Responsive Web Design: Much More Than Media Queries

Tony Walt | Lead Creative Technologist, EffectiveUI

Summary

The Web is littered with articles attempting to define responsive Web design (RWD), adaptive Web design (AWD) and mobile-first design (MFD). Too many of them are solely focused on how a visual design reflows with display size. Meanwhile, others advocating for a "mobile first" approach are laden with inefficient code that performs poorly on a brand new smart phone (much less a four-year-old one).

The debate is too narrowly focused. The silver bullet isn't AWD, RWD or even MFD. It's using *all* of these tools to create *good* sites that deliver a quality and consistent brand experience across a broad spectrum of platforms.

To do this, a site must be designed and developed while accounting for numerous factors beyond the visual flow. Here is an overview of the many elements to consider, including content, asset management, user experience (UX) design, visual design, development and alternate formats.

CONTENT

While many people espouse that "content is king," little attention is actually given to copy. The delivery of content is the reason most sites exist, so it should be both flexible and efficient.

Web articles are often read with limited available time and constrained screen size. Because of this, content should be laid out in such a way that users are able to easily scan articles and choose where they wish to dive deeper. The sections they choose to read further should quickly and effectively convey their concepts.

VISUAL HIERARCHY

A strong visual hierarchy, well-written modular copy and a variety of header weights throughout an article allow a user to quickly scan the main topics of each section and decide what to read.

MODULAR COPY

Modular copy allows sub-sections to more readily exist in a CMS, stand on their own or in various other article configurations, be easily syndicated, and be broken into individual screens of an article when displayed on various devices. The last option is particularly useful on mobile devices where users prefer to "drill-down" further into content rather than the scroll through extremely long screens created by limited reflow width.

EFFICIENT CONVEYANCE OF IDEAS

Regardless of the display, copy should be heavily edited to limit the amount of text users must read. This makes concepts more concise and engaging. This concept in is covered in more depth in Ginny Reddish's book "Letting Go of the Words"[1] or Strunk and White's "The Elements of Style."[2]

Consider if text is even the proper medium. A picture can be worth a thousand words, so an animation or video could be worth 29,970 words a second. Imagery breaks up large blocks of text and makes abstract concepts concrete, so they are more memorable and able to be understood more quickly.

When using visual media, ensure they are accompanied by text either through captioning or closed captioning. This allows screen readers and search engine optimization (SEO) technology to consume the content, as well as viewers who can't play audio due to their environment. Along with chapter markers, this also opens the possibility for users to skim a video and jump to specific parts of interest.

A short video clip can be a good option to replace lengthy articles full of complex concepts. Not only will it be easier to digest, but also the progress bar will inform the user exactly how much more time it will take to complete, and it can be consumed in a more passive manner.

Providing an audio option along with text also allows users to consume it passively. Users can multi-task and listen to content when it's convenient for them. This has helped contribute to the recent large success[3] of audio books.

ASSET MANAGEMENT

Before committing to media, make sure they *make sense* and you're choosing the appropriate medium. There are many times when text is the best way to go. Also, visuals that previously would typically have been delivered as an image can now be drawn via code like CSS, SVG, Canvas and others.

If media will be used, ensure the assets are well managed. Avoid inundating users with large downloads when they aren't necessary.

For images and video, choose the proper format, compression settings and delivery method. Scripts must be compressed and their delivery properly managed. Fonts must be carefully considered and used only as needed. Taking these measures results in fewer files and smaller files, and requires less space to store and less bandwidth to transfer. This saves both time and money for the developer and for the content's consumers.

IMAGE ASSETS

Images can be saved into several different formats, but PNG is often preferred. While GIFs handle solid color block images well, oftentimes an 8-bit PNG will be a smaller file size. PNGs are lossless and support transparency. They can be run through optimization programs like OptiPNG[4] and PNGcrush[5], which can reduce their size while maintaining image quality.

Use JPGs for images like photographs that contain a wide array of colors. They are much better suited for this scenario because their lossy compression can maintain a good quality image at a fraction of the size. JPGs can be compressed on a master level, or processed with programs like Advanced JPEG Compressor[6] to compress chroma and luminance channels separately, as well as on varying levels of detail.

Also consider WebP[7], a format Google created specifically for images on the Web. It reduces image file size from 25-34% while maintaining similar quality. It also supports animation, transparency and both lossy and lossless compression schema. Facebook and YouTube are two large sites that are utilizing this new format. However, only a few browsers support it, and most graphics software doesn't. Therefore, it's best to install the PageSpeed Module[8] on the webserver to take advantage of this technology.

Images that don't need to repeat, or only repeat on one axis, can be combined into sprite sheets, further reducing the size and number of images on a site. But be sure to carefully consider the structure of a sprite sheet so that it remains easy to maintain. A few properly segmented sprite sheets sometimes work better than a single large one. It is better to serve a tiny fraction of the imagery is requested, rather than the entire site's worth of images.

Because many screens have a high dots per inch (DPI) ratio, sites often have a second set of higher-resolution images for these devices. Use media queries to detect these displays and serve those images. But remember that not all users may want to receive these larger files due to device and connectivity constraints.

```
@media (-webkit-min-device-pixel-
ratio: 1.5), (min-resolution: 192dpi)
{
/* Styles targeting 1.5DPI and
above */
}
```

https://gist.github.com/tonyjwalt/9937072#file-highdpimediaquery

Most developers focus on using high-resolution images for high-DPI displays, but those images are also useful in print CSS files, giving users high-quality graphics when printing a Web page. (See the print section below for more information.)

VIDEO ASSETS

Common practice for videos is to compress to the H.264 container with an MP4 codec. This is the most versatile of the formats, and the most efficient. It will play on most devices and browsers natively. Jan Ozer is an amazing resource for learning to encode videos, and he offers a great resource for choosing encoding settings.[9] Programs like Handbrake[10] also have a useful set of presets.

MP4 can be complex. It contains three different profiles: high, main and baseline. Opting for 'high' means a better compression algorithm that will deliver better quality at a smaller file size, but requires more computation power and battery to play back. Many mobile devices require a simpler algorithm like baseline. However, this results in a larger file that increases download time, which impacts battery life.

Each device has its own limitations regarding the resolution of a video file they can play back. So while, theoretically, they can play the format, they still may not be able to play the video if its resolution is too large.

One thing to watch out for when encoding is the moov atom.[11] This is a small set of information that tells the browser about the video and allows it to play parts that have already downloaded. When producing a video for the Web, ensure the moov atom is at the front of the file, otherwise users will have to download the whole video before beginning playback. Standard video editing and encoding software will put the moov atom at the end of the file unless specified otherwise.

MP4 is also license encumbered, so there is a fee for developers if users are being charged for the content.

WebM[12] is a good alternative to MP4 because it was made from the ground up for the Web. However there has been much debate about WebM patent issues, and this has prevented it from being widely supported. It allows files to be encoded much more easily, but only a few browsers will support those files, and good encoding software is harder to come by. A preferred method for encoding to WebM is x-media recode.[13]

Consider encoding multiple MP4 files and a few WebM files for each video asset to ensure the media are as flexible as possible.

BANDWIDTH MANAGEMENT

When serving video, there are two options: progressive download (a format allowing users to play video while it's downloading) or streaming (sending just the section of video the user needs to their player when they need it). The latter method is excellent at conserving bandwidth. It also adds an extra layer of security since the user never has the full video file.

A good option for serving video is the use of a content delivery network (CDN). A CDN can easily balance bandwidth from video requests, often handling all the encodings and offering options like adaptive bit-rate streaming.

Adaptive bit-rate streaming allows the CDN to deliver small sections of video that match the bandwidth the viewer currently has available. This accommodates both limited and volatile bandwidth conditions on the viewer's device. Most video streaming services use this technique.

It is important to detect download speeds on media-heavy sites to adjust which images and videos are served to a user during a session. This allows users with higher bandwidth to get higher-quality background images. Unfortunately this can introduce caching issues if the user's bandwidth is volatile, so it's better to present smaller assets or let the user choose a quality setting initially.

SCRIPT ASSETS

It is good practice to concatenate and minify CSS and JavaScript (JS) files. This process removes excess whitespace and developer comments and combines the files. Overall, it reduces file sizes and the number of request a browser must make.

As more websites become Web applications and developers use more automated development tools, there is a trend towards single-site CSS and JS files. These large files result in a larger initial download, but once cached, speed up the remainder of a user's page requests.

It is better to logically organize the JS and CSS into a few concatenated and minified files that match sections or functionality groupings of the site. It is good practice to create a global CSS and JS file for common elements, and then supplement those with a few assets targeting subpages, or secured sections of the site. This prevents the serving of unnecessary CSS and JS code meant for the secured portion of the site to a user who only views the homepage.

FONT ASSETS

We now have access to use so many fonts on the Web. Sources like Typekit,[14] Font Squirrel[15] and Google Fonts[16] make the design choices feel limitless. However, designers should remember to use restraint when choosing fonts for a site.

Too many different font styles and weights on a site will dilute its impact. Like in print design, limit the design to a few well-chosen fonts and treatments. Doing so will also reduce the number of fonts a user must download to view the content.

UX DESIGN

One of the biggest challenges is implementing a successful UX design for an application consumed by various device types in many different conditions. Not only are the displays different, but also the goals and context of the users themselves.

A single user may have vastly different needs when holding the exact same device in different settings, or at different times of the day. A phone may be a convenient primary device for consuming content, but could also serve as an input-intensive productivity device when no other options are available.

FUNCTIONAL PARITY

Instead of removing functionality from a site or application just because the user is on a smaller device, consider changing the emphasis or location of some features. For example, for a mobile user, it may be best to minimize the browsing navigation in favor of search and filter options.

It is important to ensure the differences introduced to the layout and design between devices or breakpoints aren't so drastic that users get lost or have to learn two different systems.

MAINTAIN MODALITIES

Therefore, it is best practice to maintain mental modalities wherever possible. For example, consider transitioning from a tab system to an accordion when the screen becomes too narrow, because the concept of clicking on a header to reveal content in a display area remains the same in both widgets. For reference, review an example 'tabcordion' widget, downloadable here: https://github.com/tonyjwalt/tabcordion.

FULL DESKTOP VERSION

If opting to completely remove some functionality for a given device to simplify the application and cognitive load on the user, consider including a link to the full desktop version. It allows users to choose to access to the full experience.

ZOOM

It is best to avoid turning off the zoom ability of touch devices just to allow the use of "fixed" elements. Headers and their included navigation seem to be the biggest offender of this. Most of the time it's because reducing the screen width, and therefore reflow width, causes *very* long scrolling pages. A better alternative may be to conceal content and move towards a "drill-in" method.

If zoom must be turned off, then provide users an alternate mechanism for enlarging fonts and pictures, especially to accommodate users with poor vision or other hindrances to viewing content at the size set.

INPUTS

Users will interact with the site through various inputs, and a robust site should strive to support many, if not all, of them. Those inputs may include, but are not limited to, mouse, keyboard, touch, gesture and voice.

While a mouse is precise and works well with small hit areas, a finger lacks the same specificity and needs a much larger hit area. Keep this in mind when designing buttons, and remember that the hit area of a button doesn't always have to match the visual asset.

Keyboard integration often gets overlooked on a site, but is greatly important to users with accessibility challenges. It is also a nice feature for other users who find navigating via the keyboard faster or easier.

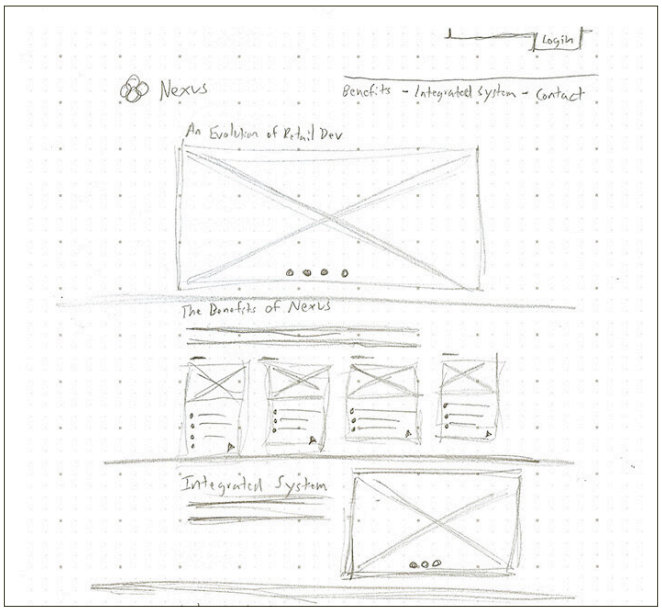
At the very least, the entire site should be navigable via only a keyboard. Using shortcut key bindings and specifying tab-index values greatly improves the experience for keyboard users. Tab-index values are set by a simple tab-index attribute, and they specify the order in which elements on a page gain focus when a user hits the tab key.

Gesture interactions on responsive sites deliver a more "native" feel on a touch-based device. While gesture-interactive areas that are targeted, like a carousel, can work well, sites may become slow or unresponsive if the gesture target area is too broad. Broad hit areas are those that encompass many different items as opposed to a specific hit area – for example, the ability to swipe anywhere on a page to navigate to another page.

VISUAL DESIGN

Most people think of 'responsive design' as visual design and how it reflows to fit on varying display sizes and aspect ratios, and it is definitely an important aspect of building a site that works well on all devices.

To effectively achieve this, it's best not to design numerous individual screens, but instead to think in terms of a fluid design framework.



Sample homepage design without borders

DESIGNING A FRAMEWORK

Begin the design process with a few individual screens, designing from the inside out. In this way, the design is not confined to a specific aspect ratio or screen size, but rather focused on the pieces and how they come together.

Once the initial designs are complete and a holistic vision is established, start extracting components, forming the basis of a design framework.

From here, the designed structures and elements combine to form additional screens, which are easier to piece together as they follow the rules established in the framework. This reduces one-off design decisions and results in the overall set of screens following a cohesive style.

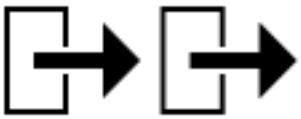
This methodology mimics object-oriented practices that are coveted for their versatility in development. In fact, designing in this method also encourages developers to follow those object-oriented coding methods.

SIMPLIFY AND SQUARE OFF

While simple, flat designs are not at all necessary, they have become very popular and many operating systems have adopted them. This isn't by chance.

Flat, squared-off designs are easier to create in code like CSS, SVG and Canvas. Drawing images in code is often the best option because such images can scale infinitely and be modified easily (even dynamically). Also, using this method reduces the number of assets the user must download.

There are, however, exceptions to this when dealing with extremely complex shapes involving numerous points. For example, when creating a halftone pattern to overlay an animation, it is much easier for the system to handle a repeating .PNG rather than draw out all of the circles.



The left icon is drawn perfectly on pixels while the right is shifted .2px to the right.

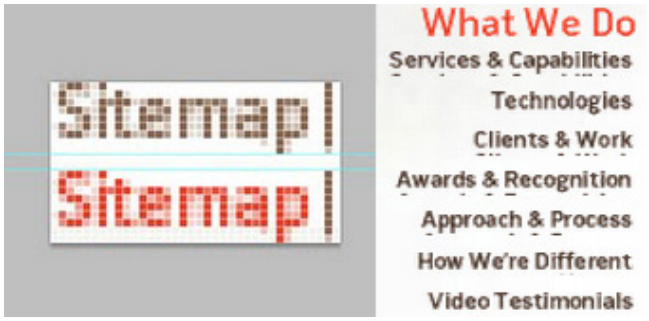
When a design is snapped directly to pixels, it appears much sharper on all displays, even at various scaled values. It also reduces the need for supersampling,[17] a memory intensive process used to calculate edge pixels that aren't precisely a single color or alpha value. In other words, it removes or reduces jagged edges ("jaggies") that may appear if the asset was not drawn on a pixel grid, or was drawn with curved edges, is scaled.

DESIGN AT 1X FIRST

When designing assets to match the pixel grid, create them at 1x before moving to the 2x (high-DPI) versions. This ensures the design assets will maintain the pixel grid as they scale up. If done in the reverse, there is the risk of having an odd-sized asset that doesn't hit the pixel grid when sized down.

SINGLE PIXEL BUFFER

It is important to leave at least a single empty pixel around all image assets. This is especially true when dealing with sprite sheets, because browsers can introduce supersampling as users zoom, causing adjacent sprites to bleed into each other, or an anti-aliased edge to get chopped.

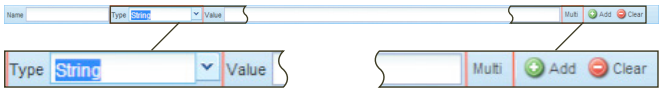


The right side shows this issue. The left side shows how it was corrected.

LARGE SCREENS

Often when dealing with responsive design, designers tend to think only of how a design behaves as the screen gets smaller. However, fluid designs can also fall apart just as easily when the screens get too large.

In particular, navigation can become too spread out, or elements that are supposed to be grouped are disassociated because of the distance between them.

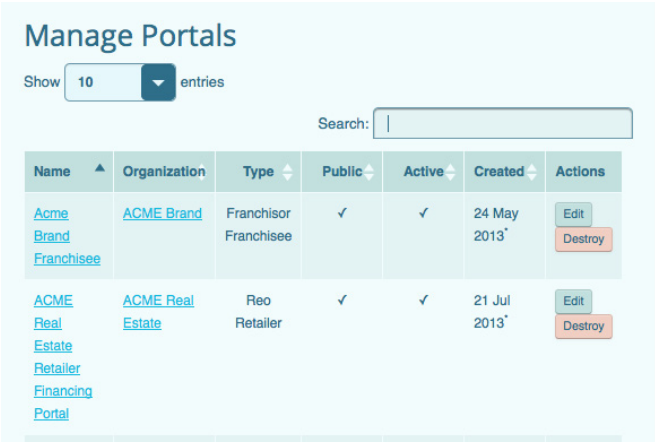


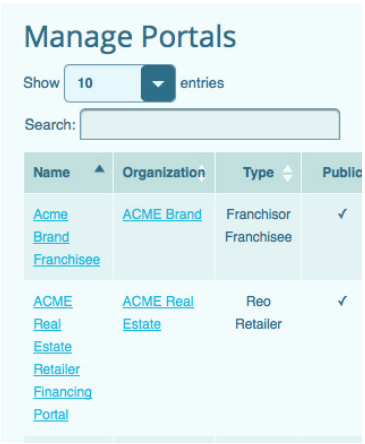
The button highlighted on the right modifies the "Type" select highlighted on the left. The larger the screen the more disassociated the two become.

A common tendency to combat this is to lock off the design at a certain size. While this is a viable option, designers should get creative and reward users for their extra display space.

TABLES

Tables are one of the main culprits for the failure of responsive design when sizing down. At some point, tables tend to lock off and stop shrinking with the rest of the design. Best practice here is to prioritize the columns and consider hiding a few as the space becomes too small.





ABOVE:
This table doesn't resize gracefully at all. It is already showing issues in the image.

LEFT:
The table is completely cut off, having lost columns.

STRESS TEST

It is important for designers to stress test their work. It is tempting to create designs with the ideal amount of content and screen size, however, once the design is publicly released, this will rarely be the case. So as a designer, it's best to look at various scenarios that could break or weaken the design.

As an example, consider how the design will handle a scenario with little to no content filling the space. Also consider what happens when a wordy copywriter injects too much text into the design. It's okay to set some ground rules around the content, but understand that at some point those rules are going to get broken.

DEVELOPMENT

When developing a site that will display on numerous devices, it is imperative to know what features will work on each of the browsers. CanIUse.com is an excellent resource for determining if a feature has enough browser support to be included.

SUPPORT MATRIX

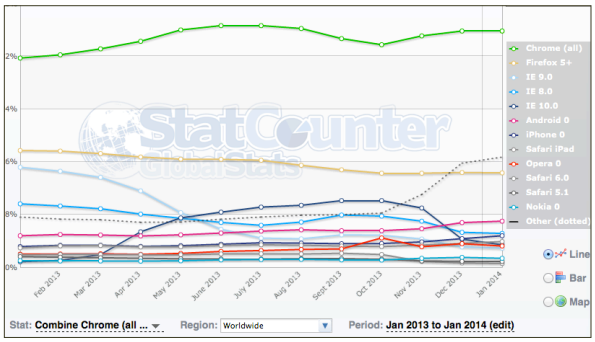
It's generally not a good idea to try to support *all* browsers, so draw the line at a logical place. Understanding the current traffic on a site and browser/device trends will help make this determination. There are two key sources for this data.

Current Site Analytics

The first is the current site statistics. This is a good place to get an initial insight into the makeup of the site's user base and how it is changing over time. However, the numbers can be misleading. A site that has poor mobile support may show low mobile traffic or high mobile drop-off rates beyond the home page, even though it may be the desired primary browser experience for the site's users. Because of this, it would be helpful to conduct user interviews to complement the analytics data.

Stat Counter

Use a stat counter[18] to gather some focused statistics about browser and device use. Google typically only supports the two most recent browser releases of a product, but most clients prefer to support additional browsers beyond that.



Tiered Matrix

The support matrix doesn't have to be a hard line; a tiered approach often works well. In this method, the site's primary browsers get the full experience while secondary browsers get a functioning experience that may simply lack some visual polish. A possible third tier may only get primary site functionality, or a message directing them to another channel for help.

It may be tempting on the business side to add too many browsers to the support matrix. Doing so is likely to greatly increase development time and cost for the site. It also can result in the use of hacks or additional bloat to the site that causes the primary user experience to suffer.

A key example of this is supporting older Internet Explorer (IE) browsers like IE8 and below, which typically requires the use of extra JS libraries and more lines of CSS code.

HTML

Conditional Comments

Fortunately conditional comment codes only work in IE, and using them to detect various versions of IE has become a standard practice. This is important to delivering IE version-specific code only to the IE users who need it, and avoiding adding bloat to the experience for other users. A typical conditional comment code block looks like this:

```
<!--[if lt IE 7]>    <html
class="no-js lt-ie9 lt-ie8 lt-ie7">
<![endif]-->

<!--[if IE 7]>    <html class="no-js
lt-ie9 lt-ie8"> <![endif]-->

<!--[if IE 8]>    <html class="no-js
lt-ie9"> <![endif]-->

<!--[if gt IE 8]><!--> <html
class="no-js"> <!--<![endif]-->
```

https://gist.github.com/tonyjwalt/9937172#file-htmlconditionals

The example above shows the assignment of various classes to the HTML tag based on the browser version. From here, use those classes to apply specific CSS.

IE Scripts

Conditional comments are also used to selectively load assets like browser-specific CSS and JS files. Here is an example of a conditional comment being used to load CSSPIE.[19]

```
<!--[if lte 9]>
  <link rel="stylesheet"
  href="scripts/vendor/pie/PIE.
  htc">
<![endif]-->
```

https://gist.github.com/tonyjwalt/9937230#file-htmlpie

CSSPIE is a great example of an IE-specific script that allows older versions of IE to work with some of the newer CSS3 styles like shadows, gradients and rounded edges. And there are many other good options for bringing older versions of IE closer to modern-day standards.

Meta Tags

Meta tags appear in the head of the HTML documents and provide meta information about that document, most commonly "description" and "keyword." The "description" tag is used to describe the contents of the page for search engines.

Meta tags are also useful for specifying how some aspects of the site should work on mobile devices. This image shows some of the mobile-specific meta tags.

```
<!-- Mobile Specific Metas
  ============================== -->
<meta name="viewport"
content="width=device-width, intial
scale=1, maximum scale=1, user-
scalable=no">
<meta name="apple-mobile-web-app-
capable" content="yes" />
<meta name="apple-mobile-web-app-
status-bar-style" content="black" />
<meta name="format-detection"
content="telephone=no">
```

https://gist.github.com/tonyjwalt/9937328#file-htmlmobilemeta

Viewport Tag

The "viewport" meta is the most important, and allows the developer to tell a mobile device what to zoom in on when the site is loaded. In this way, mobile users see the content pre-zoomed to the appropriate level instead of a tiny thumbnail of a larger site. Mozilla has a great article[20] showing an example.

The viewport tag can also turn off a user's ability to zoom by appending "user-scalable=no" to the content attribute like the image above shows. This is useful when coding fixed elements, like a header that is persistent as a user scrolls. However, be careful when choosing to do this, because some users with poor eyesight may need that ability to zoom. Either way, ensure there is a way for users to see the content at a larger size, or make the default size of text and images amply large.

Apple-mobile-web-app-capable Tag

Another useful tag is the "apple-mobile-web-app-capable" tag, which allows a site to run in full-screen mode on an Apple device. This removes the browser chrome, so it is important to ensure the

site is easily navigated without the browser's built-in "back" button.

Get more information about some mobile-specific meta tags in the Safari Developer Library.[21]

Semantic Tags

It has always been good practice to construct the HTML markup using semantic tags. This means using weighted heading tags for the headers, anchor tags for links, paragraph tags for text, etc. While this may seem logical and obvious, many sites fail to do this.

Semantic tags enable screen readers and other non-visual technology to more easily parse the page and its structure. Also, sites syndicating the content can more readily understand it and can apply appropriate styling to it.

A site built using semantic tags and properly structured will display in a clear, readable format even when associated CSS styles are removed, because the browser's built-in CSS will understand the hierarchy.

CSS

The site's CSS should reside in style sheets as opposed to inline attributes or page head tags. Style sheets are more flexible and can be cached across pages when stored in their own files. There may be edge cases where inline is necessary, but it should usually be avoided.

Preprocessors

Use a preprocessor like Sass[22] or Less[23] to write CSS files. Preprocessors are great for generating a lot of CSS quickly. They allow the use of variables and functions to automate the CSS, and also allow the minification of scripts.

Nested Inefficiency

Unfortunately, they also tend to lead inexperienced developers toward inefficient nested selectors. CSS if very fast, and this isn't typically an issue, but the more nodes there are on a page, the more this can become an issue.

Mixins

Preprocessors are important for their ability to utilize functions or "mixins" to quickly append numerous browser prefixes and other browser-specific compatible code to the CSS.

Browser Prefixes

Browser prefixes exist because support for various CSS3 features wasn't standardized when many of the browser makers implemented them. These prefixes allow us to declare CSS3 values per browser.

To generate a simple black-to-white gradient that renders well on all browsers, the CSS should look something like this:

```
.item {
  background: #000000;
  /* Old browsers */

  background: -moz-linear-
  gradient(left, #000000 0%, #ffffff
  100%); /* FF3.6+ */

  background: -webkit-linear-
  gradient(linear, left top,
  right top, color-stop(0%,#000000),
  color-stop(100%,#ffffff 100%);
  /* Chrome,Safari14+ */

  background: -webkit-linear-
  gradient(left, #000000,0%,#ffffff
  100%); /* Chrome10+, Safari5.1+ */

  background: -o-linear-
  gradient(left, #000000,0%,#ffffff
  100%); /* Opera 11.10+ */

  background: -ms-linear-
  gradient(left, #000000,0%,#ffffff
  100%); /* IE10+ */

  background: -linear-gradient(to
  right, #000000,0%,#ffffff 100%);
  /* W3C */

  filter: progid:DXImageTransform.
  Microsoft.gradient(
  startColorstr='#000000',
  enColorstr='#ffffff',GradientType=1
  ); /* IE6-9 */
}
```

https://gist.github.com/tonyjwalt/9937381#file-cssbasegradblackwhite

Information about which line of code supports which browser appears in the comments. This gradient was initially generated using the "The Ultimate CSS Gradient Generator."[24]

Fallback Values

Newer CSS3 properties like rems can be used when first providing a fallback value. Browsers that don't support the property will apply the fallback value and skip the properties it doesn't understand. Meanwhile, supporting browsers will take the last value, which should be the newer CSS3 one. This principle is applied in the browser prefix image above, and in the following sample.

```
h1 {
  font-size: 14px;
  font-size: 1.4rem;
}
```

https://gist.github.com/tonyjwalt/9937423#file-cssremfallback

Measurement Units

There are numerous units of measurement in CSS that apply to a screen display, mainly pixel, percent, em, rem and viewport units. When used together they can provide a great deal of flexibility to a design while maintaining proper consistency.

Pixel

The pixel is the most rigid of the measurement units, but it's important to note that these pixels aren't true pixels. They are a reference value, and are resized as users zoom or the pixel density on a display varies. Pixels should be the primary unit of measurement. They provide control and consistency to a design regardless of the user's display and settings.

Percent

While pixels are used for their control, percent provides fluidity to a design. This is often used on containers so that they will resize as their reference parent container does. The most common occurrence of this is column containers that are part of a grid system.

The fluidity of percentages can also be undesirable if they grow or shrink too much, so it is useful to incorporate minimum and maximum values. This can be done via min-width, min-height, max-width and max-height properties. Also, adjust percent values based on media query breakpoints.

Percentages can be a tricky because they incorporate width and padding. This means that a container with a width of 100% and padding on either side of 10px will end up 20px larger than its parent reference element. This can be avoided by setting the box sizing to "border-box" in CSS, but remember that this adjusts how padding is handled.

Em

Many responsive Web sites use ems as their preferred unit of measurement. Em measurements inherit the font-size of their parent container, so as the font-size of a container is changed, such as the body, all measurements below that using ems will adjust accordingly. In this fashion, a site can be radically resized by changing one CSS property.

Old browser versions allowed users to change their default font size. Using ems meant users could not only resize a site much like zooming does today, but it also ensured that everything stayed proportional while doing so.

However ems are quite difficult for a developer to work with due to their inheritance. A lot of math goes into figuring out the appropriate em value of an element that is more than a few layers deep.

```
<ul>
  <li>
      list item
      <a>link</a>
  </li>
</ul>
```

Example HTML
https://gist.github.com/tonyjwalt/52180ca7641beba07212#file-htmlemlist

```
body { font-size: 62.5%; }
ul { font-size: 1.8em; } /* 18px */
li { font-size: 1.3889em; } /* 25px
*/
a { font-size: .48em; } /* 12px */
```

Example CSS PX to EM Conversions
https://gist.github.com/tonyjwalt/b67a0e1ecae634d72093#file-cssemlist

Rem

Rem is a new unit of measurement that stands for root em. It behaves much like the em, but the only level of inheritance is directly from the HTML element. This provides the benefit of proportionally resizing several properties of a site based on one value, while avoiding all of the complex numbers that ems invariably create. It is best to use rems with a fallback value because they currently don't have full browser support.

Viewport

Viewport units are a fairly new unit of measurement that has somewhat spotty support; IE8 and some mobile browser versions offer no support, while IE9-10 and newer mobile browsers offer partial support. However, the current level of support is good enough, and these units can be very useful when used with fallback values.

Use a min-height value of 100vh (100% viewport height) to ensure short pages still span the height of a browser window. Previously this effort involved storing height and min-height values of 100% down a chain of elements including HTML and Body or utilizing several absolutely positioned elements. Either option can lead to excessive CSS attributes and present resize delays or bugs when too heavy.

Media Queries

Media queries allow the application of CSS rules to specific media types, and based on expressions (typically around device width, height, resolution and pixel aspect ratio). A good resource for seeing and testing the extensive list of media types and expressions is cssmediaqueries.com. And reference the CSS Media Queries article[25] on Mozilla Developer Network, about query types and expressions.

Link Media Attribute

Media queries can apply to entire CSS style sheets using a link tag, shown below. This method is best used for large sets of styles, like those for a print CSS file. It's important to note that the file will still download, regardless of whether its rules are applied or not.

```
<!-- print media query on a link
element-->

<link rel="stylesheet" media="print"
href="print.css" />
```

https://gist.github.com/tonyjwalt/c1a059f358658251a4d8#file-htmlprintquery

@Media

Media queries can be applied through the @media rule. In this way, call CSS applied to an element is kept together in the same file. This is helpful when writing object-oriented CSS because the selectors remain with the same component. The following shows an item receiving a width of 50%, but changing that value if the device width drops below 720 pixels, and the media type is "screen."

```
.test-item { width:50%; }
@media only screen and (max-width:
720px) { .test-item { width:100%; }
}
```

https://gist.github.com/tonyjwalt/8344e02c868f9aa67cd9#file-cssmediarule

@Selector

The ability to enforce CSS rules based on device width is a great step forward, but this is more appropriate for large layout structures, as stated by Andy Hume, a software engineer with a real knack for CSS. Granular items would benefit more from being able to have specific CSS applied to them based on the size of the containing element. This allows a developer to write more object-oriented CSS that is decoupled from the layout. Andy has written a bit of JavaScript that allows something just like that.[26] (But be aware that Andy's example script fires on window resize continuously, and may benefit from throttling – a concept address below.)

Breakpoints

The term breakpoint is often used to describe the points at which various CSS rules are applied. Here are typical breakpoint examples:

Min-width: 0 – this contains any rules that apply only to browsers that support media queries

Max-width: 1200px – this value can really change, but it's an upper bound at which to start locking off fluid styles

Max-width: 960px – this value represents a typical desktop browser experience

Max-width: 720px – this value is for an approximate tablet experience

Max-width: 480px – this value is for an approximate phone experience

The most common method used for responsive sites is enforcing various CSS rules based on a set of breakpoints, and using percentages to cover the gaps between them.

Device Pixel Ratio

Media queries can also be utilized to efficiently serve higher-DPI images on devices that support them. This method is preferred over a JavaScript solution because it doesn't result in the downloading of extra images or cause a flash of unstyled content.

However, it's worth noting that images delivered via a CSS background image will not print unless the user has specified that in their browser settings. Also, some users on mobile devices won't appreciate downloading twice the number of pixels just because their device supports it (this takes longer on a cell connection and drains more battery).

To deliver CSS rules based on a device's pixel ratio, use a query that looks like the image below. The 1.5 value represents be whatever pixel ratio targeted. Chris Coyier, co-founder of CodePen and founder of CSS-Tricks, has an article about this[27] on his CSS Tricks site.

```
@media (-webkit-min-device-pixel-
ratio: 1.5), {min-resolution: 192dpi)
{ /* Styles targeting 1.5DPI and
above */
}
```

https://gist.github.com/tonyjwalt/9937072#file-highdpimediaquery

Draw in CSS

Being able to serve higher-DPI images through media queries is a nice way to reward users for their high-DPI screens, but that comes at the cost of bandwidth.

If possible, add effects like shadows, gradients and rounded corners in CSS itself. This method saves on bandwidth, allows adjustments to be more easily made in code and ensures the resolution of the effect will always match the display.

Grid System

Just as designers often design on a grid system, developers can develop on one as well. This helps to maintain a responsive design that remains in proportionate sections.

The structure of a grid system is column containers and rows that contain them. The columns can either be fluid or fixed. Fluid ones will resize as the row does, while fixed ones will fit as many columns across the row as they can.

The two schools of thought for developing a grid system are using floats and inline-block. Each has its pros and cons.

The inline-block inherently adds space, and grids based on them must either use a font that doesn't have spaces or must calculate and negate that space. This can be difficult if the font or the rendering of it changes.

Alternatively, a float-based grid inherits varying float issues (like un-cleared floats), and misses out on options like vertically aligning columns.

Major frameworks like Twitter Bootstrap,[28] PureCSS[29] and Foundation[30] have grid systems. And this article's author has written a float-based grid-system, accessible here: https://github.com/tonyjwalt/grid-responsive. Some refinement may be necessary to meet individual needs.

JAVASCRIPT

JavaScript is a crucial tool for effective responsive Web design because it allows developers to adjust an application's layout and functionality based on the browser and device. Keep in mind that poor JavaScript performance will be apparent on weaker devices and can destroy a user's experience.

Browser Compatibility

This article previously explained how to determine if a user is on an IE browser, as well as which version, via HTML conditional comments. This can be very useful for catering code to old versions of IE, but they are dependent upon detecting a specific browser for support. This technique is sometimes known as browser sniffing,[31] and should be limited to some very specific use cases.

Due to the sheer number of existing browsers, with more coming all the time, it is a much better practice to understand which specific features a browser supports. Modernizr[32] is a JavaScript library that does just that, and provides developers with hook to use that information. It also has an option to include HTML5 Shiv,[33] which enables older versions of IE to properly display and style HTML5 elements.

This is extremely useful for ensuring users are provided an experience the browser is capable of delivering, and providing alternate options if it isn't.

Javascript Cross-browser Libraries

Not all events, selectors and methods in JavaScript are the same across browsers. This would typically lead to a lot of headache, testing and extra code. Luckily there are now libraries that remove many of those cross-browser compatibility issues. While they may not guarantee compatibility, they certainly go a long way toward that goal.

Jquery – jquery.com

Jquery is the most popular of these libraries, and most sites today use it. Versions below 1.x support IE6+ and versions above 2.x support IE9+.

Zepto – zeptojs.com

Zepto is similar to Jquery and attempts to mimic the same API. However, it is more targeted at mobile and modern browsers (only supporting IE10+). This allows it to be as small as 5-10K in size, as opposed to Jquery 2.x which is 84K.

When choosing to load libraries into applications and sites, be cautious about how many added. Each one will increase the size and download time for the site's users. Some libraries alleviate this somewhat by offering custom builds.

Responsive Widgets

Elements should be able to calculate their allotted space and adjust their form and functionality accordingly. It is important, however, to maintain a consistent and appropriate mental model as the element changes.

For example, if clicking on a header tab exposes related content, it should continue to do that regardless of an adjusted layout. If the mental model must be shifted, it should be heavily tested and intuitive.

Tabcordion
https://github.com/tonyjwalt/tabcordion
This widget displays as a tab system when width allows, and changes to an accordion system when it doesn't. In both instances, clicking on the header exposes related content. The widget also maintains the selected header and exposed content as it switches between views.

ListSlide
https://github.com/tonyjwalt/listslide
This is a responsive carousel of items. So the script understands if there are too few items (compared to screen width) to function as a carousel and therefore turns off. It also understands based on screen width exactly how many items to slide in order to deliver a new set of items. All of this is respectful of performance and number of event fires on resize.

PhotoSlide with Peeking
https://github.com/tonyjwalt/photoslide
This is a fully responsive, full-screen carousel that exhibits 'peeking,' showing just a hint of the previous and next images. The carousel understands if there is only one image and removes the carousel elements. It also knows to double up if there are two images, so it can properly display peeking.

Again, it is respectful of device performance and utilizes both hardware-accelerated animations, and limits function calls to a bare minimum (focusing on CSS for most of the resize).

Lastly the carousel is built so that caption HTML resides in the individual list items, keeping all slide content together and properly associated for users on screen readers.

Performance
Performance considerations for widgets are key. Poor performance can really harm an experience if the user is on a weaker device, like an older smart phone. Because of this, it is important to take special care to ensure that the JavaScript performs as efficiently as possible across all devices. There are a few methods to ensure improved performance across all devices and browsers.

Events
Scroll and resize events are two functions that developers should pay special attention to. They don't fire consistently across devices and browsers, and they can be performance killers. Some devices only fire these events upon completion of a scroll or resize, while others will fire the event every single pixel the scroll or resize traverses.

Touch-based devices fire scroll events differently in order to achieve the smooth scrolling found in many native apps. The preferred method of managing the resizing of elements like JavaScript widgets via CSS is through the use of media queries.

When opting to fire these events, it is a good idea to limit how often they fire. This can be done via JavaScript functions that will capture an event, hold it and only fire when a timer threshold is met, or upon full completion of the event (based on a delay parameter). Ben Alman, a Web developer at Bocoup, refers to these as throttling and debouncing, and he has written a JavaScript plugin[34] to handle this.

Lastly, when firing one of these events, consider storing key values in a variable that other JavaScript code can access instead of needing to fire multiple events of the same type.

Limit DOM Manipulation
Adding and removing elements in the DOM via JavaScript is expensive. It causes the browser to reflow and redraw the view. Limit the number of times needed to inject or remove an element within the DOM.

When injecting several tags, try to combine them and perform a single large injection as opposed to several small ones. Another way to avoid DOM manipulation like this is to change the visibility of a particular element.

Store Heavy Calculations
As with storing resize and scroll values, squirrel away any calculations that are expensive. For example, when storing points along a motion path so that as an element moves along the path, it can look up its position values in an array rather than recalculating the path position.

Another example of a moderately heavy calculation is locating an element in the DOM. This often manifests as $(selector) in jQuery. This can be rather expensive if done several times. So if an element is going to be located several times, it's better to store it in a local variable.

Limit Calculations in Loops

Any time JavaScript enters a loop where it will continually iterate over something, pull as much out of that loop as possible.

HARDWARE ACCELERATION

Animation and transitions can be crucial to a Web site or application. These are heavily used in native apps, and therefore can make Web-based apps feel more natural and native to a device.

These motions direct the user where to look, and better convey the message of what happened in an interface when a given action was taken. As the use of Flash has declined, much of the motion on the Web has been done via jQuery Animate. However, this is a slow and cumbersome solution.

Hardware-accelerated motion has risen as the best solution to this problem. Offloading the movements from the CPU to the GPU is much more efficient. Each hardware-accelerated element gets its own GPU layer that can be easily moved, rather than calculating every pixel on every frame..

To get an element to animate on the GPU, use a hardware-accelerated CSS3 property (primarily transitions and 3D ones) or a JavaScript (translate3d and matrix3d) that triggers a GPU layer. There are several useful sources for pre-written transitions and animations.

Transit – This JS plug-in allows animations to be called using the same API as Jquery Animate, but utilizes hardware-accelerated CSS3 animations. In certain instances, some browsers don't support CSS3 transitions, so it would be necessary to fall back to JavaScript-driven animation. http://ricostacruz.com/jquery.transit

Green Sock Animation Platform (GSAP) – This fast tweening engine can be even faster than hardware-accelerated CSS in many conditions. And they have speed tests[35] to prove it. Green Sock was initially known for their extremely efficient ActionScript animation platform. http://www.greensock.com/v12

ALTERNATE FORMATS

Many of the recent articles on responsive Web design focus solely on screen-based technologies. However, for a site to really respond across the broadest user base it must consider other formats.

ASSISTIVE TECHNOLOGIES

As sites and Web applications become more dynamic, they often do so at the expense of handicapped users. There are some basic steps to ensure these users are provided an ideal experience as well.

ARIA Tags

The Accessible Rich Internet Applications (ARIA) tags provide features such as navigation landmarks, form hints and dictating the state of various widgets. The role attribute informs the type of item an element is. For more information, reference the ARIA article[36] on Mozilla Developer Network.

Skip to Content Links

Web pages are designed with navigation and additional header content at the top of every page. Putting an anchor tag with the text "Skip to Content" that links to the content container ID as the first bit of markup on each page allows users to skip over these repeated tags.

HTML5's more semantic tags and ARIA roles also help to do the same thing, but the "Skip to Content" link is often the better choice.

This technique is sometimes misused by providing several links skipping to different sections of the page, or being hidden with the CSS display:none. Utilizing CSS in this way hides the link from screen readers as well.

Properly Hiding Elements

It is acceptable to use display:none to hide content from everyone. However, if that content should be visible to screen readers, it must be removed from the document flow and pushedt off screen, or placed in a container that clips it so that visual users can't see it. The second method is less likely to fail. Chris Coyier has written an in-depth article about this topic.[37]

Print

Print is often overlooked when it is such a simple thing to incorporate. Doing so is a pleasant surprise for users when they print a page. It will save them ink and paper while also presenting the content in a legible and intentional fashion.

Print style sheets can be incorporated by using the @media print rule or by setting the media type of a style sheet to print.

This style sheet can be fairly simple. Primarily, default styles will work as long there isn't a lot of screen-based styles spilling over. From there, maximize use of the page by checking container widths and hiding any excess elements like site navigation.

Font sizes should be set explicitly for print using point (pt) measurements. Images may also need to be sized differently and potentially replaced with higher-resolution ones. This can be done by hiding lower-resolution images and exposing ones of a higher DPI. However, this method should be used sparingly because it means more using bandwidth which could delay the time it takes a page to print.

While some users will print in color or save the page as a PDF, many others won't. Ensure the print CSS works well when a print is done in black and white and with no background images or background colors. These elements will not print unless the user's browser is set for it, which it is usually not by default.

If users are on browsers like IE6 through IE8, newer HTML5 tags won't print. Adding in HTML5 Shiv or Modernizr (which includes HTML5 Shiv) will allow these elements to print.

Hyperlinks don't work on paper, so it is helpful to use a pseudo tag like :after to expose the link on paper. A simple command like a:after { content: "("attr(href)")"; will do just that.

CSS3 expert Christian Krammer has a helpful article in Smashing Magazine called "How to Set Up a Print Style Sheet"[38] with many more print style sheet tricks.

CONCLUSION

Concepts like responsive Web design are valid and quite useful. However they only address a small part of the overarching goal of accommodating a broad swath of users and the variety of devices they employ. Focusing solely on whether to use RWD, AWD or MFD is the wrong question to ask. Instead, utilize all of these and other techniques to truly meet the needs of the users, regardless of their device or usage context.

This is not only the "right" thing to do, but also ensures a consistent, positive brand experience. A website is often the most ubiquitous touch point between a company and its users, so care should be taken to avoid negative interactions. Even a single negative brand experience can lead to much larger implications in the way users perceive that brand. This was validated in a survey Harris Interactive conducted for EffectiveUI in which 69% of respondents agreed that a poor branded mobile app experience translated to negative perception of the overall brand.[39]

Rather than asking if your company can afford to implement these tools, you will be better served asking if you can afford *not* to.

About the Author

Tony Walt is lead creative technologist at EffectiveUI. He believes in creating unique, immersive, user-friendly experiences and thrives on pushing the bounds of user interaction models while creating unexpected experiences.

Tony has worked on a variety of projects, including for the following clients: Wells Fargo, Qwest, Microsoft, The Discovery Channel, Audi, The Learning Channel, Adobe, Oakley and T-Mobile. Previously, Tony founded and ran Fusion Media Interactive (FMI), a production agency specializing in 3D, interaction and motion design, that was integrated into Effective UI in 2007. A graduate of the Art Institute of Colorado, Tony's experience includes 3-D, video, motion design, digital design and interface development.

About EffectiveUI

Navigating the most complex business problems, EffectiveUI designs and builds digital solutions for the Fortune 1000 and ambitious innovators. Driven by the belief that the interface between people and technology makes or breaks the success of every digital product, application and user experience, the company's purview spans anywhere an effective UI is needed, from smart phones to flight decks. In addition to creating award-winning digital products, mission-critical Web applications and integrated mobile apps, EffectiveUI provides its clients with strategic guidance, training and embedded experts to spur internal progress. EffectiveUI is a WPP company (NASDAQ: WPPGY). WPP is the world's largest communications services group.

References

1 http://www.amazon.com/Letting-Words-Second-Edition-Technologies/dp/0123859301

2 http://www.amazon.com/The-Elements-Style-Fourth-Edition/dp/020530902X

3 http://online.wsj.com/news/articles/SB10001424127887323854904578637850049098298

4 http://optipng.sourceforge.net

5 http://pmt.sourceforge.net/pngcrush

6 http://www.winsoftmagic.com/ajc.html

7 https://developers.google.com/speed/webp

8 https://developers.google.com/speed/pagespeed/module

9 http://streaminglearningcenter.com/articles/configuring-your-streaming-video-(for-newbies).html

10 http://handbrake.fr

11 http://www.adobe.com/devnet/video/articles/mp4_movie_atom.html

12 http://www.webmproject.org

13 http://www.xmedia-recode.de/download.html

14 https://typekit.com

15 http://www.fontsquirrel.com

16 https://www.google.com/fonts

17 http://en.wikipedia.org/wiki/Supersampling

18 http://gs.statcounter.com

19 http://css3pie.com

References Cont.

20 https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

21 https://developer.apple.com/library/safari/documentation/AppleApplications/Reference/SafariHTMLRef/Articles/MetaTags.html

22 http://sass-lang.com

23 http://lesscss.org

24 http://www.colorzilla.com/gradient-editor/

25 https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

26 https://github.com/ahume/selector-queries

27 http://css-tricks.com/snippets/css/retina-display-media-query

28 http://getbootstrap.com

29 http://purecss.io

30 http://foundation.zurb.com

31 http://en.wikipedia.org/wiki/Browser_sniffing

32 http://modernizr.com

33 https://code.google.com/p/html5shiv

34 http://benalman.com/projects/jquery-throttle-debounce-plugin

35 http://www.greensock.com/transitions

36 https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

37 http://css-tricks.com/places-its-tempting-to-use-display-none-but-dont

38 http://coding.smashingmagazine.com/2011/11/24/how-to-set-up-a-print-style-sheet

39 http://www.effectiveui.com/newsroom/press-releases/11-10-2010.php