



REDESIGN THE WEB

#3 | THE
SMASHING
BOOK

REDESIGN

THE WEB

[illegible]

FOR BEING SMASHING! THIS BOOK IS DEDICATED TO ALL MEMBERS OF THE SMASHING MAGAZINE COMMUNITY. HERE ARE ONLY 2,630 OF THEM. WE TRULY APPRECIATE ALL YOUR SUPPORT.

Megan Yoerg • Melchior Mazzone • Melissa Palus • Meryl Menezes • Mette Sofie Park • Micael Noodep V. • Micah D Montoya • Michael A. Richards • Michael Anthony Sosa • Michael Borum • Michael Brydebøl • Michael Byers • Michael Christensen • Michael Craig Bedway • Michael Dahlsbilde • Michael Dehn Chastard • Michael Eshbaster • Michael Formander • Michael Fulk • Michael Galbraith • Michael Green • Michael

• Yves Eulin • Yvonne Irene Wenzel • Zdzisław Suppli • Zdeněk Ařlý • Zdravko Jasker • Zdzisław • Zeydi Wari Hijo • Zia Zek • ZiadNasrallah • Ziv Cohen • Zoltan Hossza • Zoltan Kovács • Zvi Goldfarb

Published 2012 by Smashing Media GmbH, Freiburg, Germany.

Printed by DE Druck Europa GmbH.

Cover Design: Veerle Pieters. Illustrations: Kate McLelland.

Proofreading: Andrew Lobo, Iris Lješnjnin. Author Bios: Talita Telma Stöckle.

Editing and Quality Control: Vitaly Friedman, Iris Lješnjnin.

eBook Production: Thomas Burkert, Talita Telma Stöckle, Andrew Rogerson.

Marketing: Stephan Poppe. Technical Support: Robin Schulz.

Design and Layout: Ricardo Gimenes, Markus Seyfferth.

Typefaces used: Skolar by David Březina, Proxima Nova by Mark Simonson, Andale Mono by Steve Matteson.

The Smashing Book #3 was written by Elliot Jay Stocks, Paul Boag, Rachel Andrew, Ben Schwarz, Lea Verou, David Storey, Christian Heilmann, Dmitry Fadeyev, Marc Edwards, Aarron Walter, Aral Balkan, Stephen Hay and Andy Clarke.

The chapters were reviewed and edited by Collis Ta'eed, Ryan Carson, Harley Finkelstein, Daniel Weinand, Russ Weakley, Tab Atkins, Paul Irish, Joshua Porter, Jon Hicks, Denise Jacobs, Josh Clark, Anders M. Andersen, Bryan Rieger and Owen Gregory.

Idea and Concept: Vitaly Friedman, Sven Lennartz.

All links featured in this book can be found at www.smashing-links.com.



7	Preface <i>by Elliot Jay Stocks</i>
9	The Business Side of Redesign <i>by Paul Boag</i>
37	Selecting a Platform: Technical Considerations for Your Redesign <i>by Rachel Andrew</i>
71	Jumping Into HTML5 <i>by Ben Schwarz</i>
93	Restyle, Recode, Reimagine With CSS3 <i>by David Storey and Lea Verou</i>
135	JavaScript Rediscovered: Tricks to Replace Complex jQuery <i>by Christian Heilmann</i>
163	Techniques for Building Better User Experiences <i>by Dmitry Fadeyev</i>
197	Designing for the Future, Using Photoshop <i>by Marc Edwards</i>
231	Redesigning With Personality <i>by Aarron Walter</i>
255	Mobile Considerations in User Experience Design: Web or Native? <i>by Aral Balkan</i>
285	Workflow Redesigned: A Future-Friendly Approach <i>by Stephen Hay</i>
311	Becoming Fabulously Flexible: Designing Atoms and Elements <i>by Andy Clarke</i>
335	Index



Preface

by Elliot Jay Stocks

WHEN YOU WORK IN AN INDUSTRY that evolves at an incredibly rapid pace, every moment on that continuous journey of evolution is an exciting time. To say that, “Now is an exciting time to be working in Web design and development,” is something of a misnomer. It always has been and always will be an exciting time to live on the bleeding edge of the Web.

And yet, in recent months, I still find myself uttering those words, because now is an exciting time. In the early days of table-based layouts and novelty “new media,” we were nervous children, looking to our parents from the world of analogue media for support; in the early years of this century, we became awkward teenagers, experimenting with a variety of tools and techniques, both for better (Web standards) and for worse (proprietary plugins). Now, we’ve matured into relatively confident twenty-somethings, channeling all that we’ve learned into new, exciting experiences that finally—at long last—embrace the transient, malleable, open, and fluid nature of the Web.

It is not enough to have refined tools, nor is it enough to have the technical and creative knowledge to use them. True maturity comes from a marriage of both: a deeper understanding of what can be achieved—but more importantly—what *should* be achieved. Multiple browser support for experimental CSS features is not revolutionary; using them gracefully is. Having the choice of thousands of typefaces to use is not revolutionary; using them gracefully is. Inserting media queries to change styles according to browser width is not revolutionary; using them gracefully is.

This concept of our maturing industry—moreover, our own awareness of it—has manifested itself on the stages of conferences around the world, in the pages of our printed magazines, journals, and books, and, of course, throughout the Web’s network of blogs. It was this sense of the Web having grown up that spurred my recent redesign of Smashing Magazine’s website, and the theme of redesign is one you’ll find throughout this book. It takes the form of the business considerations that go into a redesign, as outlined by Paul Boag, as well as the modernizing of a website with tools like HTML5, CSS3 and JavaScript, as discussed by Ben Schwarz, Lea Verou, David Storey and Christian Heilmann; it’s in the responsive, mobile-friendly techniques demonstrated by Stephen Hay and Andy Clarke; and it certainly manifests itself in the broader considerations covered in the chapters by Rachel Andrew, Dmitry Fadeyev, Marc Edwards, Aral Balkan and Aaron Walter.

The desire to push things forward that drove the redesign of Smashing Magazine also inspired the creation (and content) of the third Smashing Book. It is this passion that binds us as a collaborative rather than competitive community—a passion I’m sure you’ll feel as you leaf through these pages—inspiring each and every one of us to make the Web a better place... one CSS gradient, one Web font, and one media query at a time.



The Business Side of Redesign

Written by Paul Boag

Reviewed by Collis Ta'eed

REDESIGNING A WEBSITE is the most fun Web designers can have with their clothes on. Nothing is more exciting than a blank Photoshop document, a library of CSS3 snippets and a world of possibilities. Unfortunately, I am here to burst your bubble.

Other chapters will cover cool design techniques, cooler coding possibilities and as many beautiful design examples as a Web designer can look at without going blind. I, on the other hand, will talk about the business behind redesigns.

What could my reason possibly be for focusing on such an uninspiring subject? Simple. The business behind a design has the potential not only to make a website successful, but alternately to ruin a perfectly good project. Like all of you, I have been involved in too many projects in which my excitement was dashed by changing requirements, ill-defined briefs and clients who appeared to be uncooperative. Fortunately, years of experience have taught me that if you are sufficiently prepared, these dangers can be significantly reduced.

This chapter aims to give you the techniques and knowledge needed to ensure that your next redesign project doesn't turn into a nightmare. We're going to look at subjects such as:

- The dangers of redesigning,
- Researching your project thoroughly,
- Working with the client,
- Testing your design,
- Future-proofing the website.

Before you get too excited by this list, let's begin by asking, when is it time to redesign?

Is Now the Time to Redesign?

When a client or boss asks us for a redesign, we want to leap into action. After all, this is what we love doing. Unfortunately, a redesign is not always the best move, and it falls to us, the Web experts, to explain why.

Since its birth, the Web has had a culture of periodic redesign. Every few years, somebody in senior management is horrified by the state of their website and demands a redesign. The old website is thrown out and a shiny new one is put in its place.

For a brief moment, this new website lights up the World Wide Web. However, the content does not get updated, technology moves on and tastes change. The shiny new website gradually tarnishes, until the organization is ashamed of it.

After a couple of years of stagnation, somebody in senior management once again recognizes the need to do something, and the process begins all over again.

WHY COMPLETE REDESIGNS ARE WASTEFUL

The next time a client asks for a redesign, be sure to discuss why this request might be a mistake. The reasons might include the following:

- When the website is redesigned, the entire thing has to be replaced, including elements that still work well. Everything is rebuilt from scratch.
- The website spends most of its life being ineffective because content gets out of date and design inconsistencies creep in. Although it may shine once every few years, the rest of the time it is seen as an embarrassment and, so, is underused.
- Users rarely respond well to major changes. You need only look at the outcry every time Facebook does a redesign to see this problem.¹
- Redesigning periodically is bad for cash flow, because it requires substantial investment every few years.
- Testing the effectiveness of a complete redesign is hard because so much has changed.
- Long periods with no change give users little reason to return to your website.

In many cases, I recommend to my clients that they realign instead.

GOOD DESIGNERS REDESIGN, GREAT DESIGNERS REALIGN

It was in 2005 when Cameron Moll first popularized the idea of realigning websites, rather than doing wholesale redesigns.² The concept has grown considerably since then but can still be simply defined: realignment of a website consists of a series of incremental changes over time to meet particular business objectives.

¹ The Washington Post, "Facebook Changes Confuse Users, as a Major Overhaul Looms," smashed.by/fbc

² Moll, Cameron. "Good Designers Redesign, Great Designers Realign," smashed.by/realign

In other words, proponents of realignment reject the notion that a website needs a massive overhaul every few years in order to remain up to date. Instead, they propose an ongoing program of incremental development that maximizes the effectiveness of the website to meet business objectives. This avoids the pitfalls of redesign and expensive rebuilds every few years.

This thinking moves away from the idea that a website is ever really complete or can be “signed off.” It recognizes that design needs to evolve based on continual testing. As more is learned about user behavior and preference, the design should incorporate this new knowledge so that the website can become even more effective. That being said, realignment doesn’t fit every project.

“Realigning a website is a series of incremental changes over time to meet specific business objectives.”

— Cameron Moll

WHEN REDESIGN IS PREFERABLE TO REALIGNMENT

Despite the fact that I am a fan of incrementally improving websites (i.e. realignment), the majority of the work that I do at my Web design agency, Headscape, still consists of complete redesigns. We try wherever possible to use a website’s current look and feel as a starting point, but the build itself often has to be done from scratch. This is because few existing websites are built with long-term development in mind. Little thought is given to future-proofing, as we will discuss later in this chapter.

Also, realignment itself can be a reason to do a major redesign. Even a website that has been part of an ongoing program of realignment occasionally needs a complete redesign. This is for two reasons. First, many websites that are being constantly changed will develop inconsistencies that undermine the user interface. In an article encouraging incremental change,³ Jakob Nielsen writes:

“In the long run, however, incrementalism eventually destroys cohesiveness, calling for a new UI architecture.”

In other words, every website eventually needs to be redesigned.

³ Nielsen, Jacob. “Fresh vs. Familiar: How Aggressively to Redesign,” smashed.by/nielsen

The second problem with incremental change is its impact on the underlying code. We have worked with one of our clients for well over six years in our agency. This period has been spent in a constant process of refinement and incremental change. New features have been added, while others have been dropped. Design styles have been tweaked based on user feedback. All of these changes eventually resulted in a coding nightmare. A part of the website was built on classic ASP, other parts on .NET. The CSS files became bloated with code that was no longer needed. We planned and documented as best as we could, but eventually the entire underlying code base had to be rewritten.

Unsurprisingly, the client was reluctant to pay a considerable amount of money without seeing any visual difference. So, we tied this work in with a redesign of the website, thereby killing two birds with one stone.

Although a process of incremental change is preferable, doing the occasional redesign is still perfectly fine. The key is to recognize the signals when a redesign is required.

REDESIGN INDICATORS

We have already identified two indicators that point to a need for a redesign rather than a realignment: when inconsistencies in the user interface creep in, and when the code becomes unmanageable.

A related coding issue is performance. If the website suffers from serious performance issues, and incremental changes have not been enough to fix them, then a redesign might be the answer. Building an entire website from scratch gives you the chance to optimize for performance by removing legacy code.

A redesign might also be necessary simply because the existing design has reached its limit. Although incremental change is usually possible, some aspects are extremely hard to change without profoundly affecting the rest of the design.

For example, changing the underlying grid structure of a website will affect everything from navigation to type size. Therefore, if the grid needs to be radically changed because of new incompatible content or changes in screen resolutions, then, again, a significant redesign might be necessary.

There could also be business reasons to consider a redesign rather than realignment. A major redesign brings with it promotional opportunities that you don't get with incremental changes. Also, a redesign provides the kind of radical shake-up that is sometimes necessary to give an old website new life.

Finally, another common reason I push for a redesign is if an organization has undergone major rebranding or repositioning. This wouldn't be just minor tweaks to the logo. Rather, if the company were to significantly change its market position, then that would affect everything from target audience to content and visual appearance.

Simply slapping a new logo on the website, as I have often seen done, would not be enough. Brand is more than just a logo. Whether you choose to redesign or realign, it needs to be done for the right reasons.

REASONS FOR CHANGE

As Cameron Moll points out in his article, redesigning merely because the current website looks “old” is not enough. Change needs to be driven by business objectives. Typical reasons for change are:

- Shift in market trends,
- Change in business model,
- Drop in conversion rate,
- Increase in customer support requests relating to the website,
- Repositioning of brand.

Note that not only are these drivers related to business (rather than purely aesthetics), but their results benefit from immediate action. In other words, if your conversion rate is falling or your business offering has changed, then you don't want to wait two years for the next major redesign to address it.

This is why incremental change is often preferable. Things move fast, both online and offline. If you want your website to run at peak efficiency, then you cannot wait to make major changes to it.

Depending on the actual problem, changes to a website could be as minor as textual adjustments or as major as a redesign of the entire user interface. Whatever the case, when considering whether now is the time to change your website, those decisions should be business-driven and done as part of an ongoing and incremental process of development. Unfortunately, changing your website comes with certain dangers.

Avoiding Project Pitfalls

When we start a new project, there are endless possibilities. We are excited, enthused and full of vision. Why is it, then, that by the end we just want to launch it and never mention it again?

Whether we're undertaking a complete redesign or a modest realignment, there is no shortage of pitfalls for the unwary designer. Worse still, we tend to repeat the same mistakes. Before launching into your next project, make sure to take a few moments to consider the most common issues that arise and how to deal with them.

Although every project is different, here are the biggest issues I have encountered in my sixteen years as a Web designer:

- Scope creep,
- Falling victim to fashion and trends,
- Building without considering the return on investment,
- Negative feedback.

Let's look at each in turn.

COMBATING SCOPE CREEP

Nothing sends a colder shiver down the spine of any Web designer than hearing a client utter the phrase, "I have an idea!"

The scope of any project will almost inevitably creep wider. From the perspective of clients, this is understandable. They are not Web experts like us and so do not think of everything in advance. Only by working alongside us do they start to realize the possibilities. How, then, do you deal with the goal posts moving? One way is to simply dig your heels in and say "No." However, this could lead to confrontation and damage your relationship with the client.

The methods we have found to succeed are to introduce a wish list of ideas and to work in phases. When either the client or we have an idea, it is added to the wish list. The ideas should not be censored or appraised, just added.

At the end of the project, the wish list is reviewed. The items on the list are vetted for practicality, and those that remain are prioritized and organized into development phases.

By making the client aware of this process up front, we set expectations about how changes are handled. It also encourages the client to think in terms of an ongoing relationship, which is good for repeat business.

The client will inevitably insist that some of their ideas are within the project's scope. I recommend that you avoid arguing about this mid-project. Explain that implementing the idea in this phase of development would harm the project. Suggest that the issue be parked until after the launch and discussed then.

Parking contested issues until phase 2 has three advantages:

- The client is less likely to force the issue once they see their wonderful new website.
- The new website won't be held up by features, making it more likely to launch on time.
- With the new website live, you are in a stronger position to debate what exactly was within scope.

Not that scope is the only thing that strikes terror in our hearts.

FALLING VICTIM TO FASHION AND TRENDS

Another utterance that makes Web designers blanch is something along the lines of, "My son is obsessed with Facebook. We need Facebook on our website." Of course, Facebook is not the problem. The client could just as easily say, "Our competitor's website is cool. We need to look like that," or "We need a Web 2.0. website."

The point is that clients often jump on bandwagons (sometimes mid-project), and we are expected to jump on right beside them. Not that we are any better. As Web designers, we love the latest trend, whether it's responsive design or gradients and drop-shadows.

The problem with fashion is that it changes, as do the client's requirements. What is liked when the project commences could be hated by the time the client has to sign off on the design. Even if the client remains consistent in their opinion, the website will look out of date over time, making it more difficult to realign and perhaps leading the client to view your work in a negative light.

The most powerful defense against fashion is simple: we must ask *why*. Saying that something is “cool” or “in” is not enough. We must ask why the idea is good and provide tangible business reasons. We need to apply this reasoning to our own choices and those of the client. When a client falls prey to fashion, avoid shooting them down in flames. Instead, gently ask why they feel it is a good idea. Often, some gentle prodding is enough to make them realize they are being seduced by the shiny and new.

When this does not do the trick, go further and ask them where they expect the return on investment to come from.

BUILDING WITHOUT CONSIDERING RETURN ON INVESTMENT

Many of the calls for bids I have received over the years read more like a wish list than a comprehensive brief. I see it as my job to refine each proposal into something that will benefit the client’s business, rather than give them exactly what they have asked for.

Clients (like all of us) get seduced by features. They don’t consider the cost of implementation compared to the return, because they are not in the position to make that assessment. It is up to you to help them with that process.

I remember in the early days of running our design agency, we built a website that supported multiple languages. Because this was in the days before multi-lingual support was a standard feature, it proved to be expensive.

I didn’t question why the client wanted this feature or indeed how they were going to get their content translated. In the end, the functionality was never used. I implemented their wish list rather than met their business needs and so wasted a lot of their money.

The more features are added, the greater the cost and complexity. Your job is to help the client keep things simple.

You might be noticing a recurring theme here: always favor simple solutions. Your job as a Web designer is to curb your own excesses and those of the client. Instead of a big redesign, opt for a subtler realignment. Instead of adding more features, outline a simple feature set and stick to it. Instead of following the latest trends, focus on the simple, timeless and classic.

This will ensure that the client gets the highest return on the money they spend with you. It also minimizes the final pitfall of all redesign projects: negative feedback.

DEALING WITH NEGATIVE FEEDBACK

I have already mentioned that people rarely react well to change. Some of us simply don’t like change, while others are disappointed because things haven’t changed in the

way they had hoped. In either case, any change to a website will elicit a reaction among both users and stakeholders.

We have established that minimizing change will reduce the chance of criticism. But this is not always possible, and even when it is you will still get negative feedback. Initial responses, whether positive or negative, are not to be trusted.

I remember discussing the issue with Daniel Burka when he was the lead designer at Digg. He told me how difficult he found it not to immediately respond to criticism after going live with a new design element. His inclination was to fix the perceived problem. But he learned that if you wait a couple of weeks, users would get used to a change and often accept it. In the end, this became his standard approach. He wouldn't make any further changes until a design element had been live for at least two weeks.

When a client or user sees new work, they make a snap judgment that rarely reflects their eventual perception.

Someone could be blown away by the aesthetics of a new design and yet in time discover the website to be unusable. Likewise, someone might hate certain changes when they first see them but then grow to love them.

I have learned three simple tactics to manage negative feedback effectively. The first is to allow time in the project's schedule between first showing the design to the client and making changes to it. This gives the client an opportunity to get used to the design before responding.

Secondly, actively encourage the client to take this time to digest the design. Explain to the client up front that an initial reaction is not always the best one and that they should keep going back to the design over a period of time before responding.

Finally, warn the client in advance of what will happen once the new design goes live. Explain that users might initially respond negatively. Criticism usually worries clients. This can understandably lead to knee-jerk reactions that make things worse.

Clients might also feel the same pressure when showing your work internally. Their immediate reaction might be to act on every negative comment, despite the stakeholders having lacked time to absorb the design.

Preempt this problem by discussing it with the client beforehand. You can then refer back to this moment when the problem actually occurs. This will reassure the client that this is a common occurrence and one you are prepared for.

Preparation is the key to a successful redesign or realignment. It leads to a better website for the client and a more rewarding project for you. But to prepare properly, you have to do your homework.

Do Your Research, Be Prepared

When deadlines and budgets are tight, there is an overwhelming temptation to jump right into the design process. However, doing so is unwise. First, we need a clear understanding of what we are doing and why we are doing it. This entails some research. Before explaining what I mean by research, I want to outline why it is important to the process.

WHY DO YOUR HOMEWORK?

There are two reasons to do research before starting any project. First, it will help educate you about the project. Secondly (and arguably just as important), it gives you the ammunition required to get approval for your work.

If we have a firm grasp of things such as business objectives, competition, statistics and the weaknesses of the existing website, then justifying our design decisions in terms that the client can relate to becomes much easier.

Justifying white space around a contact form as being aesthetically pleasing is not something a client can identify with. But telling them that the white space will help achieve their business objective of drawing more inquiries is something they can understand. What then is involved in research?

WHAT RESEARCH SHOULD I UNDERTAKE?

The amount of research you undertake should be proportional to the value of the project. But no matter the size, you should undertake at least *some* research. I am often surprised at the lack of basic information in the average request for proposal. Requests often fail to address fundamental questions, such as:

- Why do we have a website?
- What do we want the website to achieve?
- How will we measure its success?
- What do we want users to do on our site?

As Web designers, we must extract this information from the client before beginning work. Business objectives should be at the top of the list.

Getting the client to articulate their business objectives can be problematic. I have often asked clients why they have a website and what they wish to achieve, only to receive vague answers.

I no longer assume that this thinking has taken place. Instead, I sit down with the client at the outset of a project and brainstorm on their business objectives. I then work with the client to prioritize this list and turn it into measurable success criteria.

For example, a vague business objective such as “increase sales” needs to be turned into something more specific and directly connected with a call to action. So “increase sales” would become “increase the number of quality leads being submitted via the website’s contact form.”

Prioritizing the business objectives is important because they will sometimes clash. For example, one business objective might be to generate more sales leads, while a higher priority might be to showcase a product. So, when someone suggests forcing users to submit their email address before being able to view a product demo, you can counter by saying that showcasing the product is more important.

Establishing specific, measurable business objectives should be the minimum amount of research carried out in any project. But for most projects, we should dig a little deeper still. One way is to carry out stakeholder interviews.

THE VALUE OF STAKEHOLDER INTERVIEWS

A *stakeholder interview* is a semi-structured discussion with those who benefit from a website. It could be someone who works directly on the website (such as a content editor) or individuals who rely on the website to achieve their business objectives (such as department heads).

Stakeholder interviews provide four benefits:

- **They bring the Web designer up to speed on business requirements.**

When you’re faced with a complex business model in a new sector, stakeholder interviews are a valuable way to understand the requirements of the client. By speaking with stakeholders, you learn about the sector and the organization, while identifying how the website can meet the client’s business needs.

- **They provide a more comprehensive perspective.**

Most Web projects in large organizations will affect numerous parts of the business. To fully understand a project's potential impact, you need to discuss the project with all parties. Most projects are commissioned by a single department, which will have a particular perspective on the objectives. By talking to other stakeholders, you ensure that the project helps rather than hinders others within the organization.

- **They are politically advantageous.**

Unfortunately, internal politics is a reality in most large organizations. This means that there is no shortage of people who want to be heard. Stakeholder interviews provide an environment in which they can express their opinions. I have found that if people in an organization feel that their opinions have been heard, they are much more likely to support the design further down the line. Stakeholder interviews also help to justify a design because you can refer back to their comments as justification for certain design elements.

- **They provide access to the real decision-makers.**

When working on a big project, you will often be dealing with a relatively junior employee, with the real decision-makers staying behind the scenes, which can be problematic. Stakeholder interviews allow you to talk to these people and understand what they want to achieve.

Well-run stakeholder interviews ensure that the Web project gets clearly defined goals that benefit everyone in the organization, while at the same time confirming buy-in from all parties. Stakeholder interviews focus on the organization and the future of the website. But a lot is to be learned by reviewing what's already there.

RESEARCHING WHAT CURRENTLY EXISTS

We begin many projects by reviewing a client's online presence. This review is often commissioned as a separate project, before we even consider a website redesign. This gives the client an opportunity to work with us on a small task before committing to a large project, making it easy for the client to decide whether to work with us long term without having to commit up front. Assuming that the client is open to a research phase, we have five options (picking only one or two in most cases):

- **A strategic review**

A strategic review looks at their current Web presence. It outlines strengths and weaknesses, and goes on to make recommendations for improvement. This report is an opportunity for the client to benefit from your experience, rather than just use you as a pixel-pusher.

- **A heuristic review**

Similar to a strategic review, a heuristic review analyzes the strengths and weaknesses of the existing website. The difference is that a heuristic review uses a set number of criteria, rated between 1 and 3. A heuristic review provides an analysis of the website but not much in the way of strategies for improvements.

- **A competitive analysis**

A competitive analysis uses similar criteria as a heuristic review but applies them to the competition. This provides valuable insight and helps the client learn from their competitors' mistakes and adopt their successes.

- **An analytics report**

An analytics report identifies problems with the website and is a benchmark by which to measure change. It can reveal much about user behavior. For example, you can learn which parts of the website convert best. It also shows exit points, thereby revealing problem pages.

- **Personas**

Personas are a powerful tool for focusing on users. A set of personas tells us about the website's users and what they are trying to achieve. They provide more than demographic information, by identifying use cases and user journeys through the website.

Although time-consuming, this kind of research is valuable.

We've gone over the benefits of understanding and justifying design decisions, and touched on how this helps the client get to know you better. The last point we'll make is important if you want a collaborative working relationship with your client.

A Collaborative Approach to Redesign

Many of the relationships I see between designers and clients are broken. The designer is subservient to the client and ends up working in isolation.

This traditional client-supplier relationship is harmful for a number of reasons:

- The client does not benefit from the full experience of the designer.
- The designer becomes frustrated because they are reduced to being a pixelpusher.
- The client feels excluded from the process.
- There is a lack of communication between the two parties, which leads to misunderstanding. Because the designer works in isolation, the chance of producing something that the client will reject increases.
- The client is left to make important decisions—some they really aren't educated to make.

My approach is to work in a collaborative and equal relationship with clients, including them in every part of the process as well as taking a much more active role in decision-making myself.

Instead of working on the design in isolation, I work *alongside* the client, showing them sketches, wireframes, mood boards and design comps. Once the final design is submitted, the client is more likely to approve it for three reasons.

First, the design will not come as a surprise. The client will have seen the work that went into it, and so the final piece will seem to be a natural extension of that. For this approach to work, you need a design process that includes the client.

Secondly, the client will feel a sense of ownership over the design. They will have provided feedback at every step of the process and so would consider the design to be as much theirs as yours.

Finally, the client will be able to understand where the design came from because they had a hand in the thinking that went into it. The bonus is that, because the client understands and feels connected to the design, they will do a better job of convincing others of its merits. For this approach to work, you need a design process that includes the client.

THE RIGHT DESIGN PROCESS

If you are a designer who works intuitively towards the final design, then you will find collaboration difficult. The key to working collaboratively is to engage the client in design decisions. I am not suggesting that the client should sit with you as you design. But they need to get involved in multiple stages of the design process.

I include the client in several stages of the design process:

- **At the ideas stage**

Before beginning to work on a design, I sit down with the client to brainstorm ideas. This is the time to look at websites that inspire them and to discuss color, typography and imagery. It is also the time to discuss personality. A great question I ask is, “If the website was a famous person, who would it be?” This helps both parties visualize a character that you can emulate in the design.

- **After mood-boarding**

After the initial brainstorming session, I go away and produce mood boards that reflect different design directions. I then discuss these with the client and refine them further. I don’t make them too polished. They need to be easy to produce so that I can iterate quickly. This is my chance to explore different aesthetic approaches at once.

- **While wireframing**

I always schedule a wireframing meeting in which the client and I (and even other stakeholders) sit down and sketch out different layouts for key pages. The wireframes don’t need to be high fidelity—just enough to leave the client feeling like they have contributed to the direction of the design. I then refine them after the meeting.

- **When refining design comps**

When the time comes to present the client with a design (or HTML prototype), it won’t come as a surprise because it will have been based on mood boards and wireframes. But I still give the client an opportunity to discuss any last-minute issues before producing the final iteration.

You will be pleasantly surprised by how little iteration is required when following this approach. Clients are often happy to sign off on work with only a few changes because they have been included in the process from the beginning. A word of warning, though: things can go wrong when the design is shown to people who were not included in this process. This is when the presentation of the design really matters.

PRESENTING THE DESIGN

All of your hard work on collaborating with the client could be for nothing if they are not the final decision-maker. Working closely with the client makes it much more likely that they will defend the design internally, but someone seeing the design for the first time won't have that wealth of information with which to make an informed judgment.

The only way I have found around this issue is to present the design to these other decision-makers. This way, I ensure that they see the thoroughness of my process and hear the thinking that went into the design as well.

Ideally, this meeting would be face to face or over a conference call. But that is not always possible. Another solution I have found to work is to submit the design with an accompanying explanation. This works well as a video file with a voice-over. This ensures that nobody sees the design without also hearing the explanation of the design process.

Thus, viewers will not judge the design purely on first impression. In my experience, clients love it, too. Many clients have commented on how much more impressive this way of presenting a design is than a static image. A video also works well now that websites are becoming increasingly dynamic. Instead of a static image, you can show JavaScript elements and even responsive design when needed. Best of all, the website doesn't need to work across browsers; as long as it works in one browser, you can record it. Of course, no matter how well you present the design, things can go horribly wrong when stakeholders start providing feedback.

Dealing With Feedback Through Testing

We like to think of ourselves as rational human beings. In reality, we are not. We are swayed by everything from childhood experience to having missed our cup of coffee in the morning!

Nowhere is this kind of emotional response more apparent than in people's reaction to design. Everyone knows what they don't like, and we all like different things. One of my clients even rejected the color of a design because it reminded them of the dress that an elderly relative wore!

With design being so subjective, finding consensus on the look and feel of a website is hard. Although there are underlying principles of design, there are also factors that can make a piece look great to one person and terrible to another. Left unchecked, this turns the approval phase into a lottery. Fortunately, there are things that can be done. The secret is: psychology.

As humans, we like to feel that we are consistent in our views. That is why a client who has been involved in creating a design is much less likely to reject it. That would be a contradiction.

The other way this consistency works to our advantage is that people like to consider themselves as being logical. If you present a logical, objective argument for a design, the listener is more likely to accept it because they feel they should respond logically. This desire to be consistent with their self-image can even outweigh their personal dislike of the design.

I remember producing a design for the higher-education sector a few years ago. The website was squarely aimed at what was then the MySpace generation. I personally hated the design, and so did the client. However, user testing showed conclusively that the busy design and garish colors struck the right tone with the users we were aiming at. Our personal opinion had to be set aside in face of the data.

This is why testing the design is so important. It moves decisions away from subjective opinion and towards objective research. It also puts the focus back on the user. The client might hate the design, but if it resonates with users and achieves the client's business objectives, then they can be convinced.

How then do you persuade a client to adopt a testing-centric approach when redesigning their website?

SELLING USER TESTING TO THE CLIENT

The problem with testing is that it takes time and money. Many clients are unwilling to pay for this, instead preferring to rely on your "expertise" and their personal preference. Because cost is their primary reason for not testing, money has to be part of your justification for testing. When encouraging clients to test, I mention two financial benefits.

First, testing can significantly increase the effectiveness of a website. It ensures that the design communicates your messages clearly, and it guides the user to complete calls to action. Ultimately, this makes the website more likely to generate a significant return on investment.

Secondly, user testing can save a significant amount of money. Without testing, having to reverse course after going down a path that is found to be ineffective is all too common. There are even times when you will drop entire features because testing shows that the user does not need them. By testing early and often, you catch issues in the user interface and feature set before having invested too much time.

In short, without user testing, the redesign process is blind. There is no guarantee that the result will be effective, and mistakes will inevitably be made along the way. Once the client is on board with testing, you need to choose the most appropriate type of testing.

WHAT TYPE OF TESTING TO CONDUCT

Several options are available, and picking the right one for the job is important. I'll focus on the three that I use most often.

Surveying is not used extensively by the Web design community as a method of soliciting user feedback. But it can be effective in certain circumstances. I have worked with clients from the sciences who question the value of testing a handful of subjects. From their perspective, the sample size would be too small to be statistically relevant. A survey proved to be the solution because it involved considerably more people.

For this to be most effective, the questions need to be limited to multiple choice, thus making comparison possible.

Surveying is well suited to certain questions:

- Choosing between two subtly different design approaches;
- Getting feedback on whether a design communicates the right brand keywords (for example, is the design casual or professional, conservative or liberal?);
- Asking the user which of a limited set of options they would click on;
- Conducting closed card sorting (where the user has to organize pages into set categories).

There is no shortage of tools to support this kind of online user testing. Two that I personally use are Verify⁴ and WebSort.⁵ The benefit of surveying is that a large statistically relevant set of responses constitutes a strong business case for a particular design approach, which will give the client confidence in your decisions.

GETTING DESIGN FEEDBACK

The second type of user testing is *design testing*. This is where the interface is tested for usability and aesthetics.

Typically, I run design testing near the beginning of the design phase, with a relatively large group of users (20+). They can be interviewed individually or in small groups. I run a number of tests, but two of the most common are the emotional response test and the flash test.

In the emotional response test, you show the user a design and (as described above) ask them to choose between keyword pairs. Is the design busy or spacious, classic or modern? The idea is to judge whether the design is in line with the image you are trying to communicate. The keywords that users choose should include some of the desired results. If users pick these words, then I know I am on the right track with aesthetics.

Sometimes, though, these emotional tests can have surprising results. My colleague showed one of our designs to an elderly lady, and on seeing it, she burst into tears! The dog featured in the design just happened to look like her deceased pet. This shows that when it comes to design testing, you need to look for trends in results rather than individual comments. The flash test focuses on content and visual hierarchy. You show the design to the user for a few seconds and then remove it. You then ask them to recall items from the page.

The items recalled and their order gives a good indication of whether the design puts emphasis in the right places. For example, if a user fails to mention your primary call to action, then something obviously has to change.

Both of these tests can be carried out as surveys and, thus, include considerably more users. But speaking with users face to face has benefits. It gives you not only a better understanding of users, but the chance to follow up with questions and delve a little deeper.

⁴ Verify App, smashed.by/verify

⁵ WebSort, smashed.by/websort

USABILITY TESTING

Probably the best known form of user testing is *usability testing*. As the name implies, this tests the usability of your design, rather than its aesthetics. The subject has been written about extensively by the likes of Steve Krug, Jakob Nielsen and Jared Spool, so I will not go much in depth here.

I do want to emphasize that the success of usability testing depends on doing multiple rounds throughout the lifecycle of the project. For years, I left user testing until the end of a project, when it is too late to make changes. Testing at the end also leaves no opportunity to carry out further rounds to confirm the effectiveness of improvements.

For testing to be effective, it needs to be integrated in every stage of the redesign process. We need to test everything from sketches to mood boards to the final build. Doing one-off tests is not enough.

WHEN AND WHO TO TEST

The reason why testing is so often reduced to a token session at the end of the project is because of its perceived expense. People believe that finding participants and running sessions is time-consuming and expensive. Let me assure you that this is not the case.

Usability testing seldom requires demographically representative participants to yield valuable insight. Most of the usability hurdles we encounter are common to everybody.

Recruitment for survey participants needs little more than a link on the existing website. The exception to this is design testing, which does require a little more effort for recruitment. The number of participants needed is relatively high, and getting the right people does matter. But the tests themselves can be run anywhere and require no special equipment.

I have only ever attended one user testing session that was held in a proper test center with two-way mirrors and video hookup. In my opinion, it was actually less effective than guerrilla-style testing. By keeping the testing lightweight, you are much more likely to hold sessions regularly.

Ultimately, testing produces better websites and makes it easier to get client approval. That said, clients will still want to have their say, and so we need to be able to manage their feedback.

DEALING WITH CLIENT FEEDBACK

Although testing with real users helps guide the client away from personal opinion, they sometimes also need help providing the right kind of feedback on a design.

We make things worse by asking them, “What do you think?” and “Can you give me your feedback?” By talking in this way, we are encouraging them to express personal opinion. Instead, I ask clients a series of specific questions when seeking their feedback on a particular design. Some good questions are:

- Do you agree that the redesign reflects the organization’s brand values?
- Does the redesign meet our agreed business objectives?
- Is the redesign in line with the personality we discussed?
- Will this redesign encourage users to complete the agreed calls to action?
- Does the redesign reflect the approved mood boards and wireframes?
- Have we accommodated all pertinent feedback from the user testing sessions?

This is when all of your work up front pays off. Being able to refer back to approved mood boards, wireframes, user testing and so on focuses the client less on personal opinion and more on business objectives and user needs.

Fixation on personal opinion is not the only problem. Client feedback also often consists of a list of design changes they want you to make. Unfortunately, these changes are not always for the better.

I actively encourage my clients to suggest how a design can be improved. After all, I believe that clients can provide valuable insight, especially if you have been engaging and educating them throughout the design process.

Things become challenging when the client suggests a solution to a problem that they have not articulated. For example, a client may well tell you to change the color of the design from blue to pink without explaining that the reason is because the audience is pre-teen girls.

Without knowing the problem, you cannot suggest a better solution or judge whether the client’s idea is appropriate. Getting the client to express problems rather than solutions, therefore, is important.

When the client falls into the habit of suggesting solutions rather than expressing problems, a simple reminder often gets them back on track. Failing that, fall back on asking why. As I have said, asking why helps the client work back to the root of the problem.

The focus of this chapter has been on the logistics of working with clients to produce business-driven websites. We have looked at how to research and test your project, as well as the best way to work with clients. Before concluding, let's look at the long-term viability of the website you are building. In particular, how do we future-proof a redesign?

Redesigns That Last

I must confess that, as a Web designer, I have sometimes been short-sighted in my approach to building websites for clients, to the detriment of both my clients and my own business. I am sure I am not alone.

This short-sightedness is not entirely our fault. It is largely born of the culture of redesign that I talked about earlier. When the client commissions a new website every few years (often with a different designer each time), there is little point in planning for the long term.

I have explained the drawbacks of this approach, both for the client (in terms of the effectiveness of the website) and the designer (in terms of repeat business). Therefore, we need to establish an ongoing working relationship with our clients, rather than one-off engagements. This holds true whether we are realigning a website or redesigning it.

ESTABLISHING AN ONGOING PROGRAM OF WORK

As we have discussed, there is a strong case to be made for ongoing investment in a website. However, realizing something in principle and doing it in practice are different things. Putting in place mechanisms that ensure ongoing investment is important.

I have already suggested wish lists and phased development as ways to encourage your client to plan for the future. But those are not the only options. We also arrange quarterly calls with clients to discuss ways in which their websites can be improved. We also offer an annual website review, in which we suggest improvements for the coming year.

Whatever mechanism you use, whether email newsletters or annual reviews, the aim is the same: to demonstrate how the client's website can be taken to the next level. And remember that you cannot just show them new functionality or technology. You need to explain the business benefits it provides. Only then will they see the return on the cost of employing you to implement the idea.

Not that an ongoing program of investment is the only way to future-proof a website. There are also technological solutions.

USING TECHNOLOGY TO FUTURE-PROOF WEBSITES

We should remember why we do what we do as Web designers. If we forget, then bad practices start to creep in. A case in point is Web standards.

At the heart of Web standards lies a simple principle: that we should separate content, design and behavior. One of the many benefits of this approach is that it makes it easier to change a website in future.

Web standards help to future-proof our websites. And yet, I see an alarming number of websites built with CSS, HTML and JavaScript that do not make this clean separation: JavaScript either is inline or relies on the presence of certain HTML elements, and HTML is stuffed with classes and IDs whose sole purpose is to set the design. Don't misunderstand me: I am not a code purist. I know that a degree of crossover is sometimes inevitable. But as you code, ask yourself how the decisions you make will affect updates a year or two down the line.

Another issue that arises related to future-proofing is browser support. As Web designers, we talk a lot about supporting old browsers but little about supporting future browsers. We need to ensure that our websites are accessible on older browsers, but we also need to build for the future. Accommodating an old browser whose market share is only going to decline at the expense of support for upcoming browsers makes little financial sense.

I am a great believer in building websites using HTML5 and CSS3 because we know that these technologies will become more widely adopted. We are building for the future, not fixating on browsers whose market share is dwindling. Obviously, there is a balance to be struck, but one could argue that supporting older browsers is not the best investment of limited resources. Speaking of support, I cannot devote a section to future-proofing without mentioning mobile.

PLANNING FOR A MOBILE FUTURE

As Web designers, we get excited about mobile. However, mobile is not yet at the top of most clients' agendas. For many clients, mobile usage is still relatively low, and so they are unwilling to invest.

Despite this, there is little skepticism that mobile will play a significant role in the future of the Web. This means that even though the business case for mobile development is weak in the short term, clients need to start planning for it now.

The steps that need to be taken will depend on the mobile strategy preferred by the client. The client has a number of choices that we must guide them through.

First, they need to decide whether they need an app or a website. Apps are task-oriented and highly focused, while a website tends to be content-driven and much broader in scope. If the client decides on an app, then the next question is whether the app should be Web-based or native. This is a complex decision and not one we can fully address in this chapter.⁶

However, here are a few things to consider when discussing with your client:

- What native features do you need to access?
- Can you afford to develop for multiple platforms?
- Are you willing to share revenue with the app store owner?
- Will your app even be accepted by the app store owner?
- Is there a marketing benefit to distributing through an app store?

If the client decides on a website, then the choice becomes simpler. If the website will go through a major redesign, then now is the time to make it responsive (i.e. make it respond to the available screen space). On small screens (such as smartphones), the layout would become simple and touch-friendly, while on large screens you would have a traditional desktop layout.

If the website won't be redesigned soon, then an adaptive approach is preferable. Adaptive design is easier to implement on an existing website, while still enabling the layout to change at key screen sizes (such as the tablet's screen size). However, unlike responsive design, it will not adapt to *any* layout, so although an adaptive website might look great on today's devices, it will not be optimized for future devices.

The Web is undergoing a yet another major transition at the moment, encountering new technology, new devices and new ways of being used. It falls to us as Web designers to help our clients prepare for this brave new world.

⁶ You can learn more about mobile considerations in redesign in Aral Balkan's chapter of this book (p. 255).

Where From Here?

We have covered a lot of ground in this chapter, but gone into little depth. The idea was to get you thinking beyond technology and design, to focus on the challenges faced by clients and ensure that their websites are as effective as possible. What you need to do now is put some of these principles into action. Here are some suggestions:

- **Become more than an implementer.**

Work hard to change your relationship with clients. Stop being a pixel-pusher, and work collaboratively with clients. Be willing to challenge them, especially when they request a major overhaul of their website. Suggest a realignment instead, and adopt a process that includes them.

- **Prepare before jumping in.**

Resist the urge to leap into a redesign; rather, do some homework first. Make sure you are well prepared for the risks, such as scope creep. Ensure that you understand the business through stakeholder interviews and reviewing the current Web presence.

- **Test everything.**

Do not rely on your experience as a designer. Test as a way to make the design process less subjective and to justify change. Testing will also help quantify the potential return that a client sees from your work.

- **Plan for the future.**

Work with your clients to establish an ongoing program of development that takes their website into the future. Encourage them to plan for the mobile Web and for future browsers, not old technology whose market share is declining.

So, there you have it. Hopefully, I haven't completely demoralized you with all this talk of ROI and business drivers. Actually, this can be exciting stuff. We are designers, not artists; the main difference being that we produce things that solve problems. Sometimes they are user problems, but usually they are problems that our clients are experiencing.



About the Author

Paul Boag (1972) grew up in Washford, England, attended the University of Portsmouth and initiated his Web career at IBM in 1994 when Web design consisted of Notepad, gray backgrounds and no layout options. He is the co-founder of Headscape, through which he works on Web strategy, writes about successful websites and speaks at conferences around the globe. Occasionally, he releases podcast episodes. Paul lives in a small rural town in the heart of the English countryside, where he is heavily involved in the local church. He loves to spend time playing Skyrim. Paul considers himself all pet'ed out because his father is a wildlife photographer and they had everything from owls to deer passing through their home over the years.

His favorite colors are all shades of gray, and the biggest lessons he has learned in his career are to be passionate and enthusiastic, never to lose one's love of the Web and to play with innovations. Paul's message to readers is that undertaking a successful redesign has to be a collaborative approach with your client. You may not find your client easy to work with, but without their knowledge of their business and sector, the website will fail. What's more, the client has to love the design. If they don't love their website, they will not invest in it or use it to its full potential. You have an obligation to work collaboratively.



About the Reviewer

Collis is a co-founder and CEO of Envato. He started the company as a Web designer, Photoshopping and slicing and dicing most of the early Envato websites himself. These days, Collis spends more of his time planning, strategizing and emailing, but Web design will always be where his heart is!



Selecting a Platform: Technical Considerations for Your Redesign

Written by Rachel Andrew

Reviewed by Ryan Carson and Harley Finkelstein

THIS CHAPTER IS FOR ANYONE INVOLVED in planning the redesign of a website. In the previous chapter, Paul Boag discussed the business behind redesigning or “realigning” your website. In this chapter, we will look at some of the technical issues you might encounter as part of this task. A redesign of the user interface of your website might not require an entirely new back end, and we’ll give you some things to think about when making that decision.

Following Paul’s advice about research, we will need to learn everything we can about the current website and the back end that drives it before jumping into the redesign. Throwing away a lot of knowledge when replacing a system is very easy, so even if you are completely redeveloping or moving to a new platform, learn all you can from the existing one and avoid replicating the mistakes that previous developers have made and already fixed.

Before making any decisions, you need to understand all of the technical requirements. Perhaps your redesign will bring new functionality that needs to be supported. For example, if you are adding e-commerce functionality to the website, you will need to understand how to take payments on the Web and what services you will need to use in order to do so. You will have to be satisfied that your hosting arrangements meet the requirements of the new website or of your development of the existing platform, so we’ll also think a bit about hosting, what you’ll need and how to choose a good host.

The beginning of a project is a good chance to take stock and make sure that all of your working practices are in good shape. So, we’ll finish this chapter with some thoughts on development environments, version control and how to deal with replacing a live website with a new version. As in the last chapter, we’ll be covering a lot of ground here, and space limits us from doing so very deeply. Rather, our aim is to get you thinking about the different aspects of your website and to pick out the bits that apply to you as a basis for further thinking and research.

Who Is This Chapter For?

This chapter will be useful to you no matter where you fit on the team that will be redeveloping the website. Whether you are developing your own website, outsourcing development to another company or working on a team with developers (perhaps in a non-technical capacity), understanding the process will be beneficial to you.

Perhaps you are the person actually doing the development, either of your own website (in which case you are the client) or of a website for a client or employer. In this

case, all of the information in this chapter should be of interest to you. Redesigns are a fantastic opportunity for the developer. Your existing website or application will have strong and weak points, but it will also have an existing user base and traffic that you can study. Ignore this history and data at your peril; you might end up simply recreating existing problems or not including important features that users rely on.

When redesigning a website, you have a chance to find out from users and the owner what the current problems and pain points are. Is the content management system (CMS) so difficult to use that no one updates the website any more? Do customers of the online store constantly phone for help to place an order? Does the design constrain the addition of text or images? If you fix these problems—by creating a CMS that people actually enjoy using, or by halving the calls from confused customers—then your client or boss will be hugely appreciative of your work. Hearing that your work has made someone else's daily life better or has improved their business is very satisfying.

If you will be outsourcing development of the website, this chapter should still be of interest. We won't be covering the more technical topics in great depth. Being able to converse with your developers with a common understanding of the issues involved should make communication easier.

Perhaps you are serving in a non-technical role; for example, working purely on design or doing project management or copywriting. Likewise, understanding what the developers are doing will only help with communication.

Learning From the Existing Platform

As mentioned, learning everything you can about the website you will be replacing is vital. This involves assessing the existing website but, more importantly, speaking with the people who use it. Users include the website's visitors as well as the people who own and run the website.

In assessing the website, find out what it does well. This could be anything: perhaps the client really likes the look of the website and feels it represents the brand well, or perhaps it performs well on search engines, or perhaps users will tell you that they find the website easy to use even without all of the bells and whistles being requested by the client.

When speaking with the people who add content to the website, find out if they consider certain features of the current system invaluable. If you are considering replacing the platform, take note of the things that people use to do their job. Many people

build their entire workflow around a system, and if you take away their ability to run a certain report on orders or to edit a blog a certain way, then you could make it hard for them to do their job. If you don't replace that functionality, then you had better present a superior system!

Every system has something that drives everyone who uses it insane. But even if you were involved in maintaining the current website, don't assume you know what that thing is. Time and again, I have seen non-technical users assume that a bug in the system was their fault and so do not report it. Every time the system fails to save some data, they think, "I've done it again!" and then repeat the task. Rather than raise the problem, they mark it down to their inability to use computers.

Ultimately, even if the user is at fault, if the system allows work to be lost on a regular basis, you might be able to put something in place to prevent this from happening. And yet glaring bugs often go unreported for months, with users simply working around them. If you will be keeping the existing system, then finding and fixing these pain points will really benefit the people who use the system every day.

For systems that people spend a lot of time working with, it is worth seeing those people's workflows. What jobs do they perform on the system day to day? Very often the person doing the job is not aware that a new system could actually take away some of the grunt work in their job.

I have often seen people do repetitive data entry—for example, entering the same information in several places or copying data from one report to another—when auto-generating a report or adding a script to the CMS that copies data from one place to another would be very simple. By looking only at a content-managed website or e-commerce system, you would not see these workflows; so, if at all possible, sit down with the administrator and see what they do.

If you are considering a redesign, then the challenges are probably not just visual; rather, you need the website to do things that it currently cannot do. Do you want to start selling items online? Have you decided that the website finally needs a CMS? Is the current CMS difficult to use, or does it not support the type of content you want to create?

Knowing what it cannot do will give you something to go on when deciding whether to replace the system or build on top of it. Once you understand how the website or application serves users and administrators, you can move on to gathering the technical requirements for redevelopment.

Gathering the Technical Requirements

The previous chapter looked at website requirements. In this chapter, we are looking at how to fulfill these requirements technically. Be wary of any specification that says, “We want it to do everything that the current system does”—unless you built the current system! If the client wants that, make sure to get the details of everything that will be required in the new system.

Failing to do so will almost guarantee that, just as you’re close to launching, you will be asked how the system supports a task that half of the staff does every day (when, in fact, it doesn’t), thus forcing major additions to your project. I speak from experience!

CONTENT MANAGEMENT

Almost every website requires some form of content management. This could take the form of updating pages of content, adding products to a store, or editing bits of text in a Web application. How much management over the content will the client need, and who will be editing the content? In this chapter, we will use the term “CMS” very loosely to describe any tool for editing content—be it a simple editor for changing text to a full-featured enterprise-level CMS.



Figure 2.1. Someone choosing a CMS will face dozens of options. The only way to decide what will work for your project is to understand the requirements.

Here are some considerations to make when choosing a CMS:

- Will all content editors need the same level of permissions?
- Will the content management be done by dedicated editors or as part of other people's jobs?
- Do multiple languages need to be supported?
- What type of content will be edited?
- What kind of editing environment is needed?

Will all content editors need the same level of permissions?

Will there be one editor (e.g. the owner of a small business), or will several editors need to work on the website? If the latter, will the editors have equal privileges, or will some be able to access parts of the system that others cannot? Here are some scenarios:

- The website for a large company is managed by one person, who signs off on all content, although several people produce content for the website. The client would like these editors to be able to create content and submit it for review. Once approved, the content would be published by the managing editor.
- A company wants its HR department to be able to publish and take down posts for job openings at the company and to manage the section of the website that deals with HR issues. These users should not be able to change other sections of the website.
- The owner of an e-commerce website does not want their content editors to be able to view sales reports or customer information collected by the system. Conversely, they want to give their accountant access to sales figures but not to the content on the rest of the system.
- The owner of a website wants volunteers to be able to post to the blog but not to change content in other parts of the website.

Will the content management be done by dedicated editors or as part of other people's jobs?

Understanding the abilities of the people who create and edit the content is important. This refers not only to technical ability, such as whether they are familiar with using a CMS or are even technologically literate, but to whether they have design sense

and would expect some measure of control over how things look. Also, consider their ability as copywriters. If the content will initially be written by a skilled copywriter but then maintained by the owner or by an employee who is not a copywriter, then the CMS could help them figure out what to write and how to write it. I have addressed this in the article “Your CMS as Curator of Your Design and Content”.¹

Even if the website will launch with only one language, but support for additional languages will be required in the near term, then provisions should be made for these languages. Retrofitting a website to support multiple languages is much harder than building in support from the start. Of course, this requirement could narrow your options of off-the-shelf software.

If a website is to be translated, then find out how the translation process will work so that you can support it in the system. Will translators simply work in Word documents or the like, translating all of the content and then sending the documents back to the editor for inputting? Would it be more helpful if the translators could read and translate the content directly from the CMS?

What type of content will be edited?

Most websites that require a CMS have relatively static pages and a fairly standard tree-like structure. The most logical way to organize this content is based on pages, so that administrators can easily find the pages they want to edit.

Some websites essentially have a blog as their main feature, with some supporting pages. So, you and the client might decide to use a CMS such as WordPress, which has a strong blog at its core but with the ability to add traditional pages.

Larger websites will have more complex requirements for content. For example, we are currently involved in redesigning a website for an arts festival. The festival is in the fortunate position of having many years’ worth of video and audio material and thousands of high-quality photographs that were taken each year by professional photographers.

While much of the website is page-based, expecting content editors to sift through an offline archive and re-upload relevant material every time would be very inefficient and not the best use of the material. Instead, we have created a separate media server, with tools to search through and tag these resources, making them easy to find and embed on pages. The CMS also prepares the images at the correct sizes, including the sizes needed for the website’s responsive design. Learning about this archive of mate-

¹ Andrew, Rachel. “Your CMS as Curator of Your Design and Content,” smashed.by/cms-curator

rial and the problems inherent in dealing with it enabled us to propose such a solution. Other websites could be described more as Web applications, be they e-commerce systems or traditional Web apps. These systems are not focused on page-based content (although they might have some), but rather on various types of content that need to be easily updatable. All too often, all of the microcopy is hardcoded into the application itself, so that changing the text in a call to action would require work of the developer. Ideally, this text would be handled in such a way that non-technical users could change it.

A/B TESTING

With many websites, particularly those that sell products, you will need to test different versions of pages in order to see which content, layout or flow leads to the most conversions. Depending on the type of website, a conversion might be someone buying a product through the website, signing up for a product trial or a mailing list, or filling out a form. If this kind of testing is required, how will that be managed in the CMS? If you need to send some visitors to one version of a page and others to another, then you will need a strategy for this. If you would like to learn more about this type of testing, there is a comprehensive article on “The Ultimate Guide to A/B Testing” on Smashing Magazine.²

E-COMMERCE

Adding e-commerce functionality to a website can be as simple as adding a few PayPal “Buy now” buttons or as complex as rolling out a large third-party store or developing one yourself. Selling products directly from a website doesn’t need to be hugely complex these days, but you do have a lot of options to consider depending on the type of product. In this section, we’ll run through some of the options to think about. This will be particularly helpful if you will be implementing e-commerce functionality for the first time, because it can feel like stepping into unfamiliar land.

What are you selling?

Perhaps your online store sells physical products that will be shipped to customers via postal service or courier. Or perhaps the products are delivered electronically, such as eBooks, music or software. Donations and subscriptions are types of transactions to consider as well. If the products are downloadable, then consider how they will be delivered to the customer upon payment.

² Chopra, Paras. “The Ultimate Guide to A/B Testing,” smashed.by/abtesting.

What will the shopping experience be like?

Will only a single item be sold (such as an eBook) or will visitors need to be able to browse products and add multiple items to their cart? Do the products come with options (for example, size and color for t-shirts)? Are categories needed to make browsing easier? Should an item be restricted to one category or be found in several? Would tags be useful, or links between related items (say, to allow the owner to promote accessories for a product)?

Will the website have special offers: “Buy one, get one free”, “20% off”, “Two for the price of one” or “Buy X, get Y at half off”? Setting up these kinds of offers on a custom-built system can be quite complex. And if you will be using an off-the-shelf CMS, then you will need to know whether it supports them.

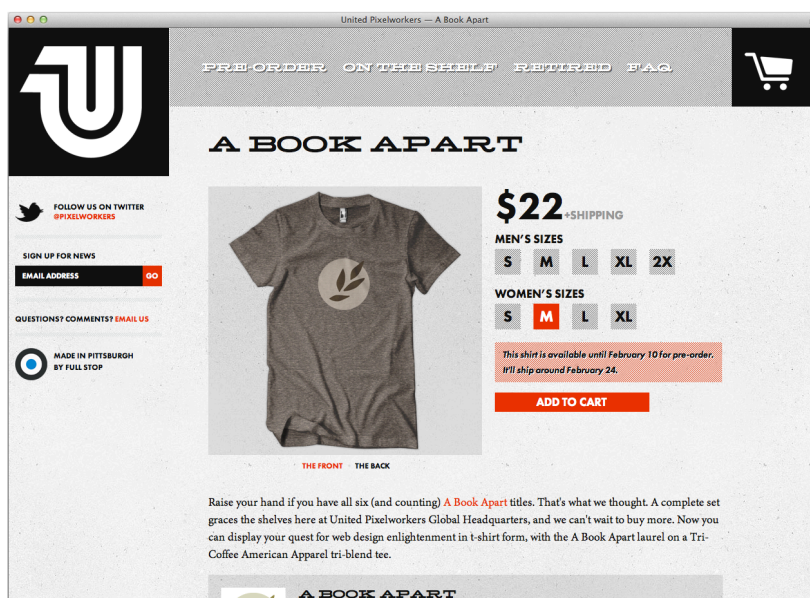


Figure 2.2. What might seem like a simple product can have a number of options; for example, t-shirts in men’s and women’s fits and in various sizes.

The devil (and the budget) is in the details. Of all the websites I have worked on, the type most prone to scope creep is online stores. Ask all of these questions as you plan the process. Wanting every possible feature is very tempting, but if you are building the CMS yourself rather than using third-party software, you can cut development time significantly by doing things more simply.

Accounts and tracking orders

Part of the user experience could include managing an account and tracking orders. Must users create an account, or will it be optional? Can they track their orders and watch them move from “Processing” to “Shipped”? Accounts should include basic management functionality, such as resetting forgotten passwords and updating contact details.

How will you take payment?

You will likely need to accept credit- and debit-card payments from customers. There are a number of options, varying in complexity and cost. Common ways to process payments include PayPal, a custom merchant account and payment gateway, and third-party software as a service (SaaS).

Taking payments online with PayPal is straightforward. The advantages are that creating an account is easy, no credit check is needed, and integration can be as simple as hardcoding a button on the page and as elaborate as full integration. Google Checkout offers a similar service (and a similarly low barrier to entry), as does Amazon (in the US) through Amazon Payments.

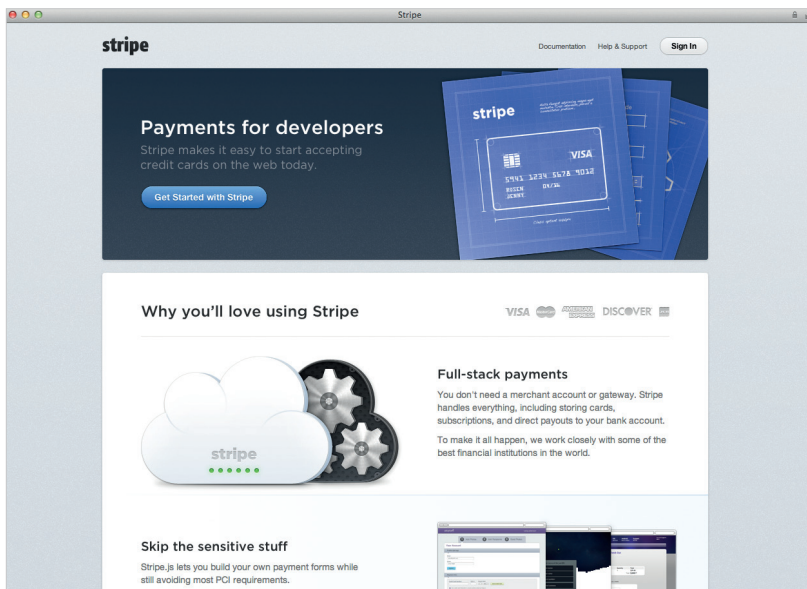


Figure 2.3. Stripe.com is a new player in the market, offering an easy way to take payments.

The payments market has other players, but many of them operate in only a few countries, the reason being that many of the laws for processing payments are country-specific. If you need to take recurring payments, then Chargify and Recurly might be useful. And although currently US-only, Stripe looks promising as a simple method to take payments online.

To accept card payments directly, rather than through a third party, you will need an Internet merchant account. This enables you to take credit-card payments and to process the money to your bank account. If you have an existing merchant account for brick-and-mortar or telephone sales, you might not be able to use it for online transactions. Internet transactions are riskier, so to start transacting online, you'll need to contact your bank. The bank will require that you adopt measures to secure payments, in most circumstances via a payment service provider (PSP, sometimes called a *payment gateway*).

What you should definitely *not* do is store credit-card details in order to enter them in an offline terminal later. This would be against the terms of the merchant agreement. So, unless you have written permission from your bank to do this and are complying with the PCI DSS (which we will discuss later), just don't.

THE PAYMENT GATEWAY

The purpose of the payment gateway is to enable you to take a card payment from a customer, validate the card number and amount, and then pass the payment to your bank securely. You can interact with a payment gateway in one of two ways:

- **Pay page**

The user moves from your website to a secure page on the payment gateway server to enter their details.

- **API integration**

The user enters their card details on your website (on a page installed with a secure certificate, running SSL), and those details are then passed to the gateway. Your website acts as the intermediary; the user is not aware of the bank transaction taking place, having seen only the transaction on your website.

The advantage of integrating a pay page is that your website never touches the card's details, so you are not liable for the customer's security. The most significant disadvantage is that you lose some control over the payment process, because the final step

requires gathering all of the details to pass to the payment server. Also, you are seldom able to customize the payment screen, even with just a logo.

Many store owners are concerned about this break in the user experience; they fear the user will abandon the transaction before reaching the payment page on WorldPay or another server. But transferring the user to the website of a known bank to enter card details might actually instill confidence in the legitimacy of your website.

When an unfamiliar website (perhaps that of a small retailer) asks me to enter my card details, I immediately worry about how it will handle them. Will my card's numbers be displayed in plain text? Will the details be stored in a database on the website's server somewhere? Even if the page has a secure certificate and checks out, I still have no idea what will happen to my details after I hit "Submit." But if I am taken to a known PSP page in the final step, then I can be confident that my details are safe and that the unfamiliar website isn't handling them at all. I would trust WorldPay with my details far more than Joe Blow's Widget Store.

Another advantage of a pay page is that, should the regulations for card payments change, they will be handled by the PSP. For example, 3-D Secure (which is the protocol underlying Verified by Visa and MasterCard SecureCode) was a requirement of one of our clients so that they could accept Maestro payments. 3-D Secure requires that users verify their payment on a page from their bank before the payment can be authorized.

If we had used an API, we would have had to edit the code in order to integrate 3-D Secure; but because we used a payment page, we simply notified the PSP, which switched on this functionality for the account.

These points have swayed many website owners to use a pay page, with many owners recognizing that being responsible for credit-card details is more trouble than it is worth.

Integrating a pay page should work with most off-the-shelf software. After a payment is made, the page typically sends back something that enables your website (which has a script running for this) to identify the user and the transaction and to process any post-purchase data that may be needed (such as marking the order as "Paid" in the database or giving the customer access to the digital product).

The advantage of full API integration is that you control the payment process from beginning to end, including the look and feel of the payment pages. But you are also responsible for the security of the user's card details, and regulations require that you prove you are following best practices.

PCI DSS

The *Payment Card Industry Data Security Standard* (PCI DSS) is a set of 12 requirements that all businesses that accept card payments must comply with. This covers not only online transactions; a brick-and-mortar store that takes payments online must also comply with the PCI DSS for both its offline and online payment methods.

If you are merely taking online payments via a pay page and do not receive, process or store any card details at any time, then you can complete the shortened PCI DSS questionnaire (SAQ A) to confirm that your PSP is PCI DSS-compliant. If you use API integration, then you will need to comply fully with the PCI DSS (even if you do not store card details), including by allowing quarterly security checks that verify ongoing compliance. Explaining PCI DSS compliance in detail is beyond the scope of this article, but if you are involved in developing a website that will take card details without a pay page, then you should acquaint yourself with it, or retain the services of someone who knows it.

Storing card data

I strongly advise against storing card data on the client's server, even in encrypted form. Doing so would require you to comply with the PCI DSS and to maintain a server and network capable of keeping this data safe. If you need access to card data in order to charge for recurring fees, for example, you could look into payment gateways that offer data-storage services.

If you are considering storing card data only to be able to offer “one-click” ordering (as Amazon does), please be careful. Do you really want to be liable for your customer's data? Are you willing to deal with the extra and ongoing expenses of maintaining compliance?

Multiple currencies and local taxes

You will likely need to account for local taxes, or VAT in Europe. Understanding exactly what taxes to collect can be difficult enough, but you also need to ensure that your system can process them correctly. For example, my company has a downloadable product, a mini-CMS named Perch. Our company is registered in the UK, so we need to collect VAT from UK buyers. We also need to collect VAT from European Union buyers unless they have a valid VAT number. If the buyer lives outside of the European Union, then we do not need to charge VAT. So, our system has to be able to validate VAT numbers as well as correctly calculate prices with and without VAT.

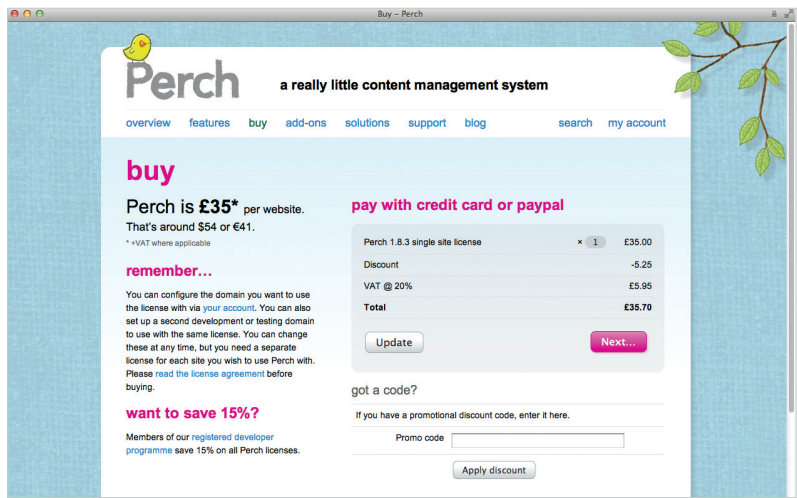


Figure 2.4. Perch’s payment page, listing the VAT, a discount and approximate pricing in Euros and US dollars. Despite only a single product being sold, several variables still need to be taken into account.

Most stores take payment in a single currency. To accept multiple currencies—that is, to allow visitors to select their currency, see the price in that currency and make their payment—you will need to set the required currencies in your merchant account. Another option is to display prices in other currencies while accepting payment only in your local currency; you could either update the exchange rates manually or automate the process with an API. If customers may pay only in your local currency, then they will need to understand that the conversions are displayed solely for information and that the actual price might differ slightly (owing to fluctuating exchange rates).

Getting advice from an accountant when dealing with money is always worthwhile, especially if you will be accepting payments in a foreign currency or in multiple currencies. Getting advice from the start on how to handle payments and exchange rates could save you a lot of trouble later on.

WHAT ABOUT DELIVERY?

If you are selling physical products that need to be shipped, you will have to somehow charge for shipping costs and perhaps arrange for order-tracking. Because you are selling online, you could attract customers from other countries, so you will have to decide how to calculate shipping to different destinations, or else limit shopping to people in your country or a few countries.

Typically, online retailers offer free shipping on orders of a certain price or higher. They also typically offer shipping with different carriers, such as the regular postal service and priority courier (depending on when the customer wants to receive the product). Consider these things when planning your website.

DIGITAL PRODUCTS

Customers expect to be able to download digital products (such as eBooks, music and software) quickly after purchasing them. Delivery could take the form of a Web link or a downloads page in their account profile (along with a license code if required).

The system will need to be able to secure products prior to purchase and provide an area in the customer's account where the product can be accessed (or at least to send a link by email). Product codes might also need to be generated. Again, third-party systems can handle this as part of a bundled payment service.

REPORTING AND OTHER FUNCTIONALITY

Your client will want to process orders as soon as they come in, and perhaps also mark items as shipped once they have been processed. Being able to load this data into some form of CSV file to be merged onto mailing-address labels would likely be helpful, as would being able to export payment information to an offline accounting system.

Here are some other questions to ask yourself or the client:

- Do you need to be able to control stock through the website?
- How do you want to deal with orders that can only be partially fulfilled?
- Should the website generate invoices, or will this happen offline?

INTEGRATION WITH OTHER SYSTEMS

Many websites do not exist as standalone systems but rather are integrated with other systems and services. Integrating log-in credentials from third-party networks is fairly standard, and you might also need to communicate with third-party systems for stock control or accounting, particularly with e-commerce stores.

A few years ago we developed a website for a university that enabled students and staff to update their details and to request certain forms from the university office. Therefore, log-ins to the website needed to be secure. For a typical website, we would

use our own authentication classes with the CMS; in this case, we were required to work with the existing credentials of students. This information was stored in LDAP, so we had to have the website authenticate users against the university's LDAP server. This entailed writing a new interface for our standard CMS authentication system that authenticated via the LDAP server. But when working with a third-party system such as this, actually writing the code is usually the easy part.

A lot of time was spent getting access to the LDAP server and finding out what was needed in order to confirm a log-in. Because we were deploying our own CMS and could write a new interface for the authentication, the actual coding part of this job was straightforward. A requirement like this, though, could effectively rule out certain third-party CMS' if there is not an easy way to change how authentication works while leaving the software in an upgradeable state.

For another job, we developed a custom e-commerce system for a clothing store that wanted to sell its products online. The store sold limited-edition designer clothes. For many of the designs, only one shirt per size was available. And because the shirts could be sold both in the physical store and online, the inventory had to be kept up to date for both; an item sold online had to be immediately removed from the rack in the store, and if an item was sold in the store the website had to be updated.

This set-up has obvious problems (an item could very well be sold online just as someone comes to the till to pay for it in store), but the best we could do was ensure two-way communication between the website and the in-store electronic point-of-sale (EPOS) system.

The EPOS system was developed by another company, and so to link these systems we needed to work with those developers. But any request made to this company could mean a three-week wait for them to gather the resources to fulfill the request; this was out of our hands, but it obviously had a big implication on the schedule of the project.

Being very clear in our communication and providing full details as well as a testing interface helped to minimize the risk of delay. But if you know you will have to work with a third party in this way, get details on the way they work and the expected turnaround time for requests. This way, you can plan around their schedule.

Understanding from the outset all of the third-party systems that you will need to work with is important. These requirements could well affect your choice of technology and off-the-shelf software. You will also need to account for them in your and the other companies' timelines.

Constraints

Every project has some constraints. Personal ventures aside, most projects tend to be limited in time, budget, available skills and a myriad of other factors. Work out what these are before making any technical decisions, because you may find that the constraints close off some paths.

BUDGET

When thinking about budget, also consider time. Employing an external developer or company to build the website will put further restrictions on the budget and timeline within which you are able to complete the work. Even if the third party gives a firm quote, it will fluctuate according to the amount of time they take for the work. Developing the project internally using your own resources also comes with budget and time constraints—whether it's the deadline for launch or simply a need to account for the time spent by your team on the project.

In addition to pure development costs, you might need to take into account third-party costs, because we are increasingly relying on third parties to provide resources for our websites. Will you need to purchase stock photography or fonts? The hosting costs could increase if your current provider does not meet the requirements of the new website. An e-commerce store could incur additional charges if you move to a different payment method.

Third-party software could bring licensing fees, although it should reduce development time. You might also have to pay monthly charges for hosted services. Account for as many of these costs as possible in the budget at the beginning of the project.

PROGRAMMING LANGUAGE DECISIONS

Perhaps your internal team or external development agency is skilled in a certain development language. This will likely determine the language on which the website is based, unless your reason to use something else is incredibly compelling. Good developers have a lot of pre-written code and ways to solve particular problems in their language. Switching to another language just because it's hot will require a lot of retraining and additional time, because the developers will have to write new libraries for basic things.

Choosing a language other than the one your team is skilled in would be justified if moving the project forward with the current language would be difficult (whether

because few developers know the language or because its development has stalled). Say your system is written in Classic ASP, and the team that has been maintaining the system for years knows Classic ASP. Because this language has been superseded by ASP.NET and is no longer under active development, using it to build the new system would not be sensible; after all, finding new developers who know Classic ASP will be difficult, and the language, being so old, is not suited to the things we, as do on modern websites.

If you find yourself in such a situation, then retraining your team in the new language in preparation for the redevelopment would be the sensible approach. If the external agency suggests that you rebuild in Classic ASP, then speak with them about why they are pushing for this technology, and explain that being tied to outdated technology for a website that you hope to last for a few years is not forward-thinking.

If you are developing the website yourself and you feel drawn to learning a particular language, then as long as you can manage the delay in the project as you learn the language, the choice is yours. Learning new things is always fun, but don't drift from one new thing to another just because everyone is talking about it. Knowing one language deeply will always serve you better than skimming the surface of several.

If the project is for a client, then they will likely impose some constraints on your time and budget. The constraints could also be technical (such as requiring you to host internally on a particular server architecture) or political (for example, requiring you to use open-source software).

TECHNICAL CONSTRAINTS

If the organization hosts its own websites, then it will probably have its own server architecture. In this case, you will need to find out exactly what you will be deploying to. For example, the server could be Windows, and the company could have or be able to install PHP; this will save you trouble when deploying, because there are some differences between PHP on Linux and PHP on Windows (particularly if the server is run on IIS, rather than Apache).

If the organization runs its own servers, then find out if you will have access to test and deploy the application. In some projects, I was given no access at all, instead having to package the code and databases and send them to an internal IT technician for deployment. If that is your situation, then you will need to be able to replicate the hosting environment for your own testing, because testing and making changes in a live environment are difficult and time-consuming.

Having to integrate with any other system is, as stated, a constraint of some kind, and certain third-party applications could be ruled out if you are not able to write plugins to integrate them.

POLITICAL CONSTRAINTS

You might come upon constraints that derive from some organizational policy or the preferences of key stakeholders. For example, a policy could dictate that any third-party software be open source or that only Microsoft technology be used.

As an aside, the term “open source” is often misunderstood. When people specify that software must be open source, they usually do not mean that it should have an open-source license or even be free of charge; what they really want is to be able to modify the code if required, so that they are protected in case the provider goes under or abandons the product. If open source is a requirement, clarify what is meant by that. Many commercial products have unencrypted and modifiable sources, despite not having open-source licenses.

You can sometimes overcome political constraints by arguing why one solution would be preferable to the one being requested. But you might have to make a strong case, because if the stakeholder’s belief about a certain technology is deeply held, then they will automatically be more skeptical of yours.

Should You Refactor or Rebuild?

The decision of whether to start with an entirely new back end for a website is never an easy one. Even if the existing system has many problems, it can seem as if you are throwing away a lot of money and effort by replacing it. In this section, we will look at reasons why we might maintain a platform or build an entirely new one.

The temptation for a developer is to march in and build something new. After all, who likes picking at someone else’s code? We all have our own way of doing things; we have our own coding standards, and there are certain frameworks and systems that we know well and trust. But by throwing everything out and starting over, we could lose a lot of the knowledge built up from the existing system. If the website is for straightforward content management, then this is less likely to be an issue than if it is a complex intranet or an e-commerce system to which a lot of small elements have been introduced over time to solve particular business needs.

Ask yourself whether the current system really is so ill-suited to your requirements that it should be completely replaced or whether it simply does not fit the way you prefer to work.

Also consider the experience of those who are using the system. If many people are adding content, updating products or performing other tasks with the software every day, then they would need to be retrained. Refactoring and improving what is already there instead, perhaps in stages, would save you that work.

Not to mention budget and time constraints. Starting over would mean that the client would have to pay the full cost, both in money and time, to develop a new website. Refactoring enables you to spread those costs out, rolling out changes as they are completed.

But there are times when rebuilding from scratch makes sense. As mentioned, if the developers are new to the website, then they would likely be happy to replace the system with something they know well.

If you have hired an in-house developer or team or have started collaborating with a third party with whom you intend to work long term, then moving to a platform that they are experienced in could be reasonable.

As also mentioned, perhaps the coding language is outdated, or the website relies on abandoned software, or the functionality you want to add would strain the current framework. Avoid throwing good money after bad.

Another reason to discard the existing platform would be if it is making design decisions for you. For example, a third-party e-commerce platform that imposes a template design and user flow on the purchasing process could prevent you from making improvements that you know would boost sales. Spending time identifying problems and coming up with solutions, only to be prevented from implementing those solutions, is frustrating for everyone involved.

Ultimately, deciding whether to start over or to refactor will require looking long and hard at the current system and seeing how it aligns with current and future requirements.

Custom or Third-Party System?

Assuming that you decide to replace some or all of the back end, you will need to figure out whether to develop a custom solution or rely on third-party software and services—or perhaps a combination of both. In this section, we will assess the advantages of each approach.

WHY GO CUSTOM?

If your requirements are specific or unusual, then a custom system might be the best way forward. Third-party software has to appeal to a large market in order for it to be worth the developer's time; also, adding a lot of features to make the software more appealing to more people is tempting. The problem is that the system could end up doing hundreds of things that you will never need. Unless it has been designed well, such functionality could slow down the speed of your website, hanging around as unneeded elements and adding overhead.

If your website gets a lot of traffic, and you know from the start that every line of code in both the front and back end must be optimized, then developing your own solution will enable you to consider this throughout the stack. My personal bugbear is the inefficient use of SQL; some applications make hundreds of unnecessary database queries, which becomes a problem under heavy load.

It may be that the license for some third-party software is inappropriate for the application you are developing. It may be that the system has almost everything you need, but adding that 10% of remaining functionality would require hacking fairly deep into the software's core. The problem with this is that getting support from the software's developer might be impossible if you have hacked the core, and upgrading will be difficult, too. Making your changes through an official API is preferable; if you cannot do that, then avoid hacking the software, because what you end up with will be hard to maintain.

The advantage of designing and developing your own system is that you can tailor it precisely to your requirements. Using off-the-shelf software would probably entail some compromises, however well-written and customizable it is. Assess whether the advantages of using something ready-made outweigh the benefits of tailoring something precisely to your needs.

WHY CHOOSE OFF THE SHELF?

Off-the-shelf software has obvious benefits. Instead of starting with a blank slate, you can get something up and running relatively soon. If you need an easy way to manage content or run an online store, and your requirements are fairly straightforward, then one of the many solutions out there will likely meet your needs.

Some third-party software is free; others come with a licensing fee. With some, the core product is free but you have to pay for add-ons. These can add up quickly, so think about which add-ons you might need to complete the project. One system might include

everything in the licensing fee, making it cheaper than a free alternative for which you would need several commercial add-ons.

Also find out how much support you are entitled to. With a lot of free software, official support costs money or does not exist. Either way, there is probably a community forum where users answer each other’s questions. Look in the forum to see whether questions are answered quickly and how many people post entries.

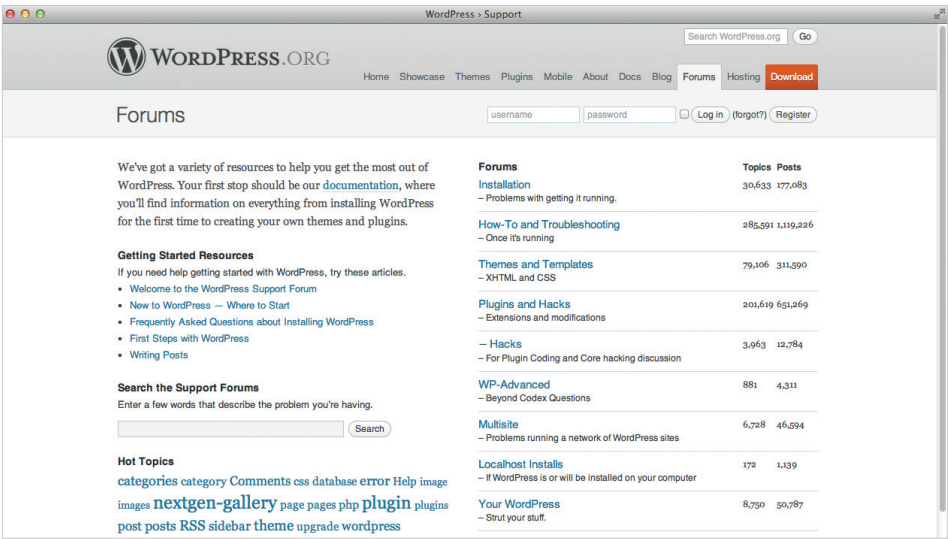


Figure 2.5. WordPress has busy forums for people who need help with the software.

If support is included, find out what form it takes. Are there restrictions on it? Is there a waiting time? If there is a public support area, are questions answered promptly by the staff? Twitter is also a good way to find out about the support and community around a product; simply ask for opinions and see what response you get. A paid product might be more valuable than a free alternative if the support is good.

HOSTED SOFTWARE-AS-A-SERVICE SOLUTIONS

A third type of solution to consider is hosted services, also known as *software as a service* (SaaS). These are CMS’ that could serve, for example, as e-commerce solutions, ranging from a hosted basket and payment page right through to a full-blown store. The solution is attractive because you do not need to install anything to get up and running; it might be a case of just configuring some options and altering a template.

Using hosted services can be a quick way to get your website up and running. Particularly with e-commerce projects, hosted services offload the worries of security and complicated functionality to a company that specializes in it. But there are some things to consider.

The client will probably have to pay a monthly charge to use the service, so this should be factored into the budget. Check whether the price varies according to your traffic or number of orders.

Also, you cannot change how a hosted service operates. If the service changes something that affects your business, you have to live with it. You cannot code around the problem the way you would with software that is hosted on your own server and whose source code you have access to.

Your website will also rely on that third party sticking around, not going out of business, and continuing to offer the service at a price that the client can afford. With software installed on your own server, even if the provider goes out of business, you still have the software; support for the software would probably discontinue, but you could at least transition to something else in your own time. If you will be relying on a hosted service, research it carefully to ensure that it is robust and that the company is stable.

BEWARE THE EVANGELISTS

Users of certain products exhibit curious behavior sometimes. You can see it in action by going to any public forum about Web development and simply asking, “What is the best CMS?”

This question is practically useless, because it does not shed light on the problem you are trying to solve. The reader has no idea whether your website is tiny, with only a few bits of editable text, or vast, comprising thousands of documents and in three languages. Yet within minutes, someone will tell you what their favorite CMS is and then cheerfully argue why it is the best under any circumstance. These people are evangelists: they love the software they use and cannot imagine why anyone would use anything different. I’m a CMS vendor myself, and so I sometimes benefit from the evangelists among our users, but I am also the first to say something when our solution does not fit someone’s project.

When researching any third-party solution, whether an entire platform or just a script, beware of the evangelists. Ask specific questions based on your requirements, and you will likely get a better quality of opinion.

Hosting

We are frequently asked to recommend hosting providers and how to choose a good host. There are thousands of hosting providers, offering what appear to be very similar services at widely different prices. How does one pick? Should you just choose the cheapest provider that has the specifications you need?

CHEAP HOSTING IS EXPENSIVE

Many hosting providers market purely on being the cheapest; hosting can be found for as little as \$10 a year. But just because such cheap hosting exists does not mean you should use it. It could cost you a lot more than the price of reasonable-quality hosting in the long run. A good host is worth the cost, and if your client is pushing for a cheap provider, then hopefully this section will arm you with good counterarguments.

A Web host that charges \$10 per year would have to put hundreds of websites on each physical server in order to make any money. This would likely push the servers over capacity, and one website that gets heavy traffic could slow down the others or even take them offline. Google now uses website speed as a metric for its search rankings,³ so slow response times not only would annoy visitors but might compromise your ranking on Google.

For \$10 a year, you probably will not get any technical support; offering full support for that price would be difficult for any provider. If your website goes down or a problem is affecting your users, you need to know that you can contact someone quickly and that the problem will be a priority. Customer service costs money; so, a low price for hosting could well be at the expense of responsive support.

HOW TO CHOOSE A HOST

Your website will have certain requirements. For example, to run WordPress, you need the following:

- PHP version 5.2.4 or greater,
- MySQL version 5.0.15 or greater,
- Apache mod_rewrite module (for clean URIs, known as permalinks—optional but required for MultiSite set-ups).

³ Google Webmaster Central Blog, “Using Site Speed in Web Search Ranking,” smashed.by/googlespeed

If you are using a server-side language or database, check that your host supports it and supports the minimum version required by the software. As a rule of thumb, avoid hosts that run very old versions of server-side languages. You can see what the current version of PHP is by visiting *php.net* and checking under “Stable Releases.” At the time of writing, the latest release is 5.3.9. A host that keeps up to date would be running a version of 5.3, and certainly no version older than 5.2.4, as required for WordPress.



Figure 2.6. Check the latest stable version of PHP on *php.net*.

The host might maintain servers with older versions of PHP if it is supporting users with legacy applications that cannot easily be updated, but it should be able to put new accounts onto newer servers or to upgrade accounts when asked.

Databases

If you will need a database for your website (a WordPress blog, for example, stores its content in a MySQL database), then check that the host gives you at least one database. If you will be running a number of scripts, then installing each in its own database might be easier, so check how many databases are allowed. As with server-side scripting languages, check that the version of the database software offered by the host is compatible with the scripts you want to run.

Will you need email from the host?

If you would like to use the same provider for both email and hosting, check what it offers. Does it have Web mail if you need it? Can you use POP or IMAP email? Is the interface for setting up mailboxes easy to use? Is spam filtering available?

Will you need to point multiple domains at the website?

Some shared hosting companies allow only one domain to be pointed at a website. If you intend to point several domains at the same website, check that this is allowed.

MAKING YOUR DECISION

There are thousands of hosting companies to choose from, and the consequences of picking an unreliable company can be severe for a company whose website is a core part of its business. How should you go about choosing a reliable host?

With hosting, a good recommendation is a positive sign, especially if the person making the recommendation has used the host for several years. If you are working with a Web developer, they will probably be able to suggest reliable providers they have worked with—and probably suggest who to stay away from!

Researching hosting providers on Google is always worthwhile. If people are having trouble with one, then they are probably posting about it in forums or on blogs. If you notice a lot of bad experiences with a particular company, steer clear. Twitter is another good source of information, particularly because people tend to tweet about their problems in real time, even if they are not inclined to write a lengthy post about it in a forum or on a blog.

As mentioned, getting very cheap hosting is possible, which is fine if the website is personal or for a friend. But if the website is at all important to you or your client's business, then remember that you get what you pay for. If you want problems to be fixed quickly, then paying a bit extra is worth it.

Find out how the provider deals with support. Some hosts offer only email support, which might suffice, but your client might want a provider they can phone up when their email or website goes down. If you know someone who uses the provider you are considering, find out if they have ever had to contact support and what the experience was like.

As also mentioned, server-side languages need to be kept up to date to ensure security and compatibility, just like desktop computers. If a hosting provider is running a very old version of a language, it could indicate that its servers are not up to date and,

thus, are vulnerable. You might also run into problems trying to install an open-source script that relies on the current version of that programming language.

Many providers offer an uptime guarantee, usually as a percentage: a “100% uptime guarantee” means that they guarantee your website will be up 100% of the time. These guarantees tend not to mean a lot. The host will generally compensate you if your website goes down for longer than the time it guarantees, but the onus is on you to catch it and alert the host before it comes back up—not a lot of good if your website routinely disappears at 2:00 am! Also, uptime guarantees tend to come into effect only if the server itself is down, not if an external problem is affecting connectivity.

I don’t take much notice of uptime guarantees when choosing a host. I rely much more on reports from users themselves about how good a host’s uptime and support actually are.

Be aware that many hosting companies actually resell the services of larger vendors. Anyone can buy a server from a large hosting provider and start reselling the space on it, without much knowledge of hosting at all. If your website goes down, you may find that your host cannot do anything about it other than open a ticket with the larger company. You have ended up with a middleman between you and the people who can actually solve your problem.

MEMSET
Virtual server, Dedicated VPS, Virtual servers | Memset

Sales Hotline 0800 634 9270
Live Chat [Online](#) | [Add to Basket](#)

Home | Solutions | Support | About Us | Press | Contact | Client Login

Miniserver VM® Virtual Server / VPS Packages - Virtual Servers
Memset first launched their Miniserver VM® Virtual Servers (also called VPS or VDS) in 2002, and have remained leaders in the field since. Each virtual server comes with dedicated resources (RAM, CPU & Disk) and is operationally indistinguishable from a normal dedicated server. The host servers are powerful Dell PowerEdge R310 servers with a Quad Core Xeon processor and RAID1 disks. We typically put around 15 VMs on each server.

	VM1000	VM2000	VM4000	VM8000	VM16000	VM32000
CPU, Xeon cores min. equiv.	1x 0.35GHz Xeon	1x 0.70GHz Xeon	2x 0.70GHz Xeon	2x 1.4GHz Xeon	4 x 1.4GHz Xeon	4 x 2.8GHz Xeon
Dedicated RAM	512 Mbytes	1024 Mbytes	2048 Mbytes	4096 Mbytes	8192 Mbytes	16384 Mbytes
RAID, disk space	40 Gbytes	80 Gbytes	160 Gbytes	320 Gbytes	640 Gbytes	1280 Gbytes
Disk transaction/sec	10	20	40	80	160	320
Opt. 1 - Unmetered connection	5Mbps 10:1	5Mbps 5:1	10Mbps 7:1	10Mbps 4:1	20Mbps 4:1	40Mbps 4:1
Opt. 2 - Insulative transfer	500GB (20Mbps)	1000GB (20Mbps)	1500GB (20Mbps)	2500GB (20Mbps)	5000GB (20Mbps)	10000GB (20Mbps)
IP addresses included	Up to 2	Up to 2	Up to 2	Up to 2	Up to 2	Up to 2
cPanel/WHM with Fantastico	\$12.50/mo	\$12.50/mo	\$12.50/mo	\$12.50/mo	\$12.50/mo	\$12.50/mo
Windows 2008 Server R2	n/a	\$7.50/mo	\$12.50/mo	\$20.00/mo	\$36.00/mo	\$65.00/mo
Uptime guarantee	99.995%	99.995%	99.995%	99.995%	99.995%	99.995%
Basic Perl/Python monitoring	Included	Included	Included	Included	Included	Included
Approx. lead time from payment	<20 mins	<20 mins	<20 mins	<20 mins	<20 mins	<20 mins
Cancellation notice period	7 days	7 days	7 days	7 days	7 days	7 days
Setup charge	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Monthly Price	\$16.50	\$25.00*	\$41.00*	\$79.00*	\$159.00*	\$329.00

* special offer, limited time only!

Easy multiple-site Web hosting with cPanel/WHM
If you are looking for a comprehensive, easy-to-use way to host one or many Web

What is a Miniserver VM® Virtual Server?
Using Miniserver VM® virtualisation allows us to divide a single powerful server

Figure 2.7. Memset, a hosting company, offers a range of virtual private server packages.

TYPES OF HOSTING

For many years, *shared hosting* was the only cost-effective way to host all but the largest websites. A website on a shared host would share a server with up to hundreds of other websites. A certain amount of space would be allocated to you, and you would not be able to configure or install software on the server.

The alternative to shared hosting has generally been to have a *dedicated server*, where an entire physical computer runs your website. This is far more flexible, because you can configure the server and the software hosted on it. But it is expensive, and most websites do not need the space or resources available on a dedicated server.

In the last few years, a new type of hosting has emerged. *The virtual private server* offers what appears to be a dedicated server, but in fact you are sharing one physical server with a number of other virtual servers. The advantage of this to shared hosting is that you can run whichever versions of the operating system and software you like; thus, you can configure the server and host multiple websites on it.

Most companies that offer this type of hosting provide a control panel by default, to make managing the server easy even for people who have never administered such a system before. The two most common control panels are Plesk and cPanel (which includes Web Host Manager). These tools help you set up new websites on the server, and they usually enable you to schedule updates to software on the server and to configure what services are available to individual websites. In effect, with a virtual private server, you become a Web host yourself.

Virtual private servers are also handy for setting up website demos for clients, perhaps on a subdomain. For a designer whose policy is not to hand over files until the client has paid, this set-up allows them to show the entire website to the client for approval before moving it onto the production server.

A virtual private server gives businesses the ultimate in flexibility. But if your needs are simple, then they might be met by a shared hosting package that has adequate specifications and reliable support and is kept up to date.

Cloud hosting is different from shared hosting or a virtual private server. With it, the resources required by your website might be spread across a number of servers, and you have the freedom to add resources or to scale back as required. This type of hosting is useful for applications that require heavy resources at certain times and light resources at others.

A ticket sales website is a good example; people will flood the website when tickets for a popular event go on sale, but traffic will be low the rest of the time. With cloud

hosting, you can access extra capacity when you need it (paying for it, of course), saving on a lot of redundant capacity all year round.

Development Environments

This section is less for the developer working on a large team and more for the designer or developer working on a redesign for the first time. We'll share good practices for redesigning and then going live with the redesign.

Whether you are replacing a live website or refactoring it with new elements or a new UI, you need to do it without affecting users. Even if the changes are minor, never update a live website without testing the changes. Something will always go wrong!

One method many people follow is to develop in a subfolder of the live website. This is a bad idea, too, because the paths from the root folder will be incorrect; and when you go live and move the website up a level to the root, your links and images could break, because many server-side scripts locate them relative to the root. You would then need to do a repair job—not the best start to a launch.

Instead, set up a local environment to develop in, including server-side scripts and correct paths from the root. Then upload to a staging server, where you would test the site on the Web, show it to the client, populate the content and get approval to go live.

If you are refactoring a live website or redesigning the UI but leaving the back end more or less intact, then I suggest exporting the database and downloading all files (as described in the section “Moving Your Website” later), so that your development and staging versions of the website are set up exactly the same as the live website. If the live database contains any confidential information about users, you will want to delete it or use placeholders, not only to protect your users' data but to avoid accidentally emailing 10,000 users from your local system as you're testing something!

THE STAGING SERVER

A local server is a set-up on your own machine or another computer in your office that you use to develop the website. Once you are ready to show your work to the client, you might need to move it somewhere else to make it accessible from outside your network; that “somewhere” would be a *staging server*.

It could be as simple as a subdomain that points to a folder on your current host's server; this enables you to test the website in an environment that approximates the live website, with all of the correct paths. If you build a lot of websites for clients and tend

to deploy to the same type of environment, you could take out a cheap virtual private server and point a subdomain to it for each client’s website, enabling you to test easily.

If you do either of these things, you should secure the entire subdomain using directory security, which forces visitors to enter a password in order to see the website. This will prevent anyone from seeing your half-finished work and will prevent Google from indexing it.

When deploying large content-managed websites for clients, we give them access to a staging version of their website, where they can add all of their content and test the website thoroughly. We then move the website, database and assets over to the live server, a very quick process at that point, where all of their content goes live.

VERSION CONTROL

We couldn’t have a section on development environments without mentioning version control. Even if you are working on your own, you should be using version control. It really becomes useful, though, when you are working in a team. Team members can modify files without worrying about overwriting each other’s work, and you can always roll back to a previous version if something goes wrong.

If you have decided to refactor rather than redo a website, your first move should be to import all of the files into some form of version control system. This way, you can always roll back if one of your changes has an unintended effect on part of the system.

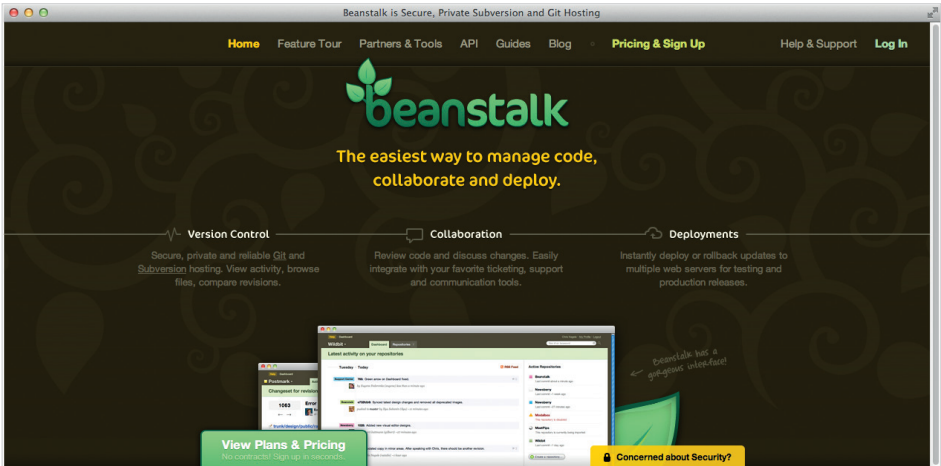


Figure 2.8. Beanstalk is a hosted service for Subversion and Git.

Even when you're working alone, version control can save you from yourself—for example, by keeping you from accidentally deleting a file or rashly modifying an application. I use version control to port work between the multiple computers and locations that I work from. At the end of every day, I commit my work. Should my daughter need to stay home from school the next morning and I find myself working from home, I can retrieve my files and pick up from the exact point where I left off the night before at the office.

There are a number of version control systems; you have probably heard of Subversion and Git. The choice comes down mostly to personal preference. While you can configure a server to do the job, you can get started quickly by using a hosted service, such as GitHub or Beanstalk. Beanstalk happens to have an excellent beginners' guide to version control on its website.⁴

Moving Your Website

Having to move a website from one host to another worries a lot of people, particularly if the website is popular. But if you follow the steps below, you will find that doing it with no visible downtime to visitors is possible. The process is the same for moving a website from the staging server to a live server.

The instructions below are for deploying a website to a regular shared hosting environment, with SFTP access. If your environment is more complex than that, then you probably already have a developer to help you work through it.

SET UP THE NEW HOSTING ENVIRONMENT

First, you need to set up the new hosting environment. Check that it has all of the capabilities you require, including support for the server-side language and database of your choice.

TRANSFER THE FILES TO THE NEW HOST

Use SFTP to upload the website's files to the new host. The new host might provide a temporary domain for you to test the website. If not, I usually point a subdomain of mine to the server temporarily while I check that everything is working properly.

⁴ Beanstalk, "An Introduction to Version Control," smashed.by/version-control

Avoid previewing websites that have a tilde (~) or your user name in the domain—otherwise, paths from the root will be incorrect.

TRANSFER THE DATABASE TO THE NEW HOST

Unless you are working on Microsoft-based applications, you will most likely be using MySQL. MySQL is the database used for most open-source applications and is a common choice among developers who work with PHP, Python and Ruby on Rails.

The first thing to know when working with a database such as MySQL is that there is no database file to upload or download. The database is held within the MySQL server; to get at the data, you need to connect to the server. Many hosted servers come installed with PHPMyAdmin, a Web-based application that enables you to manage the database in a browser. You can also download and install PHPMyAdmin yourself, which makes for a handy way to move a MySQL database between the local, staging and live servers.

SET UP THE MAILBOXES

If your email will also be hosted on the new domain, then set up the mail accounts so that they are ready once the domain transfers over.

CHANGE NAMESERVERS, OR REPOINT THE DOMAIN A RECORD

If you are changing the DNS of the domain to your new host, you should now be able to change the nameservers. If your DNS is hosted elsewhere and you just want to change where the domain points to, then you would just repoint the A record. It can take some time for the domain to resolve to the new host everywhere on the Internet, so continue to check mail through your old host for a day or so to make sure you don't lose any. If the old host allows users to check mail in a Web interface without going to the domain itself, you can do it that way, until mail stops coming through there.

MOVING A LIVE DATABASE-DRIVEN WEBSITE

The problem with moving a live database-driven website is the time it takes for the DNS to propagate across the Internet and for the caching servers to update. At some point, some visitors will be directed to the old host and some to the new. If your database is meant only for internal content management, and visitors do not add to it, then you simply need to let your content creators know not to change any content until they

are sure they are accessing the website through the new host. In this situation—and after testing the website with a copy of the database—I would move over a new copy of the database just before switching nameservers, to make sure I’ve captured the most up-to-date content. If visitors are able to add data to the database (for example, one that captures orders for an online store), then you will need to avoid losing any data in the transfer. The safest way to do this is to take the website offline and put up a holding page while you perform the transfer.

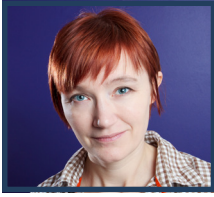
If you cannot do this, there are a couple of workarounds. First, do the transfer as described above, and then once you are sure the DNS has fully switched, compare the two databases, and sync any records from the old to the new. If the website gets fairly low traffic and you expect just a couple of orders or comments during that time, this should work fine. Alternatively, you could move the database first. As long as the new hosting space can be set up to allow an external website to connect to the database, you can move the database and then connect both the old and new hosted websites to the database in the new environment. If you are in this situation, speak to your hosting provider (and your developer if you have one) to work out the best method.

Where to Go From Here?

We’ve just skimmed the surface of the technical considerations to make when redesigning a website or embarking on a new stage of development. Not everything in this chapter will have been of use to you, but by thinking through the technical decisions you need to make, even if you are not the one who will be implementing them, you avoid rushing down a wrong and expensive path!

To echo Paul Boag’s words from the previous chapter, do your homework before diving into any solution. There are many ways to achieve whatever it is you want to do, and the CMS or framework that someone is trying to sell you might not be best for your project. By always returning to your research, business requirements, available resources and constraints, you should be able to make wise decisions. A solution that has been well thought out should enable the website to grow and develop.

If you have to completely rebuild a website, treat this as the last time you will be able to do this, and pick a solution that will take the website and the business forward.



About the Author

Rachel Andrew is a Web developer with skills in both server-side languages and front-end development. She has written numerous books, including *The CSS Anthology: 101 Essential Tips, Tricks & Hacks*, the fourth edition of which is to be published in March 2012. Rachel is the founder and managing director of edgeofmyseat.com, and she co-created Perch, the really little content management system.

Rachel has a personal website at rachelandrew.co.uk, where she writes about many things, including the Web and running a business, and can be found on Twitter @rachelandrew.



About the Reviewer

Harley (1983) is a Canadian citizen, born in Montreal, where he pursued his degree in economy. He earned an MBA in law from the University of Ottawa, home to the city where he lives nowadays. He describes the city as having a beautiful small-town feel with big-town fun, and very cold. Apart from the Internet, he loves skiing, boxing, amateur sushi rolling, DJ'ing and the color red.

Harley started his first company at the age of 17 and has been building stuff ever since. Today, he is the Chief Platform Officer at Shopify, the Web's leading e-commerce platform. Throughout his career, Harley has learned to go after things he is passionate about and practice them everyday. His personal advice to the readers is #Hustle&Hack.



About the Reviewer

Ryan Carson is an American living in England and a father. Having graduated from Colorado State University with a degree in computer science, he moved to the UK in 2000. He loves Web tech, coffee and movies. (Is it wrong to have seen *The Matrix* seven times in the cinema?) He's passionate about connecting and encouraging people, which is why he's passionate about running events for the Web community. He has successfully built and sold two businesses and is now working on his third, Treehouse.



Jumping Into HTML5

Written by Ben Schwarz

Reviewed by Russ Weakley

IF YOU ASKED ME TO EXPLAIN HTML5 TO YOU, I would probably start by explaining that your role as a Web developer has changed from the days of old. I would expect you to be an expert in HTML (the markup language), CSS (and all of its permutations across browsers), JavaScript (and the subtle differences between its APIs in browsers). And then I would roll on to design theory, animation, 3-D, server-side technologies and sound engineering.

After a moment of silence, you would probably wonder why so many technologies are brought under the umbrella of HTML and perhaps wonder why you decided to build for the Web in the first place.

HTML5 (as a specification) is broken into many pieces, covering distinct areas of specialization, so try not to fret. Getting a solid, basic understanding of HTML, CSS and JavaScript will enable you to continue on your own and develop specialized skills that others do not possess. In essence, you will become invaluable to your team or company by focusing on “non-core” technology.

Best practices have not been established for many of these fancy new features, so if you want to learn something cool (and maybe become famous in the process), then it is time to download a beta browser and start experimenting.

Most browser vendors release beta versions of their browsers to allow developers to experiment with cutting-edge features. The “big five” all have betas that you can download:

- Google Chrome has three not-ready-for-prime-time versions: “Beta” (for developers), “Dev channel” (for developers who want to use features that have been released within a week) and “Canary” (a nightly release, totally untested). You can get all of them from smashed.by/chromedev.
- Apple’s Safari browser has one version: WebKit (webkit.org).
- Opera has a “Next” version: smashed.by/operadev.
- Firefox has a nightly edition (smashed.by/ffndev) and a pre-beta build called Aurora (smashed.by/ffadev).
- Last but certainly not least, Microsoft releases new builds of IE manually (i.e. not nightly): smashed.by/iedev.

Browser support for new features is rolled out in a modular fashion. And with browser vendors (notably, Google and Mozilla) now releasing on a six- to eight-week cycle, version numbers are clearly less important than they used to be. One could liken this to the way developers make changes to websites. A website has a version, but it is unimportant to the end user. So, as a Web developer, concern yourself with which features to use to best tell your story and to translate your designs into living, breathing products.

As Web technologies evolve, we have to be constantly aware of the past. Thankfully, this goal is shared by both of HTML's standards bodies, so you will be relieved to know that HTML5 will not alienate your user base or make your job harder. Which-ever DOCTYPE you use, the user's browser will render the website as best it can. If you use a new HTML5 feature with an old DOCTYPE, it will still render correctly.

In this chapter, we will not talk about WebGL, audio and video, device APIs, Web sockets or SVG. I will leave them for you to discover because each warrants its own chapter. Instead, I will give you a tour of the ground floor. We will cover everything that is important to get right before moving on to advanced topics.

Where We've Come From, Where We're Going

HTML5 is a lot of things. And since the last major "version" of HTML, we have come a long way. The Web Hypertext Application Technology Working Group (WHATWG) refers to it as "HTML: The Living Standard" (it drops the 5). That is, HTML is defined as version-less technology. And as mentioned, browser vendors cherry-pick which features to implement, which is why browsers vary in their support.

WHATWG, W3C AND "THE COMPANIES"

You have probably heard of the World Wide Web Consortium (W3C). More recently (perhaps as recently as the last few paragraphs), you have seen reference to WHATWG. WHATWG was formed by representatives of Apple, Mozilla and Opera, who were concerned about the lack of development of HTML by the W3C and thus decided to form their own group.

Much of the work of WHATWG is shared by the W3C, and the license for the specification states that, "You are granted a license to use, reproduce and create derivative works of this document."

The W3C indeed shares the work. It does not create standards, but rather makes recommendations. And while the W3C is funded by all of the big computing and browser companies, it is dedicated to open standards that do not put any one company at an advantage.

So, as a Web developer, you can be sure that all new developments in HTML (particularly with regard to Web applications) are developed with considerable financial backing from browser implementors (Webkit, Gecko and Opera) and are approved by the W3C over time.

This odd relationship has led to a situation in which technology that comes with licensing fees or that is strictly proprietary is not looked upon favorably by many. The browser race is as competitive as when it began.

KNOWING WHAT FEATURES TO USE

A modern Web developer has to understand the audience they are serving, be able to choose the right technology for the job, and know what the impact will be if a feature is not supported by the browsers used by their audience.

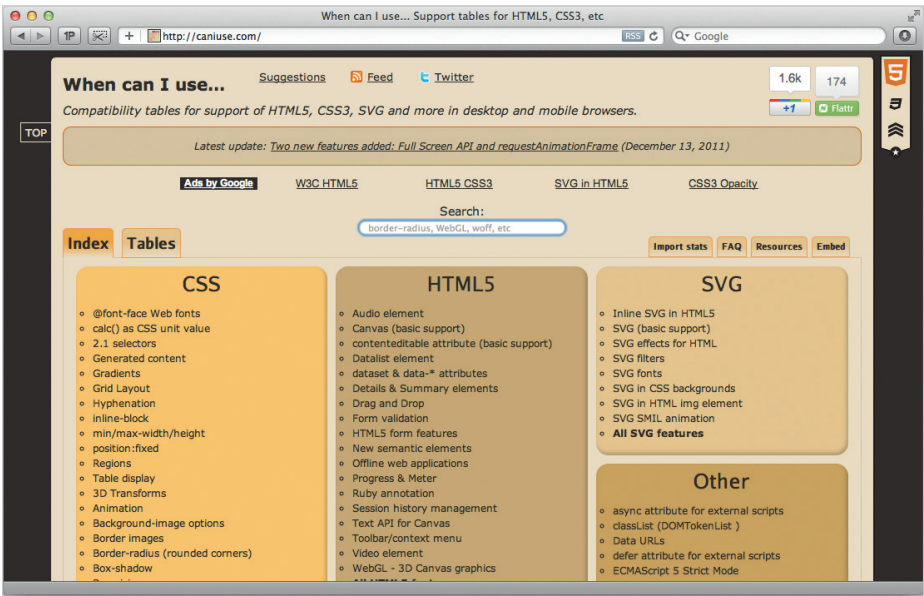


Figure 3.1. Caniuse (smashed.by/ciu) illustrates when you can, or should use a HTML5 feature.

Only wizards magically know whether a given feature is widely supported. If you're not one of them, you can be thankful for *When Can I Use*.¹ The site lists what features are supported in current versions of all the major desktop and mobile browsers and what features will be in future versions. It is searchable, and it even hooks up to Google Analytics to show you what browsers your audience is using. Now, let's dive in and look at HTML, from the ground floor.

THE DOCTYPE

Think back as far as you can remember. Did you ever remember the full DOCTYPE for HTML 4.01 (or for XHTML, for that matter)? I didn't think so. Let me show you the HTML5 DOCTYPE:

```
<!doctype html>
```

That's it! It can be in uppercase or lowercase letters, and it is all you need to put the browser in standards-compliance mode. It makes you wonder why we had to copy and paste the top of our HTML documents all the time.

Of course, we have been littering our HTML with a bunch of other important tags for years now. Let's look at what else has been simplified.

META CHARACTER SET

```
<meta http-equiv="Content-Type" content="text/html;  
charset=utf-8">
```

Argh, what a mess! This meta tag is rather important and should be added before the title tag to ensure that the browser sets the character encoding correctly. Thankfully, it has been simplified to something memorable:

```
<meta charset="utf-8">
```

Some XML parsers have trouble with tags that are not self-closing, which is why some Web developers choose to use self-closing tags (i.e. XHTML style). It is entirely up to you, but we suggest leaving the tags open.

¹ smashed.by/wcai

STYLE SHEET LINK AND SCRIPT TAGS

The type attribute can be omitted from both the `<link rel="stylesheet" href="layout.css">` and script tags.

In the old days, the type attribute could be used in the script tag if you wanted to use VBScript instead of JavaScript, but these days it is not at all required.

Being able to omit these details that made our documents longer and harder to write feels great. But we have just scratched the surface. Let's add a little something to the script tag.

ASYNCHRONOUS SCRIPT DOWNLOADS

First, a word on how the browser downloads files. After the browser downloads and parses HTML, it collects a list of assets (i.e. images, CSS, JavaScripts, etc.) and prioritizes them for downloading in order of appearance.

In the past, we connected to the Internet through dial-up, which did not handle multiple concurrent connections very well. Now, because bandwidths vary drastically (especially with mobile devices in the picture), browsers are limited to downloading only a few assets at a time per top-level domain.

This is why some developers use content delivery networks or assign assets to a subdomain (such as `assets.example.com`); using different top-level domains gives the developer more download "slots" for scripts, style sheets, images and iframes. Be aware that this comes with a performance hit!

When browsers download JavaScript, they do so one script at a time, allowing the browser to parse and pre-run magical optimizations. Now, rather than leaving the whole experience to chance, we are able to use script loaders (such as LABjs, Yepnope, RequireJS and many others) to load multiple scripts at once, set up dependencies and determine whether a particular script file is required at all.

Improving the performance of pages where possible makes sense. Amazon claims that an increase of 100 milliseconds in page-loading speed leads to a decrease in sales of 1%.² With that in mind, let's look at my favorite of script loaders, Yepnope.³

Yepnope can be used to conditionally load scripts based on tests. Simply put, you can request a JavaScript only if the browser needs it.

² smashed.by/amzspeed

³ smashed.by/yepnope

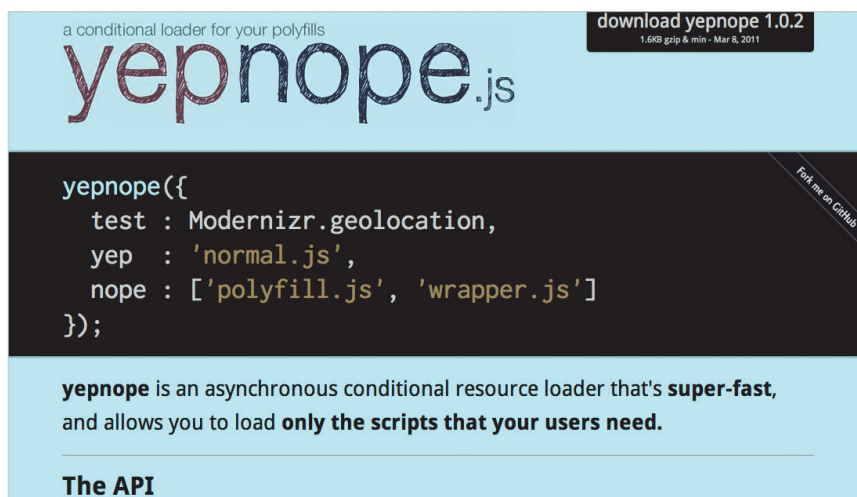


Figure 3.2. Yepnope: a conditional script loader.

For example:

```
yepnope([
  {
    test: window.JSON,
    nope: '/javascripts/json2.js'
  }
])
```

CODE

This clever bit of JavaScript checks whether the browser has a native JSON parser, and for those that do not (which is IE 6 and 7), it loads `/javascripts/json2.js`, which is a JSON polyfill.

Now that we have (briefly) covered the basics of script loaders and talked about loading scripts in parallel, it is time to look at two new attributes in the script tag. First up, `async`:

```
<script src="/javascripts/application.js" async></script>
```

The `async` tag is a boolean attribute, which means that its mere presence in the browser indicates true, or “Yes, please use this feature.” It tells the browser to execute `application.js` as soon as it is available. Scripts that are loaded using `async` are executed as soon as they are downloaded—that is, not in the order of their appearance in the HTML.

GETTING THE FILES TO THE CLIENT FASTER

It is worth mentioning that the biggest performance gain we get is in reducing the size of the scripts as a whole. The first way to reduce the size of scripts (as well as of style sheets and HTML files) is to serve them using gzip. To add gzip support to your website, check out the HTML5 Boilerplate Webserver configuration repository on GitHub.⁴

If you are not sure how your website is being served, it is time to familiarize yourself with the Web developer toolbar. In Webkit-based browsers (i.e. Safari and Chrome), you can open the Web developer toolbar by pressing Command + Option + I on a Mac and Control + Shift + I in Windows.

Under the “Network” tab, you will see a list of files that were loaded for the current page. You can inspect the request and the response headers for each file listed.

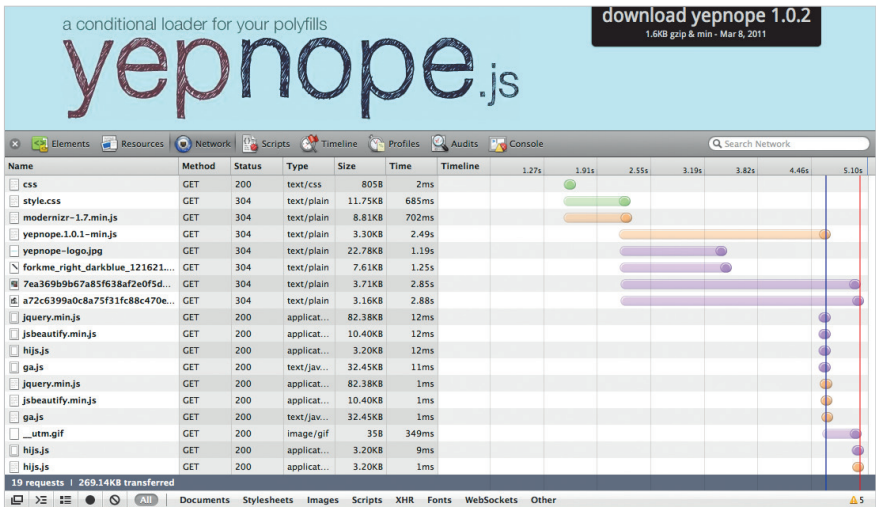


Figure 3.3. Safari’s developer toolbar that shows the network activity on Yepnope.

The second tip for improving your website’s performance (actually, the best way to improve performance when it comes to scripts) is to concatenate and compress your files. I recommend the UglifyJS compiler.⁵

Clever people such as Steve Souders have devoted themselves to understanding how all browsers download, parse and display websites. If you are interested in producing better-performing websites and applications, follow Steve’s work.

⁴ smashed.by/configs

⁵ smashed.by/uglify

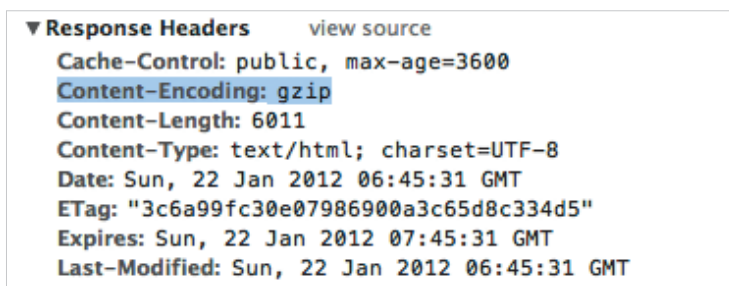


Figure 3.4. This file was returned to the browser using gzip encoding.

NEW SEMANTIC TAGS AND WHEN TO USE THEM

Because we are talking about HTML5, all of this talk of performance and scripting probably feels a bit foreign. Let's look at the new semantic tags to work out when to use them.

Before using any new tags, be sure to use what is known as the HTML5 Shiv. The script is essential because without it, Internet Explorer 6, 7 and 8 will ignore any unknown styles (i.e. the new HTML5 tags, which were unknown when those versions were built). You can get a copy of HTML5 Shiv from Google Code, where the project is hosted.⁶

You can also get HTML5 Shiv by using Modernizr.⁷ We won't cover Modernizr in this chapter, but do check it out. I have been using it on every website that I have built in the least two years.

RESETTING DEFAULT HTML STYLES

Browsers render unstyled elements slightly differently; so, to normalize the code base for a better cross-browser development and maintenance, the second thing you will want to do is use a CSS reset.

Use one of the newer CSS resets, because the older ones do not style HTML5 elements. I strongly recommend Normalize.css⁸ by Nicolas Gallagher⁹ and Jonathan Neal.¹⁰ Many of the older reset scripts (Eric Meyer's classic Reset CSS, for instance) are somewhat heavy-handed: they reset every element, and some of their changes are debatable,

⁶ smashed.by/steve

⁷ smashed.by/modernizr

⁸ smashed.by/normalize

⁹ smashed.by/nicolas

¹⁰ smashed.by/jon

such as setting strong to be unbolded by default. Normalize.css resets elements more gracefully, and it handles a few browser quirks. It gives you as close to an even playing field as you have ever had in a browser. Jon Neal and Nicolas Gallagher have carefully explained everything the script does. Read the heavily commented code—it’s fantastic!

Rebuilding Your Website

Having reached this point, you’re probably thinking, “OK, it’s time to start writing the website using the latest and greatest tags.” When the new semantic tags were established, developers had to do some work in analyzing what classes and IDs they were applying to their websites. What they discovered really wasn’t astounding at all: they were all using the same naming conventions (or slight permutations thereof). So, the names for the new tags will probably fit what we are already doing.

SECTION

A section tag could be used to break up distinct parts of the home page. Perhaps your blog consists of personal information about you, presentations that you’ve given and regular posts. You could probably break these up into section elements:

CODE

```
<section class="articles"></section>
<section class="about-me"></section>
<section class="presentations"></section>
```

Each of these parts could probably take its own header, so we could regard them all as reasonably important, thus justifying our decision to make them sections. If you were writing a book like the one you’re holding in your hands right now, you might make each chapter its own section, and then make each section within a chapter a nested section. As with everything related to semantics on the Web, don’t get hung up on the details. Choose an element based on the best information you have at the time and move on. Semantics are subjective—don’t sweat the small stuff!

ARTICLE

When diving into HTML5, you might have wondered about the difference between section and article. Perhaps you guessed that an article tag is used primarily for blogs and online news. You would not have been far from the truth.

If you remember one thing about the `article` tag, remember this rule of thumb: if the piece of content would still make sense outside of its current context (that is, if the user could not see any of the page surrounding that piece of content), then it is probably an article. Hence, blogs and online news.

I use `article` for collections of content—say, a list of presentations that I have given in the past, perhaps with synopses:

```
<section class="presentations">
  <header>
    <h1>My Presentations</h1>
  </header>
  <article>
    <h2>An Introduction to HTML5</h2>
    <p>A four-hour workshop that I ran around Australia.</p>
  </article>
  <article>
    <h2>Compass and SASS</h2>
    <p>Use a well-written library of CSS so that you can focus
on the important things.</p>
  </article>
</section>
```

CODE

Eagle-eyed readers might ask, “That looks like a list! Why not use `ul`?” You would not be wrong; it absolutely could be a list element. But `article` indicates that, while these elements are similar, they are not related to one another. We could argue about this for hours; in the end, you will have to make up your own mind.

HEADER

Have you ever used a class or ID like `masthead`, `banner` or even `top` for the header section of a website? The `header` tag can be used for much more than just the head of the website. It can be used within an article or section, but it is entirely optional. Just use it when you need a block-level element to mark off space on the page for clarity. For example, I often keep titles and meta information in the head of a blog post.

FOOTER

The footer tag is just like header. You can use it within article or section or globally within body.

ASIDE

The aside tag can be used at the top level or within article. Its contents can be regarded as useful but not essential information.

For the mobile version of your website, for example, you could choose to hide aside elements. However you treat it, the tag forces you to make some decisions about your content. A blog post could be set up as follows:

CODE

```
<article>
  <header>
    <h1>All About Tractors</h1>
    <time datetime="2012-01-01">1 January 2012</time>
  </header>
  <aside>
    <p>Written entirely by Bruce Lawson</p>
  </aside>
  <!-- The body of the post goes here. -->
</article>
```

TIME

Did you spot the new tag in the last code snippet? The time tag is simple: use it to display the time. You can provide a machine-readable version, too.

CODE

```
<article>
  <p>Published on <time datetime="1984-04-03"
    pubdate>3 April 1984</time></p>
</article>
```

The pubdate attribute can be used to indicate the initial publication date of an article. The specification states that the pubdate attribute should be used only once for an article.

NAVIGATION

The nav element is, obviously, for the navigation of a website. You can nest nav tags to create a drop-down menu. The tag would not be suitable for that list of presentations I referred to earlier when talking about the article tag. Reserve nav for the structural navigation of the website itself. For example:

```
<nav>
  <ul role="navigation">
    <a href="/products">Products</a>
    <a href="/contact">Contact and Locations</a>
    <a href="/about">About Our Company</a>
  </ul>
</nav>
```

CODE

(Wondering what the role attribute is for? You will have to keep reading!)

FIGURE AND FIGURE CAPTION

You probably add a lot images to your pages. Have you ever considered the best way to apply captions to those images? Wouldn't it be nice to be able to wrap a caption tidily with its image? Well, that is what the figure tag is for.

```
<figure>
  
  <figcaption>A glass of whisky, with a side of
    water in a small jug.</figcaption>
</figure>
```

CODE

It doesn't end there. You can also use the figure tag for video, svg and pretty much anything that is visual and could take a caption.

DIV

With all of these new tags, you would think that div was a thing of the past. It is not. Developers have been using the humble div tag for everything under the sun for years now, to the point of some contracting the debilitating disease of "divitus."

A `div` is a “division” and sometimes there is no better tag with which to describe a piece of content. Perhaps all you need is a box in which to add some styles. It happens. I don’t blame you. Semantics are tricky. If you really cannot describe a piece of content using any of the HTML tags mentioned above, then use a `div` and don’t feel guilty about it.

A FEW WORDS ON SEMANTIC OUTLINING

Now that we have some new sectioning elements (i.e. `section` and `article`), the plain old document outline that we have been used to has changed a bit. Sectioning elements can be thought of almost as documents of their own. In other words, the heading levels `h1` through `h6` can be used within them.

But hold on! This means you could encounter something like the following:

CODE

```
<body>
  <header role="banner">
    <h1>Full Frame: A Blog About Cycling</h1>
  </header>
  <article>
    <h1>Early Morning Over Black Spur</h1>
    ...
  </article>
</body>
```

Multiple `h1`s in the same document? That’s crazy! Instead, I use headings to show the structure within a given section:

CODE

```
<body>
  <header role="banner">
    <h1>Full Frame: A Blog About Photography</h1>
  </header>
  <article>
    <h2>Early Morning Over Black Spur</h2>
    ...
  </article>
  <section>
    <h2>Buy Our Book!</h2>
```

```
<section>
  <h3>Print</h3>
  ...
  <button>Purchase: $90</button>
</section>
<section>
  <h3>Electronic: PDF or eBook</h3>
  ...
  <button>Purchase: $15</button>
</section>
</section>
</body>
```

Not only does this make styling the headings easier, but it just feels much better: less confusing, and no swimming against the current.

Before having tags such as `section` and `article` at our disposal, we really only had `h1` to `h6` to describe the depth or hierarchy of a website. Now, we can describe infinite levels of depth and can represent each level of content accurately.

If after all this, you are still not sure which element to use, check out the fabulous flowchart on HTML5 sectioning elements¹¹ that was developed by Oli Studholme and Piotr Petrus. Print it out, stick it on your wall, and you will always know which elements to use. Maybe—no promises. As always, you will probably want to validate your HTML to keep it in check. I prefer Validator.nu.¹²

WORKING WITH WAI-ARIA ROLES FROM THE GROUND FLOOR

Roles for WAI-ARIA (short for Web Accessibility Initiative: Accessible Rich Internet Applications) have always been a part of modern HTML technology—so much that perhaps most developers glaze over as soon as they’re mentioned.

The roles are designed to make websites and applications more accessible to users with screen readers. Any professional accessibility expert (there really are not enough of them!) would attest to the importance of WAI-ARIA; and for the most part, they are largely ignored, too.

¹¹ smashed.by/h5doc

¹² smashed.by/vldnu

Companies may talk a lot about how important accessibility is; whether they are conducting studies on it or designing for it is a different story. But your job as a website builder is to enable everyone to consume your content.

For blind and low-vision users, WAI-ARIA roles describe the context and purpose of the information laid out for them. A section of a page is not just visually different—it is contextually different, and screen-reading software can explain that difference to the user, enabling the user to interact with that section without missing a beat.

I am by no means an accessibility expert, but I will try to give you the best no-nonsense rationale for why the importance of WAI-ARIA goes far beyond accessibility. If you have ever used any of the sectioning elements described in this chapter, then you have probably encountered something like the following:

CODE

```
<body>
  <header>
    <h1>Tractors: An Interactive Guide</h1>
  </header>
  <article>
    <header>
      <h2>Tractor Maintenance</h2>
    </header>
  </article>
</body>
```

Did you spot it? The document has two header tags, both legitimately used. The problem lies in the CSS:

```
header {
  margin: 0 2em;
}
```

This element selector targets both headers. We could use a descendent selector (i.e. `body > header`), but that feels a little heavy-handed, not to mention that the top-level header might be the masthead for the entire website. We can use a WAI-ARIA role to our advantage here simply by adding `role="banner"` to the HTML:

```
<body>
  <header role="banner">
    <h1>Tractors: An Interactive Guide</h1>
  </header>
  <article>
    <header>
      <h2>Tractor Maintenance</h2>
    </header>
  </article>
</body>
```

This role attribute states that `<header role="banner">` is a “global” element that contains content that applies to the entire website, rather than just the current page. This is a good fit and, thanks to a simple attribute selector, not too difficult to style:

```
header[role="banner"] {
  margin: 0 2em;
}
```

Because the header and footer tags can be used in multiple places, we seem to be left without a tag especially for the main content. Thankfully again, an ARIA role is ripe for the picking.

By adding `role="main"` to article (thus, `<article role="main">`), we can easily specify that the main content for the current document is contained within article. (You might have noticed in the snippets above that `h1` is used in the top (root-level) header, and `h2` is used in the nested header. Combined, this best describes the hierarchy of the document.)

You are probably starting to appreciate the gracefulness of this approach. The descriptions of content are becoming more detailed, and we are able to apply styles to our new tags with minimal effort. A third ARIA role is `contentinfo`, which is often used for copyright notices, privacy statements and general information about the current page or website. (Some people would call this “meta information.”)

Finally, a fourth highly useful ARIA role to be aware of is `navigation`, which easily distinguishes a navigational section from a regular old list of links. Adding ARIA roles is a good way to make the content and context of your existing website more descriptive. Then, when you decide to overhaul the website, you can use the newer tags.

Hopefully, this short introduction has helped you see the benefits that semantic content brings to everyone. ARIA roles are an excellent example of this.

CLIENT-SIDE STORAGE

Now for an entirely new subject: HTML5 client-side storage. To date, we have had few options for storing data on the client's side. The most common has been the humble cookie-based session; but cookies are beset by a host of small problems, the more bothersome of which are these:

- The data you store in the session is transported back and forth between client and server with every request,
- The data you store has a limit of 4 KB;
- All cookies are timed to expire.

Cookies are not all gloom and doom, though. A cookie is what stores a user's data for logging into a website, and it helps the server to identify who the user is. It is clear, then, that we need some other options just for storing data. Thankfully, we have a fantastic solution in local and session storage. What are they? I'm glad you asked.

With `localStorage` and `sessionStorage`, we have two JavaScript APIs for storing strings of text to the browser. The `sessionStorage` API is purged when the user's session has ended (i.e. the tab or browser is closed), while `localStorage` sticks around until the developer (through JavaScript) or the user (through their browser settings) decides to remove it. The APIs are virtually identical—the only difference being the length of time of the storage.

Open your developer toolbar in a modern browser (i.e. one released within the last three years). Type in `localStorage.setItem("name", "Ben")`. In Webkit-based browsers, you will see my name stored under the “Resources” tab (you will have to expand “Local Storage” to see it). You have just stored your first item in `localStorage`.

Now, let's retrieve what we've stored by using `localStorage.getItem("name")`. You should see “Ben” printed neatly in the console. Finally, to clean up after yourself, use either `localStorage.removeItem("name")` to delete my name or `localStorage.clear()` to remove everything in `localStorage`. When the user calls `localStorage.clear()`, they are only clearing it for the current domain. So, if the user stores some data on the website hosted at `example.com` and then switches tabs to `google.com`, they would see that they cannot access the data that they stored in the `example.com` tab.

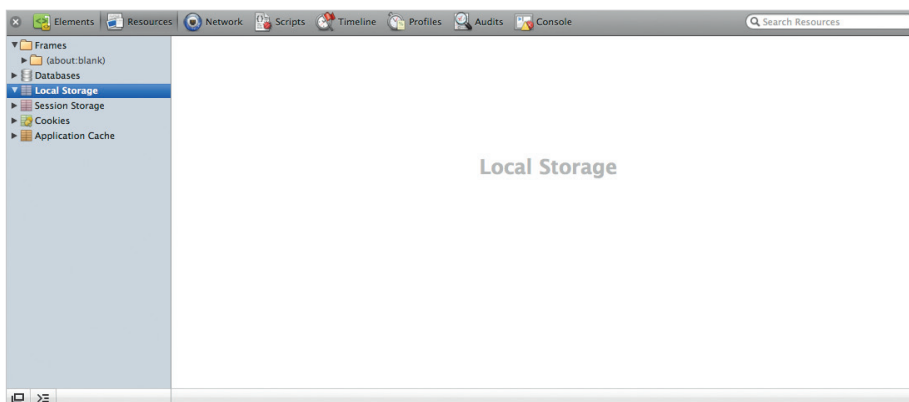


Figure 3.6. The local storage object explorer in Safari.

The `localStorage` API is highly useful. Say you are building a Twitter client which you want to be able to do the following things:

- Use in your desktop browser and on your mobile phone;
- When online, view tweets from your last online session;
- When offline, queue tweets to be posted later.

With `localStorage`, this is all possible. The snippet below illustrates this. (It is purely hypothetical, so don't sweat the small stuff.)

```
postTweet = function(tweetText) {  
  // Check if we're online  
  if(navigator.onLine) {  
    // Hey, we're online! Send that tweet, baby!  
  } else {  
    // Hm, we aren't online right now. Better store it for an-  
    other day.  
    localStorage.setItem("queue-" + +new Date(), tweetText)  
  }  
}
```

CODE

That wasn't hard, was it? To see all of the items in `localStorage`, we can iterate over them, just like an array:

```
for (item in localStorage) { console.debug(item) }
```

This will print out a list of all of the keys of items you have stored. Say you want to display your queued tweet? Here is how you would do that:

```
for (item in localStorage) { console.debug(localStorage[item]) }
```

The `localStorage` and `sessionStorage` API can be found in all modern browsers (including IE 8+), so there is no reason why you couldn't start building your own applications or just start experimenting with it in client-side applications.

In Summary

Before you replace all of the markup in your current website, take some time to study ARIA roles and browser performance and to generally learn how to structure code. Using a new tag will make only you feel good, whereas using ARIA roles will make a lot of people feel good. It sounds almost poetic, but it really is far simpler than that: it's your job.

To start using the new technology, don't feel like you have to use the new minimal DOCTYPE. Browsers will use whatever features they can when displaying your website. There is no "HTML5 mode," so you might as well dive in!

This is just a taste of the platform we call HTML5. We could go on for days about it, but instead, we will leave you with a few references to bookmark:

- HTML5 Please, html5please.us
Want to know when you need to patch older browsers? Or when a super-new tag isn't quite ready for prime time? This website will give you the grounding to really turn it up a notch.
- HTML5: A Technical Specification for Web Developers, smashed.by/whatwg
This guide is an abbreviated version of the full HTML5 specification. It removes all of those obtuse details that browser vendors need to build browsers. It is searchable, works on mobile devices (even offline), and was created by yours truly.
- HTML5 Rocks, html5rocks.com
This website is maintained by staff at Google, and nearly every article posted is not only enlightening, but mind-blowing.

- HTML5 Doctor, html5doctor.com

Aside from being written by a bunch of stand-up chaps, HTML5 Doctor has penetrated to depths that no others would dare. An excellent resource.

I'm an excellent name-dropper; throughout this chapter I have been dropping names like nobody's business. But it has not been gratuitous. The people and websites mentioned are leaders in the industry. I highly suggest you follow them on Twitter or Google+, subscribe to their blogs or buy them a beer. Nothing will teach you more about the Web than helping to build a strong online community. I'll leave you to start rebuilding your website. Good luck!



About the Author

Ben Schwarz funds his love of good food (at home) and sake (in bars) by designing sophisticated Web applications using standards-based technology. More than anything else, he is driven by a maniacal desire to produce not only elegant code, but also beautiful software in the hands of its users. He's also a committee member of Ruby Australia and joined the W3C CSS Working Group as an "invited expert" in December 2011.



About the Reviewer

Russ Weakley (1965) was born in Sydney, Australia, and lives in Chatswood West, a leafy suburb of northern Sydney. He has a diploma in visual arts and graphic design and works on user-focused Web design, markup and code, project management, user experience, accessibility and training. He has been working on the Web since 1995.

Russ has two young kids and, therefore, no time for hobbies. He used to have three dogs, but since all died, he doesn't have any. Russ' favorite color is black. An important lesson he's learned during his career is that "This too shall pass"—it applies to everything in life, including business. His personal message to readers is "Get busy!"



Restyle, Recode, Reimagine With CSS3

Written by David Storey and Lea Verou

Reviewed by Tab Atkins

BY NOW, WE'VE ALL GOTTEN THE MESSAGE that Web standards are the fundamental baseline for our work and that semantic HTML is the best thing since sliced bread. We all build table-less layouts and struggle for semantic correctness. However, many of us still code according to the methods popularized during the first years when Web standards became popular among authors—some of them include verbose markup and nasty CSS workarounds, to name just a few.

While these approaches aren't wrong per se, they are no longer optimal, and they sometimes hold us back from becoming better, more efficient designers.

In the previous chapter we learned how to recode markup to be more lean, semantic and modern. In this chapter, we will learn how to recode CSS to reduce the number of images, HTTP requests, presentational JavaScript and wrapper divs, while making the style more flexible and maintainable.

WEBSITES DO NOT HAVE TO LOOK THE SAME IN EVERY BROWSER

Before we continue, there is one preconception that we finally have to shed and help our friends and colleagues get rid of as well. Although clients and old-school designers often disagree with this, we all need to understand and accept that websites do not have to look the same in every browser. The truth is that only you, your client and your colleagues will check a website in multiple browsers. Your visitors mostly use one, and you're lucky if they actually know what a browser is.

If a website is not broken in their browser, they won't fire up four different browsers to compare—they will just keep using the one they have. If you replace some of your verbose hacks with CSS3, some visitors will get the eye candy, some won't. And that's OK. Not a single person will complain that a website is broken because it doesn't have rounded corners, shadows or gradients. As long as you take care to provide proper fallbacks, no one will think that anything is wrong.

PROVIDING FALLBACKS

Taking advantage of CSS' graceful error handling and the cascade is the easiest way to provide fallbacks. The main idea is simple: in CSS, browsers ignore things they don't understand. So, if they don't understand a property or a value, they will ignore the entire declaration. If they don't understand a selector, they will ignore the entire rule. If they don't understand an @ rule, they will ignore everything inside it.

Take this simple CSS code:

```
a {  
  color: black;  
  color: super-cool-new-css-color;  
  super-cool-new-css-property: awesomesauce;  
}  
a:hawt-new-pseudoclass(awesomeness) {  
  color: hotpink;  
}
```

CODE

In browsers that support `:hawt-new-pseudoclass`, the color of the links that match it will be hotpink. In all other cases, the color will be determined by the first rule. In those cases, if `super-cool-new-css-color` is a supported color, then links will have that color. Otherwise, they'll be black. And if `super-cool-new-css-property` is supported, it will apply to all links; otherwise, it will be ignored and will cause no problems. Sometimes (rarely), this method won't help. For example, when new layout methods are not supported, you will need to define an entirely different layout using many properties that won't get overridden by the new ones. In these cases, feature detection can help. Modernizr¹ by Faruk Ateş and Paul Irish is one of the most used feature-detection libraries. It adds classes to the root element, which you can then use in your CSS to branch accordingly:

```
.no-flexbox section {  
  /* old-style layout goes here */  
}  
.flexbox section {  
  /* cool new flexbox stuff go here */  
}
```

CODE

When downloading the library, you can even customize it and “generate” a light version to detect only the features you need, thus keeping the file's size minimal.

HOW TO READ THIS CHAPTER

Many CSS3 properties featured in this chapter are still vendor prefixed. As soon as a feature becomes a standard and is widely supported in modern browsers, the prefixes can be dropped. This means that in order to use them today, you will likely need to precede the property's name with one or more of the following prefixes:

¹ smashed.by/modernizr

Prefix	Browser Engine
-o-	Opera
-ms-	Internet Explorer
-moz-	Firefox
-webkit-	WebKit-based browsers, such as Chrome and Safari

There are other prefixes,² but they’re usually not worth the hassle. Annoying as it may be most of the time, vendor prefixes are very useful when implementations differ, and they save us from the horrible CSS hacks that we had to use in the past.³

For reasons of brevity and simplicity, the snippets in this chapter feature only un-prefixed code, unless the code needs to be different for some implementations.

Vendor prefixes are often necessary; having -o-, -ms-, -moz- and -webkit- altogether in your stylesheet is the worst-case scenario—not every feature needs them. You can see which prefixes are still needed by checking the reference tables for browser support at [When Can I Use?](#),⁴ [HTML5 Please](#)⁵ or the [Mozilla Developer Network’s](#) documentation.⁶ Also, unless otherwise noted, every snippet in this section relies on the following simple HTML(5) markup:

CODE

```
<html>
<head>
  <meta charset="utf-8" />
  <title>Learning CSS3</title>
</head>
<body>
<section>
  <h1>Learning CSS3</h1>
  <p>This is just some sample content. Don't even bother reading
it; you will just waste your time. Why do you keep reading? Do
I have to use Lorem Ipsum to stop you? OK, here goes: Lorem
ipsum dolor sit amet, consectetur adipi sicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Still
reading? Gosh, you're impossible. I'll stop here to spare you.</p>
```

² For a full list of vendor prefixes, see [smashed.by/vndrprfx](#).
³ If you don't have time to keep up with which prefixes are needed for each feature, you can use a postprocessor like [prefix-free](#) ([smashed.by/prfxfree](#)) that allows you to write CSS without prefixes and adds them via JavaScript.
⁴ [smashed.by/ciuse](#)
⁵ [smashed.by/html5pl](#)
⁶ [smashed.by/mozdevnet](#)

```
</section>  
</body>  
</html>
```

We will cover some basics at the beginning of each section, but then we will quickly jump to more advanced techniques, so please bear with us. We will explore different ways to style the simple page above, learning different CSS3 features along the way. Ready? OK, let's get started.

Web Typography and Text Techniques

The Web has until recently been a typographically dull place. Unless we resorted to image replacement techniques—each with their own benefits and drawbacks—we were stuck with the same set of core and operating-system default fonts that we've had since time immemorial. The problem wasn't just with the fonts, though. We have also lacked the fine-grained control of text layout and ligatures that we have come to expect in other media.

CSS3 is changing all of this. You've never had a better excuse to drop generic fonts and broaden your typographical horizons. We'll go over how to add some rhythm to your typography with the rem unit, experiment with new fonts, control hyphenation, and improve your type all around. After all, text is probably the most important element on the page, so it deserves special attention.

INTRODUCING REM

If you have any experience with elastic layouts, then you have probably used the em unit. While ems are the cornerstone of adaptive Web design, using them can be tricky. They're based on the current element's font size (or its parent for font-size), so some math is involved to calculate how big 1 em actually is. And let's face it, not everybody likes doing math.

The new rem ("root em") unit is a close cousin of the em, with the same benefits but much easier to use. The rem unit is set to the font size of the root element (which would be html in HTML). So, 1 rem is always the same size, no matter where you use it in the document.

To make rems as easy to use as pixels, you can set the font-size of the html element to be 62.5%, which computes to 10px if the user has not adjusted the browser's default

font size. Now when you specify a length in rems, you can just multiply by 10 to get the value in pixels. Because 10 pixels is probably too small for body copy, you can set the size you desire in rems on the body element. This value will then be inherited by all other elements. For example, if you would like your body copy to be 15 pixels, you could do the following:

CODE

```
html { font-size: 62.5%; }  
body { font-size: 1.5rem; /* 15px */ }
```

The only real downside to rem units is browser support. It is now supported by all major desktop browsers, but it is not supported by Internet Explorer (IE) 8 and below. To provide a fallback, you can take advantage of the cascade and set the size in pixels before setting it in rems. This can add considerable heft to your style sheet, so you could use a separate style sheet with conditional comments for IE 9 and below, unless you need to support iOS 3.2, Safari 4 or older versions of Opera.

Setting Up the Rhythm With rem

Now that we have a unit at our disposal that is as easy to use as pixels and as flexible as ems, composing to a vertical rhythm becomes so much easier. What is this vertical rhythm we're talking about? The basic idea is that as you move down the page, the type and page elements follow a set rhythm.

If we draw imaginary lines down the page at fixed intervals, each line of type or each margin or any other page element will take up either one line or exact multiples of lines. They keep to the beat of the page, like a bassist in a blues band. This tightens up the design, ensuring that page elements line up correctly, even across columns. Sticking to the rhythm will help you achieve visual consistency in the page's layout.

First things first. We have to set the rhythm for the page. In the example on the next page, we've set this to 2.3 rem, or 23 pixels. This is set on the body element using the line-height property. We've also set the font-size to 1.5 rem, so that all of the basic typographic elements except the headings inherit this. Having a visual guide to see whether you've skipped a beat is handy, so we've created an SVG image that shows lines at 23-pixel intervals. We've also used a basic reset to remove margins and padding so that they don't interfere with our rhythm:

```
html {
  font-size: 62.5%;
  background: url(line.svg) no-repeat;
}
/* reset */
body, div, dl, dt, dd, h1, h2, h3, h4, h5, h6, pre, p, th, td,
article, section, figure, img {
  margin: 0;
  padding: 0;
}
body {
  font-size: 1.5rem;
  line-height: 2.3rem;
}
```

CODE

If we had only paragraphs of text, our job would almost be complete, even though the paragraphs would be crunched together without any margins.

Making the Headings Follow the Rhythm

Setting up the paragraphs was a piece of cake, as each line of text has a line height smaller than the page's 23 pixel vertical rhythm. But what do we do with elements such as headings, which need either a font size bigger than our line height or extra vertical spacing? The trick here is to make the combination of the top and bottom margins and the line height be exact multiples of our base rhythm:

```
h2 {
  font-size: 2.6rem;
  line-height: 4.6rem; /* two x base rhythm */
  margin: 2.3rem 0 0 0; /* base rhythm top and bottom */
}
h3 {
  font-size: 2.1rem;
  line-height: 2.3rem; /* base rhythm */
  margin: 2rem 0 .3rem 0; /* 2 top + 0.3 bottom == base rhythm */
}
```

CODE

It doesn't matter how many lines our headings take up; they'll always take up exact multiples of the base rhythm, so the beat will continue down the page. We can then follow the same technique for all elements on the page, such as images and code snippets.

Remember that borders also take up space, so you will have to account for them, such as by reducing the margins. Box shadows, on the other hand, do not take up space, so you are free to do what you like with them.

If we had used ems in this example, we would have had to recalculate how big the line height and margins have to be based on the text's size. With rems, we can almost ignore the font size completely, except to make sure that the line height is big enough to include the text without overlapping.⁷

GEORGIA ON MY MIND

We love Helvetica and Georgia, but isn't it about time these venerable fonts took a vacation in the mountains to ease their overworked shoulders and stems? A whole typographical world is out there, and Web fonts are our passport to experience the delights of these exotic fonts.

Font formats have a whole political back story, which we won't bore you with here. The fallout is that you will need to use a multitude of formats if you want a robust cross-browser solution. Fortunately, a host of solutions have popped up to help us deal with this, so you don't have to think about it too much—unless you want to.

Rolling Your Own @font-face

Your first option is to use a font that you've sourced yourself, served from your own server. We can't stress enough that you should check the font's license before doing this to make sure you have the proper rights. Not every free or commercial font allows you to use it as a Web font; and even if it does, it might require you to buy an additional license for this purpose.

Once you've chosen a font, you have to prepare it in the right format and link to it via the @font-face rule. We've found the best method is to use the excellent Font Squirrel service.⁸ It enables you to upload a font, and then it spits out the font packaged in all of the formats you will need, along with the @font-face rule itself. All you need to do is move the fonts to your server, copy the code into your style sheet, and then reference the font wherever you want to use it.

⁷ You can view the example at smashed.by/fntexmpl.

⁸ smashed.by/fntsqr

In the example for this section, we've used Museo Sans 500 font for the body copy and Museo 700 for the headings. Both of these are provided for free by Jos Buivenga.⁹ Let's look at the CSS created by Font Squirrel for the Museo 700 font:

```
@font-face {  
  font-family: 'Museo-700';  
  src: url('1F9920_0_0.eot');  
  src: url('1F9920_0_0.eot?#iefix') format('embedded-opentype'),  
       url('1F9920_0_0.woff') format('woff'),  
       url('1F9920_0_0.ttf') format('truetype');  
}
```

CODE

The `font-family` property sets the name by which you will need to refer to the font in the rest of the style sheet, and the `src` property links to the font itself. The browser will check each URL in turn until it finds a format it supports. The initial `src` property is to support older versions of IE, which have a bug with the regular method. Once you insert this in your style sheet, you can refer to the font the regular way:

```
h2 {  
  font-family: "Museo-700", serif;  
}
```

CODE

You should be all set, but you need to be aware of an additional detail. The typeface has a 700 weight, which is bold. Note that, by default, browsers will assume that the font file you specify in `@font-face` is unbolded, unitalicized, and un-everything-else-that-fonts-can-be. If you use it in an element with `font-weight: bold`, they'll *artificially* bold the font for you. Since the Museo font is used in headings, it's already bolded, and stacking artificial bolding on top will result in a muddy mess. To avoid this, we have to tell the `@font-face` rule that this is already a bold font, using the `font-weight` descriptor:

```
@font-face {  
  font-family: 'Museo-700';  
  ...  
  font-weight: bold;  
}
```

CODE

⁹ smashed.by/exjlb

Now modern browsers will know it is a bold font and won't try to embolden it further. By including additional `@font-face` rules with the same font-family but different values for font-weight, font-style and font-stretch, you can combine multiple font files into a single font name and the browser will use the correct one when switching to bold or italic.

Using an Online Font Service

Instead of rolling your own, you could use one of the many online font-serving services available. Two of the most popular are Google Web Fonts for free fonts and TypeKit for subscription-based fonts. These subscription services are often the only legal way to access well-known commercial fonts.

Each service has its own way of including the font. In our example, we've used Google Web Fonts for the main page's heading. Google provides three methods: via the link element, via `@import` and via JavaScript. We've opted for the link element and added it to the head of the document, like so:

CODE

```
<head>
<meta charset="UTF-8" />
<title>Typography in the CSS3 era</title>
<link href='http://fonts.googleapis.com/css?family=Abril+Fatface'
rel='stylesheet' type='text/css'>
...
</head>
```

The font can then be referred to in the style sheet as before by its name “Abril Fatface”:

CODE

```
h1 {
  font-family: "Abril Fatface";
  font-weight: normal;
  ...
}
```

In this case, we set the font-weight to normal because Firefox tries to artificially bold an already bold font. Because we can't control the `@font-face` rule to set it as a bold font, we had to do the opposite and specify a normal weight every time it is used. As you can see, the result has beautiful rhythmic type, without a whiff of core fonts:

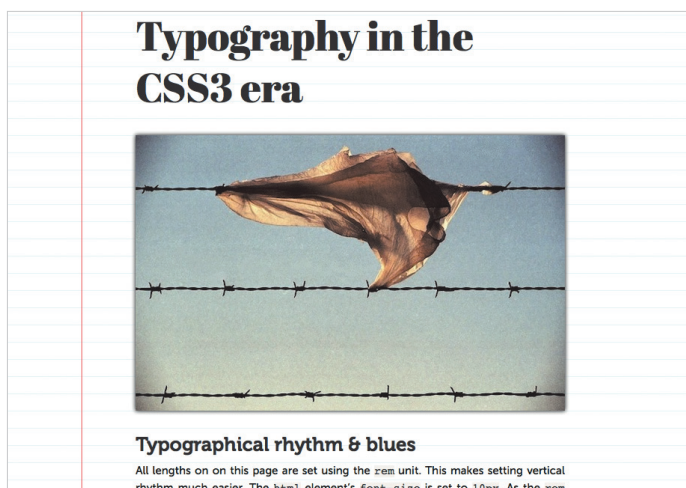


Figure 4.1. A beautiful rhythmic type, using vertical rhythm and the `@font-face` attribute.

MEET NEW MEMBERS OF THE FAMILY USING FONT-STRETCH

The `font-stretch` property has perhaps one of the most unfortunate names in CSS. It doesn't artificially stretch the font (a typographic no-no), but instead allows you to access the expanded and condensed fonts in a typeface. Without `font-stretch`, if you wanted to use the Helvetica Neue Condensed font, for example, you would need to use it as a Web font (license permitting) and map it to a different font-family name. With `font-stretch`, it's as easy as this:

```
h1 {
  font-family: "Helvetica Neue";
  font-stretch: condensed;
}
```

CODE

Most typefaces don't come with expanded or condensed fonts, so outside of Helvetica Neue, you'll probably find most use for this when using custom fonts via Web fonts. The values for `font-stretch` are: `normal`, `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `extra-expanded` and `ultra-expanded`. The browser will map the values to the closest font in the typeface. In the case of Helvetica Neue on OS X, all of the condensed values map to Helvetica Neue Condensed, unless you have additional fonts installed.

HYPHENATE YOUR WORDS

We’re used to seeing text aligned left and set ragged right on the Web. In print, justification along with hyphenation are often used to avoid rivers in the text (large gaps between words and characters). Until recently, hyphenation has not been available on the Web, but this is starting to change. The `hyphens` property is available with prefixes in Safari (since 5.1) and Firefox and is coming in IE 10. Chrome currently does not support it, although it does parse the declaration.

In the example below, we’ve set the text to be justified and the hyphenation to be automatic:

CODE

```
p {
  text-align: justify;
  hyphens: auto;
  ...
}
```

The `auto` keyword tells the browser to use its hyphenation resource to break words at the appropriate places and to insert hyphens at the ends of lines. Each language requires its own hyphenation resource, so support will vary from language to language. If you insert your own soft-hyphen character (`­`) that will overrule the browser’s automatic behavior. If you would like to manually set hyphens, you can use the `manual` value. Disabling hyphenation (the default) can be done with the `none` keyword.

Here is an example with (left) and without (right) hyphenation. Notice the large uneven spacing between words on the second and sixth lines in the unhyphenated text.

What is typography?

Typography (from the Greek words *τύπος* (typos) = form and *γραφή* (graphe) = writing) is the art and technique of arranging type in order to make language visible. The arrangement of type involves the selection of typefaces, point size, line length, leading (line spacing), adjusting the spaces between groups of letters (tracking) and adjusting the space between pairs of letters (kerning). Type design is a closely related craft, which some consider distinct and others a part of typography; most typographers do not design typefaces, and some type designers do not consider themselves typographers.[2][3] In modern times, typography has been put into motion — in film, television and online broadcasts — to add emotion to mass communication.

What is typography?

Typography (from the Greek words *τύπος* (typos) = form and *γραφή* (graphe) = writing) is the art and technique of arranging type in order to make language visible. The arrangement of type involves the selection of typefaces, point size, line length, leading (line spacing), adjusting the spaces between groups of letters (tracking) and adjusting the space between pairs of letters (kerning). Type design is a closely related craft, which some consider distinct and others a part of typography; most typographers do not design typefaces, and some type designers do not consider themselves typographers.[2][3] In modern times, typography has been put into motion — in film, television and online broadcasts — to add emotion to mass communication.

Figure 4.2. An example of a paragraph with (left) and without (right) hyphenation.

Magazine-Like Layouts

In books and magazines, laying out text over multiple columns is common. This can aid readability by keeping line lengths short, while still allowing the designer to take full advantage of the width of the page (or the browser's window in the case of the Web).

The Multi-column specification brings this capability to CSS. It enables you to specify a number of attributes for columns, such as number, width and gap size. Let's transform the demo above to use columns to get a taste of what can be achieved.

SPECIFYING THE COLUMNS

You can set either the width or the number of columns. The browser will calculate the other value based on the one specified and the available width of the container. In this example, we will specify two columns only for sections that are nested two levels deep within an article. This will avoid giving us multiple columns in the introductory section and prevent columns from being nested if we ever add sections that are more deeply nested.

The number of columns can be set with the `column-count` property, which accepts an integer as its value. We will also apply a gap between the columns so that they're nicely spaced out. This can be applied using the `column-gap` property:

```
article > section > section {  
  column-count: 2;  
  column-gap: 2.6rem;  
}
```

CODE

To specify the width of the column instead, you can use the `column-width` property, which accepts length values in a similar way to `column-gap`.

SPANNING COLUMNS

Making text and elements such as images span multiple columns is often useful. This can be achieved using the `column-span` property. It is currently an all-or-nothing affair: you can either make it span all columns using the `all` keyword or not span at all with the `none` keyword. Specifying an exact number of columns to span is not possible. In our example, we will make the `h3` headings span the full width of the columns.¹⁰

¹⁰ The `column-span` property is not widely supported yet, so your layout could break if it relies heavily on this feature.

```
h3 { column-span: all; }
```

The result should look something like the figure below.¹¹

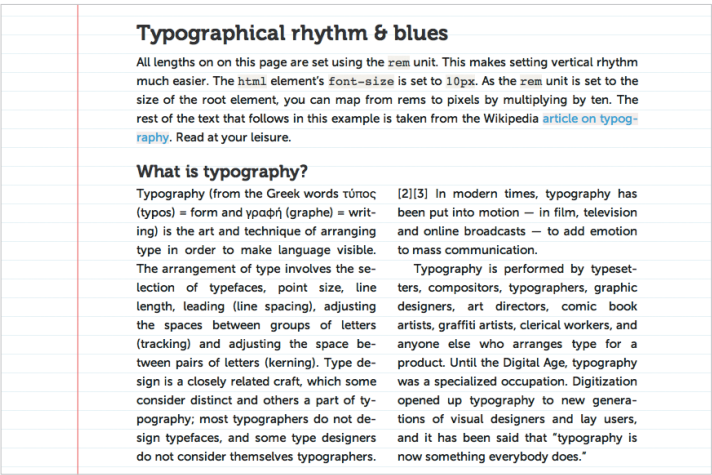


Figure 4.3. An advanced multi-column, magazine-like layout. The text block at the top spans multiple columns at the bottom.

Another word of caution with multiple columns. Using it on long passages of text can cause readability issues by forcing the reader to scroll up and down the page for each column. Consider using it only for short passages where the length of the column would likely be shorter than the user’s browser window. However, when a page is printed, multi-column elements will break and continue properly across pages, so it’s fine to use multiple columns on long passages in print stylesheets.

Layout Techniques

Perhaps the hardest thing to do with CSS to this day is laying out a page. Something as simple as centering an element on the page can be quite difficult to achieve. Implementing a two- or three-column layout with columns of equal height can turn your hair gray (if it isn’t already).

These issues boil down to the fact that CSS has never really provided a method for laying out pages. We’ve been (ab)using float-based layout for years, but it is just one

¹¹ You can try it for yourself by visiting smashed.by/mltclmn.

big hack. Just as tables were not designed for layout, neither were floats. They were designed to do things like float an image within a block of text. But when we lack the proper tools, we improvise, and we should thank the humble float for playing out of position all these years. Now, with CSS3, we can ask the float to move over, because we have new layout toys to play with. One of the most promising is the Flexible Box Layout.

FLEXIBLE BOX LAYOUT

Flexible Box Layout (or Flexbox) is a new box model optimized for UI design. The children of a box that is set to use the Flexbox model are laid out along either the horizontal or vertical axis. The widths of these children expand or contract to fill the available space, based on the flexible length they are assigned.

Flexbox has had a storied existence. It started as a feature for Mozilla's XUL and has been rewritten multiple times. The specification is only now reaching maturity, and we have fairly complete support in WebKit and Chrome nightlies. It is definitely worth knowing about because the specification will probably be implemented quickly, if it hasn't already happened by the time you read this, especially with the quick release schedules of browsers such as Chrome and Firefox.

EXAMPLE: HORIZONTAL AND VERTICAL CENTERING (OR THE HOLY GRAIL OF WEB DESIGN)

Being able to center an element on the page is perhaps the number one request among Web designers—yes, probably even more so than the venerable parent selector or putting IE 6 out of its misery (OK, maybe a close second then). With Flexbox, this is trivially easy. Let's start with a basic HTML template, with a heading that we want to center:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <title>Centering an Element on the Page</title>
  </head>
  <body>
    <h1>OMG, I'm centered</h1>
  </body>
</html>
```

CODE

Nothing special here, not even a wrapper div. The magic all happens in the CSS:

CODE

```
html {  
  height: 100%;  
}  
body {  
  display: flexbox; /* this value needs prefixes */  
  flex-align: center; /* these properties also need prefixes */  
  flex-pack: center;  
  margin: 0;  
  height: 100%;  
}  
h1 {  
  display: flexbox;  
  flex-align: center;  
  height: 10rem;  
}
```

This is not exactly all the CSS needed for the example, because we've stripped out the extra styling that you probably already know how to use in order to save space. We've also left out the prefixes for the same reason. Only WebKit supports these (with the `-webkit-` prefix), but don't be surprised if Mozilla, Opera and IE support them in the near future. Best to add the prefixes just in case. Let's look at the CSS that is needed to center the heading on the page. First, we set the `html` and `body` elements to have a 100% height and remove any margins. This will make the container of our `h1` take up the full height of the browser's window. We just need to center everything now.

ENABLING FLEXBOX

Because the `body` element contains the heading we want to center, we will set its `display` value to `flexbox`:

```
body {  
  display: flexbox; /* this value needs prefixes */  
}
```

This switches the body element to use the Flexbox layout, rather than the regular block layout. All of its children in the flow of the document (i.e. not absolutely positioned elements) will now become Flexbox items.

What do we gain now that our elements have been to yoga class and become all flexible? They gain untold powers: they can flex their size and position relative to the available space, they can be laid out either horizontally or vertically, and they can even achieve source-order independence. (Two holy grails in one specification? We're doing well.)

CENTERING HORIZONTALLY

Next, we want to horizontally center our h1 element. No big deal, you might say; and it is somewhat easier than playing around with auto margins. We just need to tell the Flexbox to center its Flexbox items. By default, Flexbox items are laid out horizontally, so setting the flex-pack property will align the items along the main axis:

```
body {  
  display: flexbox;  
  flex-pack: center;  
}
```

[CODE](#)

The other possible values are start, end and justify. The start value aligns to the left (or to the right with right-to-left text), end aligns to the right and justify evenly distributes the elements along the axis.

If you want to explicitly set the axis that the element is aligned along, you can do this with the flex-flow property. The default is row, which will give us the same result that we just achieved. To align along the vertical axis, we can use flex-flow: column. If we add this to our example, you will notice the element will be vertically centered but will lose the horizontal centering. Reversing the order by appending -reverse to the row or column values is also possible (flex-flow: row-reverse or flex-flow: column-reverse), but that won't do much in our example because we have only one item.

CENTERING VERTICALLY

Centering vertically is as easy as centering horizontally. We just need to use the appropriate property to align along the “cross axis.” The what? The cross axis is basically the axis other than the main one; so, if Flexbox items are aligned horizontally, then the cross axis would be vertical, and vice versa. We set this with the flex-align property:

CODE

```
body {  
  display: flexbox;  
  flex-pack: center;  
  flex-align: center;  
}
```

This is all there is to centering elements with Flexbox! We can also use the start and end values, as well as baseline and stretch. Let's see the finished example. To try it, point your Web browser to smashed.by/flxbox1.

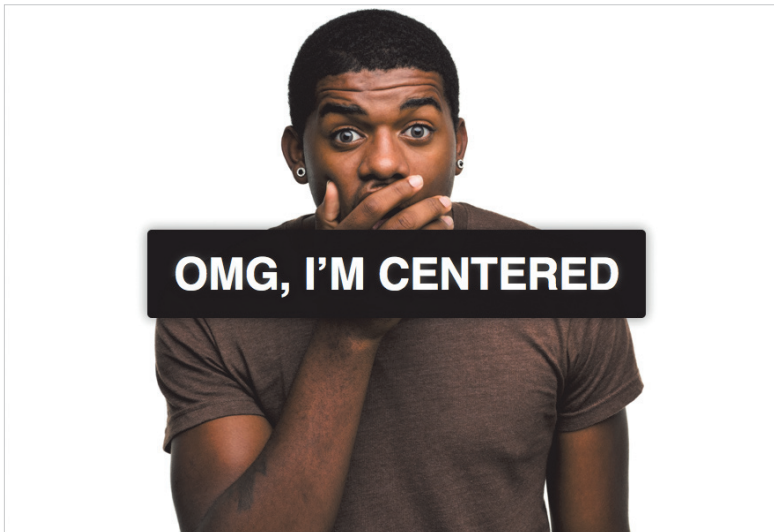


Figure 4.4. Simple horizontal and vertical centering using Flexbox.

You might notice that the text is also center-aligned vertically inside the `h1` element. This could have been done with margins or a line height, but we used Flexbox again to show that it works with anonymous boxes (in this case, the line of text inside the `h1` element). No matter how high the `h1` element gets, the text will always be in the center:

CODE

```
h1 {  
  display: flexbox;  
  flex-align: center;  
  height: 10rem;  
}
```

FLEXIBLE SIZES

If centering elements was all Flexbox could do, it'd be pretty darn cool, but there is more. Let's see how Flexbox items can expand and contract to fit the available space within a Flexbox element. For this, we'll use another example, so point your browsers to smashed.by/flexbox2.



Figure 4.5. An interactive slideshow built using Flexbox.

The HTML and CSS for this example are similar to the previous one. We're enabling Flexbox and centering the elements on the page in the same way. In addition to this, we want to make the title (inside the header element) remain consistent in size, while the five boxes (the section elements) should adjust in size to fill the width of the window. To do this, we use the new flex function as the value of the width property:

```
section {
  /* removed other styles to save space */
  flex: 1;
  height: 250px;
}
```

CODE

What we've just done here is to make each section element take up 1 flex unit. Because we haven't set any explicit width, each of the five boxes will be the same width. The header element will take up a set width (277 pixels) because it is not flexible. We divide the remaining width inside the body element by 5 to calculate the width of each of the section elements. Now if we resize the browser window, they will grow or shrink.

In this example, we’ve set a consistent height, but this could be set to be flexible, too, in exactly the same way. We probably wouldn’t always want all elements to be the same size, so let’s make one bigger. On hover, we’ve set the element to take up two flex units:

CODE

```
section:hover {  
  flex: 2;  
  cursor: pointer;  
}
```

Now the available space is divided by 6 rather than 5, and the hovered element gets twice the base amount. Note that an element with two flex units does not necessarily become twice as wide as one with one unit. It just gets twice the share of the available space added to its “preferred width.” In our examples the “preferred width” is 0 (the default).

SOURCE-ORDER INDEPENDENCE

For our last party trick in this section, we’ll study how we can achieve source-order independence in our layouts. When clicking on a box, we will tell that element to move to the left of all the other boxes, directly after the title. All we have to do is set the order with the `flex-order` property. By default, all flex items are in the 0 position. Because they’re in the same position, they follow source order.

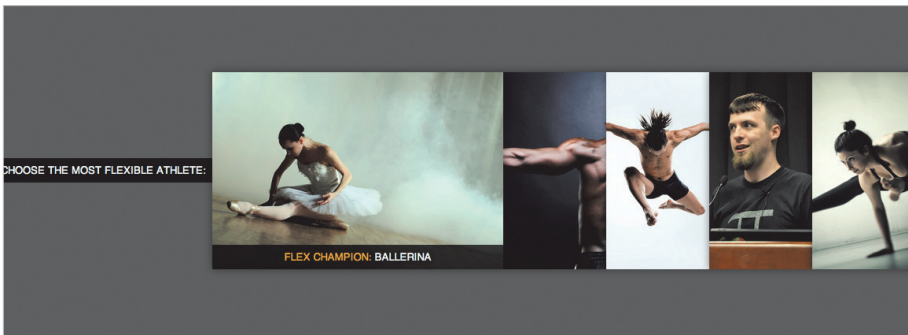


Figure 4.6. An interactive slideshow with `flex-order`.

To make our chosen element move to the first position, we just have to set a lower number. We chose -1. We also need to set the header to -1 so that the selected section element doesn’t get moved before it:

```

header {
  flex-order: -1;
}

section[aria-pressed="true"] {
  flex-order: -1; /* lower than 0 so moves before other section elements
*/
  flex: 3;
  max-width: 370px; /* stops it from getting too wide */
}

```

Hopefully, this has given you some inspiration and enough introductory knowledge of Flexbox to enable you to experiment with your own designs.

Working With Images

Now that we've got a better understanding of how we can build advanced layouts with CSS3, let's move on to some visual techniques that we can use to spice up our layouts a bit.

MULTIPLE BACKGROUNDS AND GRADIENTS

Have a look at the following style, at two different sizes:¹²



Figure 4.7. Wide view of multiple backgrounds and gradients.



Figure 4.8. Narrow view.

Going just by the image, how would you go about coding it in CSS? You would probably whip up the basic layout in a couple of minutes, as shown on the next page.

¹² smashed.by/multbgs

CODE

```
html {  
  background: #222;  
  min-height: 100%;  
}  
body {  
  margin: 0;  
}  
section {  
  max-width: 25em;  
  margin: 0 auto;  
  padding-top: 150px;  
  color: white;  
  font: italic 100%/1.5 'Palatino Linotype', Georgia, serif;  
}
```

But how to add the stars, skyline and moon? We have only two elements to use (html and body—the section element is too narrow), but we need four background images.

In the past, we'd probably just shrug and add a few wrapper divs. Or we would give up on flexibility and combine some of the images. But these days, we can have our cake and eat it, too. Meet multiple background images:

CODE

```
html {  
  background: url("moon.png") no-repeat 100% 1em,  
    url("stars.png") repeat-x 0 0,  
    url("city.png") repeat-x bottom,  
    linear-gradient(black, #444);  
  background-color: #222;  
  min-height: 100%;  
}
```

You probably noticed that the last background image is not an external URL. Indeed, we can now generate gradients right in the CSS file. The gradient used above is a simple top-to-bottom gradient with two colors. But you can do much more complex things with CSS gradients. You can have multiple colors at different positions or different angles and even radial gradients. You can also create all sorts of geometric patterns with a few CSS gradients.

However, current support for CSS gradients is not yet as good as the support for multiple backgrounds. The way we wrote our code above, even though we get the fallback color when multiple backgrounds are not supported, we still get it when only gradients are not supported. A better option would be to provide two fallbacks:

```
html {  
  background: url("moon.png") no-repeat 100% 1em,  
    url("stars.png") repeat-x 0 0,  
    url("city.png") repeat-x bottom;  
  background: url("moon.png") no-repeat 100% 1em,  
    url("stars.png") repeat-x 0 0,  
    url("city.png") repeat-x bottom,  
    linear-gradient(black, #444);  
  background-color: #222;  
  min-height: 100%;  
}
```

CODE

A bit repetitive, isn't it? And it's even worse with vendor prefixes. But we also have the body element, so we can take advantage of that and move the gradient to an element other than the one the rest of our backgrounds are in:

```
html {  
  background: #222;  
  background: linear-gradient(black, #444);  
  height: 100%;  
}  
body {  
  margin: 0;  
  background: url("stars.png") repeat-x 0 0;  
  background: url("moon.png") no-repeat 100% 1em,  
    url("stars.png") repeat-x 0 0,  
    url("city.png") repeat-x bottom;  
  min-height: 100%;  
}
```

CODE

Much better, and this gets the most out of each browser's capabilities.

BACKGROUND-ORIGIN, BACKGROUND-SIZE, OUTLINE

Let's say we have been assigned to create the design displayed on the right using CSS.¹³ Again, we're sure you could quickly whip up the basic layout and perhaps integrate some of the basics that you probably know about RGBA and text shadows:

CODE

```
html {
  min-height: 100%;
  background: url('rainbow-wood.jpg');
}

section {
  max-width: 25em;
  padding: 3em;
  margin: 4em auto;
  background: black url('logo.svg') no-repeat bottom right;
  color: white;
  font: bold 100%/1.5 sans-serif;
  text-shadow: 0 -.1em .2em black;
}

h1 {
  margin-top: 0;
}
```

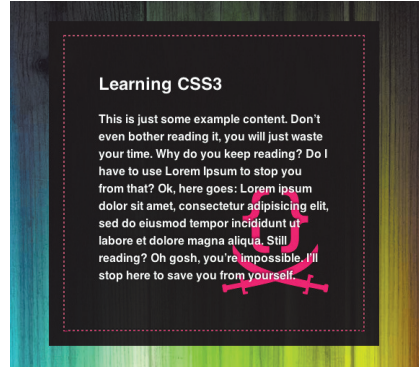


Figure 4.9. A design that we are going to reproduce with CSS3.

The result (see next page) is close to what we want, but there are several problems:

- The html background image is huge. That's good for a huge screen, but generally we would want to resize it for smaller screens.
- The pink logo is too small and is stuck in the bottom-right corner with no spacing from the edge.
- There is no pink “stitching.”

Let's tackle these issues one by one.

¹³ smashed.by/bgexample1

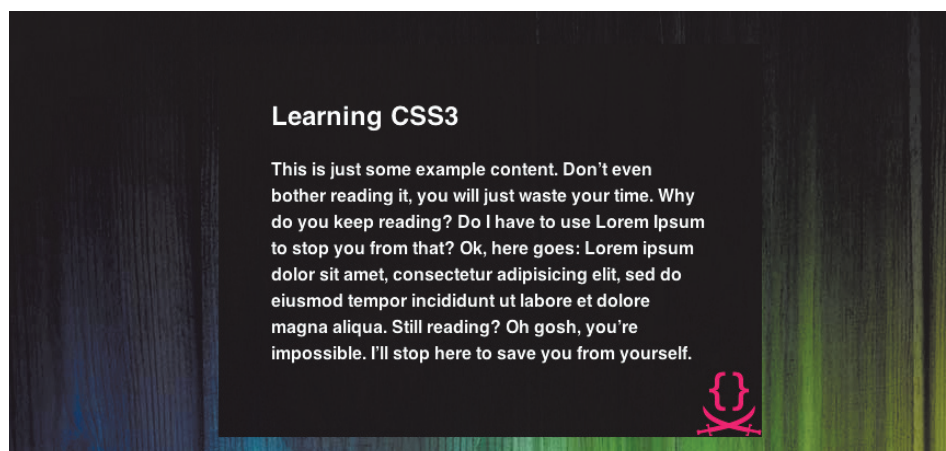


Figure 4.10. The result after we apply our CSS 2.1 knowledge.

RESIZING THE BACKGROUND TO FIT

CSS3 gives us a new property to control the size of a background image: `background-size`. This enables us to use the same image at multiple sizes. We can either explicitly define a size or use one of the two sizing keywords, `contain` and `cover`:

- `cover`
The image will adjust to completely contain the element. This is what we would use to make Flickr-like square thumbnails, where every image is cropped to a square.
- `contain`
The image will adjust to fit the element but without being cropped.

In this case, we can't use `background-size: contain` because we don't want the background color to show through. Instead, we want our image to shrink or stretch to cover the entire viewport.

FIXING THE LOGO

To enlarge the SVG logo, we use `background-size` again, but explicitly setting a size this time:

```
background-size: 150px;
```


By providing only one value instead of two, we preserve the aspect ratio, while the image stretches to a width of 150 pixels. But it's still stuck at the bottom. In the old days, we would just shrug and edit the image to include the spacing within it. Now with CSS3, we have a bit more granular control over the placement of the image. If we analyze the examples above a bit more carefully, we would see the problem—the image is in the bottom right of the padding area, but we'd like it to be at the bottom right of the text. CSS3 gives us a new property, `background-origin`, that controls whether the image is placed relative to the border box, the padding box or the content box.

In this case, we'd like to place the image at the bottom-right edge of the content box and not of the padding box, which is the default. This line of CSS will do that:

```
background-origin: content-box;
```

ADDING THE PINK STITCHING

This would have been straightforward if the pink-dashed border was at the edge of the container. This CSS snippet might have done it:

```
border: 1px dashed #f06;
```

But how do we move this border inside the container? We can't. Instead, we'll shrink the container and then add the extra color outside of the border with the `outline` property:

```
border: 1px dashed #f06;  
outline: 20px solid rgba(0,0,0,.8);
```

A different approach would be to use only the `outline` property with an `outline-offset` of `-20px` (we will not use this technique here). Here is the full snippet:

CODE

```
html {  
  min-height: 100%;  
  background: url('rainbow-wood.jpg');  
  background-size: cover;  
}  
section {  
  max-width: 20em;  
  padding: 3em;
```

```

margin: 4em auto;
border: 1px dashed #f06;
outline: 20px solid rgba(0,0,0,.8);
background: url(logo.svg) no-repeat bottom right;
background-color: black;
background-color: rgba(0,0,0,.8);
background-origin: content-box;
background-size: 150px;
color: white;
font: 100%/1.5 sans-serif;
text-shadow: 0 -.1em .2em black;
}
h1 {
  margin-top: 0;
}

```

Background Clipping

Now, consider this simpler variation of the previous style, as seen below.¹⁴

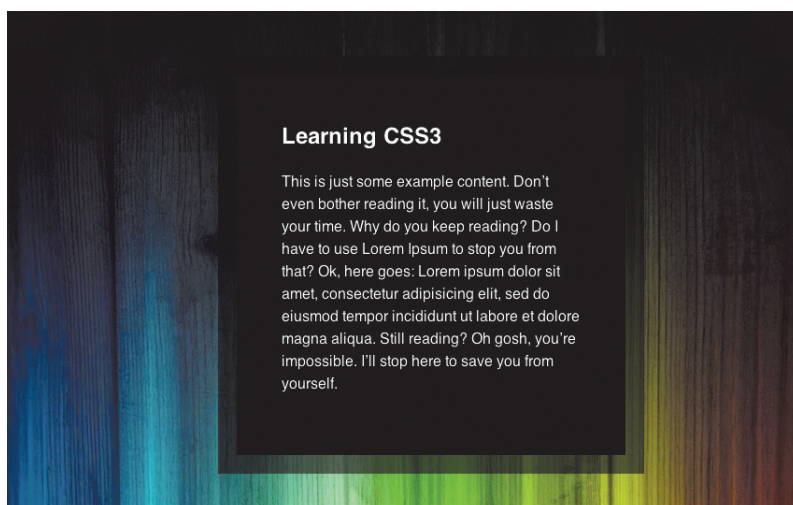


Figure 4.11. Allowing semi-transparent borders with background clipping.

¹⁴ smashed.by/bgclipping

Your first crack at achieving this would probably be something like this:

CODE

```
html {
  min-height: 100%;
  background: url('rainbow-wood.jpg') top;
  background-size: cover;
}
section {
  max-width: 20em;
  padding: 3em;
  margin: 4em auto;
  border: 20px solid rgba(0,0,0,.5);
  background-color: black;
  color: white;
  font: 100%/1.5 sans-serif;
  text-shadow: 0 -.1em .2em black;
}
h1 {
  margin-top: 0;
}
```

But when you try it out, you will notice that the border is not actually semi-transparent. Why did this happen? The reason is that, by default, the background extends beneath the border as well. Remember all those times when you used border styles with gaps (i.e. dotted, dashed and double) in the days of CSS2.1? It's the same thing, but now we have tools to control this behavior, in particular the `background-clip` property. Its default value is `border-box`, which results in the behavior you've experienced so far. But we can change it to `padding-box`:

```
background-clip: padding-box;
```

This will cause the background to clip exactly where we want it.

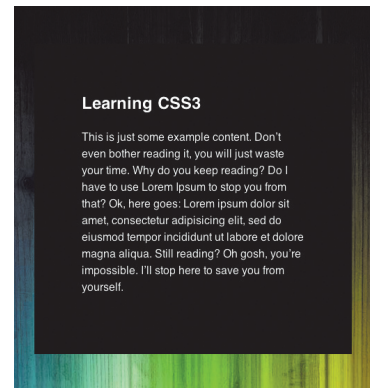


Figure 4.12. Our border is not actually semi-transparent. Why?

BORDER IMAGES

What about this style?¹⁵ How would you go about getting the zig-zag border? You would probably be forced to use a fixed width and height and a huge background image with the no-repeat property. If the viewer had set a font size larger than the browser's default or didn't have a font installed close enough to the one you're using, the text would flow beyond the fixed background image. We've all been there, seen that.

The border-image property is a powerful new feature that lets us use a single small image for both the background and the border of an element. We define how it's scaled or repeated, and it can be different for each edge. In this case, we'll use Figure 4.14.

And here is the CSS snippet that goes with it:

```
html {
  min-height:100%;
  background: white url(background.jpg);
}
section {
  max-width: 20em;
  padding: 3em;
  margin: 4em auto;
  border: 20px solid transparent;
  -webkit-border-image: url(cloth.svg) 33.33% round;
  -moz-border-image: url(cloth.png) 33.33% round;
  -o-border-image: url(cloth.svg) 33.33% round;
  border-image: url(cloth.svg) 33.33% round fill;
  font: 100%/1.5 sans-serif;
  text-shadow: 0 1px white;
}
h1 {
  margin-top: 0
}
```

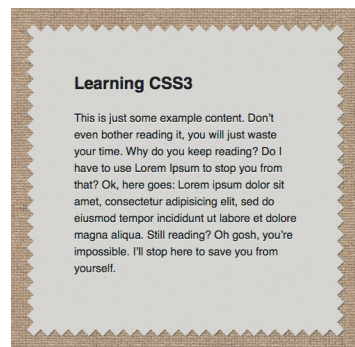


Figure 4.13. A border image example.



Figure 4.14. The cloth.svg image we are going to use for the fancy borders.

¹⁵ smashed.by/brdimages

The `border-image` property specifies which image to use, how wide each slice should be (in this case, we want to cut it in thirds, both horizontally and vertically, so it's just one number) and how these slices should be repeated or stretched (the `round` keyword repeats it but scales it a bit so that we have a round number of repetitions). The `fill` keyword keeps the middle slice as a background, but it's not supported by the prefixed implementations. Luckily, the prefixed implementations have this behavior by default. Also, you might have noticed that we are serving a SVG to every browser except Firefox. This is because at the time of writing this chapter, Firefox is very buggy with SVG border images. But Mozilla is working on it, so you might want to check for yourself.

When we use `border-image`, the border style and color that we define are ignored as long as the border's style is not set to `none`. Only the border-width has an effect. But we usually want to specify something like `solid transparent`, so that when `border-image` is not supported we don't get an ugly thick border. Your mileage may vary, and in some cases a "real" border fallback is better than none.

Transforms

CSS transforms provide the ability to apply one or a series of transformations to an element. These include scaling, skewing, rotating and translating. Many of these effects have not been possible in HTML before, beyond converting the content into an image, with all of the drawbacks that this entails.

We will construct a simple image gallery to demonstrate how to apply a number of these transforms. Hovering over an image and clicking on it will trigger different transformations. You can load this in your browser by going to smashed.by/trnsfrms.

Applying a Simple Transform

In the demo, each image is included in a figure element, like so:

CODE

```
<figure>
  
</figure>
<!-- repeat figure for each image -->
```

We are including all of the images at their full size and scaled down, rather than using separate thumbnail images. This can be done with the `scale` transform:

```
figure {  
  float: left;  
  z-index: 1;  
  margin: 1rem;  
  width: 160px;  
  height: 160px;  
  transition-duration: 1s;  
  transform: scale(0.25);  
}
```

The transform property accepts a space-separated list of transform functions as a value. Here we are using the scale function to reduce the element to a quarter of its size. This will scale from the center of the element by default. Values lower than 1 will scale the element down, and values higher than 1 will scale it up.

Bear in mind that transforms do not effect the flow of the document, so each element will take up the same space as it did before it was transformed. For this reason we have also set a width and height that is one fourth of the original image's dimensions, so that we don't have a lot of empty space around our scaled images.

After applying some additional styling to the body and to the images themselves, we should get something like the following:



Figure 4.15. A simple image gallery without transformations.

APPLYING ADDITIONAL TRANSFORMS

Transforms can get more interesting when we chain them together and combine them with transitions. We've already set up our transition in the previous code snippet, so we just have to change the transform on hover to make it take effect:

CODE

```
figure:hover {  
    transform: scale(0.33) rotate(2deg);  
    z-index: 100;  
    cursor: pointer;  
}
```

Here we are applying two transforms. Any number of transforms can be applied by specifying them in a space-separated list. Here we are scaling the figure by a third, then rotating it by 2 degrees. The rotate function transforms the element around the transform axis clockwise. To rotate counterclockwise, we use a negative value:

CODE

```
figure:nth-of-type(even):hover {  
    transform: scale(0.33) rotate(-2deg);  
}
```

Each transform function is applied in turn, so the element is first scaled and then rotated. Grasping this is important because the order will affect the size of units if scaled and will affect the direction of each axis if rotated.

Because all transform functions are applied using one property, if you just want to transition one value, you will still have to specify the full list, or else the other values will return to their defaults. On hovering over an image in our demo, the image will both scale and rotate as follows. You might have noticed that the image after the hovered element also gets transitioned and transformed after a short delay. Here we've added a `translateX` transform, which moves the element along the x axis:



Figure 4.16. Transformation applied to an image on hover using CSS3.


```
figure:hover + figure {  
  transform: scale(0.25) rotate(-1deg) translateX(15px) ;  
  transition-duration: 1.5s;  
  transition-delay: .2s;  
}
```

CODE

You'll notice that it doesn't strictly move along the x axis or translate by 15 pixels, because the scale transform reduces the length by a quarter, and the rotate transform moves the x axis by 1 degree counterclockwise.

The `translateX` function takes one value, using any valid length unit. There is also a corresponding `translateY` function. Both can be set together using the `translate` function, which accepts two values (x then y), separated by commas.

The final set of transform functions are `skewX`, `skewY` and `skew`. They are specified the same as the transform functions, but they skew the element on one or both axes. This is commonly used to apply a simulated 3-D perspective.

ADJUSTING THE TRANSFORM'S ORIGIN

All of the transforms in the demo use the default origin for the transform, which is the center of the element. This can be specified using the `transform-origin` property. It accepts one to four space-separated values, which can be lengths, percentages or the keywords of top, left, bottom, right or center. If only one value is specified, then the second value will default to center. If one of the values isn't a keyword, then the first value will be for the horizontal position. When using a keyword, you can define an optional offset value, specified as a percentage or length, directly after it:

```
transform-origin: top 10% left 25%;
```

This would set the transform's origin to be the point that intersects 10% from the top of the element and 25% from the left of the element.

Selectors

CSS selectors are often considered to be the least fancy playground for CSS developers. You might think that we don't really need any additional selectors to target elements in our mark-up, or you might be struggling with nasty jQuery workarounds to overwrite default CSS values in particular situations. In both cases, with CSS3 selectors, you are

in for a treat. Let's see what possibilities we have now that CSS3 selectors are gaining support in Web browsers.

HIGHLIGHTING THE CURRENT LINK TARGET

Let's return to the vertical rhythm example from the section on typography. As you probably know, you can link to specific sections on the page by using a hash identifier after the URL; so, we could use something like `http://example.com/index.html#def` to link to the section on typography definitions. When a page has a few large sections, it's immediately obvious to the user which section they have landed on. However, when there are many possible link targets, the user might not be certain where they are supposed to look. In these cases, highlighting the link target can be useful.

In the past, we would have needed JavaScript to accomplish this. CSS3 gives us a new pseudo-class to target the current link target, i.e. the element whose ID attribute matches the current URL hash. This pseudo-class is predictably named `:target`.

Let's highlight each heading in our example with a semi-transparent yellow to make it clear to the user where they have landed within the document. We could use the following rule for this:

CODE

```
h1:target, h2:target,  
h3:target, h4:target,  
h5:target, h6:target {  
  background: hsla(65,100%,50%,.5);  
}
```

A fallback for the HSLA color is not needed in this case, because practically any browser that supports `:target` also supports HSLA colors. Old browsers that don't support `:target` are not really a problem because the selector is a usability enhancement and not crucial functionality. Wikipedia uses `:target` when you click on a reference to highlight it within a (usually long) list of references.¹⁶ It really helps you spot the linked reference quickly, doesn't it?

TARGETING ELEMENTS BASED ON THEIR POSITION IN THE DOM

We've all been there. Sometimes we want to target odd-numbered table rows or every third image on the page or the last item in a list or the first four paragraphs of an article.

¹⁶ Try it at smashed.by/wikitarget.

In CSS 2.1 we had only one structural pseudo-class, `:first-child`. CSS3 expands on this, giving us a plethora of new pseudo-classes that solve not only these use cases but many more.

- `nth-child`
- `last-child`
- `nth-last-child`
- `only-child`
- `first-of-type`
- `nth-of-type`
- `last-of-type`
- `nth-last-of-type`
- `only-of-type`

The number of pseudo-classes in this list might seem daunting, but once you understand the possibilities of `:nth-child` and how it works, then understanding the rest and knowing how to use them become very easy because they are merely variations or shortcuts. The ones starting with `nth` introduce a concept that didn't exist in CSS 2.1: parameterized pseudo-classes. Similar to functions, they accept a parameter in parentheses that differentiates their behavior. The syntax of this parameter could be any of the following:

- A number, with `:nth-child(1)` being equivalent to CSS 2.1's `:first-child` pseudo-class. To express `:nth-child(5)` in CSS 2.1, you would have had to write `:first-child + * + * + *`, which is unacceptably verbose, especially for big numbers.
- An expression like $5n$ or $3n+2$, where n represents any number from 0 to infinity. For example, `:nth-child(3n+1)` is equivalent to `:nth-child(1)`, `:nth-child(4)`, `:nth-child(7)`, `:nth-child(10)`, etc., with the list being infinite.
- One of the keywords `odd` or `even`, which is equivalent to $2n+1$ and $2n$, respectively.

For example, to darken the background of every odd-numbered table row, we could write something along the lines of this:

```
tr:nth-child(odd) {  
  background: rgba(0,0,0,.15);  
}
```

This is essentially the zebra-stripe effect we commonly use JavaScript for.

Please note that the difference between `:nth-*` and `:nth-last-*` is merely the direction of numbering: `:nth-child` starts from the first sibling, whereas `:nth-last-child` starts from the last sibling. Therefore, `:last-child` is basically equivalent to `:nth-last-child(1)`, and `:only-child` is equivalent to `:first-child:last-child` because it matches elements with no siblings. Interestingly, we can generalize the `:only-child` pseudo-class, so that when we need to target elements with exactly five siblings, we can use `:first-child:nth-last-child(6)` to target the first one and then use `:first-child:nth-last-child(6) ~ *` for the rest.

The difference between the `*-child` and `*-of-type` pseudo-classes is that the latter maintains a separate count per tag name. For example, `body:first-child` would never match because `body` always has a head sibling, but `body:first-of-type` would always match because we have only one `body` element. This might not be particularly useful for targeting `body`, but it is incredibly useful if we want to target, say, every third image in markup that has a varying number of paragraphs between images, in which case `:nth-child` would render inconsistently because it operates on all siblings, regardless of their type.

WHAT ABOUT OLDER BROWSERS?

Usually the functionality added by these selectors is not crucial, so a Web page will still work fine without it. But if you absolutely need to support legacy browsers, a polyfill can help. The most popular one at the moment is `Selectivizr`.¹⁷

Mix Units Without Problems

Once again, let's return to our example in section on working with images.¹⁸ Suppose we now need to change from static text to a Web form, with the text being inside a `textarea` element, allowing people to edit it. We gave our `textarea` a padding of 1 em, a 1-pixel border and a width of 100% because we wanted it to occupy the full width of the container. You probably see where this is heading—to the dreaded old CSS Percentage Problem™.¹⁹

Learning CSS3

This is just some example content. Don't even bother reading it, you will just waste your time. Why do you keep reading? Do I have to use Lorem Ipsum to stop you from that? Ok, here goes: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Still reading? Oh gosh, you're impossible. I'll stop here to save you from yourself.

Figure 4.17. Our starting point: a style from the previous section.

¹⁷ smashed.by/slctvzr

¹⁸ smashed.by/mixunits

¹⁹ smashed.by/percproblem

Paddings and borders are *added* to that 100%, causing the entire box to be much bigger than 100% and look not so fancy. In the past, we would have had to specify our padding and borders in percentages, too, and specify a width of 100% minus the padding minus the border's width. Luckily, in CSS3 we now have the power to change the way widths are calculated and to do what we have always thought more natural—make the width include padding and borders. The box-sizing property is responsible for this amazing switch. It accepts 3 values:

- **content-box**
The default, which we already know and dislike.
- **padding-box**
With this, padding is included in the width, but the border is not. It doesn't have very good browser support, so avoid it for now.
- **border-box**
Both padding and borders are included in the width.

By applying box-sizing: border-box, our issue is now solved.²⁰ We've saved the best for last: this property is not only supported by every modern browser but also by IE 8!

Transitions

Until fairly recently, websites designed to Web standards were fairly static. If you wanted to add any kind of animation between elements on a page, you had to do it either with Flash or complex JavaScript. Now, if you have the freedom to design to modern browsers, you can achieve these effects using CSS, with the added advantage that they will perform better on mobile devices due to the animation routines being optimized in the browser.

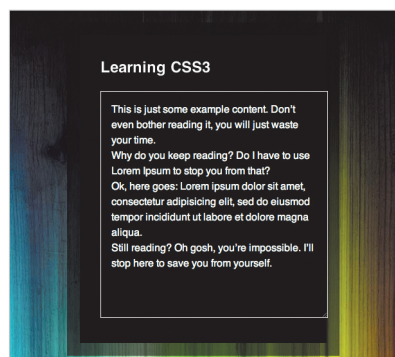


Figure 4.18. The dreaded Old CSS Percentage Problem™.

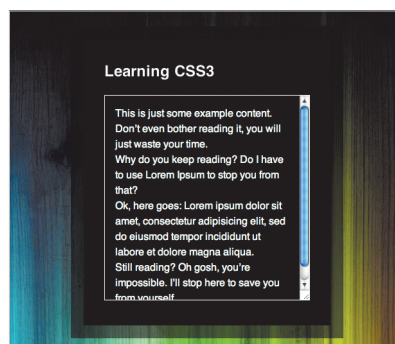


Figure 4.19. Mixing percentages with pixels and ems, using box-sizing.

²⁰ smashed.by/box-sizing

Transitions are one half of the animation capabilities of CSS. These are designed for simple animations of CSS property values between one state and another. Traditionally in CSS, if you changed a property's value, it would switch instantly between the old value and the new one. With CSS transitions, the browser interpolates between the old and new value for the specified duration.

USING TRANSITIONS

To demonstrate how to use transitions, we're going to have a day at the races. Do you remember those horse-racing slot machines when you were a kid? If not, don't worry; the idea is simple: horses will move along a track at varying speeds, and you have to guess which will come in first. This game has been recreated using CSS transitions. You can play along at home by going to smashed.by/trnsxpl. Just hover over the track and watch the horses go!



Figure 4.20. A horse-racing slot machine game created with CSS transitions.

The mark-up for the track is as follows:

```
<div id="track">  
  <h1>The <em>Smashing</em> Derby</h1>
```

```
<ol>
  <li><div></div></li>
  <!-- additional horses -->
</ol>
</div>
```

Each `li` represents a lane, and the `div` within holds the horse. We then transition the width of the `div` on hover over a set duration:

```
#track div {
  width: 3em;
  height: 3em;
  background: url(horse.png) no-repeat right center;
  transition-property: width;
  transition-duration: 6s;
}
#track:hover div {
  width: 40em;
}
```

CODE

Here we are saying that the `width` property should be transitioned over 6 seconds from 3 ems to 40 ems when the track is hovered over. The `transition-property` property defaults to all, so if you do not set it explicitly, it will transition every animatable property that changes.

All transition properties accept a comma-separated list of values, so you can specify multiple properties to transition. If the number of values in properties such as `transition-duration` is fewer than the number of properties to transition, it will match them up similar to what we get with the background properties.

ADJUSTING THE VELOCITY OF THE HORSES

A race wouldn't be much fun if the horses all ran at exactly the same speed and finished at the same time. So, let's adjust the velocity of each horse with the `transition-timing-function` property. All types of timing functions take the same duration to complete, but they will speed up and slow down at different rates, depending on the bezier curve that is specified. If this sounds like too much math, don't worry: you can choose from a set of built-in timing functions. These presets have the following keyword values:

- ease
This is the default.
- linear
Transitions at a constant speed from A to B.
- ease-in
Transitions slowly and then accelerates as it nears point B.
- ease-out
Transitions quickly and then slows down as it nears point B.
- ease-in-out
Transitions quickly until the halfway point and then slows as it nears point B.

Each of these functions is used in this order for the horses in the demo:

CODE

```
li:nth-of-type(1) div { transition-timing-function: ease; }  
li:nth-of-type(2) div {  
  transition-timing-function: linear;  
}  
li:nth-of-type(3) div {  
  transition-timing-function: ease-in;  
}  
li:nth-of-type(4) div {  
  transition-timing-function: ease-out;  
}  
li:nth-of-type(5) div {  
  transition-timing-function: ease-in-out;  
}
```

Watching the demo, you'll see that horse 1 (ease) flies out into the lead, a certain winner. But oh no! It runs out of steam in the final third. And in a photo finish, all five cross the line at exactly the same time!

If you are not happy with the presets, you can define your own using the cubic-bezier function. To define a cubic bezier, you have to specify the x and y coordinates for the curve's control points. The start is always anchored at 0,0, and the end is anchored at 1,1, so these do not have to be defined. The ease-in transition function would be defined as the following using the cubic-bezier function on the next page:

```
transition-timing-function: cubic-bezier(0.42, 0, 1, 1);
```

A number of tools enable you to visually specify and tweak the cubic-bezier function, such as Lea Verou’s Cubic Bezier preview tool.²¹

HOLD YOUR HORSES

You might not want the transition to happen as soon as the value changes. You can control this with transition-delay property. This works in exactly the same way as the transition-duration property:

```
#track div {  
  ...  
  transition-property: width;  
  transition-duration: 6s;  
  transition-delay: 1s;  
}
```

CODE

With all of these properties under your belt, you will have elements flying across the page in no time!

Conclusion

While designers and developers keep coming up with creative technical solutions, one thing is clear: CSS3 is here to stay. Not only does it greatly reduce the time spent converting designs from visuals to code, but it also helps us produce flexible styling that adapts to all kinds of circumstances: code changes, different content and different kinds of devices.

As browsers catch up to all of these CSS3 goodies, the CSS Working Group is already planning the next iteration of CSS, commonly referred to as CSS4. This will bring highly anticipated features such as parent selectors, variables, nesting and conical gradients, most of which are still under debate.

Exciting as CSS3 and CSS4 might sound as buzzwords, keep in mind that CSS as a whole is basically a living standard. As every member of the CSS Working Group can attest, there is no such thing as “CSS3” or “CSS4” in Web standards. In fact, there is no

²¹ smashed.by/cbcbz

global version of CSS anymore. After CSS 2.1, CSS was modularized, and each module now has its own version. And some Level 1 modules might actually have come out later than some Level 4 modules. But we don't need buzzwords to get excited about everything that's coming out in the world of CSS, do we?



About the Author

David Storey holds a Master's degree in Internet and Distributed Systems from the University of Durham, UK. He is an evangelist for open Web standards and a member of the CSS Working Group. David currently lives in Mountain View, California, and works for Motorola Mobility. Previously, he worked at Opera, where he founded the Developer Relations team and was the Product Manager for Opera Dragonfly. He also worked for CERN and was an author for CSS3.info during its golden period. His specialties are HTML, CSS, SVG and JavaScript.



About the Author

Lea Verou has a long-standing passion for open Web standards, and has been often called a "CSS guru". She loves researching new ways to take advantage of modern Web technologies and shares her findings through her blog, lea.verou.me. Lea also makes popular tools and libraries that help Web developers learn and use these standards. She speaks at a number of well-known international Web development conferences and writes for leading industry publications. Lea also co-organized and occasionally lectures the Web development course at the Athens University of Economics and Business.



About the Reviewer

Tab Atkins Jr. wears many hats. He works for Google on the Chrome browser as a Web standards hacker, although his code contribution is fairly minimal. Tab works mainly with HTML and CSS specifications. He's also a member of the CSS Working Group and contributes to several other working groups in the W3C.



JavaScript Rediscovered: Tricks to Replace Complex jQuery

Written by Christian Heilmann

Reviewed by Paul Irish

WHEN JQUERY APPEARED, it was an utter revelation. Its first and foremost job was to make browsers behave. Until then, support for basic features such as accessing parts of the document, responding to user interaction and even styling elements varied vastly from browser to browser.

jQuery replaced the DOM specification, which defined accessing content on the page with `getElementById()` and `getElementsByTagName()`, with a simpler approach: using CSS selectors. This opened a whole new world of development to designers who knew their CSS but suffered the daily frustration of browsers that did not support complex selectors. In other words, jQuery enabled us to use the CSS of tomorrow, today. That and the chaining approach of jQuery (which meant much less code to write) were the reasons for its quick rise to success.

Fast-forward some years to now (and by the time you hold this book in your hands, we will be even further along). We have HTML5, we have CSS3 support, and we have many more things to play with in the browsers that we and our users have installed. Yes, the scourge of IE 6 is still upon us, and IE 8 will be with us for quite some time, too, but all in all our position is much better. Libraries such as jQuery still offer the main benefit of fixing things in older browsers, but they also cause discontent, and the reason is because we are overusing them.

As a community, we have become dependent on jQuery. This is understandable, but not good. jQuery is written in JavaScript but is not the same as JavaScript; it is also not built natively into the browser. With the rise of the mobile Web, quite many people are turning away from jQuery because it is too slow and heavy for those fancy devices in our pockets. Finding good JavaScript developers is hard: for every position you advertise for a JavaScript developer, you will get about 20 CVs from people who have never written anything but jQuery. This dilutes our craft.

Let's look, then, at some of the things that browsers offer us these days that we can use to write incredibly small and usable solutions without resorting to jQuery. A lot of these things will also help us write cleaner and faster jQuery code. Because the jQuery library abstracts away a lot of the issues we face as developers, it is all too easy to write code that looks simple but deep down results in a lot of looping and comparing. And this is the reason for slow websites.

THE POWER OF MIXING AND MATCHING

On the Web, the main trick to developing code that is concise, stable, efficient and easy to maintain is separation and delegation. With jQuery, we have forgotten much of this. Excessive length makes the CSS selectors in our scripts break when the HTML chang-

es. Overusing class selectors forces the HTML developer to add a lot of superfluous classes in order to be able to use a certain plugin. Creating and styling HTML in jQuery makes it harder to find where a background color comes from when you are asked to change it. To write code that is easy to maintain, remind yourself of which technologies are good at doing what. This has shifted quite a bit since the inception of jQuery:

- **HTML is the structure and base on which to build.**

Your HTML should make sense and provide basic functionality: links that point to other websites, buttons that send forms to a script to reload the page, and elements that structure the content. The reason is that, when everything breaks, the browser is left with the HTML. If the HTML makes sense, you win. If there is a button that does not do anything, though, you will annoy users.

- **CSS defines everything interactive and visual.**

We have gone beyond fonts and color to allow for animation and transitions. Media queries enable us to define different layouts for different devices. Using content generation, we can create elements to achieve a certain visual effect without soiling the HTML with `div`s and `span`s.

- **JavaScript brings extra functionality.**

With script loading and AJAX, you can load content on demand. You can add event listeners to enable elements to be touchable and clickable, to read the orientation of a device, and to find out how far a user has scrolled or where the mouse pointer is.

The trick is to embrace these new opportunities and to not give old browsers effects that they will choke on. Those old methods won't help anyone in the long run. Yes, you could animate a menu in IE 6, but why bother writing this “nice to have” functionality when it is built into other browsers?

FEATURES WE CAN USE NOW

Let's look at some features in browsers that we can use now. To get the latest information on which browsers support them, check out the great resource *When Can I Use*,¹ which is constantly updated. Anything I tell you about browser support now would be outdated in a few weeks' time. This is the speed we have to keep up with these days.

¹ <http://smashed.by/caniuse>

CASHING IN THE \$: QUERYSELECTOR AND QUERYSELECTORALL

Now that we have learned from jQuery's success, browsers have a way to target elements on the page using CSS selectors. The `querySelector` method targets a single element, and `querySelectorAll` targets a list of matched elements. The syntax of the selector is similar to CSS. So, `document.querySelector('#content p')` would target the first paragraph in the element with the ID `content`; `document.querySelector('nav li:last-child')` would target the last list item in the first `nav` element; and `document.querySelectorAll('p')` would target all paragraphs in the document. As simple as that.

KEEPING IT CLASSY: CLASSLIST

A big use case of jQuery is accessing many elements at once and changing their styles by manipulating their `styles` collection with the `css()` method. This is handy, but also annoying because you are putting styling information in the JavaScript. Much simpler would be to add a class to the element in question and leave the rest to CSS. When you think about it, we often repeat CSS selectors in jQuery and in our style sheets. In many cases, we had to do this because browsers did not support CSS3 selectors—now they do.

Being able to test for classes in HTML elements and dynamically add and remove them is incredibly powerful. In JavaScript, we now have a `classList` property in HTML elements that contains a collection of applied CSS classes. In the past, this was done with `className`, which contained a simple string, and it was up to us to find other strings in it and to add and remove substrings from it. With `classList`, we have methods for that. We can use `element.classList.add(name)` to add a class, `element.classList.remove(name)` to remove it, `element.classList.contains(name)` to check whether a class is applied, and `element.classList.toggle(name)` to toggle a class on and off. Later in this chapter, we will see just how powerful this is; we can avoid a lot of looping simply by adding a class to the parent element.

KEEPING IT SMOOTH: CSS TRANSITIONS

Animation in jQuery is easy indeed, and it looks very smooth. The reason is that jQuery includes the easing equations,² and now we have those in CSS, too. So, if you want to expand a heading and change its background color from light-green to orange, you can use the CSS snippet on the following page.³

² smashed.by/easing

³ Check out this example in action: smashed.by/transition.

CODE

```
h1 {  
  background: #c0ffee;  
  line-height: 1em;  
  padding: 0.5em 1em;  
  -webkit-transition: 1s;  
  -moz-transition: 1s;  
  -ms-transition: 1s;  
  -o-transition: 1s;  
  transition: 1s;  
}  
  
h1:hover, h1:focus {  
  background: goldenrod;  
  line-height: 3em;  
}
```

The best thing about this is that, by defining a time for the transition but not defining which properties to change, we can make the browser move smoothly from one state to another without having to know what future maintainers might want to change. In this case, we are altering the background color and line height, but that could easily change in the future. With jQuery, this would have meant a JavaScript rewrite. Annoying as it may be that we have to repeat the transition information for all of the browser prefixes, we just have to live with this for now.

Another benefit is that these transitions are hardware-accelerated, which means they will run more smoothly and use less battery on mobile devices. Not all browsers do this yet (some need a 3-D transformation of 0 on the z-axis as a hack), but this will surely become a standard. You can read more about CSS3 transitions in the chapter 4 by Lea Verou and David Storey.

CREATING FLUFF: CSS-GENERATED CONTENT

Sometimes designers need more HTML elements to add some styling (a situation we ran into in the past with, say, rounded corners). Most of the time we use jQuery for that. With CSS-generated content, that is no longer needed. We can generate elements in CSS and style them all at once. Say you want all external links to have a red arrow behind them:

CODE

```
a[href^="http"]:after {  
  content: ' ↗';  
  color: #c00;  
}
```

This simple CSS code does the trick. We are specifying that for each link with an `href` attribute value that begins with `http`, a text node—in this case, a red-colored arrow—should be added to the link’s content.

DELEGATING EVENTS: REDUCING MANY TO ONE

One big feature of jQuery is that you can quickly iterate over many elements to change things in them or assign event handlers. But this is not really needed. *Event delegation*⁴ is an incredibly powerful tool when you are building Web interfaces. Essentially, instead of assigning event listeners to every element within a main container element, you assign one event handler to the main container and allow the browser to bubble the events up.

This has a few advantages. For starters, you end up assigning many fewer event handlers in the document, which is always good for memory consumption. More interesting, though, is that you keep the event handling independent of the number of elements it applies to. For example, if you have a to-do list and add new items to it, you would not have to reassign the handlers at all. jQuery borrowed this concept when it added the `live` event handler. However, many jQuery solutions will add `live` handlers to ID selectors without any children—and by definition, an ID may appear only once in a document and, thus, does not need any delegation.

USING THESE TECHNIQUES IN A FEW EXAMPLES

Let’s use some of these techniques in a few examples. We will start with a to-do list that uses event delegation and generated content, then progress to a brochure website that uses transitions, and then go wild at the end by using HTML5 canvas to create thumbnails in the browser.

⁴ smashed.by/sandbox

EXAMPLE 1: A SIMPLE TO-DO LIST⁵

To create a to-do list for all of the browsers out there, we will need a server-side solution to grab the list items, store them in a database and display them as a list in the browser. We won't do that here; instead, we will only use a client-side solution, including for storage of the data. But with a real product, you should have a server fallback.

Using the browser technology of today, we can do this in a few lines of code, without any looping over elements. The HTML markup is pretty simple:

```
<ul id="todolist"></ul>
<form action="#" method="post">
  <div>
    <label for="newitem">Add item</label>
    <input type="text" name="newitem" id="newitem"
      placeholder="new item">
    <input type="submit" value="Add">
  </div>
</form>
```

CODE

The JavaScript code is nothing special, either:

```
var todo = document.querySelector( '#todolist' ),
    form = document.querySelector( 'form' ),
    field = document.querySelector( '#newitem' );

form.addEventListener( 'submit', function( ev ) {
  var text = field.value;
  if ( text !== '' ) {
    todo.innerHTML += '<li>' + text + '</li>';
    field.value = '';
    field.focus();
  }
  ev.preventDefault();
}, false);
```

CODE

⁵ Check out this example in action: smashed.by/todolist.

CODE

```
todo.addEventListener( 'click', function( ev ) {  
    var t = ev.target;  
    if ( t.tagName === 'LI' ) {  
        t.parentNode.removeChild( t );  
    };  
    ev.preventDefault();  
}, false);
```

In the code above, we start by grabbing the elements in the document that we want, using `querySelector`. In this case, we will grab the list that we want to add elements to, the form where new elements come from, and the field in which the new entry has been inputted.

We then add an event listener to the form that reads out the value of the field and checks whether a value was entered when the form was submitted. (This means that the user can add new items by hitting “Enter,” in addition to clicking the button. Sadly, too many jQuery solutions use click handlers on the button instead.) If there is some content, we add a new item to the list using `innerHTML`. We then delete the current text in the form field and put the cursor focus on it (to make it easy to add another item).

To enable the user to remove completed items from the list, we add a click handler to the list, read out the `target` of the event, and compare its `tagName` to the `li` element. If the target is a list element, we remove it using the old-school DOM `removeChild()` method. This is what we have to do to create a to-do list with unlimited items using event delegation. Nothing more, nothing less.

ADVANCED CSS SELECTORS AND GENERATED CONTENT FOR STYLING

Now we want to give the list items alternating colors. We also want a check box to appear on hover, indicating that clicking on the list item will mark it as completed. To do this, we need neither JavaScript nor images:

CODE

```
#todolist li {  
    background: #eee;  
    min-height: 20px;  
    position: relative;  
}
```

```
#todolist li:nth-child(2n) {  
  background: #ccc;  
}  
  
#todolist li:hover:after {  
  content: '✓';  
  color: #060;  
  position: absolute;  
  right: 5px;  
}
```

CODE

The `nth-child(2n)` selector tells the browser to color every other row a darker gray while leaving the other rows a lighter shade. To show a check mark when the user hovers over a list item, we use the `:after` selector and create a UTF-8 check mark. Because each list item is relatively positioned, any absolutely positioned element will fall inside it, and thus the correct value will show the green check mark within the box when the user hovers over the list item.

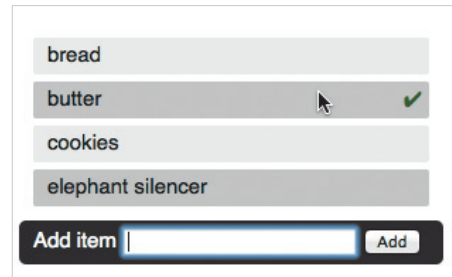


Figure 5.1. Our to-do list with check marks.

ADDING TWO-STEP DELETION FOR LIST ITEMS

What if we wanted items in our list not only to be marked as completed but to be deleted on a second click? Simple: we just add another state and use classes. When the user clicks on an item the first time, a class named `done` is added, and when they click it a second time, the item is removed. All we need to do is change the event handler:

```
todo.addEventListener( 'click', function( ev ) {  
  var t = ev.target;  
  if ( t.tagName === 'LI' ) {  
    if ( t.classList.contains( 'done' ) ) {  
      t.parentNode.removeChild( t );  
    }  
  }  
}
```

CODE

CODE

```

    } else {
        t.classList.add( 'done' );
    }
};
ev.preventDefault();
}, false);

```

If the clicked element is not assigned the class `done`, then we add the class to the element. If it is assigned the class, then we remove the element. This takes care of the functionality, but it also gives us an extra class to play with in our CSS. We can use it to add a prompt for deletion (an “x”) that appears when you hover over a completed item:

CODE

```

#todolist li:hover:after,
#todolist li.done:after {
    content: '✓';
    color: #060;
    position: absolute;
    right: 5px;
}

#todolist li.done:hover:after {
    content: 'x';
    font-weight: bold;
    color: #c00;
    position: absolute;
    right: 5px;
}

```

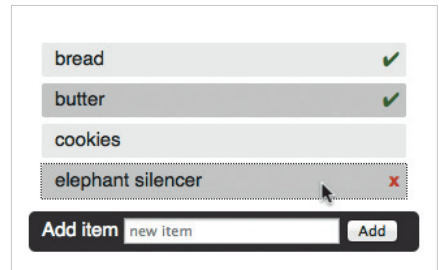


Figure 5.2. To-do list with check marks and deletion icons in the second state.

SELF-VALIDATING FORMS

As you may remember, we are checking for content in the field before generating a new list item. Right now this is done in JavaScript, but if we stay true to the environment we are working in, then we don't need that. Adding a required attribute in the HTML makes sure the browser validates the form field before sending the form:

```
<ul id="todolist"></ul>
<form action="#" method="post">
  <div>
    <label for="newitem">Add item</label>
    <input type="text" name="newitem" id="newitem"
      placeholder="new item" required>
    <input type="submit" value="Add">
  </div>
</form>
```

CODE

If the user tries to submit the form without entering any information, the submit handler never gets fired. This means we can shorten the JavaScript code by yet another line:

```
form.addEventListener( 'submit', function( ev ) {
  todo.innerHTML += '<li>' + field.value + '</li>';
  field.value = '';
  field.focus();
  ev.preventDefault();
}, false);
```

CODE

Falsely submitted form fields will now automatically be flagged by the browser—something we had to do ourselves in the past (see the image on the right). If the browser does not support the required attribute, the form gets submitted, which is what happens when a user (or, in many cases, a hacker) has turned off JavaScript anyway. Testing in JavaScript is a measure of convenience, not security. You will have to check server-side in any case.



Figure 5.3. Firefox showing a form-field error and highlighting the field.

STORING THE STATE OF THE LIST

The normal procedure now would be to find a way to store the information in a database and ask the user for credentials to make them storable. Using HTML5—and assuming that this app will be used on one computer and not have to be synced across multiple devices—we can use Local Storage to simply take a snapshot of the list every time it changes.

Session and Local Storage are not HTML5 per se, but rather their own standard. For large data sets to be stored in the client, there is IndexedDB and Web SQL. However, Local Storage is incredibly easy to use and more than enough for our needs.

To store the list's state and load it when the user comes back to the page, all we have to do is write two functions, `storestate()` and `retrievestate()`:

CODE

```
function storestate() {
    localStorage.todolist = todo.innerHTML;
};

function retrievestate() {
    if ( localStorage.todolist ) {
        todo.innerHTML = localStorage.todolist;
    }
};
```

Then we need to call these function whenever the list is changed:

CODE

```
form.addEventListener( 'submit', function( ev ) {
    todo.innerHTML += '<li>' + field.value + '</li>';
    field.value = '';
    field.focus();
    storestate();
    ev.preventDefault();
}, false);
```

```
todo.addEventListener( 'click', function( ev ) {  
  var t = ev.target;  
  if ( t.tagName === 'LI' ) {  
    if ( t.classList.contains( 'done' ) ) {  
      t.parentNode.removeChild( t );  
    } else {  
      t.classList.add( 'done' );  
    }  
    storestate();  
  };  
  ev.preventDefault();  
}, false);
```

We retrieve the data when the page is loaded again:

```
document.addEventListener( 'DOMContentLoaded', retrievestate,  
false );
```

That's it! Caching whole interfaces in Local Storage may seem like a dirty hack, but nothing is unsavory about it. Because HTML is not too verbose a data format and we have 5 MB of storage across browsers to work with, this is a simple solution to a common problem.

EXAMPLE 2: ANIMATED PAGE ELEMENTS USING CSS3⁶

Let's take another quick look at the trick of assigning classes and event delegation. This time, we will play with CSS and JavaScript to animate parts of a website without any library or animation tool.

Take the following website, which I whipped up for my favourite café (where I am writing this right now). Without JavaScript, it would look something like the image on the next page:

⁶ Check out this example in action: smashed.by/cafevintage.



Figure 5.4. The plain-vanilla Cafe Vintage website, showing all of the sections one after the other.

We have some headings, some images and accompanying descriptions on a very long page—that’s all. The HTML is very simple: a navigation menu pointing at targets in the document:

CODE

```
<header>
  <h1>Cafe Vintage</h1>
  <p>88 Mountgrove Road, London N5 2LT, England</p>
  <nav>
    <ul>
      <li><a href="#cafe">The Cafe</a></li>
      <li><a href="#fashion">Fashion</a></li>
      <li><a href="#food">Food</a></li>
      <li><a href="#gifts">Gifts</a></li>
    </ul>
  </nav>
</header>
<section>
  <article id="cafe">[...]</article>
  <article id="fashion">[...]</article>
  <article id="food">[...]</article>
  <article id="gifts">[...]</article>
</section>
<aside>
  <p>Opening hours:</p>
  [...]
</aside>
<footer>
  <p>&copy; 2012 Cafe Vintage and Chris Heilmann</p>
</footer>
```

This is old-school HTML with a few new semantic elements introduced by Ben Schwarz in Chapter 3. It works in every browser, and there is a logical connection between the navigation and the main content: the IDs.

When JavaScript is available, the page shows one section at a time. There is also an animation to transition between the sections: the last page moves up and the new one drops down, and the descriptions animate from right to left (see Figures 5.5-5.7.):



Figures 5.5-5.7. When JavaScript is available, the articles on the page animate when the user clicks a navigation item.

To make this happen, all we need to do is add and remove some classes. The rest happens in the CSS. Here is the logic we are following:

- Apply a class named `js` to the body, and hide all `article` elements in the CSS with `.js article {...}` (in this case, positioning them absolutely and moving them off screen).
- Add a class to the user's chosen article, named `current`, which is overridden in the CSS with `.js article.current {...}`.
- Add a class named `current` to the menu link that corresponds to the currently visible article, to show the user where they are in the navigation.

First, we add a class named `js` to the body of the document. This enables us to define styles for the non-JavaScript and JavaScript versions. Then we get the elements that we need—in this case, we need the first `article` element and the first link, because they will be enabled first by default.

```
document.body.classList.add( 'js' );
var nav = document.querySelector( 'nav' ),
    article = document.querySelector( 'article' ),
    link = document.querySelector( 'nav a' );
```

CODE

We set their classes to `current`:

```
link.classList.add( 'current' );
article.classList.add( 'current' );
```

The rest of the functionality is event delegation:

```
nav.addEventListener( 'click', function (ev) {
  var t = ev.target;
  if ( t.tagName === 'A' ) {
    article.classList.remove( 'current' );
    link.classList.remove( 'current' );
    article = document.querySelector( t.getAttribute( 'href' ) );
    link = t;
  }
});
```

```

        article.classList.add( 'current' );
        link.classList.add( 'current' );
    }
}, false);

```

Now we assign a click-event listener to the navigation, check that a link was clicked and remove the `current` class from the link and from the article that was being shown. We then reach the new article to be shown by reaching the `href` attribute of the link that was clicked and then assign the `current` class to the new elements.

That is almost everything we need to do. One other use case to account for, though, is of a user coming to the website via a link containing a hash (that is, arriving on an article other than the default one); we would have to show the correct article in this case. For this, we can use two selectors:

CODE

```

if ( document.location.hash ) {
    var cleanhash = document.location.hash.replace( /^#/, '' );
    article = document.querySelector( document.location.hash );
    link = document.querySelector( 'nav a[href$=' + cleanhash + ']' );
}

```

The CSS selector tests whether the link ends in the string that we give it. The animation is done in CSS using transitions:

```

section {
    overflow: hidden;
    min-height: 340px;
    position: relative;
}

article {
    position: relative;
    height: 350px;
}

```

```
body.js article {
  width: 700px;
  position: absolute;
  top: -700px;
  -webkit-transition: 0.8s;
  -moz-transition: 0.8s;
  -ms-transition: 0.8s;
  -o-transition: 0.8s;
  transition: 0.8s;
}

body.js article.current {
  position: absolute;
  top: 0;
}
```

Here we are modifying the articles. We are just telling the browser to position them relatively in the section when no JavaScript is available, and to position them absolutely 700 pixels above the top of the container when JavaScript is available. Because `overflow: hidden` is applied to the section, they will never show up.

When an article is the current one, the value of `top` is changed to 0, which moves the article down from the top.

The paragraphs work the same way:

```
article p {
  position: absolute;
  left: 320px;
  width: 370px;
}

.js article p {
  left: 900px;
  opacity: 0;
  -webkit-transition: 1s ease 0.7s;
  -moz-transition: 1s ease 0.7s;
```

CODE

```

        -ms-transition: 1s ease 0.7s;
        -o-transition: 1s ease 0.7s;
        transition: 1s ease 0.7s;
    }

    .js article.current p {
        position: absolute;
        left: 320px;
        width: 370px;
        opacity: 1;
    }

```

In this instance, we create a 1-second transition with 0.7 seconds delay. Notice that we are animating both `left` and `opacity` in one fell swoop, without having to do anything in JavaScript.

EXAMPLE 3: THUMBNAIL GENERATION IN THE BROWSER⁷

To wrap this up, let's go wild! One of the best things about HTML5 is the `canvas` element. It might seem merely like an element to be painted on (and seemingly even

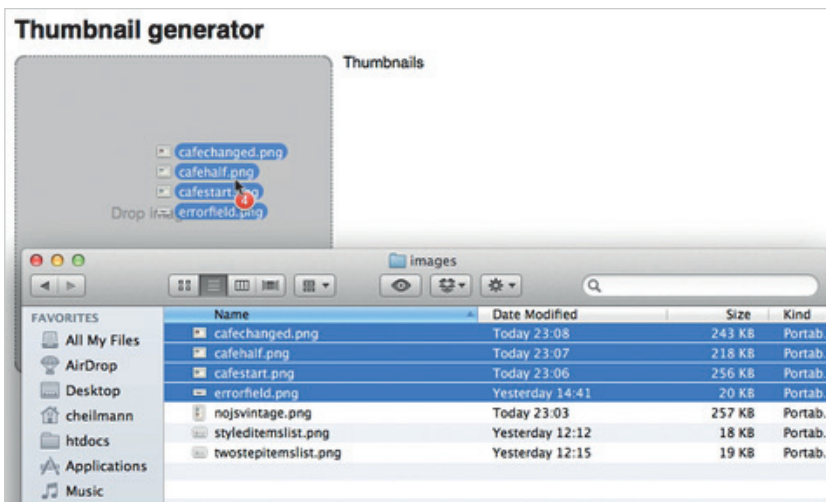


Figure 5.8. Drag and drop the images by using `canvas` and `FileReader`.

⁷ Check out this example in action: smashed.by/thumbnails.

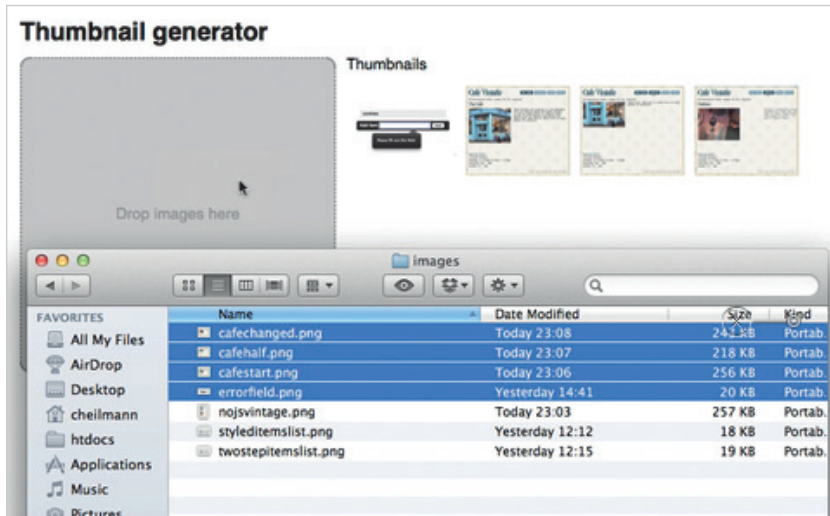


Figure 5.9. Thumbnails generated.

pointless without JavaScript), but it is a powerful tool to read and manipulate image and video data, too. Together with the FileReader and Drag and Drop functionality of modern browsers, we can really go to town. Why don't we generate thumbnails in the browser?

Most content management and blog systems have file uploaders to create thumbnails from images, so let's use them as our fallback (again, we won't go into the server code here):

```
<section>
  <form>
    <label for="upload">Pick image</label>
    <input type="file" id="upload" name="upload">
    <input type="submit" value="Make it so!">
  </form>
</section>
<output><p>Thumbnails</p></output>
```

CODE

Let's add various styles. And, if the browser supports it, we will replace the form with a message inviting the user to drag images onto the section:

CODE

```

if ( window.FileReader && ( ( 'draggable' in document.body ) ||
( 'ondragstart' in document.body && 'ondrop' in document.body ) ) ) {

    var s = document.querySelector( 'section' ),
        o = document.querySelector( 'output' ),
        c = document.createElement( 'canvas' ),
        cx = c.getContext( '2d' ),
        thumbsize = 100;
    c.width = c.height = thumbsize;
    document.body.classList.add( 'dragdrop' );
    s.innerHTML = 'Drop images here';

```

Here we are testing whether the browser supports FileReader and Drag and Drop (annoyingly, Safari still does not support the former, nor Opera the latter). If the browser does support them, then we grab the elements that we need. We create a canvas element and store its 2-D context in *cx*. We then define the thumbnails' dimensions and resize the canvas accordingly. Finally, we add a *dragdrop* class to allow for styling, and replace the form with a message prompting users to drag images over.

Normally, when you drag an image onto a browser, the browser simply replaces the current document with the image. We want to avoid this, which is why we prevent the default behavior on *dragover*. We have also added a class to give the user an indication that something is happening:

CODE

```

s.addEventListener( 'dragover', function ( evt ) {
    s.classList.add( 'active' );
    evt.preventDefault();
}, false );

```

We remove the class if the drop is cancelled:

```

s.addEventListener( 'dragleave', function ( evt ) {
    s.classList.remove( 'active' );
    evt.preventDefault();
}, false );

```

All of the other functionality happens in the drop handler:

```
s.addEventListener( 'drop', function (ev) {
    s.classList.remove( 'active' );
    var files = ev.dataTransfer.files;
    if ( files.length > 0 ) {
        var i = files.length;
        while ( i-- ) {
            var file = files[ i ];
            if ( file.type.indexOf( 'image' ) !== -1 ) {
                createthumb( file );
            }
        }
        ev.preventDefault();
    }, false );
```

CODE

The first thing we have done here is remove the active class, because we are done with the drop. The drop event gives us a `dataTransfer` object that contains the files that have been dropped. We check if at least one file was in the drop and then start iterating over all of them. (The `while{}` loop is a fancy way of doing a `for{}` loop without the need to cache the length or use a second iterator variable.) We test whether the current file is an image by testing its type, and then initiate and send it to the `createthumb()` function. Finally, we prevent the default behavior of dropping in an image, and then we are done.

```
function createthumb( file ) {
    var reader = new FileReader();
    reader.readAsDataURL( file );
    reader.onload = function ( ev ) {
        var img = new Image();
        img.src = ev.target.result;
        img.onload = function() {
            cx.clearRect( 0, 0, thumbsize, thumbsize );
```

CODE


```

        var thumbgeometry = resize( this.width, this.height,
                                    thumbsize, thumbsize );
        cx.drawImage( img, thumbgeometry.x, thumbgeometry.y,
                      thumbgeometry.w, thumbgeometry.h );
        var thumb = new Image();
        thumb.src = c.toDataURL();
        o.appendChild( thumb );
    };
}

```

The `createthumb()` function initiates a new `FileReader` and reads the image as a string of data. If the reader successfully loads the file, then we create a new image in the browser and set its `src` to the result of the `FileReader` transaction.

When the image has successfully loaded, we clear the canvas element using the `clearRect()` method. This is required, otherwise we would create thumbnails that add on to each other every time the function is called. Then we get the size of the thumbnail that we want to generate from the `resize()` function and call the `drawImage` method of the canvas. This method takes five parameters: the image to get the pixel information from, the coordinates of the top-left corner to draw the image from, and the width and height. We then create a new image, grab the pixel content of the canvas with the `toDataURL()` method, and add the new image to the outputted element.

CODE

```

function resize( imagewidth, imageheight, thumbwidth, thumbheight )
{
    var w = 0, h = 0, x = 0, y = 0,
        widthratio = imagewidth / thumbwidth,
        heightratio = imageheight / thumbheight,
        maxratio = Math.max( widthratio, heightratio );
    if ( maxratio > 1 ) {
        w = imagewidth / maxratio;
        h = imageheight / maxratio;
    } else {
        w = imagewidth;
        h = imageheight;
    }
}

```

```
x = ( thumbwidth - w ) / 2;  
y = ( thumbheight - h ) / 2;  
return { w:w, h:h, x:x, y:y };  
};
```

The `resize` function is a mathematical helper to resize an image of a certain width and height to perfectly fit into a smaller one. Nothing magical here—just practical for this case. And with that, we have thumbnails in the browser!

Summary

I hope I was able to convince you that a lot of great stuff is native to browsers these days. Of course, not all browsers support these features yet, but at least all of the vendors are working together on the standards, and we are not in the situation of the first browsers war, when innovation happened in the dark. By mixing and matching the different technologies of the Web (HTML, CSS, JavaScript), we can produce quite an amazing amount of interaction in a few lines of code. All we need to do is use what browsers are giving us.

THE RIGHT TECHNOLOGY FOR THE JOB

The interplay of CSS transitions, transformations, animations and JavaScript is powerful and something we should be using much more. Right now there seems to be a battle between people who do everything with jQuery or JavaScript and those who use CSS exclusively. This does not help our users, and it keeps us from writing concise, effective solutions. A good Web developer plays all technologies to their strengths and is agnostic about them. There is nothing clever about using one technology for everything and then leaving the product to work in only one browser or on one device.

If you need all of your features to be supported in old browsers, use a library such as jQuery. You can also find patches for old browsers in the form of polyfills. But in general, let's stop trying to make outdated technology support interactions that are meant for newer technology. Just because IE 6 does not transition smoothly from one state to another, no one will be blocked from using the product—and that is the most important feature of a good Web product.

EVERYTHING COUNTS IN LARGE AMOUNTS

As Web developers, we always complain that technology is falling behind our needs: people are using browsers that are far too outdated, and the browsers are not delivering what we need. The main reason why people have outdated browsers is that, in the past, developers like us built only for those browsers because they were the state of the art and those developers assumed that nothing better would come along.

Browser vendors do not add every feature that developers want to see because new technology is not being implemented widely enough. Complaining that something does not work is not enough. Browser vendors need this technology to be used in the wild and to get feedback on how it performs. If the only feedback they get is, “Why do you not support feature X,” they may always answer, “Because no one uses it.”

The biggest example of this is the new semantic elements of HTML5. We cannot complain about the lack of support for the outline algorithm in browsers if we do not use the right elements and then give browser vendors feedback on what works and what fails. Every browser vendor out there has feedback mechanisms in place. It is up to us as developers to give them real-world examples to fix, rather than wait for the perfect implementation.

EMBRACING THE FUTURE AND KEEPING IT SAFE AT ONCE

We should embrace new technology and use it whenever we can. Let’s not think that we have to wait until a certain technology is stable and supported by all browsers before we use it. If we do not test in the wild the technology being defined by the working groups of the WHATWG and W3C, we will never get anywhere.

And yet, we should not force technology on people. A message like, “Upgrade to browser X in order to see this content,” is incredibly frustrating if the user does not have permission to download a new browser onto their machine or they are on a slow connection. Instead of building rocket packs, let’s build escalators, because escalators enable people to reach higher levels conveniently, but they also work as stairs when there is an electricity outage or mechanical failure.



About the Author

Christian Heilmann (1975) was born in Schweinfurt, Bavaria, and has a diploma in German, English, history and astronomy. His motto is, “Start something and play with it; if you don’t want to play with it, stop doing it.” Christian currently lives in North London, a mixing pot of people from many cool places. He works to bring technology to people and people to technology, and when he’s not busy working, films are his diversion of choice. Blue and green are his favorite colors, and since he’s rarely at home, his only pets are a lot of rubber ducks. Christian’s message to readers is to stay hungry and stay inquisitive; something new is always around the corner.



About the Reviewer

Paul Irish (1982) was born in Pittsfield, Massachusetts, and graduated from Worcester Polytechnic Institute with a BS in technical communication. He has been a professional front-end developer for seven years and in the last two years has been teaching other developers full time how to make the Web more amazing.

Paul lives in a studio apartment in the Mission district in San Francisco, and in his free time he loves to listen to eclectic music and go to pre-Prohibition cocktails. The most important lesson he has learned in his career is to inject whimsy into all of the hard work. As personal advice to readers, Paul recommends, “Publish what you learn.” These are the words of an expert, so it’s worth a try.



Techniques for Building Better User Experiences

Written by Dmitry Fadeyev

Reviewed by Joshua Porter

THE DESIGN AND USABILITY of websites and apps are getting better. As more businesses and software move online, competition heats up, and so companies start seeking every competitive edge to stay ahead. If you are solving a problem that has not been solved yet, you can get away with a scrappy website; but when you're fighting off twenty competitors, the usability and user experience of your product begins to matter a whole lot more.

User Experience (UX) is a hot buzzword these days, so much so that there are full-time job positions that include this word in the title. It's not a word that I like using because its meaning isn't particularly clear; nevertheless, it's popular because what it does stand for is very important. Simply put, UX means good design. Design not in the sense of superficial eye candy, but in the sense of how everything fits together, how the product works and how well it satisfies your users' expectations.

The aim of design is not to decorate but to solve problems: whether that means getting more sign-ups, inviting users to post more content or making an interface easier and faster to use—this is the sort of design that will ultimately end up delivering a great user experience. Looks matter, too, of course, and they will directly influence the experience of using your product, but it's important not to surrender all your energy into designing the prettiest interface while leaving the rest—things like copywriting, flow, content and usability—without the due attention they deserve.

This chapter will arm you with new and powerful UX techniques that you can apply to your own products and websites to get an edge over your competition by delivering an experience your users will truly love. The chapter is split into four sections. We'll start by looking at how you can improve the sign-up process and forms, follow on with some newer and more experimental techniques, cover ideas on how you can improve customer service and finally discuss some design techniques you may wish to avoid.

“If we want users to like our software, we should design it to behave like a likeable person.”

— Alan Cooper

Improving Sign-Ups and Forms

It is only when real people begin using your product will you know whether or not you're on the right track with your solution and your feature set. Matt Mullenweg, the founding developer of WordPress, makes a beautiful analogy between usage and oxygen: "You can never fully anticipate how an audience is going to react to something you've created until it's out there. That means every moment you're working on something without it being in the public it's actually dying, deprived of the oxygen of the real world." This section features a collection of techniques to help you gather initial product interest and build a solid sign-up process once you've launched.

"I never design a building before I've seen the site and met the people who will be using it."

— Frank Lloyd Wright



Figure 6.1. SquidChef's "Coming soon" page.


HIDDEN SURVEY

The first thing any great product needs is users, and you can start getting them even if you haven't yet launched your product—or even begun creating it. You may be familiar with the “Coming soon” page. It's a page designed to promote a new product or service that's yet to launch. The page highlights the features and benefits of the product, and asks for your email if you're interested in being notified of the launch.

These landing pages are a great tool to measure demand for your product, especially if you haven't yet started building it. The number of sign-ups will tell you whether there is a market for your product, and if there is, whether it's big enough to pursue. The quality of the leads will depend on how much information is provided on the landing page. Giving more information about what your product will solve, as well as the potential pricing options, will help you get emails from people who are really interested in your product, as opposed to being merely curious.

Here's a way to take this landing page a step further so that it not only generates leads, but gives you valuable information on what features you should focus on. Monotask used this clever technique on its coming-soon page. When someone interested in the product filled in their email to get on the launch notification list, a short survey titled “Help Us Build Something You Want” would appear. This hidden survey asked a few questions, such as what methods the person used at that moment to solve the problem, and how much they would be willing to pay for the app.

Hiding the survey behind an email-capture form serves two purposes. First, the initial call to action must be clear and simple—the only thing the user should have to provide is their email address. Showing the form right away would detract from that action and clutter the page. Secondly, only people who have shown interest in your product will be shown the survey, and so the signal-to-noise ratio improves. Using a survey like



Monotask helps you ...

- lock down the internet (or just the parts you want)
- create productive blocks of time
- work on the stuff that's valuable to you

Enter your e-mail below, and we'll tell you when we open up the beta.

Your e-mail:

Please remind me when Monotask launches

Figure 6.2. Monotask's email-capture form.

this is not only a great way to collect data to help you build something people want, but can also arm you with the common pain points you're solving and the objections your users may have about using your product. Once you launch, addressing these issues on your sales pages will help strengthen them.

BURY THE SIGN-UP

There is a UX myth that says that you should present your sign-up link or your sign-up form right away—that is, at the top of your landing page. The logic behind this myth comes from the idea of removing all possible barriers from your sign-up process. The flaw in this logic, though, lies in what gets classified as a barrier—in this case, it is everything other than the sign-up form.

The sign-up form is not the only thing that your visitors need to sign up. Yes, the sign-up link and form are essential for the process, but before people sign up, two more things need to happen. First, the visitor must clearly know what they're going to get once they sign up. Secondly, they must want it. This is where good copywriting



Thanks for signing up! We'll let you know when we launch.

In the meantime, it would be awesome beyond words if you filled out a quick survey, to help us make Monotask something you'll really love. ↓

Help Us Build Something You Want

We haven't even launched yet, and we're looking to make Monotask more useful. Help us?

How do you currently block online distractions?

How damaging are distractions to you?

Business or Pleasure?

What would you be willing to invest in a service that helped you stay on target?

<input type="radio"/> \$6 / month	<input type="radio"/> \$8 / month
<input type="radio"/> \$10 / month	<input type="radio"/> \$12 / month
<input type="radio"/> \$14 / month	<input type="radio"/> \$16 / month
<input type="radio"/> I only use free tools	<input type="radio"/> more than \$16 / month

Survey powered by Wufoo

Figure 6.3. Submitting your email address exposed a short survey.

comes in. Unless the visitor already knows about your product from external sources, they will need to be educated about *what* it does and, more importantly, *why* they should want it.

For example, when ZURB designed the home page for one of its clients and moved the sign-up button to the bottom of the page, they discovered that conversions went up 350%¹—a massive improvement. Vendio had a sign-up form as the first item on one of its pages, pushing the copy aside. In its redesign, they moved the form to its own page, leaving a link to it on the home page. The result? Conversions went up 60%.²

People will not sign up for anything until they are convinced of its merit, and that certainly will not happen when the first thing on the page is a sign-up form or link. Instead, move the sign-up form down the page—or, more accurately, move your copy up the page to take the front row. When people have read your message and are ready to act, they’ll do so.



Figure 6.4. The Vendio website, with the sign-up form right on the home page (initial version).

¹ ZURB, “Why Burying Sign Up Buttons Helps Get More Sign Ups”, smashed.by/signupbuttons.
² VWO blog, “Signups increased by 60% after actually removing the signup form,” smashed.by/abtesting.



Figure 6.5. Vendio with the sign-up form moved to its own page (final version).

GRADUAL ENGAGEMENT

Another UX myth is making the sign-up form as short as possible. In reality, this may or may not be important. What's important is what actually happens *during* the sign-up process and whether it benefits the user's experience. In other words, will the extra step you want to introduce add value for the user?

For example, when Twitter redesigned its sign-up form, it introduced an extra step to the process. Initially, Twitter would try and connect to the visitor's email address to search for friends who are already using Twitter. If the user didn't want to give their email details to Twitter, they would be directed to an empty new profile page—not exactly an ideal starting experience.

The redesign added a step for suggestions. The user can now select topics that interest them, and Twitter will suggest popular accounts to follow. This way, even if the user doesn't find any friends, they will know of interesting people to follow, and the new profile page will start showing those people's tweets right away. Did the redesign work? It did: conversions went up 29%.³

³ Wroblewski, Luke. "Gradual Engagement Boosts Twitter Sign-Ups by 29%," smashed.by/twsignups.

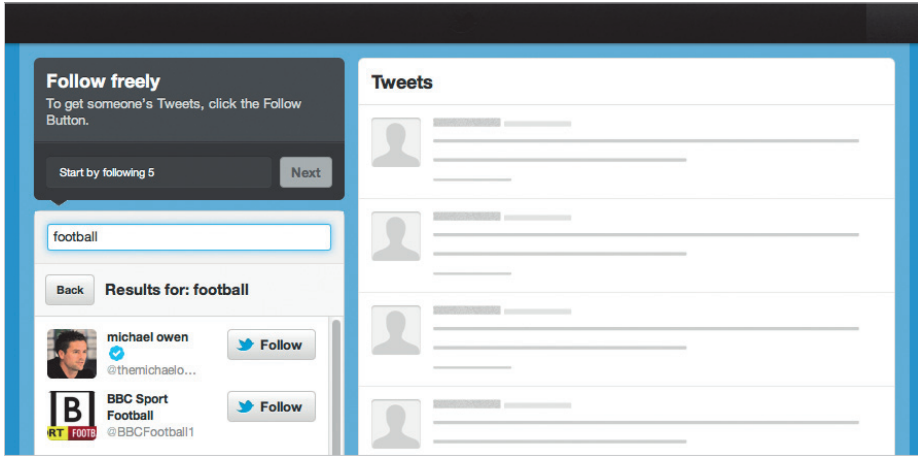


Figure 6.6. Twitter helps you find people to follow during the sign-up process.

Why did it work? The extra step enabled users to engage more meaningfully with the service. Instead of starting you off with a blank slate, Twitter helps you find people to follow right away. Even if you don't know exactly who to follow, you can search through most popular profiles by topic; if you're interested in Web design, for example, you can search for that term right in the sign-up process and get a list of accounts to follow. This makes the service useful right from the start, and deepens engagement during the sign-up process as people get to customize their new account.

PROGRESSIVE SIGN-UP

While lengthening the sign-up process is fine if it makes sense for the user, you could also shorten it, and in some cases even get rid of it altogether during the initial interaction. One of the best ways of educating your potential users about what your product does and how it works is to get them to try it. In some cases, you do not need the user to go through a sign-up form before you can show them your product. If that's the case, why not let the user try it right away?

QuietWrite does this. The first thing the visitor sees is a landing page that briefly explains what the product does. If they're interested, they can click a link and start using it. If they like what they see, they can take the next step of setting a user name and password. QuietWrite uses cookies to track the user during the initial interaction, so that even if the person does not set a user name right away, they can continue working. There is an incentive to setting a user name, though, particularly to access the app from multiple computers. The user then has a good reason to proceed to the sign-up area.

QuietWrite segments the process even further by asking for additional information later. For example, to publish your writings on QuietWrite, you need to pick a display name. This information wasn't necessary to save your account, so it's asked later, when it's actually needed.

Stripe, a payment processor, is another good example of this. On its sign-up page is a “Skip this step” link, which sends you directly to the app. You can look around the app and then get a user name and password if you decide to start using it. Showing your potential customers what they'll be using before they sign up is another way to help sell the product.

On contrast, some services will always need your details before they can be useful. Social networks, such as Facebook, are based around social interaction, which cannot happen unless you identify yourself. Facebook gets you past the sign-up stage right away by putting the form on their landing page. This contradicts the point made earlier about burying the sign-up form, but it works here because Facebook's brand recognition is so high that you can expect people coming to the page to already know about the

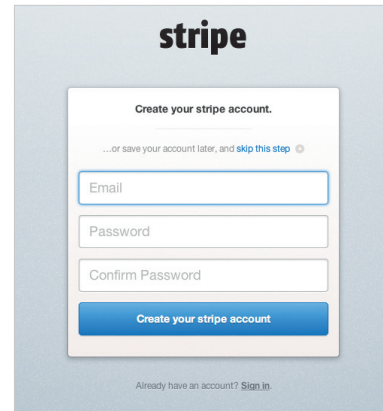


Figure 6.7. The Stripe sign-up screen lets you “skip this step.”

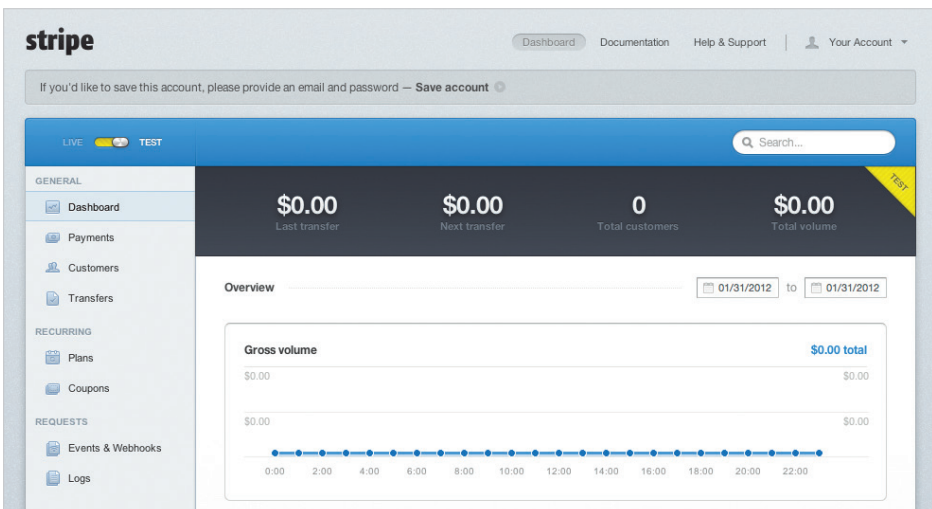


Figure 6.8. You can skip right to the Stripe dashboard without signing up. The “Save account” link at the top lets you sign up later.

service and what it does. One interesting element of Facebook's form is that the user is asked to confirm their email address, rather than their password. This makes sense because if you forget your password, a reset link can always be sent by email, but if your email is incorrect, then logging back in would be a whole lot harder.

ACCORDION FORMS

Whenever you have a very long form, the question may arise of whether to split it into multiple pages to make it more manageable. Showing the long form on one page makes the task look daunting to the user, but so is introducing additional steps to the process. There's a technique that solves this: the accordion form.

Accordion forms are forms that are made up of multiple sections, each with its own sub-heading—all of them placed on a single page. The section that the user is currently filling in is visible, while the rest are hidden with only their sub-headings showing. Once the user is ready to move onto the next section, the next section expands and the last collapses, hence the name.

How well does this technique work in practice? Etre, a London-based usability firm, ran a test on accordion forms.⁴ The test included 24 average users with typical e-commerce experience and ages ranging from 19 to 48. They tested two variations of the form against multi-page layouts and a single page layout with everything shown at once. One variation of the accordion form required the user to click on the next heading to expand it, while the other had a button at the end of each section.

What they found was that the forms really didn't differ much in terms of accuracy rates or user satisfaction, but did differ in the speeds at which users filled out the forms. Surprisingly, the accordion form performed the fastest, even faster than a single page form that displayed everything at once. This means that if your content is time-sensitive—if you are running an auction, for example—the accordion form may be a good choice.

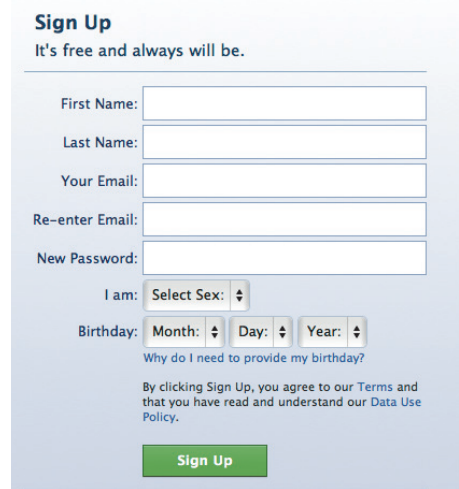


Figure 6.9. Facebook's sign-up form with an email address confirmation.

⁴ Wroblewski, Luke. "Testing Accordion Forms," smashed.by/testing-forms.

The screenshot displays the Apple Store checkout process. The page is divided into several sections, each with a heading and a 'Continue' button at the bottom right of the section.

- Items to be Shipped**: This section contains two sub-sections:
 - Shipping Contact**: Includes input fields for First Name, Last Name, Area Code, Primary Phone, and Email Address (optional). A 'Continue' button is at the bottom right.
 - Shipping Address**: Includes input fields for Company Name (optional), Street Address, Apt. Suite, Bldg. (optional), Zip Code, and a prompt to 'Enter Zip for City and State'. A 'Continue' button is at the bottom right.
- Shipping Method**: Shows 'Standard Shipping — Free' with an 'Edit Shipping Method' link. A 'Continue' button is at the bottom right.
- MacBook Air, 11-inch**: Displays the product image, price (\$1,199.00), and configuration details (1.6GHz Dual-Core Intel Core i5, 4GB 1333MHz DDR3 SDRAM, 128GB Flash Storage, Keyboard (English) & User's Guide, Accessory Kit). A 'Continue' button is at the bottom right.
- Order Summary**: Shows 'Cart subtotal \$1,199.00', 'Free Shipping \$0.00', and a 'Total \$1,199.00'. It also includes a link to '6 or 12 month special financing options' and a disclaimer about Apple's Sales and Refund Policy.
- Just Ask**: A section with a phone icon and the number '1-800-MY-APPLE'.
- Frequently Asked Questions**: A section with several questions related to shipping and tracking.

Figure 6.10. Apple uses the accordion form for their online store checkout. Note that the placement of the “Continue” button is placed inside each section, not at the bottom.


The variation of the accordion form that performed best in Etre’s test was the one with a button at the bottom of each section, not the one that required people to click on the headings to expand them. When people fill out forms, they look for a “Submit” button, and clickable headings stray too far away from that convention to make sense. So if you’re going to use the accordion form, make sure to put a “Submit” button at the bottom of each section.

THE GENDER QUESTION

Sometimes a Web service needs to know a user’s gender in order to enable whatever functionality depends on it. Your users may not want to reveal this information and may wonder why you need to know it at all. Worse, they may suspect that you’ll want to sell it off to advertisers. On the other hand, our actual use may be genuine and harmless. For example, we may want this information to help us better construct status

updates on our social app, e.g. “John has updated *his* profile,” or “Jane has uploaded a new photo to *her* holiday album.”

The team at Bagcheck has come up with a great way to ask this question. Instead of asking the user whether they’re male or female, it asks them to pick a preferred possessive pronoun: “his”, “her” or “their.” This approach makes it clear what you want to use the information for and allows people to opt out by picking the gender neutral “their.”



The screenshot shows the Bagcheck website's account creation page. On the left is a cartoon illustration of a chef with purple hair, sunglasses, and a white chef's coat, holding a whisk and a rolling pin. The main heading is "Create Your Account". Below it are several form fields: "Full Name" with a text input field and a note "Your real name is required."; "Location" with a text input field; "Possessive Pronoun" with three radio button options: "His", "Her", and "Their" (which is selected); "Your Email Address" with a text input field; and "Create a Password for Bagcheck" with a text input field. At the bottom right is a green button labeled "CREATE ACCOUNT".

Figure 6.11. The gender question on the Bagcheck sign-up form.

GOOD DEFAULTS

When your users are entering new data, it may be a good idea to present them with some typical default values. For example, Etsy tries to guess your regional settings when you first arrive on its website. A box pops up at the bottom of the screen asking you to verify your location, which it guesses based on the language settings of your system. Your selection affects things such as the currency used to display item prices. While this guess may work in most cases, Etsy understands that it may still make a mistake and so a verification message is used.

Another good example would be Homesite, a home insurance website that prefills some common values in its forms. On the page for property information, default values are filled in for the property coverage deductible, personal liability and so on. Using defaults in this way won't work everywhere. For example, when adding a new event on a

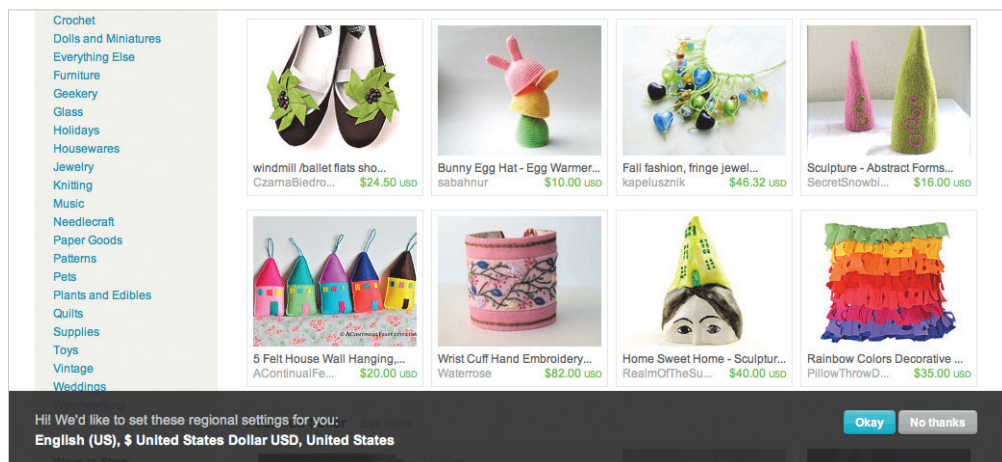


Figure 6.12. Etsy tries to guess your location based on the language settings of your system.

calendar, you really want to see an empty form because every event is different. On the other hand, if there is a pattern in how the user may want to use your product, smart defaults can be a good way to simplify their experience.

UI DESIGN IS COPYWRITING

One of the most important aspects of any visual interface is the copy—that is, the labels on buttons and forms, the instruction dialogs, the headings, the error messages, and so on. Your copy is the most direct way of communicating with your user, and while advice is often given to cut copy to the bare minimum because people don't read, we should never do this at the expense of clarity. Additionally, the copy must always be written for your users, not for you; a distinction that is easy to forget when you spend a lot of time using your own product and so already know how everything works.



Figure 6.13. Amazon uses the label “Proceed to checkout” for its check-out button.

Context can shift the meaning of words. Baymard Institute ran a usability study that found that the word ‘continue’ in an e-commerce interface confused 3 out of 10 testers.⁵

⁵ The Baymard Institute. “Contextual Words Like ‘Continue’ are Usability Poison,” [smashed.by/poison](https://www.smashed.by/poison).

Why did this happen? Context is key. To somebody who has added an item to their shopping cart and wants to proceed to checkout, the word “continue” means “continue to checkout.” To someone else who wants to keep shopping, the same word means “continue shopping.” Whenever a label is tied to context, make sure to clarify as much as possible. So, instead of “continue,” use more descriptive labels like “continue shopping” or “continue to checkout.” Clarity trumps brevity.

Here’s another example: HubSpot had a label on their domain setup screen named “Add domain.” This was confusing. What the developers of HubSpot meant was that this interface could be used by customers to add an existing domain to their account. What the customers actually thought it meant was that the software was asking them to create a new domain—something they didn’t want to do since they already had one. The verb “add” was causing confusion. HubSpot changed the label to “Connect your domain,” and the confusion went away. The original label made sense to HubSpot’s developers using the product because they thought about the problem from their own perspective, but this perspective was not shared by the customers. Write for someone encountering your product for the first time, and double-check that the labels you want to use do not have other meanings.

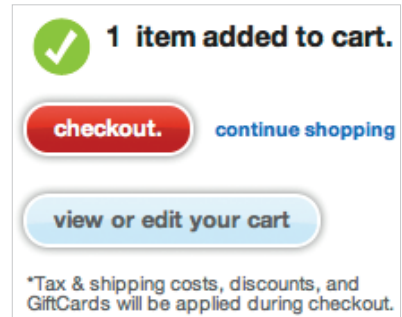


Figure 6.14. Target drops the verb and simply uses the word “checkout.”

EXTRA ATTENTION TO DETAIL

By now we’re all used to forms that use JavaScript to validate data on the fly. Square, a new payment processor that lets you use your phone as a card reader, goes beyond this by adding other dynamic elements to make the experience of interacting with the app even more polished.

When you’re filling in your card details using the Square app, the card icon will change its cover image according to whatever card you have entered (VISA, MasterCard, etc.). When the card’s number has been filled in, all but the last four digits are faded out. The little card icon will then flip over, and the area where you can find the card verification value (CVV) is highlighted.

Square has plenty more touches like this. When an American Express number is typed in, the grouping changes from 4 digits to 4, 6 and 5, reflecting the number structure on the card you're using. Since the CVV for American Express is on the front, the little card icon won't flip around but will instead show you where to find it on the front of your card. If you enter an incorrect number of CVV digits, the field will shake and turn red to tell you it's not right. Also, entering an incorrect expiration date is impossible—the interface simply won't let you do it.

This attention to detail is what contributes to making products that people love using. It guides the user through the form and nudges them in the right direction when they stray aside. The different use cases are also covered, for example, taking into account that the CVV on a particular card may be on the front and not on the back. These use cases have been given their own implementation to ensure that everyone gets the same experience.



Figure 6.15. Square hides the card's full number and highlights the location of the CVV in the icon.

New and Experimental Techniques

“It's really hard to design products by focus groups. A lot of times, people don't know what they want until you show it to them.”

— Steve Jobs

This quote from Jobs goes well with the even more famous one by Henry Ford: “If I'd asked my customers what they wanted, they'd have said a faster horse.” Innovation happens when you try new things and don't get stuck in the old ways of approaching a given problem. Accepted design patterns and best practices tell us what works well, but they don't tell us how to build something that's even better. Simply following the patterns will ensure you have a solid product, but it will mean that you'll always be one step behind those companies and designers who come up with novel solutions to see if they work better.

Sometimes we have to break rules and experiment. For example, during the beta stages of their Lion operating system, Apple tried a new button design for a radio button style control, where only one item could be selected at one time, e.g. picking the calendar view mode, which could either be day, week, month or year. The new design highlighted the currently selected item, and made the rest look depressed. It was meant to look like the button could be moved around left and right to pick the desired mode.

It didn't work though. Many users didn't take well to the change,⁶ and Apple went back to the previous design, in which the buttons actually looked like buttons, and the currently selected one was depressed. The experiment wasn't a failure because it gave the designers valuable information on what works best in that scenario, and the better solution was ultimately shipped. If the new solution you come up with doesn't follow accepted patterns but happens to solve the problem better, then it will inevitably become a recommended pattern.

In this section, we'll cover experimental UX techniques. They may or may not work for your products, but they should give you fresh ideas on how you can push your designs to the next level.

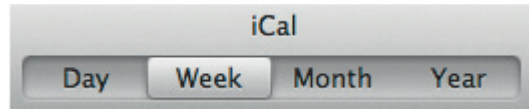


Figure 6.16. Apple's button style experiment had a selector that looked like a slider.

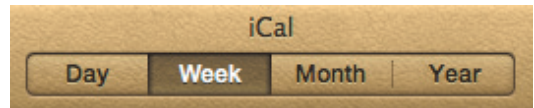


Figure 6.17. Apple ultimately went back to a clearer solution, with all buttons looking like buttons, and the currently selected item clearly active.

INTERACTIVE STORYTELLING

Storytelling has always been a great way to introduce ideas and products. The browser isn't limited to just showing text and video. We can go further and create interactive experiences that allow visitors take part in the story we want to tell. Creating an interactive story isn't easy, but the results can engage the user on a much deeper level than the passive skimming we're used to.

When Nerd Communications was developing Ben the Bodyguard app, it designed its website as an interactive story. The website featured a top-down view of a dark, dangerous street. In the center was Ben the Bodyguard—the app represented as a character. As you scrolled down the page, Ben would begin to walk with you, and as you

⁶ Apple Insider, "Inside Mac OS X 10.7: Lion," smashed.by/appinside.



Figure 6.18. Ben the Bodyguard guides you on an interactive walk through The Mean Streets.

scrolled further, he would reveal the concept of the app and why you may need it. The makers of the app managed to turn your typical bullet list of features and benefits into an interactive and engaging experience.

Spent, a donation website for the Urban Ministries of Durham, demonstrates this, too. Asking for donations isn't easy, so why not turn it into a game? Spent is a Web-based game in which you play the part of a typical unemployed American. The objective is to survive a month without running out of money. Along the way, you have to find a job, look after your kid and juggle a variety of mini crises. The successful outcome is then to make the player empathize more with the jobless and give a donation at the end of the game.

Both of these websites are not only great ways to introduce their respective concepts and engage their visitors, but also great marketing tools. Both Ben the Bodyguard and Spent went viral when they were first released. People shared them because the



Figure 6.19. Make it to the end of the month without running out of money in Spent.

sites were interesting enough to view on their own, even if you didn’t end up buying the app or donating. Now, of course, the success of these designs will ultimately be judged by whether people purchase the app or donate, but if you know that a large portion of your audience spends a lot of time on social networks, then a viral quality that attracts this sort of traffic may work for you.

COMPLETENESS METER

Do you want your users to take optional action, such as add additional information to their user profile? Show them that there are still things left to do with a completeness meter. The completeness meter is just like a progress bar; but unlike a typical progress bar, which shows the progress of an action, a completeness meter focuses on how much still needs to be done by taking additional actions. It doesn’t have to be a bar, but it does need to have a way of indicating progress. If the user doesn’t like to leave things undone, they will be compelled to go through the list.

For example, Klout has a completeness meter under the headline “Connect.” It’s just a plain text to-do list that invites you to take various actions, such as connecting your Klout account with Twitter and Facebook, following Klout on Twitter, sharing your

Klout score, and so on. As you do each of the actions, they get crossed off the list. Another example would be Groupon. Groupon's completeness meter in the sidebar is a to-do list just like Klout's, but with a progress bar at the top that fills up as more items get crossed off.

Why does this work? There are two psychological drivers at play here. The first is curiosity. People want to find out what happens when the meter reaches 100%. Will there be some reward at the end? The second driver is the feedback loop. When the user completes a task, it gets checked off and the meter goes up. This establishes a clear goal and gives the user directions on what to do next.

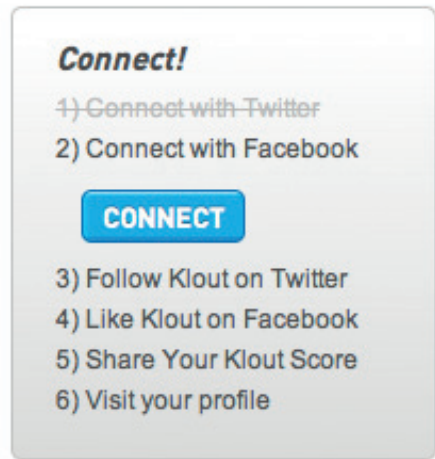


Figure 6.20. Klout's completeness meter invites users to become active and check off items on their to-do list.

REAL TIME

Real-time updates have always been possible, but a new technology makes them even easier now. WebSockets is an HTML5 feature that lets you keep an interactive communication session open on the server without having to poll it every few seconds and wait for a reply. What this really does is make it much easier to provide real-time experiences that run faster and don't flood your server with requests.

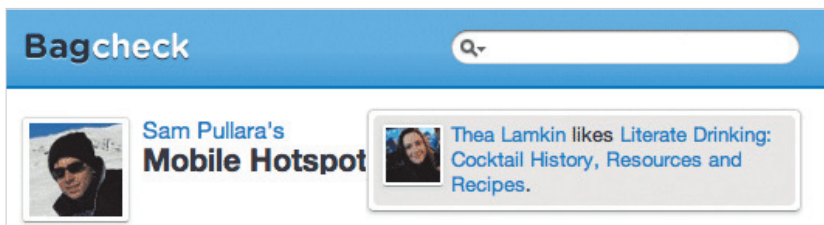


Figure 6.21. Real-time notifications in Bagcheck.

Bagcheck uses WebSockets to provide real-time updates on the views counter on each user's bag page. The counter shows the number of views a page has, and will update instantly whenever someone else opens that page. Bagcheck also displays real-time notification messages whenever a person you follow creates, likes or comments on something.

The app has even more real-time stuff; for example, whenever someone comments on an item, the comment appears instantly.

WebSockets opens up new ground for Web apps and websites to explore. Real-time analytics and games are the obvious uses, but as Bagcheck shows, it can also be applied to interface components that we're used to seeing as static, such as a list of comments. But just because we can implement real time doesn't mean we should, because the obvious risk is that this can distract users who do not expect the page to change.

COUNTRY SELECTOR

Country selectors tend to be drop-down menus that open up a page-tall list of countries that you then have to scroll through to get to the one you want to pick. Can we do better? Yes. Christian Holst and Jamie Appleseed from Baymard Institute managed to redesign the country selector to make the process painless.⁷

The idea is to use a live search field for country selection, so when the user begins to type in the name of their country, a search would be run and the list of matches would drop down below the field. The top match will be preselected, so the user simply has to hit the "Enter" key once there is a correct match to fill in the rest of the country's name. The user already knows what they want, so typing the first letters of the country's name is much faster than scrolling through a list that may include over a hundred items.

This implementation works because it tackles three underlying issues. First, you have to account for typos and sequencing; the user may type the words in the wrong order or spell them incorrectly. Secondly, some countries go by multiple names. For example, another name for the Netherlands is Holland, and so people typing in "Holland" should expect to get a correct match. Same thing with typing in "America" instead of "United States."

Finally, some countries are probably more common than others among your user base, so "United States" should likely be the first match when someone starts typing "United," and not "United Arab Emirates"—unless of course, many of your users do live in the United Arab Emirates.



Figure 6.22. The country selector in action.

⁷ Christian Holst. "Redesigning The Country Selector," smashed.by/selector.

The live search text box can be used for things other than countries, but the three issues I've just covered will still remain so it's important they're tackled to ensure that your implementation is solid.

PROGRESSIVE LOG-IN

Many websites today use third-party vendors to authorize user log-ins, i.e. people who already have a Google or a Facebook account—or any other service that allows this—can use those services to identify and authorize them on other websites.

Sometimes several log-in providers are used at once, which can lead to people forgetting which one they used to sign up with.

The team at Bagcheck has experimented with a progressive log-in process that solves this problem. Instead of your typical log-in form with the two fields for your user name and password, as well as the third party log-in buttons, Bagcheck shows just a single text field for the person's name (or email address). As the user starts typing, a live search is run to find the person's account. If the account is found, further log-in options are shown, including the button for the third-party log-in vendor that the person used to register with Bagcheck, allowing them to now sign in.

This approach eliminates the problem of people forgetting which third party log-in vendor they've used to sign up for the site. However, two caveats apply. First, people can no longer sign in with one click—something they could do when all the log-in options were shown right away. Secondly, the live search finds other people who have signed up to Bagcheck. Because Bagcheck accounts are public by the nature of the service, showing other people is fine here, but this will not work for services that need to keep accounts private.

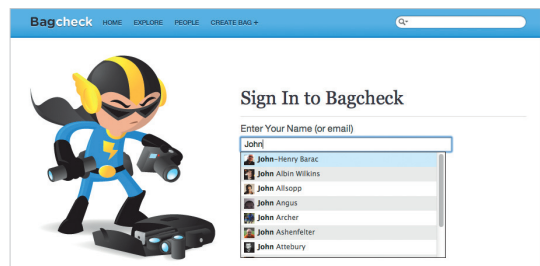


Figure 6.23. Typing your name on the log-in page initiates a search of Bagcheck's user base.

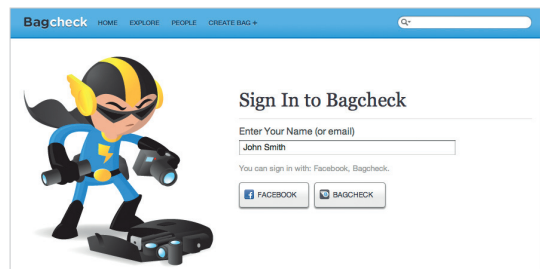


Figure 6.24. Once you've found your account, you will see the options you can use to sign in.

RESPONSIVE MOBILE UI

Web-based mobile interfaces bring up a number of usability challenges. A fingertip is less accurate than a mouse cursor, so it's often unclear whether you've pressed the right link or not. Slow connection speeds make it difficult to know whether a page is loading or stuck. If data is cached, it's not obvious if and when it will be refreshed, which is particularly a problem if the user is looking for new content.

When 37signals set out to design the Web-based mobile version of its Basecamp project management app, it sought to tackle these problems and create an experience that rivalled that of a native app. To handle touch, 37signals made sure that everything had a selected state, so that a button would highlight instantly when touched, letting the user know that their selection has been registered, and that they had pressed the correct button.

When the app loads for the first time, it fetches a lot of data that gets cached on the device. This can take a while, so 37signals designed a special loading screen for the first launch. The loading screen counts the time it takes for the app to load, and when that time passes a certain threshold, a message pops up below to let the user know that the app is still alive and loading. If the app takes far too long to load, another message pops up to give the user the option to try again or switch to the desktop version of the app.

Because the app uses a lot of caching, the user needs to know whether any given content shows the most recent data. Instead of refreshing the whole page when checking for new content, the app shows a spinning icon in the top-right corner, which tells the user that Basecamp is communicating with the server while allowing them to continue working.

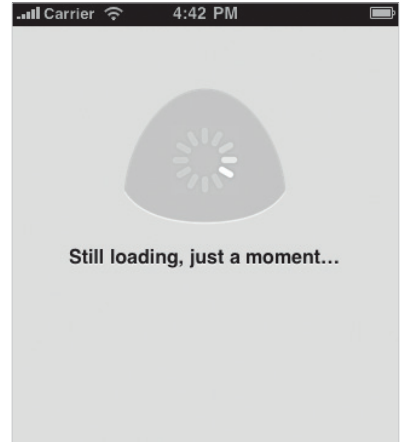


Figure 6.25. If the initial loading takes longer than normal, a message will appear to let you know that the app is still working.



Figure 6.26. An unobtrusive spinning icon in the top-right corner tells the user that the app is checking for updates on the server.

ICON MENUS

Icons are not only reserved for apps. You can use them in product selection menus on e-commerce sites, too. For example, Bonlook, a vintage eyeglasses store, uses icons in their drop-down menu to let visitors browse by the shape they want. Eyeglass shapes are not easy to describe in words, but icons communicate them instantly.

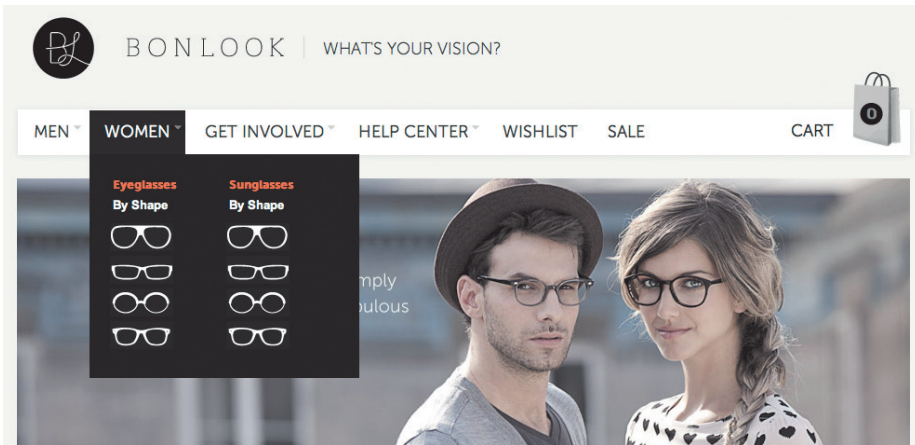


Figure 6.27. Bonlook's selector for eyeglasses.

Another example is Heppo.se, a Swedish shoe store. Here again, we'll find a drop-down menu that shows you the various types of shoes and boots you can buy.



Figure 6.28. Heppo.se shows an icon for every type of shoe they sell.

Each menu item is accompanied by an icon showing the shape of the shoe. The store also sells scarves, hats, brushes and other items, each accompanied by its own little image. Using icons in this way enables visitors to browse the store even if they do not speak Swedish. This sort of interface really shines when an item comes in multiple varieties, such as different shapes of boots, because the customer is able to scan and process the options at a glance.

Customer Service Is UX

Tony Hsieh made Zappos a success story. Since joining Zappos as CEO in 2000, he has doubled the company's revenue every year, reaching \$1 billion in 2009. Zappos is an online shoe store, but the interesting thing here is that Hsieh wasn't even interested in shoes. What he was interested in was customer service, and that passion for providing a great customer experience was what really made the company so successful.

When we talk about UX, we normally think of user interfaces. But your users interact with more than just the user interface: they read through your documentation and communicate with your support staff. All of this contributes to the overall user experience, and making sure things run well here will help make your customers happy. Below are some ideas on how to improve the customer service side of things, from innovating your support to providing better documentation.

“We asked ourselves what we wanted this company to stand for. We didn't want to just sell shoes. I wasn't even into shoes—but I was passionate about customer service.”

— Tony Hsieh, Founder of Zappos

SMILEY SUPPORT

How does one visualize the quality of support a company provides? 37signals designed a rating system by which people rate the support they have received at the end of every email exchange. The rating system is just 3 links: one for “great,” another for “just OK” and the last for “not so good.” Each rating has an illustration of a smiley face to make the selection even more straightforward at a glance. These smileys are colored like traffic lights: green for good, yellow for OK and red for bad.



Figure 6.29. 37signals shows off its support ratings on the Smiley page.

Once a customer submits a rating, they are invited to provide additional information. This is optional but a great way to gather feedback that can be used to improve support. As customers rate the response they've received to each support request, 37signals collects data on how well the company is doing, and can even track ratings of individual staff members to see if anyone is falling behind. This can be visualized by stacking all these smiley icons together to get one big image, let's say for the last 100 ratings. 37signals also uses this system as a great marketing tool. A page has been created to display the last 100 ratings, visualized with the help of traffic-light colored smileys. This way, the predominance of green on the big wall of smiley faces gives the customer an idea of how healthy the 37signals support actually is.

DOCUMENTATION IS ALSO UX

Documentation is often the weaker part of the overall user experience. It's not particularly exciting to write, and not very fun to read. But it doesn't have to be this way. The team at MailChimp has developed a great method of producing documentation. First, it covers the basics, writing about all of the major features. Then, whenever it gets a support request, the company writes out a great, detailed answer. After sending the answer off to the customer, MailChimp files it in its knowledge base. This way, the customer receives great service, and the time spent will also go into building up an in-depth knowledge base asset that all other customers can use.

To ensure your documentation is accurate, actually go through the process as you write it. This will also give you an opportunity to take plenty of screenshots, which will help to explain the steps quicker. Documentation doesn't have to be boring to write either, so have fun with it if possible—it will take away the tedium of reading it. On one page that explains how to set time zones, MailChimp has a photo of walls clocks sitting above a mural of Chimpzilla destroying city buildings. This playfulness makes the instructions much more human and interesting.

Be careful, though. Jokes work only when the reader is receptive to them. When troubleshooting a critical problem, the reader will probably be stressed out, and humor will have no place on those pages. Handle these issues professionally and with care. If the pages are not critical, though, learn from MailChimp and have some fun; it will break the tedium of an otherwise boring task.



Figure 6.30. The Chimpzilla picture from the MailChimp support page on time zones.

ADD CREDITS

Ryan Carson at Think Vitamin advocates giving users credits whenever things go wrong. If a user runs into a problem with his service, Ryan compensates them by crediting their account (offering a free subscription period, for example). This goes beyond fixing the problem; it tells customers that if something goes wrong, Think Vitamin will take full responsibility.

This tactic worked tremendously well for Think Vitamin. People are so unaccustomed to great customer service that when they get to experience it, they will be truly grateful. Not only that, but they will want to share their experience, too. The gains here are much bigger than the loss of giving your customer more than they've paid for: you'll be making them happier, more loyal and ready to recommend your service to their friends.

PLEASE-REPLY

Every interaction with your customers is an opportunity for meaningful engagement that strengthens your relationship. It is surprising, then, how many companies still use no-reply email addresses today. After all, what better way to tell your customers that you care about what they think than to tell them that they shouldn't reply back to you, right?

But it's worse than that. No-reply email addresses may negatively affect delivery rates. For example, Google's Priority Inbox feature monitors addresses people respond to most, and so marks them as more important. It is very likely that their spam algorithm also takes replies into account—after all, if you're replying to an email, it makes it more likely that the email is genuine and not spam.

What to do? Turn this problem into an opportunity by turning the no-reply into a please-reply. If you have a support ticket system, you can forward the replies you receive there, and then handle them like you normally would.

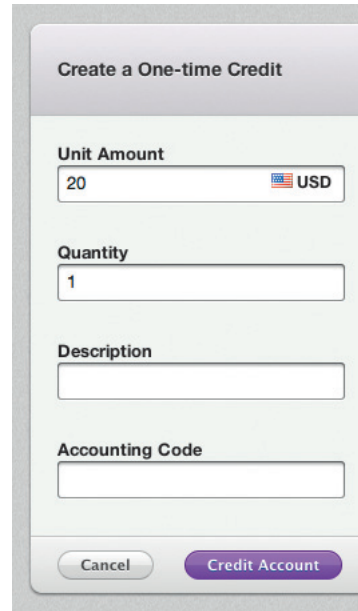
A screenshot of a web form titled "Create a One-time Credit". The form has a light gray header with the title. Below the header, there are four input fields: "Unit Amount" with the value "20" and a currency selector showing "USD" with a flag icon; "Quantity" with the value "1"; "Description" which is an empty text box; and "Accounting Code" which is also an empty text box. At the bottom of the form, there are two buttons: a "Cancel" button and a "Credit Account" button.

Figure 6.31. Modern payment-management systems usually have an option to assign credits to active accounts. The example above is a credits page from Recurly.

You should also set up filters to differentiate between genuine replies, automated replies (like out of office) and delivery failure messages. You want to get the automated stuff out of the way to ensure you’re not swamped. If your system doesn’t have this, Gmail offers some good filtering options as well as strong spam protection. Above all, remember to use this opportunity to build stronger relationships with your customers when they do reply.

REWARD PASSIONATE USERS

Marketing your product doesn’t have to mean throwing lots of money at ads. The best salespeople your product can have are passionate customers. They’re the people who use your product every day and so know all about the things that make it great and worth using. A recommendation from a friend has no commercial bias, which makes it much more effective than a paid ad.



Figure 6.32. Dropbox has set up ways to easily refer people via email and social media.

Dropbox used to buy ads,⁸ but it found that it spent more money on acquiring paid customers than it was getting back in revenue. The company rethought its strategy and focused on customer-driven marketing. Its new approach was to reward passionate customers for referring their friends. Dropbox gave incentives to both parties: referrers got free extra storage, and their friends started with more storage space than they otherwise would have had.

Encouraging passionate users to invite their friends and rewarding them for the gesture has proven to be very successful for Dropbox. Another advantage of this approach compared to typical affiliate marketing is that you are rewarding users through the product itself, rather than with money. This increases the value of your product for the people you reward and eliminates the hassle of processing payments.

THANK YOU

How often do you thank your customers for choosing your product or service? The typical response you'll get from most companies is an automated email, so doing something special is relatively easy. Why not delight your best customers with a written thank-you card? Not an email—an actual card. It's a small gesture and won't cost much, but it will make your customers happy and it will make your brand really stand out.

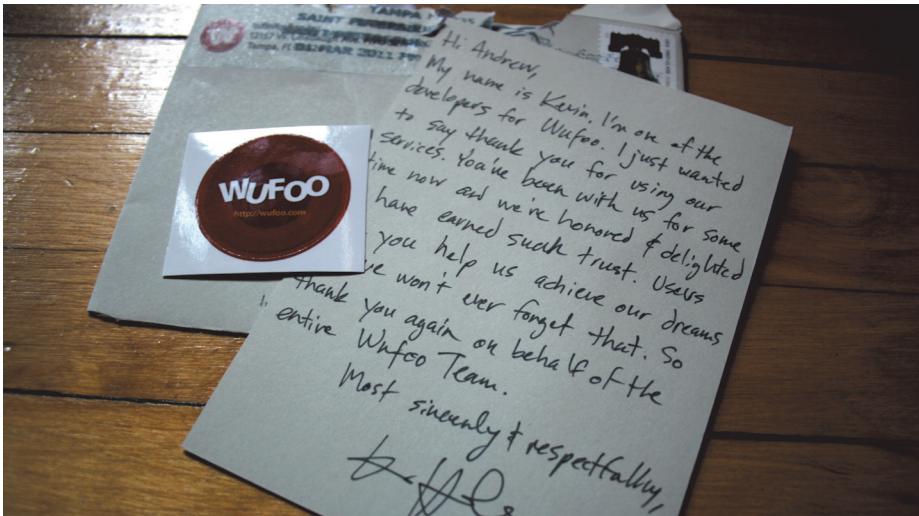


Figure 6.33. A thank-you card from Wufoo.

⁸ Porter, Joshua. "Reward The Passionates," smashed.by/dropboxads.

This is what Wufoo does with its top customers. It has even gone so far as to handcraft thank-you cards, with staff members handwriting each one. To speed things up, Wufoo brings everyone on board; on Fridays, the team members would write about 10 cards each to the people who have helped make their company successful.

Your customers won't expect you to put so much effort into saying "thank you," so cards like these would come as a delightful surprise.

Anti-UX: Dark Patterns

Generally, bad design is a product of incompetence. There are times, however, when poor design is intentional, aimed at tricking the user into doing a certain thing for the benefit of the designer. This technique even has a name: *dark patterns*. We'll cover a few examples here. Avoid such patterns in your own work because they all cross the line from persuasion to deception—and deceiving users is not a viable long-term strategy—neither from an ethical or a business standpoint.

Here's a typical dark pattern used by advertisers on Softpedia: a disguised ad. Softpedia shows a file download link, however, just over the top of it and to the left are advertisement banners. Each banner image itself looks like a download link, together with little rating stars. A user who has just arrived at the page and is looking around for a download link may fall into the trap and click on one of the two fake ones presented by the ads.



Figure 6.34. Disguised ads on Softpedia.

Here's another dark pattern: a trick question. The sign-up page of the Wired magazine used to have some check boxes at the bottom that asked you whether or not you wanted to receive offers from them or their partners. There were 8 boxes in total. Why so many? Wired alternated between opt-in and opt-out on every line, and it danced this jig twice. It distinguished between phone and mail offers and between Wired and its partners, and it had separate boxes for opting out. It has since cut this in half to a more manageable 4 radio buttons.

The screenshot shows the 'WIRED Subscription - Address Details' form. At the top is the 'WIRED' logo. Below it, the form fields include: Magazine (WIRED), Subscription Term (Choose a subscription...), and This subscription is (For me, A gift). A red message states: 'Leave checked if Delivery details are to be the same as Payer details'. The form is divided into two columns: 'Payer Details' and 'Delivery Details'. Each column has fields for Title, Forename, Surname, Address, Town/City, County/State, Post/Zipcode, Country, Email, Confirm Email, Tel. No, and Mobile No. At the bottom, there are two sections of checkboxes. The first section asks if the user wants to receive offers and opportunities from Wired and Conde Nast, with options for email, sms, and phone. The second section asks if the user wants to receive special offers from partners, also with options for email, sms, and phone. A 'Continue' button is located at the bottom right of the form.

Figure 6.35. Wired tried to get you to sign up on their mailing list with 8 different checkboxes.

One other dark pattern example: *forced continuity*. This one is used to get users to sign up for a recurring subscription. For example, signing up to a free trial and then having your account automatically upgraded to paid plan when the trial ends without an adequate reminder beforehand. At one time, Audible tried to push its membership plan without making the monthly charge clear. The checkout process simply displayed the price, so some users assumed this was a one-time payment, only to be surprised when

their card got charged the following month. Fortunately, Audible has listened to complaints and has since modified it’s process to make it more clear.

Steering away from these patterns is recommended. Sometimes you may get close to crossing the line between persuasion and deception, but you’ll always know when you’ve crossed it. Using a design technique to trick the user to take some action usually means that they’re not aware of it happening. This may mean not seeing a hidden fee, clicking the wrong checkbox, clicking on a disguised ad, and so on. The user takes an action that they would not have taken had they been given more information. Dark patterns always try and withhold information from users, and if you knowingly use them, you should also know that you’re crossing the line from persuasion to deception. Deceiving your users and your customers has never been, and will never be a good long-term strategy.

Your Cart	
Item	Price
New credit applied to:	
Freedom: A Novel	\$0.00
AudibleListener Discount Gold Membership	\$7.49
Subtotal	
\$7.49	

Figure 6.35. Audible didn’t make it clear that their Gold Membership was actually a monthly subscription.

The Value of Good Design

On 24 January 2012, Apple announced its financial results for the first quarter of its fiscal year: a record revenue of \$46.33 billion, and a record net profit of \$13.06 billion. It’s one of the highest quarterly profits on record, ever. Apple’s success stems from their focus on exceptional product design. Its marketing is effective, but it wouldn’t work without a strong product, and likewise, the culture just wouldn’t be the same if the people working there didn’t love the products they were building.

Here’s a story to illustrate just how important product design was to Apple during its resurrection under Steve Jobs. Jonathan Ive, Apple’s chief designer, wanted to add a handle on the new translucent iMac he was working on. At the time, he felt that people weren’t comfortable with technology and so were scared to even touch it. The handle would give people permission to touch the computer and make it more approachable.

The handle would have been very expensive to build, and thus, Ive faced strong resistance from other executives and engineers. They wanted to know how the handle would pay off and what its return on investment (ROI) would be. When Steve Jobs saw

the handle, he instantly understood its purpose and gave it the go-ahead. He overruled all objections and sent a clear message to his staff that strong product design was the most important thing to Apple, not cost analyses or ROI calculations.

Years later, we can see that his strategy has paid off, not only from the standpoint of creating products and a brand that people love, but that of building one of the most profitable companies in the world. If you are in doubt whether you should spend that extra time and effort designing a product you and your customers will love using, remember the iMac handle. While their immediate value may not be clear, strong design and attention to detail will pay off in the long run by helping you build a strong brand, loyal customers and a product that you are passionate about.



About the Author

Dmitry Fadeyev is the founder of the UsabilityPost blog where he posts his thoughts on good design and covers interesting interface design techniques. He's been doing freelance Web design over the years, with a recent notable project being the UX site for the Stack Exchange network. His latest project is a Web app called Usaura which helps designers run micro usability tests using screenshots of their interfaces and prototypes.



About the Reviewer

Joshua Porter is an interface designer and founder of Massachusetts-based Bokardo Design, where he focuses on the design of social Web applications. A Web geek for over a decade, Josh designs simple, usable interfaces for clients, from startups to giants. He also consults with companies suffering from severe cases of feature creep and those that merely need objective advice. Josh wrote the book *Designing for the Social Web* and speaks regularly at Web design conferences and events around the world. Since 2003, he has written the popular design blog bokardo.com, which is quite well known for making design issues either easy to understand or more complicated than ever. When he is not designing, consulting or writing about both, Josh is either rock climbing or trying to keep up with his two-year old.



Designing for the Future, Using Photoshop

Written by Marc Edwards

Reviewed by Jon Hicks

AS AN EVER EVOLVING PLATFORM, the Web has undergone significant changes since its inception. It has continued to diversify and become available to more people on more devices and in more locations. Having often been ignored, good design and user experience are now almost mandatory for success. The lines between desktop software, mobile software and the Web continue to blur as designers and developers use a combination of what's familiar and what's best for the situation at hand. Tools, techniques and requirements continue to change.

Today, native desktop and native mobile software often contains HTML, CSS, JavaScript and other languages—methods and techniques that were created with the Web in mind. And as websites and Web apps grow in both size and function, it is common for designers to use techniques that were developed with native software in mind. The underlying code and technologies may or may not be of interest to the designer, but the restrictions they put on what's possible and on the designer's assets, such as images, probably will be. Luckily, the design challenges for native apps and for the Web are similar, so a lot of the solutions can be shared.

Let's look at some of the challenges that designers face now and will face in the future. We will then address ways to tackle these challenges using one of the most popular tools available: Adobe Photoshop.

Screen Sizes

The explosion of mobile devices has changed the game. If you're building a website, you can't assume screen size, resolution or pixel density of the device your design will be seen on. Mobile and desktop apps are a little more targeted, but designers still need to be mindful of how their designs will be seen across different devices.

It almost seems inevitable that post-PC devices will overtake desktop and laptop computers in numbers. Factoring in how your website will appear on a 4- or 10-inch mobile screen, as well as a 15-inch laptop or 27-inch desktop screen, is becoming increasingly important.

Pixel Density

Pixels—the square picture elements that make up everything we see on a computer display—are shrinking, allowing crisper, more printed paper-like results. What is currently possible is fast approaching the limit of what is desirable.

Pixel density is typically measured in pixels per inch (PPI). A pixel density of 100 PPI means that 100 pixels are contained within a 1-inch row. A 1 x 1 inch area on a 100-PPI display would contain 100 × 100 (or 10,000) pixels. This might all sound a little dry, but the concept is vitally important when working out how large your design elements need to be and how they will scale. Let's look at some real-world examples.

The iPhone 3GS' display is 320 pixels wide by 480 pixels tall. The screen itself is 3.5 inches diagonally, making the pixel density 163 PPI. By contrast, the iPhone 4's "Retina" display is exactly the same physical size, 3.5 inches, but the resolution is 640 pixels wide by 960 pixels tall: exactly double the iPhone 3GS' screen. The pixels on the iPhone 4 are exactly half the width across and half the height, making its pixel density 326 PPI. Another way to look at it is that for every pixel on an iPhone 3GS, there are 4 pixels in a 2 × 2 grid on an iPhone 4.

Some common display pixel densities:

Device	Pixels	Inches	PPI
iMac (2011)	2560 × 1440	27	109
iPad	1024 × 768	9.7	132
iPhone 3GS	320 × 480	3.5	163
Samsung Galaxy SII	480 × 800	4.27	218.5
Nokia Lumia 800	480 × 800	3.7	252
iPad (3rd generation)	2048 × 1536	9.7	264
Samsung Galaxy Nexus	720 × 1280	4.65	316
iPhone 4 and 4S	640 × 960	3.5	326

Figure 7.1. Resolution and screen size of iMac and some mobile devices.

Since the very first display was invented, the trend has been to increase pixel density. This will likely continue until all or most displays are in the 200 to 350 PPI range because that is around the threshold at which human eyesight can no longer distinguish individual pixels (although the exact PPI of that threshold will vary according to the viewing distance, how good your eyesight is and when you had your last coffee). We'll look at how to construct Photoshop documents so that they scale seamlessly for all of the graphics targets required for the Web, iOS, Android, Windows Metro and other platforms.

Location

In the past, websites and applications were viewed only on desktop computers. Today, you can't assume that your design will be viewed on a bright screen in an office, study or lounge. If your mobile app is for finding great local cafés, then there's a good chance it will be used in broad daylight on a sunny day, while the viewer squints as cloud reflections bounce off their phone's screen. Getting your design's contrast right is vital, as is testing on different devices in typical scenarios. We'll cover some methods of testing designs on devices in the final part of this chapter.

Realism

Higher-pixel-density displays, multi-touch input and faster graphics processing have created great opportunities to make user interface (UI) design and animation far more realistic. Beautiful designs with subtle shading, shadows and textures are now possible. Metaphors can be taken further. Ornamental design cues, often called “skeuomorphics,” can be added, hinting at how to interact with content. The edge of an open book could show a stack of pages, suggesting that swiping from right to left to turn the page is possible. A synthesizer app might feature patch leads that can be virtually plugged in to re-route audio signals.

These examples aren't requirements—the software doesn't need pages or patch leads in order to function. They're a hat tip to the analogue world. If used properly,



Figure 7.2. The Early Edition 2 by Glasshouse Apps.

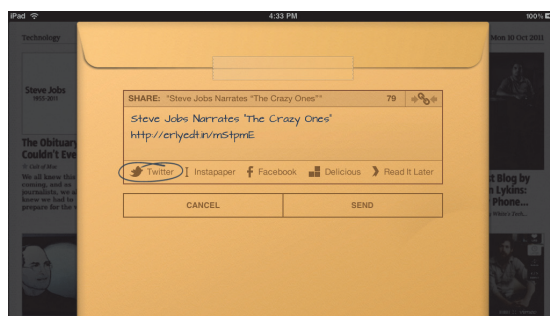


Figure 7.3. Nostalgic ornamentation by Glasshouse Apps. The Early Edition's sharing options mimic a real-life envelope.

these garnishes should also make a design more usable because they help explain functions in a familiar way.

Be careful, though. Adding real-world or nostalgic ornamentation might hold promises that users expect you to deliver on. If a page looks swipeable, then it should be. Some real-world connections might also be too tenuous; if your users are young enough, they might not know what a vinyl record, 3.5-inch floppy disk or Rolodex is.

Building to Scale

As discussed, some devices already have high-pixel-density displays, and we'll see more high PPI mobile devices coming out. We'll also very likely see high PPI desktops and laptops, too—Windows, Mac OS X and the Web as a whole will follow the exact same trend as mobile.

We're in the middle of a transition. The older, lower PPI screens will be around for some time, so both low and high PPI screens will need to be supported. Different platforms are handling this transition slightly differently, but typically you'll need a complete set of images for each pixel density you are targeting.

iOS

For native iOS development, Apple has opted for two display densities and, therefore, two UI scales. The newer displays are exactly double the pixel density of the early-generation devices; for the iPhone, this means 163 PPI for the early models and 326 PPI “Retina” displays for the iPhone 4 and subsequent models.

This is ideal for scaling because anything built with the right techniques for the smaller size will scale perfectly to the Retina display. However, you will need two complete sets of images: one for the non-Retina displays and one for the Retina displays. Apple's convention is to add @2x to the file names of Retina-sized images, so `myImage.png` would have a corresponding `myImage@2x.png` file.

Thus, if your initial files are at non-Retina sizes, then the Retina images would have to be scaled to 200%.

ANDROID

Android is similar to iOS, except that it has four pixel-density targets instead of two, because Android is used on a huge range of devices. Android caters to pixel densities

of 120 PPI (low density), 160 PPI (medium density), 240 PPI (high density) and 320 PPI (extra-high density). The UIs for all Android devices are at a scale based on one of these four pixel densities.

To support all four densities, you would need a complete set of PNG images for each. Low-density Android devices are uncommon, so you will probably only want to support the other three. If your initial design is at 160 PPI, then you would need to scale those densities to 100%, 150% and 200%.

WINDOWS METRO

Like Android, Windows Metro has been designed to accommodate a wide variety of devices, so multiple sets of images are needed. Windows Metro assets are created at 100%, 140% and 180%, unless scalable vector graphics (SVG) are used.

MAC OS X

Although unannounced, Mac OS X will very likely follow the pattern of iOS and cater to non-Retina and Retina displays with 100% and 200% image sets. Where appropriate, PDF images can also be used for Mac OS X UI elements, with one file covering both sizes.

WEBSITES AND WEB APPS

How different display pixel densities will be handled on the Web in the long term is a little unclear, but the methods employed will likely be very similar to iOS, Android, Windows Metro and other native platforms.

Some websites already switch sets of images based on the display PPI, similar to iOS, Android and Windows Metro. Other designers try to draw everything with code—by using CSS and SVG images or by embedding icons and glyphs in fonts—so that image sets are not required.

One thing is clear: as high-PPI displays become more mainstream, different methods will need to be refined for the Web. Any native design or Web design of the future will need to be built to scale to multiple sizes. So, how exactly do we achieve this in Photoshop?

PHOTOSHOP DOCUMENTS THAT SCALE

When it comes to building elements that scale easily in Photoshop, avoid bitmaps. Bitmaps are, by nature, a grid of square picture elements. This means that detail cannot be added when a bitmap is enlarged because no additional detail exists (extra pixels could be interpolated from neighboring pixels, but that leads to blurriness).

Elements cannot be scaled down, either; doing so creates obvious scaling artefacts and, therefore, lower-quality artwork.

The solution is to build everything using vector shapes and effects that can be regenerated at any size. In Photoshop, that means using solid-color, pattern or gradient layers with vector masks and layer styles. This allows artwork to be scaled up or down and then exported as bitmap images at the fixed pixel densities.



Figure 7.4. Avoid bitmaps and always use vector shapes and effects.

PDFS, SVGS, CSS AND DRAWING WITH CODE

Another approach to handling wildly different resolutions and different pixel densities is to draw everything using vector-based images (such as PDFs or SVGs), or to draw using code, or to build elements with CSS at runtime. These methods work very well in some situations, typically for simple objects. The techniques are also good if the size, color or other properties of the objects need to be changed dynamically.

However, the more complex the objects are, the more resource-hungry they tend to be, which might lead to performance problems. The increased demand on resources can be a considerable problem for mobile devices. Also, using one vector SVG or PDF image for all pixel densities means you won't have pixel-level control over how the result looks at each size, which can be important when scaling to sizes other than exact multiples. An SVG or PDF image scaled by 200% will look crisp, but scaling to 140%, 150% or 180% will likely blur the edges.

But before diving deep into Photoshop techniques, let's prepare our workspace.

Preparing Your Workspace

Let's get started on a new project in Photoshop. You will want the width and height of your document to match the size of the final website or app. In our case, we're designing an iPhone app in portrait orientation, so we're starting with a 320×480 pixel canvas. I've chosen the non-Retina (i.e. smaller) iPhone pixel density as the starting point because I favor building at $1\times$ and then scaling up for the finishing touches and exporting the $2\times$ images—being able to snap to the pixel grid and having sufficient workspace are important to me. This is a personal choice, and you might prefer starting at the larger Retina size and scaling down. Each approach has its pros and cons, and you would need to make a similar decision if you were designing an Android app, a Windows Metro app or a website targeted at high- and low-PPI displays.

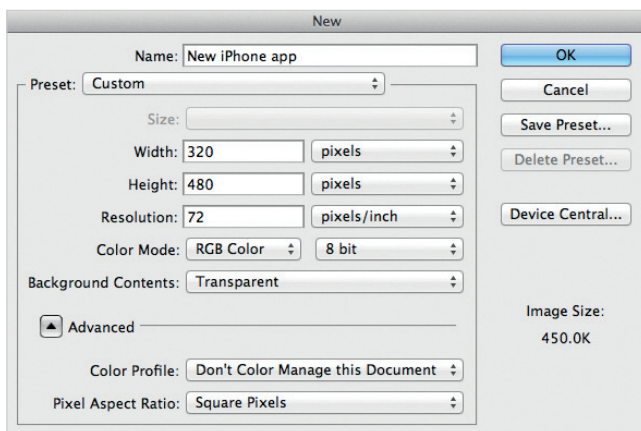


Figure 7.5. Decide whether to start new documents with smaller pixel density and then scale up or vice versa.

PIXEL GRID

Starting at the smaller, $1\times$ size ensures everything you do is locked to the $1\times$ pixel grid. If we started at the $2\times$ size, it would mean that we would have to ensure we only used even positioning, even heights, even widths and even layer style values. If we didn't, uneven values (1, 3, 5, etc.) would land on half pixels (0.5, 1.5, 2.5) when scaled down, resulting in blurry edges or rounding errors.

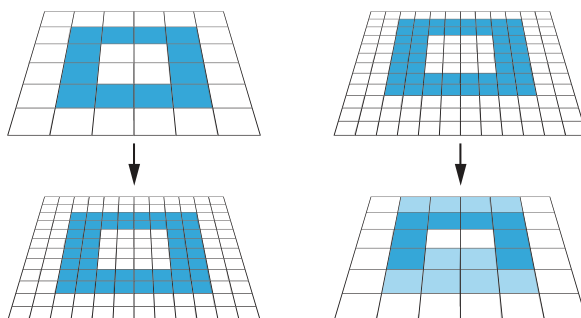


Figure 7.6. Scaling up to exact multiples always works. Scaling down can cause issues.

PREVIEW SIZE

Working at 1× means that a 1:1 pixel preview (where 1 pixel on your computer display represents 1 pixel in your design) on your computer's display will be smaller. In some cases, this is vital because your display may not be big enough to show a full portrait iPhone Retina preview (960 pixels high, plus room to fit the menu bar, window bezels and dock). With a Retina display iPad, the situation only gets worse.

I do not know of many displays that can fit 2048 pixels vertically. The current 27-inch Mac Cinema Display is 1440 pixels high, so even with the largest Apple display available, you wouldn't be able to work on portrait designs for a Retina iPad and see the entire iPad screen uncropped. A Retina iPad has 3,145,728 pixels, just shy of a 27-inch Cinema Display's 3,686,400 pixels. But once high-PPI computer displays are released, preview size will not be an issue.

DOCUMENT PPI

You might have noticed that the new document is set to 72 PPI. This might seem a little counterintuitive, as it does not match the device's pixel density. However, there are good reasons to work like this.

Let's say you have one document at 100 PPI and another at 200 PPI. If you use Copy Layer Style on a layer in the 100 PPI document and Paste Layer Style on a layer in the 200 PPI document, the Layer Style will be scaled. A 1px drop shadow would become a 2px drop shadow.

This sounds like a reasonable thing to do from Photoshop's perspective, but probably not what you're after in most situations. Here's why: if you're being particular about your document PPI, you would probably set up your iPhone documents at 163 PPI or 326 PPI and your iPad documents at 132 PPI and 264 PPI.

If you are working on an app that is for both iPhone and iPad, there's a strong chance you will copy elements and layer styles between your iPhone layout and iPad layout. If you have the document PPIs set to match the devices, your layer styles will get scaled by about 20% each time.

That is not an issue with 1px shadows, as they will be rounded up or down to the nearest pixel, and likely stay the same, but any value larger than about 5px will be scaled and you may be left wondering why elements look subtly different after moving them between documents. The same may be true if you are targeting different Android or Windows Metro devices.

Also, as discussed previously, image pixel density doesn't matter for the Web and apps—it is the pixel dimensions of the final images that count, not the PPI you have set in Photoshop.

Therefore, my strong recommendation is to always set documents to 72 PPI. Doing so will make Photoshop far more predictable.

COLOR PROFILE

Our color profile is set to “Don't Color Manage This Document.” This is deliberate and required if you'd like the colors in Photoshop and Illustrator to match other applications and not shift when exported.

In the print world, color management typically involves calibrating your entire workflow, from scanner or digital camera to computer display to hard proofs to the final press output. This can be a tall order, especially when the devices use different color spaces—matching RGB and CMYK devices is notoriously hard.

When designing or editing for TV, calibrating the main editing display and using a broadcast monitor are common—these will show a real-time proof of how the image might look on a typical TV in a viewer's home.

In these scenarios, color management offers many benefits and is highly recommended.

When building Web and application interfaces, the situation is a little different. The final output will be displayed on the same (or same type of) device that you used to create the artwork: a computer display. There are some complications, though. Even though the device on which you created the Web or app interface will be the same as or similar to the one on which the final product will be used, you will have various sources for color rendering: images (typically PNG, GIF and JPEG), style markup (CSS) and code (JavaScript, Java, HTML, Objective-C, etc.). Getting them all to match can be tricky.

When designing website or app interfaces, we want the colors that are displayed on screen in Photoshop and that are saved in files to perfectly match what is displayed in other applications, including Firefox, Safari and the iOS simulator.

Colors should not shift or appear to shift in any way under any circumstance. Therefore, we do not want Photoshop's in-app color management to alter colors on screen or in saved files.

DISABLING PHOTOSHOP'S RGB COLOR MANAGEMENT

To disable Photoshop's RGB color management, choose **Edit → Color Settings**, and set the working space for RGB to "Monitor RGB." For each document you work on, you will need to ensure that the document color profile is set to "Don't Color Manage This Document." This can be done by choosing **Edit → Assign Profile**, or by configuring the advanced options when creating a new document. If you don't do this for every single document you work on, the colors will appear incorrectly in Photoshop itself.

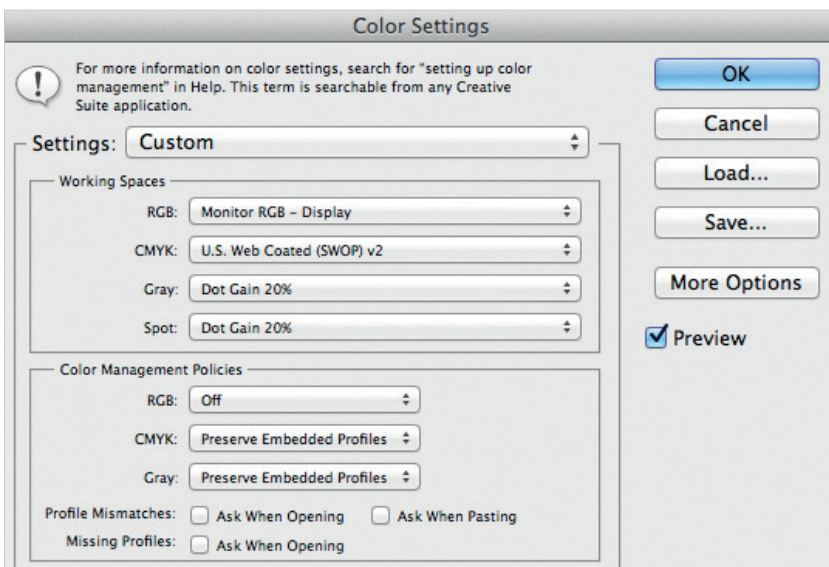


Figure 7.7. Setting Photoshop's RGB Color Management off helps avoid Photoshop's in-app color adjustments on screen or in saved files.

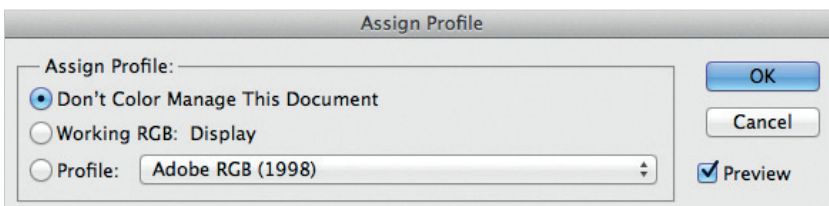


Figure 7.8. With "Assign Profile" as a non-destructive action, you change the way your document appears on screen without effecting the color data.

Every Photoshop document contains a color profile that is separate from the actual color data stored for each pixel. “Assign Profile” simply changes the profile in the document, without affecting any of the color data. The action is non-destructive—you can assign a new color profile to your documents as often as you like without doing any damage. Assigning a new profile might change the way the document appears on screen, but the data contained in the file will be unaltered.

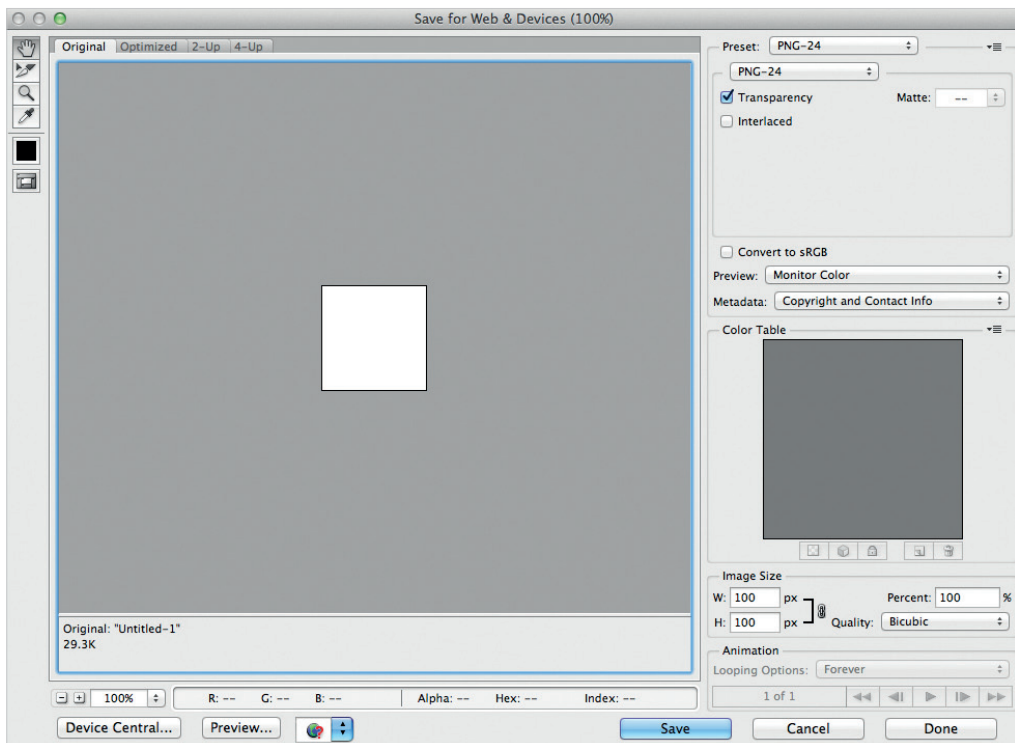


Figure 7.9. When saving files with “Save For Web” in Photoshop, ensure “Convert to sRGB” is turned off, also for saving as JPEG file. Otherwise you will alter and mismatch the color values.

“Convert to Profile” is quite different. It not only assigns a color profile to the document, but tries to keep your image looking the same on screen. It does this by processing the color data contained in the file for each pixel. Converting to a new profile will more likely maintain the way your document appears on screen, but the data contained in the file will be permanently changed. Use with caution.

If you are copying layers from one Photoshop document to another, ensure that both documents have been assigned the same color profile. If they haven't, then color information could be destructively changed as it moves between documents. You will also need to ensure that **View → Proof Colors** is turned off. If enabled, Proof Colors will change the way colors are displayed in your document, meaning that they won't match other applications.

Finally, when saving files with “Save for Web,” ensure that “Convert to sRGB” is turned off. If it is enabled, the image will be converted from the current color profile to the sRGB color profile, thereby altering the color values, destructively changing the file and mismatching colors with colors in code. If you are saving a JPEG file, then also turn off “Embed Color Profile” (you might want this to be turned on for photos in some cases, but chances are you'll want it off for interface elements and icons).

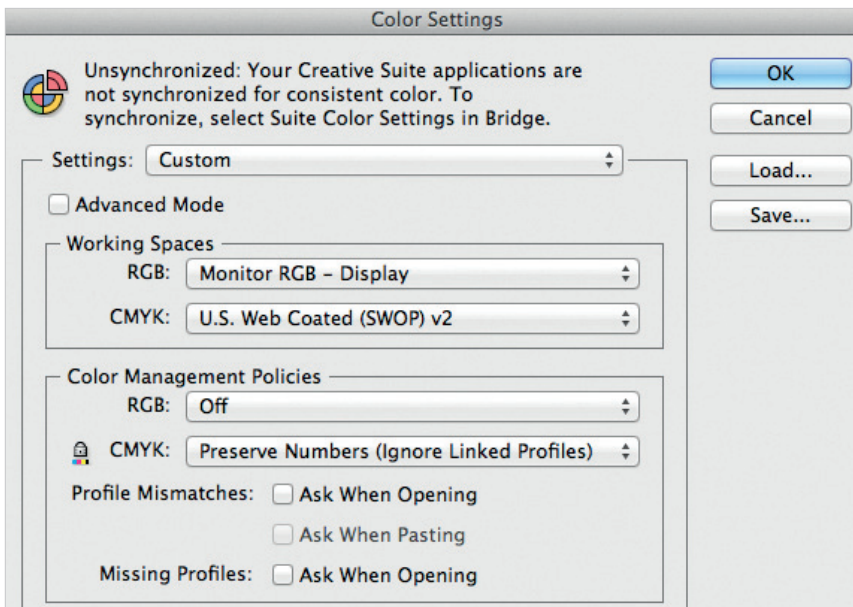


Figure 7.10. When building interfaces always set Illustrator's document color profile to “Don't Color Manage this Document” via **Edit → Assign Profile**.

If you are using Adobe Illustrator together with Photoshop and want the colors to remain consistent when pasting elements between the two, then you will need to set up Illustrator in a similar fashion.

DISABLING ILLUSTRATOR’S RGB COLOR MANAGEMENT

Color management in Illustrator is very similar to Photoshop, as are the settings required for Web and application design. To disable Illustrator’s RGB color management, go to **Edit → Color Settings** and set the working space for RGB to “Monitor RGB.” For each document you work on, you will need to ensure that the document color profile is set to “Don’t Color Manage This Document.” To do this, choose **Edit → Assign Profile**. This must be done for every single document you work on.

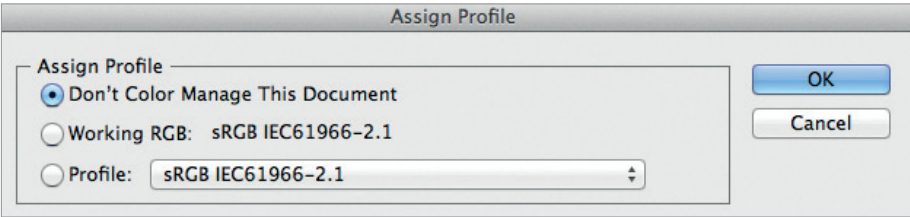


Figure 7.11. Set “Don’t Color Manage This Document” for every single Illustrator-document you work on. Also set **View → Proof Colors** turned off.

You will also need to turn off **View → Proof Colors**. If enabled, Proof Colors will change the way colors are displayed in your document, meaning that they will not match other applications. When saving files with “Save For Web,” ensure that “Convert to sRGB” is turned off.

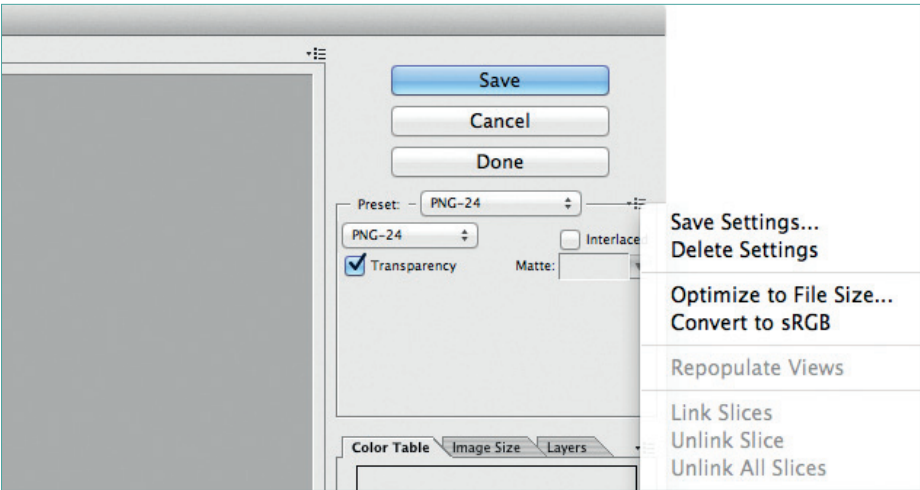


Figure 7.12. When saving files with “Save For Web And Devices” in Illustrator, ensure “Convert to sRGB” is turned off.

Shapes

Photoshop's shape layers are created and edited as vector paths, which means they can be rendered at optimal quality at any size. They are an ideal starting point for scalable, flexible glyphs, icons and interface elements.

Almost any hard-edged shape can be created using a combination of Photoshop's shape tools (Rectangle, Rounded Rectangle, Ellipse, Polygon, Line and Custom Shape).

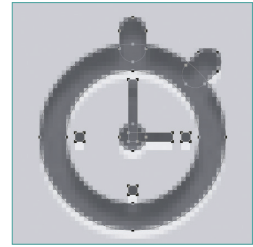


Figure 7.13. The icon above is a single vector layer, created from several paths.

SNAPPING TO THE PIXEL GRID

The Rectangle and Rounded Rectangle tools both have a well-hidden option that allows them to snap a drawing to the pixel grid, ensuring sharp edges on all sides. The “Snap to Pixels” checkbox can be found in the Options bar, usually at the top of the screen. Unfortunately, the Ellipse tool doesn't have the “Snap to Pixel” option. If you need to draw a pixel-snapped circle, the Rounded Rectangle with a large corner radius works well.

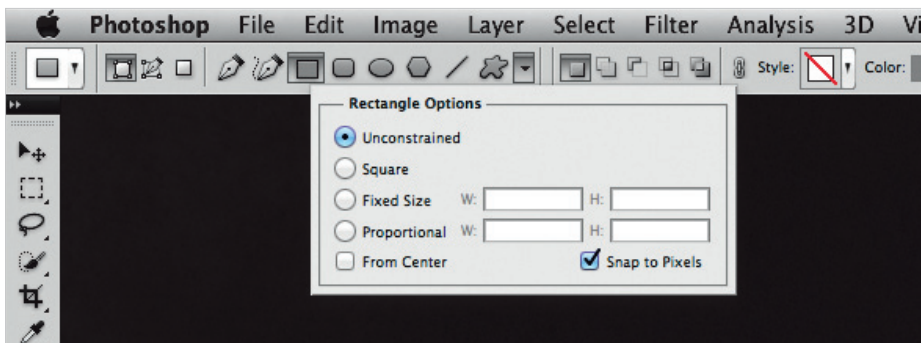
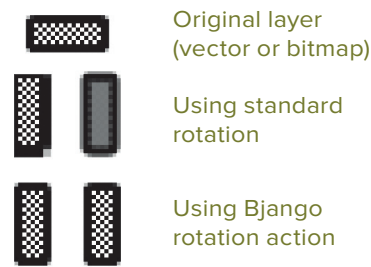


Figure 7.14. Photoshop's Rectangle and Rounded Rectangle tools allow sharp edges on all sides. The “Snap to Pixels” check box can be found in the Options bar, usually at the top of the screen.

FIXING ROTATION ISSUES

Rotating layers in Photoshop by 90 or 270 degrees by selecting either “Free Transform Path,” “Rotate 90° CW” or “Rotate 90° CCW” in the Edit menu can cause problems with vector and bitmap layers. The quality of the outcome will be determined by the artwork's size. If the layer has an even width and height, you'll be fine.

If the width and height are odd, you'll also be OK. But if it is odd by even or even by odd, then you'll see something similar to the results on the right. Changing the origin to the top left, top right, bottom left or bottom right prior to rotating will ensure that everything stays crisp after transforming.



NUDDING POINTS EXACTLY 1 PIXEL

When nudging vector path anchor points, Photoshop can exhibit some strange behavior depending on how far you're zoomed in. At 100%, nudging using the arrow keys will move your vector point exactly 1 pixel. At 200%, nudging moves the point by half a pixel. At 300%, it will move a third of a pixel. If you want pixelperfect vector shapes, you'll probably want to nudge in single-pixel increments, even if you're editing while zoomed in.

We can take advantage of Photoshop nudging however we want, even at 100%. With your document open, choose **Window → Arrange → New Window** to create a second window of the document. You can then resize the new window and place it somewhere out of the way. Edit in the original window as normal, zooming in as far as you'd like.

Figure 7.15. Bypass Photoshop's rotation bug by using Bjango rotation. smashed.by/rotation

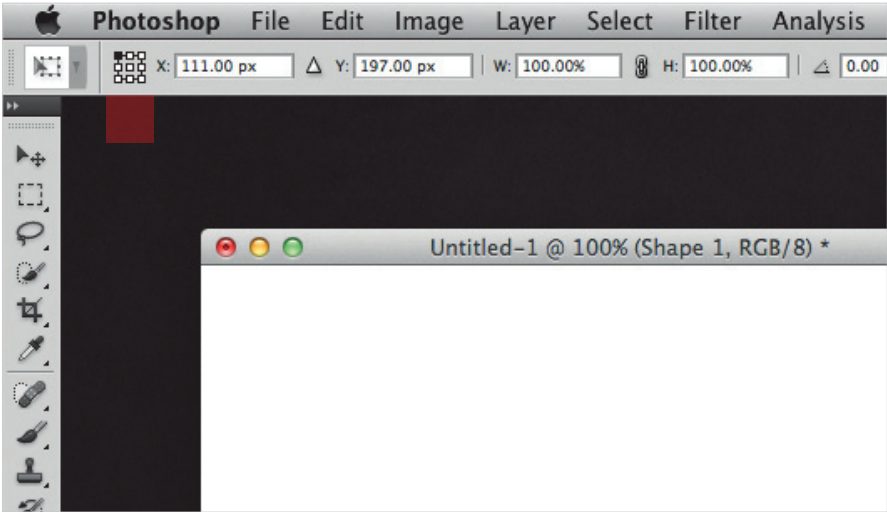


Figure 7.16. Changing the origin to the top left before rotating will ensure that the quality is maintained.

When you need to nudge a point, simply press `Command + `` to switch to the window that's zoomed to 100%, nudge using the arrow keys, and then press `Command + `` to switch back again. Because the other window is zoomed to 100%, nudging will move the selected vector points exactly 1 pixel. It's a little awkward, but far quicker than having to zoom out to 100% and back in again to edit fine details.

However, dragging vector path anchor points with the mouse does snap to the pixel grid. Also, holding `Shift` while using the arrow keys to nudge always moves 10 pixels, no matter how far you are zoomed in.

Shading and Form

Shading and shadows can add body and form to a design, making elements look like they exist in three dimensions. Raised convex elements look like they can be pressed. Lowered concave elements look like they are carved out. Shadows can indicate height, lending structure and hierarchy. These are important hints that show at first glance how a UI functions, playing on our experience with real-world objects and lighting.

However, before we start exploring shading, shadows and other techniques, we need to decide on a light source. UI designs and illustrations commonly appear to be lit from the top of the screen, with parallel light rays. This mimics a typical daylight scene, where the sun is above and far away. It is also similar to typical indoor scenes, where lights are mounted in the ceiling directly above. You might decide to use a different light source or multiple light sources, which is OK, provided that you are consistent and that the entire design follows the same rules.

In both Photoshop and Illustrator, shading is often accomplished with gradients. In Photoshop, this is best achieved using gradient fill layers or gradient layer styles because both can be scaled infinitely. In Illustrator, a gradient fill can be applied to any path.

CONVEX SHADING

Convex shapes bulge outwards, towards the viewer. These are often expressed through linear gradients from light to dark because the light source is directly above. Convex shapes look raised, so they're great for buttons.

CONCAVE SHADING

Concave shapes appear hollowed out or depressed and can be drawn as a linear gradient from dark to light (the opposite of convex shapes). Using a combination of a few shapes with gradient fills, we can create a simple scene with some dimension.

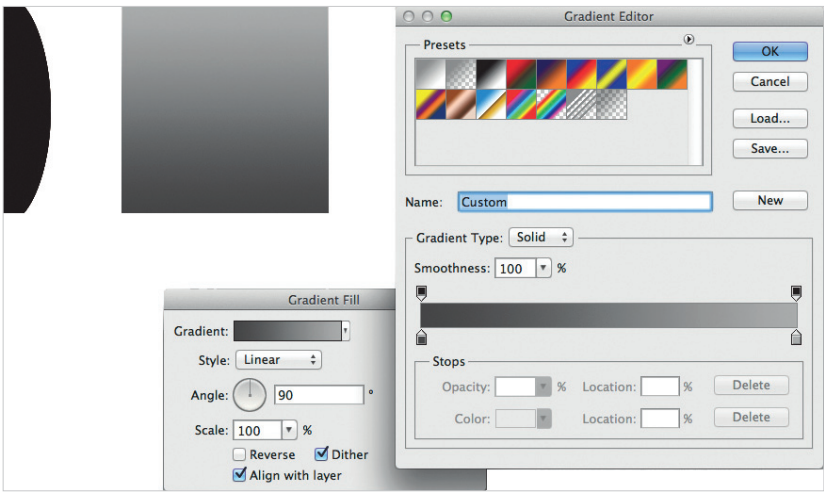


Figure 7.17. Convex shapes bulge outwards, expressed through linear gradients from light to dark. Convex shapes look raised, so they're great for buttons.

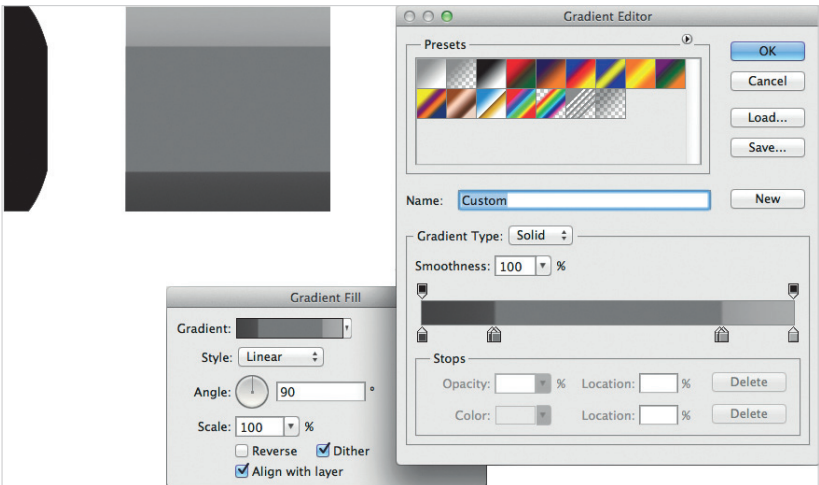


Figure 7.18. By changing the gradient slightly, we can make the shape look like it has a large flat section in the middle. This can also be achieved by adding a second layer with a flat color fill.

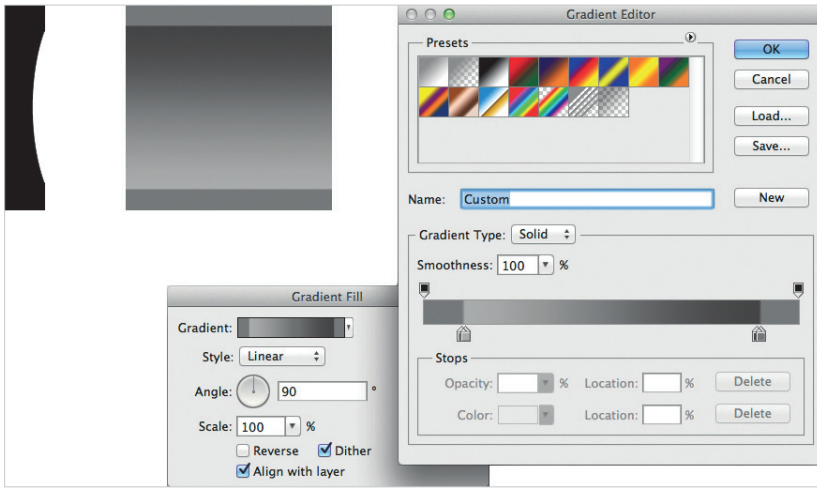


Figure 7.19. Concave shapes can be drawn as a linear gradient from dark to light.



Cross-section (left) Elements with gradient fills (right)

Figure 7.20. The lines at the top and bottom look like channels carved out. Recessed lines are often used as dividers in UIs.

SPHERICAL SHAPES

Spherical shapes are often constructed with radial gradients. Radial gradients start from the center and grow outward in a circular pattern. In Photoshop, the center point of the gradient can be moved by clicking and dragging on the canvas while the gradient window is open (for gradient fills) or when the layer styles window is open (for layer-style gradients).

It is common for spherical shapes to reflect diffused light from a surface below. Replicating this effect is possible with a few minor changes to the gradient. Using photographs and other references might help you fine-tune the amount of contrast needed for the material you are reproducing (see Figure 7.22.).

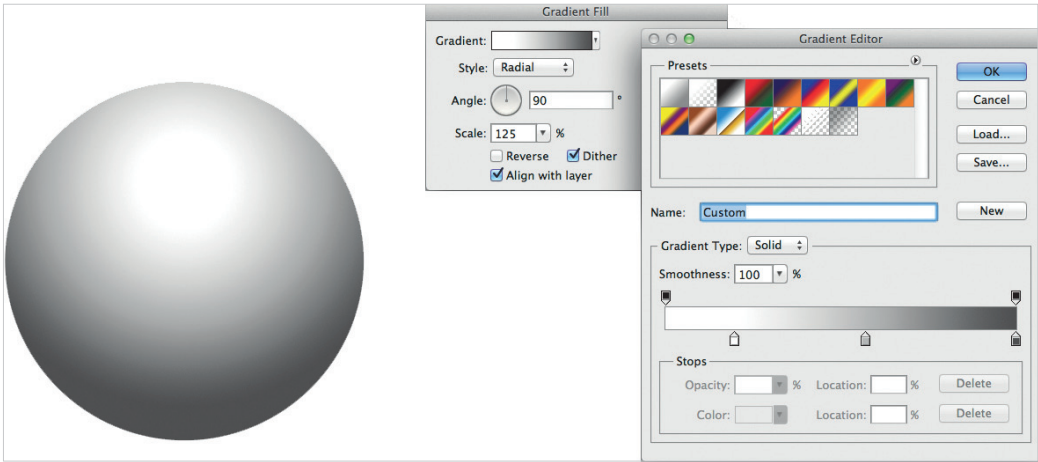


Figure 7.21. For gradient fills, the center point of the gradient can be moved by clicking and dragging on the canvas while the gradient window is open.

REFLECTED GRADIENTS

Reflected gradients in Photoshop contain a linear gradient that is editable, plus a mirrored repeat of the same gradient. Reflected gradients make editing perfectly symmetrical gradients a little less tedious, provided they produce the effect you are trying to achieve.

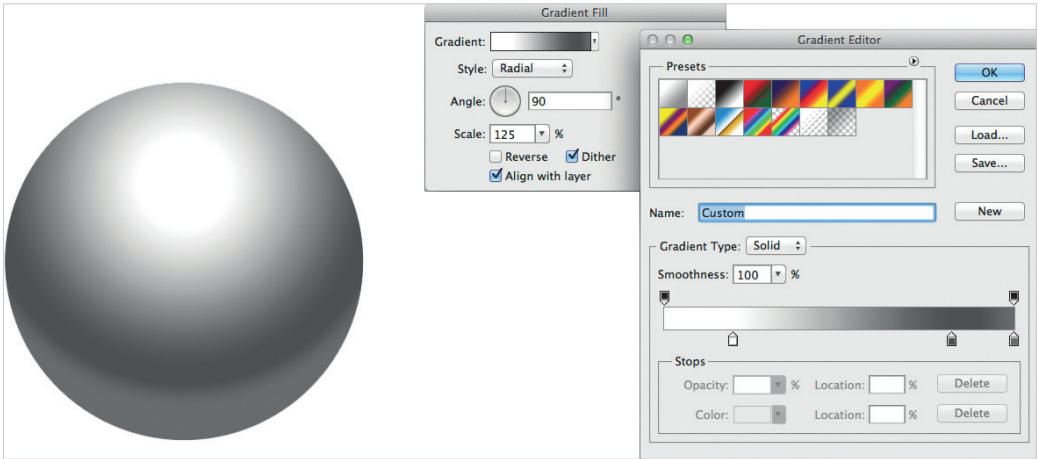


Figure 7.22. The same diffused reflections can exist in other real-world objects. Careful use of light-to-dark-to-light gradients can look spectacular as well as add realism to a design.

ANGLE GRADIENTS

Angle gradients can be a great way to mimic the kind of environmental reflections that are found on highly polished metallic objects. As with radial gradients, the center point of the gradient can be moved in Photoshop by clicking and dragging on the canvas while the gradient window is open (for gradient fills) or when the layer styles window is open (for layer-style gradients).

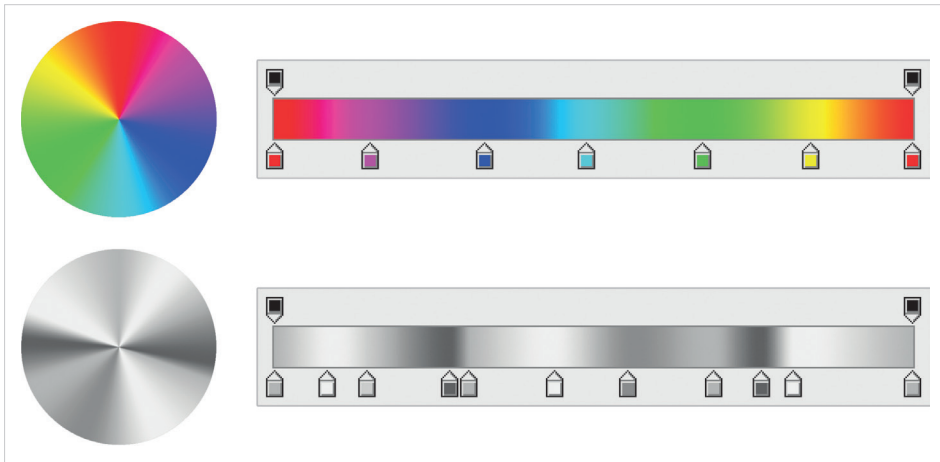


Figure 7.23. The center point of an angle gradient can be moved by clicking and dragging on the canvas while the gradient window is open.

GRADIENTS ON GRADIENTS

Combining a gradient fill layer with a gradient layer style enables you to overlay two different gradients, creating more complex and dynamic results. To combine the gradients, you will need to set a blending mode for the gradient layer style. This will allow the gradient fill layer to appear through the gradient layer style. In the examples below, I've used either Screen (good for lightening) or Multiply (good for darkening).

GRADIENT DITHERING

Adding dithering to a gradient produces smoother results because it adds subtle patterned or random noise. Non-dithered gradients often contain visible banding. Dithering is even more important if your artwork will be viewed on cheap 6-bit-per-channel “twisted nematic” LCDs and certain other display types, which tend to amplify posterization problems. Photoshop can dither gradient fill layers as well as gradients drawn

with the Gradient tool, so turning that option on is recommended. Gradients drawn in Illustrator cannot be dithered, nor can vector Smart Objects that have been pasted into Photoshop from Illustrator.

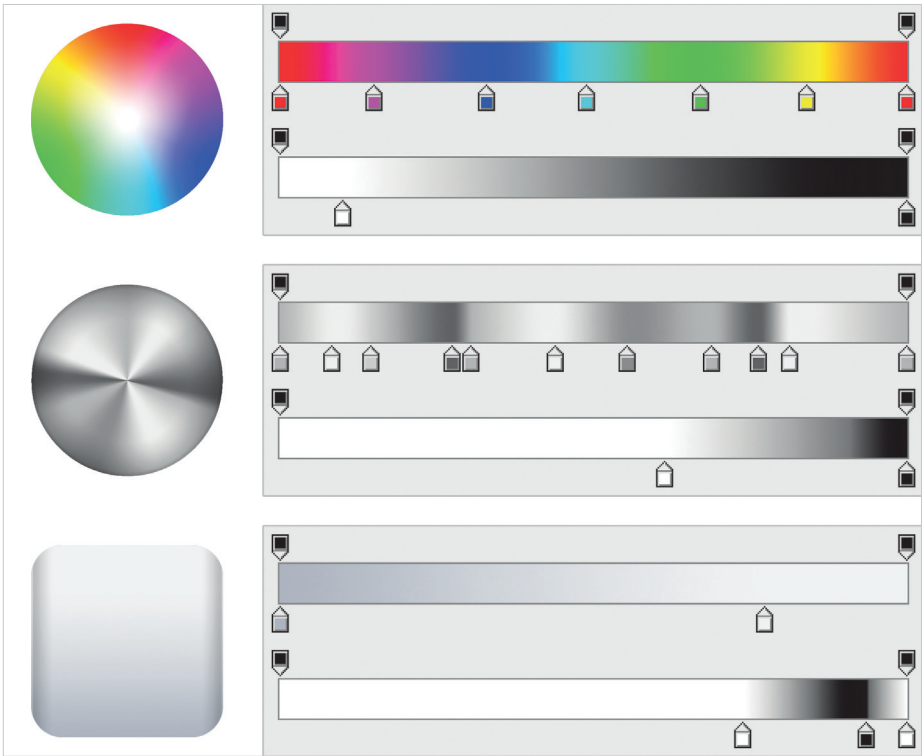


Figure 7.24. To combine the gradients, you will need to set a blending mode for the gradient layer style, e.g. Screen (good for lightening) or Multiply (good for darkening).

If you use transparency as part of a gradient, it will not be dithered either, which sometimes results in visible banding. There is a solution for particular cases: if you are using a gradient with transparency in order to lighten an area with white, then using a non-transparent gradient with a Screen layer blending mode will let you dither it. The same technique can be used for darkening, with the Mul-

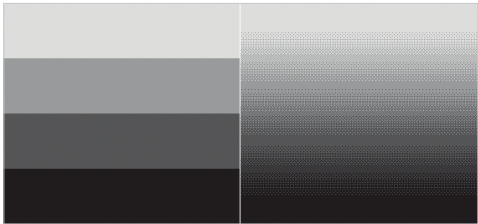


Figure 7.25. A non-dithered gradient (on the left) versus a dithered gradient (on the right). The gradient on the right looks far smoother.

tiply blending mode. Dithering is subtle and often difficult to see. The contrast has been increased in the example on the left to emphasize the dithering pattern. The gradient on the right looks far smoother because it has been dithered.

Textures

In real life, most items have some kind of texture. The texture might be obvious (such as heavy grain in a plank of timber) or subtle (such as the fine pattern of brushed aluminium). Adding texture can be a great way to denote different areas and surfaces, giving elements a more tactile, realistic look.

There is a slight complication, though. Textures are typically bitmap-based, since they need to appear photographic or have a photo-like quality. As we have discussed, bitmap images and bitmap layers do not scale well and should be avoided where possible. Photoshop allows three methods for adding texture that will also scale your documents non-destructively: “Pattern Fill Layers”, “Pattern Layer Styles” and “Smart Objects.”

CREATING A PATTERN

Patterns can be created in Photoshop by using the Marquee Selection tool to select a rectangular area, and then choosing Edit → Define Pattern. Once created, the pattern will be available for use as a Pattern Fill Layer and a Pattern Layer Style pattern.

If you are planning to scale up your document, then you will want to create your pattern texture at the largest size it needs to be.

If you would like precise control over how the pattern is viewed at particular scales, then you will need to create a version for each size required. For example, you might create two patterns for an iOS texture: one for the Retina scale and one for the non-Retina scale. You would then switch the pattern used in your Pattern Fill Layers and Pattern Layer Styles as you export all of your final image assets.

PATTERN FILL LAYERS

Pattern Fill Layers are precisely what their name suggests: they are layers filled with a pattern. They may also optionally contain a vector mask, so that even if the bitmap contained inside the pattern layer is softened by scaling, the edge itself will remain sharp. Double-clicking a Pattern Fill Layer’s thumbnail in the Layers panel opens up the options, allowing you to set the pattern scale independent of the document’s scale.

PATTERN LAYER STYLES

Pattern Layer Styles are similar to Pattern Fills, but they are applied as a layer style. This means they can be used in conjunction with Solid Color Fill Layers, Gradient Fill Layers and even other Pattern Fill Layers.

SMART OBJECTS

In Photoshop, Smart Objects are documents contained within a layer, making them an ideal way to embed a high-resolution texture in a lower-resolution document. Smart Objects are rendered as their original file is rendered. Smart Objects created within Photoshop are rendered at their original size, then scaled up or down using bitmap scaling.

The best way to use Smart Objects in Photoshop is as a rectangular region with a vector mask applied for the shape. This means that the texture itself will be bitmap scaled, but the shape will be redrawn at size as a vector, making for a crisp edge.

To create a Smart Object in Photoshop, right-click or **Control**-click on a layer or group, and choose “Convert to Smart Object.”

Smart Objects created by pasting or importing a file from Illustrator are natively vector, so they can be scaled to any size and will re-render as required. Smart Objects from Illustrator can contain multiple color fills, strokes and features that are not possible with Photoshop’s vector layers. However, some caution is recommended: Smart Objects from Illustrator are not drawn with dithered gradients and so can have anti-aliasing issues.

To create a Smart Object from an Illustrator file, choose **File** → **Place** from Photoshop. Copying objects in Illustrator and then pasting them into Photoshop is also possible.

The Handover

Designers, developers and teams all work differently. Some see design as a process that can happen parallel to development. Others see it as a task best completed prior to any code being written. Where possible, I favor the latter because it allows for fast-paced design iteration without having to change code.

Whichever way you work, you will need to keep in mind quite a few things when handing over the design to the developer or development team. In addition to providing a complete set of images, you will need to consider other aspects.

FONT INFORMATION

Some parts of your design will likely require text to be rendered with code. Send the complete specifications for each instance of type—including font, font size, color and any relevant shadows—to the developer.

Ensure that the fonts used exist on the target devices. For websites and Web apps, you might need to create a font stack, a list that specifies the order of preference for fonts in the event that certain fonts are not available. If a font is substituted, words or phrases might widen, so be sure to test with all of the fonts in your stack.

Also, what is possible in code is usually only a subset of what is possible in Photoshop. A gradient fill or complex layer styles on text in Photoshop might not be easy, possible or desirable in an app or website.

If you are designing for iOS, setting your Photoshop document to 72 PPI will make the font sizes correlate closely to Xcode: 10-point type in Photoshop at the 1× size (320×480 for iPhone) equates to 10-point type in iOS. Please note, however, that there will be some differences in the way text looks, due to rendering differences between Photoshop and iOS.

STRETCHABLE IMAGES

UI images with dynamic widths or heights commonly stretch by repeating a section in the middle. If your stretchable images contain patterns or dithered gradients, then you will probably want to add a note to the developer; 1-pixel-wide repeating textures negate the benefit of dithering and usually look awful because the dithered pattern needs more space to work its magic.

IMAGES AND PIXEL DENSITY

PPI information is almost always ignored when images are used on the Web or as UI elements in iOS, Android, Windows Metro and Mac apps.

However, the pixel dimensions of your images do matter, so do get those right. For iOS, ensure that your 2× images are exactly double the dimensions of your 1× images and that elements within the images are in the same positions; your Retina images should be identical to their smaller counterparts, but with more detail.

Please note that the PNG image format stores its pixel density as pixels per meter (PPM). This can cause rounding errors when values are converted between PPI and PPM. If you've ever seen an image change from 300 to 299.999 PPI when saved as a PNG, this is why.

PNG COMPRESSION AND IOS APPS

At face value, running your images through a PNG compression tool, such as OptiPNG, PNGcrush, AdvPNG or PNGOUT, might seem like a great idea. The tool grinds away, shaving kilobytes or bytes off of each file, hopefully improving the application's downloading and launch speeds.

But to dramatically increase the drawing performance of iOS apps, Xcode recompresses PNG files as it builds. It premultiplies the alpha channel, and it swaps the bytes in the red, green and blue channels to be sequenced as blue, green and red. Xcode then recompresses the images using PNGCrush. The result is optimized for iOS' purpose, but as a side effect, any prior compression gets undone, due to the images being rebuilt.

There are definitely some good tools and reasons for optimizing your images for the Web, but if you are creating a Cocoa iOS app and using Xcode's built-in PNG compression, you gain no advantage.

DEVICE TESTING

If you are designing for mobile, test your final mockups on the target device or devices themselves. The advantages in doing so are significant. You will be able to test ergonomics and tap areas, and you'll see the design in context, with the layout's characteristics in full force, all while having the luxury of being able to make changes.

Testing devices is important because screen types, warmth of colors and even sub-pixel patterns vary greatly, so you might want to tweak the design after seeing everything in situ. Some display types, such as AMOLED, can appear far more saturated and with much higher contrast than typical computer LCDs. Not to mention, seeing your design on the device is exciting.

There are many ways to get your final mockup onto a mobile device. Emailing images to yourself or using services like Dropbox works well, as do more tailored solutions such as LiveView and Skala Preview. If you can, test with a method that maintains image quality: 32-bit PNG images are perfect, but lossy compression such as JPEG could introduce artefacts.

SUB-PIXEL PATTERNS

A display's sub-pixel pattern is the matrix used for the red, green and blue elements on the screen itself. Some sub-pixel patterns—such as the PenTile matrix, used commonly in OLED screens—can cause vertical edges to look different and irregular.

Techniques for Exporting Images From Photoshop

Exporting all of the image assets needed to build a website or app is probably the least interesting part of the design process, usually entailing hours of grinding. Saving images to multiple scales, as required by iOS, Android, Windows Metro and other platforms, adds complication to the task. But there are ways to streamline or automate exporting.

COPY MERGED

Cutting up your design with “Copy Merged” is fairly easy: make the relevant layers visible, draw a Marquee selection around your element, choose **Edit → Copy Merged**, then **File → New**, hit **Return**, and then **Paste**. The result is a new document with your item isolated, trimmed to the absolute smallest possible size. From here, all that’s needed is to save the image using “Save As” or “Save For Web And Devices.”

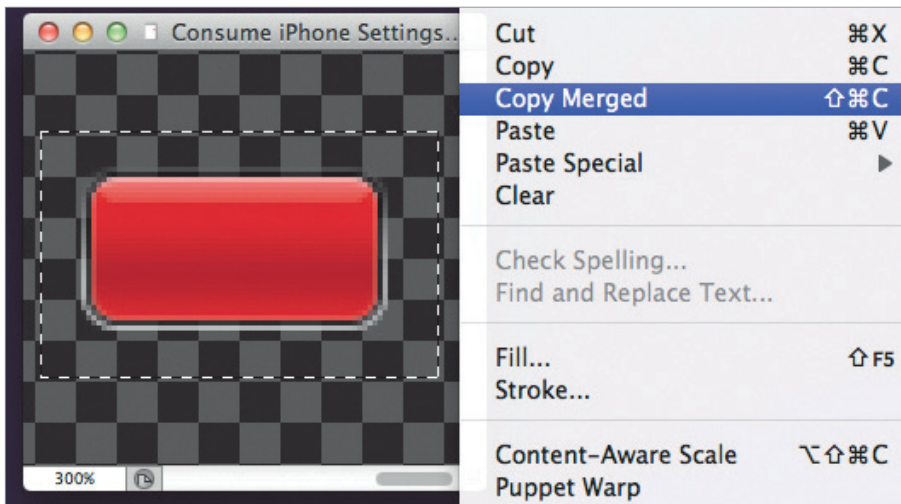


Figure 7.26. Cutting up your design with “Copy Merged” is fairly easy: make the relevant layers visible, draw a Marquee selection around your element, choose **Edit → Copy Merged**, then **File → New**, hit **Return**, and then **Paste**.

Rinse and repeat for every image for your app or website. The technique is simple and quick but requires a lot of repetitive work, and if you ever need to export the images again, you’ll have to start from scratch.

In my experience, this is the most common, and often the only, method that most designers use, which is a shame because better techniques exist.

You could create an action that triggers the Copy Merged, New, Paste sequence, which would be a small time-saver, although not much of an improvement to the workflow.¹

EXPORT LAYERS TO FILES

If you are lucky to be exporting a lot of similar images (typically with identical dimensions), you might be able to use Photoshop's Export Layers to Files script.

By choosing File → Scripts → Export Layers to Files, each layer of your document will be saved as a separate file, with a file name that matches the layer's name. This means you will probably have to prepare your document by flattening down to bitmap layers all of the elements you'd like to export—a time-consuming process, but often quicker than using Copy Merged. This can also trim the resulting file, if you'd like to remove completely transparent areas. Export Layers to Files is handy if the desired result fits its limited range of use cases.

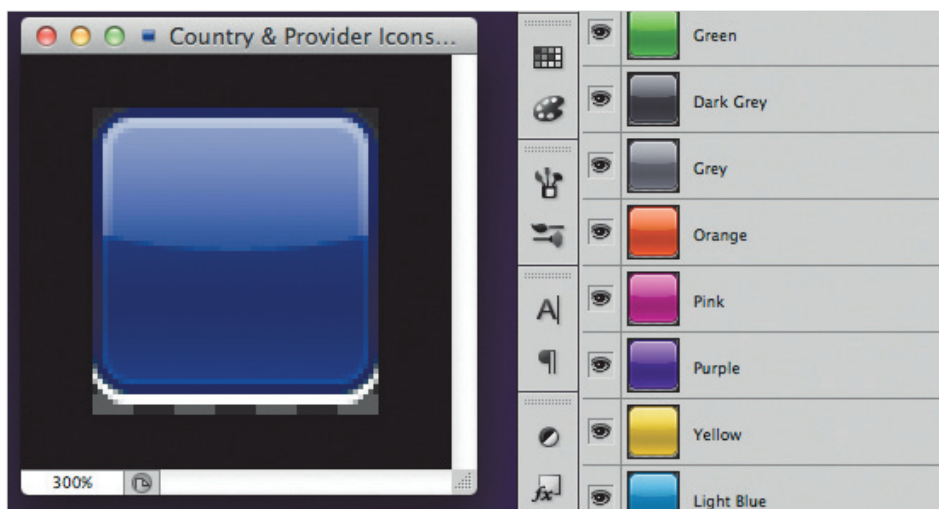


Figure 7.27. By choosing File → Scripts → Export Layers to Files, each layer of your document will be saved as a separate file, with a file name that matches the layer's name.

¹ You can use my action for Copy Merged at smashed.by/copyaction.

SLICES

Photoshop's Slice tool lets you define rectangular areas to export as individual images, with some limitations: only one set of slices can exist per document, and slices may not overlap (if they do, smaller rectangle slices will be formed). During the 90s, the Slice tool was a good way to create table-based Web layouts filled with images. These days, we need finer control over how images are sliced, especially if we want our designs to be efficient and dynamic, with images that have transparency. However, with a twist on the concept, the Slice tool can be put to great use.

SPRITE SHEETS WITH SLICES

Sprite sheets are commonly used in CSS and OpenGL games, where texture atlasing can have significant performance benefits (a texture atlas is a large image that contains small images within it). This can be advantageous for websites because only one image needs to be sent to the Web browser, saving on HTTP requests. A similar performance boost occurs with OpenGL games, where a single file stored in the GPU's memory can be used to reference a lot of smaller images contained within it.

A visually similar method can be employed to export UI elements from Photoshop, even if the result is a set of images rather than one large image.

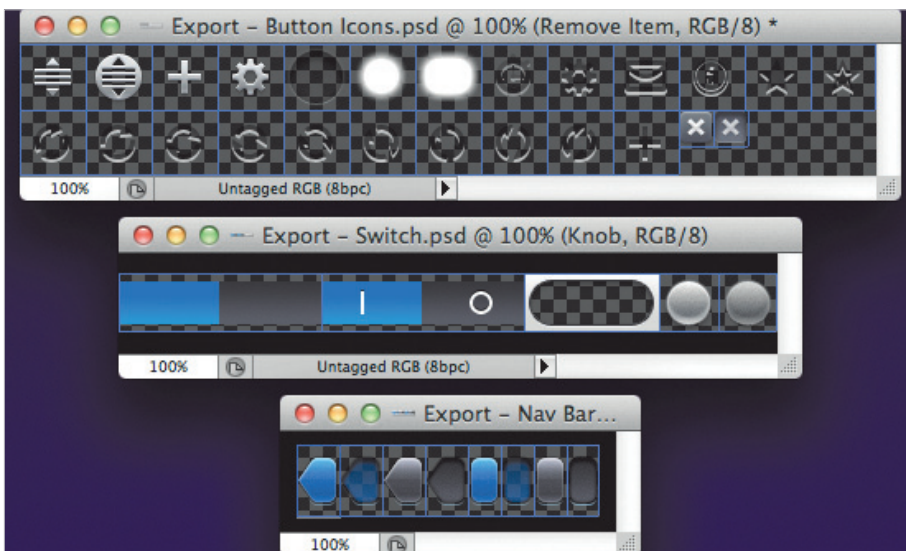


Figure 7.28. Sprite sheets require time to set up, but help automate image exporting.

By spreading out the elements that you need to export as a flat sprite sheet, you negate the need for slices to overlap. If there are too many elements to comfortably fit in one document, then you can create multiple documents, negating the need for more than one set of slices per document.

The bonus to working like this is that you no longer need to build your main design documents with the same level of precision. Occasionally using a bitmap or forgetting to name a layer is fine because you will have a chance to fix things when preparing your sprite sheet to export. But this does mean that your original mockup document could get out of sync with your latest changes to the export documents (if you make color or layer effect changes, for example).

Because we are interested only in user-created slices, you might also want to click “Hide Auto Slices” (in the options bar when using the Slice Select tool) and turn off “Show Slice Numbers” (under “Guides, Grid & Slices” in Preferences) to remove unnecessary clutter from Photoshop’s slices UI. After creating a sprite sheet with slices all set up correctly, you will be able to export all of the images at once, using “Save for Web & Devices.” And assuming you’ve done things properly, you will be able to scale up by 200%, save all of your Retina images, and then add @2x to the file names by batch renaming them (or scale down, if you built everything at Retina size to begin with).

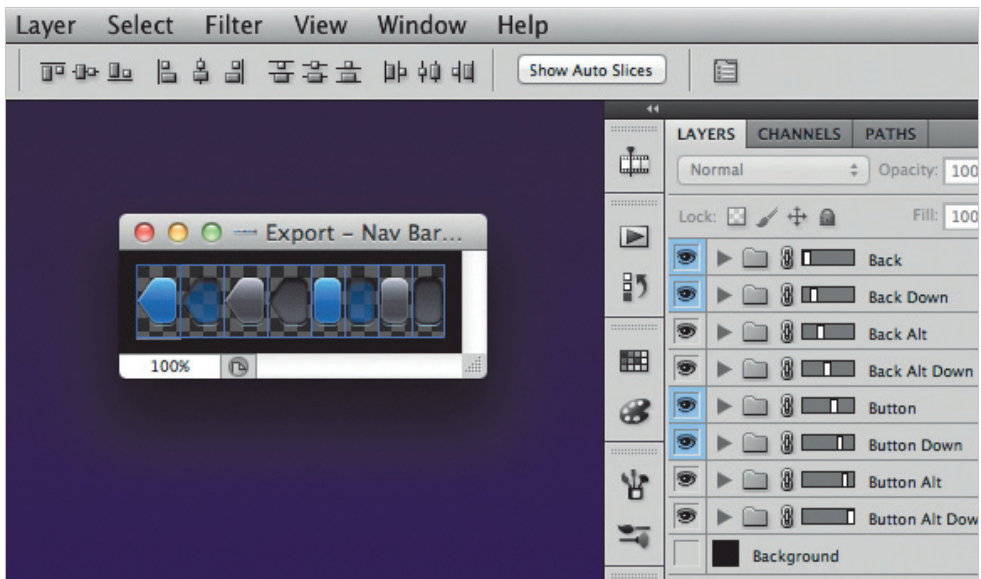


Figure 7.29. After creating a sprite sheet with slices all set up correctly, you will be able to export all of the images at once, using “Save for Web & Devices.”

LAYER BASED SLICES

If your UI element is one layer and you would like the exported image to be as small as possible, you might want to consider using a Layer Based Slice. To create one for the currently selected layer, choose “New Layer Based Slice” from the Layers menu. Layer Based Slices move, grow and shrink with the layer that they’re linked to. They also take into account layer effects: strokes and shadows are included and so increase the size of the Layer Based Slice. Less control, but more automated.

MY EXPORTING WORKFLOW

For years, I used “Copy Merged” as my primary exporting method and used “Export Layers to Files” when doing so made sense. That was a poor choice. Sprite sheets have so many advantages, especially for medium-sized and large projects, that the initial set-up time is made up for very quickly. This is even truer when exporting multiple sets of images for different pixel densities; each set can be exported in a few clicks and is far less likely to have problems with its file name or size, due to the automated process.

It also fosters an environment in which making changes to production assets is easy, allowing for faster iteration and more experimentation. It lowers the barrier to improving your artwork during development and for each revision of your app or website. And that is a very good thing.

Comparison and Adjustments

Having spent time sweating the details, we now have to ensure that the final product matches the mockups from Photoshop. My preferred method of checking the live app or website for errors is to take a screenshot, open it in Photoshop, and then place the original mockup over top, with the green channel removed. This can be done from the Blending Options in the Layer Style window.

This works incredibly well for predominately neutral designs. Where there is a difference, the original mockup will appear green and the live app or website will appear magenta. Spotting the differences, measuring the changes required and sending a note to the developer (or doing the tweaks yourself if you are the developer) is easy.

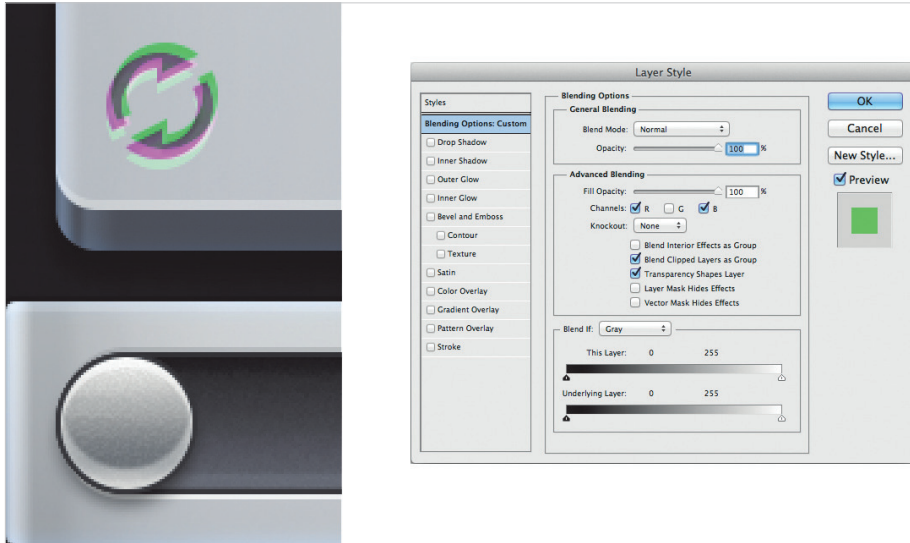


Figure 7.30. Disabling the green channel makes it easy to see the difference between the mock-up and the real app.

As you're comparing, the text in the final product will probably look quite different from the text in the mockups in Photoshop or Illustrator. This is to be expected. iOS, Android, Windows Metro and all Web browsers render text differently, sometimes subtly, sometimes radically.

New Challenges

The world of software and, by extension, software design has always been in flux. New technology and new capabilities bring with them new challenges. But the current list of challenges faced by software designers is as long and complex as it has ever been.

Along with our more familiar creative challenges, we must now face the technical challenges that high-pixel-density displays have created. In a lot of ways, our designs are becoming codified. They're machines; they need to be scaleable and liquid. They also need to retain the human element because humans are who they speak to. I hope this chapter has helped you prepare to meet these new challenges.



About the Author

Marc Edwards (@marcedwards) is the director and lead designer at Bjango, an independent Mac and iOS development company based in Australia. Marc co-hosts the Iterate podcast, occasionally speaks at conferences, and writes design articles for Smashing Magazine and on Bjango's website. (Photo: Matt Adams)



About the Reviewer

Jon Hicks (1972) was born in Leamington Spa, England. Nowadays, he lives in a medium-sized market town just outside of Oxford. It is right on the edge of a beautiful area called The Cotswolds, and it's a great place to bring up the family and go out riding! He loves cycling and bike geekery in general, as well as all things lego and Dr. Who. Jon has been working as a designer for 18 years, 10 of those as a freelancer. He is known for the Firefox, MailChimp and Shopify logos, but he works in a variety of media. Jon has two guinea pigs (Tom Tom and Tufty) and a golden retriever named Olive. His favorite colors are orange and black. The biggest lesson he has learned in his career is to spend time mucking about on small personal projects. If you can, dedicate a half day every week to just experimenting; it always pays dividends. He follows this maxim in life: You reap what you sow.



Redesigning With Personality

Written by Aaron Walter

Reviewed by Denise Jacobs

REDESIGNING A WEBSITE can be the seven-layer taco dip of hell. You've searched for inspiration on dozens of websites, captured screenshots, jotted down notes, consulted friends and colleagues, maybe even interviewed users. But despite your due diligence, your vision for the new website remains unclear.

I feel your pain, my friend. I have been there many times. A redesign brings with it the pressure to innovate, to reimagine, to make a better version of the website so that it lasts for years to come. It can be paralyzing.

Whether the website is for a client or for yourself, if you're struggling to find your way, it's probably because you are starting from the wrong place. The inspiration you seek is not where you think it is. It's not in a blog post entitled "25 Amazingly Beautiful Websites." It's not in your Twitter stream, nor on Facebook. It's not even on the Web. It's right there on your seat. It's you.

Just for a moment, stop thinking about HTML semantics, CSS magic and jQuery tricks. Instead, ask yourself, "Who am I, and what do I want to say?" What do you stand for, what's important to you, and who are you speaking to? Let's make the answers to these questions the trailhead of your redesign journey.

We Web designers have many tantalizing tools at our fingertips, and because the Web is a large community centered on sharing, new ideas and fancy techniques enter our field of vision daily. But in this chapter, I would like to turn your gaze from those shiny objects and focus it on what we're really trying to do with our medium. Our true aim is to communicate clearly and to create human connections.

We achieve that goal not by collecting bells and whistles for our next project, but by discovering who we are and what our message is. The interfaces we design are not walls upon which our users click and tap. They are windows through which we show the world who we really are. As we will see in the principles and examples to come, sharing our personality can help us create lasting relationships with the people who use our websites, and it can improve the bottom line of our business.

Personality will set your brand apart from competitors and help you connect with a passionate audience. Making personality central to the ethos of your redesign might sound scary, especially if you're working with a big corporation accustomed to speaking like the Borg.¹ But even the biggest corporations can communicate with a human voice.

¹ Wikipedia: Borg (Star Trek), smashed.by/borg

Who Are “They”?

Big redesign projects often begin by researching users. We sit down with people to discuss their goals for our website and the expectations they have; we look at demographics, analytics and search logs. It is a lot of data to sift through, but it's not idle footwork. From this research we can create portraits of our archetypal users. This dossier on individuals in our target audience is called a user persona. It answers an important question in the redesign process: who are “they,” the people we're communicating with, and what do they expect of us?

Chances are, if you've spent even a little time working in Web design, you have probably heard of user personas. Maybe you've even created a few. We have been asking ourselves “Who are they?” since Alan Cooper introduced user personas to interaction design in 1995, and they have been a staple of user-centered design ever since. If personas are new territory for you, you will find a concise introduction to the topic in *The Project Guide to UX Design*² by Russ Unger and Carolyn Chandler. If you would like to dig deeper into user research, check out Alan Cooper's industry-changing book *The Inmates Are Running the Asylum*.³

With personas in hand, we have a solid starting point for a redesign. But something is missing. Personas show us only half of what we need to see. Truly effective communication is bidirectional. We now know who “they” are, but who are we? If we share a bit of ourselves in our design, we can not only gain the trust of our audience, but also inspire impassioned users.

PERSONALITY

Lasting relationships center on the unique qualities and perspectives we all possess. We call this amalgam of traits personality. Through our personality, we express the entire gamut of emotions. Personality is the mysterious force that attracts us to certain people and repels us from others. It is like a signpost for compatibility, stirring an emotional response that we cannot ignore.

We have all experienced the magic of meeting someone whose personality captivates us. A chance encounter brings us together, and the magnetism of our personalities keeps us together. Personality helps our brains perform a simple cost-benefit analysis when we meet someone.

² Unger, Russ and Chandler, Carolyn. “A Project Guide to UX Design: For user experience designers in the field or in the making,” New Riders Press, 1st edition, 2009

³ Cooper, Alan. “The Inmates Are Running the Asylum,” Sams - Pearson Education, 1st edition, 2004.

PERSONA

THE INFLUENCER

Julia

Age: 19 - 22; Sophomore; Journalism & Communications

Goals: Get a "Big City College" education; cosmopolitan experience; Build resume with internship; Take new/different courses; Make new/different friends; Experience different cultures

Pain Points: Limited courses offered; Costs; Organization (too much or not enough); Advantages are hidden; Challenging to transfer credits

My internship provided me with the opportunity to work in Times Square. I just love all of the lights, action, and excitement!

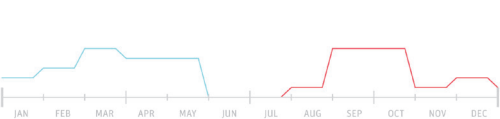
Julia has been taking Spanish since high school and is excited to study abroad in Buenos Aires next spring. She's traveled a little in the past—to Great Britain for a vacation with her family and to Mexico for a missions trip—but this is her first time going abroad alone. Though she has other friends who also plan to study abroad, she wanted to go at a different time so she would be forced to make friends with the locals and truly immerse herself in the culture. She's heard from friends that the maturity level of some of the students plummets the moment they step on the plane to study abroad. She hopes they don't make her look like a "stupid American."

She's also heard that the dorms in Buenos Aires aren't great, which solidified her decision to do a homestay. However, she's concerned about commuting to classes, which she hopes to take at the NYU campus as well as a local university—if the credits transfer. She doesn't have a lot of extra cash and is interested in a work study to pay for souvenirs and some travel around Argentina. Speaking Spanish on the job would also be great practice, but she isn't sure what sort of opportunities there are, or even if she's allowed to work.

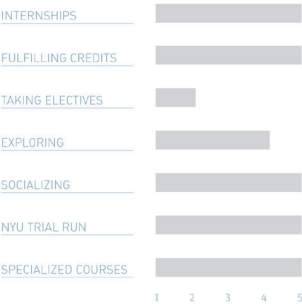
Knowledge



Lifecycle



Activities and Interest



Influencers

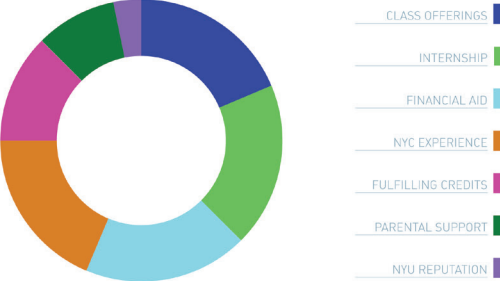


Figure 8.1. A persona paints a portrait of an archetypal user in our target audience and informs our design decisions. This one was created by Todd Zaki Warfel of messagefirst.com.

Though we may not always be conscious of it, we evaluate the world around us with a simple question, “Is this good or bad for me?” Personality provides us with all of the cues we need to determine whether a relationship with a new acquaintance is in our best interest or could be harmful. Because personality plays such an important role in our decision-making process in our social circles, it can be a powerful tool in design as well.

A LONG TIME AGO IN A GALAXY FAR, FAR AWAY

It wasn’t that long ago when those of us publishing on the Web felt compelled to inflate ourselves for the public. Did you have a website in the late 1990s during the dot-com boom? I did, and like so many others, I wrote copy for my websites in the royal “we” to give the impression that I had a corporation as big as iXL or Razorfish (kids, go ask your parents). Throw in stock photography of fictitious partners and meetings, and you’re on your way to transforming a one-man show operating out of a bedroom into a global corporation.

It was a facade I had created because I believed no one would take me seriously if I was honest. I wasn’t the only one duping my audience. At the time, most small businesses and freelancers were painting a picture of corporate grandeur when there was nothing but one or two individuals behind the curtain.

Things are different these days. We’ve opened up our lives to the world on Twitter, Facebook and other social media. We no longer maintain separate personas for our personal and professional lives. The line that once separated them is now blurred, and we now offer one authentic personality for the world to experience, for better or worse.

Because we’ve opened ourselves up to one another, we have come to expect the same of the brands we interact with. In a consumption-centric world where products are pushed on us and companies are always taking instead of giving, we crave real human interaction that is reciprocal and respectful.

A few companies have figured this out and are forging emotional connections with their customers by sharing their personality. We have now come to expect this of small startups, such as Carbonmade,⁴ a zany Web app that helps designers create a portfolio, and Photojojo,⁵ a playground for photographers. These companies are free of corporate constraints and brimming with youthful creativity. Their design and marketing are filled with wit that sets them apart from competitors. But personality is a powerful design tool that works even with gigantic conglomerates.

⁴ smashed.by/carbon

⁵ smashed.by/jojo



Figure 8.2. Carbonmade and Photojojo both have distinct personalities that set them apart from competitors and attract impassioned users.

General Electric, a Fortune 100 company with 287,000 employees (making it, um, 36 times bigger than the town where I grew up), dwarfs nearly every company on the planet. Most corporate behemoths like to market themselves with generic copy, stock photography and whitewashed promises, but GE is trying something different. It is telling stories through the eyes of its employees about how it is changing the world.

In a video on GE’s home page,⁶ workers from Durham, North Carolina, share with us a glimpse of the work they do to create massive jet engines that lift commercial airplanes 3,000 feet in the air. The video is not about the mind-boggling complexity of the technology that goes into the engines, though, but a collection of personal stories about Seth, Mark, Kareem, Tom and their colleagues putting their hearts into their work. The stories express their love for what they do. It is clear that this small team of individuals goes to work each day not just for a paycheck, but because they feel a sense of responsibility and because they care for the passengers that their engines carry.

⁶ ge.com



Figure 8.3. GE shares personal stories on its home page from the individuals who make its products. The company’s personality shines through in individuals who have a passion for their work.

As the video concludes, the team members are taken to a runway where they get to see a jet outfitted with their engines take flight. All stand rapt as the jet makes its ascent.

Tears spill from the eyes of one man who is overcome with pride. The honesty in their message and their earnest delivery make it hard for viewers not to feel emotionally engaged and inspired.

Products Are People, Too

Personality determines how we express emotions and the degree to which we do it. It’s the framework within which we share jokes, select our circle of friends and even find a mate. It is at the heart of all human interaction.

Steven Pinker, Johnstone Family Professor in the Department of Psychology at Harvard University, points out in his bestselling book *How the Mind Works*,⁷ “Much of the variation in personality—about fifty percent—has genetic causes.” That’s right: the moment you enter this world, half of your personality is already predetermined.

⁷ Pinker, Steven. “How the Mind Works,” W. W. Norton & Company, Reissue edition, 2009.

The other half is primarily shaped by social and cultural influences. Only about 5% of your personality is influenced by your parents' nurturing. As a dad, I find this painfully depressing. The fact that so much of our personality is genetic indicates the extent to which it shapes our lives and ensures our survival.

So what if the websites, products and services we design could be imbued with personality? Personality is a natural interface that is familiar to humans. We already know how to respond to people we meet based on cues from their tone of voice, language, appearance and posture. We process this information subconsciously and behave accordingly. If we detect courteousness and trustworthiness in a person, then we might share more of ourselves and linger in conversation. If a person is rude or suspect, we are likely to make excuses for a sudden departure.

The cues we naturally take from people's personalities can also come through in design. Color, type, imagery, copy and interaction patterns can all serve as channels for a personality. Just as our personality influences the behavior of the people around us, personality in design can shape the behavior of visitors to our websites.

There are four key benefits to expressing your personality in design:

1. In a crowded market, personality helps distinguish you from competitors.
2. Personality elicits an emotional response from the audience that encourages long-term memory of your brand.
3. Personality attracts those who get you and deters those who don't.
4. Personality impassions users, who will become your most powerful marketing channel.

Let's look at each of these in detail.

ONE OF THESE WEBSITES IS NOT LIKE THE OTHERS

No matter what kind of website you're publishing, dozens of others like it are on the Web. Put yourself in your audience's position. How will they distinguish your website from all of the others? What makes yours different? The travel-booking company Hipmunk⁸ carefully considered these questions before launching into a very competitive marketplace. Travelocity, Orbitz, Expedia and several others have had a tight grip on online travel booking for some time, but Hipmunk has managed to stand out from the crowd.

⁸ smashed.by/hipmunk

Visit a few well-known travel websites and you'll see common traits among them that express something about their personalities. Advertisements for last-minute travel deals and money-back guarantees litter their home pages, each distracting users from the central goal of the page, which is to get them to book a flight. Each of these elements is asking something of users. "Gimme, gimme!" they scream.

I hate making travel plans because it's such a stressful experience. Coordinating schedules, figuring out time constraints, paying high fees for basic services, and fearing the airline will once again screw something up—all leaving me frazzled and gun-shy.

This emotional state is not helped by the attention-assaulting design of most travel websites. A calm, focused personality is the best medicine for a stressful situation. That is exactly what Hipmunk offers on its website. Its home page is squarely focused on one thing: choosing a flight. A cheery chipmunk adds much-needed levity to an otherwise stressful interaction. No discount offers distract users or add to their stress.

Hipmunk's search results are equally as focused, showing an empathy for users that is unseen in any competitor. Instead of simply listing flights by time and airline, it presents an "Agony" index, showing users which flights will be the most painful. Flights with early departures, late arrivals or long layovers rank higher in the Agony index. One could argue that this feature is simply a clever design pattern, but some-

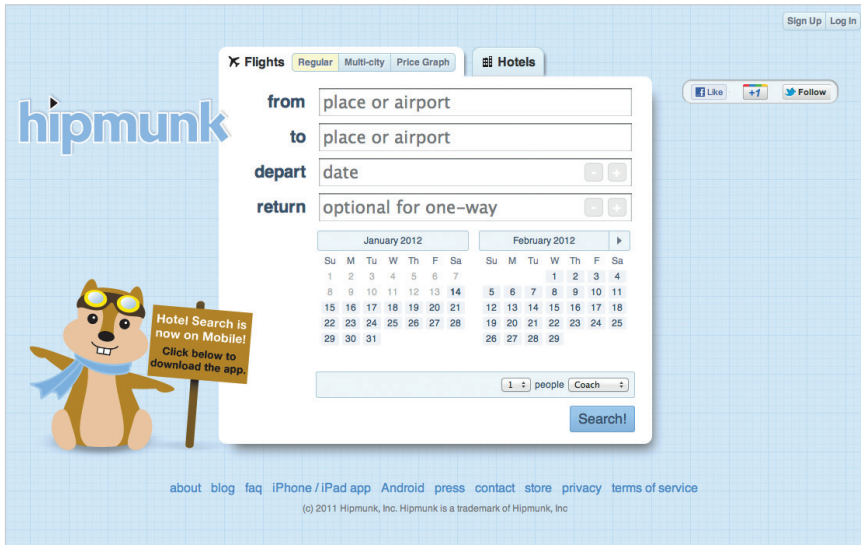


Figure 8.4. Hipmunk's cheery personality and focus on one task are the perfect remedy for the stressed-out emotional state of so many users of travel-booking services.

thing more is going on. The Agony index literally communicates to users that Hipmunk is sympathetic: it feels your pain.

Very few websites possess these traits. Just as an act of kindness on the street says something about the individual who extends it, sympathetic interaction design can convey a designer's compassion for their audience. It's the sort of thing users will not soon forget.

I Remember

Emotional experiences make a profound imprint on our long-term memory. Both the generation of emotion and the recording of memories happen in the limbic system, a collection of glands and structures under all of the folded gray matter of the brain. There is a good reason why the limbic system unites these essential functions. The brain couples emotion and long-term memory because, otherwise, humans would be doomed to repeat negative experiences and would not be able to consciously repeat positive experiences. As John Medina explains in his book *Brain Rules: 12 Principles of Surviving and Thriving at Work, Home and School*,⁹ our brains take note of emotionally charged events:

“The amygdala is chock-full of the neurotransmitter dopamine, and it uses dopamine the way an office assistant uses Post-It notes. When the brain detects an emotionally charged event, the amygdala releases dopamine into the system. Because dopamine greatly aids memory and information processing, you could say the Post-It note reads “Remember this!” Getting the brain to put a chemical Post-It note on a given piece of information means that information is going to be more robustly processed. It is what every teacher, parent, and ad executive wants.”

Experiences that lack an emotional charge tend to fade from memory. Thus, conservative, familiar designs are likely to be forgotten. Medina eloquently sums up the main reason why we designers should employ personality and emotion in design: “The brain doesn't pay attention to boring things.” When we design with personality, we are building a framework through which emotional experiences will remain in the memories of our users.

I'll be honest: there is some risk in designing with personality. Not everyone will like the result. But if you design to please everyone, you will please no one. As we will see in the next section, an interesting dichotomy of positive and negative emotions is elicited by the expression of personality.

⁹ Medina, John. “Brain Rules: 12 Principles for Surviving and Thriving at Work, Home, and School,” Pear Press, Reprint edition, 2009.

I LOVE YOU, I HATE YOU

It's OK if some people hate your redesign and do not connect with the personality you're sharing. That's a sign that you are indeed engaging with your audience on an emotional level. Disdain is always better than apathy.

Showing personality in your design always carries some risk. Some people will feel very connected to it, while others will be turned off. When you share a bit of yourself with the world, someone is not going to like you. But that's fine. The people who are turned off by your personality are not the people you want to court. They are the ones who would cause the most problems in your product support queue or who would constantly insist that you change your product into something it isn't.

If you're a freelancer or agency on the hunt for new projects, you can ward off clients from hell by expressing your personality on your website. The people who are turned off by your website will not want to work with you—and I can tell you from experience, that's not a bad thing!

Even the design process we typically follow for a project reminds us that we are not designing for everyone. The reason we do user research is to find out who we're actually designing for. If the goal was to design for everyone, research would be unnecessary.

In our personal lives, we have all encountered individuals with whom we just don't see eye to eye. Although painful to accept, some people not liking our personality is perfectly fine. It is a fool's game to try to cater to the desires of every person we encounter. The best we can do is be ourselves and trust that some people on this planet will accept us as we are. (OMG! Did we just sneak a life lesson into a Web design book?)

This lesson holds true with design. Personality will help you filter the audience down to those with whom you share common values, interests and goals. These folks are your passionate users. They are the ones to cater to, and they will express their love for your brand openly.

As the user experience lead at MailChimp, I've seen this first hand. The humor, bon-mots and good times I experience every day with my colleagues is visible in the copy, illustrations and interaction design of the stuff we make. The quips of Freddie von Chimpenheimer IV, our chimp mascot, that sit atop each page of the app are collected from people in the company, providing a snapshot of our sense of humor. When you interact with MailChimp, you interact with the people who make it.

We have heard from a few people that the humor woven into the interface is distracting or annoying. When you put your personality out there, that kind of feedback can sting.

We were concerned that these perceptions were widely held by our users, so we did some research. We created an option called Party Pooper mode, which disables Freddie’s jokes and buttons up the informal copy throughout the app. After tracking its usage, we discovered that only 0.007% of our customers actually turned off our personality. The lesson learned was that a bit of criticism is no reason to stop being yourself.

We have found that expressing our personality in the things we make has overwhelmingly greater benefit than risk. It distinguishes our brand from competitors; it helps our customers remember their experiences with us; and it makes many of our customers fall in love with our products. Oh, and it doesn’t exactly hurt your marketing budget either, as we will see in the next section.

YOUR USERS HAVE A WAY WITH WORDS

Personality in design will help spread the word about your website, no ad campaign required. Wufoo,¹⁰ creator of a Web app that makes it easy to build Web forms and to manage the data they collect, eliminated its marketing budget entirely when it discovered that users market the app for it.

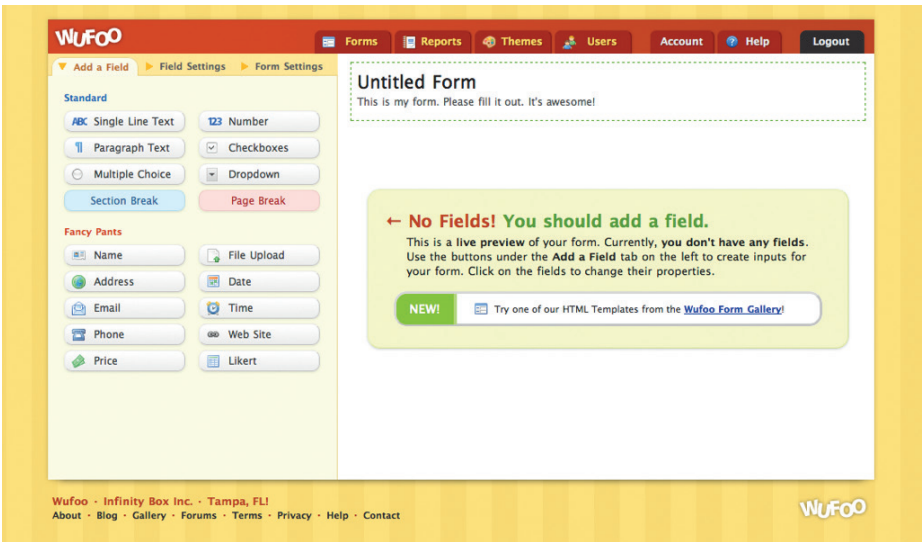


Figure 8.5. Wufoo’s unique approach to designing a business-focused Web app has elicited admiration and joy in customers, who are anxious to tell the world about their positive experiences.

¹⁰ smashed.by/wufoo

Tweets, blog posts and word of mouth brought in new users more effectively than banner ads. Its youthful design, witty copy and occasional rogue dinosaur set Wufoo in stark contrast to its competitors, which are much more reluctant to express personality. When using Wufoo, we can easily see the personalities of the people who made it.¹¹

Having a product that is beloved by users and that grows constantly by word of mouth made Wufoo an attractive acquisition for SurveyMonkey,¹² which purchased it in 2011 for an estimated \$35 million.

PERSONALITY: NOT JUST FOR DESIGNERS ANYMORE

Respected user experience designers such as Stephen Anderson and Andy Budd have been thinking and talking about the connections between personality, emotion and design for a few years now. The topic has gained traction with designers. But we are not the only ones who recognize the power of personality in a crowded marketplace. The people who fund startups are getting it, too.

On his blog, Fred Wilson, venture capitalist and principle of Union Square Ventures, advocates for clarity of voice and personality in product design, suggesting that the two are “critical to building a successful product.”¹³

Dave McClure, angel investor and founder of the business incubator 500 Startups, is of the same mind. He sees personality and emotion as important factors in the success of a product. In his talk at the Warm Gun conference in 2011, McClure told designers and entrepreneurs, “Be yourself, your super-self. What can you be authentic about and comfortable with? Find an essence that you think is you and amplify it. Find a feeling or emotion that you can sustain.”¹⁴

Business news outlets are exploring the role of personality in design, too. Forbes contributor Anthony Wing Kosner attests that personality can make a website more memorable:

*Why are some companies’ websites more memorable than others? On the surface, it might seem to have to do with originality, visual impact and branding. But what if I were to tell you that the most important factor is how a site makes a visitor feel?*¹⁵

¹¹ As mentioned in Dmitry Fadeyev’s chapter, Wufoo goes beyond the usual to make its personality really stand out by sending out handwritten thank-you cards to their best customers.

¹² smashed.by/monkey

¹³ Wilson, Fred. “Minimum Viable Personality,” smashed.by/mvp

¹⁴ Wroblewski, Luke. “Warm Gun: Designing for Emotion,” smashed.by/warmgun

¹⁵ Forbes, “Why Does Emotional Design Work on the Web,” smashed.by/forbes

Wilson, McClure and Kosner are all describing something we saw earlier in this chapter. GE made its giant corporation feel more human by sharing the individual personalities of its workforce. It showed us the passion and pride it has for its craft, and we felt it.

The informal personality of Hipmunk's website allays the stress and negativity that so many travellers feel when booking a flight. The Agony index literally ranks results based on emotional response, which not only helps customers make a decision, but makes them grateful for Hipmunk's empathy.

And Wufoo eliminated its entire marketing budget after realizing that passionate users were spreading the word for it. The personality in its app brightens the day of thousands of workers who carry out the data-collection orders of their corporate overlords while festering in gray cubicles. The levity, color and personality of Wufoo are to users as water is to a desert wanderer.

Once upon a time, personality and emotion in design were a novelty of small-scale websites powered by one or two creative individuals. Today, we are seeing personality being carefully infused into the websites of many brands, small, gigantic and everything in between. Designers are not the only ones who see the value in making the user experience more human; investors now recognize that personality is a key factor in the success of a product.

However, starting a redesign by thinking about personality can be nebulous. Clearly defining the traits of your personality before jumping into Photoshop or Illustrator would be more helpful. That is exactly what a design persona is for.

DEFINING YOUR PERSONALITY WITH A DESIGN PERSONA

Earlier in the chapter we talked about the user persona, which is like a dossier of your archetypal user. It answers the question, "Who are they?" A design persona flips that question on its head, asking, "Who are we?" Both user and design personas set the parameters for the design process. They help us overcome blank-canvas syndrome.

With so many possibilities, where do we start? By understanding both your users and yourself, the options are no longer vast, and the direction is clearer.

Think about it: if your website were a person, who would it be? Would the person be a serious, buttoned-up, all-business type, yet trustworthy and capable? Or a wise-cracking buddy who makes mundane tasks fun? A design persona is a document that outlines the key traits of the personality you wish to convey in a design. We'll look at a real-world example of a design persona momentarily, but let's first look at the structure of the document.

A design persona has nine parts:¹⁶

1. Brand name

The name of your company or product.

2. Overview

A short overview of your brand's personality. What makes it unique?

3. Personality image

This is an actual picture of a person who embodies the traits you wish to convey. This makes the personality less abstract. Pick a famous person or someone with whom your team is familiar. If your brand already has a mascot or representative that does this, use it. Describe the attributes of the mascot that communicate the brand's personality.

4. Brand traits

List five to seven traits that best describe your brand, along with one trait you want to avoid. This will help those who design and write the website to construct a consistent personality, while avoiding traits that would take your brand in the wrong direction.

5. Personality map

We can map this personality on a graph. The x axis ranges from unfriendly to friendly, and the y axis ranges from submissive to dominant.¹⁷

6. Voice

If your brand could talk, how would it speak? What would it say? Would it speak in a folksy vernacular or a refined style? Describe the particular aspects of your brand's voice and how it might change in various situations. People change their language and tone to fit the situation, and so should your brand.

7. Copy examples

Provide examples of copy that might be used in different scenarios on your website. This will help the writers understand how the design should communicate.

8. Visual lexicon

If you are creating this document for yourself as the designer or for a design team, develop a visual lexicon that summarizes the colors, typography and

¹⁶ Download a template for user personae from smashed.by/persona.

¹⁷ To learn more about personality mapping and the research behind it, see "Emotional Design With A.C.T.: Part 1" by Trevor van Gorp on Boxes and Arrows: smashed.by/emodesign.

visual style of your brand's personality. You can be general in concept or include a mood board.

9. Engagement methods

Describe the methods by which you might emotionally engage users in order to create a memorable experience. Stephen Anderson's Mental Notes card deck is a handy collection of such methods.¹⁸

CREATING YOUR DESIGN PERSONA

The process of creating a design persona is as valuable as the document itself. When you stop to consider the traits you want in the design, you gain clarity of what you wouldn't have had, had you jumped straight into your favorite design app. Defining a personality through a team brainstorming session will help the writers, designers, information architects and developers think about your website as a person and recognize the boundaries they are working within.

To get started on your design persona, download the template and sample files first.¹⁹ If you're working on a team, gather everyone together with some snacks and a whiteboard to work through the initial ideas of the persona. It'll be fun—and a cooler full of adult beverages wouldn't hurt either.

Start the discussion by asking, "What seven words best describe who we are?" Be honest. You might hear some traits mentioned that you're not proud of. Now tweak the question: "What seven words best describe who we hope to become with this redesign?" Redesigns are inherently aspirational, and being specific about your aspirations early on is helpful.

Now look for overlap between the two lists. Discuss how you might go about transforming the traits you don't like into some of the aspirational traits. If the remaining list contains more than seven traits, continue whittling it down to seven or fewer, because a personality can become diluted and insincere if you try to be everything to everyone.

With the final list of traits scribbled on the whiteboard, reflect on the boundaries of each trait. What don't you want your personality to become? For instance, if you have listed "fun" as a trait, that can mean a lot of things. Is it fun like Tickle Me Elmo, or fun like driving a Ferrari California along the winding Blue Ridge Parkway? Boundaries add clarity to a design persona and will help you see when a line has been crossed during the project.

¹⁸ Anderson, Stephen P. "Mental Notes," smashed.by/notes.

¹⁹ smashed.by/persona

To get a better sense of how traits and boundaries work, let's look at the ones we defined for MailChimp's persona:

1. **Fun** but not childish,
2. **Funny** but not goofy,
3. **Powerful** but not complicated,
4. **Hip** but not alienating,
5. **Easy** but not simplistic,
6. **Trustworthy** but not stodgy,
7. **Informal** but not sloppy.

A little list like this one is a value system. It tells you who you are, guides your voice and helps shape the audience's perception of your brand.

Now that you know what personality traits you want and don't want, can you think of a person—either a celebrity or someone you know—who could serve as a common reference point for you and your team? Putting a face to these traits will make it even easier during the design process to answer the question, “What would my persona do in this situation?”

Write an overview of this personality to flesh it out further. Describe the voice of the personality, and write out examples of copy to illustrate. How would the personality manifest itself in color, typography and visual style? At the end of this exercise, you will have learned a lot about your starting point for the design.

When creating a personality for your website, keep one important rule in mind. Make the personality an honest reflection of the company you're working for or yourself. You will have a hard time staying true to the design persona if the personality is a stretch. In all of the examples we saw earlier in this chapter, the personalities of the people who created the websites were evident. They were not idealistic concoctions.

We can tell in an instant when a person is pretending to be someone they aren't, and the same holds true in design. The whole point of conveying personality in design is to share more of ourselves so that we can forge meaningful connections with other people. Faking it won't work; not only is it difficult to do, but you'll miss out on connecting with your ideal audience.

When we created our design persona at MailChimp, we were simply documenting the collective personality of the people on our teams. Actually, we found that the

exercise wasn't difficult because we had hired like-minded people with similar senses of humor and complementary personalities. Although our primary goal in putting together a team was to find people who would work well together, the process turned out to be helpful for defining the brand's personality, too. Here's the design persona we created for ourselves:²⁰

Brand name

MailChimp

Overview

Freddie von Chimpenheimer IV is the face of MailChimp and the embodiment of the brand's personality. Freddie's stout frame communicates the power of the application, and his on-the-go pose lets people know that this brand means business.

Freddie always has a kind smile that welcomes users and makes them feel at home. The cartoon style communicates that the brand offers a fun and informal experience. Yes, he's a cartoon ape, but Freddie is still cool. He likes to crack witty jokes, but when the situation is serious, the funny business stops.

MailChimp often surprises users with a fun Easter egg or a link to a gut-busting YouTube video. Fun is around every corner, but never gets in the way of the workflow.

Brand traits

Fun but not childish. Funny but not goofy. Powerful but not complicated. Hip but not alienating. Easy but not simplistic. Trustworthy but not stodgy. Informal but not sloppy.

Voice

MailChimp's voice is familiar, friendly and—above all—human. The personalities of the people behind the brand shine through honestly. MailChimp cracks jokes (ones you can share with your mom), tells stories and communicates in the folksy voice you might use with an old friend.

MailChimp uses contractions like “don't” instead of “do not” because that's how real humans speak to one another. MailChimp uses casual interjections, like “Hmm” when he's thinking hard, and “Blech! That's awful” to communicate empathy.

²⁰ You can download this example from smashed.by/mailchimp.



Figure 8.6. The design persona that was created to guide the personality of MailChimp.

Copy examples

- **Success message:** “High fives! Your list has been imported.”
- **Error message:** “Oops! Looks like you forgot to enter an email address.”
- **Critical failure:** “One of our servers is temporarily down. Our engineers are already on the case and will have it back online shortly. Thanks for your patience.”

Visual lexicon

- **Color:** MailChimp’s bright yet slightly desaturated palette conveys fun—although not to the point of being Romper Roomy. In line with the brand’s traits, the colors convey humor and yet are powerful and refined.
- **Typography:** MailChimp is easygoing, efficient and simple to use, and its typography reflects this. Simple sans-serif headings and body copy vary appropriately in scale, weight and color to communicate information hierarchy, making MailChimp feel like a familiar, comfortable cardigan that is both functional and beloved.
- **General style:** Interface elements are flat and simple, keeping everything easy to understand and unintimidating. Soft, subtle textures may appear in spots to warm up the space and make it feel human. Freddie should be used sparingly, and only to inject a bit of humor. Freddie never gives application-related feedback or statistics, nor does he help with tasks.

Engagement methods

- **Surprise and delight:** Themed log-in screens commemorate holidays, cultural events and beloved figures in history. Easter eggs create unexpected moments of humor, sometimes conveying nostalgia or referencing kitschy pop culture.
- **Anticipation:** Freddie's random funny greetings at the top of each main page create anticipation for the next page. These greetings do not provide information or feedback; they are a fun layer that never interferes with functionality or usability. We created this design persona primarily to guide the design of the application. It helped us dial in the voice we wanted for important elements such as success and error messages, and it helped us reconsider small design elements that were starting to deviate from our personality.

The design persona is not a style guide. It's not intended to dictate how a logo can or cannot be used. It contains no color or typography specifications. It is simply a summary of the spirit of a design. When creating a design persona, you discover so much about yourself and your colleagues, which you can then share with your audience. It's not a document to be policed, but rather a compass to keep you pointed in the right direction.

With personality on our minds, we started to recognize its presence in everything we were creating, especially our writing. Blog posts, product copy, knowledge-base articles and support guides all convey our personality. We didn't always do a great job of it, though. We were injecting humor at inappropriate times. When a user is stressed out because they can't figure out a workflow that they desperately need to complete in order to meet a deadline, a joke or informal tone is not welcome.



Figure 8.7. Freddie von Chimpenheimer IV, MailChimp's mascot.

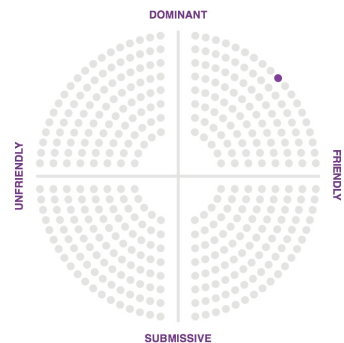


Figure 8.8. MailChimp's personality map.

While the design persona helped to establish our voice, we found that further clarity was needed. Our content curator, Kate Kiefer-Lee, discovered that although a brand's voice must remain consistent in design and writing, the tone should change to match the emotional context of users. Determining the relationship between voice and tone is tricky, but Kate found an interesting solution.

DEFINING VOICE AND ADAPTING TONE

Have you ever tried to write a style guide that sets the design or writing standards for a company? Style guides identify a myriad of scenarios in which design or copy might get mangled, and they provide a framework within which everyone on a team can stay on the same path.

Many redesigns start with a well-intentioned style guide, but more often than not it's abandoned because having to constantly refer to it is impractical, and it makes people feel like the creativity police are peering over their shoulder. Kate Kiefer-Lee penned the content style guide for us to help current and incoming writers, and she had some reservations about how colleagues would receive it. Would people actually use it, or would it be seen as a burden? Having guidance on grammar and punctuation was handy, but the essence of what she wanted to communicate to the team was the relationship between voice and tone.

As Kate combed through blog posts, the knowledge base, microcopy, tutorials and guides to take stock of the company's voice, she noticed areas where MailChimp's voice was clear but the tone was off.

Would you tell a joke while comforting a widow at a funeral? Would you use informal language when meeting the Queen of England? I hope not! You'd certainly want to be yourself, but you would adjust your tone to suit the occasion.

In the design persona laid out above, a few examples of copy were included to show the language style and, to some degree, the variance in tone for different situations. Error messages in an application are the wrong place for a joke because the user is likely confused or stressed at the time. But a success message is a great place to employ humor or an informal tone because the user has just had a positive experience.

Rather than write a formal style guide to govern communication, Kate decided to take a different approach, one that would make understanding the voice of the brand and the variations in tone easy and fun. With the help of some colleagues, Kate created Voice & Tone,²¹ an interactive guide that ties tone of voice to the emotional state of readers.

²¹ smashed.by/voicetone

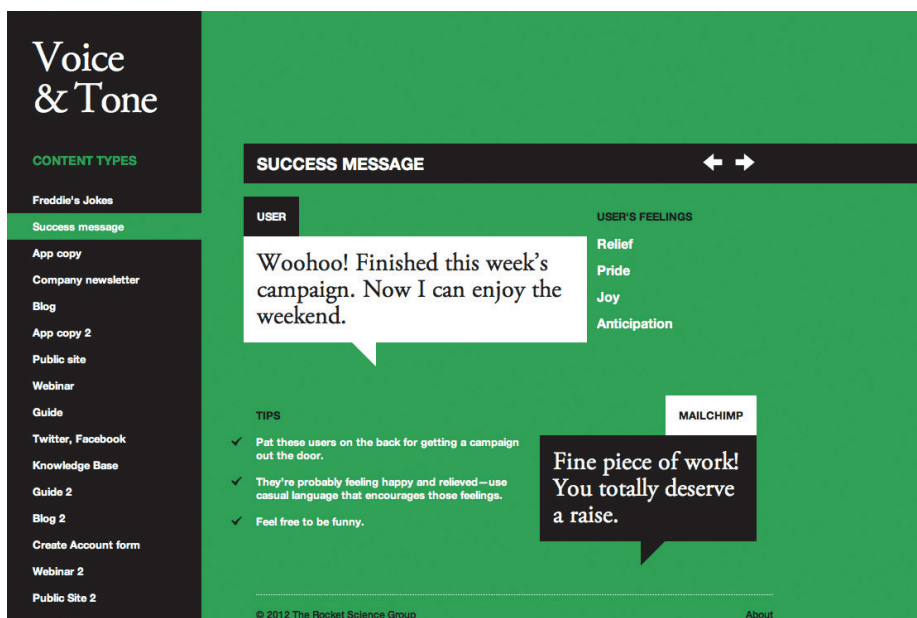


Figure 8.9. Voice & Tone helps anyone who writes for MailChimp maintain the voice and personality of the brand while adapting the tone to match the reader's emotional state.

This approach to defining personality and writing style has been well received; it's easy for people to understand, and it's fun for people to use. It's not full of rules and regulations. No one wants to be governed by the personality police. This guide gives people the freedom to write using the MailChimp voice while guiding tone in the right direction.

Kate sees a close connection between voice and personality: Voice is closely connected to personality. It's who we are. It's our perspective and it's what we bring to every piece of content our customers are reading. Voice is from our end. It's all about us. And tone is something we adapt to match our readers' feelings.

The examples of copy on Voice & Tone not only identify the feelings of readers in particular situations, but are accompanied by matching colors on the page's background. A red background indicates anger or frustration in the reader, while green signifies delight or joy. Like any style guide, Voice & Tone helps authors establish and maintain consistency, an often-preached brand trait that fosters customer trust and strengthens relationships. But it also suggests that adaptability is important; rather than being dogmatic about style, it simply illustrates the spectrum of communication styles that writers can use in various situations.²²

²² Check out the bonus content for this chapter: smashed.by/bonus.

Conclusion

So often in our industry, the motivation for starting a project is to use a new technology or technique. While expanding our skill set is lovely, technique is not the reason we fell in love with this medium. The real reward of our work is the human connections we build on the Web with our fellow designers and the people we design for. Sharing more of ourselves on our websites amplifies those connections.

We've seen many examples of personality and many methods of cultivating it in this chapter. Hipmunk expresses empathy for its users with its Agony index. Wufoo managed to eliminate its marketing budget by sharing its personality in its interface and letting customers spread the word. MailChimp shaped its personality through design personas and an interactive writing guide.

These examples are shared here not to suggest that you do the same in your next project, but to spark new ideas in your work and to help you see the power of personality in design. A design that conveys a clear personality stands out in a crowded market. It elicits an emotional response from the audience that fosters long-term memory of your brand. It attracts the people you want and deters those who will be burdensome. And it impassions those users who will be your most powerful marketing channel.

Make the starting point of your next redesign this simple question, "Who am I, and what do I want to say?" Once you've answered that, then design elements and a writing style will emerge that bring out the personality at the heart of your work.



About the Author

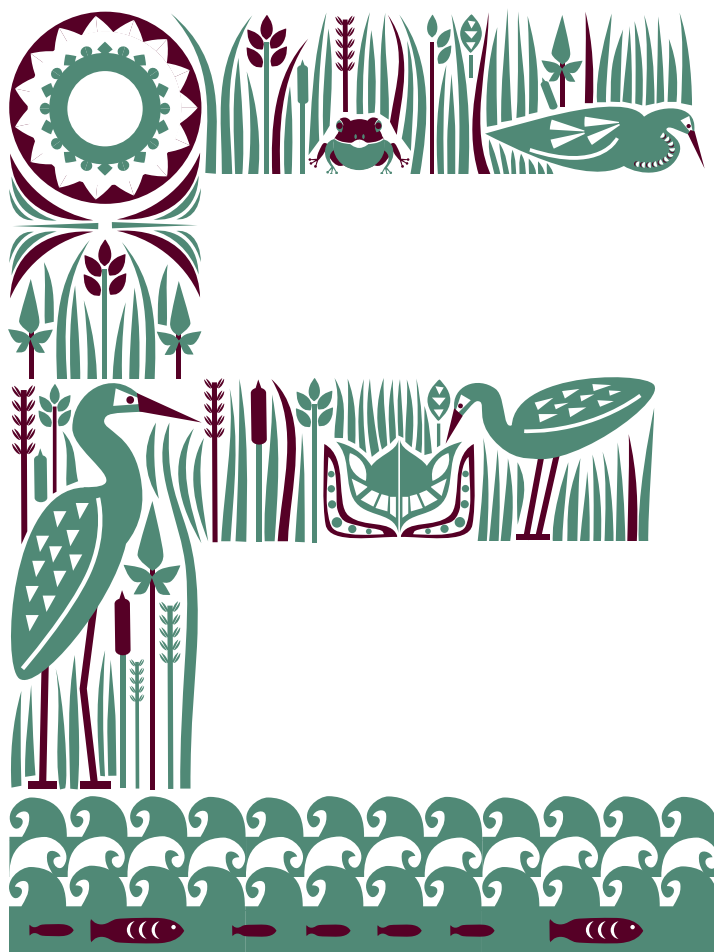
Aaron Walter was born after the dinosaurs but before the Web in a town not far from where Captain Kirk will be born in 200 years. His education includes a BFA degree in painting from the University of Iowa and a MFA graduate degree in painting from Tyler School of Art, Temple University. He lives in Athens, Georgia, where the music is as beautiful as the people. Aaron is the leader of the User Experience Design team at MailChimp. He has written two books, the latest of which is *Designing for Emotion* (2011). Before joining MailChimp in 2008, he spent eight years teaching interactive design courses at a few colleges around the US. His favorite color is black, and he dreams of becoming a barista when he grows up. As advice to readers, he claims that trying is always better than wondering “What if?”



About the Reviewer

Denise Jacobs (1968) was born in Springfield, Ohio, and graduated from Stanford University, the University of Washington in Seattle and Université de Paul-Valéry, Montpellier 3 in France. The motto of her life is, “Feel the fear and do it anyway.” Denise lives in a medium-sized home close to Coral Gables in Miami, Florida. She had almost completely landscaped her backyard in 2009—but then started writing a book. She’s never been able to get the garden back to its former glory since then, but she does have banana trees, and her neighbor’s avocado tree is abundant.

Denise speaks at conferences and writes books and articles. She designs stuff and likes to teach people stuff. Denise loves organic gardening, making jewelry, dancing (especially samba and salsa), improv comedy and acting, self-improvement and reading. The most important lesson Denise has learned is that you just can’t do everything. Trying to be super-human is vastly overrated, and there is more to life than working. Her personal message to readers is to follow your passions and your dreams, and dream big. It will change your life for the better.



Mobile Considerations in User Experience Design: “Web or Native?”

Written by Aral Balkan

Reviewed by Josh Clark and Anders M. Andersen

AS YOU PROBABLY KNOW, user experience design is the discipline concerned with all aspects of the design of interactive products. Although it incorporates important elements of graphic design and motion design, it is primarily concerned with the design of interaction. Its closest cousin in the design realm is product design. As user experience designers, we design virtual products. Furthermore, since hardware design and software design are so intrinsically linked and inseparable, the line that separates product design from interaction design—if it exists at all—is a faint one.

A WEB DESIGNER IS A USER EXPERIENCE DESIGNER

Essentially, a Web designer is a user experience designer with specialized knowledge of the medium of the World Wide Web. The materials of a Web designer are the core (HTML, CSS, JavaScript) and auxiliary (LESS, Stylus, etc.; HAML, Jade, etc.; jQuery, MooTools, etc.) frameworks of the Web and the components within those frameworks. These frameworks and the components within them are made of code. It is this code that determines the design limits and behavior of these materials.

Just as an automotive designer must have intimate knowledge of the various materials that go into making a car, so too must a Web designer understand the materials that go into a website or application.

As interaction designers, we are interested not simply in the aesthetics of an interactive object but in how it behaves. This is especially important when you are designing applications (which are behavior-based objects) as opposed to designing documents (which are content-based objects).

Designing an application is as much about drawing a pretty picture of an application as designing a car is about drawing a pretty picture of a car.

DESIGNING DOCUMENTS VS. DESIGNING APPLICATIONS

Even designing interactive documents well—especially in a responsive manner for the Web—requires specialized knowledge. At a bare minimum, it involves an understanding of responsive design principles and progressive enhancement. Drawing pretty pictures, on the other hand, is art, not design.

Interactive products or applications, however, are a completely different ball game. Designing for an interactive medium requires skills in graphic design, motion design

and, most importantly, interaction design. The most important aspect of an interactive product is its interactions. These interactions are constructed in code.

Unfortunately, due to increased specialization on teams, the role of Web designer and Web developer has been artificially separated. While such a separation may be necessary when working on a team, these labels should define the current tasks of the team members, not sandbox their knowledge. You might focus on certain areas more—especially in particular projects—but you must understand that the primary reason we build products is to satisfy user needs and that every role on a development team has an effect on the user experience. This is why working in small interdisciplinary teams is imperative, where every member is responsible for always thinking of the user first.

DESIGNING FOR THE USER FIRST

When building a product, design leads development and development informs design. This is a cyclical, iterative process in which the goal is to continually improve the product to better meet the needs of users.¹

Every decision you make for your product should stem from the user. You must think of the user’s needs first, before considering your own needs. In other words, practice what we call “outside-in design.” Think about the user’s needs and their context, design what the user will see and interact with, and then go about deciding how to solve all the problems that creates for you.

Outside In Is Good, Inside Out Is Bad.

Is your first question in a new project which server-side technology you will use or what your database schema will look like? Stop! This is a wrong approach. You’re trying to solve your own problems, not the user’s. That’s inside-out design, and that’s a Very Bad Thing.TM

PURPOSE OF THIS CHAPTER

Two of the core decisions you will need to make during the design process is whether to make the product cross-platform and whether it should be native.

The purpose of this chapter is to empower you, as a user experience designer, to understand your medium so that you can answer these questions informedly, starting by looking at what exactly we mean by an application being “native.”

¹ To learn more about working this way, read up on agile methodologies and user-centered product development. The merging of these two worlds—that is, adding sufficient design and user testing to every iteration in an agile process—is what I called *user-centered agile product development* back in 2003: smashed.by/cfe



Figure 9.1. Yes, a huge number of devices are out there. No, your application does not have to support all of them. Focusing on the user means identifying your target audience and optimizing your applications for the platforms and devices they use. Supporting a large number of platforms via progressive enhancement is easier if you are primarily building content-based websites as opposed to behavior-based applications. (Image: David Jones, smashed.by/davidjones)

What is “Native”?

Have you noticed how people throw around the term “native” willy-nilly without grasping what it really means? Let’s change that, starting with what native is *not*.

NATIVE IS (AND IS NOT) ONES AND ZEROS

If we are going to be pedantic, “native”—insofar as digital devices are concerned—refers to the absence or presence of electric current in the transistors that power our computing devices. We usually visualize this as the basic cliché of digital computing: binary code, a series of ones and zeros. We call these binary instructions machine language.

While it is true that computers were once programmed in binary using switches, we no longer write applications at a level that is so close to the metal. However, every other computer programming language we have devised—assembly language, C, Python, JavaScript, etc.—is eventually translated into the ones and zeros of machine language. These are further translated into the presence or absence of electric current in transistors.

Each of these technologies is built upon layers of abstraction. Python is written in C, for example. The purpose of each level of abstraction—each higher layer in the layer cake of technologies that comprise the modern computing ecosystem—is to make it easier for developers to author applications. So, although technically correct, using “native” to mean programming in binary is a rather meaningless definition in today’s world.

So, now that we know what native is not, let’s figure out what it is.



Figure 9.2. Web technologies can be the native technologies for certain operating systems. Here we have a Samsung laptop running Chrome OS, on which HTML, CSS and JavaScript—and Web applications—are first-class citizens. (Image: Google’s promotional image.)

Native as Culture

“Native” refers to the technologies—i.e. languages and frameworks—that form the culture, language, conventions and norms of a platform. It is the base level of abstraction that comprises the core symbols, gestures and interactions that users employ to communicate with applications on a given platform. These elements are of utmost importance because they constitute the culture and norms of a platform.² They are the

² These norms are usually expressed in interface guidelines for the platforms, such as the human interface guidelines that exist for the Mac, iPhone, iPad and Android platforms. (Android is the odd one out here since it is not really a single platform but has many flavors, each customized by device manufacturers and mobile carriers. This is why it is very difficult for Google to exercise control over the user experience of the Android platform. Good user experience is a function of control—and Google has very little control over the user experience of phones based on the Android platform.)

language—both visual and behavioral—that users learn to communicate in when using a platform. Conversely, they are also the words, phrases and concepts that applications on a given platform use to communicate with users. The more usable and consistent these are on a given platform, the more advantages there are to creating native applications for that platform.

At one end of the spectrum we have Apple’s iOS, with its detailed “Human Interface Guidelines”³ and its elegant and consistent Cocoa Touch framework. A native productivity app on that platform that conforms to the guidelines will inherit much of the usability inherent in the core frameworks and will seem instantly familiar to users who are already familiar with other applications on the platform.

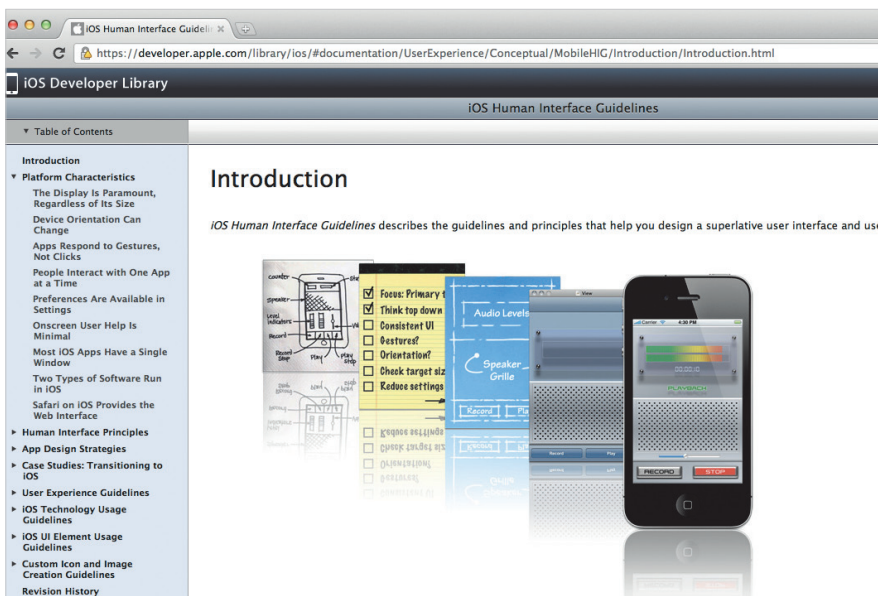


Figure 9.3. Apple’s “Human Interface Guidelines” provide clear instructions on how native apps on the iOS platform should look and behave. They help define the culture of the platform.

At the other end of the scale, we have native platforms like Android that are heavily customized by manufacturers, carriers and users to the point that there is little, if any, consistency between different Android phones and applications. Designers of native applications on such platforms might have a harder time providing a consistent user experience.

³ [smashed.by/apple](https://developer.apple.com/design/human-interface-guidelines/)



Figure 9.4. The Swype keyboard is actually quite amazing. Simply slide your finger from letter to letter, and it automatically guesses the word you’re thinking of. Unfortunately, it is not on every Android device, just some. Others come with keyboards that are more similar to the one on iPhone, and users can buy and use a multitude of other third-party keyboards. While this array of choice might seem good initially, it means that there is no single unified Android user experience. There are, in effect, as many Android user experiences as there are different versions of Android as customized by manufacturers, carriers and users themselves. This makes it very difficult to use a common design language when building apps.

For example, my iPhone app, *Feathers*,⁴ has a custom keyboard that I made for users to enter extended Unicode symbols. On iPhone, it works exactly like the built-in iPhone software keyboard. Achieving this took some effort, but it was not impossible. If I were to port the application to Android, I would have to know which of the many software keyboards the user has installed and then customize its behavior to match. Needless to say, this would involve a lot more effort and might not even be feasible. The Swype keyboard⁵ that comes on some Android phones, for example, is patented. So, on an Android device with a Swype keyboard, I cannot make my keyboard behave exactly the same way as the system keyboard.

The compromise would be to make a single custom keyboard and use that regardless of the user’s system keyboard. Of course, it would neither look nor feel like the main keyboard and thus wouldn’t provide the same seamless experience of *Feathers* on iOS. Instead, users of the app would have to learn to use two different types of keyboards in the app and have to exert cognitive effort when switching from one to the other.

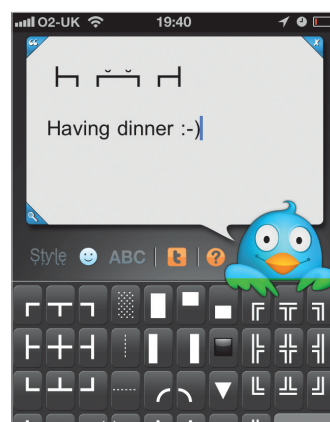


Figure 9.5. Porting *Feathers* to Android would require making different versions of the custom keyboard to support many different keyboard types.

⁴ smashed.by/fapp

⁵ swype.com

Similarly, on native platforms that do not have a strong, consistent visual language and culture—such as Android—or on native platforms that are notorious for being confusing to use—such as Windows—non-native applications will have less of a handicap. They might even provide better usability and user experiences in some situations.

A great example of a non-native application providing a better user experience on certain native platforms is Gmail. Using a desktop mail client on an operating system (OS) such as Windows could require you to find and install the application itself, keep it updated, keep your mail synced between your various devices, and make sure messages are checked for viruses and other malware. Compare that to the simplicity of Gmail, in which you enter a URL in the Web browser of any device and—boom!—you

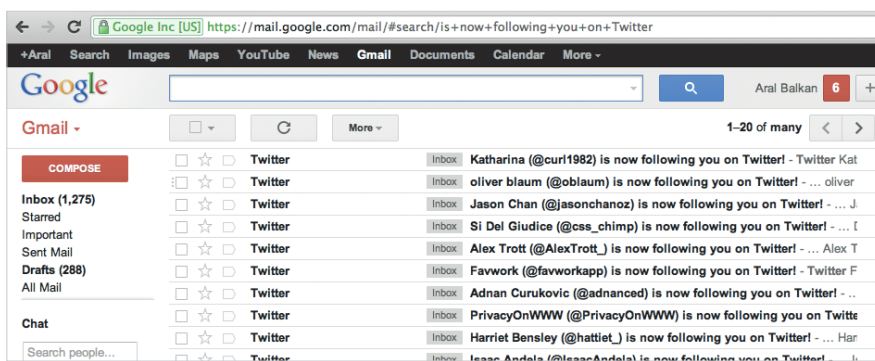


Figure 9.6. The Gmail Web app provides a consistent experience across platforms. Users don’t have to install anything or worry about syncing their email to multiple devices.

have your mail. End of story. Beautiful. Gmail is also a great example of how creating good cross-platform user experiences can require a lot of platform-specific optimizations. Although the Gmail app runs across desktop and mobile platforms, there are actually several highly optimized versions of the app.

The Web as a platform itself, however, has few user experience consistencies of its own. Although Web applications share common features, there is no “Human Interface Guidelines” document for the Web (maybe there should be).⁶ Instead, we focus on documenting good coding and design practices, such as progressive enhancement. Different browsers (that is, native applications that run Web apps) implement the behavior of core form controls differently. And that’s why a Web application could behave differently in different browsers even if it has the same markup, components and code.

⁶ See Tantek Çelik’s call for “Web Human Interface Guidelines” (smashed.by/wehuin) and read Joe Hewitt’s post calling for more focus and vision for Web technologies (smashed.by/owned).

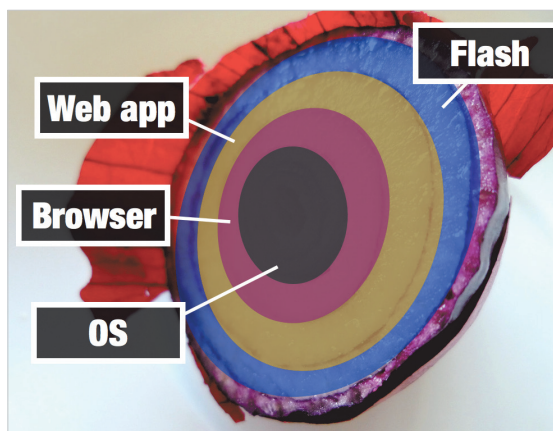


Figure 9.7. A browser is an application that runs in the context of the OS. In other words, a browser is a native application. A Web app, on the other hand, is an application that runs in the context of the browser. It is not a native application since it has one degree of separation (the browser) between it and the OS. Similarly, a Flash app runs in the context of a Web app. It is not a native Web application since it has one degree of separation (the Web app) between it and the browser. Flash apps, therefore, are not native to the browser, just as Web apps are not native to the OS. (Image: Rosmarie Voegtli, smashed.by/voegtli)

Hybrid Applications

So, we’ve got native applications, which refer to the culture of the platform that they run on, and we’ve got Web applications, which run in a browser on the Web. But we’ve missed a third category that many modern applications fall into: hybrid apps.

We need to understand the strengths of various technologies and use them where appropriate. Declarative Web authoring technologies (primarily HTML and CSS) are great for creating complex documents and styling them beautifully. Thus, many designers of native applications use HTML and CSS when they need to display rich content. These types of applications are called *hybrid applications* because they employ a mix of native and Web technologies.

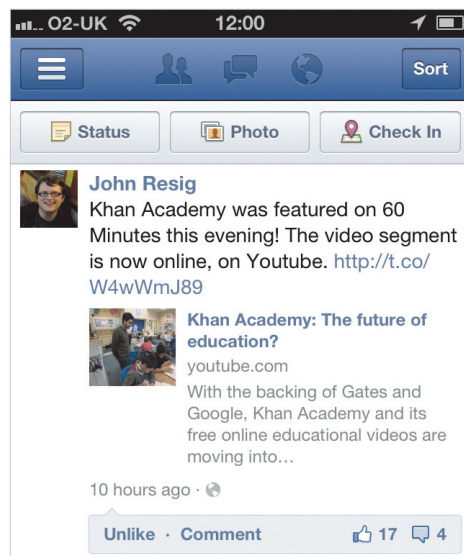


Figure 9.8. The Facebook app on iPhone is a hybrid app.

The Facebook app on iPhone is one example of a hybrid app, in which certain sections (such as the news feed) are rendered using Web technologies.

Similarly, as we saw before, Web applications can also be hybrid. A website authored in HTML, CSS and JavaScript and that uses Flash to display rich interactive content is an example of a hybrid Web application.

Many applications today are hybrids, and if you are accomplished in HTML, CSS and JavaScript, it is safe to say you will never go hungry regardless of which platform or platforms end up becoming the most popular in the decades ahead.

Overcoming Ideological Bias

All too often, technology and design decisions are made based not on a desire to choose the best materials and tools for the job but on ideology. The Web standards advocate who blindly recommends the Web platform and Web authoring technologies for any project, regardless of the users’ needs is, unfortunately, all too common. Developers who blindly recommend Flash and native iPhone or iPad apps for any project are all too common, too. The old adage “*When all you have is a hammer, everything looks like a nail*” comes to mind. So, it is important to recognize such biases and to base your decisions on the needs of your users, not on ideology. It is also important to be able to recognize ideological views so that you can steer the discussion back into the realm of design.

Some advocates of Web standards hold a core assumption that the Web platform and Web authoring technologies are, by nature, a force for good. One wouldn’t dispute that the Internet and, by extension, the Web have had as radical a democratizing effect on the world as the Gutenberg Press (if not more so). However, the Web platform and authoring technologies are inherently neither good nor bad, and they could easily be used for either end.

In the case of the Web platform, the common assumption is that it is inherently good because any document or application placed on it is universally accessible. While this may be true for open collections of documents—as was the norm for content in the early days of the Web—it is generally not true for what we would consider a modern Web application today. Take Facebook. Facebook is a Web application. It is not open. It is free, but what this means is that you, the user, are not Facebook’s customer. You are Facebook’s product.⁷

Your personal information and behavioral traits are what Facebook sells to its real

⁷ As Andrew Lewis states, “If you are not paying for it, you’re not the customer; you’re the product being sold.” smashed.by/discont

customers, advertisers. Is this in any way inherently better or more open than having to buy a commercial application from Apple’s App Store?

Not really.

In fact, you could easily argue that buying a license for a commercial iPhone app is a more honest and up-front transaction. You pay for it and thereby own a license to use it. You become the customer of the company or individual who made the application. There is transparency in how the company makes its money. In many ways, this is a much more traditional commercial relationship.

Of course, even commercial apps can use your data in devious ways, so being vigilant about your personal information these days is important. But the point is that simply being a Web application does not somehow magically make it a force for good in the world.

Are Native Applications and Platforms Endangered Species?

As Jeremy Keith famously put it at the Update Conference, “Writing a native app is like coding for LaserDisc.” The implication here is that native platforms, like CD-ROMs and LaserDiscs, will be obsolete soon since the Web is “catching up.”

I sure hope that is not the case, because the Web itself is a native platform and is becoming even more so (in the traditional sense) with the rise of OS’s such as WebOS and Chromium, which are based on native Web authoring technologies. We have to understand that what the Web is supposedly catching up to is a moving target. New features, user experience enhancements and more are being added to native platforms all the time. It’s not like Apple will decide on 1 September 2012 that iOS is “done” and stop creating new versions of the OS or new models of iPhone and iPad, at which point the Web can take a few years to finish catching up.

We know that “native” refers to the core culture and language of a platform, so predicting the demise of “native” is analogous to saying that different devices will not have distinct cultures in the future.

The assumption inherent in this view is that a monoculture will arise in the future in which every application on every device will speak via the components of the Web and be authored using native Web authoring technologies, and moreover that these applications will all be served from the Web platform. If anything, this is a gray and chilling vision of the future, in which people will have even less control of their own

data and in which their devices will simply become dumb terminals that hook up to great silos in the sky (“the cloud”) that are controlled by large corporations.

Instead of owning a license to a word-processing application, for example, you would write everything in Google Docs. Google, for its part, will be analyzing every word and sentence, trying to understand more about who you are and what makes you tick so that it can use that information to better manipulate your commercial behavior.

This is not to say that Web applications are necessarily evil, but they are definitely not inherently good.

Blurring of the Lines

We keep hearing that “the Web is catching up to native.” What people mean when they say this is that Web authoring technologies are getting better access to device features such as touch, hardware-accelerated graphics, GPS, accelerometer support and cameras. What is rarely mentioned, however, is how native applications are catching up to Web applications. In some respects, the strides that native applications are making are far more important because they threaten the core user experience advantages that Web applications have historically enjoyed over native applications.

The three main areas in which native applications are catching up to Web applications are ease of deployment and access, automatic updates and seamless access to data.

EASE OF DEPLOYMENT AND ACCESS

One of the core user experience advantages that Web applications have historically had over native applications is the ease with which they can be deployed and accessed. Click the “Upload” button in your FTP application of choice⁸ and your app becomes accessible to anyone who has the URL anywhere in the world.⁹ Simple.

No need to download a Zip file, then search for an application to unzip it with, then look for where you downloaded it to, then unzip it, then install it, then run it, only to find out that it doesn’t support your graphics card. Eek! Is it any surprise that Web applications like Gmail and Google Docs have enjoyed such phenomenal success, especially on native platforms with poor usability?

However, native apps are catching up to the ease of deployment and access offered by

⁸ Or, if you’re really savvy and use a Git repository, you could use a post-commit hook to automatically deploy your latest commit to the server.

⁹ At least where the URL isn’t blocked by an autocratic regime.

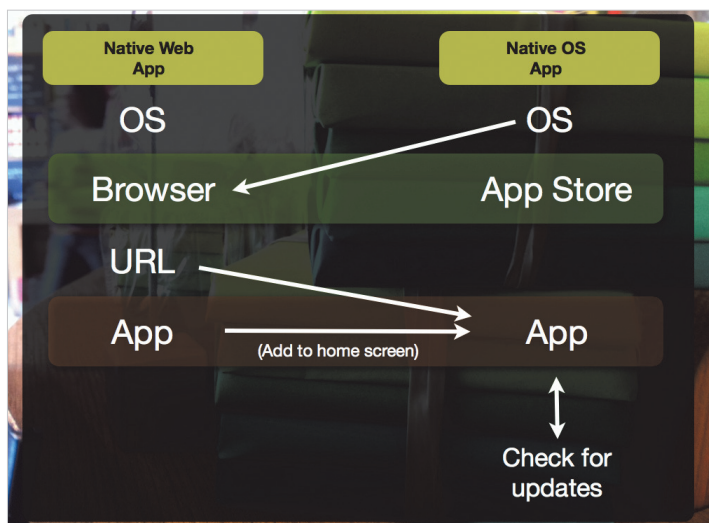


Figure 9.9. The line between the user flows of Web applications and native OS applications is blurring. In fact, in OS’s based on Web technologies, Web applications are native OS applications.

Web applications thanks to the development of app stores. With an app store like Apple’s, the process of finding an application is as easy as hitting a URL in a browser. In fact, you can hit a URL to reach an app in the App Store and, from there, simply click a button to download and install it.

AUTOMATIC UPDATES

Web applications, by nature, have always provided automatic updates. In fact, we don’t even think about the “version” of a Web application because the Web is inherently version free.

You’re always using the latest version of Gmail, and you don’t care what version it is. It’s not Gmail 7; it’s just Gmail.¹⁰

Native applications, by contrast, have usually had clunky update mechanisms that interrupt the user’s flow. That, too, is changing with more and more native applications implementing seamless updates. When, for example, was the last time you noticed Google Chrome updating? Never. It does it silently.

¹⁰ Read up on the “one version manifesto” and the versionless character of the Web in my opinion piece in .Net magazine titled “My Websites Will Only Support the Latest Browser Versions”: smashed.by/netsup

SEAMLESS ACCESS TO DATA

The other huge advantage that users of Web applications have traditionally enjoyed is the ability to access their data from any device. You never have to worry about syncing email from your desktop to your mobile phone when using Gmail. It’s always there. Compare this to the nightmare that has traditionally plagued syncing on native platforms.

Native apps, however, are again catching up. With Apple’s iCloud, for instance, manual synchronization is becoming a thing of the past. Your data is simply, transparently and automatically available on your Apple devices and is kept constantly in sync without your having to worry about it. Although iCloud is primarily an Apple-centric continuous client solution, other cross-platform technologies, such as Dropbox, bring similar advantages to other platforms.

Just Another Client

Did you read the previous section thoroughly? Good. Then you may have noticed the common thread in all three areas where native applications are catching up to Web applications. They are all areas where the advantage in user experience is due to a characteristic of the Internet and not the Web. The Web is just a stack of technologies—namely, HTTP and URLs—on top of the Internet stack. So, native applications are catching up to the Web by taking advantage of the very same characteristics of the *Internet* that the Web does.

Furthermore, we are seeing the rise of a new type of user experience pattern, called the *continuous client*. A continuous client experience—as originally proposed by Joshua Topolsky¹¹—lets a user seamlessly continue an experience across devices and contexts.

For example, if you are reading your Twitter stream on your computer and then grab your phone, you should be able to continue reading the stream from the same place. And when you get home, you should be able to continue from the same place again on your TV.

A good example of a continuous client in the wild is the Trillian instant-messaging application. It can store chats in the cloud, share chats between all of your devices in real time, keep track of and synchronize your read and unread messages, and even make use of “presence technology” to know on which device you’re currently active so that it can send current notifications only to that device.

¹¹ Topolsky, Joshua. “A modest proposal: the Continuous Client,” smashed.by/client.

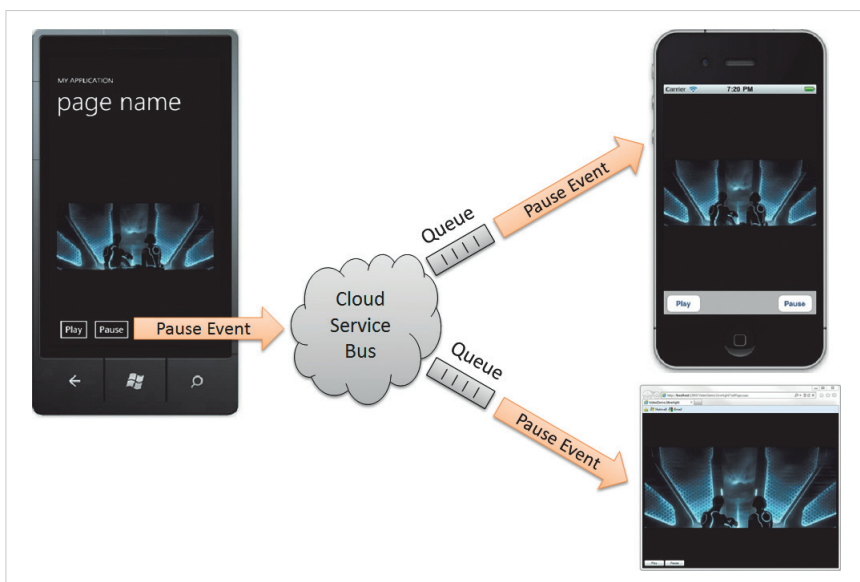


Figure 9.10. Kelly Sommers has a nice sample app that showcases a continuous client experience: smashed.by/multi. Users can start watching a video on their Windows Phone 7, continue in a Web client, and then on their iPhones.

If you think about it, in the age of continuous client experiences, the Web becomes JAC (just another client). It may be the best client to use in certain contexts, but users have the choice of switching to native clients without worrying about synchronizing data. Soon, continuous clients will be a core expectation instead of a novelty, especially as high-level technologies, e.g. iCloud, make it easier for developers to implement them.



Figure 9.11. The instant messaging app Trillian is a good example of a continuous client. (Image: Trillian Blog, smashed.by/trillian.)

The Future is Native

The Web today is just another native platform and one that, going forward, has to compete on its own unique merits and not on the advantages bestowed upon it by the Internet, because other native platforms are also implementing those same advantages.

The tricky question to answer when deciding whether to use the Web platform or other platforms for your next application is whether the advantage lies in exposing the user interface of your application as a native app or in exposing it via a URL and serving it via HTTP.

As Web applications catch up to native platforms with features like Local Storage and the ability to run while the device is offline, the line between Web and native applications blurs even further. This is taken to its logical extreme in OS's like Palm's WebOS and Google Chrome, where the native technologies are Web technologies.

We need to understand that a Web application running on such an OS is a native application. At that point, our decision is between different native OS's and native frameworks. That choice will be greatly influenced by which native OS's offer a superior user experience. Subsequently, our technology choices will be between different native authoring technologies—HTML, CSS and JavaScript for native Web applications, Objective-C and Cocoa Touch for native iOS apps, Java and Android SDK for Android apps, C# and .NET for Windows Phone apps, and so on and so forth.

Regardless of which platforms and technologies win out in the end, the future is clearly native and the Web is JAC. Now, the billion-dollar question is not “Do we go Web or native?” but rather “Which platform or platforms and which client technology or technologies should our new product support?”

To answer this, we need to understand the nature of our product and, specifically, where it falls on the documents-to-applications continuum.

The Documents-to-Applications Continuum

On the Web, one way to classify products is according to whether they are content-centric or behavior-centric. We call a content-centric collection of documents a *website*. A behavior-centric product is called an *application* (or “app”). Instead of falling entirely in one camp or the other, your product will probably lie somewhere between the two extremes on the documents-to-applications continuum.

When a product falls closer to the documents side of the continuum, we can use progressive enhancement to layer features and interactions on top of the content-based core while keeping that core accessible to the largest number of people possible. These

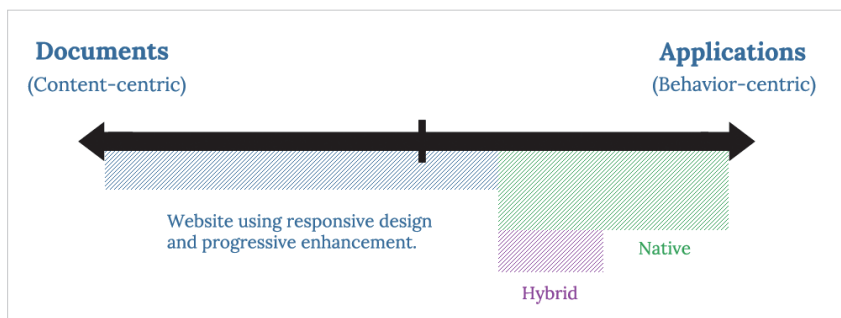


Figure 9.12. The documents-to-applications continuum.

progressively enhanced features usually either layer advanced formatting or layout on these documents or add fanciful interactions for navigating within or between them. We can make the content adapt to different screen sizes and make the limited interactions that are used to navigate the content adapt to different input mechanisms. This isn't an easy task, but nor is it impossible.

As products shift from the documents side of the spectrum to the applications side, however, implementing progressive enhancement gets harder. In fact, it might become entirely meaningless or impossible. How would you gracefully degrade an online im-

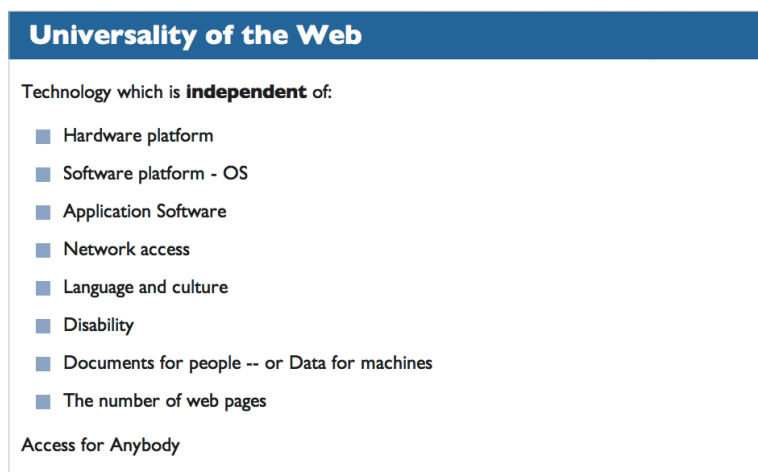


Figure 9.13. The principle of universality, as composed by the creator of the World Wide Web, Tim Berners-Lee, was written in an age when the Web was mostly a collection of interlinked documents. It doesn't necessarily apply wholesale to applications, which have very different design constraints and requirements. See Lea Verou's comment on John Allsopp's retort to my one-version manifesto on .Net magazine: smashed.by/netmag

age editor, for example? How would an image editor work on feature handsets without graphical displays? What content would you actually fall back to displaying?

Applications are not content-based; they are behavior-based. Gracefully degrading to a simpler representation of whatever content an application might have does not always make sense. Applications are often made up entirely of behaviors that let users create content. Consider the image editor again: it doesn’t have any content itself, but it enables users to create content.

In order to create exemplary user experiences, we need to maintain focus. This focus has to be placed squarely on meeting the needs of our users in the best way possible.

Given unlimited time and resources, we could optimize the user experience of our apps on every device and platform known to humankind. However, given limited time and budget we have to work with in the real world, we must be selective with our audience, problem domain, platforms and devices. We do this not to exclude people unnecessarily but because we realize that including everyone and giving everyone a great user experience is impractical.

No product team on Earth has the resources to create applications that provide the best possible user experience for every user.

Mobile Design Considerations for Document-Centric Products (Websites)

On the documents side of the continuum are websites. A website comprises content as well as the presentation of that content. The content may be text, images, audio, video, etc., that is marked up with HTML in order to add semantics and structure. The presentation includes both the visual layout and the interactive elements (such as navigation between pages or states) and is usually implemented using a combination of CSS and JavaScript.

If you have a website, the first thing you should do is test it out in mobile contexts to see how it displays and performs. I’ve encountered far too many companies that overlook the importance of making their websites mobile-friendly and jump straight into creating a native application. Be careful of making this mistake, especially if your website is an important revenue generator. For document-centric websites, use progressive enhancement whenever possible to make sure the content and core interactions for navigating and consuming that content remain accessible to as wide a base of your users as possible.

To do so, you can follow these broad steps:

1. Make sure your content is accessible to everyone, and mark it up with proper semantics. Separating content from presentation is key.
2. Progressively enhance your content to provide a more optimized experience for people viewing it at different screen sizes (this is the focus of responsive design currently).
3. Progressively enhance your content to provide a more optimized experience for families of devices based on supported features and capabilities (for example, touch). In other words, make the website responsive in behavior, not just in layout. At this point, you are optimizing for features, not for specific devices (say, for all mobile phones that support touch, not just the iPhone).
4. Progressively enhance your content to support the unique culture and capabilities of the various devices you want to support. At this point, optimizing for specific devices is all right. There’s nothing wrong with trying to make the user experience as beautiful as possible, even if it is for a specific subset of your users at a time.

Of course, each of these steps will take time and more resources, and you will need to plan and budget accordingly. But what if making your current website responsive does not meet your users’ needs in this particular instance? Perhaps you need to provide a more optimized and focused experience than progressive enhancement allows given your budget and schedule, or perhaps you need a level of device integration that is simply not possible through a browser currently? In this case, you might want to start thinking about whether to build an app, and if so, whether it will support multiple platforms and which technologies you should use to build it.

Make sure to test your designs on actual devices

A simulator or emulator is great for testing the effects of code changes while you’re developing, but they cannot recreate the unique ergonomics of the device itself. Context is also a key factor that affects the usability of mobile experiences, and a design that works perfectly well under the perfect lighting conditions of an office might not work as well in bright sunlight, for example.

Also, remember that when you test with a simulator, it uses your computer’s powerful hardware to run the application. You might see slower performance—and even slightly different behavior—when running on an actual device.

Cross-Platform or Single Platform?

Can you best meet your users’ needs with an application that runs on multiple platforms or—at least initially—on a single platform?

When answering this question, keep in mind that creating an application for a single platform does not mean that you cannot create a separate app for a different platform later on. It just means that you will be focusing your design and development efforts on a single platform (or perhaps even a single device) to start with.

It also does not, by itself, imply whether you will be creating an application that runs as a native binary or whether your application will use the native components of the device or devices that it runs on. (Using cross-platform authoring technologies to create an application that you have optimized for a single platform is perfectly possible.)

The answer to this question will, however, determine how much optimization you must do on different platforms. If you care about the user experience, you must optimize your app on each and every platform that you support. It will also affect how much testing you must do (because you will need to test on every platform that you support), the size of your support department (because you will need to troubleshoot user issues on every platform that you support), and how much time and budget you need to set aside for these various functions.

THE MYTH OF WRITE ONCE, RUN ANYWHERE

A common mistake I see many designers make is to assume that by using cross-platform authoring technologies they will be able to write once, run anywhere. This is a myth. And acting on the myth can lead to rather costly underestimations.

The only way to get an application to run well on multiple platforms is to optimize it for each one of those platforms and devices. As mentioned earlier, every platform has its own culture, language and norms that users expect apps to conform to. And most users do not care how many other devices your application runs on—they care only about how well your application runs on their device.

Designers who do not take the unique cultures, customs, language and norms of their respective platforms into consideration risk making their applications look and sound out of place. The applications will appear noticeably foreign, unnecessarily loud and usually rather arrogant, simply because they are culturally insensitive.

Your application might run on multiple platforms, but this rarely—if ever—means that it will run well on multiple platforms.

Because we don’t want our applications to exhibit such obnoxious behavior, we must optimize them for every platform we support. Our aim is to create applications that are culturally sensitive to the language, norms and customs of the platforms they run on. Anything less and we would be showing varying degrees of disrespect to certain segments of our users.

DEATH BY A THOUSAND CUTS

The worst thing you could do, of course, is disrespect all of your users by creating a lowest-common-denominator application that gives every user on every platform an unoptimized user experience. At that point, you would be at your most vulnerable. Even though your cross-platform application might have the potential to reach a large number of users because it runs on a large number of devices and platforms, it will be at a disadvantage in user experience on every one of those platforms. To put it bluntly, who cares if it runs on a certain platform if it runs badly?

More importantly, what happens when a competitor comes by on one of those platforms with a beautifully designed, precisely optimized, delightful user experience and blows your app out of the water?

And then again on a different platform by a different competitor?

This is death by a thousand cuts, whereby your application is abandoned one platform at a time for superior alternatives. Supporting multiple platforms is not a feature unless you can support them all well. You may have first-to-market advantage, but that will last only until you are outdone by your best-in-market competitor.

WRITE ONCE, OPTIMIZE EVERYWHERE

So, write once, run anywhere is a dangerous myth. Cross-platform applications that compete successfully are write once, optimize everywhere. You must understand the implications this will have on your budget and schedule and plan for optimizing, testing and supporting your application on every platform you choose to support.

Please do not be taken in by the “create a cross-platform application in five minutes” demos by various tool vendors. They are rubbish, unless you are building the simplest of to-do list apps. The true test of a development technology is not how easily it enables you to create a five-minute demo or how quickly it gets you 70% of the way to your goal, but rather how easily it enables you to tackle the last 30% of your development, including the all-important user experience optimizations that could take up the last 10%. These details could end up taking up the bulk of your time and effort in developing the application.

WEB AND OTHER CROSS-PLATFORM TECHNOLOGIES

Broadly speaking, cross-platform technologies can be split into two groups: those that create native binaries and those that don’t. We can also further categorize them as those that use native frameworks and those that don’t. The combination gives us the binary-framework matrix shown below:

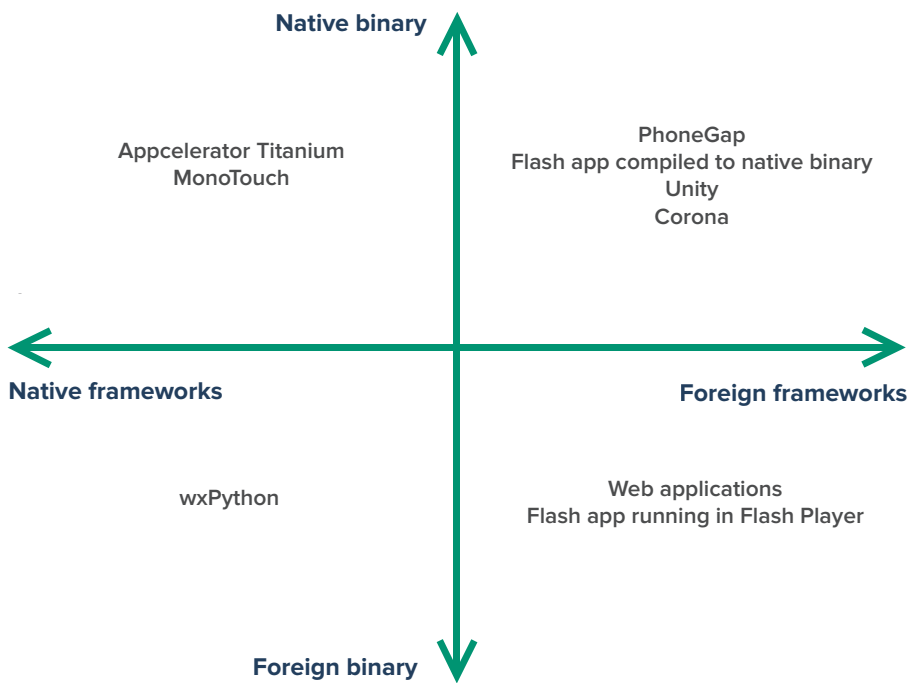


Figure 9.14. The binary-framework matrix.

A native binary is an application bundle that can be run directly by the OS of a given device. It is what we think of commonly when we refer to a “native app.”

However, the far more important test of nativeness is whether the application makes use of native frameworks. These frameworks are what embody the culture, language, gestures, symbols and norms of a platform.

Foreign Apps Wrapped in Native Binaries (or Wolves in Sheep’s Clothing)

Creating a native binary for iPhone that does not use any of the native components in the Cocoa Touch framework for iOS is possible (and quite easy) by using a technology such as Adobe’s PhoneGap. PhoneGap wraps an existing Web application—that uses native Web components—and creates a native operating system application from it (in the example above, a native iOS application). While your app might look like a native app on the iPhone’s home screen and might launch like a native app, the UI of the app will be rendered using Web components.¹²

Your binary application is just a shell that contains a WebKit component. This WebKit component is what renders your Web application using Web components. Because Web components cannot meet native expectations, I would not recommend using PhoneGap to create non-immersive applications such as productivity apps.¹³

When building immersive applications such as eBooks and games, however, the lack of native framework support in similar technologies is not as big a problem. Adobe Flash and ActionScript, Unity and Anscá’s Corona are favorites among native game and eBook app makers, even though they do not use native frameworks or components. This is because immersive apps rarely, if ever, use native components. Instead, their designers aim to create a completely different world—perhaps a skeuomorphic one that looks and behaves like a real-world book—with its own rules, interactions and culture.

I do not recommend Adobe Flex (or applications that use components from the Flash framework) for the same reason that I do not recommend using PhoneGap to create non-immersive native applications: they do not conform to the culture of the native platform and will behave differently than native components on the system.

In summary, be careful when creating native binaries that simply wrap applications that do not use native components. These apps have a tendency to look like native applications, but they cannot behave like native applications because they do not use

¹² PhoneGap does not dictate the use of any particular Web UI framework (for example, jQuery Mobile). All it does is let you wrap a Web application in a native application. However, regardless of which UI framework you use (or even if you decide not to use a Web UI framework at all), the rendered components will be Web UI components, not the native UI components of the platform that your app runs on.

¹³ The same can be said for Web applications that are added to the home screens of phones. Again, the Web application in question looks like a native app, launches like a native app, but does not behave like a native app. A Web application running in the browser does not have these shortcomings because it does not create the expectations of a native OS app in the first place. A Web application running in the browser is a native Web application, and native Web applications—just like native apps on other platforms—have their own culture, conventions, language, norms and expectations (even though these might not be as strongly defined as on some other native platforms).

native components in native frameworks. A PhoneGap application that uses the jQTouch framework might display what looks like an iOS table view when running on an iPhone, but this is simply an HTML look alike brought to life by clever use of CSS and JavaScript. It pretends to be an iOS table view, but it cannot meet the behavioral characteristics of a real table view component from the Cocoa Touch framework, and thus it ends up creating expectations that it cannot meet.¹⁴

Creating unmet expectations is a major faux pas of usability. Avoid it like the plague.

Immersive vs. Non-Immersive Applications

Understanding the distinction between immersive and non-immersive applications is important because the nativeness of an application is considerably less of an issue for immersive applications.

Immersive applications usually take over the whole device and, by definition, create their own culture and language. A good example is games. A game could have its own control system for running, jumping and firing. Far from conforming to the culture of the platform, a good game transports the user to a world of its own creation. As such, immersive applications usually employ few, if any, of the platform’s conventions. As such, they are prime candidates for the use of non-native technologies.

This explains why Flash, while not native to the Web, is such a popular technology for delivering immersive experiences such as games on the Web; or why Unity, while not using the native components in Cocoa Touch, is used to make many of the top games on iOS and other platforms.

Non-immersive applications (such as productivity apps) usually make heavy use of standard user-interface elements like buttons and table views. They use platform-specific interactions like full-screen “master-detail” transitions on iOS, whereas they might use a tree-view control on a Windows or OS X application. It is thus very important that non-immersive applications speak the native language and conform to the native culture and norms of the platforms they run on.

All that being said, even for immersive applications, performance could still be an issue that influences your choice of whether to use native or non-native technologies.

¹⁴ The biggest usability faux pas you can commit is to style or skin non-native components to look like native components (as the jQTouch framework does). Whereas components in a native app that do not look like native components at least offer a visual clue that they will probably not behave like native components, non-native components that pretend to be native components will confuse users even more by appearing a certain way but behaving differently. The best thing to do, of course, is to use native components in your native apps. Failing that, at least make your components different enough visually so that you do not create behavioral expectations that you cannot meet.

Certain games—such as first-person shooters—need to squeeze every bit of performance out of a system in order to shine. In these situations, some non-native technologies might not be performant enough for your needs. The early versions of Adobe AIR-based Flash apps on iPhone, for example, were notoriously slow. Adobe has improved performance in the latest versions of Adobe AIR on iOS, though.

Native Apps Translated From Non-Native Languages

If your goal is to provide an optimized user experience, a better cross-platform approach when creating non-immersive native applications would be to use native frameworks and components. This does not necessarily mean that you have to use the native programming language for a given platform to author your application.

Using Appcelerator’s Titanium Mobile SDK, for example, you could write JavaScript to instantiate and populate native components.¹⁵ Thus, on iOS, when you create a table view in Titanium Mobile, a native Cocoa Touch table view component (a UI TableView instance) is created in your user interface. This not only looks like a native table view (as could also be the case in a PhoneGap application that mimics the native components) but actually is a native table view. Most importantly, it behaves as a table view should.

The use of a scripting language like JavaScript can make it easier to author applications and reuse your team’s existing Web development skills, while still affording you all of the advantages of using native components in your native applications.

Also, because Titanium Mobile is a cross-platform technology, it knows to instantiate native iOS components for your native iOS app and use native Android components instead when compiling your native Android app. The advantages are clear: you use a single scripting language (JavaScript) and have to maintain just one code base, instead of having to learn and use Objective-C on iOS and Java on Android and maintain two separate code bases.

The disadvantage is that you have yet another layer of abstraction to work with. Ultimately, the quality of your applications will depend on the quality of the code that Appcelerator writes in its abstraction frameworks. And you will be dependent on how quickly Appcelerator supports new features that are released to the native frameworks and SDKs. Although Appcelerator tries to make the differences between platforms as transparent as possible in Titanium, there are differences—not least of all cultural

¹⁵ Don’t confuse Titanium Mobile with Titanium Desktop. The latter works just like PhoneGap. Appcelerator has recently open-sourced Titanium Desktop (now called Desktop) and is focusing its efforts on improving Titanium Mobile: smashed.by/tita.

Product	Authoring technology	Deployment		
		Native binary?	Native components?	Cross-platform?
Web app	Web (HTML, CSS, JavaScript)	No ¹	Yes ² & No ³	Yes
Appcelerator Titanium	JavaScript	Yes	Yes	Yes
PhoneGap	Web (HTML, CSS, JavaScript)	Yes	No	Yes
Xcode (and/or GCC, LLVM)	Objective-C, Cocoa Touch (and C, C++, possibly Ruby, Python)	Yes	Yes	No
Unity	C# and other languages	Yes	No	Yes
Mono (for Android & MonoTouch)	C#, core device frameworks	Yes	Yes	Yes

*1 Does not use binaries
*2 If running in the browser
*3 If running from home screen link

Figure 9.15. Comparison of some common native and cross-platform technologies.

Native is not necessarily better, but it is native.

The only way to create applications that conform to the norms—that is, the culture and language—of a given platform is to use native technologies. E.g., while creating Flash applications that are served on the Web platform is possible, they will not look or feel like native Web applications that use native Web authoring technologies like HTML, CSS and JavaScript. Similarly, while using these technologies to create applications for platforms like the iPhone is possible, the applications will not look or feel like native iPhone applications that are created using the components in the Cocoa Touch framework. That is not to say that Flash applications cannot perform better than HTML applications. In certain use cases—especially immersive apps like games—Flash applications might provide a better user experience. Machinarium, for instance, is a lovely game created in Flash that runs beautifully on iPad. Again, especially for immersive applications like games and eBooks, a cross-platform technology like Unity or Corona could reduce development time and make it easier to implement features that would be more difficult to create using native technologies (such as a 3-D environment or a physics engine).



Figure 9.16. Machinarium on iPad, an immersive native app created in Flash.

ones—that you will still need to address and optimize for (remember that our credo is write once, optimize everywhere).

This is not to say that you should fear cross-platform technologies, but rather that you should do your research, weigh the pros and cons and make an informed decision of whether to add another layer of abstraction to your development process. Each cross-platform technology has different advantages and disadvantages and use cases that make it a better fit for certain types of applications. While Corona might be the perfect choice for a 2-D physics-based game, Titanium Mobile might be better for building a cross-platform productivity application.

Of course, Titanium is not the only cross-platform technology that can create native binaries and use native frameworks. If your team has skills in C# and .NET development, you might also want to consider Mono (and specifically MonoTouch for iOS and Mono for Android). Mono works in much the same way as Titanium, but instead of using JavaScript, you use the C# programming language (... and, with caveats, other .NET programming languages) and .NET patterns and tools to create native apps.

Web Applications

If you are reading this book, you are probably either a Web designer or a Web developer (or both), or you are learning to become one. Your role as a Web designer might involve anything from designing collections of documents (in which case you would rely heav-

ily on your graphic design skills) to designing behavior-rich applications (in which case you would be exercising your interaction design skills).

You also have a rich selection of materials to choose from when designing your products. Depending on the needs of your audience, you might choose to use native Web authoring technologies such as HTML, CSS and JavaScript to create native Web applications; alternatively, you might use non-native cross-platform application authoring technologies such as Adobe Flash or Unity to create non-native Web applications. As a third option, you might use a combination of both native and non-native authoring technologies to create hybrid Web applications—for example, a website built primarily with HTML, CSS and JavaScript but complemented with Flash to deliver a rich gaming experience.

Regardless of your choice of native or non-native Web technologies, what makes a Web application a Web application is that it is deployed to, and accessed from, the World Wide Web platform. This means, at its simplest, that your website or application has a URL¹⁶ and is served via HTTP.

Single-Platform Native Applications

Having worked through the process above, you might decide that you would best serve the needs of your users by tackling a single platform or device for your application and investing your limited time and budget into optimizing the user experience for that platform.

If you do decide to support a single platform, you still have technology choices to make. If you’re making a game or other immersive application, you can choose a cross-platform technology that is specifically geared to making it easy to author such applications, such as Anscá’s Corona, Unity or Adobe Flash.

If you are building a non-immersive application, you still have the option of using a cross platform toolset like Titanium Mobile or Mono. The downside there is that your development process will involve one additional translation layer, and you will have less control over optimizing the performance of the application because you will be reliant to some degree on the native code that Titanium’s engineers have written. Even if you do use Titanium (or a similar framework like Mono), remember that you will still have to learn the native frameworks (or APIs) of the platform that you are developing for. Learning new frameworks is much harder than learning new programming languages.

¹⁶ More precisely known as a URI, although there are technical differences between the two that would make a W3C standards geek froth at the mouth if they were to see as cursory a dismissal of those differences as is being displayed here.

Whereas a seasoned programmer could pick up a programming language like Objective-C in a matter of days, it could take weeks (if not months or years) for a developer to fully grok and become comfortable with the patterns, culture and intricacies of a framework like Cocoa Touch.

Finally, you could use the native toolset of your platform of choice. For example, you can use Xcode with Objective-C and Cocoa Touch to build a native iOS application, or Eclipse with the Android SDK to build a native Android application, or Visual Studio with C# and the .NET framework to build a Windows Phone application.

This could involve learning a new programming language (Objective-C for iOS, Java for Android and C# for Windows Phone) or hiring team members who know the language and have experience with the native framework. As stated, picking up a new language is easy, but learning a new framework is much harder. Keep that in mind when deciding whether to go this route if you or your team do not already possess the required skills.

The advantages of building native applications using native technologies are numerous. For one thing, you have complete flexibility in optimizing the application and user experience. When you use native components and adhere to the human interface guidelines for your chosen platform, your application will conform to the culture, language and norms of that platform. It will be easier for users to learn and use. Also, since you are not reliant on a foreign abstraction or translation layer, you will always be working with the latest code and frameworks from the manufacturer, giving you additional flexibility to accommodate new features and updates as they are released for the platform. Most importantly, concentrating your efforts on a single platform means that you can put the time that you would otherwise have spent optimizing, testing and supporting other platforms into refining and optimizing the user experience of your application first, and if necessary add support to other platforms later on.

DESIGNING FOR HUMANS

The platforms and technologies that you decide to use for your next product will have a fundamental impact on how the product is accessed, how it looks and feels, and whether it meets—and hopefully exceeds—the needs of your users. This is not a decision to be taken lightly or brushed over.

If your choice of platforms and technologies is based simply on your perceived short-term business needs or on the current competencies of your team, then you are making a decision that solves your own problems, not the user’s problems. This may

have short-term advantages, but you will not be able to compete in the long term with those who solve the user’s problems first. Your choice of technologies and platforms should be based on how best you can meet the user’s needs, not on ideological bias or on obtaining short-term gain at the risk of long-term loss.

Remember that there are already too many things out there. We don’t need more things. We need things that work better. Make things that inform, empower and delight. And use the right tools and technologies for the job.



About the Author

Aral Balkan is an experience designer and developer working to improve the world through design that empowers, amuses, and delights. These days, he’s organizing the Update Conference as part of the Brighton Digital Festival and makes iPhone apps such as his critically acclaimed Feathers.



About the Reviewer

Josh Clark holds a B.A. from Harvard College in Cambridge, Massachusetts. Josh is a regular speaker at international technology conferences, sharing his insight into mobile strategy and designing for phones, tablets and other emerging devices. His motto is the same for both fitness and software user experience: no pain, no gain. Josh is the founder of Global Moxie.



About the Reviewer

Anders M. Andersen holds a Masters in computer science from La Trobe University in Melbourne. His life is driven by this line from Douglas Adams: “I love deadlines. I like the whooshing sound they make as they fly by.” Through his work, he strikes to make information available to people. The biggest lesson he’s learned in his career is that if you want something done, do it yourself. And his personal message is, make sure to learn something new every day.



Workflow Redesigned: A Future-Friendly Approach

Written by Stephen Hay

Reviewed by Bryan Rieger

I AM OFTEN REMINDED OF THOSE WALL MAPS that show the territory of an office building, shopping mall, subway or city zone, along with a prominent, “You are here,” usually with a red arrow and dot marking where I’m standing. These maps show us where we are in context and enable us to get directions to where we want to go. We need both, of course: we obviously need to know where we are going, but directions don’t make sense unless we know where we are now.



Figure 10.1. While having a goal is important, knowing your starting point is essential. (Image credit: Joe Goldberg, smashed.by/joegph.)

The map makes this easy for us. “You are here” presents our current location as a given; finding our destination is a matter of looking it up in an index and finding the coordinates on the matrix. Assuming you know where you want to go in the first place, all that is left to do is travel.

Philosopher Alfred Korzybski observed that a map is not the territory: a model of reality is not reality itself. This is why most project plans do not work. I can’t help but feel we all know this deep down. Experience hammers it home in any case: traffic jams, road construction, one-way streets are all things we did not see and could not have seen on our map, things we come across while travelling.

Travel is a form of workflow. We dart in and out of traffic, take shortcuts, avoid obstacles, stop for coffee, encounter an old friend who tells us the store we’re looking for has moved to another address. Grab that map again and revise the plan.

A workflow should be fluid. It should adapt to different circumstances. It should be... responsive, if you will. We, pattern-seeking creatures and lovers of step-1-2-3 tutorials, would love nothing more than a recipe for our workflow:

- Go straight two blocks,
- Turn left,
- Go another two blocks,
- First building on the left.

That would be fantastic in a world where the map is the territory. But it’s not. Ever.

First, we often do not know where we are at the beginning of a project. There is no “You are here.” Clients often *tell* us “You are here,” but our job as designers is to ask questions to discover whether that really is the case. Even “I think we are here” is a gamble. If you want to get to New York and you are in Paris rather than Chicago, you’re in some serious trouble: the car you rented is not the best mode of transportation. You might still get to New York, but it will take much longer.

Thus, a workflow should remain fluid, because every factor influences other factors. The combination of the starting point and the end point (the goal or target) shapes our choices for getting from A to B. Plane to New York, cab to the hotel, walk to the building.

In Web design, the territory changes quickly. Maps are only marginally useful. In multi-platform design, where websites and apps will be used on various and many devices, we are confronted with multiple destinations. The list of devices to be supported

might be our map, but its usefulness in this changing landscape is limited. We cannot say, “It has to look good on Android,” because what does that even mean? You could probably come up with a list of variables that you have to deal with: screen resolution, pixel density, screen size, CSS support (or support for *any* technology, for that matter), accessibility, keyboard and mouse input versus touch input, the list goes on. And these are only technical factors; let’s not forget fuzzy (albeit still technical) variables, such as why the website looks different on the client’s BlackBerry than it does in your Photo-shop mockups. Oops!

So, how do we set up workflows for our website (re)designs in such a way that we remain flexible and adaptable, yet do not wander all over the place? A good first step is to start at the beginning. Let’s look at what we have and see where we are and where we want to begin our journey. To give ourselves optimum flexibility, let’s first focus only on the content.

Structured Content First: Platform- and Device-Agnostic Thinking

Purposely ignoring our destination, and focusing only on where we are and whether it is indeed where we want to begin, affords us great flexibility. One of the biggest problems on the Web today is that technology influences decisions more than it should. Device capabilities become reasons for implementing functionality. Content management systems are chosen before even deciding what the application will do or who it is for. “We’d like to use HTML5,” we declare.

People’s focus on technology is natural. Technology is fascinating and great fun, after all. Web technologies are toys, tools, stuff that “everyone” is using. We see cool demos and think up reasons to use the technology behind them, instead of allowing our problem to lead us to the right technology for the solution. We forget that these demos are made by people who... well, make demos. If you are in a redesign project, that is real life. You’ve got to make stuff that works. Cool is OK, but only if it works and solves a problem. The focus today is far too often on solutions instead of problems. We need to become temporary pessimists and focus on all of our problems—focus on them intently, let them incubate. To those whose business is to keep up with Web technology (um, that’s you), the answers will literally come to you in the shower.

CONTENT INVENTORY

There is a certain way of thinking about content that can help the designer (and their client) focus on what really matters about the project. And what really matters, 100% of the time, is the base content and functionality of the website or app. It's all about the reasons why people would use the product in the first place. Ignore the platform. Ignore user agents. Focus on your reasons for being. Let's call this *platform- and device-agnostic thinking*, because we don't care about those things just yet. Let's apply this thinking to the base content and functionality.

As usual, asking questions will help us think critically:

1. Why would people use this website in the first place? Come up with good reasons.... I'm not convinced. Come up with more. Now refine those reasons. They must be *true*.
2. If only one browser existed in the world, and it rendered only HTML, with default styles and all functionality being handled on the server, what is the sum of the content and functionality you would offer? In other words, what should all users be able to see and do in all situations?

Put your answers to these two questions in a list. We'll call it our *content inventory*. For each piece of content or functionality that you add to the content inventory, it must pass the *purpose test*: it must actively support the reasons you have listed in response to the first question.

When you're done (this will take a while and is a significant project milestone), look at the result—closely. It says, “You are here.”

In this chapter, we will introduce some subtle (and not so subtle) changes to a common Web design and development workflow, to make it more future-friendly. We will get a sense of how to apply our new workflow to a simple project. The project will be small and intentionally not comprehensive. It will merely illustrate how these ideas can be put into practice, rather than serving as a case study. Try out some or all of the techniques in your own projects; use what works for you, and alter or dispose of what doesn't. I have used all of the elements in this workflow in actual projects and have been very pleased with the results. I hope you get as much out of these ideas as I have.

AN EXAMPLE: “THREE LITTLE BOXES”

Let’s say we are working on a website that will teach developers about the “CSS Flexible Box Layout Module” (or Flexbox), a relatively new module for layout that has great potential in the multi-platform space.¹

The point is to keep the explanations and examples ridiculously simple. Because most CSS properties can be explained by rendering text and a few boxes, let’s call our website “Three Little Boxes.” Eventually, we might expand the lessons to include other CSS modules.

“Three Little Boxes” will consist of theory and syntax, as well as several exercises, each building on the concept of the preceding exercise. The user will be able to write their code in an in-page editor, and the “Result” space will show the result of their code. Clicking on the “Run” button will run the inputted code. Each exercise has two types of textual content: theory and assignment. The theory explains syntax and concepts, and the assignments are the exercises for the student to complete.

Let’s brainstorm a content inventory:

1. Logo,
2. Global navigation,
3. Introductory text (only on the home page),
4. Code editor (only on lesson pages),
5. Result area (only on lesson pages),
6. Progress bar (only on lesson pages),
7. Theory text (only on lesson pages),
8. Exercise text (only on lesson pages),
9. Lesson navigation (only on lesson pages),
10. Sign-up form (only on the home page).

We’ll have two page types:

- Home page (for the introductory text and sign-up form) and
- Lesson page.

¹ The CSS Flexible Box Layout Module (smashed.by/w3flexbox) is one of a few powerful layout modules currently in development. Thus far, its implementation in browsers is limited. I have written a tutorial that demonstrates the Flexbox specification using three boxes on a page: smashed.by/learnflexbox.

Let's stop here. Obviously, this is laughably simple and incomplete, but it suffices as an example. We are certainly missing a lot of stuff here (profile page, statistics on lesson completion, etc.). On a large website, the content inventory might be huge and would probably be split into several ones. But remember, this is not a functional specification. We do not have to include every detail of every item of content. We're dividing content into the largest unique groups that make sense.

We would normally assign a number or letter ID to each item of content. Because our example is simple, we'll just use the numbers 1 through 10, as listed. The design of the inventory is not important; a plain old text file will do, but you can make it as fancy as you like.

The great thing about a content inventory is that it never breaks. Anything goes. The client can yell anything they want, and you can just jot it down. We will find out soon enough what will work and what won't.

The entries in a content inventory can be embellished (for example, with descriptions or screenshots of predesigned components), but don't go overboard. I find that mapping items of content to the page types is useful, like so:

- Home page → 1, 2, 3, 10
- Lesson pages → 1, 2, 4, 5, 6, 7, 8, 9

The order of items doesn't matter at this time. We will get to that during wireframing.

CONTENT REFERENCE WIREFRAMING

One of the first steps many of us take in a Web design project (hopefully after a thorough inventory of content) is to create wireframes. Wireframes, now arguably the primary deliverable of the interaction designer, have evolved from simple line drawings of boxes on a canvas to almost full-blown Web designs, devoid only of color, images and detailed typography. Originally, they were a means through which to play with design: placement, proximity, visual hierarchy, priority of space, etc. The content was represented by crude boxes. Some designers still use them that way. But for many, their purpose has changed.

Today, wireframes are often used as a deliverable to show the client what the website will look like and how it will work before it has actually been "designed." In fact, these types of wireframes *are* designs; but they are unfinished and, thus, potentially dangerous to show to the client. If the client likes certain aspects of it, any deviation

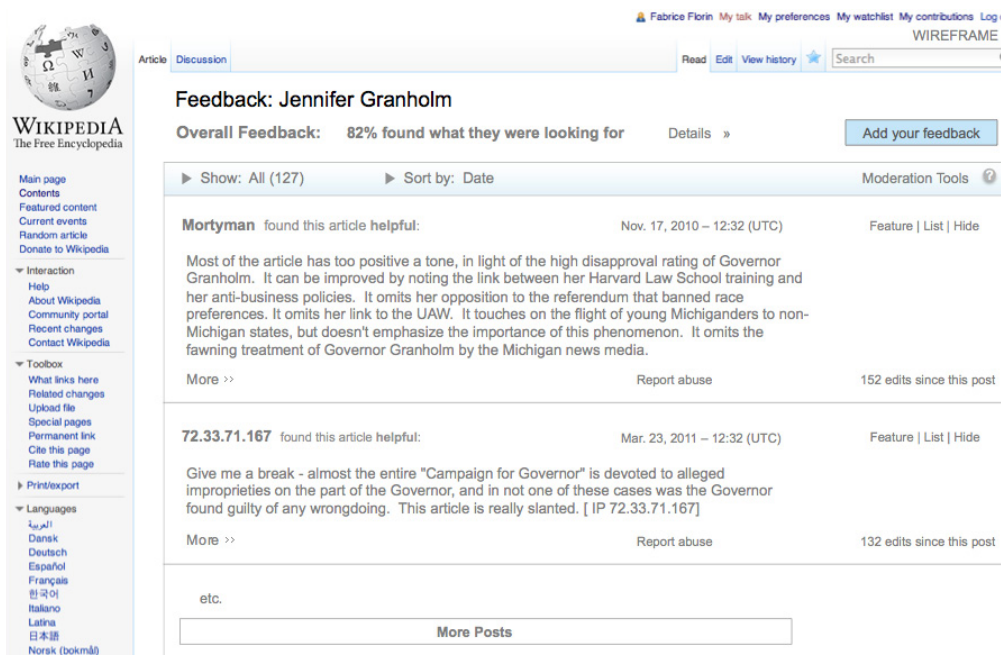


Figure 10.2. This style of wireframe is common. While most likely useful, it looks so much like an actual website that only the “Wireframe” label in the top right reveals that it is not. Both the visual design and the content get detailed treatment here, a combination that would arguably be best handled through an HTML prototype, because creating several variations for different screen sizes so early on in the project would be tedious at best. To be fair, this early wireframe of a Wikipedia page perhaps served its purpose. But its disadvantages for a future-friendly workflow are obvious. Image source: Fabrice Florin, Wikimedia Foundation, licensed under the CC-BY-SA and GFDL license. smashed.by/wkpmckp.

from it (whether in type, color, etc.) could turn them off. If they end up not liking the design itself, well, then there goes a lot of hard work.

The wireframes of long ago (which many still use) are more akin to the thumbnail sketches of traditional graphic designers. Many thumbnail sketches can be made quickly, allowing for sufficient exploration of what would work and what would not. The designer has to pay attention only to visual prominence and composition. These types of wireframes are a tool for them, not a deliverable for the client. That doesn’t mean we cannot involve the client this early on, but it would be to offer a glimpse into our process, rather than to get them to sign off on our work. Approval is less of a problem when the client has accompanied you through the design process.

I call these thumbnail-ish wireframes “content reference wireframes.” If the client does not like your content reference wireframes (if you have even shared them), then you’ve lost only a few minutes of work. The purpose is not to show the client a “design”

but to show which content will be visible in a given context, roughly indicating placement and composition, as a precursor to the visual design. This contrasts with the way many wireframes are done today, which reduces graphic design to a coloring-book exercise.

Making content reference wireframes is simple. Using your content inventory, determine which content and functionality must be on a given page, and then draw boxes, each box representing one of those pieces of content or functionality. Do this for every type of page necessary. You can name each box, or label it with a letter or number corresponding to an item in the content inventory. That's it. Absolutely unsexy, but quite effective—especially if we do not try to oversell it to the client.

This approach might sound strange at first, but we are still at the stage where figuring out whether a log-in form will appear on the page and knowing roughly where it will go would be useful. We just need to confirm these basic facts; there is no need to visualize it yet.

Content reference wireframes can be done on paper or in a drawing application. But I encourage you to create them in HTML and CSS. Yes, this does entail doing some CSS layout, but it's trivial, and the advantage is that you can create wireframes that adapt to screen sizes, enabling you to start making decisions about responsiveness early on.

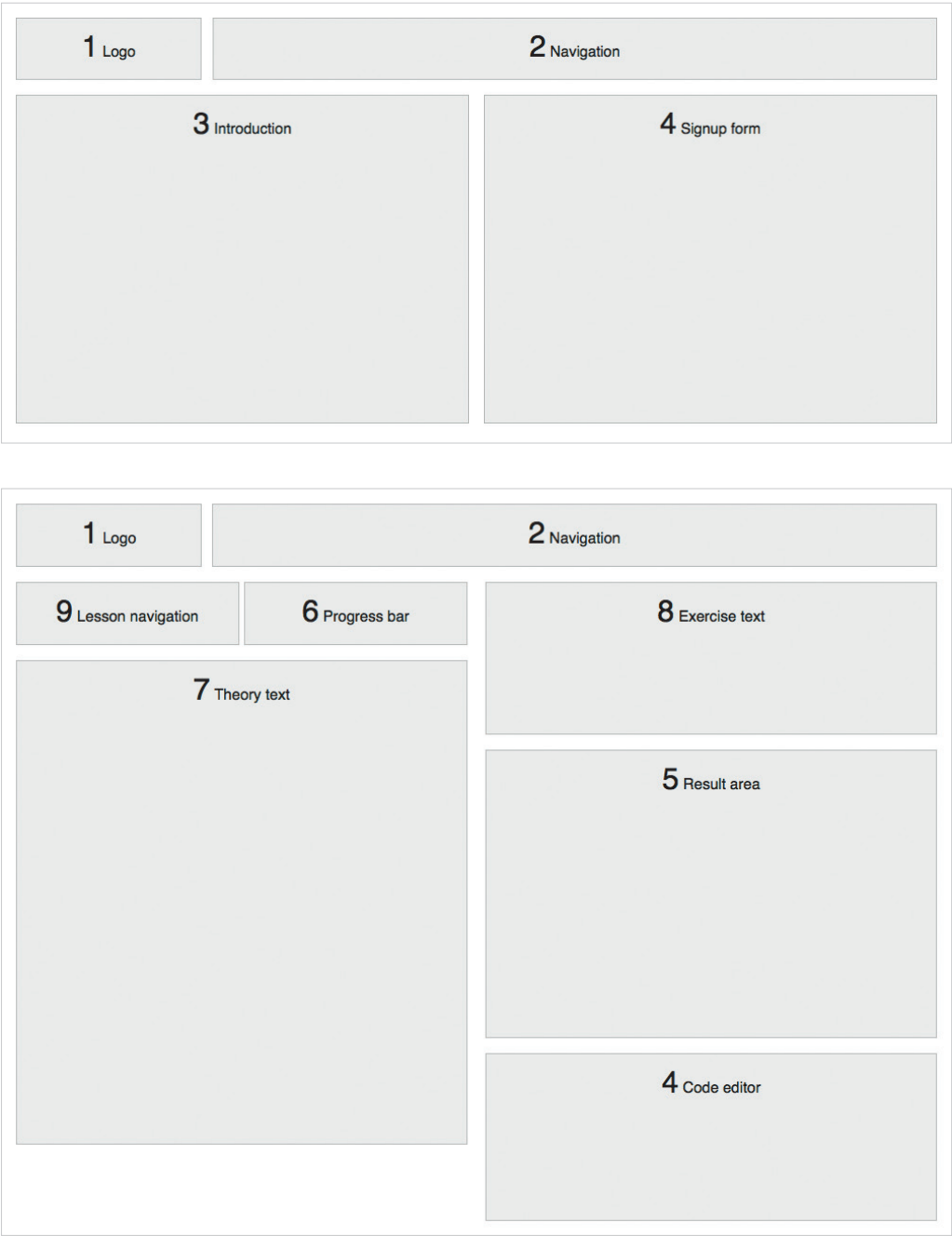
Content reference wireframes can be “designed” in the sense that a box can have a background color rather than just an outline. That's fine. Typographical features are also OK. But other than that, keep it simple.

Here's where content reference wireframes get interesting. By using CSS media queries, we can start to work on rough layouts for different screen dimensions. This will turn our two wireframes into countless wireframes, because loading them in a browser on any device is easy; we can then explore the layout possibilities and show the wireframes to the client.²

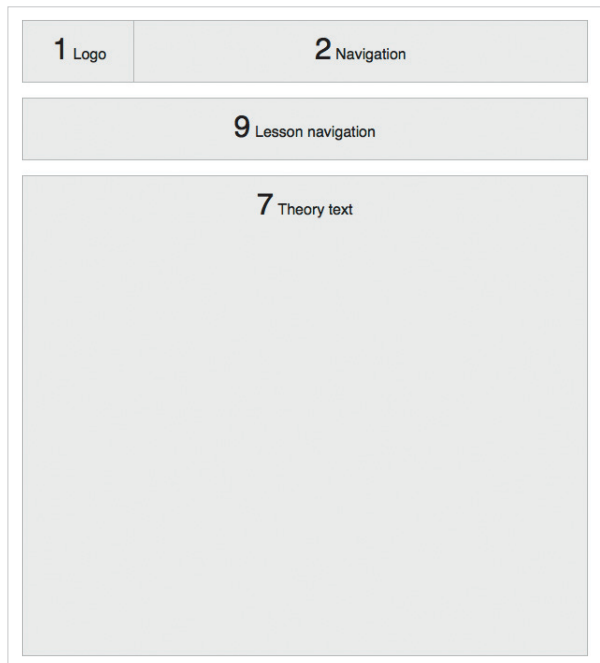
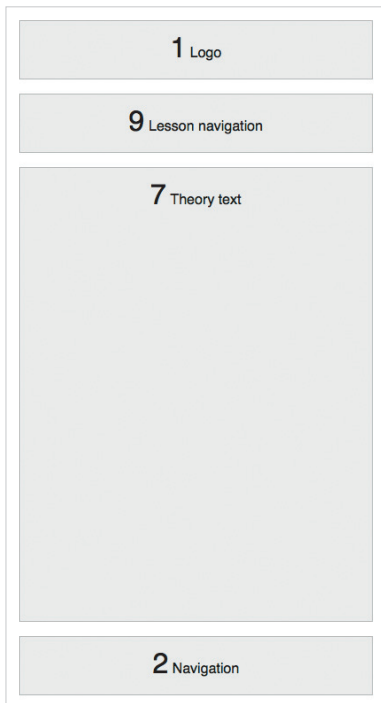
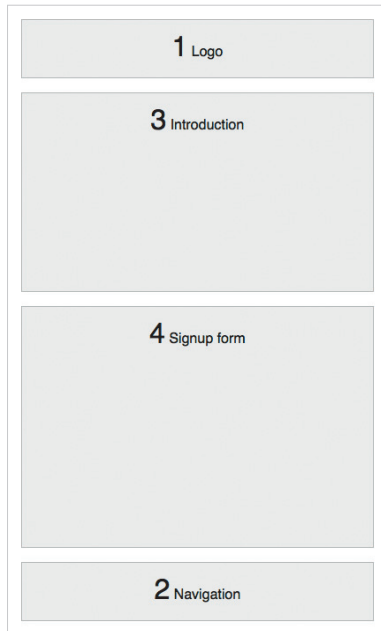
I often show clients *screenshots* of wireframes in different dimensions (to avoid giving the impression that the development process will be quick), but showing the wireframes themselves on different devices does have its advantages.

We have coded two wireframes (Figures 10.3. and 10.4.) for our “Three Little Boxes” website, one for each major page type, and each numbered according to the corresponding items in our content inventory. I have added the names of the items, too, so that we don't have to constantly refer back to the inventory:

² Ethan Marcotte's wonderful article on “Responsive Web Design” (smashed.by/rwdala) is about more than media queries, but media queries are put in such a useful context that I recommend the article to anyone who is unfamiliar with them.



Figures 10.3. and 10.4.



Figures 10.5. - 10.8.

If we change something in the inventory, updating the wireframe is trivial. We have used divs here to speed up the process, and we have added some CSS to style the boxes and arrange the layout.

Figures 10.3–10.8. show our two—yes, only two—wireframes at varying screen widths. Equipped with CSS media queries, our HTML-based wireframes enable us to explore multi-platform layouts early on. We can already start thinking about the priority and proximity of content at different screen sizes. Imagine trying to use a drawing application to render these changes in a wireframe that is as rich in content as the one we saw above for Wikipedia. Imagine the work involved for even minor variations.

STRUCTURED CONTENT DESIGN

As a starting point for wireframing, consider how our content would be designed if no layout other than a linear one were possible—that is, the kind often seen in mobile versions of websites, one item of content under another. This “mobile first” philosophy, popularized by Luke Wroblewski, is rooted in accessibility and progressive enhancement.³ You could say that the purpose is to design a base view of the app that works reasonably well anywhere that plain HTML can be rendered.

```
# Learn Flexbox with Three Little Boxes!

We've discovered that CSS layout with CSS3 Flexible Box Module can be learned by
observing the effects of the properties on a few simple boxes on the page. Once
developers observe the effects the various properties have on these simple forms, it's
easy to move on to complex layouts.

It's all about learning the basics without a lot of noise; we'll leave the creativity up
to you!

We're pretty sure you can do the first three lessons in about an hour. You can run
through the lessons anonymously, but if you'd like to save your progress, you can sign
up below.

Have fun!

<form>
  <label>What's your name? <input type="text"></input></label>
  <label>What's your email address? <input type="email"></input></label>
  <button>Sign me up!</button>
</form>

Please note that we don't share your information *with anyone*.
```

Figure 10.9. Simple structured content. Here’s the Three Little Boxes home page as I wrote it in Markdown. Converting from Markdown to HTML is easy. I use Pandoc for this purpose. Pandoc is a flexible, universal document converter: smashed.by/pandoc.

³ Luke has spoken and written a lot about the mobile-first approach: smashed.by/mfirst.

Ask yourself which content should go first? Which should go second? Which last? And how will this affect the order of our source code?

Designer and developer Bryan Rieger actually designs with text early on in the design process. This is essentially what we are talking about here: structuring and prioritizing content as if we were designing a linear document.

Learn Flexbox with Three Little Boxes!

We've discovered that CSS layout with CSS3 Flexible Box Module can be learned by observing the effects of the properties on a few simple boxes on the page. Once developers observe the effects the various properties have on these simple forms, it's easy to move on to complex layouts.

It's all about learning the basics without a lot of noise; we'll leave the

Learn Flexbox with Three Little Boxes!

We've discovered that CSS layout with CSS3 Flexible Box Module can be learned by observing the effects of the properties on a few simple boxes on the page. Once developers observe the effects the various properties have on these simple forms, it's easy to move on to complex layouts.

It's all about learning the basics without a lot of noise; we'll leave the creativity up to you!

We're pretty sure you can do the first three lessons in about an hour. You can run through the lessons anonymously, but if you'd like to save your progress, you can sign up below.

Have fun!

What's your name? What's your email address?

Learn Flexbox with Three Little Boxes!

We've discovered that CSS layout with CSS3 Flexible Box Module can be learned by observing the effects of the properties on a few simple boxes on the page. Once developers observe the effects the various properties have on these simple forms, it's easy to move on to complex layouts.

It's all about learning the basics without a lot of noise; we'll leave the creativity up to you!

We're pretty sure you can do the first three lessons in about an hour. You can run through the lessons anonymously, but if you'd like to save your progress, you can sign up below.

Have fun!

What's your name? What's your email address?

Please note that we don't share your information *with anyone*.

Figure 10.10. - 10.12. These screenshots show our HTML-structured content in the browser. The top left and bottom images show the browser's default styles in Opera Mobile Emulator and Firefox on the desktop, respectively. For the image at the top right we've started adding typographical styling.

You could even think of it as designing for a word-processing document or article or book. We are interested in the underlying structure of the content: this is a heading, this is a quote, this is a list, etc.

Ideally, we should be using real content, which is why this step follows content reference wireframing in our workflow. Once our content is structured, we can start designing typographically: applying typefaces and type sizes, determining the baseline grid, and adjusting the leading (or line height).

By the way, this can all be done in HTML and CSS. If you use plain-text markup, such as Markdown, and convert it to HTML later, you will be able to do the structured content design just as quickly as as you could in a word processor. But this way you will also have some basic HTML and CSS to use for a functioning responsive prototype.

Once we have finished this step, we will have a very basic design for small screens. Load the page on a mobile device and have a look.

Second, Enhance the Experience: Platform- and Device-Specific Thinking About Browser and Device Capabilities

If we have done it right, then our base content and functionality will work in most user agents. A word of caution if you are building a Web app that requires specific technology (such as a mapping application that needs CSS, JavaScript, images and GPS capabilities as a baseline). In this case, a linear design might not work or even be relevant. In a way, that's too bad because we will be shutting users out. Just try to make as much of the content as accessible as possible, and use your technological baseline as the starting point. Our linear design will give us an idea of how the website might look and work on some mobile devices.

But now we'd like to enhance the design and overall experience on browsers and platforms that support various enhancements. A few examples:

- A linear design is just a single column of content. We can change this for larger screens and for when the user switches between portrait and landscape mode on some devices. The layout could be made to fit more columns when more space is available. Positioning elements differently might also make sense, because users might interact differently with a larger viewport; you might want to tweak or reposition the navigation, for example. You would also have to

rethink important pieces of content, since it is no longer simply a matter of putting them near the top.

- Some devices have capabilities that we would want to take advantage of, such as camera or GPS functionality. Most of us would want JavaScript functionality when it's available (and it often is—but not always, especially on low-end devices, and plenty of those are around). What about visual enhancements such as font embedding and CSS gradients? Designing in actual browsers on actual devices enables us to test not only whether these features work (and work properly), but also the impact that these features have on performance. We would gain insight into which features would be better excluded on certain devices and platforms, while supporting them on others.
- We might even want to add, remove or alter the content itself depending on the platform or device. For example, we would certainly want the smaller versions of images to be the default on small screens and mobile devices, while serving larger versions elsewhere. Also, we would not want to offer content that is irrelevant to a particular use case; GPS-related content would only be relevant where GPS is supported, so we would add it only for those devices.

To enhance the experience of structured content, we would at a minimum list the types of devices that we would like to support, grouping them into “classes” of devices. In other words, group devices with similar characteristics, so that you focus on classes of devices rather than individual devices. We could cater only to iOS or Android, but that would be too limiting. In the end, we can tweak things to get the app to look and work just right on any device.

Don't classify according to general physical shape, such as desktop, smartphone, tablet, etc. These categories are less relevant than you might think. Instead, break down devices (and, by extension, their default browsers) according to features your app might require. Any factor can be relevant, be it touch capability, screen size, pixel density, geolocation, local storage, SVG support and so on.

Focusing on features ensures that our decisions remain relevant even when new devices come along that are hard to slot into conventional consumer and marketing categories.

Marketing categories do not tell us what we need to know (such as, does SVG support exist and perform well in this browser on this device?).

UP CLOSE: DEVICE CLASSES

For “Three Little Boxes”, we really need to pay attention to device classes. The website will be educational and will involve the user typing code in order to learn what the code does. The code will be interpreted in the browser, and the result rendered in a section of the screen.

Let’s be realistic: pulling this off on most mobile phones would be impossible. In general, here are the device classes we will be dealing with:

- HTML support: required for the text-based educational content (theory and syntax);
- JavaScript support: imperative for the interactive coding exercises;
- Large screens: not required for the text-based content, but useful for the coding exercises;
- Hardware keyboards preferable for typing code;
- The browser should also support the latest Flexbox specification (otherwise, the exercises will not work). Devices that meet all of these requirements are ideal for our project.

But we have a serious problem. At the time of this writing, the newest Flexbox specification is supported only in Chrome Canary, which severely limits our multi-device opportunities for this website. If this were a real website, we would be facing some hard choices. Assuming we do not want to write a Flexbox layout engine in JavaScript, we are stuck with Chrome Canary as the only browser in which the code for the exercise will execute.

However, according to our content inventory and device classes, we could certainly provide valuable textual content (Flexbox theory, syntax, etc.), and then provide interactive elements only when they would actually be used (i.e. when the required features are available).

Deciding the best way to do this is complex and beyond the scope of this chapter, but let’s imagine that we use JavaScript to test for support of certain Flexbox properties in the browser. If supported, then the interactive components could be added to the website.

EXPERIENCE ENHANCEMENT VS. KEY SCENARIOS

When entering the enhancement phase, we should constantly ask ourselves, “Is this functionality in its current form absolutely essential to using the website?” If it is, then it is what I and some others like to call a “key scenario”—that is, one of the primary tasks of the user on the website, which should in most cases not be considered an “enhancement.”

Does this imply that enhancements are frivolous or non-essential? Certainly not. Take the example of a simple Web app for a to-do list. The goal would be to make the app usable regardless of the platform; the user agent would only have to be able to support HTML, including basic functionality for forms. It stands to reason that a to-do app would have to allow the user to do at least a few things:

- Add to-do items,
- Edit current items,
- Mark items as completed,
- Archive or delete old items.

These are our key scenarios. Of course, we could have more (and most to-do apps do have more), but for now let’s focus on these actions, which are essential to the purpose of the app.

In a world without JavaScript, we would display a form containing one or more text fields in which the user could input one or more items, as well as a button to submit the items. The content of this form would be sent to the server, and a new page would be returned to the user with their to-do items. Each item might have a couple of options (perhaps as checkboxes), and you could perform actions on the items at the click of a button. Again, the data would have to travel to the server and back, but at least it would work.

In any browser that supports JavaScript, we would want instantaneous action: marking a task as completed would perhaps strike out the text and/or add a check mark, and the user would see the change immediately. This is neither a key scenario nor a frivolous enhancement: it’s a highly useful enhancement that adds a lot to the usability of our app. JavaScript-dependence is not necessarily a bad thing—such dependence could probably be avoided in some cases—but if it can be avoided, why not start with the basic functionality at the lowest level? Remember that most of the high-traffic

and compelling websites that we know today were built when JavaScript was not nearly as widely used. Still, these websites cater to millions of users. Building a fantastic user experience without JavaScript is possible; a great experience is about content, form and execution, not tools.

“Three Little Boxes” poses a slightly different problem. While the exercises are the primary use of the website, there is definite educational value in presenting clear information about the Flexbox specification, its syntax and how to use it. It is not all or nothing; being able to read the textual information is also a key scenario. This textual information contains code examples and reads like a tutorial.

However, the experience is enhanced in Chrome Canary (most likely on desktop and laptop computers), where the interactive components are added to the exercises. This is an example of altering content based on device and browser features. When it’s done well, users of an unsupported platform won’t feel like they are missing essential content. The user of a seven-year-old feature phone would not expect the latest Flexbox code they input to render properly, and neither should they be confronted with non-functioning interactive components.

We need to make sense of where and when to present certain content and layouts. I use a visual tool that I call a *breakpoint graph*, which helps me to map out the points where content and design might change.

BREAKPOINT GRAPHING

Earlier we looked at three things we might want to change for certain device classes and/or screen sizes:

- Design and layout,
- Functionality and features,
- Content.

The points at which these changes take place can be called *breakpoints*. They are most often set in relation to the width of the viewport in pixels, in which case they are usually implemented with CSS media queries and/or JavaScript. For example, you could say that when the viewport is wider than 600 pixels, resort to layout X.

Breakpoints are not limited to viewport widths. Any characteristic of a device class can be a breakpoint: GPS (versus no GPS), camera (versus no camera), etc. We can plot these points out on a breakpoint graph.⁴

A *breakpoint graph* gives the designer and developer an overview of the aspects of a website that will change according to certain parameters of the device. These changes can be represented by thumbnail-sized wireframes (to show changes in layout) or labels that identify particular aspects (to show changes in function or content). We can plot device classes (and even particular devices) on the breakpoint graph. Not only is this useful for the development team, but it offers the client insight into how the website will adapt in certain situations.

Making a breakpoint graph is fairly quick and easy. It looks like a timeline, with a horizontal line representing the full spectrum of parameters. These parameters could be anything: screen size, support for particular technology or even a device class. The parameters are then logically grouped and plotted above or below the horizontal line. Plotting the parameters in such a way to show their relationship to each other can be helpful (for example, a device class might correspond to a screen size), but that is not always possible. After research, once you have determined the breakpoints, each can be represented by a symbol on the line (we have used a circle in the graph below). Feature sets are represented by rectangular shading behind the horizontal line, spanning the breakpoints between which they are supported. Thumbnail sketches of the layout at different screen sizes can be included under the corresponding breakpoints.

Be aware that setting breakpoints for particular devices, while sometimes a requirement, can cause problems. The changes you make for a platform might look good and work well on certain devices, but perhaps not on others with similar features. Plenty of websites today use device detection for this reason.

What we're doing here is planning for progressive enhancement. The goal is to provide the best possible experience in every device context. We do this by providing the most basic experience for the most basic devices (in some cases, with just some HTML) and enhancing the experience with features as increasingly sophisticated devices and browsers support them. Progressive enhancement minimizes the risk that the product won't work at all on major classes of devices. Not everyone is on iOS. This design planning is essential. It provides valuable information to the designer and developer and gives them a chance to experience the fun, freedom and speed of responsive prototyping.

⁴ Stephen Few's "Bullet Graph Design Specification" was the inspiration for my breakpoint graphs: smashed.by/bullgraph (PDF). See the Wikipedia article for more information: smashed.by/bullgraphwiki. Breakpoint graphs and bullet graphs differ, however, and I have not (yet) written a formal specification for breakpoint graphs.

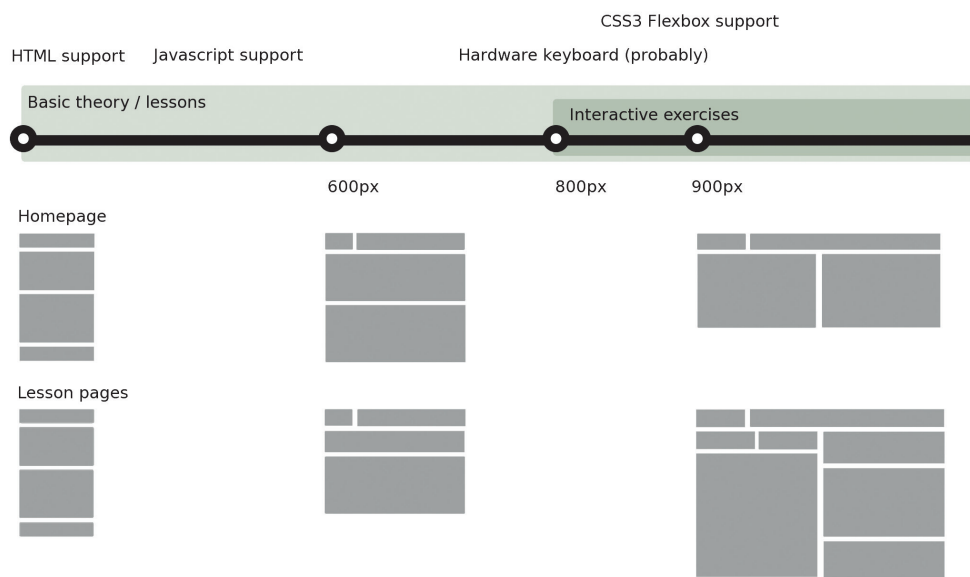


Figure 10.13. A breakpoint graph for "Three Little Boxes."

DESIGNING IN THE BROWSER:
ADAPTIVE AND RESPONSIVE PROTOTYPING

You may have heard of the practice of designing in the browser. You may have thought, as I certainly have, “That sounds fantastic, but how can I do that quickly when the design hasn’t been approved yet? Isn’t that the same thing as building static templates during the design phase? Making a mockup in Photoshop is much quicker.” This is the dilemma: we want to produce something that the client can see and approve, but we do not want to start building the actual website yet.

I have struggled with this problem for a long time, and the answer varies according to the project. I have found that if I start in HTML during the wireframing phase, then simply extending those wireframes into a full-blown visual design is much quicker and easier than having started in an image editor. The same advantages that apply to HTML wireframing apply here: you get to work in the target technology, and you can evaluate (and show to the client) variations such as changes in viewport width.

We can view these HTML mockups as an evolution of the Web-based wireframe. In this case, the wireframe gives us an HTML base to build upon. We simply take the structured content design, plug it into the wireframe, and we are well on our way to having a responsive design—right in the browser! This has the added benefit of

enabling us to test the design on actual devices very early on. As technologies evolve—high-density displays, for example—and widen the gap between the Web experience and static images, the need for designing in the browser will increase.

We have seen that our basic “linear” structured content design is best done in HTML and CSS, and the reason is because we are going to start fleshing out that basic typographical design with CSS. We will load this file (or the multiple files, if you have them) into several browsers on several devices so that we can see what happens and adjust accordingly.

This might sound like ad-hoc design, but it doesn’t have to be. I am not suggesting that you skip the tried and true steps in graphic design of thumbnail sketching and rough sketching. We should always think and sketch before executing on the computer. Please, keep doing those things. I am suggesting to execute the design or design proposal in Web technology, rather than in an image-editing program such as Photoshop.

Blasphemy? I think not. Our medium affords us the luxury of instant publication. Anyone can create a Web page in seconds. We could not do that in print; when work comes off the printing press, there is always a slight tension as we go to check how everything has turned out. Why do we insist on designing static images, having clients approve them, and then producing the very same work again in HTML? Perhaps we think designers cannot or should not code. While debatable, that is not necessarily an obstacle. In the worst case, designers could sketch on their own and then sit down with a front-end developer to produce a design in the browser.

To put it simply, we can skip the image-editing apps for designing and use them solely to edit images, which makes sense. Producing our design in HTML and CSS brings the following benefits:

- The designer sees where to tweak the design to make it look better in certain browsers and on certain platforms. As a bonus, they learn details about Web development along the way.
- The developer sees where things go right and wrong from a technical stand point. As a bonus, they gain insight into the graphic design process (unless they are the designer).
- The client and other stakeholder see—and learn to accept—the differences between browsers and platforms. In seeing their project come to life, they start to understand just a wee bit more about how the Web works and come to appreciate that the power of the Web lies in content being viewable on all devices.

Showing the client screenshots of these HTML prototypes is helpful. I will take screenshots in different browsers, viewport widths and/or devices⁵ and say to the client, “Here are some images of how this design could work in different scenarios.” Presenting them as images first helps to keep the discussion on design issues and avoid in-depth analysis of particular pieces of content. But if the client does want to change the style for, say, a heading, you can do it in CSS and take new screenshots very quickly. Imagine making those changes in several different Photoshop documents—not fun! The client won’t notice (or care) that these images have not come from Photoshop.

Sharing prototypes as images is not intended to deceive the client. Rather, it avoids giving the impression that you are further along in development than you actually are, which is important because the focus should remain on the design at this point; you will get to usability and implementation later. Actually, you *are* further along than you otherwise would be, but your client should not be concerned with this. This is an advantage to you and your team in following a technologically consistent workflow. The client will also be pleased that, when the website is finished, it will look almost exactly like the wireframes.

How you create the design prototypes is up to you. Some static website generators can

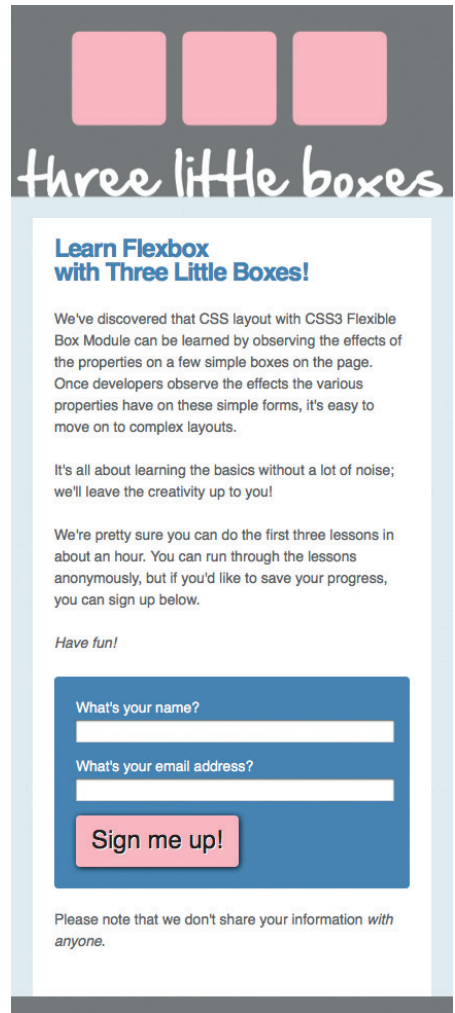


Figure 10.14. A Taking our plain, structured content in HTML, we have created a design for it in linear form. Based on the design sketches, we have started adding styles to it. Then, referring to our breakpoint graph, we can use media queries to start coding the design for increasingly larger screens and browsers that support our desired features.

⁵I cannot stress enough the importance of testing in actual browsers on actual devices. When possible, even take screenshots on the devices, because simply resizing a desktop browser window seldom emulates what happens on a particular device well enough. Emulators do not always tell the whole story either, but they are still better than a resized desktop browser.

be helpful: content is submitted in Markdown files, which are piped through templates to create a static website. In the end, these are merely tools; you will still have to make a few HTML templates and CSS files for the design. If you have already made the initial wireframes this way, then it is usually a matter of piping the content into the various boxes on the page and applying CSS based on your design sketches.

Many designers effectively use CSS meta languages and/or preprocessors or compilers such as SASS and LESS to make significant changes to the CSS in relatively short time. Combined with version control, these tools can make designing in the browser quicker, easier and more fun than working in an image editor. Trying out the client's (or your own) Great New Idea becomes trivial, and the process can help keep the project on track and on budget by cutting down on the effort required to explore new ideas.

Once the client has approved the basic designs, you can fine-tune a bit and present some important pages as full-blown HTML prototypes as opposed to screenshots. This is easy: you are most of the way there because your design is already in HTML. Use this to your advantage by focusing on contingency design: error messages and state changes and the like.

You now also have the time to consider these things, because you have skipped the step of visually parsing the Photoshop file and inferring design subtleties from it.

Let's not be too harsh on image editors, though. Applications such as Photoshop are still fantastic for quickly coming up with design and color studies and for developing visual assets. Use Photoshop to quickly experiment with things like color, image and some of the typography. Photoshop works well for putting together mood boards of these elements in the conceptual phase, before producing design impressions.

In the end, designing in the browser in this way brings huge benefits:

- Little to no time is spent in an image editor. All time spent on the design is time spent on code—code that could be used in whole or in part during development of the website itself. Using a version control system such as Git will allow you freedom to experiment with your design in code without worrying about the permanence of changes.
- Executing most requests from the client is usually trivial (except perhaps for changes to the layout). And the changes will be immediately visible in the various viewport widths, etc. Making such changes in an image editor would be extremely time-consuming—and not fun.

- Showing the client changes in state (for example, logged in versus logged out) is possible with a minimum of effort.
- If you are attentive to the quality of your code in the prototyping phase, a lot of it will be usable in the production website.

Building a prototype as both a design impression and a working prototype might seem counterintuitive, but I urge you to try it out a few times to get used to the idea. It really can save a lot of time, and you never have to leave your target technology. It yields few surprises compared to the traditional process of designing websites.

A New Way of Thinking, A New Way of Designing

Let's see our new workflow, then:

1. Create an inventory, listing our content and functionality.
2. Create quick wireframes in HTML and CSS, labelling the boxes in the wire frames according to the content in our inventory.
3. Put some real content into a structure, in much the way we would for a book or article.
4. Produce a linear design for this content.
5. Consider device classes, and create a breakpoint graph.
6. Sketch, brainstorm and conceptualize the design for the various breakpoints.
7. Combine the content from the linear design and the HTML wireframes, building the design as a prototype in HTML.
8. Show the client screenshots of this prototype, presenting them as design impressions.
9. Upon getting approval for the design impressions, (optionally) build on the HTML prototype and present it (for, say, user testing).
10. If you have done things right, most of the code can be used in the production website.

This workflow is effective for both redesigns and new designs. It is a future-friendly approach, because we are considering various device classes from the beginning. Thus, new devices that come on the market probably will not break the website.

The workflow is efficient because each step is incorporated into the following step. Both the designer and the client are consistently confronted with real-world problems, such as browser rendering.

In my experience, clients like this workflow. It does not build to a visual anticlimax the way we see with the traditional design workflow, where production rains on the Photoshop mockup's parade. It builds slowly, incrementally, and the client could be involved in the entire process. There are no surprises. Each step yields a more appealing result than the last. Things get concrete very quickly, and the client sees early on the effects of their decisions.

Some designers and developers are already working this way. For some, perhaps including you, it will take some getting used to. Try out bits and pieces in your projects today and discover what works for you. The process is probably drastically different than the way you are used to working, but it is an effective way to navigate your journey as a Web designer.



About the Author

Stephen Hay (1970) was born in Orange, California. He now lives in Leeuwarden, a city in the northern part of The Netherlands. After getting his BA in graphic design and fine art, he put his dreams of becoming a painter aside and worked his way from designer to art director in a design agency, where he specialized in identity and packaging design. Stephen made his first website in 1995 and fell in love with the medium; the Web seemed the ideal mix of design and technology. After 10 years as creative director at a front-end Web development firm, he became an independent consultant, which affords him more time to work on the things he loves to do. He loves to help people learn, and he enjoys speaking and writing about the Web. But he's not all work and no play: Stephen played baritone saxophone in a jazz quintet and enjoys sleight-of-hand magic. He is also an art and film lover. The greatest lessons he has learned in his career are to think critically, to do what you enjoy, to do it as well as you are able, and to keep learning.



About the Reviewer

Bryan Rieger was born in the year that the Beatles broke up, Jimi Hendrix had a number one and the world's population reached 3.6 billion. He is from Toronto and tends to live much of the year in Edinburgh and tries to spend the rest of it in Bangkok. He has moved around fairly regularly and has considered many places to be home, including Charlottetown, Vancouver, Surabaya, Phuket, Hong Kong, Manila, London, Glasgow and Brighton. Home is wherever he happens to be now, which fits his life motto well: "Whatever works." Bryan's education includes a little design, theater and animation at various schools, times and levels. Bryan began creating HyperCard stacks shortly after leaving art school and eventually fell into UI design. He worked at various agencies throughout the dot-com years, and after burning out in 2003 he took a sabbatical to wander Southeast Asia for a bit and dive into this thing called mobile. Ever since, he's been working on mobile operating systems, apps and websites for clients around the world. Bryan loves to travel, and his favorite color is white—it just has such a sense of possibility. The biggest lesson he has learned over his career is to question everything, always. His message to readers is, "Plus ça change, plus c'est la même chose." Be flexible, embrace change...



Becoming Fabulously Flexible: Designing Atoms and Elements

Written by Andy Clarke

THERE ARE THREE WORDS that I believe sum up working on the Web right now for many of us. They are:

- Responsive.
- Web.
- Design.

It was Ethan Marcotte who gave that combination of fluid grids, liquid images and CSS3 media queries a name, and after he did, developers all over the world have been publishing some pretty fabulous boilerplates, packages, scripts and template solutions to many of the challenges created by this thing Ethan called *Responsive Web design*.

Responsive Web design isn't simply about how designers and developers use technologies like CSS3 media queries, though. It isn't about how we handle serving different-sized images or tackle data tables responsively either. These are just technical challenges; becoming responsive isn't merely about overcoming technical problems. It doesn't mean learning new languages or how best to use them either. I wish it were that simple.

But it isn't.

Designing responsively is much, much more difficult.

Like it or not, responsive Web design challenges everything we know about Web design for everybody involved in the process. That's why, in this chapter, I'm going to demonstrate how the old ways of designing are no longer relevant, and I'll introduce a new way of designing responsively that has worked for me and my clients.

CORE TO WHAT YOU DO

I don't think I'm alone in believing that responsive Web design represents a fundamental shift in what it means for us to design for the Web. Andy Hume wrote:¹

"For me responsive design is another example of web design getting back some of its Dao. That's why it's not an added extra or a feature. It's core to what you do."

I agree with Andy, and although I know some people see responsive Web design as just a trend like many others, I believe that it's perhaps the biggest and most important change in Web design since the Web began. I wrote once:²

¹ Hume, Andy. "Responsive by Default," smashed.by/respdef

² Clarke, Andy. "I Don't Care About Responsive Web Design," smashed.by/respcare

“Anything that’s fixed and unresponsive isn’t web design anymore, it’s something else. If you don’t embrace the inherent fluidity of the web, you’re not a web designer, you’re something else. Web design is responsive design. Responsive Web Design is web design, done right.”

I wasn’t exaggerating. I stand by that, even if some people think I was getting a little carried away. I believe that considering how a design will respond as it’s displayed across a myriad of device sizes, shapes and capabilities is one of the most important aspects of a Web designer’s job today.

HOW, WHEN AND WHY?

Responsive Web design changes what we make for the Web, so that means how we make it changes, too. It’s not just designers and developers who are affected. Responsive Web design affects everyone who thinks up, designs, builds, pays for or uses the Web.

- Content strategist? You.
- Interaction, experience or graphic designer? Yup.
- Front- or back-end developer? You too.
- Boss, client or customer? You betcha.
- User? You too, and in a really good way.

Responsive Web design asks more questions than it offers answers. It affects the working relationships and interactions between everyone involved in every process—from content specialists to designers and developers of all kinds, to our bosses and customers who ultimately approve and pay for the work that we do.

It challenges the *how*, *when* and *why*. These challenges won’t always be easy to overcome and the changes they bring won’t always be popular. There will likely be plenty of resistance from people who can’t or won’t see the need to adapt.

EASY. REALLY EASY.

In 1998, when I started my own tiny design studio, the biggest technical issues I faced were the differences in how Internet Explorer 4 and Netscape 4 rendered my designs. Truth be told, even though I had to work with immature technologies and around terrible browsers, I had it easy back then. We all did.

Really easy.

Here's how I worked. I bet you did the same. Maybe you still do?

1. I'd make a design in Adobe Photoshop or Macromedia Fireworks—a single design that I intended for everyone to see, no matter which browser they used or how big their screen was.
2. Then I'd show that design—as a comp or mock-up (I call them static visuals because they're flat and non-interactive)—to my clients.
3. After that, I'd make changes based on their feedback. I'd rework and export new static visuals. When they were approved in a second or third or fourth round of approval, I'd chop up the design into HTML and CSS, publish it and be home in time for tea and reruns of *Animal Magic*.³

Back then, making a single design was acceptable because, for us and our bosses and customers, the Web was pretty much only accessed from the desktop. But we can't make just one design today. It isn't that simple anymore because whereas once we had two important desktop browsers, today we have potentially hundreds on all kinds of devices that people use to access the Web.

ACCEPT THE EBB AND FLOW OF THINGS

Even as far back as the turn of the millennium, we should have realized that the Web is fundamentally different to other media and that part of its uniqueness is our lack of control over how people view and interact with the content and services we provide on it. The truth is, we should have always designed for the flexible nature of the Web.

Actually, some people did realize, and they tried to warn us of our foolish ways. In 2000, John Allsopp wrote what many consider to be the most important article about Web design ever written: "A Dao of Web Design." In it, John wrote:⁴

"The control which designers know in the print medium, and often desire in the web medium, is simply a function of the limitation of the printed page. We should embrace the fact that the web doesn't have the same constraints, and design for this flexibility. But first, we must accept the ebb and flow of things."

³ smashed.by/anmag

⁴ Allsopp, John. "A Dao of Web Design," smashed.by/dao

Good advice.
Did we take it?
No. Silly person.
What did we do?

We fooled ourselves into thinking we were in control. We tried to exert the same levels of control that we had over print. We largely ignored that troublesome flexible Web and we tried to make the Web fixed instead. Here's how.

DESIGNING FROM THE EDGES OF A CONTAINER

When I moved from print to the Web, the Web was new. There were no rules. So I, and designers like me, imported principles, ideas and tools we knew well. The people who hired us were new to the Web, too, so they brought with them the processes from pre-press and print that had worked for them.

I used Photoshop to design websites, and I began every project by making a new canvas—I drew a landscape-format rectangle—then I filled it with stuff.

Those first rectangles measured 640 pixels wide by 480 pixels high because that was the resolution of most PC monitors at the time. Designers were used to being in control, so we went to extraordinary lengths to maintain it and did crazy things like scripting browser windows to snap to a size and letter-boxing content using five or more frames in a frameset.⁵ That 640 pixels rectangle was tight because—if you can remember back that far—clients didn't want people to scroll (at all) and everything had—just had—to appear above the fold. 640-pixel screens soon gave way to 800 (by 600), so designers drew bigger rectangles to match. The feeling of wide open space was intoxicating, but it wasn't long before even those big rectangles felt cramped and we started to look beyond 800 to 1024 (by 768). Over and over again, at every step up, we studied server logs and statistics and agonized over whether it was safe to go wider.

It might seem strange today looking back, but in 2005, fixed versus fluid layouts was a really hot topic. No, no, it really was.

I interviewed Jason Santa Maria about his redesign of *A List Apart*, the website for people who make websites.⁶ I wondered why Jason had decided on fixing *A List Apart*'s width to 1240 pixels, rather than implement a fluid layout based on percentages. Jason replied:⁷

⁵ Dreamweaver FAQ: Using Frames To Align Page Content, smashed.by/tut

⁶ Santa Maria, Jason. "A List Apart Redesign," smashed.by/mari

⁷ Clarke, Andy. "A List (taken) Apart," smashed.by/andy

“ALA has always tried to be one of those sites at the front of the pack. We don’t support 800 × 600 anymore, nor do we 640 × 480. Do you? People flipped when sites stopped supporting 640 × 480... now no one says a word. Things change. Trust me, you are going to see more sites stretching out their legs and putting their feet up.”

People were outraged at the time by that new, wider A List Apart. Well, not really outraged.

Jeremy Keith (who’s nothing if not consistent) echoed Jon Hicks’ concerns⁸ about the apparent dichotomy between “designer sensibilities vs. user preference.” Jeremy said:⁹

“Arguing about 640, 800 or 1024 pixels is like arguing about whether Pepsi tastes better than Coke when really, a nice glass of water would be much more refreshing. The numbers game is a red herring. A big fixed-width red herring.”

For years we fooled ourselves into thinking that because 640, 800, 1024 or above were commonly used screen resolutions, we could design for those fixed dimensions. We desperately clung to fixed-dimension design because the reality—that the Web is a fluid medium with no common canvas size, no edges—was simply too daunting.

THE WEB HAS NO EDGES

Because today we can’t predict the size or format that our content will be viewed in, the Web effectively has no edges. So, what do designers do?

We create them. When 640 turned to 800 and then to 1024, we used progressively larger fixed dimension canvases as the starting points for our designs. We drew bigger and bigger rectangles and we filled them with our content from the edges of the canvas working inward.

When Steve Jobs (bless his soul) pulled the first iPhone, with its Safari browser, from his pocket in 2007, it had two orientations within one device. The iPhone’s pan, pinch and spread gestures also reinforced the fact that edges are irrelevant. The Web changed forever that day, thanks to that one device. So, what did designers do?

We drew a small rectangle. 320 pixels wide and 480 pixels tall.

⁸ Hicks, Jon. “Is 1024 OK?,” smashed.by/hicks

⁹ Keith, Jeremy. “A List Too Far Apart?,” smashed.by/runaway

After Steve sat on a couch and unveiled the first iPad, we sat and made our canvases 1024 by 768 pixels again. We stuck with canvas-in, fixed-dimension design because that's what we knew and that's what our bosses and customers expected.

Have you heard that definition of insanity: doing the same thing over and over while expecting a different outcome? That's how we've worked until now. The rectangles grow, then shrink, then grow again. All the time they multiply and all the while our thinking stays the same.

BRINGING A KNIFE TO A GUNFIGHT

We shouldn't be too hard on ourselves. The software tools we've cherished have done their level best to keep fixed-dimension alive. Think about it. What's the first action we take when we start a new design in Fireworks and Photoshop?

File → New

⌘ N.

Then we give that new document a fixed canvas size.

Ask yourself, is the reason why so many websites are fixed-width and centered a direct result of our clients seeing, then signing off on, fixed-width visuals?

Although software vendors like Adobe have incorporated Web design tools into their products, there's nothing in those tools that can help us design responsively. They can't even help us demonstrate hover states or other transitions, let alone demonstrate the ways that flexible layouts affect how type and other elements reflow. Jason Santa Maria again:¹⁰

“Every element on a webpage has the ability to affect the layout of other elements. We should be able to specify what actions to take (float, clear, wrap, etc.) when that happens. Additionally, a browser window is a fluid canvas; desktop design apps only work with a fixed canvas size, making comping a fluid/flexible design little more than a guess.”

Our current software tools, in particular Photoshop and Fireworks, are simply incapable of handling the demands of responsive design. They're bringing a knife to a gunfight.

¹⁰ Santa Maria, Jason. “A Real Web Design Application,” smashed.by/rwdapp

THAT AWKWARD FACT

For the longest time, the hardest part of designing websites wasn't dealing with the fact that people experienced our designs on different-sized screens. That's because most of time we *didn't* deal with it.

Instead, we ignored that awkward fact and carried on believing that if the majority of people had a screen big enough to display our design, then everything would be OK.

When the iPhone made us realize that our work wouldn't always be “(best) viewed in a modern browser at 1024 pixels or above,” our first response was to create an iPhone-specific design, in addition to one for the desktop. This immediately doubled the time for design, feedback, corrections and approval.

As more smartphones, eReaders and tablets appeared, it was common for bosses and clients to ask for specific versions of a website or application for those, too.¹¹

- ☐ iPhone
- ☐ Android
- ☐ iPad
- ☐ Kindle Fire

But designing separate versions might mean three, four, sometimes five times the design work. Three, four, five times the length of the approval cycle. Three, four, five times the meetings!

With so many characteristics, capabilities and sizes to deal with now, making multiple fixed-dimension designs isn't appropriate, economical or practical, nor is it the best use of a designer's time and talent. Creating several fixed-dimension designs wouldn't benefit us technically either, when we come to convert those visuals into code. As Stephanie Rieger writes in “The ‘Trouble’ With Android”:¹²

“Designing to fixed screen sizes is in fact never a good idea... there is just too much variation, even amongst ‘popular’ devices.”

So, instead of thinking about individual designs for separate devices, we should start to think, as Ethan Marcotte suggests, about a single design that has many facets, which lie along what we should think of as a *fluid continuum*:¹³

¹¹ See chapter seven for an overview of common smartphone screen resolutions and pixel densities.

¹² Rieger, Stephanie. “The ‘trouble’ with Android,” smashed.by/trouble.

¹³ Marcotte, Ethan. “Responsive Web Design,” smashed.by/rwdala.

“Rather than tailoring disconnected designs to each of an ever-increasing number of web devices, we can treat them as facets of the same experience.”

I love this way of explaining design in a responsive context— many facets of one experience—because it echoes so well the “ebb and flow” that John Allsopp described in his Dao. I also like to think of design as a river, starting narrow at the source and becoming wider towards the sea. We can never predict where along its course someone will experience our design, so we must make it flexible.

This raises questions. If we shouldn’t make separate fixed-width designs, what should we make? What should be our process? How can we design for a Web without edges?

“Hey, This Is What I Made.”

It’s normal today for us to make one or several fixed-width static visuals to represent the website we’re creating. There are several reasons why we do this.

FOR DESIGNERS

Photoshop and Fireworks are fabulous tools for creative experimentation and for helping us solve problems and define a project’s visual direction. It’s sometimes even the act of using Photoshop and Fireworks that can help us —sometimes accidentally—arrive at a result that would be difficult to accomplish by any other means. Finally, the result is something we can point to and say to our bosses and customers, “Hey, this is what I made.”

The problem with static visuals—as I’ve written many times—is not simply that they are bad at conveying the realities of a modern interactive website: it’s that, when used incorrectly, they set the wrong expectations. Because designers use them as a way of getting sign-off for a design, our bosses and customers no doubt expect that the finished website will look precisely like the visual. After all, that’s why we showed it to them.

But not all browsers can render a design in the same way. All have different capabilities that the static visual ignores. Therefore, it was nobody’s stupid fault but ours that we then spent hours hacking our HTML and CSS or applying JavaScript patches in a vain attempt to make our website look the same as the visual in every browser.

That’s why, in the future, designers need to accept that what they include in a static visual may not appear the same to every person or on every device. They must also learn to set the right expectations from static visuals, and make it clear to their bosses and customers that the visuals represent only how a website *might* look in some browsers on some devices.

FOR DEVELOPERS

Static visuals can be useful tools for developers tasked with implementing a design in code. Visuals act both as a blueprint for developers and as a benchmark for bosses and customers to measure how well the final website matches the design vision as expressed through that visual.

But from now on, developers must accept that when they receive a static visual, it represents only a guide for how a design might look. Developers must learn to make layout decisions, including how design layouts adapt to different screen sizes and devices.

FOR CLIENTS

It's conversations around these static visuals and between designers and our bosses and customers that can be the most valuable parts of a creative process. If the correct expectations are set—and bosses and customers understand the meaning of what they're looking at—then static visuals can be enormously helpful in describing intent and soliciting comments and constructive criticism.

The trouble is, the correct expectations are rarely set, and the more we ask of static fixed-dimension visuals, the less appropriate they become and the less effective they are as tools through which to communicate design. In particular, static fixed-dimension visuals don't work in a responsive context because they make it easy for people to confuse design with layout.

Design Isn't (Always) Layout

Has a client ever said to you, “I don't like the design”?

If that has happened, did you dig a little deeper and discover that it wasn't the details of your design they objected to? I'll bet it wasn't the typefaces or type treatments you chose. They went unremarked. It wasn't the way you'd used color, either. Nor your finely crafted line work, borders or shading. Perhaps it was, “The sidebar should be on the left, not the right.”

In other words, your client was commenting on layout but expressing their criticism of the design.

It's not unusual for bosses and customers to include layout in the same conversation as other aspects of a design: typography, color and texture. Designers do it, too, because for years we've been making and demonstrating fixed-width visuals that have included all these things.

From now on we need to accept that design and layout must be separate from each other. The design of components and the arrangement of those components horizontally and vertically on a grid are now two different issues. Whereas the layout's arrangement of components will undoubtedly be different across screen sizes, the design of those same components will almost certainly transcend layout.

DESIGNING COMPONENTS FIRST

Think for a moment about the components that you place in almost every design. Your list will almost certainly include:

- boxes;
- data tables and other data panel types;
- images (content and iconography);
- interface elements (carousels or scrollers, and so on);
- navigation (several levels);
- type.

Pay attention. This is an important part. Design in the absence of layout becomes the styling of these components. How these parts look is now what we mean by the design. More specifically, the look and feel of these parts is what designers can and should work on first, long before any thought is given to layout. I like to think of the approach as designing page components at an *atomic* level first.

Now, I know that the idea of designing components like this, out of context and without layout, might sound strange—particularly if, like most of us, you've been used to designing Web pages the traditional way. But, truth be told, we've been abstracting design ideas like this for the longest time.

MOOD BOARDS

Mood boards have long been a fabulous way to collect inspiring materials. Whether we keep collections on our computers in applications like Evernote, use online services like Pinterest, or simply stick them to a large mat or mounting board, it's the combination of different elements that expresses a mood.

Our mood boards might have a contemporary, chintzy or traditional feel, but however we choose to name it—a look, mood or personality—the important thing to remember is that we’re describing style. Of course, we don’t have to start a design process with paper, scissors and glue. We can keep our hands clean if we prefer.

Now, when we design responsively, we could start by styling components as individual, but connected, parts of a design.

Adobe Photoshop and Fireworks may not be the perfect Web design tools, but they are still valuable for designing every kind of component. To help you break away from thinking about layout at this stage, try starting with a Photoshop and Fireworks canvas that’s 10,000 by 10,000 pixels or bigger. On this design sheet, group elements into headings, body text styles, form elements, buttons, error messages and more. When needed, move components around the sheet to see how they look in relation to each other.

Trent Walton wrote:¹⁴

“Web designers will have to look beyond the layout in front of them to envision how its elements will reflow & lockup at various widths while maintaining form & hierarchy. Media queries can be used to do more than patch broken layouts: with proper planning, we can begin to choreograph content proportional to screen size, serving the best possible experience at any width.”

The separation of design from layout that we achieve when we design at an atomic level is important because it helps everyone—the designer and their boss or client—focus on the details in a design while setting no expectations for how components will ultimately be arranged across various screen sizes or devices.

For the last several months, I’ve been using design sheets as a way to demonstrate designs to my clients. I’ve found them to be extremely effective, although they require a few minutes of explanation to clients who haven’t experienced this way of working before. We can ask everybody involved, “Is this the mood of the design we’re striving for?”

Design sheets are a fabulous tool for clarifying a client’s aspirations very early in the design process. Instead of encouraging vague statements such as “I don’t like the design,” the approach helps us to be precise when we discuss design.

¹⁴ Walton, Trent. “Content Choreography,” smashed.by/trent.

- How clean or minimal should the design be?
- How will we use color to convey actions and intent?
- How many steps in tone or contrast will there be?
- How will typefaces be used to distinguish types of content?
- How many increments in the typographic hierarchy will there be?

Designing components this way can help us find answers to these questions and gives everyone involved options and opportunities to change their minds before any major investment is made in design or development.

Using design sheets can also allow us to continue with design iterations, even while other aspects of a website's development are in progress. This helps us to break the archaic workflow that we imported from pre-press and print to the Web. You know the one:

Plan → Sign-Off → Design → Sign-Off → Development.

By working in a more flexible way, design can now take place at various stages, sometimes even before planning has been completed or after development has started. This way, design becomes more deeply integrated and is continued throughout the development cycle.

Style Tiles

The abstraction of design into the look and feel of elements is something that designer Samantha Warren has been thinking about, too. She calls her technique *style tiles*. Samantha explains:¹⁵

“A style tile is a visual “tray” of paint chips, fabric patterns, and color choices that support the client’s goals. I have a Photoshop template with specifically masked areas where [...] I display samples of button styles, navigational treatments, and typographic possibilities.”

For Samantha, style tiles are an effective tool for communication as well as component-level design.¹⁶

¹⁵ Warren, Samantha. “Style Tiles as a Web Design Process Tool,” smashed.by/styletiles.

¹⁶ A more detailed overview of Samantha's design process is presented on www.styletiles.com.

BOOTSTRAP

For others, page components can be building blocks for future designs. Take Mark Otto and Jacob Thornton, a designer-developer combination who work for Twitter. They're the pair behind Bootstrap.¹⁷ The Web might not have edges, but it does have content. We express that content through appropriate HTML elements:

- headings, and their levels 1 to 6;
- paragraphs, block quotes and text-level elements;
- lists: definition, ordered and unordered;
- media: figures and captions and images;
- tables;
- forms: buttons, elements and labels.

As well as Twitter's own grid system, Bootstrap includes design defaults for "typography, forms, buttons, tables, grids, navigation, and more." Bootstrap is interesting for a number of reasons. Not only is it a solid development boilerplate, but the fact that it includes a comprehensive list of HTML elements, along with some default styles, makes it an ideal starting point for a new design.

DO YOU DRIBBBLE?

Dribbble¹⁸ is a website where designers share small screenshots (maximum 400 by 300 pixels) of designs they're working on.

Take a trip through Dribbble's pages and you'll find plenty of examples of component-level design and the details that designers sweat. You'll find navigation designs and treatments for inline images. If forms are your thing, you'll find styles for every kind of form element, plus button styles for every state. Typography isn't missing either, with as many examples of typefaces and type treatments as you could want.

Dribbble's shots are the perfect example of showing designed components without layout. They show how we can design and then demonstrate to our bosses and customers a design direction in a format that doesn't confuse style with layout.

¹⁷ Twitter Bootstrap, <http://smashed.by/boots>,

¹⁸ smashed.by/drbb



About Bootstrap

Brief history, browser support, and more

History

Engineers at Twitter have historically used almost any library they were familiar with to meet front-end requirements. Bootstrap began as an answer to the challenges that presented. With the help of many awesome folks, Bootstrap has grown significantly.

Read more on dev.twitter.com.

Browser support

Bootstrap is tested and supported in major modern browsers like Chrome, Safari, Internet Explorer, and Firefox.



• Latest Safari

What's included

Bootstrap comes complete with compiled CSS, uncompiled, and example templates.

- Javascript plugins
- All original .less files
- Fully compiled and minified CSS
- Complete styleguide documentation
- Three example pages with different layouts

Figure 11.1. At first you might wonder why a developer toolkit like Bootstrap is relevant to a responsive designer's workflow. You might be surprised to know that Bootstrap is the perfect starting point for designing page components.

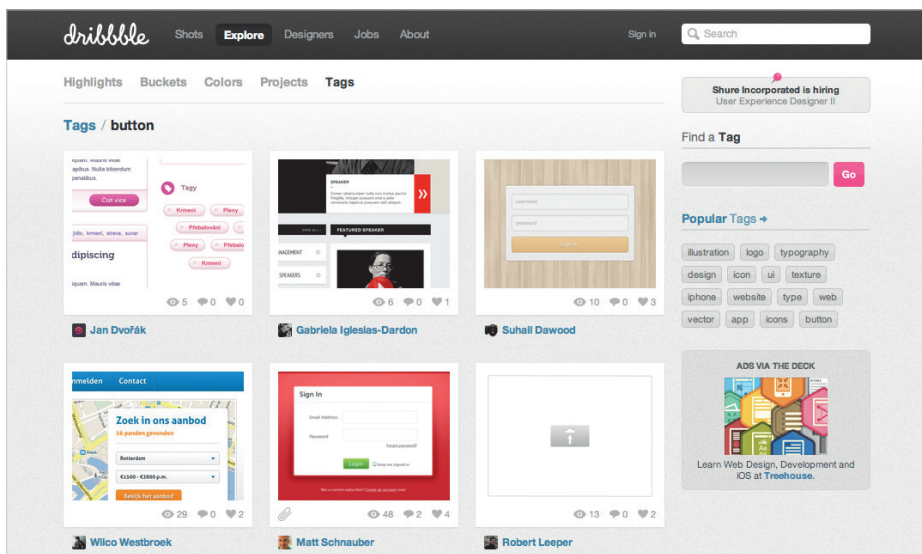


Figure 11.2. Dribbble shows designed components in a neutral environment.

STYLE GUIDES

When you work on a project in which teams of people contribute, a thorough and well-written style guide can be vital for maintaining a design’s integrity. While organizations use style guides for many purposes—publishers use them for spelling and punctuation rules, and publicists use them for setting the tone of copy in written publications—on the Web, the best style guides clearly set out a website’s design house style.

One of the best recent examples of a comprehensive set of style guidelines comes from the BBC. Its Global Experience Language (GEL)¹⁹ documents the BBC’s complete visual language as used on the Web, with detailed information on typography, iconography and the design of interface components such as overlays, accordions and carousels.

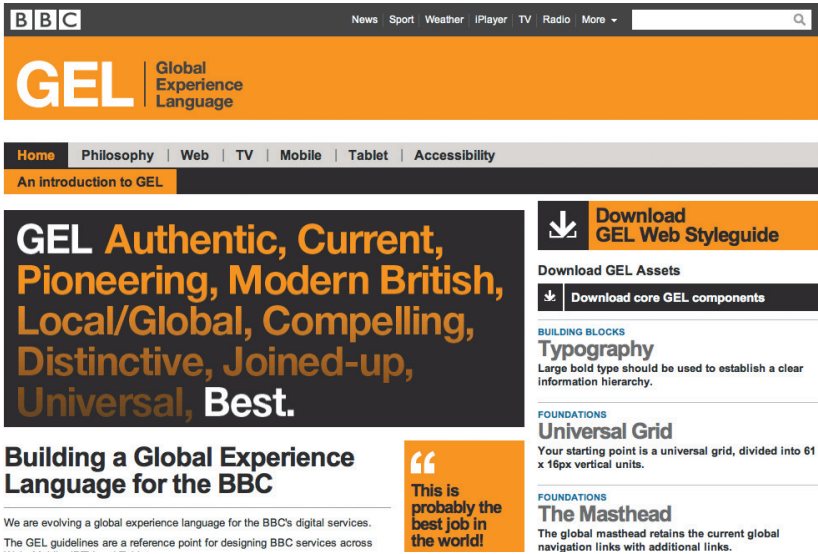


Figure 11.3. BBC’s Global Experience Language (GEL) 18 documents the BBC’s complete visual language as used on the Web.

Email expert MailChimp²⁰ has created its own user interface pattern library to document how elements such as buttons, forms, tables and tabs should be styled. You might wonder why style guides are important in a conversation about responsive design, especially as style guides are commonly created to document design principles after a website has been designed.

¹⁹ smashed.by/gel
²⁰ smashed.by/mc

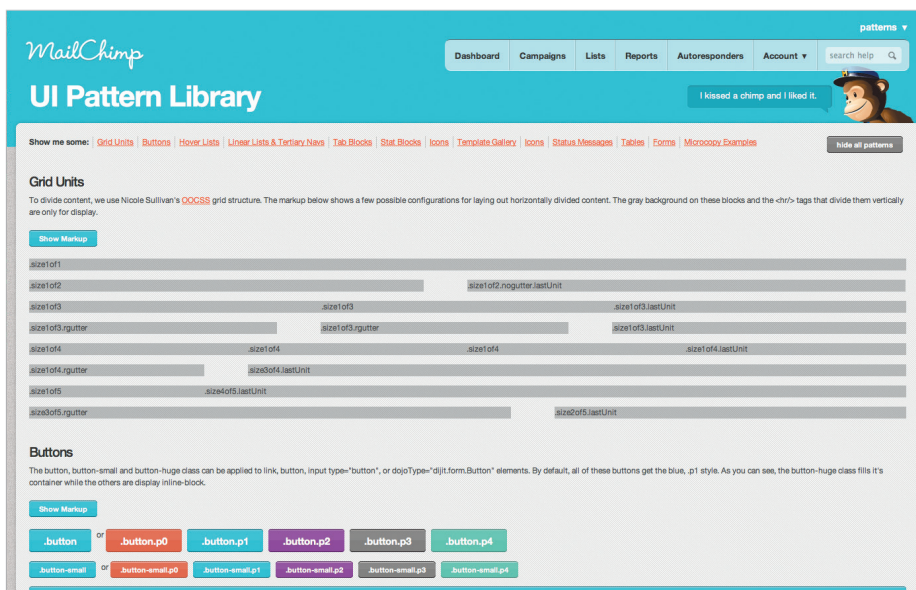


Figure 11.4. MailChimp has created its own user interface pattern library to document how its UI elements should be styled.

Like Twitter's Bootstrap, both the BBC's GEL and MailChimp's UI pattern library are wonderful examples of component design. They detail how every element that will appear in a design should look, and they do so with little or no reference to layout.

The most thorough style guide doesn't simply document a website's design—it is its design. So, why wait until the end of a process before making a style guide? Instead, treat style guides as another example of component-level design. Use them to document and then design every element that will appear on your pages. You can then use them to demonstrate your design direction without being layout-specific, and use them as tools to help you get sign-off without setting expectations about layout.

RETAINING STATIC VISUALS WHEN APPROPRIATE

I've found that designing components first pays real dividends with my clients. I've seen projects run smoother, faster and with fewer misunderstandings. But designing components first is a process that some people find hard to grasp. Because others work in organizations processes where layout-specific static design visuals are still the currency between designers, developers and their bosses or customers, not everyone can leave static visuals behind.

Designers need to see elements in relation to each other. That's because sizes and proportions matter, and there's nothing quite like seeing how everything fits together. In this situation, making a static visual of a design with one layout can be very informative. It's something I do during almost every project.

We also can't escape the fact that, over the years, our bosses and customers have been so indoctrinated into expecting a full-page composition that some of them may be unnerved when we show them a sheet containing just the design of components and no layout. This is something that I experienced early on after switching to component-first design, and it can sometimes require careful handling and reassurances that the result will be better after adopting this process. It's also why sometimes I'm forced to demonstrate an example design to a client as a full-page static visual.

If our workflow forces us to stick with full-page static visuals, at least for a while, does that mean we must abandon the advantages of designing components first? No. Not if we set the right expectations. Specifically, our bosses and customers must learn that the static visuals we might show them are simply extensions of the atmosphere of a design and only one potential expression of it.

Design Atmosphere

If we must carry on using full-page static visuals, we can still extract the component design and use it across screen sizes and devices. Let's break down a design, any design, into its components:

- **Typography:** typefaces, type treatments and white space.
- **Color:** evoking mood and highlighting actions.
- **Texture:** decorative aspects, including borders, shading and shapes.
- **Layout:** elements arranged on a grid, horizontally and vertically.

First, let's separate layout from those other constituents. What remains—the color, texture and typography—I like to call the *atmosphere of a design*, because I imagine everyone's been to a concert or a ball game where the atmosphere could have been described as electric. I'll bet we've all been to a party or two where the atmosphere was flat. Perhaps you've been unlucky enough to sense two people fighting just by the atmosphere in the room.

In design, atmosphere describes the feelings we get that are evoked by color, texture and typography. You might already think of atmosphere in different terms. You might call it “feel”, “mood” or even “visual identity.” Whatever words you choose, the atmosphere of a design doesn’t depend on layout. It’s independent of arrangement and visual placement. It will be seen, or felt, at every screen size and on every device.

While we or our organizations make the transition to newer and better design workflows,²¹ understanding how to extract the atmosphere from a static visual or even an existing website is an important skill. It’s one that developers as well as designers should be quick to learn, because knowing how to see, extract and then rearrange elements as a layout changes is the key to good responsive Web design.

EXTRACTING THE ATMOSPHERE

Let’s look closely at a small selection of website designs. We’ll separate their atmosphere and layout, and with each one we’ll look for what makes its atmosphere distinctive.

Food Sense

First is Food Sense,²² a beautifully designed, responsive website that’s all about “plant-based eating” (see the next page). If we think in static terms, we might comment first on its two-column desktop layout. But we’re looking deeper than that. We’re looking for its atmosphere. We’ll break that down into color, texture and typography.

Color: Within the atmosphere of any design, color evokes moods and elicits emotions, but we can also use color vocabulary for calls to actions, messages and other touch points. Food Sense uses subtle shades of its green, black and white to bind elements of the design together.

Texture: When I talk about texture, I don’t mean physical texture, although nothing is stopping you from breaking out the faux leather and torn paper if you’re into that kind of thing. Instead, texture refers to decorative aspects in a design, including line work (borders, dividers and separators), shading and shape.

For example, we can describe line work. Are lines single, double, dashed? Are they of a consistent width? Is there a hierarchy of dividing-line widths between minor and major sections of content? If so, how’s that hierarchy structured?

²¹ Stephen Hay outlines a future-friendly workflow in the previous chapter in this book.

²² smashed.by/food

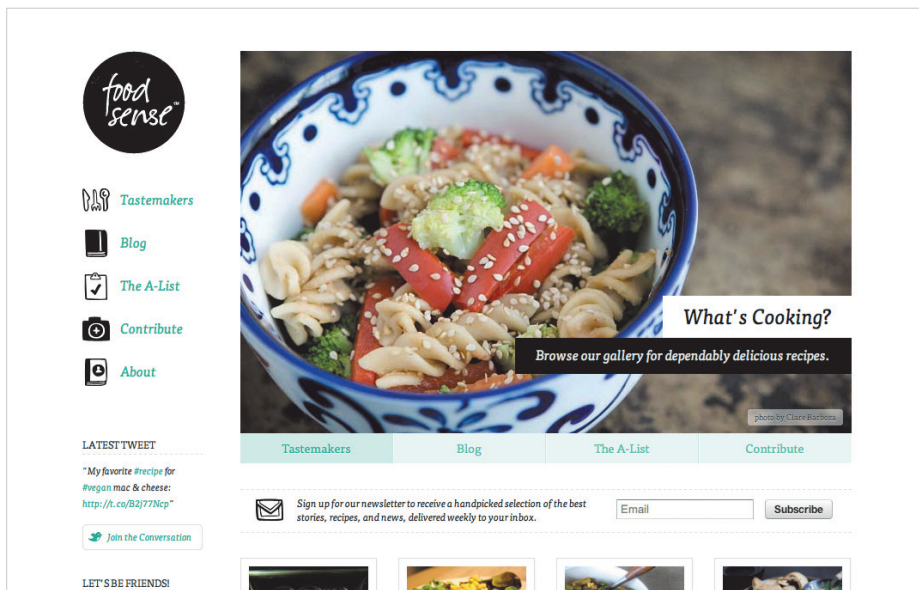


Figure 11.5. Food Sense website.

How is content shaded? Do boxes have a flat color background or are they graduated or patterned? Are the edges of boxes rounded or square? This is also a matter of texture.

Food Sense has flat colored shading for boxes and simple gray dashed and solid lines that divide content regions. It has buttons and action links with subtle gray borders and shadows. Inline content images have been given a white frame, made visible by a gray border. Hand-drawn icons reinforce the friendly atmosphere of Food Sense's design.

Typography is perhaps the easiest of atmospheric elements to extract, but typography means more than just choosing a typeface. Typographic atmosphere includes leading (line height), tracking (letter spacing) as well as the white space separating elements.

Elliot Jay Stocks writes:²³

"I feel that a typography-first, content-out approach to web design moves us one step further away from the unnecessary distractions of design-for-design's- sake and one step closer to becoming true typographers."

²³ Jay Stocks, Elliot. "The Typography-Out Approach In The World Of Browser-Based Web Design," smashed.by/typeout

Food Sense's designers chose lowercase FF Tisa Web Pro for serif headings, body copy and navigation. Their design allows for generous amounts of white space, giving it an airy, open atmosphere.

Jamie Oliver

From studying the atmosphere of designs, our eyes quickly become attuned to their differences. Staying with food, Jamie Oliver's website²⁴ has a very different look from Food Sense. Now it's your turn to extract the atmosphere from Jamie Oliver's website and break it down into color, texture and typography.

Color: The website uses a core combination of brown and blue, as evident in the logo. Can you find a system of colors for links? How is color used in navigation?

Texture: How does the website use background images to create blocks of content? Do you notice design patterns inside those panels? For example, how are headings separated from the content below them? How are buttons styled? Do they have rounded corners or square ones? Is there consistency?

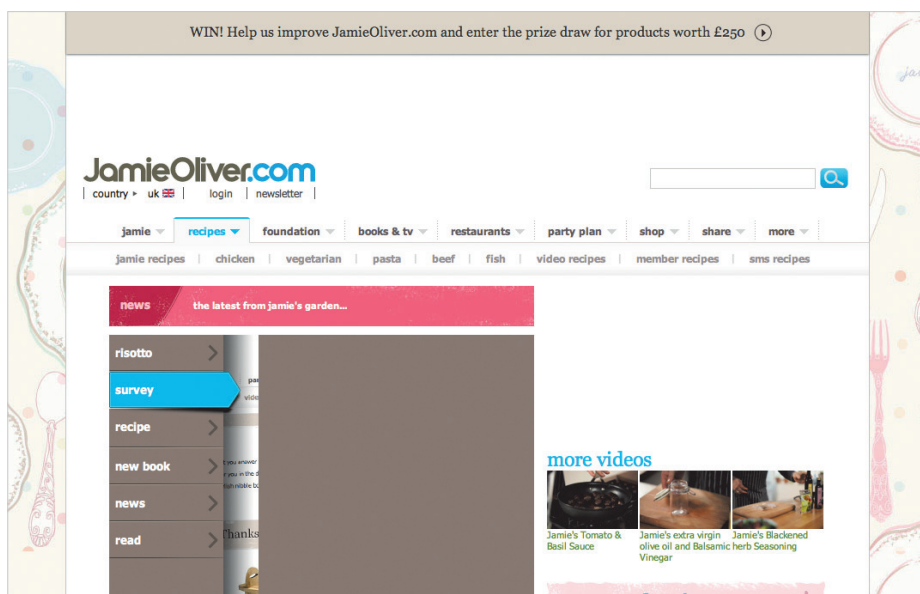


Figure 11.6. Jamie Oliviers website.

²⁴ smashed.by/jam

Typography: The website uses commonly installed fonts. Is there a recognizable typographic hierarchy of headings? Are there differences between the types of body copy? If so, what are they? Does the website feel open or cramped? How do padding and margins affect the feeling of space within the design?

BBC Food

Finally, and staying with food once again, let's examine the BBC's food website,²⁵ where we can see parts of the BBC's GEL guidelines making an appearance. Once again, we're not looking for layout. Instead, we're looking for atmosphere and what makes this website's design distinctive from others with similar content or purpose.

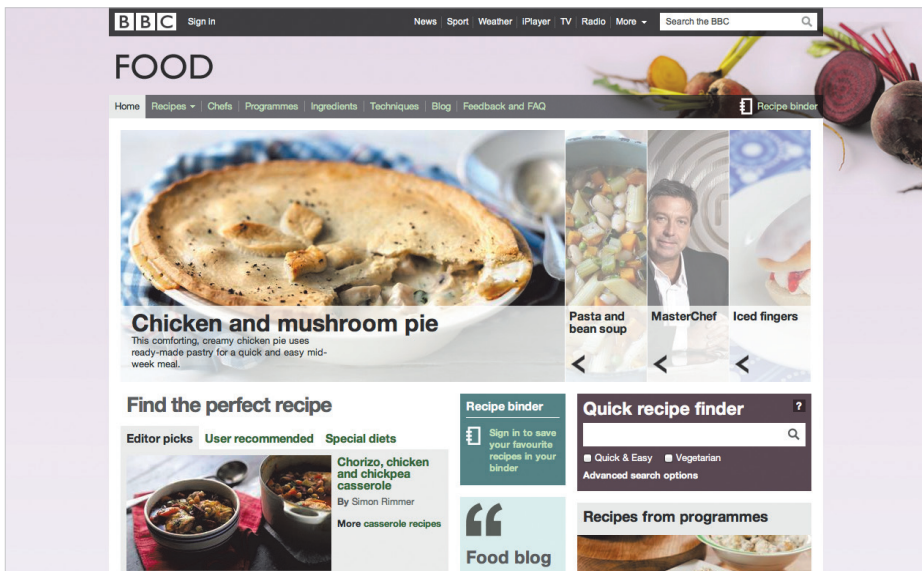


Figure 11.7. BBC Food website.

BBC Food uses color to explain which elements are links. Its palette is subdued, perhaps to emphasize the color in its photography. Where the BBC uses icons from its GEL icon set, the icons are flat and single color, as are the backgrounds of content boxes.

The BBC keeps the edges of its boxes square and sharp. Where content overlaps images, BBC Food's designers use semi-transparent blocks of white or black. Images themselves are largely untreated and are without borders of any kind.

²⁵ smashed.by/bbc

The BBC uses bold typography to “create strong hierarchies and drama across the site.”²⁶ The BBC’s default Web font is Arial, although dig in and you’ll find Helvetica Neue for the Mac. It uses those typefaces both for headings and for body copy.

This is an exercise that will work on any website and especially with groups of designers and developers working together. It’s become a key part of my responsive Web design workshops and has proven particularly effective.

Becoming Fabulously Flexible

Whether we learn to design page components first or we learn to extract the atmosphere from static visuals, it’s no longer necessary for us to make separate Photoshop or Fireworks visuals of layouts for every page—let alone every device. In fact, I’d go so far as to say that the days of designing multiple static visuals are over, as we, our bosses and our customers realize their inherent inaccuracies and inefficiencies. This means that, whether we like it or not, we must find new ways to design responsively.

I know from my own work over the last twelve months, as I get deeper and deeper into responsive Web design, that finding new ways to design and then communicate those designs isn’t easy. Although I’ve been promoting the notion that websites needn’t look the same in every browser for years, somehow letting go of control over layout has been tough. I’ve come to realize that my job as a designer has fundamentally changed. I can no longer expect to have the same level of control over layout across screen sizes and devices as I experienced before.

There are significant upsides to responsive Web design for designers, though, especially in workflows that embrace flexibility. Now I can focus on sweating the details of a design throughout the entire development process, not just at the beginning. Instead of needing to explicitly describe how every layout for every screen size and device should look, I can explain my thinking behind a design and then leave others to interpret that design and adapt it while they develop. This is a workflow that’s worked fabulously for me and my clients.

For this to work well, the developers I work with must also learn about design and how to extract the atmosphere from it. They must take responsibility for and make decisions about design. For some, this might seem like a daunting challenge, but the developers I’ve worked with have viewed it more as an opportunity to develop an understanding of and skills in a new area.

²⁶ smashed.by/bbcfonts

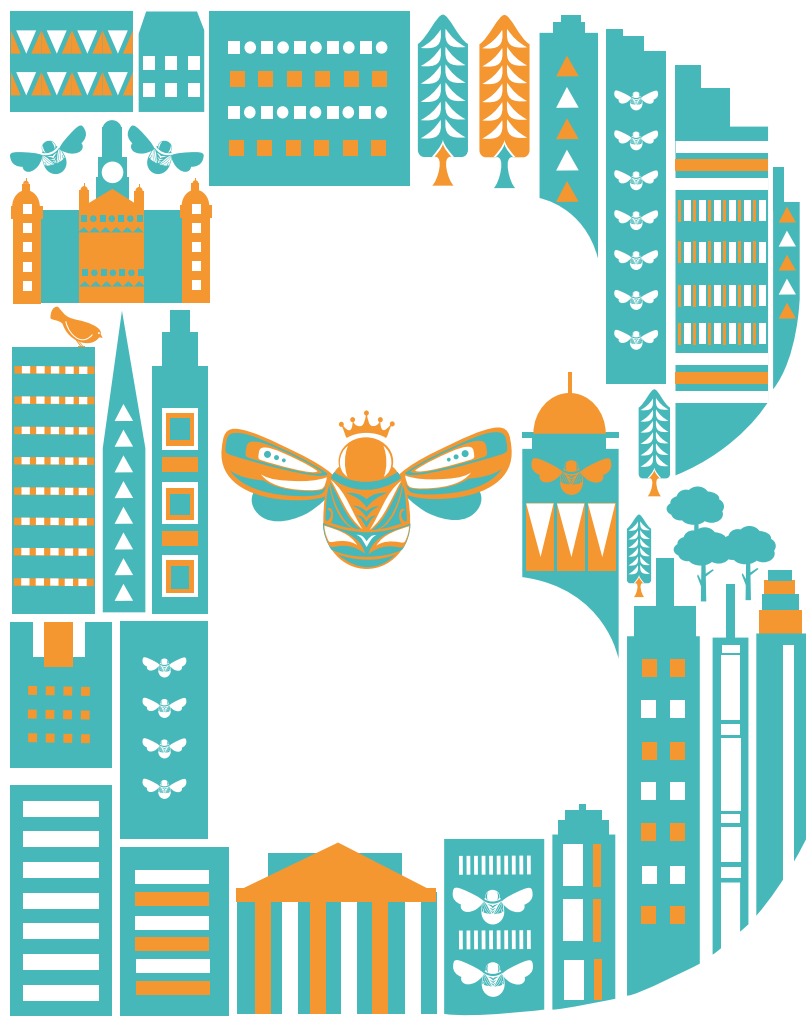
I said at the beginning of this chapter that responsive Web design asks more questions than it offers answers, that it challenges the working relationships and interactions between everyone involved in every process. I know that these challenges won't always be easy to overcome. But I've seen firsthand how keen designers and developers are to solve these challenges, and how receptive our bosses and customers can be when they experience the benefits that responsive Web design can bring. By working together, we can make the Web the responsive place it was always meant to be.



About the Author

Andy Clarke has been called many things since he started designing for the Web 10 years ago. His ego likes terms such as “Ambassador for CSS,” “industry prophet” and “inspiring,” but he’s most proud that Jeffrey Zeldman (the godfather of Web standards) once called him a “triple-talented bastard.”

Andy runs Stuff and Nonsense, a tiny Web design studio where he works with clients such as ISO, STV and the UK government. He presents at Web design conferences worldwide and is the author of *Transcending CSS* and the highly acclaimed *Hardboiled Web Design*.



“one-click” ordering, 49
 @font-face, 100ff
 <article>, 81
 <aside>, 82
 <figcaption>, 83
 <figure>, 83
 <footer>, 82
 <header>, 82
 <nav>, 83
 <section>, 80, 81
 <time>, 83
 3-D Secure, 48
 A/B testing, 44
 accordion forms, 172
 adaptive design, 33
 adaptive prototyping, 304
 Android, 201
 angle gradients, 217
 anticipation, 250
 background-clip, 120
 binary-framework matrix, 276
 bitmaps, 203, 211, 220, 226
 blank-canvas syndrome, 244
 border images, 121
 box-sizing, 129
 brand, 39, 14, 232
 breakpoint graph, 302
 breakpoints, 302
 browser engines, 96
 browser support, 32
 budget, 53
 business, 10, 12, 20
 card payments, 46, 176
 centering, vertical and horizontal, 107ff.
 client-side storage, 88
 cloud hosting, 64
 CMS, 39ff.
 Cocoa, 222, 260, 270, 277ff.
 color management, 207, 210
 color profile, 206
 completeness meter, 180
 component-level design, 327
 components first, 321
 concave and convex shading, 213ff.
 content inventory, 289
 content reference wireframing, 291ff.
 continuous client, 268ff.
 copywriting, 175, 167
 cross-platform, 257, 276
 CSS media queries, 137, 293, 312, 322
 CSS Percentage Problem, 128
 CSS selectors, 136, 138, 142
 CSS selectors, 125
 CSS transitions, 129, 138, 159
 CSS workarounds, 94
 CSS-generated content, 139
 CSS3, 94
 customer service, 186ff., 60
 dark patterns, 192
 databases, 61
 dedicated server, 64
 default values, 174
 delight, 191, 250
 design atmosphere, 328
 design patterns, 239
 design persona, 244
 device classes, 303
 device testing, 222
 device-agnostic thinking, 288
 disguised ad, 192
 DNS, 68, 69
 DOCTYPE, 75

documents-to-applications continuum, 271
 DOM, 136, 142
 drag and drop, 155, 156
 e-commerce, 38ff, 44, 186
 Electronic Point-Of-Sale (EPOS) System, 52
 emotional experiences, 240, 244
 emotional response test, 28
 engagement methods, 250, 169
 event delegation, 140, 103
 experience enhancement, 301
 exporting images, 223
 fallbacks, 94
 feature-detection, 95
 FileReader, 155, 156, 158
 flash test, 28
 flex unit, 111
 Flexbox (Flexible Box layout), 107ff, 246
 font-stretch, 103
 forms validation, 47, 145, 176
 forms, sign-up, 165ff., 193, 290
 gradients, 217, 114
 gradual engagement, 169
 hidden survey, 165
 hosting, 60
 HTML5, 72
 HTML5 Canvas, 140
 HTML5 Local Storage, 88, 98
 HTML5 semantic outlining, 84
 HTML5 Session Storage, 88, 98
 HTML5 Websockets, 181
 hyphenation, 104
 IE 6, 136, 137, 159, 88
 IE 7, 77
 IE 8, 90, 129, 136
 immersive apps, 277ff.
 incremental change, 12
 interfaces, 175, 164, 184, 206, 238
 iOS, 201, 222, 260, 283, 303
 JavaScript, 76
 jQuery, 136
 layout techniques, 106
 linear design, 298
 marketing, 190, 179, 187, 238
 mobile, 184, 32ff.
 mobile first, 296
 mood boards, 321, 246, 23ff.
 multiple backgrounds, 113
 multiple gradients, 114
 native, 258, 33, 184, 198, 201
 navigation, 83
 no-reply, 189
 non-immersive apps, 278ff.
 Objective-C, 279, 283, 238
 off-the-shelf software, 43, 48, 52, 57
 open source, 55
 outline, 116ff.
 outside-in design, 257
 patterns, 177, 219, 114
 payment gateway, 47
 Payment Service Provider (PSP), 47, 48, 49
 PayPal, 44, 46
 Payment Card Industry Data Security Standard (PCI DSS), 49
 personality, 233
 PhoneGap, 276ff.
 Photoshop, 198
 pixel density, 199, 221
 pixel grid, 204, 211
 Pixels Per Meter (PPM), 221
 please-reply, 189
 polyfills, 159, 111
 PPI, 199, 205

principle of universality, 271
 progressive enhancement, 256, 303
 progressive log-in, 183
 progressive sign-up, 170
 proprietary plugins, 8
 psychology, 26
 querySelector, 138
 querySelectorAll, 138
 real-time updates, 181
 realignment, 12
 realism, 200, 216
 redesign indicators, 13
 refactoring, 56, 65
 rem unit, 97
 reset styles, 80
 responsive context, 319
 responsive prototyping, 304
 responsive Web design, 312ff, 33, 43, 256, 273, 304
 Retina display, 199, 205, 219ff., 226
 scaling, 201
 scope creep, 15
 screen sizes, 199
 server-side languages, 61
 shading, 213
 shapes, 211
 shared hosting, 64, 67
 simulators, 273
 skeuomorphics, 200
 slices, 225
 Smart Objects, 220
 Smashing Book 4, 2013
 Software As A Service (SaaS), 46, 58
 source-order independence, 112, 297
 spherical shapes, 215
 staging server, 65, 67
 stakeholder interviews, 20
 static visuals, 314, 327
 storytelling, 178
 structured content, 288, 296
 style guides, 250, 326
 style tiles, 323
 sub-pixel patterns, 222
 surveying, 27
 SVG, 98, 122, 203
 target, 126
 textures, 219, 249
 third-party systems, 51
 typography, 249, 328
 universally accessible, 264
 uptime guarantee, 63
 usability testing, 29
 user experience, 164, 244, 256
 user interface, 200, 12
 user personas, 233, 22
 user research, 233, 19
 user testing, 25
 vendor prefixes, 96
 version control, 66
 vertical rhythm, 98ff.
 virtual private server, 64
 voice and tone, 251, 323
 W3C, 73, 74
 WAI-ARIA, 86
 Web apps, 202, 262ff.
 Web platform, 264
 Web Standards, 8
 Web typography, 97
 WebKit, 74, 96, 107, 277
 Windows Metro, 202, 221
 wireframing, 24, 291ff.
 WordPress, 43, 58, 60
 Xcode, 221, 283