Learning iOS Design
Visit informit.com/learningseries for a complete list of available publications.

The Addison-Wesley Learning Series is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you’ve learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.
To Buzz and CeeCee; Touichi and Risako
Contents at a Glance

Foreword           xix
Preface            xxi
Acknowledgments    xxix
About the Author   xxxi

I Turning Ideas into Software       1
  1 The Outlines             3
  2 The Sketches             15
  3 Getting Familiar with iOS  31
  4 The Wireframes           55
  5 The Mockups              81
  6 The Prototypes           111
  7 Going Cross-Platform     127

II Principles          143
  8 The Graceful Interface  145
  9 The Gracious Interface  167
 10 The Whole Experience  195

III Finding Equilibrium 221
 11 Focused and Versatile   223
 12 Quiet and Forthcoming   237
 13 Friction and Guidance   255
 14 Consistency and Specialization  271
 15 Rich and Plain         285

Index           303
Contents

Foreword xix
Preface xxii
Acknowledgments xxix
About the Author xxxi

I Turning Ideas into Software 1

1 The Outlines 3
The Process: Nonlinear but Orderly 3
Writing about Software 4
The Mental Sweep 6
More Inputs to Outlining 7
Outlining Requirements 8
Introducing SnackLog 8
Antirequirements 9
Define a Platform 10
Listing Ramifications 11
iOS and Featurefulness 11
Reducing Problems 12
Outlining Architecture 13
Your Outline Is Your To-Do List 14
Summary 14
Exercises 14

2 The Sketches 15
Thinking by Drawing 15
Design Happens in Conversations 16
Tools for Sketching 18
Sketches Are Sketchy 19
When to Sketch 20
Using Precedents 21
Playing Devil’s Advocate 22
Sketching Interfaces 22
Sketching Interactions 24
Contents

Sketching Workflows 26
Summary 29
Exercises 29

3 Getting Familiar with iOS 31
Navigation: Screen to Screen 31
  Navigation Controller 31
  Split View 34
  Tabs 35
  Segmented-Controls-as-Tabs 36
  Multiple Personalities 36
  Modal View 37
  Popover 39
  Custom Navigation 39
Advice on the Standard Elements 41
  Bars 41
  Content Views 43
  Alerts 46
  Action Sheets 47
  Standard Controls 48
Custom Controls 52
Summary 53
Exercises 53

4 The Wireframes 55
Thinking in Screens 56
Thinking in Points 57
Optical Measurements 57
  Measuring Text Optically 59
  Measuring Images and Controls Optically 60
  Techniques for Measuring 60
Tools for Wireframing 61
Principles of Layout 63
  Unity Is the Goal 63
  Visual Weight 64
  Similarity and Distinction 65
  Proximity and Distance 66
Alignment 66
Rhythm 68
Margin and Padding 70
Balance 71
Understatement 71
Typography 72
Layout: A Place for Everything... 74
Content and Controls 74
Thinking in Layers 74
Controls in Content Areas 75
Information Density 75
Dimensionality 76
Orientation on iPhone 77
Orientation on iPad 78
The Worst-Case Height-Compression Scenario 78
Summary 79
Exercises 80

5 The Mockups 81
When to Mock Up 81
Styling: The Apparent Design Discipline 82
Rendering 83
Communication 84
Tastefulness 84
Mockup Tools 85
Color: Thinking in HSB 86
Good Old RGB 86
Introducing HSB 87
Get Serious about Value 88
Contrast: Thinking in Figure/Ground Relationships 89
Styling for Good Contrast and Visual Weight 89
Good Backgrounds 92
Transparency 93
1+1 = 3 94
Presenting Image Content 95
Evaluating Contrast: Posterize It 95
Contrast Examples 98
Table Cells  98
Action Sheet Buttons  99
iBooks Page Metadata  99
Birth of a Button  100
  Step 0: Set Up the Canvas  100
  Step 1: Create a Shape Layer  101
  Step 2: Choose a Fill Color  102
  Step 3: Apply a Gradient  102
  Step 4: Add a Stroke  103
  Step 5: Add a Bevel  104
  Step 6: Add Texture  105
  Step 7: Add an Underhighlight  105
  Step 8: Add Contents  106
Onward  106
Mockup Assembly  106
Resizable Images  107
Retina Resources  107
Designing for Layers  108
Summary  109
Exercises  109

6  The Prototypes  111
  Test on the Device  111
  Kinds of Prototypes  112
  Paper Prototypes  112
  Wizard of Oz Prototypes  114
  Motion Sketches  115
  Preemptive Demo Videos  117
  Interactive Prototypes  118
  Proof-of-Concept Software  121
  Why Do Usability Testing?  123
  How to Do Usability Testing  124
Summary  126
Exercises  126

7  Going Cross-Platform  127
  Platform Catalog  127
  Standalone, Mini, and Companion Apps  129
Visible Status 178
Contextual Status 179
Invisible Status 180
  Adaptation 180
  Learning 182
  Resourcefulness 182
The Sense of Adventure 183
Capability 184
Defensive Design 185
Forgiveness 187
  Undo 187
  Manual Undo 189
  Confirmation 190
Making SnackLog Gracious 191
Summary 193
Exercises 193

10 The Whole Experience 195
Serve the Soul 197
Conveying Capability 198
  The Name 199
  The Icon 199
  Launch Images 202
  The App Store Listing 202
  The Price 205
Documentation 206
  Comprehensive Documentation 206
  Problem-Solving Documentation 207
  Tutorials 208
  Release Notes 209
  Characteristics of Good Documentation 210
Support 211
Localizaion 211
Accessibility 213
  VoiceOver 214
  AssistiveTouch 214
Ethos 215
Respect  
   Respect for Time and Attention  215
   Respect for Data  216
   Speaking of Betrayals of Trust...  216
Summary  219
Exercises  219

III Finding Equilibrium  221

11 Focused and Versatile  223
   Debunking “Simple” and “Complex”  223
   The Focused Design  224
      Focused Apps Are About Real-World Goals  225
      iOS Loves Focus  225
      Massacre Features  225
      Consolidate Functionality  226
      Save It for Later  227
      Scaling Back  227
   Focusing SnackLog: Labeling  228
      Scaling Back on Labeling  230
   The Versatile Design  230
      Versatile Apps: Bring Your Own Goals  231
      iOS Loves Versatility  231
      When to Go Versatile  233
      How to Go Versatile  233
      Triangulation  233
      Pattern Recognition  235
      Finding the Boundaries  235
Summary  236
Exercises  236

12 Quiet and Forthcoming  237
   Adjacent in Space  238
   Stacked in Time  239
   Progressive Disclosure  240
   Group by Meaning, Arrange by Importance  242
   Promotion and Demotion  243
Splitting the Difference 246
iOS Loves Context 246
Hide, Don’t Disable 248
Disappear 248
Taps Are Cheap 250
Loud and Clear 250
Making SnackLog Quiet 251
Making SnackLog Forthcoming 252
Summary 253
Exercises 253

13 Friction and Guidance 255
The Difficulty Curve 255
Experience Weight 257
Why Add Friction? 257
How to Add Friction 258
Unintended Friction 259
  Don’t Expose Underlying Mechanisms 261
  Streamline Input 261
Guidance 262
  Zero Options 262
  One Option 263
  Guidance among More Options 264
Sensible Defaults 266
  The Blank Slate 267
  Templates 268
  Presets 268
Summary 270
Exercises 270

14 Consistency and Specialization 271
How It All Works Out 271
Getting the Most Out of the HIG 272
The Consistent Design 273
  Precedents, Motifs, Patterns, Shorthands 275
  Avoiding Cargo Cult Design 277
<table>
<thead>
<tr>
<th>Sections</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Specialized Design</td>
<td>278</td>
</tr>
<tr>
<td>Harmless Distinctiveness</td>
<td>279</td>
</tr>
<tr>
<td>Conscientious Divergence</td>
<td>279</td>
</tr>
<tr>
<td>One Free Novel Interaction</td>
<td>280</td>
</tr>
<tr>
<td>Novelty Is Hard</td>
<td>282</td>
</tr>
<tr>
<td>Summary</td>
<td>283</td>
</tr>
<tr>
<td>Exercises</td>
<td>284</td>
</tr>
</tbody>
</table>

### 15 Rich and Plain  285
- Color versus Monochrome       286
  - Using Hue                   286
  - Using Saturation            288
  - Using Brightness            289
- Depth versus Flatness         290
- Lighting                     291
  - Extremes of Flatness and Depth 294
- Realism versus Digitality     296
- Texture and Tactility        297
- Metaphor                     297
- Ornamentation                298
- Simulation                   299
- Take It Easy                 301
- Summary                      301
- Exercises                    302

Index  303
This page intentionally left blank
When Apple introduced Mac OS X, Mac users' feelings were ambivalent. Sure, this looked like a fantastic operating system, but a huge part of what made the Mac unique was its software. Photoshop, Illustrator, Claris Works, MacPaint—these were the reasons we used Macs. And with Mac OS X, all of these applications effectively stopped working. There were few native applications for Mac OS X, and fewer still that weren’t horrible.

There was, however, one company that consistently developed fantastic software for Mac OS X right from the start. And they kept doing it. For the last decade, The Omni Group has been a sure bet for quality products. Applications like OmniGraffle combine ease of use and sheer power in a way that is unique, yet feels completely natural. On the one hand, these applications are incredibly accessible. It takes very little to create fantastic output. On the other hand, they have great depth. Recently, the Omni Group has expanded their reach to iOS, and they’ve done something almost nobody else outside of Apple has achieved: they’ve brought their applications to the iPad in a way that makes them feel native to these portable touchscreen devices, but doesn’t diminish their power and depth.

I’m probably not the only designer who has more than once looked at applications like OmniOutliner, OmniGraffle, or the somewhat exorbitantly named OmniGraph-Sketcher and wondered to themselves: How do they do it? How do these people consistently create software that seems to effortlessly present incredibly powerful features in a way that is easily accessible, and a pleasure to use? And even more puzzling, how do they manage to achieve this feat on iOS, a platform famous for its abundance of shallow, poorly designed, one-trick-pony, cash-grab apps?

Well, today’s your lucky day, because you’re holding the answer to this question in your hands. My friend Bill, who wrote this book, happens to be Omni’s User Experience Lead. And he’s lifting his kilt, just for you.

I first consciously heard of Bill when he became Internet-famous for talking about Omni’s 1:1 replicas of iPads made from wood, cardboard, Plexiglas, and 3-D-printed parts. Who would want to make 1:1 replicas of iPads? Well, Apple had announced the iPad, but had not yet started shipping it. Having already started designing apps for the iPad, Bill’s team needed to get an idea for how these apps would feel on an actual device. At this point, less dedicated people would just postpone the whole thing for a few months. But not Bill’s team. They went ahead and made their own iPads.

Most UX designers eventually manage to come up with a design that works well. It’s this kind of relentless dedication to detail, this kind of work ethic, though, that is
the difference between a designer who can come up with a good design, and one who will come up with a mind-blowingly awesome design.

But there’s something else that makes Bill unique among his peers. Any designer will tell you that their goal is to make the product they’re working on beautiful and easy to use and efficient and pleasant. But Bill goes one step further. His goal isn’t just to make apps user-friendly, but to touch the user’s soul, to help people make more beautiful things, be more successful, and be happier. In one of his presentations, he recounts how one man converted his classic VW Beetle into an electric car with the help of OmniGraffle. To Bill, that’s the ultimate goal. Software design isn’t just about making an application easy to use, it’s about making the application have a positive impact on people’s lives. It’s about helping people be better.

This book contains everything you need to know to create awesome, life-altering applications, just like Omni’s. While it’s targeted at iOS designers, you’re going to learn a lot from reading this book regardless of the platform you design for. I pride myself on knowing a lot about design, but when reading this book, I probably didn’t encounter a single page that didn’t offer at least one interesting idea, new concept, or clever design technique. From learning how to make your application more forgiving to a section on how pricing influences how people perceive your app (yep, its price is part of the app’s design), you’re in for a treat.

Even better, this book doesn’t just offer invaluable content that will forever change the way you design applications, it’s also written in a way that prevents you from putting it down. So grab a hot cup of cocoa, put on your favorite music, and settle down into your most comfortable chair, my friend, because you’ll be sitting here, staring at this book, for quite a while.

Enjoy it.

—Lukas Mathis, ignorethecode.net; author of Designed for Use: Create Usable Interfaces for Applications and the Web (Pragmatic Bookshelf, 2011)

March 2013
Hello

It took a while for the world to notice, but design really matters.

A perfect story of the power of design can be found by traveling back to April 2007 to eavesdrop on a chat with Microsoft CEO Steve Ballmer. Apple’s Steve Jobs had announced the iPhone that January, and everyone had had a good while to process the announcement and decide what they thought of it. Ballmer, in an interview with USA Today, opined on the iPhone’s chances to make a dent in the well-populated smartphone market: “There’s no chance that the iPhone is going to get any significant market share. No chance.”

I’m not normally one to indulge in schadenfreude, but the wrongness of that prediction is too illuminating to ignore. iPhone went on to become an icon that redefined the public’s concept of what a mobile phone is, and nearly every “smartphone” on the market takes inspiration from it. Its sibling, iPad, finally popularized the stagnant tablet concept and is on its way to replacing the traditional desktop or notebook computer for millions. iPhone and iPad each own about half of the market share of their respective markets. The App Store model has redefined the way people buy software and has paid out more than $7 billion to third-party developers. As of the beginning of 2013, nearly half a billion iOS devices have been sold.

Why? How did iOS become so successful? What did Ballmer and the rest of the early-2007 iPhone scoffers miss? Ask any authority who followed the story closely to pick one word to describe Apple’s advantage, and they’ll say design. (Some cynics might say marketing, but they’re wrong.)

iOS is arguably the first technology platform to truly put design first. Instead of the puffed-up and bulleted feature lists, the contortions to accommodate legacy systems, the assumptions about how a phone was supposed to look or behave, and the obsession with being the first to the market, iPhone prioritized beauty, responsiveness, and fun. (And anything that Apple couldn’t get just right was omitted until they could.) This view of design is about creating happiness, about cultivating a relationship with the user, about imagining the most positive user experience possible and then doing whatever it takes to produce that imagined outcome.

You could almost say that iPhone refused to compromise on its user experience. But as this book argues, all designs are compromises. Surely, countless tradeoffs and tough decisions were made in the process of bringing iOS into being. But what’s important is
that wherever possible, those compromises erred on the side of paying attention to detail, abandoning conventional wisdom, and putting in more work to make users happier.

Not solely because of Apple and iOS, but in large part, the world is learning that design counts. It’s getting harder to compete without good design. It’s harder to find good designers than it is to find good engineers (and that itself is pretty hard). Well-designed software really can improve people’s lives, help them be more productive, and yes, make them happy. This book aims to give you the practices, examples, and advice you need to make it happen yourself.

You’re a Designer

Design is deciding how a thing should be. In every act of design, that decision-making is done to accommodate constraints and to satisfy the needs of some audience or “user.” The needs are paramount, because an artifact that doesn’t do anything useful for anyone is more a piece of art than a design. And the constraints are your friends, because they narrow the space of possibilities, making your job much more approachable. Almost everything you think about and do as a designer can be narrowed down to these concepts: How are you serving the needs of the user? How are you working within the constraints?

Everything artificial was designed by someone. Most of the time you don’t think about the people who decided how the things around us should be: the height of a chair’s seat, the shape of a battery charger, the hem of a blanket. That blissful ignorance is the goal of many designers. If people don’t think about the design of an object, the designer has probably done a fantastic job. More than two thousand years ago, Ovid said it like this: Si latet, ars prodest. If the art is concealed, it succeeds. That’s one to print and hang on your wall.

If you’ve ever made something, then you’re a designer. Ever built a couch fort? Arranged some flowers in a vase? Sketched a map for someone? Whether or not you thought very much about it, whether or not you followed well-researched principles, you designed that thing. That’s design, with a lowercase d. You could take that approach to designing an iOS app, but the result isn’t likely to be compelling. Books like this one aim to help you do Design with a capital D. That means absorbing and imagining as much as you can about how things could be better. It means making the smartest, most informed decisions possible about the needs and constraints involved. And it almost always means creating plans, sketches, and models along the way to a final product. The good news is that you can get there from here, one step at a time, always experimenting and learning as you go.

Meet the Book

This book introduces and explores the topic of designing iOS apps, even if you don’t consider yourself a designer (yet). Even if you’ve never taken an art or design course, if you consider yourself to have more of an engineering or analytical mind than a
creative one, or if you're mystified by what actually goes on in the process of design, you're very welcome here.

At conferences, I've presented the topic of design to a largely engineering-minded audience. Lots of programmers know that they should care about design, but the practice of design seems from the outside to be mysterious or even arbitrary, leaving them disillusioned or apathetic about it. But after some demystification and conversation, some folks have told me that they finally get why design is important and how they can think about it systematically.

This book presents the art and science of design in an accessible, sensible way.

**Part I: Turning Ideas into Software** steps through the phases of design, turning a vague idea for an app into a fully fleshed-out design. It goes from outlines to sketches to wireframes to mockups and prototypes. Each step of the way, you'll find advice about how to think carefully, critically, and cleverly about your project. Each chapter concludes with exercises conceived to encourage you in planning the design of your own app. Part I includes the following seven chapters.

- **Chapter 1: The Outlines**—This is all about planning, writing things down, and making sense of your app idea. You'll learn about the ways you can use structured thinking and writing to figure out what your app is about and stay on track throughout the project.

- **Chapter 2: The Sketches**—Sketching is the central activity of design. It's all about getting ideas out there and seeing where they lead. You can never know the merits of an idea until it's on a page, a whiteboard, or a screen. This chapter will help you sketch with the right blend of adventurousness and discipline.

- **Chapter 3: Getting Familiar with iOS**—Understanding the constraints of the platform is crucial. iOS offers a versatile kit for building interfaces and experiences; you should know it well enough to decide when to take advantage of it and when to diverge from it.

- **Chapter 4: The Wireframes**—Eventually you need to turn your sketches into precise, screen-by-screen definitions of how the app should be organized. A wireframe is a document that specifies layout and navigation without getting bogged down in pixel-perfect styling just yet.

- **Chapter 5: The Mockups**—It's not the only concern of design by far, but it matters what your application looks like on the surface. In this chapter you'll break out the graphics apps and learn how to assemble beautiful assets into a convincing, pleasant whole.

- **Chapter 6: The Prototypes**—Sometimes a static drawing of an interface is not enough. You need to know how it behaves. This chapter is all about simulating and testing the interactions that make up your app.

- **Chapter 7: Going Cross-Platform**—Plenty of apps exist not as completely standalone experiences, but as parts of a multiplatform suite. This chapter explores the concerns you'll need to deal with if you want to build the same app
for more than one device. It uses an app that appears on iPhone, iPad, and Mac as a case study to illustrate how a single idea can wear three different interfaces.

Part II: Principles presents universal principles that apply to any design and that you should follow if you want to craft an effective app that people will appreciate and even love. To make sure your app works on every level, each chapter in this part is based on one of the three levels of cognition identified by psychologist Donald Norman. Many of these principles are applicable to all software design, but here they’re tailored to the strengths and challenges of iOS. The exercises for each chapter present sample situations to help you learn how to apply each principle.

- Chapter 8: The Graceful Interface—This chapter examines the visceral level of cognition, which relates to the way people feel from instant to instant as they interact with software. It deals with things like touch input, timing, and feel. Most of the concerns here are subconscious. Users may not notice them, but they subtly affect how pleasant the software is to use.

- Chapter 9: The Gracious Interface—Here you’ll learn about concerns at the behavioral level of cognition. That means how users make decisions moment to moment and how the app communicates possibilities and status. The chapter also discusses how the app can encourage a sense of adventure so that users feel welcome and safe as they explore its possibilities.

- Chapter 10: The Whole Experience—The biggest, vaguest, most intangible, and most important level of cognition is the reflective level. This chapter explains how people feel about your app in the long run: whether they rate it well, whether they recommend it to friends, whether they respect you as a developer, and whether they’d buy from you again. Happiness is the ultimate goal.

Part III: Finding Equilibrium is meant to function as a reference, inspiration, and exploratory guide to the various decision points you may encounter in designing an app. It embraces the concept that all designs are compromises and that many decisions have no single correct answer. This means that many answers to the same design problem can coexist, and every design, no matter how unfashionable or unsophisticated it seems, has something to teach (a fact that many critics seem to forget). You can look at each chapter’s opposed approaches as a sort of slider control, with a continuum of answers between the extremes at either end. For each challenge, a smart designer like you should seek an answer that works best for your app’s unique philosophy. Over time you may find yourself preferring one side of a given slider over the other. Maybe you like to err on the side of focused rather than versatile. Or perhaps you’d rather seek the Aristotelian golden mean, straight down the middle. That’s great. That’s what it means to have a style. Each type of decision is illustrated by examples of different solutions to the same problem, depending on the angle you prefer. The exercises encourage you to find your own favorite solution for a situation that may have several possible answers.
- **Chapter 11: Focused and Versatile**—One of the biggest decisions you need to make about your app is its scope. Do you want to do one thing flawlessly, or many things competently? What's feasible depends on the resources available and your ability to be aggressive about defining what you expect of the project.

- **Chapter 12: Quiet and Forthcoming**—When most people talk about a design being “simple,” what they usually mean is that it’s in good order and presents an understandable amount of information and control at once. In contrast, designs feel empowering when they simultaneously present as much as possible. This chapter describes how to control the apparent simplicity of your app from screen to screen, depending on the emotion you prefer to evoke.

- **Chapter 13: Friction and Guidance**—Part of the job of a software designer is to make many things possible, but also to gently guide people through an experience. This chapter is about the ways an interface puts down grooves that encourage a user to move this way or that way next, or slow down before taking the next step.

- **Chapter 14: Consistency and Specialization**—Differentiating yourself from the rest of the apps out there is both an advantage and a risk. When you think of well-designed apps, the examples that come readily to mind are the ones that break from convention and get away with it. But respecting the established guidelines is usually the wiser path. This chapter will help you decide when to stick to the script and when to diverge.

- **Chapter 15: Rich and Plain**—The visual styling of an app is the most conspicuous outward manifestation of its design. Independent of its functionality, your app can look extravagant or subdued, lifelike or digital. This chapter will help you tune the depth, color, and realism of your interface to set its tone and personality.

**Meet the Web Site**

The web site for this book is [http://learningiosdesign.com](http://learningiosdesign.com). There, you will find resources such as the Photoshop and OmniGraffle source files for the examples given throughout the book. You can also offer feedback about the book and find updates of its content.

**You and Your Team**

You can follow this book as you work on your own app idea, especially by working through the practices described in Part I. Even if you don’t yet have an app project, or if your app already exists and you want to revise it for a new version, you should be able to benefit from the book. Parts II and III are compatible with dipping into for inspiration or advice.
From time to time, the book may talk as if you are a designer working with a software engineer or a team of engineers. That of course doesn’t need to be the case. Maybe you’re one of that noble species, the lone programmer/designer hybrid. Maybe you’re a product manager looking to understand design better. It doesn’t really matter; whenever this book mentions “your engineers,” it’s fine if that means you!

**Art/Science Duality**

Design is full of what are called “wicked problems”: they’re difficult to define, they’re impossible to come up with definitive answers to, and they’re never finished. That’s likely to spook some people, but it’s also what makes design so much fun. You never know what you’re going to get. There’s always some way to improve on your work. Everything is a matter of taste, and yet some answers are unequivocally better than others. There’s no recipe, and yet there are morsels of wisdom and inspiration to be found everywhere.

Design is an art. And it’s a science. And it’s neither. Steve Jobs liked to say that what Apple does falls “at the intersection of technology and liberal arts.” You may find your team arguing about how to make a decision. One side is showing numbers; usability test metrics clearly indicate that design $A$ is more efficient than design $B$. The other side is arguing that based on aesthetics, design $A$ just doesn’t feel right. Who wins? Maybe it’s one of those two options; maybe it’s a third, new option. Figuring it out is part of the thrill of design.

You could take a completely scientific approach, refusing to budge on anything until you’ve run a statistically significant study. You could also take a completely artistic approach, following your muse and composing your personal magnum opus in app form. But you won’t get very far with either one alone—data and heart both matter.

**Inspiration Is Everywhere**

This book can give you specific advice on specific topics and situations that occur often in the work of designing apps for iOS. But your growth as a designer depends, more than anything else, on your willingness to absorb inspiration from around you. Pay attention to all kinds of design: graphics, interiors, architecture, games, anything. Read widely: psychology, art, history, biology, everything. The most seemingly irrelevant knowledge may end up informing your work as a designer someday, in some oblique way. If you do nothing else, use lots of well-regarded apps and think about what makes them successful. The more you examine and ponder great work of all kinds, the better you’ll get at it yourself.

Again, growing as a designer is a lifelong journey, but here is a necessarily short list of reading material to get you started. Some of these books are mentioned again in the chapters where they’re especially relevant.
• *Universal Principles of Design* by William Lidwell, Kritina Holden, and Jill Butler—A delightful collection of 125 concepts that apply to all categories of design. Very compatible with flipping through for quick inspiration.

• *The Elements of Typographic Style* by Robert Bringhurst—One of the most carefully built, wisdom-packed books of all time. Yes, Bringhurst will make you knowledgeable about type, but he will also inspire you with his methodical, tasteful approach to design in general.

• *Visual Explanations: Images and Quantities, Evidence and Narrative* by Edward Tufte—Or any of his four main books, really. Tufte tends to lean toward information design for print, but the principles he espouses should be useful to anyone who has any interest in making things understandable and beautiful.

• *Designing Interactions* by Bill Moggridge—This book is a collection of captivating interviews (included on DVD) from original Macintosh software lead Bill Atkinson to legendary game designer Will Wright.

• *Sketching User Experiences: Getting the Design Right and the Right Design* by Bill Buxton—Much of the reverence that technology designers have for the practice of sketching can be credited to Buxton. Sketching is good for your brain and good for your work.

• *The Design of Everyday Things* by Donald Norman—A classic that has stood the test of time. This book pioneered the dissatisfaction with poorly designed experiences and set the stage for a generation of designers to make the world a more agreeable place to live in.

• *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests* by Jeffrey Rubin and Dana Chisnell—If you’re interested in the scientific side of design, this is an excellent walkthrough of the procedures and principles of collecting data from a sample of the target audience using your app.

• “The Nature of Design Practice and Implications for Interaction Design Research” by Erik Stolterman—A brief academic paper, chock full of references to other influential papers, about what design really is and how to deal with its complexity.

• *Basic Visual Concepts and Principles: For Artists, Architects and Designers* by Charles Wallschlaeger and Cynthia Busic-Snyder—A solid grounding in perception and the construction of visuals.

• *Revolution in the Valley: The Insanely Great Story of How the Mac Was Made* by Andy Hertzfeld—This book is a treasure trove of firsthand anecdotes about the culture and creativity surrounding the development of the original Macintosh. If it doesn’t get you excited about making technology, nothing will.

• *How the Mind Works* by Steven Pinker—A comprehensive tour of what we understand so far about human psychology. Not directly related to software design, but a surprising source of insight into how people think and why design principles work the way they do.
Thinking, Fast and Slow by Daniel Kahneman—An up-to-date psychology book about how people pay attention, judge situations, and make decisions. Another surprisingly enlightening read for science-minded designers.

And here are a couple of things that aren't books.

- “Inventing on Principle”—A one-hour talk by Bret Victor, interaction designer for iPad (among many other impressive accomplishments). Victor has among the most thoughtful and inspirational minds in technology design, and this talk is a fantastic place to start learning from him. This is the sort of talk you’ll want to come back to once a year or so.

- Ideo Method Cards—A deck of cards from the legendary product design firm Ideo. Each card describes a “user-centered” practice that can be of use to designers working through an interesting problem. You can casually flip through the deck for ideas, assemble a mini-deck for a given project, or make up your own ways of getting the most out of them.

- Oblique Strategies—A set of cards, each bearing an enigmatic phrase meant to motivate and give direction to a person facing a creative problem. They were originally created by Brian Eno and Peter Schmidt for musicians, but creative people of all kinds have since found them useful for breaking through difficulty. The cards themselves are rare, but plenty of web- and app-based editions are available.

I found these resources helpful. Hopefully some of them will be at home in your own garden of influences and inspirations.

Now…let’s make some software.
Acknowledgments

Turns out writing a book is hard! Mountains of thanks go out to all these people for making it possible.

Thanks to Barbara Gavin and Erica Sadun for taking a chance on a shy and inexperienced speaker and inviting me to speak at the Voices That Matter series of conferences, which eventually led to this book project. Thanks to Trina MacDonald at Addison-Wesley for guiding me through the writing process. Thanks to Betsy Hardinger for editing that makes me seem like a much better writer than I am. Monumental thanks to my review board: Lukas Mathis, Jim Correia, and Jon Bell; my trust in their wisdom is the reason I’ve been able to maintain confidence in this endeavor.

Thanks to all my colleagues at the Omni Group for giving me the chance to make good software and talk to brilliant people all day as my job. Every day, I feel as if I’m getting away with something. Thanks to my instructors and classmates at the University of Washington’s Human-Centered Design & Engineering professional M.S. program, where I’ve finally been able to get an academic grounding in the thing I’ve been doing all this time. Thanks to my dear friends in #rosa for their endless support and encouragement.

Admiration and thanks go to Yasunori Mitsuda, whose Xenogears albums provided the soundtrack that kept me pushing keys. Thanks, too, to the various coffee shops of Seattle, for providing the perfect writing environment.

It seems as if every book’s acknowledgments page mentions family members’ patience; now I understand why. Copious gratitude and love to my wife, Hiroko, for her steadfast patience and support. Ultimately, everything is thanks to her.
This page intentionally left blank
Since 2004, William Van Hecke has been User Experience Lead at the Omni Group, one of the world’s most accomplished and affable Mac and iOS developers. Bill got his start designing software by reverse-engineering his older brother’s text adventures in MS Basic on the Macintosh Plus, and then graduated to creating HyperCard games to mail to his cousins on floppy disk.

Bill’s primary hobby is hobby-collecting: reading fiction and science; playing bass guitar; appreciating, translating, and developing niche video games; studying the Japanese language; mastering tabletop gaming; and exploring 3-D modeling. You can find Bill on Twitter, prattling on about these topics and more (@fet).
This page intentionally left blank
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

You can email or write me directly to let us know what you did or didn’t like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book’s title and author as well as your name and phone or email address.

Email: trina.macdonald@pearson.com
Mail: Reader Feedback
Addison-Wesley Learning Series
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our web site and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
Getting Familiar with iOS

A major part of getting good at designing for iOS is simply coming to understand what the platform has to offer. Much of your design work will consist of choosing from a set of standard navigation schemes and controls. Occasionally you’ll need to dream up a custom element, but even your original designs should be in the spirit of the platform.

Although you may see these standard elements and behaviors in the apps you already use every day, you might not realize the reasons Apple and savvy third-party developers use them the way they do. This chapter supplements the iOS Human Interface Guidelines with further explanation and advice about the standard choices available. Then I talk about how to create your own custom designs that feel at home on iOS.

After you read this chapter, you’ll be ready to start wireframing. While outlining and sketching, you can get away with scrawling a vague blob in the general vicinity of a feature and saying something like, “We’ll need to provide some way to turn this setting on and off.” When it comes time to wireframe, though, you’d better know your options for a setting that needs a two-way toggle (probably a switch, a segmented control, or a table with checkmarks, depending on the needs of the design).

Navigation: Screen to Screen

Put too simply, the primary challenge of wireframing is figuring out how to fit a list of features onto a series of two-dimensional screens. Part of that challenge is providing navigation between the screens in a way that makes sense and is easy for users. Let’s look at some dependable ways to construct a sensible navigation scheme for your app.

Navigation Controller

A navigation controller is the most common way to get between screens on iOS (see Figure 3.1 for an example). A navigation bar at the top of a screen indicates the current location and contains a back button; rightward-pointing chevrons in the content area offer ways to proceed down the hierarchy. This arrangement allows for any number of branching paths, with a consistent way of getting back up to the top.
The result is a navigation scheme in which a user can scroll vertically on tall screens of information, and move horizontally to step through the greater hierarchy of screens. Navigation controllers work well only for a limited number of levels. If your hierarchy requires users to routinely delve four or five levels deep, you may need to flatten it out. (Chapter 12, Quiet and Forthcoming, explains how to flatten interface hierarchies.)

Navigation controllers are very familiar to iOS users. They trace their heritage all the way back to 2001, on the original iPod, where selecting a row required spinning a physical wheel. They’re still the most reliable, most predictable way on iOS to present a treelike hierarchy of information. Users take it for granted that many iOS apps have a navigation bar at the top of each screen, and that’s the first place they look to check which screen they’re on and how to go back a screen. Yes, the navigation controller is the go-to workhorse for getting around in an iOS app, and it’ll seldom do you wrong.
Most navigation controllers use standard table views (described later in this chapter) to list the options on each screen, but that’s not the only approach you can try. (See Figure 3.2 for examples of other approaches.) Following are the only strong rules for creating a navigation controller experience.

- Delving down a level should involve tapping an element that either bears a rightward-pointing chevron or otherwise presents an obviously tappable piece of content that the user would want to navigate to.
- There should be a recognizable back button in the upper left, labeled with the name of the screen that it takes you back to (not with the word “Back”).
- Moving between screens should use a horizontal slide animation—the content slides leftward for delving down, and rightward for going back up.

As long as you follow those guidelines, the navigation experience should feel comfortable and familiar to users. In that spirit, here are a few ways of presenting options that differ from the ordinary table view.

- A map, where users can tap a pin to open a label and then tap a disclosure button (bearing a chevron) to navigate to a detail screen. The Maps app on iPhone is an obvious example, because a map is a much more appropriate way to display geographical data than a list would be.
- A collection of big, expressive images, especially when there is a strong case for relying on images rather than text to identify your items. This would be an exception to the need for rightward chevrons, because they would unbalance the composition of each icon. Podcasts’ cover view is an example; people react more immediately and emotionally to colorful cover images than to a list of titles.
- A grid of carefully laid-out previews of content. For an app that focuses on a specific kind of content, this approach can be far more expressive and inviting.
than a simple list of titles. Instapaper offers cleanly arranged previews of web articles to help you decide which one to read.

**Split View**

Only available on iPad, a **split view** offers a way to present navigation and content at the same time, something that helps flatten the navigation hierarchy. Most iPad apps that have a branching hierarchy work well with a split view. The relationship between the two sides is simple: what’s selected on the left pane appears in detail on the right pane. Either side can have a navigation controller with ordinary horizontal navigation, but both shouldn’t have navigation. (Really, they shouldn’t. It has been tried, and the result is a confusing mess.) This means that you have two options for spreading navigation across a split view (see Figure 3.3 for an illustration of these two approaches).

- One is like the Settings app, where the sidebar always portrays the top level, and the content area can navigate. This makes it easier to jump around between lots of top-level items and then delve into detail for items on the right side if necessary.

- The other option is like Mail, where the sidebar can navigate, and the content area always displays details about what’s selected on the left. This is a good approach when you have a well-defined, consistent type of content being displayed on the right, such as emails. People can navigate around their mailbox structure, and as soon as they tap an email, the right side updates to show it.

Remember that when you use a split view, you need to decide what happens in portrait orientation: either the split should remain visible, or the sidebar should be

![Figure 3.3 At top, Settings-like split view navigation. At bottom, Mail-like split view navigation.](image)
hidden, sliding in when summoned by the button in the upper-left corner. The answer is easy when you use Settings-style navigation as just described: keep the sidebar visible. The back button in the upper-left means you wouldn’t have a place to put a sidebar toggle button. (Some apps, such as Facebook, do it anyway. The sidebar is hidden, and it’s unavailable at any level other than the top.)

For a Mail-style app, the answer depends on how important it is to keep the sidebar visible at all times versus how likely it is that the user will want to focus on the content area. In Mail, the sidebar is hidden in portrait orientation because users want to focus on a generously sized message area. (See Figure 3.4 for an illustration of these options.)

**Tabs**

A **tab bar** provides ever-present top-level navigation at the bottom of the screen. It’s perfect for an app that needs to provide quick access to a few distinct top-level screens. The quintessential example is the Music app on iPhone: it offers tabs for Artists, Playlists, Audiobooks, and so on. Listeners on the go probably want to quickly jump to a certain category and then quickly navigate to the content they’re interested in. The tab bar means that even if they last left the app several levels deep in the Artists hierarchy,
they can still immediately tap the Audiobooks tab and start navigating that hierarchy instead. (Or they could even tap the already selected tab to jump to its top level and start over.) If you use a tab bar, it should provide the top-level navigation of the entire app. Tabbed views can contain navigation controllers, but not the other way around.

Deciding to go tabbed is a big choice, because that tab bar will be visible for most or all of the time people are using the app. You can offer only four or five top-level categories at a time. When you have more than five, the burden is on the user to decide which ones she thinks she’ll need most often; everything else gets hidden behind the More tab. Even if you have only a few tabs now, this decision can come back to haunt you when you want to add more of them later. Unless your app strongly benefits from the always-there top-level navigation provided by a tab bar, you might want to consider an ordinary navigation controller instead.

Every screen, except for some special screens dedicated to specific tasks (modal views, described shortly), needs to dedicate 49 points of height to that heavy black bar. (And even though you can tint it any color, you should keep it relatively heavy.) The tab bar’s size, color, and shiny highlighting effects give it a dominant role in the visual composition of a screen, compared with lesser elements like toolbars. That reinforces the significance of the tab bar’s role as the top-level navigation of the whole app, with the power to send users to different branches of hierarchy with only one tap.

**Segmented-Controls-as-Tabs**

From time to time, you may want to offer a couple of views of the same information, or variants of the same screen. You might be able to offer a switch between those options in the form of a segmented control that behaves like tabs. A proper tab bar controls the whole app, jumping from one top-level section to another, but this lightweight tab style (let’s call it segmented-controls-as-tabs) affects only the content or presentation of the current screen. As a result, you can offer two or three personalities for a single screen. If you have a popover to give quick access to more than one set of controls, such a segmented control may be the answer. iWork makes good use of this technique in its inspector popovers.

Don’t rely too heavily on this technique. It can feel arbitrary and confusing when a navigation scheme switches back and forth between the horizontal sliding of a navigation controller and the control-swapping of segmented-controls-as-tabs.

**Multiple Personalities**

Here is a way to gracefully cram a couple of completely distinct interfaces and navigation structures into one app. In response to the tap of a button (usually in an upper corner of the screen), the entire interface transitions to reveal a new interface, sometimes with its own navigation scheme. The most prominent example is iBooks, with its entire store-shopping experience presented on the reverse side of its reading experience, complete with a three-dimensional flip transition between the two. The big, fancy animation makes it feel as if you’re taking a major navigational leap—“going to the store”—without having to open a separate app. When you buy a book, it hovers in
midair as the app flips back to the bookshelf, thus logically connecting the two sides. It’s a fairly rare scheme, and you should use it only when you’re certain you need it, but it can be invaluable when you need to include two similar but distinctive interfaces in one app.

**Modal View**

You can use a modal view to handle a specific task that doesn’t quite fit into your ordinary navigation hierarchy. While the modal view is open, the normal navigation and functionality of the app are temporarily unavailable; the app is in a specific “mode,” hence the name. A classic example is composing a message in the Mail app. This mode is available from anywhere and has nothing to do with where you are in the hierarchy, so it takes you out of the hierarchy momentarily to deal with the task of writing a message. Then it drops you back where you were.

On iPad, you have several choices for presenting modal views (see Figure 3.5), each with its own personality.

- **Full screen**—The iPad screen is pretty big, so a full-screen modal view is a big deal. This option makes sense when users will be spending a lot of time in the mode and it’s OK for them to forget about the main app itself in the meantime. A full-screen modal view takes over the whole device, behaving like an app within an app, dedicated to a specific task. Think of things like a web browser built into a Twitter app. You could very well follow a link that someone tweeted and end up spending 45 minutes reading an engrossing article or watching a video; that experience needs to use the whole screen, and not relegate you to a fragment of the screen while the rest of the app peeks out at you in the background.

- **Page sheet**—This is a step down from the full-screen style, in that a page sheet has a constrained width. In portrait orientation, it looks like a full-screen modal view, filling the width of the screen at 768 points wide; but in landscape orientation, it’s still only 768 points wide, leaving some of the underlying interface visible but dimmed. The Mail app uses this style for its composition window. The resulting experience is close to the full-screen style. Users can easily spend a lot of time and thought writing an email, so they should be given a quiet, dedicated space to do so. But there is one key difference from the full-screen style: a page sheet keeps the interface from getting too wide. A 1,024-point-wide composition area would result in really long lines of text, a classic blunder of poor typography; the distance the eye needs to travel to go from the end of one line to the beginning of the next is too far, causing reading mistakes and cognitive fatigue. (See more about typography in Chapter 4.)

- **Form sheet**—This is a step down from a page sheet. A form sheet takes up only 540 × 620 points, hovering in the middle of the screen, with the rest of the interface dimmed. This doesn’t depart very far from the normal context of the app, so it feels more lightweight. It’s useful for tasks that need a bit of space but that you expect to take only a moment, such as entering account credentials for a web service, exporting a document, or changing appwide settings.
• **Current context**—Sometimes you need to present a mode inside an existing view, such as a popover or one side of a split view. Perhaps you want to allow access to the sidebar pane while a mode is under way in the content pane. Or you need to break out of the navigation hierarchy of a popover to take care of a special task. You shouldn’t need to use this style of modal view very often, but it’s good to know you have it for special situations.

• **Popover** (as an alternative to a modal view)—A popover isn’t usually a modal view, but you can use it to fulfill the role of one in a lightweight way. When you’re considering a modal view, ask yourself whether a popover might do the job better. A popover keeps users much more in context rather than pull them away from what they’re working on. (If you need to make a popover feel more modal, you can give it Cancel and Done buttons and disallow tapping away to dismiss it, but this design is falling out of fashion.) So if the task at hand is related to the existing interface on the screen, and doesn’t need all the space afforded by a modal view, try a popover instead.

On iPhone, modal views are simpler: they’re always full-screen, because anything smaller would be barely worthwhile (although the **partial curl** transition style does leave much of the existing screen in place). Most of the time you should use the straightforward vertical transition, which simply slides the modal view up from the bottom of the screen and then slides it away when the task is finished. Notice that the transition doesn’t push the previous screen out of the way, as horizontal navigation does; instead, it merely covers it up, thus promising that it’ll still be there when the mode is over. Users recognize and understand that type of transition as being a momentary diversion from normal navigation. Other transitions might leave users guessing about whether they’ve been transported to some other part of the app. See Chapter 6, The Prototypes, for more about transitions.

![Figure 3.5 Types of modal view. The “current context” type can appear anywhere; in this example it appears in a sidebar. A popover can be modal or nonmodal.](image-url)
**Popover**

A popover is a marvelous little iPad-specific element that seems unremarkable at first. It’s a little window with a triangular stem pointing at the object that summoned it. But the humble popover contributes great sophistication to the iPad experience, for a few subtle reasons.

- You don’t need to manage it like a window on the desktop. It can’t be moved, and it’s generally only as big as it needs to be.
- It appears at the time and place you need it.
- It disappears as soon as you’re finished with it, either because you reached the end of its little workflow or because you decided to stop using it and tapped away.
- It generally keeps you in the context of the surrounding interface so that the interlude of using the popover doesn’t interfere too much with the train of thought you had before opening it.
- It assumes that you’re editing in-place and that you want your work to be saved, unless you manually cancel or undo it. Simply tapping away from a popover shouldn’t discard your charges.

All this combines to make using popovers a lightweight experience. Consequently, interfaces can be designed to be “quieter” than they would be otherwise. Functionality can be easily summoned, used, and then put away for next time, rather than spread out all over the screen all the time so that you can get at it easily. (See Chapter 14 for more about quiet interfaces.)

Another exciting thing about popovers is that they can contain screens and navigation hierarchies all their own, distinct from what’s going on in the main app. You can think of a popover as a little iPhone sitting inside the iPad screen, with its own miniature app that deals with one concern of the greater app. Popover navigation can use navigation controllers, segmented-controls-as-tabs, and modal views.

**Custom Navigation**

So far, we’ve looked at the standard navigation methods offered out of the box with iOS. If you need to—and if you have the software engineering wherewithal at your disposal—you can create almost any navigation scheme that you dream up. But be careful. Lots of apps with unique needs present their experience in a clever, original way, but not all apps have those unique needs. Applying an unexpectedly distinctive design to a problem that could have been solved with a standard approach can backfire. Think carefully before deciding to use a clever and unique navigation scheme as a primary way of differentiating your app. (See Chapter 14 for advice about unique designs.)

The most important thing to keep in mind when thinking up new navigation schemes is that they should feel spatially consistent. The conventional navigation controller scheme works well in users’ minds because they can easily build a map of how
it works (often called a mental model.) Deeper levels are further to the right, higher levels are off to the left, and modal views slide in from off-screen and then slide away when they’re no longer needed. In iBooks, tapping a book on the shelf causes it to zoom toward the viewer and open. The iWork apps present a similar experience by showing a grid of documents that zoom up when you tap on them and zoom back when you’re finished editing. GarageBand is more ambitious. It starts with that same document grid but adds a carousel of various instruments, and a tracks overview is revealed by a vertical rotation effect. (See Figure 3.6 for a map of GarageBand’s navigation scheme.)

Figure 3.6  GarageBand on iPad has a custom navigation scheme that’s complex but consistent.
Even if users never consciously think about the mental map they’ve made of the app, they’ll be put off if that mental map is violated. Use smooth, sensible transitions every time the screen changes, especially when the changes are big ones. The simpler you can make the spatial representation of your navigation, the easier it will be to make it feel stable and consistent to users.

Advice on the Standard Elements

Elements are the building blocks that make up screens: views, controls, alerts, and so on. In wireframing, you just need to find the right elements and put them together on the right screens in the right arrangement. Of course, that is like saying that writing a best-selling novel is only a matter of picking the right words and putting them in the right order. You’ll need a lot of wisdom to put together great app designs, but to begin with you should make sure you’re familiar with the building blocks at your disposal.

As a rule, you should choose standard elements provided by the OS over building your own. For nearly every need, there’s a standard control that does the job reliably and predictably. Standard controls have the benefit of being familiar to users, who of course spend most of their time in other apps. Chapter 14, Consistency and Specialization, goes into more depth about the choice to go standard or custom.

For a basic understanding of the standard elements the platform offers, you should read the *iOS Human Interface Guidelines*. The beginning of the chapter said that, too, and so did Chapter 2. Really, you should read them. That will give you Apple’s official baseline position on how these elements are to be used. But for further advice and suggestions about how to put them to use effectively, based on actual Apple and third-party examples, here is a rundown of each individual element.

Bars

These basic screen-spanning bars show content and controls.

- **Status bar**—The only decision you need to make about the status bar is whether to hide it and, on the iPhone, what style it should be. The different states of the status bar mainly affect the immersiveness of the app; the more immersive treatments avoid distracting the user from the content being shown. The standard light status bar blends in with the interface. This works best when there’s no immersion necessary, as in communication or productivity apps. On the opposite end of the spectrum are media or entertainment apps, where you’ll want to hide the status bar to avoid distracting from the content. But don’t hide the bar just to show off how cool and immersive you can be. Users tend to care about the time and their battery level. Only if the immersiveness is more important than that information can you justify hiding the status bar. If you’re not hiding the bar, remember to accommodate 20 points at the top of each screen in your wireframes and mockups, and consider what your screens will do when the double-height status bar appears. (See The Worst-Case Height Compression Scenario in Chapter 4.)
• **Navigation bar**—This bar spans the entire top of the screen (on iPhone) or a specific view (on iPad). It’s 44 points tall, except in landscape mode on an iPhone; then it’s 32. A navigation bar is the primary way people move through the various screens. It sits at the top of the view, framing it with a reminder of where you are and how to go back up a level. Don’t confuse the navigation bar with a toolbar. Navigation bars are only about showing a back button, a title, and possibly a single (bordered-style) button for managing content (an Add button, an Edit button, a View button, etc.).

• **Toolbar**—This bar spans the entire screen (on iPhone) or a specific view (on iPad) and serves to contain controls. It’s 44 points tall, except in landscape mode on an iPhone; then it’s 32. If the navigation controller is the workhorse for moving between screens, the toolbar is the equivalent for choosing commands on screens. A toolbar can contain a limited number of controls, so you’ll need to be scrupulous about picking them. On iPhone, you’re limited to five 44×44-point buttons. On iPad there isn’t a hard limit, but you should do your best not to clutter the bar. Unlike desktop computer users, iOS users aren’t accustomed to wading through lots of controls. Instead you should think of ways to consolidate functionality behind a single button, by using popovers, modal views, and action sheets. (See Chapter 14.)

You have two choices of button style in a toolbar: regular and bordered. The bordered style is great for emphasizing that something is a button, when going borderless would make it ambiguous. A button with a text label is the prime example of something that needs an extra bit of emphasis. Another example is when there’s non-buttony stuff hanging around in the same bar, and that’s why a button in a navigation bar is always bordered; otherwise, it could be construed as being part of the screen title. The HIG advises against using both button styles in the same toolbar, but on iPad such a design isn’t a big deal. iWork keeps bordered buttons on the left side of the toolbar, and borderless ones on the right. That’s fine; just don’t mix the styles willy-nilly. (See Figure 3.7 for the example set by Keynote’s toolbar.)

• **Tab bar**—This is a 49-point-tall bar that spans the entire screen and always appears below the content it switches. (The purpose of a tab bar is described in detail earlier in this chapter, in the section called Tabs.) When you’re
wireframing for tab bars, remember they’re slightly taller than toolbars, and you can put only five tabs into one on iPhone. For a while, it was fashionable to make a custom tab bar that promotes one tab by having it protrude outward from the top of the bar. Instagram made that design famous. But the practice seems to be falling out of favor and doesn’t appear in that app anymore. Fads.

**Content Views**

These are the basic, general-purpose views for presenting content and controls.

- **Popover**—Earlier you learned why popovers are awesome, so here are a few tips for keeping them that way. For the most part, users expect popovers to be 320 points wide, for that iPhone-within-an-iPad feeling. Popovers much wider than that feel awkward hanging off that tiny triangular stem, and a form sheet would probably work better. Make popovers as tall as you need, of course, but no taller. When navigating among the screens of a popover, don’t worry if you leave empty space (because the popover is too tall for the content) or if scrolling is required (because it is too short). Such minor imperfections are better than making users sit through a resize animation every time they navigate somewhere. So just find a height that works reasonably well for all screens in the popover, and stick to it.

- **Split view**—Remember that the sidebar is always 320 points wide, mimicking the width of an iPhone screen and allowing you to use similar layout strategies for both sidebars and iPhone screens. Normally, the sidebar is always visible in landscape orientation; in portrait view, it is often hidden and needs to be slid in from the side. But thanks to the official Facebook app, it’s fashionable to offer a slide-in sidebar that is always hidden until summoned, regardless of the orientation. This design puts more focus on the content area, at the expense of quick access to navigation. The tradeoff is that it’s a bit nonstandard and thus is harder to implement and maintain. Weigh that against the importance of emphasizing the content in both orientations.

- **Table view**—Table views are the go-to element for displaying information, editable or otherwise. Table views appear in sidebars, main content areas, popovers, modal views...everywhere. They can be used for navigation (with chevrons or detail disclosure buttons), for selection (with checkmarks), or for data editing (usually using the value styles, described shortly). The name is a bit misleading: you might expect a table to offer multiple columns, but iOS table views are a single column (although each cell can display several bits of information). There are two standard table styles and four standard cell styles to choose from, each with a terribly nondescript name that makes it hard to keep them straight.

- **Text view**—Standard text views are straightforward. They’re good when you need to display, or the user needs to enter, lots of text. Of course, “lots” is relative.

- **Web view**—There is about one good reason to use a web view: to load an actual page from the web for tasks like logging in to sites or following a link in the user’s content. You formerly had to use web views to display rich text with
styling applied to it, because the standard text view was plain-text only; but as of iOS 6, you can put rich text in labels and text views. (Some apps try to put most of their functionality inside the web view, actually running a web app wrapped thinly in an iOS app. For advice on that particular venture, see the hybrid web-app admonishment in Chapter 14.)

Here are the styles of cells you can use in your table views.

- **Plain table**—This table style pushes the content all the way to the edges of the view; an example is the message list in Mail. A plain table is great for presenting a single, homogenous list of items, especially when the list is likely to get long. That's why you see it used for email messages, contact names, music tracks, to-do list items, or any other uniform collection of items that can get arbitrarily long. Even within such homogenous lists, though, there can be subdivision indicators, such as letters of the alphabet. In those cases you can use **section headers**, which float among the rows as little eye-catching milestones without being tappable. And if a sectioned list is likely to be really long, you can provide an index along the edge of the screen for quickly jumping to a certain group. The index tends to look silly or awkward if you don't have lots of data or if you use it for sections that aren't strictly ordered in a familiar way such as the alphabet.

- **Grouped table**—This table style keeps the content in self-contained stacks with rounded corners. An example is the Settings app in iPhone. A grouped table works best when you have heterogeneous kinds of information to present on one screen, when the separation of sections is critical, or when you offer controls and labels that don't fit into table cells. Imagine if the Settings app had a single huge plain-style table for all the settings. There would be no strong separation between the kinds of options; there would be no place to put the explanatory labels; and lots of sections would consist of a contrived header and a single item. See Table 3.1 for a comparison of plain and grouped tables.

<table>
<thead>
<tr>
<th>Plain Table</th>
<th>Grouped Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>It’s good for any length of list.</td>
<td>It’s best for shorter lists.</td>
</tr>
<tr>
<td>It’s good for homogenous data.</td>
<td>It works fine with wildly different kinds of data.</td>
</tr>
<tr>
<td>Labeled sections help delineate subgroupings like alphabetization.</td>
<td>Groups provide stronger separation than sections.</td>
</tr>
<tr>
<td>All sections need to be labeled.</td>
<td>Groups can be labeled or unlabeled.</td>
</tr>
<tr>
<td>All sections are expected to hold roughly equivalent kinds of data.</td>
<td>Each group can hold a different kind of data.</td>
</tr>
<tr>
<td>It can offer an index for jumping quickly to a section.</td>
<td>A single index doesn’t make sense, because there is more than one table on the screen.</td>
</tr>
</tbody>
</table>
- **Default cells**—The default cell style simply presents a text label and, optionally, an image. This style is generally used in navigation to help users identify the information they want to navigate to next. It's used to identify artists in the Music app, people in the Contacts app, and so on. The designers could have used the subtitle style to include plenty of extra data, but in these cases they recognized that almost every time, users want to navigate to the full detail screen. No single piece of information would be most helpful to promote from the detail screen to the cell, so they left the cell simple, encouraging users to delve into the detail screen. After all, taps are cheap, and it costs very little to delve inside and get a dedicated screen of information about the item you're interested in.

- **Subtitle cells**—This cell style includes smaller, gray text below the title to give additional detail about each item. It's useful when you need to let users compare a certain key piece of information between items without having to delve into the individual detail screens. For instance, the app list in the Notifications screen of Settings uses a subtitle to indicate the kinds of notifications set up for each app. Users can see at a glance the answer to their likely question: “Which apps have notifications enabled, and what kinds?” It's a clear case of a piece of information that saves people time and trouble when it's pulled up a level into the cell.

- **Value 1 cells**—Yep, that's the official name of this style. In the Interface Builder component of Xcode, it's called “right detail,” which is a bit more descriptive. This style combines a label and a value, usually to allow users to edit the value. The label is bold and left-aligned; the value is grayish-blue and right-aligned. This layout emphasizes the label text, because (at least in left-to-right languages), people find it easier to compare strings of text that are left-aligned; scanning down the beginnings of the labels is easy. You can also offer switches or other small controls in the value area instead of a text string. Value 1 cells are seen in various places throughout the Settings app, accommodating users' need to scan the names of the various options presented there, looking for the one they're interested in. Scanning the values wouldn't help that process very much.

- **Value 2 cells**—Also known as “left detail,” this style combines a label with a value that can usually be edited. The text label is deemphasized by its small size and its fainter color, whereas the value is dark and bold. In this layout, it's easy to recognize and compare the values, because they're left-aligned in the middle of the cell. This cell type works well when you have pieces of information that are more recognizable by their values than by their labels, and that's why it is used in the Contacts app on iPhone. Names, phone numbers, addresses, email addresses, and other bits of contact information are pretty recognizable without seeing the label, so you can easily scan the values on the screen and find the one you need. (See Figure 3.8 for a comparison of the various cell styles.)
Custom cell styles—You can put almost anything in a custom table cell. So if you need something a bit different (or very different) from the standard choices, and you have a good reason for it, you can create your own style. For an example, see the message list in Mail. Each row provides a large, bold sender name, a small, regular-weight subject line, a small gray message preview, and a blue time indicator. When you dig through your email looking for a specific message, all these bits of information are likely to be helpful; none of them could be omitted safely. To implement the message list using one of the standard styles would have done a disservice to the task of browsing email messages. (See Figure 3.9 for some custom table cell styles.)

Alerts
Alerts are an efficient, effortless way to drive your user crazy. For decades on the desktop and on the web, the alert was a way to pester users with whatever the developer felt like saying, whether the user cared or not. iOS did a lot to reduce the number of alerts that users have to deal with, reserving them for times when something is seriously amiss and the user needs to be notified or asked for immediate input. Consequently, when an alert appears it usually heralds an important moment. If it instead says something inane, banal, or cryptic, it waters down (or poisons) the purpose of alerts and reduces the attention users give them.

Here are some good times to use alerts:

- When the app can’t proceed unless users enters their account credentials
- When a background process encounters a problem, such as a sync conflict, that needs an immediate decision from the user

Figure 3.9 Left to right: Mail, Podcasts, Tweetbot, and Instapaper.
Custom table cell styles support any layout you can dream up.
The following are some terrible times to use alerts.

- When an operation completes normally. The interface should make this apparent without interrupting the user.
- When something the user just tried to interact with, still visible on the screen, has a problem. In that case, the thing itself should show its status.
- When you want to influence the user’s behavior (see Chapter 10).

Look at the animation that a new alert performs when it appears on the screen. It emerges from nothing, fades in, floats perpendicular to the screen toward the user, and then bounces back into place. Almost everything else that happens on an iOS device is connected to something already on the screen, giving it context and meaning. An alert is for those rare cases when something happening behind the scenes, something that normally proceeds without the user’s needing to care about it, suddenly needs the user to care about it.

**Action Sheets**

Action sheets are another humble but heroic player in iOS’s quest to make software quieter and more respectful. Action sheets are a stack of buttons that appears in response to an action by the user. Their effect is profound.

On iPhone, action sheets always slide up from the bottom of the screen and always offer a cancel button in addition to their action buttons. On iPad, action sheets might do the same thing in the virtual-iPhone-interface that is a popover, or they might appear in a dedicated popover of their own. If they appear in a popover of their own, there is no cancel button; instead, the rest of the screen serves as a safe place to tap away and cancel the action.

Here’s what is cool about using an action sheet.

- It hides several similar actions behind a single button, thus simplifying the interface until the moment of need.
- It doesn’t necessarily offer explanatory text, thus making it feel lightweight and reinforcing the fact that it always appears because of the last thing the user did. (You can add explanatory text if you really can’t get the situation across with the button labels.)
- It’s easy for a user to open, check the available options, and then cancel if none of them is what the user wanted.

An especially thoughtful case is the action sheet that shows a single action button. In old-fashioned interfaces, where an especially consequential action would be accompanied by a fussy dialog box with long explanatory text and buttons for proceeding or canceling, iOS offers a single, clearly labeled confirmation button. If you really intend to proceed, you can move your finger an inch and tap the action button. If not, you just tap away. That’s elegant.
Standard Controls

Most controls are fairly straightforward and are well documented in the HIG. Here are some tips for using them.

- **Activity indicator**—This is also known as the **indeterminate progress indicator**, or spinny. On iOS, these are much more common than progress bars. If something takes less than a few seconds, you just put up a spinny in a location connected to the work being done, and don’t bother the user with guesses about how much time is remaining. An activity indicator should suggest what it’s for by an associated text label, by its location on the screen, or both.

- **Date and time picker**—This is also known as the **wheels of time**. Nine times out of ten, this is the right way to get date input from users. Spinning to dates that are even decades away is quick and easy. If multiple fields are visible on the screen, this control helps highlight or otherwise call out the value that the wheels are editing.

- **Detail disclosure button**—Normally, when you need to delve inside an item to see more detail on another screen, you tap a table cell that has a chevron on the right side. A detail disclosure button serves as a backup “delve inside” tap target for times when you can’t follow that pattern. You might need to use it in two cases.
  - The item to be delved inside isn’t a table cell and thus isn’t obviously tappable for more information—for example, the bubble that emerges from a pin on the Maps app for iPhone, or a photo in Messages.
  - The table cell itself has some other function. In the Phone app for iPhone, tapping the cell for a favorite contact starts calling the person, whereas tapping the detail disclosure button delves into a detail screen about the person.

In that second case, adding a detail disclosure button makes the most sense when both choices are about equally likely and when the shiny blue button doesn’t compromise the cleanliness of your visual design. Another option is to split the two functions between normal mode and Edit mode.

- **Info button**—This venerable emblem, which is used on the desktop mainly for editing content details, is supposedly for revealing “configuration details” on iOS. For a while the same icon was used in the iWork apps on iOS to summon the style inspector popovers, but it has been replaced by a more expressive paintbrush icon. Meanwhile, the tools popover, which is closer to “configuration details,” is summoned from a **wrench** icon. Many third-party apps that have configuration screens opt instead for a gear icon, because the meaning of the info icon is so muddled.

- **Label**—This is an ordinary little string of text that you can use to…label things. Generally, it’s best to match the style and layout of the default labels on a grouped table view.
– To name something, put a bold label immediately (10–12 optical points) above it. Keep it on one line.
– To offer additional explanation about something, put a regular-weight label immediately (10–12 optical points) below it.
– To offer freestanding explanatory text that’s not related to a particular element, put some empty space (20–24 optical points) between it and the nearest controls.

You don’t need to label everything. A group of obviously color-related controls doesn’t need to be called “Colors.” The only table on a screen titled “Addresses” doesn’t also need to be labeled “Addresses.” Make sure you add labels only when they actually communicate something that wouldn’t get across otherwise.

- **Network activity indicator**—This activity indicator in the status bar informs the user of communication happening over the network. Users look here to see whether their network connection is being used, especially if they’re expecting some stale information on the screen to be updated. This indicator is a subtle hint to keep waiting, because the update is on the way.

- **Page indicator**—This is yet another quietly heroic interface element. Thanks to its presence on the home screen, this simple series of dots is immediately recognizable to most users as an invitation to swipe sideways for more content. It gives you the opportunity to display lots of screen-sized chunks of information without actually requiring navigation from screen to screen. (See Chapter 12 for more praise of pagination.)

- **Picker**—This generalized variant of the wheels of time is used for pop-up menus on web sites but is pretty rarely seen natively. Most times that you would use it, you could instead use a table view; the interactions of scrolling through a table view and of spinning the wheel are almost identical. The main benefit of the picker is that it lets you stay in context, and that’s why it works well on the web. (On a web site, you need to stay in context in order to see the identifying information around the pop-up, so you can’t just navigate to a dedicated screen for a table view. Nor can you insert an arbitrarily tall table view into a web site that wasn’t designed for it.) Another benefit is that it’s slightly lighter weight than a table, because the user simply scrolls to update the value; there’s no need to tap an item to select it.

- **Progress view**—This is equivalent to the old-fashioned progress bar seen often on the desktop. As mentioned in the description of the activity indicator, most of the time you don’t need a progress view. First, most operations should not take so long that you need to show the user how far along they are. Second, people need to see a progress bar only when they have no choice except to wait for the process to complete in order to get something done. Here are good examples of using progress views:
  - Waiting for a document to be downloaded from iCloud so that you can work on it
– Waiting for an iMovie project to be exported so that you can send it to someone
– Waiting for an email with heavy attachments to be sent so that you can make sure it succeeds
If the process usually takes less than a few seconds, or if waiting for it doesn’t affect the user’s ability to get work done, you’re probably better off with a spinny activity indicator.

**Rounded rectangle button**—This is the one general-purpose, standard style of button that you can place in content areas. When you’re using buttons in the content area, make sure you use them for *actions*. Don’t use them for the following other purposes (as always, unless you have a good reason to).
– Navigation is usually better handled with table cells bearing chevrons or detail disclosure buttons.
– Choosing from a number of options is usually better handled by a table view with checkmarks or a segmented control.
– Toggling a setting on or off is usually better handled by a switch.

**Search bar and scope bar**—These are handy when a screen shows a number of items and it takes more than a few moments to scroll through them and find one manually. A common trick is to include the search bar at the top of the content area and load the screen so that it’s scrolled just out of view. This design lets people scroll to the search bar if they need it but otherwise leaves it tucked away out of sight.

**Segmented control**—A segmented control is a concise way to offer a very short list of mutually exclusive options. Often, it’s a headache to come up with good labels that fit inside the narrow buttons, and you should use a table view with checkmark selection instead. The following are some good uses for segmented controls.
– Selecting from a handful of options, if you can get your point across with recognizable images or very short text labels. You can even offer a label just above or below the segmented control that updates to reinforce the current choice with a text description.
– Providing options that show or hide other controls based on the setting. The visual weight (see Chapter 4) and experience weight (see Chapter 13) of the content-area style of segmented control lend it well to this use. Pushing a big segment and watching it highlight in intense blue feels appropriately consequential to the subsequent appearance or disappearance of controls.
– Switching between views on a screen in the same way tabs do. See Segmented-Controls—as-Tabs earlier in this chapter.

Whatever you do, don’t make a segmented control behave like a button. It’s for choosing between options, and not for performing actions. And remember that
Advice on the Standard Elements 51

for simple on/off toggles, you have the switch at your disposal; a segmented control with “on” and “off” segments doesn’t make a lot of sense.

- **Slider**—This is a great way to provide quick control over a continuous setting when the actual numbers aren’t very important. Excellent examples in the operating system are the brightness and volume sliders. Nobody ever thinks, “I could hear better if the volume was at 86% right now” or, “It’s getting dark; I should turn the brightness to 39%.” Instead, they think, “Quite a bit louder” or, “A little dimmer.” That’s the sort of thing sliders excel at. People don’t know exactly what setting they need beforehand; instead, they need continuous feedback while moving the knob. So make sure users can see or hear the result as they move the slider. It’s frustrating having to go somewhere or do something to get the feedback they need.

- **Stepper**—A stepper is good for numerical settings when the number matters but adjustments tend to be within a small range. Poking the plus or minus button until you see the number you want is a cognitively cheap interaction, compared with typing in a number. A setting that was almost always set to 1, 2, or 3, for instance, would work well with a stepper.

- **Switch**—You can put a switch in a table cell to offer a simple on/off toggle. Flipping a switch feels fairly weighty, so you can easily use it for consequential settings or let it show or hide other controls. Make sure that the two opposing settings are easily gleaned from looking at the label. Something like “Automatically download new items” goes well with a switch. A vague title like “Horizontal Layout,” where the opposing option is not immediately apparent, is not as good. (That would be better served by a segmented control with the label “Layout” and segments called “Horizontal” and “Vertical.”)

- **Text field**—The text field is commonplace on the desktop and the web, but it often feels a bit crusty on iOS, especially if it’s just sitting in a content area, lacking any placeholder text. Sometimes, you can offer a better way of entering information, such as picking from a table view. Text input is even more of a pain on a touchscreen than on a physical keyboard, so avoid it when you can. In content areas, a table cell with text input enabled is often more attractive. But when you need text input and you can’t use a table cell, then a text field makes sense. (See a comparison between text fields and table cells in Figure 3.10.)

![Figure 3.10](image.png)

Figure 3.10  Text fields (left) look dated, feel cramped, and are hard to balance. A table cell with text input enabled looks nicer, and using placeholder text instead of a value cell style leaves more room for typing.
Custom Controls

If you work on a sufficiently complex software project for a long enough time, you’re bound to run into cases where the standard building blocks don’t quite provide the best experience you can imagine. This section isn’t about customizing the appearance of standard controls; that’s a topic for the mockups phase in Chapter 5. Instead, it’s about making controls that behave in a new way.

Most of the time, you can and should base your custom control on a standard one. There’s likely to be a standard control that kinda does what you want but that you need to tweak. By carefully adjusting its characteristics to satisfy the interaction you have in mind, you can keep your custom control as close as possible to the spirit of the platform.

Of course, it’s possible to create an entirely original control from whole cloth, without basing it on anything that came before. If you and your team can pull this off perfectly, you’ll be heralded as UI design heroes. If you execute it anything less than perfectly, however, it’ll come off as awkward and painful to use.

Suppose you want to provide a long list of options in a quick, easily browsed way, without taking up a lot of space. These options are easily represented by small square icons, so a big tall table view with labels would be overkill. But a segmented control can’t hold all the options you’re planning. What kind of custom control could you create to do the job?

Start by looking at the standard controls that offer a way to pick from a list. The picker control’s vertical spinning wheels do a good job of offering a lot of options without taking up much space. What if you could adapt the picker concept to a more compact, icon-based set of options? Presenting…the **horizontal mini-picker**. (See Figure 3.11.)

This control can hold any number of options, as long as they’re represented by distinctive, square icons. It takes up only as much space as a single table cell row, thanks to its horizontal orientation. And it’s immediately familiar, because it takes advantage of an existing control’s interaction metaphor. In fact, to typical users, it probably wouldn’t be obvious that the control was custom made. For all they know, it’s simply a standard control offered by the operating system. That’s great! Blending in as a natural part of the platform is a noble goal.

There are plenty of ways a control can give away its custom status. If you miss any of the following considerations, your control is a lot less likely to be perceived as fitting in with the rest of the system.

---

**Figure 3.11** A horizontal mini-picker for choosing fill styles in OmniGraffle. It tweaks the concept of a vertical picker for a slightly different purpose.
Exercises 53

- How does the control react to various gestures—tap, double-tap, touch and hold, drag/swipe, and the like?
- What happens if you accidentally touch the control and then try to drag your finger away without letting go? (On a standard button, this lets you cancel a mistap.)
- How does the control adapt to different amounts of available space, especially when the orientation of the device changes?
- What does the highlight look like while you’re in the middle of tapping the control?
- How does the control work with accessibility features, especially VoiceOver?

Summary

iOS offers a healthy collection of carefully thought-out building blocks that you can use to craft your own navigation hierarchies and screens. Plenty of life-improving apps can be built using only these standard elements and navigation schemes. But if you need to, you can build your own custom navigation and custom controls. Just make sure that anything you create conforms to the spirit of the platform.

Now that you’ve familiarized yourself with the toolkit available (by reading the HIG and this chapter), you’re ready to start building wireframes in earnest.

Exercises

It’s time to try out your new knowledge. Give these exercises a shot to solidify your familiarity with the standard iOS elements and your understanding of when to customize beyond them. Do each one a few times, choosing a different example for each iteration, if you like.

1. Think of a single feature in your own app. What screens and elements might you need? Sketch out a couple of approaches using different kinds of controls to see which one feels right.

2. Choose a standard control. Imagine how you could design a custom version of it that serves a slightly different purpose. Can you make it more precise (or less, if that’s what is needed), more compact, or more expressive? What purpose would your custom control serve better than any existing standard control?

3. Draw the geography of your app the way GarageBand’s is drawn in Figure 3.6. Can you make spatial sense of the navigation scheme you’re using?
This page intentionally left blank
1-D layouts, 76
1+1 = 3 effect, 94
1.5-D layouts, 76
1Password app, 281
2-D layouts, 76

A
Accessibility, 213–215
Accessibility Programming Guide for iOS, 214
Accounts, Mac Mail, 132–133
Acorn, graphics tool, 86
Action sheets
confirming actions with, 190–191
cost for buttons, 99
overview of, 47
paying attention to context with, 247
using hue for, 286
Activity indicator, as standard control, 48
Adaptation, invisible status of apps and, 180–181
Administrative debris, UI paraphernalia as, 238
Aesthetics (rich and plain)
color vs. monochrome, 286–290
depth vs. flatness, 290–296
exercises, 302
overview of, 285–286
realism vs. digitality, 296–301
summary review, 301
Affordances, 169–170
Alerts
animating with motion sketches, 115
appropriate use of, 190
avoiding annoying, 216, 218
delivering important text message with, 173
overview of, 46–47
showing contextual status with badges, 179
Alignment, 66
Alpha software, 121–123
Anatomical components, of elements, 90
Animations, 115–117, 161–163
Annotations, 19, 25
Antialiasing, 59
Antirequirements
keeping rejected ideas as, 6
pruning features for focused apps, 226
specifying in versatile apps, 235–236
specifying what app is not for, 9–10
App Store
creating market for life-improving software, 197
couraging experimentation, 278
listing app in, 202–204
listing app name in, 199
release notes for, 209–210
tap target size for purchase button in, 161
App Store Review Guidelines, 204
Appearance. See Aesthetics (rich and plain)
Apple, xix, xxii–xxiii
Architecture outline, 13, 20
Architecture sketches, 20
AssistiveTouch, 214–215
Attention
budget, 237–238
respecting user, 215–218
Autosave, manual, 263

B
Back button, 33
Background
adding underhighlights to, 105
safe hues for, 286–287
using alerts for processes in, 46
Background contrast
overview of, 92–93
presenting image content, 81
with visual weight, 64–65, 90–92
Badges, for contextual status alerts, 179
Balance, as layout principle, 71
Balsamiq tool, prototypes, 118–119
Baseline, measuring text optically, 59
Basic scale, rhythm in layout, 68–69
Behavioral level of cognition, defined, 167
Betrayal of trust, 216–218
Binocular vision, 291–293
Blank slate, 267–268
Blending modes, applying gradients, 103
Borders
1+1 = 3 effect in, 94
applying understated layout to, 72
for contrast and visual weight, 90–92
toolbar button, 42
Bounds, optical measurements and, 58–60
Branding, with certain hue, 287–288
“A Brief Rant on the Future of Interaction Design”
(Victor), 146
Brightness
HSB color model and, 87–88
perceived as value, 88–89
slider, 51
using, 289–290
Index

304

Brushes app, 171

Bug reporting
  overview of, 121–123
  using tickets in bug-tracking database, 5

Buttons
  avoid making segmented controls behave as, 50
  for contrast in action sheet, 99
  generous tap targets for, 159–161
  rounded rectangle, 50
  styling communication cues, 84
  styling instantaneous feedback, 147–149
  styling with understatement, 72

Buttons, custom
  bevel, 104
  contents, 106
  fill color, 102
  gradient, 102–103
  overview of, 100–101
  shape layer, 101–102
  stroke, 103–104
  texture, 105
  underhighlight, 105–106

C

Calendar app
  orientation on iPhone, 78
  ornamentation in iPad, 299
  replicating office supplies, 298
  resourcefulness of, 183
  scaling back in iPad, 228

Camouflage, contrasting objects to avoid, 89

Canvas, 100–101, 107

Cap height, 59, 70–71

Capability, conveying
  App Store listing, 202–204
  icon, 199–202
  interface interaction design for, 184–185
  launch image, 202
  name, 199
  overview of, 198
  price, 205–206

Cargo cult design, avoiding, 277

Case study. See Mail app, case study

Cell styles, content views, 44–46

Center alignment, in layout, 67

Center case, versatile app design, 234–235

Characters, principles of typography, 73

Chrome, UI paraphernalia as, 238

Clarity, from text and visual weight, 250–251

Clock app, 78

Clock screens, 33

Coherence, of animation, 117

Color
  customizing with tints, 279
  fill, 102
  HSB, 87–88
  perceived brightness of, 88–89

  RGB, 86–87
  styling communication with, 84
  styling contrast and visual weight with, 89–92
  vs. monochrome, 286–290

Colors, programmer’s, 288–289

Columns
  in 2-D layouts, 77
  principles of typography, 73

Commands
  Mac Mail, 133–134
  Mail for iPhone, 136, 138

Communication
  breakdown of, 176–177
  as styling attribute, 84

Communication apps. See also Mail app
  adding friction to protect user, 258
  immersive status bar for, 41
  on mobile platforms, 128

Companion apps, 129–130

Competitive analysis, in outlining, 7–8

Complexity of design. See Versatile apps

Comprehensive documentation, 206–209

Conciseness, of written text, 174–175

Connotation, 168–171, 172

Consistent design
  avoiding cargo cult design, 277–278
  difficulty of novelty apps, 282–283
  exercises, 284
  getting the most of HIG, 272–273
  guidelines, 271
  how it all works out, 271–272
  overview of, 273–275
  precedents, motifs, patterns, and shorthands, 275–276
  specialization vs., 272

Consumption-oriented apps, full-screen mode, 249

Contacts app
  attaching commands to objects, 138
  resourcefulness of, 183
  respecting user data, 216
  value 1 cells for, 45

Content
  2-D layouts and, 77
  adding to custom button, 106
  adding to mockups, 1–6
  bright elements stealing from, 289
  designing layers for, 108
  information density and, 75
  layout of controls vs., 74
  neutral interface of apps focused on, 286
  presenting controls in areas of, 75
  presenting with split view navigation, 33–34
  rounded rectangle button in areas of, 50
  styling for contrast and visual weight, 90–92
  transparency, and reading of, 94
  views, 43–46

Contents, anatomical component of an element, 90
Index

Context, iOS paying attention to, 246–247
Contextual controls (documentation), 178
Contextual inquiry, in outlining, 7
Contextual menus
  paying attention to context with, 247–248
  providing guidance, 178
  sketching interactions for, 26
Contextual status, 179–180
Contour, 89
Contrast
  brightness for, 289
  designing layers with, 108
  examples of, 97–99
  importance in visual design, 89
  measuring images/controls optically, 60
  posterizing to evaluate, 95–97
  transparency for, 93–94
  using low internal background, 92–93
  visual weight for, 64–65, 89–92
Controls
  in content areas, 75
  custom, 52–53
  designing for layers, 108
  guidance at point of need for, 178
  hiding vs. disabling, 248
  instantaneous feedback, 147–149
  layout of content vs., 74
  measuring optically, 60
  segmented-controls-as-tabs navigation, 36
  sketching on-screen, 22–24
  sketching workflow, 26–29
  standard, 48–51
  text label with icon for crucial, 176
  tints for customizing, 279
  toolbar, 42
  understatement for, 72
  undo for, 187, 189
  viewing gradient, 103
Conventions, design
  conscientious divergence from, 279–280
  harmless distinctiveness from, 279
  overview of, 271–272
Conversational documentation, 210
Conversations, sketching during, 16–18, 20
Converbot app, 281
Copycats, design, 277–278
Credentials, signup experience, 260–261
Cross-platform
  case study of Apple Mail, 131–141
  evaluating virtues of all platforms, 127–129
  exercises, 142
  outlining, 130–131
  overview of, 127
  standalone, mini, and companion apps, 129–130
  starting from scratch, 130
  summary review, 141–142
Cues
  adding friction with scary, 259
  combining imagery/text with, 176
  false, 171
  as guidance, 265
  interaction, 169–171
Current context modal view, iPad, 38
Curves, animation, 162–163
Custom
  appearance, 279
  buttons, 100–106
  cell styles, 46
  controls, 52–53
  navigation, 39–41
D
Data, respectfulness of user, 216–218
Date and time picker, 48
Dead-end (rejected) ideas, 6, 17
Decision fatigue, human attention budget, 238
Deep prototypes, 119–120
Default cell view, information in, 44–45
Defensive design, 185–187
Delay, 147–149
Delete button, 160, 190–191
Delete Contact button, Contacts app, 65
Delicious Generation, 278
Demoting features, 243–246, 259
Denotation, 167, 172–174
Depth vs. flatness
  extreme examples, 293–296
  lighting, 291–293
  overview of, 290–291
Design apps, 119
Design bugs, 121
The Design of Everyday Things (Norman), 170
Design specification, in outlining, 5–6
Desktop apps, Mac Mail, 132–134
Desktop computers, mouse-based input on, 158–159
Detail disclosure button, 48
Devil’s advocate, in sketching, 22
Dictionary, 182
Diet Coda web editor, iPad, 263–264
Digitality. See Realism vs. digitality
Dimension lines, in wireframing, 62
Dimensionality, and layout, 76–77
Disabling controls, 248
Disappearing interfaces, 248–249
Distance, layout principle of, 66
Distinction, layout principle of, 64
Division of labor
  scaling back features, 228, 230
  software design philosophy, 266–267
Documentation
  bugs, proof-of-concept software, 121–122
  characteristics of good, 210
Index

Documentation (continued)
- comprehensive, 206–207
- problem-solving, 207–208
- release notes, 209–210
- tutorials, 208–209
  - in usability testing, 125
Double-taps
  - overview of, 152
  - single-taps vs., 148
  - zooming to 100% with, 154
Drag
  - creating realistic, 154–155
  - hysteresis and, 156
  - pull-to-refresh threshold in Mail using, 158
    - as reliable gesture, 152
Drag to Move, 156–157
Drag to Resize, 156–157
Drop shadows
  - communication of, 84
  - home screen icons with, 202
  - overview of, 291–293
  - underhighlight effect with, 105–106

E
- Ease-in animation curve, 162
- Ease-in/ease-out animation curve, 116, 162–163
- Ease-out animation curve, 162
- Edge alignment, layout, 66–67, 70
- Edge cases, 233
- Edit mode, as visible status, 179
- Editing-oriented apps, full-screen mode on, 249
- Elements
  - action sheets, 47
  - adding depth to give permanence, 291
  - alerts, 46–47
  - applying styling to. See Styling
  - bars, 41–43
  - content views, 43–46
  - creating paper prototypes, 113–114
  - standard, 41
  - standard controls, 48–51
  - titling, 172
  - understatement of, 71–72
- The Elements of Typographic Style (Bringhurst), xxvii, 69, 74, 271–272

Email
- avoid exposing underlying mechanisms of, 261
  - ramifications outline for, 11
  - reducing friction in, 260
- Engineering bugs, 121
- Ethos, cultivating a good reputation, 215
- Experience weight, and friction, 257

F
- Failed feedback, 147
- Failed inputs, 146–147
- Fair app pricing, 206
- Fallback gestures, 154
- FAQs, as problem-solving documentation, 207–208
- Feature creep, Mac Mail, 132–133
- Features
  - avoid exposing underlying mechanisms, 261
  - complexity vs. usefulness of, 231–232
  - comprehensive documentation of, 207
  - grouping/arranging, 243–245
  - iOS and, 11–12
  - Mac Mail, 132–134
  - placing usefulness, 238–239
  - promoting/demoting, 243–245
  - pruning for focused apps, 225–228
  - reducing problems, 12–13
  - scaling back for focused app, 228–230
  - streamlining on Mail for iPhone, 134, 138
  - versatile design for, 233
- Feedback. See also User feedback
  - giving instantaneous, 147–149
  - keeping out of hand shadow area, 150–151
  - moment of uncertainty caused by lack of immediate, 147
  - realistic gestures and, 154–155
- Figure/ground relationship, contrast and, 89
  - Fill color, mockups, 102
- Find My Friends app, 279, 299
- Fitts’s Law, tap target sizes and, 161
- Five Whys process, 197–198
- Flatness vs. depth
  - extreme examples of, 293–296
  - lighting, 291–293
  - overview of, 290–291
  - tastefulness of flat interfaces, 85
- Focused apps
  - consolidating features, 226–227
  - designing, 224–225
  - example app, 228–230
  - exercises, 236
  - as forthcoming or quiet, 223–224
  - iOS love of, 225
  - pruning features, 225–227
  - real-world goals of, 225
  - saving feature for later, 227
  - scaling back features, 227–228
  - summary review, 236
- Forgiveness, user error
  - confirmation, 190–191
  - overview of, 187
  - undo, 187–189
- Form sheet modal view, iPad, 37
- Forthcoming interface design
  - adjacent in space, 238–239
  - disappearing interfaces, 248–249
  - example of, 252–253
  - exercises, 253–254
of focused and versatile apps, 223–224
  grouping/arranging features, 242–243
  hiding vs. disabling controls, 248
  overview of, 237–238
  paying attention to context in, 246–248
  progressive disclosure, 240–241
  promoting/demoting features, 243–244
  quiet design vs., 237
  splitting difference of features, 246
  stacking in time, 239–240
  summary review, 253
taps, 250
text and visual weight, 250–251
Friction
defined, 255
depending on context weight and, 257
how to add, 258–259
modulating app, 270
reasons to add, 257–258
slope of difficulty curve and, 255–257
summary review, 270
unintended, 259–264
Full-screen mode, 37, 249
Functionality
  adding friction for changes to, 258
  complexity of design, 223–224
  consolidating in focused apps, 226–227
  of popover navigation, 39

G
GarageBand app
  aggressive use of depth, 294–296
  custom navigation scheme, 40
  help overlay documentation, 208
  orientation on iPhone/iPad, 78
  simulation, 300–301
Gear icon, 48, 172
General preferences, Mac Mail, 132
Guides
  measuring pixels with, 61
gestures, 154
  hysteresis of, 155–157
  introducing one novel interaction per app, 280–281
  keeping feedback out of hand shadow, 150–151
  realistic, 154–155
  sandwich problem, 153–154
  six reliable, 151–153
  thresholds and, 157–158
Graceful interface. See Interface, crafting graceful
Gracious interface. See Interface, crafting gracious
Gradients, 102–104, 291–293
Graphics software, sketching with, 19
Groups
  measuring pixels with, 61
  using 2-D layouts, 77
  wireframing, 62
Grouped table views, 44, 66
Grouping
  by meaning, 242–243
  with usefulness stacked in time, 240
Guidance. See also Friction
  among more options, 265–266
  modulating app, 270
  one option, 263–264
  at point of need, 177–178
  sensible defaults, 266–269
  summary review, 270
  zero options, 262–263
Guidelines, design
  overview of, 271–272
  using HIG. See iOS Human Interface Guidelines (HIG)
Guides
  measuring pixels with, 61
testing alignment of layout with, 67–68
H
Hand shadows, 150–151
Handbook of Usability Testing (Rubin and Chisnell), xxvii, 125
Hardware display, 56
Hardware prototypes, 114
Help overlay documentation, 208
Helvetica Neue typeface, 59, 73
Helvetica typeface, 73
Hiding
  controls, 248
  status bars, 41
Hierarchical navigation view
  iPad, 139
  Mac Mail, 133
  Mail for iPhone, 135
  of navigation controllers, 31–34
  sketching interactions for, 26
HIG. See iOS Human Interface Guidelines
High contrast, posterization process, 97
High fidelity prototypes, 112, 118–120
Hints, coexisting with interface, 208
Horizontal slide animation, navigation controllers, 33
HSB color model
  action sheet contrast, 99
  brightness, 289–290
  hue, 286–288
  overview of, 87–88
  saturation, 288–289
Hue
  feelings associated with, 286
  HSB color model and, 87–88
  perceived brightness of, 88
  using, 286–288
Human Interface Guidelines. See iOS Human Interface Guidelines
Hysteresis, 155–157
Index

iA Writer, disappearing interface of, 249
iBooks app
  custom navigation scheme, 40
disappearing interface, 249
experience weight, 257
interfaces/navigation structure, 36–37
internal background, 92–93
page metadata contrast, 99
presentation of image content, 81
smart approach to brightness, 289–290
transparency of toolbar buttons, 93–94
iCab app, 223
iCloud, 227–228
Icons, conveying capability via, 199–202
Ideo Method Cards, xxviii, 8
iLife design, 278
Illustrative documentation, 210
Illustrator, as mockup tool, 86
Image resources
  creating mockups using canvas, 100–101
  creating mockups with Paintcode, 86
  creating mockups with resizable, 107
  creating Retina versions of, 107
  exporting for mockup assembly, 106–107
Image Size command, 107
Images
  App Store listing, 202–204
  combining with cues and text, 176
  for interface interaction design, 171–172
  launch, 202–203
  margin and padding guidelines, 70–71
  measuring optically, 60
  presenting, 95
Immersion, 41, 145–146
Inconvenience hand-off, scaling back features, 227–228, 230
Indeterminate progress indicator (spinny). See Spinning indicator
Indexes, in plain table view, 44
Info button, as standard control, 48
Information density, and layout, 75
Inner bevels, 291–293
Inner shadow, 291–293
Input
  creating suspension of disbelief, 145–146
  failed, 146–147
  improving using hysteresis, 155–157
  instantaneous feedback for, 147–149
  mouse-based vs. touch, 158–159
  outlining, 6–8
  streamlining, 261–262
Insert popover, 227
Insight, from users, 7
Inspiration, xxvi–xxviii
Instapaper app
  interface adjusting for time of day, 183
  novel interaction, 281
  quiet presentation, 223
Interactions
  cues, 169–170
  design precedents for, 275
difficulty of novel, 282–283
  introducing novel, 280–281
  sketches, 24–26
  styling precedents, 84
  suspension of disbelief in touch-based, 146
  updating original, 276
  usability testing for, 123–124
Interactive prototypes, 55, 112, 118–120
Interface
  constraining width using page sheet, 37
  creating paper prototypes, 113–114
  including two in one app, 36–37
  layers, 66, 74–75, 108
  modal view navigation and, 37–38
  orientation on iPhone, 77
  paraphernalia, 238
  plotting out screens, 56–57
  sketches, 22–24
  tastefulness, 85
Interface, crafting graceful
  defined, 145
  example app, 163–164
  exercises, 164–165
generous taps, 158–161
hysteresis and, 155–157
instantaneous feedback in, 147–148
layout, 149–151
meaningful animation, 161–163
moment of uncertainty, 146–147
realistic gestures, 154–155
sandwich problem, 153–154
six reliable gestures, 151–153
summary review, 164
suspension of disbelief, 145–146
thresholds, 157–158
using exotic gestures as shortcuts, 154
Interface, crafting gracious
  capability, 184–185
  communication breakdown, 176–177
contextual status, 179–180
cues, 168–171
defensive design, 185–187
denotation and connotation, 167–168
example app, 191–193
exercises, 193–194
forgiveness, 187–191
guidance at point of need, 177–178
imagery, 171–172
invisible status, 179–183
overview of, 167
redundant messages, 176
sense of adventure, 183–184
summary review, 193
text, 172–174
visible status, 178–179
writing, 174–176

Interior, anatomical component of an element, 90

Internal contrast, 92–93, 97

Interviews, outlining using input from, 7

Invisible status, 180–182

iOS Human Interface Guidelines (HIG)

80 percent solution for defensive design, 186
Apple developer site, 21
Apple’s icon guidelines, 200, 202
button styles within same toolbar, 42
design guidelines, 271–272
getting most out of, 272–273
iPad and iPhone tap targets, 161
iPhone tab bar limits, 23
resourcefulness, 182–183
standard controls, 48–51
standard system imagery, 172

custom controls, 52–53

elements. See Elements exercises, 53

navigation scheme. See Navigation overview of, 31

summary review, 53

iPad

target views, 38
Music app tab bars on, 35

navigation controller, 32–33

sketching interface for, 21–24

sleek/lean apps of, 11–12
tap target sizes, 161

worst-case height-compression scenario, 78–79

iTunes app

contextual status in, 180
localization and, 212–213

specialized design of, 278

iWork apps

custom navigation scheme, 40

disabling vs. hiding undo in, 248

fallback gestures of, 154

interactive tutorial of, 208–209

precedent for browsing local documents, 275–276

redo feature in, 188

scaling back features in, 227–228

segmented-controls-as-tabs navigation in, 36
templates, 268

versatile design of, 230–231

Jobs, Steve, xxii, xxvi, 82, 128, 134, 138, 259, 278

K

Kaleidoscope tool, 8

Keyboards branch, navigation controller hierarchy, 32

Keynote app

animation curves in, 162–163

building wide prototype in, 120

consolidating functionality in, 227

good guidance of, 266

handling orientation on iPad, 78

interactive prototypes with, 119

prototyping animations in, 116–117
templates, 268–269

versatile design using, 230–231

L

Labels

creating paper prototypes, 114

for groupings, 242

scaling back for focused app, 230

as standard controls, 48–49

text used for, 172

value 1 and 2 cells emphasizing text, 45

Lag time, realistic drag and, 154–155

Landscape orientation

on iPad, 78

on iPhone, 77–78

keeping platform in mind while sketching app, 21

page sheet modal view in, 37
Language
  localizing app, 211–213
  using resourcefulness for, 183
  worst-case height-compression scenario, 79
Launch image, 202–204
Layer Vault tool, 8
Layers
  depth styling hinting at, 291
  designing for, 108
  dimension lines and, 62
  interface, 66, 74–75, 108
  mockup assembly with, 106–107
  shape, 101–102
  thinking in, 74–75
  transparent, 93–94
  as wireframe tool, 62
  Wizard of Oz prototypes in, 114–115
Layers palette, 102–104
Layout
  alignment, 66–68
  balance, 71
  consistent design for, 274
  content and controls, 74
  for graceful interface, 149–151
  localizing app and, 212
  margin and padding, 70–71
  overview of, 63
  proximity and distance, 66
  rhythm, 68–69
  similarity and distinction, 65
  understatement, 71–72
  unity, 63–64
  visual weight, 64–65
Left detail (value 2 cells) style, 45, 51
Letterpress app, as flattened, 294
Life-improving software, iOS, 197
Lighting effects, and depth, 291–293
Linear animation curves, 162–163
Linguistic gimmicks, avoiding in localization, 212
Links, consistent design for, 274
Lion, Mail on, 140–141
LiveView, for interactive prototypes, 119
Localization, 183, 211–213
Location Services, respecting user data, 216
Logos, 84, 287–288
Loudness, with text/visual weight, 250–251
Low fidelity prototypes
  defined, 112
  interactive prototypes as, 118–120
  paper prototypes as, 112–114
Mac
  designing Mail for, 140–141
  going cross-platform with, 128
  specialized design of, 278
Mac OS X Leopard, and Mail, 131–134, 137
Mail app
  1.5-D message list in, 76
  adaptation of, 180–181
  depth cues in, 291
  guidance at point of need in, 178
  handling orientation on iPhone, 78
  learning of, 182
  paying attention to context in, 247
  pull-to-refresh threshold in, 158
  split view navigation, 33–34
  text used for unread messages on, 172
  undo feature in, 188
  using page sheet modal view, 37
Mail app, case study
  back to the Mac, 140–141
  implementing on different platforms, 131
  iPad, 138–139
  iPhone, 134–138
  Mac OS X Leopard, 131–134
Maps app
  detail disclosure button on iPhone, 48
  double-tap in, 152
  navigation on iPhone, 33
  rotate in, 153
  sandwich problem of, 153–154
Margins, as layout principle, 70–71
Marketing
  creating preemptive demo videos for, 118
  evaluating proof-of-concept software for bugs, 122
  of iOS gestures, 153
Master/detail approach, with workflow sketches, 26
Matte surface, mockups, 105
Meaning, grouping by, 242–243
Meaningful animation, 161–163
Measurement, 58–61
Medium contrast, in posterization, 97
Mental model, 40–41
Mental sweep, outlining using, 6–7
Menus, Mac Mail, 133–134
Message list screen, Mail for iPhone, 136
Messages
  redundant, 176
  rewriting, 175
  writing text, 174–176
Messages app, conscientious divergence of design in, 280
Metaphors, mimicking real objects, 297–298
Mini apps, 129–130
Mission statement, App Store listing, 204
Mobile platforms, going cross-platform, 128
Mockups
  assembly, 106–107
  backgrounds, 92–93
  color for, 86–88
  color vs. monochrome, 286–290
contrast, 89–92
contrast, evaluating with posterize, 95–97
contrast, examples, 97–99
creating button, 100–106
designing for layers, 108
exercises, 109
overview of, 81
pixels and, 57
presenting image content, 95
resizable images, 107
retina resources, 107–108
sketches vs., 19
styling, 82–85
summary review, 109
tools for, 85–86
transparency, 93–94
value, 88–89
when to create, 81–82
when to skip, 82
Modal views
context and, 248
manually undoing interactions, 189
for motion sketches, 115
presenting, 37–38
Modes
hues for, 286
as visible status, 179
Modular scale, 69
Monochrome, color vs., 286–290
Motion sketches, 112, 115–118
Mouse-based input, vs. touch, 158–159
Multiple personalities, 36–37
Multiple-user support, rarely offered in iOS, 12–13
Multithreading, contextual status and, 180
Music app, iPhone
presenting image content, 95
tab bar, 29, 35–36
volume knob lighting, 292
Mystery meat navigation, 172

Naming
apps, 199
groupings, 242
Navigable documentation, 210
Navigation
customizing, 39–41
modal view, 37–38
of multiple interfaces in one app, 36–37
mystery meat, 172
navigation controllers, 31–34
overview of, 31
popovers, 39
segmented-controls-as-tabs, 36–37
split view, 34–35
tab bar, 35–36
with table cells/detail disclosure buttons, 50
with table views, 43
Navigation bar, 31–32, 42, 279
Navigation controllers
consistent design for, 274
creating motion sketches, 115
overview of, 31–34
tab bar navigation vs., 36
Negative feedback, 147
Negotiation bugs, 122
Network activity indicator, 49
Nextstep operating system, Mail on Leopard, 131–132
Night theme, iBooks, 290
No-hand holding, for iPhone/iPad, 150
Noise layer, for matte surface, 105
Norman, Donald
on affordances, 170
on behavioral level of cognition, 167
on reflective level of cognition, 195–196
on visceral level of cognition, 145
Notations
adding to screenshots in App Store, 204
denotation vs. connotation, 167–168
using Remarks app for, 19
Notes app
replicating office supplies, 298
streamlining input, 261
using architecture sketches for, 20
Notifications
betrayal of user trust, 216–218
respecting user time/attention, 215–216
subtitle cells of Settings app screen, 45
Novel interactions, 280–283
Number pads, 163–164
Numbers app, 230–231, 268
Numerical settings, with stepper, 51

O
OmniFocus app, 186
OmniGraffle app, 19, 119
On-screen controls, sketching interfaces, 22–24
One-handed holding, iPhone layout for, 149–150
One not many, scaling features, 227, 230
One option, guiding user with, 264–266
Online resources
accessibility, 215
quiet vs. forthcoming presentations, 223–224
registering this book for reader services, xxxiii
web site for this book, xxv
Opacity, 103–104
Optical measurements, wireframes, 58–61
Orientation
sketching app with platform in mind, 21
wireframing iPhone/iPad, 77–78
worst-case height-compression scenario, 78–79
Ornamentation, 298–299
Outlines
antirequirements, 9–10
architecture, 13
avoid exposing underlying mechanism, 261
defining platform, 10–11
as to-do list, 14
exercises, 14
exploring design ideas with, 15
features and, 11–12
listing ramifications, 11
mental sweep before beginning, 6–7
more inputs to, 7–8
nonlinear but orderly process of, 3–4
overview of, 3
problem reduction, 12–13
requirements, 8–9
software design with, 4–6
starting new platform with, 130

Padding, as layout principle, 70–71
Page indicator, 49
Page metadata contrast, iBooks, 99
Page sheet style, modal view, 37
Pages app
borrowing materials from real world, 297
complexity on Mac vs. iPad, 231–232
presets, 269
templates, 268
versatile design of, 223, 230–231
Paintcode tool, mockups, 86
Paper app, 18–19, 281
Paper prototypes, 112–114
Partial curl transition style, iPhone, 38
Pathways, workflow sketch, 26–29
Pattern recognition, versatile apps, 235
Patterns, of good backgrounds, 93
Penultimate app, writing/sketching, 19
Perceived brightness (values), 88–89, 95–97
Permanence, 291, 297
Photos, 81, 249
Photoshop
converting image resources to Retina, 107–108
creating custom button, 100–106
mockup assembly in, 85, 106–107
Picker control, 49, 52–53
Pinch/unpinch
getting out of sync with fingers, 155
hysteresis and, 157
pitfalls of thresholds, 158
sandwich problem in Maps app and, 153–154
zoom in/out with, 152–153
Pixelmator tool, mockups, 86
Pixels
adding bevel, 104
and grids, 62
measuring, 58–61
and points, 57–58
Placeholder text, 51, 178
Plain apps. See Aesthetics (rich and plain)
Plain table view, 43–44
Plain text files, for software design, 5
Platform definition outline, 10–11
Platforms
creating new sketches based on precedents, 22
going cross-platform. See Cross-platform
keeping in mind while sketching app, 21
Podcast screens, navigation controllers for, 33
Points
tap targets and, 158–161
using scale for margins, 70
using scale for rhythm, 68–69
wireframing iOS displays in, 57–58
worst-case height-compression scenario, 79

Powers of 10, instantaneous feedback, 149
Precedents, 21–22, 275–276
Preemptive demo videos, 112, 117–118
Preferences, Mail, 132–135
Premium app pricing, 205–206
Presentation
functional complexity of. See Versatile apps
functional simplicity of. See Focused apps
simplicity vs. complexity of. See
Forthcoming interface design;
Quiet interface design
Presets, 228, 230, 268–269
Previews of content, navigation controllers, 33–34
Pricing, app, 205–206
Priorities, bug reporting, 122–123
Problem reduction outlines, 12–13
Problem-solving documentation, 207–208
Productivity apps, 41
Programmer’s colors, and saturation, 288–289
Progress indicators. See also Spinning indicator
quietness of spinnies vs., 251
for response of more than three seconds, 148
threshold for, 148
Progress view, as standard control, 49–50
Progressive disclosure experience, in iOS, 240–241
Project management software, outlining using, 5
Promoting features, 243–245
Proof-of-concept software, 112, 121–123
Prototypes
  exercises, 126
  interactive, 118–120
  kinds of, 112
  motion sketches as, 115–117
  overview of, 111
  paper, 112–114
  preemptive demo videos, 117–118
  proof-of-concept software, 121–123
  sketches vs., 19
  summary review, 126
  testing, 111–112, 123–126
  Wizard of Oz, 114–115
Proximity, layout principle of, 66
Pull-to-refresh
cargo cult design example, 277
as successful novel interaction, 281
threshold example, 157–158
Q
Quiet interface design
  adjacent in space, 238–239
  disappearing interfaces, 248–249
  example of, 251–252
  exercises, 253–254
  of focused and versatile apps, 223–224
  forthcoming design vs., 237
  grouping/arranging in, 242–243
  hiding vs. disabling controls, 248
  overview of, 237–238
  paying attention to context, 246–248
  progressive disclosure, 240–241
  promoting/demoting features, 243–244
  splitting difference of features, 246
  stacking in time, 239–240
  summary review, 253
  taps, 250
  text and visual weight, 250–251
R
Ramifications outline, 11
Read-only, scaling back features, 228, 230
Real-world goals, focused apps, 225
Real-world objects. See Skeuomorphic design
Real-world textures, 92–93
Realism vs. digitality
  metaphor, 297–298
ornamentation, 298–299
overview of, 296–297
simulation, 299–301
taking it easy, 301
texture and tactility, 297
Realistic gestures, 154–155
Reassurance, of elements adjacent in space, 238
Records, user feedback, 8
Redo feature, 188
Redundant messages, 176
Reflective level of cognition
  judging app quality, 124
  overview of, 195–196
Rejected (dead-end) ideas, 6, 17
Release notes, 209–210
Reliable gestures, 151–153
Remarks app, writing/sketching tool, 19
Rendering, as styling attribute, 83
Requirements outline
  creating, 8–9
  creating interface sketch from, 23
  starting new platform using, 129
Resizable images, mockups, 107
Resourcefulness, 181–182
Resources
  focused vs. versatile apps, 225
  helpful, xxvii–xxviii
  versatile app requirements, 233
Respect, establishing user, 215–219
Retina resolutions
  converting image resources to, 107–108
  Helvetica Neue typeface on, 73
  points in, 57
Rewriting messages, 175
RGB colorspace, 86–88
Rhythm, as layout principle, 68–69
Rich apps. See Aesthetics (rich and plain)
Right detail (value 1 cells) style, 45, 51
Rotate, performing gesture, 153
Rounded rectangle button, as standard control, 50
Rubber ducking, 17–18
Ruler objects, measuring pixels, 61
S
Safari, tap targets in, 160
Safety mechanism, custom controls, 187
Saturation
  HSB color model and, 87–88
  using, 288–289
  visual weight and, 91
Saving work, 262
Scale
  basic, 68–69
  modular, 69
Scale Styles setting, 107
Scaling back features in focused apps, 227–228, 230
Scope bar, as standard control, 50
Scope, choosing app, 224–225
Screens
  elements adjacent in space on single, 238–239
  elements as building blocks of. See Elements
manual undo and, 189
mockup assembly and, 106–107
navigating. See Navigation
for paper prototypes, 113–114
tab bar dominance on, 36
thinking in terms of, 55–57
for Wizard of Oz prototypes, 114–115
workflow sketches of paths between, 26–29
Screenshot Journal app, 60
Screenshots, 81–82, 204
Scrolling, 74–75, 79
Search bar, as standard control, 50
Section headers, plain table view, 44
Security, respecting user data, 216
Segmented controls, 50
Segmented-controls-as-tabs, 36–37
Selection, as visible status, 178–179
Self-guided tour, of your app, 240–241
Semiotic engineering, 169
  The Semiotic Engineering of Human-Computer Interaction
  (de Souza), 169
Sensible defaults, 265–269
Sepia theme, iBooks, 289–290
Service, customer, 211
Settings app
  gear imagery for, 171
grouped table view in, 44, 66
  subtitle cell style for Notifications screen of, 45
  value 1 cells for, 45
Settings-like split view navigation, 33–34
Shading, 58–59, 62–63
Shake to Undo gesture, 188
Shape layer, creating custom buttons, 101–102
Shine effect, app icons, 202
Shortcuts, 27–29, 154
Shorthand, using precedents, 276
Signatures, Mac Mail, 132
Signup experience, reducing friction in, 260–261
Silence, in failed feedback, 147
Similarity, layout principle of, 64
Simulation, of real-world objects, 299–301
Single-taps, 148
Size, visual weight and, 64–65
Sketching
  creating paper prototypes, 114
  creating versatile app, 233–235
  creating Wizard of Oz prototypes, 114–115
  exercises, 29
  exploring design ideas with, 15
  interactions, 24–26
  interfaces, 22–24
  playing devil’s advocate using, 22
  rubber ducking and, 17–18
  situations for, 20–21
  sketchiness of, 19–20
  summary review, 29
  thinking by, 15–16
  through conversation, 16–18
  tools for, 18–19
  using precedents, 21–22
  wireframes vs., 55–56
  workflows, 26–29
Sketching User Experiences (Buxton), xxvii, 15
Skeuomorphic design
  metaphors, 297–298
  ornamentation, 298–299
  overview of, 301
  simulation, 299–301
  taking it easy, 301
  texture and tactility, 297
Skeuomorphism, 301
Skinner, B.F., 183
Skinning standard controls, harmless distinctiveness, 279
Slicy app, 106–107, 202–203
Slide to unlock, adding friction with, 259
Slider, as standard control, 51
SnackLog sample app
  Five Whys and, 197–198
  as focused app, 228–230
  introduction to, 8–9
  making forthcoming, 252–253
  making graceful, 163–164
  making gracious, 191–193
  making quiet, 251–252
Specialized design
  conscientious divergence in, 279–280
  consistency vs., 271–272
  difficulty of novelty, 282–283
  exercises, 284
  getting the most of HIG, 272–273
  harmless distinctiveness in, 279
  how it all works out, 271–272
  one novel interaction per app, 280–281
  overview of, 278
Spinning indicator
  progress indicators vs. quietness of, 251
  pull-to-refresh in Mail using, 158
  for response of more than three seconds, 148
  threshold for, 148
Split view
  as content view, 43
  current context modal view in, 38
  presenting navigation with, 34–35
Stacked in time, 237, 239–240
Standalone apps, 129–130

Standard controls
  - custom controls based on, 52
  - customizing appearance with tints, 279
  - types of, 48–51

Standard resolution, 57, 100

Status
  - contextual, 179–180
  - invisible, 180–182
  - visible, 178–179

Status bar
  - network activity indicator in, 49
  - showing content/controls, 41
  - worst-case height-compression scenario, 79

Steering wheel zone, iPad layout, 149–150

Stencil tools, wireframes, 62

Stepper control, numerical settings, 51

Steps, adding friction with more, 259

Stocks app, 20

Stretchable images, mockups, 107

Strings, localizing app, 212

Stroke, mockups, 103–104

Styling
  - backgrounds, 92–93
  - color, 86–88
  - for communication, 84
  - with consistency. See Consistent design
  - with contrast, 89, 95–97
  - as design discipline, 82–83
  - for good contrast and visual weight, 89–92
  - image content, 95
  - in layers, 108
  - rendering and, 83
  - specialized. See Specialized design
  - tastefulness and, 84–85
  - transparency, 93–94
  - value and, 88–89
  - wireframes, 62–63

Subtitle cells, as content view, 45

Support, designing for user, 211

Surface, adding matte to, 105

Suspension of disbelief
  - breaking, 146
  - instantaneous feedback preserving, 148–149
  - iOS devices preserving, 145–146
  - moment of uncertainty, 146–147

Sustainable app pricing, 205

Swipe gesture, 152

Swipe-to-delete convention, 152

Switches
  - consistent design for, 274
  - manually undoing interactions, 189
  - as standard control, 51
  - toggling setting on/off, 50

Tab bar
  - showing content/controls, 42–43
  - top-level navigation with, 35–36
  - workflow sketches of, 26

Table cells
  - consistent design for, 274
  - contrast in, 98
  - generous tap targets for, 159–161
  - Mail for iPhone using, 136
  - with text input, 51

Table view
  - choosing options in, 50
  - contrast in, 98
  - information displayed in, 43
  - Mail for iPhone using, 135, 136
  - navigation controller options, 33
  - picker vs., 49
  - styling for contrast, 91

Tactility, borrowing materials from real world, 297

Taps
  - ease of using, 250
  - forgiving accidental/exploratory, 259
  - hysteresis and, 156
  - instantaneous feedback of, 148–149, 154–155
  - as most reliable gesture, 152
  - navigation controllers using, 33
  - sketching interactions for, 26
  - targets for, 158–161

Target audience
  - accessing for outlining, 7–8
  - usability testing with, 124–126

Tasks, outlining, 5

Text
  - aligning in layout, 67
  - combining with cues/imagery, 176
  - demanding attention/requiring reading, 174
  - depth styling for legibility, 291
  - giving loudness and clarity to, 250–251
  - in interface interaction design, 172–174
  - label, 45, 48–49
  - margin and padding guidelines, 70–74
  - measuring optically, 59
  - principles of typography, 73–74
  - understated layout for, 72
Text fields, 51
Text view, 46
Texture, 105, 297
Themes, iBooks, 289–290
Thinking, Fast and Slow (Kahneman), xxviii, 238
Thinking with Type (Lupton), 74
Thresholds, in graceful interface, 157–158
Thumb field, iPhone layout, 149–150
Time, user
  elements adjacent in space saving, 238
  precedents saving, 276
  respecting, 215–218
Timing, iOS animations, 116
Tints, customizing app, 279
To-do applications, tasks in, 5
To-do lists, outlines as, 14
Toolbar buttons
  bordered/unbordered, 42
  center alignment of borderless, 67
  iBooks transparent, 93–94
  iPhone Mail commands, 136
  margin and padding, 70
Toolbars
  customizing with tints, 279
  safe hues for, 286–287
  showing content/controls, 42
Tools
  graphics, 85–86
  prototyping, 118–119
  sketching, 18–19
Touch and hold, 152
Toyota, Five Whys process, 197–198
Track 8 music app, 294
Traditional outlines, 5
Transitions, iPhone, 38
Transparency, mockups and, 93–94
Trends, design, 272
Triangulation, versatile app design, 233–235
Trust, respecting user, 215–219
Tutorials, introducing interface via, 208–209
Tweetbot, Use Last Photo Taken button, 265–266
Tweetie, 157–158, 277, 281
Two-handed holding, iPhone/iPad, 150
Typography
  Apple points vs. points in, 57–58
  page sheet modal view and, 37
  principles of, 73–74
Undo feature
  arrow buttons, 171
  disabling vs. hiding in iWork, 248
  overview of, 187–188
  prominence of, 243
Unicode character set, 73, 211–213
Unintended friction, 259–264
Unitaskers, 225
Unity, layout, 63–64
Updating sketches, as you go, 18
Usability testing, 123–126
Use cases
  defined, 121–122
  starting new platform, 130
  versatile app design, 233–236
User experience design
  accessibility, 213–215
  conveying capability, 198–206
  documentation, 206–210
  ethos, 215
  exercises, 219
  following precedents to save effort, 276
  localization, 211–213
  overview of, 195–196
  respect, 215–219
  serving the soul, 197–198
  summary review, 219
  support, 211
User feedback
  keeping records of, 8
  often-requested features vs. antirequirements, 10
  in usability testing, 126, 208
User Interface Design Labs, 273
Users
  accessing for outlines, 7–8
  betrayal of trust, 216–218
  guessing intentions using hysteresis, 157
  sketching interactions, 26
  use of term in this book, 218–219
V
Value 1 cells (right detail) style, 45, 51
Value 2 cells (left detail) style, 45, 51
Value bar, 163–164
Values (perceived brightness)
  contrast and, 89
  overview of, 88–89
Vectors, defining shape layer, 102
Versatile apps
  bringing own goals to, 231
  creating, 233
  designing, 230–231
  exercises, 236
  finding boundaries, 235–236
  as forthcoming or quiet, 223–224
iOS love of versatility, 231–232
pattern recognition for, 235
resources required for, 233
summary review, 236
using triangulation, 233–235
Version control, for design resources, 8
Versions tool, by Black Pixel, 8
Videos, preemptive demo, 117–118
Visceral level of cognition
crafting graceful interface. See Interface, crafting graceful
defined, 145
judging app quality at, 124
Visible status, interface interaction design, 178–179
Visual cues, 259
Visual rhythm, layout, 68–69
Visual weight
adding friction by increasing, 259–260
adjusting for balance, 71
adjusting for contrast, 89–92
giving loudness and clarity, 250–251
as layout principle, 64–65
using hue for, 287
VoiceOver, for accessibility, 214
Volume slider, 51

W
W3C (World Wide Web Consortium), 89
Wait indicator threshold, 148
Warning cues, 259
Weather app
adjusting for time of day, 183
focused design of, 223
status images of, 171
Web, going cross-platform with, 128–129
Web service, sketching interactions for, 26
Web view, 46
“What’s New,” App Store, 209–210
Wheels of time, 48–49
White theme, iBooks, 289–290
Whiteboards, 5, 16–18
Wide prototypes, 119–120
Widget-type apps, starting out, 20
Widths, 37, 76
Windows, cross-platform with, 128
Wireframes
content and controls layout, 75
controls in content areas, 75
dimensionality, 76–77
exercises, 80
in graceful interface layout, 149–151
information density, 75
layout principles. See Layout
optical measurements, 58–61
orientation on iPad, 78
orientation on iPhone, 77–78
sketches vs., 55–56
summary review, 79
for tab bars, 42–43
thinking in layers, 75–76
thinking in points, 57–58
thinking in screens, 55–57
tools, 61–63
typography, 72–74
in Wizard of Oz prototypes, 114–115
worst-case height-compression scenario, 78–79
Wizard of Oz prototypes, 112, 114–115
Workflow sketches, 26–29
World Wide Web Consortium (W3C), 89
Worldwide Developers Conference, Apple, 273
Wrench icon, 48
Writing
about software, 4–6
interface interaction with good, 174–176

X
Xcode, 61, 70
xScope app, 61, 101, 108, 115

Z
Z dimension, 74
Zero options, 262–263
Zoom in
measuring pixels with, 60
pinch/unpinch for. See Pinch/unpinch
two-fingered double tap for, 154