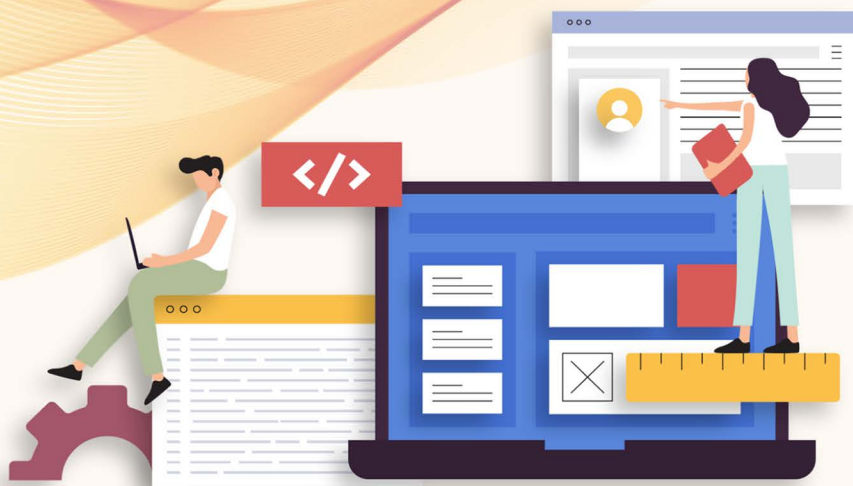


101 UX **PRINCIPLES**

«packt»

Actionable Solutions for Product Design Success
2nd Edition



WILL GRANT

Includes invitation to an online UX professionals community

101 UX Principles

Second Edition

Actionable Solutions for Product Design
Success

Will Grant



BIRMINGHAM—MUMBAI

101 UX Principles

Second Edition

Copyright © 2022 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Senior Publishing Product Manager: Manish Nainani

Contracting Acquisition Editor: Tushar Gupta

Acquisition Editor – Peer Reviews: Gaurav Gavas

Project Editor: Janice Gonsalves

Content Development Editor: Edward Doxey

Copy Editor: Safis Editing

Technical Editor: Aniket Shetty

Proofreader: Safis Editing

Indexer: Hemangini Bari

Presentation Designer: Pranit Padwal

First published: December 2018

Second edition: May 2022

Production reference: 1160522

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-80323-488-5

www.packt.com

“For Claire, Noah, and Alfie”

Contributors

About the author

Will Grant is a veteran product designer and UX professional with over 20 years' experience overseeing the design, information architecture, and usability of web and mobile products that have reached over a billion users.

After his Computer Science degree, Will trained with Jakob Nielsen and Bruce Tognazzini at the Nielsen Norman Group—the world leaders in usable design. He has since devoted his career to working on designing usable, powerful products that solve problems and generate value for organizations. With a deep understanding of mobile, enterprise, and consumer software products, Will is adept at identifying and addressing users' problems with thoughtful design.

Will is the author of Amazon bestseller *101 UX Principles* and UX contributor to *net*, *Econsultancy* and other publications. He also co-founded and designed the web's biggest data migration tool, SQLizer.io, and open-source visualization product QueryTree.

I'd like to thank Claudio Gomboli, Kate Shaw, and Ricardo Megger for their tireless and invaluable help in reviewing this book.

About the reviewers

Claudio Gomboli is the Head of User Experience at Canonical, the company behind Ubuntu Linux. Previously at Intel, he manages the UX team at Canonical, working on web applications and services for enterprise cloud solutions, IoT, and embedded devices.

Currently based in London, Claudio has lived in Italy, Japan, and Germany over the last 20 years, working in various design roles with web agencies, start-ups, design studios, and large IT corporations. During his career, Claudio has worked with exceptional people and multidisciplinary teams, delivering products from end to end.

Kate Shaw is a senior product designer, communicator, creator, problem solver, mum of three, and wannabe revolutionary.

With over 20 years' experience in digital product design and seeking the balance between commercial and customer needs, Kate has designed intuitive experiences for start-ups, FTSE 100 companies, and agencies. Her clients have included The Telegraph, the BBC, The Guardian, John Lewis, and Expedia, and she is now happily relocated to Switzerland, working for the Swiss leader in online banking.

Ricardo Megger is a Brazilian UX/UI designer. He has degrees in Design and Marketing and works at Grupo Services, where he helps create amazing experiences for users. He likes to transition between different areas of knowledge, making analogies between them; as such, he is very interested in biomimicry. In his spare time, he enjoys reading classical poetry and watching horror movies. His major professional aspiration is helping the future happen.

Join us on Discord!

Read this book alongside other users, UX professionals, and author Will Grant.

Ask questions, contribute to discussions with other readers, chat with Will via *Ask Me Anything* sessions and much more.

Scan the QR code or visit the link to join the **UX Squad** community.



<https://packt.link/101uxprinciples>

Table of Contents

Preface	xix
UX Field	1
Principle 1: Everyone Can Be Great at UX	3
Principle 2: Be Strategic About Using These Principles	7
Principle 3: Don't Be Afraid to Ship Something Simple...	11
Principle 4: ...But Complexity Can Be Good for Some Users	15
Principle 5: Use A/B Testing to Test Your Ideas	19
Principle 6: Test with Real Users	23
Principle 7: Nobody Cares About Your Brand	27
Typography	31
Principle 8: Don't Use More than Two Typefaces	33
Principle 9: Users Already Have Fonts on Their Computers, So Use Them	35
Principle 10: Use Type Size and Weight to Depict an Information Hierarchy	39
Principle 11: Use a Sensible Default Size for Body Copy	45

Controls	47
Principle 12: Use an Ellipsis to Indicate That There's a Further Step	49
Principle 13: Make Interactive Elements Obvious and Discoverable	53
Principle 14: Make Buttons a Sensible Size And Group Them Together by Function	59
Principle 15: Make the Whole Button Clickable, Not Just the Text	63
Principle 16: Don't Invent New, Arbitrary Controls	65
Principle 17: Search Should Be a Text Field with a Button Labeled "Search"	67
Principle 18: Sliders Should Be Used for Non-Quantifiable Values Only	73
Principle 19: Use Numeric Entry Fields for Precise Integers	75
Principle 20: Don't Use a Drop-Down Menu If You Only Have a Few Options	77
Principle 21: Allow Users to Undo Destructive Actions	81
Principle 22: Optimize Your Interface for Mobile	85

Content	91
Principle 23: Use “Infinite Scroll” For Feed-Style Content Only	93
Principle 24: If Your Content Has a Beginning, Middle, and End, Use Pagination	97
Principle 25: Allow Users to Accept or Reject Cookies with One Click	101
Principle 26: Help Users Understand Their Next Steps from “Empty States”	105
Principle 27: Make “Getting Started” Tips Easily Dismissable	109
Principle 28: When a User Refreshes a Feed, Move Them to the Last Unread Item	113
Navigation	115
Principle 29: Don’t Hide Items Away in a “Hamburger” Menu	117
Principle 30: Make Your Links Look like Links	121
Principle 31: Split Menu Items Down Into Subsections, so Users Don’t Have to Remember Large Lists	125
Principle 32: Categorize Settings in an Accessible Way	129
Principle 33: Repeat Menu Items in the Footer or Lower Down in the View	135

Iconography	139
Principle 34: Use Consistent Icons Across the Product	141
Principle 35: Don't Use Obsolete Icons	143
Principle 36: Don't Try to Depict a New Idea with an Existing Icon	147
Principle 37: Never Use Text on Icons	151
Principle 38: Always Give Icons a Text Label	153
Input	157
Principle 39: Use Device-Native Input Features Where Possible	159
Principle 40: Streamline Creating and Entering Passwords	165
Principle 41: Always Allow the User to Paste into Password Fields	169
Principle 42: Don't Attempt to Validate Email Addresses	173
Principle 43: Respect Users' Time and Effort in Your Forms	177
Principle 44: Pick a Sensible Size for Multiline Input Fields	181
Principle 45: Use Animation with Care in User Interfaces	185
Principle 46: Use the Same Date Picker Controls Consistently	189

Principle 47: Pre-Fill the Username in “Forgot Password” Fields	191
Principle 48: Make Your Input Systems Case-Insensitive	195
Principle 49: Chatbots Are Usually a Bad Idea	197
Forms	201
Principle 50: If Your Forms Are Good, Your Product Is Good	203
Principle 51: Validate Data Entry as Soon as Possible	207
Principle 52: If the Form Fails Validation, Show the User Which Field Needs Their Attention	211
Principle 53: Users Don’t Know (and Don’t Care) About Your Data Formats	215
Principle 54: Pick the Right Control for the Job	219
User Data	223
Principle 55: Allow Users to Enter Phone Numbers However They Wish	225
Principle 56: Use Dropdowns Sensibly for Date Entry	229
Principle 57: Capture the Bare Minimum When Requesting Payment Card Details	233
Principle 58: Make It Easy for Users to Enter Postal or ZIP Codes	239

Table of Contents

Principle 59: Don't Add Decimal Places to Currency Input	245
Principle 60: Make It Painless for the User to Add Images	249
Progress	253
Principle 61: Use a "Linear" Progress Bar If a Task Will Take a Determinate Amount of Time	255
Principle 62: Show a Numeric Progress Indicator on the Progress Bar	259
Principle 63: Show a "Spinner" If the Task Will Take an Indeterminate Amount of Time	263
Accessible Design	265
Principle 64: Contrast Ratios Are Your Friends	267
Principle 65: If You Must Use "Flat Design" Then Add Some Visual Affordances to Controls	271
Principle 66: Avoid Ambiguous Symbols	277
Principle 67: Make Links Make Sense Out of Context	279
Principle 68: Add "Skip to Content" Links Above the Header and Navigation	281
Principle 69: Never Use Color Alone to Convey Information	285
Principle 70: If You Turn off Device Zoom with a Meta Tag, You're Evil	289

Principle 71: Give Navigation Elements a Logical Tab Order	293
Principle 72: Write Clear Labels for Controls	295
Principle 73: Make Tappable Areas Finger-Sized	299
Journeys and State	303
Principle 74: Let Users Turn off Specific Notifications	305
Principle 75: Each Aspect of a User's Journey Should Have a Beginning and End	309
Principle 76: The User Should Always Know What Stage They Are at in Any Given Journey	313
Principle 77: Use Breadcrumb Navigation	317
Principle 78: Users Rarely Care About Your Company	321
Principle 79: Follow the Standard E-Commerce Pattern	325
Principle 80: Show an Indicator If the User's Work Is Unsaved	329
Principle 81: Let Users Give Feedback, but Don't Hassle Them	331
Principle 82: Don't Use a Vanity Splash Screen	335
Principle 83: Make Your Favicon Distinctive	339
Principle 84: Add a "Create From Existing" Flow	343
Principle 85: Make It Easy for Users to Pay You	345
Principle 86: Give Users the Ability to Filter Search Results	349

Table of Contents

Principle 87: Your Users Probably Don't Understand the Filesystem	353
Principle 88: Show, Don't Tell	357
Terminology	361
Principle 89: Be Consistent with Terminology	363
Principle 90: Use "Sign In" and "Sign Out", Not "Log In" and "Log Out"	365
Principle 91: Make It Clear to Users If They're Joining or Signing In	367
Principle 92: Standardize the Password Reset Experience	371
Principle 93: Write Like a Human Being	375
Principle 94: Choose Active Verbs over Passive	377
Expectations	381
Principle 95: Search Results Pages Should Show the Most Relevant Result at the Top of the Page	383
Principle 96: Pick Good Defaults	387
Principle 97: Only Use Modal Views for Blocking Actions	391
Principle 98: Give Users the Experience They Expect	395
Principle 99: Decide Whether an Interaction Should Be Obvious, Easy, or Possible	399

Principle 100: “Does It Work on Mobile?” Is Obsolete	403
UX Philosophy	407
<hr/>	
Principle 101: Don’t Join the Dark Side	409
Bonus: Strive for Simplicity	415
Other Books You May Enjoy	419
Index	425

Preface

These 101 principles are a broad set of guidelines for designing digital products. There are no doubt thousands more, but these are the core principles that will make most products more usable and effective. They'll save you time and make users happier.

This book discusses many aspects of User Experience (UX) but focuses heavily on the User Interface (UI). The fact is, the pixels on the screen are still a huge part of the customer experience with digital products.

The UI (the buttons on the screen, the controls on the form) is the only way the user has to interact with your product—making the quality of the UX (the overall experience) tightly coupled with the quality and usability of the interface. UX is a bigger domain than UI, but we can't ignore how essential the UI is to UX.

Somewhere along the journey of the web maturing, we forgot something important: UX is not art. It's the opposite of art. UX design should perform a function: serving users. It must still *look* great, but not at the expense of *actually working*. Poor design has crept in over the years and some digital products have become worse in 100 tiny ways.

So how did we get here? Branding agencies got involved. They insisted that because as a company we always refer to photos as “memories,” the photo menu should be called memories too. Nobody knows what it means or how to find their photos.

The CEO *personally* picked the shade of sea breeze that the company uses for its headings everywhere, so all the headings are pale blue. This means nobody can read them against a white background on their mobile phone screen.

The marketing department decided that a full-screen pop-up collecting users' email addresses would be good for the Quarter 4 CRM metrics. Then they said, “Oh, don't make the close icon too big because we don't want customers to actually close it.”

Preface

In these three examples, found all over the web, the company lost sight of the user's needs and forgot to put the user first. Over the past 20 years, I've learned a lot about designing digital products. It's hard to pick all these individual lessons out because it feels like they've been compiled into a big UX operating system in my brain.

I'm not ashamed to admit that I'm a design purist. Of course, I value aesthetics, but I see them as a "hygiene factor" and a necessity. Beyond the veneer of aesthetics, I've always strived to produce software that's usable and powerful, where the features are instantly obvious or easy to discover and learn.

This book is a "shortcut to success" for less experienced designers and a challenge to some accepted thinking for seasoned UX professionals.

The principles are structured into broad sections such as typography, controls, journeys, and the wider field of UX practice. Feel free to dip in and out and use the book as a reference, although it has been designed to be read through in order, if you wish.

You might find yourself disagreeing with some of the principles, and that's OK. This is, after all, an opinionated book—but the disagreement will sometimes be a prompt to examine your thinking and evaluate if there might be a better way to accomplish your users' goals.

I hope you enjoy the book and that it helps you to become a better UX professional so that you can implement experiences that work, avoid common pitfalls, and grow your confidence to fight for the user.

Will Grant, May 2022

Share your thoughts

Once you've read *101 UX Principles, Second edition*, we'd love to hear your thoughts! Please [click here to go straight to the Amazon review page](#) for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

UX Field

The first principles of UX put the user at the center of your thinking. This section introduces the underlying principles needed to design user-centric products and highlights the importance of testing and iterative improvement of your creations.

#1

**Everyone Can Be
Great at UX**

This guide is for anyone who designs software products as part of their work. You may be a full-time designer, a UX professional, or someone who has to make decisions about UX in your organization's products. Regardless of your role, the principles in this guide will improve your products, help you to serve your users' needs better, and make your customers more likely to return to you.

Although various examples in this book feature a mobile app, website, web app, or some desktop software, the principles are ready to be tailored to a wide range of applications, from in-car UIs, mobile games, and VR, to washing machine interfaces and everything in between.

Developing the primary skills of empathy and objectivity is essential for anyone to be great at UX. This is not to undermine those who have spent many years studying and working in the UX field—their insights and experience are valuable—rather to say that study and practice alone are not enough.

You need empathy to understand your users' needs, goals, and frustrations. To “walk a mile in their shoes” requires you to approach user problems with respect—they're not stupid, your software is just too hard to use. You need objectivity to look at your product with fresh eyes, spot the flaws, and fix them.

User testing is essential and will reveal flaws you never imagined were there—talk to users early and often. It's easier to fix a product at the beginning—and almost impossible to fix at the end.

With this foundation of empathy and objectivity, you can learn everything else it takes to be great at UX.

Learning points

- UX isn't a talent you're born with—you can learn how to be good in this field.
- Objectivity and empathy are the two key personality traits you need to display—your problems and needs aren't always the same as your users'.
- This book aims to provide a shortcut to success with 101 tried-and-tested principles.

#2

**Be Strategic About
Using These Principles**

The principles we look at in this book are great and all, but how do we put them into action? Since the first edition of this book was released in 2018, the number 1 question I've been asked is a variation of "How do I put these principles into practice in my day-to-day work?" The best intentions of a principled UX professional are one thing, but the messy reality of working in a modern business is quite another. Here are the best tips I've successfully used over the years:

1. **Fight for the user.** Day-to-day business involves competing departments and priorities. KPIs and OKRs are fine for marketing and sales departments, but your job is to fight for the user—putting their needs front and center of the plans. Sometimes this will mean clashing with other teams, it's unavoidable, and you won't win every battle—but it's important you're there and your voice is heard. The sales team has goals to hit, but your job is to convince them—with best practice and data—that a full-screen unskippable video advert will do more to drive customers away than engage them.
2. **Build allies in your organization.** The most effective way to "get UX done" in an organization is to build allies across the business. Product managers, developers, and senior stakeholders (yes, including the "C-level" suits) all need to be onside and understand the value of UX. Take them along on your process.
3. **Understand the business goals.** You're the voice of the user, but you can't work in a vacuum—you need to understand the business goals and the plans for your products and services. You won't survive for long if you're *just* the voice of the user without enabling results for the business. Engage with product managers and help shape the roadmap and business plan.

4. **Build a culture of user research.** Involve your colleagues in user research, bring them to testing sessions, and disseminate your results to a wide group of people. UX designers and product people alike should be thinking about user needs and testing ideas with them every step of the way. Finally, bring these ideas back to the stakeholders in the business: there's nothing as powerful in changing a CEO or CMO's mind than presenting them with a trove of research findings from real customers.
5. **Drive data-driven decisions.** Use data to steer your high-level decision-making—this point is less about qualitative data (data you can describe with language, not numbers) from user testing, and more about quantitative data (data you can measure with numbers) from analytics and other metrics. Use this data to help you make better decisions in terms of product features, missing services, and whole approaches to doing business.

Learning points

- Fight for the user
- Build allies in your organization
- Understand the business goals
- Build a culture of user research
- Drive data-driven decisions

#3

**Don't Be Afraid to Ship
Something Simple...**

One of the (many) great things to come out of the software industry's adoption of Agile methodologies in the early 2000s is the behavior of shipping early and often. From the Agile Manifesto principles:



Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

—Agile Manifesto,

<https://agilemanifesto.org/principles.html>

You don't wait for a “big bang” release; you don't perfect every aspect of your product—you release *valuable* software out to users as soon as it's ready, and you update it continuously. Some of the ways that this principle is often violated in modern software development are:

- You want to add just *one* more feature before you think the product is ready
- The marketing team wants to wait until the campaign is ready to promote the feature
- Your competitor offers more features and you need to match them

Trust me—your users don't care about these reasons. Your product doesn't need to be a *whole new category* of product; it just needs to help users *get their shit done*. Fewer, better features are better for the user experience than trying to cram too much in, pushing deadlines and developers until your user ends up with 100 half-baked features instead of 25 excellent ones—and a later release date.

Try to remember that, in most cases, most users will only use your app for 1% of their day—you work in it all the time so it's easy to lose objectivity. Ask yourself: “Do we really need *Y*? Would users be happier with a better *X*?”

Don't Be Afraid to Ship Something Simple...

Create a well-researched, well-defined scope for your first version, so that—when stakeholders inevitably get *the fear* about missing features compared to competitors—there is justification and strategy for continuing with the minimal version release and then iterating later based on what you learned.

Learning points

- Keep it simple; don't reinvent the wheel
- Ship early and often, delivering valuable features
- Don't chase competitors' feature sets; sometimes less is more

#4

**...But Complexity Can
Be Good for Some
Users**

A few years back I did some work for an insurance company—I know, I get all the exciting projects, right? The software was a web-based UI that call center staff used to manage incoming customer inquiries. In this web application I was tasked with improving, a customer would call up to change their policy, or get a new one, cancel their cover, and so on, and the user would need to search, edit, save, or delete as necessary.

It had recently undergone a redesign and my job was to work with users to understand how successful the redesign had been and what features were succeeding and which weren't. The new UI was clean, modern, airy—based on **Material Design**, with lots of negative space and big, clear typefaces.

It turned out that all the users hated the new system. The information density was too low. Instead of seeing all the customer information on one (or two) screens, tightly packed together, the user now had to navigate through multiple screens, scrolling view after view to see the details they needed. They even had to navigate back and forth between two views—waiting on the page load each time—to do basic things like checking a policy expiry date.

Nobody had asked users, “What is wrong with the current system?” Instead, they optimized the new design for user needs that didn't exist (see #6, *Test with Real Users*).

Many skilled users who work in an enterprise environment rely on software to do lots of the same thing, repeatedly. Making enterprise software too simple—by trying to be more “consumer-looking”—just hides necessary functions away from the user and makes it frustrating and slow, increasing cognitive load.

...But Complexity Can Be Good for Some Users

For example, an aircraft cockpit is dense and complex—but it needs to be, because complexity is good for these users:



Figure 4.1: Look at all those controls. Image by Honglin Shaw on Unsplash

Similarly, professional audio tool Ableton Live has a dense, complex interface, but musicians spend all day, every day here and love the productivity it offers:



Figure 4.2: Ableton Live UI

Learning points

- Be wary of making complex products too “consumer-looking” and simple
- Don’t follow trends; focus on what your user needs
- Complexity doesn’t always mean a bad user experience

#5

**Use A/B Testing to Test
Your Ideas**

We can learn a great deal from our users with interviews and feedback studies, but it's hard to achieve a large scale with these techniques—it's just too time-consuming to talk to 1,000 or 100,000 users. A/B testing (where you compare two designs) and multivariate testing (where you change multiple design elements), are a great way to gather results and test your designs with large audiences because they can be run by robots, not humans.

A/B testing is a technique that involves splitting a user base into two groups or populations and showing two different designs ("A" and "B") to each group to see which design performs better. A/B testing gives the best results with larger populations of users; over 1,000 is a good sensible minimum to give meaningful results.

It's that simple: set up two different versions of your UI and use some software (many free and paid services are available) to serve them equally to visitors. Tag each group with a label in your analytics software and you can see easily which design performed better against your chosen conversion metric: whether that's checkout conversions, sign-ups, or something else.

Start with a hypothesis to test. In the example below: “We suspect that making the purchase more instant and presenting the option in a brighter color will cause more checkout conversions.”

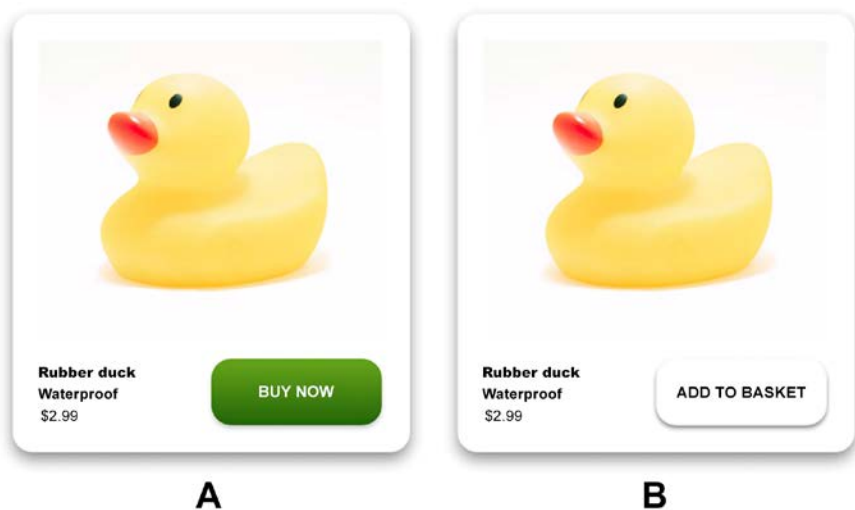


Figure 5.1: Run version A and B past 1,000 users and see how many ducks you sell for each cohort

All well and good, and a useful research tool to have in your UX research toolbox. One last word on ethics—make sure you’re helping the user, not just optimizing for company needs. That way lie **deceptive patterns** (see #101, *Don’t Join the Dark Side*).

Learning points

- Use A/B testing to work out which design performs better
- Make sure “performs better” means the design performs better for the user
- A/B testing works well with two distinctly different designs, rather than just tiny changes

#6

Test with Real Users

Nothing in this book means anything unless you test with real people. Moreover, you need to test with real *users*: not your colleagues, not your boss, and not your partner. You need to test with a diverse mix of people, from the widest section of society you can get access to (within your target audience, of course).

User testing is an essential step to understanding not just your product, but the users you're testing—what their goals really are, how they want to achieve them, and where your product delivers or falls short. You'll not only understand your users better, but you'll reduce development time by short-circuiting the feedback loop and getting problems fixed much earlier in the product life cycle.

It's never too early to start testing—an unfinished prototype or even paper prototype (cards or notes that you move around on a desk) can yield valuable insights—so get your product in front of users as soon as you can. Additionally, you can save yourself (and your developers) a huge amount of work and rework by heading in the right direction early, rather than making lots of false starts.

So, what are you testing? User tests are, in themselves, a broad spectrum of activities ranging from “guerilla-style” tests—where you approach a random person and ask them to perform a task in the app—through to specific feature-based tests, where an expert user (usually with domain knowledge) is asked to perform a complex task. Either way, you need to start with an idea of what you're testing, tuned to both the complexity level of the product and the domain knowledge that a user needs to operate it.

A few of the most common usability testing methods are:

- Guerilla testing: informal and ad hoc, as mentioned above. This method is cheap, rapid, and great for getting an early steer on your proposed solutions.
- Lab tests: performed in controlled conditions with a moderator. This method provides greater insight into the users' motivations, but they can be costly and slow to yield results.

- Remote testing: performed unmoderated, where the user is left to their own devices. This method does not provide the same depth of feedback, but you can scale these tests up to thousands of participants.

There's a myth that user testing is expensive and time-consuming, but the reality is that even very small test groups (fewer than 10 people) can provide fascinating insights. The nature of tests with a small number of participants is that they don't lend themselves well to quantitative analysis, but you can get a lot of qualitative feedback from working with a small sample set of fewer than 10 users.

There's research to show that testing with as few as five users will uncover 85% of usability problems in a single test. This startlingly high number is arrived at thanks to the **Poisson distribution** and some math.



Why you only need to test with 5 users, NNG: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>

Too often, products aren't tested, the thinking being that "we'll just hear what users don't like and fix it." The problem is that *your users won't tell you*; they'll just leave. The near-infinite choice of products and services on the web, app stores, and a myriad of devices means that the user has no incentive to stay, complain, and help you to improve your product—it will simply fail.

The cost of switching to an alternative for a user is almost zero—a quick Google for a competitor and they're gone for good. Test with real users and listen to them, and you'll build something they love.

Learning points

- Test your product early and with real users within your target audience
- Test with a diverse group of people from different ethnicities, ages, genders, and backgrounds
- You only need to test with a small group to get huge benefits

#7

**Nobody Cares
About Your Brand**

I don't mean brand in the sense of visual identity—a good logo, wordmark, or tagline is a great idea. I mean *brand* in the modern sense—a woolly definition that's come to be commonplace over the past 10 years or so.

The word brand has come to allude to the company or to stand for the entire personality of a corporation or product. It is seen as the “feeling” of interacting with products and services, and inevitably the core interactions of those products.

The problem with this approach, developed for over a decade by multinational branding corporations and advertising agencies, is that *we already have a discipline for this: UX*. By crafting a product to adhere to a brand (in the modern sense of the word), we defer control of the UX to the marketing and branding teams, not the UX professionals.

I'm not talking either about the megabrands with a billion customers; Apple, Google, Coca-Cola, Microsoft, Nike, and so on are so big and their brands so powerful that it *does* and *should* make a difference when it comes to how their products are designed.

What about your brand, with a few thousand or tens of thousands of customers, or your small company, product, or newly launched start-up? *Nobody cares*. Harsh, but true. None of your users care about your brand. They care about what your product or service *lets them do*. They care about how your product improves their lives and enhances their productivity, and so on.

The experience of your product *is* your brand and it shouldn't be designed by a marketing team, but by UX people. This is also your competitive advantage against the big, lumbering dinosaurs that have to adhere rigidly to brand guides.

Don't let the brand guide ruin your product with:

- **Unreadable brand typefaces:** Just use the native system font stack.
- **Branded splash screens:** Just show me the damn app.
- **Build-your-own nightmarish UI controls:** Oh, the things I've seen...

- **Awful, unreadable contrast ratios:** Don't stick to the brand palette if it doesn't work in your product.
- **Unnecessarily quirky copy:** The wacky humor on the side of a smoothie bottle—just get to the point!

A brand can help to enforce consistency, but, if you're a decent designer, you shouldn't need a brand guide to tell you how to build a consistent UI. Brands are bullshit, so focus on the UX and the experience *becomes* the brand.

Learning points

- Nobody cares about your brand, only about what your product lets them do
- A good UX is better than a good brand
- Fight for the user, not the brand guide

Typography

Typography helps to create a visual hierarchy that can guide users through your product in a logical and easy-to-follow manner, which can in turn help users to achieve their goals. Well-designed typography can help to give your product a more professional and polished look, which can instil trust and confidence in your users.

#8

**Don't Use More than
Two Typefaces**

It's a pretty common point of confusion, but typefaces and fonts are different things. “Proper” design professionals call them “typefaces,” while fonts are the files on the device that the software *uses* to display the typeface. Fonts are the paint on the palette, while the typeface is the masterpiece on the canvas.

For example, Helvetica is one of the most widely used typefaces, but a font is a particular set of glyphs within that font family: for example, Helvetica Condensed Bold, 10pt.

Regardless, too often designers add too many typefaces to their products. You should aim to use two typefaces maximum: one for headings and titles, and another for body copy that is intended to be read in depth.

Use weights and italics *within* that font family for emphasis—rather than switching to another family. Typically, this means using your corporate brand font as the heading, while leaving the controls, dialogs, and in-app copy (which need to be clearly legible) in a more proven, readable typeface.

Using too many typefaces creates too much visual “noise” and increases the effort that the user has to put into understanding the view in front of them. What's more, many custom-designed brand typefaces are often made with a punchy visual impact in mind, not readability.

Learning points

- Use two typefaces, maximum.
- Use one typeface for headings and titles.
- Use another typeface for body copy.

#9

**Users Already
Have Fonts on
Their Computers,
So Use Them**

Yes, your corporate brand font is lovely. It's so playful and charming but it delays the web page display by 4 seconds as the font needs to be downloaded from the server and rendered—and nothing appears until it loads—driving your users crazy.

Including custom display fonts for headings and titles is fine; it helps to brand the product and adds some visual interest. However, using custom fonts for body copy is generally a bad idea.

First of all, these fonts have to be loaded from somewhere, whether it's Google Fonts, Typekit, or your own **content delivery network (CDN)**. This means that there is an overhead in getting the font files down to the user's machine. Content-heavy pages will often break while the correct fonts are downloaded and rendered—the dreaded **Flash of unstyled content** or **Flash of unstyled text**.

Secondly, if, by specifying wild and wonderful body copy typefaces, you think you're exerting some control over the end result, then think again: responsive design, assistive technologies for disabled people, and thousands of different devices out in the wild mean your pages will look different for everyone.

Luckily, whether your user is on a phone or a desktop, Windows or Mac (or Linux), they have some beautiful, highly readable fonts already installed and waiting to be used. The “system font stack” is a CSS rule that tells modern browsers to render type in the system-native typeface.

In most cases, using system-native fonts makes pages appear more quickly, and the type looks sharper and more readable.

The CSS code below tells the browser to display the text in one of three commonly used styles, only using the built-in font files on the device.

- Sans-serif typefaces like Helvetica or Futura—most modern sites and applications use sans-serif typefaces.

```
font-family: -apple-system, BlinkMacSystemFont,  
avenir next, avenir, segoe ui, helvetica neue,  
helvetica, Ubuntu, roboto, noto, arial, sans-serif;
```

- Serif typefaces—the more “traditional” looking typeface with little “hooks” (called serifs) hanging off each letter.

```
font-family: Iowan Old Style, Apple Garamond,  
Baskerville, Times New Roman, Droid Serif, Times,  
Source Serif Pro, serif, Apple Color Emoji, Segoe  
UI Emoji, Segoe UI Symbol;
```

- Monospace typefaces—so called because the spacing between each letter is the same, designed to increase readability for code examples.

```
font-family: Menlo, Consolas, Monaco, Liberation  
Mono, Lucida Console, monospace;
```

Please, just use the system font stack.

Learning points

- Use the system-native fonts that your users already have installed.
- System fonts will typically render better than custom ones—and will include glyphs for many non-English languages too.
- Using native fonts speeds up page load time.

#10

**Use Type Size and
Weight to Depict an
Information Hierarchy**

This is a simple, but effective, method for organizing your views and making them instantly understandable for a wide range of users. Let's look at an example of how not to do this in an imagined "Calendar" app user interface:

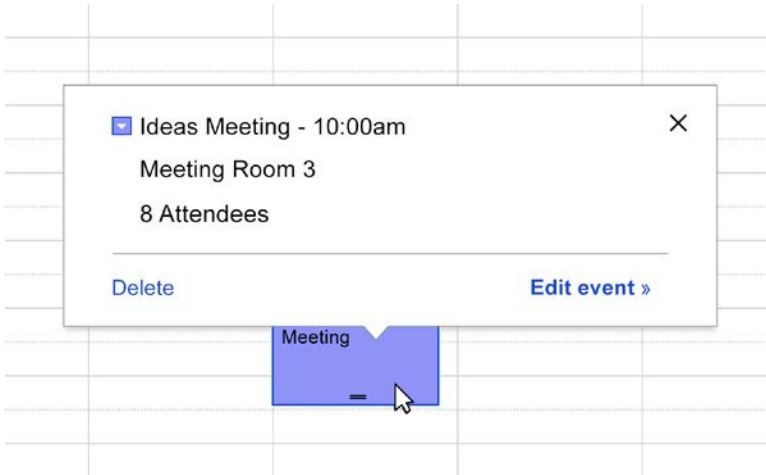


Figure 10.1: This example shows all the type in the pop-over at the same size and weight

Simply by altering the type size by a noticeable factor, we can show the user the most relevant information first:

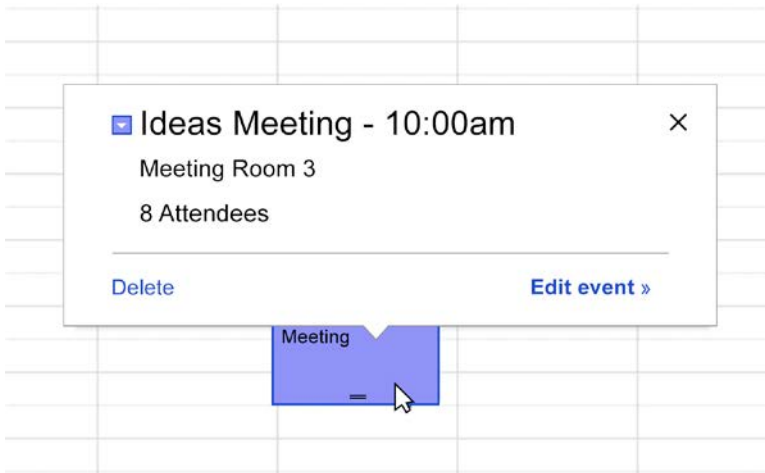


Figure 10.2: This example shows how using different type sizes (or weights) can help differentiate information

Scale up the information that you want users to see first, or that you think they'll find most useful, and they can read on further for extra detail.

Another option is to use type weight instead of scale, as in this example. Using a selection of bold, regular, and light typefaces has a similar effect to altering type size, but reduces the demand on the screen space.

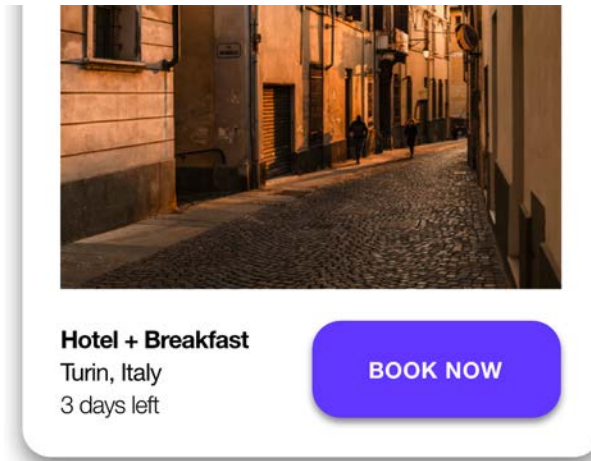


Figure 10.3: Three different weights, all the same size

Many news sites and factual journalism sites have settled on this format:

Headline that tells you something

Subtitle that adds context and poses more questions

Body copy that expands on the story by adding detail progressively through the copy. Keep reading to the end to learn less and less important details.

Use Type Size and Weight to Depict an Information Hierarchy

The exact same technique can be employed in user interface design to great effect.

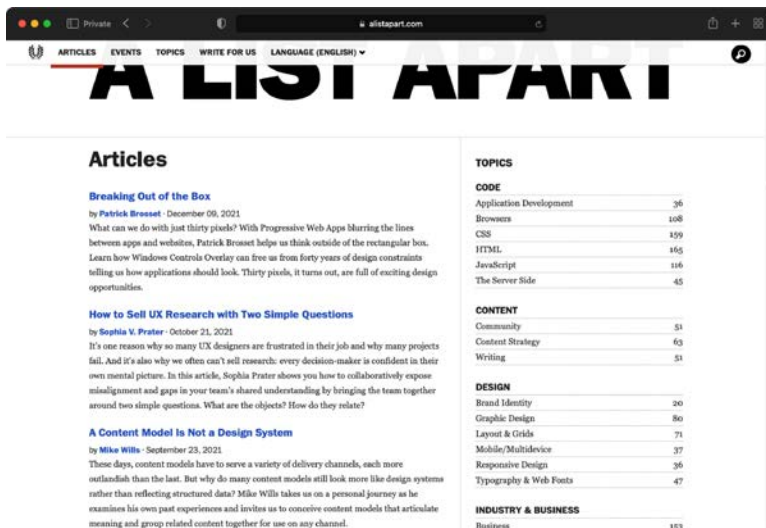



Figure 10.4: Design blog 'A List Apart' uses typographic hierarchy to excellent effect on its article list



Find a balance and don't overdo it. If too many elements on the page are large, then they lose any sense of hierarchy and emphasis. When everything is important, nothing is important.

Learning points

- Type size and weight can indicate the importance of information to users.
- Use at least two, but no more than three, type sizes.
- Think about which fragments of information are most important to your users.

#11

**Use a Sensible Default
Size for Body Copy**

Your customers will be reading a lot of text across your app or site, so how big should the type be?

The days of fixed-size type are long gone. Most browsers on desktop and mobile will let users scale type up and down, switch into “reading mode,” and apply system-wide accessibility settings, like large type and high-contrast colors.

With that in mind, all you’re doing here is setting the *default* type size that appears when the product is first opened. Ideally, the type should be big enough to be readable, but not so big as to overwhelm the user or take up too much space in a crowded view.

Body copy in 16px, with a 1.5 line height and “auto” or “default” character spacing, is usually a safe bet and a good default for the vast majority of your users.

This can be achieved easily with the following CSS:

```
body {  
    font-size: 16px;  
    line-height: 1.5;  
}
```

Trying to set your own character spacing is usually unnecessary for body copy because the browser will do a better job of text rendering than you can.

Learning points

- Body copy in 16px, with a 1.5 line height and “auto” or “default” character spacing, is the “gold standard” for readable text.
- Allow users to scale your type up and down for their device.
- Allow users to display your content in a way that works for their needs.

Controls

There are many different types of controls in UX and UI design. Some of the most common are buttons, checkboxes, sliders, and text fields. Each of these controls serves a different purpose, so it's essential to choose the right one for the task at hand.

#12

**Use an Ellipsis to
Indicate That There's
a Further Step**

If your user sees a “Remove” button, how do they know if pressing it will:

- Remove the thing they’re looking at?
- Ask which thing needs to be removed?
- Ask them if they really want to remove the thing?
- Instantly remove all their stuff?

Label the button “Remove...” and the user will have a good idea that there’s another step before all their stuff is removed. Most users will infer from this that the button is the first part of a multi-part process and there will be a second step to confirm or cancel the action. If a control requires an extra step to perform its action, include an ellipsis (...) in the control:

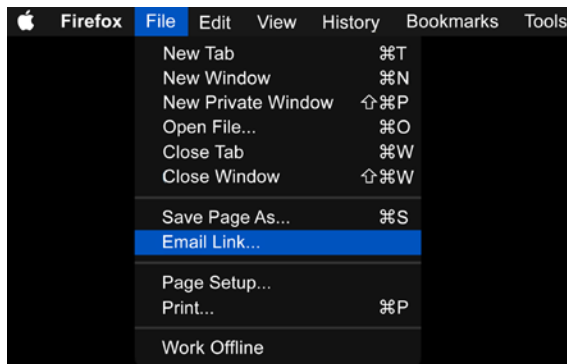


Figure 12.1: New Tab just opens a new tab, while Email Link... will ask for more information in the next step

These little dots are a great example of invisible design: most users will never have even noticed them, but they impart a subtle message as a user’s experience builds over time. They don’t get in the way and they “just work.”

Learning points

- If the user needs to perform an additional action, show an ellipsis.
- An ellipsis can give the user more confidence that there's a further step to confirm an action.
- Users may well have unconsciously learned what these dots mean in a UI, but you should validate this understanding while testing your product.

#13

**Make Interactive
Elements Obvious and
Discoverable**

The flat design aesthetic, born out of Microsoft's Metro user interface, rose to near ubiquity in the late 2000s. In iOS 7 and Android's Material Design, these extremely minimal visuals are still the go-to look for modern web applications. Although, the flat design of the late 2000s has "softened" over the years to introduce more visual affordances. Flat design has performed poorly in user testing over the years. It's style over substance and it forces your users to think more about every interaction they make with your product. Stop making it hard for your customers to find the buttons:

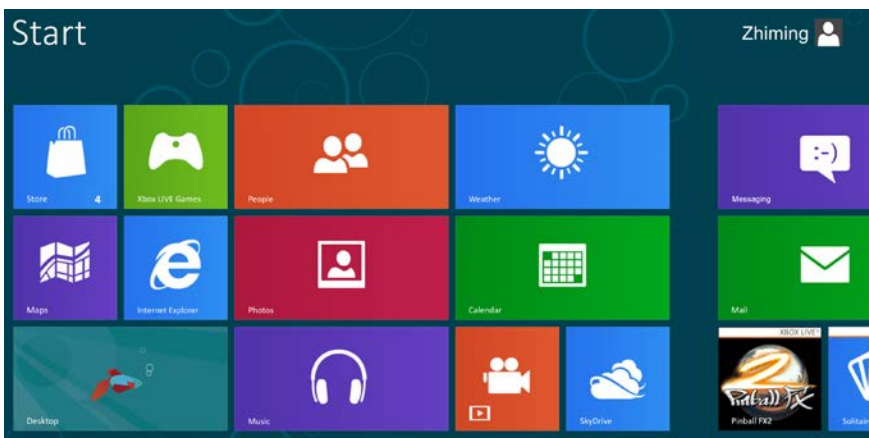


Figure 13.1: The "Metro" user interface in all its "what is clickable?" splendor

There are parts of your UI that can be interacted with, but your user neither knows which parts these are, nor wants to spend time learning this. They have used buttons in real life, many times—on elevator controls, on their oven, and in their car—so they understand how a button works:



Figure 13.2: Buttons that exhibit visual affordances such as texture and pseudo-3D shadows (left) consistently perform better in user tests than those without them (right)

By drawing on real-world examples, we can make UI buttons that are obvious and instantly familiar. The human visual system is tuned to see depth, and by removing the illusion of depth from your UI, you remove a whole layer of information for the user.

Buttons in real life look pushable: they're raised or they suggest an obvious way that they might move if pushed. For example, they might have an indicator light and look more prominent when enabled. You should copy these features into your UI.

The inverse is also true: there are real-world buttons that *don't* look pushable—flat capacitive buttons on car park machines and coffee machines spring to mind—and these buttons are often accompanied by a stuck-on, handwritten *press here for ticket* note.

Using real-life inspiration to create affordances, a new user can identify the controls right away.

As with Jakob Nielsen's second usability heuristic: "Match between the system and the real world," the concepts in your system should map onto concepts the user is already familiar with from the real world.



Nielsen's second heuristic can be read about in more detail here:
<https://www.nngroup.com/articles/match-system-real-world/>

Create the visual cues your user needs to know instantly that they're looking at a button that can be tapped or clicked.



Figure 13.3: Bringing flat design to the real world has consequences

Lastly, the opposite is also true: don't make non-button elements look like buttons if they're not.

I've talked a lot about "buttons" in this chapter, but the same is true of all interactive elements. Frequently badly designed controls with few affordances include:

- Accordion controls: Plus or minus, will it open the accordion, or add an item to a list?
- Sliders: Handles that are too small or not visually distinct enough.
- Form input fields: Simply making an empty rectangle isn't always enough to indicate to users "this is where your data goes!"

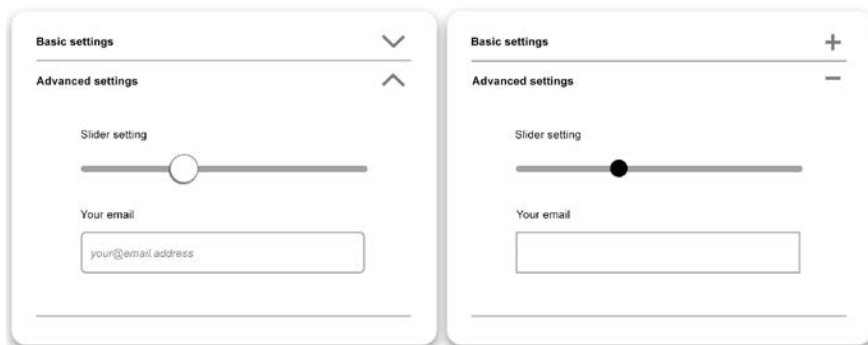


Figure 13.4: Which one of these example user interfaces is more instantly approachable, easier to use, and familiar for users?

Learning points

- Make buttons look like buttons.
- Make anything that can be interacted with look interactable.
- Borrow ideas from real-world experiences in your UI.

#14

**Make Buttons a
Sensible Size And
Group Them Together
by Function**

The US psychologist Paul Fitts wrote a paper in 1954 called *The information capacity of the human motor system in controlling the amplitude of movement* (<https://www.ncbi.nlm.nih.gov/pubmed/13174710>), which was published in the *Journal of Experimental Psychology*. Fitts' work would go on to be one of the most well-studied models of human motion.

To dumb Fitts' law down for us UX people, rather than psychologists, the core concept that applies to us is:

“*The time required to rapidly move to a target area is a function of the ratio between the distance to the target and the size of the target.*”

If you're building a user interface, it's really simple to do this: make buttons big enough, and placed in such a way that users can efficiently find them and move between them—ideally grouped by function:

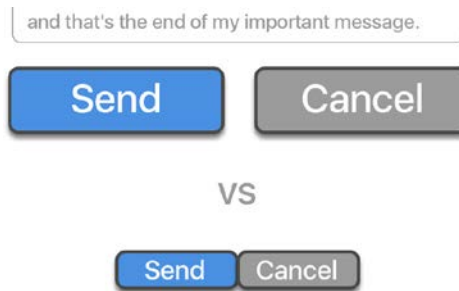


Figure 14.1: Which is easier to use and less prone to a misclick?

This rule is most often forgotten when designers “scale down” a desktop interface for mobile—without considering that controls still need to be a reasonable size to tap on (see #73, *Make Tappable Areas Finger-Sized*).

A great anti-pattern example is those tiny x buttons to close pop-up modals: it's almost as if the advertisers don't want you to close them...

Learning points

- Make buttons big enough that they can be tapped or clicked easily.
- When you design for small screens, remember the controls still need to be big enough.
- Don't cause misclicks by placing buttons too close together.

#15

**Make the Whole
Button Clickable,
Not Just the Text**

Maybe this is simply a pet hate of mine, but I see it often enough to mention it here. Buttons often feature text and sometimes developers *only make the text clickable*, not the whole button. Meaning that, if you're a couple of pixels out and miss the text (but hit the button)... nothing happens.

We're all familiar with thinking, "Did I not click that?" and often this is the cause. If you're imitating the real-world behavior of a button, then make it behave like a real button. This includes giving the user some feedback that the button has been successfully clicked (or tapped). This could be with a change of shade, a slight 1-pixel-movement "down" or a subtle audio effect.

All interactive elements should have default, active, hover, and disabled states, with subtle visual differences so the user can infer their interaction state with that control.

You get bonus points for showing the "hand pointer" to desktop users. Sloppy CSS code means that some web apps don't show this and it's unforgivable.

Learning points

- Your button should look and behave like a button—clicking anywhere on the button should activate it.
- Make the pointer turn into a hand when you hover over the button on desktop.
- Give the user some visual feedback that the button has been clicked—on mobile this could be audio or vibration, but avoid this for desktop.

#16

**Don't Invent New,
Arbitrary Controls**

These could be:

- An isometric pseudo-3D wheel to choose the color of your car
- A volume dial that you must click and drag up-and-down to “rotate”
- A button you must click and hold for a few seconds to indicate that you *really* want to do this action

Just don’t invent them. As designers, we already have a rich palette of existing controls to choose from. If you’re thinking about making a new UI control, please stop and think about how hard it will be for users to learn yet another interface pattern. I promise you this—there’s already a way to do what you want to do.

However, every now and then, something new comes along that is genuinely an advance in UI. Back in 2008, Loren Brichter made a Twitter app called **Tweetie**, with a unique pull-to-refresh interaction. Pulling the view down would show “release to refresh” and releasing would show a spinner. The pull-to-refresh interaction went on to be included in Twitter, which bought Tweetie, and then in iOS and Android apps in their hundreds.

“Swipe right” in dating apps to choose a potential partner and “double-tap to like” in Instagram get honorable mentions as some more recent, novel interaction patterns. As you can see, inventing something new that works and doesn’t confuse users is the exception rather than the rule.

So, don’t design new controls, or invent new interaction patterns... unless they’re astonishingly good.

Learning points

- Don’t invent your own UI.
- There’s almost certainly a UI component out there that does what you need.
- Don’t make users learn your new thing.

#17

**Search Should Be
a Text Field with a
Button Labeled “Search”**

The search function has, over the years, been over-designed. One common anti-pattern is hiding search behind a control to activate it. Slowing the user down and adding an extra step might remove an input field from your view, but at the expense of familiarity and ease of use.

If you're offering your users a search function, then show them a text field with a search button. If you're using an icon, then use a "magnifying glass" icon. This is the archetype and using anything else no longer makes any sense.



Figure 17.1: The “gold standard”

The placeholder text (“type to search” in the above example) is a vital and often overlooked piece of UI copy. In a couple of well-chosen words, the UX designer can give valuable hints and context to the user. For example: “type to search” indicates to the user they’re searching within a set of data—but “filter results” would indicate that they’re instead limiting the data set already in view.

Care should be taken with regard to accessibility; ensure that placeholder text is still readable and not low-contrast, and don’t replace perfectly good form labels with placeholder text. It’s an added bonus and not a replacement for form labels.

Instant search, where typing instantly shows results to the user, is often better than showing a separate search results page. Instant results are obviously more immediate and can let the user navigate to a result without breaking their flow inside your product.

Search Should Be a Text Field with a Button Labeled “Search”

On a mobile phone screen, there may not be enough space to always show the search field, but I’d still encourage you to evaluate whether you can fit one in. Tucking the search field into the top of a scrolling view can work well.

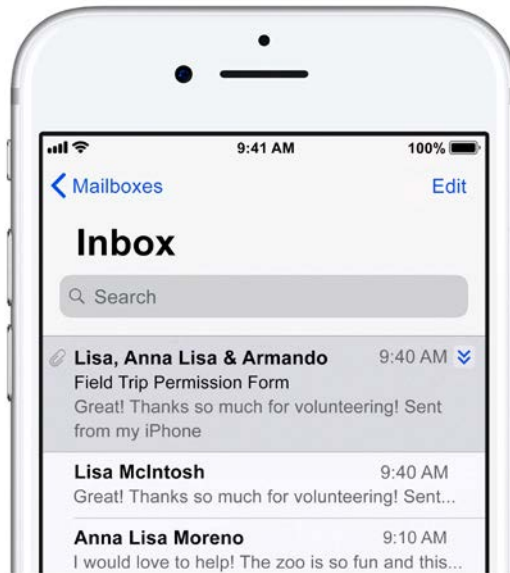


Figure 17.2: Search at the top of a list view that only appears when “pulled down”

Bonus points: when the user taps the **Search** tab in a mobile app, show the search view, move the cursor to the search field, and show the device keyboard for them. The Spotify mobile app does this particularly well:

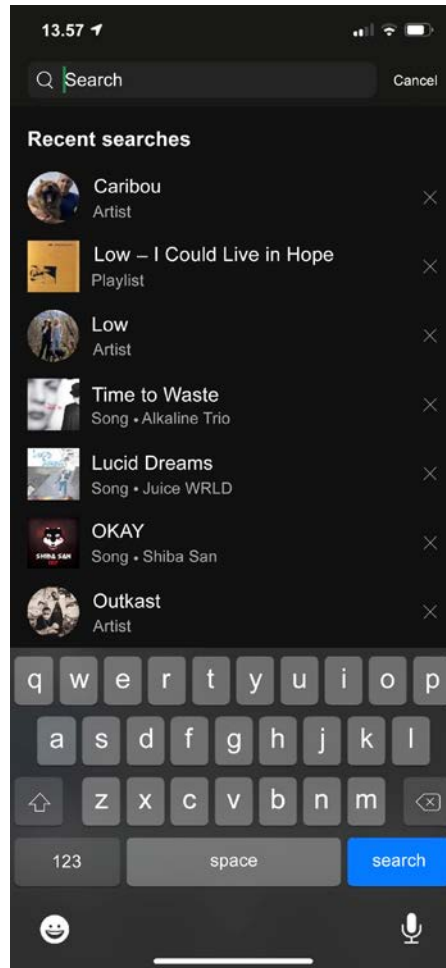


Figure 17.3: Spotify search function

Tapping **Search** in Spotify takes you straight to the search view with the input focused and the keyboard visible—and even includes your recent searches.

Search Should Be a Text Field with a Button Labeled “Search”

Learning points

- Search should be a text field with a search button
- Only use the “magnifying glass” icon for search
- Move the focus to the search field when the **Search** tab is tapped on mobile

#18

**Sliders Should Be Used
for Non-Quantifiable
Values Only**

Designer: “Oh, cool, this UI kit has a nice-looking slider; let’s use it for everything!”

User (trying to set a value): *smashes up phone*



Figure 18.1: I was trying to select 86

If you’ve ever fiddled with a tiny touchscreen while trying to set a value with a slider, you’ll be familiar with the preceding scenario. Even on a desktop screen with a mouse, it’s a pain.

Slider controls should never be used for setting specific numeric values. They are, however, great for volume controls, brightness, and color mix values, where the slider can be used to pick a value in a more qualitative way, and the actual numeric value itself doesn’t need to be precise.

For precise numbers, refer to #19, *Use Numeric Entry Fields For Precise Integers*.

Learning points

- Slider controls should never be used for setting specific numeric values
- Use sliders for adjusting qualitative settings such as volume and brightness, where a precise value isn’t needed
- Make the slider control a sensible size that can be easily grabbed by the user’s pointing device

#19

**Use Numeric Entry
Fields for Precise
Integers**

If you're trying to get an integer (a whole number) from a user—for example the number of widgets they want to order or the number of days an event runs for—it makes no sense to offer them a free text input field where they can enter “a few” or an emoji. A numeric entry field in HTML is:

```
<input type="number">
```

This will display slightly differently on different devices—and that's the whole point. By adapting to the control system of the client's device, the user gets a simpler entry and makes fewer mistakes. You also get fewer emojis in your database.

Users abandon forms because:

- a. they're too long,
- b. they ask for too many details, or
- c. because it's difficult to enter information into the form.

Therefore, a huge benefit here is that this will improve your form conversion rates by giving users on both desktop and mobile a quick, painless way to enter numbers into forms.

Learning points

- Numeric input controls should be used for setting specific numeric values
- Let the browser or device determine the best input method; don't build your own numeric entry control
- Forms always require more effort from users than just consuming content, so minimize the number of “things” you ask them for

#20

**Don't Use a Drop-Down
Menu If You Only Have
a Few Options**

A drop-down menu in the user interface is designed to expand when clicked and present a range of options. This is fine for customization, where there *genuinely are* plenty of options.

There is, however, an overhead to operating a drop-down menu: the user needs to click to open, scroll to the correct item, and then click to select. On a mobile device, this can be even slower, as the user will be using a smaller screen.

If you only have two or three options, then don't jump to using a drop-down straightaway. Consider whether the options could be better presented to users with a different kind of control (radio buttons—for single choices, sliders, and so on).



Figure 20.1: The anti-pattern: this should clearly be a cheese toggle, not a cheese drop-down

Sort your options into a sensible order—alphabetical or numerical—rather than random. Don't be the app that asks users to select a floor of a building in alphabetical order: "First, Fourth, Ground, Second, Third." Yes, I've seen this in the wild!

Very long drop-downs—for example, country selection—can benefit from a mini search or filter control: begin typing "U" and only see "Ukraine, United Arab Emirates, United Kingdom," and so on. This allows your user to skip to the section they need.

Don't Use a Drop-Down Menu If You Only Have a Few Options

Mobile users have a head start here: most mobile operating systems will show a full width “picker” control for drop-down selection, which is much less fiddly to use on a small touchscreen.

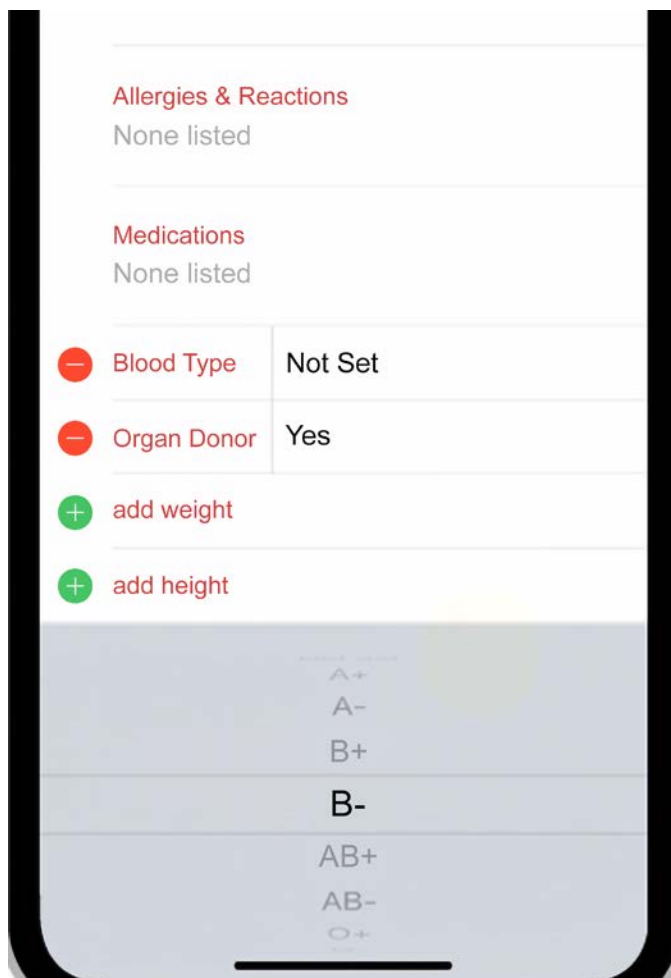


Figure 20.2: Picking a blood type with a mobile “picker” UI

Learning points

- Drop-down menus can be a pain, so only use them if you need to
- Offer the ability to search in very long drop-downs
- Drop-downs can be useful in mobile apps, as the user will have a specialized UI to use

#21

**Allow Users to Undo
Destructive Actions**

The ohnosecond (<https://en.oxforddictionaries.com/definition/ohnosecond>) is the split second when you realize you've made a terrible mistake. Your stomach sinks, your trembling hands lift from the keyboard, and you freeze. This moment of horror could be deleting a customer's records, emailing what you really think of your boss directly to your boss, or hitting "buy now" on 111 items when you only wanted one.

The best apps allow users to back out of such actions, either with undo controls or by giving users the ability to edit actions before they're final. Google's Gmail has had an optional "undo send" feature for quite some time. This stores your sent message in a "buffer" for 20 seconds, giving you that short grace period to cancel sending. If you just ignore it, you know the message will be sent shortly. This particular feature has saved me many times.

Users will feel more in control of your product because knowing they can undo every action and recover from mistakes will free them to experiment more with the product and hopefully get more from it.

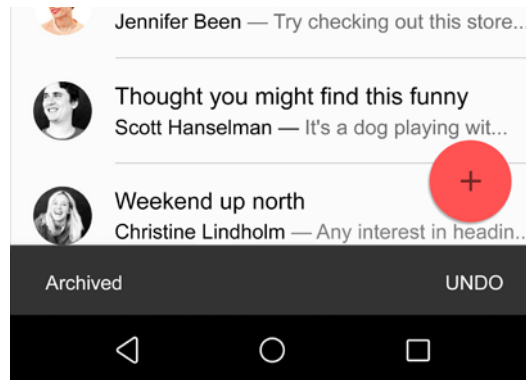


Figure 21.1: A toast-style notification with an optional "undo" control

From a UI perspective, a nice pattern is to include the **UNDO** control on a banner (or toast in Material design) that appears after an action. The user is informed that their action worked, they see a recap of what the action was, and they're given a quick shortcut to reverse that action.

There's a scale of destructiveness too—"delete all my stuff" should require a modal and for the user to double-confirm, maybe by typing "cancel" into a text field, while less severe actions should use the second-chance "undo" option mentioned above. Be forgiving because people make hundreds of mistakes every day and your users will love you if your product saves their ass just once.

Learning points

- Allow users to undo their mistakes
- Give users a greater sense of freedom and control
- Be forgiving; people will make mistakes

#22

**Optimize Your Interface
for Mobile**

The experience of using mobile apps (and mobile-optimized websites) can be either a joy or a deep frustration for users. This principle aims to summarize other points from across this book into a “UX for mobile cheatsheet” to help you deliver better mobile experiences.

Much of the frustration experienced by mobile users stems from the constraints the devices place on us. They have a much smaller screen, their touchscreens are more error-prone, and they’re typically on lower bandwidth connections than desktop computers. As a result, getting information out of the device and getting user input into the interface are both higher friction processes than would be the case on a desktop screen with a keyboard and mouse. The UI of these devices attempts to mitigate these constraints with the device-native features listed below.

Device-native features

The two main mobile operating systems (iOS and Android) have a great number of adaptations in their user interface and UX strategy compared to desktop operating systems.

Designers should use these device-native features where possible, and not re-invent the wheel each time.

Ensure navigation bars use the host OS style so that users can feel instantly familiar and at home, not wasting valuable cognitive load on working out a new navigation system.

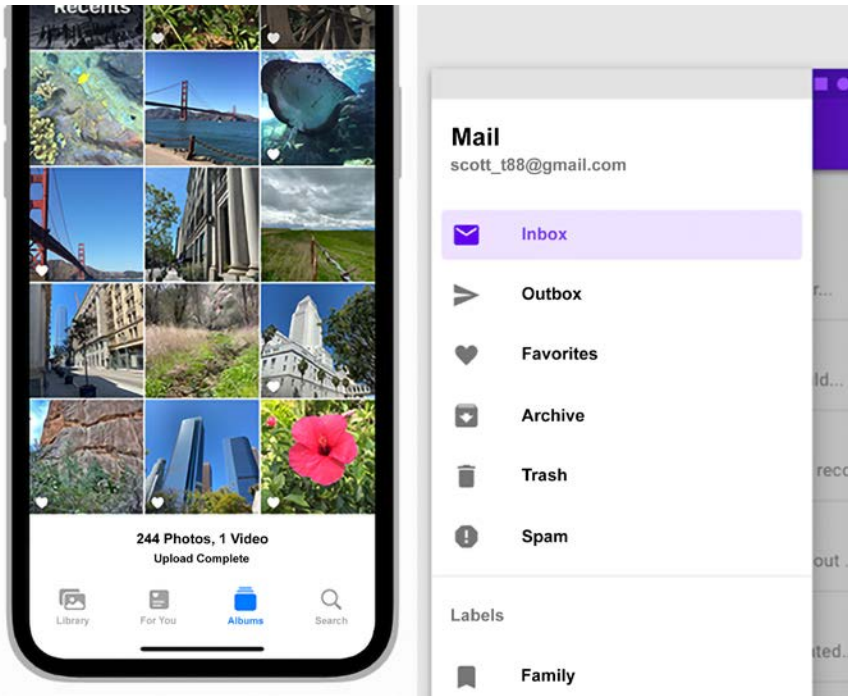


Figure 22.1: The iOS navigation tabs (left) and the Android navigation drawer (right)

There are also a great many specialized user interface components designed for date picking, numeric range entry, and telephone number input: use them.

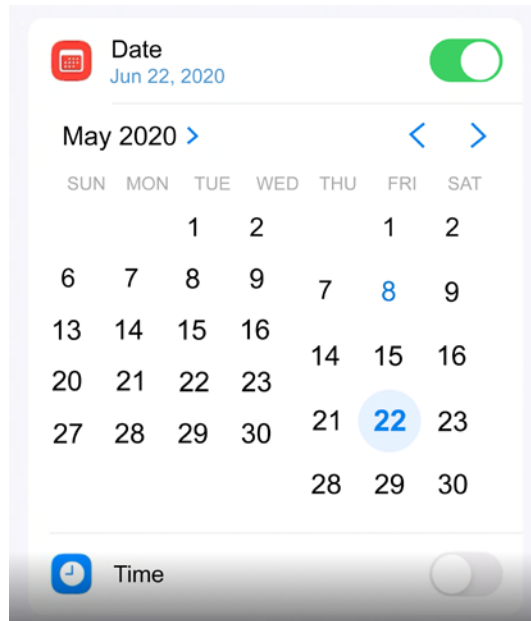


Figure 22.2: The iOS date picker control

Human-usable buttons

Accidental taps are among the most frustrating of mobile UX errors, and they can often be avoided by simply making buttons usable at mobile scale and not simply copying the desktop interface and scaling it down.

You can avoid many of these problems by adhering to Fitts' Law (see #14, *Make Buttons a Sensible Size and Group Them Together by Function*)—the Apple human interface guidelines recommend a minimum button size (along the smallest dimension) of 44 display pixels, while Android recommends 48 pixels.

Information entry

Form-filling on mobile is a constant source of angst for many users. Here are two tips to radically improve the experience of information entry on phones and tablets:

1. When the user finishes entering a selection, perhaps a drop-down select or a toggle, move them to the next field automatically
2. Split long forms up into sections so the user only has to navigate a few fields at a time, saving their entries as they go

Page shape and format

Desktop computers have been landscape in aspect ratio for decades, but mobile phones are portrait orientation. Even more confusingly, they can be rotated to landscape orientation easily.

The upshot of this is that desktop design doesn't easily translate across to mobile. We've all experienced those terrible sites where the desktop version has simply been scaled down to fit mobile. It doesn't work at all. Responsive design fixes most of these issues, but it does need to be done sensibly.

Responsive design allows the page styling to detect the viewport size and adapt accordingly. All modern frontend frameworks come with sensible defaults built-in for page layout and control sizing. It's almost always better to trust these defaults than to try to build your own responsive rules from scratch.

One example is scrolling—users are very happy to scroll vertically through pages of content, newsfeeds, and lists of items. Horizontal scrolling, however, is a big problem on mobile; it feels unnatural because of the limited width of the viewport and, even worse, it can interfere with the “back” gesture on most modern mobile browsers. Avoid horizontal scrolling.

Learning points

- Use device-native features where possible
- Make controls a human-usable size
- Don't force users to scroll horizontally

Content

Not every application is focused around content, but the vast majority are—posting, reading, searching, sorting, and so on. This section explores the best practices for giving your users a seamless experience when dealing with content of all types.

#23

**Use “Infinite Scroll” For
Feed-Style Content Only**

Infinite scroll—where the page just keeps scrolling, loading more items asynchronously as the user hits the bottom—is extremely handy for users.

Scrolling with a mouse wheel or a touchscreen is inherently quicker, and simpler, than clicking through pages, and, when the content is a newsfeed of Instagram photos or tweets, it's perfect. Always give the user an indication that there is more content loading, or tell them if they've reached the end.



Figure 23.1: Loading the next few items. I hope

However, infinite scroll should be limited to only a few types of content. If applied to finite lists (messages, emails, to-do items, and so on), then the user has no way of determining a beginning, middle, and end to the content. When used with this kind of content, infinite scroll is confusing and slower to use, so save it for feeds.

Although feeds used to be predominantly chronological—with the newest items first—more products (Facebook and Twitter, for example) are opting to offer users “algorithmically sorted” or “smart” timelines. The idea is (I assume) to offer users more relevant tweets or news stories at the top of their feed, and to allow promoted content like ads and sponsored posts to appear more prominently.

It may be personal taste, but I really dislike these smart timelines. First, they aim to serve companies and advertisers over the user, but, secondly, there is a real discoverability problem: you can't be sure what you're going to see when you open the timeline.

Is it the latest item? Is it the most relevant? What happens when you navigate away and come back? Often, you’re shown a new, regenerated list, making it impossible to find the item you just saw and thought you could come back to.

Often, a user will leave your infinite-scroll feed to take an action: maybe to favorite an item or post a comment. From there, they’ll hit back (on desktop), swipe back (iOS), or use the hardware back button (Android).

The question is, where should they end up?

- a. Right back at the top of the infinite-scroll feed again
- b. Exactly where they left off

The answer is obviously b), unless you *really* dislike your users. Sadly, a) is often the case on e-commerce sites, when browsing a long list of products.

Although the technical implementation details can be challenging, it’s worth putting in the effort to avoid disorientating users. Viewing a product and then navigating back should always return the user to the point where they left off.

Whether smart or normal, infinite scroll pages have a couple of other, often overlooked, problems. They “break” the scroll bar: the scroll position on the browser window is no longer accurate and it can no longer be used to navigate up and down the page. Lastly, page footers become impossible to reach. Bear that in mind, and consider giving the user a “back to top” control.

Learning points

- Use pagination for long lists of items
- Use infinite scroll for newsfeed-style content only
- Remember the user’s position if they navigate away from the feed

#24

**If Your Content Has a
Beginning, Middle, and
End, Use Pagination**

Continuing from #23, *Use “Infinite Scroll” For Feed-Style Content Only*, a paginated, multi-page list may seem “old school” but it has a few major benefits:

- It’s goal-oriented, so the user is trying to find the item they need in a list and pagination feels intuitive, instead of having to search through an endless list
- It remembers the user’s position and displays the current page to them
- It conveys a beginning, middle, and end to the content
- Users can use the scroll bar to navigate the page and they can reach the footer if they need to

If the user sees that there are “9,999 pages,” then they can choose to use a search, sort, or filter control. They can’t make that choice if they have no idea how many pages there may be.

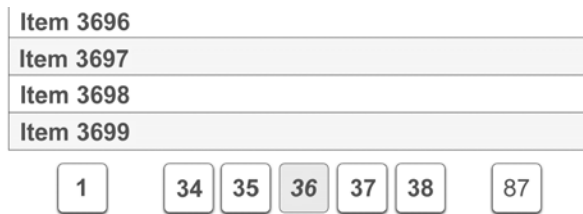


Figure 24.1: *A great paginator*

Show the user the current page, some pages before and after it, and the lowest and highest pages in the range. Adding “next” and “previous” buttons can be a useful aid for mobile users who may find the page numbers too fiddly to tap.

Given all of this, users won’t find it easy to search through very long lists: it’s just too cognitively arduous. A search, sort, or filter control should be considered mandatory on lists of more than a few pages.

Learning points

- Use pagination when the content is finite
- Show the user the current page, nearby pages, and the highest and lowest pages in the range
- Offer the user search, sort, and filter controls

#25

**Allow Users to Accept
or Reject Cookies with
One Click**

The law (and resulting directives) around cookies is often misunderstood, leading to bizarre experiences where users are blocked from visiting sites or inundated with complex modal dialogs they must navigate through before they can view the site.

Like most laws, it began with the best intentions. Ten years ago, the web was a “wild west” of tracking and privacy-invading profiling of users across websites. The EU aimed to give people back some rights over their personal information and protect them from non-consensual harvesting by third parties.

The EU ePrivacy Directive states that:



No cookies and trackers must be placed before prior consent from the user, besides those strictly necessary for the basic function of a website.

Analytics cookies allow teams to understand detailed information about who visits the site and their on-site behavior, and these don’t count as “basic functions,” so the user needs to give their consent. Analytics cookies matter to marketing people, so naturally, they would rather the users didn’t turn them off.

Most users fall into these categories.

- Don’t care, accept and visit the site
- Care about their privacy or don’t trust the site, reject
- Care a lot, and want to tailor specific trackers

Given these three obvious use cases, it is remarkable that most cookie popups only offer the following:

- Accept it all
- Turn it all off manually

Allow Users to Accept or Reject Cookies with One Click

It's a dark pattern (see #101, *Don't Join the Dark Side*) that's everywhere, and it's ironic that a set of directives designed to give people better control over their privacy has done the opposite: we're nagged into blindly accepting the defaults the site wants us to use.

The obvious user-friendly solution for this is to do the following:

- Don't use marketing/analytics cookies or,
- If you absolutely *must*, then give your visitors two simple options:
 - a. Accept analytics cookies
 - b. Reject analytics cookies

It's that simple. Here are a couple of examples of large sites that do this well.

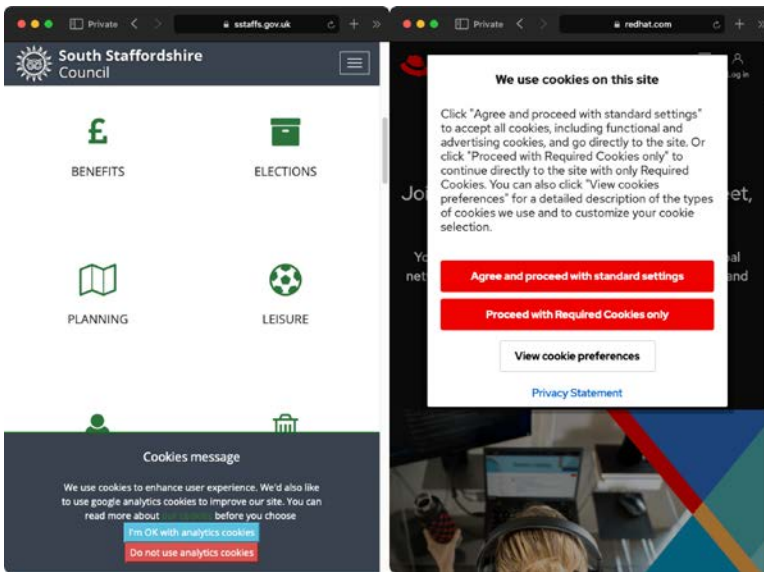


Figure 25.1: Both sites show clear options for users, respecting their preferences, and not delaying entry to the content by forcing the user to configure too much

Learning points

- Give users a clear way to accept or reject optional tracking cookies
- Don't force users to manually configure trackers one by one
- Don't delay access to the content of your site unless you absolutely must

#26

**Help Users Understand
Their Next Steps from
“Empty States”**

An empty state is a view that would normally show a lot of information to a user—a list of projects, albums, tasks, and so on—but because the user is new, they haven't yet created anything.

The default behavior of many apps is to simply show an empty view where the content would be. For a new user, this is a pretty poor experience and a massive missed opportunity for you to give them some extra orientation and guidance.

An empty state should usually show some helpful text, hints, and maybe a friendly graphic or icon. Now, because these views can appear on a per-feature basis, it's easy to be very task-oriented in the advice you give. If the user views the to-do list, you can advise on making the first to-do item.

On a profile, you can guide the user to include a bio or add an avatar picture.

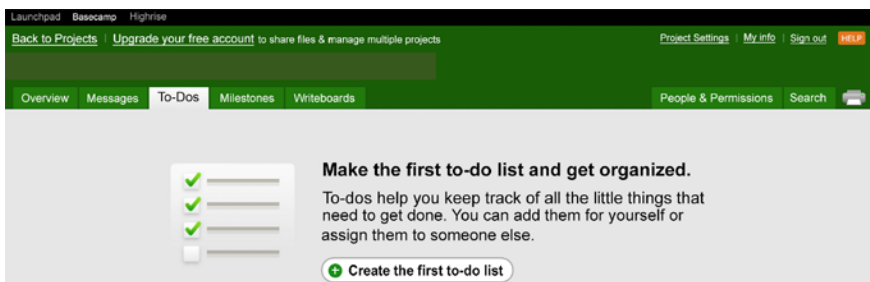


Figure 26.1: The Basecamp to-do list before the user has created any items

Help Users Understand Their Next Steps from “Empty States”

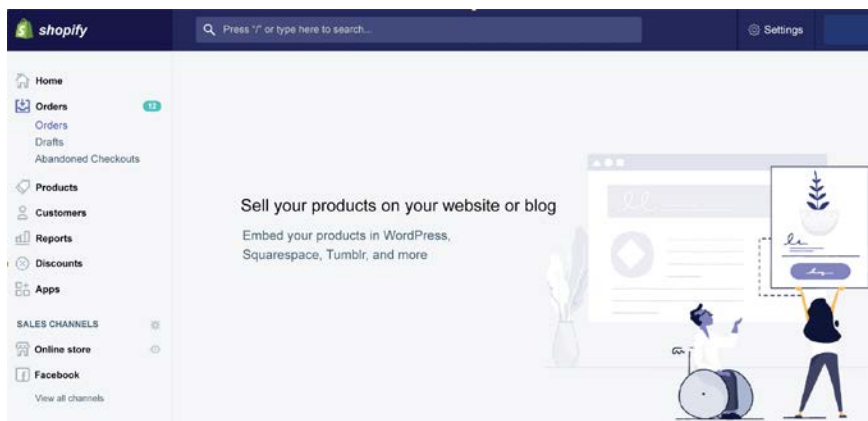


Figure 26.2: Shopify welcomes new users with a recap of what they can do

The empty state is only shown once (before the user has generated any content), so it's an ideal way of orienting people to the functions of your product while getting out of the way of more established users who will hopefully “know the ropes” a little better. For that reason, it should be considered mandatory for UX designers to offer users a useful empty state.

What's more, offering clear calls to action (CTAs) in these empty states can be the difference between a user never engaging with your product, or a successful onboarding experience, and should be considered vital.

Learning points

- Use empty states to orient new users
- Be task-oriented in the advice you give to users
- Be specific in your advice if you offer empty states on a per-feature basis

#27

**Make “Getting Started”
Tips Easily Dismissable**

An empty state (refer to #26, *Help Users Understand Their Next Steps from “Empty States”*) won’t show once the user adds some content or performs an initial task, which is ideal.

Too often, apps force users to view their “getting started” guide or “tips for beginners.” They are often good for new users, but if you’re coming back to an app you’ve used before then they’re incredibly frustrating.

An extra level of rage is induced when an app update “resets” these tips and existing users are forced to sit through the tutorial all over again just to use the app. To avoid this, tell users what will be covered in the tutorial and make tips optional and dismissible. You’ll get bonus points for letting users exit the entire “onboarding wizard” with one tap—at any point in the journey or at the very start:



Figure 27.1: This dialog tells the user what’s covered in the tutorial so they can decide to skip it if they wish

However, beware of overdoing these tips: if you’ve followed conventions (and basically, the rest of this book), there shouldn’t be a need to explain every little detail of your app’s UI.

If you have designed and delivered a UI where you must explain “You can search for things here,” and “Your past entries appear here,” and “Click here to create a new entry,” then your UI is too complicated and needs to change.

Larry Tesler was a computer scientist and human-centered designer who coined “Tesler’s Law” in the 1980s. It states that every application has an inherent amount of complexity that cannot be removed or hidden. Instead, this complexity must be dealt with: either by the system or by the user.

Onboarding tutorials do have a place, therefore, and are best used to explain general product concepts to new users—to help them unravel this complexity—rather than to explain individual UI elements or controls.

Learning points

- Explain to users what’s covered in the onboarding tutorial
- Allow the whole tutorial to be skipped in one action
- Allow users to return to onboarding tips or help easily

#28

**When a User Refreshes
a Feed, Move Them to
the Last Unread Item**

Typically, a feed (or any list of items) will have links *on each item* to view them or perform actions on them. This means that users may well be navigating back and forth to these lists.

Imagine a list of news items; it's likely that a user would read the list, then choose one or more news items to read, each time navigating back to the list view. Don't simply reload the feed and put the user back to the start again, you monster!

As an example, Twitter shows the user how many tweets behind they are, allowing them to manually reload if they wish, but not altering the feed without their explicit action:

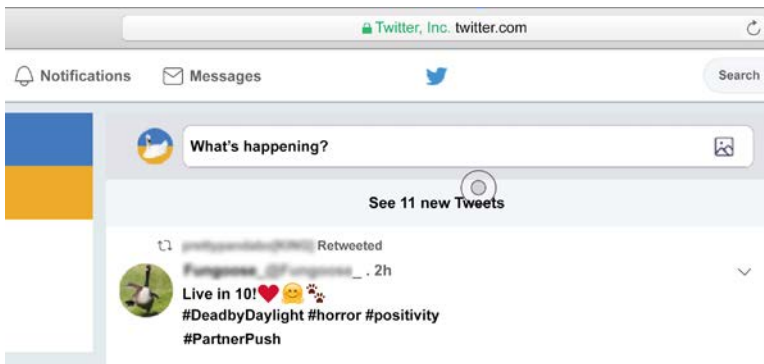


Figure 28.1: Twitter getting something right for a change

Of course, *technically*, the feed may well have changed in the time it took the user to read the story, but if it keeps updating, it's disorienting and difficult to use. Yes, this means additionally keeping track of where your user's scroll position is, but it's worth it for the usability benefit.

Learning points

- Return users to the same place that they came from
- Don't reload or refresh feeds while a user is using them
- Give the user an option to manually refresh the feed while they're using it

Navigation

Navigation is important in UX design because it allows users to move through an interface and interact with content. Good navigation should be easy to use and understand, and it should help users find the information they need quickly and easily. Navigation can be created using various elements, such as menus, breadcrumbs, and buttons.

#29

**Don't Hide Items Away
in a "Hamburger"
Menu**

Few UI patterns can be as controversial as the hamburger menu. Over the past five years, it's become the de facto way of offering a menu on small displays, typically as a website scales into mobile or tablet width using responsive design:

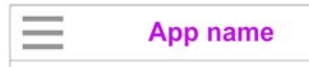


Figure 29.1: The dreaded hamburger

Research shows (from *Hamburger Menus and Hidden Navigation Hurt UX Metrics*, <https://www.nngroup.com/articles/hamburger-menus/>) that hamburger menus:

- Slow down discovery time for users
- Increase perceived task difficulty
- Increase the time it takes to complete a task

Simply put, the hamburger menu hides items away from users and makes them less discoverable. Additionally, because the menu is hidden, users can't gain a sense of "where they are" in the product.

It's worth noting that since the research linked above was undertaken, users have been exposed to the hamburger in thousands of apps and websites and will be experiencing less friction today as they will have learned the pattern. The reduction in discoverability however is still a very real issue.

Some alternative design patterns to the hamburger menu:

- **Navigation on the bottom of the view:** Made popular by iOS apps, you can get four or five key features into an ever-present bottom menu and maybe make the fifth item "fly out" containing advanced tools.

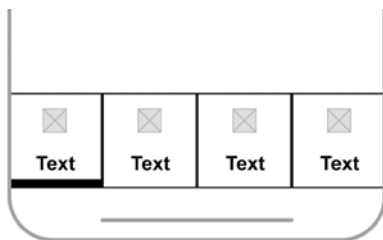


Figure 29.2: Navigation at the bottom of a mobile view

- **Tabbed navigation:** Inverting the above, and popularized by Android apps, items can live at the top of the view.

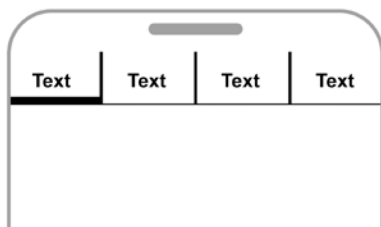


Figure 29.3: Tabbed navigation at the top of a mobile view

- **Breadcrumbs:** The good old tried-and-tested breadcrumb navigation (see #77, *Use Breadcrumb Navigation*) is a simple list of underlined links. It is well understood by users and combines easy navigation with showing the current page.

In some circumstances, for example, if your app has a lot of features that need to be “possible” (see #99, *Decide Whether an Interaction Should Be Obvious, Easy, or Possible*), the usability trade-off seems worth making, in order to offer these features on mobile rather than removing them, but never use a hamburger menu on the desktop.

If you **must** use a hamburger menu, then label it **Menu** and spare the user the much-maligned and ambiguous “three lines” icon.

Learning points

- The hamburger menu slows down discovery for users
- By hiding menu controls in this way, users can't get a sense of their location
- Consider alternatives to the hamburger menu, but if you must use one, label it

#30

**Make Your Links Look
like Links**

Links, or *hyperlinks*, are the basis of the web and were one of the key advances when Sir Tim Berners-Lee invented HTML in 1989. In the original browsers, clickable links were blue, italic, and underlined. They looked gaudy and out of place, but that was the point: it was a brand-new concept and users needed a way of telling a link apart from the rest of the text on the page.

Fast-forward to the present day and the practice of styling links has largely been abandoned in favor of only highlighting them when they're hovered over or, worse, adding no visual affordances to them whatsoever.

The style-on-hover approach is less than ideal: users on touchscreen devices have no hover state. Meanwhile, users with a mouse end up “hunting” for links by hovering over parts of text bit by bit, hoping to find a link, or just never finding them at all.

Make Your Links Look like Links

☐ Yes ☒ No

24 Please confirm whether, within the past 3 years, your organisation or any of your partner organisations has:

- been guilty of serious misrepresentation of the information required for the fulfilment of the selection criteria

☐ Yes ☒ No

25 Please confirm whether, within the past 3 years, your organisation or any of your partner organisations has:

- withheld such information or failed to submit supporting documents required under [Regulation 59 of the Public Contracts Regulations 2015](#) [\(link opens in a new tab\)](#)

☐ Yes ☐ No

26 Please confirm whether, within the past 3 years, your organisation or any of your partner organisations has:

Figure 30.1: A gov.uk page with clear links and nice controls

When adding visual affordances to links, don't forget to include the state of the link (active, followed, and so on), and ensure that disabled links (and other controls) are sufficiently different from your link styling.

One of the earliest conventions of the web: styling visited links a different color, is still vital—it helps users work out where they've already been and avoids frustrating wasted clicks. (See #75, *Each Aspect of a User's Journey Should Have a Beginning and End*).

Asking the user to click things just to work out what they do (or whether they do anything at all) is insane. This kind of design decision is a classic example of form over function. If you're making your users guess what links do, because you think that "minimalism" means adding so few affordances that controls are impossible to use, then you're wrong. I don't care what the marketing guys say: make your links underlined.

Learning points

- Make links look like links with visual affordances
- Don't make non-links look like links
- Don't make your users hunt for clickable controls

#31

**Split Menu Items Down
Into Subsections, so
Users Don't Have to
Remember Large Lists**

Humans are better at some things than others. We're really good, for example, at drawing a pretty picture of a flower, but we're not so good at instantly recalling the precise genus of that flower and its scientific name. Computers are better at that kind of thing.

The rule of thumb for the number of items that a person can reasonably remember and juggle in a list is “seven, plus or minus two.” (Stipulated in George A. Miller's *The Magical Number Seven, Plus or Minus Two*, first published in 1956 in *Psychological Review*.) This research has been around since the 1950s and has sadly been misunderstood and misinterpreted over the years.

The rule doesn't mean that people *can't remember* more than 7 or so items; it refers to the limit of short-term memory processing. An expert in a complex computer system will regularly be able to juggle many more items in their heads—but a first-time user, approaching a list of choices for the first time, would benefit from there being “about 7” items in the list as a maximum.

The “magic number seven” will change depending on the items being recalled, the context, and environmental factors like state of mind, but it's as good a starting point as any. The point is: users can't manipulate and recall long lists of items in their minds.

If you're presenting a user with a list of options, keep in mind that by the time they've read the seventh or eighth option, they will likely have “filled the buffer” in their mind to capacity, and will struggle to remember what the first option was.

Split Menu Items Down Into Subsections, so Users Don't Have to Remember Large Lists

Collectables	Garden
Antiques	Appliances
Sports memorabilia	DIY materials
Coins	Furniture & homeware
Mobile phones	Cycling
Sound & vision	Fishing
Video games	Fitness, running & yoga
Computers & tablets	Golf
Women's clothing	Radio controlled toys
Men's clothing	Construction toys
Shoes	Outdoor toys
Kid's fashion	Action figures
Sneakers	Books, comics & magazines
Luxury Watches	Health & beauty
Costume jewellery	Musical instruments
Vintage & antique jewellery	Business, office & industrial
Fine jewellery	

Figure 31.1: A long list of items that's hard to mentally juggle

Collectables & antiques	Fashion	Home & garden	Toys & games
Collectables	Women's clothing	Garden	Radio controlled
Antiques	Men's clothing	Appliances	Construction toys
Sports memorabilia	Shoes	DIY materials	Outdoor toys
Coins	Kid's fashion	Furniture & homeware	Action figures
	Sneakers		
Electronics	Jewellery & watches	Sporting goods	Other categories
Mobile phones	Luxury Watches	Cycling	Books, comics & magazines
Sound & vision	Costume jewellery	Fishing	Health & beauty
Video games	Vintage & antique jewellery	Fitness, running & yoga	Musical instruments
Computers & tablets	Fine jewellery	Golf	Business, office & industrial

Figure 31.2: The same content, but split up into easy-to-parse, shorter sections

Navigation menus, options in a drop-down menu, section headings, and category lists—all over your UI you can optimize these by shortening them, consolidating them, and dividing them into shorter logical groups.

Try to group menus into sections, or reduce the complexity of options, so that the user doesn't have to struggle to recall them. Hide extra settings in “advanced” settings, for example. Your users are (probably) humans, not robots.

Learning points

- Users can read, manipulate, and recall *roughly* seven items in a list in their short-term memory
- After more than seven or so items, the user will struggle to use the list
- Group related items into sections

#32

**Categorize Settings
in an Accessible Way**

There's no need to include every possible menu option in your menu when you can hide advanced settings away. Group settings together, but separate out the more obscure into their own section of "power user" settings, which should also be grouped into sections if there are a lot of them (don't just throw all the advanced items in at random).

The assumptions you make about *which* features are advanced should be backed up by user research; techniques like **card sorting** or **tree testing** with real users can expose the right set of advanced settings.

Not only does hiding advanced settings have the effect of reducing the number of items for a user to mentally juggle (refer to #31, *Split Menu Items Down into Subsections, so Users Don't Have to Remember Large Lists*), it also makes the app appear less daunting, by hiding complex settings from most users.

By picking good defaults (refer to #96, *Pick Good Defaults*), you can ensure that the vast majority of users will never need to alter advanced settings. For the ones that do, an advanced menu section is a well-used pattern.

Once you've worked out the right set of advanced settings, it's time to structure them into a usable set.

Categorize Settings in an Accessible Way

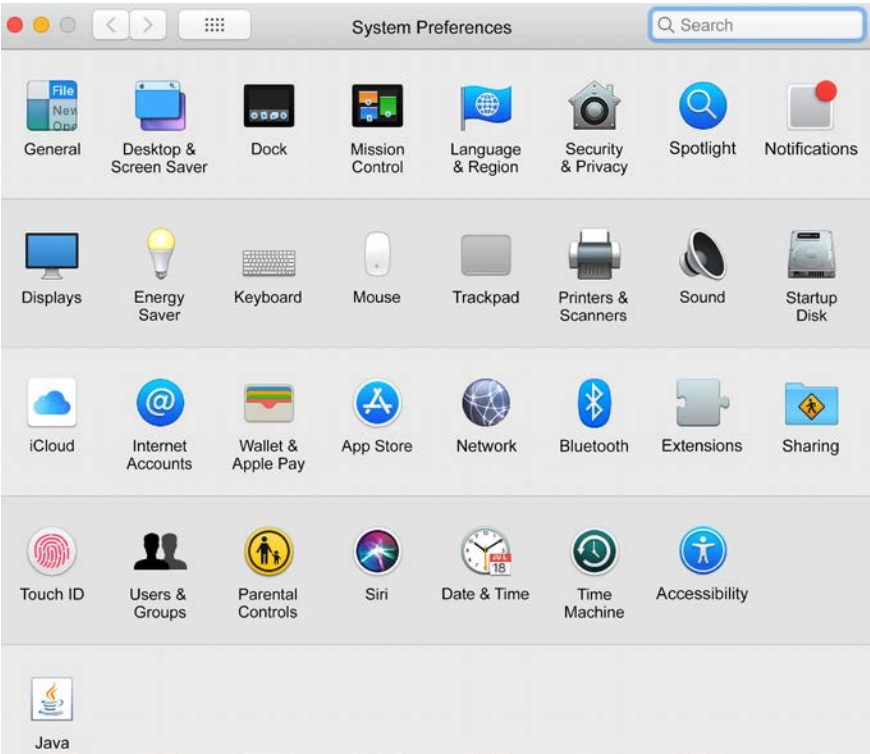


Figure 32.1: The macOS system preferences panel is well categorized



“Jobs to be done” is an innovation methodology devised by Anthony W. Ulwick in the early 2000s as part of the “Outcome-Driven Innovation” product development framework. It asks the designer to focus on the functional, emotional, and social outcomes the user wishes to achieve—their “jobs to be done”—and then meeting those outcomes with product features and benefits.

Settings pages should be structured based on “jobs to be done,” not necessarily on system function. For example, all the settings for “sound” are in one place and “video” in another. This seems obvious and many operating systems get this right, but many software products don’t, instead throwing all the settings into one long settings menu that is too dense and long to work with.

The macOS system preferences panel (in *Figure 32.1*) does this well by sorting items by conceptual area, rather than system function. **Keyboard**, **Mouse**, and **Trackpad** all have their own views, instead of calling them “Input” and lumping them together into one confusing view.

Categorize Settings in an Accessible Way

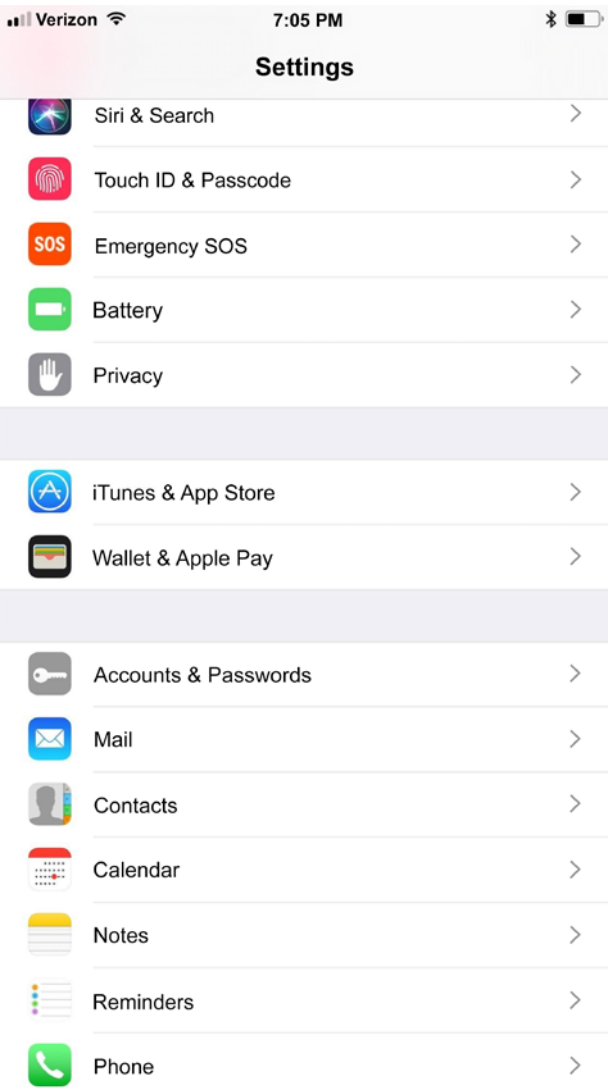


Figure 32.2: Although there are a lot of items, iOS groups them into sections

You get bonus points for putting a “search” field on particularly long or complex settings views.

Learning points

- Hide advanced settings behind another level of navigation
- Group items together by jobs to be done or conceptual areas
- Remember the “seven plus or minus two” rule for long lists of items

#33

**Repeat Menu Items in
the Footer or Lower
Down in the View**

Your site’s navigation is at the top of the view, but the user has scrolled right down the view—no doubt captivated by the wonderful, engaging content you’ve provided—so how do they return to the top of the page?

Most mobile browsers have a shortcut where tapping the top bar of the app will scroll the page up. There’s no need to provide a “back to top” link that floats down the page: it’s a waste of space.

A great solution is to repeat main menu items in the footer of the page or, at the very least, add some shortcuts to popular parts of the site. Including a “mini breadcrumb” (a shrunk-down version of the typical breadcrumb control) is more useful than a “back to top” link, as the user can hop back up a level to find the next item.

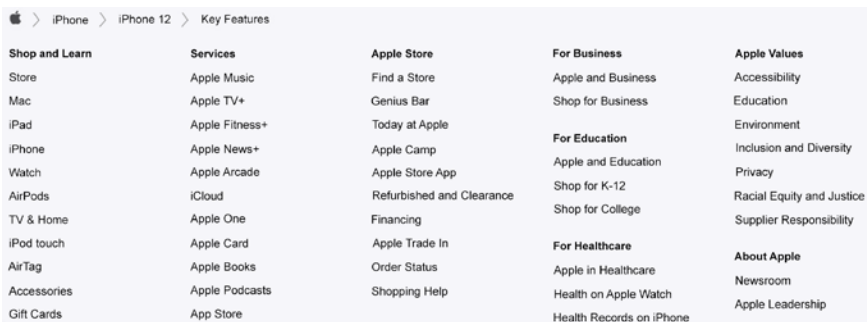


Figure 33.1: Apple.com features a “mini breadcrumb” in the footer to great effect

Repeat Menu Items in the Footer or Lower Down in the View

Bonus: persistent links in the footer are good for helping search engines index your site and can help with SEO in some situations.

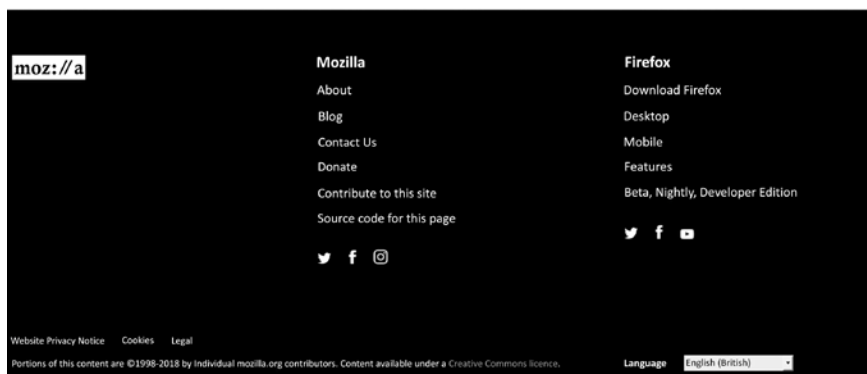
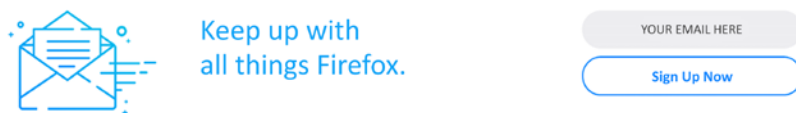


Figure 33.2: Mozilla's footer

Mozilla's footer (above) strikes a nice balance between not being overly cluttered and providing useful links to the top-level sections of its navigation hierarchy. Some sites decide to include a search control in the footer, which can be a smart idea: the user may not have found what they were looking for on the page, so this gives them a way to search the site.

Learning points

- Repeat navigation items in the footer
- Don't make footers a "dead end"
- Consider offering a search function in the footer if it makes sense to do so



Iconography

Icons are visual shorthand—a way of encoding lots of information and meaning into a simple, easy-to-remember picture. Too many icons can become overwhelming but when used correctly, they can be a powerful tool for simplifying complex user interfaces.

#34

**Use Consistent Icons
Across the Product**

What is an icon for anyway? An icon is a visual shorthand—a way for the user to glance at the screen, and in a few milliseconds, understand the purpose of a button. The user will not learn their way around a UI from icons alone, but once learned they can speed up recognition and recall massively.

A UI packed with seemingly random, disparate icons is a UX disaster. I know what happened: you started using an icon set because it looked cool, then you realized it didn't have an icon for “upload” or “download.” The UI review meeting is later today and you'd better get an icon in there quick! So, you use  and  instead. But they look different to the rest of the app and users must spend those extra few seconds working out that, yes, they are actually part of the UI too.

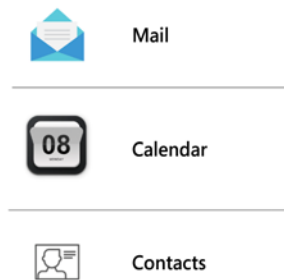


Figure 34.1: Inconsistent icon styles looks unprofessional and is confusing for the user to recognize and recall

Don't be lazy when it comes to icons: pick a metaphor and stick with it. This may mean extra illustration effort to produce new icon elements that are in keeping with the icon style, but that effort will pay off in increased usability for the end user.

Learning points

- Use a consistent icon style across the product
- Don't take a shortcut by including disparate icons
- Take the extra time to build a coherent icon style

#35

**Don't Use Obsolete
Icons**

For about 20 years, the “floppy diskette” icon has meant “save” and this connection persists in UIs across desktop and web apps. It was a great visual metaphor for a long time, but things have changed and many users under the age of 20 will have never laid eyes on a floppy disk—if they know it at all, it’ll be from seeing the icon, not the real object.

Other examples include old telephones with handsets, curly cords, and rotary dials; radio microphones from the 1950s; and reel-to-reel tape recorder icons to mean “voicemail.”

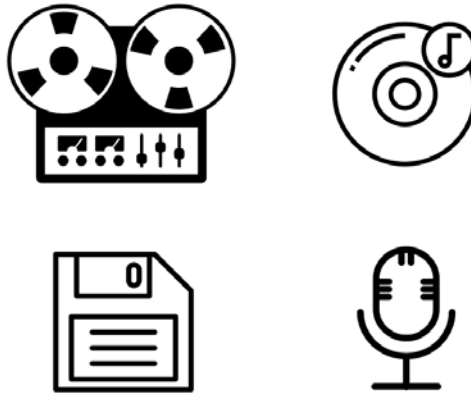


Figure 35.1: Ten years from now, nobody will know what any of these are

Try to think about how the visual metaphors you use will work for different age groups, cultures, and languages. Searching for the right visual metaphor for an icon is often challenging work but rewarding, and your users will benefit from increased familiarity with your product.

Increasingly, apps will auto-save your progress as you go, but even so, we're in need of a new standardized icon for "saving" (which these days means sending your data to a web-based service), rather than a removable or hard disk.



*Figure 35.2: "Upload to the cloud," perhaps? Icon by The Noun Project/
Jeevan Kumar*

Icons are simple, so they will always have a degree of ambiguity, but they should always be shown with a text label to reduce this (see #38, *Always Give Icons a Text Label*). The icon should serve primarily as a visual cue or shorthand, as well as a tappable (or clickable) target.

Like most things in design, icon selection benefits from testing with real users (refer to #6, *Test with Real Users*). Ask users what they think a proposed icon means to them and see if they can recall your icon later.

Learning points

- Don't use icons that depict obsolete technologies or visual metaphors
- Always show icons with a text label to reduce ambiguity
- Test your icons with real users

#36

**Don't Try to Depict
a New Idea with an
Existing Icon**

Occasionally, you will need to invent a whole new icon. If the concept you're trying to describe is novel, then your users will need an icon that doesn't confuse them by referencing another idea. It needs to be new, yet recognizable, and mappable to a real-world example. If this sounds difficult to you, that's because it is.

Thankfully, the need to create an entirely new icon is rare because most of the concepts in your app will be better served by existing UX patterns and UI conventions, but there may be a case where you have a new concept.

The middle ground here (using an existing icon to depict a new concept) is the worst of all worlds: it's confusing for users who have seen the icon before in other products, with a different meaning behind it.



Figure 36.1: Spare a thought for these poor folks...

Some of the most misused icons in this category are:

- The WiFi “fan” icon
- The generic “cloud” icon
- The globe icon
- The chain “link” icon

These icons can be seen frequently depicting ideas as diverse as “upload,” “save,” “share,” “email,” and more. I’ve seen the WiFi icon being used to depict “pay with your contactless card.” It’s jarring and utterly confusing.

Don't Try to Depict a New Idea with an Existing Icon

It's understandable that sometimes mistakes are made, but too often this is simply laziness: it can be hard to find the right icon and even harder to create a new one.

There are many large searchable directories of icons online (some of them are royalty-free, like my favorite at the moment, called The Noun Project (<https://thenounproject.com/>)) and it's always worth a quick search on such sites to see what other designers have used to depict concepts.

This is another case where copying the patterns of others (many of these icons are available for reuse without a fee) is great for the user: they will be familiar with the patterns from other applications and uses, and you can save them learning and cognitive time by reusing these icons.

Learning points

- There is probably an icon out there that suits your needs already
- Don't use an existing icon for a new concept
- Check whether there are open source or public domain icons already

#37

Never Use Text on Icons

Icons are *supposed* to be simple pictures that depict a concept. They are a short-hand visual reminder for users that what they’re about to click is the thing they want.

Often, icon designers get frustrated that their pictogram isn’t *quite right*, as it isn’t quite recognizable or distinct enough. Instead of solving the problems with the picture, they opt to simply add a line of text *into the icon design*. Note that I’m not talking about text labels for icons—those are essential. I’m referring to the lazy practice of including text *within* the icon itself.



Figure 37.1: Three icons with text as part of the icon

First off, this is low-effort design, but secondly, and more importantly, it breaks translation and accessibility functionality. Whether your product is a website, which can be translated with an online service like Google Translate, or a self-contained app, which will be internationalized with “strings” of copy, the text in icons won’t get translated and users will get confused.

Users with accessibility needs who are using screen-reading software will run into problems too: the software won’t be able to “speak” text that’s included on the icon. Spend that extra bit of time and effort building (or sourcing) icons that convey their meaning without text—and always add descriptive “alternative” text for screen readers.

Learning points

- Don’t include text inside icons—icons should make sense without text
- Text within an icon can’t be translated or read with assistive technologies
- Include text *labels* with icons but not text *in the icon itself*

#38

**Always Give Icons a
Text Label**

Now, I *don't* mean text on the icon (see #37, *Never Use Text on Icons*)—I mean a text label near the icon, not just an icon on a button on its own.

Small, nondescript buttons, with obscure mystery icons on them, are next to useless and consistently perform terribly in user tests.

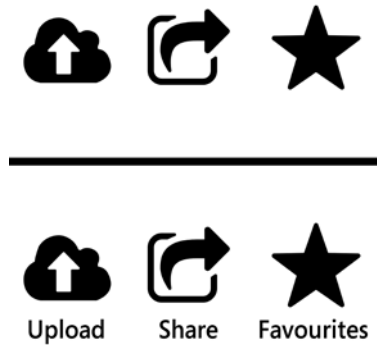


Figure 38.1: Which is easier to understand?

Let's go back to the original purpose of an icon—to provide a quick visual shorthand by which the user can instantly recognize a control, and to provide a target for the user to click or tap. The icon isn't meant to describe a button the first time that the user sees it—the user will need a text label for that. However, if the icon is distinct and recognizable, then the user will locate the control and recall its purpose more quickly with it.

Our old favorites, the Nielsen Norman Group, have a great shortcut for this, the “5-second rule”:



If it takes you more than 5 seconds to think of an appropriate icon for something, it is unlikely that an icon can effectively communicate that meaning.

Icons are used and misused so relentlessly, across so many products, that you can't rely on any single icon to convey a definitive meaning. For example, if you're offering a "history" feature—there's a wide range of pictogram clocks, arrows, clocks within arrows, hourglasses, and parchment scrolls to choose from—the user needs a text label to understand what *this* icon means in *this* context within your product.

Often, a designer will decide to sacrifice the icon label on mobile-responsive views. Don't do this. Mobile users still need the label for context. The icon and the label will then work in tandem to provide context and instruction, and offer recall to the user, whether they're new to your product or use it every day.

There are exceptions to this—frequently used controls (like bold, italic, underline, and so on) can be recognized without a text label—but icons in a main menu or toolbar really need descriptive text next to them. It's also possible that in frequently used "expert" software (for example, the internal control panel at an insurance company where a phone operative spends all day, every day) the user will be sufficiently familiar with or will have had sufficient training to recognize the tools. These kinds of expert users don't need the icon label; they're already very familiar with it. This hardly ever applies to most consumer or B2B software, however—users just don't spend enough time in your app to learn its icons.

Learning points

- Always show text labels with icons
- Don't hide or obscure labels on mobile versions
- Icons without labels are a major source of frustration for users

Input

How do you get your users' intentions into your system? In this section we'll look at simple input fields, through special cases like passwords and even chatbot interfaces—with shortcuts to make your users' lives easier.

#39

**Use Device-Native
Input Features Where
Possible**

If you're using a smartphone or tablet to dial a telephone number, the device's built-in "Phone" app will have a large numeric keypad, which won't force you to use a fiddly QWERTY keyboard for numeric entry.

Sadly, too often, we ask users to use the wrong input features in our products. By leveraging what's already there, we can turn painful form entry experiences into effortless interactions.

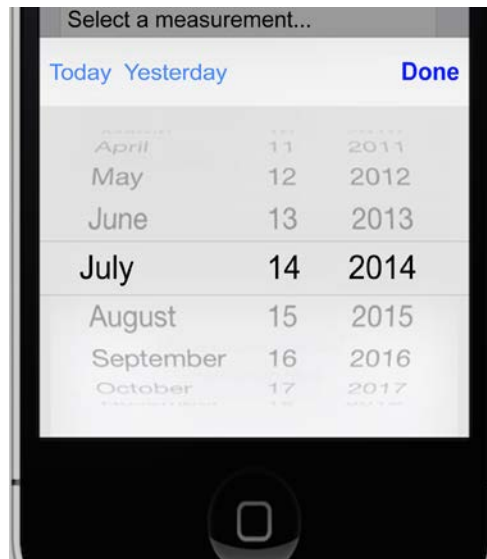


Figure 39.1: The iOS “picker” control replaces fiddly drop-down menus

Dropdowns should let users use the device's full-width picker control and numeric entry should show a numeric keypad. For example, you can display a numeric keypad in web forms by adding the `type=tel` attribute to the input field in HTML:

```
<label for="phone">Your telephone number:</label>
<input type="tel" id="phone" name="phone">
```

This will show the telephone keypad in both iOS and Android browsers:

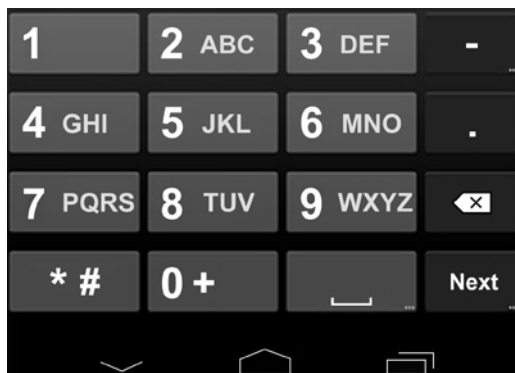


Figure 39.2: The telephone keypad in Android

If you're asking users to enter an amount of money to send, why not give users a numeric keypad? Monzo Bank forces users to increment the amount they want to send, up and down £1 at a time. Bad luck if you want to send someone £1,000.

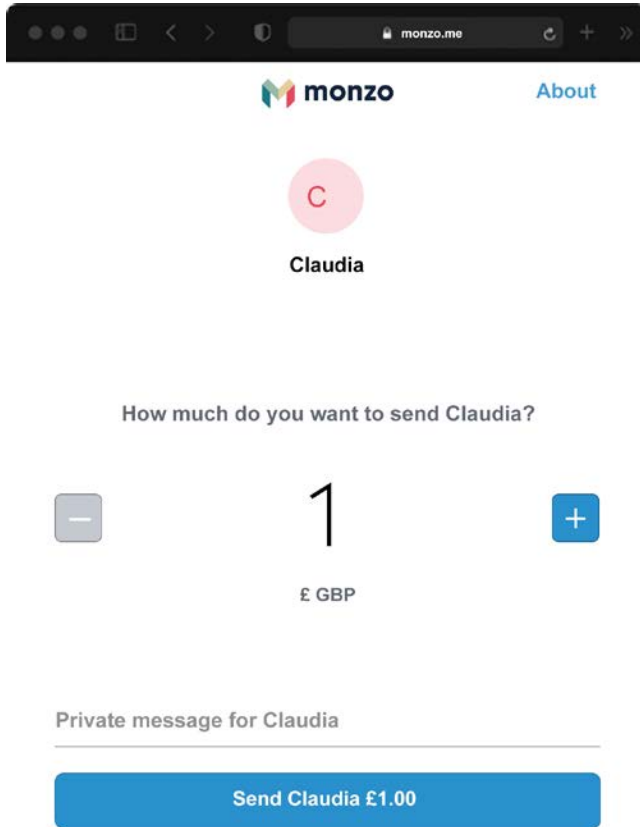


Figure 39.3: The anti-pattern—just keep pressing “+”

No matter how good you are, you can't justify spending the time and money that these companies have spent on making usable system controls. Even *if* you get it right, it's still yet another UI for your user to learn, when there's a perfectly good one already built into their device. Use that one.

Of course, there are some rare exceptions to every rule, and sometimes date picker controls need a bit of extra functionality. We'll explore some of these in #46, *Use the Same Date Picker Controls Consistently*.

Learning points

- Shortcut your way to success by using a UI already built for you
- Using device-native input controls means that users have one less thing to learn
- This isn't just for mobile users: desktop software should also use the right controls for those input methods

#40

**Streamline Creating and
Entering Passwords**

It still makes sense to obfuscate passwords as they're being entered, but let's be real, shoulder-surfing isn't possible when you're signing in to an app on your couch.

Providing a “show password” toggle is not only great for usability but also improves security: users can enter longer, more complex pass-phrases and be confident that they can retype them correctly. Default to obfuscating the password, but provide a checkbox or toggle that allows the user to see their password.

Yes, I know we should all be using a password manager (a plugin that generates and stores all your site passwords for you), but the fact remains that most regular users don't.

Show the password strength rules. Don't make users try and try again to enter passwords, only to be told later that they need to have a certain obscure combination of letters, numbers, and symbols. Show the user the rules the whole time that the password field is visible.

One quick word on password managers. Most password managers overlay a control in the browser—directly into the password field—to allow you to “tap to fill.” You should be aware of this and accommodate it in your UI.

Sign In
Use your Google Account

Email or phone 🔍

[Forgot email?](#)

Not your computer? Use a Private Window to sign in

Figure 40.1: The LastPass control is usable and shown in the field, but this can sometimes be obscured and unusable by custom input styles

The preceding screenshot should also serve as a reminder to see #38, *Always Give Icons a Text Label*—the icon is not very discoverable.

Finally, there's no need to force users to enter a password twice, just to check that they got the password correct. It slows users down, creates an unnecessary “test” for them to pass, and serves little purpose: if they did misspell their password, they can simply do a password reset later down the line.

Learning points

- Obfuscate passwords but let the user toggle them to visible when creating and signing back in
- Show the user any rules that they need to follow when creating a password
- Don't ask the user to retype the password when setting it

#41

**Always Allow the User
to Paste into Password
Fields**

It's difficult to fathom where this pattern of disabling paste came from or what possible security issue it's supposed to address. Back in the 1990s, "webmasters" would disable right-click to prevent users from copying images. This worked for about 5 seconds, until people realized that they could just screen capture the image. Using some JavaScript on the page to *prevent* users from pasting into a password field is *insane* and potentially harmful for security.



Figure 41.1: Screenshot from security expert Troy Hunt's blog post, "The 'Cobra Effect': <https://www.troyhunt.com/the-cobra-effect-that-is-disabling/>

A user with a password manager app will have a long, impossible-to-remember password that has to be pasted into the field (especially on mobile, where it's more tricky to autofill the field). However, disabling paste in a password field forces users to only use weak, easy-to-remember passwords. If you've disabled paste on a password field, then you need to have your laptop taken away from you.

It's a good general rule across the board to not interfere with standard system behaviors (copy, paste, find, zoom, right-click, and so on), as they are all basic interactions that the user will have grown accustomed to over years of working with various devices. To deliberately disable these behaviors on your product is nonsensical, yet it still happens. Designers think that they can improve security, reduce plagiarism, or other factors that aren't user-centric.

Learning points

- Don't disable paste on password fields
- Don't interfere with any basic system interactions like copy, paste, find, and right-click
- Allow users to use password managers with your product

#42


**Don't Attempt
to Validate Email
Addresses**

Your user is entering an email address and you're thinking about writing some code to validate it (check that it's in a sensible format and they haven't entered gibberish or mistyped it). Think again.

It used to be so simple to validate email addresses on the client side. A little bit of JavaScript was all it took to check that the domain was in the format:

```
user@domain.tld
```

If it didn't match this pattern, it wasn't a valid email and the user couldn't sign up. We used to only have a handful of **top-level domains (TLDs)**. Now, we have over 1,000 TLDs, with more being added all the time:

```
stealthy+user@example.ninja  
stealthy+user@example.ninja  
holidays@.ws  
email@www.co  
website@email.website
```

The above addresses are all valid domains (please don't email these people, in case they are real!), but the TLD list changes all the time, so good luck writing the JavaScript to validate them—there are too many edge cases.

The side effect of this is that any errors will prevent legitimate users from signing up or using your product, leading to device-smashing levels of frustration and lost signups for your product.

Simply make your input field an “email” input (in HTML: `<input type="email">`) and let the browser and device figure the rest out (some will autofill or suggest the user's email address). You still may want to verify these addresses on the server side by sending a one-click link in an email to verify.

Learning points

- Don't validate emails on the client side
- Tell the browser or device that you're collecting an email address
- Verify emails on the server side with a one-click verification link

#43

**Respect Users' Time
and Effort in Your
Forms**

Your long-suffering user has painstakingly entered field after field of data into your form, often on a tiny mobile screen with an on-screen keyboard.

Don't clear this data unless the user specifically abandons the flow (maybe by hitting cancel). If clicking something is going to reload the page, and this might result in a blank form, then make sure you save the user-entered data first.

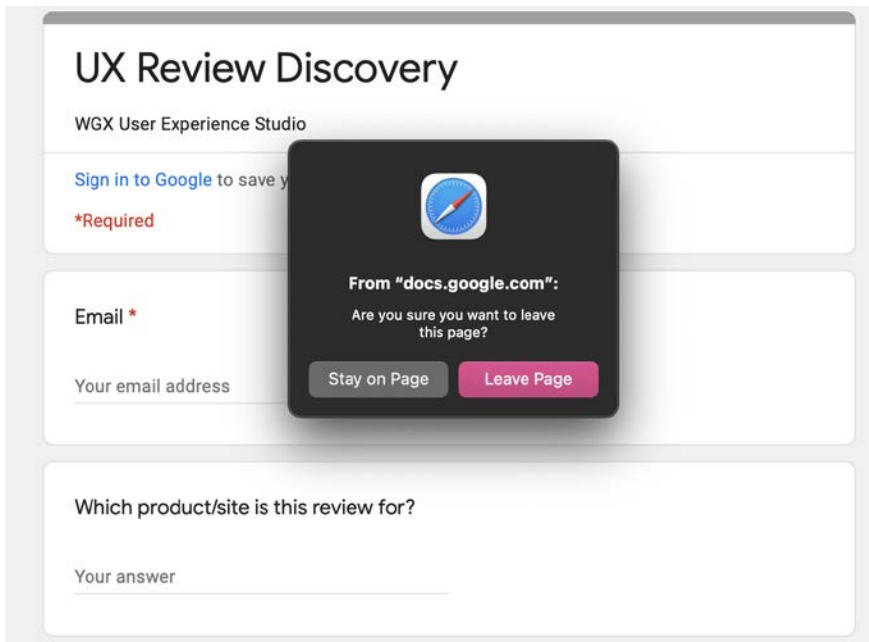
The image shows a web form titled "UX Review Discovery" from "WGX User Experience Studio". The form includes a "Sign in to Google" link, a red asterisk indicating required fields, an "Email" field with an asterisk, and a question "Which product/site is this review for?". A dark browser warning dialog is overlaid on the form, asking "From 'docs.google.com': Are you sure you want to leave this page?" with "Stay on Page" and "Leave Page" buttons. The form fields are currently empty.

Figure 43.1: Some sites and browser combinations will warn you before leaving or reloading to prevent accidental loss of user-entered data

This is a great example of where technical reality runs against UX goals. On the one hand, if a browser could speak, it would likely argue that reloading a form *should* clear it, as we're telling the browser to fetch the empty form again.

However, we are not robots, we are humans, and so much of good UX design is about empathy and respect. This includes respecting the user's time and effort and demonstrating empathy for what they're trying to achieve. Reloading a form with all the user's data removed is one of the most careless UX failures and nothing makes people angrier.

Learning points

- Don't ever clear user-entered data without explicit permission—or adequate warning of what will happen
- Urge your technical teams to build features that prevent loss of user effort
- Put yourself in their shoes: would you want to type all this stuff in again?

#44

**Pick a Sensible Size for
Multiline Input Fields**

Forms need to be as frictionless as possible, because they are a huge barrier for users to painstakingly navigate. Conversion rates are generally low, so make forms as easy for the user to complete as possible.

Sometimes, we need to ask users for more than a simple one- or two-word answer (like a name) and a multiline input field (or “text area”) is needed. A common mistake on the web (and in some desktop apps) is to provide a text area that is way too big or way too small.

If the text area is way too big, and the user has to manipulate the viewport to see what they’re typing, then you’re wasting valuable screen space.

A screenshot of a web form. At the top, there is a label "Your pet's name" in a light gray font. Below the label is a large, empty text area with a thin gray border. The text area is significantly larger than the text "Mr Wiggles" that is entered at the top. Below the text area is a blue button with the word "Send" in white text.

Figure 44.1: A text area that is way too big for the intended input

If the text area is way too small, then the user has to scroll around inside the field to see what they’ve written.

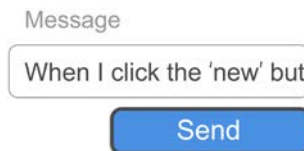
A screenshot of a web form. At the top, there is a label "Message" in a light gray font. Below the label is a small text area with a thin gray border. The text area contains the text "When I click the 'new' but". Below the text area is a blue button with the word "Send" in white text.

Figure 44.2: An impossibly small input field for lots of text

Pick a Sensible Size for Multiline Input Fields

HTML makes it very easy to specify a single-line input field:

```
<label for="fname">First name:</label>  
<input type="text" id="fname" name="fname"><br><br>
```

Which appears like this in the browser:

First name:

Figure 44.3: A single-line input field

Or a multiline, resizable text area:

```
<label for="aboutYou">All about you:</label>  
<textarea id="aboutYou" name="aboutYou" rows="4"  
cols="50">  
    Enter some information about yourself.  
</textarea>
```

Appears in the browser like this:

All about you:

Enter some information about yourself.

Figure 44.4: A multiline text input area

Think through the common responses that would be entered in these fields and judge the size accordingly. This is a classic example of how a little UX thought before the UI design phase can massively improve the experience for most users.

Learning points

- Pick a sensible input size based on how much text the user has to enter
- Don't just use default sizes: adjust them for each use case
- Consider these things early in the design phase

#45

**Use Animation with
Care in User Interfaces**

Only a psychopath would deliberately make elements of their UI move while users attempt to interact with it, forcing users to “press and guess” as they try to tap or click controls—that’s more like a video game than a usable digital product.

The prevalence of Flash on the web in the late 1990s and early 2000s led to many designers introducing UI animation just because they could, and it’s almost always a bad idea. Unfortunately, the UI can and does move due to unintentional factors and users are left frustrated.

Do the following scenarios feel familiar?

Have you had a web page load, but the advertising elements (or custom fonts) are served from a different, slower server? As the page loads, the introduction of these ads or fonts “shunts” the page elements around, meaning that you click or tap on the wrong part of the page.

This can be solved by testing, then introducing *placeholders* to reserve space for slow-loading elements, preventing the page from moving as it loads.

Or how about clicking a menu that, while it animates open, disorients you into clicking the wrong option? I feel like I do this on a daily basis on my travels around the web.

Maybe you’re operating a control in a mobile app, when a time-sensitive notification appears—just under your finger as you go to tap something else—taking you out of your app and into another, unintended app. So-called “micro-animations,” where UI controls fade in and out as they’re presented, or menus animate in or out, aren’t necessarily a bad thing. These subtle transitions can add some feeling and personality to your product but care should be taken to ensure micro-animations are:

- Subtle, so as not to distract the user
- Short, so as not to interrupt the key task

If you're asking users to control your software, don't make those same controls move.

Learning points

- Always keep UI control elements static
- Micro-animations are generally fine, but keep them short and subtle
- Test how your interfaces appear on a range of devices and connection speeds, use placeholders to prevent “element shunt”

#46

**Use the Same Date
Picker Controls
Consistently**

This problem of fiddly, arbitrary, and unusable date pickers is less pronounced than it used to be, thanks to browsers and mobile device makers producing more consistent date picker UIs. By triggering the device-native date picker, you can give the user an experience they're familiar with and a UI that has been designed for their device.

There are cases where the native UI won't cut it, however, as some tools need a more complex or more advanced interface for selecting dates, ranges of dates, or comparison date ranges. When this is the case, always use the *same* date picker control everywhere in your app.

Showing a *different* set of controls for the *same* task in a *different* part of your product will confuse users and reduce your conversion rates. A common place where this mistake is made is on holiday or hotel booking sites. The home page will often have a big, clear date picker, designed to convert casual visitors into “searchers” when they land on the site. Once the user is deep into their journey, and they're asked to refine a date range, or pick flights or a hire car, that's when the bad UI creeps in and they're shown a *different* date picker—another new thing to learn.

Please be consistent with your UI—don't confuse your customer with different types of the same UI. It might only take your user an extra second but respect your user's time. Life really is too short to wrestle with bad UIs.

Learning points

- Use the same style of date picker across your product
- Using system-native controls can help to enforce this consistency
- Forcing users to adapt to multiple versions of the same control will confuse them and reduce your conversion rates

#47

**Pre-Fill the Username
in “Forgot Password”
Fields**

If your user has tried to sign in and failed, it's a safe bet that their next action will be to click “forgot password.” Don't make them enter their email again—pre-fill the username field with the entry from their earlier sign-in attempt, so that the user can just tap “reset password” and be on their way.

The forgot password flow of an app is—certainly from metrics I've seen—a very well-used feature. In fact, a user who uses a difficult password, forgets it, then resets it every time is probably more secure than a user who just uses a weak password. So, let's make the forgot password process easy by following these rules:

- If the user gets their password wrong, pre-fill the username field with the last-used username (or email) and show a “forgot password” button



A quick security note on this—ensure you don't confirm or deny the existence of a username (or email) in your system. This could be used by bad actors to assist in phishing attacks (for example).

- When they hit the button, email (or SMS) them a link that expires within a sensible time period, even as short as 30 minutes
- The link, when tapped, should open a page for them to type a new password
- If the link is used more than once, it should still work (users accidentally double-click links often)
- When the new password is set, the user should be automatically signed in to the product

Reducing the frustration of not being able to sign in for returning users is a great move that will dramatically improve their experience.

Learning points

- A user performing a password reset has already given you their username, so reuse it
- Allow them to reset their password with a simple tap or click of a link
- Sign them in once the password is reset

#48

**Make Your Input
Systems Case-
Insensitive**

Lots of systems are case-insensitive by default—that is, they don’t care if characters are UPPERCASE or lowercase—but you don’t notice, because that’s how it should be, and it works well.

For example, emailing `Will@WillGrant.org` goes to the same place as `will@willgrant.org`. Visiting `www.WikiPedia.ORG` takes you to the same site as `www.wikipedia.org`.

The email system and domain name system are both case-insensitive, which was a good call back in the 1970s. Thousands of years of combined technical support time have been avoided by this decision.

Despite this, you can still find apps and websites where you have to sign in with a case-sensitive username or email address. Not only does this lead to errors—a user who cannot sign in because their username had a capital letter they forgot about—but even if they *do* remember, switching between lowercase and uppercase letters on a fiddly mobile keyboard is a pain in the ass.

Misusing case-sensitivity creates a very *opaque* error for the user—they’re usually not sure *why* it doesn’t work—and that’s often the most frustrating type of error.

If you *must* add case-sensitivity for some reason, throw users a lifeline with a hint: “Usernames are case sensitive,” for example. Passwords should always be case-sensitive. For everything else, default to case-insensitive unless you have a very good reason for case-sensitivity.

Learning points

- Default to case-insensitive if you are not sure
- Always make passwords case-sensitive
- If something has to be case-sensitive, tell the user this is the case

#49

**Chatbots Are Usually
a Bad Idea**

Conversational user interfaces (CUIs) are big right now—right at the top of the “hype cycle”—and you can’t move around the web without seeing the “Chat now” bubble in the corner of everything from banking and insurance apps through to online stores like Amazon and Apple.

CUIs are not in themselves a bad thing—they’re focused, simple, and widely understood by users. The problem is: what’s on the other end? Who is your user talking to?

Some of my best user experiences as a customer have come from using online chat: telling Amazon about a faulty product and getting an instant refund, or telling the electricity people that I need a new smart meter in a matter of minutes.

Do you want to know what some of my worst user experiences have been? Chatbots.

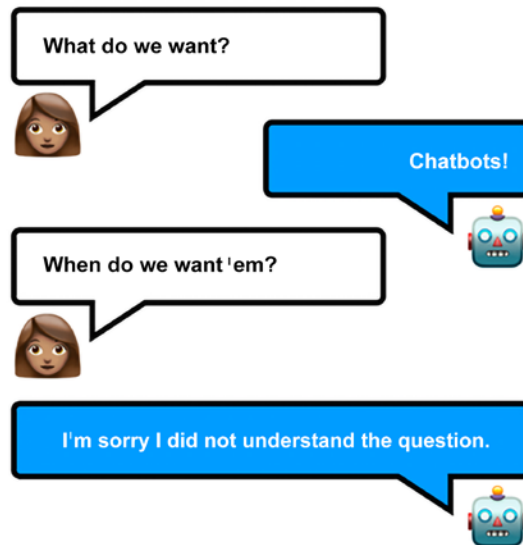


Figure 49.1: Let me know when there's a good chatbot I can try

The key problem with chatbots is this: the AI or language processing on the other end just isn't good enough to deal with my requests properly. Chatbots take the one good thing from the interaction—the human assistant—and throw it away, leaving you with a web-based equivalent of the awful “press 1 for customer support” phone systems.

So instead, treat your user to a real customer service person—letting them know how long it will be before they get a response (or what queue position they're at), and show an indicator when the other person is typing. You'll free up phone lines in your call center and end up with some very happy users.

Learning points

- If you're going to use chat, have a human on the other end
- Give users a realistic wait time for them to get attention
- Let users know that someone is typing

Forms

Filling in forms in applications and websites is often a pain, but it needn't be. A few simple steps and the application of best practices can be the difference between hair-pulling frustration and a seamless form experience.

#50

**If Your Forms Are Good,
Your Product Is Good**

We like to think of our designs in high-minded terms, but let's be real with each other: most apps are really just forms. Complex forms that let you order food, date people, or buy a house—but forms, nonetheless.

Almost every kind of software product features a form (a page with inputs for text, numbers, and other data that the user has to fill in). They are often the source of major frustration, but if you make your forms and data entry work well, your customers will thank you and your conversion rates will improve.

People generally hate filling in forms—it's slow and can be clunky and cumbersome—so let's make users' lives easier by streamlining and optimizing the data entry process.

The first rule of Form Club is: *don't ask for more information than you need*. Time and again, users are asked to sign up to sites that ask for:

- First name
- Last name
- Middle initial
- Email address
- Title
- Organization
- Street address
- Town
- County
- State
- Postal code (ZIP code)
- Telephone number (home)
- Telephone number (cell)
- Telephone number (office)
- Password (with some arbitrary password complexity rules)
- Password (type it again)

If Your Forms Are Good, Your Product Is Good

At this point, your users are close to giving up on joining your product or service. You just don't need all this stuff. Your engineers might have designed the user tables to support it, and maybe your marketing people want it for demographics or direct mailing, but your users don't want it. Kill it.

***required**

Name*	First <input type="text"/>
	Middle Initial <input type="text"/>
	Last <input type="text"/>
E-mail Address*	<input type="text"/> * Please enter your SQUARE ENIX Account Email Address.
Confirm E-mail Address*	<input type="text"/>
Square Enix ID	<input type="text"/>
Security Question	Select a security question.
Answer	<input type="text"/>
Detail*	<div><div></div><div>* Please describe as much as possible in 4000 byte letters.</div></div>

[Next](#)

[Jump back to previous page](#)

© 2009-2018 SQUARE ENIX CO., LTD. All Rights Reserved.

Figure 50.1: This support request form asks for information that the system already knows

In the ideal situation, a user should be able to join your product with:

- Email or cell phone number
- Password (only once—if they get it wrong, they can do a reset)

If you really need other stuff, then:

- Name (any number of names, separated by spaces). The user could add this to an optional profile, rather than it being part of mandatory onboarding.
 - Remember also that not all names are in the “western” form of [Firstname Surname] and your forms shouldn’t throw errors for names from other cultures
 - Also think about field length, don’t make the character limit too short: many non-English language names are much longer than “Joe Smith”
- Address (house number, street, and postal code/ZIP code should be enough). But seriously, if you’re not shipping physical items to the customer, why ask for this? It’s organization-centric, not user-centric—and we should be designing for users’ needs where possible.

Asking your user to enter reams of information on forms is a sure-fire way to reduce conversion to a fraction-of-a-percentage level. If it’s the kind of form they *have* to fill in, at least tell them why you’re asking for the data and how it will be used. So many products get this wrong, so it’s a great opportunity to deliver a good form experience and build a product that people love to use.

Read on for several more form-related principles that may just *change your life*—and your users’.

Learning points

- Don’t ask for more information than you need
- Explain to the user why you’re collecting it and what you will do with it
- Every additional field you add to a form reduces conversion


#51

**Validate Data Entry as
Soon as Possible**

Validation on a form means showing the user some feedback that there's a problem with some of the information they've painstakingly entered.

Validate data entered into a field as soon as possible, when the user moves to the next field, so you know they're done typing in the current one.

Client-side validation isn't always technically possible, but you should aim for it wherever you can because the “round trip” to the server and back is frustrating if there are errors.



The image shows a form validation example. At the top, the text "Pick a username" is displayed in red. Below it is a text input field with a red border. The field contains the text "will" and a red "X" icon on the right side. Below the input field, the message "Sorry, that username's taken. Try another?" is displayed in red.

Figure 51.1: Tell the user to try again before they submit the form

There are lots of techniques for doing this, including plenty of third-party validation libraries for popular programming languages and frameworks. In the bad old days, the user would get a (sometimes partially filled) form back after submitting, with errors marked in red like school homework.

Nowadays, it should be possible to show the user what they've done wrong in real time (for example, too few digits for a phone number) and the steps they can take to rectify it.

The same goes for less common inputs like date pickers—they should include the logic to know that, for example, a hotel guest can't check out *before* they've been checked in. This is a simple check that can help you to avoid a whole raft of common problems.

Don't *ever* clear the form data just because the user made a mistake (see #43, *Respect Users' Time and Effort in Your Forms*)—the exception to this can be payment card information, which is often obfuscated after a form is submitted.

There are bonus points for correcting common errors, for example, the user typing an email address ending in `gmail.con` could display the suggestion **Did you mean gmail.com? Fix it for me!**

Learning points

- Show the user where they've made mistakes as soon as possible—but after they've had the chance to complete the field
- Don't wait until the user has submitted the form
- Use analytics to identify common errors and get ahead of them with checks and suggestions

#52

**If the Form Fails
Validation, Show the
User Which Field Needs
Their Attention**

If you really must validate on the server side and can't do it on the client side (see #51, *Validate Data Entry as Soon as Possible*), then never send a user back to a form without telling them what to do next, and never with a generic message such as “there was an error.”

The user will have likely entered several different bits of data and they'll need to get the context of the form back into their head again, once it comes back from server-side validation. The worst way to do this is by forcing them to scan the whole form again, looking for what they might have got wrong.

Having to scan the whole form again for errors is worse still for partially sighted users with screen readers (or other assistive technology) where the attention needs to be drawn *to the specific field* that needs to be amended. Highlight the problem (or problems) with the form and show the user *where* they need to correct items.



*Figure 52.1: Showing the user exactly where their attention is needed—
not at the “whole form” level*

The W3C's Web Content Accessibility Guidelines (WCAG) include a section on this, *Providing a text description when user input falls outside the required format or values* (<https://www.w3.org/WAI/WCAG21/Techniques/general/G85>), which explicitly advises designers to show which fields need attention.

Sending the user back to an identical form to the one they just submitted, then expecting them to work out what went wrong—like a puzzle—is the world's worst video game.

Learning points

- In server-side validation, there's a delay before the user gets feedback, so help them to remember the context
- Show the user exactly which areas need their attention
- Avoid generic “something is wrong” messages

#53

**Users Don't Know (and
Don't Care) About Your
Data Formats**

The overarching principle of both forms and wider UX practice could be summarized as “be forgiving.”

Things that users do can often seem strange and unpredictable, but they probably have good reasons:

- The user who can’t save their name because it has a special character (like an accent or apostrophe)
- The user who can’t enter a phone number because you’re validating for phone number rules of the wrong locale
- The user who does (or doesn’t) put spaces between groups of digits in their payment card number
- The user who spells their name with an emoji (this *will* happen, I promise)

Just because your developer set telephone fields to be 12 digits and 12 digits only, don’t inflict this kind of madness on your poor users.

Your software should be forgiving—it should allow full names to consist of multiple names, with hyphens and accented characters. It should let users choose to skip non-mandatory fields. It should allow phone numbers with and without prefixes, and with extensions if users wish to enter them. It should allow users to enter postal codes in all manner of unfamiliar ways, for example, don’t force them to enter (or omit) the space.

This freedom will cause problems in your back-end, and your developers may be unhappy, but these are *your problems to solve*, not the users’.

Yes, it’s likely that some of these steps will create technical complexity or more work for your developers—but there are tried and tested technical techniques for this. Your product is there to serve the user, not to make life convenient for your internal development team.

Learning points

- Give your user flexibility in how they enter data
- Don't make your technical challenges a problem for the user
- Expect the user to do unpredictable things with your product

#54

**Pick the Right Control
for the Job**

UX designers have an extensive palette of controls and UI elements to choose from, so it's surprising to see that often the wrong controls are added to forms.

You can enhance the UX of a product considerably by using the right control for the job. HTML5 has extensive form controls, supported by all modern browsers, including color pickers, telephone input, URL input with validation, and so on—and the UI libraries of both iOS and Android contain a vast array of controls for almost all occasions.

Show users a fully featured color-picker with:

```
<input type="color" id="color" name="color">
```

This produces UI like the following:

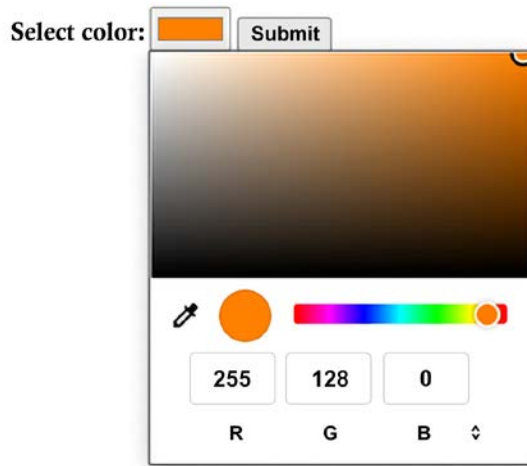


Figure 54.1: There's a full color picker, for free, in HTML!

Similarly, let users pick a month and year with a system-native date picker with:

```
<input type="month" id="month" name="month">
```

This will produce UI like the following:

Birthday (month and year): February 2022

2022

Jan	Feb	Mar	Apr
May	Jun	Jul	Aug
Sep	Oct	Nov	Dec

Clear This month

Figure 54.2: The built-in HTML date picker is highly usable

The right choice might not always be the most obvious control. Here are some examples:

- Showing users two radio buttons for a yes or no (or another binary choice), when a toggle switch would be simpler
- Overusing drop-down controls when there are only a few options (it would be better to use virtually any other control because a dropdown *obscures* the available choices from the user; see #20, *Don't Use a Drop-Down Menu If You Only Have a Few Options*)
- “Building your own” UI for color selection when the HTML color input type is widely supported and shows a bespoke control suited to the user’s device

Once again, this is an area where a little initial thought can save your users a lot of frustration.

Learning points

- Consider whether you're using the best UI control for the job
- The most commonly used approach may not be the best
- Don't build your own when there are standardized controls available to use

User Data

An extension of Input and Forms, these chapters deal with the “special cases” of handling user input. Payments, currency, shipping, and images are all covered with tried-and-tested UX principles.

#55

**Allow Users to Enter
Phone Numbers
However They Wish**

Phone number entry should be as painless as possible for the user. Don't attempt to validate them, split them into groups of numbers, apply brackets, or any of the other weird tricks you see all over the web and in apps. If you've tried to use a UK mobile number on a form that's demanding a US number, then you'll know the feeling.

My theory is that this sort of design comes from traditional paper form filling. Designers are tasked with copying or recreating what was once a paper form into the company's shiny new web application, but they take this too literally and users end up with a horrible experience as a result.

Stop and think about whether a phone number is even necessary for most registration forms. I hate using the phone. The "Phone" app is my least favorite app on my phone (I've tried deleting the app but it won't let me). However, I concede that there will be cases where you absolutely *have* to collect a phone number.

Carry out your clever phone number detection and parsing on the server side and let the user just simply key in their phone number. In front-end HTML you should be using `<input type="tel">`, which, on a mobile device, shows the telephone keypad when the field is tapped—and will show "Autofill" on modern smartphones, meaning that the user can add their number with one tap.

Allow Users to Enter Phone Numbers However They Wish

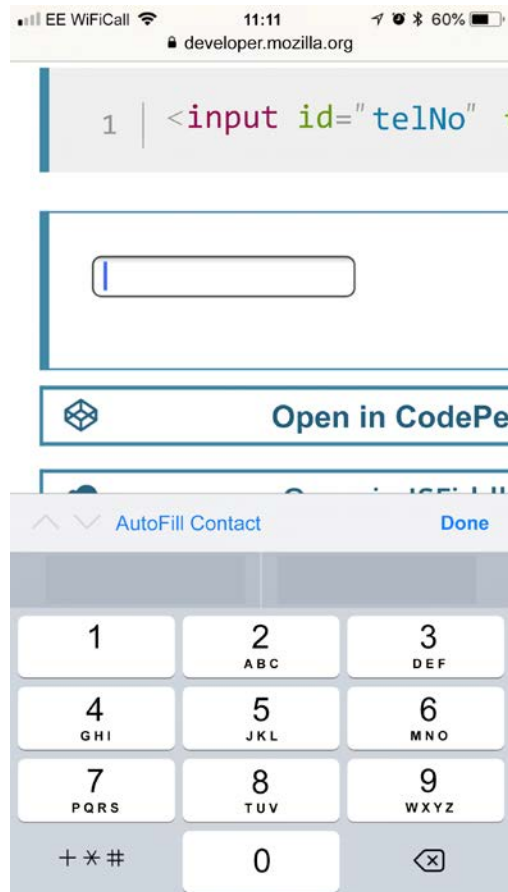


Figure 55.1: The numeric keypad being shown for a “tel” field

The HTML for this form would look like this:

```
<label for="phone">Enter your phone number:</label>
<input type="tel" id="phone" name="phone">
```

Learning points

- Don't attempt to validate or parse phone numbers in the UI
- Just let the user key in their number
- Show the user a numeric keypad on mobile

#56

**Use Dropdowns
Sensibly for Date Entry**

A user entering a full date (like a date of birth) should be offered a dropdown for the day and month, then a numeric entry for the year. Day and month are sufficiently short that a dropdown doesn't feel too cumbersome. It also solves the issue of US dates having their day and month in the opposite order to most of the rest of the world.

Don't use a dropdown for the year though: it looks crazy and forcing the elderly to scroll back to the early 1900s seems very cruel. For mobile, use responsive design to show mobile users the date picker, a custom-designed UI on iOS and Android that makes picking dates a piece of cake.

Be honest—would you rather build your own mobile date entry UI or stand on the shoulders of the designers and researchers at Apple and Google, who've done all the hard work for you?

The system-native date picker will also be familiar to users, reducing cognitive load and giving them one less thing to learn.

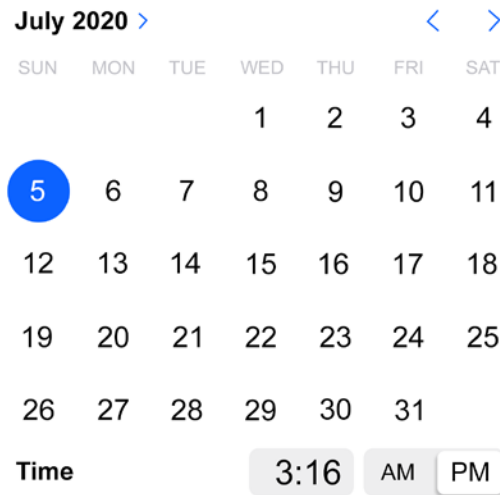


Figure 56.1: The iOS 14 date picker

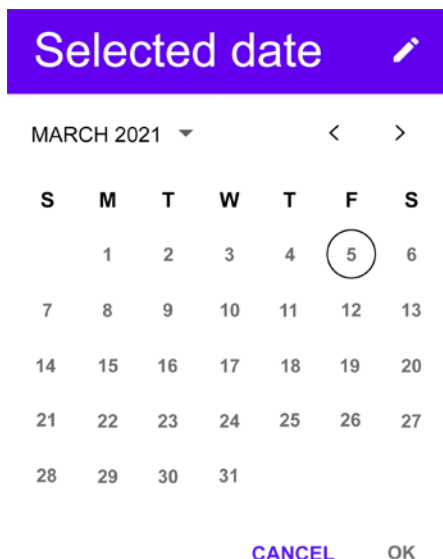


Figure 56.2: The date picker on Android

One exception to this could be enterprise “expert” systems, where a user has to enter customer dates frequently—in this case, it might be easier for these experienced daily users to just provide text `<input>` fields and let them use the keyboard numpad to enter DDMMYYYY.

Learning points

- Use drop-down selectors for the day and month
- Use numeric entry for the year
- Mobile devices should show the system-native date picker

#57

**Capture the Bare
Minimum When
Requesting Payment
Card Details**

The end goal for a lot of sites and apps is getting a customer to pay for a product or service. It's a cause for celebration: we've made something or are offering something so good that the user is happy to spend their hard-earned currency with us. So, why do we make it so hard for them to do so?

Over the years we've all pieced together a robust mental model of what it means to pay for something online. Any attempt to change this model to something different or more complex is going to create user friction and confusion. A credit or debit card number is already an unwieldy amount of data for a user to enter, so make it as easy as possible for them:

- Only collect what you need: the card number, expiry date, and CV2 code should be sufficient for almost all online purchases.
- Depending on your payment provider you may also have to collect postal (or ZIP) codes.
- Allow the user to type the full card number into one field, but visually split it into groups of four digits as they enter it. This makes errors easier to spot while preventing the user from having to move between four separate input fields.
- If the user hits the spacebar, then remove the space for them.
- Include some help text describing where to find the CV2 or card security code. It's not worth losing a customer because some people have different terms for this code.

Capture the Bare Minimum When Requesting Payment Card Details

Stripe’s checkout (below) does this particularly well with a remarkably useful icon that can be clicked for further information:

Apple Pay

Or pay with card

Email

will@willgrant.org

Card information

4242 4242 4242 4242

VISA

01 / 23123

Name on card

William Grant

Country or region

United Kingdom

Postal code

Pay \$129.00

Figure 57.1: Stripe’s default checkout behavior is pretty much perfect

Bonus points for supporting Apple and Android Pay, so your mobile users can checkout securely with one tap rather than the more error-prone route of entering their details.

If you do not need to collect the “valid from” date, issue number, or postal/ZIP code—or 10 other things you think might be useful—do not collect them. Every form field is another thing for the user to do, another bit of information to find and parse, and another chance for them to get stuck, change their mind, get bored, or otherwise abandon your payment form.

Finally, use standard terms for the form fields, as this will enable most modern browsers on desktop and mobile to detect and auto-fill card details from the secure password storage. Then all your user needs to do is quickly review the details and click “Pay.”

If you’re using a service such as Stripe or Shopify, this will be done for you, but in case you’re writing your own HTML, this markup will auto-complete nicely on all modern browsers:

```
<label for="frmNameCC">Name on card</label>
<input name="ccname" id="frmNameCC" required
placeholder="Full Name" autocomplete="cc-name">

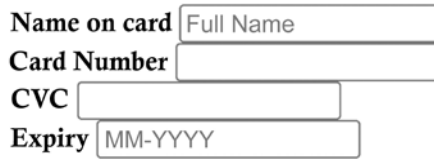
<label for="frmCCNum">Card Number</label>
<input name="cardnumber" id="frmCCNum" required
autocomplete="cc-number">

<label for="frmCCVC">CVC</label>
<input name="cvc" id="frmCCVC" required
autocomplete="cc-csc">

<label for="frmCCExp">Expiry</label>
<input name="cc-exp" id="frmCCExp" required
placeholder="MM-YYYY" autocomplete="cc-exp">
```

Capture the Bare Minimum When Requesting Payment Card Details

This will generate a UI like the following:



The form consists of four rows, each with a label on the left and a text input field on the right. The labels are 'Name on card', 'Card Number', 'CVC', and 'Expiry'. The input fields contain the placeholder text 'Full Name', an empty field, an empty field, and 'MM-YYYY' respectively.

Name on card	Full Name
Card Number	
CVC	
Expiry	MM-YYYY

Figure 57.2: Customized form fields for optimized autofill

Learning points

- Only collect the bare minimum of information you need to make the transaction
- Be forgiving: absorb accidental spaces and make numbers readable
- Optimize your markup to ensure browsers can auto-fill card information if it's stored on the user's device

#58

**Make It Easy for Users
to Enter Postal or ZIP
Codes**

Postal codes and ZIP codes vary wildly around the world. Don't try to guess the format for the user: simply give them a text entry input field and allow them to enter their code. You can carry out validation if you need to on the server side.

Some of the better forms that I've seen in recent years include a "live lookup," where entering a postal code (or part of a postal code) will return a list of possible addresses for the user to tap or click. Obviously, this reduces the keystrokes and clicks that the user needs to make to enter their address, and also reduces error rates by pre-filling fields with data that has already been sanitized. There are several free (and paid-for) services that offer this, including Google's "Place Autocomplete."

If you're dealing with a web page (as opposed to a native app, for example), then using the autocomplete attribute on an input element in HTML will prompt some browsers to offer to auto-fill that field:

```
<input autocomplete="shipping postal-code">
```

This will work on the Android and iOS browsers, offering the user the chance to populate their own postal code in one tap:

Zip code only
alternative to current approach in checkout

Please enter your postal code:

Enter a location

Mixed approach

Please enter your address:

Enter a location

^ v AutoFill Contact Done

home
B3 1QS

Q W E R T Y U I O P
A S D F G H J K L
↑ Z X C V B N M ↵
123 😊 🎤 space return

Figure 58.1: Pre-filling the postal code with one tap

Give the browser a helping hand by using this recommended markup, which will play nicely with all modern browsers:

```
<label for="frmAddressS">Address</label>
<input name="ship-address" required id="frmAddressS"
placeholder="123 Any Street" autocomplete="shipping
street-address">

<label for="frmCityS">City</label>
<input name="ship-city" required id="frmCityS"
placeholder="New York" autocomplete="shipping locality">

<label for="frmStateS">State</label>
<input name="ship-state" required id="frmStateS"
placeholder="NY" autocomplete="shipping region">

<label for="frmZipS">Zip</label>
<input name="ship-zip" required id="frmZipS"
placeholder="10011" autocomplete="shipping postal-code">

<label for="frmCountryS">Country</label>
<input name="ship-country" required id="frmCountryS"
placeholder="USA" autocomplete="shipping country">
```

This will produce a UI like the following:

Address 123 Any Street

City New York

State NY

Zip 10011

Country USA

Figure 58.2: Autocompleting form UI



Country selection is often found at the end of a form. Move it to the very start to avoid users entering all their data, then clearing it because choosing a different country changes the form's fields.

Learning points

- Form entry is a pain for users, so just let them enter their postal code however they wish and validate it on the server side later
- Offer a “live lookup” for postal-code-to-address conversion, if possible
- Allow autocompletion of form fields using HTML

#59

**Don't Add Decimal
Places to Currency
Input**

This is yet another example where keeping it simple is the best option. Many currency input situations (sending a bank payment or adding a tip, for example) require the user to enter a value, which could be a whole amount (\$10) or an arbitrary amount (£5.99).

Products sometimes try to be too helpful by auto-adding the decimal place or adding “.00” to the end of the value, which tends to lead to errors. Not the fun kind of errors either, but the kind where you bid \$1000.00 for some underpants on eBay when you meant to offer \$10.00 for them as the absolute maximum. To avoid this, allow the user to type the decimal themselves, but assume a “.00” if they don’t.

You can improve some experiences by offering pre-set amounts. Tipping is a great example—many ride-hailing apps will have pre-set tip buttons for \$1, \$5, and so on.

Enter your donation amount

\$.

DONATE

Figure 59.1: Please don’t do this

This anti-pattern is *preventing* users from entering the amount they want in numbers and instead forcing them to increment a value up one number at a time—or separating dollars and cents into different inputs.



After the entry is done, always present the value *back* to the user, so they can hit “confirm” or go back and edit it. Users don’t want to make mistakes with money and this confirmation step is a great way of building trust in your user experience.

Learning points

- Don't add decimal places to currency entry, as it can lead to errors
- Allow the user to type the smaller denominations if they wish, but assume "0" if they don't
- Always ask the user to confirm the amount after entry

#60

**Make It Painless for the
User to Add Images**

There are a lot of situations in web and mobile apps where the user is asked to upload an image. It's done in a variety of ways, but here are some principles for getting user input in the form of images:

- Give the user the choice of picking a file or taking a picture, which is especially useful on mobile or tablet, where the request can trigger the system image picker, which has more functionality than your app can provide.
- On iOS and Android, the user will be asked to give permissions the first time they're asked to add a photo—and it's really easy for them to accidentally tap “deny.” You should provide some help text or hint on how they'd go into settings to correct this.
- Consider whether you would like the user to upload multiple images. If so, allow them to do this in one go, rather than lots of separate selections. The markup to allow the browser to send multiple images back looks like this:

```
<form action="/upload_image">
  <label for="images">Select images:</label>
  <input type="file" id="images" name="images"
multiple>
  <input type="submit">
</form>
```

- Give the user “crop” and “rotate” controls when the image is previewed. It's super useful to be able to trim and rotate an image with a couple of clicks, rather than using another tool to do so.
- Try to accept a wide variety of image formats: JPEG, PNG, and GIF at the very least. Be aware that any modern smartphone will probably be storing its images as HEIC (Apple) or WebP (Google)—and it's a common blunder to not allow these images, thereby excluding a huge amount of users on mobile.

Make It Painless for the User to Add Images

- Tell the user that the image is uploading and show them the progress (uploads can be slow, especially on mobile devices)—see #61, *Use a “Linear” Progress Bar If a Task Will Take a Determinate Amount of Time*.
- For avatar images, consider using a third-party service such as Gravatar, which should mean a good proportion of your users won’t need to add an image at all. After all, the best interface is no interface.

Learning points

- Use device features for capturing images if they’re available
- Allow multiple image uploads in one go if you want to collect more than one image
- Keep the user informed about the upload progress

Progress

Not everything a user wants to do is instant, so how do you keep users informed of their progress—and your system's progress? This section deals with the best practices for letting users know that they're waiting, and how long they'll be waiting for.

#61

Use a “Linear” Progress Bar If a Task Will Take a Determinate Amount of Time

Despite your iPhone having the number-crunching power of a late-1990s supercomputer, everyday tasks still seem to take a maddeningly long time in a lot of software. Printing, for example: why does it take *so long* for the computer to send a document to a printer? It's as if the printer must work out how to be a printer every time. Regardless, it's a great idea to let users know how long they're going to be waiting for.

By the word determinate, I mean that your software “knows” the number of things it must do (or can work it out), and can work through them while updating on progress. Default to this option if you can.

Never show a series of completing progress bars, for example:

- Copying: 0...10...50..100%
- Decompressing: 0..20...60..100%
- Installing: 0...15...45...80...100%
- Finishing up: 0...20...60...100%

Another similar anti-pattern is the source of some of the most awful (and hilarious) UI disasters that I've come across have this travesty in common: an animated (often a GIF) progress bar which, once it has meandered its way to the end, restarts back at zero and does it all over again. Think of it as a “linear spinner”. If you really dislike your users, then this is an excellent way to “troll” them.

Instead, just provide *one* progress bar:



Figure 61.1: An effective progress bar

Use a “Linear” Progress Bar If a Task Will Take a Determinate Amount of Time

A progress bar with a start and end, that gradually fills as the task completes, is the gold standard for this. There’s no ambiguity and the user can get a good idea of how long this task will take, and that it’s proceeding as planned. Users are already familiar with this metaphor: a linear progress bar is really similar to the “playback” UI in video or audio software, where this can be combined with “pause” and “rewind” controls, for example.

Learning points

- Show a linear progress bar if your software is able to
- Show only one progress bar for the whole operation
- Give the progress bar a clear start and end

#62

**Show a Numeric
Progress Indicator on
the Progress Bar**

Following on from #61, *Use a “Linear” Progress Bar If a Task Will Take a Determinate Amount of Time*, we can enhance the experience by showing a numeric (percentage) indicator on the progress bar, but only if there’s time to read it.

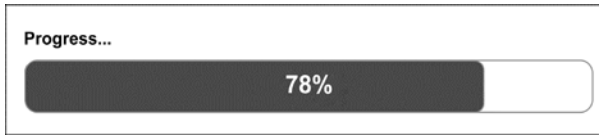


Figure 62.1: A progress bar with a numeric indicator

A progress bar and a number that appears for a fraction of a second is just confusing and adds to the visual clutter that the user needs to process. If they’re going to be stuck looking at the progress bar for a few seconds, then a percentage is a nice, universally understood way of keeping them updated.

Also, this numeric indicator could be an amount of time. So, for an update, you can show a certain number of minutes remaining. However, a percentage is more useful for shorter processes. Be careful because calculating “time remaining” is often a big technical challenge. It’s common to see an update say, “24 minutes remaining” and then see it complete in the next few seconds. If you’re not confident that you’re giving users an accurate time, then it’s better to leave it out and use a percentage instead.

Anything above a minute’s wait will start to make users ask “how long is this going to take?” and should therefore include a “time remaining” estimate if feasible to do so.

That’s it, the captivating progress bar section is over. Now, if we can all *please* get progress bars right over the next few years, we could have millions more happy users out there!

Learning points

- Show a “percentage complete” numeric indicator on a progress bar, if there’s time to read it
- For long processes, consider showing the time remaining
- If you can’t show an accurate estimate of the time remaining, just revert to a percentage

#63

**Show a “Spinner” If
the Task Will Take an
Indeterminate Amount
of Time**

In this case, by indeterminate, I mean that your software isn't sure (or has no way of knowing) how many things it must do: it just knows that it *will know* when it is done.

Showing an animated spinner gives a user less information than a progress bar, but it at least tells them that something is happening, and their task will be done when the spinner vanishes.



Figure 63.1: A spinner. Other styles of spinner are available

If something goes wrong, then make the spinner stop. Your user doesn't know whether this is a “loop forever” GIF, so they'll just carry on waiting when nothing is happening behind the scenes. Gmail shows “loading” and then, after some time, it shows “still loading”, which is a nice touch.

A spinner is also great for tasks that are very short, for example when a page reloads, or when fetching some updated state from the server, or *inside* a button or control—where a progress bar would be overkill.

Learning points

- Use a spinner when your product can't reliably show a progress bar
- Use animation to indicate that something is happening
- Stop or remove the spinner if something goes wrong

Accessible Design

As designers we have an obligation to make sure that the widest possible section of society can use our products. This section aims to ensure you design products that people with disabilities can use easily—not only is it the right thing to do, we'll also see how it improves usability for everyone.

#64

**Contrast Ratios
Are Your Friends**

Way back in 1999, the **World Wide Web Consortium (W3C)**, the main international standards organization for the internet, published the catchily-titled **Web Content Accessibility Guidelines (WCAG)**. They were revised and updated to “WCAG 2.0” in 2008 (WCAG 3 is currently in “Draft” state), with the guidelines stating that “websites must be perceivable, operable, understandable, and robust.”

The guidelines are extensive and detailed, and go beyond the scope of this book, but some key elements from them are great best-practice guidelines to incorporate into your user interface to improve the UX for everyone—not just for disabled people.

One great guideline is that on contrast:



1.4.3 Contrast (Minimum): The visual presentation of text and images of text has a contrast ratio of at least 4.5:1.

- Understanding Success Criterion 1.4.3: Contrast (Minimum)

<https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html>

There are some exemptions and caveats around logos and especially large text, but the “golden rule” contrast ratio of 4.5:1 is one to live by. See the three example buttons below. The low-contrast button would be hard to use for partially sighted people.

What’s more, people with perfect vision would also find it annoying and difficult to read (especially on a tiny mobile screen).

Example text at a 7.5 to 1 contrast ratio

Example text at a 4.5 to 1 contrast ratio

Example text at a 2.5 to 1 contrast ratio

Figure 64.1: How contrast makes a difference—you should be aiming for the top example

There are automated contrast checkers on the web, so search for one in your favorite search engine and give it a whirl on your UI contrast. A decent contrast ratio will help partially sighted people, as well as preventing fully sighted users from getting frustrated.

Remember, if the marketing team tells you that text on controls or in-app copy *has* to be in a low-contrast color combination for branding reasons, then tell them where to shove their brand guide—you'll be doing a great service to people with less-than-perfect vision.

Learning points

- A contrast ratio of 4.5:1 is an absolute minimum
- Aim for a contrast ratio of around 7.5:1 for maximum readability
- Like many accessibility tweaks, this one benefits all users, regardless of their ability

#65

**If You Must Use
“Flat Design” Then
Add Some Visual
Affordances to Controls**

Minimalism is generally good and reducing clutter and visual distractions can often help a user to find what they need more quickly. Minimalism does not, however, mean making controls so minimal that they are impossible to use.

The flat design aesthetic (refer to #13, *Make Interactive Elements Obvious and Discoverable*) tends to remove visual affordances, but not to the same extent as the newly-emerging “web brutalism.” Brutalism, inspired by the brutalist architectural style, is an aesthetic in product design that deliberately looks unstyled and raw (Craigslist is a great example):



Figure 65.1: Craigslist: no, it's not broken—it's supposed to look like that

Outside of being a joke for designers, this level of minimalism is too imposing and unnecessary and, like flat design, can degrade discoverability by removing all visual affordances.

Let’s look at some UI in the widely-used Google Calendar (iOS) app. The strict adherence to flat design here means that it’s very hard to work out what is *tappable* and what isn’t:

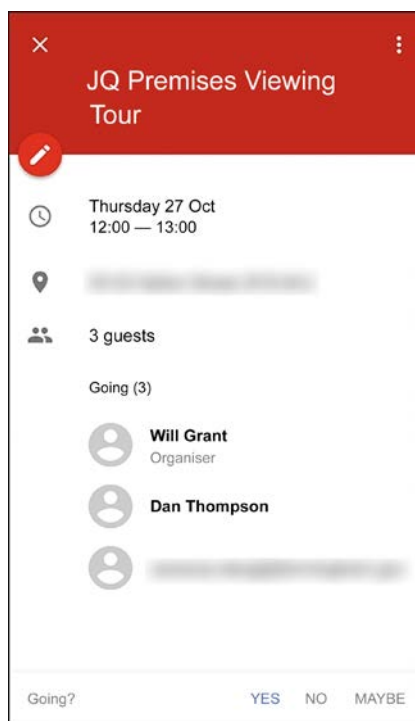


Figure 65.2: The Google Calendar app

It’s good that less frequently used controls (like “email” and “delete”) are hidden from most users, but they’re not discoverable: the menu to find them is a small, unlabelled “ellipsis” menu in the top right—the controls should be exposed at the top level, or exposed in a labeled, discoverable menu if they’re advanced features.

There's also a problem with consistency. The “pencil” edit icon is tappable and has a subtle drop shadow (yay, a visual affordance!), but none of the other tappable items do—making it hard for the user to build a mental model of how to go about interacting with this product.

Compare Google Calendar to a section of the user interface from the following control panel of **Stripe.com**:

Home Payments Balances Customers Products Reports Connect More ▾ Developers Test mode ☐

Profile Edit

Email

Name Will Grant

Password

Two-step authentication Add authentication step

Increase security for your account by using multiple authentication steps.

SMS Update...

If you lose your mobile device or security key, you can [generate a backup code](#) to sign in to your account.

Language Save

Please select a preferred language for your Dashboard, including date, time and number formatting.

Language

Figure 65.3: A section of Stripe’s control panel

It strikes a great balance between minimalism and affordances. It’s clean and simple, and here are some of the reasons why:

- It’s broken down into logical sections with titles
- The controls are clearly recognizable as tappable thanks to their subtle shadows

If You Must Use “Flat Design” Then Add Some Visual Affordances to Controls

- A pencil icon is used for **Edit**, as well as being labeled with some text
- The **Update...** button features an ellipsis, indicating that there’s another step to be completed when it’s clicked

I guarantee that users will have a better experience with this UI than with Google’s—if such a randomized test could be constructed.

Learning points

- Visual affordances on controls are still vital for all user interfaces
- Consistency across your product will help users to learn your interface more quickly
- Don’t take minimalism too far: find a balance


#66

**Avoid Ambiguous
Symbols**

This is easier said than done, but there are some symbols and iconography that are often used and misused across products.

This principle aims to encourage you to think about two things; “Am I using symbols that the user will be familiar with?” and “Do these meanings conflict with other parts of their experience?”

Here are just a couple of examples from products on the web and mobile, but there are hundreds more:

- @: The “at” symbol is a repeat offender in the context of the control. Does it mean email, a web link, to mention a user, or something else?
- : Does this mean “share” or “new window” or “open additional menu options”? I’ve seen it used to represent all of the above, as well as upside down to mean “go back.”

Some things to think about when picking iconography:

- Is there a well-used existing icon for this that can be reused? Users will already know it and you don’t have to redesign it.
- Operating systems and web frameworks will often have guidance on their standard symbols and their intended uses – check these and stick to them, as the user is already familiar with them.
- Is this proposed icon distinct from the others and memorable?
- Does this proposed icon conflict with any established patterns?

By giving your iconography and symbols a bit of extra thought, you can help to make your interface—and, therefore, your user’s experience—a whole lot better.

Learning points

- Choose your icons with care and thought
- Don’t reinvent the wheel: there’s probably an established pattern that you can reuse
- Icons are like jokes: if you have to explain them, they’ve not worked

#67

**Make Links Make Sense
Out of Context**

Q: What's the difference between these two ways of offering a web link to a user?

- To download our brochure: **click here**.
- You can **download our brochure** here.

A: The first one is harder for visually impaired people to use.

Screen-reader software often has a mode where the user can skim the page for clickable links, and these links need to make sense out of context. In this case, the first link would be read aloud as “click here,” while the second would be dictated as “download our brochure”—much more usable.

Let's take another example from an index of blog posts. Compare the following links:

- Blog post story 1: **Read more**
- Blog post item 2: **Read blog post item 2**

In this example, recapping the title in the “read more” link gives additional context and prevents the screen reader from simply reading a list of “read more, read more” over and over.



Bonus: Making your links descriptive can help some search indexes to make sense of your content.

Learning points

- Avoid “click here” links
- Use descriptive links that make sense out of context
- This will help with search indexing as well as accessibility

#68

**Add “Skip to Content”
Links Above the Header
and Navigation**

As previously mentioned, some users with a visual impairment will be using screen-reader technology to read the text elements of your interface aloud.

One problem is that it's easy for these users to get lost in the mess of links and content on an especially busy page. Users need a way to get to the navigation. For fully sighted users, the location of the navigation is a well-accepted pattern, but partially sighted users may not have the same “mental model” of a web page or web app.

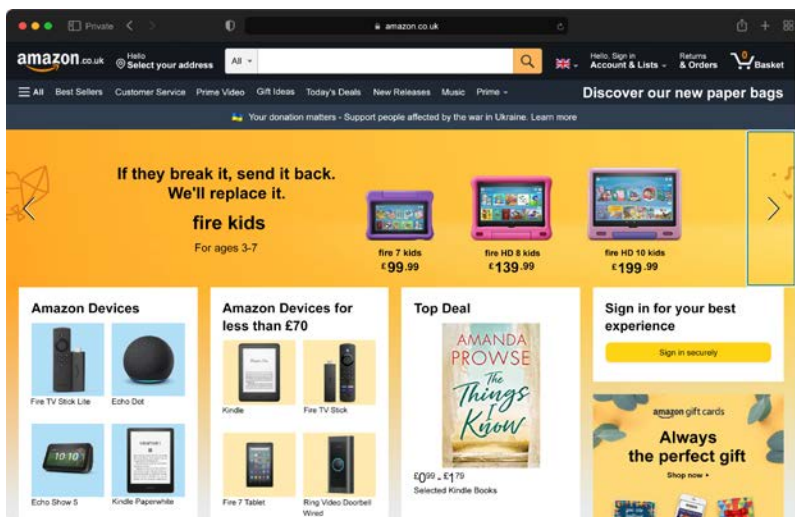


Figure 68.1: Amazon's homepage is, at the time of writing, almost impossible for visually impaired people to navigate using a keyboard.

Adding a “skip to content” link to the top of your site (it can be hidden with CSS rules and only needs to be visible to assistive technologies like screen reader software) will allow the user to skip past the navigation effortlessly. They don't want to have to hear all your menu options read aloud, over and over, each time a page is loaded.

Add “Skip to Content” Links Above the Header and Navigation

Here’s the CSS that the W3C recommends using to position the link off-screen for sighted users:

```
#skiptocontent {  
  height: 1px;  
  width: 1px;  
  position: absolute;  
  overflow: hidden;  
  top: -10px;  
}
```

And in the HTML:

```
<div class="skipnav">  
  <a href="#skiptocontent">Skip to main content</a>  
</div>
```

Learning points

- Add a “skip to content” link to the top of your site
- Use CSS positioning to hide the link for sighted users
- Include this in your site or app templates so it appears on every page automatically

#69

**Never Use Color Alone
to Convey Information**

This sometimes sounds counterintuitive: making a warning red or a success alert green is second nature to most designers. While color can act as a shorthand for most users, those with color blindness can find themselves at a disadvantage. Certain types of color blindness will mean that users can't tell the difference between a red status blob and a green one.

The best way to approach this is to use color to convey *additional* information, and not just use color alone. This makes the site usable for most people, but not at the expense of a few. Therefore I advise making links underlined (and, optionally, a different color), not *just* a different color, to differentiate them from body copy.

Another example is that a “status normal” label could show a green indicator blob, but should never *just* be the green blob on its own.



Figure 69.1: Only one of these interfaces is usable for color-blind people



Figure 69.2: People with red/green color blindness (deuteranopia) would still perceive the correct meaning from these icons

Color is a great secondary indicator: a visual cue that will help people to identify elements of your product more quickly and get information simply.

Never Use Color Alone to Convey Information

Color vision difficulty affects approximately 1 in 12 men (8%) and 1 in 200 women in the world—that's as many as 298 million people worldwide. This principle is intended to remind you that not everyone can see colors, so it shouldn't be the only way of conveying a message.

Learning points

- Don't use color on its own to convey information
- Ensure that there are other indicators along with color
- Color is still a great secondary source of information for users; there's no need to remove color entirely

#70

**If You Turn off Device
Zoom with a Meta Tag,
You're Evil**

Adding the following meta tag to the head of an HTML page will prevent the user from scaling the page, either using their browser controls or with “pinch-to-zoom” on a touchscreen device:

```
<meta name="viewport" content="width=device-width,  
initial-scale=1.0, maximum-scale=1.0, user-scalable=no"  
>
```

Although rare, it’s still seen in the wild. Designers typically do this because:

- They haven’t made their designs work responsively or don’t know how to
- They don’t understand the implications for accessibility

The result is that this prevents users with vision difficulties from scaling the page.

Don’t be these designers. Let your users choose how to view and manipulate your interfaces. Away from web pages, offer these scaling controls in desktop and native mobile software. iOS and Android both have built-in support for accessibility features that you can hook into and, as a result, respect the user’s preferences for type size and contrast.

This is the correct code:

```
<meta name="viewport" content="width=device-width,  
initial-scale=1.0" />
```

This sets the initial display size while still allowing users the freedom to scale and interpret your product as they wish, and using their own assistive technologies if needed.

A designer can’t ever know how a user will want to view their content, so don’t assume. Not being able to design a pixel-perfect outcome for every device size means that pulling these kinds of tricks (disabling scale, for example) is simply shooting yourself in the foot—you’ve made the site match your mock-ups but now some people can’t use it.

As with most adjustments for accessibility, responsive content creates a better experience for all users, regardless of ability.

Learning points

- Let go of pixel-perfect design and accept that users will want to view your products on their terms
- Use device-native accessibility features where you can
- Test your product's interfaces on multiple device sizes and with assistive technologies

#71

**Give Navigation
Elements a Logical Tab
Order**

Try an experiment: head to a website in your browser and once the page has loaded, start pressing the *Tab* key repeatedly. You should notice the “focus” (usually a colored rectangle or shaded area) move from item to item across the site.

This is one of the ways that users who are partially sighted, or have motion difficulties, use web pages. These users rely on interface designers to use common sense in the tab order that they assign to items. On some websites and web apps, this is unusable, while on some, it’s clearly been well thought through.

Filling in a form is often extra frustrating when tapping the *Tab* key takes your focus to a strange part of the page. It’s unlikely that you’ll be writing code yourself, but you may wish to tell your frontend developers that they can specify the order that items are selected in using the `tabindex` attribute:

```
<input type="text" name="field1" tabindex="1" />
<input type="text" name="field2" tabindex="2" />
```

The archetypal example here is site search. Most site layouts place the search field over on the right-hand side, after all the navigation elements.

This makes sense, but for assistive technology users it can mean a great deal of tabbing through the menus to get to search. Give the search field `tabindex="1"` and the user can reach it easily, then skip over it to the navigation if they’d rather use that route to navigate.

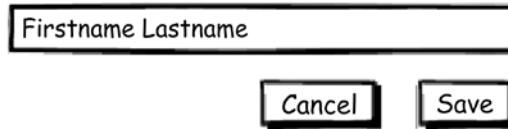
Learning points

- Ensure that tabbing around your UI takes the focus in a sensible direction
- Although this is especially important for accessibility, all users will benefit from forms that are easier to move around
- Test your designs with assistive technologies, or at the very least test keyboard controls in your own browser

#72

**Write Clear Labels for
Controls**

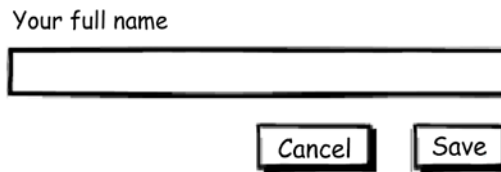
Another small change you can make, which will make the world of difference to your users using assistive technologies, is writing clear labels:



Firstname Lastname

Cancel Save

Figure 72.1: An example of bad labelling



Your full name

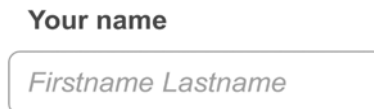
Cancel Save

Figure 72.2: An example of good labelling

Pre-filling your field with placeholder (or watermark) text may look tidy, but it's not supported in all browsers and disappears when the focus moves to the input field.

You can, however, include **both**, which allows the field to be identified and gives some assistance to users as to the kind of information that is needed for that input.

A word about alignment: from a logical “order of the content” point of view, it makes sense to add the label above the field and left align. All other alignments (right align, label to the left, and so on) have various visual and accessibility issues: your default should be above the field, left-aligned.



Your name

Firstname Lastname

Figure 72.3: An input with a clear label and a helpful watermark

I know I've tapped a field many times, planning to type some information in, only to stop and think, "Wait, what was this field for?" Yet again, this is an example of improving accessibility (screen readers use labels to make forms usable), while also improving the overall experience for your whole audience.

Learning points

- Screen readers rely on labels for partially sighted users
- Field labels benefit all users
- Placeholder labels disappear or are obscured when the user types in the field

#73

**Make Tappable Areas
Finger-Sized**

If you think your design will be used by touch, then your users' fingers are the tool that they'll use. Given that obvious statement, it's surprising to see UI controls in touch interfaces that are clearly *way too small* for users to poke at easily with their digits.

As a guide, your smartphone screen is (roughly) five fingers wide and 10 fingers high, so—if you introduce padding (the area around each control)—that's about 40 controls: a hard limit of the controls that can be comfortably used on such a display. If you were to try to fit more than four or five items horizontally across the display, they would be too small to be comfortably used.

You'll need to experiment to find the right control size, but if you're using native control elements (see #39, *Use Device-Native Input Features Where Possible*), that research has been done for you: they're already the right size (44px or 48dp are the minimum 'safe' sizes to be usable by human fingers). Just use those controls instead of re-inventing them with less user testing.



Figure 73.1: Make controls a size that humans can operate with their fingers

If you're building your own touchscreen controls, use the human finger size as a guide. Trying to grab a 1 or 2 pixel-sized control with a finger is needlessly difficult and will frustrate users no end.

Don't make elements directly adjacent if some users will be accessing your product via touch. Padding between buttons prevents the wrong button from being touched accidentally. 3mm is a good guide for padding, in however many pixels that means for your display.

An added complexity is caused by the emergence of devices that have dual input methods—a laptop with a touch screen, for example.

Make Tappable Areas Finger-Sized

This forces designers to consider touch operation in interfaces that would previously have been mouse only. The result of this is that you *always* have to consider this principle to ensure your interfaces are usable by touch.

Learning points

- Think about the size of human fingers when designing touch interfaces
- Don't make touch controls too small for users to use comfortably
- Add padding between control elements to prevent accidental mis-taps

Journeys and State

Your customers move through journeys in your product on their way to completing their goals. This section includes basics like breadcrumb navigation, through to removing common annoyances to let users flow through their tasks more efficiently.

#74

**Let Users Turn off
Specific Notifications**

Notifications, whether on desktop or, more commonly, mobile, are a great way to keep users informed of state changes while the app is closed or in the background.

It's worth thinking through carefully how users can customize or disable certain types of notifications (or all of them). The events that each user considers important, or notification-worthy, will vary and may even change over time.

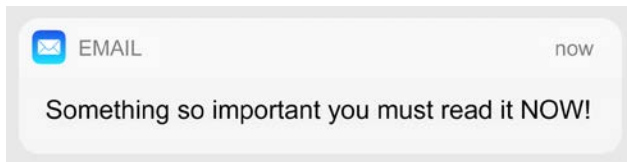


Figure 74.1: A notification

A user probably doesn't want an audio notification every time someone likes their Instagram selfie. Perhaps they *do* want a notification of a direct message because they get them less frequently.

The user's device or browser will allow them to disable notifications *entirely* for your app, which they will do if what they're seeing isn't fine-tuned enough.

It's extra technical work to allow a user this kind of fine-grained control, but allowing them to set up notifications how they want them is a serious advantage over your competitors' products.

There's an approach, particularly in mobile products, that seems to advocate bombarding users with as many push notifications as possible, to encourage repeat usage. In my experience, this does more harm than good to your product in terms of UX and retention.

It has become such a source of annoyance that the latest version of iOS (at the time of writing, iOS 15) actively prompts users with each new notification, asking them if they'd like to turn this off permanently.

Let Users Turn off Specific Notifications

A pattern that can work well is allowing users to “snooze” a notification. The snooze action will dismiss the alert temporarily and remind the user once again after a period of time.

An added bonus is that by allowing a user fine-grained control over these notifications, they are going to be happier with your app’s “noise level” and will be less likely to disable notifications at a system level—meaning your app’s *useful* updates can still get through.

Learning points

- Allow users fine-grained control over notifications
- Don’t bombard users with too many messages
- Remember that users can simply disable all notifications for your product at a system level

#75

**Each Aspect of a User's
Journey Should Have a
Beginning and End**

The user's journey can be thought of in a broad or narrow way: it can be their journey through the whole product—for a dating app, that could be from signing up to a first date—or it could be a fine-grained journey, for example, into a particular settings menu to change an option.

As the user goes through their jobs to be done (JTDB; a methodology for discovering user needs and solving them), they make a great many small journeys. In every case, the user should know that they have begun a journey, that this journey will end at some point, and when it has ended.

The classic anti-pattern here is users thinking, “Have I saved these settings or not?” For example, a rare UX mistake from Apple is that on macOS, changing the settings and then closing the window saves the settings. On the other hand, on (older) Windows applications, the user must press **Save**. In some more obscure systems, the user must click **Apply** and then **Save**: a clearer experience that gives users a sense of confidence.

The user is never sure whether this journey (to change a setting) has ended or not, so make it clear to them.

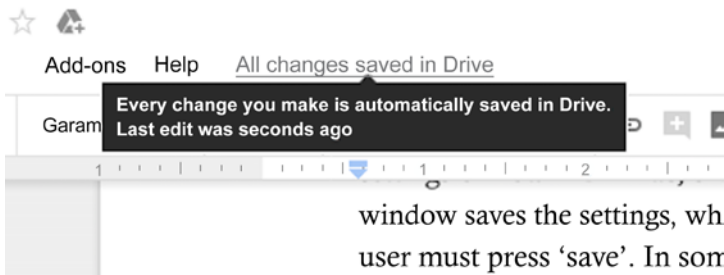


Figure 75.1: Making an edit to a Google doc lets you know that the change has been saved

Each Aspect of a User's Journey Should Have a Beginning and End

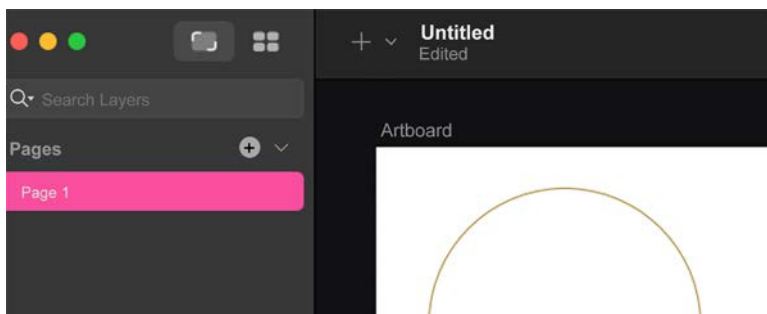


Figure 75.2: Many applications (in this case Sketch) will show “Edited” or an asterisk in the title bar if your work is not saved

Telling your users that they’ve succeeded and completed their task is an easy way to drastically improve their experience. Including these virtual signposts along the journey is relatively easy to do and provides a massive boost to the experience.



Figure 75.3: A great example of showing the user their journey is complete

Keeping users updated with the right amount of clear communication is not an easy task—and every product is different—but it’s worth testing your journeys to ensure that these signposts are visible along the way.

Learning points

- It can help to think of user tasks and journeys as needing signposts along the way
- Keep the user informed as to when they've finished doing the task they're trying to do
- Some examples of end-of-journey signposts include “message sent,” “changes saved,” and “link posted”

#76

**The User Should Always
Know What Stage They
Are at in Any Given
Journey**

Some of the worst experiences in digital products come from not adhering to this principle. For example, a user's profile page that can't be saved because there's another step to complete or the user who abandons their checkout journey because they're not sure how many more steps there will be.

Most users will approach your product with an incomplete (or non-existent) conceptual model of how it functions—this is normal. You need to expose some of this to the user, so they can understand how to use the product.

Although the user will not consciously “know” what stage they're at, at all times they should at least have a general sense of it and you can deliver that experience with some simple cues. Much like landmarks in the real world, your product should include visually different areas that serve as landmarks in the product.

The home screen should look different from the settings screen, for example. Although it sounds obvious, making screens look visually distinct will help the user to think, “I'm back at the home page.” This contributes to an overall feeling of control for the user, reinforcing their mental model.

Some of the ways you can expose the user's stage in a journey include:

- A segmented progress indicator
- A “breadcrumb” control (see #77, *Use Breadcrumb Navigation*)
- An indicator that shows their work has (or hasn't) been saved
- With words—explain to the user what they've done and what comes next

The User Should Always Know What Stage They Are at in Any Given Journey

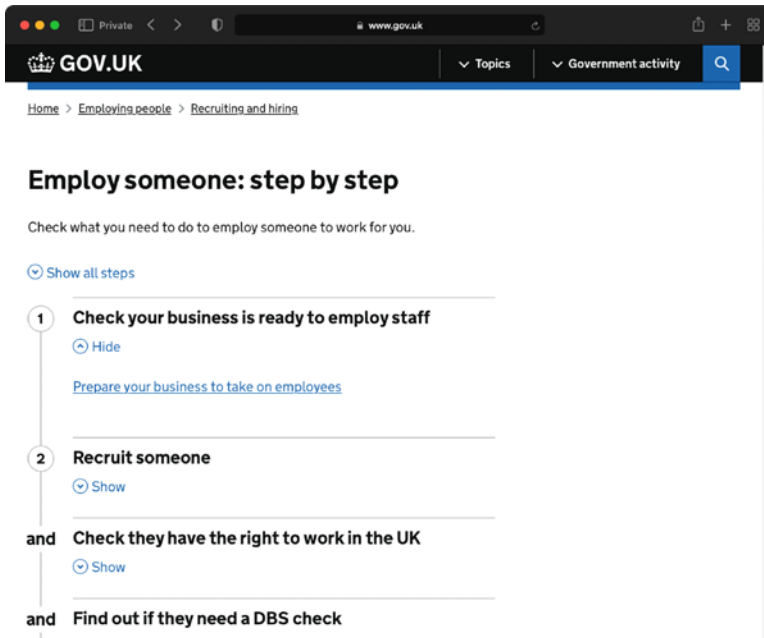


Figure 76.1: GOV.UK features a great “step-by-step” pattern that clearly shows users where they are in a journey

Stop making users feel disorientated: give them some cues—visual or otherwise—to help them to feel their place in each journey.

Learning points

- Think carefully about the journeys in your product, and which parts you must expose to users to orient them
- Tell the user clearly what stage they're at in every journey, for example, with breadcrumbs (see #77, *Use Breadcrumb Navigation*) and clear navigation (see #31, *Split Menu Items Down Into Subsections, so Users Don't Have to Remember Large Lists*)
- Show users what they've done, where they are, and what's left to do in every journey

#77

Use Breadcrumb Navigation

Breadcrumb navigation is not the sexiest of UI components, but it's a long-standing, tried-and-tested control that your users will turn to again and again. An example of breadcrumb navigation is as follows:

Home > Products > Apparel > Hoodies

Websites and apps on the desktop and tablet (and often mobile) can fit a small and unobtrusive breadcrumb into their UI with ease. They are commonly well understood by users in testing and real-world use.

The breadcrumb allows your user to see their position in the system and easily return to a previous level of the hierarchy. What's more, by displaying the path the user took to get to their current location, breadcrumb navigation helps the user to form a better mental model of the layout of the product.

Increasingly, in the world of single-page JavaScript apps, we're seeing breadcrumbs overlooked in the design phase, perhaps because they're seen as boring (they're one of the earliest inventions of web UI).

Overlooking breadcrumb navigation is a huge yet easily preventable mistake for your product's usability. Breadcrumbs also remove the need for a back button in your product, which should always be avoided, as it replicates existing browser functionality in a non-standard, site-by-site way—forcing users to learn *your* version of the back button. If the user wishes to use your breadcrumbs *or* their browser back button, that's fine—both will still work as expected.

It's vital to remember that your job as a UX professional is not to follow trends and remove breadcrumbs just because the control is seen by some as “old school.” Your job is to improve usability, and this is a great way of doing it with very little screen space and at no detriment to users who overlook it.

Learning points

- Use breadcrumb navigation to help your user both move around and understand your product
- Consider whether the breadcrumb is required on mobile—it might not always be needed
- Breadcrumbs are well understood by a wide audience of users

#78

**Users Rarely Care
About Your Company**

There's a running joke in the HBO series *Silicon Valley* about how every tech company wants to make the world a better place. The show's main antagonist, Gavin Belson, goes so far as to say, *"I don't want to live in a world where someone else makes the world a better place better than we do."*

Too many products labor the point: telling their users about their mission or vision, which is about how they're trying to change the world. Please don't do this because users simply don't care. Products are useful for what they let users *do*. This pattern of too much information is a symptom of a lack of objectivity.

If a user has installed your dating app, for example, the chances are that they have some clear goals in mind: some basic "jobs to be done" that involve setting up a profile and meeting people. They don't want a multi-screen onboarding wizard that tells them how your company "brings people together," complete with some stock photos of couples on beaches, holding hands.

When Google launched, it attracted users with its simple UI and high-quality search results. The UI looked like this:



Figure 78.1: Old-school Google

Back then, Google had no "brand" to speak of, a pretty ugly logo, and no real corporate vision or mission statement. Google *did* have a killer feature: better search result relevance than all the competitors, which made it the winner.

As a UX professional, you have to “play nice” with other teams across the business like marketing and sales, but this is one example where championing simplicity over complexity can really improve the experience for users. Once again, objectivity is the most important skill for a UX professional. Put yourself in your users’ shoes.

Learning points

- Don’t overdo the corporate vision in your product
- Users care about what your product lets them do, not what it says it does
- Strive for objectivity in your work

#79

**Follow the Standard
E-Commerce Pattern**

If you're selling items online—physical goods or digital items—then, like it or not, you're in the world of **e-commerce**. The word *e-commerce* already seems hopelessly outdated, but it's the best word we've got to mean “selling things online through a website or app.”

Now, because e-commerce generates revenue for businesses in a very direct way, it was one of the first areas of online experience to really get deep focus and attention from UX professionals. Even marginal gains could increase revenue by significant volumes, so it was worth putting in the effort of user testing and A/B trials.

The e-commerce pattern we've arrived at from the past 25 years of the consumer web is both well-refined and well-understood by users. Getting a customer through a purchase funnel is difficult, with many opportunities for “cart abandonment,” which means that it has to be as frictionless as possible, so make everything as familiar as possible.

It goes a little something like this:

- **Products:** Products are listed in categories, with attributes like price, size, color, pattern, and so on. Users can search and sort these products by their attributes. Viewing a product shows controls to adjust the size, color, and so on—if these options are available—as well as a quantity selector.
- **An add to basket button:** This adds the quantity selected to the basket. Depending on the type of item, the user may be prompted to go straight to the checkout (if they're only expected to buy one thing).
- **The basket:** The basket or cart shows the user the items and the quantity they've selected. From there, they can modify quantities, remove items, clear the basket, or proceed to check out.
- **The checkout:** The user is shown the total and asked to enter personal details, such as delivery address and payment information. If they have an account, they can optionally sign in at this stage (to avoid entering details over and over), but you should allow “guest checkout” where possible.

- **A confirmation:** The user needs to see that yes, their payment was successful, and yes, the order is complete. They can close the tab now and wait excitedly for their product to arrive.

That's all there is to it! This is the tried-and-tested pattern that's sold billions of items over the years. You'd be crazy to mess with it—you risk alienating users and reducing revenue in a very direct way.

There are, of course, a hundred ways you can enhance this journey—this is the bare minimum that users want and expect when buying online. Additional enhancements to consider are generally around giving the user confidence in the product and the company; warranty details, customer reviews, 3D tours or high-resolution images, and so on.

Finally, e-commerce is one of the functions where you really don't have to conduct a “build versus buy” exploration: for most people, out-of-the-box e-commerce patterns will work very well.

Learning points

- Every way that you can reduce friction in the purchasing funnel will increase conversions
- Users expect your store to work like every other store they've used
- Don't mess with the pattern of products, a shopping basket, and a checkout

#80

**Show an Indicator If the
User's Work Is Unsaved**

If possible, your app should be “autosaving” the user’s work, but there are, of course, cases where this needs to be a user-initiated action (for example, in an application where saving could be destructive—like writing a really long document or editing a photo).

A great way of showing the user that their work is unsaved is by displaying a visual indicator in the title bar of the app. This could be a bullet or could even explicitly say “not saved,” if space allows.

At a glance, the user can tell if they need to quickly hit *Cmd + S* (or *Ctrl + S*) to save where they’re at or if they’re just experimenting, they will know that they haven’t saved.

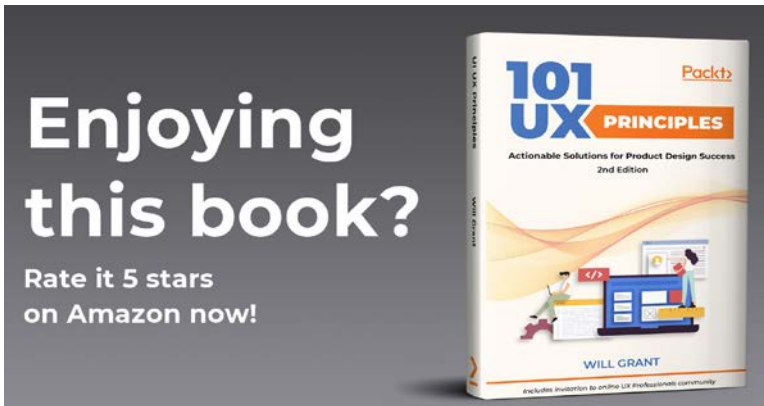
A big part of this is about respecting the time and effort that the user has put into using your product: entering data, preparing a profile or bio, and so on. They deserve to be shown the state that their work is in and not have to guess or remember whether it’s saved.

Learning points

- Show the user whether their work has been saved or not
- Consider whether autosaving a user’s work is helpful or not for your product
- Show your user that you respect the time and effort they’ve put into using your product (see #43, *Respect Users’ Time and Effort in Your Forms*)

#81

**Let Users Give
Feedback, but
Don't Hassle Them**



See how annoying that is? Yet apps do it all the time, interrupting the users' flow to nag them for a review on the relevant app store. App publishers care a lot about reviews because they build trust in a product, and app stores will rank software more highly if it has attracted a lot of great reviews. So it's easy to understand their relentless drive to ask users to rate every app.

It's the digital equivalent of wandering around a department store (remember those?) and having an eager salesperson ask "Can I help you?" within 30 seconds of you walking in.

It's counterproductive because this interruption just annoys users and leads to lower levels of user satisfaction for your product. Much better to just offer a "Give feedback" or "Tell us what you think" control in a help section of your UI.

Some of the best feedback experiences on the web or in mobile apps are as follows:

- Have an easy-to-locate control to give feedback
- Allow the user to fill in some free-form text on the problem they're having or the feedback they wish to leave—not picking from a convoluted list of possible problems

Let Users Give Feedback, but Don't Hassle Them

- Include optional diagnostic information with the feedback (for example, browser type or app version)

There are a great many useful feedback tools (free and paid) that can be customized and added to your product, like GetFeedback, InMoment, or Qualtrix. Go ahead and search for “product feedback tool” to see your options.

Nagging users into giving feedback only serves to annoy your faithful customers and drive “on the fence” customers further away.

Learning points

- Offer users an easy way to give feedback
- Allow them to give feedback in their own words, with free text
- Don't show a “rate this app” modal

#82

**Don't Use a Vanity
Splash Screen**

The splash screen—the full-screen graphic that appears when your user opens your iOS or Android app—is a great place for your company logo, brand messaging, or corporate vision statement, right?

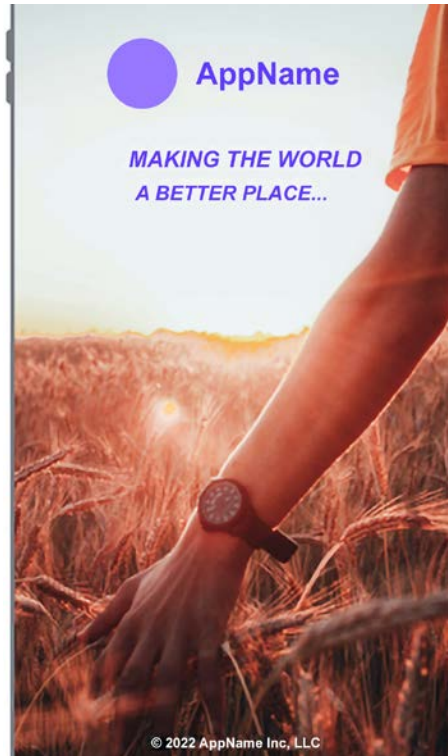


Figure 82.1: Splash screen (Photo by Artem on Unsplash)

No. Do not do this.

Users do not care about how you're making the world a better place—they just want to open the app to do whatever it is the app does, and you're just slowing them down by a few seconds every time.

Don't Use a Vanity Splash Screen

Instead, look at the first screen of your app and offer a splash screen that echoes this layout, but without content. This is called a “skeleton screen” or “content skeleton,” as seen below:

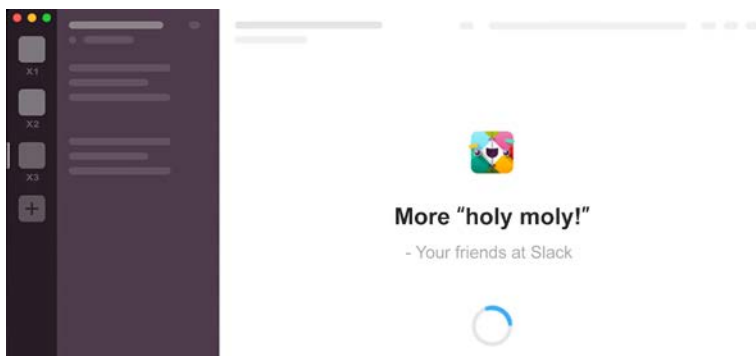


Figure 82.2: Slack makes great use of a content skeleton to get users straight into the product while it loads, rather than showing a vanity splash screen

Users will feel like the app is loading quicker if they see the expected interface and it then transitions into the “real” interface.

Load the UI quickly, and if that means some user interactions aren't ready yet, only show the user a spinner if they click them. For example, in a word processor, let them start typing as soon as the app opens and load in the pretty “add a chart” dialog later, if the user clicks it. If there is a need for a dedicated login screen before the user sees anything else, some of this branding can be done there.

Learning points

- Don't show the user company information on a splash screen
- Help the user get into your product as quickly as possible
- Put the user's needs first, not your corporation's needs

#83

**Make Your Favicon
Distinctive**

A favicon, app icon, or apple-touch-icon—whatever you like to call it—is an icon you may well have forgotten to add to your web app. It serves a useful purpose beyond branding, appearing in the browser’s tab UI and in bookmark views across your devices.

Users with lots of tabs open, lots of apps in their start menu, or lots of apps in a folder on their phone, will appreciate being able to find yours quickly.

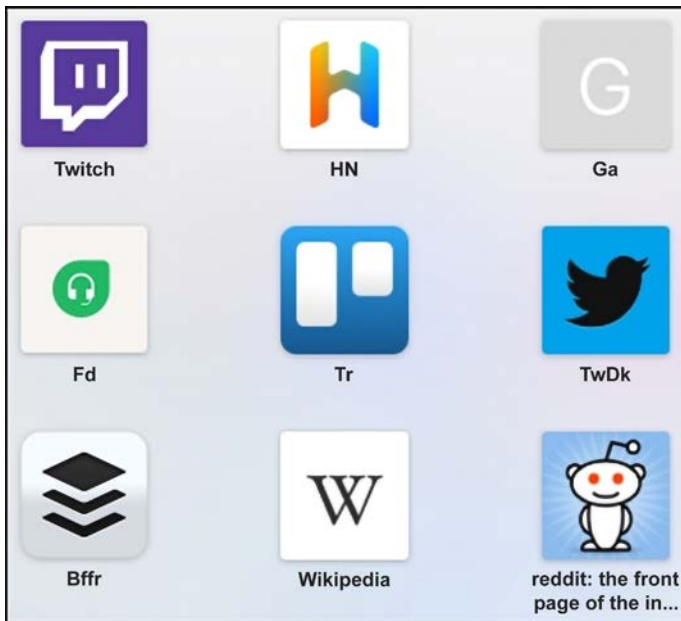


Figure 83.1: Some favicons

A bright, bold icon or letter is usually sufficient, but test it at 16 pixels in size to see that it’s legible. Use transparency because, unless your icon actually is a white square, nobody wants a white square in their tab bar!

You can cover most browsers out there with just four types of favicon, containing just six favicon files:

1. A `favicon.ico` for legacy browsers at 32px square
2. A single SVG icon with a light and dark version for modern browsers
3. A 180px PNG for Apple devices
4. A webapp manifest with 192 and 512px PNG icons

Typing any of these into your favorite search engine will give you more detailed instructions which go beyond the scope of this book.

If users can see your app instantly and switch to it, you'll be saving thousands of hours of people's time cumulatively. Good job!

Learning points

- Make your favicon clear and distinctive
- Users use favicons to identify tabs, favorites, and more
- Favicons can be displayed as small as 16 pixels, so check them at that size

#84

Add a “Create From Existing” Flow

An often-overlooked flow in many applications is the “create from existing” flow, and the ability to create templates. Whatever your user is dealing with in the app—customers, orders, or really any repetitive task—you should offer a “create from existing” flow if possible.

When given a chance to re-use work that they have meticulously created, this simple flow is a massive time saver and productivity boost for the user.

Selecting “create from existing,” “duplicate and edit,” or even “duplicate,” should make the product behave something like this:

- The system copies the item, giving it a new ID
- The user is presented with the edit view, but with a new name (perhaps with “copy” appended to the original title)
- The fields are pre-filled with the data from the original item
- The user can change as much or as little as they wish and click “save”

An extension to this could be “save as template” for work that the user will come back to again and again. Templates also have the benefit of being a great onboarding resource for new users—Canva, Miro, and Figma all offer great templates to orient new users.

This flow is useful anywhere that a user is adding items or maintaining a list of items. It’s fairly typical for business-to-business applications (for example, customer records, orders, and so on) to be comprised of a lot of detailed records. The flow also has a place in consumer-focused apps—duplicating a slide in a presentation tool, and then editing the contents, is a well-used pattern.

Learning points

- Allow users to create a copy of an existing item in the system
- Don’t force them to re-enter the same details every time
- Allow users to create, edit, and re-use their work as templates

#85

**Make It Easy for Users
to Pay You**

The routes by which products are paid for are many and varied, but—even if you're not selling tangible items—there is often the need for a product to ask a user to upgrade and enter some payment details.

Time and again, these interactions fall short of top-quality user experience. Be they complex credit card forms, asking for too much information on lengthy order forms, or unclear pricing plan details, they represent a massive missed opportunity.

To some extent, this is a solved problem in mobile apps—both iOS and Android include extensive support for in-app purchases and subscriptions, allowing designers to use native input systems rather than clunky custom ones. The user likely has their payment details saved and it's often a one-tap action to make a purchase.

The screenshot displays a web browser window at the URL meowmeowmeow.com. The page is a checkout interface for a product named "Juniper Bergamot Face Cleanser" priced at \$51.00. The layout is clean and organized into two main columns. The left column contains the checkout steps: "Cart", "Information", "Shipping", and "Payment". Under "Information", there are three payment options: "shopify", "Apple Pay", and "PayPal". Below these, there is a section for "Contact information" with a text input for "Email or mobile phone number" and a checkbox for "Email me with news and offers". The "Shipping address" section includes fields for "First name", "Last name", "Address", "City", "Country/region" (set to "United States"), "State" (set to "State"), and "ZIP code". At the bottom of the left column are two buttons: "Continue to shipping" and "Return to cart". The right column shows the product details, a "Gift card or discount code" input field with an "Apply" button, and a summary of costs: "Subtotal" (\$51.00), "Shipping" (calculated at next step), and "Total" (USD \$51.00).

Figure 85.1: The Shopify checkout experience is well-tested and almost perfect

Out on the web, however, it's a different story. Although popular online stores like Shopify have helped to standardize this to some extent, it's often far too confusing and over-complicated in many products.

First up, there are pricing pages. Many pricing pages make it hard for the user to understand the various plans, subscriptions, and add-on bundles. That's if the site even *has* a pricing page. Try to apply general UX principles to pricing pages:

- Overly long lists of features and benefits are hard to parse in the user's mind, so keep them short, or consider a comparison format that lets customers see the benefits of higher-priced options.
- Make the “buy” button obvious with visual affordances.
- Keep your pricing structure simple, for example, a few plans, with different features and at different price points. Avoid weird and wonderful add-ons and extras that are hard for users to calculate the value of.

In many organizations, your very salary depends on the success of pricing and payments, so take care to apply the rest of the principles in this book to pricing pages.

Next, let's look at order forms. Make them simple, don't ask for unnecessary information, and give the user control over what they're buying, for example, the ability to adjust quantities of items, change colors or styles, or easily remove items.

Finally, make your payment form usable. The user flow of learning about pricing, ordering a product or service, and paying should be treated as one of your most important features. It's essential to your commercial survival.

Your user loves your product so much that they want to pay you, so make it as easy as you can—use standard, accepted patterns where possible, and test this flow regularly.

Learning points

- Make payment and ordering pages as easy to use as possible
- Don't hide your pricing page away, and make it simple and clear
- Test your payment flow regularly

#86

**Give Users the Ability
to Filter Search Results**

In the early days of web interfaces, most search systems worked like this:

1. Enter a search term, such as *red shoes*
2. The system would return 100,000 pairs of red shoes
3. The user would try again: *red men's shoes in size 8*
4. The system would try an **AND** match on all the terms
5. The results page would show no results
6. The user would have to work out how to change the query to get the results they want

Some enterprise and business systems *still* work this way, much to the frustration of their long-suffering users. It's a symptom of interface design mapped directly onto the engineering requirements: this is how databases are queried, but it's not how human minds work.

Over the past 10 years, pioneered by e-commerce sites, searching has gotten a lot better—a modern example looks like this:

1. Enter a search term, such as *red shoes*
2. The system would return 100,000 pairs of red shoes
3. A filter system would let the user filter those results
4. The user chooses *Men* and *Size 8* to get the results they want

Give Users the Ability to Filter Search Results

For example, the following screenshot shows the process of searching for *wardrobe* at Habitat, a British retailer. The site suggests *1 door*, *2 door*, and so on, and includes a range of filters on the left-hand side:

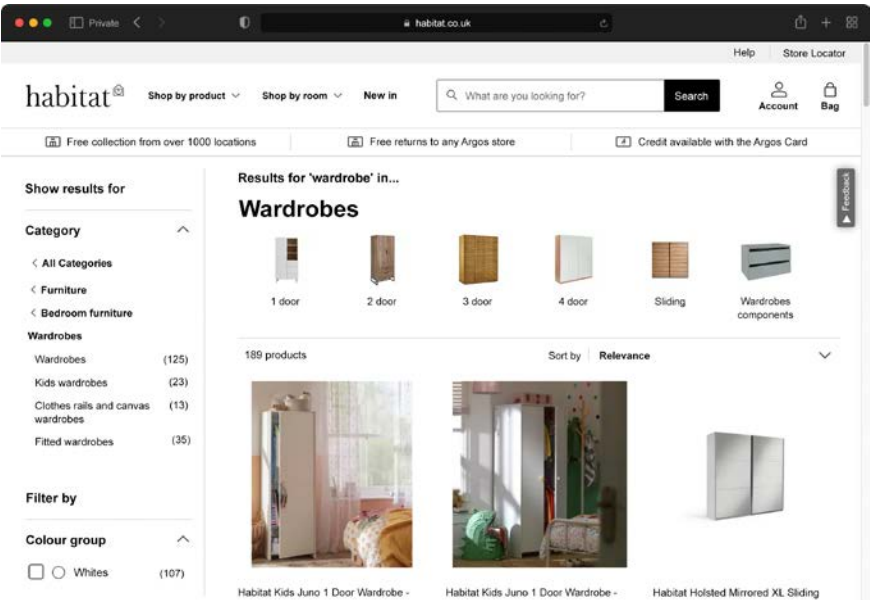


Figure 86.1: Filtering search results at Habitat

This leap in usability came from realizing that we don't always know what we want until we see the options in front of us. Only then, in the context of our results, can we see the next step to find what we're looking for.

Away from e-commerce, this pattern can be applied to a huge range of other software interfaces, for example:

- Find items near me, then filter by *used* or *new*
- List all my servers, then filter by just the Linux ones
- Show me all flights to Lisbon, then just the direct ones

Try to avoid making users play a game of *press and guess*—let them query with broad terms and then refine later.

Learning points

- Allow users to enter broad search queries and then refine them later
- Don't force users to guess the right query the first time
- Offer customized filters based on the *kind of thing* the user is looking for

#87

**Your Users Probably
Don't Understand the
Filesystem**

The filesystem of your computer is the complex tree of many, many thousands of folders and files that make up the operating system—all your apps and their resources, and all your documents, images, and music files. Your users likely don't understand this—nor should they have to.

I've witnessed people use Microsoft Word as the primary way to find and retrieve information from their computer or network. They'll open Word and use the "open" command as a way to browse around their documents. If they come across an image, they'll open it in a Word document. This probably sounds insane to most computer-literate people. But this makes perfect sense to users who mostly write and manage Word documents. These people aren't stupid—they just don't understand how files are stored on their computers.

One of the reasons why smartphones and tablets became ubiquitous is the fact that there's no way to see the files. On an iPad, you have apps and "documents" within those apps. Open an app and your documents *for that app* are in there.

Advanced users can dig around and access files if they really want to, but there's no easy way to accidentally delete an important system file and break the device.

Think about users' mental models of your products and how you store their information. When a user approaches your product fresh, they have to form a mental model of how it saves and retrieves their information. Does it save their files in the app? Do they need to download their work? If they start a task on their phone, can they continue it on their desktop?

Make this clear to people by:

- Offering an onboarding wizard to orient them
- Auto-saving their work for them
- Alerting them to destructive actions ("Do you want to save this?")
- Respect users' desire to just get shit done and not have to worry about how you're saving their work

Learning points

- Users don't and shouldn't have to understand the filesystem of their device
- Make it clear to users how and where their work is saved
- Use this principle to consider what complexity you can helpfully hide from your users to improve their experience

#88

Show, Don't Tell

The expression “show, don’t tell” comes from screenwriting and fiction. Often attributed to playwright Anton Chekhov, the technique is intended to allow the reader to experience the story through action, words, senses, and feelings, rather than through the author’s exposition and description.

Show the viewer (or user) the situation and let them approach it in their own way. It’s also a great mantra to repeat to yourself if you’re working on the experience of onboarding, feature guides, or other tuition—*showing* users how to use your product is always better than *telling* them.

The first reason for this is that *users don’t read text*. Really, they don’t. Time and again, in user test after user test, I’ve witnessed this with my own eyes—users simply don’t read blocks of onscreen copy. You have to show them how to use the product, not write a description using prose.

Onscreen tips are a good starting point. The tips should be easily dismissible for repeat users (perhaps this isn’t their first installation of this app), but present for new users. These tips can highlight areas of the app that allow the user to get started. Once they’ve been shown the key areas of the interface, you can leave them alone to discover more for themselves.

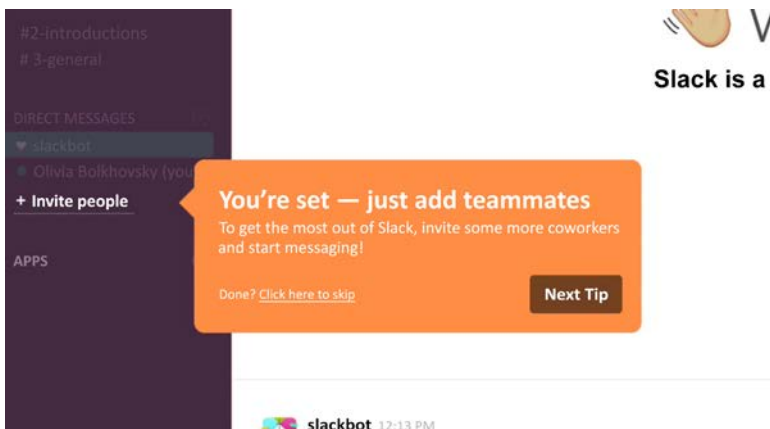


Figure 88.1: Rather than using prose, Slack gives users a big one-line description of key features, along with additional, less important text beneath

A video demo is best used for more complicated or highly specialized products. It's more intrusive and laborious to sit through, so please allow your veteran returning users to skip it. The benefits, of course, are more detailed and specific instructions on how to operate a more complicated UI. This technique is used to great effect in professional software like video editors, graphics tools, and music software. Consumer products shouldn't need this.

A final way to make your “show, don't tell” approach more effective is to build upon established products (see #98, *Give Users the Experience They Expect*). The chances are that your user has seen and experienced products like yours, so they can apply this experience to your product and be off to a flying start in no time.

Learning points

- Users seldom read text, so show them what you mean
- Video demos are great for complex software and UI
- Allow returning users to skip these demos, but also access them again if they need to

Terminology

The words you write in your product have huge power—to guide and comfort users, or if written poorly, confuse and alienate them. This section includes shortcuts and best practices to take your in-app copywriting to the next level.

#89

**Be Consistent
with Terminology**

The words (or copy) that you write in your product have a dual purpose. The first is the most obvious: they label items and views and tell the user which elements are which.

The second is less obvious, but more important: the words you use become a very precise and descriptive *language* for your product. Understanding and parsing this language is essential to a user forming a mental model of how your product works.

- If you call your e-commerce shopping cart a “cart,” then call it “cart” everywhere
- If you call your user’s profile page “profile,” then call it “profile” everywhere
- If you call your user’s email settings “email settings,” then call them “email settings” everywhere

Mix these up and it will take your user longer to ponder the inconsistent terms and work out what you mean.

Think also about the tone of voice you use in your product—for example, very formal error messages might appear jarring or unwelcome in a children’s mobile game. On the other hand, if you’re applying for a passport online, you want very clear, precise error messages with no frivolity.

The words in your UI have a lot more meaning than many people realize. Obsess over the right terminology to use and you’ll help make users happier.

Learning points

- Use consistent terms across your product
- Don’t just label things as you go—build a consistent language for your product
- Help users to form a mental model more quickly with consistent copy

#90

**Use "Sign In" and
"Sign Out", Not "Log In"
and "Log Out"**

Everyone has signed in to attend a meeting or visit a doctor or dentist. Signing in is something that people do in the real world. Nobody alive today has ever “logged in” in the real world. The term comes from the ship’s log, where the sailor would log in their times and the distance traveled that day. It’s highly unlikely that your users are 18th-century seafarers!

Despite this, it’s pretty common to see “Log in” (or, even worse, “Logon”) in software, and especially in business-to-business software that’s been designed by developers.

For reasons of familiarity, always use “Sign in” and “Sign out” in your product consistently: they relate back to the real world, unless your product is a mobile app for time-traveling pirates, of course.

Learning points

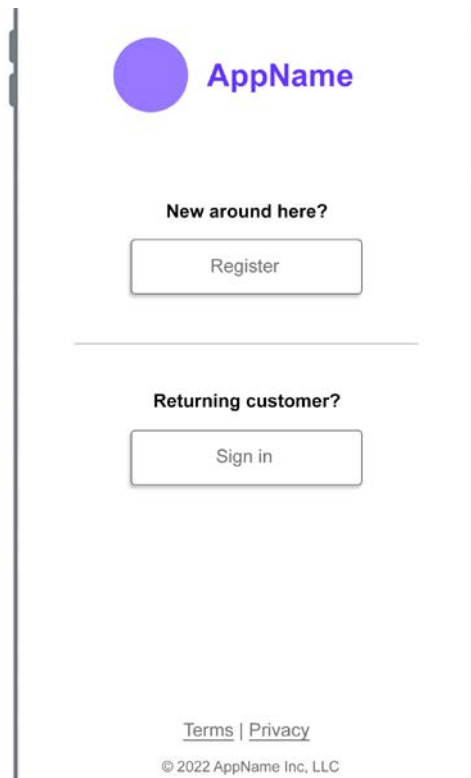
- Use “Sign in” and “Sign out” in your product
- Relate tasks like this to real-world situations for familiarity
- In particular, try to avoid the dreaded “Logon”

#91

**Make It Clear to Users
If They're Joining or
Signing In**

The principles in this section discuss the confusion of terminology, and in 2022, as I write this, there are few signs that these terms are beginning to standardize. Apps and websites still feature a confusing mix of “Log in,” “Sign in,” “Login,” and so on.

But it’s more than that—the interaction flow of your product should make it clear to users whether they’re joining (for the first time) or signing in (as a repeat user). Seldom-used products will often require a sign-in when you come back to them, and more secure applications need a sign-in every time—often with two-factor authentication. So, let’s make it easy for users to get back into your product.



*Figure 91.1: This wireframe shows a clear way to solve this problem—
why doesn't every app do this?*

A theory for this “sign-in obfuscation” is that aggressive conversion optimization often hides the sign-in option in favor of devoting the whole screen real-estate to sign-up enticements. This, of course, makes sense on one level: the user has gone to the trouble of downloading the app, and the obvious next step is to generate an account.

Over-playing this, however, is short-sighted. It just frustrates and alienates your loyal repeat customers. It might be great for the short-term sign-up metrics, but retention is more important.

Learning points

- “Sign up” and “Register” are clear and unambiguous; give users these two options
- Consider some extra hint text to disambiguate the options
- Don't take over the whole view with “Register” only

#92

**Standardize the
Password Reset
Experience**

Password resets are a frequently used part of the sign-in experience. Users will make mistakes and, as UX professionals, it's our job to help them out as best we can.

When it comes to passwords, unless you're using a password manager, you've either got a password that's way too easy to remember (and way too easy to guess), or you've forgotten your password.

Allowing users to reset their password with an email or text message is a useful pattern, and it's so well-known that it should be standardized by now. Even so, there are plenty of examples around the web and in mobile apps where unusual terminology or UI makes it unnecessarily hard to reset a password and get back into your account.

Call the control **Forgot password** and not **Reset your password**, **Can't access your account?**, or **Get a reset link**. Most users won't necessarily understand that this is the route for their most common use case: they've forgotten their password.

If they've already entered their email address or username, pre-fill the **Forgot password** form with that email or username—don't make them enter it all over again.

They should get (in their email or by SMS) a link that:

- Takes them to a page to set a new password
- Doesn't expire after one click (users double-click things frequently!)
- Does expire after a sensible time period
- Expires when the password has been successfully reset

This will ensure that you can strike a good balance between usability and security—a common trade-off we often have to make as designers.

Finally, consider allowing longer-lived sessions on your product: don't sign them out so often. Once the user has signed in, their cookie or session token is stored securely in the browser or mobile device. If the device is lost or stolen, they likely have a PIN code or password on the device.

Automatically signing people out after a short time (30 minutes for an enterprise app or a couple of days for a mobile app) is bad for usability: it means the user has to sign in more frequently. This repeated signing-in creates more hassle for them, a poorer experience, and, as a result, they choose an easier password, making their account *less* secure.

Learning points

- Use **Forgot password** so the user knows that this is the function to solve that problem
- If the user has already entered their username, pre-fill it and don't ask for it again
- Consider allowing longer-lived sign-in sessions on your product

#93

**Write Like a Human
Being**

Too often, terminology in software is written from a systems-oriented or organization-centric point of view. Consider a CRM or other business system: we often see menu options like **Edit customer** or **Create new customer**, but stop and think about this for a second—customers are people and we don’t create them. The first option doesn’t actually *edit a customer* and the second doesn’t *create a new customer*.

For the developer, customers are just database records, so of course it makes sense to edit them and create new ones, but for the users of these systems, these options should be better named: **Edit a customer’s details** and **Add a new customer**.

This principle is best achieved through objectivity and empathy. In other words, being able to step outside of your view of a product and see it through a customer’s eyes. You must take this step to build usable software and it’s worth the effort.

The words that you use for in-product copy, for menu controls, and even for marketing materials, have power and weight: you can use them to welcome or alienate people, to set them on the right path, or to confuse and bewilder them. Try to write from a user-centric point of view. Put effort into your writing and you’ll build products that people love to use.

Learning points

- Write from a user-centric, and not an organization-centric, point of view
- Don’t let “corporate speak” or internal business language creep into your product
- Consider how the words you use can affect people’s perceptions of your product, and try quick “guerilla” tests to see how they feel about your words

#94

**Choose Active
Verbs over Passive**

Most of this book is concerned with using visual design to improve the user's experience. However, the *words* that we use as designers also have a huge impact on the usability of the products we create.

10 years ago, I found myself on a half-day course by the Plain English Campaign (the body behind the “Crystal Mark,” for documents that are easy to read and understand). There were a lot of great tips on the course, but the section on the active and passive voice really stuck with me:



A verb is in the passive voice when the subject of the sentence is acted on by the verb. For example, in ‘the ball was thrown by the pitcher’, the ball (the subject) receives the action of the verb, and ‘was thrown’ is in the passive voice. The same sentence cast in the active voice would be, ‘The pitcher threw the ball.’

– Dictionary.com (definition of the active (<https://www.dictionary.com/browse/active-voice>) and passive (<http://www.dictionary.com/browse/passive-voice>) voice)

Now, because the active voice is more direct, it requires fewer mental steps for the user to unpack the meaning. In UX, this translates into interfaces that can be used and understood faster. Switching your copy to the active voice can make it sound less stuffy and bureaucratic—and users will appreciate this simplicity. Consider the following sentences:

- *This matter will be considered by us shortly* (passive verb)
- *We will consider this matter shortly* (active verb)

The active voice is crisper and uses fewer words. When applied to the field of software design, we can make onscreen copy much easier to read.

The following are some examples:

- *In order to apply updates, your computer must be restarted* is passive. Compare that to the clearer and more punchy alternative, *Restart your computer to apply updates*.
- *The “search” button should be clicked once you have entered search terms* could be replaced with the much simpler *Enter search terms and click “search”*

It’s possibly because most software is designed within large, bureaucratic organizations that this language creeps in over time. The passive voice is often associated with sounding more officious or formal when, in reality, it just sounds pompous and confusing.

As products evolve, more and more stakeholders inevitably weigh in to have their say: branding want the copy to reflect the brand values, legal want it to be factually accurate and watertight, the growth hackers want to stuff keywords in, and so on. What users are then presented with is a watered-down, passive version of the original idea, which is overly complex and indirect.

The passive voice makes your interfaces slower to use and harder to understand, so root it out and destroy it.

Learning points

- Choose the active voice over the passive voice for in-app copy
- Continually review your copy and labels to ensure they still make sense
- Test phrases on real users and work out which gets the best results

Expectations

Like it or not, your users come to your product with a set of expectations in their minds already. In this section we'll discuss how to deliver world-class user experience while still putting customers at ease with familiar experiences.

#95

**Search Results
Pages Should Show the
Most Relevant Result at
the Top of the Page**

Of all the principles in this guide, this might be the number one no-brainer. Of *course*, show the user the most relevant results first. Yet, time and again, this principle is broken and users are shown irrelevant items first in their results. So, why have you asked the user to search, then shown them a poor set of results?

Reason 1: Your search algorithm sucks.

Technically, this is the toughest one to solve. Ranking search results is, in some cases, a tricky technical problem, but there are tried and tested technologies (term frequency-inverse document frequency, or TF-IDF, is a very popular algorithm for ranking text documents, for example) and a lot of off-the-shelf search tools will include some sensible defaults.

It's a difficult task to make your search perform as well as Google's search does—but that's what users expect. Users don't understand that 1,000 years of cumulative effort has gone into Google's ranking algorithm; they expect your site to rank results just as effectively. Test searches, pore over your site analytics, see what the most popular search terms are, and make damn sure that those results are relevant.

Reason 2: Your filter defaults are bad.

Maybe your results are coming back from the database in a decent ranking, but you're applying some poorly chosen filters to them—for example, if a user is searching an auction site for items, only to be shown the closest first. It might have seemed like a good idea—because you have the user's location—but if they're getting the item shipped, then it's not relevant and there may be a better, cheaper item further down the list. Pick sensible defaults and show the user which ones you've picked, allowing them to change them at will (see #96, *Pick Good Defaults*).

Search Results Pages Should Show the Most Relevant Result at the Top of the Page

Reason 3: You're trying to sell the user something that they don't want.

A more sinister reason is that many sites will show you the items *they want you to see*, rather than the items that you want to see. This serves nothing but the internal needs of the organization. It's a surefire way to enrage users, so don't do it. You might sell a few more car rentals, but at the expense of irritating most of your customers. Allow users to filter their results how *they* wish (see #86, *Give Users the Ability to Filter Search Results*).

Learning points

- Show users the most relevant results at the top of a search results page
- Give users clear controls to modify the results with sort order and filters
- Think like your users—what results would it be *best for your users* to see first?

#96

Pick Good Defaults

The power of default settings is often overlooked, but they have huge potential to affect the UX of your product. Some examples of great defaults:

- When I get into my car, my phone detects that it's connected to a Bluetooth vehicle and the sound output switches from headset to car—a great default behavior.
- When signing in to an analytics product with a selected date range of “this week,” with a comparison date range of “last week.” Imagine if the default was “today” and showed no data—useless, right?
- When I tap a name in my “recent calls” view, my phone calls that person, rather than starting a new text message or video call. Those options are tucked away in a context menu, if I need them.

Picking a good default is a balance of factors:

- How many users you think (or know through research) want *this* default setting
- How difficult it is for the user to change to an alternative
- How discoverable that alternative setting is

As a UX professional, it's your job to weigh up these factors and a lot of that judgment will be based on gut feeling, as well as evidence. There's a temptation to expose a new feature or functionality—just because it's new—and make it the default setting. Don't do this. Your users don't care about something because it's new; they care whether it's useful or not.

How many times have you heard users complain, “They've updated the app and now it makes you do X”? If X was an *option*, rather than the new *default*, you'd have happier users. Through the use of user testing, A/B testing, and analytics research, it should be possible to identify common user journeys and optimize the defaults for the vast majority of users. These kinds of studies often yield results that follow the 80/20 rule, or **Pareto principle**: optimizing the top 20% of user journeys can have a positive impact on 80% of your users.

A final thought—be aware that the vast majority of users don't venture into settings menus and will simply use a product with its default setup. For the bulk of your users, the default setting is the *only* setting, so choose well.

Learning points

- Think carefully about the default settings you choose
- Most users will never change from the defaults
- Balance the factors of discoverability and frequency of use when deciding on defaults

#97

**Only Use Modal
Views for Blocking
Actions**

A modal view—like a dialog box, or full-screen takeover—is just that: modal. The user is forced into a mode where they have to make a decision, and they can do nothing other than deal with the modal.

It breaks the user's flow, and that can be a good thing for some actions—"are you sure you want to permanently delete all your stuff?" is a great example. But too often modals are thrown around with abandon and your customer needs to stop and deal with each one.

Dealing with the modal isn't a zero-effort task; the user must:

1. Switch out of the context they were working in
2. Read the modal text
3. Work out what to do, and how it will affect their current goal in the product
4. Operate the modal

This extra cognitive load adds up, and too many modals (where there's no need for them) will just infuriate your customers. Breaking a user's flow unintentionally is often the cause of users feeling frustrated and will also make your app come across as slow or flaky.

Using a modal view for destructive actions (like the **permanently delete** example above) is ideal—the user must make a yes or no decision, and they can't proceed without choosing an option. A good rule of thumb is to ask yourself, "can the user continue without responding to this?" If they can—don't use a modal.

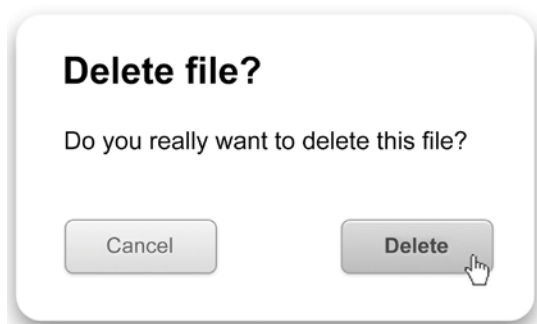


Figure 97.1: A good modal—the user needs to make a decision on this before anything else can happen

A common mistake is to give less important events too much significance—a good example is receiving a new message in a shopping app, where your browsing is interrupted by the app telling you “You received a new message!” with a modal that must be dismissed:

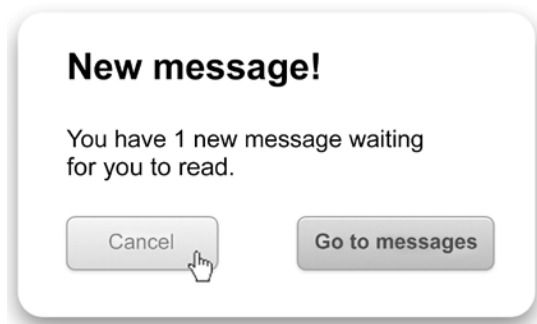


Figure 97.2: There’s no reason for this to be a modal—it interrupts the user needlessly

The app already has a **Messages** section, which can show me notifications as a numeric bubble—I don’t need a modal to tell me about something I *may not* want to view right now.

Learning points

- Only use modal dialogs or full-screen alerts when you absolutely need to
- Remember that obscuring content will break the users' flow—especially so on a mobile where modals are often shown fullscreen
- There are still occasions for modal views, but choose wisely

#98

**Give Users the
Experience They Expect**

Over the years, I've found that part of the imagined code of practice of designers is to not steal. As we train and learn, we're taught to develop our own design style and not to borrow too much. Imitation is discouraged and copying the designs of others is frowned upon, dishonest even.

In UX, this is the polar opposite of best practice. Consider Jakob Nielsen's Law of Internet User Experience (<https://www.nngroup.com/videos/jakobs-law-internet-ux/>), which states:



Users spend most of their time on other sites. This means that users prefer your site to work the same way as all the other sites they already know.

Jakob Nielsen utterly nails it with this one. Your users spend the vast majority of their lives *not using your product*. They spend that time on other sites, other web apps, and other mobile apps. The product with which they're *least familiar* is *your* product.

You should aim to build upon established patterns:

- Forms that allow simple data entry, easy movement between fields, and a **Submit** or **Save** button
- Toggle controls that adjust a setting to be either on or off
- Pages or views in your product that tell users how much the product costs, in total, without hidden fees
- Obvious controls, links that look like links, and buttons that resemble buttons
- Search that works quickly and shows the most relevant items first

Give Users the Experience They Expect

Your users have spent *many years* using products *just like yours*, so should your product work *just like* those other products or radically differently? The answer is, *just like those other products*:



Figure 98.1: iOS, Android and Tizen. There's a reason that products look similar

It's not exciting or sexy—you're not inventing a whole new class of product or interface, and you're not revolutionizing a whole product sector. What you are doing is the good work of a UX professional: building on the established practices that users know and love from years of experience.

Your satisfaction comes not from reinventing the wheel but from giving the user a wheel that they already know how to use.

This will give them the tools to get their jobs done and improve their life just a little bit.

Consider “benchmarking” competitors’ products or other similar products in your field to get a sense of their experience and usability—you’ll see where they do things well and how you can put users at ease by offering them a familiar experience.

Make your product behave like all the other products your customer already knows how to use—I don’t mean “copying and pasting” your competitors, but I do mean giving users experiences based on familiarity. Much of this book, although based on practice and real-world experience, focuses on collecting and distilling the best practices of *what’s already out there*.

Learning points

- Don’t be shy about borrowing best practices from other products
- Users want your product to work like products they already know and use
- Build upon established patterns to achieve this

Join us on Discord!

Read this book alongside other users, UX professionals, and author Will Grant.

Ask questions, contribute to discussions with other readers, chat with Will via *Ask Me Anything* sessions and much more.

Scan the QR code or visit the link to join the **UX Squad** community.



<https://packt.link/101uxprinciples>

#99

**Decide Whether an
Interaction Should
Be Obvious, Easy, or
Possible**

While we strive to make our products as intuitive and familiar as possible, there will always be “advanced” options and rarely used features (see #32, *Categorize Settings in an Accessible Way*). Giving users choice and control over their experience will naturally lead to features that are used less frequently or settings that only a small percentage of users will change.

To help decide where (and how prominently) a control or interaction should be placed, it’s useful to classify interactions into one of three types:

- **Obvious:** Obvious interactions are the core function of an app, for example, the shutter button on a camera app or the new event button on a calendar app. They’re the functions that users will likely perform every time they use your product and their controls should be visible and intuitive. Hiding these away—either accidentally or intentionally—does still happen and it’s often a cause of massive frustration for users and the failure of new products.
- **Easy:** Easy interactions are the hardest to classify and often we’ll only get these right after several rounds of iteration and user feedback. For example, an easy interaction could be switching between the front-facing and rear-facing lens in a camera app, or editing an existing event in a calendar app. The controls should be easily found, perhaps in a menu or as a secondary-level item in the main controls. They’re the toughest to get right because they’re used too frequently to be tucked away, but they are not used every time, which means designers will often de-prioritize them too heavily.
- **Possible:** Interactions we classify as possible are rarely used and they are often advanced features. They need to be discoverable, but they shouldn’t be given the same prominence as obvious or easy interactions. For example, it is possible to adjust the white balance or auto-focus on a camera app, or make an event recurring on a calendar app. These advanced controls can be tucked further away, as the majority of users will not need to see their UI cluttered with them.

The iOS camera UI balances these three classes of interaction well:



Figure 99.1: The iOS camera UI

These decisions are vital to the success of your UI and therefore the UX and product as a whole. Start early—at the paper prototyping or wireframe stage. Test often—looking at what users are doing and how they are discovering features and settings. Iterate quickly—make changes and get them shipped and tested as fast as you can. Only then will you get the balance between obvious, easy, and possible correct for your product and your users.

Learning points

- Decide whether interactions should be obvious, easy, or possible
- Test your assumptions with real users
- Iterate quickly in the early stages of your products to ensure success

#100

**“Does It Work
on Mobile?”
Is Obsolete**

In 2021, mobile browsing accounted for more than 56% of all web browsing traffic. In Africa, it's over 60%—mobile has been the most popular browsing experience for several years.



Statistics sourced from *Statcounter Global Stats*: <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet>.

With this in mind it feels like the terms “mobile-first,” “mobile-friendly,” and “responsive design” have stopped being worth mentioning—they should be a given. Everything is now assumed to be responsive and mobile-first, and it’s considered a breaking bug if your web app doesn’t work on mobile, not to mention it being a death sentence for your SEO—Google will penalize sites significantly if they don’t work well on mobile.

Modern frontend frameworks make it simple to build a web app or site that responds to different viewports, makes controls the right size for mobile, and “gracefully degrades” (hiding elements that don’t work on smaller devices). Responsive design means that the UI will adapt to different device sizes automatically, so you don’t have to build a distinct “mobile version” of your product.

What’s more, web apps are often more convenient for the user than native mobile apps. This isn’t a strict rule, as there are lots of reasons why you might need a native app—access to device features or heavy-duty computation or logic—but always consider whether a web app might be a better choice. Web apps need no installation, don’t have to be submitted to an app store, work across any platform with a web browser, and can be updated instantly without a download.

Also consider that a mobile-first approach helps you to reduce and simplify the experience in the design phase. I’ve watched users, in user testing, opt to use the mobile version of the site because it’s cleaner and simpler.

Learning points

- Your software has to work on mobile—it’s no longer optional
- Modern frontend frameworks make this easy to achieve
- Starting from a mobile-first position can help the overall design process

UX Philosophy

This section aims to provide some general advice in the form of two principles that will help you do better UX work and feel better about doing it.

#101

Don't Join the Dark Side

People check their smartphones a *lot*. One reason for this is that, in some way, it's a gamble. You check your phone, and maybe there are no notifications—or maybe there's a red blob over your favorite social media app. Maybe someone's retweeted your latest witty tweet or saved your Instagram picture of your brunch or your pet.

Each time you get a notification, you feel happy—your brain releases a little bit of dopamine. So, you wait a little while and you check your phone again, hoping for the same result and reinforcing the addictive behavior loop.

This isn't an accident. Many modern products, especially social media, are *designed* to be addictive. In *Hooked: A Guide to Building Habit-Forming Products*, psychologist Nir Eyal proposes the Hook Model:



A four-step process that, when embedded into products, subtly encourages customer behavior.

Through consecutive **hook cycles**, these products bring people back again and again without depending on costly advertising or aggressive messaging. Deliberately building subtly addictive behaviors into technology products is definitely a *Bad Thing*.

Next, there are so-called “dark patterns,” which are UI or UX patterns *designed to trick* the user into doing what the corporation or brand wants them to do:

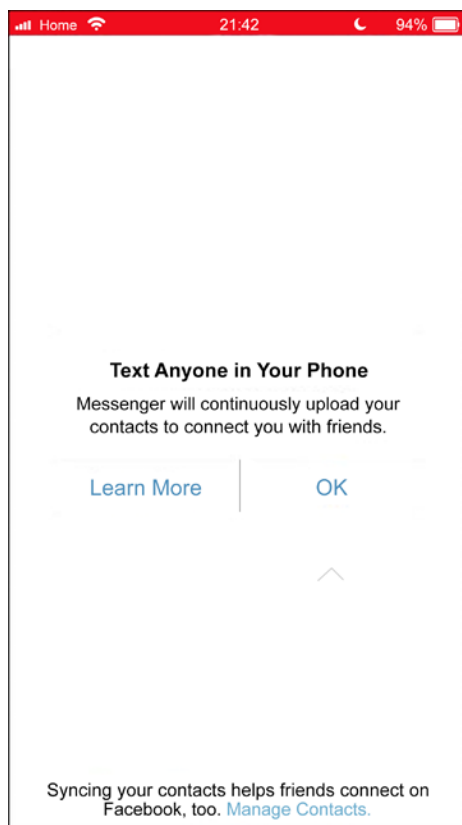


Figure 101.1: Dark pattern: In order to not send your contacts to Facebook, you need to tap Learn More

These are, in a way, exactly the same as the scams used by old-time fraudsters and rogue traders, now transplanted to the web and updated for the internet age. You'll definitely have come across some of these:

- Shopping carts that add extra “add-on” items (like insurance, protection policies, and so on) to your cart before you check out, hoping that you won't remove them
- Search results that begin their list by showing the item *they'd like to sell you* instead of the best result
- Ads that don't look like ads, so you accidentally tap them
- Changing a user's settings—edit your private profile and if you don't explicitly make it private again, the company will switch it back to public
- Unsubscribe “confirmation screens,” where you have to uncheck a ton of checkboxes *just right* to actually unsubscribe
- Cookie or privacy banners where it's easy to press **Accept all**, but hard to reject or customize tracking
- Software in an automobile engine management computer that checks whether the vehicle is being emission-tested and, if so, lowers the performance and emissions

I could go on—there are hundreds. Please don't do any of them.

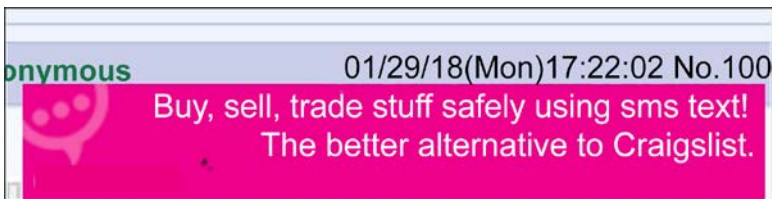


Figure 101.2: This mobile banner ad has a “speck of dirt” on the image, in the hope that the user will accidentally tap when they try to remove it

In some fields (medicine, for example), professionals have a code of conduct and ethics that informs the core of the work they do. Building software does not have such a code of conduct, but maybe it should do. All of these dark patterns and addictive products were designed by normal people working in normal software companies—they had a choice. They chose to fight for the company, not the user. Be a good UX professional and *don't join the dark side*.

Learning points

- Think about the moral and ethical implications of the software you help to create
- Design interfaces and experiences that you'd want to use
- Fight for the user, not the company

Bonus

Strive for Simplicity



A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away.

– Antoine de Saint-Exupéry

Strive for simplicity and clarity in every aspect of your work. Not just in the interfaces, copy, and experiences you design, but in the words you say in meetings and in the emails and chat messages you write. Avoid jargon, put people at ease, and try to improve the UX of everyone you interact with—users and colleagues.

Your mock-ups and designs should be clear and usable, but so should all other aspects of “you, the product”. Make yourself a delight for others to interact with, and be kind—the reward for this is a successful career in UX!

Join us on Discord!

Read this book alongside other users, UX professionals, and author Will Grant.

Ask questions, contribute to discussions with other readers, chat with Will via *Ask Me Anything* sessions and much more.

Scan the QR code or visit the link to join the **UX Squad** community.



<https://packt.link/101uxprinciples>



packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



JavaScript from Beginner to Professional

Laurence Svekis

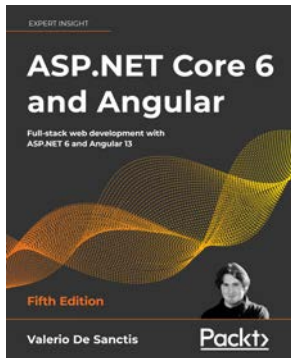
Maaïke van Putten

Rob Percival

ISBN: 9781800562523

- Use logic statements to make decisions within your code
- Save time with JavaScript loops by avoiding writing the same code repeatedly
- Use JavaScript functions and methods to selectively execute code
- Connect to HTML5 elements and bring your own web pages to life with interactive content
- Make your search patterns more effective with regular expressions
- Explore concurrency and asynchronous programming to process events efficiently and improve performance
- Get a head start on your next steps with primers on key libraries, frameworks, and APIs

Other Books You May Enjoy

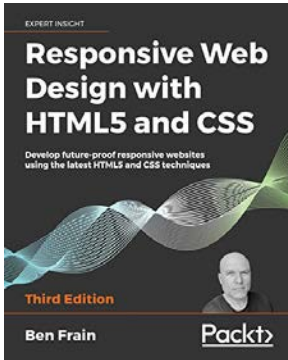


ASP.NET Core 6 and Angular, Fifth Edition

Valerio De Sanctis

ISBN: 9781803239705

- Use the new Visual Studio Standalone TypeScript Angular template
- Implement and consume a Web API interface with ASP.NET Core
- Set up an SQL database server using a local instance or a cloud datastore
- Perform C# and TypeScript debugging using Visual Studio 2022
- Create TDD and BDD unit tests using xUnit, Jasmine, and Karma
- Perform DBMS structured logging using providers such as SeriLog
- Deploy web apps to Azure App Service using IIS, Kestrel, and NGINX
- Learn to develop fast and flexible Web APIs using GraphQL
- Add real-time capabilities to Angular apps with ASP.NET Core SignalR



Responsive Web Design with HTML5 and CSS, Third Edition

Benjamin Frain

ISBN: 9781839211560

- Integrate CSS media queries into your designs; apply different styles to different devices
- Load different sets of images depending upon screen size or resolution
- Leverage the speed, semantics, and clean markup of accessible HTML patterns
- Implement SVGs into your designs to provide resolution-independent images
- Apply the latest features of CSS like custom properties, variable fonts, and CSS Grid
- Add validation and interface elements like date and color pickers to HTML forms
- Understand the multitude of ways to enhance interface elements with filters, shadows, animations, and more

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share your thoughts

Now you've finished *101 UX Principles, Second edition*, we'd love to hear your thoughts! If you purchased the book from Amazon, please [click here to go straight to the Amazon review page](#) for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index

A

Ableton Live tool 17

A/B testing 20, 21

accessible design

best-practice guidelines 268

principles 268-300

active verbs

selecting, over passive verbs 378

Agile Manifesto principles 12

alternative design patterns,
hamburger menu

breadcrumbs 119

navigation on bottom of view 118

tabbed navigation 119

autosaving feature 330

B

best-practice guidelines, UX

contrast 268

brand

building, by focusing on UX 28, 29

breadcrumb navigation 318

example 318

brutalism 272

business goals 8

C

card sorting technique 130

character spacing

setting 46

chatbots

cons 198, 199

client-side validation 208

color

using, with indicators 286, 287

complexity

benefits 16

consumer needs

focusing on 16

content

dealing with 91

principles 94-114

content screen 337

conversational user interfaces
(CUIs) 198

cookies

analytics cookies 102

obvious use cases 102

obvious user-friendly solution 103

tracking 102

create from existing flow

adding 344

currency input 246

customers

fighting for 8

D

dark patterns 21, 411, 412

data-driven decisions

driving 9

date picker control

using 190

default settings 388

examples 388

device-native features 86, 87

accessibility 290

input 160-162

dropdown 230

drop-down menu 78, 79

E

e-commerce pattern 326

add, to basket button 326

basket 326

checkout 326

confirmation 327

products 326

ellipsis

using 50

email addresses

validation, avoiding 174

empty state 106, 107

EU ePrivacy Directive 102

expectations

principles 381-404

F

favicon 340, 341

filesystem

complications 354, 355

flat design 54, 272, 273

forgot password fields

username field, prefilling 192

forgot password process

rules 192

form fields 236

forms 201, 204

complex forms 204

principles 204-221

user-entered data, saving 178, 179

validation 208

G

Google

advantages 322

Google Calendar (iOS) app

UI 273

UI, comparing to Stripe's control panel 274

guerilla testing 24

H

hamburger menu 118

alternative design patterns 118

hook cycles 410

human finger size

considering, for designing touch interfaces 300

human-usable buttons 88

hyperlinks 122

I

iconography

- ambiguous symbols, avoiding 278
- considerations 278
- principles 142-155

icons 139, 142

- consistent icons, using 142
- existing icons usage, avoiding in
 - new idea depiction 148
- obsolete icons, avoiding 144
- text label, creating 154
- text usage, avoiding 152

indicators

- using, with color 286

infinite scroll 94

- limiting, to content 94

input

- principles 157-199

input system

- case-sensitivity, adding 196

instant search 68

interactions

- classifying 400
- easy interactions 400
- obvious interactions 400
- possible interactions 400

L

labelling

- clear labels, writing for controls 296

lab tests 24

links 122

- descriptive links, using 280
- visual affordances, adding 123

live lookup 240

M

menu items

- repeating, in footer 136, 137
- splitting into subsections 126, 128

micro-animations 186

minimalism 272

mobile apps

- interface, optimizing 86

modal view 392

- using, for blocking
 - actions 392, 393

modern frontend frameworks 404

multiline input fields

- text area size, selecting 182, 183

N

navigation 115

- principles 115-137

navigation elements

- logical tab order, applying 294

Nielsen, Jakob

- principles 55, 154, 396

notification 306

Noun Project

- URL 149

numeric entry field

using 76

O

objectivity 4

ohnosecond

reference link 82

onscreen tips 358

organization

allies, building 8

P

pagination 98

benefits 98

Pareto principle 388

password

creating 166

entering 166, 167

paste option, enabling 170

rules 166

password manager 166

password resets 372, 373

pixel-perfect design

avoiding 290

Place Autocomplete 240

placeholders 186

Poisson distribution 25

postal codes 240

product demo

giving 358

video demo, giving 359

product design

performance, ensuring 20

progress

principles 253-264

progress bar 256

effective progress bar 257

example 256

linear progress bar 256

with numeric indicator 260

R

real users

testing with 24

relevant search results

displaying on top 384, 385

remote testing 25

responsive design 89

S

scrolling 89

search field, mobile phone

screen 69

search function 68

search results

filtering, by users 350

settings

categorizing 130-134

seven, plus or minus two

reference link 126

- show password toggle** 166
- Shopify example** 107, 236, 346, 347
- Sign in and Sign out**
 - using 366
- Sign up and Register**
 - using 369
- simplicity**
 - striving for 416
- skeleton screen** 337
- skip to content link**
 - adding, above header and navigation 282
 - hiding, CSS positioning used 283
- slider controls** 74
- snooze action** 307
- specific notifications**
 - turning off 306, 307
- spinner** 264
- Spotify search function** 70
- Stripe example** 235, 236, 274
- style-on-hover approach** 122
- system-native date picker** 230

T

- terminology**
 - consistency 364
 - principles 364-379
- tips**
 - making optional and dismissible 110

- top-level domains (TLDs)** 174
- touch interfaces**
 - human finger size, considering 300
- tree testing technique** 130
- Tweetie** 66
- Twitter** 66
 - examples 66, 94, 114
- type sizes**
 - used, for differentiation of information 43
- typography**
 - principles 43-46

U

- UI**
 - big buttons, creating 60
 - buttons, creating 55, 56
 - clickable buttons, creating 64
 - interactive elements, creating 54
 - real-life examples, using 55
- UI control**
 - selecting, for job 220, 221
- UI, mobile apps**
 - device-native features 86, 87
 - human-usable buttons 88
 - information entry 89
 - optimizing 86
 - page shape and format 89
- undo controls** 82, 83
- usability testing methods**
 - guerilla testing 24

- lab tests 24
 - remote testing 25
 - user**
 - navigating, to last unread item in feed 114
 - user-centric point of view**
 - writing from 376
 - user experience**
 - expected experience, providing 396-398
 - user feedback 332**
 - user input 223**
 - principles 226-251
 - user interfaces**
 - animations, using with precautions 186
 - user payments**
 - simplifying 346, 347
 - user research**
 - culture, building 9
 - users**
 - flexibility, providing for data entry 216
 - user's journey**
 - completion 311
 - principles 303-359
 - stages 314, 315
 - user testing 4, 24**
 - diverse group of people 25
 - small test groups 25
 - within target audience 24
 - UX controls**
 - principles 50-89
 - types 47
 - UX failures 179**
 - UX Field**
 - principles 1-28
 - ways of violating principle 12
 - UX Philosophy**
 - principles 407-416
 - UX principles**
 - applying, to pricing pages 347
 - UX research toolbox 21**
- V**
- validation, forms 208**
 - client-side validation 208
 - error handling 212
 - server-side validation 212
 - vanity splash screen**
 - avoiding 336, 337
 - visual indicator**
 - displaying, for unsaved user's work 330

W

web brutalism 272

Web Content Accessibility
Guidelines (WCAG) 212, 268

webmasters 170

World Wide Web Consortium
(W3C) 268

Z

ZIP codes 240

