<http://hesketh.com/publications/progressive_enhancement_and_the_future_of_web_design.html>

Degrade with grace

Cleaning House

that you should always work on a copy

The first thing we do is remove all presentational markup from our document.

Move scripts and stylesheet rulesets outside of the document.

Mark It Up

That Layered Look

The bugs can be used to selectively hide, or show, styles to whatever browser has that bug.

Bugs are Your Friends

Hack Once, Print Anywhere

For testing you can use basic browsers like a netscape navigator.

Web Moderné

<http://uxmag.com/articles/combining-responsive-and-adaptive-strategies-to-solve-mobile-design-challenges>

responsive web design (RWD) adaptive web design (AWD, or mobile websites)

*Responsive Web Design (RWD)*

Pros

**The content stays the same.**

**It can be easily achieved automatically.**

**Cost**

Cons

**Generic not optimized experience.**

**No accounting for users’ behavioral differences with different mediums.**

**Loading time issues.**

*Adaptive Web Design (AWD, or dedicated mobile websites)*

Pros

**Speed.**

**Sensory design.**

Cons

**The content nightmare.**

**Too many versions.**

**Native app ‘transformation’.**

By combining RWD and AWD, however, you can often obtain the same user experience objectives far more cost effectively and efficiently than by using AWD or RWD alone.

<http://www.lukew.com/ff/entry.asp?1390>

Underlying each of these solutions is the same philosophy: give each device only what it needs using media queries and background images, JavaScript, or a server-side solution. This philosophy can dramatically cut down file size and increase performance.

For example, the Bagcheck desktop experience has all the information about a list of content, its comments, updates, and likes at a single URL. We bundle all of these sections (or modules) into a single file then load each section dynamically (as people request them) without a page refresh. While this creates a smooth transition on the desktop, it adds up to a lot of bytes on mobile.

So the mobile Web experience uses a different URL structure. The same URL loads the same initial content but each sub-section (comments, updates, likes) is a separate page with a unique URL as visualized in the image below.

<https://joshemerson.co.uk/blog/responsive-process/>

A Responsive Workflow

How Should We Respond?

What Should We Respond To?

Responsive Images

Is Responsive Design Always Appropriate?

The Responsive Summit

[https://adaptivewebdesign.info](https://adaptivewebdesign.info/)

* [Front Matter](https://adaptivewebdesign.info/1st-edition/read/index.html)
* [Foreword](https://adaptivewebdesign.info/1st-edition/read/foreword.html)
* [Chapter 1: Think of the User, Not the Browser](https://adaptivewebdesign.info/1st-edition/read/chapter-1.html)
* [Chapter 2: Progressive Enhancement with Markup](https://adaptivewebdesign.info/1st-edition/read/chapter-2.html)
* [Chapter 3: Progressive Enhancement with CSS](https://adaptivewebdesign.info/1st-edition/read/chapter-3.html)
* [Chapter 4: Progressive Enhancement with JavaScript](https://adaptivewebdesign.info/1st-edition/read/chapter-4.html)
* [Chapter 5: Progressive Enhancement for Accessibility](https://adaptivewebdesign.info/1st-edition/read/chapter-5.html)
* [Chapter 6: Take It Away](https://adaptivewebdesign.info/1st-edition/read/chapter-6.html)
* [Back Matter](https://adaptivewebdesign.info/1st-edition/read/back.html)

Fault tolerance - (отказоустойчивость) you can view any html page in any web browsers.

Graceful degradation was the philosophical equivalent of fault tolerance’s superficial.

Tasty at any level

One analogy I like to use for progressive enhancement is the peanut M&M. At the center of a peanut M&M is, well, the peanut.

Technologies applied as layers—HTML, then HTML & CSS, then HTML, CSS & JavaScript—can create different experiences, each one equally valid (and tasty). And at the core of it all is the nut: great content.

Progressive enhancement isn’t about browsers. It’s about crafting experiences that serve your users by giving them access to content without technological restrictions. Progressive enhancement doesn’t require that you provide the same experience in different browsers, nor does it preclude you from using the latest and greatest technologies; it simply asks that you honor your content (and your users) by applying technologies in an intelligent way, layer-upon-layer, to craft an amazing experience. Browsers and technologies will come and go. Marrying progressive enhancement with your desire to be innovative and do incredible things in the browser is entirely possible, as long as you’re smart about your choices and don’t lose sight of your users.

Progressive enhancement = excellent customer service

The baseline, first level of experience its most basic form: as text

The second level of experience comes from the semantics of the HTML language itself: emphasasing, bolding and etc.

The third level of experience is the audio-visual one, expressed through the use of CSS and the use of inline images, audio, and video.

The fourth level of experience is the interactive one. In the standards world, this level relies almost entirely on JavaScript

The final level is realized through the application of enhanced semantics and best practices contained within and used in conjunction with the Web Accessibility Initiative’s Accessible Rich Internet Applications (WAI-ARIA) spec.



Google was the first search engine to **take** **semantics into account** when indexing web pages. Starting with the humble anchor (a) element, the cornerstone of their original PageRank algorithm, Google pioneered the use of semantic markup to infer meaning and relevancy. The other search engines soon followed and, as search engine spiders began hunting for other meaningful HTML elements on web pages (e.g., h1 which indicates the most important content on a page), semantic markup became more important to the business world because proper use of it meant a better ranking in search engines and, thereby, a greater opportunity to attract new customers.

With these additions, the markup for this section is now:

And, thanks to the fact that they ignore anything they don’t understand, the markup we’ve used will work in every browser, regardless of age. Sure, modern browsers may treat the newer elements differently, but even text-based browsers (such as Lynx) will be able to access the content. Devoid of style and stripped of JavaScript-based interactivity, the markup just works, providing us with the second level of support in the progressive enhancement continuum. (Remember: the content itself forms the crucial first level).

As good as this markup is, we’ve neglected a major accessibility requirement by not providing any alternate text for our logo image (expressed using the alt attribute). “Alt text,” as it’s most often known,[[3]](https://adaptivewebdesign.info/1st-edition/read/chapter-2.html#footnote-222-3) provides a text-based back-up for users who have images turned off; it is also the content that is read to users of screen reading software (such as the blind), which is why its inclusion is critical.

Returning to the example, I’ve added a simple alt attribute:

<h1><img src="i/logo.png" **alt="Retreats 4 Geeks"**/></h1>

Whereas the alt attribute is used to provide alternative content, the title attribute is used to provide advisory information about an element. In the case of the navigation links in the above example, we can use title to provide the user with information about where each link will take her:

<li><a href="#location" **title="Get the 411 on Gatlinburg, Tennessee"**>**Location**</a></li>

Similarly, further down the page in the “location” section, title provides context to the link that wraps the map:

<a href="http://maps.google.com/…" **title="View Gatlinburg, Tennessee on Google Maps"**> <img src="http://maps.google.com/…" alt="A map showing the location of Gatlinburg, Tennessee"/></a>

As all of the information for the Retreats 4 Geeks event is included on a single page, I’ve grouped each chunk of content into separate article elements[[10]](https://adaptivewebdesign.info/1st-edition/read/chapter-2.html#footnote-222-10), each with a unique id. The article element was introduced as part of HTML5 and demarcates content that forms an independent part of the document, such as a newspaper article, blog post, or, in our case, a distinct topic. Each of the articles on the page is then targeted, using its id as an anchor reference, by the navigation links. Clicking one of these links will jump a user directly to the appropriate content:

<body> <header> <h1><img src="i/logo.png" alt="Retreats 4 Geeks"/></h1> <nav> <ol> <li><a href="**#details**" title="Find out what this retreat is all about">**Details**</a></li> <li><a href="**#schedule**" title="Get familiar with what this retreat will cover">**Schedule**</a></li> **<-- … -->** </ol> </nav> </header> <div id="content"> <article **id="details"**> **<-- … -->** </article> <article **id="schedule"**> **<-- … -->** </article> **<-- … -->** </div></body>

Developed by Tantek Çelik, Matthew Mullenweg, and Eric Meyer, XFN makes use of the oft-neglected rel attribute. The purpose of rel—which you are probably familiar with in the context of the link element for inclusion of an external stylesheet (rel="stylesheet")—is to indicate the relationship of the target of an anchor to the current page. The idea was simple: if I wanted to point from my blog to the blog of a colleague, I could employ XFN and add rel="colleague" to the link. Similarly, if I was linking to my wife’s blog, I would use rel="friend co-resident spouse muse sweetheart co-worker" because she is all of those things.[[11]](https://adaptivewebdesign.info/1st-edition/read/chapter-2.html#footnote-222-11)

An example of XFN in the Retreats 4 Geeks page can be found in the footer[[13]](https://adaptivewebdesign.info/1st-edition/read/chapter-2.html#footnote-222-13):

<footer> <p id="copyright">**&copy;2010 Retreats 4 Geeks. All Rights Reserved.**</p> <p>**Retreats 4 Geeks is an** <a **rel="me"** href="http://easy-designs.net/">**Easy! Designs**</a> **venture.**</p></footer>

**CHAPTER 3:** PROGRESSIVE ENHANCEMENT WITH CSS

For properties, using parsing errors to your advantage is pretty straightforward and it opens up some awesome possibilities. Here’s a quick example using CSS3’s RGBa color scheme:

p { background-color: rgb(137, 224, 160); background-color: rgba(180, 246, 248, .43);}

Perhaps the most famous example of using this technique to selectively deliver rules to one browser over another (more for effect than practicality) is Egor Kloos’ CSS Zen Garden entry titled “Gemination.”

**/\* =Typography \*/**

img {

display:block;

}

**/\* =Layout \*/**

@media screen {

.frame .inner {

border: 10px solid;

}

}

**/\* =Color \*/**

.frame .inner {

background-color: rgb(227, 222, 215);

border-color: rgb(227, 222, 215);

}

You were probably quick to notice the @media block that contains layout rules for the screen. The use of @media here is not accidental: it ensures that every medium is given access to the typography and color rules while the layout rules are restricted to user agents that implement the “screen” media type. Following this setup, you can easily do the same for print or any other medium, but more on that in a bit.

As you can see, there are numerous ways we can use CSS to progressively enhance our web pages. Some techniques, such as taking advantage of parsing errors, are so simple and commonplace that you’re probably using them right now. Others, such as faceted style separation, may provide a slightly different take on your current practices or may be completely foreign to you. When used in combination, however, these techniques weave together, layer upon layer, to provide a tailored experience for every user, no matter what her browser or device supports.

**Apply no style before its time.**

PLANNING AND RESTRAINT

Make no mistake, progressive enhancement with JavaScript requires considerably more effort than it does with CSS or HTML. With a bit of thoughtful reflection and consideration of the numerous factors that affect the web experience, however, it quickly becomes second nature. And, when in doubt, you can always come back to the three maxims:

* Make sure all content is accessible and all necessary tasks can be completed without JavaScript turned on.
* Use JavaScript to generate any additional markup it needs.
* Apply no style before its time.

Web Accessibility Initiative’s Accessible Rich Internet Applications spec (WAI-ARIA or ARIA, for short)

Derek Featherstone surfaced this issue in his excellent article for A List Apart Entitled “ARIA and Progressive Enhancement.”[[7]](https://adaptivewebdesign.info/1st-edition/read/chapter-5.html#footnote-222-7) I highly recommend reading it to better understand the issue. At the time of this writing, a viable solution has not been proposed, but my fingers are crossed.

So what exactly is tabindex juggling? Well, some time in 2005 (it’s hard to pin down the exact origin) it was discovered that assigning a value of “-1” to the tabindex attribute of an element would remove that element from the default tab order of the document.[[10]](https://adaptivewebdesign.info/1st-edition/read/chapter-5.html#footnote-222-10) Interestingly, despite being taken out of the document’s tab order, the element remained focusable via JavaScript (element.focus()), which opened up a lot of possibilities for controlling a user’s experience.

Now get out there and make something great! =)

<https://developers.google.com/webmasters/mobile-sites/mobile-seo/responsive-design>

Responsive web design (RWD) is a setup where the server always sends the same HTML code to all devices and CSS is used to alter the rendering of the page on the device.

To signal to browsers that your page will adapt to all devices, add a meta tag to the head of the document:

<meta name="viewport" content="width=device-width, initial-scale=1.0">  
We recommend using responsive web design.

JavaScript

Common configurations

Three popular implementations of JavaScript for mobile-friendly sites are:

* **JavaScript-adaptive**: In this configuration, all devices are served the same HTML, CSS, and JavaScript content. When the JavaScript is executed on the device, the rendering or behavior of the site is altered. **If a website requires JavaScript, this is Google’s recommended configuration**.
* **Combined detection**: In this implementation, the website uses both JavaScript and server-side detection of device capabilities to serve different content to different devices.
* **Dynamically-served JavaScript**: In this configuration, all devices are served the same HTML, but the JavaScript is served from a URL that dynamically serves different JavaScript code depending on the device’s user-agent.