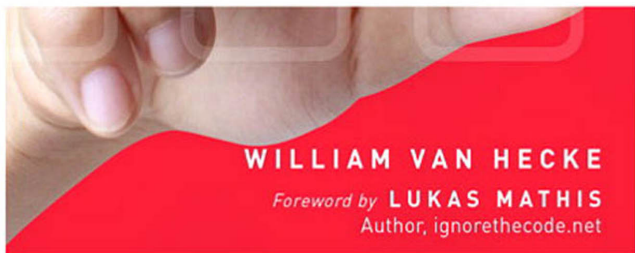




LEARNING iOS DESIGN

A Hands-On Guide for Programmers and Designers



WILLIAM VAN HECKE

Foreword by **LUKAS MATHIS**
Author, ignorethecode.net

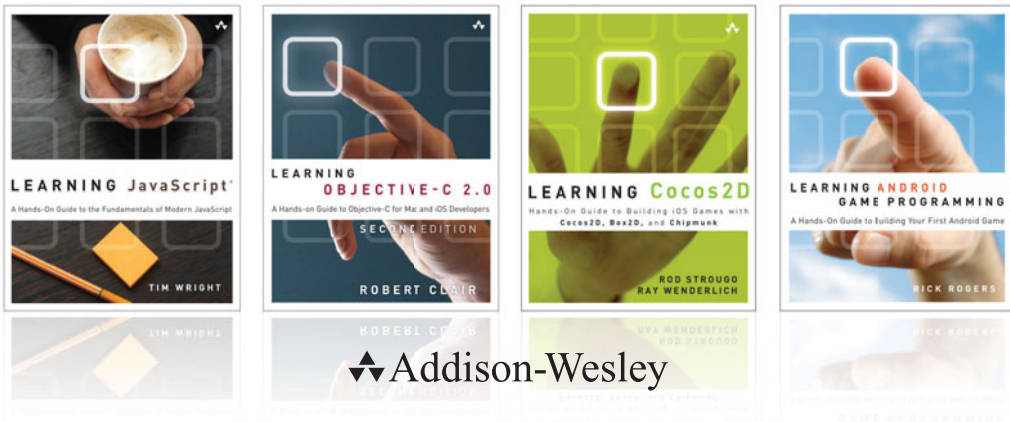
FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Learning iOS Design

Addison-Wesley Learning Series



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.

 Addison-Wesley

 **informIT**
The trusted technology learning source

 **Safari**
Books Online

Learning iOS Design

A Hands-On Guide for
Programmers and Designers

William Van Hecke

◆◆Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

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

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

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Van Hecke, William.

Learning iOS design : a hands-on guide for programmers and designers /
William Van Hecke.

pages cm

Includes index.

ISBN-13: 978-0-321-88749-8 (pbk. : alk. paper)

ISBN-10: 0-321-88749-2 (pbk. : alk. paper)

1. iOS (Electronic resource) 2. Application software—Development. 3. iPad
(Computer)—Programming. 4. iPhone (Smartphone)—Programming. I. Title.
QA76.774.I67V36 2013
004.167—dc23

2013010043

Copyright © 2013 William Van Hecke

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-88749-8

ISBN-10: 0-321-88749-2

Text printed in the United States on recycled paper at RR Donnelley in
Crawfordsville, Indiana.

First printing, June 2013

Editor-in-Chief

Mark L. Taub

**Senior Acquisitions
Editor**

Trina MacDonald

**Development
Editor**

Sheri Cain

Managing Editor

John Fuller

**Full-Service
Production
Manager**

Julie B. Nahil

Project Editor

Anna Popick

Copy Editor

Betsy Hardinger

Indexer

Jack Lewis

Proofreader

Anna Popick

**Technical
Reviewers**

Jon Bell

Jim Correia

Lukas Mathis

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Composer

Rob Mauhar



To Buzz and CeeCee; Tonichi and Risako



This page intentionally left blank

Contents at a Glance

Foreword	xix
Preface	xxi
Acknowledgments	xxix
About the Author	xxx

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
-------	------------

This page intentionally left blank

Contents

Foreword	xix
Preface	xxi
Acknowledgments	xxix
About the Author	xxxii

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

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

Start from Scratch	130
Back to the Outlines	130
Case Study: Apple Mail	131
Mac OS X Leopard	131
iPhone	134
iPad	138
Back to the Mac	140
Summary	141
Exercises	142

II Principles **143**

8 The Graceful Interface **145**

Suspension of Disbelief	145
The Moment of Uncertainty	146
Instantaneous Feedback	147
Gracefulness through Layout	149
Six Reliable Gestures	151
The Sandwich Problem	153
Exotic Gestures as Shortcuts	154
Realistic Gestures	154
Hysteresis	155
Thresholds	157
Generous Taps	158
Meaningful Animation	161
Making SnackLog Graceful	163
Summary	164
Exercises	164

9 The Gracious Interface **167**

Denotation and Connotation	167
Cues	168
Imagery	171
Text	172
Writing: The Secret Design Discipline	174
Redundant Messages	176
Communication Breakdown	176
Guidance at the Point of Need	177

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
Localization	211
Accessibility	213
VoiceOver	214
AssistiveTouch	214
Ethos	215

Respect	215	
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

The Specialized Design	278
Harmless Distinctiveness	279
Conscientious Divergence	279
One Free Novel Interaction	280
Novelty Is Hard	282
Summary	283
Exercises	284

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
--------------	------------

Supplement: The Learning iOS Design Companion	319
------------------------------------------------------	------------

This page intentionally left blank

Foreword

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

Preface

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>. 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

About the Author

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.

This page intentionally left blank

This page intentionally left blank

The Wireframes

In the grand hierarchy of pictures of imaginary software, wireframes exist somewhere between sketches and mockups. Their purpose is to nail down the details that sketches leave out: what exactly exists on each screen and how it all fits together—the *geography* of an app. A sketch is casual and usually disposable; you might attach a sketch to a design document as a reminder of the conversation that led to a decision, but the sketch probably doesn't represent a definitive understanding of how the software should end up. Wireframes, on the other hand, document some details of how the app should be built. (See Figure 4.1 for an illustration of the difference.)

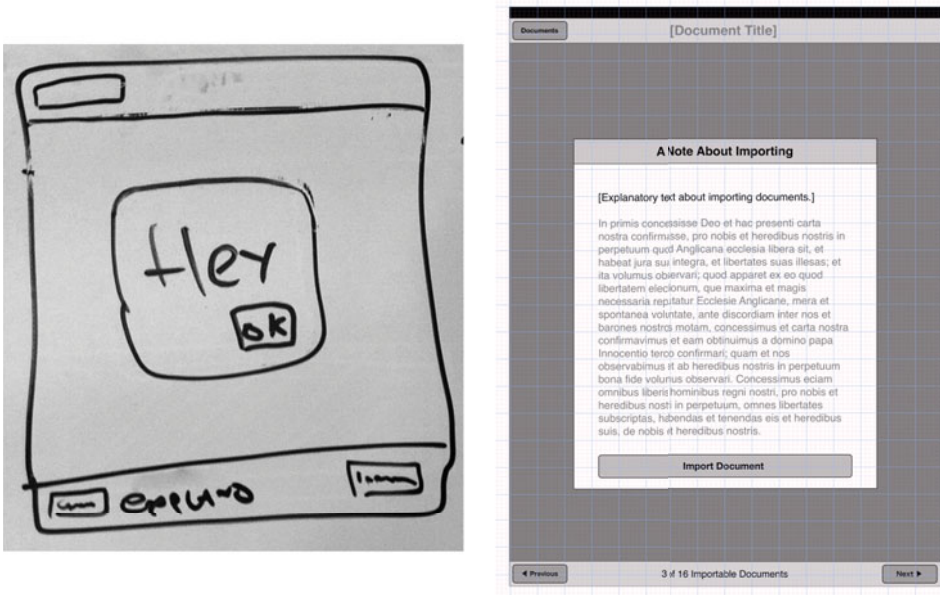


Figure 4.1 A sketch (left) becomes a wireframe, gaining detail about the screen elements and their layout.

The distinction is not always black and white. Some wireframes are kind of sketchy, and some sketches are kind of wireframey. But generally speaking, wireframes have the following unique characteristics as compared with other types of software design images.

- A wireframe includes all the elements that should go on the final screen; a sketch probably omits some.
- A wireframe delineates the type, position, and size of elements to a reasonably final degree; a sketch can drastically distort them.
- A wireframe doesn't specify an exact, pixel-perfect rendering of what elements should look like; that's what mockups are for. Instead, it uses a minimal rendering—usually plain, flat shapes, lines, and text.
- A wireframe may give some indication of the relative visual weight of elements on the screen, usually by differences in shading.

A wireframe may or may not be interactive; if it is, it's also a prototype.

Thinking in Screens

iOS apps are made of **screens**. Sure, they also include animations, audio cues, notifications, and other bits that can't be represented by static screens. And some apps do a lot of work behind the scenes that never becomes visible. But the basic constituent unit of any app, from a user's perspective, is the screen. The login screen, the home screen, the item detail screen, the archive screen, the user info screen, the settings screen, the share screen—these are the places users visit as they journey around your app. Each screen might have a number of states it can be in as well as a number of elements it might display depending on the situation.

Many **interface screens** are bigger than the physical size of the **hardware display** they appear on and can be scrolled or zoomed around. Don't confuse these different types of screens. (In this book, "screen" always means a place in the app, and not the glass on the front of the device.) On iPad, popovers give you a roughly iPhone-sized mini-display within a display, which can move between its own set of screens. All these screens and the states they move through are the skeleton of an app; you need to get them right to get the app right. (By now, it's normal if "screen" no longer looks like a word to you.)

Thanks to outlining, sketching, meeting, arguing, and deep thinking, you've got a good idea of all the screens that need to go into your app, and it's time to plot out precisely what each one should contain and how it should be organized. You need wireframing. Making yourself draw out every button, switch, field, and label will almost certainly raise some questions.

- How are we going to fit all six of these features onto one toolbar?
- Should this feature really be exposed at this point in the app?

- Do we want to present this option up front, or tuck it away somewhere else?
- Is this the right place to diverge from Apple’s guidelines and precedents?
- Should these controls really be separate, or can we combine them?
- Oh dear, what have we gotten ourselves into? Is this even the app we should be making?

And so on. That’s fine! That is part of the process. You might need to do more sketching or hold discussions to figure out how you want to answer those questions. Part III of this book is all about finding your own answers to those questions that don’t have definitive solutions, so you may want to peruse it when you run into one. Get excited: this step—figuring out how you want to answer the questions that come up during wireframing—is when you develop the unique personality of your app.

Thinking in Points

We’re used to dealing with pixels when measuring interface elements. And for a couple of years, only two sets of pixel dimensions mattered to iOS designers: the iPhone (at 320×480) and the iPad (at 768×1,024). As of the introduction of the iPhone 4 and the third-generation iPad, you now need to think about both standard and Retina resolutions, for both device families. Thankfully, the Retina resolutions are evenly divisible by the standard ones: one standard pixel split in half, in both dimensions, produces four Retina pixels. So you can simply multiply pixel distances by 2 and carry on.

But for the most part, you shouldn’t be thinking in pixels until you get to the mockup stage and start creating production graphics. For wireframes, you should be thinking in points. The **point** comes from typography and publishing, where it has held a variety of values over the centuries but which finally settled at $\frac{1}{72}$ inches, the “PostScript point.” The Apple point, though, is a new unit that indicates a resolution-independent distance on the display. On a standard-resolution display, 1 point equals 1 pixel; on a Retina display, 1 point equals 2 pixels. (See Figure 4.2.)

The important thing to remember is that you shouldn’t take the presence of more pixels to mean you can increase information density. Retina displays are not for cramming numerous comically tiny elements onto a single screen. Rather, they’re for increasing the *fidelity* of the same old identically sized elements. So Retina displays shouldn’t even affect your wireframing process. Carry on designing for a screen that’s 320×480 or 768×1,024, and enjoy the extra sharpness and fidelity you’ll get if it appears on a Retina display.

Optical Measurements

For wireframing, you’ll need to use an app that can lay out objects on a canvas, and you’ll need to be able to measure their positions and their relationships to each other

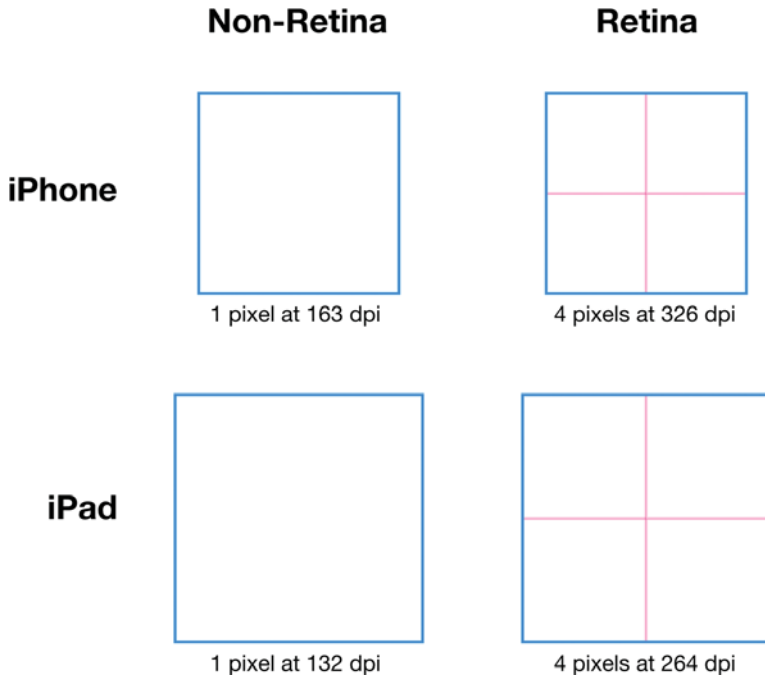


Figure 4.2 The meaning of 1 point on the four iOS displays. The size of a point can vary and can consist of 1 or 4 pixels.

very precisely. But the measurements reported by Xcode and by various graphics apps for a given object are sometimes at odds with each other, and are often at odds with what your eyes actually see. This is because the actual *visual* edges of a given object don't necessarily coincide with the bounds of that object as understood by the app displaying it. Objects have whitespace inside, or shadow effects dangling off of them, or text ascenders and descenders, or other accoutrements that make it hard for the software to know where the theoretical thing, as seen by a human, begins and ends.

Bounds and Optical Edges Are Sometimes Equal

Sometimes, of course, the visual edges of an object *do* correspond with its bounds as understood by the software drawing it. That's the case for a rectangle without any of those accoutrements like extra whitespace or shadows. For those objects, you can rely on the measurements reported by the software.

To plan clean, professional layouts, you need to think optically and measure sizes and distances optically. This often means ignoring what OmniGraffle, Photoshop, Fireworks, and Xcode say about where things are and how big they are. Instead, use the following techniques—and your eyes.

First, you need to know where to measure from. This is getting a bit into mockup territory, because you're going to see examples that include final effects. You aren't supposed to be worrying concretely about production graphics at this point, but you need to know about these techniques while you're wireframing.

Measuring Text Optically

Measuring text horizontally is easy: just look at the left- or rightmost point that's mostly opaque. If there's a faint column of antialiased pixels or a few pixels protruding from a round character shape (like an O), you can ignore them and move on to the heavily filled pixels.

Effectively every typeface—and certainly Helvetica Neue, the standard font on iOS—has its weight vertically delineated by its baseline and its cap height.

- The **baseline** is the horizontal line that the bottoms of most letters sit upon. (The curves of rounded letters go below it just a bit, and the descenders of certain letters protrude down beyond it.)
- The **cap height** is the line that the capital letters reach up to. (Some lowercase letters also reach it.)

These two lines delineate the vertical bounds of text for the purposes of optically aligning and measuring distances between elements. (See Figure 4.3.) Yes, this means that the descent of letters like y and p, and the ascent of letters like Å and Ž, may protrude outside the visual weight of the text and into the margins around it. Generous margins should more than accommodate that. Tons of apps get this wrong, either by including the descent or excluding the cap height in the weight of a text element. You can do better.

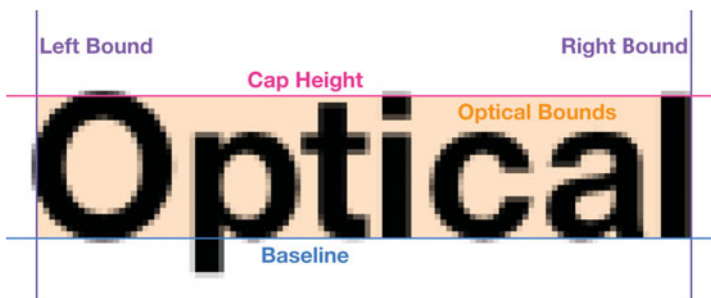


Figure 4.3 Measuring text optically. The shaded area represents the bounds of the text for the purposes of positioning and balance. Rounded shapes and faint antialiasing can protrude slightly.

Measuring Images and Controls Optically

Any control or image should have some sort of contrasting border or edge that defines its optical boundary. (If the edges seem to blur into the surface that the element is resting on, then you have a contrast problem. See Chapter 5 for more about contrast.) The boundary where the dark object transitions to the light background, or vice versa, is the optical boundary of the object. (See Figure 4.4.)

Where to Measure From

Some designers measure from the inside of the boundary; others measure from the outside. Is the edge of a table cell defined by the last white pixel inside, or the last gray pixel of the border? It doesn't matter which you pick, as long as you're consistent.

Techniques for Measuring

When you need an exact pixel measurement, here are some ways you can get it.

- **Zoom in really close**—On your Mac, go to the Accessibility pane of System Preferences and turn on the systemwide zoom. Turn off image smoothing, because you want to see those pixels nice and chunky. Now you can use the keyboard commands or the modified scroll gesture to zoom way the heck in on your designs and count the pixels yourself. Lots of times this is the quickest way to check on very small distances, but it gets tedious and error-prone at around 16 pixels. On iOS, the accessibility zoom always has blurry smoothing applied—you can't turn it off. You can work around that limitation with the app Screenshot Journal, which is intended for designers and developers, so it has a nice, chunky zoom.

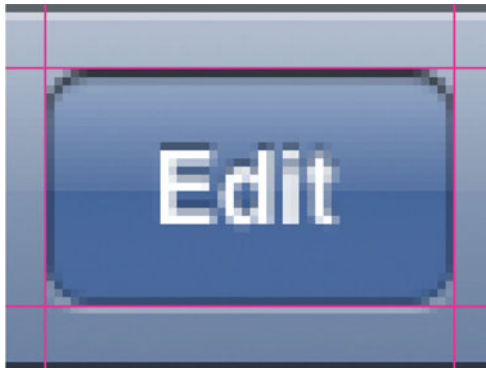


Figure 4.4 Measuring controls optically. In this example, the outside of the stroke is considered to define the bounds of the button. Note that the underhighlight for the etched effect is not part of the bounds.

- **Guides and grids**—Any decent diagramming or graphics app should offer manual guides and customizable grids that you can use to draw laser-straight lines across your canvas. These are an excellent way to make sure objects are aligned and spaced evenly.
- **Standalone measurement tools**—Expect to run into lots of situations when you need to get the exact dimensions of an element in the iOS simulator or your wireframes, align two distant objects, or perform some other feat of precision. A standalone screen measurement tool (independent of your graphics app) is invaluable. You aren't likely to do better than the Iconfactory's xScope, a suite of tools for measuring, scrutinizing, and previewing pixels; it works like the guides and grids mentioned earlier, but for your whole screen rather than within a certain app.
- **Ruler objects**—Another option is to prepare precisely sized ruler objects inside your graphics app of choice, and keep them on a layer that you can unhide when you need to check a size or distance. These objects should be ones that you know are the same size optically and are shown in the app's logic—such as a plain, filled rectangle with no outer stroke—so that you can rely on the sizes reported by the app.

Tools for Wireframing

When it comes to sketching tools, you want anything that helps you get ideas out as quickly and roughly as possible. But for wireframes, precision is paramount. Theoretically you could do wireframes on graph paper, if you have an unusually steady hand, a mathematical mind, and a superb eraser. But you're almost certainly best served by a desktop software tool. You should choose something that lets you nudge objects pixel by pixel, measure exact distances, and quickly make adjustments to get your positioning and dimensions just right. This book doesn't rely on a specific piece of software, but among your options are OmniGraffle, Fireworks, Illustrator, Photoshop, Axure, and Balsamiq.

Designing in Xcode

Some designers use Xcode itself for interface design, which has the benefit that you can use the resulting product in the actual app. But it works great only as long as you stick strictly to stock, standard controls and elements. You'll miss out on custom elements, annotations, layers, and the rest of the graphical and logical benefits offered by a proper graphic design app. Once you have the hang of designing in design software, it'll save you much more time and effort than designing in Xcode.

Here are some crucial features to look for in your wireframing tools.

- **Layers**—One of the best skills you can have for building good wireframes is layer management. Be fastidious about keeping related elements together on layers, and name your layers descriptively; an orderly layer structure will serve you well. You can use layer visibility to compare alternative approaches to the same problem. You can also describe several states of a screen on just one canvas; popovers, sheets, alerts, and so on can each live on their own layer for easy showing and hiding. Some apps (like Photoshop) let you save layer comps that remember the visibility of each layer, or (like OmniGraffle) share layers across canvases.
- **Grids**—Get used to using the grid to position objects exactly where you want them. For initial layout, a good grid setting is 44 pixels with 11 subdivisions, resulting in 4-pixel squares. The major grid lines help you see the minimum 44-point-square tap target, and the minor grid lines give you some fine movement, 4 pixels at a time, without getting too fiddly. You may occasionally need to bump the subdivisions to 44 or 22 for a moment to move elements just a pixel or two. Even if you want maximum precision, leave the subdivisions at 1 pixel to prevent yourself from accidentally placing an object between pixel boundaries. A design document showing objects off the pixel grid looks cheap and mushy.
- **Styles**—Over time, you're bound to develop your own tastes and practices for wireframe creation. But it's good to keep in mind that you have options. The term "wireframe" of course comes from the classic outlines-only style of drawing interface elements, which looks precise like a blueprint and cares very little about the eventual appearance of each element. If you want to be a bit more detailed, you can use shading—monochrome or a very limited color palette is best—to indicate the relative visual weight or the semantic grouping of elements. Going in the opposite direction, you could even make your wireframes look intentionally slightly sketchy, just to communicate clearly that they're not finalized. (See Figure 4.5 for some wireframing styles.)
- **Dimension lines**—If you're passing your designs off to be implemented, and especially if you aren't going to have a mockup step between the wireframes and the implementation, it's helpful to include dimension lines between elements to make sizes and distances explicit. Some tools (like OmniGraffle) can even automatically calculate and display the length of a line. Put dimension lines on their own layer so that you can show them when they're needed and hide them otherwise.
- **Templates and stencils**—Whatever your graphics app of choice, you can find prepopulated templates and stencils full of wireframing components for common iOS screens and controls. Assuming that the resources you use are accurate, you can save a lot of time and effort in creating elements that are the correct dimensions. Or you can always create your own internal suite of stencils that fit the way you design so that you don't have to re-create every object every time.



Figure 4.5 Several wireframing styles. Left to right: slightly sketchy, with the precise dimensions not yet nailed down; bare wires, to avoid suggesting any visual treatments; shaded, for suggesting relative visual weight; color-coded, for easy recognition of groups of elements.

Principles of Layout

Before we get into the specifics of laying out iOS screens, it's helpful to learn (or review) some basic principles. As you lay out each screen, you should be thinking about what the placement and relationships of the elements communicate to users. There's no single canonical process for turning a set of requirements into a “correct” layout. But the principles discussed here will guide you as you experiment with layout possibilities and try to find a balance between the needs and constraints of the design.

Much of this section is based on the rudiments of Gestalt psychology—a theoretical framework that has a lot to say about how human beings perceive things. It's been highly influential in visual design. It tends to show up in art school curricula, and a century after its founding it is even useful for laying out iOS app screens. You could spend a lot of time studying Gestalt theory. If this sort of thing is interesting to you, check the reading list in the Inspiration Is Everywhere section of the Preface.

Unity Is the Goal

The following principles are aimed toward the creation of **unity** in your layout. In a unified design, every component seems to be just where it needs to be. Nothing extra-neous is present. Nothing feels uncomfortable or haphazard. Each part makes sense on its own and contributes to a whole that makes sense, too. And if all is well, users don't notice this unity. Unless they're familiar with design and they're looking at the screen with a critical eye, most people don't realize what is pleasant about a layout. They just happily proceed to use it. If pressed, they might be able to say that it seems “clean” or “professional.” But if the app falls short of achieving unity, people *will*

notice. Customers can spot a disjointed layout from a mile away, and they recognize it as a warning sign that the developer hasn't put enough care and time into polishing the product.

How well an app is laid out is related to, but is not the same as, how well rendered it is graphically. That's another test for your layout to pass, and you'll get to that in Chapter 5, *The Mockups*. For now, focus on constructing good bones so that you can layer on the appealing styling later.

Read on for the principles. When you carefully consider all of them and get them working harmoniously, you'll achieve unity. Most of the principles probably seem obvious on their own, but it's surprising how easy it is to neglect them if they're not near the front of your consciousness. And it's the combination of them that creates a magical, ineffable sense of good design. If a design seems "off" somehow, it's probably time to run through this list and see which principle is being violated.

Visual Weight

Each element in a layout possesses some amount of **visual weight** relative to the others and to the layout as a whole. This weight affects how readily the element is noticed by the eye, how important it seems, and how it affects the balance of the layout. Visual weight is determined mainly by multiplying the following two attributes.

- **Size**—The bigger an element is, the more visual weight it carries.
- **Background contrast**—The more an element stands out from the background, the more visual weight it carries.

It's easy for a new designer to think that size is the only attribute that contributes to visual weight, neglecting the role of contrast. But the relationship between size and contrast means you can have elements that are different sizes but feel equally weighty, or elements that are the same size but seem to carry different weights. (See Figure 4.6 for an illustration.)

The final contrast depends largely on the visual treatment you end up giving the element in the mockup phase. But you should consider now the amount of contrast you intend to give the element so that you can weigh that against its size. For now, while wireframing, you can indicate contrast roughly by filling objects with three shades of gray to represent light, medium, and dark elements and areas.

For instance, suppose your app has a crucial button that you want to make particularly large for easy tapping. Giving it the same style as all your other buttons, while also making it much bigger, would result in an uncomfortably heavy button. It would far outweigh everything else on the screen. But if during the wireframing phase you indicate that it should be given a low-contrast treatment, you can then plan on styling it accordingly during mockups.

Or you may have a button that you need to keep small, for space reasons or even to minimize accidental taps. To keep it from getting lost among the other elements on the screen, you can increase its contrast with the background. A fine example of this



Figure 4.6 Objects of different sizes can have equivalent visual weights, as shown here with abstract wireframe objects (top row) and then again with visual treatments (bottom row).

is the purchase button in the App Store. It's actually smaller than the recommended minimum tap target size, because it's important to avoid accidental taps. But its heavy, high-contrast treatment makes its importance apparent.

Similarity and Distinction

Similar objects seem related, and dissimilar objects seem different. It seems banal to even write down. But there it is, because it's crucial to remember it as you design. When you're weighing your options for which element to use, where to place it, or how to size and orient it, compare it to existing elements on the same screen. Its similarity or dissimilarity to the other elements can tip the scale. Consider what their level of similarity says about the on-screen elements.

If you offer five similarly styled toolbar buttons, for example, a user expects them to be roughly equivalent. It's safe to assume that they are all actions that can be taken on whatever is currently visible in the content area, and that they're about equally likely to be used. Identically styled items listed in a table view are seen as peers—none particularly more important than another.

Any dissimilarities you introduce should have meaning. People tend to look for a reason when something is not like the things around it. The Done or Save button that concludes a modal view tends to be blue. Some apps give a distinctive size or style to a single toolbar button or tab to indicate that it is somehow more important than the rest. The Delete Contact button in the Contacts app lives among flat, white table views, but it's presented in a unique shiny red style to call out its distinct (and destructive) purpose.

Proximity and Distance

Objects that are close together seem related, and objects that are far apart seem different. This fact works along with similarity and distinction to determine how the eye groups things. The closer two things are in space, the more strongly their similarity links them, and vice versa.

When elements have similar purposes or meanings, put them close together. In many cases, such as in grouped table views, you can actually stick them together, sharing a border, with no space or separation between them. Or if you have two elements you need to separate conceptually, try putting some space or some sort of visual separator between them.

The grouped table views in the various screens of the Settings app are the prototypical example of using grouping and separation to associate and disassociate elements. Notice that they bear group labels only when necessary; often, the cells in each group do a fine job of explaining what the group is for. Really, the whole Settings app is a gold mine of admirably clever problem solving through layout.

Alignment

Most objects on a screen should align with something else on the screen. That is, if you draw an imaginary straight line along the edge or center of an object and continue it across the screen, the line should coincide with the edges or centers of other objects rather than miss them. (See Figure 4.7.) Look at any clean design, and you're bound to find that objects are rarely just a bit out of alignment. Either they're perfectly aligned, or they're far enough out of alignment that it's obvious the misalignment is intentional.

Alignment and Interface Layers

There's one major exception to the rule that objects should align, and it has to do with interface layers (which are not necessarily the same as the layers feature in graphics software). The Thinking in Layers section later in this chapter explains how to use interface layers to conceptually separate chunks of your interface. To reinforce the distinction, it's good for objects on different interface layers to be slightly out of alignment with each other. Notice that in Apple apps, the side margins in a navigation bar don't align with the side margins in the content area. The narrower margins in the navigation bar layer subtly suggest that it's a different interface layer that "contains" the content area layer.

Here are some tips for keeping elements aligned.

- **Edge alignment**—Aim to edge-align as many as possible of rectilinear (box-like) objects in your design: buttons, tables, square icons, and so on. For similar

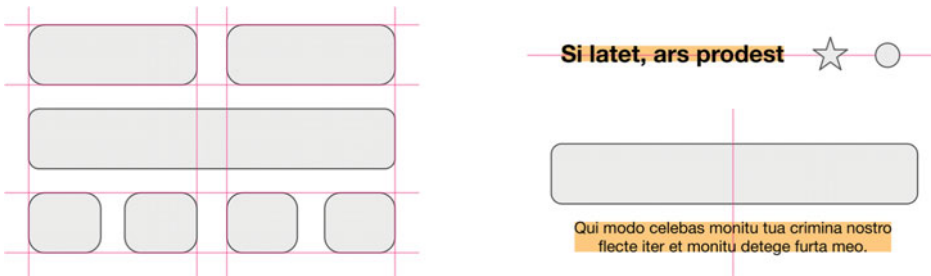


Figure 4.7 Edge alignment and center alignment. Pink lines show alignment; orange areas show the optical bounds of text elements.

objects in close proximity, edge alignment reinforces their relationship. If objects are farther apart, or less similar, then their edge alignment simply looks nice; it doesn't suggest a strong relationship.

- Center alignment**—For rectilinear objects, this design is weaker and more prone to messiness than edge alignment. But it's excellent for irregular shapes, like blobs of text or bare graphics. For example, borderless toolbar buttons are usually aligned to their vertical center rather than edge-aligned, because each one has its own unique shape. Similarly, text that doesn't live in a box is often center-aligned. The title in a navigation bar is usually centered on the screen horizontally and centered in the bar vertically.
- Aligning text**—Generally, text wants to align to other text more than it wants to align to boxes. You might at first think that a table view label should be left-aligned with the table border. But it looks much more at peace when it's left-aligned with the text *inside* the table. To vertically align different-sized pieces of text, the traditional approach is to ground them by making them share the same baseline; otherwise, both pieces of text will seem to be floating uncomfortably in space. If the two pieces of text are of very different sizes, though, then center-aligning to the vertical weight works better.
- The guide test**—This is the quickest, most straightforward way to check the alignment of a layout and then fix it. Put the layout on the canvas of a graphics app that supports manual guides (straight lines that span the whole canvas). For every visual edge you see, add a guide. The edges of aligned objects can share a guide, but misaligned objects get their own guides. When you're finished, the guides express the visual logic of your layout, whether it's neat and sensible or a chaotic mess. From there you can adjust elements and remove guides until you have the most elegant layout that still supports the needs of the design. (See Figure 4.8 for a before-and-after example.)

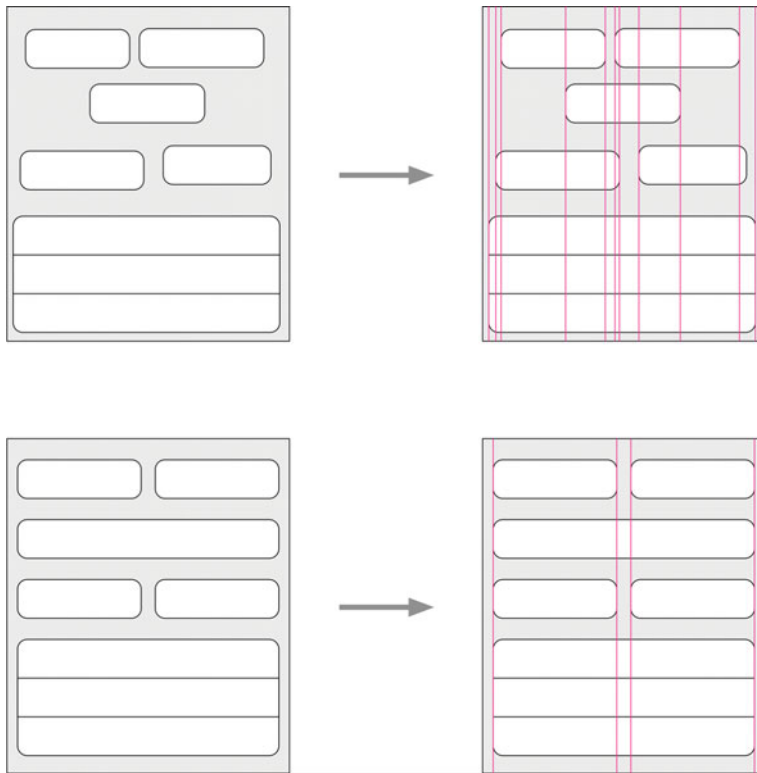


Figure 4.8 Applying the guide test to a particularly heinous layout (top), and then to a repaired version of the same layout (bottom). Only the guides for horizontal alignment are shown.

Rhythm

A good sense of **visual rhythm** is usually a cumulative effect of your spacing, alignment, and balance. But you can take an extra measure to ensure that the sizes and distances you choose are harmonious: use a scale. A **scale** is a preset list of measurements for you to rely on rather than choose an arbitrary value every time you need to position or size something. When the same harmonious numbers keep appearing throughout a design, it lends a sense of unity and order. (See Figure 4.9 for scales in use.)

Here are two scales that work well for iOS designs; you can pick one of these or invent your own.

- **Basic scale: 10, 20, 44**—For almost any design, you can get away with using this simplified scale. You use 10 points for a “normal” distance—for separating controls from each other, from labels, and from the edges of the screen or the view that contains them. Use 20 points when you need extra separation, as for groups

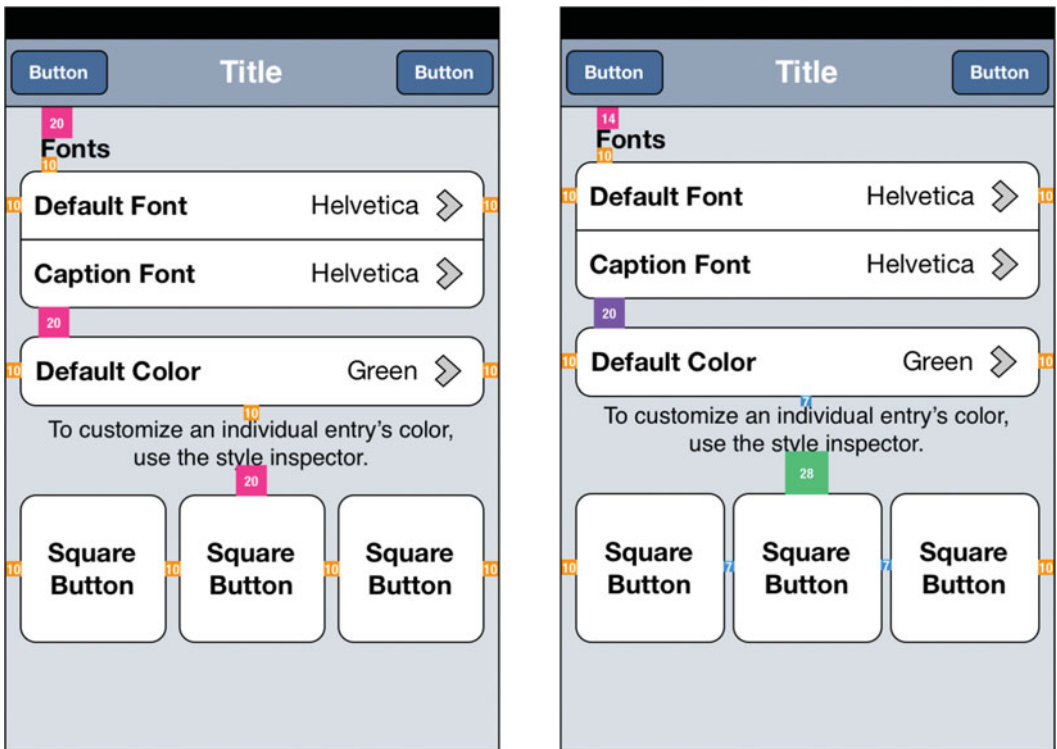


Figure 4.9 Basic-scale (left) and modular-scale (right) variants of a single design

of unrelated controls. Many standard iOS controls (such as table cells and toolbars) are 44 points; if you need much bigger spaces, using multiples of 44 when possible will coordinate well with them. You may need to use some other sizes on an ad hoc basis—for instance, a 52-point-tall table cell for holding a lot of text. That's fine! But don't also create 50- and 54-point-tall cells that will clash with it.

- Modular scale: 7, 10, 14, 20, 28, 40, 56, 80...**—A modular scale is based on iterated multiplication by some number. In this case it's $\sqrt{2}$, which results in a doubling every two iterations. You can get very involved with choosing a ratio and devising a scheme for how you use each step in the scale; ventures like this are part of the endlessly rewarding pursuit of design. If you enjoy this sort of thing and want to learn more, pick up *The Elements of Typographic Style* by Robert Bringhurst. If not, there's no shame in using the basic scale.

The guide test described earlier is also good at uncovering awkward rhythms. If the distances between the guides seem to vary willy-nilly instead of falling into a sensible pattern, you need to make some adjustments.

Margin and Padding

Layouts look more modern, more pleasant, and more approachable when they employ healthy amounts of empty space. **Margin** is the amount of space afforded around an object. For boxlike objects with borders and content, such as a table cell or a button with a text label, **padding** is the amount of space afforded from the border to the content.

On iOS, users expect a certain amount of breathing room for elements, and when it's not there the app feels cramped, cheap, and uncouth. Scrolling is almost free, cognitively and ergonomically speaking, so there is little downside to giving content more space.

Don't Trust Xcode's Guides

Xcode's automatic positioning guides do try to help with layout, but you can't rely solely on them. They're more than happy to blindly suggest positions and spacing that make no sense, given what the layout actually looks like. Remember, optical measurements are what counts, and not the bounds of objects in Interface Builder. Over time, you may develop a reflex of letting Xcode place an object wherever it wants and then adding or subtracting a certain amount of space that you know to be correct for your design.

For margins, you can rely on a scale, as described earlier; put 10 points around everything, and you're in pretty good shape. To make sure you provide adequate padding, though, it's up to you to be vigilant. There isn't a strict algorithm for figuring out how much padding to afford a given bit of content. But a few guidelines can take you a long way. Most of these are based on the cap height of the text involved; that's the height of an ordinary capital letter like "E."

- For a single line of text in a box, give about 100% of the cap height as vertical padding. (See Figure 4.10 for illustrations of text and image padding.)
- For multiple lines of text in a single box, give 50% to 100% of the cap height of the biggest text as vertical padding.
- Generally, the horizontal padding should be between 50% and 100% of the cap height of the biggest text. Once you have your vertical padding as recommended earlier, you can apply a similar amount as horizontal padding.
- Objects that are edge-aligned with each other, especially horizontally, should also edge-align their text by having equal horizontal padding. (Even though the cells in Figure 4.10 have different top and bottom padding, they have the same side padding.)
- Images inside of boxes (such as toolbar buttons) can generally stand having a bit less padding than text. Aim for between 25% and 50% of the height or width of the image.

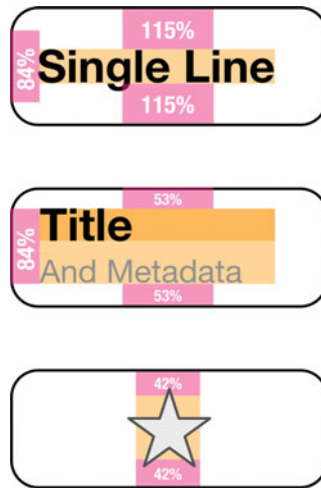


Figure 4.10 Padding examples for text, in percentages of the cap height) and an image (in percentage of the image height). Orange areas indicate optical bounds; note that the star’s optical bounds don’t reach all the way to the points, because the points contribute little to the visual weight of the overall shape.

Balance

Even if all the individual details in your layout are appropriately sized, aligned, and spaced, you still need to look at how the whole picture fits together. The main thing to keep in mind when thinking at this scale is **balance**: how the visual weight of the elements is distributed across the entirety of the screen.

A cheap way to get balance is by making your layout symmetrical. This approach is appropriate in a lot of cases, but it isn’t the only way. One thing to watch out for is the tendency of the elements that balance each other to seem equivalent or in opposition to one another.

Another way to get balance is by carefully adjusting the visual weight of the elements. Adjust the size or the background contrast (described earlier) of elements to add or remove weight from an area of the screen. You may even subtly shift elements out of strict alignment or consume some of the space reserved for margins in order to achieve better balance. This approach takes a lot of experience and wisdom.

Thankfully, iOS apps can generally bear quite a bit of unbalance before they look bad from it. Vertical imbalance is usually fine, but try to avoid leaving one side of the screen empty while populating the other side.

Understatement

Use the minimum elements necessary to satisfy the needs of the design. In other words, never include anything that you don’t need. This is the single most important

principle, and it probably seems the most obvious. But it's more subtle than it seems, and it's easy to violate without realizing it.

To separate controls, you usually need only take one step up on your distance scale. There's usually no need for a separator line. When you have two pieces of text separated by whitespace and using different styles, you don't also need to use punctuation or borders to set them apart. If you have buttons organized in a grid, you don't need to draw a border completely around each item; let them share borders like a table, and use the edge of the screen or view as a border, too. Or eliminate the borders if the buttons still end up looking tappable. (See Figure 4.11 for an application of understatement.)

Typography

Normally a design book would need a huge chapter dedicated to typography: how to use type effectively and how to avoid the common pitfalls that make type look amateurish. Luckily, iOS has done a lot of the work for you, making decent typography

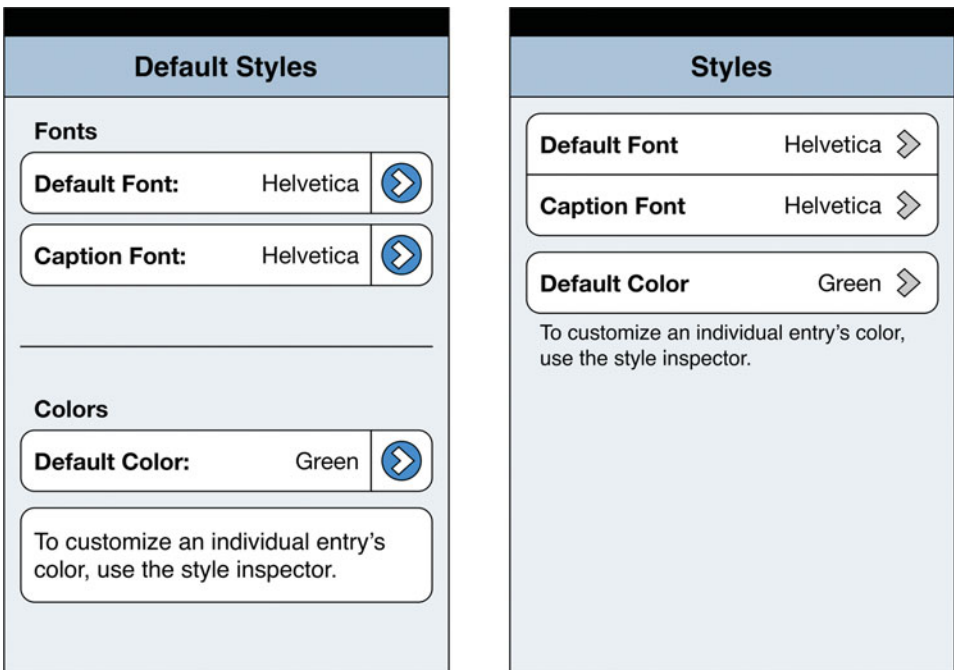


Figure 4.11 A needlessly fussy layout (left) and an understated version of the same screen (right).

easy and poor typography difficult. For most projects, you need only follow a few principles, listed here.

- Many of the principles described in this chapter apply to text as much as to other elements. Text should be balanced, aligned, understated, adequately padded, and so on. A modular scale, described earlier in this chapter, is wonderful for choosing text sizes. See the principles in this chapter that call out how they apply specifically to text.
- Almost all the time, the system font (Helvetica Neue on Retina displays, and Helvetica elsewhere) is all you need. It's a fantastic all-purpose font given the size and resolution of iOS displays, and it's so neutral that anything else tends to feel odd in an iOS interface. One exception arises when you want to put in a bit of branding by, say, using your logotype as a label on your home screen; that's fine, but use the system font everywhere else. The other exception is for long-form reading, spanning many screens' worth of text; in these cases it can be worth switching to a font specially designed for heavy reading; Georgia and Verdana are solid choices. (To learn about hunting down great fonts for inclusion in your app rather than using the iOS defaults, look up Marco Arment's "Introducing Instapaper 4.1 for iPhone, iPad" post on marco.org.)
- Use real characters, not ascii approximations. Habits from the early days of personal computing and the limitations of typical keyboards often keep people restricted to the meager ascii character set. Today's systems have a wealth of expressive characters available, thanks to the Unicode character set, but many people don't even know that there's a difference between, say, a proper apostrophe (') and a clumsy vertical tick ('). Use real quotation marks (" ") instead of neutral ones ("), use en- and em-dashes (—) instead of double-hyphens (--), and use a real ellipsis (...) rather than three periods. All these characters can be typed with key combinations or copied from a lookup app like UnicodeChecker.
- There's no need to separate sentences with two spaces. It can be a hard habit to kick if you were raised to type this way, but it's just not necessary in the age of modern digital typography. One space is all you need; the carefully tuned fonts included with iOS know how to handle it.
- Generally, columns of the correct width (about wide enough to type the alphabet once or twice) are easier to read than too-wide ones (more than two alphabets wide) or very narrow ones (less than one alphabet wide). If you do have wide columns, you can make them easier to read by putting a bit more space between lines. Then it's easier for the eye to stay on track as it moves from the end of one line to the beginning of the next.
- On an iOS screen, it's rare that you'll have enough words per line to benefit from horizontal justification; instead it's likely to lead to unsightly and random-looking gaps between words. Left-alignment, with a ragged right edge, works well for paragraphs of content. Center-alignment is common for short, stand-alone explanatory blurbs of text that live outside of elements.

For advice beyond this, get hold of Robert Bringhurst's *The Elements of Typographic Style*, or the more adventurous *Thinking with Type* by Ellen Lupton, which are a thrill to read and will put you well on your way to becoming an expert.

Layout: A Place for Everything...

Now that you're familiar with the mechanics of placing elements together on a screen, it's time to think about *where* to put them. This section deals with the semantic (meaning-oriented) implications of how you arrange elements and how iOS helps you convey meaning through layout.

Content and Controls

Recognize the difference, and the relationship, between content and controls. **Content** is the actual stuff that users come to your app to see and manipulate: documents, images, web sites, media, text, and so on. **Controls**, then, are the administrative necessities (such as buttons and switches) that the user uses to tell the app what to do. The content is on the screen because the user wants to see it and work with it. The controls are on the screen because the app needs to get input from the user about how to behave.

This relationship should make it clear that content takes precedent whenever possible. The more you can pull your controls out of the way of your content, the better. That means either putting the controls on a visually distinct layer or presenting them in a temporary form such as a popover or a contextual menu.

Thinking in Layers

Most iOS apps involve some sort of layering, with the visual suggestion that the parts of the screen are actually different contiguous surfaces, each with its own independent position, and often overlapping in the imaginary dimension perpendicular to the screen (usually called the **z dimension**). This book calls these **interface layers** to differentiate them from the layers feature of graphics software. The most basic example of this concept is the way the content of an iPhone app scrolls around while the navigation bar and toolbar stay put. The content appears to be sliding around on its own layer behind the bars. Similarly, the sidebar of an iPad app is clearly its own layer by the way it scrolls independently of the main content area (or slides in from the side, in portrait orientation).

But independent scrolling isn't the only way you can distinguish interface layers. A conspicuous change in visual treatment can also signal a transition from one layer to another. The status bar, for instance, is usually styled differently from the app interface so that it seems more like part of the hardware device than part of the app. The same goes for the keyboard; users understand it as being different from the content area where they're using it to enter text.

The value of interface layers is both practical and conceptual. Practically, of course, it's good to be able to keep some things on the screen (like toolbar buttons) while

other things (like content) scroll around. Conceptually, layers can signal the purpose of elements and content. Whatever occupies the center of the screen gets the user's attention, whereas the controls in toolbars, sidebars, and navigation bars around the periphery tend to be more utilitarian, only there to serve a purpose ancillary to the main content. Layers help you avoid polluting the content area with controls and help users' brains understand what's safe to ignore when they're just interested in looking at the content.

Occasionally apps add even more interface layers. An example is the slide-out ruler in the app Pages, which provides further, optional controls related to styling. As you lay out a screen, think about how to make interface elements feel as if they exist in discrete, modular chunks, grouped and arranged meaningfully around the main content.

Controls in Content Areas

Every rule has exceptions. Even though you just learned that controls should be arranged in chunks around the content, sometimes it makes sense to put controls inline with the content. If a control is closely related to the content, and especially if space is tight in the designated control areas, it might make sense to put the control in with the content.

The classic example of this is a “new item” placeholder presented as the last item in a table. If users tap it, a new item appears right in the tapped cell. Note that this approach doesn't guarantee that the new item button will always be visible on the screen; if you have a long list, it might be scrolled off the bottom. (Putting it at the top would be awkward because new items are usually added to the end of a list, not the beginning.)

The main challenge in presenting controls in the content area is making them look sufficiently distinctive to identify them as controls, without overpowering the precedence of the content. One minor styling difference, such as making text grayish-blue instead of black, is usually enough.

Information Density

Scrolling on iOS is cheap. The huge touch target, the simple gesture, the breezy inertial effect, and the viscerally satisfying animation make it one of the easiest things to do on an iOS device. So putting content “below the fold”—that is, beyond the first screenful of information (analogous to the fold in a paper newspaper)—is not nearly as strong a deemphasis as it was on the desktop in days of yore.

So calm any tendency you may have to cram as much data as possible onto a single screen. In some situations, an info-dense, dashboard-style, at-a-glance view is useful, but it's uncharacteristic of iOS apps. Even on a small screen, feel free to spread out your information as described earlier in the principles of layout. See more about information density in Chapter 12.

Dimensionality

Given the controls you have to lay out and the screen or view size you have to work with, it may seem reasonable to use a two-dimensional arrangement: a grid, a multi-column table, and so on. Especially on iPad, where you have a lot more screen space to work with than on iPhone, you might be tempted to get sideways. Be careful! iOS is generally a one-dimensional platform, because it's harder for users to decipher 2-D layouts.

- Most fundamentally, 1-D layouts are far easier for the human eye to scan and interpret. You can simply draw your eye down the list of available items until it arrives at the correct one. A 2-D layout instead encourages the eye to bounce around the screen haphazardly, unsure where to look next.
- Two-dimensional layouts suggest meaning that might not be there. Each column seems to have a purpose, so the eye looks for a way to group its items. The same goes for rows; anything adjacent appears to be adjacent for a reason, which your brain tries to find.
- Although traditional tables of data are common on the desktop, such as in Finder windows or the classic Mail message list, they are rare on iOS. Instead of laying out cells of data side by side, iOS more commonly provides a couple of important pieces of information, using a supplemental text style, arranged in the same cell as the main title. This makes for a sort of 1.5-dimensional layout, where the items are arranged in a linear list but each item contains information spread out horizontally and vertically. (The 1.5-D message list in Mail on iOS has even made its way to the Mail on OS X.)

Keep Widths Sensible

Especially if you stay one-dimensional, you might wonder how to keep your elements from being comically wide as they try to fill the width of an iPad display. In such cases, you should usually not have the entire width of the display to fill; elements usually appear in a popover, a sheet, a sidebar, or the content area next to a sidebar. But if you find yourself with a content area covering the entire screen anyway, use generous amounts of margin (balanced on both sides) to push the controls inward. That's better than having ridiculously sized elements that fill the space just because it's there. (The page sheet style of modal view, described in Chapter 3, does a good job of keeping things from getting too wide.)

Sometimes, though, going two-dimensional is a fine answer. After all, even the iOS home screen is a grid.

- Two-dimensional layouts work pretty well with instantly recognizable entities. A grid of big, simple, colorful icons should work great. But a grid of tiny text-labeled buttons is a hassle for the mind to process.

- The fewer items you have to lay out, the more likely they are to work in 2-D. A 2×3 grid of 120-point square icons can be understandable and comfortable to interact with. A 5×5 grid of 48-point square icons, though, would be a headache to parse.
- If you can establish a meaningful reason for the columns that helps the user identify the options, 2-D can be nice. Suppose you have three text-oriented options and three media-oriented options in a blogging app; separating them into two columns makes good sense.
- Two dimensions is more often appropriate for content than it is for controls. Don't be afraid to use a 2-D layout to present information if it makes the information clearer. Sometimes it makes perfect sense to lay out information spatially. And if a meaningful 2-D presentation already exists for the content at hand, as in a calendar, even better.

Orientation on iPhone

iPhone apps can be locked to a certain orientation (portrait or landscape), or they can adapt to either orientation. There are four basic options for dealing with device rotation.

- Locking your app to portrait orientation is understandable, but not ideal, for a typical app. If you have to pick an orientation to lock to, portrait is the one. That's the orientation of the iPhone home screen, and it's by far the most commonly used orientation in apps. iOS tends to stack UI elements vertically: navigation bars, toolbars, keyboards, and so on. So some apps are simply too hard to cram into the shorter vertical space of a landscape iPhone.
- Being locked to landscape orientation is odd. A few apps can get away with it because their interface is dependent on a unique and immutable composition of elements.
- Adapting one interface to either orientation is ideal. If you can do it without undertaking a drastic redesign of your entire interface, you should. Users appreciate being able to rotate the device to switch between seeing more items at once and seeing more detail about each item. Lots of people prefer the big keys on the landscape keyboard for typing-intensive tasks. You can even get away with a few rarely used screens being portrait-only, if you must.
- Occasionally, it makes sense to offer a different interface depending on the orientation. If you have two drastically different but comparably valuable ways of looking at the same information, an orientation change may be just the way to switch between them. That's especially true if the normal view wouldn't have benefited much from landscape anyway. But be careful with this approach. Plenty of users simply never think to rotate their phone in order to see something different. Some even keep their orientation locked all the time to avoid accidental rotation, because they never intentionally switch to landscape.

Consider the orientation handling of some of Apple's built-in and App Store apps.

- Clock is locked to portrait orientation, because switching to landscape wouldn't gain you much. The information in each item is narrow enough to be completely visible without needing more horizontal space; switching to landscape would merely reduce the number of items visible at once and would make the already sparse items look even more empty.
- GarageBand is locked to landscape orientation, an unorthodox choice. Of course, GarageBand has an unorthodox approach to its interface overall. It feels much more like a virtual hardware device, like an actual piece of recording equipment rather than a magical piece of software that can reconfigure its layout. It is already pushing hard on the boundaries of what a phone is expected to do; redesigning its painstakingly detailed and surprisingly complex interface to be orientation independent would have been prohibitively difficult, with little to be gained. If your app is as grand and complex as GarageBand, and as astoundingly useful, feel free to lock it to landscape.
- Mail adapts to both orientations, which is an easy choice because of its structure. For the most part, it's a flexible content area bounded by a navigation bar at the top and a toolbar at the bottom. When you rotate the device, the content resizes appropriately, and the bars get a bit taller or shorter. It's a small investment for a big payoff, considering how much reading and typing people do in the app.
- Calendar offers a different presentation in landscape mode: a sort of expanded version of its hour-by-hour day planner that spans multiple days and can be panned two-dimensionally. In this app, rotating the device is more like toggling a view setting than it is like simply adjusting the size of the "window."

Orientation on iPad

The question of how to deal with orientation is quite a bit simpler on iPad. The device is still easy to rotate, but it's not as easy and casual as rotating an iPhone. People tend to pick an orientation and stick with it for a long time as they work with various apps. The screen is spacious enough that as long as you design carefully, either orientation offers enough space to get things done. So it is much harder to get away with not offering the same interface in both orientations. For almost all apps, users expect the app to accommodate them and not to have to rotate the iPad to accommodate the app.

Being locked to either orientation requires a compelling reason. GarageBand and Keynote are both locked to landscape, because their interfaces are so specific to those screen dimensions that it would not make much sense to reinterpret them in portrait.

The Worst-Case Height-Compression Scenario

On both devices and in both orientations, it might seem as if you have plenty of vertical space to work with. But there's a horde of interface elements just waiting to stack

up and consume it. Whatever elements you're intentionally including in your design, make sure you consider the other elements that might invite themselves.

- Unless you're hiding the status bar because you have an especially immersive app, it'll always be there, eating up 20 points.
- On an iPhone in portrait mode, if a phone call, voice recording, or tethering session is in progress, then the status bar grows to be 40 points tall.
- On any screen that allows text entry, you must of course account for the keyboard (162 points on iPhone landscape, 216 points on iPhone portrait, 352 points on iPad landscape, and 264 points on iPad portrait). Screens that don't normally scroll may need to scroll when the keyboard is visible.
- Some languages, such as Japanese, also need a completion bar to support text input, so the space needed for the keyboard is actually taller than you think (36 points on iPhone, 54 points on iPad).
- Even if you're designing with the taller iPhone 5 in mind, with 568 points of height to work with, lots of people have older iPhones with only 480 points of height. Every screen you make should be great on the shorter display and even better on the new taller display. This may mean that less scrolling is necessary, or it could mean that screens designed to fill the display afford more space to each element. Wireframe for both heights.

For each screen you wireframe, plan how it will respond to having *all* the possible interface elements visible at once. If it can take all those elements and still have a reasonable amount of room for the content, then you know you're in good shape.

Summary

Wireframing is about figuring out which elements to use and where. It requires thinking in screens: how elements fit onto screens, how one screen flows into the next, and what variants and states of each screen exist. It also requires thinking in points: the dimensions of elements and their placement relative to the screen boundaries and the other elements. There are lots of tools for wireframing. Make sure you choose tools that help you draw precisely and measure accurately.

To get good at wireframing, you should be familiar with the fundamentals of layout, especially the goal of unity and the importance of understatement. These principles will serve you well elsewhere, too: on the desktop, on the web, in graphic design, and beyond. When it comes to iOS specifically, keep in mind the standards of layout that users expect from a well-made app.

The next stage of our journey is an exciting one. It's almost time to turn our modest, utilitarian skeleton of a design into a lovely, charming, production-grade mockup.

Exercises

The principles you've learned in this chapter should serve you well for building responsible, sturdy screens. Work through these exercises to get your own app into shape.

1. Based on reviewing the sketches you made of each screen and referring to your architecture outline, build a wireframe of each screen that includes every element just where it needs to be. Consider all the elements built in to iOS and the possibilities of custom controls. Consider what the positioning, grouping, and characteristics of the elements say about their meanings.
2. Critically scrutinize your wireframes, looking for any elements that are misaligned, unbalanced, or cramped. Use the guide test to bring all the elements into harmony as best you can.
3. Wireframe key screens in both orientations and with all the possible transitory elements visible. Make sure these changes don't cause the layout to break down.

Index

- 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
 - contrast 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
 - encouraging 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, xxi–xxii, xxvi
 - 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

- 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

- 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
- Convertbot 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

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 (Brigham), 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
 - experience 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
- Gestures
- adding friction with more-involved, 259
 - exotic, 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
- Grids
- 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

- I**
- 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 (spiny). *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
 - iOS
 - custom controls, 52–53
 - elements. *See* Elements
 - exercises, 53
 - navigation scheme. *See* Navigation
 - overview of, 31
 - summary review, 53
 - iPad
 - action sheets, 47, 190–191
 - app icon variants, 201
 - designing Mail for, 138–139
 - drawbacks of sketching with, 19
 - form factor, and sketching for, 21
 - going cross-platform, 127
 - handling orientation, 78
 - holding techniques/layout, 149–151
 - modal views, 37–38
 - popovers, 39, 56
 - sketching interface for, 21–24
 - sketching workflow for, 26
 - sleek/lean apps of, 11–12
 - tap target sizes, 161
 - Undo button for apps, 171
 - worst-case height-compression scenario, 78–79
 - iPhone
 - action sheets, 47, 190–191
 - going cross-platform with, 127
 - handling orientation on, 78
 - holding techniques/layout, 149–151
 - icon variants, 201–202
 - Mail for, 134–138
 - Mail for iPad vs., 139
 - modal 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
 - ramifications outline for, 11
 - 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
- J**
- Jobs, Steve, xxi, 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
 - toolbar configurations, 42
 - 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

M

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
- N**
- 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

P

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

Poppers

- hues for, 286

- on iPad screens, 56

- modal views vs., 38

- navigating, 36, 39

- paying attention to context with, 246–247

- styling for communication, 84

- tips for, 43

- undo feature and, 188

- workflow sketches of, 26

Portrait orientation

- on iPad, 78

- on iPhone, 77–78

- keeping in mind while sketching app, 21

- page sheet modal view in, 37

- split-view navigation and, 33–34

- worst-case height-compression scenario, 79

Posterization process, 95–98

PowerPoint, 266

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

T

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

Tastefulness, as styling attribute, 84–85

Templates, 62, 268

Terminology, designing features using, 13

Testing

- prototype animations, 117
- prototypes on device, 111–112
- usability of prototypes, 123–126
- using hysteresis to improve, 157
- VoiceOver, 214

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, 70–71
- 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, 130
 - Typography
 - Apple points vs. points in, 57–58
 - page sheet modal view and, 37
 - principles of, 73–74
- U**
- UI furniture, 239, 248–249
 - UIPreRenderedIcon shine effect, 202
 - Unbordered buttons, toolbars, 42
 - Underlying mechanisms, 261
 - Underhighlights, 105–106, 291–293
 - Understatement, 71–72, 84–85
 - 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

This page intentionally left blank