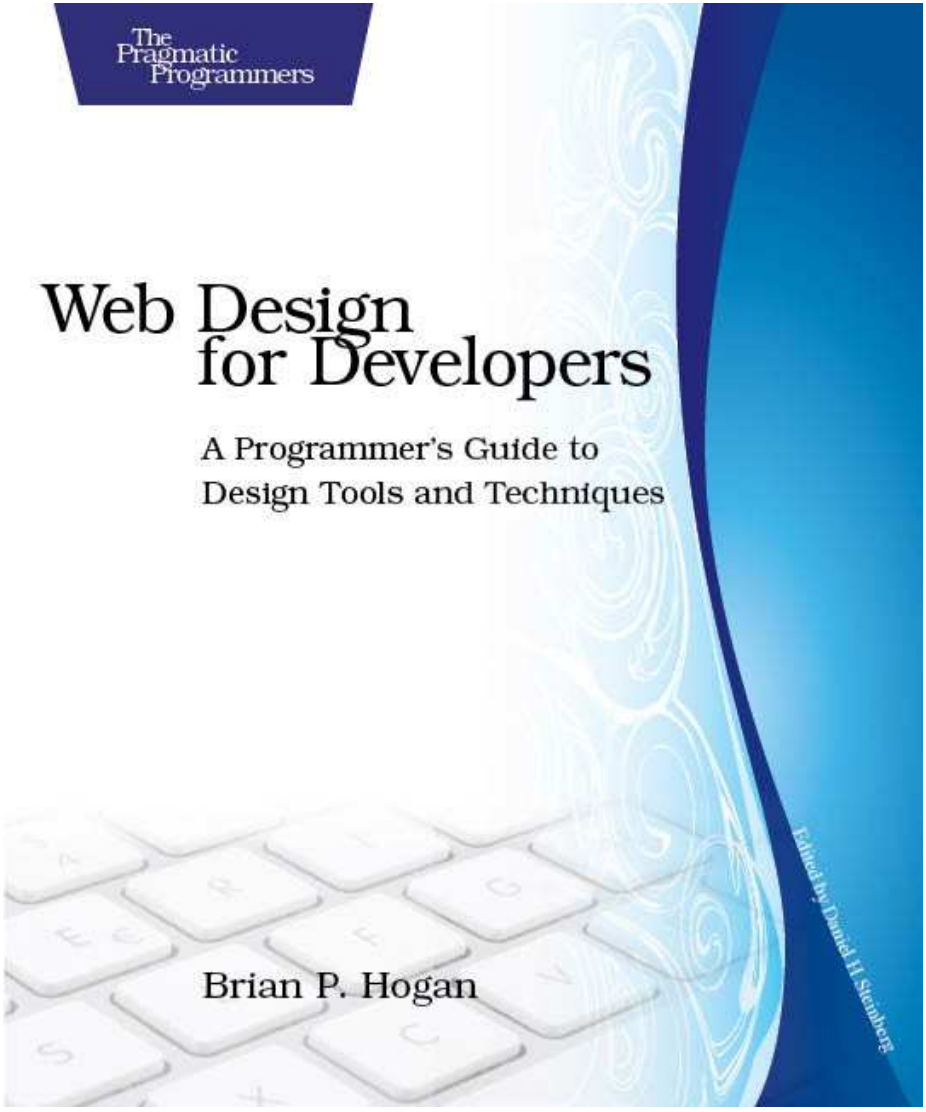
Extracted from:

# Web Design for Developers

## A Programmer's Guide to Design Tools and Techniques


The Pragmatic Programmers
Web Design for Developers
A Programmer's Guide to Design Tools and Techniques
Brian P. Hogan
Edited by Daniel H Steinberg




Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

http://www.pragprog.com



Printed in Canada.

ISBN-10: 1-934356-13-1

ISBN-13: 978-1-9343561-3-5

Printed on acid-free paper.

P1.0 printing, December 2009

Version: 2009-12-21

- The web page should be accessible to everyone, regardless of web browser, platform, or disability.
- The site meets basic usability guidelines for navigation, links, and structure.
- Behavior is separated from the content and its presentation. JavaScript that works on all platforms is used, and it degrades gracefully for platforms, devices, and users who can't use it.

This list sounds reasonable, but how do you implement something like this? You start by building a *valid* HTML document that contains your content and defines your structure. If you've ever composed simple HTML, this will be a piece of cake for you. But don't worry: even if you've never dabbled in HTML before, you'll pick up the lessons in this chapter quickly.

## 9.2 The Home-Page Structure

Try to visualize your pages as regions of content, as opposed to rows and columns, and you'll find it much easier to develop pages that not only conform to standards but are also much more flexible—you want to be able to switch out your style sheets and completely change the layout of the page.

For Foodbox, we want all the content for our sidebar to be in its own region, and we want all the content for our main area to be in its own region. We're going to do the same thing we did when we created our mock-up—divide the page up into sections.

You can divide your mock-up into four basic regions:

- header
- sidebar
- main
- footer

These four regions are easy to identify. However, you can build a flexible structure that you can manipulate easily if you further divide the page into subsections. The key to accomplishing this is to look for logical groupings of content.

For example, let's express the mock-up's regions in outline form:

- Page
  - Header
  - Middle
    - Sidebar
      - Search Recipes
      - Browse Recipes
      - Popular Ingredients
    - Main
  - Footer

In this example, we have an overall region called page. We divide this region up into a header region, a middle region, and a footer region. The outer, or parent, region, acts as a point of reference that we can use for positioning, and we can also control the overall page width by changing the width of the outer region.

The sidebar and main regions are wrapped in another region called middle. Like the outer page region, this middle region acts as a reference point, but it also serves another important purpose: it provides flexibility. We might not want a sidebar region for some of our pages; for example, we might want a full-width main region for displaying the content instead. On those pages, we could omit the sidebar and middle regions and place the content right in the main region, using CSS to resize it.

This structure is fairly common. It's the structure for your standard two-column layout with a header and a footer, one of the most common website types. The neat thing about standards-based design is that you can reuse this skeleton for another project if you want to, because your style sheets will define your column widths, colors, and other visual elements.[1]

## 9.3 Semantic Markup

Semantic markup makes sure your document is structured so that it can be interpreted by machines, devices, or people. For example, Google's web crawler uses tags such as h1 and href attributes on links to determine the importance of web pages and their content.

1. This approach works great for *skinning* a website; you could use this technique to let your users have their own themes. Visit http://www.csszengarden.com to see a great example of a single document rendered in multiple ways.

### Joe Asks...

#### Can't We Just Slice and Dice Our Mock-Up?

In the old days of web development—and by "old" I mean those medieval times of 2004—it was common practice for developers to take a Photoshop document and use tools like Fireworks or ImageReady to slice the image up and generate HTML. This approach gives you a quick-and-dirty way to make a web page, but it also has some serious problems.

For example, it almost always involves using HTML tables for layout. This was *the* way that every web designer built web pages before CSS became a viable alternative. Among the many problems with this approach was that it made life more difficult for users who browse with screen readers.

Also, this approach doesn't separate the content from the design, so you can't easily make multiple presentations of your content available, such as a version of your site for printing and another version customized for display on mobile devices.

Finally, and most important, using tables for layout means that you will duplicate all the table HTML code on every page of your site. Every time someone requests a page, that data must be transferred to the end user. On a small-scale site, this just means your pages might take longer to get to the end user. If you run a site that gets lots of hits, you might start to see it in your monthly bills from your ISP. When you host a website, you have to pay for all the traffic that you serve, so if you have a lot of traffic, it's in your best interest to reduce file sizes wherever you can.

Designing with CSS and web standards allows you to define the look and feel of a website using files that end users download only once but share across all the pages you serve them. This improves performance and saves you money.

You need to use HTML tags for their intended purpose so that they describe the content they contain properly. Your page will have headings, paragraphs, lists, and other elements. HTML has lots of tags that are designed to mark up content. Headings, for example, should use syntax something like <h1>About Us</h1>. An HTML parser will see this tag and know it's the most important headline on the page.

It would be completely inappropriate then to do something like <font size="+2">About Us</font>. Unfortunately, many developers do precisely this because they don't like the fact that, by default, the h1 tag places a margin above and a line break below this tag's content when it is rendered.[2]

You can use CSS to solve the visual issues quite easily once you understand how everything works. For example, you might use CSS to change the way all headings look, or you might use it to modify the appearance of a single heading on a single page. Best of all, one CSS file can be applied to many pages, so instead of setting every heading on 100 pages, you can add a couple of lines to your style sheet.

## 9.4 The Home-Page Skeleton

Open your favorite text editor,[3] and create a new file. Immediately save this new blank file as index.html. The index.html page will be the home page for the site. Web servers will serve up the index page whenever a request comes in for a path and a page is not specified.

### The Doctype

Each HTML page must have a doctype to help a validation tool ensure you're serving properly coded markup. It's extremely important to make sure you have a valid page before you apply style sheets or JavaScript. Invalid markup can cause styles to be applied incorrectly or cause JavaScript code to fail horribly. Your web browser relies on a well-formed document to apply styles and behaviors properly, so failing to close a tag might trip up a user's browser.

More important, doctypes force certain browsers to interpret a page differently. For example, Internet Explorer 6 has a *quirks* mode that is extremely forgiving to invalid markup, but you can spend a lot of time

2. Some WYSIWYG HTML editors write code like this too, so it's not just novices.
3. I recommend Notepad++ for Windows and TextMate for Mac.

### Default Page Names

Web servers have a concept called *default pages*. A default page is rendered whenever a page is not specified for a directory. Web servers serve files from a directory structure. You have pages within folders, and the universal resource locator (URL) contains the path to the folder and file the user requests. For example, if you requested the http://www.foo.com/products/superwidget/about.html URL, the web server at http://www.foo.com would look in the products/superwidget folder for a file called about.html.

If you requested the http://www.foo.com/products/superwidget URL, then you've requested an incomplete resource, so the web server tries to figure out what you meant. First, it looks to see what actually exists at that location on the server. If it finds a folder there, it looks at a list of default filenames and then checks to see whether any of those filenames exist within that folder. Common default filenames include index.html, index.htm, and default.htm.

If the server can't find a default file, it might return a directory listing, or it might return an error message if an administrator configured the server to not allow directory listings. Many webmasters believe that disabling directory browsing adds a level of security to their sites; however, I don't think you gain much security by doing that. If you don't want people to see something, don't publish it to the Web.

When you link to a resource that has a default page, you should either include the filename in the URL or use a trailing slash after the directory name. This *courtesy URL* tells the server that you are in fact requesting a directory from the server, and you expect the server to return the default file. Courtesy URLs work best on the home page of a site.

For maximum performance and to avoid confusion, you should always link directly to the complete resource. Links to the Foodbox home, for example, should always end with index.html. This way, the server can just serve that file and then get on with handling the next request.

scratching your head trying to make your page work in other browsers that are more strict about what they will render. However, you can use a doctype declaration that forces IE 6 into *standards* mode, which isn't perfect, but it'll get us by.

You can choose from a few different doctypes. The doctype you use dictates what tags you can use in your document, as well as the validation rules that will be used to check your markup. The two most frequently used doctypes are *XHTML 1.0 Transitional* and *HTML 4.01 Strict*.

### XHTML 1.0 Transitional

For a long time, XHTML Transitional was considered *the* way to build pages for the Web. A primary reason for its use was that it forced web browsers into standards mode. That's not much of an issue today, but XHTML continues to have some advantages over regular HTML. XHTML markup is more strict, which forces developers to think more about a page's structure. It also requires that you use lowercase letters when defining tags and attributes, which can be helpful when parsing documents. Finally, it requires every tag to have a closing tag.

Unfortunately several browser support issues undercut the benefits of using XHTML, including its extensibility. Internet Explorer does not understand how to handle XHTML unless it's served as HTML using the text/html content type instead of the more appropriate application/xhtml+xml. Serving XHTML as HTML forces browsers to deal with tag soup; the browser expects HTML tags, but it gets XHTML instead, so it spends time reworking the document.[4] You lose a lot of the benefits of XHTML that your users see, and these browser issues can in fact introduce some new problems into your page. For example, self-closing div and span tags, which are perfectly valid in XHTML, get their trailing slash removed by browsers when served as text/html, which leaves them unclosed, affecting all elements that follow.[5]

These issues have prompted some designers and developers to switch back to using regular HTML again, in the form of HTML 4.01 Strict[6] or the HTML 5 specification.

---

4. http://xhtml.com/en/xhtml/serving-xhtml-as-html/
5. http://www.webdevout.net/articles/beware-of-xhtml#myths has some great examples of how the content type affects the output of a page written in XHTML.
6. http://mezzoblue.com/archives/2009/04/20/switched/

### HTML 4.01 Strict

We're using HTML 4.01 Strict in this book's examples. With HTML 4.01 Strict, elements must still adhere to a hierarchy, but case doesn't matter, some tags don't need to be closed, and self-closing tags don't exist. It's important to remember that these are only language issues, and they don't make HTML's syntax any worse or better than XHTML's syntax. As long as you make sure you validate your documents, you'll have no trouble with browser compatibility, user experience, accessibility, CSS, or JavaScript.

We'll use HTML 4.01 Strict in these examples, but I'll make sure to stress well-formed, valid, semantic markup. This will keep a future transition to XHTML 1.0 Strict or HTML 5 simple. Whichever doctype you choose to use in your work, you should realize that you almost always serve both doctypes to browsers as HTML, so the only *real* difference between the two doctypes is syntactical. Don't let yourself get caught up in a holy war.

### Adding the Doctype

Place this doctype declaration in your document. Everything else in your document goes after the doctype.

Download homepage_html/index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

Don't bother typing it in yourself, though. Most web page editors have a template you can use, or you can go to your favorite search engine and search for *HTML 4.01 Strict doctype* to find an example.

## The HTML Tag

A web page is a hierarchy of elements, much like an XML document. The html element is the root element of the document. All other elements in the document will reside within that element. Almost all elements in a web page have an opening tag and a closing tag. You can think of the opening and closing tags as scope markers, similar to curly braces in Java.

Add the html tag to your document immediately after the doctype, and be sure to add the closing tag. This is a good habit to get into when you do web-page development. Add the element's tag, immediately add the closing tag, and then reposition the cursor between the opening and closing tags. Forgetting an element's closing tag results in invalid

### Joe Asks...

#### Is XHTML Dead?

The W3C's recent decision to stop work on the next version of XHTML to focus more resources on HTML 5* has not killed off XHTML 1.0. but it does show that HTML 5 is the way to go when it comes to web markup.

The main reason many programmers and standards advocates prefer XHTML over HTML is its strict syntax. All tags must have closing tags, all tags and attributes must be in lowercase, attribute values must be quoted, and stand-alone elements like br, img, meta, and hr need a trailing slash. With the exception of the self-closing elements, all these are perfectly legal with HTML 4.01 Strict, and you can use every one of these coding practices with HTML 5.

XHTML isn't going to be worked on anymore, so it's dead in the same way that COBOL is dead—it works, and it's not going away any time soon. You shouldn't rush out and convert all your sites to HTML 4.01 Strict or HTML 5, but you should consider all your options when you start work on a new site.

*. http://www.w3.org/News/2009#item119

markup, which in turn causes browsers to apply your styles in strange ways. Invalid markup also causes other web developers to break out in a rash of expletives or, worse, punches. You should do your best to avoid this.

Download homepage_html/index.html

```
<html lang="en">

</html>
```

## Attributes

Each tag supports various attributes that you can specify within the tag's declaration. Attributes help describe the tag in more detail. The html tag we used has an attribute that describes the language we use in this document.

**Self-closing Tags**

If you're used to XML, you might be familiar with the idea of self-closing tags, or tags that have a trailing slash when there's no closing tag. The HTML 4.01 Strict doctype doesn't support these, but the XHTML 1.0 Strict and Transitional doctypes do, and so does the HTML 5 doctype.

## The Head and Body

You can always find two elements within the scope of the html element: head and body. The head element contains all the metadata about the page, including the page's title that appears in the bookmark link and in the browser's title bar, as well as links to load JavaScript files, style sheet files, and other assets. The body element contains the visible contents of the web page.

Add the head tag and its associated closing tag to your document, immediately below the html tag you just defined:

Download homepage_html/index.html

```
<head>

</head>
```

It's a good idea to indent your tags, just as you would indent code within an if..else statement. Doing this will help you later, when your document gets bigger.

Add these two lines to the head element:

Download homepage_html/index.html

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

## Tags Without Closing Tags

Some tags in HTML don't have any scope because they don't wrap any content or perform any transformation on content. Many of these tags can be considered content themselves.

Examples of this include the img tag, which inserts an image into the document; the br tag, which adds a soft line break; and the hr tag, which creates a horizontal rule.

**Joe Asks...**

**Aren't You Supposed to Set the Content and Encoding in the HTTP Headers?**

You are absolutely supposed to set the headers correctly, but some browsers use the value of the meta tag anyway, as do the validators. Using the meta tag in the page's source can only help you describe your content better. Other developers can use the value in the meta tag to see your intentions when they follow your work.

Finally, and most important, using the meta tag lets you develop and validate HTML that's not served by a server. You can open an HTML file on your hard drive, and it will render with the correct encoding.

When you do serve the file from a server, make certain that the value for the Content-Type header matches what you specified with the meta tag.

The meta tag is an example of a content element. This tag lets us describe our document with metadata. In this case, we use a meta tag to tell the browser or interpreter what character set our content will use. Sometimes you might paste in content from another source, and this content might contain symbols, curly quotes, or other characters that can't be viewed in all browsers or on all computers. Specifying a certain character set causes HTML validators to alert us when we use content like this.

We can use meta tags to provide more information to browsers, search engines, and other consumers of our page. We'll do a lot more with these tags in Chapter 18, *Search Engine Optimization*, on page 257.

### The Page Title

The title tag is important. The text you place within that element will be displayed in the title bar of the web browser. It's also used as the default text when a person bookmarks the page, and it shows up in the search results for most search engines. In this case, the name of the site is good enough, but subsequent pages should have additional text in that element, such as *About This Site* | *Foodbox* or *Top Recipes* |

**Block and Inline Elements**

Almost all elements that reside within the body tags of your page are either block or inline elements. Understanding the difference between these types of elements can save you a lot of time when you're ready to style your pages with CSS.

By default, block elements begin on a new line. Examples of block elements include div, h1, h2, h3, p, ul, li, table, and form.

Inline elements, on the other hand, are rendered on the same line as other elements by default. Examples of inline elements include a, b, i, span, em, strong, label, select, input, textarea, u, and br.

You want to remember this point: block elements can contain other block elements or inline elements. Inline elements can contain only text and other inline elements; they cannot contain block elements.*

*. They might render in a browser, but your page won't be valid, and you will have a lot of trouble applying styles or working with JavaScript later.

*Foodbox*. The title displays in a site's bookmark and in the title bar, so we want to place the site name in all the headings. However, it might get truncated, so we also want a specific part of the title to show up first. For example, *Latest Recipes* | *Food...* looks better to users and search engines than *Foodbox* | *Latest Rec...* does.

The head section of the page will contain more elements as you move closer to the finished product, but you can begin building the visible part of the page right now. It makes no sense to do much search engine optimization or scripting at this stage.

### The Body: The Main Event

All of the visible content of your page resides within the body tag.

Add the body and closing tags to our document, leaving some space between the tags so we have some room to work. At this point, we've built a standard HTML 4.0 Strict template (see Figure 9.1, on the following page).

You learned how to break down the elements of the page into sections in Section 9.2, *The Home-Page Structure*, on page 123. Now you have

Download homepage_html/index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">

<html lang="en">

  <head>

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

  <body>

  </body>
</html>
```

homepage_html/index.html

Figure 9.1: An example of a default HTML template

to mark those sections up with code. To do that, use the div tag to *divide* the page into sections. div tags are invisible elements, so they don't take up any visible space on the page when it's rendered. They do have some special properties, though. For one thing, they are block elements, which means they begin on a new line. You can find a more detailed explanation of block elements in the sidebar on the preceding page.

### The Page Wrapper

We can constrain all the content in the page we're creating to our desired width of 900px by creating a top-level region. We will place all the other regions of the page, such as sidebar, header, and footer, in this new region. Later, you can use this outer region as a point of reference for all other elements. Good coders document their code, and HTML permits comments, so add this code immediately after the opening body tag:

Download homepage_html/index.html

```
<div id="page"> <!-- start of the page wrapper -->

</div> <!-- end of the page wrapper -->
```

You must give the browser some way to identify the regions of your page so it can apply styles and behaviors. Note that the id attribute is unique

to the document. This means you can't have more than one page id on a single page. If you do, your page won't validate, and it will likely start doing strange things when you apply styles.

The HTML comments in that code might prove a big help later, when the document gets longer and harder to read.

### The Four Content Regions

You can use div elements to stub out the header, footer, sidebar, and main regions of the page:

Download homepage_html/index.html

```
      <div id="header"> <!-- start of header -->

      </div> <!-- end of header -->
      <div id="middle"> <!-- container for the sidebar and main region -->
        <div id="sidebar"> <!-- the sidebar -->

        </div> <!-- end of the sidebar -->

        <div id="main"> <!-- start of main content -->

        </div> <!-- end of main content -->
      </div> <!-- end of middle container -->
      <div id="footer"> <!-- start of the footer -->

      </div> <!-- end of the footer -->
    </div> <!-- end of the page wrapper -->

  </body>
</html>
```

This example includes an extra div called middle. Whenever you have two regions that you'll eventually want to display side-by-side, you should wrap those two regions with another region. It doesn't add that much extra markup to the document, and it makes your design more flexible. For example, if you need to eliminate the sidebar on another page of the site, you could omit the two inner regions and style the outer region. Here we wrap the sidebar and main regions the same way we wrapped the entire page.

We've put the structure in place, so let's add the content.

**Alternative Text**

The alt attribute for images gives you an easy way to improve the usability and accessibility of your site. Alternative text is displayed when images can't be displayed. Users who are blind rely on alternative text to describe the images to them, so it's a good idea to make your descriptions *descriptive* rather than vague! "A blue car" isn't as strong as "A vintage 1957 blue Chevrolet in front of the downtown mall."

Alternative text also comes in handy for text-based browsers and mobile-phone users with low-bandwidth connections. Another reason to make sure you always include good alternative text descriptions for your images is that search engines use them. Search engines can't read your images either, and your alternative text descriptions become extremely important at that point. I'll cover this issue further in Section 16.2, *Alternative Text Attributes*, on page 233.

## 9.5 The Header

The content for the header region consists of only the Foodbox logo, which we'll include with the img tag. This tag has a src attribute that specifies the path to the image. This path works like the href attribute of the a tag; it can be a URL or a relative path to a file. We'll discuss URLs in detail in Section 9.6, *The Recipes Tag Cloud*, on page 139.

When placing an image on a web page, it's always a good idea to specify the height and width of the image. We don't have the image right now, so we'll let that go for the moment; however, we definitely want to come back later and add this. For now, specify the image source and an alt attribute for the text. This alternate text gets displayed if the image can't be loaded; it's also extremely helpful for your users who use screen-reading software.

Place your cursor within the region defined by div id="header", and insert the following code:

Download homepage_html/index.html

```
<img src="images/banner.png" alt="Foodbox">
```

# The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

# Visit Us Online

**Web Design for Developers' Home Page**
http://pragprog.com/titles/bhgwad
Source code from this book, errata, and other resources. Come give us feedback, too!

**Register for Updates**
http://pragprog.com/updates
Be notified when updates and new books become available.

**Join the Community**
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

**New and Noteworthy**
http://pragprog.com/news
Check out the latest pragmatic developments in the news.

# Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/bhgwad.

# Contact Us

| | |
|---|---|
| Phone Orders: | 1-800-699-PROG (+1 919 847 3884) |
| Online Orders: | www.pragprog.com/catalog |
| Customer Service: | orders@pragprog.com |
| Non-English Versions: | translations@pragprog.com |
| Pragmatic Teaching: | academic@pragprog.com |
| Author Proposals: | proposals@pragprog.com |