional users now have to juggle their phones and wallets and credit cards in a crowded public place, which is uncomfortable, or think ahead, which no human in the universe ever does about anything. Adding an Instant Purchase button for each recent trip at the top of Figure 8.2a, similar to Amazon's 1-Click purchase, would smooth this out even more, especially since commuters almost always travel the same route.

The app fails miserably at the greater need of commuter rail riders: accurate and timely schedule information. Again, the home screen contains no information whatsoever about schedules. If we want to see a schedule, we have to go through three steps: tapping Schedules on the home page, which then takes us to a screen where we are offered the choice between Schedules and Alerts (Figure 8.3a). The app is saying, "I know you selected Schedules, but did you really want Schedules?" After tapping Schedules again, we have to choose the line for which we want the schedule (Figure 8.3b). Only then will it show us a schedule in an ugly format that is very hard to read (Figure 8.3c).

Alerts, whatever they might be, do not appear as an option on the home page. We have to somehow intuit their existence and go digging—tap Schedules, then tap Alerts (Figure 8.3a), then look at our line to see if it has any (Figure 8.4a). The green check mark would seem to indicate that everything is OK, but despite this indicator, the Lowell line has one Upcoming alert

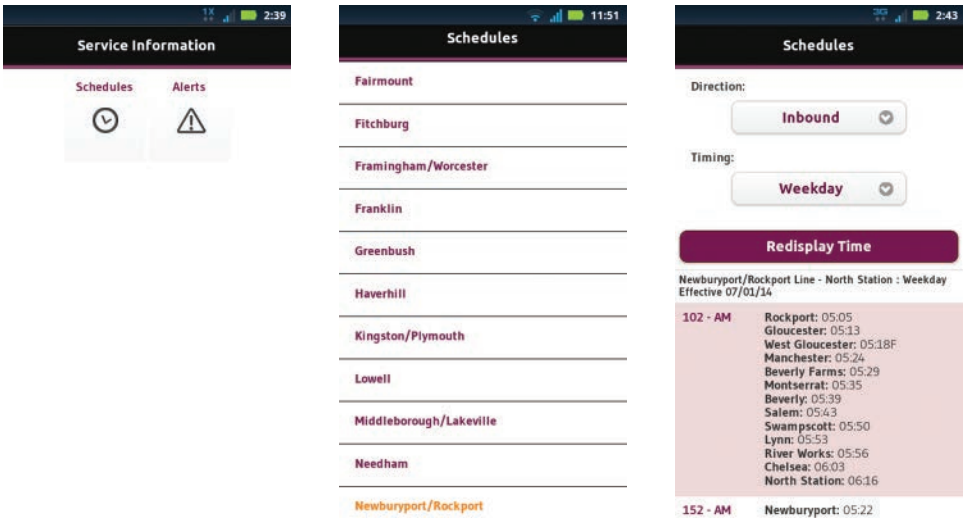


Figure 8.3 Schedules are difficult to find, then difficult to read once we've found them.

and one Ongoing alert. (What the hell is the difference between Upcoming and Ongoing? I can't tell, and when I look at the contents of each, they appear to be identical.)

If something is important enough to be called an alert ("an alarm or other signal of danger"), it surely shouldn't be buried four screens deep, should it? And isn't an "ongoing alert" a contradiction in terms? Once we look at the alert, we can see that it often impacts the schedule; note the two canceled trains (Figure 8.4b). Burying this information four levels deep ensures that no one will ever see it—exactly the opposite of what alerts are for.

The developers of the schedule portion of this app did not apply the skills that they (sort of) demonstrated in the ticket purchase portion. They didn't work from the users' perspective. They just took their paper schedules and tossed them into an app, with the awful results you would expect from such an unthinking approach.

The user has to do far more work than she should have to. The app doesn't make use of the knowledge it has about the user, or about the repetitive nature of the commuter rail relationship. The developers are saying, "Hey, it's *your* job to do all this work." Maybe that attitude was acceptable a decade ago, but it sure isn't today. If a student of mine turned in something like this, I'd flunk him so fast he'd switch his major to English.

We can do a whole lot better by following the Platt UX Protocol, putting ourselves in the users' shoes. Once we think about who the users really are and what these users actually need, we can select the items of information most relevant to them, here and now, and present them clearly and easily. That will turn this commuter rail app from a brick into an indispensable everyday aid.

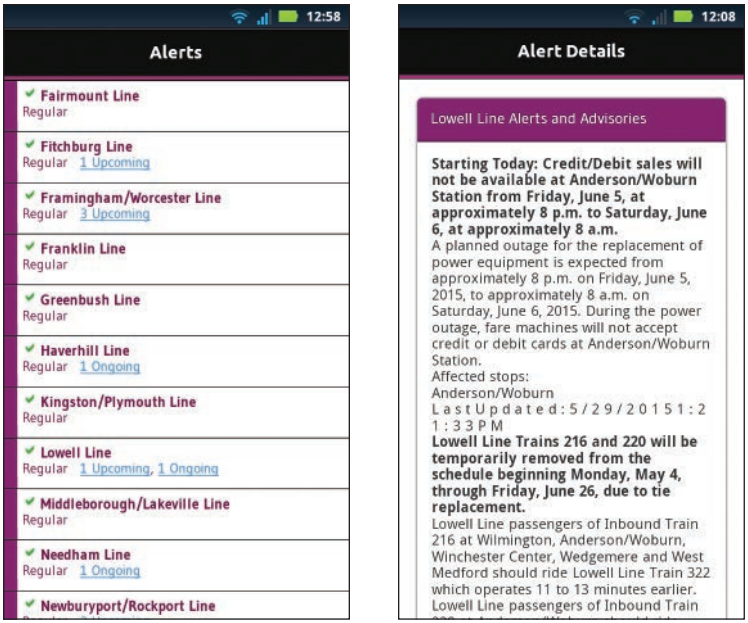


Figure 8.4 We have to select the alert for our line.

Step 1: Who?

Your first impulse, of course, is to bring up Xamarin and start dragging and dropping components onto the design surface. I hope this book has shown you that that's wrong. That's what the MBTA's developers probably did when they wrote this app, and look where it got them: publicly hammered in this book. Let's run the Platt UX Protocol on it and see where it takes us.

To begin: Who rides the commuter rail in Boston? What's our potential user population? Your immediate response is to shout, "Everyone; it's public transport." But you should know better than that by now.

The basic demographic information is easy to find. A quick search on the MBTA's Web site for the term *advertising* gave me the contact info for the company that manages the advertisements on the T's vehicles and stations. I emailed them for a customer kit about advertising on commuter rail and received it within the hour. It contained the basic demographics we need to get started. You can see an excerpt in Figure 8.5.

Commuter rail riders are just about evenly divided in gender, 52% male and 48% female. Commuter rail passengers skew older than the general populace. The largest single cohort is age 45 to 54, constituting 30% of ridership. Seventy-three percent of riders are age 35 or older. We could speculate that's because younger people don't move out to the suburbs until they

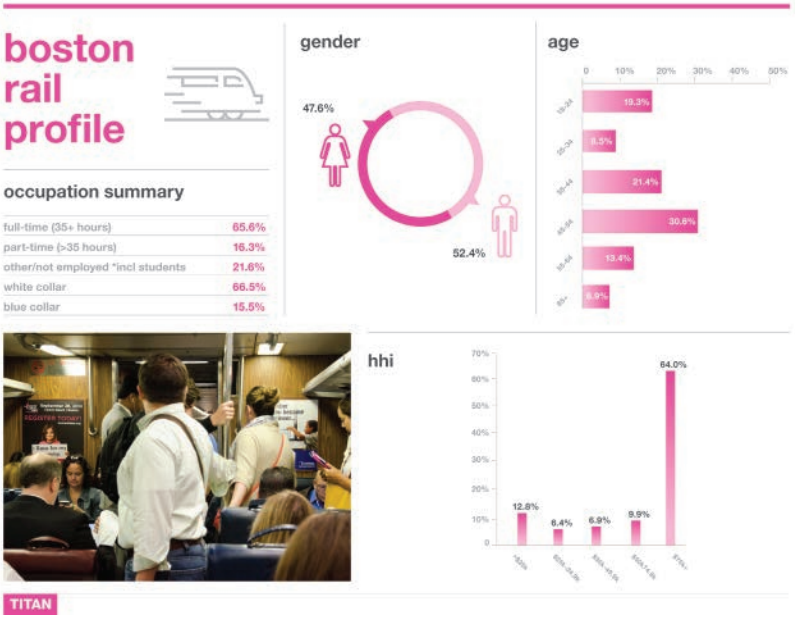


Figure 8.5 Media kit with commuter rail demographics. (Courtesy of MBTA)

marry and need room for the second kid and the swing set and the golden retriever. So no, it's not everyone. Our user population contains very few kids, and not many millennials. Our users didn't grow up with smartphones. They are not the kinds of people who say, "Way cool" when they see, for example, Snapchat. They will always speak geek with an accent.

Well, if they're that old, do they use smartphones at all? Are we barking up the wrong tree completely with the notion of any smartphone app? Fortunately, no. A quick Google search finds the *Boston Globe*'s coverage of the MBTA ticketing app initial rollout, reporting that 76% of commuter rail riders carry smartphones. That was written in 2012, and I doubt the percentage has gone down since then. We have market penetration, if we can provide customers with what they need.

Now that we have basic information about our user population—evenly split in gender, but skewing older and hence less technophilic—let's create personas to communicate that information to our development team. I worked up two personas, one who travels every workday and one who travels occasionally.

Claire (Figure 8.6) lives in Salem, Massachusetts. (You can download her full persona from this book's Web site.) She's 42, divorced, with three kids (16, 14, nine) still at home. She works at Massachusetts General Hospital as a respiratory therapy aide, on a regular eight-to-four weekday schedule. She depends on commuter rail to get herself to work every day. Commuting on buses and the subway would take her three times as long, and those kids of hers don't leave her a free

Figure 8.6 Claire, who rides the MBTA commuter rail every day to work in Boston.

minute. Driving in Boston's killer traffic at those peak times would be awful. And parking downtown costs $30 a day, which would take a huge bite out of her $18-per-hour pay.

Claire drives to the Salem commuter rail station every day and takes the 6:43 a.m. train to North Station, arriving at 7:18. She then walks or takes a shuttle bus to Mass General. She likes to take the 4:20 p.m. train home, arriving at 4:53, but often she can't get out from work in time. Then she takes the 4:45 train, arriving at 5:16. She knows the train times very well because she takes the train every day.

Her biggest headache is when the train schedule gets disrupted. It's not just snow—lightning, construction, the old equipment breaking down, vehicle accidents at a grade crossing; anything can mess it up. She doesn't know when she has to get to the station, or when to tell her family she'll be home.

Claire pays her fare with a monthly pass. Her employer kicks in 25% of it as a fringe benefit. Her phone is a four-year-old Android on a T-Mobile family plan because it's cheap.

For our second persona, I need to tell you of a Boston legend. A hypothetical man on Boston-area transit is always named Charlie, harking back to the Kingston Trio's 1959 smash hit "Charlie on the MTA," a song about a rider who never returned. Google it; it's good.

Figure 8.7 Charlie, who rides the MBTA commuter rail into Boston 25 or 30 times per year. (Photo by redjar on Flickr)

Charlie (Figure 8.7) lives in Ipswich, Massachusetts, on the same rail line as Claire, another half-hour farther out. (You can also download his persona from this book's Web site.) He is 56 and married, one child graduated from college and another halfway through, but he hasn't downsized his house yet. He is a higher-end computer consultant who sometimes works with clients in downtown Boston.

Charlie makes 25 or 30 trips per year on commuter rail. They tend to come in bunches, perhaps four trips this week and three the next, followed by several months with none. His billing rate is high enough that he could afford to drive and park in Boston, but the traffic drives him batty. He'd have to leave his house at 5:00 a.m. to beat it, and his clients don't usually like to start that early. He's happy to ride the train, drink coffee, listen to classical music on his iPod, and review the upcoming day's material on his MacBook Air. On the way home, he turns off his phone and reads a book, sometimes drinking a beer (which the conductors wink at if he keeps it in a paper bag).

He drives from his house to the commuter rail station in Ipswich. He usually takes the 7:13 a.m. train in, arriving at North Station at 8:10. He then walks or takes the subway to his client. His default return trip leaves at 5:15 p.m., arriving at 6:07. But sometimes he finishes earlier and catches an earlier train. And sometimes he has to work later, or stay in town to socialize with

clients, and catches a later train. He can't remember the train times from one trip to the next because he doesn't take trains frequently enough.

Charlie usually has to buy his tickets aboard the train for cash. There's no store near his house that sells them, and no vending machine at his station. That means he has to remember to stop at the ATM and take out a $20 bill every day for his round trip. The conductor punches out a paper ticket and gives him $1.50 change. (How nineteenth century.) He has to keep the paper receipt for his expenses and manually enter it into Quicken. He could buy several from the ticket window at the main station when he gets in, but waiting in that line at rush hour is way more trouble than it's worth.

Charlie always carries the latest iPhone from Apple. He justifies the cost by saying he has to project a tech-savvy image to his clients. But he really just likes Apple stuff. He won't camp out on the sidewalk the night before a new release, but he will pre-order it from Apple's Web site.

Note that neither of our personas is a tourist or a foreign visitor, someone new to Boston. If we were dealing with buses and subways, we would definitely want to include them. But commuter rail reaches a very different audience—more homogeneous, more suburban, more repetitive. (It also makes this example much easier.)

Step 2: What (and When, and Where, and Why)?

What problems do Claire and Charlie need to solve, and what would they consider to be the characteristics of a good solution?

Riding the commuter rail is a repetitive kind of thing. Almost everyone travels from an outlying station into the city in the morning and returns to that same station in the evening. Our app needs to recognize and cater to this repetitive behavior.

It's rare that a user will change stations, even more rare that he will change the line on which he travels. It happens occasionally—a guy will stay over at his girlfriend's house and take a different train the next day—but not often. Our app needs to be able to handle changes, and make them as easy as possible on that user, but not at the cost of complicating the much more common case of "same old, same old."

What do commuter rail riders need? More than anything else, they need to know when the trains will actually leave the station. Commuter rail is not like the subway or bus that comes along every few minutes. Commuter rail trains run every half an hour during peak times, and less frequently (an hour or two, sometimes longer) outside them. If you miss one, you might wait quite a while for the next one, and the stations (particularly inbound) are not at all comfortable.

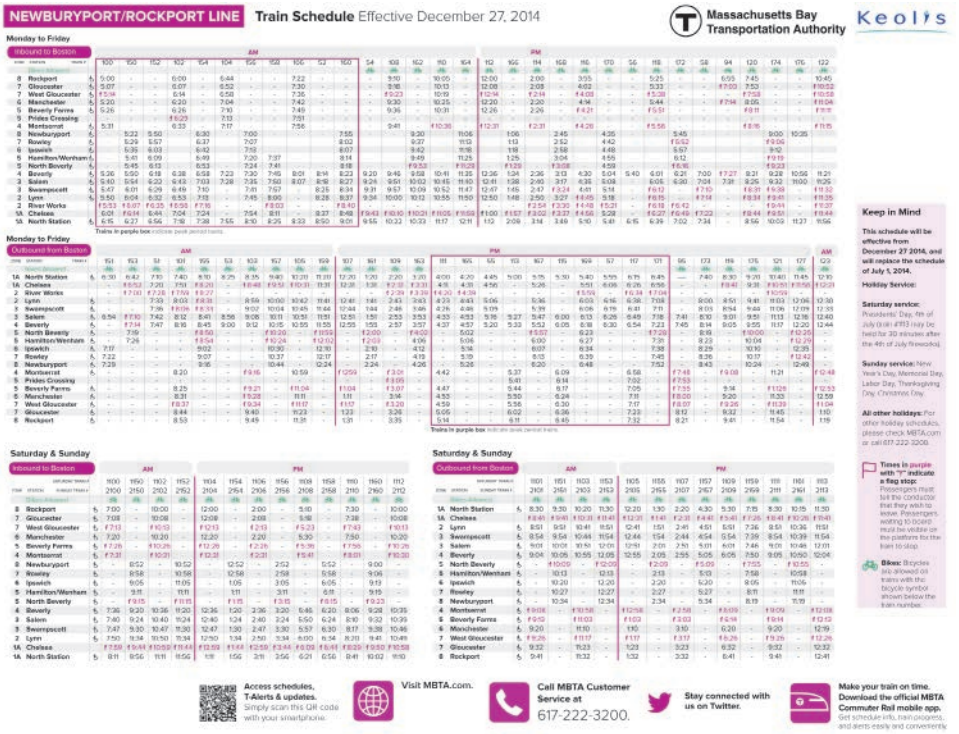


Figure 8.8 Confusing, inconvenient printed timetable. (Courtesy of MBTA)

How do riders know the schedule? Historically, the railroad has issued a printed timetable for each line (Figure 8.8). Riders have to pick up a copy, usually at the main station (where they've got scads for every line but yours), carry it with them, and then remember where they've put it and how to refold it. It's inconvenient to read, because it covers all stations on a train line and you have to pick your specific station out of it. Charlie does this every time he travels, and it's a pain. "The signal-to-noise ratio is low," he says. (Geek.)

It is not unusual that the rail schedule gets disrupted by weather or accidents or mechanical difficulties. Sometimes the delay affects just one line (a train on the Fitchburg line hits a truck), sometimes it's all of them (a presidential visit snarls the entire downtown). Riders need to know about this so they can take an earlier train to work, or drive in if they have to, or pretend to work from home, or say, "To hell with it" and fake a case of smallpox to take a sick day. Obviously the paper schedule can't tell us this.

The MBTA Web site could, but it has to cover an enormous array of topics—bus, subway, boat, and so on. It's difficult to find the specific current information that you need even with a

full-size Web browser on a PC. And we often don't have access to a PC, for example, while we're waiting on the suburban platform in blowing snow for a train that isn't there. It's almost impossible to use the MBTA site on the limited area of a mobile phone.

Riders also need help buying their tickets. A quick Google search finds that 57% buy monthly passes. The rest have a quandary. Most trips originate in the suburbs. Very few suburban stations today have staffed ticket windows, or even vending machines. Once in a while a nearby store will sell tickets as a convenience, but that's increasingly rare. That was traditionally the domain of the local tobacco shop, where the riders would also pick up a newspaper and a pack of smokes for the ride. Those shops are just about all out of business today, along with the newspapers and the smokers. So most non-monthly-pass riders have to pay cash to the conductor on board. It would be nice to be able to buy tickets on demand, with credit cards.

Now that I had some handle on the problems that Claire and Charlie need to solve, I needed to ask other riders what they thought about the app. I needed to do this quickly, so I went to my local commuter rail station on a weekday morning and interviewed as many people as I could. Here's what I said to them:

- Tell me about your ride today.
- What burns you the most about it?
- How do you pay for your train ride?
- What kind of smartphone do you have?

Note that I started with open-ended questions and moved toward more specific ones. Above all, I needed these interviews to be quick. The riders start gathering at the station only about ten minutes before the train arrives, and I needed to talk to as many as I could.

I found that most of the riders complained about not enough trains, because of cancellations. (We can't really help them with that one.) Their second complaint, almost universal, was not knowing when the trains were running. They were angriest about the times the MBTA gave out wrong information. The Web site would say that a train was on time, the riders would go to the station, and the train didn't come. They wait 15 minutes, half an hour, in their cars with the engines running; still no train. The electronic sign at the station claims that the train's on time, but in reality it's been canceled and the one behind it is two hours late and packed to the gills. While our app can provide the users with the information that they need, efficiently and in a pleasing format, we can't repeal the zeroth law of computer science: "Garbage in, garbage out." Our app is only as good as the information that the MBTA gives us to feed to it.

Hardly anybody talked about buying tickets. That wasn't on their minds when I did this research. It might increase in importance as schedules returned to more normal conditions, but the riders weren't thinking about ticket purchases when I asked them.

Riders look at schedules more often than they deal with tickets. Charlie buys a round-trip ticket once per day when he travels. Claire sets up a monthly pass with auto-renewal and then doesn't touch it again. They display their tickets to the conductor once per trip, or twice per day, often not even that when the train is so crowded the conductor can't get through to check. But they look at the schedule a lot: at least once or twice the night before, the same again in the morning, and the same again in the afternoon. Charlie will probably check it more often per day than Claire, who can settle into a routine. But they both need to know about any service disruptions.

So here's what our users, Charlie and Claire, need:

- Good, up-to-the-minute schedule info, including any changes
- Good, easy ticket purchase and display
- And all of it easy, easy, easy to use

Now that I knew what users needed, I started writing it up in the form of stories so that the geeks who would code this app can understand. Here's what I wrote:

Story 1

Claire is at home in the evening, getting ready for work tomorrow. She doesn't know what's up with that stupid commuter rail schedule, due to all the snow they've had lately. She needs to know when the trains are running tomorrow, so she can know when to set her alarm for. She pulls out her Android phone, taps our app. The app sees that it's evening and that she's currently located in the suburbs. It knows from the pass she's purchased which stations she travels from and to. So it automatically comes up showing the trains inbound from that station for tomorrow morning. (She can change that with a few taps in case it's wrong, but it usually isn't.) The app says that everything's currently on schedule for tomorrow, but Claire doesn't believe that for a microsecond. She sighs and wishes she could get a job locally and not have to deal with this damn commute. But she's got seniority at her current job, her kids are headed toward college—she's stuck with it for the foreseeable future. She sets her alarm clock early anyway and goes to bed.

Story 2

Charlie is at work in downtown Boston and his client invites him to stay in town for dinner. Of course he'd love to socialize with his client; that's how he often hears about new business coming down the pipe. He needs to know the last trains of the day, so he knows when he has to leave his social engagement. He pulls out his latest iPhone (his customer's eyes widen with longing) and taps our app. The app sees that it's late afternoon, and its current location is in the city. So it automatically comes up with the outbound trains highlighted. It knows from the ticket he purchased this morning where he's traveling to, so it shows the times for that line.

There's a train leaving North Station at 7:40 p.m.; that's probably too early. He'd have to eat too fast and probably wouldn't get around to the business discussion over coffee and brandy. The next one's at 9:20, so he has time for a good outing. But the one after that leaves at 11:45. If he misses the 9:20, he'll have to sit in North Station for two and a half hours—no fun at all. And if he misses the 11:45 train, he'll have to take a $100 cab ride out to Ipswich, or sleep on the station benches. Charlie knows what his parameters are and goes off to his dinner meeting with confidence.

Story 3

Claire wakes up in the morning and turns on the coffeemaker. She looks out the window and sees some new snow. Damn! She pulls her phone off the charger, taps our app, checks to see if the train schedule has gotten even more screwed up. Double damn! It has! They canceled her regular train, but there's an earlier one (actually an even earlier one that got delayed) that she can still grab if she hustles. She yells to her older daughter that she'll have to get the younger ones out to the school bus, throws on her clothes, and runs out the door cursing the politicians who screwed up the transport network. But she makes her train, keeps her job, doesn't even get her pay docked. She does have to pick up the load for employees who couldn't get in until noon because they didn't have our great app to warn them when their trains got screwed up. Fortunately, many of the patients got stranded, too, and missed their appointments, so the workload wasn't quite as bad as it might have been. The outbound trains are also messed up, but at least she can see which ones are running. She'll order pizza delivery for dinner tonight.

Story 4

Charlie needs to buy a ticket every time he takes the train in. There isn't a ticket outlet near his stop. He used to need a $20 bill every day to buy it on board from the conductor. But now Charlie takes out his phone, taps our app, and buys a ticket, which he displays to the conductor. The bill goes to his credit card and appears magically under the "Travel" category when he downloads his transactions into Quicken. Charlie's accountant is happy. Charlie is happy. The MBTA bean counters, who want to go cashless as soon as the politicians will let them, are happy too. The world is a better place all around.

Step 3: How?

Now that we know who our users are and what they need to do, we'll start addressing how they can do it. We'll use Balsamiq to make some quick mockups based on our initial research, show them to users, and get feedback. We won't spend any time at all making them pretty. As I've said throughout this book, the key point at this stage is to iterate quickly. Polishing the cannonball is counterproductive.

The operation that users do most often is to check schedules, both promised and actual. Four or five times per day is not unusual. The more the users perceive that the schedule is likely to change, the more frequently they'll check it. So the more critical this piece is, the more critical it becomes. It's important to get it right.

The next most common thing users do is to display a ticket to the conductor. Buying is less common. Claire sets it up once in her life and the pass continues forever. Charlie does it on days he takes the train in, usually buying a round-trip, which means he does this just once per day.

We want to minimize the number of touches that the user has to make. The original MBTA app does not do that, and it never seems to have occurred to its developers that they should try. Let's take the app's knowledge of the repetitive patterns of most commuter rail users and leverage it to the max.

I made my best initial guesses based on what my live users had told me, and on what Claire and Charlie said when they disturbed my dreams at night. Figure 8.9 shows my first mockups.

I started by trying to fit everything onto just one page. In direct contrast to the MBTA app where the home page does nothing at all, this home page does everything. We don't need any

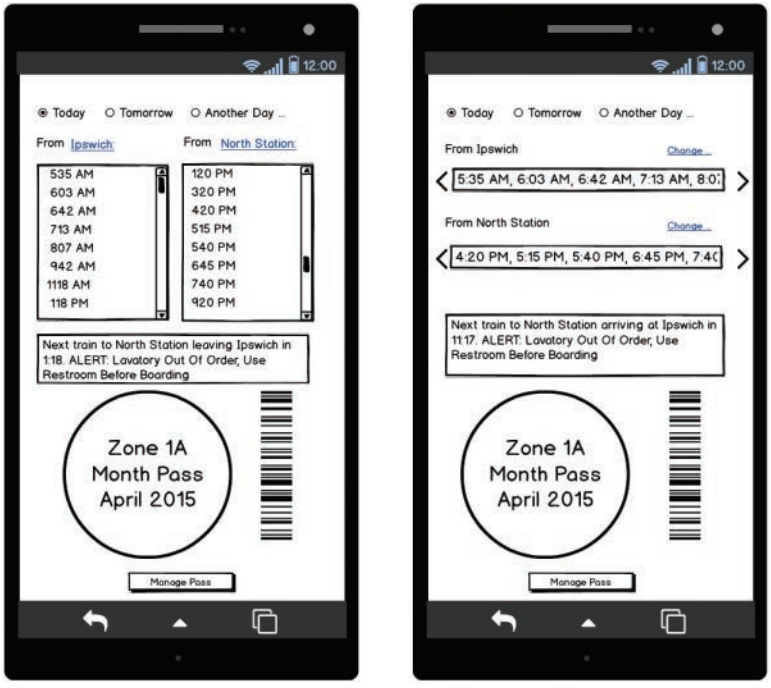


Figure 8.9 Two ideas for our first iteration of the MBTA mobile app.

navigation because we never have to go anywhere else for schedules or alerts or ticket display. Purchasing a ticket or pass requires another screen, for which we provide a button, but as I said before, this is much less common.

The first thing we see at the top is a line of three radio buttons. Users check schedules at three main times during the day: in the morning before they head out from home to the station, in the afternoon before they head out from work to the station, and in the evening at home, to see what's up for the next day. So perhaps two times out of three they want to see the current day's schedule, but in the evening they want to see the next day's schedule. The app automatically figures out which one to show, based on the time of day and the user's cell-level location. (That's precise enough for our needs and draws no extra power.) The selected radio button displays the current choice. If the user sees yesterday and needs today, or vice versa, she just taps the radio button that she wants. If she needs to see another day, perhaps on the weekend to go into town for a show, picking "Another Day" brings up a date picker to specify that date.

Next, we see the inbound and outbound trains. Here's the key improvement over the original app: because the app knows the ticket that the user bought, it knows which trains and which times to show. Claire bought a pass originating in Salem. Charlie's tickets originated in Ipswich. Both specified North Station as their destination. The app uses this knowledge to populate the labels above the list boxes and the train times within them, with the next available train at the top. Each station has a link to change it if needed. That feature likely will not get used much, but telemetry (next section) will tell us if we're wrong.

I tried two different approaches to displaying the train times. Figure 8.9a uses list boxes, displaying the train times in a vertical arrangement that the user is probably familiar with. In Figure 8.9b, I saved space by putting train times in a horizontal line. The app automatically places the next available train as the first in the list. Either scroll bars or horizontal arrows indicate to the user that she can scroll them if she needs to see more trains than fit in that space.

The app doesn't show the trains' arrival times. I figured most commuters probably wouldn't care. They know approximately how long their train ride takes and can therefore easily extrapolate. Plus, arrival time is often affected by factors beyond the rider's control, so it doesn't much matter what the schedule says. If needed, we could make train arrival information accessible with a tap on a particular train. I wonder how many users would say they need that.

Below the train displays is a text box. This carries a countdown clock, showing the time to the next train inbound (if the user is in the suburbs) or outbound (if the user is in the city). The users won't have to do mental subtraction to see if they can make that train. This text box also shows any alerts pertaining to this line or their station. Users won't have to tap down three levels to find out if there's anything they care about.

Below all this is the monthly pass or ticket display. Claire's automatically updates every day to show the correct color for validity. Charlie will have to explicitly activate one, using a button (not shown).

The main drawback with this layout is that it's a little bit cramped. Ideally it won't bother our user population much. In a reader app that they'd use for half an hour or more at a time, that would be a deal breaker, but they'll look at this app for no more than ten or 20 seconds at a pop. The main criterion was for users to be able to pick out immediately the data that they need. I carefully selected everything users needed (or so I thought), and nothing that they didn't (ditto).

The point is not that this layout is perfect, or even very good. This is my first draft. The point is that it didn't take long to dummy up these layouts *and try them on actual users*, to see what they liked and what they didn't like. To that trial we now turn our attention.

Step 4: Try It Out

I next needed to ask a number of users if they'd look at my app. I emailed my student list, asking for anyone who rode commuter rail and wouldn't mind taking a look. Obviously, this selection has a certain amount of skew. My students tend to be younger than the average rider, although since I teach in the continuing education division, they're not undergrads. They tend to be better educated than the majority of commuter rail users, although here in the Boston area we have more college grads than most places. All of them did explicitly identify themselves as commuter rail users, either current or recent, so they do know what they themselves need.

If I were going to invest major money in this mobile app, I'd need real users. I'd probably hand out cards at the commuter rail station offering $20 to any rider who'd do a half-hour Skype call to look at the app. I'd get as many riders as I wanted. Again, at this stage, I need to move quickly, so I'm thinking fewer test users rather than more. I decided to go with Steve Krug's suggestion of three. If all three of them like something, it's probably pretty good, and if all three of them hate something, it's probably pretty bad.

I arranged Skype calls with three of my former students who are regular commuter rail riders. They know me and aren't afraid to call 'em as they see 'em. Once we got our Skype session established, we chatted a little bit about their current activities to break the ice and get the conversation flowing, and then I started with open-ended prompts such as "Tell me about your commute." Then I read them the blurb from Chapter 4, how "We're not testing you. You cannot possibly make a mistake here. We're checking how well *our* software fits what you're trying to do."

I used Story 3 above, the one about Claire waking up in the morning. I figured it was the most critical one. I read it to the test subject, substituting his name for hers: "Imagine, John, that you've woken up at five in the morning to get ready for your day. You're sipping your first cup of coffee and you look out the window and damn, it's snowed six inches. You figure you had better check and see how the trains are running. You pull out your phone, tap the icon [now I share the screen in Balsamiq], and here's what you see. What do you do now?"

One of the users started talking out loud as she thought, which is helpful. I had to prompt the other two to get them started: "John, it would be really helpful if you could maybe speak out loud as you are thinking. You're obviously figuring something out; could you maybe tell me about it?" and so on.

I won't repeat the conversations verbatim, but here is a summary of what I found. The first thing every user noticed was the set of radio buttons showing Today/Tomorrow/Other. I thought those buttons were important, but the users didn't know what to make of them. I said, "OK, no problem, ignore them for now and continue. What are you thinking?" If these buttons had been a real problem, if they were too distracting to continue this test, I could have easily removed them from the Balsamiq mockup, but these users were happy enough continuing on.

They started looking at the departure times in the left list box. All of them said that they would find the one that they wanted to take and tap on it, expecting to see information about that train. I had expected the departure time to be all the information that the user needed, but these users didn't see it that way. They wanted to see the arrival time, and they wanted some sort of indicator to know if it really was on time.

Two of the three noticed the countdown timer and alerts box and understood what it was. It's a little harder to notice here in the static mockup, because in a real app you would see the time counting down and that would give a clue to its purpose. The third subject didn't notice it at first but understood it immediately when I asked him to look at it.

This was all great feedback. I thought initially that the users would not care about the arrival time, because they already knew how long the train ride takes. That wasn't the case. They all wanted to see it when making their choice of trains. They didn't want to have to add it mentally; they wanted that information in tandem with the departure times, perhaps to go to the same place in their brains. Also, they said that they didn't care about seeing the evening return trains at this point. They'd look at return trains in the afternoon when they were thinking of heading home. The return trains weren't horribly distracting, but they weren't the slightest help right then either, and their real estate could be used for something else that users cared about.

They all preferred the vertical arrangement of trains in Figure 8.9a to the horizontal line in Figure 8.9b. That's how they're shown in the timetable, that's how they're shown on the monitor in the station, and the testers found it confusing to view them otherwise. No one liked the horizontal line. OK, no problem; I learned something else.

They all said that the layout was too cramped, making it too hard to pick out the thing that they needed. In my zeal to get rid of all the navigation, I had crammed too much into one place and made the user do work of a different sort. Clearly, I hadn't found the optimax yet.

Was I angry that a bunch of idiots couldn't see the brilliance of the design? Did I scream to the heavens demanding to know how I had gotten such a brain-damaged set of test users? On the

contrary. They told me things I didn't know. They made this app better. They made me smarter. And now you too, I hope.

So back I went to Balsamiq. I split this information into two different screens, with the easiest possible navigation between them. The hamburger control is popular in mobile apps, but this app has such a small amount of navigation that the extra tap it would require (tap the hamburger to see the choices, then tap the choice you want) would have been cumbersome. Instead, I used a tab control for its visible navigation. The amount of screen real estate it consumes is small compared to the benefit of instant and obvious access to this small number of choices. I put the schedule on one tab and the ticket or pass on another. You can see it in Figure 8.10.

The test users liked this one much better. They all said it was way easier to find the times that they wanted. They didn't have to tap on anything to see the arrival times. "That table, that's how they're shown on the monitor in the station, so it's really familiar," said one, and the others

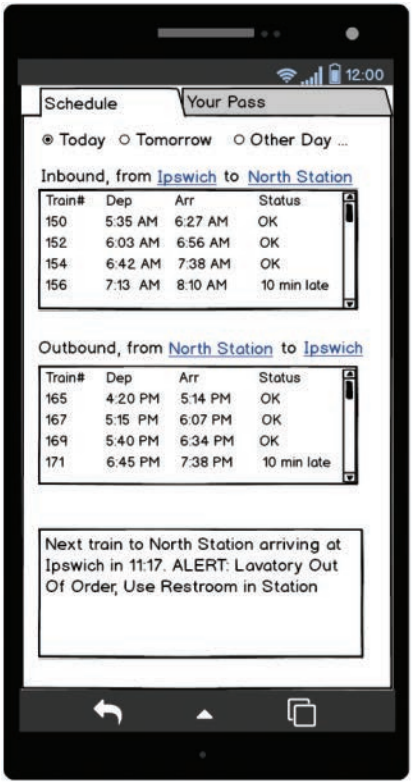


Figure 8.10 Second iteration of the MBTA mobile app, incorporating feedback from the first attempt.

agreed when I asked them. "We never use the train number at all. We always think of them as departure time, like the 8:57 train. So you might as well get rid of the column." The other users, when I asked them, said that no, they never used it either and agreed we might as well get rid of it.

They loved the tab control for navigation. I thought it might be more of a PC-based idiom, not so much for mobile phones, but they all grabbed onto it right away. Maybe it's the fact that the test users are older and more used to PCs, but then so are the actual commuter rail riders. The only disagreement was over where the tabs should go. At the top, as I originally showed? One tester wanted them at the bottom so he could select the tab with the thumb of his hand that held the phone, convenient when the train got crowded. I decided to keep them on top for now. It's not a huge deal one way or the other. That choice can easily be made in further refinements, and if we really need it both ways, we could make it configurable.

Again I encountered the desire for separation of morning inbound train times and afternoon outbound train times. Every test user said, "Better, but I still have to tell one table from another. It's easier than the first one you showed us, but why not have each in a separate tab?"

Finally, I asked each tester about the alert and countdown timer box. One said that the countdown timer was extremely important, as it told her when she had to run to make her next train. She wanted it on top for that reason. It would be nice to have the track number on it as well, if we were able to get that. That's the sort of information you will get only from conversations with your users. Telemetry can't tell you that.

Again I went back and made the changes suggested by the test users. They had said they were happy with tab navigation. They can see all the choices at once, and it takes only one tap to select any of them. They didn't care about having inbound and outbound on the same screen, so I decided to place each on a separate tab.

I moved the countdown timer to the top. Only one test user had specifically requested that, but the others all said they liked it when they saw this design.

I used the extra space on the tab to show more trains, and to make the font larger for easier reading. Think about the demographics. Half of the users are age 45 and over. The developer community skews much younger than this. It is common for them to dismiss the need for larger type as a special need; it's not the main thing, so we'll get to it when we can—definitely not version 1, probably not version 2 either, maybe version 3, or then again maybe not. But age-related presbyopia (farsightedness) begins around age 40. At least half of our user base would appreciate something easier to read. You can't call half the user population an obscure special case. They need relaxed-fit text as they need relaxed-fit jeans. They probably aren't happy about needing it, but they'd sure appreciate having it. The third tab made reading easier for everyone. The users' requirements are now working together toward the optimal solution. The result is shown in Figure 8.11.

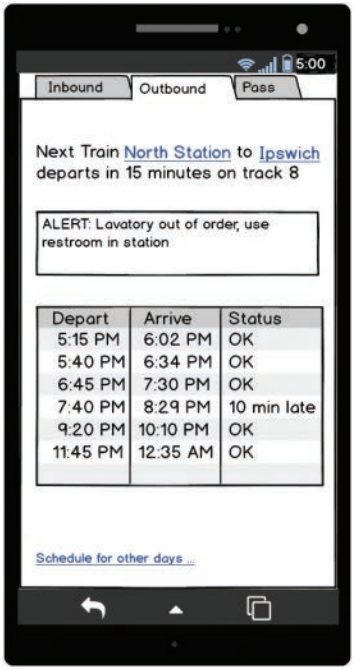


Figure 8.11 Third iteration of the MBTA mobile app, incorporating feedback from previous attempts.

I'm still stuck on the case of Charlie sitting at home Sunday night, needing to take the train in on Monday and wondering what the schedule will be. That was the point of the radio buttons in the original mockup, which all the test users hated. But we don't want Charlie to do any additional work by having to click the Schedule for Other Days link. So I brought in a pattern I see in airports. When the monitor lists flights by time, at the end of the day when there aren't many left, they show a little banner after the last of tonight's flights, and then they start showing tomorrow's flights. So we can easily see that we're at the end of today and what's happening first thing tomorrow. I made a small change and came up with the iteration shown in Figure 8.12.

The app would be smart enough to bring up the correct tab based on the user's location. If Charlie opened it while downtown, it would figure he's most likely headed home, so it would bring up the outbound tab. Story 2 discusses this. But if Charlie opens the app in the suburbs in the evening, it would automatically show the inbound tab, with the last of today's inbound trains (if any) and the first of tomorrow's inbound trains. All the users loved this when I showed it to them.

Some other ideas surfaced here as well. One tester said, "What about a space for ads?" Much as I'd personally hate to see them, the rail agency could charge a lot of money for delivering this

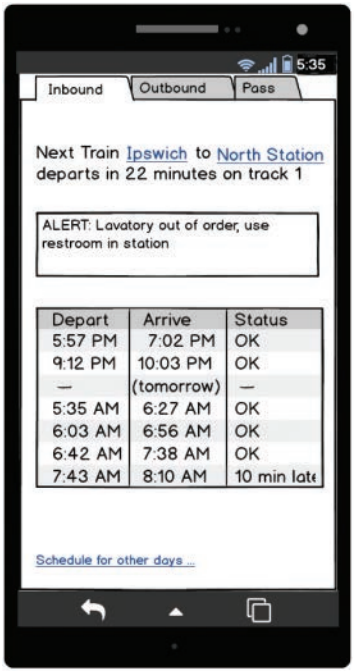


Figure 8.12 Fourth iteration of the MBTA mobile app, automatically showing tomorrow's trains at the end of the day.

clientele to advertisers, especially if it were location based and time based. Imagine that you're walking toward the station and your phone beeps. "I see you have an extra ten minutes. Here's a coupon for a free donut with purchase of a coffee at Dunkin' Donuts." Or whatever.

There was some discussion about phone-level alerts or text messages when the schedule got seriously messed up, not just notifications within the app. Interestingly, all three test users expressed a desire for some information as to how crowded a train was. After further discussion, we judged these to be impractical, as you don't know for sure how crowded a train will be until it fills up around departure time. Also, regular commuters usually know which trains are the most crowded. And there's not much they can do about a crowded train anyway.

The point of this discussion is not to learn how to make your first design perfect. Your first design will *never* be perfect. It probably won't even be all that great; mine wasn't. It is meant to stimulate discussion, to get users thinking and talking to you. You need to iterate, and therefore you need to iterate as quickly and cheaply as possible. It shouldn't take more than a week to go from sketches to testing and iterations. Lather, rinse, repeat.

There will be more detailed design, and better graphic design, as we continue development. But the point is that rapid sketching and quick feedback give us huge, *huge* advantages very

quickly and very cheaply. We were able to switch designs before we'd spent much money, or much time, or gotten emotionally invested in any design. As I put on the shoes of my users, I learned all sorts of things that I never knew that I never knew. Now I know them, and so do you.

Step 5: Telemetry Plan

Every modern app uses telemetry. What will we record with our telemetry in this MBTA commuter rail mobile app, and what use will we make of that information?

In order for our app to provide the seamless capabilities we are asking of it, we need to make steadily better guesses as to what the users want. As you have seen, much of our logic is based on the time of day and the location at which the user makes requests. For example, when the user looks at the schedule in the morning in the suburbs, we deduce that he wants to see the inbound trains and come up with that tab showing. If he's in the city in the morning, he probably has a reverse commute, or pulled an all-nighter, so we bring up the morning outbound trains.

For each UX event, we need to record the time of day and the location of the phone. We won't use GPS location because of the power drain, and also potential privacy problems. But cell-level location, easily available on all phones, will serve well enough for our needs. We won't be able to tell if the user looks at our app on First Street versus Second Street. But we'll certainly know if the guy is in Salem or in downtown Boston.

The first thing most telemetry focuses on is the feature usage profile. How often do users use each feature? We went to a great deal of trouble to figure out which schedule tab to show. Did we get it right? How often do users change it? We make our best guess as to which trains to show in the schedule displays. How often do users scroll? We infer the station from the tickets the user purchases. How often does anyone change it? Does anyone ever use the Schedule for Other Days link?

Since we're selling tickets with this app, we'll want to know how often that feature gets used. We can compare that data against other channels. What percentage of monthly pass users do as Claire did, setting up her monthly pass on a Web browser on her PC, versus doing it on her phone? How many tickets does Charlie buy at a time? At what time of day does Charlie buy his tickets, and what is the peak load?

We'd learn a lot from this feature tracking. We probably shouldn't go any further until we've digested this amount of data and iterated our UX a few times based on it. We can then evolve it to almost anything we care about. Our data miners will certainly have their requests.

If we really wanted to get fancy, we could gather data from the phone's accelerometer, compare it with location and time, and figure out how often anyone ever runs for a train.

Step 6: Security and Privacy Plan

Now we come to the security and privacy plan. At first glance, this app has very little security or privacy need. All of the data on the schedule side is public. You do not need to authenticate if you want to know when the next train is arriving. There's not a whole lot to be concerned with.

Certainly someone who steals the phone can see which train line the user has been looking at, and it's a pretty good bet that's the one she's been riding all along. A user has far worse things to worry about if her phone gets stolen, like the content of her text messages. A user will secure almost anything else, even her Kindle reading list, before she cares about locking up her train schedule. If a user does worry about her commuter rail profile becoming public, she'll get a phone that encrypts its entire contents and locks it all up with a fingerprint reader.

The only piece that needs to be secure is the credit card number used for purchasing tickets. It's important to store it in a way that the user doesn't have to take out her credit card and type it in every time. A user doesn't want pickpockets or purse snatchers in a rush hour station to get a look at her wallet. The choices are to store the number on the phone itself, perhaps encrypted, or store it on a central server, like Amazon does, and fetch it when she wants to purchase a ticket.

Which will users prefer? It might depend on when we ask them. If there had been a data breach in the last few weeks, like the one that hit Target in 2013, they'll want it stored on the handset. They'll figure a bad guy will focus on hitting big targets where he can steal millions at a time, not one at a time by stealing phones. There's not a whole lot you can do to prevent that. On the other hand, if Claire wants her monthly pass to automatically renew, she has to give the T some sort of ongoing financial authority, whether it's credit card or automatic checking account withdrawal. And once it's there, we might as well keep it all there.

With privacy, our biggest problem will be the tracking of user movements and locations. In theory, this information becomes a problem only when it is personally identifiable. We could give each user a unique identifier, a random number assigned when the app is installed. We would know that anonymous user 24168302 went into town on the 6:00 a.m. train and came home on the 4:30 p.m. train. We wouldn't know that particular user was Mary Smith, but we'd know somebody did it. We could then work out individual profiles, which could be of great use to data miners.

The problem is that if we collect this information, it's liable to get abused somewhere. We could, or someone could, correlate user movements with ticket purchases and work out who each user was.

Suppose the news leaked out that the T was collecting the movements of individual riders. "To serve you better," they'd doubtless say. "We're only doing good stuff with it." What do you suppose the local Fox News affiliate would do with that story? How long would it take for

customers to ditch our app? That's like the dentist saying, "This won't hurt." No, it won't hurt *him* a bit, will it? Suppose one of the programmers got caught stalking an ex-girlfriend?

As riders, we don't have an intimate relationship with the T. We don't have any reason for them to be tracking us as individuals. I'd suggest that this app should not collect data on specific users, even if it's anonymized. If we're not recording it, we can't possibly leak it. Project managers will sleep a whole lot more soundly at night that way.

Step 7: Make It Just Work

Now that we've worked out our designs for this project, let's go back and review them against the commandments that I gave you in Chapter 7. We're trying make this app just work. How are we doing?

Start with Good Defaults

This is probably the biggest improvement that we've made from the original app. Instead of throwing all the documentation for the entire commuter rail network at the user, and forcing her to strain out the pieces she cares about, our app automatically deduces the pieces that she cares about. We generate her default stations from her ticket purchases, and her current location from cell-level positioning. We show them to her front and center.

Remember Everything That You Should

We do this very well in this app. The most important item that we remember is the user's usual commuting route—the stations that he goes to and from. We use that information to determine the schedules to show him, and the time to the next train.

Speak Your Users' Language

We are careful to speak our users' language at all times. Probably the biggest decision I made in relation to this was omitting the train number from the schedule grid view. Certainly the T uses train numbers internally, as airlines do. But riders, according to their interviews, never, ever think in terms of those. They always think in terms of the time: "Damn, missed the 6:20 and they canceled the 6:50; the next one goes at 7:14."

Don't Make Users Do Your Work

The original app made our users do a lot of work to find and then read the information they needed. This app automatically figures out what the user wants and then shows it to her. She's doing a whole lot less work than she used to do, as close as we can get to no work at all. Even

the tabs that we show are carefully selected—the outbound trains if she's in town, the inbound trains if not. We've done this well.

Don't Let Edge Cases Dictate the Mainstream

The main case of the rail commuter is making the same trip over and over, day after day. This app is highly optimized for that case. If he wants something else, like "Hey, if I take the train in for the Bruins game on Saturday, when are the trains?" there's a link for it, but he'll have to do some work. A rail fan whose hobby is riding every line to every station will have to do much more work. We have carefully optimized for the main case.

Don't Make the User Think

This app's biggest de-thinker is the countdown timer. It shows the time until the next train, so you don't have to look at the schedule, look at the clock, and do math. It shows you the track number so you can go directly there. This app requires as little thinking as I could possibly make it.

Don't Confirm

We don't have any confirmation in this app. This app doesn't do anything that would require it.

Do Undo

The scheduling portion of the app doesn't have anything that we could undo. In the ticket purchase portion, perhaps we could allow return of purchased tickets. But the T doesn't refund paper tickets, so electronic tickets won't get refunded either.

Have the Correct Configurability

This app automatically detects the stations that the user travels to and from. If for some reason our automatic detection is wrong, or the user changes her pattern, we provide a link by which she can make changes. We also provide a link if she wants to look at other times. Nothing else is configurable. That's a good place to start for this app. We may gain a little more configurability in the future, such as the choice of putting the tabs at the top or bottom of the screen.

Lead the Witness

This app leads the witness in everything—automatically detecting which stations we use, which trains we ride, automatically showing us when the next one leaves. Compare it to the original app where the user has to dig for every last scrap. This is a damn good app, if I do say so myself. Which I do, because it is.

INDEX

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z