RESILIENT WEB DESIGN

CHAPTER 2:

Materials

At the risk of teaching grandmother to suck eggs, I'd like you to think about what happens when a browser parses an HTML element. Take, for example, a paragraph element with some text inside it. There's an opening P tag, a closing P tag, and between those tags, there's the text.

```
<p>some text</p>
```

A web browser encountering this element will display the text between the opening and closing tags. Now consider what happens when that same web browser encounters an element it doesn't recognise.

```
<marklar>some more text</marklar>
```

Once again, the browser displays the text between the opening and closing tags. What's interesting here is what the browser doesn't do. The browser does not throw an error. The browser does not stop parsing the HTML at this point, refusing to go any further. Instead, it simply ignores the tags and displays the content within.

This liberal attitude to errors allowed the vocabulary of HTML to grow over time from the original 21 elements to the 121 elements in HTML5. Whenever a new element is introduced to HTML, we know exactly how older browsers will treat it; they will ignore the tags and display the content.

That's a remarkably powerful feature. It allows browsers to implement new HTML features at different rates. We don't have to wait for every browser to recognise a new element. Instead we can start using the new element at any time, secure in the knowledge that non-supporting browsers won't choke on it.

```
<main>this text will display in any
browser</main>
```

If web browsers treat all tags the same way—displaying their contents—then what's the point of having a vocabulary of elements in HTML?

The meaning of markup

Some HTML elements are literally meaningless. The SPAN element says nothing about the contents within it. As far as a web browser is concerned, you may as well use a non-existent MARKLAR element. But that's the exception. Most HTML elements exist for a reason. They have been created and agreed upon in order to account for specific situations that authors like you and I are likely to encounter.

There are obviously special elements, like the A element, that come bundled with superpowers. In the case of the A element, its superpower lies in the HREF attribute that allows us to link out to any other resource on the web. Other elements like INPUT, SELECT, TEXTAREA, and BUTTON have their own superpowers, allowing people to enter data and submit it to a web server.

Then there are elements that describe the kind of content they contain. The contents of a P element should be considered a paragraph of text. The contents of an LI element should be considered as an item in a list. Browsers display the contents of these elements with some visual hints as to their meaning. Paragraphs are displayed with whitespace before and after their content. List items are displayed with bullet points or numbers before their content.

The early growth of HTML's vocabulary was filled with new elements that provided visual instructions to web browsers: BIG, SMALL, CENTER, FONT. In fact, the visual instructions were the only reason for those elements to exist—they provided no hint as to the *meaning* of the content they contained. HTML was in danger of becoming a visual instruction language instead of a vocabulary of meaning.

A matter of style

Håkon Wium Lie was working at CERN at the same time as Tim Berners-Lee. He immediately recognised the potential of the World Wide Web and its language, HTML. He also realised that the expressive power of the language was in danger of being swamped by visual features. Lie proposed a new format to describe the presentation of HTML documents: Cascading Style Sheets.

He was quickly joined by the Dutch programmer Bert Bos. Together they set about creating a syntax that would be powerful enough to handle the demands of designers, while remaining simple enough to learn quickly. They succeeded.

Think for a moment of all the sites out there on the web. There's a huge variation in visual style: colour schemes, typographic treatments, textures and layouts. All of that variety is made possible by one simple pattern that describes all the CSS ever written:

```
selector {
    property: value;
}
```

That’s it.

CSS shares HTML’s forgiving attitude to errors. If a web browser encounters a selector it doesn’t understand, it simply skips over whatever is between that selector’s curly braces. If a browser sees a property or a value it doesn’t understand, it just ignores that particular declaration. The browser does not throw an error. The browser does not stop parsing the CSS at this point, refusing to go any further.

```
marklar {
    marklar: marklar;
}
```

Just as with HTML, this loose error-handling has allowed CSS to grow over time. New selectors, new properties, and new values have been added to the language’s vocabulary over the years. Whenever a new feature lands in CSS, designers and developers know that they can safely use it, even if it isn’t yet widely supported in browsers. They can rest assured that old browsers will react to new features with complete indifference.

Just because a language is elegant and well-designed doesn't mean that people will use it. CSS arrived later than HTML. Designers didn't spend the intervening years waiting patiently for a way to style their documents on the web. They used what was available to them.

Killing it

In 1996 David Siegel published a book entitled *Creating Killer Websites*. In it, he outlined a series of ingenious techniques for wrangling eye-catching designs out of the raw material of HTML.

One technique involved using a transparent GIF, just one pixel by one pixel in size. If this was inserted into a page as an IMG element, but given precise values in its WIDTH and HEIGHT attributes, designers could control the amount of whitespace in their designs.

Another technique used the TABLE element. This element—along with its children TR and TD—was intended to describe tabular data. But with the right values applied to the widths and heights of table cells, it could be used to recreate just about any desired layout.

These were hacks; clever solutions to tricky problems. But they had unfortunate consequences. Designers were treating HTML as a tool for the appearance of content instead of a language for describing the meaning of content. CSS was a solution to this problem, if only designers could be convinced to use it.

Browser wars

One of the reasons why web designers weren't using CSS was the lack of browser support. Back then there were two major browsers competing for the soul of the web: Microsoft Internet Explorer and Netscape Navigator. They were incompatible by design. One browser would invent a new HTML element or attribute. The other browser would invent their own separate element or attribute to do exactly the same thing.

Perhaps the thinking behind this strategy was that web designers would have to choose which proprietary features they were going to get behind, like children being forced to choose between parents. In reality web designers had little choice but to write for both browsers which meant doing twice the work.

A group of web designers decided enough was enough. They gathered together under the banner of the Web Standards Project and began lobbying Microsoft and Netscape to abandon their proprietary ways and adopt standards such as CSS.

The tide began to turn with the launch of Internet Explorer 5 for the Mac, a browser that shipped with impressive CSS support. If this was the future of web design, life was about to get a lot more productive and creative.

Some forward-thinking web designers caught the CSS bug early. They redesigned their websites using CSS for layout instead of using TABLEs and spacer GIFs. True to the founding spirit of the web, they shared what they were learning and encouraged others to make the switch to CSS.

Perhaps the best demonstration of the power of CSS was a website called the CSS Zen Garden, created by Dave Shea. It was a showcase of beautiful and varied designs, all of them accomplished with CSS. Crucially, the HTML remained the same.

A selection from the cornucopia of designs created for the CSS Zen Garden.

Seeing the same HTML document styled in a multitude of different ways drove home one of the beneficial effects of CSS: separation of concerns.

Coupling

In any system, from urban infrastructure to a computer program, the designers of that system can choose the degree to which the pieces of the system depend on one another. In a tightly-coupled system, every piece depends on every piece. In a loosely-coupled system, all the pieces are independent, with little to no knowledge of the other pieces.

In a tightly-coupled system, each part of the system can make assumptions about the other parts. These systems can be designed quite quickly, but at a price. They lack resilience. If one piece fails, it could take the whole system with it.

Designing a loosely-coupled system can take more work. The payoff is that the overall result is more resilient to failure. Individual parts of the system can be swapped out with a minimum of knock-on effects.

The hacks pioneered by David Siegel tightly coupled structure and presentation into a single monolithic HTML file. The adoption of CSS eased this dependency, bringing the web closer to the modular approach of the UNIX philosophy. The presentational information could be moved into a separate file: a style sheet. That’s how a single HTML document at the CSS Zen Garden could have so many different designs applied to it.

The style sheet still needs to have some knowledge of the HTML document’s structure. Quite often, “hooks” are added into the markup to make it easier to style: specific values of CLASS or ID attributes, for example. So HTML and CSS aren’t completely decoupled. They form a partnership but they also have an arrangement. The markup document might decide that it wants to try seeing other style sheets sometimes. Meanwhile, the style sheet could potentially be applied to other documents. They are loosely coupled.

Dancing about architecture

It takes time for a discipline to develop its own design values. Web design is a young discipline indeed. While we slowly begin to form our own set of guiding principles, we can look to other disciplines for inspiration.

The world of architecture has accrued its own set of design values over the years. One of those values is the principle of material honesty. One material should not be used as a substitute for another. Otherwise the end result is deceptive.

Using TABLEs for layout is materially dishonest. The TABLE element is intended for marking up the structure of tabular data. The end result of using TABLEs, FONT elements, and spacer GIFs is a façade. At first glance everything looks fine, but it won't stand up to scrutiny. As soon as such a website is stress-tested by actual usage across a range of browsers, the façade crumbles.

Using CSS for presentation is materially honest—that's the intended use of CSS. It also allows HTML to be materially honest. Instead of trying to fulfil two roles—structure and presentation—HTML can return to fulfilling its true purpose, marking up the meaning of content.

It's still possible to use (or abuse) CSS to be materially dishonest. For the longest time, there was no easy way to add rounded corners to an element using CSS. Instead, web designers found ways to hack around the problem, putting background images on the element to simulate the same end effect. It worked up to a point, but just like the spacer GIF hack, it was a façade. Then the `border-radius` property arrived. Now designers can have their rounded corners in a materially honest way.

Crucially, designers were able to use new properties like `border-radius` long before every web browser supported them. That's all thanks to the liberal error-handling model of CSS. Newer browsers would display the rounded corners. Older browsers would not throw an error. Older browsers would not stop parsing the CSS and refuse to parse any further. They would simply ignore the instructions they didn't understand and move on. No harm, no foul.

Of course this means that the resulting website will look different in different browsers. Some people will see rounded corners. Others won't.

And that’s okay.