10-3-2014

Making It Work for Everyone: HTML5 and CSS Level 3 for Responsive, Accessible Design on your Library's Website

Stewart Baker
Western Oregon University, bakersc@wou.edu

Recommended Citation
Baker, S.C. (2014) Making it work for everyone: HTML5 and CSS level 3 for responsive, accessible design on your library's web site. Journal of Library & Information Services in Distance Learning, 8(3-4). doi: 10.1080/1533290X.2014.945825

This is an Accepted Manuscript of an article published by Taylor & Francis in Journal of Library and Information Services in Distance Learning *on October 03 2014, available online:* www.tandfonline.com/10.1080/1533290X.2014.945825

Making It Work for Everyone: HTML5 and CSS Level 3 for Responsive, Accessible Design on your Library's Website

Stewart C. Baker
Western Oregon University

Abstract

This article argues that accessibility and universality are essential to good web design. A brief review of library science literature sets the issue of web accessibility in context. The bulk of the article explains the design philosophies of progressive enhancement and responsive web design, and summarizes recent updates to WCAG 2.0, HTML5, CSS Level 3, and WAI-ARIA. The final section of the paper walks readers through the website creation process. The tools and techniques described in the article can be used to create a library website which can be accessed equally by all patrons without sacrificing aesthetics or usability at any level.

Introduction

Tim Berners-Lee (1998) often credited with inventing the Internet, defines the Web as a dream "of a common information space in which we communicate by sharing information," and states, "its universality is essential" (para. 3). Libraries, in particular, and other public-serving institutions need to ensure that their websites are accessible to as many of their users as possible.

However, despite the potential impact of non-accessible library websites, especially to distance students and other non-local users, library web design is often assigned to non-experts, or to teams consisting of some expert designers and some amateurs. Connell (2008) found that the main selection criterion for web design teams in university libraries was that the individual "showed an interest in web design" (p. 124). According to Connell, only 17.6% of survey respondents stating that the most important quality in a library web designer was "web authoring skills" (p. 125). While there is nothing wrong with this approach *per se*, library websites often suffer as a result, especially where accessibility and other technical issues are concerned.

This paper will provide a summary of the latest developments in web design techniques and technologies. In addition, the paper will discuss how these techniques can be used to make future maintenance and upgrades easier by fully separating content from layout, using contextual markup and creating websites that are accessible from the start and will not need extensive redesigns later down the road.

Literature Review

Numerous articles have been written on the topic of web accessibility for libraries. Hazard (2008) provides a snapshot of the literature and suggests that most articles published review accessible library websites and focus on implementation, the "legal framework for web accessibility," and/or web standards (p. 419-420). In general, these are still the major areas discussed in the literature.

As Vandenbark (2010) points out, "Because the Internet and its design standards are evolving at a dizzying rate, it is difficult to create websites that are both cutting-edge and standards-compliant" (p. 23). Likewise, any literature review on the subject will quickly become outdated. The goal of this review, then, is not to provide a comprehensive view of the field, but to point interested parties to recent articles and books which may serve as useful background reading.

Web Accessibility

Riley-Huff (2012) gives an excellent high-level summary of the basics of web accessibility, legal issues, and technologies; however, as her purpose is to provide "a primer and basic understanding of … website accessibility" (p. 35), she does not go into much detail about the principles of web design or the changes wrought by CSS Level 3 and HTML5.

Brophy and Craven (2007) provide an overview of web accessibility, and point out that web designers should not assume that assistive technologies are a valid substitute for accessible design. Hazard (2008) reinforces this, noting that a survey of ARL membership libraries found significant problems with the implementation of text-only websites, as well as a general failure to maintain them.

Librarians who wish to know more about real world implementations of web accessibility principles and guidelines may consult Zap and Montgomery (2013), for a summary of recent studies in the United States and Canada, or Comeaux and Schmetzke (2013) for accessibility trends in academic library websites in North America between the years 2002 and 2012.

Legal Issues

Fulton (2011) summarizes the legal issues of web accessibility and punctures some of the common myths about legal ramifications in the United States. Vandenbark (2010) explains section 508—the amendment to the Rehabilitation Act of 1973 which

requires that Federal agencies to make websites and other technologies accessible to those with disabilities—in great detail, breaking out the different points of the law and explaining what each means to libraries, and how they can be implemented, as well as providing additional background. McHale (2011) provides a comparison of the US government's Section 508 law with the now-outdated WCAG 1.0, which remains nonetheless useful for librarians who are having difficulties aligning the two sets of criteria.

Standards and Technologies

The literature on web standards and technologies has largely been published outside the sphere of library science. A Book Apart (http://www.abookapart.com/) publishes excellent resources for developers, as well as the well-known *A List Apart* web journal. Librarians who have limited resources, expertise, or time and find themselves responsible for maintaining websites, may wish to invest in Ethan Marcotte's *Responsive Web Design* (2011), Jeremy Keith's *HTML5 for Web Designers* (2010), and Dan Cederholm's *CSS3 for Web Designers* (2010).

Profession specific treatments include Hoy's introduction to HTML5 (2011), McHale's already-mentioned overview of web standards (2011), and Lamb and Johnson's brief summary of HTML5 and CSS 3 tools and resources for school libraries (2013). Reidsma (2012) is an excellent resource for libraries that wish to create a responsive website from a template. His presentation for the American Library Association annual conference in Anaheim in 2012 contains a sample website, downloadable source code, and a list of further reading materials. Fox (2012) provides an overview of how this kind of responsive web design can be used to overcome some of the problems of accessible websites.

The Need for Universally Accessible Websites

Link-Rodrigue (2009) makes a compelling argument for what she calls the inclusion principle: essentially, embracing both the similarities *and* the differences of each individual and group of individuals. Link-Rodrigue states, "We can embrace similarities by focusing on universal design and embrace differences by applying accessible design" (2009, Real-world inclusiveness section, para. 1).

The job of a web designer is best approached through this principle of inclusion: Websites need to be as close to universally accessible as is possible in the real world. The design philosophies of progressive enhancement and responsive web design provide excellent frameworks to do just that.

Progressive Enhancement

In the mid- to late-90s, the so-called browser wars were in full swing, and a big problem for web designers was creating pages that rendered properly regardless of the browser-operating system combination each visitor was using. Much of the discussion

focused on the principle of graceful degradation, which argued for "building the website for the most advanced/capable browsers" and then putting workarounds into place for those who were using older browsers or alternative technologies (Gustafson, 2008, para. 5). Graceful degradation also meant that designers put alternative delivery mechanisms in place if their site used Flash or Javascript to present its content.

The problem with graceful degradation, as Steve Champeon (2003) points out, is that in the real world it is often interpreted to mean "it looks okay in the previous version of Internet Explorer for Windows" (Degrade with grace section, para. 2). Champeon, who coined the term progressive enhancement with Nick Finck, argues that instead of making websites which degrade gracefully, designers should focus their attention on the content of the site, making sure that it displays attractively and logically as content alone. Only after this has been accomplished should successive "layers" of more modern design be added to a website to enhance it. Gustafson (2008) compares the process to the production of a Peanut M&M, in which the content is the peanut, the visual presentation is the chocolate, and client-side scripting such as Javascript is the hard candy shell.

This process, wherein presentational effects are not integrated into page content, is a more logical approach to web design and makes accessibility easier to accomplish as designers no longer need to spend time establishing workarounds when display methods are not accessible. Instead, they simply do not serve the display methods to those who cannot use them. Likewise, progressive enhancement means designers do not need to worry about whether users with outdated phone browsers can see content due to unsupported or new technologies, although they will still need to ensure that the progressive layers of enhancement don't break things.

Responsive Web Design

Responsive web design, at its simplest, is about creating a website which modifies its layout based on the size or media of the device viewing it. The term was coined by Ethan Marcotte after noticing an increase in the number of clients who wanted websites built specifically for iPhones or other mobile devices. As Marcotte (2010) points out, creating a separate subdomain or page for a "mobile" site is problematic: By "quarantin[ing] the mobile experience," the designer must maintain the same content in multiple locations on any future update (para. 5). It is far simpler to change the layout of a single website on the fly, depending on the user's resolution size or other variables. Although the idea of using flexible layouts for websites already existed, Marcotte argued that it did not go far enough, as designers still did not plan for users on anything other than a desktop computer. After learning of an architectural movement called "responsive architecture," Marcotte hit upon the idea that designers should not create a series of "disconnected designs to each of an ever-increasing number of web devices," but "treat them as facets of the same experience" through what he terms responsive web design (Becoming responsive section, para. 3).

According to Marcotte (2010), the key to responsive web design is a CSS technique called media queries. Media queries allow designers to specify several

different design plans within a single CSS document, with the user's browser selecting which to display based on their screen resolution. It's also important to remember that some users will not be viewing the page at all. CSS media queries allow for these users by giving web designers the flexibility to create separate rules for non-visual access methods such as aural or Braille, which can be parsed by speech synthesizers or Braille tactile feedback devices, respectively (Avila, 2013).

By combining flexible layouts with a little CSS magic, responsive web design allows for optimization on a number of devices: Web designers can create a default, content-only page and then apply layout to that content based on whether the end user is viewing the page with a tiny phone browser, a smartphone, a tablet, a desktop computer monitor, or even listening to it via a screen reader or other assistive device.

New Tools for Web Design

Several developments in the past five years have made accessible web design easier to accomplish. The introduction of CSS media queries and other techniques in CSS3 are suggested above. Others include a revision to the W3C's Web Content Accessibility Guidelines (WCAG), an expanded semantic syntax in HTML5, and the Web Accessibility Initiative's Accessible Rich Internet Applications (WAI-ARIA) specification. Used in conjunction with progressive enhancement and responsive web design, the tools described below enable designers to create functional, yet aesthetically pleasing websites that can effectively serve users at all ability levels.

WCAG 2.0

The World Wide Web Consortium's (W3C) Web Access Initiative (WAI) interest group provides guidelines for creating accessible websites, known as the Web Content Accessibility Guidelines (WCAG), the most recent version of which, WCAG 2.0, was formally accepted as a W3C recommendation in 2008. W3C describes these guidelines as a series of "testable statements that are not technology-specific" (2008, Dec 11).

Although the WCAG 2.0 contains too much information within it to reprint here in any detail, the guidelines are based on four principles, which hold that web content must be perceivable, operable, understandable, and robust (see Figure 1).

1. Perceivable - Information and user interface components must be presentable to users in ways they can perceive.
    - This means that users must be able to perceive the information being presented (it can't be invisible to all of their senses)
2. Operable - User interface components and navigation must be operable.
    - This means that users must be able to operate the interface (the interface cannot require interaction that a user cannot perform)
3. Understandable - Information and the operation of user interface must be understandable.
    - This means that users must be able to understand the information as well as the operation of the user interface (the content or operation cannot be beyond their understanding)
4. Robust - Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.
    - This means that users must be able to access the content as technologies advance (as technologies and user agents evolve, the content should remain accessible)

Figure 1. Testable statements from the Web Content Accessibility Guidelines 2.0.

To make WCAG 2.0 more easily achievable, W3C maintains several documents which discuss how to apply the guidelines in general (Worldwide Web Consortium [W3C], 2013, Sept 5a), as well as with specific technologies such as HTML (W3C, 2012b) and CSS (W3C, 2012a). Unlike Section 508, the U. S. government's official standards for ensuring access to electronic materials, WCAG 2.0 are just recommendations which carry no legal status (McHale, 2011). However, they remain by far the clearest guidelines for accessible design.

HTML5

Hyper Text Markup Language (HTML) has been the basic building block of websites since the Web's inception, and efforts to replace it with newer technologies have generally not come to fruition (W3C, 2013, Oct 29). HTML5 is the latest officially recommended version of the language and includes many new features, as well as the extension of several aspects of older versions. Two of the biggest changes in HTML5 are (a) its vastly expanded selection of elements (e.g., is supplemented by <article>, <section>, and a number of other more specific options) and (b) its native support for multimedia content through application programming interfaces (APIs).

HTML has always been a semantic language, meaning that its markup carries a contextual meaning beyond simply being used for making a page display properly in the browser. HTML5 makes it easier to create meaningful markup with a number of new elements. Connor (2012) provides a useful overview of key elements, among them <section>, <article>, and <nav>, which can all be used to provide more specificity than the standard HTML4 element; <figure> and <figcaption> as a way to provide contextual explanation of an embedded image or video file; and a number of new elements related to multimedia content such as <video>, <audio>, <embed>, and

<canvas> (p. 6-8). As Connor mentions, a full list of the new elements in HTML5 can be found online at http://www.w3.org/TR/html5-diff/.

There is a world of difference in the level of support provided for multimedia content like video and audio in HTML5 and its predecessors. In older versions of the language, much of the functionality for audio and video had to be added in through third-party extensions such as Flash. Although this is still an option, HTML5 now offers native support for multimedia content with its own suites of APIs for video and audio.

Although a full explanation of how to create an accessible video or audio player in HTML5 is beyond the scope of this paper, three important accessibility concerns are as follows:

- You can use Javascript to make your video controls keyboard-accessible so that users who cannot see the screen can still access them (see Connor, 2012, p. 204-205 for an example).

- Add WAI-ARIA roles to your content for easier parsing by screen readers and other accessible technologies (see section below on WAI-ARIA).

- Always add fallback content: content that can be accessed by users and technologies that cannot process your video. Connor (2012) suggests that fallback content should be useful and may include "text and a link to an alternate accessible version" (p. 200).

In addition to explaining the principles and practice of using HTML5, Connor provides the full code for an accessible HTML5 video player (p. 211-214) as well as links to alternatives like jPlayer (http://jplayer.org) and jQuery UI (https://github.com/azatoth/jquery-video) which can be accessibly implemented with little additional work on the web designer's behalf (2012).

CSS Level 3

The use of Cascading Style Sheets (CSS) makes web design flexible and scalable. CSS allows you to separate your website's content from its layout: a core concept of progressive enhancement, as described above. Much like HTML, CSS has gone through multiple stages of development. The latest of these, CSS Level 3, splits the specification into modules, a process that the CSS Working Group argues will "[allow] more immediate, incremental improvement to CSS" (W3C 2011, Oct 14). Indeed, there are already a few level 3 modules that have received recommendation or candidate recommendation status, and one on selectors which has a working draft as a level 4 module (W3C 2013, Oct 24).

One of the most useful modules in CSS Level 3 is the media query module. As discussed above, media queries are bits of code that allow developers to specify different style options for different sized devices, so that "presentations can be tailored to a

specific range of output devices without changing the content itself" (W3C, 2012, Jun 19, para. 2). In short, if a screen reader or other form of assistive technology is being used, media queries let you see what size your end user's browser window is and display a different layout based on the result of that query. This could be used to build a section of the page which is visible only to users on very small screens with a link to the full 'desktop' layout, or an aside for users accessing the site with assistive technologies providing links to alternative, more accessible multimedia content. Figure 2 shows the syntax of media queries, along with several example queries as defined by a sample CSS stylesheet.

```
/* Extra small devices (phones, less than 768px) */
/* No media query since this is the default in Bootstrap */

/* CSS styles go here */

/* Small devices (tablets, 768px and up) */
@media only screen and (min-width: 768px) {

/* CSS styles go here */

}

/* Medium devices (desktops, 992px and up) */
@media only screen and (min-width: 992px) {

/* CSS styles go here */

}

/* Large devices (large desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {

/* CSS styles go here */

}

/* Screen readers ("aural" is deprecated in HTML5 but "speech" is not recognized by all browsers) */
@media aural, speech {

/* CSS styles go here */

}


/* Braille layout */
@media braille {

/* CSS styles go here */

}

/* Print layout */
@media print {

/* CSS styles go here */

}
```

Figure 2. Example CSS media queries, including those for non-visual media.

The biggest decision when it comes to media queries is generally where to create break points between your different designs. That is, how many different layouts do you need to create an effectively responsive website while still not creating too much extra work in terms of testing and actual coding? A quick search for "media queries" on Google makes it clear opinions on this run the gamut. Some designers recommend creating a different layout for each of the currently popular mobile devices and desktop screen resolution sizes, but this engenders a similar problem to the one Marcotte (2010) points out about creating multiple websites with similar content for different devices.

Bootstrap, an open source framework intended to make mobile web development easier and more standardized, takes the opposite approach. Bootstrap (2013) defines four main break points: below 768 pixels for phones and other extra-small devices; between 768 and 992 pixels for tablets and other small devices; between 992 and 1200 pixels for desktop monitors and other medium devices; and above 1200 pixels for any larger devices (2013). If you do use Bootstrap, keep in mind that it's designed with mobile users in mind first and foremost. Their media query set assumes that the default CSS is what will render on very small devices, and that any layout for a larger device will be coded in the media query which matches that device's size.

It's important also to reiterate that media queries are not just for screen sizes and other visual cues. They work with all the media types listed in HTML4, such as aural and Braille (W3C, 2012 Jun 19). This means you can theoretically tailor your non-visual visitors' experience of the site by making sure the various non-standard sections (e.g., chunks of code presented on the page) are pronounced in a sensible manner (W3C, 2011 Jun 7); although how well this works in practice is a matter of debate. Regardless, since the majority of non-sighted web visitors use screen reader software such as JAWS, Orca, or VoiceOver, web designers should take extra care to render their HTML itself in a manner that can be easily accessed. Connor (2012, p. 34-58) provides an excellent overview of screen readers and explains how to account for them when creating a website.

WAI-ARIA

The Web Accessibility Initiative's Accessible Rich Internet Applications specification (WAI-ARIA), which is often abbreviated as ARIA, is a suite designed to make dynamic applications, or other web content with heavily developed user interfaces more accessible to those with disabilities. The suite aims to "fill the gaps" of web content accessed in a desktop browser by providing additional information on user interface taxonomy, role, and states or attributes to assistive technologies like screen readers (W3C 2010, Sep 6).

ARIA is not itself a scripting language: it's a set of attributes that can be appended to HTML elements. For example, you might have a section of your library web page that shows the most recent book checked out, or the most recent article accessed by library users. This would likely be powered by Javascript or some other dynamic scripting technology in conjunction with a server-side scripting language that pulled data from the system's backend. For users with screen readers, the biggest problem would be that every time the section of the page updated, their screen readers would "focus" on it, disrupting whatever content they happened to be browsing (Connor 2012). By adding an ARIA attribute of aria-live="polite" onto the element that contains the script, you could let screen readers know to ignore updates unless the end user has purposely focused on the region (see Figure 3).

```
        <meta charset="UTF-8">
        <title>Nonesuch Library</title>
        <meta name="description" content="This page offers an example of WAI-ARIA in
application">
    </head>

    <body>
        <header>
            <h1>Nonesuch Library</h1>
        </header>

        <nav>
        <!-- Roles added to menu and menu items let assistive technologies know how
            <ul role="menu">
                <li role="menuitem">Main</li>
                <li role="menuitem">About the Library</li>
                <li role="menuitem">Search Books</li>
                <li role="menuitem">Search Articles</li>
                <li role="menuitem">Other Resources</li>
                <li role="menuitem">Contact Nonesuch Library</li>
            </ul>
        </nav>

        <aside aria-live="polite" aria-atomic="false">
            <!-- Create a side-bar section of the page, and let assistive technologi
interrupt the user's browsing unless they explicitly focus on this section -->

            <h2>Recently Accessed</h2>
            <!-- Script to pull recently accessed materials from the back-end goes h

        </aside>

        <section>
            <h2>Welcome to Nonesuch Library</h2>
```

Figure 3. WAI-ARIA attributes added to a sample HTML page.

Other examples of ARIA implementations include clarifying the purpose of non-native user interface items like graphical buttons, providing more details for buttons and other form elements, creating "desktop-type menus", and adding "document landmarks" to clarify the purpose of a section of content on a page (Connor, 2012). Regardless of how you use them, ARIA attributes will be ignored when rendering screen output, so have no impact on visually oriented site visitors. Adding roles and other ARIA attributes to your code can make a big difference to those using assistive technologies.

Building Your Accessible Website

The principles, techniques, and tools laid out above can be used to build a website that is as close to universally accessible as it is possible to get. The following section of the paper presents one possible roadmap for the creation of an actual web page or series of web pages. I have created a basic proof-of-concept website at http://wou.edu/~bakersc/ua_ex/ which presents the text of the following section of this paper as an accessible, responsive web document.

Start with Content

As Champeon (2003) says, one of the key benefits of using responsive web design is that it forces you to make sure your content is in place before adding design features. Since you will be adding presentational effects through CSS and other technologies later, the earliest part of your web design process should be to simply create your content. Or,

if you are redesigning a website, to strip the content of its layout and examine its organization.

Before you even open up your design software, you will need to figure out what is going on your website, where it is going, and how all of its parts relate to one another. If you have a complicated website, the content creation process may take some time as you determine what needs to go on any given web page, how your various pages connect to one another, and what your menus will look like. On a streamlined informational website, it may be as simple as typing up or otherwise inputting all the information that the page will contain and making sure it is arranged in a straightforward, intuitive manner (see Figure 4).

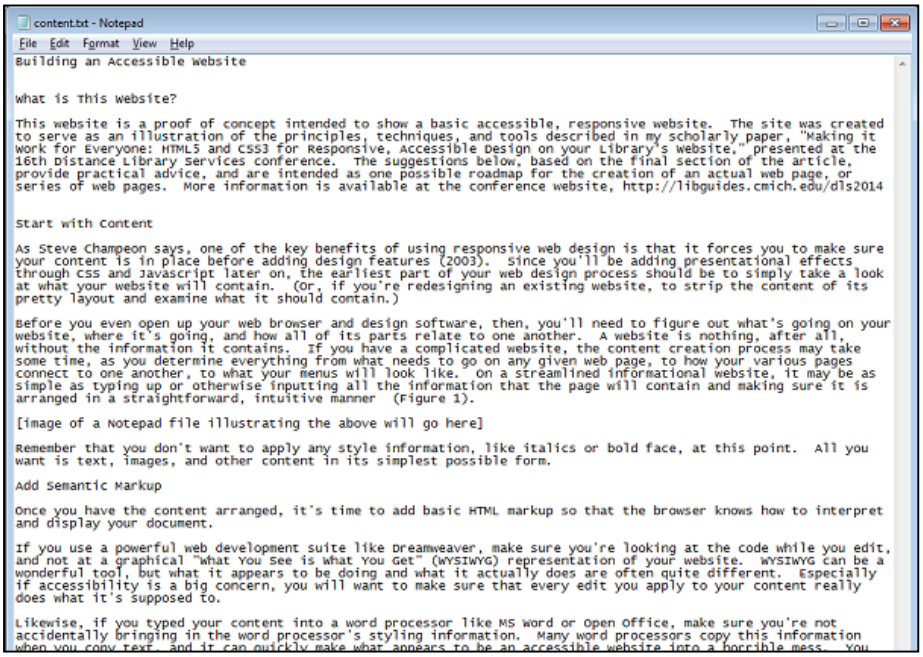

Figure 4. Content of a sample Web page in Notepad.

Remember that you do not want to apply any style information, like italics or bold face, at this point. All you want is text, images, and other content in its simplest possible form.

Add Semantic Markup

Once you have the content arranged, it is time to add basic HTML markup so that the browser knows how to interpret and display your document.

If you use a powerful web development suite like Dreamweaver, make sure you are looking at the code while you edit, and not at a graphical "What You See is What You Get" (WYSIWYG) representation of your website. WYSIWYG can be a wonderful tool, but what it appears to be doing and what it actually does are often quite different when it

comes to the resultant code. Especially if accessibility is a big concern, you will want to make sure that every edit you apply to your content really does what it is supposed to.

Likewise, if you typed your content into a word processor like MS Word or Open Office, make sure you're not accidentally bringing in the word processor's styling information. Many word processors copy this information when you copy text, and it can quickly make what appears to be an accessible website into a horrible mess. You can copy your text into a plain-text program like Notepad and strip out all the styling, just to be sure. I chose to type up my content in Notepad and then copy it into Programmer's Notepad, an open source text editor with nifty programming options available at http://pnotepad.org.

Regardless of how you add the markup, remember to take full advantage of the features offered by HTML5:

- Increased semantic tags: make it clear to anyone looking at your markup what any given piece of content does by using context-specific labels like article or section instead of vague tags like div.

- HTML-native media: if your site has audio, video, or other media, consider using the HTML5 APIs, instead of a third-party plug-in.

This is also a good time to add in WAI-ARIA attributes if you have menus, areas of the site which will be used for largely presentational purposes (e.g., banners), or intend to include heavy scripting on parts of your page.

Once you have everything marked up, you should be able to open your page in any browser and see it in its most basic form (see figure 5).

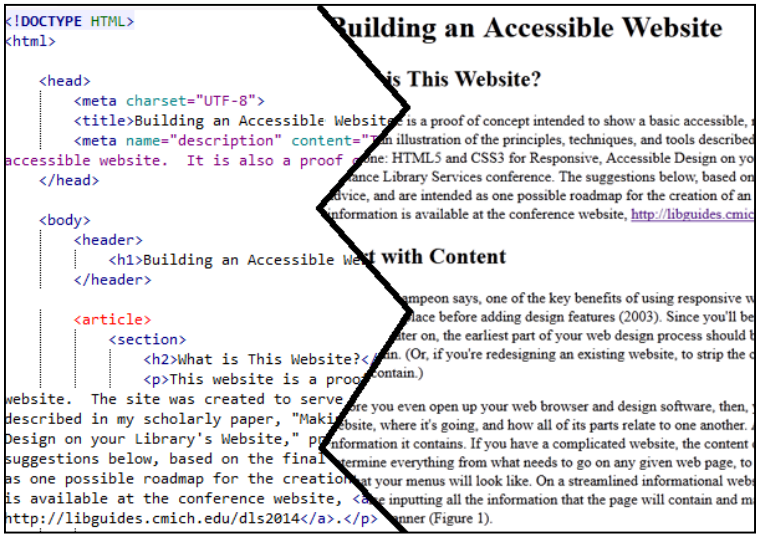

Figure 5. Sample Web page showing browser view and HTML markup view.

Style Your Site

Now that your content is arranged and marked up, it is time to add a visual design. The best way to ensure that you can easily update your site in the future is to use an external CSS file, not to define individual styles in-line throughout your document (see Figure 6). Keeping your content and its layout separated will mean that on any future updates to your site's content or design, you will not have to worry about whether a change to content will break your layout, or vice-versa.

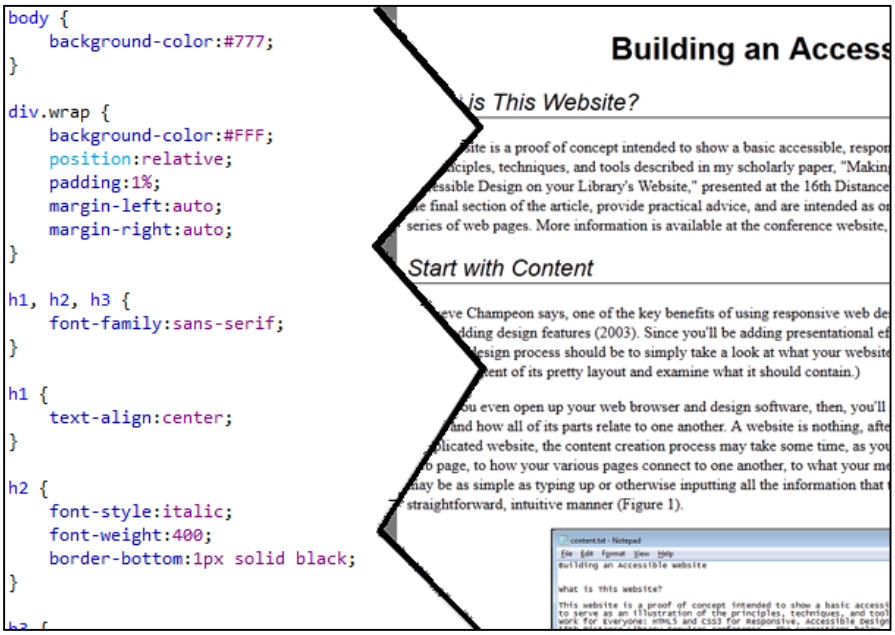

Figure 6. Sample Web page with CSS applied showing browser view and snippet of CSS.

The most important thing to remember here is that not all users will interact with your website through visual stimuli, and that even among those who do there are different thresholds for access. Avoid low-contrast color choices like black on grey or grey on white, and make sure your fonts are legible regardless of the end user's device size. You can use CSS media queries, described above, to make sure visitors to the site can experience it well on any device, and at any level of ability.

When you are done with this step, you should have a static website which is accessible to any user: a website that is, ideally, aesthetically pleasing regardless of format and device.

Add Scripting (If Necessary)

For a long time, Javascript and other technologies were seen as inimical to accessible web design, but as Connor (2012) points out, the problem lies more with "poor scripting practices" and developer apathy than any particular fault of the language itself

(p. 70). The good news here is that you can add scripts to your website to get all the bells and whistles without making it inaccessible to users with disabilities. That doesn't mean, however, you can add scripting without considering its effect on those users.

Connor (2012) lays out two principles for accessible Javascript, one of which is progressive enhancement as described above. The second is that Javascript, or any other scripting language, should be unobtrusive; which means it should be integrated into a page's functionality and should not be implemented in ways which make site controls hard to use (Connor, 2012). The bottom line is you should always be certain the added "functionality" of scripting does not make your site less functional for users with disabilities or those on older browsers or devices. If you do add scripting, remember that you can use WAI-ARIA attributes to help accessible technologies more easily parse your page.

Test

The importance of testing cannot be overstated, especially when it comes to accessibility. As books and articles on the topic make clear, just because something is written into the specification or guidelines does not mean it will actually work. As Power, Freire, Petrie, & Swallow (2012) show, WCAG 2.0 is far from easy to grasp. Moreover, several of the problems it reports do not actually affect most users with disabilities, while several real accessibility problems are not caught by it (Power et al., 2012). Indeed, the testing they carried out with a group of blind users suggested that there was no significant difference between websites written to conform to WCAG 2.0 and those which were not when it came to usability for blind users (Power et al., 2012). Although this does not mean we can safely ignore the guidelines, it does mean we cannot simply follow them and assume our websites are accessible.

Connor (2012) summarizes a number of ways to test websites, including participatory design, or including an end-user in the design process; expert accessibility audits involving an expert, third party assessment; and traditional usability testing. Each of these types of testing, and the many others, has their own pros and cons, and not all libraries will be able to do extensive testing. Regardless, web designers need to put in the effort to prove that their website is actually working, and accessible, after it has been published.

Web design in general is best approached as an iterative process, so that a website is never really "finished," it is just in its latest version. Likewise, a successful round of accessibility testing means that your website works well in its current incarnation and on current technologies; it doesn't mean you can then mark your website as finished and never revisit its accessibility. Connor (2012) suggests that in addition to making the accessible design process iterative, designers would do well to "include user involvement as early as possible in each stage of the build" (p. 293), so that you might get feedback on your content, and then on your markup, and then on your design, and so on.

The best part about testing your website like this, whether through software like JAWS or through consultations with real users, is that you will get a much better sense of what people actually want to use your website for, and how your design decisions affect real world use of your site, as well as whether or not it actually works.

Conclusion: Laziness, Accessibility, and Future-Proofing the Internet

Larry Wall, author of the Perl programming language, holds that one virtue of a good programmer is laziness or "the avoidance of future work" (Christiansen, Foy, Wall, and Orwant, 2012, p. 756). In other words, the more work you do now, the more work you save yourself, and others, later on.

The ever-changing nature of the Internet, computing, and assistive technologies make a strong case for applying Wall's virtue of laziness to accessible web design. Instead of creating more work for yourself and others by building websites that are not accessible, and that will later need updating across the board, it makes more sense to take a pro-active approach by building a site which works from the start. Although it is impossible to create a truly "future-proof" website, the techniques laid out in this paper can help make sure your website is more likely to be accessible—or at least easier to modify—for the foreseeable future. More importantly, they can make your library's website accessible today, for users of all ability levels and devices.

References

Avila, J. (2013, Jul 11). What does responsive web design have to do with accessibility? *SSB BART Group Blog.* Retrieved from https://www.ssbbartgroup.com/blog/2013/07/11/what-does-responsive-web-design-have-to-do-with-accessibility/

Berners-Lee, Tim. (1998, May 7). *The world wide web: A short personal history.* Retrieved from http://www.w3.org/People/Berners-Lee/ShortHistory.html

Bootstrap. (2013, Nov 6). "Media Queries." In *Bootstrap CSS*. Retrieved from http://getbootstrap.com/css/#grid-media-queries

Brophy, P., & Craven, J. (2007). Web accessibility. *Library Trends, 55*(4), 950-972.

Cederholm, D. (2010). *CSS3 for Web Designers.* New York, NY: A Book Apart.

Champeon, S. (2003, Mar 21). *Progressive enhancement and the future of web design.* Retrieved from http://www.hesketh.com/thought-leadership/our-publications/progressive-enhancement-and-future-web-design

Comeaux, D., & Schmetzke, A. (2013). Accessibility of academic library web sites in North America: Current status and trends (2002-2012). Library Hi Tech, *31*(1), 8-33.

Connell, R.S. (2008). Survey of web developers in academic libraries. *The Journal of Academic Librarianship, 34*(2), 121-129.

Connor, J. (2012). *Pro HTML5 accessibility*. New York, NY: Apress.

Christiansen, T., Foy, B.D., Wall, L., & Orwant, J. (2012). *Programming Perl* (4th ed.). Sebastopol, CA: O'Reilly.

Fox, R. (2012). Being responsive. *OCLC Systems & Services*, *28*(3), 119-125.

Fulton, C. (2011). Web accessibility, libraries, and the law. *Information Technology and Libraries*, *30*(1), 34-43.

Gustafson, A. (2008, Oct 7). Understanding progressive enhancement. *A List Apart*. Retrieved from http://alistapart.com/article/understandingprogressiveenhancement

Hazard, B. (2008). Separate but equal? A comparison of content on library web pages and their text versions. *Journal of Web Librarianship*, *2*(2-3), 417-428.

Hoy, M. (2011). HTML5: A new standard for the Web. *Medical Reference Services Quarterly, 30*(1), 50-55.

Keith, J. (2010). *HTML5 for Web Designers.* New York, NY: A Book Apart.

Lamb, A., & Johnson, L. (2013). Riding the winds of change: New directions for libraries and web development tools. *Teacher Librarian*, *40*(5), 58-63

Link-Rodrigue, M. (2009, Jul 21). The inclusion principle. *A List Apart*. Retrieved from http://alistapart.com/article/the-inclusion-principle

Marcotte, E. (2010, May 25). Responsive web design. *A List Apart*. Retrieved from http://alistapart.com/article/responsive-web-design

Marcotte, E. (2011). *Responsive Web Design.* New York, NY: A Book Apart.

McHale, N. (2011). An introduction to web accessibility, web standards, and web standards makers. *Journal of Web Librarianship*, *5*(2), 152-160.

Power, C., Freire, A., Petrie, H, & Swallow, D. (2012). Guidelines are only half the story: Accessibility problems encountered by blind users on the web. In *CHI '12 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 433-442. Austin, Texas: Association for Computing Machinery.

Reidsma, M. (2012, Jun 25). Responsive web design for libraries. Paper presented at American Library Association Conference 2012, Anaheim, CA, USA. Retrieved from http://matthew.reidsrow.com/articles/23

Riley-Huff, D. (2012). Web accessibility and universal design: A primer on standards and best practices for libraries. *Library Technology Reports*, *48*(7), 29-35.

Vandenbark, R. T. (2010). Tending a wild garden: Library web design for persons with disabilities. *Information Technology and Libraries*, *29*(1), 23-29.

World Wide Web Consortium (W3C). (2008, Dec 11). *Web content accessibility guidelines (WCAG) 2.0*. Retrieved from http://www.w3.org/TR/2008/REC-WCAG20-20081211/

World Wide Web Consortium (W3C). (2010, Sep 16). *WAI-ARIA 1.0 primer: An introduction to rich Internet application accessibility challenges and solutions.* Retrieved from http://www.w3.org/TR/2010/WD-wai-aria-primer-20100916/

World Wide Web Consortium (W3C). (2011, Oct 14). *Cascading Style Sheets: Levels, snapshots, modules....* Retrieved from: http://www.w3.org/Style/2011/CSS-process

World Wide Web Consortium (W3C). (2012a). *CSS techniques for WCAG 2.0*. Retrieved from http://www.w3.org/TR/2013/NOTE-WCAG20-TECHS-20130905/css.html

World Wide Web Consortium (W3C). (2012b). *HTML and XHTML techniques for WCAG 2.0*. Retrieved from http://www.w3.org/TR/2013/NOTE-WCAG20-TECHS-20130905/html.html

World Wide Web Consortium (W3C). (2012, Jun 19). *Media queries*. Retrieved from www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/

World Wide Web Consortium (W3C). (2013, Sept 5). *Techniques for WCAG 2.0: Techniques and failures for Web Content Accessibility Guidelines 2.0*. Retrieved from http://www.w3.org/TR/2013/NOTE-WCAG20-TECHS-20130905/

World Wide Web Consortium (W3C). (2013, Oct 24). *CSS current work and how to participate*. Retrieved from http://www.w3.org/Style/CSS/current-work

World Wide Web Consortium (W3C). (2013, Oct 29). *W3C HTML 5.1: A vocabulary and associated APIs for HTML and XHTML working draft* (4.8). Retrieved from http://www.w3.org/TR/2013/WD-html51-20131029/embedded-content-0.html#embedded-content-0

Zap, N., & Montgomery, C. (2013). The status of web accessibility of Canadian universities and colleges: A follow-up study 10 years later. Paper presented at EdMedia 2013, Victoria, BC. Retrieved from http://www.editlib.org/p/112322/proceeding_112322.pdf