RESILIENT WEB DESIGN

CHAPTER 6:

Steps

"Always design a thing by considering it in its next larger context", said the Finnish architect Eliel Saarinen. "A chair in a room, a room in a house, a house in an environment, an environment in a city plan."

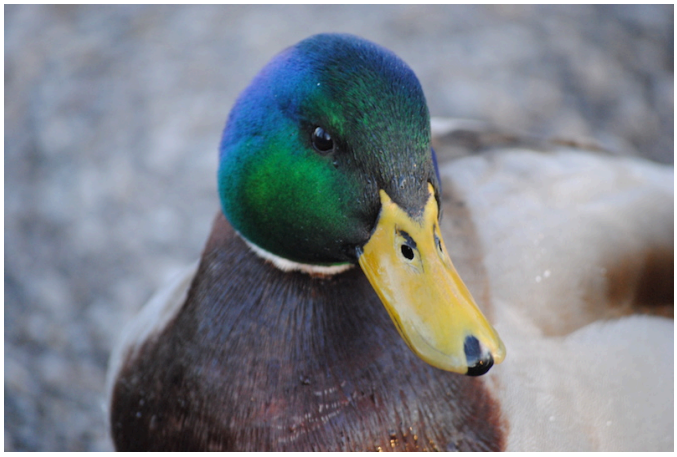
At first glance, web design appears to be a kind of graphic design. Using graphic design tools like Photoshop to design websites reinforces this view. But to get to the heart of a website's purpose, we should consider the interface in its larger context: what are people trying to accomplish?

When designing for the web, it's tempting to think in terms of interactions like swiping, tapping, clicking, scrolling, dragging and dropping. But very few people wake up in the morning looking forward to a day of scrolling and tapping. They're more likely to think in terms of reading, writing, sharing, buying and selling. Web designers need to see past the surface-level actions to find the more meaningful verbs beneath.

In their book *Designing With Progressive Enhancement*, the Filament Group describe a technique they call "the x-ray perspective":

> *Taking an X-ray perspective means looking "through" the complex widgets and visual styles of a design, identifying the core content and functional pieces that make up the page, and finding a simple HTML equivalent for each that will work universally.*

If you're not used to this approach to web design, it can take some getting used to. But after a while it becomes a habit and then it's hard *not* to examine interfaces in this way. It's like trying not to notice bad kerning, or trying not to see the arrow in the whitespace of the FedEx logo, or trying to not to remember that all ducks are actually wearing dog masks.



This duck's bill looks like the face of a dog.
Photograph by Asmaa Dee. Licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Generic license.

Here's a three-step approach I take to web design:

1. Identify core functionality.
2. Make that functionality available using the simplest possible technology.

3. Enhance!

Identifying core functionality might sound like it's pretty straightforward, and after a bit of practice, it is. But it can be tricky at first to separate what's truly necessary from what's nice to have.

Information

Let's say you're a news provider. That right there is the core functionality—to provide news. There are many, many other services you could also provide; interactive puzzles, realtime notifications, and more. Valuable as those services are, they're probably not as important as making sure that people have access to news.

A selection of news outlets.

With that core functionality identified, it's time to move on to step two: how can you make that core functionality available using the simplest possible technology?

Theoretically, a plain text file would be the simplest possible way of providing the news. But as we're talking specifically about the web, let's caveat this step: how can you make the core functionality available using the simplest possible **web** technology? That would be an HTML file served up at a URL.

Even at this early stage it's possible to overcomplicate things. The HTML could be unnecessarily bloated. The URL could be unnecessarily verbose; hard to share or recall.

Now that the news has been marked up with the appropriate HTML elements—articles, headings, paragraphs, lists, and images—it's time for step three: enhance!

By default, the news will be presented using the browser's own stylesheet. It's legible, but not exactly pleasurable. By applying your own CSS, you can sculpt the content into a more pleasing shape. Whitespace, leading, colour and contrast are all at your disposal. You can even use custom fonts—an enhancement that was impossible on the web for many years.

There's no guarantee that every browser will be capable of executing every CSS declaration that you throw at it. That's okay. Those browsers will ignore what they don't understand. Crucially, the news is still available to everyone, regardless of the CSS capabilities of their browser.

For browsers on large-screen devices, you can introduce layout. It might seem odd at first to think of layout as an enhancement, but that's the lesson of mobile-first responsive design. Consider the content first, then mark it up with a sensible source order, then apply layout declarations within media queries.

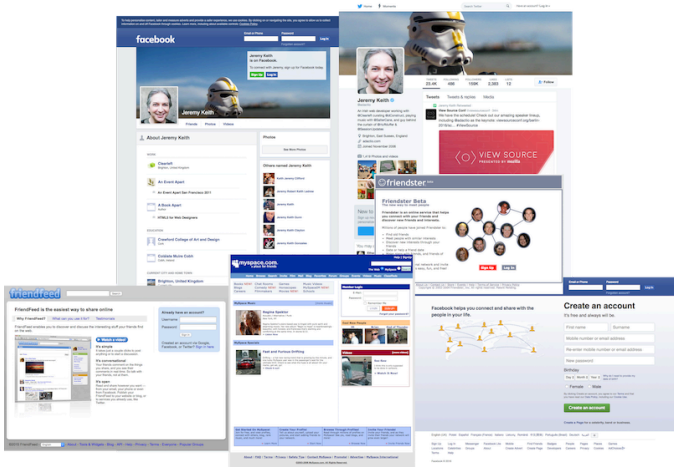
Thanks to ever-evolving nature of CSS, there are multiple ways of applying layout. Like Andy Tanenbaum said:

> *The nice thing about standards is that you have so many to choose from.*

Communication

Applying the three-step process to a news site is relatively straightforward. Catching up with the news is a fairly passive act. To really test this process, we need to apply it to something more interactive.

Suppose we were building a social network. The people using our tool need to be able to communicate with one another regardless of where in the world they are. The core functionality is sending and receiving messages.



Just some of the social networking sites that have graced the web.

Displaying messages in a web browser isn't difficult. There might be a lot of complexity on the server involving databases, syncing, queueing, and load balancing, but the HTML needed to structure a reverse-chronological list isn't very different from the HTML needed for a news site.

Sending a message from the browser to the web server requires HTML that is interactive. That's where forms come in. In this case, a form with a text input and a submit button should be enough, at least for the basic functionality.

People can now receive and respond to messages on our social network, no matter what kind of device or browser they are using. Now the trick is to improve the experience without breaking that fundamental activity.

If we were to leave the site in this HTML-only state, I don't think we'd be celebrating our company's IPO anytime soon. To really distinguish our service from the competition, we need that third step in the process: enhance!

At the very least, we can apply the same logic we used for the news site and style our service. Using CSS we can provide colour, texture, contrast, web fonts, and for larger screens, layout. But let's not stop with the presentation. Let's improve the interaction too.

Right now this social network has the same kind of page-based interaction as a news site. Every time someone sends a message to the server, the server sends back a whole new page to the browser. We can do better than that. Time for some Ajax.

We can intercept the form submission and send the data to the server using Ajax—I like using the word Hijax to describe this kind of Ajax interception. If there's a response from the server, we can also update part of the current page instead of refreshing the whole page. This would also be a good time to introduce some suitable animation.

We can go further. Browsers that support WebSockets can receive messages from the server. People using those browsers could get updates as soon as they've been sent. It's even possible to use peer-to-peer connections between browsers to allow people to communicate directly.

Not every browser supports this advanced functionality. That's okay. The core functionality—sending and receiving messages—is still available to everyone.

Creation

What if our social network were more specialised? Let's make it a photo-sharing service. That raises the bar a bit. Instead of sending and receiving messages, the core functionality is now sending and receiving images.

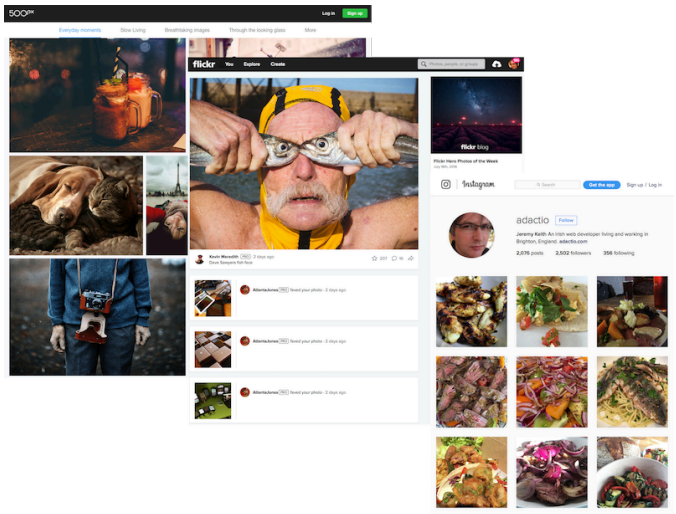


Photo-sharing websites.

The interface needs to show a reverse-chronological list of images. HTML can handle that. Once again we need a form to send data to the server, but this time it needs to be a file upload instead of a text field.

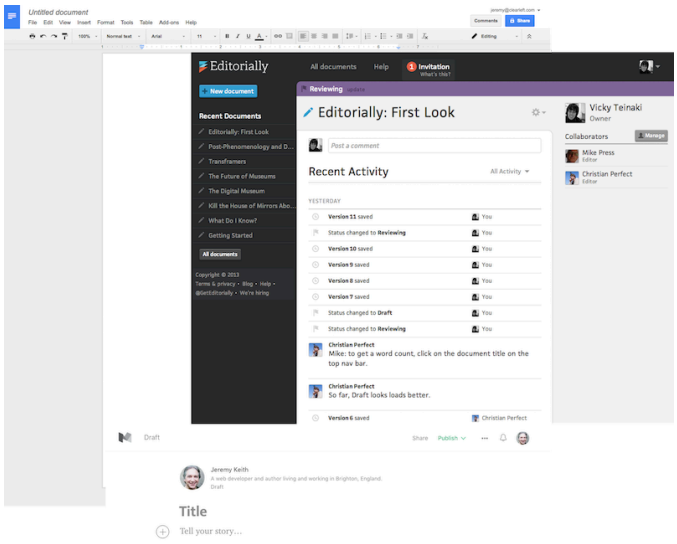
With those changes, the core functionality is in place. Time to enhance.

As well as all the existing enhancements—CSS, web fonts, Ajax, WebSockets—we could make use of the File API introduced in HTML5. This allows us to manipulate the image directly in the browser. We could apply effects to the image before sending it to the server. Using CSS filters, we can offer a range of image enhancements from sepia tones to vignettes. But if a browser doesn't support the File API or CSS filters, people can still upload their duck-facing selfies.

Collaboration

There was a time when using software meant installing separate programs on your computer. Today it's possible to have a machine with nothing more than a web browser installed on it. Writing emails, looking up contact details, making calendar appointments, bookkeeping and other financial tasks can all be done without having to install bespoke applications. Instead, the act of visiting a URL can conjure up the tool you need when you need it.

Delivering software over the web doesn't just replace the desktop-centric way of working. The presence of an internet connection opens up possibilities for all kinds of collaboration. Take, for example, the kinds of applications that were once called "word processors." As long as those programs were tethered to individual machines, trying to collaboratively edit a document was bound to be a tricky task requiring plenty of coordination, sending files back and forth. Using the web, the act of sharing a single URL could allow multiple people to work on the same document.



Collaborative writing tools on the web.

Let's apply the three-step process to a web-based word processor:

> *Identify core functionality.*

The tautological answer would be "processing words." Not very helpful. What do people actually *do* with this software? They write. They share. They edit.

> *Make that functionality available using the simplest possible technology.*

Looking at our three verbs—writing, sharing, and editing—we get one of them for free just by using URLs: sharing. The other two—writing and editing—require the use of a form. A basic TEXTAREA element can act as the receptacle for the words, sentences, and paragraphs that will make up everything from technical reports to the great American novel. Submitting that content to a web server means it can be saved for later.

Technically, that's a web-based word processor, accessible to anyone with a web browser and an internet connection. But the experience is clunky and dull. It would be a shame not to take advantage of some of slicker options available in modern browsers.

> *Enhance!*

Using JavaScript, the humble TEXTAREA can be replaced with a richer editing interface, detecting each keystroke and applying styling on the fly. Web fonts can make the writing experience more beautiful. Ajax will allow work to be saved to the server almost constantly, without the need for a form submission. WebSockets provide the means for multiple people to work on the same document at the same time.

Both Ajax and WebSockets require an internet connection to work. There's no guarantee of a stable internet connection, especially if you're trying to work on a train or in a hotel. Modern browsers provide features which, after the initial page load, can turn the network itself into an enhancement.

If a browser supports some form of local storage, then data can be stored in a client-side database. Flaky network connections or unexpected power outages won't get in the way of saving that important document. Using Service Workers, web developers can provide instructions on what to do when the browser (or the server) is offline.

These are modern browser features that we should be taking full advantage of …once we've made sure that we're providing a basic experience for everyone.

Scale

Ward Cunningham, the creator of the wiki, coined the term "technical debt" to describe a common problem in the world of software. Decisions made in haste at the beginning of a project lead to a cascade of issues further down the line. I like to think of the three-step layered approach as a kind of "technical credit." Taking the time to provide core functionality at the beginning gives you the freedom to go wild with experimentation from then on.

Some people have misunderstood progressive enhancement to mean foregoing the latest and greatest browser technologies, but in fact the opposite is true. Taking a layered approach to building on the web gives you permission to try cutting-edge JavaScript APIs, regardless of how many or how few browsers currently implement them.

We've looked at some examples of applying the three-step approach to a few products and services—news, social networking, photo sharing, and word processing. You can apply this approach to many more services: making and updating items in a to-do list, managing calendar appointments, looking up directions, making reservations at nearby restaurants. Each one can be built with the same process:

1. Identify core functionality.
2. Make that functionality available using the simplest possible technology.
3. Enhance!

This approach works at different scales. It doesn't just work at the highest level of the service; it can also be applied at the level of individual URLs within.

Ask "what is the core functionality of this URL?", make that functionality available using the simplest possible technology, and then enhance from there. This can really clarify which content is most important, something that's important in a mobile-first responsive workflow. Once you've established that, make sure that content is sent from the server as HTML (the simplest possible technology). Then, using conditional loading, you could decide to make Ajax requests for supporting content if the screen real-estate is available. For the URL of an individual news story, the story itself would be sent in the initial response, but related stories or comments could be pushed from the server only as needed (although you can still provide links to the related stories and comments for everyone).

We can go deeper. We can apply the three-step process at the scale of individual components within a page. "What is the core functionality of this component? How can I make that functionality available using the simplest possible technology? Now how can I enhance it?"

A component might be designed to be an all-singing all-dancing interactive map. With x-ray goggles, the core functionality reveals itself to be something much simpler: showing a location. Provide the address of that location in text: the simplest possible technology. Now you can enhance.

It's worth remembering that enhancements can be provided on a sliding scale. The first enhancement for a text address might be to provide a static image. The next level up from that would be to swap out the static image with an interactive Ajax-powered slippy map. If a browser supports the geolocation API, you could show the distance to the location. Layer on some animations and transitions to help convey the directions better.

Site navigation is another discrete component that lends itself well to a sliding scale of enhancements. The core functionality of navigation is to provide links to resources. The simplest—and still the best—technology to enable that is the humble hyperlink. A list of links should do the trick. With that in place, you are now free to enhance it into something really compelling. Off-canvas navigation, progressive disclosure, sliding, swiping, fading, expanding …the sky's the limit.

Because enhancements can be layered on according to the capabilities of each browser, it quickly becomes clear that this approach doesn't simply reduce down to having two versions of everything (the basic version and the enhanced version). Instead the service, the URLs, and the components you are designing could be experienced in any number of ways. And that's okay.

Websites do not need to look exactly the same in every browser.