FOR MARJOLEIN, COLIN, CHRISTOPHER, SARAH, AND LEX.
This page intentionally left blank
Acknowledgements

Writing a book is hard (let’s go shopping). And while this isn’t a big book, I’m amazed at how much work—by so many people—has gone into it.

I’d like to thank Michael Nolan, who saw writing potential in me nine years ago, and again last year when I finally had the mental space to jump into the deep end and try. A friendly man with impeccable taste in authors.

A huge thanks to Margaret Anderson, Secret Weapon of Authors™, who was this book’s emergency power supply. Margaret is half psychologist, half development editor, half mental coach, and half project manager. But wait, you say. That’s four ha—yup. Indeed. Margaret made my first trek into book-writing territory as painless as it could be. It only hurts when I laugh.

Thanks also to copy editor Gretchen Dykstra, who spiced up all my boring black text by adding red. Lots and lots of red. Gretchen taught me lots about the English language, especially the fact that I don’t know how to write it. I think it should become public knowledge that authors are only as good as their copy editors.

A huge thank you to Charlene Will for this book’s design. Also Rebecca Winter, Patricia Pane, Danielle Foster, Jack Lewis, and the rest of the Peachpit/New Riders team. An incredible amount of work done by a bunch of friendly and talented people.

But that’s not all. Oh, no, that’s not all. Many thanks to . . .

Jake Archibald, eerily talented developer, for agreeing to tech edit this book for me. I chose him because of his superior knowledge, friendly demeanor, and politically incorrect humor. He repaid me by telling me that my JavaScript should be more JavaScripty. What does that even mean? He’s an oracle.

Ana Nelson, author of Dexy, which now plays an important role in my work. Thanks to Ana for spending suspicious amounts of time with me on Skype answering all of my questions, and even adding stuff to Dexy so it could accommodate my bizarre use cases. She even taught me a little Python along the way. I’m the first official Ana Nelson fanboy; group therapy awaits.

Ethan Marcotte, distinguished gentleman of web design, for his wonderful foreword. He has inspired me for years.
Tim Kadlec, who had just finished his book and was my big example and took all my questions gracefully. Bruce Lawson, for recommending the Secret Weapon™. Aarron Walter, Mike Rohde, and Travis Holmes for their image contributions.

And all those who have inspired my work—whether they know it or not—in person, online, during conversations, or through their great work. These include Stephanie and Bryan Rieger, Jeremy Keith, Scott Jehl, Christian Heilmann, Remy Sharp, Brad Frost, Lyza Danger Gardner, Karen McGrane, Jason Grigsby, Kristina Halvorson, Peter-Paul Koch, Krijn Hoetmer, Jennifer Robbins, Robert Jan Verkade, Marrije Schaake, Bert Bos, Luke Wroblewski, Vasilis van Gemert, and many, many others. I’m privileged to call some of these people my friends.

My mother and my sister, of course, who are always encouraging, and to my father, who would have loved to see this book come to be.

My beautiful, wonderful kids, for having lost some free time with me and for having put up with some serious moodiness on my part.

And finally, Marjolein, my partner in crime. Her support, advice, love, and encouragement are ultimately the reason these words are in print.
# Contents

## Foreword
by Ethan Marcotte

### In Splendid Variety These Changes Come

- The birth of static hi-fi mockups ........................................ 2
- The static mockup comfort zone ....................................... 4
- The specialist invasion .................................................. 5
- We’re all interaction designers ....................................... 8
- Jump from the waterfall ............................................... 8
- The straw that broke..................................................... 9
- The elephant in the room ............................................. 10
- This is not gospel .......................................................... 11
- This is a challenge ..................................................... 12

## From the Content Out

- Microstructure vs. modular structure .................................. 14
- The lazy person’s content inventory ................................ 16
- Our universal example: This book’s website .................... 16
- Progressive enhancement as design principle:
  - The zero interface .................................................... 17
- Creating the example content inventory ....................... 19
- Try it out ............................................................... 22
3 Content Reference Wireframes

Stop making this stuff so complicated ...........................................27
Baby steps: Creating low-fi web-based wireframes .......................29
  Setting up your base markup .................................................29
  Setting up your base styles ..................................................32
  Adjusting the wireframe to be “mobile first” ............................35
  Adding navigation ...............................................................37
  Creating variants for larger screen sizes ..................................39
Let’s bust some myths ...............................................................46
  Interaction designers should make wireframes .........................46
  Wireframes should be detailed ..............................................47
  Do content reference wireframes limit design choices? ...............47
  Isn’t it too early to think about layout? ..................................48
  What should I wireframe? .....................................................48
  When do I involve the client (a.k.a. “Where’s my fancy deliverable?”) ....49
Try it out ..................................................................................49
# Designing in Text

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>It’s all content</td>
<td>52</td>
</tr>
<tr>
<td><em>Starting design with plain text</em></td>
<td>54</td>
</tr>
<tr>
<td>Marking up plain text</td>
<td>55</td>
</tr>
<tr>
<td><em>The book page text in Markdown</em></td>
<td>56</td>
</tr>
<tr>
<td><em>What changes mean at this point.</em></td>
<td>59</td>
</tr>
<tr>
<td><em>It’s about thinking</em></td>
<td>61</td>
</tr>
<tr>
<td>Converting plain text to HTML</td>
<td>61</td>
</tr>
<tr>
<td><em>Using the command line</em></td>
<td>62</td>
</tr>
<tr>
<td><em>Converting to HTML</em></td>
<td>66</td>
</tr>
</tbody>
</table>

# Linear Design

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing a design language</td>
<td>70</td>
</tr>
<tr>
<td><em>Using the Design Funnel</em></td>
<td>71</td>
</tr>
<tr>
<td><em>Serve your design to actual devices</em></td>
<td>74</td>
</tr>
<tr>
<td>Enhancing your structured content</td>
<td>76</td>
</tr>
<tr>
<td><em>Introducing templates</em></td>
<td>78</td>
</tr>
<tr>
<td><em>Your project folder so far</em></td>
<td>81</td>
</tr>
<tr>
<td><em>Think and sketch</em></td>
<td>81</td>
</tr>
<tr>
<td><em>Playing with type and color</em></td>
<td>83</td>
</tr>
<tr>
<td><em>Don’t do too much just yet</em></td>
<td>87</td>
</tr>
</tbody>
</table>
## Breakpoint Graphs

- Documentation for breakpoints. .............................. 92
  - Anatomy of a breakpoint ................................. 92
  - Visualizing breakpoints .................................. 95
  - Breakpoint graph components ............................ 95
- Creating a simple breakpoint graph ......................... 97
- Major and minor breakpoints ............................... 100
- Adding more complexity .................................... 101
  - A more complex example: A podcast player .......... 102
- What we’ve covered ........................................... 105

## Designing for Breakpoints

- First, a bit about sketching .................................. 108
  - How to sketch .............................................. 109
  - Sketching on devices ..................................... 113
  - Sketching as a habit ..................................... 115
- Only sweat the major breakpoints (for now) ............... 116
- Think about your content while sketching ................. 118
  - Text ......................................................... 119
  - Navigation ................................................ 119
  - Tables ..................................................... 120
- What to do if you get stuck ................................. 123
# Creating a Web-Based Design Mockup

Hurdles to acceptance ................................. 127

*Clients (generally) don’t care* ......................... 127

*Other people* ........................................ 128

*You* .................................................. 130

*Presenting your mockups* ............................ 132

Let’s get to work ........................................ 132

*Evolving your responsive wireframe* ................. 133

From static page to static site generator ............... 139

*Templating* ......................................... 139

*Choosing an SSG* .................................... 140

*Introducing Dexy* .................................... 141

*Installing Dexy* ...................................... 142

*Get your assets in here!* ............................. 146

*Including style sheets* ............................... 146

*Adding content* ....................................... 148

*Sectioning content* ................................... 148

*Dexy’s command center: The dexy.yaml file* ....... 151

*Finishing your design mockup with CSS* .......... 153

*Multiple pages* ....................................... 154

What we’ve covered ..................................... 156
Presentation, Round One: Screenshots  

Why not present in the browser right away? ........................................ 159

The presentation/realism balance ...................................................... 159

Screenshots: Going from web-based (back) to images ..................... 161

How to make screenshots ............................................................... 163

Manual screenshots ........................................................................ 163

Automated screenshots ................................................................... 164

Presenting screenshots ..................................................................... 169

Presentation, Round Two: In the Browser  

You’ll find many bugs in your design ............................................... 172

Collaboration and communication ................................................... 173

How to present your interactive mockups ....................................... 175

Use devices to impress .................................................................... 175

Explaining your design .................................................................... 176

Testing and client review .................................................................. 177

Client review ..................................................................................... 178

Take good notes ................................................................................ 179

Using your notes and making revisions .......................................... 184
II Creating Design Guidelines

Design manuals and the web ........................................ 189
The content and structure of design guidelines. ................. 191
  Websites are different ........................................ 192
My wish list for design guideline software ...................... 193
Creating your design documentation ............................ 195
  Writing the documentation .................................. 196
  Inserting example material .................................. 198
  Creating screenshots ......................................... 199
  Making the Dexy configuration file ......................... 200
Testing your Dexy project. ...................................... 201
  Taking screenshots of specific elements ................... 202
  Including rendered HTML .................................. 204
  Including syntax-highlighted code .......................... 206
  Making the documentation your own ..................... 210
Now it’s time to go .................................................. 211

Index ......................................................... 213
This page intentionally left blank
Foreword
by Ethan Marcotte

I have to be blunt: this is a wonderful book you’re about to read.

There’s a quote by Ludwig Wittgenstein that I’ve always loved: “The limits of my language are the limits of my world.” Something’s always seemed magical about that image: the broader your vocabulary, the broader your horizons.

I think of it often, especially as I remember my first studio job. Because looking back, I realize my introduction to web design was, well, pretty narrow, framed as it was by four little words: discover, design, develop, and deploy. Those were, I was taught, the discrete, task-based phases into which each design project was segmented. Research preceded design, and then coding followed, leading to site launch. Simple. Straightforward. Linear.

That model of working felt a bit like a relay race: teams would have to finish their work before the next could begin, handing work down the line before a site could launch. Of course, the truth was often quite a bit messier. And as we began designing beyond the desktop, bringing our work to more and more screens, that old, linear workflow began to show its limitations. Teams need to collaborate more; research, design, and development are more closely related than we once thought, and that old waterfall method kept them siloed.

Thankfully, in these pages, Stephen shares his years of thinking about a more web-native, responsive design process. And as he leads us from design exercises, to a new mode of wireframing, to introducing clients to responsive design, one thing becomes clear: this is a better way to work.

If the limits of our world are set by our language, then Stephen’s book is a veritable dictionary: one full of concepts and techniques to reinvent the way you think about not only design, but the web in general.

This book is going to make your world so much wider. Enjoy.
This page intentionally left blank
“Plain text is the underlying content stream to which formatting can be applied. [...] Plain text is public, standardized, and universally readable.”

—THE UNICODE STANDARD, VERSION 6.1
The world’s first web page was practically mobile ready. And that was in a time when there were no tablets or smartphones. There’s one simple reason for this: the web page consisted of plain text. Actually, it consisted of structured text with hyperlinks. This was how the web started, and the premise of structured content is still the basic foundation of the web (Figure 4.1).

We’ve added to the basic building blocks of the web, which were essentially (structured) plain text, hyperlinks, and images. We have technologies like JavaScript and SVG that let us draw on the screen, or even create entire user interfaces. Many developers advocate creating a distinction between web apps and web documents, referring to information-based websites, such as the W3C’s first web page—or your blog—as web documents.

It’s all content

We need not concern ourselves with the heated discussions about apps versus documents. For our purposes, if it’s on the web, it’s probably both. Most websites composed of documents are run by an application such as a content management system that requires a user interface. This tends to be the case even if the site’s purpose is informational.
By the same token, I’ve never seen a web app without content. No app simply contains buttons with nothing on them, text fields without labels, and absolutely no text on the page.

One of the biggest problems in web accessibility is that many people start with the advanced user interface, take that as a given, and try to “add accessibility.” Many websites are designed from the UI in rather than from the content out.

Take a geographical mapping application such as Open Street Maps or Google Maps as an example. It’s easy for developers to plot out company locations on a map. When embedding these types of apps within a web page, there may be a focus on making that interface accessible—ensuring that the user can navigate with nothing more than a keyboard, for example. This is great, but there’s a problem: technically complex or advanced user interfaces can’t be viewed or used on every device. There’s only one type of content that can be viewed on virtually any web-enabled device, and that is plain text, or rather, plain text that’s been structured with HTML. Like it or not, HTML is the way we structure plain text for the web, because HTML is the single most portable and universally readable format at the time of this writing. Anything that can show websites can read and display HTML.

This means that there’s an alternative approach to accessibility for complex interfaces, and for making any content universally available: start with the text-based foundation of the website or application you’re designing, and then add the complex interface as a layer on top of this text base.

This might sound weird at first, but when you think about it, a mapping application does contain actual data and textual content. It’s simply obscured from the user. We have to deal with a layer of abstraction to get to that information. Responsive design starts at the base: the structured content level. This allows sites and apps to respond to the user’s environment, rather than expecting the user to respond to the interface (perhaps by grabbing a different device so she can actually use what we’ve built).

The way to do this is to start with the data, that is, the plain old textual content that’s always available somewhere. Don’t bury this under an avalanche of UI, but expose it from the very beginning. Allow that content to be the base on which you build.

So what about those company location maps? Well, the base data is probably a list of company location addresses, and perhaps other data such as phone
numbers and URLs. My opinion? Don’t hide this data in a map. If you need or want a map, that’s fine, but leave the textual data open for the user as well, instead of taking perfectly accessible data and hiding it from some users.

If you think this sounds a lot like progressive enhancement, you’re absolutely right. That’s what it is. And of course there are exceptions; there always are. But generally, many web apps are web forms at their core. Many websites are simply structured text at their core. By starting the design at the core, we can build websites that are more accessible and more usable by more and larger groups of people.

Starting design with plain text

Designer Bryan Rieger shares my love of plain text. During some correspondence with me about the subject, he shared these thoughts:

“One technique I’ve used for years is to ‘design in text’… not necessarily describing everything in textual form […] essentially what is the message that needs to be communicated if I was only able to provide the user with unstyled HTML?” —BRYAN RIEGER

Rieger’s statement embodies most of what this book is all about: creating from the most basic, important content and working from that point forward. The technique of designing in text—that is, unstyled HTML—has some absolute benefits:

- As in content inventory and content reference wireframes, the focus is on content. In the case of designing in text, it’s all about the structure of the textual content. Irrelevant content becomes easy to spot, as it’s not hidden by the design.
Designing in text utilizes one of the most important building blocks of the web: HTML.

The linear form of structured text prepares us for the starting point for responsive design: the smallest screens with the least capabilities. (Remember the web’s first page!)

Clients familiar with word processors are probably familiar with the idea of linear, structured content (although some require an explanation of the difference between visual and structural formatting). It’s relatively easy to convert word processor documents into structured plain text.

When you create a page with unstyled HTML, you have created a web page that’s mobile ready. From a design standpoint, it’s also mobile first. With the default width of 100%, you might say it’s on a single-column, flexible grid. This is a perfect starting point for responsive design.

Where content reference wireframes get us thinking about content at a block level, designing in text shifts the focus to the smaller bits of content. Let’s take a look at how these ideas apply to the book website.

Marking up plain text

As Rieger pointed out, it’s not enough to simply use plain text. We need to structure our textual content with HTML. There are several ways to do this, from writing the HTML by hand in a text editor to using a WYSIWYG editor. However, my preference is plain text markup, for which I use Markdown, although many other plain text markup languages exist.¹

Plain text markup languages like Markdown let you write text in a very human-readable way (similar to how you would write in a text-based email program) and offer you tools to convert this human-readable format to HTML quickly and easily. Simple conventions denote structure. For example, a hash symbol (#) in Markdown denotes a level 1 heading. A word enclosed in asterisks (*hello*) denotes emphasis, in the same way you might use italics in a word processor.

¹ To learn more about Markdown, check out:
http://daringfireball.net/projects/markdown/
http://en.wikipedia.org/wiki/Markdown
The best thing about using plain text markup is this: if your client or another party has created the content you’ll use in your design, using a plain text markup syntax means all you have to do is copy the textual content, paste it into a text editor, and make use of the simple markup conventions. This is much simpler than, say, turning the text into HTML by hand.

The book page text in Markdown

The following is some text we’ll use for the book site design. This is the minimum amount of content I’d like to use to communicate with visitors to the page. Save the following text (or your own example text) to your project folder and call it whatever you please. Since it represents the content of a home page, I’m calling it `index.markdown`.

THE TOOL RULE

In this book, there’s a lot of talk about process, and there’s a lot of talk about tools. Let’s agree on the Tool Rule: it’s not about the tools. The process is most important.

I need to use tools to demonstrate the workflow laid out in this book, and the best way to do that is to demonstrate using the tools I use in real projects when incorporating this workflow. This doesn’t mean that the tools are right for you. Admittedly, some are quite geeky, and where I use a command line program, you might prefer a graphical equivalent. That’s okay. Just remember that for all the talk of tools in this book, the specific tools used here are not essential for putting the responsive design workflow into practice. You’ll work most effectively when using tools you’re familiar and comfortable with.

That said, if you don’t already have appropriate tools for any of the steps in this workflow, why not try out the ones mentioned in the book? If you’re a designer, don’t be afraid of text-based tools or the command line. You might be pleasantly surprised at how quick, effective, and fun they can be!

So if you prefer Textile or reStructuredText to Markdown, it’s fine to stick with them. If you prefer your own home-brewed Markdown converter to Pandoc, more power to you. (You’re missing out, though!)

You get the idea. Whenever I use a specific tool in this book, remember the Tool Rule and use a tool that you’re comfortable with, as long as it helps you get the job done. Tools are the means, not the end.
# Responsive Design Workflow

by Stephen Hay

In our industry, everything changes quickly, usually for the better. We have more and better tools for creating websites and applications that work across multiple platforms. Oddly enough, design workflow hasn't changed much, and what has changed is often for worse. Through the years, increasing focus on bloated client deliverables has hurt both content and design, often reducing these disciplines to fill-in-the-blank and color-by-numbers exercises, respectively. Old-school workflow is simply not effective on our multiplatform web.

Responsive Design Workflow explores:

- A content-based approach to design workflow that's grounded in our multiplatform reality, not fixed-width Photoshop comps and overproduced wireframes.
- How to avoid being surprised by the realities of multiplatform websites when practicing responsive web design.
- How to better manage client expectations and development requirements.
- A practical approach for designing in the browser.
- A method of design documentation that will prove more useful than static Photoshop comps.

## Purchase the book

Responsive Design Workflow is available in paperback or as an e-book. The book is available now and can be ordered through one of the booksellers below.
- [Order from Amazon.com]
- [Order from Peachpit Press]
- [Order from Barnes & Noble]

## Resources

[List of resources per chapter?]

## Errata

[List of errata per chapter?]

Now this is interesting. In previous chapters, we discussed how problems can arise from insufficient thinking about how content will actually work—think back to the example where I “forgot” that we might need some sort of navigation. This may seem an unlikely example, but we’ve all experienced situations where we realize that things have been omitted or not thought through. Working in small steps from the content out can expose issues and avoid these problems. This Markdown document exposes a flaw in my thinking about resources and errata that becomes clear when we start designing the page in text. If I have both resources and errata for some of the same chapters, I’ll get something like this:

## Resources

* [Chapter 1](http://www.example.com/resources/chapter1)
* [Chapter 2](http://www.example.com/resources/chapter2)
* [Chapter 3](http://www.example.com/resources/chapter3)

## Errata

* [Chapter 1](http://www.example.com/errata/chapter1)
* [Chapter 2](http://www.example.com/errata/chapter2)
* [Chapter 3](http://www.example.com/errata/chapter3)
Although this won’t get us arrested, it’s redundant. It makes more sense to have a page for each chapter, and having anything relevant to a given chapter on that particular chapter’s page. This means that I’m going to change my mind at this point in the process (clients tend to do that, as you probably know).

Keep in mind that any mind-changing going on at this point is not only not a problem, it’s also a good thing. Better to make content-related and structural changes now.

Instead of Resources and Errata, I want a list of chapters, with each item being a link to that particular chapter’s page, which can contain sections for Resources and Errata. Come to think of it, we’ll need to put up code samples as well, so the chapter pages are a great place to put those.

What changes mean at this point

Changes at this point in the process entail relatively non-labor-intensive edits to one or more of three things: the content inventory, the content reference wireframes, and the structured text design (that is, your Markdown document).

Of course, since our example is currently one page, changing these is simple. Remember that most websites, no matter how big, are a combination of user interfaces for interaction and containers for content that’s poured in from a database. The focus of the first three steps in this workflow is these larger entities: types of pages, types of user interfaces, and types of content. While you’re free to use these steps to design individual pages, you probably won’t want to do that for the individual articles in a site like nytimes.com.

Think in terms of types. Think in terms of components. There are never many page types, so don’t think too much about pages. One of my most challenging conversations with a client involved me explaining that I didn’t have to redesign their site’s 10,000 pages. Rather, we had to spend a lot of time analyzing their content, and then I would design about 10 or 12 pages (plus a battery of small components). I would be designing a system. Not individual pages.

The change I’ve made to the book page means there’s another page type, since the content of that page type dictates that it should be different than the home page. That last point is important. There’s been a lot written about designing the user interface first. I agree in most cases, but not all. The user interface serves a purpose and contains content relevant to that purpose.

TIP
It’s easy to forget that clients don’t always think the way we do. We’re influenced by our experience with technology and the heavy use of abstraction in modern web work. Be patient with clients and explain things in plain language, like a chef giving a tour of his kitchen to a guest.
Both purpose and content are the foundation of the interface. And that thinking is also part of the whole design process.

Before we make the changes to our example, let's think about what these changes mean for us while utilizing the responsive workflow:

1. We need to create a content inventory for the second page type and change the existing one accordingly. This would have to be done within any workflow that involves content inventories, not just the workflow described in this book.

2. We need to create a new wireframe and modify the existing one. In plain English: we need to remove one box from our existing wireframe and create another HTML page with a few boxes on it. Oh, the pain.

3. We need to change the last section of the Markdown document and create an additional one. Since we don't actually have resources or errata yet, we'll have to define the "shape" of that content by coming up with a real-world example we can test and discuss with the client.

These steps are not difficult. If you're a graphic or visual designer, you may not find it exciting. In fact, someone else can do these steps. But that person is absolutely part of design, and as I mentioned previously, all parties should be involved. Yes, the client, too—especially the client. You'll reap the benefits later.

Now, contrast this with the waterfall process. In the traditional workflow, there is no designing in text step. The wireframes are detailed and intricate. The content inventory may or may not exist. So the problems with changes start with wireframes. Sure, we need to change the text in the Markdown document, much as we'd change text in a complex wireframe. But the main difference is that when designing in text, the changes we make are text-based. Detailed wireframes contain text, but this text is still presented in a highly visual form in relation to other elements on the page. There may be no color, but there is typography. There is layout, to a certain extent. By contrast, plain text markup is all the same font and denotes only textual structure. Changes in a Markdown document don't require typographical changes or changes in layout. New pages don't require a new layout. We're not at that point in our process yet, so changes are much easier to make.

Content reference wireframes are also very easy to change. After all, they're just boxes, aren't they? They become more important down the line, but for now, we're compartmentalizing changes. Content changes should be content changes, not content-typography-layout changes.
This approach allows us to “help” the content. In return, content will help us down the line. Because we’ve given it the attention it deserves, it will help us with layout, with determining breakpoints, and with many other aspects of the design. There are always cases where huge changes to a project will come during the final stages of design, and this process is designed to minimize the chance of those changes occurring—and minimizing the impact of those changes if they do occur.

It’s about thinking

Again, the book page is a very simple example, but this could just as easily have been a sign-up page for a product or service (or any other page or component), where designing in text might help you make better decisions. As with content inventory and content reference wireframes, designing in text gives you an opportunity to change things before doing so endangers the schedule or results in high costs.

We’re designing in text here simply by putting text down in a Markdown document. That fact should make it clear that the process of designing in text is not about Markdown; it’s about thinking. It’s (as in the first two steps) about content and its structure.

Personally, this is one of my favorite steps in the workflow. The beauty is not only in its simplicity, but also in the fact that once you convert this document into HTML, you have a mobile-ready mockup of a web page in structured text that you can load into virtually any browser that parses HTML. This is a huge advantage.

Converting plain text to HTML

Practically every plain text markup language has a way to convert structured text into HTML. In fact, Markdown might be one of the most complex, because there are so many different versions of Markdown itself. (These versions are often called flavors. Don’t ask me why—hungry techies, perhaps.)

The original Markdown lacks an equivalent for every single element available in HTML. This is actually a strong point: it contains equivalents for the most commonly used text elements, and lets you use plain HTML when needed. In that sense, Markdown is not a language of its own; it’s a front end for HTML.

TIP

Remember the Tool Rule. If Markdown isn’t your thing, or you’re just interested in exploring some other options, see http://en.wikipedia.org/wiki/Lightweight_markup_language for an overview of similar markup languages.
This means that if you want tables, you can enter them in HTML and that’s a totally legal Markdown document.

But this makes Markdown just that much more difficult for nontechnical people (perhaps even your clients) who’ll be preparing the documents for this step in the workflow. As I was developing this particular workflow, I toyed with the idea of using a more “complete” text markup language, but I was already so familiar with Markdown from use in email and other applications that I wasn’t too keen on switching. Luckily for me, as I mentioned, there are several different implementations of Markdown.2

I chose an implementation called Pandoc.3 Pandoc supports the original Markdown and offers extremely useful optional extensions, such as definition lists, numbered example lists, tables, and more. Pandoc can convert to and from a bunch of file formats, which is wonderful and has so many uses beyond web design workflow.

This will be the first of several instances in this workflow when you’ll be typing things into the command line. In case you’re not familiar with the command line, you’ll most likely be using the Terminal application on OS X or Cygwin on Windows. If you use Linux, there’s a good chance you’ve already used your terminal application at some point.

**Using the command line**

The command line interface (CLI) provides a simple means of interacting with your computer. Its design is actually quite elegant: on the screen there’s a prompt, at which you can tell the computer what you want it to do, and it will do what you ask (Figure 4.2). If it doesn’t understand your command, it will tell you so.

People like to bring up the potential drawbacks of the CLI: yes, there are commands that will erase your entire hard disk or a portion thereof. Just *don’t type those commands*. In the graphical interface of your computer you wouldn’t willingly select all your folders, move them to the trash, and then empty the trash.

---

3 [http://johnmacfarlane.net/pandoc/](http://johnmacfarlane.net/pandoc/) If you’re interested in Pandoc, you can try it before installing it at [http://johnmacfarlane.net/pandoc/try](http://johnmacfarlane.net/pandoc/try).
The argument is that it’s easier to do something stupid like that in the command line. And arguably it is. In fact, many things are easier to do in the command line, not just stupid things. Commands aren’t difficult, though some of them can be difficult to remember. But practice helps with memorization, just as it does when you need to remember which submenu item to choose in a graphical interface.

So don’t be afraid. The command line is a very useful tool, just as graphical applications can be. And as with any computer interface, you need to think about what you’re doing. Just remember that the command line does what you tell it to, nothing more, nothing less. Don’t tell it to do stupid things and it won’t.

The beauty of the command line is that you don’t need to know everything about it. It helps to know some basic commands—such as those allowing you to navigate around your system and create, copy, move, and delete files and folders—but mostly what you’ll need to know are the commands specific to the software you’re using.

If you’re skeptical, consider Adobe Photoshop (Figure 4.3). Photoshop—which this workflow deems unnecessary for creating design mockups for the web—is one of the most complex and sophisticated pieces of software available to consumers today. There are hundreds of books on how to use Photoshop, as well as whole books that cover only a specific functionality. If you’re a designer, you’ve most likely used Photoshop. So there you are, proficient in this really advanced piece of software, but worried about the command line. Believe me, you are absolutely smart enough to learn commands in the command line. And someday when you discover more command line tools and are able to do things like resize fifty images in about three seconds,

**NOTE**

I’m making the assumption that you either have a terminal application or are capable of installing one. Linux and OS X have terminals built in. Windows users can install Cygwin from www.cygwin.com.
you’ll feel the power that your developer friends do. If you’re a developer and reading this, yes, go ahead and gloat.

I recommend that you consult a resource like Zed Shaw’s CLI Crash Course to become familiar with the command line. The online version is free, easy to follow, and you really will learn all the basics.4

That said, there are few commands you should know now: `pwd`, `cd`, and `ls`. Go on, open up your terminal application. This puts you in a shell. On OS X, the system I currently work with, the standard shell is called Bash.5

As a web worker, you’ve very likely seen a terminal before, but if you’re not familiar, that little blurb of text you see to the left of the cursor is called a prompt. It’s customizable, and it tends to look different depending on both the system and the user. It may or may not tell you information about the system or the folder you’re in. No matter. You type commands at the point where the cursor is. Type one now: `pwd`. You’ll see something like this:

```
$ pwd
```

Now press the Return key. The CLI returns a path. Just like paths on a web server, this is a path of folders on your computer. The path you see leads from the root folder of your computer to the folder you’re currently in. `pwd` means

---

4 http://cli.learncodethehardway.org

5 Bash stands for “Bourne-again shell”. See http://en.wikipedia.org/wiki/Bash_(Unix_shell)
print working directory. The command tells the computer to print the working directory, which is the folder you’re currently working in. This is useful because, in contrast to your system’s graphical interface, you don’t always have a visual clue of where you are when you’re in the command line. This is what I get when I execute `pwd`:

```
$ pwd
/Users/stephenhay
$ 
```

Yours will be different, unless your name is Stephen Hay (in which case, nice name!). No problem; now you know where you are. Let’s see what’s in this folder. We can list the files in the current folder with `ls`:

```
$ ls
Applications   Documents   Library   Movies
Pictures   Desktop   Downloads   Mail
Music   Public
$ 
```

Your results might be shown differently, depending on how many files you have and the width of your terminal window.

You’ll also want to change your directory, which you can do with `cd`:

```
$ cd Applications
$ 
```

This puts me in the Applications folder. You might not have the same folder; just enter the same command in one of your own folders.

My own prompt contains information about where I am (it actually contains the name of the folder I’m in), but I’ve customized it to do so. You needn’t worry about that at this point. As you become more comfortable with the command line, you can learn how to customize your environment.⁶

---

⁶ Many years ago I learned how to customize my own prompt by reading Daniel Robbins’s easy-to-understand article on the subject, which can be found at http://www.ibm.com/developerworks/linux/library/l-tip-prompt/.
So moving down is easy: just type `ls`, note the directory you’d like to move to, and `cd` to that directory. Also note that many shells let you use the tab key for completion. This means that instead of typing the full word `Applications` in the previous example, I could type an `A` or `Ap` followed by the Tab key, which would complete the word for me. When the completion matches several words, these will be shown and you’ll need to add one or more letters accordingly before pressing Tab again. This is a huge time-saver:

```bash
$ cd A [press Tab key]
$ cd Applications [press Return key]
```

Now you know how to move down a directory. Moving up is easier. A single dot is the symbol for the current directory, and two dots is the symbol for the parent directory. Moving up a directory is done thus:

```
$ cd ..
```

followed by pressing the Return key. Moving up two directories would entail:

```
$ cd ../../../
```

and so forth.

That’s enough to get you started. If you have no previous CLI experience, try these commands out for a while. You can’t do anything bad to your system, because these commands don’t alter anything.

## Converting to HTML

The first step in using a command line tool—unless it comes with your system—is to install it. I’m assuming Pandoc doesn’t come with your system, so you’ll need to install it if you’re planning to follow along in the book. On Linux, you might find it and be able install it via your package manager. For OS X or Windows, there are install packages available.\(^7\) Go ahead and install Pandoc (or your preferred plain text converter) and then come back.

---

\(^7\) [http://johnmacfarlane.net/pandoc/installing.html](http://johnmacfarlane.net/pandoc/installing.html)
Once you’ve installed Pandoc, use `cd` to navigate to the folder that contains your Markdown document. Then type the following command:

```
$ pandoc index.markdown -o index.html
```

This says, “run Pandoc on the file `index.markdown`, convert it to HTML, and save the output of this command as `index.html`.” If you run `ls`, you should see that `index.html` has been created (Figure 4.4). Open this file in a web browser. Your structured content is now HTML. And it works on practically every device.

It just doesn’t look very pretty yet, so let’s do something about that. In the following chapter, we’ll start thinking about the more visual aspects of the design process, using this content as a base.

---

**Figure 4.4**

Using a command line tool like Pandoc, it takes only a second to turn plain-text markup into a basic HTML page. This can be a huge time-saver.
Index

SYMBOLS
# (hash symbol), Markdown, 55
$ (dollar sign)
   command line prompt, 64, 143
   variables, 78
* (Asterisk), Markdown, 55
*** (Asterisks), comments, 208

NUMBERS
24ways.org, 189
80/20 principle, 48

A
Accessibility
   content first for, 53–54
   sketching navigation for, 119–120
   testing in web-based design mockup, 177–179
Anatomy, breakpoint, 92–94
Anti-aliasing, on static mockups, 4–5
Assembly line approach, avoiding, 174
Assets, linking web-based design docs to, 189–190
Asterisk (*), Markdown, 55
Asterisks (**), comments, 208
Automated screenshots
   web-based design guidelines software with, 193
   of web-based design mockups, 164–165

B
Background
   design guidelines for, 191–192
   exporting sketches for image, 114
   setting up base styles, 33–34
Bands, breakpoint graph, 96
Base markup, low-fi web-based wireframes, 29–32
Base styles
   creating low-fi web-based wireframes, 32–35
   sculpting unstyled HTML, 85
   web-based design mockups, 134–135
base.css
   adding links to style sheets for web-based mockup, 147
   adding style to web-based design mockups, 134
creating low-fi web-based wireframes, 32–35
creating templates, 78
creating variants for larger screen sizes, 42
inserting syntax-highlighted code into design docs, 208
specifying font stack in, 84
taking notes in browser, 183–184
using for all styles applied to site, 137
_base.html
   creating note-taking app, 181
   Dexy installation, 143–144
   mockups of several pages, 155
   sectioning content for screen layout, 149
Bash shell, 64
Big Reveal, avoiding, 159–160
Blockable tables, 121–122
Blocks, breakpoint graphs, 96
body element
   creating low-fi web-based wireframes, 30–32
   setting up base styles, 32–35
Borders, setting up base styles, 33–34
Bottom-of-page navigation, 37–39
Boulton, Mark, 27, 159, 190
Branding guidelines, 91
Breakpoint graphs
   adding major and minor breakpoints to, 100–101
   components, 95–97
   creating complex, 101–104
   creating simple, 97–100
   creating web-based design mockups, 136–137
   defined, 92
   inserting into design documentation, 198–199
   visualizing breakpoints with, 92, 95
Breakpoints
   anatomy of, 92–94
   defined, 40
   documentation for, 91–92
   in linear design, 88
   presenting interactive mockups to client, 176
   progressive enhancement using implicit, 94
   using separate style sheets for, 43
Breakpoints, designing
creating comps from best rough sketches, 113–116
major breakpoints, 116–118
overview of, 108
rough sketches, 112–113
selecting thumbnail sketches, 111–112
sketching, overview, 108–109
thinking about content while sketching, 118–122
thumbnail sketches, 109–111
what to do if you get stuck, 123

Browser
creating low-fi web-based wireframes, 29–32
creating web-based design mockups, 132
creating working linear design in. See Linear design developer tools, 136
not presenting web-based mockups to client in, 159–160
not worrying about compatibility of, 138
removing browser chrome from screenshots, 163–164
resizing to check style sheets, 35–37
setting up base styles, 34
showing screenshots of web-based mockups in, 129
support for new CSS layout modules, 40

Browsers, web-based mockup presentation
collaboration and communication, 173–174
finding bugs, 172–173
interactive mockups, 175–176
note-taking app, 181–184
note taking in browser, 179–181
notes and making revisions, 184–185
overview of, 172
testing and client review, 177–179
version control for revisions, 185–186

Bugs, finding/fixing design, 172–173
Bullet graphs, visualizing breakpoints, 95
taking screenshots of specific elements, 202–204
writing screenshot script, 166–169
casper . then() function block, 203–204
cd command
converting plain text to HTML, 67
Dexy installation, 143
overview of, 65
using template with Pandoc, 79
Chartable tables, 121–122
Chartjunk, 100
class attribute, 31–33
CLI. See Command line interface (CLI)
CLI Crash Course (Shaw), 64
Clickability, designing link, 119
Client presentation. See also Browsers, web-based mockup presentation
automated screenshots for, 164–165
installing PhantomJS, 165
involving clients from beginning, 158
manual screenshots for, 163–164
not tricking clients, 162
overview of, 158–159
presentation/reality balance, 159–160, 177
psychology of, 132
of screenshots from web-based mockups, 161–162, 169–170
writing screenshot script, 166–169

Clients
involving with wireframes, 49
managing expectations of, 127–129
overloading with detailed wireframes, 47

Code
in design guidelines, 194
design guidelines software features, 193–194
inserting into design documentation, 198–199, 206–210

Collaboration
building into each phase, 8, 173–174
creating content inventories, 23
design guidelines aiding in, 193

Color
content/structure of design guidelines, 191–192
for major changes on breakpoint graphs, 101
rough sketches using shading for, 112
sculpting unstyled HTML, 85
setting up base styles, 33–34
as visual language component, 71
Column width, sketching text, 119
Command line interface (CLI)
  converting plain text to HTML, 66–67
  reading commands, 79
  using template with Pandoc, 79
  working with, 62–66
Comments
  cutting code into sections using idio, 207–209
  inserting rendered HTML into design doc and, 204–205
  typing in page being reviewed, 184
Communication
  documentation improving team, 196
during each phase, 173–174
Components
  breakpoint graph, 95–97
  content inventory, 19–22
defined, 15
  features for design guidelines software, 193
  making changes in terms of, 59–61
  microstructure vs. modular structure, 15
  visual language, 71
Comps, best rough sketches based on, 113–116
Computed Style view, browser developer tools, 136–137
Configuration file, Dexy, 200–201
Content
  adding to web-based design mockups, 148–151
  changing at breakpoints, 92–94
  client presentation of interactive mockups and, 176
design guidelines starting with, 191–192, 196–198
  thinking about when sketching, 118–122
  wireframing pages of, 48
Content inventory
  creating low-fi web-based wireframes, 30–32
  as documentation, 90
  as essential throughout project, 174
example of, 19–22
  experimenting with, 22–23
  as first step, 14
involving client in first draft of, 158
listing only things needed, 16
making changes to, 59–61
microstructure vs. modular structure of, 14–15
this book’s website as example of, 16–17
zero interface and, 17–18
Content reference wireframes
  adding navigation, 37–39
  adjusting for mobile devices, 35–37
  creating low-fi web-based, 29–32
CSS modules for source-order independence, 40
  as documentation, 90
  experimenting with, 49
  implementing media queries, 43
  introduction to, 26–27
  making changes to, 59–61
  as minimal, 27–28
  myths about, 46–49
  not limiting design choices, 47
  setting up base styles, 32–35
  variants for larger screen sizes, 39–45
in web-based design mockups, 133–134
contenteditable demo, 180, 182–183
CSS
  adjusting in browser, 35–37
  for base style setup, 32–35
  for evolving responsive wireframes, 134
  for implementing changes at breakpoints, 92–94
  for inserting syntax-highlighted code into design docs, 206–210
learning, 86, 130, 154
for low-fi web-based wireframes, 29–32
for navigation, 38
new layout modules in, 40
for own documentation, 210–211
for sculpting unstyled HTML, 85
for showing code in design guidelines, 194
sketching navigation with, 119–120
sketching tables with, 122
variants for larger screen sizes, 39–45
for web-based design mockups, 132–137, 153–154
CTRL-D, stopping web server, 75
Cygwin, 62–63
Debenham, Anna, 189

Deliverables
  aiding design process, 90
  building communication/collaboration into each phase, 173–174
  in design documentation, 90
  screenshots of web-based mockups, 129–130
  version control for all, 185–186
  for web-based design mockups, 146

Design Funnel, 71–73

Design guidelines
  communicating what mockups cannot, 192
  content and structure of, 191–192
  creating documentation, 195–199
  creating screenshots, 199–200
  design manuals and web, 189–190
  including rendered HTML, 204–206
  including syntax-highlighted code, 206–210
  making Dexy configuration file, 200–201
  making documentation your own, 210–211
  overview of, 188
  software, 193–194
  taking screenshots of specific elements, 202–204
  testing Dexy project, 201–202
  using code in, 194
  for websites, 192–193

Design language
  creating with Design Funnel, 71–73
  designing for breakpoints, 108
  developing, 71
  example of, 72
  presenting web-based mockups to client as, 160

Designing in text
  making changes, 59–61
  marking up plain text, 55–59
  overview of, 54–55

Detailed wireframes, 26–28, 47

Developer tools, browser, 136

Device classes
  linear design based on. See Linear design
  major breakpoints and, 117
  responsive design working with, 40–41

Dexy
  adding navigation between pages, 155
  command center, 151–153
  creating design documentation, 194, 195–196, 207–210
  features of, 141–142
  finishing web-based design mockup with CSS, 153–154
  installing, 142–145
  making configuration file, 200–201
  rerunning, 147
  taking screenshots of specific elements, 202–204

dexy -r, rerunning Dexy, 147

dexy viewer, 205–206, 208

dexy .conf, 144

dexy .yaml, 144, 149–153, 200–201

Difficult tables, 121–122

Directories, moving to, 65

Discussion, content inventory, 21

Do stuff sites, 5–6

Documentation
  breakpoint, 92
  creating with Dexy, 141–145
  overview of, 90–91
  style guide, 186
  training, 91

web-based design guidelines. See Design guidelines

Dollar sign ($)
  command line prompt, 64, 143
  variables, 78

Elements
  adding on as-needed basis, 9
  design guidelines software and, 193
  taking screenshots of specific, 202–204

Ems, 99–100, 198

Emulators, 75–76, 79–80

Evolutionary prototypes, 126–127

Exporting
  sketches to create detailed mockups, 114
  smaller images from image editor, 86
Filters, Dexy
defined, 141
inserting rendered HTML into design guidelines, 204–206
using, 143
Flexible Box Layout Module, CSS, 40
Font stack, specifying for page, 84
Fonts
sculpting unstyled HTML, 85
serif vs. sans-serif, 84
setting up base styles, 34–35
sketching text and, 119
Frameworks, constructing pages within, 17
Freeform writable, design guidelines software as, 193
Front-end developers
adding navigation, 37–39
creating content inventories, 23
making wireframes, 46–47
Garrett, James, 6
Get stuff sites, 5–6
git, version control system, 185–186
Goals, Design Funnel translating, 72
Gregory, Danny, 116
Grid Layout Module, CSS, 40
guidelines.html file
including rendered HTML, 206
including syntax-highlighted code, 209–210
taking screenshots of specific elements, 204
testing Dexy project, 201
guidelines.markdown file
including rendered HTML, 205–206
including syntax-highlighted code, 208
making Dexy configuration file, 200–201
testing Dexy project, 201
writing design documentation, 196–197
Habit, sketching as, 115–116
Hash symbol (#), Markdown, 55
_head.html, 144, 146–147
Headless WebKit browsers, 164–165
Home pages, wireframing, 48
Horizontal line, breakpoint graphs, 95–96, 99–100, 103–104
HTML
adding image content to, 86
best practices for, 135
converting plain text to, 61–67
creating low-fi web-based wireframes, 29–32
creating own documentation with, 210–211
creating page with unstyled, 55
creating web-based design mockups with, 132
design guidelines with rendered, 204–206
enhancing structured content, 77
evolving responsive wireframes, 137
resistance to learning, 130
sculpting unstyled, 85
sectioning content for screen layout in, 148–151
sketching navigation with, 119–120
structuring textual content with, 55–61
testing Dexy project, 201–202
using media queries with, 43
htmlsections filter, Dexy, 204–208
Hybrid HTML/Markdown, 148–149
id attributes, 30–31, 150
Identity/branding guidelines, 91
idio comments, 207–209
Illustrator, static mockups on web and, 5
Image editors
creating/editing images with, 86
learning HTML/CSS vs. learning, 130–131
as old school for static mockups, 10–11
Images
adding textual documentation to, 90–92
adding to design, 86
as building blocks of web, 52
in complex breakpoint graph, 104
for design guidelines, 191–192
screenshots of web-based mockups as. See Screenshots, web-based design mockups in simple breakpoint graph, 99–100
using exported sketches as background, 114
as visual language component, 71
visualizing breakpoint graph changes, 96
\@import, media queries, 43

\texttt{index.markdown} file
- converting plain text to HTML, 67
- creating web-based design mockup, 148, 153, 155
- as home page of mockup, 144
- text for book site design, 56
- using template with Pandoc, 79
- web documents vs., 52

Information architects, 6

Interaction designers
- adding navigation, 37–39
- birth of, 5–7
- creating content inventories, 19–23
- front-end disciplines overlapping with, 28
- involving throughout design process, 46–47
- as obstacle to web-based mockups, 130
- overview of, 6
- as visual designers, 8
- in waterfall model, 7

interactive mockups, presenting, 175–176

Interface, zero, 17–18

Introduction
- birth of specialists, 5–7
- birth of static hi-fi mockups, 2–4
- creative ways to design on web, 2
- interaction designers, 8
- leaving waterfall method, 8–9
- responsive design vs. static mockups, 10–11
- responsive design workflow, 9–10, 12, 212
- on static mockups, 4–5

IP address, starting ad hoc web server, 74–75

Iterative design process, 8–9

J

JavaScript
- creating web-based design mockups, 132
- implementing changes at breakpoints, 92–93
- note-taking app, 181–184
- sketching navigation, 119–120
- sketching tables, 122
- using breakpoints in progressive enhancement, 94
- writing screenshot script, 166–169

L

Laptop-screen presentations
- of interactive mockups, 175–176
- of screenshots, 169
- taking notes, 184

\texttt{large.css}
- adding links to style sheets for web-based mockup, 147
- creating variants for larger screen sizes, 44
- as smallest style, 137

Layout. See also Content reference wireframes
- content/structure of design guidelines, 191–192
- sculpting unstyled HTML, 85
- sketching with content reference wireframes, 48
- specialists, 8
- static hi-fi mockups, 2–4
- static mockup comfort zone, 4–5
- as visual language component, 71
- zero interface design principle, 17–18

Layout viewport, mobile devices, 41

Linear design
- adding images to, 86
- adding style to web-based design mockups, 134–135
- adding type and color to, 83–84
- Design Funnel for, 71–73
- developing design language for, 70–71
- evolving responsive wireframes, 134
- form elements and touch devices, 87
- overview of, 70
- project folder setup, 81
- reading commands, 79–80
- sculpting unstyled HTML, 85
- serving to actual devices, 74
- starting ad hoc web server, 74–75
- structured content enhancement in, 76–77
- summary, 87–88
- templates, 78–80
- thinking and sketching, 81–82

Linear form, structured content, 55

\texttt{<link>} element, media queries, 43
Links
- adding navigation to small screens, 37–39
- to assets, from design guidelines, 189–190
- designing touchability vs. clickability for, 119
- sketching, 119
- as web building blocks, 52
- localStorage app, 179–184
- Login form, 15–16
- ls command, 65

**M**
- macros/_footer.html, 144, 149
- macros/_head.html, 144
- macros/nav.jinja, 144, 155
- MailChimp, design language, 72–73
- Major breakpoints
  - creating web-based design mockups, 137–138
  - minor vs., 100–101
  - rough sketches of, 116–118
  - taking screenshots of each screen at, 164
- Margins, 34–35, 44–45
- Markdown
  - converting plain text to HTML with, 61
  - implementations of, 62
  - making changes to document, 59–61
  - plain text markup using, 55–59
  - sectioning content for screen layout, 148–151
  - testing Dexy project, 201–202
  - using templates in, 80
  - writing design documentation with, 196–198
- Marker renderers, 3
- Markers, breakpoint, 96
- Markup
  - choosing static site generator, 140
  - plain text, 55–61
  - setting up base, 29–32
- @media, 43
- Media queries
  - implementing, 43
  - using em-based, 99
- medium.css
  - adding links to style sheets for web-based mockup, 147
  - creating variants for larger screen sizes, 42
  - implementing media queries, 43
  - for layout changes, 137
- Mental Models: Aligning Design Strategy with Human Behavior (Young), 18
- Menu button, adding navigation via, 37–39
- meta element, 41, 77
- Microstructure
  - creating content inventory vs., 19–22
  - modular structure vs., 14–15
- Minor breakpoints, 100–101, 118
- Mobile devices
  - checking files first on, 35–37
  - device classes for, 40–41
  - linear design on, 87
  - presenting interactive mockups on multiple, 175–176
  - sketching designs on, 81–82, 113–115
  - viewing your design on, 74
  - viewports, 41
- Mockups
  - birth of hi-fi static, 3–4
  - creating content inventory, 19–22
  - creating web-based. See Web-based design mockups
  - early wireframes as precursors to, 26, 48
  - exporting sketches to create detailed, 114
- Modules
  - defined, 15
  - microstructure vs. modular structure, 15
  - new CSS layout, 40
- Mood boards, involving client in, 159

**N**
- Naming
  - project files and folders, 81
  - templates, 78
- Nanoc, 139
- Navigation
  - design principle of zero interface for, 18
  - for larger screens, 41–42
  - between pages using Dexy, 155
  - sketching, 119–120
  - for small screens, 37–39

Note taking
  app, 181–184
  during client review sessions, 179
  on mockup page, 179–181
  revising mockups using, 184–185

Numbering, content inventory items, 19–22

Oblique Strategies (Eno and Schmidt), 123

Old-school wireframes
  as content reference wireframes, 27–28
  simplicity of, 5–6, 26

Online references
  browser support for new CSS layout modules, 40
  CasperJS, 202
  Cygwin, 63
  git (version control system), 185–186
  learning CSS, 154
  Markdown, 55
  markup languages, 61
  media queries, 43
  mobile viewports, 41
  Python installation, 142
  simulators and emulators, 76
  terminal applications, 63
  this book’s website as example, 16–17

Page types
  making changes, 59–61
  wireframe pages representing, 48

Pandoc
  command line tool, 62
  converting plain text to HTML, 66–67
  enhancing structured content, 77
  sectioning content for screen layout, 151
  using Dyex filters with, 143
  using templates with, 79–80

PDFs, for screenshot presentations, 169

PhantomJS, 164–165

Photoshop
  learning HTML/CSS vs. learning, 130–132
  as old school for static mockups, 10–11
  responsive design workflow and, 9–10
  sketching with, 83
  static hi-fi mockups in, 3, 5
  using command line interface vs., 63–64

pip (Python package manager), installing Dyex, 142–143

Pixels
  inserting screenshots into design documentation, 198
  presenting interactive mockups to client, 176
  using ems vs., 99–100

Plain text
  benefits of starting design with, 54–55
  converting to HTML, 61–67
  making changes to, 59–61
  marking up, 55–59
  Unicode standard on, 51
  web accessibility with, 53–54
  world’s first website using, 52

Points, breakpoint, 96

Positioning context, 45

Preprocessors, CSS, 133, 138

Presentation. See Client presentation

Presentation board, screenshots, 169

Presentation psychology, 132

“print screen,” manual screenshots, 163–164

Printed manuals, design guidelines vs., 189–190

Programming language, static site generators, 140

Progressive enhancement
  for accessible websites, 54
  creating complex breakpoint graphs, 104
  implicit breakpoints in, 94
  visualizing with breakpoint graphs, 95

Projector presentations, of screenshots, 169–170

Prompt, 64

pwd command, 64–65, 79

Pygments, Dyex utilizing, 141

Python
  Dyex software written in, 141
  installing for Dyex, 142
  starting ad hoc web server, 74–75

Python package manager (pip), installing Dyex, 142–143
Quantitative ranges, breakpoint graphs, 96, 99–100
Quantitative scale, breakpoint graphs, 95–96, 99–100

Rand, Paul, 14

rdw:mockup template
creating multiple page design manual, 196
creating own documentation, 210–211
installing Dexy, 143
Reference design, 70
Rendered code, inserting into design documentation, 198–199
Rendered HTML, including in design guidelines, 204–206
Responsive design workflow
birth of, 9–10
challenges of, 11
steps in, 212
this book’s website examples of, 16–17
Revised mockups
notes guiding, 184–185
version control for, 185
Rieger, Bryan, 40, 54, 70, 85, 119
Rieger, Stephanie, 81–82, 114
Rohde, Mike, 110, 112, 115–116
Rough layouts. See Client presentation
Rough sketches
creating comps based on, 113–116
of major breakpoints, 116–118
redrawing winning thumbnail sketches as, 112–113

Sans serif fonts, 84
Schematics. See Content reference wireframes; Wireframes
Screen sizes
adding navigation for small, 37–39
adjusting wireframe to for mobile devices, 35–37
creating variants for larger, 39–45
setting up base styles regardless of, 32–35
thinking/sketching designs for all, 81–82
Screenshots
creating for design documentation, 199–200
making Dexy configuration file, 200–201
taking of specific elements, 202–204
testing Dexy project, 201–202
Screenshots, web-based design mockups
creating automatically, 164–165
creating manually, 163–164
getting client commitment, 161–162
getting images as deliverables using, 129
inserting into design documentation, 198–199
not about tricking clients, 162
writing script, 166–169
Scriptable browsers, for automated screenshots, 164–165
SDKs (software development kits), 76
<section> element, content for screen layout, 148
Sectioning content for screen layout, 148–151
Serif fonts, 84
Shell, command line interface, 64
Simulators, 75–76
Sketching
on actual devices, 81–82
creating comps based on best rough, 113
creating rough sketches, 112–113
on devices, 113–115
as habit, 115–116
helpful practices, 109
not involving clients with results of, 159
overview, 108–109
with Photoshop, 83
selecting thumbnail sketches, 111–112
thinking about content while, 118–122
thumbnail sketches, 109–111
what to do if you get stuck, 123
The Sketchnote Handbook (Rohde), 115
Small-screen-friendly tables, sketching, 121
Software
web-based design documentation, 193–194
web-based design mockup, 133
Software development kits (SDKs), 76
Specifications, writing, 90
Static mockups
comfort zone of, 4–5
image editors as old school for, 10–11
Photoshop comps as, 23
static hi-fi mockups, 2–4
web-based mockups vs., 126–127, 160
Static site generators (SSGs)
- choosing, 140–141
- speeding things up with, 138
- from static page to, 139
- templating language and, 139–140
- for web-based design mockups, 133

Sticky note apps, 179–181
Structured content. See also Content inventory
- as building blocks of web, 52
- creating low-fi web-based wireframes, 30
- of design guidelines, 191–192
- enhancing, 76–77
- making changes, 59–61
- overview of, 14
- starting with plain text, 53–54

Structured text designs, as documentation, 90

<style> element, using media queries, 43

Style guides
- creating. See Design guidelines
- providing textual documentation with images, 91

Styles
- adjusting wireframe for mobiles first, 35–37
- sculpting unstyled HTML, 85
- setting up base, 32–35
- specifying font, 84

Syntax-highlighted code
- design guidelines software for, 194
- inserting into design documentation, 206–210

Testing
- Dexy project, 201–202
- web-based design mockups, 177–179
- websites with PhantomJS, 165

Text
- annotating rough sketches with, 112
- annotating thumbnail sketches with, 111–112
- copying all notes into file, 184
- creating simple breakpoint graph with, 99–100
- labeling and annotating breakpoints, 97
- sketching issues, 119

Text, designing in
- converting plain text to HTML, 61–67
- focus on content, 52–54
- marking up plain text, 55–61
- overview of, 52
- starting design with plain text, 54–55
- Unicode standard on plain text, 51

Text editors
- creating low-fi web-based wireframes, 29–32
- creating web-based design mockups, 132
- setting up base styles, 32–35

Text-level semantics, 14–15

Thinking
- about content while sketching, 118–122
- sketching as visual tool for, 115–116

Thumbnail sketches
- exploring ideas quickly with, 109–111
- helpful practices for, 109
- making selection of, 111–112
- redrawing as rough sketch, 112–113
- rough sketches vs., 117

Tool Rule, 56, 61, 97

Touchability, designing links for, 119

Touchscreen stylus, sketching with, 114

Training documentation, 91

Typography
- content/structure of design guidelines, 191–192
- starting design with, 83–84
- as visual language component, 71
Unicode standard, plain text, 51
Usability, testing web-based design mockups for, 177–179

Variables, as placeholders for content, 78
Version control, revised mockups, 185–186
Viewports
  mobile device, 41
  screenshots of mockups at varying widths of, 162–163
  using breakpoint graph for web-based mockups, 136–137
  variants for larger screen sizes, 39–45
  viewing width of, 137
Visual designers
  adding navigation, 37–39
  creating content inventories, 22–23
  implementing changes at breakpoints, 92–94
  interaction designers as, 8
  as obstacle to web-based mockups, 130
  in waterfall model, 7
Visual language components, 71
Visual viewport, mobile devices, 41
Visual vocabulary, site structure/interaction, 6
Visualizing breakpoints, 95

W
W3C, Media Query specification, 43
Waterfall model of website development
  making changes in, 60
  multidisciplinary, iterative approach vs., 8–9
  overview of, 7
  responsive design workflow vs., 10–11
Web accessibility
  content first for, 53–54
  sketching navigation, 119–120
  testing in web-based design mockup, 177–179
Web applications
  content and, 52–54
  designing for interaction, 6
Web-based design mockups
  adding style, 134–135
  adding style sheets, 146–147
  breakpoint graph, 136–137
  choosing SSG, 140–141
  Dexy command center, 151–153
  Dexy installation, 142–145
  Dexy software, 141–142
  evolving responsive wireframe, 133–134
  finishing with CSS, 153–154
  how perfect HTML needs to be, 135
  hurdles to acceptance of, 127–132
  major breakpoints, 137–138
  multiple pages, 154–155
  overview of, 126–127
  presenting. See Browsers, web-based mockup presentation
  reasons to not involve clients in, 159–160
  requirements, 132–133
  sectioning content, 148–151
  speeding things up, 138
  from static page to static site generator, 139
  templating, 139–140
Web documents, content and, 52–54
WebKit browsers, 164–165
Wireframes. See also Content reference wireframes
  birth of Web design specialists, 7
  multidisciplinary, iterative approach to, 9
  problems with today’s detailed, 7, 26–27
  simplicity of early, 5–6, 26
Workflow, steps in, 212
Writing design documentation, 196–198

Z
Zero interface, 17–18
Zoom, 41