2 Process

Find the perfect workflow.

Focus on interactive prototypes.

Follow best practices for creating prototypes.

Searching for a Perfect Workflow

During 2005 I worked with a company that was driven by technology and process rather than creativity. Planning and procedures were considered more important than creative experimentation. In this company, dry-wipe boards full of mantras and flow diagrams lined the walls, making me almost long for those dreadful "soar like an eagle" motivational posters so loved by business managers who read *1001 Ways to Energize Employees* on their days off!

Every day, designers and developers alike were subject to the ritual humiliation of reporting on what they had achieved the previous day and what they hoped to accomplish the next. (Sometimes, out of sheer belligerence, I would say, "I thought about tabs.") It was a miserable experience that left us all handcuffed due to a rigid process that gave little room for creativity.

Most Web design workflows continue to follow traditional patterns for design and development. Despite the best intentions, adopting these outdated patterns can limit both your creativity and your efficiency. They can also negatively impact how you work and how you communicate with others during the process.

Looking for a better way

Visiting with a variety of other designers in organizations and companies large and small, I see many similarities with my own experiences, particularly in the way designers often work separately from the technical developers and others working on the same project.

We need to find a better way, one that values the creative process, enhances the overall quality of the final product, and improves the working relationships between all involved.

The key to developing a new process is to give everyone a central point around which to focus. This focal point should not be presentation, as it has been in the past, but the meaningful content that we convey to our visitors. And although some of the techniques that designers and developers have used in the past still have many merits, these techniques are often disconnected from each other. To work more effectively, we need a way to better connect them.

The Web is a dynamic and interactive medium. The best way to join these disconnected tasks is to focus the efforts of everyone involved—information architects and others working primarily with content, visual designers, and technical developers—on the same things that make up the essence of the Web. Using content, meaningful XHTML markup, and CSS to develop interactive prototypes makes the connection and helps designers, developers, and other specialists do their jobs better and more efficiently.

It's easy to understand why many devices of the early planning stages—content site maps, flow diagrams, and wireframes—use static images, but the Web is an interactive medium. Whereas wireframes and static designs can only hint at the interactivity of a finished Web site, interactive prototypes can do so much more. To make the most of the medium, we should work *using* the medium.

Why should we do this?

We should create a new workflow for the following reasons:

- To improve our knowledge and understanding of all parts of the design and development process
- To help us better communicate the organization and relationships in content
- To convey this meaning and these relationships through our visual designs and layouts
- To improve our efficiency and enable technical developers to implement a design and add functionality much earlier in the process

In this part, you will find new ways to make all this happen. You will learn best practices for making wireframes and interactive prototypes and will finish by making an interactive prototype using all the techniques you learn about in this part.

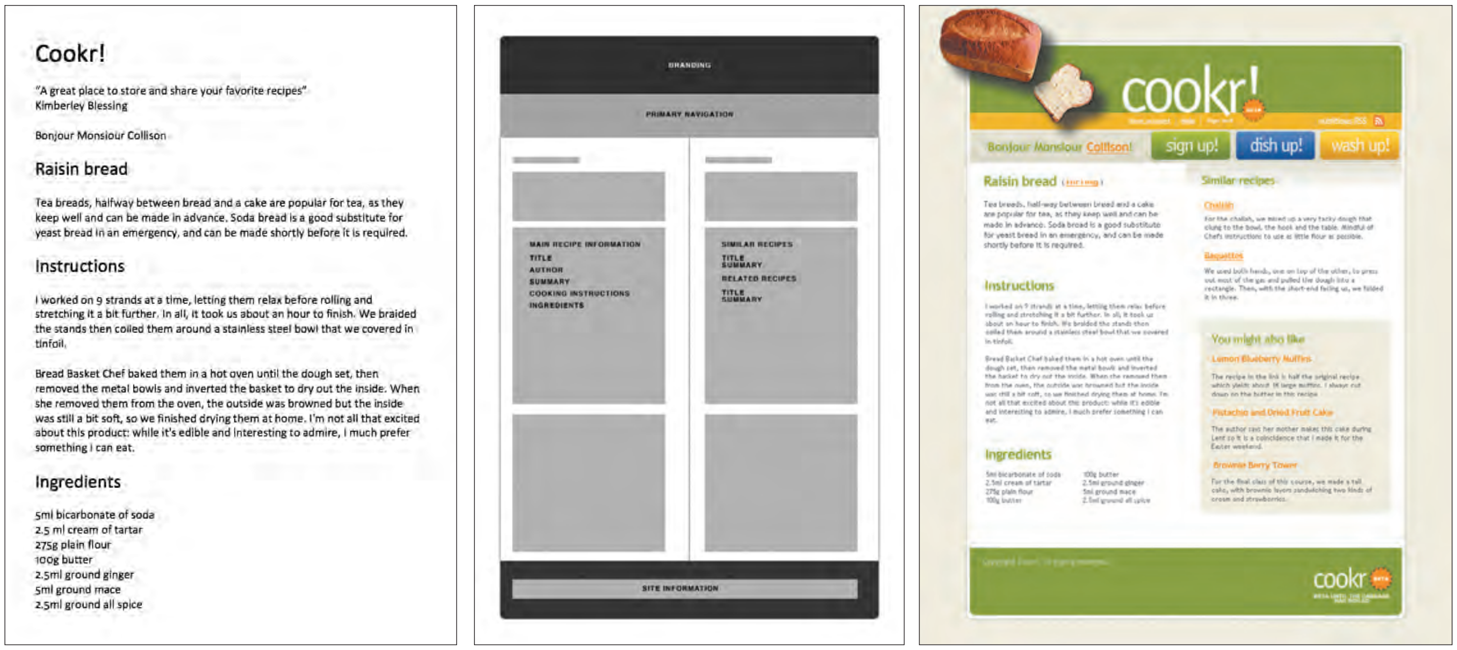
Following a content-based process

Content is often said to be king, a king who demands the attention of everyone in the design and development life cycle, including information architects, user experience and accessibility strategists, visual designers and technical developers, and copywriters and editors. Content in the form of text, links, photos, audio, and video is the foundation of any site.

In Part 1, "Discovery," you were introduced to the content-out approach to markup. Taking this approach a stage further, you will see how this approach creates an opportunity for you to build a completely new type of workflow. In this new process, content is the primary focus, influencing every stage of planning, designing, and developing. The content-based process includes many familiar steps (**Figure 2.1, left to right**):

1. Gather the content.
2. Work with wireframes to organize and present that content.
3. Create static designs to demonstrate creative concepts and layout ideas.
4. Write meaningful markup that structures the content.
5. Work with CSS to implement the design.

2.1 **Following the steps of a content-based process**



Cookr!

"A great place to store and share your favorite recipes"
Kimberley Blessing

Bonjour Monsiour Collison

Raisin bread

Tea breads, halfway between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

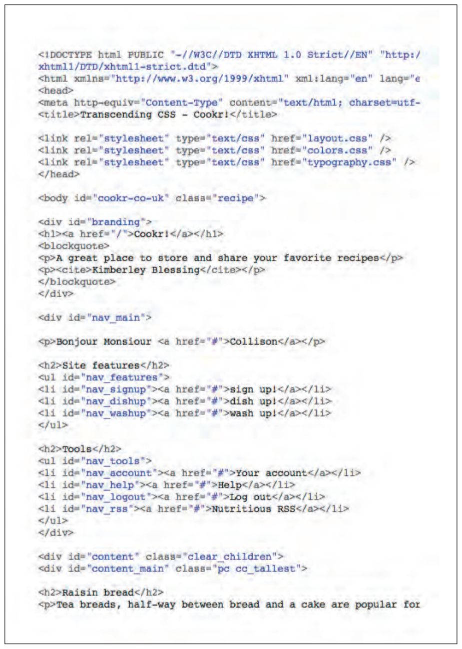
Bread Basket Chef baked them in a hot oven until the dough set, then removed the metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

Ingredients

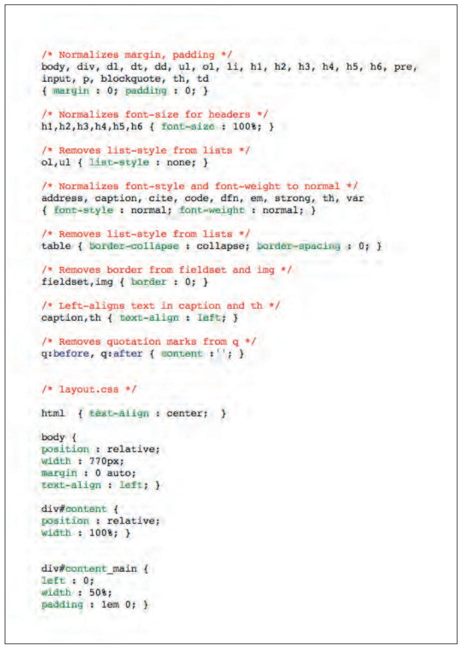
5ml bicarbonate of soda
2.5 ml cream of tartar
275g plain flour
100g butter
2.5ml ground ginger
5ml ground mace
2.5ml ground all spice

All these steps lead to the creation of an interactive prototype, which you can use to better communicate with everybody on your team and with your clients. The interactive prototype helps designers make design iterations and refinements and test them with visitors and with clients. The interactive prototype also provides a solid platform for technical developers to create fully interactive experiences with scripting and programming.

If by now your heart is pounding in anticipation, start by looking at gathering content from your clients and other sources.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http:/
xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="e
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-
<title>Transcending CSS - Cookr!</title>

<link rel="stylesheet" type="text/css" href="layout.css" />
<link rel="stylesheet" type="text/css" href="colors.css" />
<link rel="stylesheet" type="text/css" href="typography.css" />
</head>

<body id="cookr-co-uk" class="recipe">

<div id="branding">
<h1><a href="/">Cookr!</a></h1>
<blockquote>
<p>A great place to store and share your favorite recipes</p>
<p><cite>Kimberley Blessing</cite></p>
</blockquote>
</div>

<div id="nav_main">

<p>Bonjour Monsiour <a href="#">Collison</a></p>

<h2>Site features</h2>
<ul id="nav_features">
<li id="nav_signup"><a href="#">sign up!</a></li>
<li id="nav_dishup"><a href="#">dish up!</a></li>
<li id="nav_washup"><a href="#">wash up!</a></li>
</ul>

<h2>Tools</h2>
<ul id="nav_tools">
<li id="nav_account"><a href="#">Your account</a></li>
<li id="nav_help"><a href="#">Help</a></li>
<li id="nav_logout"><a href="#">Log out</a></li>
<li id="nav_rss"><a href="#">Nutritious RSS</a></li>
</ul>
</div>

<div id="content" class="clear_children">
<div id="content_main" class="pc cc_tallest">

<h2>Raisin bread</h2>
<p>Tea breads, half-way between bread and a cake are popular for
```



```
/* Normalizes margin, padding */
body, div, dl, dt, dd, ul, ol, li, h1, h2, h3, h4, h5, h6, pre,
input, p, blockquote, th, td
{ margin : 0; padding : 0; }

/* Normalizes font-size for headers */
h1,h2,h3,h4,h5,h6 { font-size : 100%; }

/* Removes list-style from lists */
ol,ul { list-style : none; }

/* Normalizes font-style and font-weight to normal */
address, caption, cite, code, dfn, em, strong, th, var
{ font-style : normal; font-weight : normal; }

/* Removes list-style from lists */
table { border-collapse : collapse; border-spacing : 0; }

/* Removes border from fieldset and img */
fieldset,img { border : 0; }

/* Left-aligns text in caption and th */
caption,th { text-align : left; }

/* Removes quotation marks from q */
q:before, q:after { content :''; }


/* layout.css */

html  { text-align : center;  }

body {
position : relative;
width : 770px;
margin : 0 auto;
text-align : left; }

div#content {
position : relative;
width : 100%; }


div#content_main {
left : 0;
width : 50%;
padding : 1em 0; }
```

As long as we have the outline and breakdown of how content is organized and prioritized on a page, we do not need to have the final content in place until the site is completely built.

—KELLY GOTO, Goto Media
www.wise-women.org/features/kelly_goto/

Gathering Your Content

Goto's assertion (**facing page**) that "we do not need to have the final content" is effective when you are working at a higher level and are dealing with the overall shape and structure of your pages. However, this process breaks down when working with the specifics of markup. This perspective begs the question, how do you convey the meaning of content through markup when you have no precise content to work with?

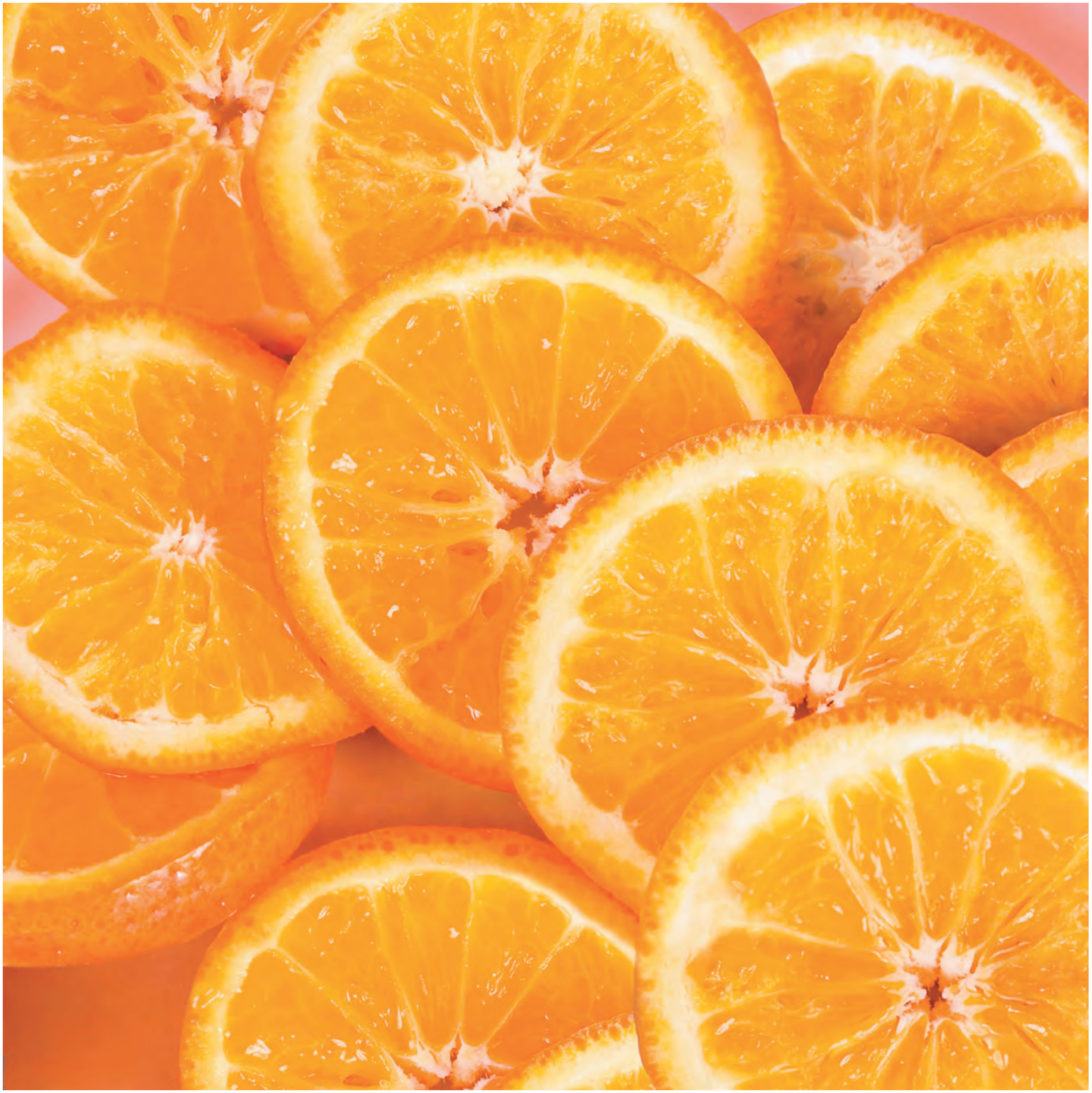
Note: It has been a common working practice in both print and Web design for designers to lay out their pages using Greeking text (the familiar *Lorem Ipsum*). Other designers prefer to use short paragraphs of text from works in the public domain, such as *Moby Dick*, or covered under the Creative Commons license. Mock text doesn't provide the meaning you want in order to begin immediately marking up your documents semantically.

Ensuring that content is delivered on time and in the right format is not only a concern for visual designers and technical developers. Information architects and others who organize content also depend on it, as visual designers and even search engine optimization specialists.

You have many ways to ensure you receive content so your job can run more smoothly:

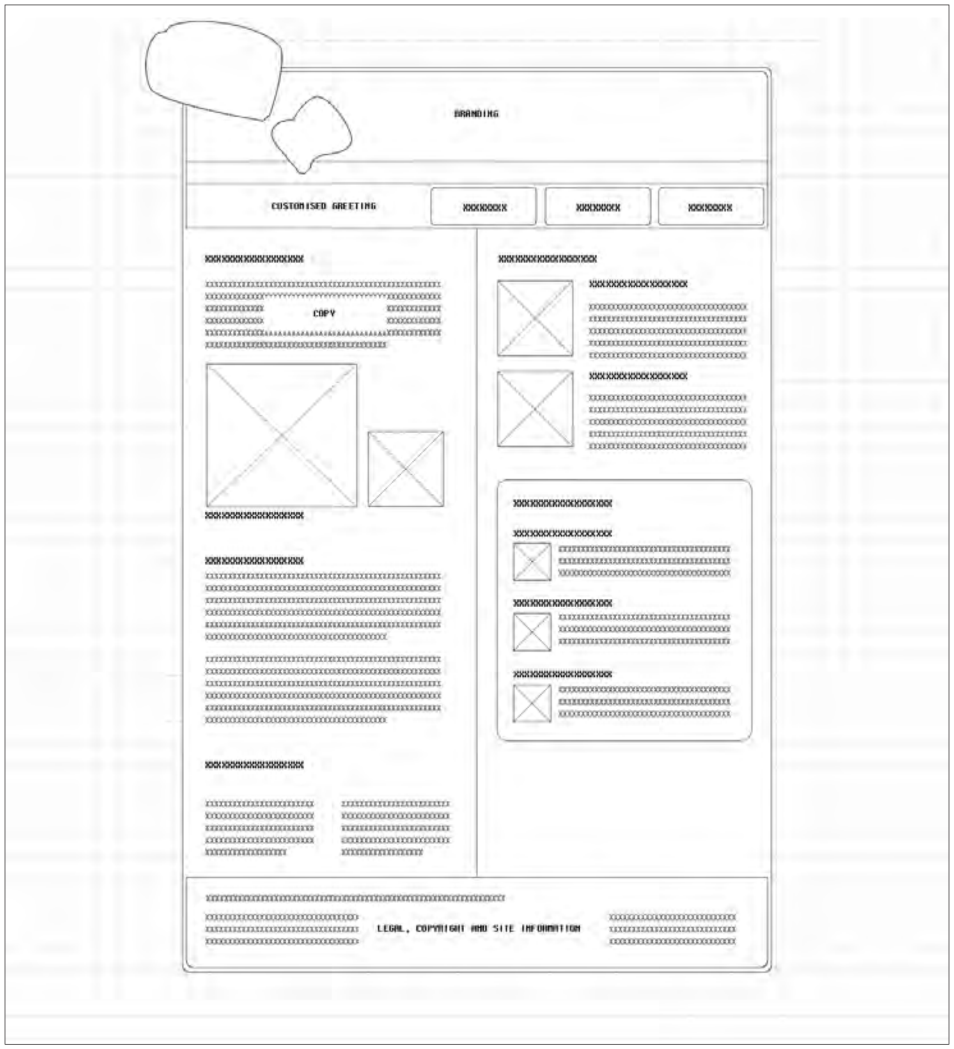
- Work with your clients to create inventories of existing content.
- Provide your clients with a *new content brief*, which gives an overview of the new content that will likely be required and who will be required to work on the different areas.
- Include milestone dates in your contracts, and use Basecamp or even a simple spreadsheet to keep track of content delivery.

Note: You can learn more about content briefs from D. Keith Robinson's article "Content Brief" at www.7nights.com/asterisk/archives05/2005/05/content-brief. You can learn about the collaborative project management tool Basecamp at www.basecamphq.com.



Working with Wireframes

Traditional wireframes are black-and-white diagrams that illustrate blocks of content, navigation, or functionality. They have been a familiar sight to Web designers and developers and are broadly understood by both clients and Web professionals (**Figure 2.2**).



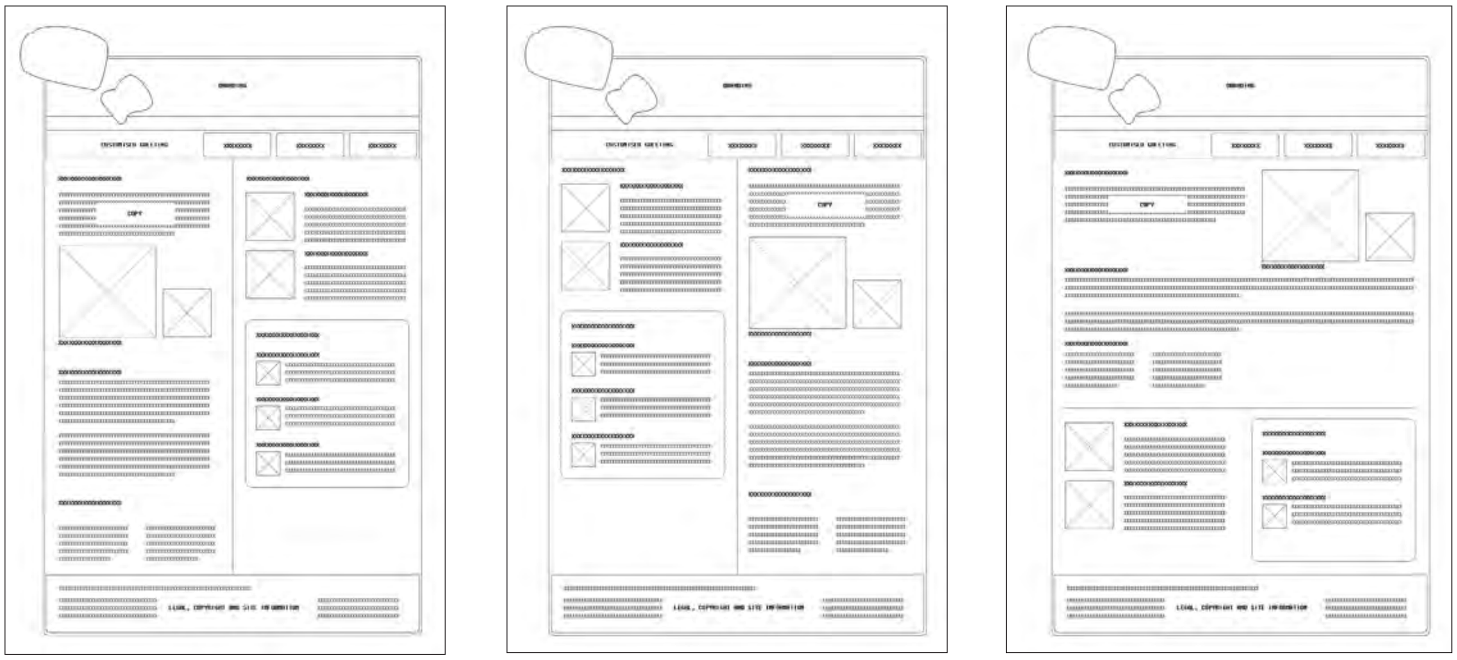
2.2 **Detailing a wireframe**

Used as a tool to communicate content and structure without the distractions of color and imagery, wireframes remain an important part of the design process.

They can help designers do the following:

- Storyboard a visitor's path through a site.
- Work quickly through a series of layout iterations (**Figure 2.3**) before the costly job of creative design and technical development begins.

Although wireframes are not quite shake-and-bake, almost anyone can easily create wireframes with OmniGraffle from the Omni Group or even PowerPoint from Microsoft, with little or no knowledge of Web technologies. Just like pouring boiling water onto an instant Pot Noodle, using common software to create simple wireframes is easy. It is perhaps because of this ease that many people have come to regard making wireframes as an inexpensive way of proofing concepts.



2.3 **Working through a series of layout iterations**

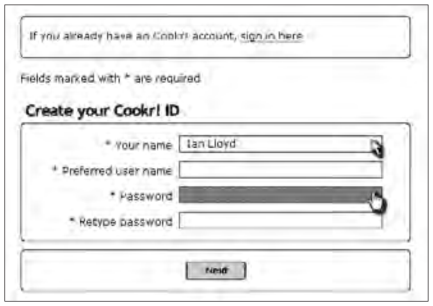
Where traditional wireframes fail

Frederick Barnard has been credited with saying “a picture is worth a thousand words” in the advertising journal *Printer’s Ink*. Sadly, what worked for Barnard in 1921 is not altogether relevant eighty-five years later. However much care you take in creating them and however detailed or well annotated you make them, traditional wireframe images can only hint at what will become rich content and navigation on the Web.

Images work just fine for static designs to show creative concepts, color, and typography, but using them as prototypes for interactive Web pages is flawed from the start. Using images makes it difficult to mimic even the simplest forms of interaction such as `:hover`, `:focus`, or `:target` states (**Figure 2.4**).

A further drawback of wireframes is that they are often created long before a visual designer begins the job of creating page-layout concepts. Sometimes they include not only information about content and relationships, but also page-layout instructions such as the location of branding and content areas, sidebars, navigation, and footers.

Many visual designers, when presented with wireframes that contain so much detail, feel they have little room to express their creativity. Decisions over design and layout have already been made *before* the job of visual design has even started. Cases like these are common and problematic because the designer’s valuable input has been overlooked.



If you already have an Cookr! account, sign in here

Fields marked with * are required

Create your Cookr! ID

* Your name	Ian Lloyd
* Preferred user name	
* Password	
* Retype password	

2.4 **Using images makes it hard to show interaction**

Improving the approach using granular wireframes

To help avoid being overly prescriptive and dictating the complete layout of a page, some have advocated using more granular wireframes. These break down important features into smaller pieces, such as the following for a Web application or an e-commerce site: account creation, customer sign-in, customization options, e-commerce checkout, navigation, and search interfaces.

This type of wireframe helps designers stay creative, because they do not specify the layout of an entire page. However, they are still far from ideal:

- They lack the capability to describe the semantic meaning of elements or the relationships between them to visual designers.
- Despite their precision, they often fail to describe in sufficient detail all the information that technical developers need to understand complex functionality or interactivity.

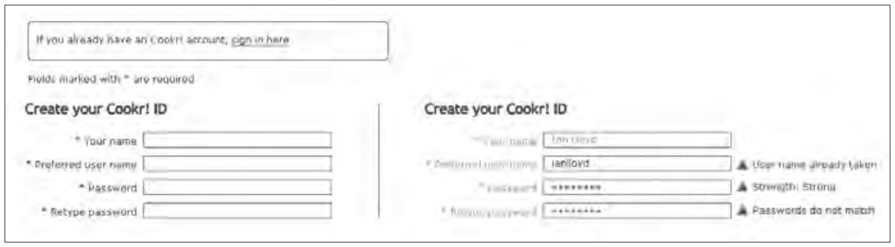
Although traditional wireframes can provide a broad indication of a finished product, they are often mistakenly used as benchmarks for how a layout will ultimately look. Worst of all, they reinforce the idea that Web pages should be pixel-perfect reproductions of frozen images. Often, they lock a design into one fixed display type and rarely take into account the need to design for users of alternative browsers, such as screen readers or mobile devices.

Are wireframes a good value for the money?

It is a popular misconception that making wireframes is an inexpensive part of design and development; in the context of a modern workflow, they represent much less value for the money.

Prototyping of any kind is often thought to involve working on materials that will ultimately be discarded before creating a final product. If you start your process with images as a wireframe, you can quickly turn this notion into a self-fulfilling prophecy. When I look at the hundreds of wireframes I have made, all now gathering digital dust in my archives, I can picture the thousands of hours that went into creating them. Because I approached these wireframes as part of a throwaway process, that is exactly what they became.

In addition, they may seem quick and easy to make, but in reality wireframes rarely consider every aspect of the complex nature of interactive Web pages. Issues such as pagination, error or status messages, and visitor feedback are rarely tackled at the wireframing stage, and by not including them, the missing work is simply moved to a stage later in the process (**Figure 2.5**).



If you already have an Cookr! account, sign in here

Fields marked with * are required

Create your Cookr! ID

* Your name
* Preferred user name
* Password
* Retype password

Create your Cookr! ID

* Your name: Ian Lloyd
* Preferred user name: ianlloyd — User name already taken
* Password: ******** — Strength: Strong
* Retype password: ******** — Passwords do not match

2.5 **Dealing with error messages**

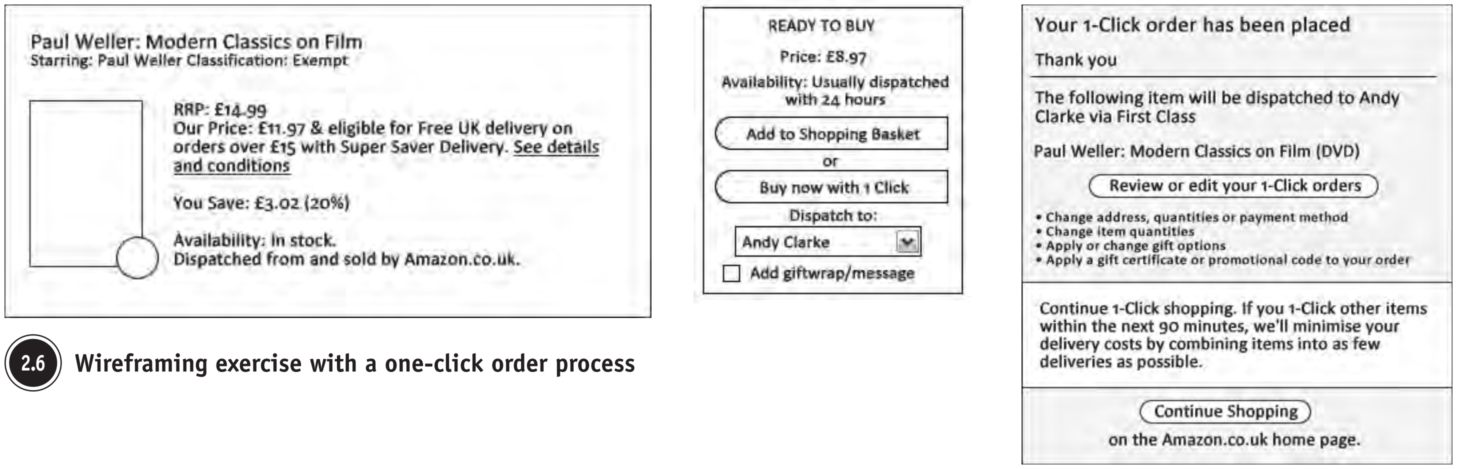
Finally, wireframes rarely convey all the complete information, and a range of supporting material such as site maps, page descriptions, or functional specifications almost always accompanies them. On large-scale projects, writing, reading, and keeping track of this documentation can add to the complexity and costs involved.

Traditional wireframes and interaction

If the simple interaction of a flexible, liquid layout is hard to convey using images, try to imagine how difficult the complex functionality now common on sites using interactive media such as Macromedia Flash and technologies such as Ajax and DOM (Document Object Model) scripting might be. Jeffrey Zeldman may have said it best:

> *Wireframing Ajax is a bitch.*
>
> —Jeffrey Zeldman (www.alistapart.com/articles/web3point0)

When you are designing e-commerce sites similar to Amazon, you can more easily convey the process of adding an item to a shopping cart and proceeding through the checkout process using images (**Figure 2.6**). Even some of Amazon's slickest user features such as 1-Click ordering present no real challenges to an experienced designer.



Paul Weller: Modern Classics on Film
Starring: Paul Weller Classification: Exempt

RRP: £14.99
Our Price: £11.97 & eligible for Free UK delivery on orders over £15 with Super Saver Delivery. See details and conditions

You Save: £3.02 (20%)

Availability: In stock.
Dispatched from and sold by Amazon.co.uk.

READY TO BUY
Price: £8.97
Availability: Usually dispatched with 24 hours

Your 1-Click order has been placed
Thank you

The following item will be dispatched to Andy Clarke via First Class

Paul Weller: Modern Classics on Film (DVD)

- Change address, quantities or payment method
- Change item quantities
- Apply or change gift options
- Apply a gift certificate or promotional code to your order

Continue 1-Click shopping. If you 1-Click other items within the next 90 minutes, we'll minimise your delivery costs by combining items into as few deliveries as possible.

on the Amazon.co.uk home page.

2.6 **Wireframing exercise with a one-click order process**

Note: For more thoughts on prototyping Ajax, read Kevin Hale's article "A Designer's Guide to Prototyping Ajax" at http://particletree.com/features/a-designers-guide-to-prototyping-ajax/.

But in Ajax-driven applications, where users can invoke complex behaviors with simple text input or a mouse click, images are rarely capable of describing sufficient detail on a single screen.

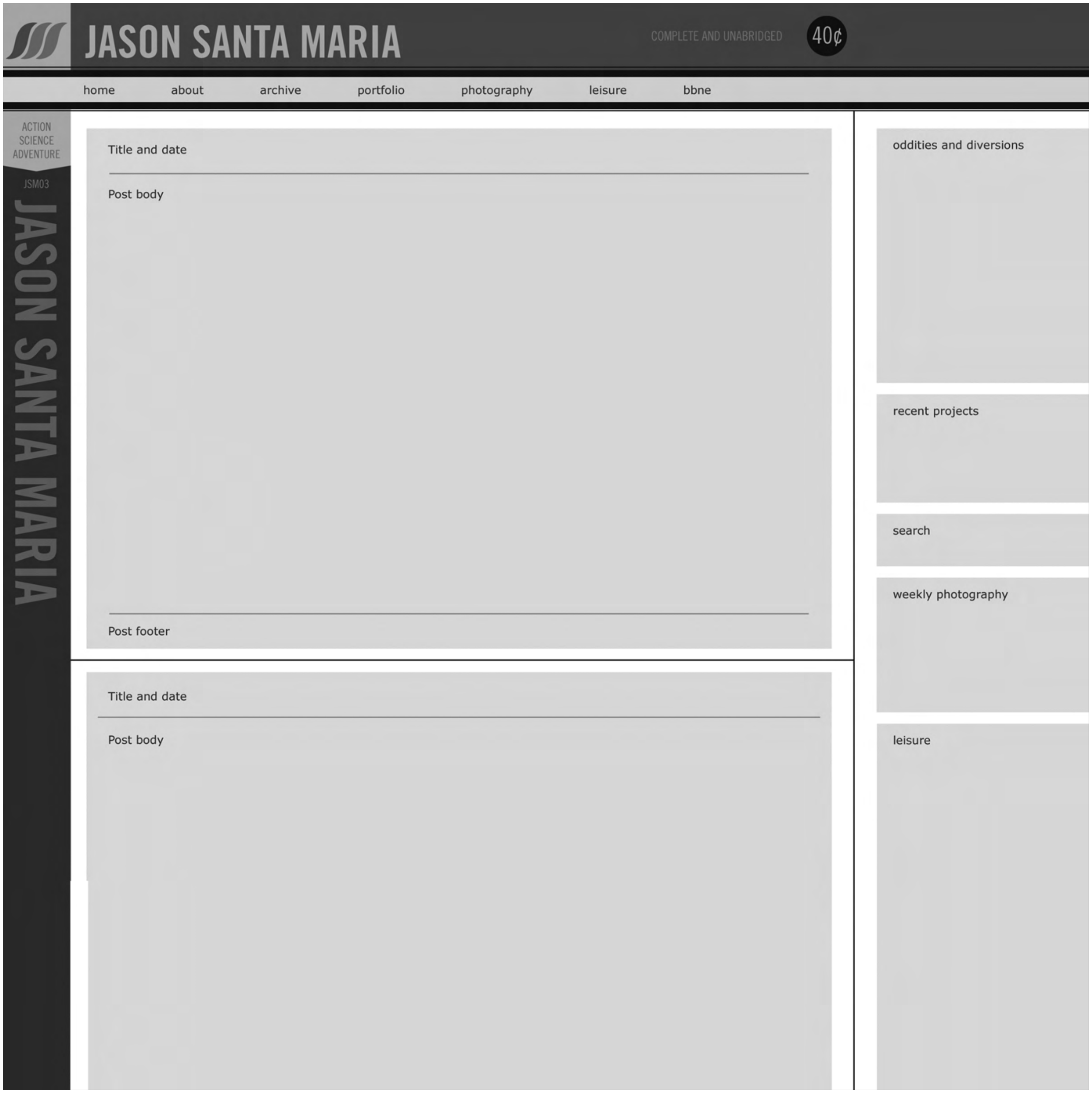
An interesting case in point is Flickr, Yahoo's popular photo storage and sharing application. Flickr is a complex mix of markup, CSS, Flash, and Ajax that gives its visitors an immersive user experience. Some of Flickr's tools, such as adding a tag, editing a title, or changing a description, might cause fewer headaches in wireframing with images.

Flickr's more complex user features for adding a note to a photo or organizing photos into sets through its drag-and-drop interface would prove extremely difficult to represent in a traditional wireframe with even a long series of images.

Proofing functionality of this complexity would be a tough job for even the most patient designer and would involve creating images of every stage of user interaction, clearly a job few designers would relish.

Many variables get overlooked when creating wireframes or other paper documents. Factors such as state, security, error messages, level of effort, page flow, DOM scripting and other dynamic elements can be ignored or misrepresented.

—GARRETT DIMON
www.digitalweb.com/articles/
just_build_it_html_prototyping_and_agile_development


JASON SANTA MARIA
COMPLETE AND UNABRIDGED
40¢
home
about
archive
portfolio
photography
leisure
bbne
ACTION SCIENCE ADVENTURE
JSM03
JASON SANTA MARIA
Title and date
Post body
Post footer
Title and date
Post body
oddities and diversions
recent projects
search
weekly photography
leisure

Improving the Approach with the Grey Box Method

With many people unhappy about the limitations of traditional wireframes, we needed to find a new answer to the question of how to communicate the organization and relationships of content to each other, without being too precise or limiting creative options. One solution has come from the everyday work of a visual designer, Jason Santa Maria.

This solution is based on using a series of simplified grey boxes as the starting point of the graphic design process. When extended to other roles, grey boxes encourage information and content specialists to define only the loose placement of content areas; this leaves designers free to make the important decisions over layout and composition.

The grey box method does the following:

- Allows the design process to be spread over shorter stages
- Places focus only on the *structural* elements of the page design
- Allows designers to work creatively without limiting their options

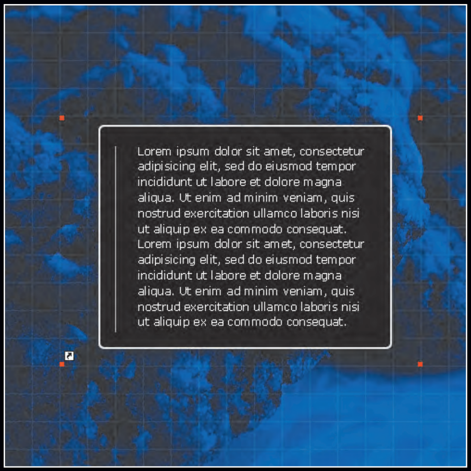
From its simple beginnings, the grey box method is appealing because it allows you to extend its approach to suit many of the situations and environments you may find yourself working in.

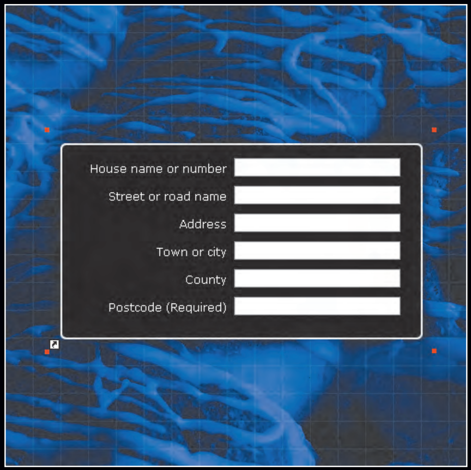
Note: You can read Jason Santa Maria's original "Grey Box Methodology" article at www.jasonsantamaria.com/archive/2004/05/24/grey_box_method.php.

Annotations and page descriptions

When you need more detail than simple grey boxes can deliver, you can add short notes about specific content and relationships. When you need further detailed explanations, you can use accompanying page description documents to express other key issues and goals.

This combination of grey boxes, notes, and descriptions is also highly effective during conversations with clients, because the combination can rarely be misinterpreted for anything other than what it is; in addition, the combination does not set design or layout expectations.


Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.


House name or number

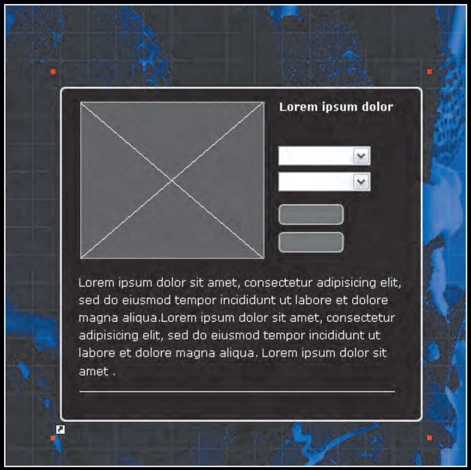
Street or road name

Address

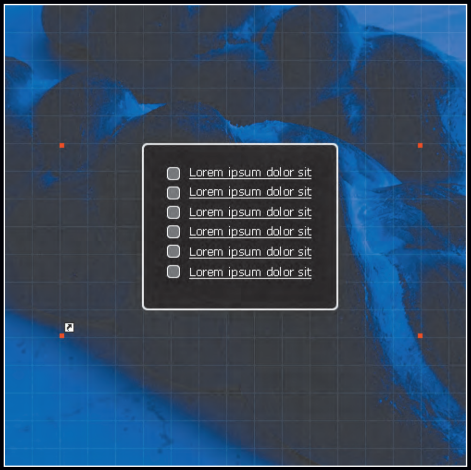
Town or city

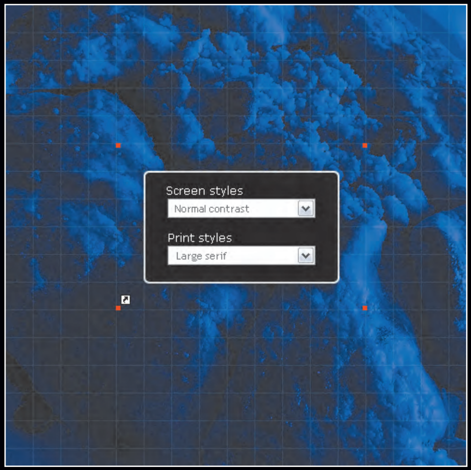
County

Postcode (Required)

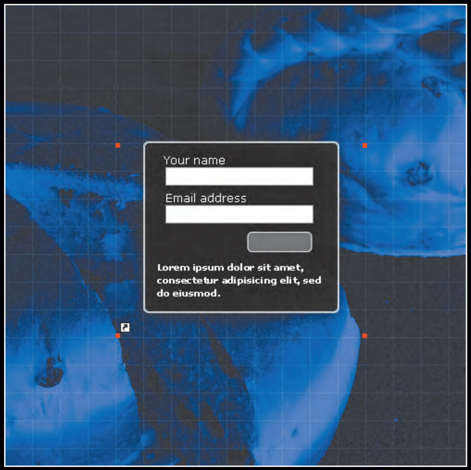

Lorem ipsum dolor

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Lorem ipsum dolor sit amet .


- Lorem ipsum dolor sit
- Lorem ipsum dolor sit
- Lorem ipsum dolor sit
- Lorem ipsum dolor sit
- Lorem ipsum dolor sit
- Lorem ipsum dolor sit

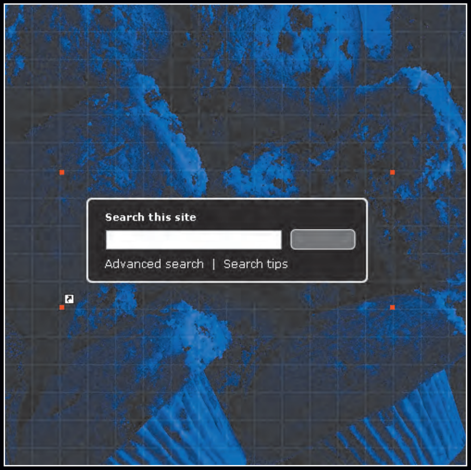

Screen styles: Normal contrast

Print styles: Large serif

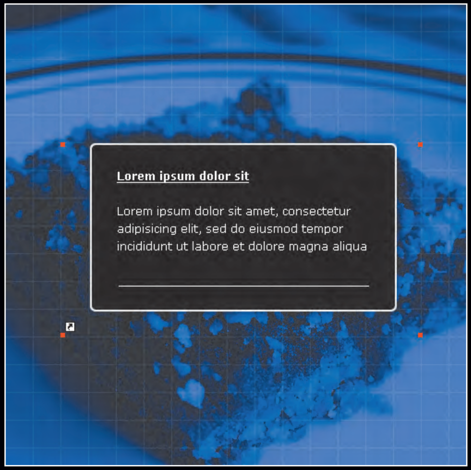

Your name

Email address

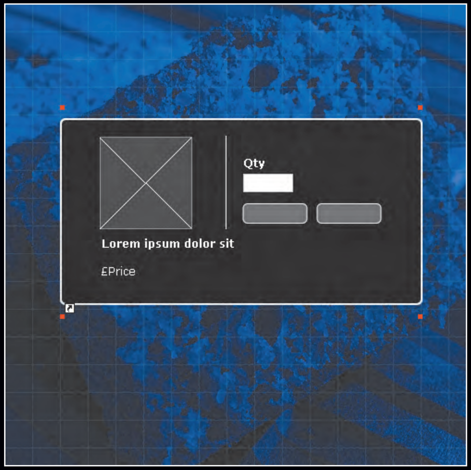
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod.


Search this site

Advanced search | Search tips


Lorem ipsum dolor sit

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua


Qty

Lorem ipsum dolor sit

£Price

Using symbols to add greater detail

Sometimes you will need to provide more detail in a visual way. Developing a library of reusable graphic components, or *symbols*, that provide this extra level of detail and can be dragged and dropped on grey boxes can be highly effective.

For much of my own design and layout work, my preferred layout application is Macromedia Fireworks, not least for its capability to store graphic symbols that I use from project to project in its library. I have made a simple symbol for each of the common user interface elements I use on a regular basis (**Figure 2.7**):

- Grid design variations
- Greeking text
- Common e-commerce interface elements
- Search areas
- Form fieldsets and other form elements

Tip: This technique is also possible in other graphic and layout tools.

You should keep these symbols deliberately minimal and free from any creative flourishes. By dragging and dropping these symbols on grey boxes, you can add more detail to the parts of your wireframes that require it.

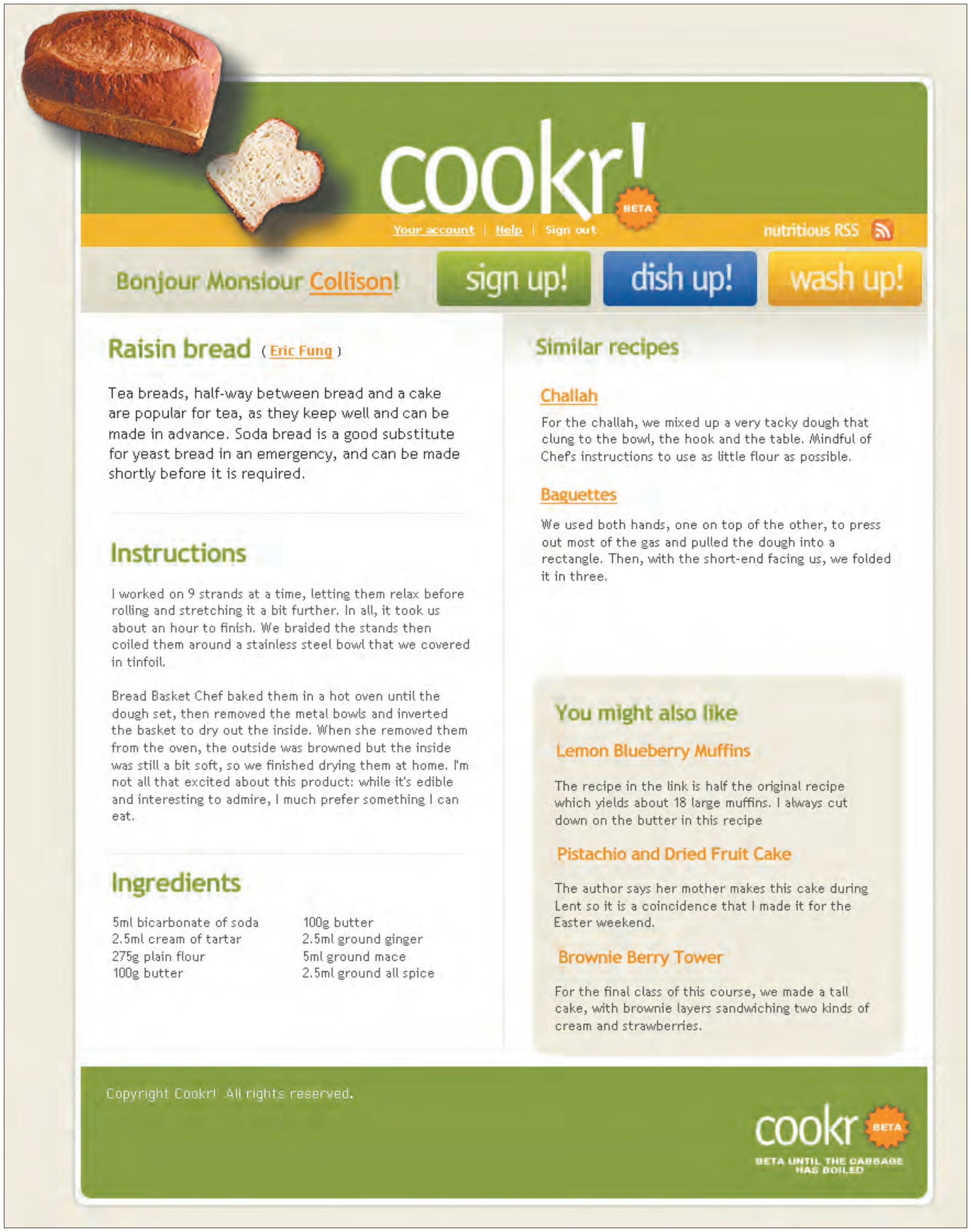
2.7 Several elements created in Fireworks as "symbols"

Tips for working with grey boxes

Grey boxes can help make collaboration between information architects, visual designers, and technical developers better.

When organizing information for Web sites, you can implement the following tips to better relate your work to the work of visual designers and developers while at the same time preserve the meaning of the content relationships you have defined:

- Define the source order of the content.
- Think about how that content will look without style; will it be easy to read, and will the order make sense?
- Visually highlight any relationships: This will ultimately help identify necessary divisions and show areas of similar but not related information.
- Use established naming conventions



Raisin bread (Eric Fung)

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

Bread Basket Chef baked them in a hot oven until the dough set, then removed the metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

Ingredients

5ml bicarbonate of soda	100g butter
2.5ml cream of tartar	2.5ml ground ginger
275g plain flour	5ml ground mace
100g butter	2.5ml ground all spice

Similar recipes

Challah

For the challah, we mixed up a very tacky dough that clung to the bowl, the hook and the table. Mindful of Chef's instructions to use as little flour as possible.

Baguettes

We used both hands, one on top of the other, to press out most of the gas and pulled the dough into a rectangle. Then, with the short-end facing us, we folded it in three.

You might also like

Lemon Blueberry Muffins

The recipe in the link is half the original recipe which yields about 18 large muffins. I always cut down on the butter in this recipe

Pistachio and Dried Fruit Cake

The author says her mother makes this cake during Lent so it is a coincidence that I made it for the Easter weekend.

Brownie Berry Tower

For the final class of this course, we made a tall cake, with brownie layers sandwiching two kinds of cream and strawberries.

Copyright Cookr! All rights reserved.

2.8 Creating a static design

Creating Static Designs

Static designs are important in conveying ideas about look and feel, page layouts, and interface designs to clients and other stakeholders (**Figure 2.8**). Although static designs should indicate how the finished product will look, they are also mistakenly used as benchmarks, leaving little room for later flexibility or even behaviors such as liquid page layouts.

Many designers have asked when static design visuals, or *comps* as they're often referred to, fit best in a contemporary Web design workflow. Should they come at the beginning? Should they come before making wireframes or grey boxes, or should they come later? The answer will depend on your own working environment.

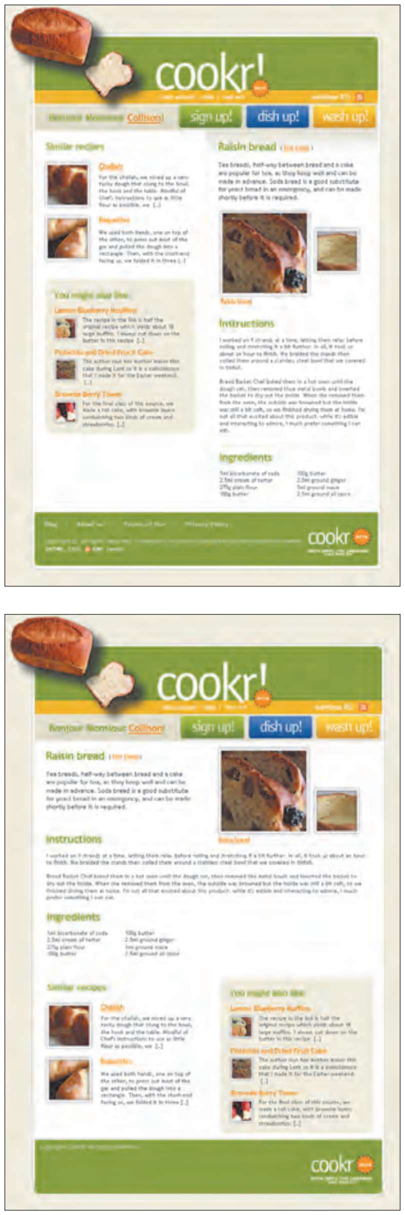
To answer this question, you should ask yourself, what purpose are these static designs going to accomplish? If they are designed to illustrate only general shape and style, you can work on them throughout the process and refine them right up until the end because the specific details of the layout may not depend on them. However, if static designs are intended to show the specifics of layout details, they will likely be completed and signed off on far earlier in the process.

Moving faster through the design workflow

In the past, finalizing and signing off on static designs has always preceded other stages in the design and development process. Writing markup and CSS waited while static designs were completed, and it was even less likely that expensive activities such as scripting or programming would start until later.

One of the most important advantages of the content-based workflow is that these other areas of work do not have to wait until static designs are complete. Grey boxes and other supporting materials can give not only visual designers but also technical developers and others all the information they need, giving them a head start to begin their work from the same basis of content as designers.

Even when markup has been written and perhaps more complex functionality added to a prototype, visual designers can continue to experiment with design ideas or even different layouts, safe in the knowledge that CSS will allow them to implement their designs without breaking the work of other people in their team (**Figure 2.9**).



2.9 **Giving variety to layout**

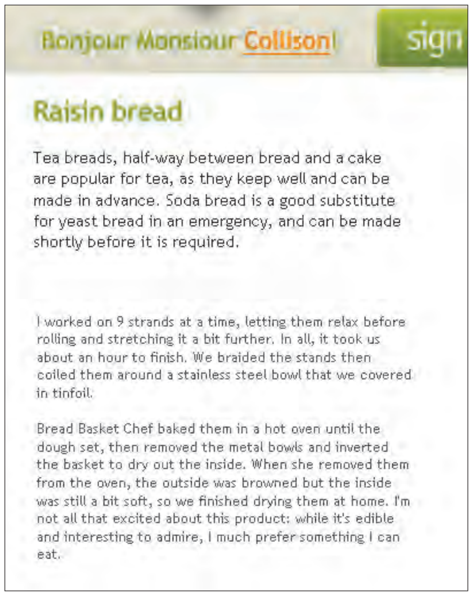
Adding markup guides to static designs

Markup guides are designed to show the simple outline of the markup that is most appropriate to convey the meaning of any element during the design process. They do this directly on the static design and can be used to do the following:

- Show the hierarchy or structure of the content.
- Help advise technical developers of the most appropriate markup to use.

Imagine for a moment that a visual designer shows two blocks of text. The first is plainly styled; the second is styled bolder and in larger type. One problem that technical developers often face is how to interpret the visual design as markup when the precise meaning of an element is not obvious through the visual design. For example, in **Figure 2.10**, can you tell that the second block of text is a quotation rather than a normal paragraph?

Whether you choose to work on paper by printing your grey boxes and writing your markup and any accompanying notes onto that or to work electronically by adding the guides directly to your grey box files, using markup guides will help everyone involved in the process (**Figure 2.11**).



Raisin bread

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

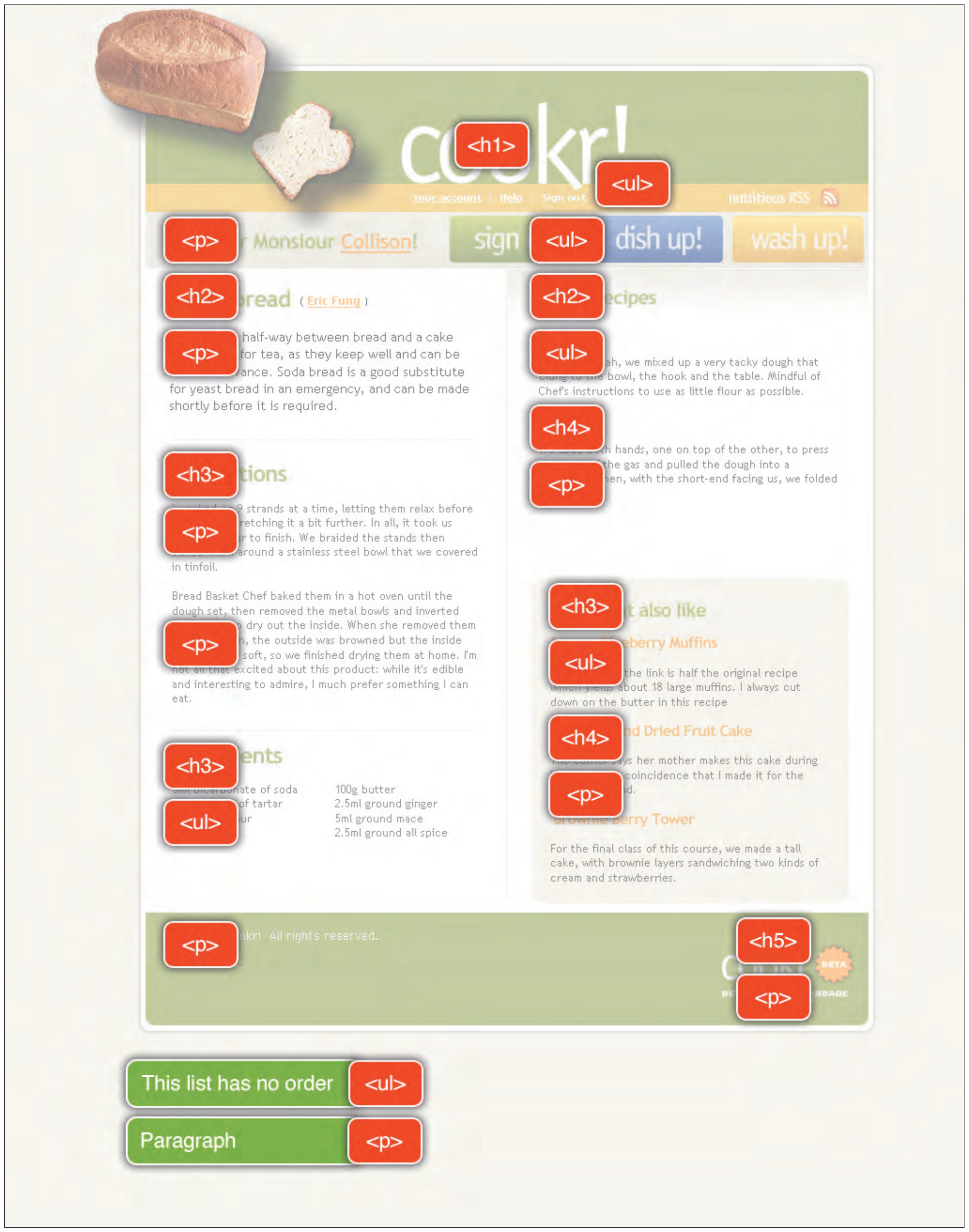
Bread Basket Chef baked them in a hot oven until the dough set, then removed the metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

2.10 Highlighting the difficulties when precise meaning is not obvious

Markup guides as training aids

Some information architects and designers who are new to the concepts of semantics and writing meaningful rather than presentational markup find it difficult to relate a visual layout to XHTML.

Markup guides help them visually associate markup to design. Working regularly with these guides can help demystify markup and the XHTML elements that are commonly used.



2.11 **Laying markup over a static design**

Using Interactive Prototypes

It's easy to understand why many devices of the early planning stages—content site maps, flow diagrams, and wireframes—use static images, but the Web is an interactive medium. Whereas wireframes and static designs can only hint at the interactivity of a finished Web site, interactive prototypes can do so much more. To make the most of the medium, we should work *using* the medium.

By creating interactive prototypes using valid, meaningful markup and CSS, designers can use the prototypes to demonstrate their designs, and developers can easily add more functionality with Ajax and related technologies to create a fully working prototype.

You might at first think that making prototypes using hand-coded XHTML and CSS would take more time than creating images, particularly if you have experience with your wireframing application or have a library of reusable symbols you use by dragging and dropping.

However, using XHTML and CSS does allow you to work faster:

- You can use one or multiple CSS files to lay out your prototypes.
- You can use CSS styles for layout, color, and typography across any number of pages.
- You can make rapid changes without changing your markup.
- You can preview multiple design variations using the same content.

These interactive prototypes also enable visual designers to make faster and more frequent design iterations, try new ideas, and rearrange layouts, all without altering the structure or order of the content.

Interactive prototypes make it real

When you demonstrate your designs in a Web browser, you allow your clients to interact with them in a more meaningful way. Rather than them imagining how a feature might appear from looking at an image, they can interact with it directly, even though it may not be fully functional, reducing the opportunities for misunderstandings.

If you are following modern Web standards practices, these pages are probably built with XHTML for structure and CSS for markup. XHTML is an excellent structure that can serve as the basis for a wireframe that can later be transitioned into a prototype and eventually designed via CSS.

—NICK FINCK
www.blueflavor.com/ed/information_architecture/
recyclable_information_archite.php

Clients become engaged when they can interact with HTML wireframes. Clients not only enjoy the process more, but they also get a better contextual understanding of the features than with paper prototypes.

—Jeff Gothelf, Boxes and Arrows (www.boxesandarrows.com/view/practical_applications_visio_or_html_for_wireframes)

Interactive prototypes are powerful tools for presenting your designs to your clients. When clients provide feedback, you can implement suggestions immediately; if the changes don't work, your clients will see the results right away, and you can easily roll back to a previous iteration. Working in this way rapidly speeds up the process of gaining client approval for your designs, even when working with clients spread across different continents and time zones.

Creating reusable code

When you develop using meaningful markup and CSS, you can reuse much of your work. This will save you significant amounts of development time because your work is far less likely to need duplicating. When you follow the same strict coding practices as you would when making your final product, your work during prototyping will not be thrown away.

If you approach your work with the goal of keeping and reusing it, then most everything you do will survive to the end. By adopting this method, you will almost always increase your speed and reduce costs for your studio, organization, managers, and clients.

Model behavior for wireframes and prototypes

With new ways of improving communication between all of those working on organizing content, it is time to learn best practices and learn how to use a Web browser and a range of extensions for organizing your style sheets efficiently.

WYSIWYG: What you see, or short-sighted?

The WYSIWYG design environment in applications such as Macromedia Dreamweaver, with its built-in templates and drag-and-drop library items, have already played a large part in the transition to using HTML for prototyping. These tools have made it far easier for people to make HTML prototypes without a wide knowledge of markup, CSS, or best practices.

With WYSIWYG editors being designed to create the markup for you, you might at first think that hand-coding XHTML and CSS would take more time and be less efficient than using a WYSIWYG editor, particularly if you have experience with your wireframing application or have a library of reusable assets that you use by dragging and dropping. But that isn't necessarily the case.

The following are some advantages of using XHTML and CSS prototypes over those created with WYSIWYG editors:

WYSIWYG	**Standards-based markup and CSS**
Requires you to buy a WYSIWYG application such as Dreamweaver or Adobe GoLive.	Requires only a basic Web editor or plain-text editor such as Notepad for Windows or TextEdit for Mac OS X.
Requires you to be experienced in an application that will rarely be useful in developing the final pages.	Requires you to have only a basic knowledge of markup and CSS.
Your markup is likely to be presentational and more likely to be difficult to maintain and reuse.	Your markup will be structured, well ordered, and meaningful. You will reuse much of your markup and CSS.
Changes to your visual layout will often require you to change markup and source order across many pages.	Editing linked or imported CSS files can update any number of pages from only one file.

In the same way an author makes an outline before writing a book, [the grey box method] serves as my visual outline before creating a design. Breaking this into steps makes you consider your design choices and foundation before you are swept away by the details of your visual decisions.

—JASON SANTA MARIA
www.jasonsantamaria.com/archive/2004/05/24/grey_box_method.php

Following Best Practices for Interactive Prototyping

It is important to understand that even though I present the steps in a linear order, the process is not. The content-based process is not a set of hard-and-fast guidelines but a series of steps to improve your workflow, whether you are a lone designer with multiple roles or you work as part of a larger team.

Choosing a development browser

When developing, testing, and demonstrating your interactive prototypes, you should avoid the quirks and issues of older browsers. Having a stable browser platform to act as your development environment and sticking with it throughout the workflow process is essential.

It is important to let others know about your choice so they understand that if they look at your work in any different browser, the results may not look the same.

Your choice of browser will depend on several factors. If you are designing for an internal company environment where the majority of people reading your pages use Microsoft Internet Explorer 5 on Windows 2000, Explorer 5 might, sadly, be the most logical choice because even after several years this browser still ships as part of that operating system. It might also be appropriate to choose Safari if your visitors largely use that browser. Using a browser that has strong support for CSS and a range of development tools available to work with it will make the job of developing and testing that much easier.

Using browser extensions

Although Internet Explorer 7 has its own tools and a developer toolbar, Mozilla Firefox is the development browser most standards-aware designers will choose because of the sheer quantity and quality of its developer extensions. The Mozilla Web site currently contains more than 190 developer extensions for Firefox, with more being added almost daily.

You will be using browser extensions throughout the exercises in this book. Two of my favorite extensions for Firefox are Chris Pederick's Web Developer extension and Firebug.

Note: You can find the developer extensions for Firefox at http://addons.mozilla.org/.

Use the Web Developer extension

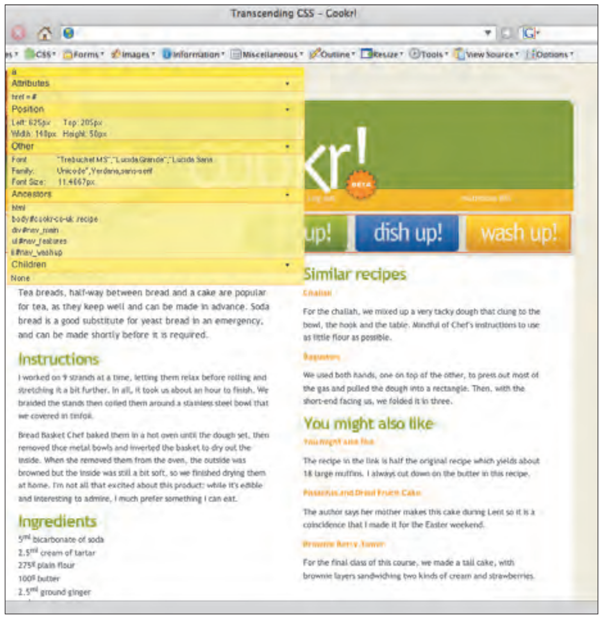
The most essential developer extension is Chris Pederick's Web Developer extension for Firefox and other Mozilla browsers. Pederick's browser toolbar extension includes so many useful features that an entire book could be written (and perhaps will be) on how to use it (**Figure 2.12**).

Note: Download the free Web Developer extension at http://chrispederick.com/work/webdeveloper/.

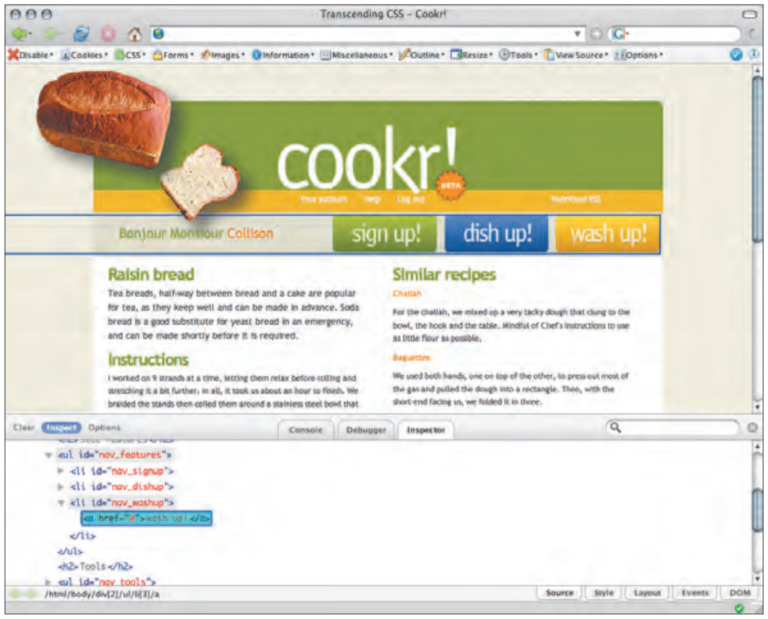
Explore the DOM with Firebug

Firebug is a useful Firefox extension that makes it easy to explore the DOM. Firebug then logs JavaScript, CSS, and other errors to a console. You can also use your keyboard to move through the DOM, and any node you select will be highlighted in the page (**Figure 2.13**).

Note: You can find Firebug at http://addons.mozilla.org/firefox/1843/.



2.12 **The Show Element Information feature in the Web Developer extension**



```
<ul id="nav_features">
  <li id="nav_signup">
  <li id="nav_dishup">
  <li id="nav_washup">
    <a href="#">wash up!</a>
  </li>
</ul>
<h2>Tools</h2>
<ul id="nav_tools">
/html/body/div[2]/ul/li[3]/a
```

2.13 **Using Firebug to explore the DOM**

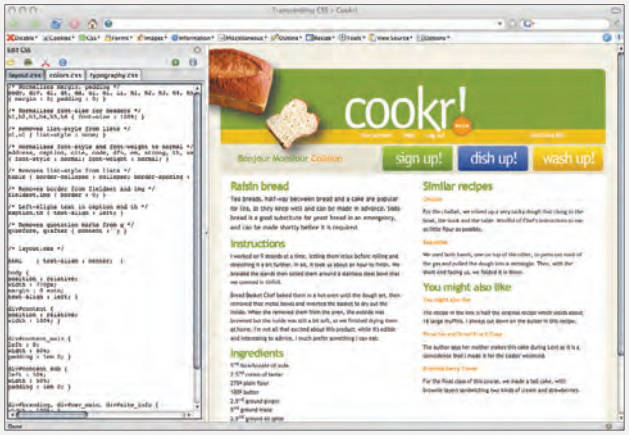
Live editing your CSS with the Web Developer extension

Some of the powerful tools within the Web Developer extension are the controls for the specific CSS files and styles that are loaded into the browser. These tools make it simple for you to change the look of your prototype page without directly editing your CSS files in an external editor, and this process is particularly effective when you want immediate feedback.

One of the most useful features of the Web Developer extension is the Edit CSS panel, which allows you to change styles and then preview the results without ever leaving your browser.

Imagine for a moment that your client or manager has had feedback from user tests. The tests tell you that the default type size you originally chose is too small to meet their needs. This is not an unusual comment, so don't take it to heart; many designers love text so small that it can leave older visitors with their noses pressed against the screen.

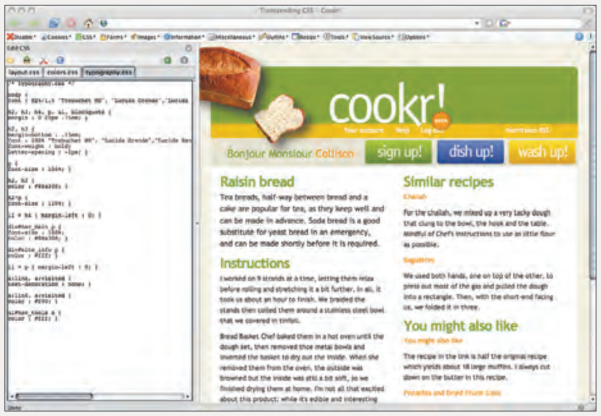
Select Edit CSS from the toolbar menu, and a panel will appear with all the styling information from your inline styles and external style sheets, all organized neatly into tabs.



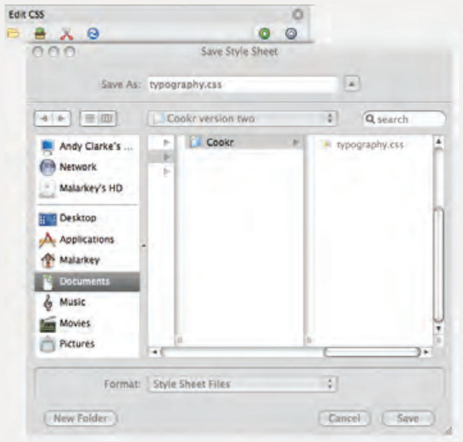
To change your base font size, select the correct tab (here typography.css), and increase the percentage text size that you defined on the <body> element:

```
body {
font : 82%/1.5 "Trebuchet MS", "Lucida Grande","Lucida
Sans Unicode", Verdana, sans-serif; }
```

The results will appear immediately in the browser.



When any changes have been approved, use the Edit CSS panel to save the new font size to your external style sheet.



Working directly in your development browser can save you considerable time. You can try, test, and (if successful) save even large-scale changes to layout, all without ever reaching for your favorite Web editor.

Tip

The HTML Validator extension for Firefox and the Safari Tidy plug-in for Safari will nag you by showing warning icons in your browser's status bar when your pages contain errors. Validation is a great learning tool as well. Be forewarned! No matter how well your site might conform, many content management systems and ad server services will add invalid code. Many designers and developers have no control over this. Still, using validators is an important part of the process that should be built into your workflow.

Keeping your <div> elements to a minimum

Adding more `<div>` elements than necessary will make the likelihood of errors far greater. To avoid this problem during the markup phase, you should keep your markup as minimal as possible.

Start with only structural elements such as headers, paragraphs, lists, and quotations, and work from the content out, before adding any divisions.

Then, keep your `<div>` elements to a minimum, adding them progressively as needed but not more than you need. This approach will help you keep your working documents and the final prototype as free from presentational or unnecessary markup as possible.

Ensuring your markup stays valid

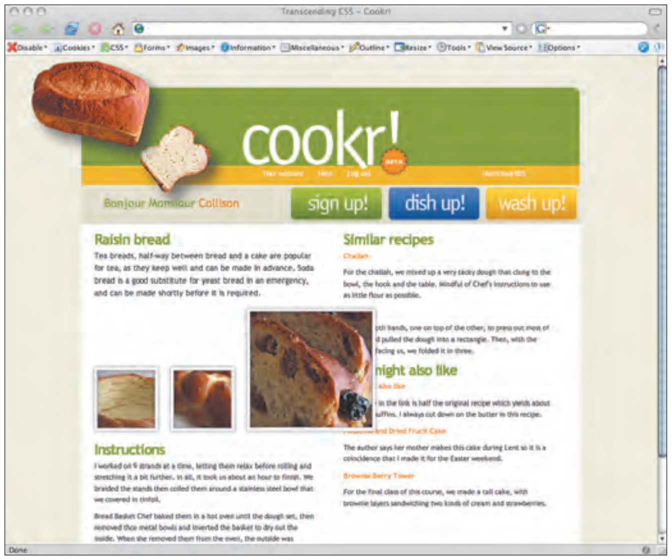
Poorly written markup will always eat into your valuable time by forcing you to first find errors and then correct them. Testing your markup regularly to ensure that no validation errors have slipped in is not only wise, but it can be essential to working efficiently. Writing valid markup will pay dividends; it will reduce any margin for error and give you a firm foundation on which to build your designs.

Choosing positioning over floats

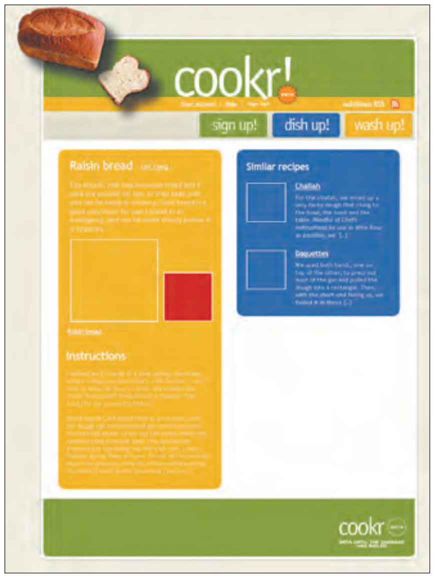
Using floats has become almost a de facto standard method for creating column layouts using CSS. Floats were originally not intended for page layout, but they do their best of a tough job. However, they have a fragility that can often lead to frustration during this phase because sometimes all it takes for float-dependent layouts to fall apart is the addition of italicized text or an image that is a single pixel wider than the floated column that contains it.

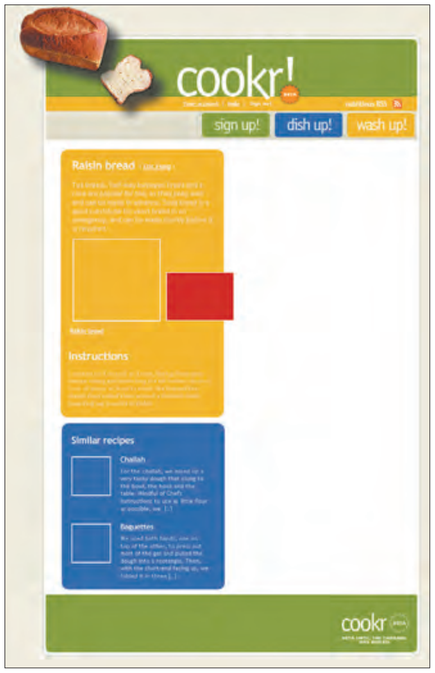
The solution lies in CSS positioning. Understanding the basics of absolute, relative, and fixed positioning can sometimes be more difficult than understanding the relative simplicity of floats. However, mastering positioning with its enormous potential for layout flexibility and its more robust behavior will be one of the most rewarding challenges you can take on when learning CSS.

Whereas float-dependant layouts can easily fall apart at the slightest nudge, positioned layouts can support supersized images or gigantic text without failing, making them ideal



2.14 **Sticking out of the side of a parent element**





2.15 **A floated element dropping down**

to use during this phase. Although you may later choose to rework the CSS to use floats and some positioning, the generally accepted current recommendation is to use positioning in this phase and make changes later.

The problem with floats

Contemporary Web browsers follow the W3C (World Wide Web Consortium) specification in which a 200-pixel-wide element, such as a division, should always be 200 pixels wide. Wider elements placed inside it would simply stick out of the side; the result might look ugly, but the layout will remain intact (**Figure 2.14**).

The developers of early versions of Internet Explorer had other ideas. They developed their browser to expand a container to fit the width of its contents. So, the same 200-pixel-wide column containing a 220-pixel-wide image would expand to 220 pixels. This can cause a floated column to drop underneath its neighbor column, breaking your layout (**Figure 2.15**).

Although you can solve problems with floats when you have more time, time is often in short supply when you are making your prototypes.

PNG alpha transparency in the workflow

During the static design phase, the background colors of your columns and other divisions are likely to change more than once. Few jobs can be more frustrating than creating and exporting images several times over to ensure that their background colors match that of the CSS `background-color` value.

2.16 **PNG versus GIF transparency**

If you are working with a browser with PNG alpha transparency support as your development environment, using PNG images in place of transparent GIF images can save you many hours of frustrating work (**Figure 2.16**).

PNG images make it possible for you to export a single set of images with alpha-transparent backgrounds and then change the background colors in your style sheets as many times as you need, without ever returning to your image editor.

Organizing your CSS

So many different ways to organize and comment on your CSS files exist that if a group of CSS designers were to meet for a drink in pub, a discussion over which method is best would no doubt last well beyond closing time.

Note: The Mozilla.org Markup Reference is a wonderful example of how to document standards for authoring markup and CSS (www.mozilla.org/contribute/writing/markup).

As your style sheets become longer and more complex, organizing your styles into a clearly understandable structure is essential. Organizing styles helps you write more efficient CSS and ensures that others can easily understand and edit your CSS documents. Of course, every designer will have a preferred method.

Organizing by location

Some organize their rules by divisions, with all the `#branding` rules in one group and all the `#content` rules in another:

```
/* =content_main */
div#content_main { width : 70%; }
div#content_main p { font-size : 100%; }
div#content_main p > a { text-decoration : underline; }
```

```
/* =content_sub */
div#content_sub { width : 30%; }
div#content_sub p { color : #666; }
div#content_sub p > strong { font-weight : normal; }
```

Marking sections in your CSS

Making sections easily distinguishable from each other by using a combination of CSS comments, section markers, and dashes as separators is one method to help you and later developers find certain rules and know which rule applies to which part of your design:

```
/* Main content
----------------------------------------------- */
```

This method of highlighting the start of each section can save you time when trouble-shooting or returning to a project several months after first writing your CSS.

Organizing by element

Others prefer to organize their rules by element, grouping all the headings, paragraphs, and lists together:

```
/* p */
p { line-height : 110%; }
blockquote p { padding-left : 1em; }
div#site_info p { text-align : center }

/* ul */
ul { list-style-type : disc; }
div#nav_main ul { list-style-type : none; }
div#content_sub ul { border : 1px solid #ccc; }
```

CSS flags

Adding a simple flag, perhaps the `=` character immediately before your commented section marker text, can help finding and jumping to that section much easier:

```
/* =p */
```

Using your text editor's Find command to find `=p` will take you straight to your section for paragraphs and will ignore erroneous results such as `list-style-type` or `padding`.

Dividing your CSS into multiple files

Whereas people might disagree over whether a single, linked, or imported CSS file is more manageable in a final product than many separate ones, you can best solve that argument by studying the context of the situation. One fact is certain, however; while building the interactive prototype, using multiple files has distinct advantages.

For example, you could break a prototype into the following separate files:

- Layout styles including display properties, floats and positioning, widths and heights, and padding and margins (layout.css)
- Color styles including background properties, colors, and images, as well as text colors (color.css)
- Typographical information including font families and sizes, line heights, letter spacing, and text decorations (type.css)

For simplicity and to reduce the number of style sheets that are linked to and from your markup, you might choose to link to one file and import your additional style sheets into that using the `@import` at-rule.

To work correctly, your imported style sheets must appear at the top of the style sheet above any other rules:

```
@import url(color.css);
@import url(type.css);
[ remaining layout.css rules ]
```

So far, you have learned that using meaningful markup and CSS to make interactive prototypes will help you achieve lightweight, semantic code; accessible content; and flexible design. This kind of prototype also helps you communicate more efficiently with your colleagues and clients.

HTML prototyping and full-on agile development of Web applications are increasingly viable options that help minimize communication gaps and assumptions and deliver more accurate results sooner. If you haven't considered it, now may be the time.

—GARRETT DIMON
www.digital-web.com/articles/
just_build_it_html_prototyping_and_agile_development/

Practicing the Process

It's time for you put into practice the workflow and techniques you have learned. In this section, you will move through the stages of the content-based workflow with the goal of creating an interactive prototype from one of three static design visuals using meaningful markup and CSS.

Looking at the ingredients

Let's imagine for a moment that you have the pleasure of working on a new design for a start-up company. This company has shiny new offices and enough venture capital to run a small country with change to spare. This new company, Cookr!, is building an exciting Web application that will enable its visitors to upload and share recipes on the Web.

The first stage in the process is for you to gather all the content and organize it, taking care to be sensitive to search engine, usability, and accessibility concerns.

Opening the grey boxes

Here the grey box method is ideal for describing the content that will appear on any given page and the relationships inherent between areas of that content. Grey boxes are easy to create, and they give visual designers just the right amount of information without limiting any of the ideas they might have for how the page layout should look.

For this simple page, grey boxes are representing two primary areas (**Figure 2.17**):

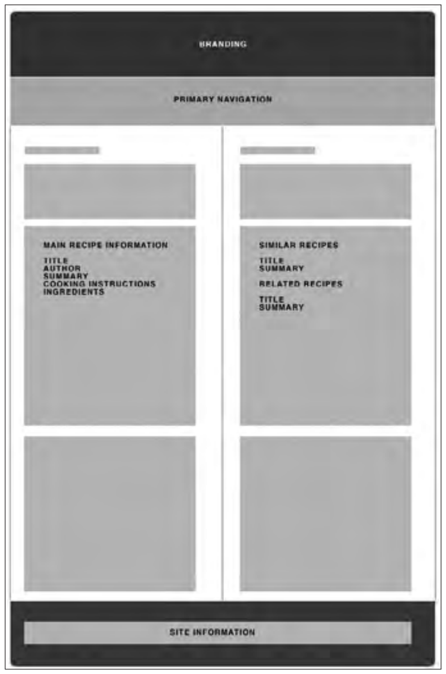
- Content of interest
- Navigation and tools

Content and navigation have been further divided into the following:

- Content includes an individual recipe's main content, including its description, ingredients, and cooking instructions.
- Navigation includes links to account features and tools that help visitors use the site.

To provide more detailed information, these grey boxes could be part of a larger set of documentation that might include a page description document or other notes about content and functionality.

2.17 Representing the Cookr! prototype with a grey box



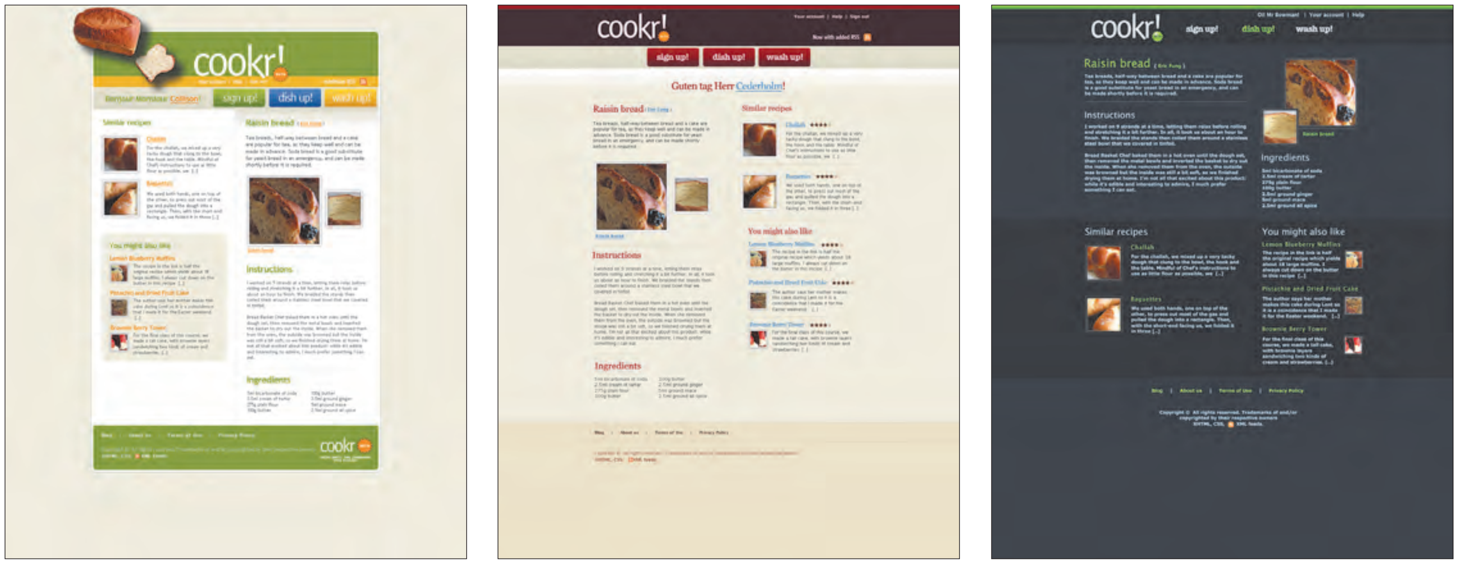
Looking at the static design visuals

Creating static designs will likely always be part of the creative design process. For this project, assume you have created three static designs. The client has initially chosen one that best reflects their branding and the emotions they want the site to evoke (**Figure 2.18**).

Writing content-out markup

Whatever role you play in the process, work can begin now on creating a meaningful XHTML document. You will build this document from the content that was described and organized into grey boxes. You can work on this markup even while the intricacies of the visual design are still being finalized. This gives everybody involved in the project a head start.

Although it might be tempting to use the visual layout as the basis for the structure of your markup, this could lead to you overusing elements, particularly `<div>` elements. This will also result in establishing your content order primarily to accomplish the visual layout rather than it making sense when no style sheets are available. To avoid presentational markup and ordering problems, begin by first looking at the content and then working out from the meaning.



2.18 **Giving three static designs for the Cookr! project**

Visualizing the structure

If you refer to the grey boxes and static design visuals, you will see that the page you are prototyping has several important areas of content:

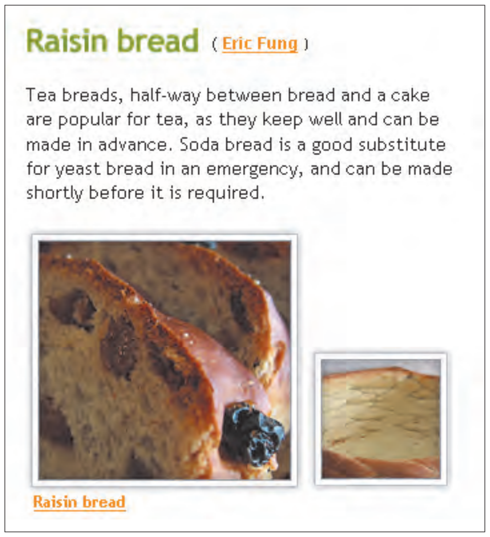
- A main recipe that contains information about how to make it, plus its ingredients
- Similar and alternative recipes

The static design contains expected site features including branding, navigation, and site information that are unrelated to the main content. I often refer to these features as *site furniture*.

Main content

We already know that the name of the site, Cookr!, is important for identifying the site. It's so important, in fact, that it is appropriate to use the top-level heading <h1> to describe it in markup.

Now turn your attention to the main recipe information, which includes its name followed by a short description (**Figure 2.19**).


Raisin bread (Eric Fung)

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

2.19 Getting a close-up look at the main recipe information

Note

Some developers prefer to use a top-level heading, on the home page only, for the name of the site. On other pages, they use an <h1> element for the page's unique title and not for the name of the site.

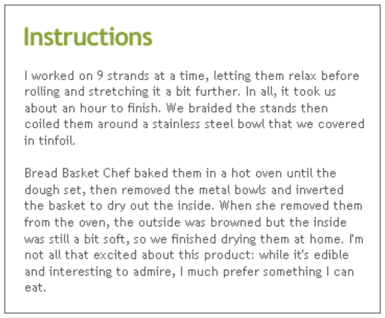
Looking at this content, two elements should spring to mind:

- A heading for the recipe name, in this case a second-level heading, `<h2>`
- A paragraph, `<p>`, for the recipe description

Here are the elements:

```
<h2>Raisin bread</h2>

<p>Tea breads, halfway between bread and a cake, are popular for tea, as they
keep well and can be made in advance. Soda bread is a good substitute for
yeast bread in an emergency and can be made shortly before it is required.</p>
```

You should see a similar pattern to the structure of the cooking instructions, but with one subtle difference. If you refer to the grey boxes, you will see that these instructions and ingredients are part of the main recipe content marked up with an `<h2>` element. To maintain a well-structured outline, you should choose lower-level headings for all the content that falls under that heading (**Figure 2.20**).



Instructions

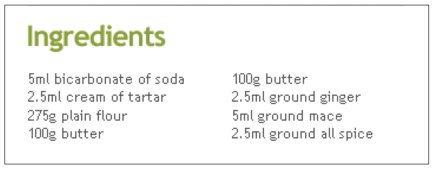
I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

Bread Basket Chef baked them in a hot oven until the dough set, then removed the metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

2.20 **Close-up of the instructions**

```
<h3>Instructions</h3>

<p>I worked on 9 strands at a time, letting them relax before rolling and
stretching them a bit further. In all, it took us about an hour to finish. We
braided the strands and then coiled them around a stainless steel bowl that we
covered in tinfoil.</p>

<p>Bread Basket Chef baked them in a hot oven until the dough set, then
removed the metal bowls, and inverted the basket to dry out the inside. When
she removed them from the oven, the outside was browned but the inside was
still a bit soft, so we finished drying them at home. I'm not all that excited
about this product: while it's edible and interesting to admire, I much prefer
something I can eat.</p>
```

Finally, the main recipe content contains a list of ingredients, in no particular order. Here you should use an unordered list, rather than any other type, to provide the structure (**Figure 2.21**).



Ingredients

5ml bicarbonate of soda	100g butter
2.5ml cream of tartar	2.5ml ground ginger
275g plain flour	5ml ground mace
100g butter	2.5ml ground all spice

2.21 **Close-up of the ingredients**

```
<h3>Ingredients</h3>
```

```
<ul>
<li>5<sup title="Mililitres">ml</sup> bicarbonate of soda</li>
<li>2.5<sup>ml</sup> cream of tartar</li>
<li>275<sup title="Grammes">g</sup> plain flour</li>
<li>100<sup>g</sup> butter</li>
<li>2.5<sup>ml</sup> ground ginger</li>
<li>5<sup>ml</sup> ground mace</li>
<li>2.5<sup>ml</sup> ground all spice</li>
</ul>
```

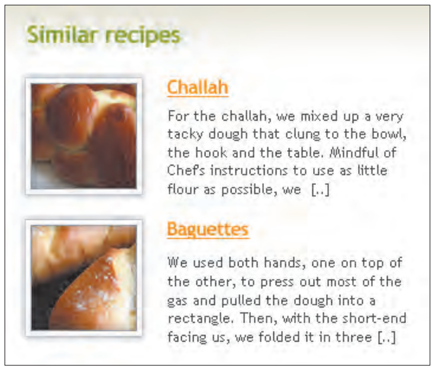
You might have noticed that at no stage have I been asking any questions about how this content is going to look, concentrating instead on the content's meaning and the elements most appropriate to describe it.

Secondary content

Now turn your attention to the similar recipes area. They appear on the right of the static design, but you should not be concerned about that at this stage because it does not matter where this content will be positioned visually.

The similar recipes links give visitors an opportunity to look at related recipes should the main recipe not quite be what they need. Here you can see two alternative recipes listed, although many more could be listed (**Figure 2.22**). The heading is simple enough to deal with; in this case I have chosen to use a third-level heading, `<h3>`.

```
<h3>Similar recipes</h3>
```



Similar recipes

Challah

For the challah, we mixed up a very tacky dough that clung to the bowl, the hook and the table. Mindful of Chef's instructions to use as little flour as possible, we [..]

Baguettes

We used both hands, one on top of the other, to press out most of the gas and pulled the dough into a rectangle. Then, with the short-end facing us, we folded it in three [..]

2.22 **Close-up of similar recipe links**

Tip

Both the HTML Validator extension for Firefox and the Safari Tidy plug-in for Safari will nag you by showing warning icons in the status bar of the browser when your pages contain errors.

What about the alternative recipes information that follows this heading?

```
Challah

For the challah, we mixed up a very tacky dough that clung to the bowl, the
hook, and the table. Mindful of Chef's instructions to use as little flour as
possible.

Baguettes

We used both hands, one on top of the other, to press out most of the gas and
pulled the dough into a rectangle. Then, with the short end facing us, we
folded it in three.
```

You would be correct if you thought this content suggests that headings and paragraphs would be appropriate, but you would also be overlooking that these recipes are part of a series, and as such they form a list. But what type of list?

The titles and accompanying text are not strictly definition terms and descriptions, so using a definition list would be stretching the semantic use of the `<dl>` element. Because the list has no order, it is appropriate to use an unordered list with list items that contain each heading and a paragraph. This forms a meaningful XHTML compound:

```
<ul>
<li>
<h4><a href="#">Challah</a></h4>
<p>For the challah, we mixed up a very tacky dough that clung to the bowl, the
hook, and the table. Mindful of Chef's instructions to use as little flour as
possible.</p>
</li>

<li>
<h4><a href="#">Baguettes</a></h4>
<p>We used both hands, one on top of the other, to press out most of the gas
and pulled the dough into a rectangle. Then, with the short end facing us, we
folded it in three.</p>
</li>
</ul>
```

Note: XHTML *compounds* are combinations of two or more elements in XHTML that each have their own meaning. When combined, the elements create a more precise meaning together than they do separately. The concept of XHTML compounds emerged from the microformats community rather than the W3C. You can learn more about XHTML compounds in "The Elements of Meaningful XHTML" by Tantek Çelik at www.tantek.com/presentations/2005/09/elements-of-xhtml/.

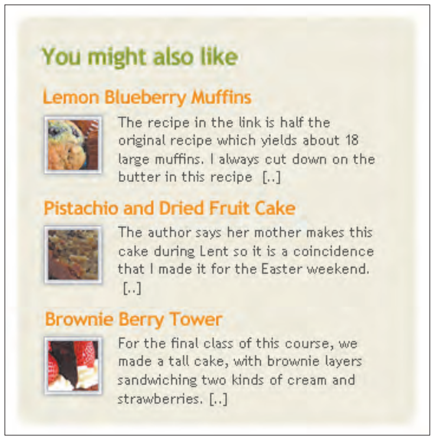
Finally, you can take an identical approach for the links to related recipes (**Figure 2.23**):

```
<h3>You might also like</h3>

<ul>
<li>
<h4><a href="#">You might also like</a></h4>
<p>The recipe in the link is half the original recipe that yields about 18
large muffins. I always cut down on the butter in this recipe.</p>
</li>

<li>
<h4><a href="#">Pistachio and Dried Fruit Cake</a></h4>
<p>The author says her mother makes this cake during Lent, so it is a
coincidence that I made it for the Easter weekend.</p>
</li>

<li>
<h4><a href="#">Brownie Berry Tower</a></h4>
<p>For the final class of this course, we made a tall cake, with brownie
layers sandwiching two kinds of cream and strawberries.</p>
</li>
</ul>
```

With both the main and additional areas of content complete, it's a good time to preview your document in your development browser to see the structure of your content. It's also a great time to validate your markup to ensure that no errors have crept in along the way.

Adding your first divisions

By now it should be clear that these two content areas are separate but are also related. You can group each into a division, and to add further meaning, you can give each one an identity that relates to the meaning of the content it contains:



You might also like

Lemon Blueberry Muffins

The recipe in the link is half the original recipe which yields about 18 large muffins. I always cut down on the butter in this recipe [..]

Pistachio and Dried Fruit Cake

The author says her mother makes this cake during Lent so it is a coincidence that I made it for the Easter weekend. [..]

Brownie Berry Tower

For the final class of this course, we made a tall cake, with brownie layers sandwiching two kinds of cream and strawberries. [..]

2.23 **Close-up of related recipes**

```
<div id="content_main">
 Main content
</div>

<div id="content_sub">
 Secondary content
</div>
```

Because the two areas are related, you can enclose them both inside a content division to further cement their relationship:

```
<div id="content">
<div id="content_main">
 Main content
</div>

<div id="content_sub">
 Secondary content
</div>
</div>
```

Each of these three meaningfully labeled divisions will soon become an opportunity for you to add your visual style.

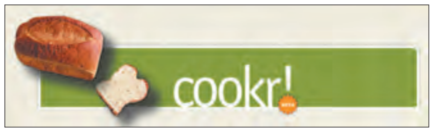
Adding the site furniture

With your content areas complete, it is time to add the branding area, navigation, and site information that form the site's furniture. This furniture will appear on every page of your site. Once again, start working from the content out, tackling each area in turn.

Branding

Your document contains two pieces of branding information: the site's name that is graphically presented as a logo in the static design and a tag line that the visual design has intended not to be visible. Think of this hidden tag line as a piece of embedded information that will be useful to visitors using browsers that do not support style sheets and also to search engines (**Figure 2.24**):

```
Cookr!
A great place to store and share your favorite recipes
Kimberly Blessing
```



2.24 **Hiding the content**

You have already decided on an `<h1>` element for the site name, but what about the tag line? In its basic form, this tag line is a quotation from a happy customer, and the `<blockquote>` element is a perfect choice. You can also take this opportunity to cite the source of the quotation by including the `<cite>` element:

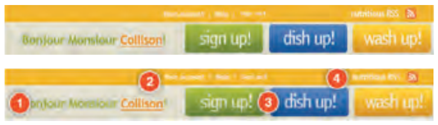
```
<blockquote>
<p>A great place to store and share your favorite recipes</p>
<p><cite>Kimberly Blessing</cite></p>
</blockquote>
```

You can now relate both the site name and the tag line in a division to give them an extra level of meaning and get the added advantage of a hook for styling those elements:

```
<div id="branding">
<h1>Cookr</h1>
<blockquote>
<p>A great place to store and share your favorite recipes</p>
<p><cite>Kimberly Blessing</cite></p>
</blockquote>
</div>
```

Adding navigation

It is now time for you to address the elements you will need for the navigation area. Although this navigation might at first look complex with its different visual styles, by working from the content out you will realize that it is no more complex than other areas of your document (**Figure 2.25**).



2.25 **Close-up of the navigation**

Looking at this navigation in the static design and grey boxes, you will see several distinct types of links:

- A personalized welcome message containing a link from the visitor's name
- Account, help, and sign-out links (tools)
- Image links for the "sign up," "dish up," and "wash up" features
- A link to the site's RSS feed

I suggest you start by establishing the order of importance of these links.

1. Welcome message (for a returning visitor)
2. Features
3. Tools
4. RSS feed

You can now choose the appropriate markup for these lists, working in that order:

```
<p>Bonjour Monsiour <a href="#">Collison</a></p>

<ul id="nav_features">
<li><a href="#">sign up!</a></li>
<li><a href="#">dish up!</a></li>
<li><a href="#">wash up!</a></li>
</ul>

<ul id="nav_tools">
<li><a href="#">Your account</a></li>
<li><a href="#">Help</a></li>
<li><a href="#">Log out</a></li>
<li><a href="#">Nutritious RSS</a></li>
</ul>
```

You can now extend the meaning of these navigation list items by giving each a unique identity that reflects the function of the list that each contains:

```
<p>Bonjour Monsiour <a href="#">Collison</a></p>

<ul id="nav_features">
<li id="nav_signup"><a href="#">sign up!</a></li>
<li id="nav_dishup"><a href="#">dish up!</a></li>
<li id="nav_washup"><a href="#">wash up!</a></li>
</ul>
```

```
<ul id="nav_tools">
<li id="nav_account"><a href="#">Your account</a></li>
<li id="nav_help"><a href="#">Help</a></li>
<li id="nav_logout"><a href="#">Log out</a></li>
<li id="nav_rss"><a href="#">Nutritious RSS</a></li>
</ul>
```

I also suggest you add headings to both of these lists:

```
<h2>Site features</h2>

<h2>Tools</h2>
```

These headings will not be visible in the browser but will help visitors who will not be able see the visual design. You can think about these hidden headings as embedded helpers to further clarify the lists that follow them.

Cement the relationship between headings and lists by grouping these elements inside their own uniquely identified division:

```
<div id="nav_main">
<p>Bonjour Monsiour <a href="#">Collison</a></p>

<h2>Site features</h2>
<ul id="nav_features">
<li id="nav_signup"><a href="#">sign up!</a></li>
<li id="nav_dishup"><a href="#">dish up!</a></li>
<li id="nav_washup"><a href="#">wash up!</a></li>
</ul>
```

```
<h2>Tools</h2>
<ul id="nav_tools">
<li id="nav_account"><a href="#">Your account</a></li>
<li id="nav_help"><a href="#">Help</a></li>
<li id="nav_logout"><a href="#">Log out</a></li>
<li id="nav_rss"><a href="#">Nutritious RSS</a></li>
</ul>
</div>
```

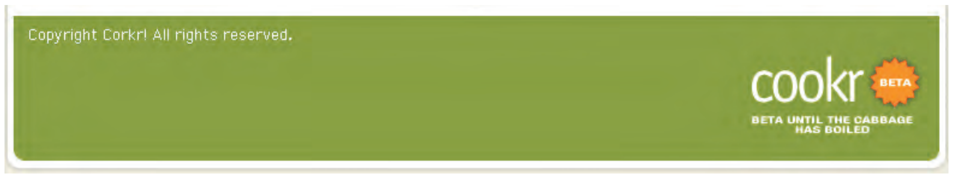
Site information footer

Finally, you can turn to the footer that will contain the site information. Typically this might include a copyright statement, legal information, and perhaps a link to the start of the page (**Figure 2.26**).

Although the visual designer has chosen not to show these elements, you will see by looking at the grey boxes that the content should be present in the document. Deciding on the elements for this content should be straightforward to you by now: a heading containing an anchor to the top of the page and two paragraphs all grouped together within a site-information division:

```
<div id="site_info">

<h5><a href="#cookr-co-uk" title="Back to top">Cookr!</a></h5>
<p>Beta until the cabbage has boiled</p>
<p>Copyright Cookr! All Rights Reserved</p>

</div>
```

Once again, this is a good opportunity to preview your document in your development browser and to validate your markup to ensure it doesn't contain any errors.

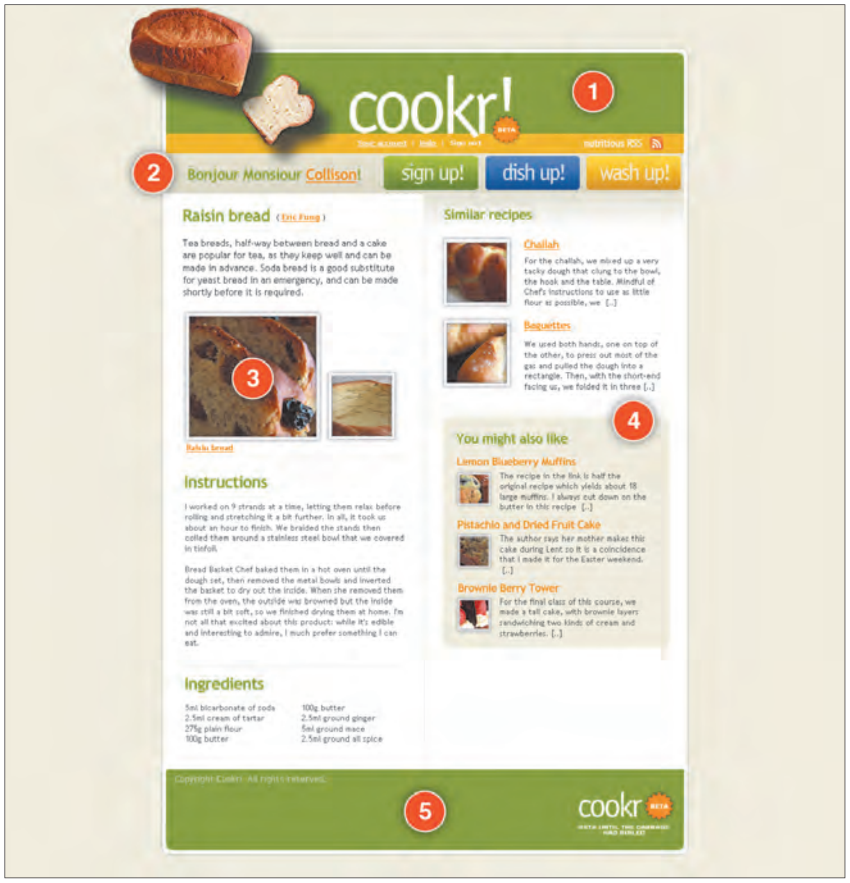

Copyright Corkr! All rights reserved.
cookr
BETA
BETA UNTIL THE CABBAGE HAS BOILED

2.26 **Showing detail of site-information footer including a link to the top of the page**

Arranging content into a meaningful order

It is time for you to put all these elements together in logical order. Before you dive back into your markup, I suggest you start by listing the content in the most logical order (**Figure 2.27**):

1. Branding (site name and tag line)
2. Navigation
3. Main content
4. Supplementary content
5. Site information



Raisin bread (Eric Fung)

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

Bread Basket Chef baked them in a hot oven until the dough set, then removed the metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

Ingredients

5ml bicarbonate of soda	100g butter
2.5ml cream of tartar	2.5ml ground ginger
275g plain flour	5ml ground mace
100g butter	2.5ml ground all spice

Similar recipes

Challah

For the challah, we mixed up a very tacky dough that clung to the bowl, the hook and the table. Mindful of Chef's instructions to use as little flour as possible, we [..]

Baguettes

We used both hands, one on top of the other, to press out most of the gas and pulled the dough into a rectangle. Then, with the short-end facing us, we folded it in three [..]

You might also like

Lemon Blueberry Muffins

The recipe in the link is half the original recipe which yields about 18 large muffins. I always cut down on the butter in this recipe [..]

Pistachio and Dried Fruit Cake

The author says her mother makes this cake during Lent so it is a coincidence that I made it for the Easter weekend. [..]

Brownie Berry Tower

For the final class of this course, we made a tall cake, with brownie layers sandwiching two kinds of cream and strawberries. [..]

Copyright Cookr. All rights reserved.

2.27 **Ordering the content in a logical manner**

This list will become your document's content order. For easy scanning through the order, I have added numbered comments that relate to this order.

I have now also added the `<html>`, `<head>`, and `<body>` elements and my chosen `DOCTYPE` to form a complete XHTML document:

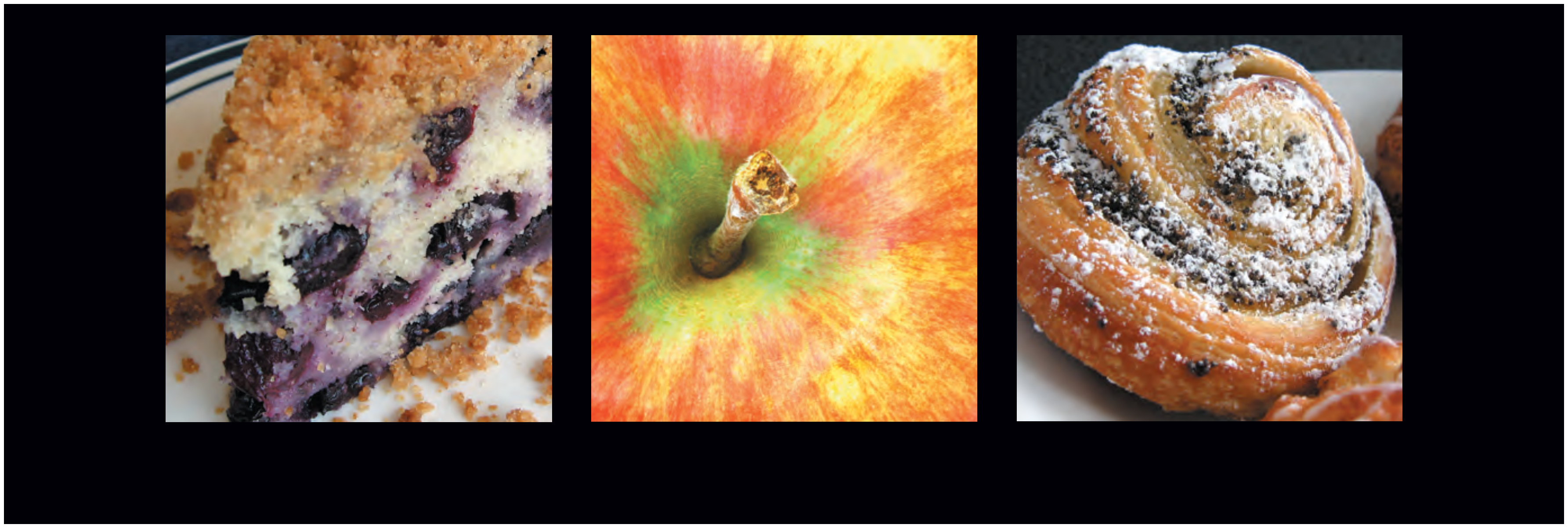
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Cookr! | Raisin bread</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" /> </head>

<body id="cookr-co-uk" class="recipe">

<!-- 1. Branding -->
<div id="branding">
<h1>Cookr!</h1>
<blockquote>
<p>A great place to store and share your favorite recipes</p>
<p><cite>Kimberly Blessing</cite></p>
</blockquote>
</div>

<!-- 2. Navigation -->
<div id="nav_main">

<p>Bonjour Monsiour <a href="#">Collison</a></p>

<h2>Site features</h2>
<ul id="nav_features">
<li id="nav_signup"><a href="#">sign up!</a></li>
<li id="nav_dishup"><a href="#">dish up!</a></li>
<li id="nav_washup"><a href="#">wash up!</a></li>
</ul>

<h2>Tools</h2>
<ul id="nav_tools">
<li id="nav_account"><a href="#">Your account</a></li>
```

```
<li id="nav_help"><a href="#">Help</a></li>
<li id="nav_logout"><a href="#">Log out</a></li>
<li id="nav_rss"><a href="#">Nutritious RSS</a></li>
</ul>
</div>

<div id="content ">
<!-- 3. Main content -->
<div id="content_main">

<h2>Raisin bread</h2>
<p>Tea breads, halfway between bread and a cake are popular for tea, as they
keep well and can be made in advance. Soda bread is a good substitute for
yeast bread in an emergency and can be made shortly before it is required.</p>

<h3>Instructions</h3>
<p>I worked on 9 strands at a time, letting them relax before rolling and
stretching them a bit further. In all, it took us about an hour to finish. We
braided the strands and then coiled them around a stainless steel bowl that we
covered in tinfoil.</p>

<p>Bread Basket Chef baked them in a hot oven until the dough set, then
removed the metal bowls, and inverted the basket to dry out the inside. When
she removed them from the oven, the outside was browned but the inside was
still a bit soft, so we finished drying them at home. I'm not all that excited
about this product: while it's edible and interesting to admire, I much prefer
something I can eat.</p>

<h3>Ingredients</h3>
<ul>
<li>5<abbr title="Mililitres">ml</abbr> bicarbonate of soda</li>
<li>2.5<abbr>ml</abbr> cream of tartar</li>
<li>275<abbr title="Grammes">g</abbr> plain flour</li>
<li>100<abbr>g</abbr> butter</li>
<li>2.5<abbr>ml</abbr> ground ginger</li>
<li>5<abbr>ml</abbr> ground mace</li>
<li>2.5<abbr>ml</abbr> ground all spice</li>
</ul>
</div>
```

```
<!-- 4. Supplementary content -->
<div id="content_sub">

<h3>Similar recipes</h3>

<ul>
<li>
<h4><a href="#">Challah</a></h4>
<p>For the challah, we mixed up a very tacky dough that clung to the bowl, the
hook, and the table. Mindful of Chef's instructions to use as little flour as
possible.</p>
</li>
<li>
<h4><a href="#">Baguettes</a></h4>
<p>We used both hands, one on top of the other, to press out most of the gas
and pulled the dough into a rectangle. Then, with the short end facing us, we
folded it in three.</p>
</li>
</ul>

<h3>You might also like</h3>

<ul>
<li>
<h4><a href="#">You might also like</a></h4>
<p>The recipe in the link is half the original recipe that yields about 18
large muffins. I always cut down on the butter in this recipe.</p>
</li>
<li>
```

```
<h4><a href="#">Pistachio and Dried Fruit Cake</a></h4>
<p>The author says her mother makes this cake during Lent, so it is a
coincidence that I made it for the Easter weekend.</p>
</li>

<li>
<h4><a href="#">Brownie Berry Tower</a></h4>
<p>For the final class of this course, we made a tall cake, with brownie
layers sandwiching two kinds of cream and strawberries.</p>
</li>
</ul>
</div>
</div>

<!-- 5. Site information -->
<div id="site_info">
<h5><a href="#cookr-co-uk" title="Back to top">Cookr!</a></h5>
<p>Beta until the cabbage has boiled</p>
<p>Copyright Cookr! All Rights Reserved</p>
</div>

</body>
</html>
```

With all your markup elements in place, it is a great idea to once again preview your document in your browser and take the opportunity to validate your markup before you move on to implementing the design with CSS.



Implementing the static design with CSS

With your markup written and validated, it is time to develop the CSS to implement the static design.

For flexibility, I suggest you divide your CSS across three separate style sheets, in the way I demonstrated earlier:

- Layout styles including display properties, floats and positioning, widths and heights, and margins and padding. Name this style sheet layout.css.
- Color styles including background properties, colors, and images, as well as text colors. Name this style sheet color.css.
- Typographical information including font families and sizes, line heights, letter spacing, and text decorations. Name this style sheet typography.css.

To reduce the number of style sheets that are linked from your XHTML document, it is common practice to link to only one style sheet, as follows:

```
<link rel="stylesheet" type="text/css" href="layout.css" />
```

Then import the remaining style sheets into that by using the `@import` at-rule:

```
@import url(color.css);
@import url(typography.css);
```

Building your layout

The first style sheet you will work with will contain all aspects of the visual layout that has been defined in the static design; in this instance, it's a two-column layout with branding at the top, navigation, and a site-information area at the bottom.

I suggest you start by overriding all the browser styles, which will appear if you do not provide a style. Furthermore, different browsers have different style defaults. By overriding, or *normalizing*, the way elements are styled by the browser, you'll gain far more control—both within the CSS and across the browsers that will interpret it:

```
/* Normalizes margin, padding */
body, div, dl, dt, dd, ul, ol, li, h1, h2, h3, h4, h5, h6, pre, form, fieldset,
input, p, blockquote, th, td
{ margin : 0; padding : 0; }
```

```
/* Normalizes font-size for headers */
h1,h2,h3,h4,h5,h6
{ font-size : 100%; }

/* Removes list-style from lists */
ol,ul
{ list-style : none; }

/* Normalizes font-style and font-weight to normal */
address, caption, cite, code, dfn, em, strong, th, var
{ font-style : normal; font-weight : normal; }

/* Removes list-style from lists */
table
{ border-collapse : collapse; border-spacing : 0; }

/* Removes border from fieldset and img */
fieldset,img
{ border : 0; }

/* Left-aligns text in caption and th */
caption,th
{ text-align : left; }

/* Removes quotation marks from q */
q:before, q:after
{ content :''; }
```

Working from the body

The static design in this example is a common, centered, fixed-pixel-width layout. In this type of design, you can use an outer container or wrapper division to constrain the design to the center of the browser window. You can eliminate the need for this division by using the `<html>` and `<body>` elements to fix and center your design:

```
html {
text-align :  center; }
```

Note

Read more about normalizing browser styles at http://tantek.com/log/2004/09.html#d06t2354 and at http://meyerweb.com/eric/thoughts/2004/09/15/emreallyem-undoing-htmlcss/.

```
body {
width : 770px;
margin : 0 auto;
text-align : left; }
```

Here, a fixed width and autoright and autoleft margins placed in the `<body>` element will center the design within `<html>`.

Creating a positioning context

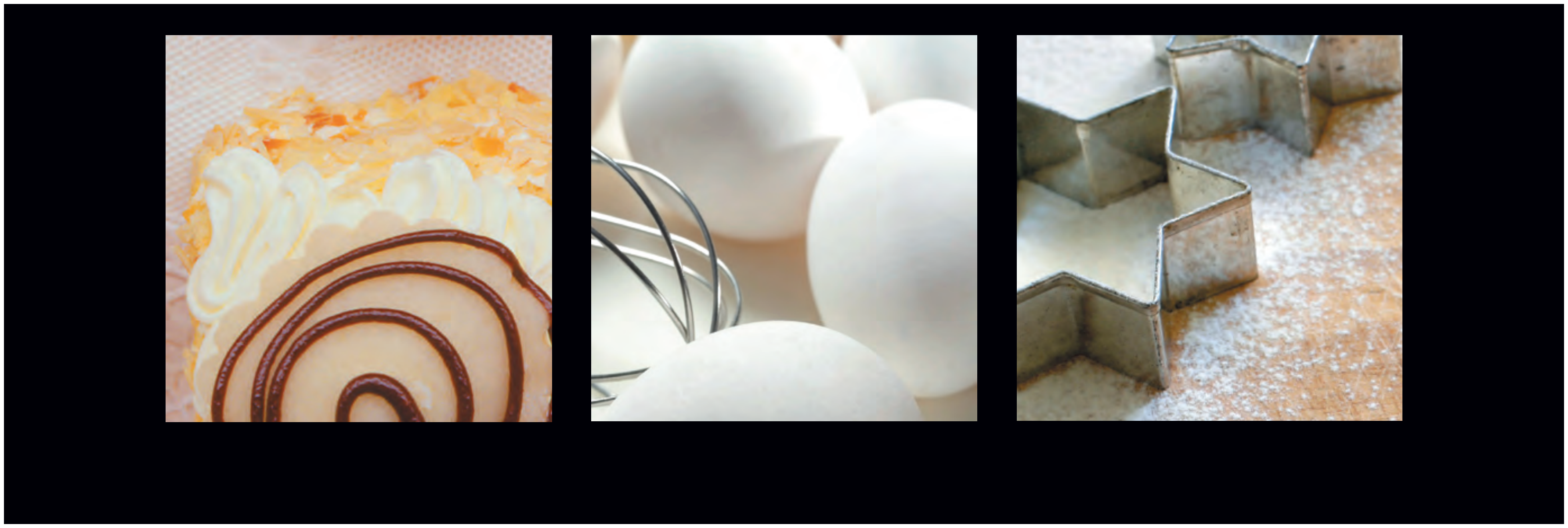
The expected behavior of any absolutely positioned element is to be positioned according to any offsets in relation to the following:

- Its closest positioned ancestor
- In the absence of a positioned ancestor, the root element `<html>`

Because your design will use `<body>`, rather than a container `<div>`, to center the design, you can apply relative positioning to the `<body>` element to establish it as a positioning context for other positioned elements within the design:

```
body {
position : relative;
width : 770px;
margin : 0 auto;
text-align : left; }
```

To implement the static design using positioning (as I recommended earlier in the discussion about CSS in the workflow), you will create two equal columns.



In your markup, both of the columns, content_main and content_sub, are situated inside an outer division, content. To create a positioning context for these two columns, you will add position : relative; to the division labeled content:

```
div#content {
position : relative;
width : 100%; }
```

Creating your two columns

Now you are ready to make two equal columns and use absolute positioning to place the columns to the left and right. To accomplish the static design, you will give each column an equal width, with the main content positioned to the left edge of its container and the additional content positioned 50 percent from the container's left edge:

```
div#content_main {
left : 0;
width : 50%;
padding : 1em 0; }

div#content_sub {
left : 50%;
width : 50%;
padding : 1em 0; }
```

Using this method will create two robust columns that can support supersized images or even gigantic text without breaking the integrity of your design.

Switching the columns

What will you do if your client asks at this point, "Do you think the main content would look better on the right, rather than on the left?"

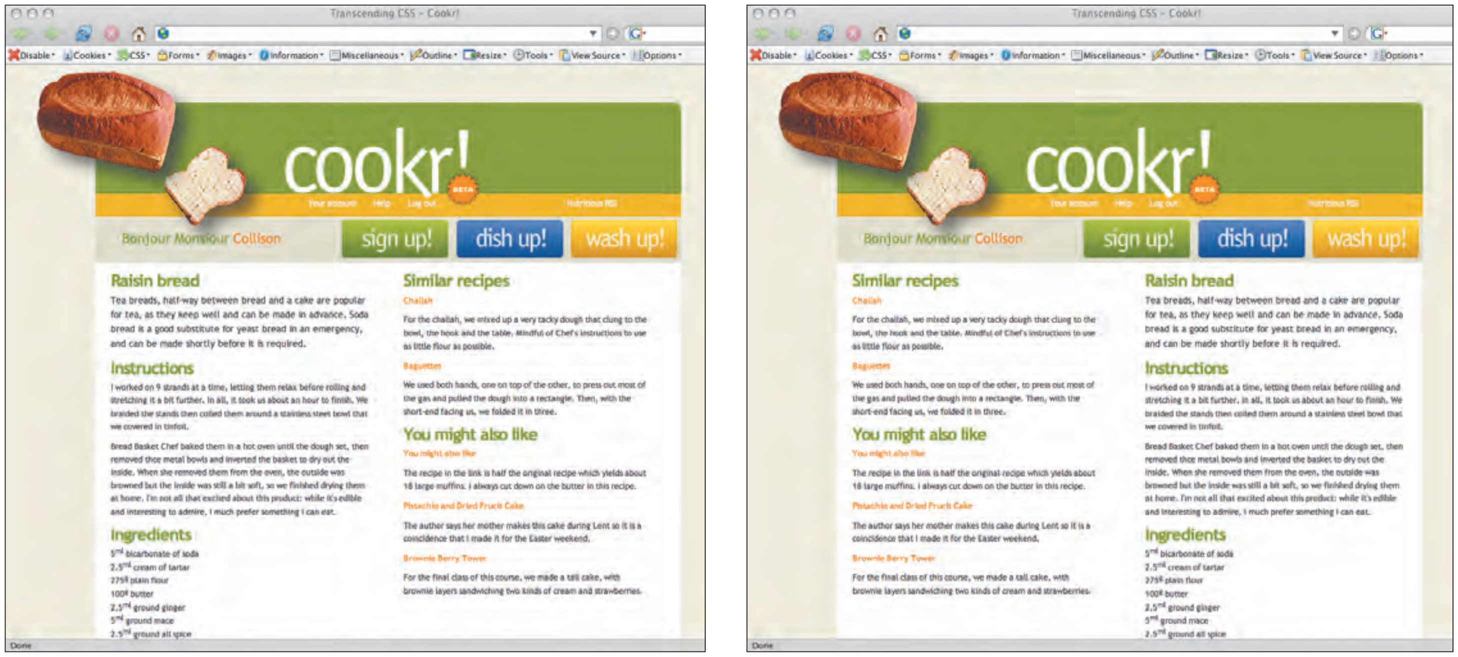
CSS positioning allows you to switch the position of the two columns without changing the source order of your document, even if one of your colleagues is still developing the markup (**Figure 2.28**):

```
div#content_main {
position : absolute;
left : 50%;
width : 50%; }
```

```
div#content_sub {
position : absolute;
left : 0;
width : 50%; }
```

It is equally as simple to change the proportions of your layout by adjusting both the widths of the two columns and their horizontal positions. For example, to alter the layout to 70/30 percent proportions, you can alter the column rules to the following:

```
div#content_main {
position : absolute;
left : 70%;
width : 30%; }

div#content_sub {
position : absolute;
left : 0;
width : 70%; }
```



Raisin bread

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

Bread Basket Chef baked them in a hot oven until the dough set, then removed thoe metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

Ingredients

5ml bicarbonate of soda
2.5ml cream of tartar
275g plain flour
100g butter
2.5ml ground ginger
5ml ground mace
2.5ml ground all spice

Similar recipes

Challah

For the challah, we mixed up a very tacky dough that clung to the bowl, the hook and the table. Mindful of Chef's instructions to use as little flour as possible.

Baguettes

We used both hands, one on top of the other, to press out most of the gas and pulled the dough into a rectangle. Then, with the short-end facing us, we folded it in three.

You might also like

You might also like

The recipe in the link is half the original recipe which yields about 18 large muffins. I always cut down on the butter in this recipe.

Pistachio and Dried Frucit Cake

The author says her mother makes this cake during Lent so it is a coincidence that I made it for the Easter weekend.

Brownie Berry Tower

For the final class of this course, we made a tall cake, with brownie layers sandwiching two kinds of cream and strawberries.

2.28 **Switching the position of the two columns**

Arranging the furniture

With much of the more complex layout positioning now complete, you should add a combined rule that will give all of your remaining layout divisions a width:

```
div#branding, div#nav_main, div#site_info {
width : 100%; }
```

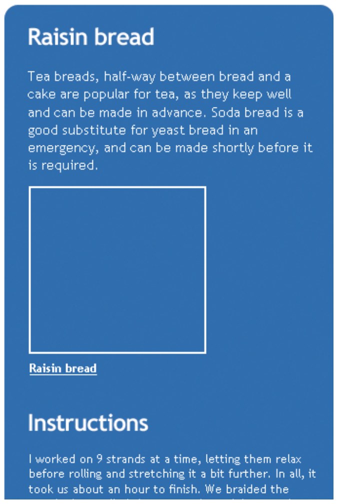
Ah, but a gotcha is hiding here. If you preview your layout in a browser, you will notice immediately that the site-information section, which you intended to be at the foot of the page, has risen up and overlaps the content of your columns. This is not a bug, either in your CSS or in your browser; it is the result of using absolute positioning to create your columns, which means you can easily fix this using the Inman position clearing method.

RELATIVELY SPEAKING

Relative positioning often confuses people who are new to CSS positioning, largely because the term *relative* makes people wonder "relative to what?"

Relatively positioned elements are positioned relative to the normal flow. *Normal flow* is the natural, expected flow of the content within the browser window. Imagine looking at a plain document, marked up with content, headers, and paragraphs only. Then resize the browser window using the lower-right corner. Watch the text while making your browser window smaller and you'll notice the text flows down and to the left. Any element in the normal flow flows with the document logic (**Figure 2.29**).

A relatively positioned box is offset in relation to its natural position within the normal flow. When an element is relatively positioned from its original location in the normal flow, it leaves behind it the space it would normally have occupied. Other elements cannot flow into this "ghosted" space because the element is still considered by the browser to actually be in the normal flow, not out of it (**Figure 2.30**).


Raisin bread

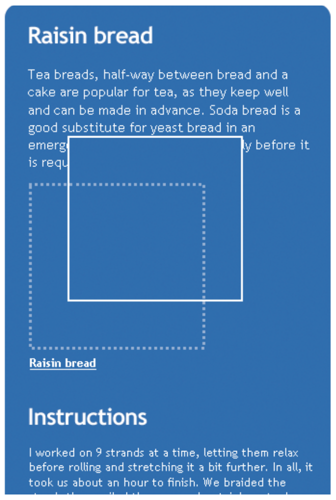
Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

Raisin bread

Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the

2.29 Box in normal flow


Raisin bread

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emerg y before it is requ

Raisin bread

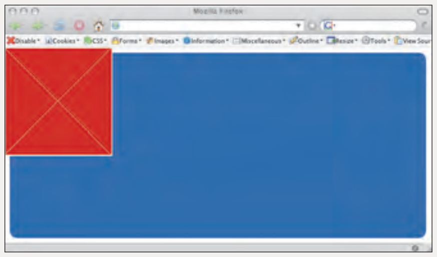
Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the

2.30 Relatively positioned box leaving a space behind

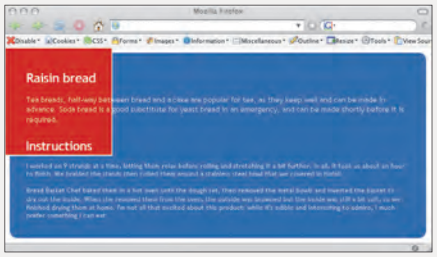
Understanding absolute positioning

An *absolutely positioned* element is positioned first to its closest positioned ancestor. If there is no positioned ancestor, the element is positioned to the root element `<html>` (**Figure 2.31**).



2.31 **An element absolutely positioned to root**

Absolutely positioned elements are considered to be out of the normal flow of the document. Therefore, text and other elements can flow up into any space the element was taking up prior to being offset (**Figure 2.32**).



2.32 **Normal flow intact, with other elements flowing up to fill the space**

Using Inman position clearing to fix your footer

During the prototyping phase, you can use a combination of JavaScript and CSS to force this site information to drop below the absolutely positioned columns. Inman position clearing uses JavaScript to position the site-information `<div>` element underneath the absolutely positioned columns once the browser has calculated their heights.

For this solution to work, you must place a link to the script immediately prior to the closing `</body>` tag in your XHTML:

```
<script type="text/javascript" src="si-clear-children.
js"></script>
</body>
```

To enable this script to work its magic, you need to add several extra `class` attributes to your content divisions in your XHTML:

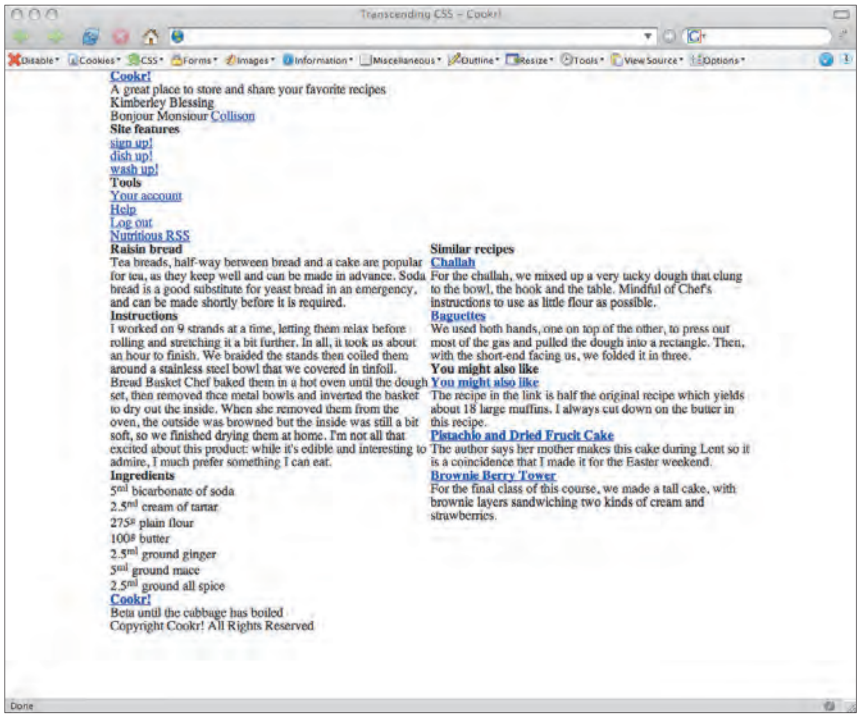
```
<div id="content" class="c clear_children">
<div id="content_main" class="pc cc_tallest">
Main content
</div>
<div id="content_sub" class="sc">
Additional content
</div>
</div>

/* =si_clear_children */
.pc,.sc { position : absolute; top: 0; left: 0; }
.clear_children,.cc_tallest { position: relative; }
/*\*/* html .clear_children { display: inline;}/**/
.cc_tallest:after { content: ''; } /* PREVENTS A REDRAW
BUG IN SAFARI */
```

Your site-information footer will now take its rightful place underneath the two columns.

Note: You can find out more about Inman position clearing at www.shauninman.com/plete/2006/05/clearance-position-inline-absolute.

Moving on, or handing over?

In a team environment, this is another opportunity for you to hand over your work to another team member or a different department, for example, to add functionality or perhaps CMS (Content Management System) integration. This basic layout (**Figure 2.33**) provides your technical developers with a robust platform on which to develop while you continue working to accomplish the full static design.



Cookr!
A great place to store and share your favorite recipes
Kimberley Blessing
Bonjour Monsiour Collison
Site features
sign up!
dish up!
wash up!
Tools
Your account
Help
Log out
Nutritious RSS
Raisin bread
Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.
Instructions
I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.
Bread Basket Chef baked them in a hot oven until the dough set, then removed thce metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.
Ingredients
5^{ml} bicarbonate of soda
2.5^{ml} cream of tartar
275^{g} plain flour
100^{g} butter
2.5^{ml} ground ginger
5^{ml} ground mace
2.5^{ml} ground all spice
Cookr!
Beta until the cabbage has boiled
Copyright Cookr! All Rights Reserved

Similar recipes
Challah
For the challah, we mixed up a very tacky dough that clung to the bowl, the hook and the table. Mindful of Chef's instructions to use as little flour as possible.
Baguettes
We used both hands, one on top of the other, to press out most of the gas and pulled the dough into a rectangle. Then, with the short-end facing us, we folded it in three.
You might also like
You might also like
The recipe in the link is half the original recipe which yields about 18 large muffins. I always cut down on the butter in this recipe.
Pistachio and Dried Frucit Cake
The author says her mother makes this cake during Lent so it is a coincidence that I made it for the Easter weekend.
Brownie Berry Tower
For the final class of this course, we made a tall cake, with brownie layers sandwiching two kinds of cream and strawberries.

2.33 Viewing the prototype with only layout styles applied

Basic color styles

Now it's time for you to add the background colors and images for your page backgrounds and content areas:

```
html {
background-color : #f1efe2; }
```

```
div#content {
background-color : #fff; }

div#site_info {
background : transparent url(site_info.png) no-repeat 0 0; }
```

Building brand

If you refer to the static design, you'll see the branding area features a rounded-corner, green background image, and a logo and image that break out of the top and left of the design. Here's the markup that was chosen for this area:

```
<div id="branding">
<h1>Cookr!</h1>
<blockquote>
<p>A great place to store and share your favorite recipes</p>
<p><cite>Kimberly Blessing</cite></p>
</blockquote>
</div>
```

You'll add some room above the branding area to make space for positioning your logo and image by adding top padding to your <body> element:

```
body {
padding-top : 50px; }
```

Style the branding by giving it a height that matches its background image:

```
div#branding {
height : 120px;
background : transparent url(branding.png) no-repeat 0 0;  }
```

Adding the logo

To create the effect of the logo and image breaking out of the branding area, you will use negative offset values on the absolutely positioned <h1> element. This will move them outside the <body> element:

```
h1 {
position : absolute;
top : -10px;
left : -80px;
}
```

To accomplish the static design, you will need the branding images to overlap the navigation below it. To ensure that these images always remain in the foreground and above any other positioned elements, give your heading a high `z-index`:

```
div#branding {
position : relative;
z-index : 10; }
```

Note: You can read all about `z-index` and image replacement in my article "Z's not dead baby, Z's not dead" at http://24ways.org/advent/zs-not-dead-baby-zs-not-dead/.

For this design, you will need to replace the `<h1>` element with an alpha-transparent PNG image. So many image replacement techniques now exist that it can be hard to keep up-to-date. I suggest you use the simple and reliable Phark method:

```
h1 {
position : absolute;
top : -60px;
left : -80px;
width : 588px;
height : 253px;
background : transparent url(h1.png) no-repeat;
text-indent : -9999px; }
```

This method hides the header text by using negative text indentation to the point where it disappears off the left edge of the browser viewport. You can use a similar technique for moving the tag line out of view. This text will still be available to visitors who cannot see the visual design:

```
div#branding blockquote {
position : absolute;
top : -9999px;
}
```

2.34 **Completed branding**

The branding area is now complete with the header styled, the site logo in place, and the tag line moved out of view (**Figure 2.34**).

Styling the navigation

With your branding complete, it is time to turn your attention to the slightly more complex navigation area. If you refer to the markup you created for the navigation division, you will see you included a paragraph and two unordered lists. You gave each of the lists, and the items they contain, unique identities:

```
<div id="nav_main">

<p>Bonjour Monsiour <a href="#">Collison</a></p>
<h2>Site features</h2>
<ul id="nav_features">
<li id="nav_signup"><a href="#">sign up!</a></li>
<li id="nav_dishup"><a href="#">dish up!</a></li>
<li id="nav_washup"><a href="#">wash up!</a></li>
</ul>

<h2>Tools</h2>
<ul id="nav_tools">
<li id="nav_account"><a href="#">Your account</a></li>
<li id="nav_help"><a href="#">Help</a></li>
<li id="nav_logout"><a href="#">Log out</a></li>
<li id="nav_rss"><a href="#">Nutritious RSS</a></li>
</ul>
</div>
```

You can now put each of those attributes to good use by selecting these elements using their id attributes.

But before you turn your attention to the elements, you'll start by preparing their parent, the division you labeled nav_main. Add a background image and a height that matches that image:

```
div#nav_main {
position: relative;
height : 50px;
background : #edc025 url(nav_main.png) no-repeat 0 0; }
```

Because so many of its children will be positioned, you should establish the parent as the positioning context and set a low z-index because you'll want some of these positioned elements to sit behind others on the page:

```
div#nav_main {
z-index : 1;
height : 90px;
background : #edc025 url(nav_main.png) no-repeat 0 0; }
```

Features navigation

You will start by using absolute positioning to place the features navigation list on the right of the navigation area:

```
ul#nav_features {
position : absolute;
top : 35px;
left : 325px;
margin : 0;
width : 440px;
height : 50px; }
```

Ready to add the images that form the features navigation buttons? Opt for a simple solution that places all three buttons as a background image to the features list. Then, lay each of the anchors over the background image by using absolute positioning and giving each anchor an explicit height and width:

```
ul#nav_features {
position : absolute;
top : 35px;
left : 325px;
margin : 0;
width : 440px;
height : 50px;
background : transparent url(nav_features.png) no-repeat; }

ul#nav_features li {
display : inline;  }

li#nav_signup {
left : 0; }

li#nav_dishup {
left : 150px; }
```

```
li#nav_washup {
left : 300px; }

li#nav_signup a, li#nav_dishup a, li#nav_washup a {
display : block;
height : 50px;
width : 140px;
text-indent : -9999px; }
```

Tools navigation

You will form the tools navigation list from simple text links. Once again you will use absolute positioning to place this list where the design demands:

```
ul#nav_tools {
position : absolute;
top : 3px;
left : 280px;
margin : 0;
width : 460px; }
```

You should style each list item to display inline rather than as a block that will occupy a space on a line below:

```
ul#nav_tools li {
display : inline; }
```

To create an even space between each of the anchors, add a margin and padding to each of the anchors, and then reset these styles on specific links by selecting them with their list item's `id name`:

```
ul#nav_tools li a {
margin-right : 10px;
padding-right : 10px; }

li#nav_logout a. li#nav_rss a {
margin-right : 0;
padding-right : 0; }
```

You can position the link to the site's RSS feed using both positioning and image replacement:

```
li#nav_rss {
position : absolute;
right : 0;
width : 120px;
height : 25px; }

li#nav_rss a {
display : block;
width : 120px;
height : 25px;
text-indent : -9999px; }
```

Getting up close and personal

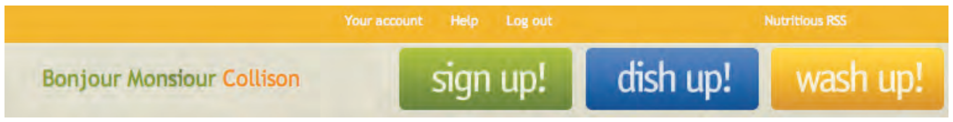
Style and position the paragraph containing the site's personalized welcome message to returning visitors:

```
div#nav_main p {
position : absolute;
top : 45px;
left : 35px;
margin : 0; }
```

Hide the embedded alternate headers by moving them off the top of the screen:

```
div#nav_main h2 {
position : absolute;
top : -9999px; }
```

You've styled each of the different navigation elements using a combination of text and images (**Figure 2.35**).



2.35 **Completed navigation**

Styling the footer

All that remains is to complete the styling of the site information. This area contains a level-five heading, <h5>, and two paragraphs:

```
<div id="site_info">
<h5><a href="#cookr-co-uk" title="Back to top">Cookr!</a></h5>
<p>Beta until the cabbage has boiled</p>
<p>Copyright Cookr! All Rights Reserved</p>
</div>
```

Once again you will use absolute positioning to style the heading and "hide" one of the paragraphs from view. Your first tasks are to establish the division as a positioning context for its absolutely positioned child elements and give it a height that matches its background image:

```
div#site_info {
position : relative;
width : 100%;
height : 120px;
background : transparent url(site_info.png) no-repeat 0 0; }
```

Using image replacement

Replacing the text of this heading should be a familiar process to you by now. Once again I suggest using the Phark method for its simplicity:

```
div#site_info h5 {
position : absolute;
right : 10px;
bottom : 10px;
width : 150px;
height : 70px;
background : transparent url(h5.png) no-repeat; }

div#site_info h5 a {
display : block;
width : 150px;
height : 70px;
text-indent : -9999px; }
```

The semantic framing of the pages in (X)HTML makes it all the way to production in 95% of the projects. The time savings with XHTML wireframes has been about a quarter to a third of the development time saved.

—THOMAS VANDER WAL
www.vanderwal.net/random/category.php?cat=84

Positioning the paragraphs

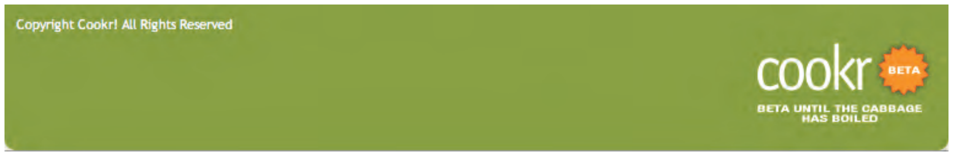
Your final task in styling the footer is to position the paragraph containing the copyright notice. Although your document has two paragraphs of text, the static design dictates that only the second of the two is visible.

Here you can use the power of adjacent sibling selectors to target the second paragraph without affecting the first, all without adding further `class` or `id` attributes to achieve this presentational result.

Apply styles to both of the paragraphs in the site-information area, once again using absolute positioning to place both paragraphs in the same place:

```
div#site_info p {
position : absolute;
left : 10px;
top : 10px;
margin : 0; }

div#site_info h5 + p {
text-indent : -9999px; }
```

If you preview the result in your browser, you will see that both paragraphs now occupy the same space, hardly an attractive result. Your next move will be to hide only the first paragraph, moving it off the left of the browser viewport, by using an adjacent sibling selector and a large amount of negative text indent (**Figure 2.36**).


Copyright Cookr! All Rights Reserved
cookr
BETA
BETA UNTIL THE CABBAGE HAS BOILED

2.36 Completed footer styles

Understanding elements of typographical style

With your main design styling complete, it's time to turn your attention to typography. I suggest you write all your typographical styles in a separate typography.css style sheet:

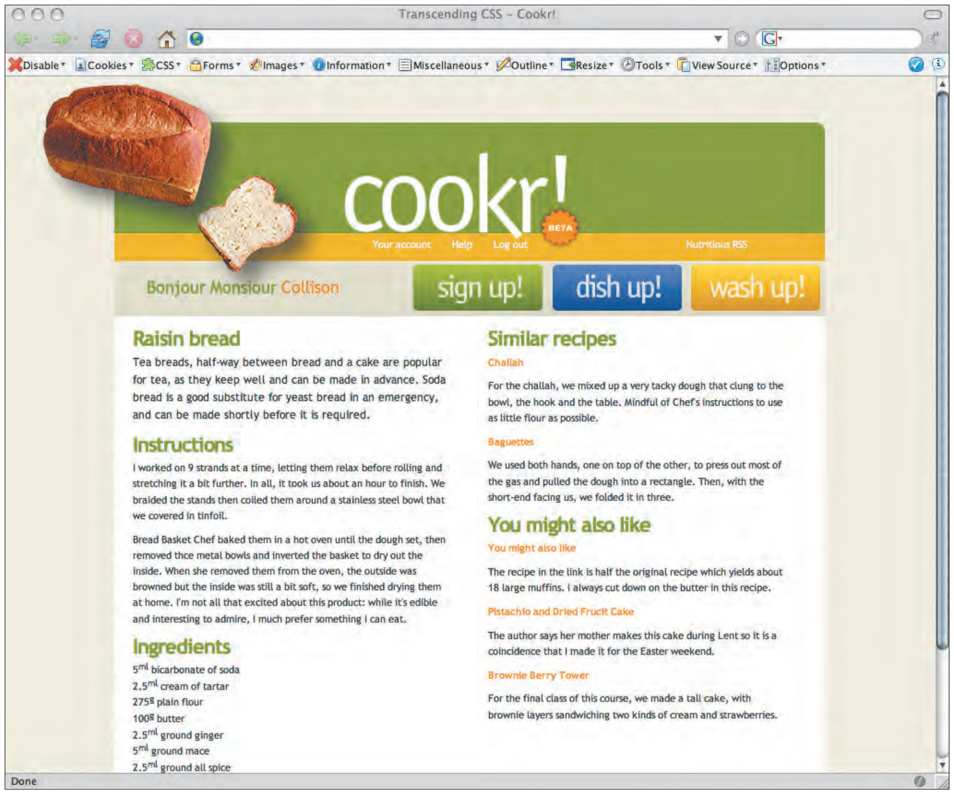
```
body {
font : 72%/1.5 "Trebuchet MS", "Lucida Grande","Lucida Sans Unicode", Verdana,
sans-serif; }

h2, h3, h4, p, ul, blockquote {
margin : 0 20px .75em; }

h2, h3 {
margin-bottom : .15em;
font : 200% "Trebuchet MS", "Lucida Grande","Lucida Sans Unicode", Verdana,
sans-serif;
font-weight : bold;
letter-spacing : -1px; }

li > h4 { margin-left : 0; }

p {
font-size : 100%; }

h2+p {
font-size : 110%; }

li > p { margin-left : 0; }

a:link, a:visited {
text-decoration : none; }
```

You can follow these with your text colors that you have derived from your static design:

```
body {
color : #333; }

h2, h3 {
color : #88a308; }

div#nav_main p {
font-size : 160%;
color : #88a308; }
```

```
div#site_info p {
color : #fff; }

a:link, a:visited {
color : #f90; }

ul#nav_tools a {
color : #fff; }
```

Add the typographic styles, and you've completed the interactive prototype (**Figure 2.37**).



Raisin bread

Tea breads, half-way between bread and a cake are popular for tea, as they keep well and can be made in advance. Soda bread is a good substitute for yeast bread in an emergency, and can be made shortly before it is required.

Instructions

I worked on 9 strands at a time, letting them relax before rolling and stretching it a bit further. In all, it took us about an hour to finish. We braided the stands then coiled them around a stainless steel bowl that we covered in tinfoil.

Bread Basket Chef baked them in a hot oven until the dough set, then removed thce metal bowls and inverted the basket to dry out the inside. When she removed them from the oven, the outside was browned but the inside was still a bit soft, so we finished drying them at home. I'm not all that excited about this product: while it's edible and interesting to admire, I much prefer something I can eat.

Ingredients

5^{ml} bicarbonate of soda
2.5^{ml} cream of tartar
275^{g} plain flour
100^{g} butter
2.5^{ml} ground ginger
5^{ml} ground mace
2.5^{ml} ground all spice

Similar recipes

Challah

For the challah, we mixed up a very tacky dough that clung to the bowl, the hook and the table. Mindful of Chef's instructions to use as little flour as possible.

Baguettes

We used both hands, one on top of the other, to press out most of the gas and pulled the dough into a rectangle. Then, with the short-end facing us, we folded it in three.

You might also like

You might also like

The recipe in the link is half the original recipe which yields about 18 large muffins. I always cut down on the butter in this recipe.

Pistachio and Dried Frucit Cake

The author says her mother makes this cake during Lent so it is a coincidence that I made it for the Easter weekend.

Brownie Berry Tower

For the final class of this course, we made a tall cake, with brownie layers sandwiching two kinds of cream and strawberries.

2.37 The completed interactive prototype for Cookr!

Putting It All Together

You now have a working interactive prototype, a far cry from the traditional result of a static image. This prototype, built with meaningful markup and CSS, enables interaction designers and developers to take development further by adding enhanced functionality using Ajax or related technologies.

I hope you have seen that this method not only makes the most of lightweight, meaningful markup and CSS, which are two of the major advantages when working with Web standards, but that this method is about far more than the mechanics of markup and style sheets.

If you are a lone designer who creates static designs, markup, and CSS, this method is an ideal design framework to give you more flexibility. Proofing your ideas using markup and CSS, in addition to working in Photoshop, enables you to see the realities of your designs far earlier in the process. It allows you to try new ideas and to rapidly see what works and what doesn't. You can find out earlier how your designs will work when adapted for the Web and see how your layouts will behave when implemented as a flexible, rather than as a frozen, layout.

If you work as part of a larger team where designers work separately from technical developers, these prototypes have many advantages for both designers and developers. For designers, they give all the creative advantages and also provide an ideal way to collaborate with technical developers. They show the real meaning of the content and demonstrate the meaningful markup that should be used and maintained throughout the entire process.

For technical developers, these standards-based prototypes offer the firm foundation for development that no other method of static design, wireframe, or prototype can offer: valid, meaningful markup on which to continue developing with microformats, DOM scripting, and any other type of technical programming and development. These prototypes are the ideal starting point for developing feature-rich content by using Ajax or similar combinations of technologies, all without breaking the carefully crafted work of a visual designer.

In large organizations where many people with many different skills work together to make the final product, quality of workmanship and time spent in design and development are both critical factors. By working in parallel, iterative tracks from the same sound foundation of meaningful XHTML, visual designers and technical developers of all kinds can work on a project in sync, with fewer margins for error and less time spent undoing other people's hard work.