First Edition

Six Revisions

45 Articles for Web Developers

Written by **Alexander Dawson**

Six Revisions: 45 Articles for Web Developers, First Edition

Find me on the Web at: https://alexanderdawson.com/

To report errors, send me a message at: alex@hitechy.com

Editor: Jacob Gube

Writer: Alexander Dawson

Publisher: Six Revisions

Credits: HiTechy

For technical support relating to anything on this page visit my website.

First published and showcased online through the now defunct web design and developer magazine Six Revisions, of which I was their most active guest contributor.

Printed and published in the United Kingdom and United States.

Table of Contents

The Anatomy of a Website	6
Web Languages: Decoded	24
Semantic CSS3 Lightboxes	42
Problems Using Web Validation Services	58
Sexy Tooltips with Just CSS	68
250 Quick Web Design Tips (Part 1)	79
250 Quick Web Design Tips (Part 2)	97
The Web's Undead	109
5 Web Files That Will Improve Your Website	119
A Guide on Layout Types in Web Design	135
Reductionism in Web Design	149
The Art of Distinction in Web Design	161
A Comprehensive Guide Inside Your <head>	177
Mobile Web Design: Best Practices	191
CSS3 Card Trick: A Fun CSS3 Experiment	208
Designing By Numbers: Data Analysis for Web Designers	222
The Science Behind a Single Page Website	233
Improve Site Usability by Studying Museums	251
Human Behavior Theories That Can be Applied to Web Design	268
Evolution of Websites: A Darwinian Tale	280
Privacy and the User Experience	291

100 Exceedingly Useful CSS Tips and Tricks	298
The A-Z List for Web Designers	312
Ultimate Guide to Microformats: Reference and Examples	332
Becoming a Better Web Designer	358
Ways to Horrify Website Designers	370
60 Questions to Consider When Designing a Website	378
Situational Design for the Web	389
The Importance of Historiography on the Web	399
Why IE9 is a Web Designer's Nightmare	407
Progressive Disclosure in User Interfaces	417
Effective Communication Tips for Web Designers	433
Designing for Different Age Groups	444
Smarter Web Designs: Responsive and Customizable	456
A Guide to CSS Colors in Web Design	464
5 Little-Known Web Files That Can Enhance Your Website	476
The Evolution of Internet-Enabled Devices	492
The Proxority Principle in Web Design	498
Getting the Most Out of QR Codes Using URI Schemes	508
Why We Still Need Web-safe Fonts	519
15 HTML Questions for Testing Your Knowledge	530
15 CSS Questions to Test Your Knowledge	535
Web Agility: Pushing for Performance	543

A Guide to Styling with SVG	549
Another 6 Web Files to Help Improve Your Website	555

The Anatomy of a Website

Many people find it hard to picture a website as more than a bundle of content. This often makes explaining the mixture of languages used and the way everything comes together a difficult task.

Because what makes up a website can be related and linked to the physiology of a human body, this article's comparison should help clients and beginners alike understand the complex nature of a site's creation and components.

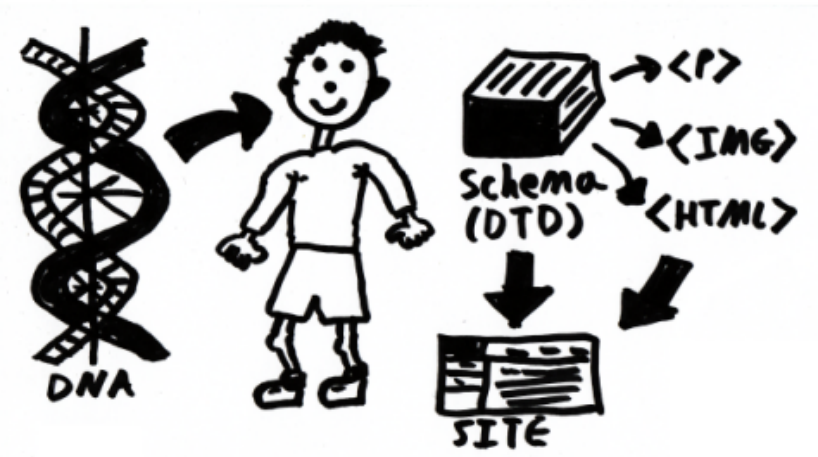
Disclaimer: It's probably worth pointing out before we start the "autopsy" that I'm not a doctor. Therefore, I recommend you don't practice this literally on your friends and family — they won't appreciate you peeking inside their ears to look for meta information!

Designer DNA: Schemas and DTDs

Humans have predefined characteristics for how we look. These building blocks of life are passed down to us through genetics, and when arranged properly, give us our unique appearance.

This process of evolution takes millions of years to adapt to changing environments and certainly plays a part in limiting both our structural and visual appearance.

In terms of the Web, the regulators of code "genetics" are commonly known as Schemas (you'll be aware of these devices from DTDs).



The human body contains DNA, just like a website. It explains how your body should react.

Of course, while the process of creating a schema doesn't take millions of years, it does take a certain length of time for new languages to appear and become widely adopted, thereby evolving the building blocks of your website.

As a result, while sites may look different, you can be assured that they only use one of a family of structural languages that predefine many of its characteristics, and what you end up with will share common elements and tags with many millions of others.

Bonus: The very inclusion of a DTD in your site can set standards for your code and avoid the obscurities that quirks mode can present to your web browser. Therefore, having this DNA, which describes the language used, can prove beneficial in inheriting Web standards.

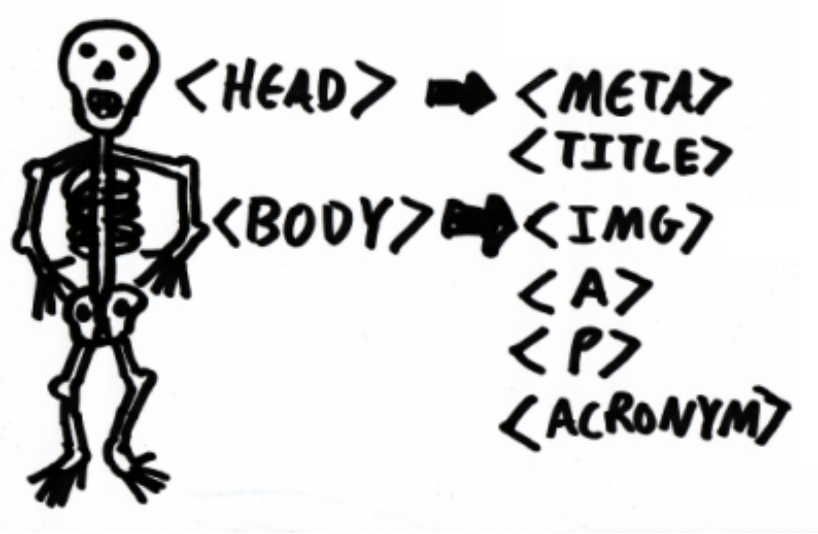
A language specification and DTD provide the genetic material all websites use and inherit.

Skeletal Structure: The Structural Markup

The structure of the human body is made up of bones that define our basic shape — the same is true of web documents in the sense that they are shaped from various interlinking elements that form the backbone of the Web.

Most web documents are formed through languages which describe the skeletal structure of the document, such as HTML and XML. RSS is also a classic example of a markup that structures a website's content.

Without these core markup languages, your website would not be able to maintain its layout.



All the parts of a website join up, like the bones that interconnect within your body.

While each bone in the human body serves a specific purpose, entire groups of bones can serve a single job, such as the ribs (each protects your lungs) or your finger bones which help you grasp objects.

Because this repeating purpose can exist within a website's body, they can be distinguished by attaching conventions like microformats that can give additional semantic characteristics and value beyond what a "generic" or reused element would offer, acting as a point of bodily recognition.

Bonus: Microformats are descriptive elements (usually as class or ID values) which give your structure some recognizable semantic values — this is much like recognizing each finger bone by its appearance and unique characteristic. It's labeling your anatomy for referral!

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="en-US">
<head profile="http://gmpg.org/xfn/11">
<script src="http://widgets.digg.com/buttons.js" type="text/javascript"></script>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Six Revisions » Web Development and Design Information</title>
<meta name="tweetmeme-title" content="" />
<link rel="stylesheet" href="http://sixrevisions.com/wp-content/themes/SixRevisions/style.css" type="text/css"
media="screen" />
<link rel="alternate" type="application/rss+xml" title="Six Revisions RSS Feed"
href="http://feeds.feedburner.com/SixRevisions" />
<link rel="pingback" href="http://sixrevisions.com/xmlrpc.php" />
<link rel="icon" href="http://sixrevisions.com/favicon.ico" type="image/x-icon" />
<link rel="EditURI" type="application/rsd+xml" title="RSD" href="http://sixrevisions.com/xmlrpc.php?rsd" />
<link rel="wlwmanifest" type="application/wlwmanifest+xml" href="http://sixrevisions.com/wp-includes/wlwmanifest.xml"
/>
<link rel='index' title='Six Revisions' href='http://sixrevisions.com' />
<meta name="generator" content="WordPress 2.8.4" />

<!-- All in One SEO Pack 1.6.7 by Michael Torbert of Semper Fi Web Design[280,300] -->
<meta name="description" content="Six Revisions is a blog that shares useful information about web development and
design, dedicated to people who build websites." />
<meta name="keywords" content="web design, web development" />
<!-- /all in one seo pack -->

<!-- Start Of Script Generated By WP-PageNavi 2.40 -->
<link rel="stylesheet" href="http://sixrevisions.com/wp-content/plugins/wp-pagenavi/pagenavi-css.css" type="text/css"
media="screen" />
<!-- End Of Script Generated By WP-PageNavi 2.40 -->
</head>
<body>
<script type="text/javascript" src="//s3.buysellads.com/ac/bsa.js" id="_bsap_js"></script>
<div id="wrapper">
```

Bones are like web page elements: They build up a logical structure that gives the body its core appearance.

Mechanical Muscles: Client-side Scripting

Being able to move allows you to interact and engage with people you meet. Without muscles, we can't turn thoughts into a reaction.

As people expect a certain level of involvement with your site, not enough interactivity could make your site appear unemotional.

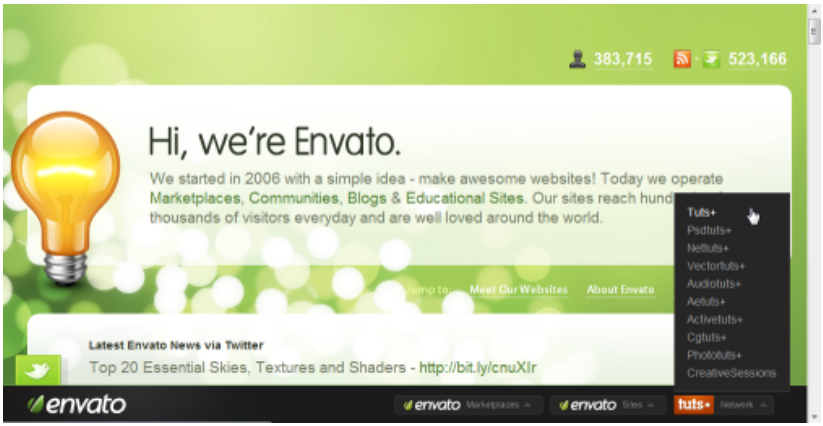
Muscles work between the skin and bones to allow both to fluidly play their part in the interaction. The same is true about sites where behaviour underpins the style and structure of a site to "flex" only when interaction is required.

When you flex your muscles, movement occurs. When a website flexes, a reaction also occurs.

Client-side scripting is the muscular component of a site. Languages like JavaScript allow interaction when visitors click, move their mouse, press a key on their keyboard or make any other noticeable gesture. This response mechanism functions just like the body in that it reacts based on its surroundings.

Simply put, the "muscles" act as a way to interact and make noticeable changes in structure (standing up rather than sitting down) or appearance (smiling instead of frowning).

Bonus: Just as humans have multiple methods of input (such as sensory mechanisms like touch, taste, sight, smell and hearing), JavaScript and other client-side scripting languages can interact and react based on its own input methods like touch, speech, automated actions and movement.



Movement and reaction are key components to both human survival and website interaction.

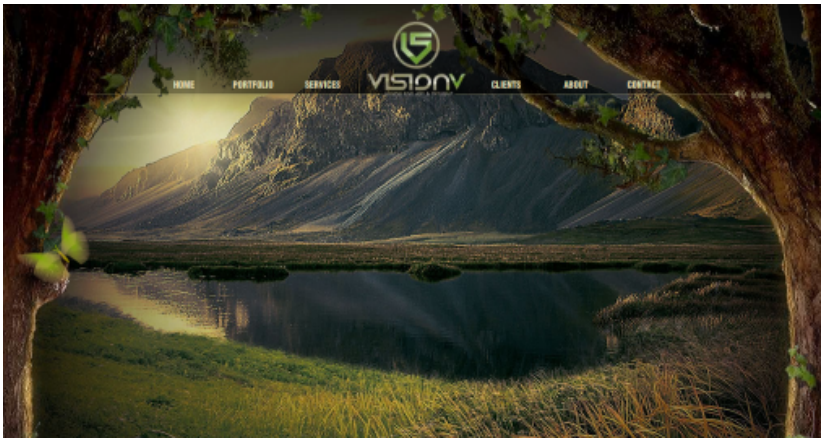
Nervous Reactions: The Web Browser

With scripting included in your website comes the need to send and receive information that acts upon the interaction occurring within your design. In a website, the mechanism of communicating these signals belongs to the user-agent or server that handles the requests and reflects those requests to act into a mechanism that is visible to the end user. In short: Your web browser works the mojo!

In the human body, such requests are sent as electrical signals that pass through the various organs and are broadcasted to conduct the action determined from the receptor, such as when you feel pain.

In a website, while pain doesn't exist (except for the end user who encounters an unusable interface element like a nasty webform) the code fires signals to the browser upon examination and triggers structure, style and behavioral reactions in turn.

Bonus: The rendering engine of a browser does everything from ensuring the sites "body" appears correctly, right down to reacting upon interaction. Even the likes of Flash, which attach themselves to a browser, have their own method of "nerve"-based interaction!



Flash websites are a great example of how information is rendered to progress effects.

The Heart: Content and the Community

The heart of a website is its content — if you've read any "Content is King" articles, you'll know what I mean.

With the human heart, a constant supply of oxygen needs to be pumped around the body to the vital organs — otherwise you'll suffer long-term damage.

The same is true for the Web, when a lack of quality regularly updated content exists, the site will become inefficient in producing visitors (the life blood of a site) and will starve (as it's abandoned), thereby giving the site's body a fatal blow.

If the heart is healthy, the pathways will be clear. If too much damage occurs, it may break.

The website's content is encased within the structure, securing it where it needs to appear.

The more pages you add, the stronger the structure — both internal and external — of the website will become. Therefore, as a result, it will become more resilient for when illness appears. The outdated but still constantly visited W3Schools is proof of this.

The balance of getting visitors (blood) around the body of the site will depend on how much depth and energy is put into a content-rich website.

Bonus: While not enough visitors can cause the heart to be starved of oxygen, too many visitors can have a similar effect. Just like when your heart works too hard, a sudden spike in traffic could make the server where your website is located collapse under the pressure, just like a heart attack!

Poor quality content of little visitor value will simply result in your community dying.

Blood Vessels: Information Architecture

As mentioned above, the ability to get people (the blood) to the places they need to get (feeding the site's popularity and architecture) is one of the key elements of building a website.

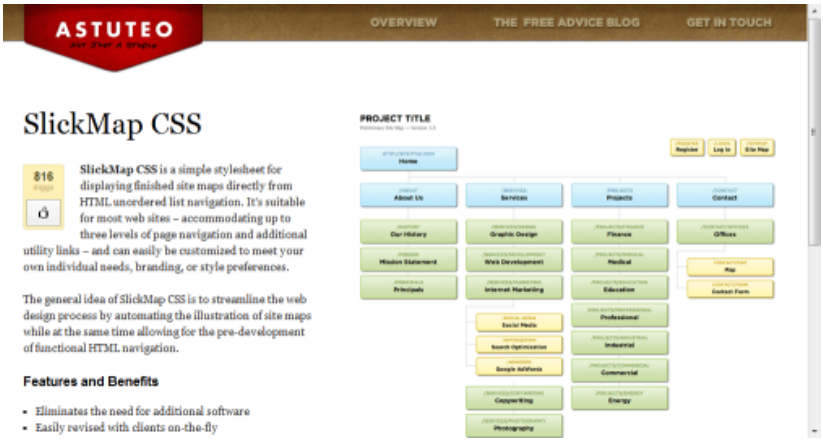
The way a site takes to interlink all the bodily elements which comprise a website is commonly referred to the information architecture. This — in simple terms — is the way we organize, structure and relate pages of a web design together (and how each page is structured in itself).

This well-organized method of interlinking the needs of various components of a website can be easily be recognized in the human body in terms of the blood vessels.

As described before, if visitors are the life blood of a website, the blood vessels would be the way we address the interlinking of pages, the findability of information and how files connect to feed each other (relative to the overall site structure).

Just as blood flows in the body, humans should flow between pages and sections.

Bonus: Usability and accessibility have their place in this analysis. When dead links or poor navigation occurs, the damage caused can hemorrhage visitors (as they leave your site). The case of where a button cost millions of dollars shows this relative link in full effect!



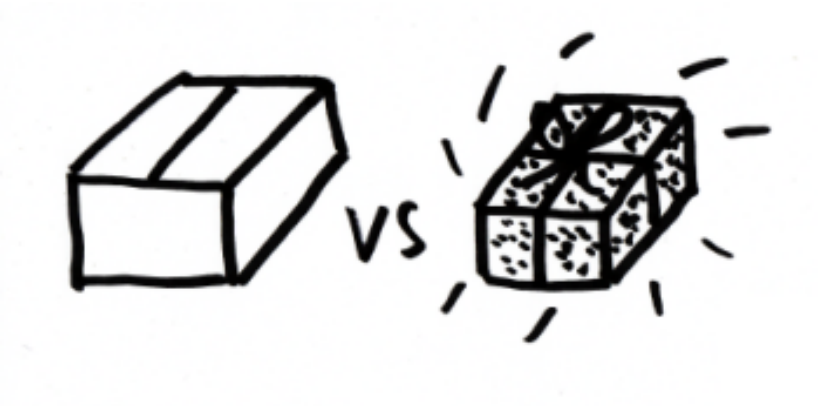
If pages don't link together appropriately, it can result in a loss of your sites visitors.

Sexy Skin: Aesthetics and Web Design

What makes people look human? Well, part of it is their features such as their eyes, nose and mouth, but their skin and visual appearance play an important role.

Whether someone is tall, short, fat or thin, the skin adopts the form, and is fairly elastic and flexible in how it covers all of the structures of the body.

While you'll want your site's skin to appear beautiful (as design is as relevant as human beauty) you also want to make sure that nothing "hangs out", looks ugly, or out of place!



Things always look prettier with skin attached; your website is no different!

In terms of web design, the primary language dealing with the style of a website is CSS. This attaches itself to your structure and layers on elements of style, which give the visible physical appearance you desire.

If you want your site's skin to be neon green and flashing pink, though not advised, it is entirely possible!

Just like in humans, the skin is simply the outer layer that works with the internal elements but has its own unique method of affecting how the body will look to the visitor.

Bonus: Unlike humans, a website design can alter its own physical appearance dynamically. Techniques include using behaviour like JavaScript to alter style on demand. This gives sites a chameleon-like quality because they can adapt to their conditions or surroundings.

When the skin is wrapped around something, it feels like an entirely different creature.

Brain Retain: Server-side Scripting and DBMS

The thing in your head allowing you to think, remember, and behave according to your surroundings, is the closest thing to a computer you have.

The brain allows decisions — the best method of dealing with complex situations — to be made.

How does this apply to a site? There are three things that must be taken into account which apply directly between human and website anatomy. These are referred to as the acts of behaviour, memory and identity.

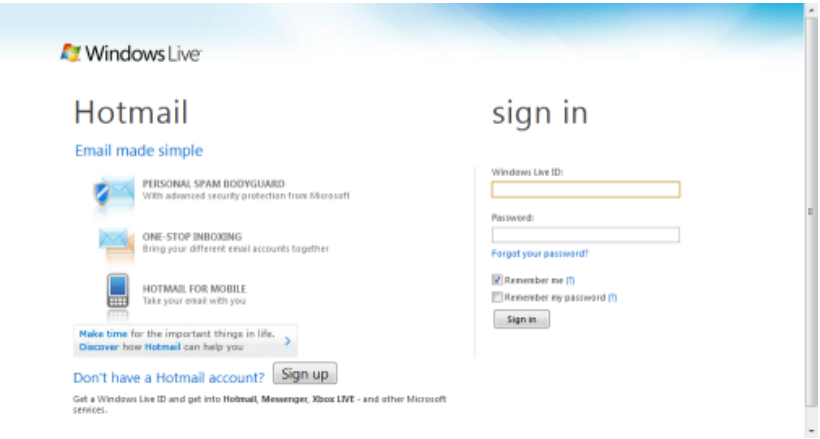
The brain handles a lot of stuff, even your website's server has to take in everything it's told!

When we talk about the behaviour of the brain, I am referring to the things we think about doing and then override our bodies to achieve, such as when you instruct your body to punch someone (thereby carrying out the punching action).

While we have already mentioned behaviour in terms of muscles, you should remember that you do not think (actively) about breathing or walking — you go through a methodology of stimulus and response (and the brain controls reactions which occur).

Bonus: Server-side scripting for dynamic websites showcase a site's brain at work, allowing your website to make decisions based on a situation (or previous knowledge as held in storage) and act upon them rather than automatically carrying out an action (like muscles).

The brain has the ability to remember many thousands of things, and computers can do the same. When you store information within a database, it holds the information until it is requested, deleted or damaged (this acts the same as a human's long-term memory). This information is usually organized relative to what type of information is contained, and can be easily searched or referenced to access the information without suffering digital amnesia.



Servers have to relay the content you input to a place where it can be recalled, it's just like human memory.

Of course, everyone also has short-term memory: Both cookies (which retain stimulus specific data like usernames and passwords) and browser caching (which contain re-usable more generic visual stimulus — like images and client-side scripts) hold the purpose of storing small pieces of information for a limited period of time.

With both long and short-term memory that can be retrieved and used until deleted or committed to a permanent and secure form of memory, it's easy to see the human resemblance.

Bonus: Of course, the human brain is much more powerful than that of current computer systems, but the diversity of information management, memory and organization gives computers a strong relationship in the likeness the human anatomy has to websites.

Last is the act of identity: being able to know who, what and why you are.

We all have our own personalities and this is something we take for granted, but sites can have their own unique sense of self in the form of metadata.

This information is held in the head (or as I call it, "thinking" code) of each page which is not visible to visitors but explains to search engines, browsers or applications wanting to associate with the contents contextual or semantic value, just like in a reference library.

Your head does the thinking and your body visibly reacts, just like your website.

Bonus: While a site's identity may be produced by its title and meta information, the actual information and abilities which a website contains are ultimately what determine the real nature of a website (sort of like humans having personalities reflected in their appearance).

Nature versus Nurture

While all of the above may help you explain the process of web design to clients in a way they can understand (or perhaps just give you something fun to pass around the office), it's important to know that lessons can — and should — be taken from the relative interlinking that a site can have with how the human body is formed.

A site, just like humans, can suffer imperfections. Some can be overcome, some can't (without a total redesign), and therefore care and attention should be given to helping overcome problems of significance.

The evolution of a website can equally be put down to a mixture of nature (what the coder puts into the site) and nurture (what the end-user adds with growth), both of which have significant importance and shouldn't be ignored.

A website's survival depends on many things working perfectly in synchronization. I think most of us underestimate how complex rendering a website is. Frequent testing can spot early onsets of problems.

But the most important thing to consider is that websites (just like humans) are formed of many layers, all interacting together, to which you need to apply as required (without making the site obese).

Think of your website like a child, you don't just feed it once and abandon it. The child requires time, money, effort, care and attention to keep it healthy and help it survive into adulthood.

Websites are like humans, they have layers, all interacting and working in synchronicity.

I think it's a fair comment to make that while standards and the way we build websites evolve, the need to make our work less static and more interactive and dynamic will aid our continued progression towards the next level of the Web. Who knows, in a few years we could yet again find ourselves becoming even more involved and emotionally tied into our brands than we are today!

Sources:

- https://en.wikipedia.org/wiki/XML_schema
- https://en.wikipedia.org/wiki/Document_Type_Definition
- https://www.w3.org/QA/2002/04/valid-dtd-list.html
- http://microformats.org/
- http://www.w3schools.com/
- https://www.lipsum.com/
- http://cybernetnews.com/one-button-costs-google-110-million/

- https://www.astuteo.com/slickmap/
- http://alistapart.com/
- http://login.live.com/

Web Languages: Decoded

Those of us who have become well seasoned to the dyslexia-inducing array of web languages often overlook the diversity and additional interactivity we can gain by learning another language or two.

Perhaps you are a beginner trying to understand what you need to spend time learning, or perhaps you're an experienced individual looking for something new to play with.

Whichever situation applies to you, this article aims to underline the various languages at your disposal and where they fit in the puzzle.

It should be an interesting ride and seasoned experts may find languages they've not yet encountered!

Too Many Cooks

When it comes to the diversity of coding for the web, the well-known phrase "too many cooks spoil the broth" springs to mind. Not only in the way browsers support modern standards but the ever increasing range of competing formats that exist.

Should you use HTML or XHTML, RSS or Atom, PHP or ASP.NET, SVG or VML, JavaScript or VBS? The list is near endless.

I wouldn't blame anyone for getting confused at this point because the question itself often relies on personal preference — something you probably won't have established until you understand the language and use it regularly.

This Catch-22 situation is usually only resolved when experienced people come together and educate each other on why their language is better the others (or as I like to call it, a flame war).

Enough abbreviations to make an English scholar tremble (and give you a migraine).

What Web Language Should You Learn?

So what is the answer you seek, to which languages you should learn? The simple answer is... it depends.

The deciding factor is not only reliant on the type of site you are trying to produce but also the depth of complexity you wish to delve into.

Therefore, before we can hope to determine which languages you should choose, we need to categorize all of the available options based on their purpose.

At first, this may seem like a complex task, but luckily for us, web languages are well-documented and each explains its purpose in context to the various existing layers of the web.

Every language has a specification which explains its purpose and function.

While documentation for each language exists, picking those worthy of your consideration and how they relate to each other in the function they undertake is something beginners regularly struggle with. You literally have to wade through the heavy number of choices and select one that you looks interesting and useful to you.

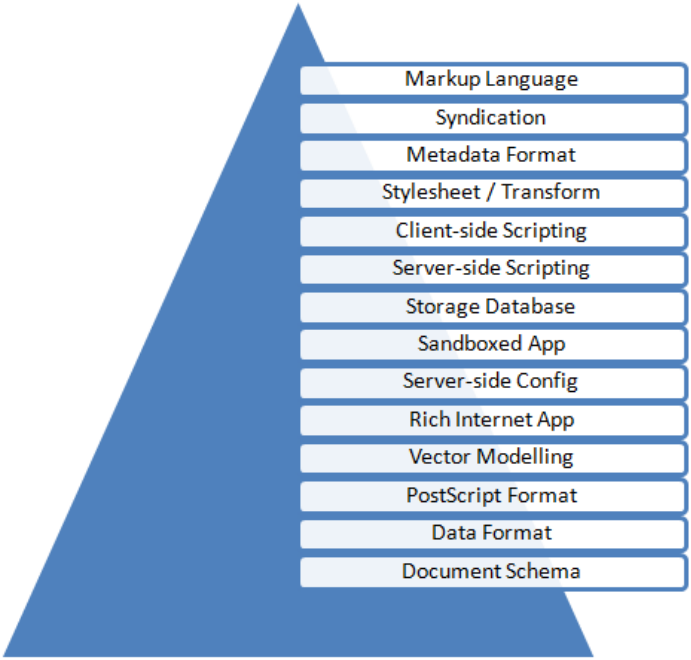
While I won't even attempt to individually explain each language's history (as this article will become encyclopedic in length at that point – instead, I'll link up to their specifications) we shall coordinate the choices into easy-to-recognize segments, and from there, you can decide which choice best meets your needs.

Language Layers

So how many language layers exist? Most people recognize layers like structural markup (HTML), stylesheet (CSS), client-side scripting (JavaScript) and server-side scripting (PHP) but you may be surprised to hear that if you account for every variant based on its intended purpose, a whopping 15 different layers exist.

Of course at this stage it's worth pointing out that learning 15 languages to cover every possible layer isn't going to be in your best interests as you'll simply be spending all your time learning, but

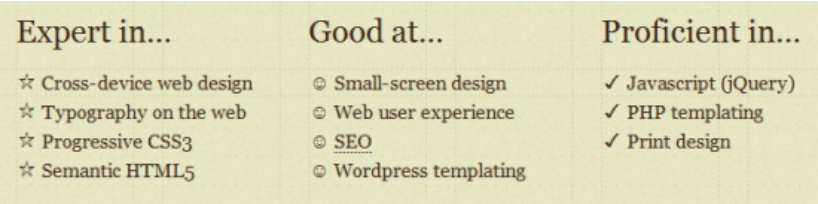
learning a new layer as you need the skill can be of genuine benefit to you.



There are 15 language layers which comprise the full spectrum of web development.

Each layer represents a unique piece of functionality such as the ability to add behaviour that interacts with the end user (in the case of JavaScript) or a method of providing dynamic vector graphics on the screen (in the case of SVG).

Having the knowledge to experiment and implement the various independent layers will give you an advantage both in operating as a professional (that you will be able to cover a wider range of skills) and as a hobbyist or newbie (where you can jump for joy at the extended level of fun you can have while experimenting with the web's offerings).



Having a wide range of skills will showcase your ever growing competency.

Of course, while these layers constitute endless possibilities of functionality which can be injected into your website, some people may simply require only a couple of these layers to produce a basic website (such as HTML and CSS).

There is absolutely nothing wrong with limiting or making a niche for your skills and become a master of either a single layer (and language choice).

This article simply acts as a starting point to which you can examine your current level of knowledge against the wider array of web languages (to determine what you can follow on from – if you know any, that is).

Sometimes a simple website requires nothing more than a couple of languages.

Collective Choices

In the language layer diagram that you saw earlier, it became apparent that there's a whole bunch of layers which comprise a website's unique structure, but as it currently stands, it's not much use to you as there's no actual languages listed.

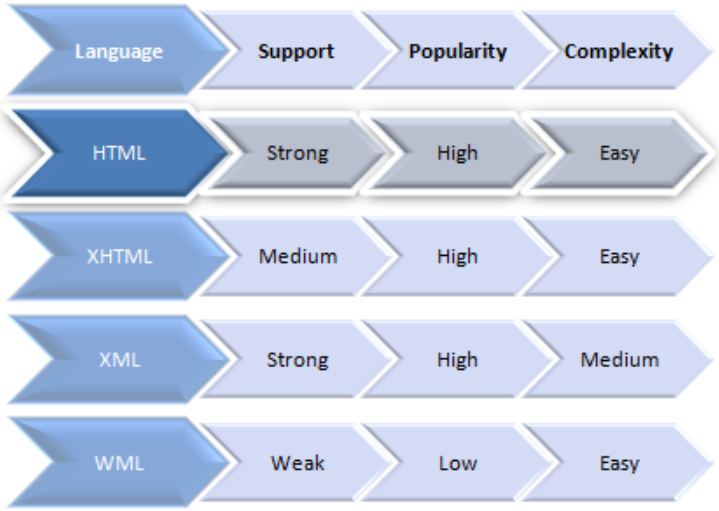
Well this is where all of those awesome abbreviations come in.

For each language layer that exists, you will find (below) a list of the languages which fit the category, their browser support levels, popularity status (useful for seeking help) and details such as the level of complexity involved, and some recommendations based on that information (highlighted from the other languages).

Note: Some languages are a subset of another language listed and some languages may be derived off of a certain implementation, but due to public awareness they have been referenced separately.

It's also worth noting that recommendations are provided based on my own experience, and so there may be some contention on some points depending on who you talk to.

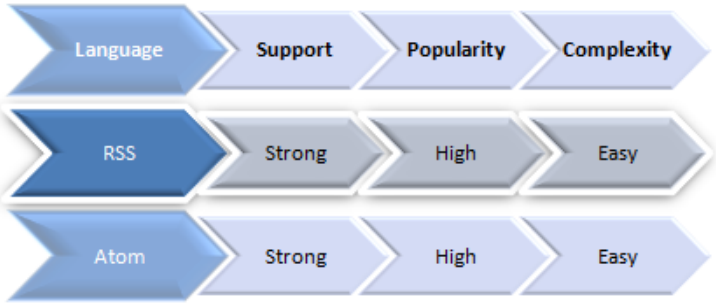
Markup Languages



There are four markup languages involved in web development.

- HTML - https://www.w3.org/TR/html5/
- XHTML - https://www.w3.org/TR/xhtml1/
- XML - https://www.w3.org/TR/xml/
- WML - http://www.wapforum.org/what/technical.htm
- Others: MHTML and SGML

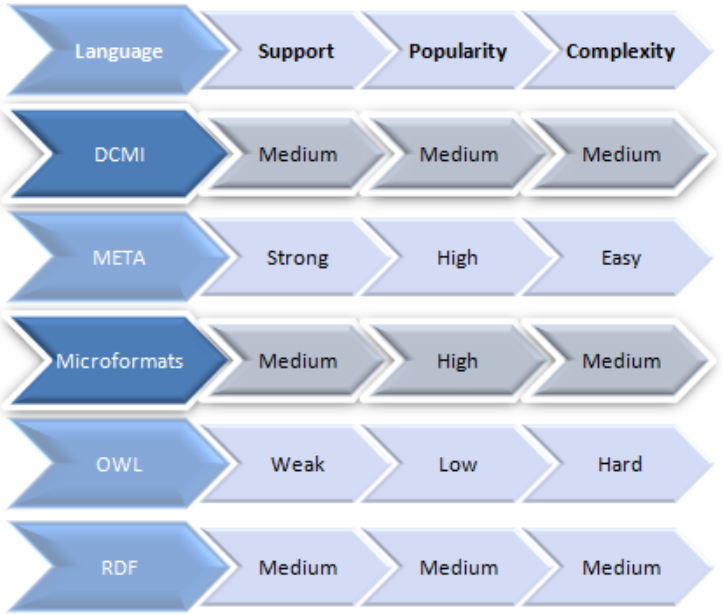
Syndication Languages



There are two syndication languages for content delivery.

- Atom - https://tools.ietf.org/html/rfc4287
- RSS - http://www.rssboard.org/rss-specification
- Others: EventsML, GeoRSS, MRSS, NewsML, OPML, SportsML and XBEL

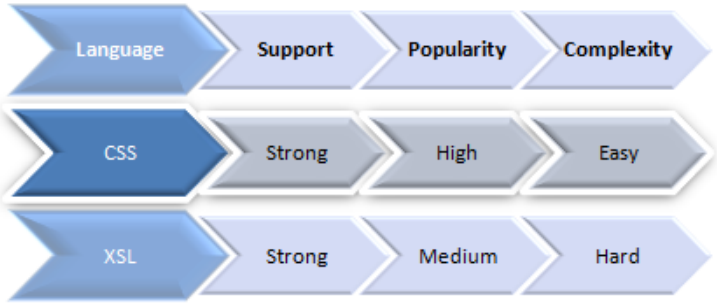
Metadata Languages



There are five methods to embed rich contextual metadata.

- DCMI - http://www.dublincore.org/specifications/
- META (Classic) - https://wiki.whatwg.org/wiki/MetaExtensions
- Microformats - http://microformats.org/wiki/Main_Page#Specifications
- OWL - https://www.w3.org/TR/owl2-overview/
- RDF - https://www.w3.org/RDF/
- Others: APML, FOAF, hSlice, OpenService Accelerators, P3P, PICS (Deprecated), SIOC and XFN

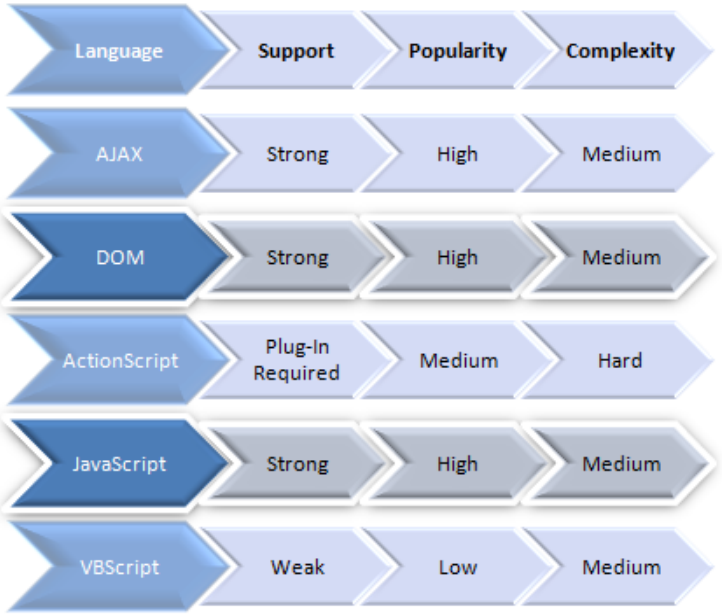
Stylesheet and Transform Languages



There is a single stylesheet language and a single transformation language for the web.

- CSS - https://www.w3.org/TR/CSS2/
- XSL - https://www.w3.org/TR/xsl/
- Others: DSSSL and JSSS (Deprecated)

Client-Side Scripting



There are a number of client side languages, though most are connected to JavaScript!

- AJAX (XHR) - https://www.w3.org/TR/XMLHttpRequest/
- DOM Scripting - https://www.w3.org/DOM/
- Flex (ActionScript) - https://help.adobe.com/livedocs/specs/actionscript/3/
- JavaScript - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- VBScript - https://msdn.microsoft.com/en-us/library/t0aew7h6%28v=VS.85%29.aspx
- Others: E4X, ECMAScript, JScript, JScript.NET and WMLScript (Deprecated)

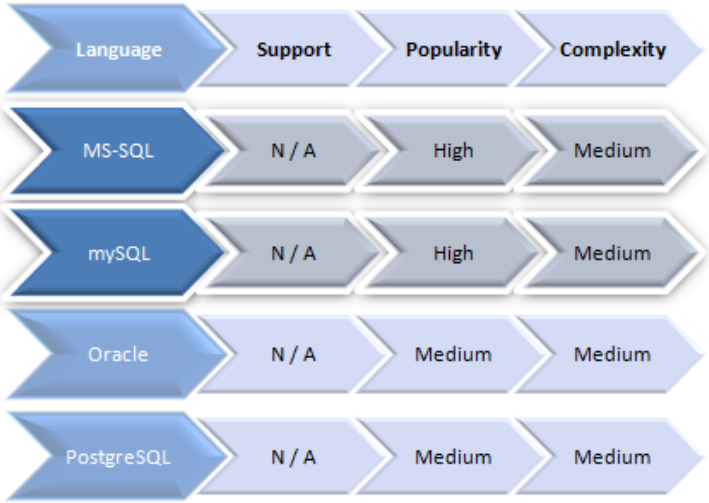
Server-Side Scripting

There are a huge number of server-side languages to choose from.

- ASP - https://msdn.microsoft.com/en-us/library/aa286483.aspx
- ASP.NET - https://www.asp.net/get-started
- ColdFusion - https://help.adobe.com/en_US/ColdFusion/9.0/Developing/index.html

- JSP - https://jcp.org/en/jsr/detail?id=245
- Perl - http://perldoc.perl.org/
- PHP - http://php.net/manual/en/
- Python - https://docs.djangoproject.com/en/2.0/
- Ruby On Rails - http://guides.rubyonrails.org/
- Others: Lasso, OpenLaszlo, Smalltalk, SMX, SSI and SSJS

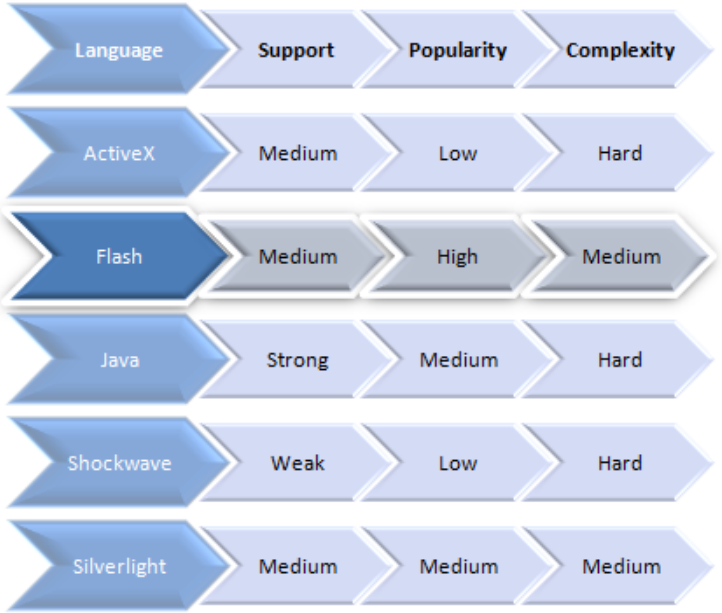
Database Management Systems



There are four popular SQL-based relational databases worthy of consideration.

- MS-SQL - https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017
- mySQL - https://dev.mysql.com/doc/
- Oracle - https://docs.oracle.com/en/database/
- PostgreSQL - https://www.postgresql.org/docs/
- Others: Derby, MongoDB and SQLite

Sandboxed Languages



The likes of Flash and Silverlight run in an independent sandboxed environment.

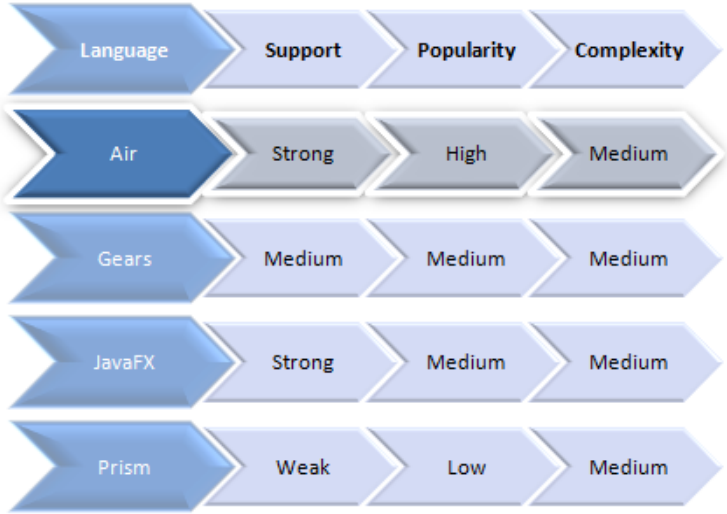
- ActiveX - https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/aa751972(v=vs.85)
- Flash - https://www.adobe.com/devnet/flashplayer.html?view=gettingstarted
- Java - https://docs.oracle.com/javase/specs/
- Shockwave - https://helpx.adobe.com/director/topics.html
- Silverlight - https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/mt788662(v=msdn.10)

Server-Side/Web Server Settings

There are two server languages and one search engine language file!

- .htaccess - http://httpd.apache.org/docs/current/
- Robots.txt - http://www.conman.org/people/spc/robots2.html
- Web.config - https://msdn.microsoft.com/en-us/library/ff400235.aspx

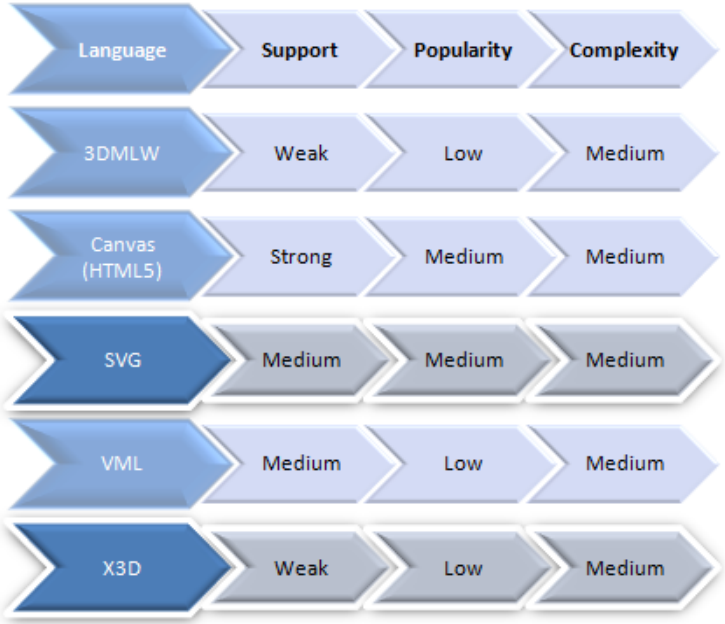
Rich Internet Applications



Several frameworks will take your web applications to the desktop.

- Air - https://www.adobe.com/devnet/air.html
- Gears - http://gears.google.com/
- JavaFX - https://docs.oracle.com/javase/8/javafx/api/toc.htm
- Prism - https://developer.mozilla.org/en-US/docs/Archive/Mozilla/Prism
- Others: Cappuccino, Curl and Titanium

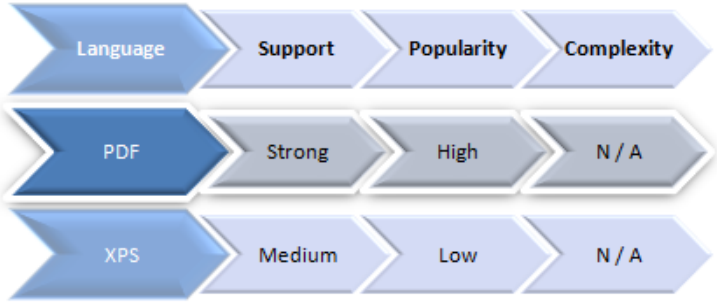
Vector Modeling Languages



There are 5 popular modeling languages for both 2D and 3D graphic and chart rendering.

- 3DMLW - https://en.wikipedia.org/wiki/3DMLW
- Canvas (HTML5) - https://developer.mozilla.org/en/docs/Web/HTML/Element/canvas
- SVG - https://www.w3.org/TR/SVG/
- VML - https://www.w3.org/TR/NOTE-VML
- X3D - http://www.web3d.org/what-x3d-graphics
- Others: 3DML, 3DXML, SMIL, UML, VRML and XVRML

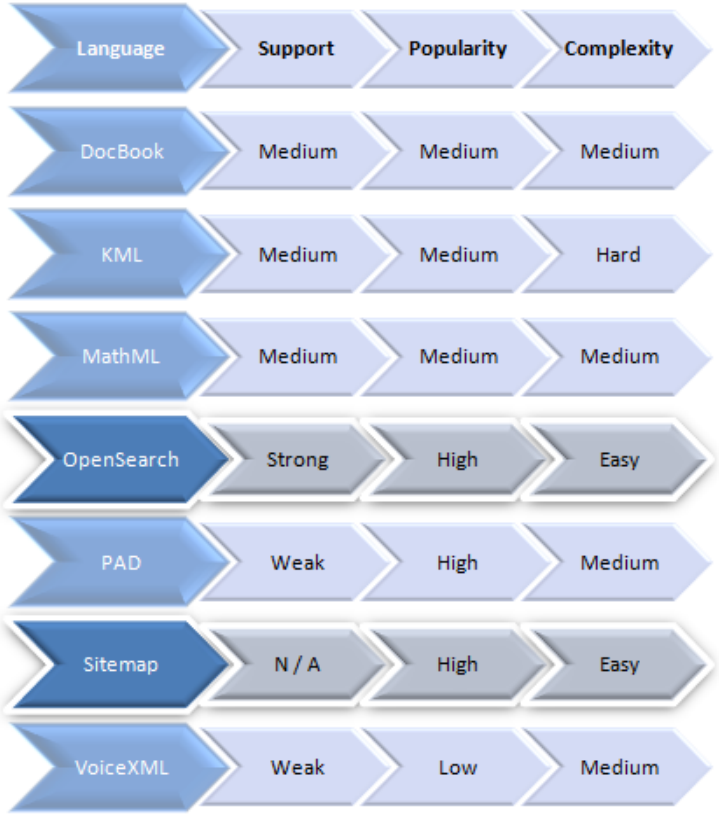
PostScript Format Languages



There are two web compatible PostScript formats for document manuscripts.

- PDF - https://www.adobe.com/devnet/pdf.html
- XPS - https://msdn.microsoft.com/en-us/library/windows/desktop/dd145058(v=vs.85).aspx
- Others: FlashPaper and OpenXML

Data Formatting Languages

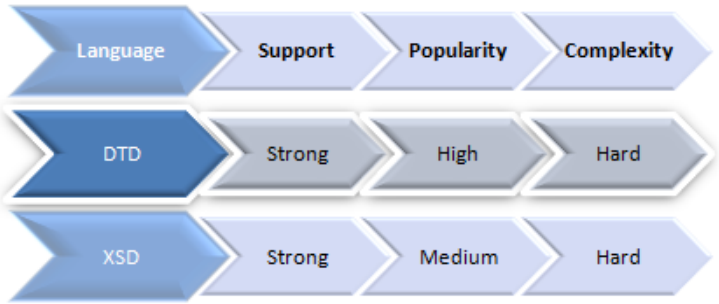


All of the above languages formats of various different mediums.

- DocBook - https://docbook.org/specs/
- KML - https://developers.google.com/kml/documentation/kmlreference
- MathML - https://www.w3.org/TR/MathML/
- OpenSearch - http://www.opensearch.org/Home
- PAD - http://pad.asp-software.org/spec/spec.php

- Sitemap - https://www.sitemaps.org/protocol.html
- VoiceXML - https://www.w3.org/TR/voicexml20/
- Others: DOAC, DOAP, GML, GraphML, InkML, OpenMath, SISR, SRGS, SSML and XMLTV

Document Schema Languages



There are two popular schemas for rendering markup languages on the web.

- DTD - https://www.w3.org/QA/2002/04/valid-dtd-list.html
- XSD - https://www.w3.org/XML/Schema
- Others: DSD, RelaxNG and Schema XML

A Caveat about Comprehensiveness

While only languages that are well recognized and have a reasonable level of support are provided in this list, it's worth pointing out that many others exist and could perhaps be worthy of inspection if you feel the need to dive into something a little more obscure and possibly interesting.

Going Forward

Now that you are aware of the sheer multitude of options at your fingertips, I'm sure you'll want to run off and investigate these languages further and perhaps learn another skill that you can add to your resume (or further enhance the way you approach building a website).

Just Starting Out?

If you are a beginner to the whole web design/development scene, my general advice would be to stick to one of the languages I have highlighted as recommended in each category (if you feel that researching for yourself would just confuse you) and to follow through each layer in order as it's been listed to give you a general roadmap to cover whatever you may wish to employ (HTML is a great starter language).

Depending on the needs of your website, you can wrap additional layers around it.

It's worth making a note at this point that the order, breakdown and recommendations provided are simply my interpretation of how the task of finding a new language can be undertaken.

Factors That Can Affect Your Choice

It's important (before we leave this discussion) to underscore the general point that your choices for each layer should be made upon a mixture of intended compatibility (so you know your choice will work for your visitors), relevancy (if the language is current or deprecated) and what you feel comfortable using.

Don't Sweat It

Beginners to the world of web design/development should remember that even the most experienced gurus of the web started from the bottom and worked their way up one skill/language at a time.

If you feel that all the choice at your disposal is too much for you, simply follow my recommendations and you'll not go far off course. While all of the considerations and lists may seem like a lot to take in, the great thing about coding for the web is that you only need a couple of languages to get started. The rest of your knowledge can evolve over time and evolve as you add more layers to your work.

Hopefully, you've been given the inspiration and pathway to perhaps look beyond the conventional HTML, CSS and JavaScript towards languages you might have never considered before, and I look forward to seeing what comes of your learning a brand new exciting web technology!

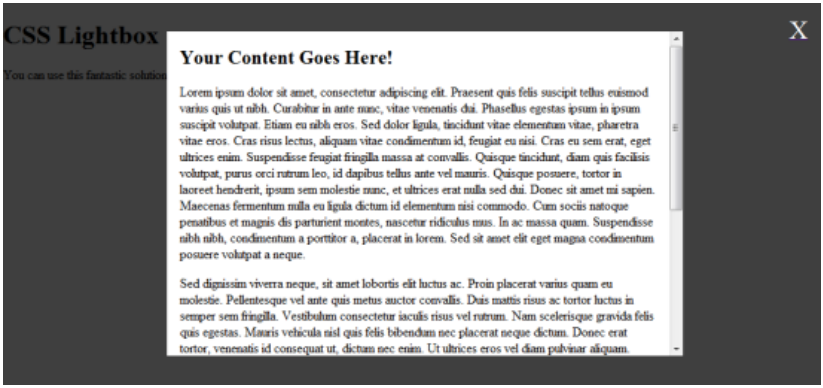
Sources:

- http://madebysofa.com/

Semantic CSS3 Lightboxes

The rise of jQuery, MooTools, and JavaScript frameworks has given many web designers a new lease on life, adding more unique functionality into their sites.

Most notably among the various cool and interesting features you can find being injected into a design is the humble lightbox (modal window).



If you've ever come across a link or image which — upon clicking — increases in size and where the rest of the screen gets "shaded" to focus on the content, you'll know what I'm talking about.

This tutorial aims to showcase a method of displaying content based on the lightbox, which is web accessible and (excluding Internet Explorer) will require no scripting at all. Sound like fun? Well, let's explore the subject further!

What About Scripting?

There's a lot to be said about the benefits of using client-side scripting for this. While we are certainly not in the cut-and-paste era that leads to the kind of crimes against JavaScript many coders would like to see punished, we do, to this day, still have an uncomfortable reliance on scripts and frameworks in order for our websites to function.

Perhaps it seems a little hypocritical in this instance for me to critique the use of scripting (as Internet Explorer users will require scripting enabled to use this functionality), but like all things, we sometimes need to compromise our code to ensure compatibility (especially with that browser)!

Why Not Just Use JavaScript?

In the sense of the current scripting frameworks, I'm not going to say that they are inherently bad because the likes of jQuery often gracefully degrade with good levels of success — you guys can put away the knives and flame-lit torches now!

However, the sad facts are that, due to security issues, widespread abuse and the intrusive potential of client-side scripting, it's quite easy to understand why a large number of people do not (or cannot) make use of scripting.

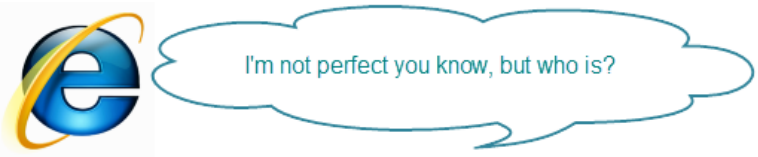
Therefore, a solution is required. And hopefully for you CSS3 fans, you'll find this simple, easy-to-use code to be a welcome improvement.

Browser Support of this CSS Lightbox

To give you a quick introduction before we begin producing the code itself, it's worth mentioning that I've tried the code in a variety of browsers and can say for sure that it works in all recent versions of Mozilla Firefox, Google Chrome, Apple Safari and Opera (which is all great news).

With a little bit of unobtrusive scripting that replicates the CSS3 techniques, it'll also run smoothly within IE 6, 7 and 8.

The other fantastic news is that current test versions of IE9 support CSS3, so this may very well be the best JavaScript-free solution there is in the near future.



Internet Explorer is the usual culprit when it comes to lacking in standards support.

Magic Markup!

To begin our quest for a much more compatible lightbox, the first thing we need is some general HTML markup.

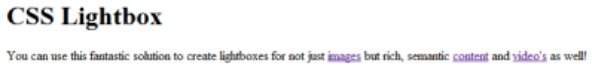
Much of the below won't surprise you in the slightest as it's pretty standard. However, for this example, and to showcase how durable this cool method is, we'll produce three individual lightboxes.

One will be for an image, one will be for a block of scrollable content and the final one will hold a YouTube video. What more could you ask for?

To begin, let's create the basics and have three fragment links which will go to the correct lightbox (you'll learn how a bit later on).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
    <head>
        <meta http-equiv="content-type" content="text/html;
        charset=utf-8" />
        <title>CSS Lightbox</title>
    </head>
    <body>
        <h1>CSS Lightbox</h1>
        <p>You can use this fantastic solution to create lightboxes
        for not just <a href="#image" title="Image
        Lightbox">images</a> but rich, semantic <a href="#content"
        title="Content Lightbox">content</a> and <a href="#video"
        title="Video Lightbox">video's</a> as well!</p>
```

```
    </body>
</html>
```



Not much going on at this stage, a simple HTML document with extra links.

So far, things are pretty simple; you have a heading and paragraph with some links that, upon clicking, will currently do nothing.

The next stage in the process, however, is all important.

We need to add the container for each lightbox to appear within and give them their appropriate values.

To give you an idea as to how each of the three lightboxes work, you'll need to examine (and add) the code into your source code editor.

To better understand how it works, let's talk about each item individually.

```
<div class="lightbox" id="image">
    <div class="w300 h60">
        <img alt="Six Revisions" height="98" width="255"
        src="logo.jpg" />
    </div>
    <p class="close"><a href="#" title="Close This Image
    Lightbox">Close <span>X</span></a></p>
</div>
```

Each lightbox you wish to provide will require a div element container that contains the id attribute you wish to link too. That id will reference what you'll need to attach to the href anchor to start the navigation process, not forgetting you'll also need the lightbox class value on each you include.

Are you with me so far? Good!

Once you have the container that references the lightbox, you'll need to include both a div (which will hold the lightbox content and will have the width and height declared through class values like in the above) and a paragraph with an anchor to act as the "shader".

Using the above source code, you should find yourself with a working lightbox that contains an image. In this example, I used the Six Revisions logo.

You can indeed link to any image of any size, taking into account the viewport space available.

A Little Bit More About the "Shader"

To better understand how this gets its visual appearance, the secondary div layer works independently from the close anchor which filters in the background. It's essentially a cascading/tiling effect at work.

The first lightbox has appeared, and it contains an image.

If you look at any modern lightbox, the most common characteristic you will see is that you have a semi-transparent layer which covers over the rest of the page drawing focus to the lightbox content. Upon clicking on this layer, it will return you to the page. This is the very reason why each lightbox requires a closing anchor.

What Should the href Value Be?

The great thing is that you could even link the href to another lightbox if you wanted it to do something other than close (using the null fragment link).

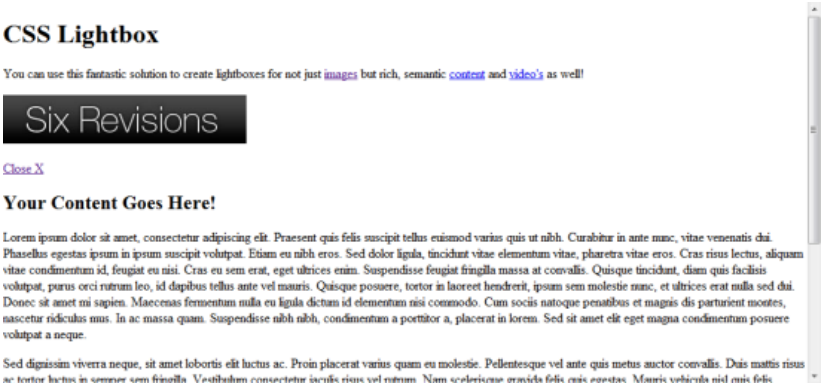
What Can the divs Contain?

As for what's inside the div, it's entirely up to you. You can use any block or inline elements (the lightbox will house any HTML element).

In the first example, we had an image; let's add some content within the code:

```
<div class="lightbox" id="content">
    <div class="w60p h400 scroll">
        <h2>Your Content Goes Here!</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Praesent quis felis suscipit tellus euismod varius quis ut nibh.
        Curabitur in ante nunc, vitae venenatis dui. Phasellus egestas
        ipsum in ipsum suscipit volutpat. Etiam eu nibh eros. Sed
        dolor ligula, tincidunt vitae elementum vitae, pharetra vitae
        eros. Cras risus lectus, aliquam vitae condimentum id,
        feugiat eu nisi. Cras eu sem erat, eget ultrices enim.
        Suspendisse feugiat fringilla massa at convallis. Quisque
        tincidunt, diam quis facilisis volutpat, purus orci rutrum leo,
        id dapibus tellus ante vel mauris. Quisque posuere, tortor in
        laoreet hendrerit, ipsum sem molestie nunc, et ultrices erat
        nulla sed dui. Donec sit amet mi sapien. Maecenas
        fermentum nulla eu ligula dictum id elementum nisi
        commodo. Cum sociis natoque penatibus et magnis dis
        parturient montes, nascetur ridiculus mus. In ac massa quam.
        Suspendisse nibh nibh, condimentum a porttitor a, placerat
```

```
        in lorem. Sed sit amet elit eget magna condimentum
        posuere volutpat a neque.</p>
        <p>Sed dignissim viverra neque, sit amet lobortis elit luctus
        ac. Proin placerat varius quam eu molestie. Pellentesque vel
        ante quis metus auctor convallis. Duis mattis risus ac tortor
        luctus in semper sem fringilla. Vestibulum consectetur iaculis
        risus vel rutrum. Nam scelerisque gravida felis quis egestas.
        Mauris vehicula nisl quis felis bibendum nec placerat neque
        dictum. Donec erat tortor, venenatis id consequat ut, dictum
        nec enim. Ut ultrices eros vel diam pulvinar aliquam.
        Phasellus non nisi vitae ligula imperdiet dapibus eleifend sed
        neque. Morbi gravida dignissim turpis eu auctor. Morbi
        pellentesque urna vitae nunc dictum elementum. Aliquam
        erat volutpat. Aenean urna nibh, pretium ut accumsan ut,
        luctus eget nibh. Nulla sollicitudin fermentum turpis eget
        rutrum. Integer dignissim dui turpis. Morbi metus libero,
        suscipit blandit dignissim aliquam, sodales non mi. Proin id
        augue odio, sit amet gravida mauris.</p>
        <p>Proin a dignissim orci. Nam nec urna nisi, in blandit
        lorem. Nulla cursus ornare rhoncus. Nunc lectus orci,
        tristique et aliquet sed, venenatis et felis. Nullam sodales orci
        at est pharetra nec aliquam risus scelerisque. Nullam varius,
        nisi ut sagittis scelerisque, nulla mauris tempus magna, quis
        pellentesque sem erat a diam. Cras lectus est, dictum ut
        consequat ut, adipiscing sit amet sapien. Curabitur tincidunt
        varius gravida. Quisque augue sem, commodo sed molestie
        venenatis, cursus vehicula lorem. Praesent scelerisque, tortor
        a euismod malesuada, ipsum ligula semper odio, ultricies
        molestie sapien metus condimentum felis. Vivamus
        hendrerit gravida interdum. Fusce at purus eu orci laoreet
        lobortis. Aliquam cursus mi at tellus fringilla dictum.</p>
    </div>
    <p class="close"><a href="#" title="Close This Content
    Lightbox">Close <span>X</span></a></p>
</div>
```



For all whom understand the value of dummies, some Lipsum content is added.

Add a YouTube Video

And to round things off even further, let's add a YouTube video into the mix:

```
<div class="lightbox" id="video">
    <div class="w640 h386">
        <object type="application/x-shockwave-flash" width="640"
        height="385" data="http://www.youtube.com/v/
        bsGEWHNJ3s8"><param name="movie" value="http://
        www.youtube.com/v/bsGEWHNJ3s8" /></object>
    </div>
    <p class="close"><a href="#" title="Close This Video
    Lightbox">Close <span>X</span></a></p>
</div>
```

Video on demand, that'll work in the lightbox too! Pretty cool isn't it?

At this stage, we have completed the entire HTML file that this demonstration will use.

If you want to preview the results in your web browser, you'll notice that as well as the heading and paragraph, there will now be the three pieces of information on the page (the image, the content block and the video), each with a "Close" link right below it.

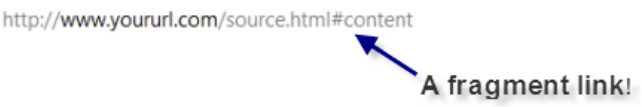
Right on :target

At this stage, you're probably wondering how we're going to ensure that only the right content appears when it's needed. And that's a good question! We are going to use a cool CSS3 pseudo-selector called :target (which styles based on the anchor).

Perhaps you've come across the :target selector before, perhaps not. Basically, it cleverly allows you to apply specific style to an element if its id matches the fragment identifier, which is the hash (#) symbol followed by text usually found in page bookmarks. For example: index.html#content (you'll see it in the address bar).

So, without any further ado, it's time we began formatting our lightbox to perform the required task whenever this fantastic selector is activated.

You now know why we give the three links in the first piece of code those unique URLs (and gave each div a matching id).



Fragment links are best indicated by the # (hash) character at the end of a URL.

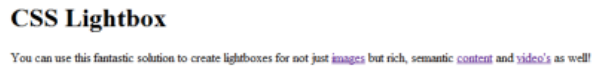
For the following piece of code, I'll provide everything you need using a style element within the <head> of the document.

Generally, this is considered bad practice because all CSS styles should be separated from the structure. However, as this is simply an example and it works unobtrusively, you can easily place all of the stylistic code in an external file. For simplicity, I'll keep it in-line.

To begin, we'll start by adding in the easy-to-follow snippets of code such as removing the padding and margins from the document. Also, we'll hide the lightboxes off-screen until they're required using some absolute positioning and negative margins.

```
<style type="text/css">
    html, body {
        height: 100%;
        overflow: hidden;
        width: 100%;
        margin: 0;
        padding: 0;
    }
    body { overflow-y: auto; }
    .lightbox {
        left: -999em;
        position: absolute;
    }
</style>
```

Note: It is entirely possible for you to hide your three lightboxes when they're not in use using the property display: none. However, as this can affect how some screen reading software will interact with your page, I've used negative positioning to get the same effect.



With lightboxes offset, things disappear.

Upon refreshing your browser at this stage, you'll notice that the three previously visible lightboxes will be hidden, which is great.

However, when you click on the links, nothing happens! Don't worry, your lightbox isn't broken.

All we need to do is add in the code to activate the two layers (the "close this" lightbox anchor, that gives the fade effect and the lightbox content itself).

```
.lightbox:target { bottom: 0; left: 0; right: 0; top: 0; position:
absolute; }
.lightbox:target .close a { background: rgba(0, 0, 0, 0.75); bottom: 0;
left: 0; right: 0; top: 0; position: absolute; z-index: 1; }
.close span { color: #FFFFFF; font-size: 2em; text-indent: 0; position:
absolute; right: 0.5em; top: 0.5em; }
.close {text-indent: -999em;}
```

Keeping things simple, the above code will span the lightbox to every corner of the browser viewport using absolute positioning.

The second line of code will reinforce this code on the anchor itself and will give it a z-index (for priority over the other stuff on the page) and a background which makes use of opacity in browsers that natively support CSS3 alpha transparency.

The third line will use the span element on the "close" button to give it some emphasis for people to know that clicking in the anchor closes the lightbox.

The last line will hide the text in the faded section (for full effect).

Note: It's worth pointing out that if your content scrolls and you wish the lightbox to follow the scroll action, you can replace position: absolute with position: fixed, however IE6 will not support it. For the example, I've used absolute positioning for simplicity and to showcase its potential usage.

Now when you click on a lightbox link, the transparency effect flourishes.

If you refresh your browser with the above code, you'll notice upon clicking any element that you get that amazing faded effect which will span the entire content. And you should also have a neat little "X" button in the top-right corner just to help reinforce the effect upon clicking outside of the lightbox.

The problem now is we need to get the lightbox contents itself positioned correctly and looking great on the screen.

So below, you'll find the last bits of CSS to align the content and offset it based on the width of the contents specified.

```
.lightbox:target div { background: #FFFFFF; position: absolute; left: 50%; top: 50%; z-index: 99; }
.w60p { margin-left: -30%; width: 60%; } .w300 { margin-left: -150px; width: 300px; } .w640 { margin-left: -320px; width: 640px; }
.h60 { height: 60px; margin-top: -30px; } .h400 { height: 400px; margin-top: -200px; } .h386 { height: 386px; margin-top: -193px; }
.scroll { overflow-y: scroll; padding: 0 1em; }
```

If you refer back to the HTML code, you'll see that each lightbox container has class values such as w300 (for width 300px) or h400 (for height 400px) or w60p (for width 60%).

The values for each lightbox were simply calculated by measuring the width and height of the contents required and then setting the container width and height to match the dimensions.

Take for example the image we used: As that had specific width and height needs, a specific width and height was applied to the lightbox container, and for each, a negative margin for half that value was given to give it accurate centering.

Note: You may have noticed that a scroll class was also added to match the width and height classes for each lightbox.

Simply put, if you have lots of content within the container and you give it a fixed width and height, the scroll mechanism will allow it to overflow with scroll bars attached. Lovely!

Adding the next bundle of code will properly align the lightboxes content.

Now try refreshing the page again and — Voila! — just like magic, whenever you click one of the links, its content will load in a container, in the center of the screen.

And the fantastic thing is, it will look and function like any other lightbox. Clicking inside the lightbox will act normally, but clicking outside of it will restore the main screen (and thereby hide the lightbox contents as they're not the fragment).

If you use Firefox, Chrome, Safari or Opera, this code will work perfectly and is a 100% pure CSS way of implementing a lightbox.

But for Internet Explorer, we (as usual) have compatibility issues to deal with!

Tip: Just imagine what you can do with these target lightboxes: You could have anything from the content shown above, right through to setting the width and height of the container to 100% and have a literal different "page" appear with overflow enabled on a single page web design. Woot!

Compatibility Crisis

Without going into a great deal of detail, the below JavaScript code has been produced to deal with Internet Explorer's lack of support for :target (this could well be the first effective and simple solution for this issue).

As with the style, I've attached the content inline using <script> tags in the header, but it's entirely graceful and can appear in an external file.

As long as the script is placed below the <style> element, you shouldn't encounter any issues in regards to the emulation of the code.

```
<script type="text/javascript">
<!--
/*@cc_on @if (@_jscript_version > 5.6)
 function bootup(){
    var tds = document.getElementsByTagName("a"); lightbox();
    for( var x=0; x < tds.length; x++ ){tds[x].onclick = function()
    {setTimeout(lightbox, 1);};}
}
function lightbox(){
    var counted = document.getElementsByTagName("div");
    for( var x=0; x < counted.length; x++ ){ if
    ( counted[x].className == "boxfocus" ) { counted[x].className
    = "lightbox"; } }
    if (location.hash.substr(1) == "") {} else
    { document.getElementById(location.hash.substr(1)).className
    = "boxfocus"; }
}
window.onload=bootup;
@end @*/
// -->
</script>
```

Keeping the above code as simple as possible, it makes use of conditional comments (for JavaScript) to ensure you are using Internet

Explorer (hence why it references Jscript) thereby ensuring the code will not interfere with good browsers which render correctly.

When IE is verified, and the page loads, the bootup function monitors whenever an anchor is clicked.

When this event is triggered, the lightbox function is activated and it simply applies a set of classes to the div that matches the hash (#) in the address bar (fragment link), while restoring all other lightbox instances back to the default.

Essentially it says "You're using IE? OK. You clicked a link or loaded a page? OK. The link clicked references a lightbox? OK, let's set that in motion and reset any others which may be active".

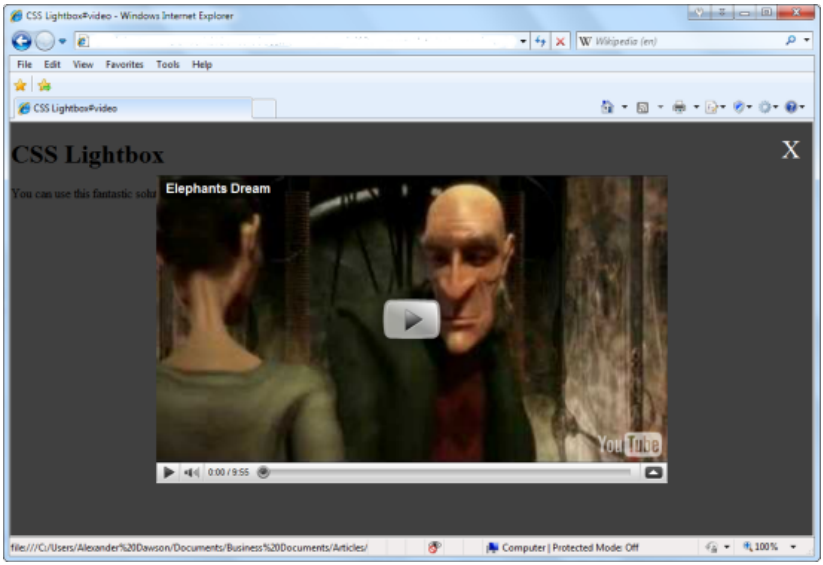
If you're not too JavaScript-savvy, it may not make a lot of sense, but the code requires no user maintenance and if you use the code, you'll just have to trust me that it'll do the job. Except for those classes to be applied, which have the alternative to the :target stuff, we'll need to add these into the style in the header.

And to overcome IE's lack of opacity support, we'll add a transparent repeating PNG image in its place.

```
.boxfocus { bottom: 0; left: 0; right: 0; top: 0; position: absolute; }
.boxfocus div {background: #FFFFFF; position: absolute; left: 50%;
top: 50%; z-index: 99; }
.boxfocus .close a { background-image: url('trans.png'); bottom: 0;
left: 0; right: 0; top: 0; position: absolute; z-index: 1; }
```

Those of you with eagle eyes may have noticed that the properties of the boxfocus classes, which sit in place of those using the :target pseudo, contain mostly the same properties and values as their counterparts. So you may well ask: Why not group them together?

While it may seem like a smart idea to reduce repeating code, unfortunately the default behaviour for IE and other browsers is that unknown or invalid code (as it would appear to IE) should be totally ignored and deleted. Which means, if grouped, it wouldn't work, so we need the repeated CSS as a separate entity.



With a bit of caring JavaScript, Internet Explorer will work like any other browser.

If you now refresh the page again and open it in Internet Explorer, you will find that the lightbox should now work properly in that browser.

It may not be a perfect solution to require extra CSS and some conditional JavaScript to work the mojo, but alas, this isn't a perfect world, so we must use it to keep things as compatible as possible.

On the bright side of things, Internet Explorer 9 has full support for target so in the future, none of that extra code will be needed.

Setting Standards

And on that note, the example is complete.

- It's totally semantic: with div containers and the use of any HTML elements you wish within the container
- It's accessible: it'll work with screen readers
- It's not as dependent on scripting as the jQuery solutions and you can bookmark their behaviour
- And — all things considered — it's more graceful in that the future looks good for the selectors' widespread browser support without the need of any scripting

Get Involved with CSS3. Please.

Hopefully this tutorial will inspire you to get involved in testing modern standards like CSS3 because many of these elements already have basic native support.

While there is a real need at this time to make things gracefully degrade (and this option certainly can achieve that), it's worth highlighting that the potential for the :target pseudo-class goes far beyond lightboxes: It could remove the future need for JavaScript in single-page websites, content swapping, and other process-intensive functions.

While the need for workarounds (like the JavaScript we needed to use) won't disappear overnight, the future of web standards is pretty bright!

Sources:

- https://en.wikipedia.org/wiki/Modal_window

Problems Using Web Validation Services

Amongst the basic skills that fledgling designers and developers should know is the art of website validation. Website validation consists of using a series of tools such as W3C's Markup Validation Service that can actively seek out and explain the problems and inconsistencies within our work.

While the use of such tools has benefits (in the sense of being an automated fresh pair of eyes), a worrying trend of either over or under-dependence keeps rearing its ugly head.

This article aims to underpin the inherent issues of validating your websites through automated web services/tools and how using these tools to meet certain requirements can miss the point entirely.

Current Practices

Before we begin critiquing the valiant efforts that our noble code validators undertake (they have good intentions at least), it's important to note that with all things in life, a balance must be struck between the practical application of validation and common sense.

We live in a modern era of enlightened thought where web standards have become a white knight, always charging towards slaying code which fails to best represent our work.

But while current practices actively request and promote using these validation tools, no web-based tool is a substitute for good judgment.

177	#main, #primary, div div:target, .image:target img	Property -moz-border-radius doesn't exist : 5px 5
177	#main, #primary, div div:target, .image:target img	Property -webkit-border-radius doesn't exist : 5px
178	#main, #primary, div div:target, .image:target	Property -webkit-box-shadow doesn't exist : 5px 5

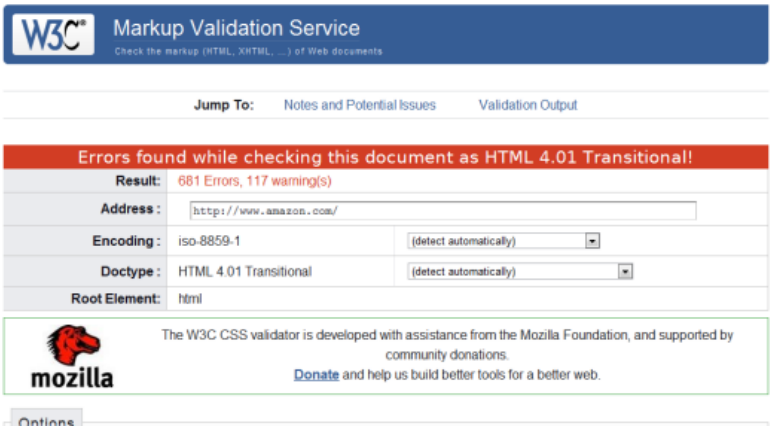
The above won't validate, but it's acceptable if there's no alternative.

Not Using Valid Code

The case for under-dependence can be seen by examining the Alexa Top 100 sites and using some basic W3C validation tests.

The eye-watering number of errors these popular sites produce (which escalate into the hundreds) is rather unsettling for some.

The problems for those ignoring validation altogether has been well-documented to the detriment of end users (as has the justification for following web standards) and as ignoring validation entirely makes you as guilty as those using it as a crutch, it's worth recommending not to forsake these tools even with their shortcomings.

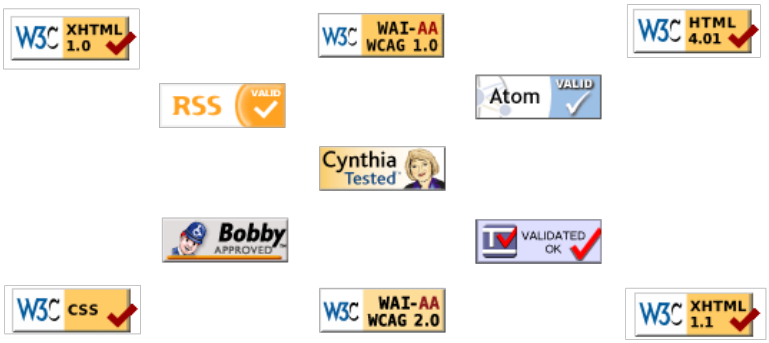


Amazon's front page doesn't validate, but it doesn't mean they don't care.

Blindly Following the "Rules"

The case for over-dependence is something we need to worry about too. Those who form drug-like addictions to making everything validate or meet a certain criteria just to please an innate need for approval are on the rise.

While ensuring your code validates is generally a good thing, there are professionals who take it to such an extent that they resort to hacking their code to pieces, ignoring new and evolving standards or breaking their designs just to get the "valid" confirmation. Quite a price for a badge. And there are even people who think validation automatically means everything's perfect, which is worse.



There are a lot of tools, be wary about which you rely upon.

Context is King

The thing about validation tools that beginners (and some seasoned professionals) often overlook is the value of context.

The most common problem that validation tools encounter can be summed up in the sense that they are only machines, not humans.

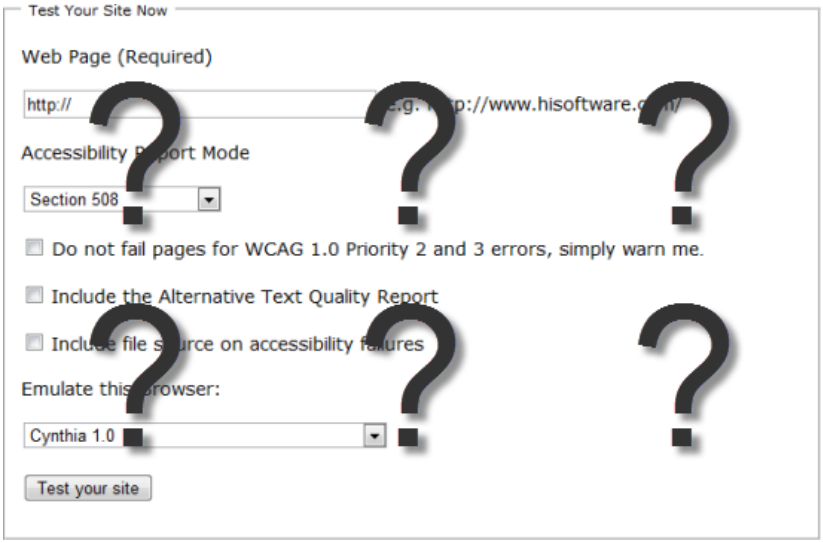
You see, while checking if the code you wrote is written correctly, on the surface, may seem like a simple task, that sites meet disabled users' needs, or that text on the screen is translated properly — the very obvious truth (for those who understand the mechanics involved) is that the complexity of how humans adapt cannot be replicated effectively.

What does this design say to a machine? Nothing! It only sees the code and that's it.

You Can Make Decisions That Robots Can't

If you've seen the "The Terminator" movie, you probably have a mental image of a not-too-distant future where machines can think like humans and therefore make decisions based on adaptive thought processes, such as being able to intelligently and emotively know what makes sense.

But unlike that film, the levels of which such tools can understand context and meaning (beyond what's physically there) simply doesn't exist today—though that's probably a good thing as we don't want the W3C validator going on a rampaging killing spree against <blink> tag users, right?

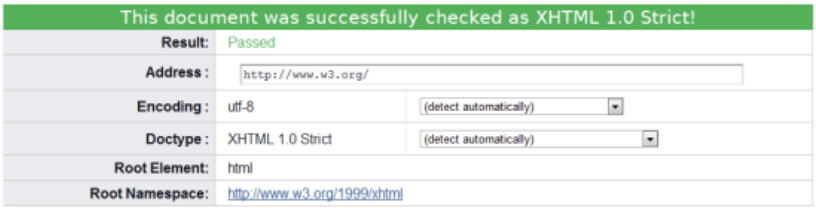


There may be a time where machines are as smart as humans, but that's not today.

Code Validation

The most notable form of code validation in use today is that of the W3C HTML and CSS validators.

The level of obedience from some designers and developers to ensure their code validates is best reflected in the way many websites actually proclaim (through the use of badges) to the end user that their code is perfect (possibly to the point of sterility).



Proclaiming the validity of your code doesn't mean what you've produced is perfect.

This reminds me of the way software developers proclaim awards from download sites as justification to use their product. As I've mentioned previously, however, the W3C validator (despite its association) is not perfect.

Failing Because of Future Standards

It's an established fact that the W3C validator not only examines the structure of your site but the elements or properties themselves (though they don't understand semantic value!).

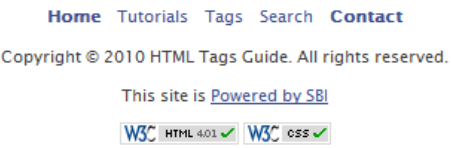
The key issue with such due diligence from these tools is that elements which are not recognized (such as those of upcoming standards like CSS3 or equally valid proprietary extensions) are often misinterpreted by developers as "unusable" or "not approved" and therefore get rejected.

Taking Stuff Out For The Sake of a Badge

It seems rather amusing to me that people are willing to omit the value of somewhat acceptable but non-standard code — CSS3 attributes specific to particular browsers, for example — to satisfy the validators, like they're trying not to anger the Tiki gods.

What ends up happening is the validators themselves make it seem like legitimate practices which defies convention are automatically wrong, and this results in a strange psychological condition in which people too quickly limit their own actions for the sake of a machine (or the ideology it provides).

While it makes sense not to use deprecated/future-standards code, validators simply can only test against what they know.



Nothing says: "I want approval" like these well-known badges of honor from the W3C.

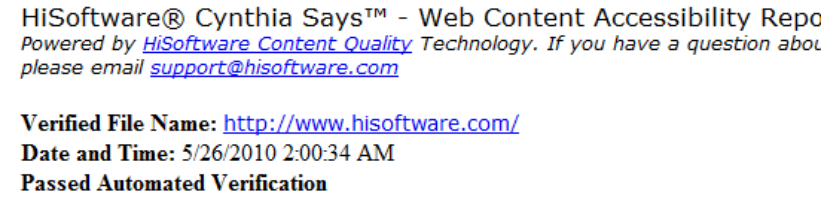
It should be made quite clear that people who proclaim HTML and CSS validation on the page are doing so to make themselves feel better. Unfortunately, none of your users (unless you cater specifically and strictly to web designers and developers) are likely to know what HTML even is, let alone understand or trust what the fancy validator badge is telling them!

Web Accessibility Validation

If you want a case where validators totally miss the point and where their limited testing ability is abused (to proclaim the work which is being tested against is complete), you need look no further than the accessibility validation services like Cynthia, the now debunked "Bobby" and their kin.

One key issue with validators are that they can only test against what they can see (in almost all cases this only accounts for source code).

While some of the issues in WCAG can be resolved with some helpful coding (like alt attributes on images), code doesn't account for everything in accessibility.



Cynthia "says" your site meets WCAG guidelines, but it's often missing several points!

The Only Way to Test for Accessibility and Usability is through People

Accessibility and usability are highly subjective issues that affect many people in many different ways, and often the way code is presented (or even the content) does not establish where key problems may lie.

Too often beginners actively use the checklist nature of these services to claim their work is accessible on the basis that a validator covered what it could (omitting the complexities which it cannot account for – such as the sensory criteria and their inhibiting factors). This key lack of understanding and the wish for a quick fix showcases that reliance of these tools isn't ideal.

How many validators can tell you how easy to read your content is? How many of them run a screen reader over the top of your site to denote the way a blind user may find your information?

While some factors can be mechanically replicated, the problem is the tools primarily focus on code alone and therefore miss the bigger picture (and those who rely on them also get caught up in this lack of awareness).

Without the background knowledge of how such web or software applications function, a scary number of people simply use them as an alternative to properly learning what they're doing.

Do you speak Latin? No? This content would pass as accessible, readable and valid!

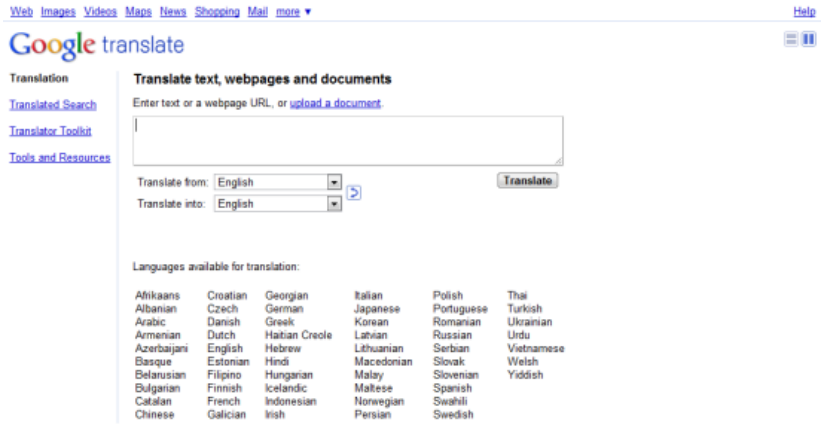
The state of accessibility tools is so bad that I advise people not to use them in favor of proper human checking.

The myth that tools can do things at an equal skill level of a human is far from the truth and while the W3C validators can be helpful, accessibility tools are too biased to be credited.

Translation Troubles

Confusion (as denoted above) in relation to such validation tools comes in many forms. Whether it's the mystical messages the W3C validator produces (which beginners may not understand), the lack of fair warning that these tools should be used "as part of a balanced diet", or that these tools are often much more limited in what they can offer than you would be led to believe.

One of the more comical examples of automated tools going crazy can be seen through translation tools such as those provided by Babel fish or Google, which again proves that nothing is better than humans.



Google Translate is popular amongst websites for giving "rough" language translations.

One of the key elements of human languages is that words can have more than one meaning (and deciding which instance is in use can be tricky for machines – a case of context).

In accessibility, the issues of vision can be anything from total loss of eyesight right down to a case of color blindness. Because of this, a language translator will simply go for a literal meaning of the word rather than the context in which it's used which can reduce your content into a scrambled illegible mess which doesn't help your visitors (especially if they have learning difficulties).

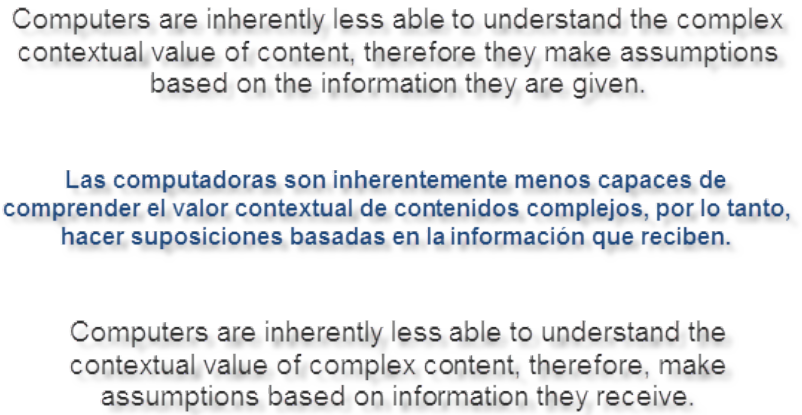
While of course translation tools aren't code validators, they do in fact perform a similar service. By taking a known list of criteria (whether code, words or something else), they attempt to check that something accurately portrays what it's intended to.

If, however, you use something it doesn't expect (like a new word in translation tools or a new property in the W3C validator), it will report it as a failing on your behalf.

Such reliance on validation tools for "perfect" results is therefore unjustified and can limit yourself to the detriment of your audience.

A Translation Exercise to Test the Idea

If you take a block of content from a website, paste it into Google Translate, translate it to another language, and then translate it back into English, you'll see for yourself how badly these validators of content conversion are at the job. It can give you hours (if you're really that much of a geek) of comedy in a few sessions!



See how the same sentence has been wrongly translated? It's not uncommon!

The Silver Bullet

Knowing that validation tools are far from perfect is an important lesson to learn. Many people assume that such tools are an all-knowing oracle that accounts for everything your users or browsers may suffer.

While it's wrong to say that these tools aren't useful, it's important to understand that the validation tools should not be used as a guarantee of accuracy, conformance or accessibility (in your visitor's best interests).

A valid site should never be achieved if it sacrifices the progression of web standards, unjustly acts as a badge of honor or attempts to justify the end of the build process.

Knowing how and when to use code and the difference between right and wrong is a tough process we all undergo during our education.

The truth about validators is that sometimes being invalid is the right thing to do, and there are many occasions where a "valid" website is nowhere near as valid as you might like to think it is in terms of code semantics, accessibility or the user experience.

I hope that all of this will serve as a wakeup call to the generation of coders who either ignore or abuse validation services. These tools are not a silver bullet or a substitute for being human!

Sources:

- https://validator.w3.org/
- https://www.alexa.com/topsites/countries
- http://venturas.org/badges
- http://jigsaw.w3.org/css-validator/
- https://www.st-andrews.ac.uk/itsold/web/accessibility/cynthia_validator.shtml
- https://www.lipsum.com/
- https://translate.google.com/

Sexy Tooltips with Just CSS

Providing supplementary information about potentially complex elements of a user interface is a central part of any website designer or developer's workflow in creating usable and accessible websites.

One of the most common mechanisms for providing extra details beyond what you can see on the page is the tooltip (a design pattern for showing tips about a particular element on a screen).

While many innovative solutions exist using CSS and JavaScript (with and without JavaScript frameworks like jQuery), it's sometimes useful to look towards new technologies to examine the impact they may have on our current techniques.

Thus, we're going to look at how using the evolving CSS standard can enhance some lovely cross-browser tooltips.

Tooltips are Terrific!

Whether you need to explain an abbreviation or acronym, to define a word or if you simply want to give additional details based on what someone hovers over, tooltips provide a simple but effective solution.

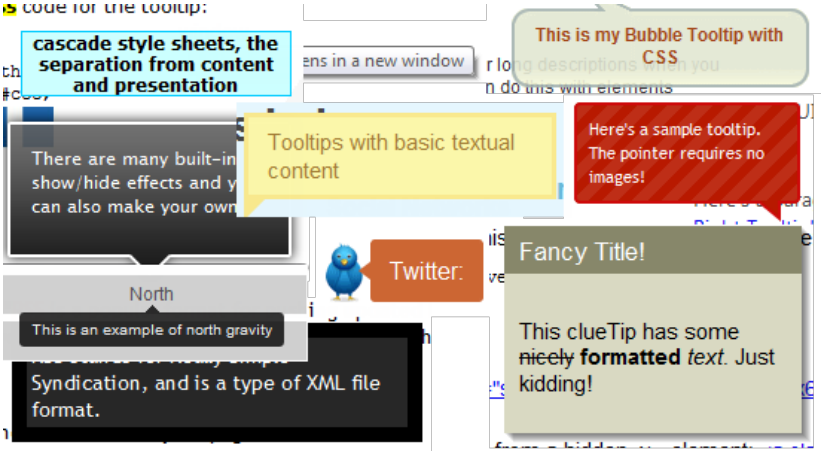
From the little yellow block of text that appears over elements such as images with a title attribute (or the alt attribute if you're unfortunate enough to use Internet Explorer), right down to the CSS and script-powered solutions that exist, they're fantastic devices that unfortunately seem to get little interest from the design community.

Most browsers have a default style of tooltip, though it's not very pretty.

Leveraging Progressive Enhancement in Tooltips

As standards rapidly evolve and support for new techniques appear within browsers on a more consistent basis, the advances of CSS allow us to produce tooltips (which serve as a replacement for the somewhat boring browser defaults as shown in the image above) to a brand new level of detail and beauty.

If you already use a jQuery-powered example — fear not! — there are still many things JavaScript can accomplish which CSS cannot (like adding a delay on when tooltips disappear), but highlighting the ways we can use CSS to better equip our designs may inspire you to create some beautiful solutions of your own, outside of tooltips.



There are many existing solutions for tooltips you can choose between.

What We're Going to Make

In this example, we are going to be producing a pure CSS tooltip mechanism that is aesthetically enhanced using CSS3 (rather than using it to achieve some higher purpose).

What this means is that it will work on browsers that don't support CSS3 (such as Internet Explorer 8 and below) — it just will not look as pretty. This concept is known as progressive enhancement.

Subtle effects such as the colors, fonts, imagery and border you give your tooltip may differ depending on the end user's machine (such as their browser, installed fonts or monitor contrast).

CSS3 Extras

Using simple yet effective extras like the now well-established border-radius property and the box-shadow property will give what used to be a generic-looking "boxy" popups a new and more sophisticated visual appearance.

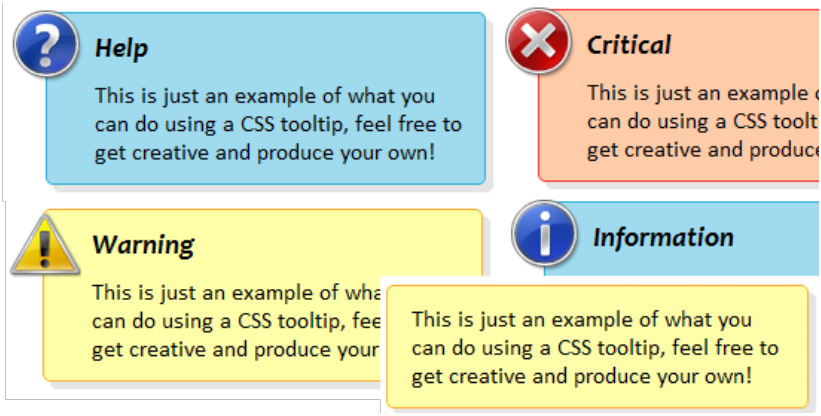
Under the Hood

As always, it makes sense to get some basic code down in your chosen source code editor and we shall begin with the HTML source code for our examples.

Different Types of Tooltips

For the benefit of giving you enough ideas to build upon or implement directly into your own work, we shall produce five different tooltips.

Each will look very similar in order to maintain a standardized appearance, but you should feel free to experiment further and continue evolving these techniques.



You can see how pretty a tooltip can actually be, much better than the default!

Cross-Browser Compatibility

This mechanism should work across all browsers, however, if you feel the need, you can tweak it to better suit your own requirements.

Basic Markup

On the code block below, we have a generic XHTML 1.0 template with the usual <head> elements that are required.

As we add the CSS into the design, it's worth pointing out right now that for the purpose of this tutorial, the CSS will be embedded into the document using the <style> tag.

Separating structure from style is (of course) important, I highly recommend putting your CSS into a separate stylesheet in production usage.

For ease of discussion, I decided to place the styles inside the HTML document.

I also highly recommend that if you feel that adding jQuery or other JavaScript enhancements may assist the usability of this lovely eye candy, feel free to do so!

HTML Markup

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
    <head>
        <meta http-equiv="content-type" content="text/html;
        charset=utf-8" />
        <title>ToolTips Example</title>
    </head>
    <body>
        <h1>Examples of CSS ToolTips!</h1>
        <p>Here are some examples of a <a class="tooltip"
        href="#">Classic<span class="classic">This is just an example
        of what you can do using a CSS tooltip, feel free to get
        creative and produce your own!</span></a>, <a
        class="tooltip" href="#">Critical<span class="custom
        critical"><img src="Critical.png" alt="Error" height="48"
        width="48" /><em>Critical</em>This is just an example of
        what you can do using a CSS tooltip, feel free to get
```

```
        creative and produce your own!</span></a>, <a
        class="tooltip" href="#">Help<span class="custom
        help"><img src="Help.png" alt="Help" height="48" width="48"
        /><em>Help</em>This is just an example of what you can
        do using a CSS tooltip, feel free to get creative and produce
        your own!</span></a>, <a class="tooltip"
        href="#">Information<span class="custom info"><img
        src="Info.png" alt="Information" height="48" width="48" /
        ><em>Information</em>This is just an example of what you
        can do using a CSS tooltip, feel free to get creative and
        produce your own!</span></a> and <a class="tooltip"
        href="#">Warning<span class="custom warning"><img
        src="Warning.png" alt="Warning" height="48" width="48" /
        ><em>Warning</em>This is just an example of what you
        can do using a CSS tooltip, feel free to get creative and
        produce your own!</span></a> CSS powered tooltip. This is
        just an example of what you can do so feel free to get
        creative and produce your own!</p>
    </body>
</html>
```

Within the above code, you will notice that we have a heading (<h1>) element (nothing special so far) and a paragraph (<p>) of text which contains some anchor (<a>) elements (with a class value of tooltip).

Why Use Anchor Tags for Tooltips?

The reason why anchors rather than abbr, dfn or another element like span is used at this level is due to IE6's total lack of support for the :hover pseudo-selector beyond anchor elements.

Therefore, for compatibility reasons I recommend using <a>, though if you don't like IE6, you can change it.

Each anchor also contains a span with the content of the tooltip.

With some basic HTML code, things don't look or function as they will end up.

The anchor effectively acts as a point of reference for the hover effect to occur within, and as we add style into the document, the content of the spans will be hidden off-screen until they're required.

Each span in this example either has a class value of classic (denoting a general tooltip) or custom (with critical, help, info or warning to match denoting the color scheme to use).

Those using the custom style also have a couple of bonus features like the em element (denoting the overview text) and an image proceeding it (which acts as the icon for the tooltip, you can even use your own images).

Basic CSS

As you will now have the HTML on the page, it's time we make these tooltips do their job (rather than spanning the page as normal linked text).

By adding in the code block below into your <head> element, you will give each link containing a tooltip a nice dotted underline (differentiating it from normal links which typically have a standard solid underline) and a help cursor (again for visual differentiation).

It'll also remove the outline and set the color (so it feels less like a link and more like a general hover element).

The second bit of code simply hides the tooltips off-screen until their needed. Easy as pie!

Basic CSS Styles for .tooltip Class

```
<style type="text/css">
.tooltip {
  border-bottom: 1px dotted #000000;
  color: #000000; outline: none;
  cursor: help; text-decoration: none;
  position: relative;
}
.tooltip span {
  margin-left: -999em;
  position: absolute;
}
</style>
```

Web Accessibility Considerations

On a note about maintaining accessibility: the outline is removed visually as the link is effectively redundant — it's only included for the purpose of compatibility with old versions of IE.

Therefore, the outline itself isn't needed to show that the tooltip is a clickable link unless you want it to be, in which case I advise removing that bit of code.

Also for the benefit of screen readers, the tooltip contents are being moved using a negative margin rather than using display: none or visibility: hidden as some screen readers may ignore the content – which would be bad news for screen reader users.

It's amazing what a bit of CSS can do; now the page appears ready to host the tooltips.

CSS for Displaying the Tooltips

At this stage, hovering over the links will do nothing.

Soon we will have a functioning tooltip that will look about the same through whichever browser you prefer to use, but for now, it's time to place the next lines of CSS code in below what you already have.

By adding in the code block that follows, you should have a mechanism that works and displays the tooltips, though it will look very bland and will have little to no appeal (visually) apart from the fact that everything is contained within a box-like shape (which sets the standard for future code to customize further).

CSS for Showing the Tooltips

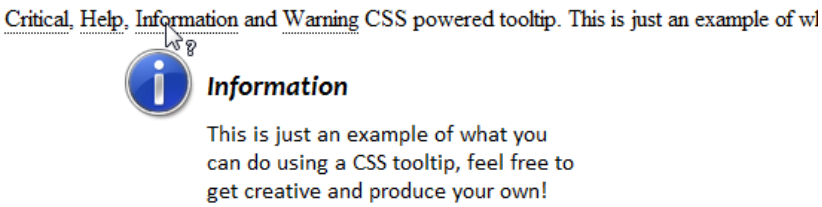
```
.tooltip:hover span {
    font-family: Calibri, Tahoma, Geneva, sans-serif;
    position: absolute;
    left: 1em;
    top: 2em;
    z-index: 99;
    margin-left: 0;
    width: 250px;
}
.tooltip:hover img {
    border: 0;
    margin: -10px 0 0 -55px;
    float: left;
    position: absolute;
}
.tooltip:hover em {
    font-family: Candara, Tahoma, Geneva, sans-serif;
    font-size: 1.2em;
    font-weight: bold;
    display: block;
    padding: 0.2em 0 0.6em 0;
}
.classic { padding: 0.8em 1em; }
```

```
.custom { padding: 0.5em 0.8em 0.8em 2em; }
* html a:hover { background: transparent; }
```

Star HTML Hack Necessity

You may have noticed that the very last line of code above uses the HTML star hack to apply a transparent background to IE6. Why was this included?

Well, while testing the tooltip, I encountered a strange quirk where the hover pseudo was not obeyed unless a background reference existed!



A lot of CSS later, we have something that functions generally as a tooltip!

With the above code in place, everything works and is visible on the page. But it's hard to read because there's no color scheme that pulls the contrast out from the page and that's something we need to fix right now to make them usable.

CSS for Giving the Tooltips Some Color

The following code will give each of the five styles of tooltip a color scheme which fits the icon (if one exists).

Upon giving your page a quick refresh and hovering over one of the links, you'll see a pretty tooltip that looks and works equally across the various web browsers.

Though as you will soon discover, there's a bit more to the story still to come!

CSS for Color Scheme

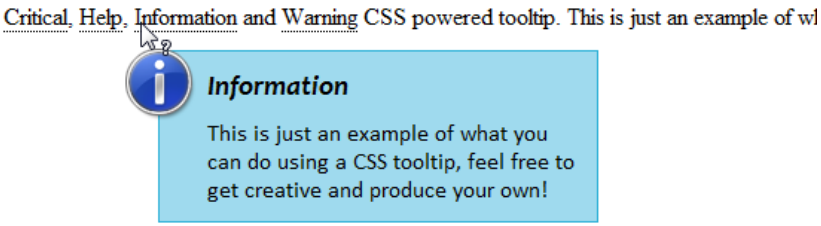
```
.classic { background: #FFFFAA; border: 1px solid #FFAD33; }
.critical { background: #FFCCAA; border: 1px solid #FF3334; }
.help { background: #9FDAEE; border: 1px solid #2BB0D7; }
.info { background: #9FDAEE; border: 1px solid #2BB0D7; }
.warning { background: #FFFFAA; border: 1px solid #FFAD33; }
```

With what you already have (as mentioned above), you'll have something a bit basic, but that looks good and does its job in giving you a colorful tooltip that you can give your website design.

CSS3 for Progressive Enhancement of Sexiness

Before we leave the example, it's worth bringing up CSS3 as we can (literally) take the edge off our tooltips, making it feel a bit less boxy using border-radius and give it a bit of extra depth using the box-shadow property.

Because neither of these elements is globally supported, it won't work for every browser, but for those that it does work within, they'll look a lot sleeker and sexier!



Much better! A little style gives the tooltip a visually unique appearance.

Add the below code into the .tooltip:hover span selector and refresh the page.

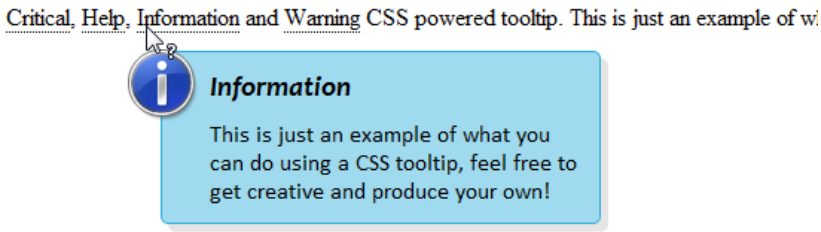
The visual effect of the border, shadow and opacity will help lift the tooltips from the page, and that may well make the information and contrast a bit easier to read.

You'll see that not only are the official CSS3 properties provided but the Mozilla and WebKit proprietary extensions too.

It's worth pointing out that this means your code may not validate (even though it's an acceptable bending-of-the-rules scenario), but the experience benefit your visitors will get may well be worth the lack of validation.

Additional CSS for Modern Browsers

```
border-radius: 5px 5px;
-moz-border-radius: 5px;
-webkit-border-radius: 5px;
box-shadow: 5px 5px 5px rgba(0, 0, 0, 0.1);
-webkit-box-shadow: 5px 5px rgba(0, 0, 0, 0.1);
-moz-box-shadow: 5px 5px rgba(0, 0, 0, 0.1);
```



Using some CSS3 properties, we can improve upon the existing CSS code.

Food for Thought

As this tutorial shows, it doesn't take much in the way of visual eye candy to give our readers a better experience.

While this style of tooltip may have been done before, adding flourishes like image-based icons, CSS3 effects and perhaps the odd bit of opacity (and typography) can help you go beyond a boxy (and rather ugly or limited) tooltip to have something a bit more elegant and pretty.

When using CSS, the key to any successful implementation is ensuring the experience degrades gracefully and what you do implement is done tastefully and with subtle enhancements.

Giving your website a few useful tooltips to notify users of the meanings of certain terms or words, explaining a general concept you talk about, or even to provide some meaningful alternative details can not only give your content some added context but it may well help your readers get more from the experience you offer them.

Perhaps the greatest thing the future of CSS offers us is a fresh opportunity to restudy and rethink our craft to better meet our users' needs. I certainly recommend taking this example's opportunity to look for other ways to give your content an added layer of interactivity.

Sources:

- https://en.wikipedia.org/wiki/Tooltip
- https://en.wikipedia.org/wiki/Progressive_enhancement
- http://css-discuss.incutio.com/wiki/Star_Html_Hack

250 Quick Web Design Tips (Part 1)

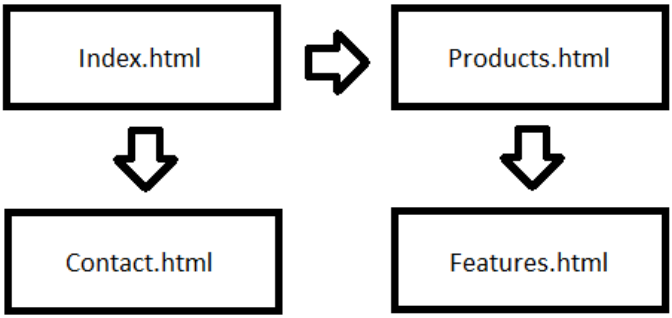
As web professionals, we're always looking for ways to improve our knowledge and skills. Tips, tricks and checklists are often one of the most underused yet potentially useful models of providing great, quick and easy to follow pieces of useful information.

You may or may not know some of the tips below — and you may or may not agree with everything listed — but hopefully it will give you some ideas for your own sites or motivate you to create a checklist to help cover your bases.

Perhaps a few items may even inspire you investigating a subject further, and that would be pretty awesome too.

This is the first part of a 2-part series. In this first part, we will cover planning, content creation, and design elements.

Planning and Getting Into the Web Design Profession



Planning what your website needs to contain can help you scale the project size.

One fundamental aspect of creating a website is the planning stage. This includes things like looking for a domain registrar and hosting package, seeking out inspiration for your design, building the information architecture, and much more.

Getting your website's purpose mapped out will help you better write content (to match your needs) and more effectively create a design that will retain the look-and-feel you want to put across.

Below are some tips and tricks which may prove useful when you're making decisions before putting your (or your clients') website together.

Picking Domain Names

1. Many people are used to seeing the www at the beginning of a website address (e.g. www.sixrevisions.com). Ensure your website functions both with and without this famous subdomain.

2. Reserving a subdomain called m (e.g. m.sixrevisions.com) for mobile devices has become a common web design convention. It's cheaper than — and as widely recognized as — the .mobi top-level domain (TLD).

3. Most of the non-technical general public tend to only recognize .com, .net and .org. It's worth checking the TLDs you want are available before dedicating yourself to a brand name.

4. Avoid using dashes in your domain name. (e.g. sixrevisions.com versus six-revisions.com).

5. Domain hacks like del.icio.us have become pretty popular, and while they may be harder to spell, they can give you an awesome alternative to a simple but unavailable .com address.

6. If you want to target a local audience, it may well benefit you to purchase a country code top-level domain (ccTLD) in your own country. Something like co.uk may be great for grabbing regional visitors in the UK.

7. Remember that some ccTLD domains require you to be a resident of a certain country. If you don't live there, you could forfeit the TLD as a violation of the registrar's agreement.

8. WHOIS privacy can be a dicey affair when you allow your registrar to put their details in place of your own. You run the risk that you may lose the domain if a conflict occurs.

9. Domain auctions like Sedo can be a great place to get a domain that's already been taken. While it can be somewhat expensive to pick up a rare domain, you might find yourself the owner of your preferred domain name.

Web Hosting

10. When picking a website host, ensure that you check what you'll get in the package. Disk space, bandwidth, CPU usage and other specified features may decide the cost you'll encounter. If you already have a web host, test their performance using these tools.

11. Beware of hosts proclaiming unlimited bandwidth or resources. Everything in this world is finite and you may find yourself falling short of contractual small print and fair use policies.

12. If you're starting a website or service of your own, it pays to start off with shared or grid hosting rather than a VPS or dedicated because you won't know how many visitors you will need to cater for.

13. Free hosting for commercial use is not a good idea. If you plan on having a commercial website, it makes sense to avoid the intrusive advertisements and purchase some basic web hosting.

Development Platform

14. If you want to have a good testing environment that will run PHP and mySQL on your own PC, install XAMPP. It's quick, easy and will help you get things running before you go live.

15. Unless you know what you're doing and have the money to finance the infrastructure, hosting your own website may not be the best or most economical idea (as fun as it sounds).

16. Pick your development platform carefully as some products (such as WYSIWYG editors) inherently produce less reliable code than a classic text editor that allows you to write by hand.

Tools

17. You don't need to rely on Adobe or Microsoft software to create a fantastic website as there are lots of free and open source products which can do the job without cost.

18. Ideas: GIMP, Inkscape, Dia, FileZilla, IcoFX, Audacity, Paint.NET, Scribus, Eclipse, Skype, KeePass, Xenu Link Sleuth, Tweetdeck, FoxIt Reader and Notepad++ are great free products for designers. For more great open source products, read the article called 30 Useful Open Source Apps for Web Designers.

19. Finding a good selection of checklists and cheat sheets can give the fledgling designer some quick, easy places to get advice on how best to approach a task.

Project Management

20. Set yourself aside a decent workspace environment. The less distractions your workspace has, the better off you will be in terms of productivity.

21. Always have realistic expectations about how long a project will take to complete. Rushing your work and releasing it "half-baked" can cause issues — just look at Windows Vista.

22. Getting some decent time-tracking or project management software is important. It's far too easy to get distracted and lose sight of the big picture if you've lots of small tasks to achieve.

23. To-do lists may seem inconsequential and rather trivial, but you may find them useful in structuring all the various tasks you need to deal with and setting yourself deadlines.

Learning

24. Always keep learning because there is no excuse in allowing your education to lapse or become deprecated. You could keep up to date with news through design blogs or perhaps learn a new web language.

25. There are many fantastic web design books and magazines out there. They also cover a wide range of subjects with ever-increasing depth as a source of education they are second to none.

26. Web resources like Six Revisions are great for learning new techniques. While perhaps not as in-depth as books, many web resources offer you useful and up-to-date advice on the web industry.

27. Remember to verify anything you learn through a third party resource. There's an awful lot of outdated information out there (like W3Schools) that could encourage bad habits.

28. Sites are beginning to teach classroom-style lessons and video-based instruction classes (e.g. Lynda.com) on web design and development. They can get pricey, but may be good alternatives to a degree.

Specialization and Competitive Analysis

29. There are many sectors you can work in as a web professional (web designer, UX, UI, front-end development, etc). You shouldn't restrict yourself to a core subject unless you know exactly what you want to end up doing.

30. Whether you decide to become a Jack of all trades or a specialist is entirely up to what you prefer. It's worth noting that there is enough work in the industry to cater to both work styles.

31. Investigate what your competitors are doing with their services as you can learn so much from the mistakes or successes that others have had — they can be a goldmine of ideas.

Learning About Your Target Audience

32. Research is the mother of all invention if you're going to work on any project. It pays to ensure what you're planning will meet the needs of the audience you're trying to gain.

33. Always try to be inventive with what you create. There's no point cloning another successful website when you could improve upon it to convert some of their existing user base.

34. If you plan to produce a blog or an informative website, ensure that you know your subject. Trying to create a medical blog with no knowledge is not a good idea. You should be passionate and be well read about your subject matter.

35. Seek out the kind of people who might want to use the service your planning and ask them what they would like to see in such a website and what popular topics is worthy of inclusion.

Inspiration

36. If you're stuck for ideas for what kind of site to create, browse around the web looking for subjects that are popular. You could serve a niche market where there's existing demand.

37. Finding inspiration for a site can come from the most unlikely sources. Watching movies or TV, taking a walk, or even talking to your friends and family can help you get business ideas.

Handling Data

38. Deciding whether you need an SSL certificate or not depends on whether sensitive personal details like credit cards or login information will be processed. It may be worth buying one.

39. Handling your customer's information is of critical importance. Never store passwords as plain text documents and do what you can to encrypt details that are stored in databases.

Conceptualization & Information Architecture (IA)

40. Creating a visual sitemap before you start building the website can do wonders for your core structure. If you know what pages you may need initially, you can plan the content ahead.

41. Certain types of websites require certain types of documents. Most portfolio websites, for example, have a contact page. Seek other likeminded websites to get required page ideas.

42. When in doubt, always do what works and the norm. There's a reason why certain types of websites succeed. It's because they follow conventional practices that visitors will adapt to quickly.

43. Concept sketches are useful for developing your ideas. Sometimes a piece of paper or a napkin with some doodles can assist you in turning what's in your mind into a workable design.

44. Wireframes are a simple, underused method of planning and plotting out an idea. You can create something as simple as basic shapes, right down to mapping out your site structure.

45. Beyond wireframes, you could also consider a working prototype when planning your site. Mocking up a quick and simple website can eliminate potential feature flaws quickly and easily.

46. Brainstorming is another fantastic but underused method to evolve your business or website ideas. Picking a loose concept and mapping related ideas to it can give quick but abstract results.

47. Some site owners write a business plan to scope out a project's evolution before it happens. If you find yourself too easily distracted, it might prove to be a useful document to make.

48. Determine what kind of person you are, and the way you use websites. It's quite subjective, but provides a good grounding point in conceptualizing how an idea can become a real product.

Miscellaneous

49. Products like EverNote or Microsoft OneNote provide you with a great platform to gather and store research and ideas. Think of it like a sketchbook you can turn to for inspiration.

50. Never give up. It's so easy to think an idea has fallen flat, and most people tend to move on far too quickly. Most ideas can become what they're intended to be with enough hard work.

Content Creation

Even something as simple as an About page should have purposeful content produced.

Everyone keeps reiterating the same term over and over: "Content is king" has almost become a mantra which writers of web copy sing from the rooftops. And they're right to do so!

Whether your content is provided in textual form, vivid imagery or some beautifully implemented audio and video media, ensuring your website's content is up-to-scratch will help you turn visitors into customers.

When you come to producing the content that will help visitors understand what the website is about, the following tips may give you some relevant advice to keeping your users hooked.

51. There is more to content than text. Providing polls, infographics, or interactive elements that have content-based value can help improve the interest and readability of on-page information.

52. People respond to engaging prose.

Copyright, Content Licensing and Legalities

53. If you're intending to build for other people, ensure you have some good solid contracts to work from. You don't want to be unprepared if the client refuses to meet their obligations.

54. Creating paperwork such as invoices, receipts of purchase, questionnaires (for contract work) and other useful materials will reduce your workload if you start doing freelance jobs.

55. Word of mouth constitutes a binding contract, though it's harder to prove you shouldn't say you can or will do something unless you fully intend to follow through what you state.

56. All services should have good terms of service, privacy policy and copyright agreements. It's important that your end-users know what you expect from them (and that works in reverse)!

57. You don't need to have a copyright statement on your website (though it's good as a reference). Ignorance of intellectual property does not qualify as a valid excuse.

58. When deciding how to license your finished design, you may want to check out creative commons or open source licenses; they're pre-written and flexible (which is great).

59. A cheap way of writing agreements or contracts for your website is to examine others and then write your own based on it. You can save yourself a lot of money in potential legal fees.

60. Avoid legal jargon whenever possible and simply state outright what you want to say in an agreement. Your clients will be more likely to read what you say if they can understand it,

61. If you write your own contracts, it might pay to have them read over by a lawyer to get them as watertight as possible. Verifying is often cheaper than having it custom written.

62. Accessibility statements aren't as important as they used to be (as being natively accessible is more of a requirement), but providing one may be useful to your website's audience.

Content Formats and Considerations

63. Get the hang of compression — whether it's using GZIP for content, caching for external files or squeezing extra bytes from images and media. It will increase the speed of your website.

64. Consider the best image format for what you are trying to achieve, while GIF makes for good basic animations, JPEG or its less lossy friend PNG will be better for high-resolution photos. Read The Comprehensive Guide to Saving Images for the Web for more information.

65. Be careful as to what you use images to portray. Not everyone can see images (like search engines) and this may present readability problems if you use them in place of text.

66. When adding video, audio or graphics into your site, make sure alternative content is available for those who cannot take advantage of these mediums due to accessibility issues.

Images

67. Opacity in images is a tricky issue with Internet Explorer. There are fixes for issues in IE6, but you should remember that only full alpha transparency has issues, not single colors.

68. Your logo is one of the most important aspects of your website as it's what people will recognize you for. Therefore, it pays to have a good, memorable one created for your brand.

69. While the favicon is one of the smallest graphics you're likely to encounter on a website, it provides a fantastically unique way of gaining recognition in bookmarks and social networks.

70. Producing an Apple touch icon at 57×57 pixels can be useful for users of the iPhone, iPad and iPod touch who can proudly display your site in their home screens (using web clip).

71. There are loads of sites that provide free stock images, audio and video if you're not much of a pixel-pusher.

Content Writing

72. Even if you're not an articulate individual, trying to ensure that your spelling and grammar are correct should be at the top of your agenda.

73. If you're at a loss for what to write, taking a break or using one of the many techniques to help remove writer's block can prove indispensable to the content creation process. See the Content Strategy category for tips.

74. A simple way to reduce the complexity of content is to take what you have and boil it down to 50%. It may seem a lot, but reductionism can seriously help eliminate the waffle!

75. Writing your content before you start designing your website can help you better approach the coding stage as you can pick the right elements that describe your content's value.

76. Content is king. If you sacrifice the quality of the content for the design of the website, your visitors may likely hit the back button in their browser and never return as a result.

77. Much of writing for the web is down to practice. Don't be afraid to start off small with the likes of Twitter or forum posts before building up your credibility as a web content writer.

78. Making content fun and involving is important to being successful. While dry humorless copy might get across the point, being quirky will emote passion.

79. Never be afraid to ask for help and feedback or get colleagues to proofread what you have to say. Often, a bit of critique will help you become a better professional.

80. When linking to another website, ensure you notify the visitor of how the target site relates to the content or element of the website so they don't end up at an undesired location.

81. Break your content down into easy to manage segments. Using unordered lists, for example, can help increase the content readability.

82. Fluff and poor quality marketing speak is unnecessary. Always keep to the point and avoid redundant technical language. We all hate junk and in the recycling bin it all belongs.

83. Ensure that what you say is factually correct. Citing references will give your words added credibility.

84. Don't plagiarize or steal other people's content. If you find people stealing yours, it's worth taking the time to learn how to send DMCA takedown notices and cease and desist letters.

85. When writing content of your own, simplicity is valuable. If you can strike a balance between being informative and being overly wordy, you could avoid wasting your reader's time.

86. Don't span long documents over multiple pages if you can avoid it. Such practices can reduce the readability of content as readers will be forced to break their natural flow to jump pages.

87. If you're planning on having a blog, ensure that you state if you're reviewing something and have been paid to do so.

88. There are so many fantastic CMS solutions (i.e. WordPress). If you find less technical people are going to contribute to a site you make, they can be ideal in removing some complexity and speeding up content production.

89. Consistency is important with everything you write. Maintaining a core set of standards and values helps ensure regularity.

90. Always try to put across information in a friendly and non-aggressive tone. Being overly sarcastic or rude can lead to arguments that can degrade the value of your content.

91. Feedback can be just as important in content writing as the written material itself. Using blog comments, for example, can give entertaining and potentially informative extra reading.

92. Write for people, not search engines. Your users are more important than your Google PageRank.

93. If you plan on providing translated content for international users, nothing beats a human translator. With that said, there are some decent translation tools out there.

Multimedia Content

94. If you create a podcast for your website, a good compression-to-quality ratio is 96kbs MP3 (for voice recordings). Large file sizes are a pain, and at this level, you can save a lot of bandwidth.

95. MP3 is arguably the most compatible audio format around. If you're providing alternative formats like OGG or FLAC, then ensure an MP3 version exists for more restrictive audio players.

96. Embedding Windows Media Player or Apple QuickTime into a page may have problems if people don't have the players installed. Flash has a higher market penetration than both.

97. Automatically playing music is a sin — it's annoying, so don't do it.

98. Remember that Flash-dependent components are not reliable: People with vision and hand-mobility impairments limit them in accessing a lot of Flash-based content.

99. If you are planning to provide content through email, keep subscribers' email addresses private (don't use the CC feature when sending out emails en masse).

100. Don't spam or send out heavy streams of email – people hate it.

Design Elements

Color is a critically important part of your design as it may invoke or reflect emotion.

One of the most subjective parts of creating a website is its design. Whether you're looking at accessibility, usability, the user experience, or even something as fundamental as the psychology of color, giving your users the best possible experience with as little effort as possible can prove tricky.

101. Your web design does not need to be pixel perfect. Every device, platform and browser render things slightly differently but that's not always a bad thing if your site's still usable.

102. If you are requesting users to sign up for a service on your website, always keep the amount of required information to a bare minimum. Keep things simple.

103. Keeping file sizes as small as possible is important for improved page response times.

104. Mobile web designs should be simple. If you have less content, no Flash dependence, a single column layout and a liquid design, you should have few problems with visibility.

105. mobiForge has an excellent mobile web development guide that is full of best practices and some useful guidelines to helping make the mobile experience better for your end-users.

106. Don't rely on fixed-width designs. Toolbars, sidebars, add-ins, viewport sizes, window sizes, screen resolutions and many other factors can affect the amount of real estate available to users.

Colors

107. Color can invoke a wide range of subtle psychological influences over people. Knowing how to use color and various contrasts may help you better engage with your audience.

108. Consider how people associate color with feelings: red for hot, blue for cold, white with purity and clarity, black with darkness and death, yellow with happiness and sunshine, etc.

109. Contrast is important when using colors. For certain people like the color-blind, the ability to distinguish various shades may be diminished and they may struggle to read content.

110. The idea of web-safe colors is relatively redundant due to the way screens have evolved, but making sure your site is color accessible for visual impairments is worthy of your consideration.

111. Color theory and harmony are important parts of design. Understanding how such devices influence the way information is perceived is worth studying.

Typography

112. Typography is an ever-increasing variable of importance within web design. As the range of fonts that can be used within designs increases, the legibility of those fonts becomes vital.

113. Producing a font stack is easy! Have your chosen font followed by an alternative that looks similar, then its closest relation that's likely to be available, and finally the type (like serif).

114. Size is another variable of typography within design that you should consider. The larger the scale, the more readable it becomes and the increased attention it will receive.

115. Giving emphasis through styled italics, strength through bold visual styles or underlying and striking through content can affect the perceived importance design elements receives.

Arranging Design Elements

116. White space/negative space is a valuable commodity. Don't pack your design full of stuff! Having enough breathing space will improve the readability of your design and help the reader "scan."

117. Scanning is the act of an end-user flicking through content on your pages to determine the information they are looking for. The ease they can do this will affect how they use your site.

118. Websafe typography is a big deal unless you embed a font (which has legal implications). You can't guarantee the end-user will have any font installed, even common ones like Arial.

119. Organizing your information on-screen can be a tricky task. Using conventions and patterns like the logo appearing in the top-left hand side can improve the ease of use for visitors.

120. Knowing how to appropriately display content like navigation menus is an art form and a science. Seeing how others implement such devices can help measure success rates; check out the site called Pattern Tap for common design patterns such as site navigation.

121. Remember that most people read content in a left-to-right manner. Therefore, it makes sense to have important details as high and as far to the left as possible in your design.

End-User Considerations

122. Your design should directly reflect the needs of the end-user. Don't pack it with useless features and widgets like clocks or weather applets. Only give them what they need, as they need it.

123. When updating your website (which you should do often), check your website statistics to see how people navigate around your website. It can be helpful to find where issues occur.

124. Understanding how people perceive and respond to your brand can be the difference between trust and abandonment. Your visitor's views are more important than your own.

125. I've noted it earlier, but it's worth reinforcing: update your website often! People gauge the prevalence and accuracy of websites by the rate at which they are maintained.

126. If you have the time, consider reading about psychology and sociology topics. They're not strictly dedicated for the web, but they apply in so many regions of the industry that it's worth learning about.

127. Don't design a website for yourself. As much as you may like that scrolling animated reel you just implemented, you will spend little time visiting the website compared to your audience (who matter).

128. In a websites design, people look for an experience. If you give them something positive to remember, you'll give them satisfaction, which may result in a long-term relationship.

129. The satisfaction a user gets is directly related to the way you provide information. If the user struggles to find their way to a document, you'll make them angry.

130. Interaction-based design is important. While static content has uses in certain situations, giving users something to explore and play with will result in a more memorable experience.

131. Unnecessary interaction should be eliminated from the project. While subtle or useful enhancements are great for the end-user, added barriers may cause a user to abandon ship.

132. Applications tend to follow different rules to conventional content distribution. A need for logical structure and purpose will be of increased importance within the user interface.

133. If developing apps for a mobile device, it may prove useful to attempt an offline version for when Internet access is not an option. Dead zones for cell phone signals still exist.

Web Accessibility

134. Accessibility is an important aspect of any web design. If certain people can't access the site or the content, that's a group of people who you could have converted to a customer.

135. Numbers in relation to accessibility are highly biased. Don't think of people with disabilities as a minority; just think of how many people need glasses for reading — that's a major one!

136. Impairments come in all shapes and sizes: they can be physical, intellectual, emotional, social or even technological (e.g. people without broadband connections or people using mobile devices as their browser agent).

137. The scale and duration of disabilities differ: someone may be paralyzed (which would be long-term) or they may have a broken arm (short term). Don't just think of lifelong issues.

138. There are plenty of helpful specifications and laws in relation to accessibility. Have a read through WCAG 1 and 2, Section 508, PAS 78 and the Six Revisions guide on quick web accessibility tips, to name a few accessibility guidelines and best practices resources.

139. Always provide alternative content for images or media and don't rely on <img> elements without alt attributes or Flash content without text variants. You're hurting some of your visitors.

140. Checking your work in a screen reader is quite easy to do. There are free tools out there such as WebAnywhere, a browser-based screen reader simulator, as well as commercial alternatives like JAWS that you can install and test your website through.

141. Don't become too reliant on tools like Cynthia as accessibility validators because these tools only examine machine-readable code. They're not perfect solutions for checking your design, semantic structure, content flow, and visual elements of accessibility. Read more about the problems with website validation services.

Usability

142. Usability.gov has a selection of great guidelines in PDF format that can help you improve a website for your end-users. These PDFs are well worth reading through to see what they can offer in your design process.

143. Steve Krug and Jakob Nielsen are two highly respected experts in the field of usability. If you pick up books they have written, you'll find lots of fantastic usability guides in them.

144. Usability is about making a website as seamless and functional for the end-user as possible. Do whatever you can to help users find and accomplish what they set out to achieve.

145. Before you launch a website to the general public, it's worth getting a group of people together to test out your design and find any bugs which exist in the system and to see how usable your design is.

146. Carrying out a usability study can be as simple as asking a group of visitors to carry out a specific task and getting their feedback in the form of a questionnaire to improve upon.

147. Ensure your design degrades gracefully to create a usable design that is as universally designed as possible.

148. Progressive enhancement should be what you aim for. Simply put: you want to make sure everything can be used at a core level, and increase functionally for devices that can cope. A scenario would be using CSS3: make sure that your design still works and looks decent on browsers that don't yet support CSS3.

149. Encourage people to get involved in helping improve your design. Ask for useful critiques and feedback that can give you future website evolution ideas.

150. Keep striving for perfection. It's probably not possible to have a perfect web design, but if you always aim for the best, it'll encourage you to continue making an effort in maintenance.

Sources:

- https://en.wikipedia.org/wiki/Top-level_domain
- https://en.wikipedia.org/wiki/Domain_hack
- https://en.wikipedia.org/wiki/Country_code_top-level_domain
- https://en.wikipedia.org/wiki/WHOIS
- https://sedo.com/us/
- https://www.apachefriends.org/index.html
- https://www.gimp.org/
- https://inkscape.org/en/
- http://dia-installer.de/
- https://filezilla-project.org/
- https://www.audacityteam.org/
- https://www.getpaint.net/
- https://www.scribus.net/
- https://www.eclipse.org/downloads/
- https://www.skype.com/en/
- https://keepass.info/
- http://home.snafu.de/tilman/xenulink.html
- https://notepad-plus-plus.org/
- https://en.wikipedia.org/wiki/Transport_Layer_Security
- https://davidwalsh.name/set-sites-apple-touch-icon
- https://en.wikipedia.org/wiki/Digital_Millennium_Copyright_Act
- https://mobiforge.com/
- https://en.wikipedia.org/wiki/Web_colors
- https://www.w3.org/TR/WCAG20/
- https://www.section508.gov/
- https://en.wikipedia.org/wiki/PAS_78

- https://guidelines.usability.gov/

250 Quick Web Design Tips (Part 2)

As web professionals, we're always looking for ways to improve our knowledge and skills. Continuing on with our list of quick web design tips (see Part 1), these tips will cover the website development process as well as positive marketing and promotional tips for web designers.

This is the second part of a 2-part series. In this second part, we will cover development and marketing tips.

Development

Something every web professional should get themselves swimming in on a regular basis is code. Code is the backbone of our entire industry and turns pretty graphics and content into a formal structure that everyone can enjoy.

While there are far too many languages out there to be able to learn them all, some languages like HTML, CSS and JavaScript seem to be centrally focused in whatever we do online.

The tips you'll find in this section all relate to coding for the web, and while some of them should be common sense, it doesn't hurt to reiterate some valuable things which may improve your code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>A List Apart</title>
                <meta http-equiv="Content-Type" content="text/ht
        <meta name="description" content="" />
        <meta name="keywords" content="" />
        <link rel="alternate" type="application/rss+xml" title="
        <link rel="stylesheet" type="text/css" href="/css/print.
        <script type="text/javascript">if(top!=self){top.locatio
        <link rel="stylesheet" type="text/css" href="/css/home.c
        <style type="text/css">
                @import url(/css/307.css);
        </style>
</head>
```

Applying markup in the correct proportions will give your website some inner beauty.

151. Web standards are important. If you can provide semantic code that accurately describes the content, then you may score a better search position or contextually richer value.

152. Semantics should not stop at using the right element for the right job. For example, giving your classes and IDs effective names (like with microformats) can improve the site's contextual value.

153. Specifications exist for every web language you'll encounter (even the more obscure ones). While they can be quite technical, if you have the patience, it's well worth reading them.

154. Validate your syntax using the W3C's validation tools. While you should use them to help find and fix bugs in code, remember that you shouldn't sacrifice evolution for compliance.

155. I recommend learning these technologies in this order: HTML, CSS and then JavaScript (if you're a beginner) as this order allows you to understand the underlying layers that the sequential languages attach themselves to.

Debugging and Testing

156. When debugging code, it makes sense to eliminate the causes by removing chunks of code temporarily to see if the issue resolves itself (you can narrow down the culprit this way).

157. Tools like Firebug can be exceptionally useful for looking through how your code is being applied (live on the web). It's worth having a copy of Firefox with it installed for testing.

158. Services that test your designs for cross-browser support like BrowserShots are not perfect, but they do have a benefit of giving you quick glances of how your site might render on different browsers.

159. If you want to test your website in IE6, there are lots of options such as a virtual environment or tools like IETester or the Spoon installers. Don't just leave the elderly product to chance!

160. Internet Explorer 6 may be the bane of our lives, but I highly recommend testing in any web browser that holds more than 1% of all traffic to your site, as it's a significant proportion.

161. I recommend separate folders for style (CSS), feeds (RSS/Atom), scripts (JavaScript) and images (GIF/JPEG /PNG). It can be helpful when you organize, test, and debug your site's content.

162. Relying on pseudo protocols like mailto isn't a good idea without a fallback mechanism. If the end-user doesn't have a supporting product, they will effectively become a dead link.

Browsers and User Agents

163. Be consistent with the browsers you test your site in. If you go with the big five ones — Internet Explorer, Firefox, Chrome, Safari, Opera — then remain loyal to those as a bare minimum.

164. Using browser hacks is generally a bad idea as there's no future support guaranteed. If you do need to make IE behave properly, conditional comments are a friendlier solution.

165. Remember that mobile devices may render your website differently. Checking on various platforms like the iPhone can be extremely important to encouraging mobile device traffic.

166. Jscript (within IE) has its own set of conditional comments called conditional compilation which means you can target JavaScript with IE-specific code just like with stylesheets.

167. Google Chrome surprisingly doesn't natively support RSS and Atom without a plug-in, therefore — like many things (i.e. Flash) — have it as a supplement rather than a requirement.

168. Device detection on mobile devices is a sticky subject. There are far too many devices for PHP header checks to be effective and handheld media types aren't strictly and uniformly implemented.

169. WML is a dead language. If you're producing a mobile-friendly website, it doesn't pay to cater to such a limited audience unless you know you have people using old mobile phones.

170. Apart from fixed CSS units (px, cm, mm, etc.) you can use elastic (em), liquid (%) fluid (min/max-width) or even a hybrid of various measurements to size your design to flex to the end-user's demands.

Behaviour

171. JavaScript frameworks like jQuery and MooTools can be helpful if you want a quick plug-in method of displaying your code, but be wary of the amount of bloat or unnecessary complexity they may have.

172. Intrusive scripting like preventing right-clicking and forcing pop-ups is damaging to the end-user. You might think it will aid your design but it more likely causes users to hit the back button.

173. Don't use HTML frames or elements like <marquee> and <blink>. It has become an unwritten rule not to use such user-experience-damaging markup at the cost is greater than the reward.

174. Replicating browser functionality is a pointless endeavour. The user's browser has print, font resizing, and bookmarking functions for a reason. Having these things is redundant.

175. Code protection is impossible. If you want to make sure people can't take what you've created, don't put it on the web! Protection scripts only harm genuine users of your site.

Markup

176. Remember to include a DOCTYPE declaration at the top of your (x)HTML documents. You would be surprised how many browser inconsistencies occur as a result of omitting the DOCTYPE.

177. One thing that bugs many web developers is the use of the term, "Alt tag." The alternative content item which is being referred to is an attribute, not a tag.

178. Don't include natively block-level elements within inline elements. elements were not intended to be used

generically within paragraphs or lists. You'll break the semantics if you do things poorly.

179. Deprecated elements should not be used unless you have a seriously good reason to make use of them. If there's a replacement intended for the same role, use that in its place.

180. HTML5 may be a useful addition to the languages we can use for the web, but always keep in mind that browser compatibility may be a major issue for elders with scripting disabled!

Styles

181. CSS3 offers several useful, gracefully degrading properties that can benefit your visitors. If you take the time to learn how things are evolving, you'll be ready for the next generation.

182. Optimizing your CSS can be achieved by making use of inheritance, using selectors carefully, avoiding repeat declarations and grouping your style based on what it does for your design.

183. Remember not to forget about printers. Having a print-specific stylesheet will help eliminate ink and paper wastage and will make your content look professional when printed.

Various Web Technologies

184. If you need to display large documents, rather than have a Word document, consider using a PDF or an XPS file, which are formats oriented for the web.

185. While Canvas, SVG and VML (and their relations) have a lot of potential for vector graphics on the web, it's worth recognizing the serious cross-browser compatibility issues they have.

186. Flash shouldn't be avoided altogether — it has a higher penetration of compatibility at this time than HTML5 and it can offer several things that, without scripting, would be impossible.

187. The key to successful Flash implementation is alternative content for when it's unavailable. I certainly wouldn't rule out the use of Flash in the future, but dependence is a serious flaw.

188. Flash is a great platform for providing dynamic content like video and audio, but don't use it in place of mission-critical components like navigation or the whole website itself!

189. Silverlight and Java are options worthy of recognition if you're not a Flash junkie, though it's worth recognizing that they have less market share and will therefore be less compatible.

JavaScript

190. JavaScript can be useful for validating web forms and aiding the end-user's experience, but remember to use server-side scripting to verify things in case scripting is unavailable.

191. Unobtrusive scripting is the key to any website. Don't allow scripts to infest anchor links using the JavaScript: pseudo and avoid scripting JS-dependent functionality without a fallback.

Miscellaneous

192. There are some great bookmarklets that can aid your development cycle using any web browser. Quix, Spry Toolkit, Firebug Lite and Aardvark are perfect examples of these.

193. The best way to improve a site's speed is to take advantage of caching. Separate your site's structure from the style (CSS) and from the behaviour (JS), using individual files to achieve this.

194. Table-based designs aren't a good idea unless you're forced to use them for the likes of HTML email. It's poor semantics and affects screen-reader accessibility.

195. Picking the right server-side language is important. Options include PHP, classic ASP, ASP.NET, JSP, Ruby, Perl, ColdFusion and Python. Research your options before deciding.

196. Your server environment will generally dictate what languages you can code in. Depending on the setup, you may find that certain server-side languages (like ColdFusion) have limited support.

197. If you find yourself wanting to get involved with specific packages like WordPress, you may find yourself needing to know

a specific language. Account for this in choices you make concerning what server-side language to learn.

198. If you want to remember certain user data only temporarily, perhaps consider saving the information as a browser-specific cookie in preference to storing it in a server-side database.

199. Database formats are pretty varied, though most of them make use of the SQL format. Popular options include mySQL and MS SQL Server.

200. Have you ever considered taking a web application offline? There are solutions such as Adobe Air, Titanium and Mozilla Prism that allow you to create quick, packaged solutions.

Marketing

Finally, we're going to look at some useful pieces of advice for when you have the finished website but want to ensure search engines and visitors see it.

Using a mixture of general marketing advice, search engine optimization tips (of the ethical variety) and social media usage, you may find these tips useful in taking your web project to the visitors you seek.

Getting your website an audience — and keeping the interest in your website alive — is probably one of the hardest and most time-intensive aspects of owning a site. Hopefully these tips may inspire both you and your clients.

Advertisements are a core component of getting your brand around the web.

201. Branding yourself or using your own name to represent you online can present risks of its own. You may encounter confusion with people who share the same name as yourself!

Search Engines and Rank

202. Give your website a creative name. It should be short, snappy and easy to remember (and spell). Visitors need to be able to recall who you are when they find a need for your goods.

203. Don't try to trick search engines. Keyword-stuffing into hidden sections and using deceptive techniques aimed to gain bonus PageRank may result in you being booted by Google.

204. Keep things organic without targeting search engines whenever possible. High quality content, for example, will seek better long-term rewards than a poor quality link-farm.

205. Spending time reducing errors in your code and using elements (like headings) appropriately will ensure that search engines not only find your content, but also index it accurately.

206. Remember that marked-up text will hold a higher placement in search engines than non-indexible content such as Flash or PDF

files that tend to have more limited semantic value embedded within.

207. Your Alexa rank is not important. Alexa ranks are highly biased and provided by a select few who install the toolbar. It's honestly not worth the value many people seem to place on it.

208. Search engines do ignore keyword <meta> tags. I know plenty of people still claim that this isn't the case, but its downright wrong information. They do, however, follow descriptions.

209. The DCMI (Dublin Core Metadata Initiative) is a great way of giving your site added value and meaning. If you're into metadata, I recommend you check the specification out.

210. Adding a Robots.txt file to your website can help you better assist search engines in knowing what not to index. Robots.txt files are very easy to produce and are widely recognized.

211. Remember that a robots.txt file will not prevent people from stealing your content. Search engines unfortunately don't have to follow them, so they could just ignore them if they wish.

212. OpenSearch is an XML specification with little recognition and a powerful function. It lets you add a custom search entry to the web browser! You could produce a cross-browser search function, for example.

Best Practices

213. Every website should have an XML Sitemap. It's generally an XML list (though HTML or text ones are supported) of every page on your website, indicating how often content is updated.

214. <title> elements should be unique on every page in your website. I also recommend having the unique name before any static content (like the site name) for the sake of bookmarks.

Advertising

215. Advertising in the right areas may find you some great traffic but it pays to research the keywords you wish to target carefully, otherwise, you could be pouring money down a drain.

216. If you find the likes of Google AdWords isn't turning the amount of results you want to gain, consider getting your services listed directly with useful related sites by buying ad-space.

217. Sponsorship is a great way to get involved with a community whose audience your site may appeal to. Most Industry-related events (like web design conferences) allow advertising, for example.

218. There are many different ways you can monetize a site. Ones you might consider include advertisements, premium access or support, donations, subscriptions or even exclusive features!

219. Monetizing a website isn't an easy task. Beware of get "rich quick schemes" that claim guaranteed earnings that sound too good to be true (because they almost always are).

Branding, Reputation, Networking

220. If you're planning to make money through your venture, ensure you have all the relevant paperwork filled out (and notify the tax people). You will need to report all of your earnings.

221. Don't just think of your project as a digital venture, consider having some business cards, fliers or printed resumes created for when you're out looking for clients in your local area.

222. The domain extension you choose will not impact your search engine ranking, Google, and Bing do not account for the TLD used (unless you require a local regional position).

223. Trust is a mission-critical brand element. Only put your name to project ideas you believe are in the user's best interests — you don't want to end up like the privacy ridiculed AOL.

224. When you drop a link to your own website, do it only in places where it's deemed suitable. Spam will reflect poorly on your reputation and may get you kicked out of communities.

225. Joining chat rooms, forums or likeminded individuals in online spaces can be a great place to network, learn additional skills and perhaps even find some extra paying customers.

226. Getting your sites featured on other websites can boost your ranking in search engines (if the link is dofollow) and reputation online, however, you shouldn't bother with "free for all" directories as the quality return will be low.

227. If you want to get your name out to other professionals, taking some business cards along to a web design conference can be a great place to meet others on a face-to-face basis!

228. Don't underestimate the value of marketing yourself through fliers or adverts in a local paper or magazine (or even the Yellow Pages). Local businesses want websites too!

229. If you're looking to apply for some existing jobs to showcase your skills to clients, check job boards on reputable, well known niche development sites rather than the likes of monster.

230. Social Media is one of the simplest ways you can build up a reputation for yourself on the web. Using services that allow you to build a community of followers can turn into job leads.

Selling

231. If you're offering physical goods, consider getting them featured on well-known merchants like Amazon where existing customers may find what you're offering as service on the side.

232. Want to showcase your skills to the active community? Consider expanding your portfolio by writing an e-book or contributing to an open source project that will get your name out.

233. Digital Rights Management and product activation is something I recommend avoiding. The bad people will still crack the item and it's the legitimate customers who eventually suffer.

234. Selling goods online is quite a simple process thanks to "middle man" merchants like PayPal, Google Checkout and Amazon Flexible Payments, so consider your options wisely!

235. Occasionally, giving something away for free can drive a lot of interest and potential clients to your website. Whether it's coding for a charity or handing out a free e-book, it all helps.

236. If you're offering a desktop, device or web-based piece of software, submitting them to download portals such as CNET Download.com can get your product featured, win you more customers and even awards.

Social Media, Social Networking and Blogging

237. Twitter is a great method of getting into blogging if you find your time is stretched. You have a limited amount of space to be interesting. This service is basically a crash course on writing.

238. Setting up a blog can be a great way to gain some extra traffic. If you think you can write about your chosen subject on a regular basis, you could turn a profit on it and gain clients.

239. If you have the voice or face for it, having a podcast with a few work colleagues or friends (or even on your own) can give your sites a unique additional layer of marketing exposure.

240. Join communities who promote best practices in your particular industry. Being associated with these groups gives you a hint of professionalism and allows you to help good causes.

241. Facebook is used by a good proportion of the world population, therefore, it may be worth setting up a base camp for your project. You can share stuff with fans of the service too!

242. StumbleUpon is not only a great resource of inspiration (and a place to find awesome learning materials), it's also a good place to give your services some non-spammy exposure.

243. Getting your professional profile (or a copy of your resume) uploaded on sites like LinkedIn and Google profile can give people a place to find you professionally (so free exposure).

244. Whichever social networks you decide to use, don't overdo the self-promotion, use the services for their intended purpose and spend time building up some high quality content.

245. Social networks provide you with a place to allow customers to get in touch with you, gain useful feedback on your site and testimonials. Don't think of them as just a linking service!

Miscellaneous

246. Always take the time to review any marketing plans you have for your website or services, mistakes can be costly and you want to ensure your reputation is as positive as possible.

247. Having a large number of visitors or a high PageRank does not mean your site is successful. It's the conversion from visitor to regular user that ultimately decides the fate of your service.

248. Try to be unique with the way you market yourself. Look outside of the web for inspiration and you may find a unique method to advertise your services with few active competitors.

249. Having side projects either for fun or to expand your business can give you client-crossovers where users from one service may discover what you provide elsewhere (and join up).

250. Enjoy what you do and don't be too critical. If you find yourself struggling to meet your expectations, your clients will notice and you could lose potential visitors or fail as a result.

It's worth re-iterating that what's listed here only scratches the surface of the many facets within our industry.

Perhaps it's fair to say that beginners may find some of the tips more useful than a seasoned expert, but I've always been of the opinion that it doesn't hurt the most established web architects to keep learning, reinforce their skills and perhaps remember what they previously may have forgotten.

I hope some of the tips will prove useful in your everyday process, and perhaps to those who it may benefit: like Chinese whispers, you can pass the tips to the next generation.

Sources:

- https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes
- http://dublincore.org/specifications/
- http://www.robotstxt.org/

- https://www.sitemaps.org/protocol.html

The Web's Undead

For most people, the web looks and feels like things are all peachy — vibrant, alive, new, fresh. However for those of us in the know, below this facade exists a consistent cycle of death and rebirth.

While many technologies and practices have left this world and passed on to the next (R.I.P Netscape), some have been more resilient. Supposedly dead elements of the web are rising from the grave, continuing to haunt us.

This article will explore the state of the web zombie invasion!

Nature of the Beast

I'm an avid horror film fan. I love television shows like Buffy the Vampire Slayer and movies like Night of the Living Dead. The idea of "beings" which shouldn't exist (like vampires, ghosts, mummies, and zombies) highlights the similar thoughts and feelings I receive when viewing the source code of some pretty awful websites from back in the early days.

For the novice coder who hasn't explored the history of our craft, these undead beings may blend into the landscape rather well. But unbeknownst to them is the debris of the "abandoned web" — and the perpetuation of this cycle.

IE6 is considered dead to such an extent that an unofficial funeral was given in its honor!

When I talk about zombies on the web, I'm not referring to the stereotype of the old-school "web surfer" who naively wanders around the internet, clicking on every get rich in 24 hours link to get malware infections — no, not those guys.

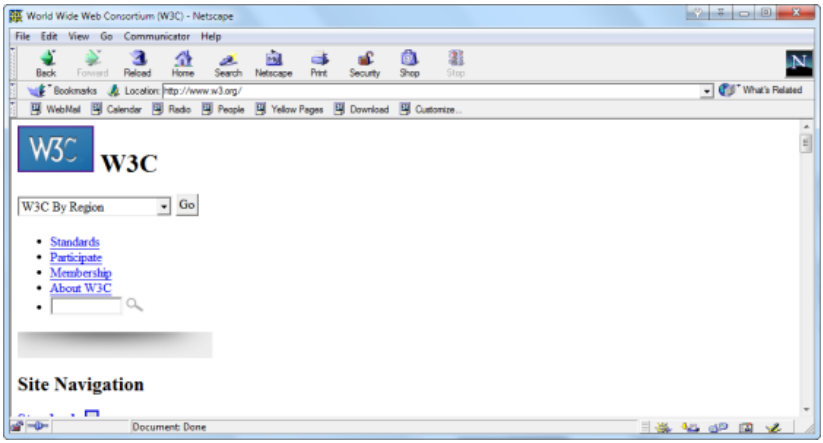
On the web, my zombies refer to the browsers, technologies, code and design practices that are officially dead, but continue to live.

Let's talk about the walking dead, starting with web browsers.

Zombie Browsers

Of the many different types of web zombies that exist, the noticeable case of outdated versions of web browsers hold the potential for being most dangerous.

Ironically, these are the types of creatures that we hold the least amount of control over. We all know the agony of giving post-mortem support for Internet Explorer 6 (which passed its use-by-date eons ago when Microsoft issued its replacement, IE7). And we fondly remember the Netscape browser that IE killed. However, the scariest thing is that, even today, there are people who can't or won't let go of their undead browsers by taking five minutes to upgrade.



The invasion of zombie browsers is still an ongoing battle.

Because we can't control the zombie browsers, the issue of those infected (staggering around using these dead shells) often becomes a matter of containment (patching our work) or in Zombieland style, killing their life support.

It's a scary thought how many dead browsers exist out there on our clients' machines.

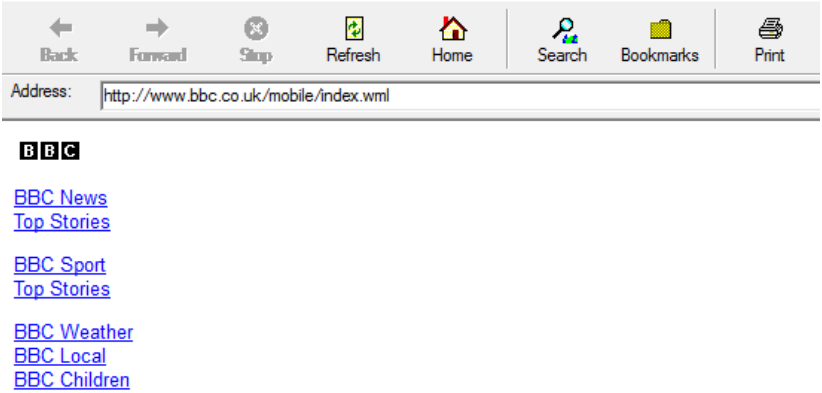
In regards to the ethics of zombie support, some of us take the Shaun of the Dead approach. Because some people still have an attachment to their "un-deceased" browsers (e.g. IE6), rather than shooting them up with "Upgrade your browser now!" messages or forcing them into a wasteland of zero tolerance, we keep them alive through hacks and special stylesheets — the developer equivalent of how Shaun from the movie kept his best friend, zombie Ed, alive in his shed.

Rather comical perhaps — but in many ways, some of us go out of our way to give leniency towards zombie browsers.

Zombie Technologies

Whereas we can easily spot the zombie browser — they stagger around the web confused at what CSS3, HTML5, and other modern standards mean — one of the more frustrating types of zombies are web technologies and standards that have already died, but developers still cling onto.

One perfect example of a zombie technology is Wireless Markup Language (WML). Due to the evolution of the smartphone market, modern mobile devices can now render regular HTML.



The peak days of WML may be over, but the BBC still shows this web zombie some love!

While WML itself is deprecated (W3C's way of pronouncing something dead) — and let's face it, it wasn't exactly the real web in the first place — there are still some with old mobile phones wanting to access the web even through a subpar viewing experience.

To this day, there are still developers who insist on providing or maintaining WML versions of their website to cater to this zombie technology, and while their care for users with old cell phones is admirable, their contribution to the proliferation of a zombie web standard is not.

Testing your website using a variety of older handsets shows how bad things are getting.

Old technologies being replaced by new ones is nothing new to the nature of the web. And I suppose that like web browsers, there will become an epidemic point where the number of undead languages goes far beyond the number of living ones, which may be problematic for beginners deciding what they need to learn.

The case of undead technologies isn't so much of an issue of support — as we modern web developers tend to comply with current web

standards — but that of excess baggage that the web's future is going to have to deal with.

Zombie Code

This zombie is something which most of us want to see dealt with in the harshest possible manner because it's something that we have control and choice over.

While undead languages maintain some level of support for the sake of older browsers or devices, using deprecated HTML tags (e.g. <font>, <marquee>, <blink>) and non-standard/proprietary CSS (e.g. -ms-overflow-y) to solve today's design tasks becomes proof of poor quality craftsmanship and thought by certain developers.

Revenge of the fallen markup -- deprecated code still exists in modern web designs.

While we may consider zombie code as just an annoyance, let's be clear and state they're not completely benign.

The most worrying thing about zombie code is the danger of future browsers stopping the support of these deprecated and non-standard coding practices. What happens to these sites? They will still be floating around in cyberspace, waiting to be visited by a potential client, who'll later come to us asking for their site's logo to blink and scroll.

From past experience, I know of developers even today who still maintain and produce websites (professionally, I might add) using the kind of source code we would have expected to see in the early 90s

— and it shocks me just as if I saw a real zombie straight out of 28 Days Later.

```
    <td valign="top" align="center" height="313"
colspan="2"><table width="100%" border="0" cellspacing="0"
cellpadding="0">
      <tr>
        <td valign="middle" align="center"><table width="100%"
border="0" cellspacing="0" cellpadding="0">
            <tr>
              <td valign="top" align="center" width="50%"
height="505"><table width="423" height="311" border="0"
cellpadding="0" cellspacing="0">
        <td bgcolor="#FFFFFF" align="center" valign="top"><img
        src="images/buttonsdropshadow.gif" width="816" height="8"
        alt="button drop shadow" /></td>
        </tr>
```

Separating structure from style is the modern convention, yet zombie code still works in modern browsers

In much the same way as that of dead browsers or dead technologies, education will ultimately be the way to combat this epidemic of outdated code — code that "works" but does so using undead coding habits.

The number of casualties of the original browser wars has served us a lesson of what happens when militant code becomes so disproportionate that web professionals are forced to deal with each browser individually (with the mobile device war, it could happen again).

Zombie Design Practices

Finally, we have something that is near and dear to my heart — the sympathetic case of what could easily qualify as design zombies. We all remember the days of the early web: Table-based layouts (a zombie practice still widespread), obtrusive JavaScripts, spacer gifs, statistics counters, flashing banners, animated clipart, "designed for" banners, phony website awards and background music (often blended to form an epileptic massacre of color).

While it could be seen that many of these practices have evolved into new strains, the issue of outdated design is as apparent today as ever.

Sites as bad of this can still be found on the web, and in many cases they're still maintained!

Design is one subject that — with the web's evolution — has managed to maintain a level of historical value with itself. If you've ever visited a newly launched website and thought, "Wow, this website looks retro in a bad way" — that's a sign that you're on a site designed using undead practices.

While zombie designs seem insignificant — as the code can itself be very well crafted using best practices and standards — they do nourish a sentiment of a lack of regard towards usability, accessibility, user experience and modern aesthetic appeal, making the design zombie an interesting foe.

Another well-intentioned website, with the aesthetic design value of a zombie.

In Buffy the Vampire Slayer, when asked to spot a vampire, Buffy looked for people in the club wearing seriously outdated clothes. While this is funny — it's also true that having something so old looking that we could probably carbon date it will ultimately affect our users' experience.

Education (surprise, surprise) seems to be the best way forward in eliminating undead designs.

The Circle of Life

With future standards like HTML5 and CSS3 emerging, brand new zombies from the array of existing standards will continue to rise.

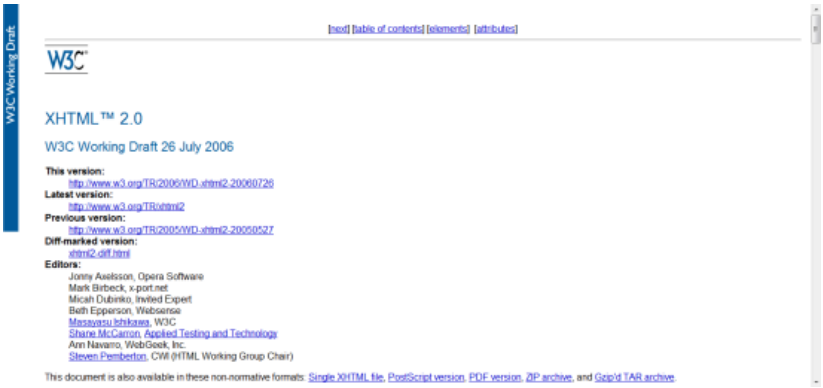
Maintaining a skill set and knowledge base that is up to date — and staying ahead of the curve — is the best way to avoid the reoccurrence of zombie practices and habits.

And while some of the web's afterlife will continue to exist without causing too much harm, there comes a time where such undead beings can ultimately lead us into a spiral of escalating annoyance and rot.

Details about standards aren't that hard to come by when you know where to look.

It's worth pointing out that the web has an interestingly rich history full of technologies which, though ousted by something newer, may still hold a place in our world.

While in a perfect world, the transcendence from one to the next should be the ideal solution, newborn standards (like XHTML 2.0) can die before their time. As such, don't think of zombies simply as the old stuff — they can be new stuff that didn't quite fully form yet but may have been early-adopted by some. Perceptions can lead to accidental shootings and you don't want to give the death sentence to a practice that has legitimate value.



XHTML 2.0 unfortunately didn't make it to fruition and thus became a newborn zombie.

Old standards die and new standards appear in their place — that's just the way of the web. The circle of life is well intentioned, it moves us forward to bigger and brighter things. The solution isn't to stop innovation — that's just crazy — but culling the ever-increasing zombie population that still exists.

As an industry, it's our duty to use what skill, knowledge and network we have to push back the zombie invasion. And while I'm not saying you should go after IE6 users with holy water and a crucifix, you could take a more civil approach through education and conversations.

If you know someone with web zombies, why not spend a few minutes explaining the problem and helping them make an informed

choice? Every outdated element on the web we can eliminate is worth fighting against. Especially if we don't want the web to be a haunted graveyard.

Sources:

- https://techcrunch.com/2010/03/05/ie6-funeral/
- https://en.wikipedia.org/wiki/Wireless_Markup_Language
- http://www.zeldman.com/2009/07/02/xhtml-wtf/

5 Web Files That Will Improve Your Website

The amount of code that developers encounter regularly is staggering. At any one time, a single site can make use of over five different web languages (i.e. MySQL, PHP, JavaScript, CSS, HTML).

There are a number of lesser-known and underused ways to enhance your site with a few simple but powerful files. This article aims to highlight five of these unsung heroes that can assist your site. They're pretty easy to use and understand, and thus, can be great additions to the websites you deploy or currently run.

An Overview

Which files are we going to be examining (and producing)? Deciding which files to cover was certainly not an easy task for me, and there are many other files (such as .htaccess which we won't cover) you can implement that can provide your website a boost.

The files I'll talk about here were chosen for their usefulness as well as their ease of implementation. Maximum bang for our buck.

We're going to cover robots.txt, favicon.ico, sitemap.xml, dublin.rdf and opensearch.xml. Their purposes range from helping search engines index your site accurately, to acting as usability and interoperability aids.

Let's start with the most familiar one: robots.txt.

Robots.txt

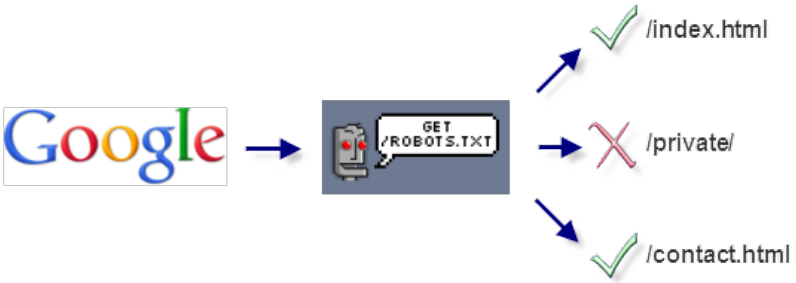
The primary function of a robots.txt file is to declare which parts of your site should be off-limits for crawling.

By definition, the use of this file acts as an opt-out process. If there are no robots.txt for a directory on your website, by default, it's fair game for web robots such as search engine crawlers to access and index.

While you can state exclusion commands within an HTML document through the use of a meta tag (<meta name="robots"

content="noindex" />), the benefits of controlling omitted pages through a single text file is the added ease of maintenance.

Note: It's worth mentioning that obeying the robots.txt file isn't mandatory, so it's not a good privacy mechanism.



This is how the robots.txt file interacts between a search engine and your website.

Creating a Robots.txt File

To create a robots.txt file the first and most obvious thing you will need is a text editor. It's also worth pointing out that the file should be called robots.txt (or it won't work) and it needs to exist within the root directory of your website because by default, that's where web robots look for the file.

The next thing we need to do is figure out a list of instructions for the search engine spiders to follow. In many ways, the robot.txt's structure is similar to CSS in that it is comprised of attribute and value pairs that dictate rules.

Another thing to note is that you can include comments inside your robots.txt file using the # (hash) character before it. This is handy for documenting your work.

Here's a basic example telling web robots not to crawl the /members/ and /private/ directory:

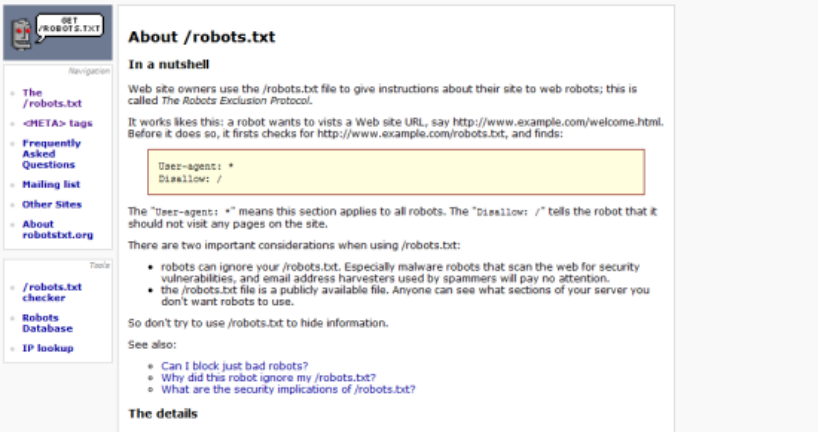
```
User-agent: *
Disallow: /members/
Disallow: /private/
```

The robots.txt exclusion standard only has two directives (there are also a few non-standard directives like Crawl-delay, which we'll cover shortly).

The first standard directive is User-agent. Each robots.txt file should begin by declaring a User-agent value that explains which web robots (i.e. search crawlers) the file applies to.

Using * for the value of User-agent indicates that all web robots should follow the directives within the file; * represents a wildcard match.

The Disallow directive points to the folders on your server that shouldn't be accessed. The directive can point to a directory (i.e. /myprivatefolder/) or a particular file (i.e. /myfolder/folder1/myprivatefile.html).



There is a specification for robots.txt, but the rules and syntax are exceptionally simple.

Robots.txt Non-Standard Directives

Of course, whilst having a list of search engines and files you want hidden is useful, there are a few non-standard extensions to the robots.txt specification that will further boost its value to you and your

website. Although these are non-standard directives, all major search crawlers acknowledge and support them.

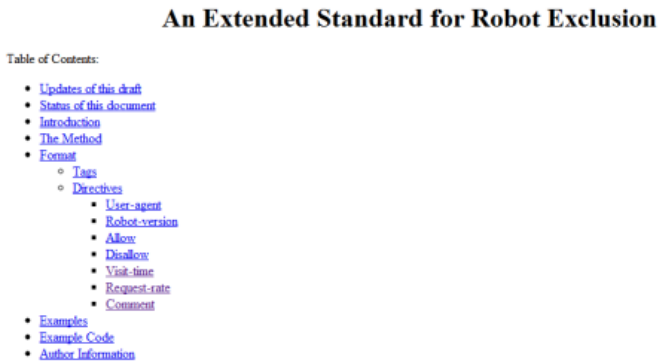
Some of these more popular non-standard directives are:

- **Sitemap:** where your Sitemap.xml file is
- **Allow:** opposite of Disallow
- **Crawl-delay:** sets the number of seconds between server requests that can be made by spiders

There are other less supported directives such as Visit-time, which restricts web robots to indexing your site only between certain hours of the day.

Here's an example of a more complex robots.txt file using non-standard directives:

```
Allow: /private/public.html
Comment: I love you Google, come on in!
Crawl-delay: 10
Request-rate: 1/10m # one page every 10 minutes
Robot-version: 2.0
Sitemap: /sitemap.xml
Visit-time: 0500-1300 # military time format
```



An Extended Standard for Robot Exclusion

Table of Contents:

- Updates of this draft
- Status of this document
- Introduction
- The Method
- Format
  - Tags
  - Directives
    - User-agent
    - Robot-version
    - Allow
    - Disallow
    - Visit-time
    - Request-rate
    - Comment
- Examples
- Example Code
- Author Information

Whilst not a standard, there is an extension for robots.txt which has mainstream support.

Favicon.ico

A favicon (short for "favorites icon") is a small image (like a desktop application's shortcut) that represents a site.

Shown in the browser's address bar, the favicon gives you a unique opportunity to stylize your site in a way that will add identity to browser favorites/bookmarks (both locally and through social networks).

The great thing about this file is that every major browser has built-in support for it, so it's a solid extra file to provide.

This is how the favicon.ico file affects your site visually through the browser.

Creating a Favicon.ico file

To create a favicon, you'll need an image or icon editor. I am a fan of Axialis IconWorkshop, but there are free editors like IcoFX that do the job well.

You can also find quite a few free online favicon tools by viewing this list of web-based favicon generators.

You need to have a 16x16px icon (or 32x32px, scaled down) that matches what you want to see in the browser.

Once you are done creating your icon's design, save the file as "favicon.ico" in the root directory of your web server (that's where browsers look for it by default).

Note: It's a good idea to use the .ico file type, as some browsers don't support PNG, GIF or JPEG file types.

To make this file work properly, refer to the location of your favicon in the <head> tags of all your HTML documents, as such:

```
<head>
    <link rel="shortcut icon" type="image/vnd.microsoft.icon"
    href="favicon.ico" />
<head>
```

The rel attribute values of "shortcut icon" or "icon" are considered acceptable and the MIME type of "vnd.microsoft.icon" (as of 2003) replaced the older type ("image/x-icon") as the official standardized favicon MIME type for .ico files on the web.

Note: While Internet Explorer (and some other browsers) will actively seek out your favicon in the root directory of your site by default (which is why you should have it there), it's worth adding the above code into the <head> of your HTML just to make it explicitly known by other types of browser agents.

There are multiple online tools which can create a favicon from existing images.

Favicon's in Apple Devices

Another standard (of sorts) has appeared in light of Apple's iPod, iPad, and iPhone. In this situation, you can offer a 57×57 PNG, ICO or GIF file (alpha transparency supported) that can be displayed on the devices' Home screen using the web clip feature.

Apple also recommends that you use 90-degree corners (not rounded corners, which the devices will do for you automatically) to maintain the "feel" of such icons.

To make this file work properly, place the following code into every page within your <head> tags:

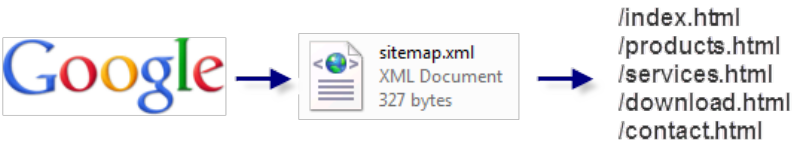
```
<head>
    <link rel="apple-touch-icon" href="images/icon.png" />
</head>
```

For users of Apple devices, a specially produced "favicon" can be produced.

Sitemap.xml

One thing website owners worry about is getting their website indexed correctly by the major search engines like Google.

While the robots.txt file explains what files you want excluded from results, the Sitemap.xml file lists the structure of your site and its pages. It gives search engine crawlers an idea of where things are on your site.



This is how the Sitemap.xml file interacts between a search engine and your website.

As always, the first recommended course of action to produce a Sitemap is to create the XML file that will contain its code. It's recommended that you name the file as "sitemap.xml" and provide it within the root directory of your website (as some search engines automatically seek it there).

It's also worth noting that while you can submit your Sitemap file location directly to search engines, adding the non-standard Sitemap directive to your robots.txt file can be useful as it's widely supported and gives spiders a push in the right direction.

Below is a basic example of how a Sitemap looks like.

```
<?xml version="1.0" encoding="UTF-8"?>
    <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/
    0.9">
    <url><loc>index.html</loc></url>
    <url><loc>contact.html</loc></url>
</urlset>
```

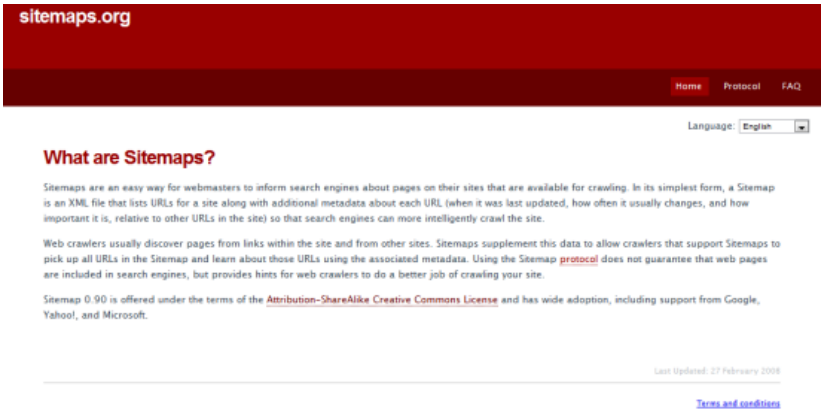
Each Sitemap file begins with a Document Type Definition (DTD) that states that the file is UTF-8 encoded, written in XML, and uses the official Sitemap schema.

Following those formalities, you simply produce a list of your URLs that exist within your website's structure.

Each URL must be contained within two elements: <url> and <loc>. This is a very simple specification to follow, so even less experienced developers should be able to replicate this basic mechanism with little effort.

To reference your Sitemap inside your HTML documents, place this code between the <head> tags:

```
<head>
    <link rel="sitemap" type="application/xml" title="Sitemap"
    href="sitemap.xml" />
<head>
```



Just like most XML-based schemas, there is a protocol and specification to follow.

Other Sitemap Tags

While you could limit yourself to simply listing every file, there are a number of other meta-information that can be included within the <url> tag to help further define how spiders deal with or treat each page in the site — and this is where the Sitemap's true power lies.

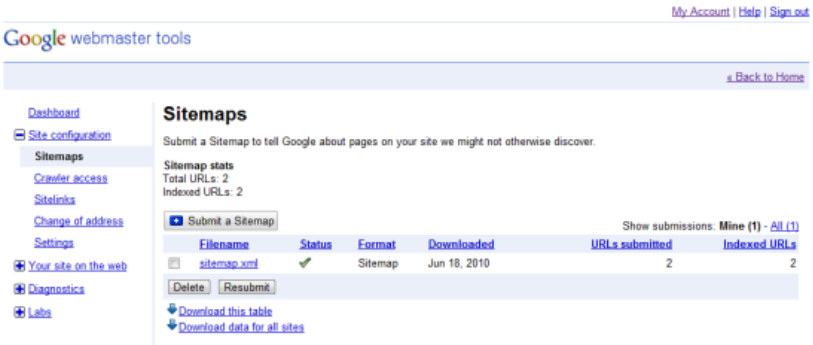
You can use <lastmod>, for example, to state when the resource was last modified (formatted using YYYY-MM-DD). You can add the <changefreq> element, which uses values of always, hourly, daily, weekly, monthly, yearly, and never to suggest how often a web page

changes (for example, the front page of Six Revisions has a value of daily).

There is also the <priority> tag, which uses a scale of 0.0 to 1.0 that you can utilize to indicate how important a web page is to a website.

Here's an example of using the above tags:

```
<lastmod>2010-05-13</lastmod>
<changefreq>monthly</changefreq>
<priority>0.8</priority>
```



Google allows you to submit your Sitemap to initiate its analysis of your site structure.

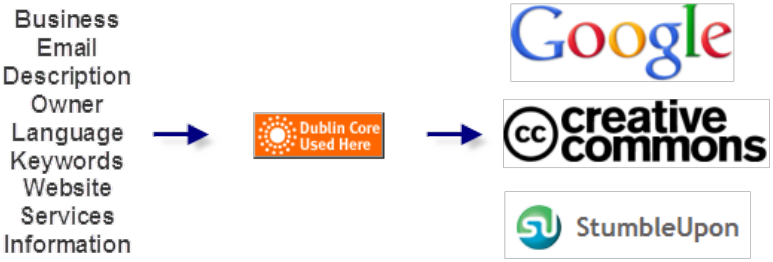
Dublin.rdf

Ensuring you provide metadata has become big business among SEO professionals and semantics advocates. The appropriate use of HTML, metadata, microformats and well-written content improves the chances of appearing in the right search results. They also allow an increasing number of browsers and social networks to aggregate and filter the data so that they can accurately understand what your content represents.

The Dublin.rdf file acts as a container for officially recognized meta elements (provided by the DCMI specification) which can augment the semantic value of the media you provide.

If you've ever visited a library and tried to locate a book, you know that you will often have to flick through the library catalogs to find

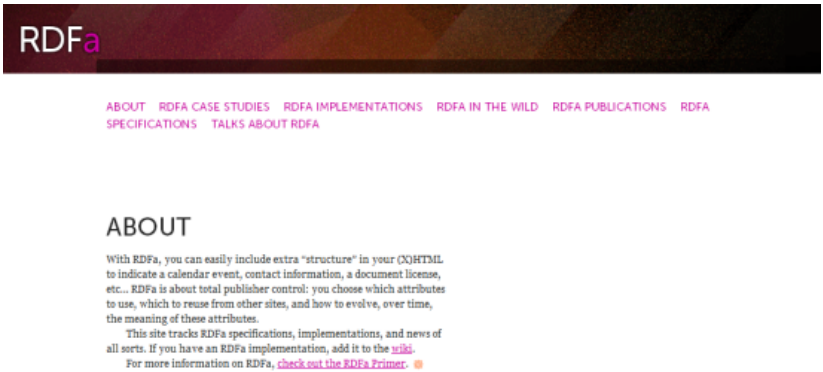
books based on their subject, their author, or perhaps even their title. The aim of the DCMI is to produce such a reference card for your website that will help search engines, social networks, web browsers, and other web technologies understand what your site is.



This is how the Dublin.rdf file interacts with supporting social networking mediums.

Creating a Dublin.rdf File

To begin, you need to produce the file itself (which we shall name "Dublin.rdf"). In order to maintain consistent meta details about the site (as opposed to individual DCMI meta tags for specific pages and resources), we shall create an RDF file (formatted as XML) with a reference within the HTML document to indicate that the information is available. While you can embed DCMI meta tags within HTML, RDF allows you to cache the data.



This is how the OpenSearch file interacts with your site through the browser.

When a supporting spider or other resource that acknowledges the DCMI core sees the file, they can cache and directly relate to the information. This doesn't mean you shouldn't use traditional meta tags, but the file can serve as a useful supplement.

```
<?xml version="1.0"?>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
    ns#" xmlns:dc= "http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://www.yoursite.com/">
    <dc:contributor>Your Name</dc:contributor>
    <dc:date>2008-07-26</dc:date>
    <dc:description>This is my website.</dc:description>
    <dc:language>EN</dc:language>
    <dc:publisher>Company</dc:publisher>
    <dc:source>http://www.yoursite.com/</dc:source>
    </rdf:Description>
</rdf:RDF>
```

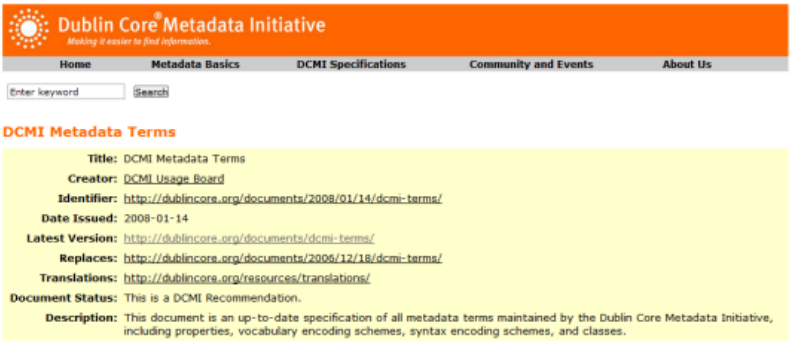
Like most XML files, this RDF document has a DTD — and within that, you have the description element (which links to the resource being referenced).

Within the description, as you can see from the above, there are several elements (beginning with the prefix of dc:) — these hold the metadata of the page.

There's a whole range of terms you can add (see this list of DCMI metadata terms), it's simply a case of adding the term's name, then giving a value as denoted by the DCMI specification. You'll end up with a library of useful data that can improve your site's semantics and interoperability with other sites and applications!

To make this file work properly, place the following code into every HTML document within the <head> tags:

```
<head>
    <link rel="meta" type="application/rdf+xml" title="Dublin"
    href="dublin.rdf" />
<head>
```



The Dublin.rdf file makes use of the DCMI specification to provide meta information.

OpenSearch.xml

The ability to search a website is one of the most important ways people locate content.

The OpenSearch file allows you to add a custom search engine listing (on your own site) through the search feature that appears in all modern browsers. All of the major browsers can take advantage of OpenSearch; it's pretty durable.

While you will still want to provide a search mechanism on your website, this core enhancement complements the user's in-browser search capabilities.

This is how the OpenSearch file interacts with your site through the browser.

Like with all things we've discussed thus far, we need to produce the file for the code to be placed in.

As this particular type of file doesn't have assumed name reservations like robots.txt or sitemap.xml, we could call the file whatever we like. However, the convention for OpenSearch files is to name the file, "opensearch.xml".

You'll want to include the code below as your starting template, then proceed to customizing the required tags such as <ShortName>, <Url> and <Description> (they are case-sensitive) to describe your site.

The example used below is for Six Revisions using Google Search.

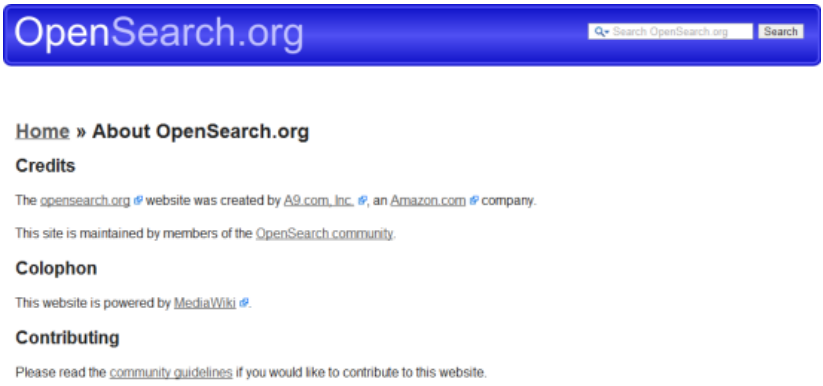
```
<?xml version="1.0" encoding="UTF-8" ?>
    <OpenSearchDescription xmlns="http://a9.com/-/spec/
    opensearch/1.1/">
    <ShortName>Six Revisions</ShortName>
    <Description>Search this website.</Description>
    <Image>favicon.ico</Image>
    <Url type="text/html" template="http://www.google.com/
    search?
    sitesearch=http%3A%2F%2Fwww.sixrevisions.com%2F&amp;as
    _q={searchTerms}"/>
</OpenSearchDescription>
```

The tags included above are:

- **ShortName:** the title you want for your search extension
- **Description:** explains the purpose of the search box
- **Image:** this isn't required like the others, but I recommend referencing your Favicon with it so the search feature has a unique icon
- **Url:** requires a MIME type and a template attribute which links to the search terms

To make this file work properly, place the following code into every page within the <head> tag:

```
<link rel="search" type="application/opensearchdescription+xml"
title="Website" href="opensearch.xml" />
```



This is how the OpenSearch file interacts with your site through the browser.

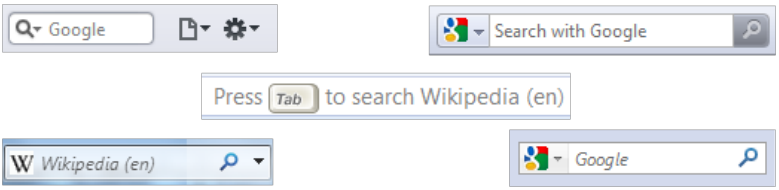
Other OpenSearch Tags

There's a range of additional tags we can provide. Among these are:

- **AdultContent:** if the site has adult material needing to be filtered, set to false
- **Attribution:** your copyright terms
- **Contact:** an email address for the point-of-contact of your site
- **Developer:** who made the site?
- **InputEncoding and OutputEncoding:** The MIME type used
- **Language:** i.e. EN for English
- **Query:** for more detailed search terms
- **Tags:** keywords, separated by a space
- **SyndicationRight:** The degree to which people can request, display or send results

Example usage of these other tags:

```
<AdultContent>false</AdultContent>
<Attribution>Copyright, Your Name 2010, Some Rights Reserved.</
Attribution>
<Contact>None@none.com</Contact>
<Developer>Your Name</Developer>
<InputEncoding>UTF-8</InputEncoding>
<Language>en-us</Language>
<OutputEncoding>UTF-8</OutputEncoding>
<Query role="example" searchTerms="terms" />
<Tags>Example Tags Element Website</Tags>
<SyndicationRight>open</SyndicationRight>
```



This is how the OpenSearch file interacts with your site through the browser.

Simple, Small and Effective

While this guide represents a crash course in producing these useful files, it's worth pointing out that taking the time to understand the syntax of any language is important in order to determine what the impact of these files on your website.

These files represent a truth that there's more to a website than HTML, CSS, and JavaScript, and while producing these files will certainly not act as a replacement for your existing code workflow, their inherent benefits make them worthy of consideration to supplement your projects. Give them a try for yourself!

Sources:

- https://en.wikipedia.org/wiki/Web_crawler
- http://www.robotstxt.org/
- http://www.conman.org/people/spc/robots2.html
- https://www.axialis.com/iconworkshop/
- https://www.kodlian.com/apps/icon-slate
- https://www.favicon.cc/
- https://www.sitemaps.org/protocol.html
- https://www.google.com/webmasters/tools/home
- http://dublincore.org/
- http://dublincore.org/documents/dcmi-terms/
- http://www.opensearch.org/Home

A Guide on Layout Types in Web Design

One of the most variable aspects of web design is the way in which we approach width and height in terms of measurements and flexibility.

For many years, we have rotated between the benefits and pitfalls of using fixed, elastic, and liquid measurements in a quest to give optimal viewing experiences in highly varied situations, while balancing our need to control things in our web pages.

But, as Bob Dylan proclaimed a long time ago, "The times, they are a-changin'," and with these changes come a variety of new ways for laying out your website's pages and an even more variable landscape of methods for viewing websites.

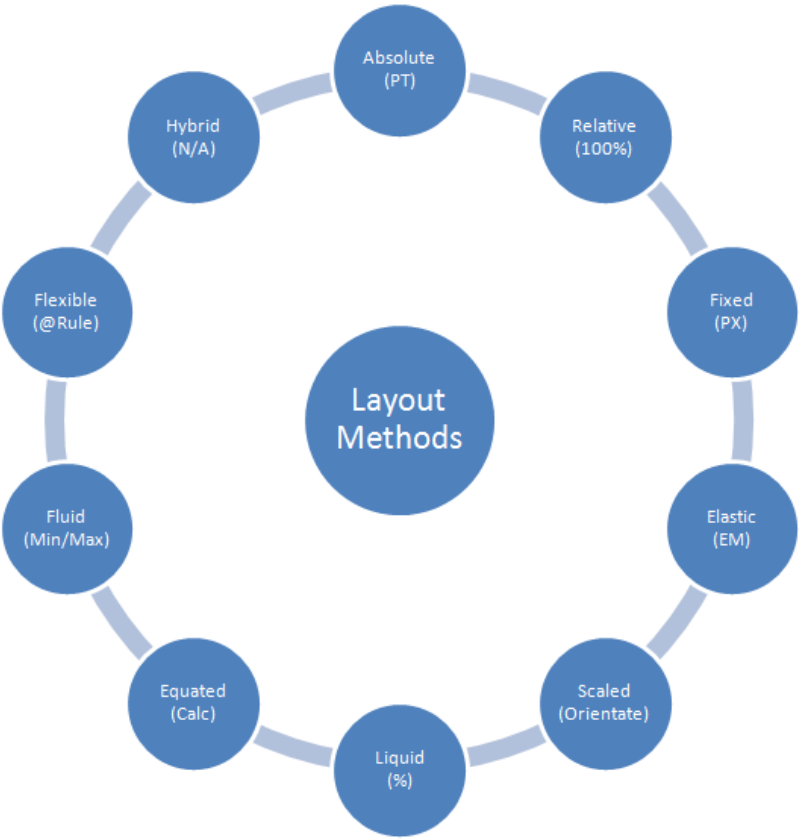
In this article, we will examine web layout types — old, new, and the future. We will explore the subject in the context that websites are being viewed in a diverse amount of ways, such as through mobile phones, netbooks, and touchscreen personal devices like the iPad.

About Your Options

Let's set our objectives for this exploration of layout types:

- We shall examine the variety of options that exist
- For each layout type, I'll try to suggest some situations they are best used in
- The pros and cons of a layout type compared to others

We will discuss 10 types of web layouts.



While pixel perfection is a pipe dream, there's more to layouts than fixed, liquid or elastic!

The main lesson to take away from these choices is to think carefully about why an option is suitable for a particular situation and how your choice will affect your audience.

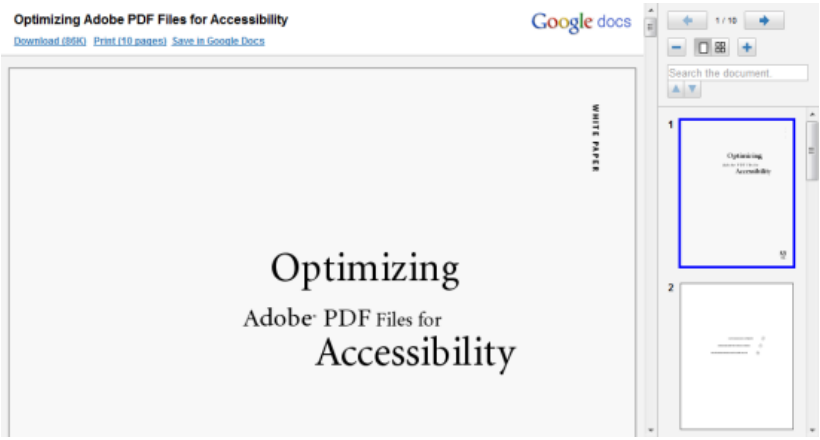
Let's dig in, starting with absolute layouts.

Absolute Layouts

One of the least commonly used methods of measurement employed in web design is absolute measurement (i.e. inches, cm, mm and picas). Absolute units and positioning is traditionally found in print media, which natively use these units of measurement.

The conversion of print to web format can be seen in word processing software such as Microsoft Word, which still uses these conventions when formatting text and sizing the dimensions of a document in order to make it appear as close as possible to printing on paper.

Absolute layouts have limited use in web designs.



A use for absolute layouts on the web is for PDF documents where content remains static.

Of course, just because it isn't popular doesn't mean it doesn't have its place on the web designer's bevy of options. If you are someone who utilizes printer-friendly stylesheets — yes, people do still print web pages — the absolute measurements of cm, mm, inches, and pt can help you prepare a page layout for printers more accurately.

Relative Layout

Relative positioning and layouts adjust in size depending on the size of the user's browser viewport.

The area inside the red border is the browser's viewport. You can change the size of the viewport by resizing the window. Different monitor sizes have various maximum sizes for the view port.

Typically, this type of layout relies on everything working at 100% width, whether it's a small screen (like a netbook) or a 24-inch widescreen desktop monitor. This means that the layout will scale according to the viewer's situation.


useit.com → Alertbox → Aug. 2002 Font size | Search

Jakob Nielsen's Alertbox, August 19, 2002:

Let Users Control Font Size

Summary:
Tiny text tyrannizes users by dramatically reducing task throughput. IE4 had a great UI that let users easily change font sizes; let's get this design back in the next generation of browsers.

Sometimes technological progress backfires, and the "better" technology turns out to be worse for users. The Web is no stranger to this problem, and has experienced many innovations that would have been best avoided. Examples include frames, changing the color of browser scrollbars, and scrolling text.

Another example of harmful Web technology comes with the increasing use of style sheets, which let web designers specify the exact size of text down to the pixel. Unfortunately, many designers are using this ability, leading to **reduced readability of an increasing number of websites**.

The State of Font Control

I'm hereby launching a campaign to get Microsoft to **make user preferences override** any fixed font size specification in Web designs.

It may be okay for the browser to initially render the page with the designer's text size, but users should be able to easily enlarge text, no matter what the style sheet says. After all, it's my screen, my computer, and my software, and they should do what I say.

Granted, some web browsers have a geeky feature that lets users specify their own style sheets. Fine for experts,

Very few sites make use of 100% widths, but it does work.

Fixed Layout

Commonly regarded as one of the least flexible methods of laying out a web design, the use of pixel-based measurements has almost a digital resonance associated with it that transfers across from the print industry, in that the medium relies on fixed/static measurements.

This unit of measurement is accurate and leaves little guessing as to how a web design will appear across different web browsers and has become exceptionally popular among sites that favor control and predictability over optimizing the layout for the audiences' particular viewing situation.

A fixed width layout is used on Six Revisions.

We all know that problems can arise from having to scroll in all sorts of directions, and the fixed measurement of a px-based layout has this general issue in spades.

While many people seek out some sort of ideal width to ensure maximum compatibility, it's worth mentioning that if you use a lot of elements that require fixed layout rules like non-repeating background images or borders with other non-relative elements, fixed measurement layouts can do the job well and act as the best all-around solution.

Elastic Layout

One of the most used methods of laying out a design's content is using the relative em unit of measurement.

Commonly referred to as an elastic layout design (due to the way it flexes by growing and shrinking to meet the content's needs), it has shown a great deal of appreciation within the web design community due to its ability to scale content, text sizes, and such.

Unlike with fixed units of measurements where absolute-unit elements like images are best suited (due to maintaining without distorting), elastic layouts work best when flexible content (such as text blocks)

takes the front seat (though there are ways to have images scale elastically as well).

Popular for its elastic nature, em measurements are recommended for font sizes.

Of all the methods listed, the elastic layout type is the most subservient to your content as it gives the content itself the deciding position as to how the layout should scale.

Making the text smaller in such a design will reduce the width or height, and enlarging the text will have the opposite effect.

This unique attribute allows the layout to resize based on the content rather than the needs of the layout.

Using an elastic solution is perfect if you want the layout to be determined by the content, but it can have issues if the text scales beyond the viewport (causing unwanted horizontal scrolling).

Scaled Layout

One of the latest methods in CSS3 allows the manipulation of the available viewport around certain device orientations (i.e. portrait and landscape).

Depending on the way in which the device is held, the design has the potential to alter its visual layout (altering the amount of space given to the content itself).

Unlike the others, this type of layout does not rely on measurement units, but rather a specific layout type. However, this notion shouldn't be underestimated as a way of dealing with complex columns on small screens.

10 years ago, we wouldn't have considered a screen's orientation. How times have changed!

Scaled layouts truly shine in the smartphone market where the display can be rotated or moved frequently (such as the iPhone, for example).

The iPhone adjusts orientation of your websites on-the-fly.

With such limited space being available on handheld mobile devices, you cannot only maximize the way your pixels are allocated, but you can also allow people the option to choose whichever method they prefer to visualize the information.

Each person will use his or her web-enabled mobile device in a different way, and by allowing your design to relate your content in a transformative way depending on the orientation, you can maximize the usability of your content.

Liquid (or Fluid) Layout

The most relaxed method of providing a dynamically expanding or contracting design makes use of the ever-popular percentage (%) unit measurement.

This layout type has gained mass popularity because it is the ultimate way of allowing the total opposite of a fixed layout where the content will simply take whatever space is available to it.

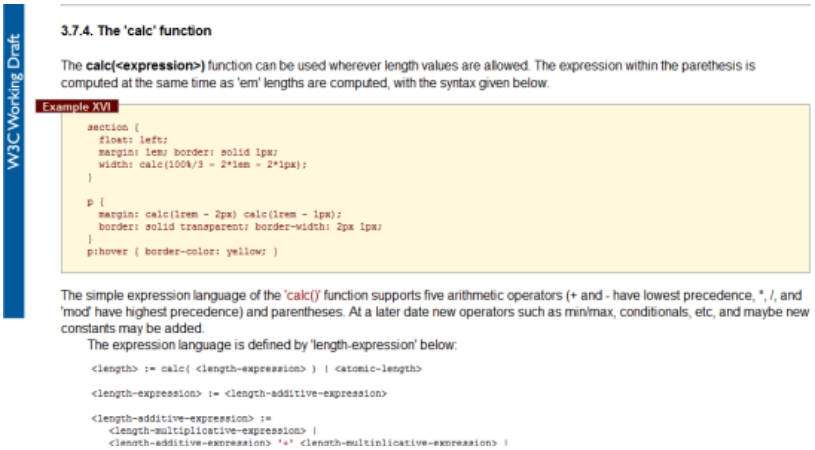
Percentages require careful calculation as you can't give more than 100% without issues!

The limited guarantees you hold on the viewport being used goes beyond screen resolutions (imagine your site on a 6-inch screen versus a 100-inch screen, even just at 80% width).

Though it goes without saying that a liquid layout is useful in almost every web-based situation because it adjusts its width depending on how big or small the user's viewport is — so it's definitely worth looking into.

Equated Layout

The next method of laying out content we shall look at is the equated layout, which makes use of a new CSS function called calc (see W3C calc spec).



W3C Working Draft

3.7.4. The 'calc' function

The **calc(<expression>)** function can be used wherever length values are allowed. The expression within the parethesis is computed at the same time as 'em' lengths are computed, with the syntax given below.

Example XVI

```
section {
  float: left;
  margin: 1em; border: solid 1px;
  width: calc(100%/3 - 2*1em - 2*1px);
}

p {
  margin: calc(1rem - 2px) calc(1rem - 1px);
  border: solid transparent; border-width: 2px 1px;
}
p:hover { border-color: yellow; }
```

The simple expression language of the 'calc()' function supports five arithmetic operators (+ and - have lowest precedence, *, /, and 'mod' have highest precedence) and parentheses. At a later date new operators such as min/max, conditionals, etc, and maybe new constants may be added.

The expression language is defined by 'length-expression' below:

```
<length> := calc( <length-expression> ) | <atomic-length>

<length-expression> := <length-additive-expression>

<length-additive-expression> :=
   <length-multiplicative-expression> |
   <length-additive-expression> '+' <length-multiplicative-expression> |
```

When this measurement capability reaches browsers, a new level of control will exist.

While the previous layouts we've covered rely on specific widths or heights being provided, an equated layout allows you to mix a fixed and relative value by using a calculation like width: calc(50% - 200px).

Have you ever had a situation where you wished that you could make up the full 100% but also account for things like divs with borders and elements that have fixed widths (such as an image)? If you're anything like me, it's certainly something that has crossed your mind.

The calc CSS3 function, which has not been widely adopted yet (but is part of the CSS3 spec) may just be the thing you are looking for. While the function still isn't widely supported by existing web browsers, this can be a future-forward option for building layouts with an added layer of pliancy.

Fluid-Min/Max Layout

A common problem that we have as designers is that whenever the amount of space we have becomes either too wide or too narrow (or too tall or too short), the relatively-measured and flexible content we have gets too diluted or too compressed (which is bad news).

Using minimum and maximum widths (or heights), you can set limits on how much the design can scale so that you can still have flexibility — but only to a certain extent. Rather than spanning the viewport like a liquid layout, this layout type flows only up to where it's told ('atta boy).

A fluid/"jello" layout will scale only to a certain fixed width or height.

If there's one thing that causes problems with layouts, it's us making assumptions as to the amount of space that we will have available for our design elements.

The benefits of the CSS min-width, max-width, min-height and max-height properties are most widely noticed when you want your layout to be confined within certain dimensions (like within a fixed-width design) but don't want to suffer the wrath of horizontal scrolling.

For example, if you wanted to have your width scale to 100% for small screens but only up to, say, 1,500px so that your layout doesn't get too wide for larger screens, then you can use a max-width:1500px.

As this method of laying out a web page provides a safety net that browsers can rely on (based on the min and max values you supply), you can give your fixed work a bit of added flexibility.

Conditional Layout

With the rise in devices like the iPhone, a need has appeared for a way of altering web designs beyond conventional layouts to ensure that mobile device users can have an optimized experience.

The ability to serve a unique stylesheet based on the device or viewport width and height (through CSS3's media queries) gives rise to an even more flexible and friendly way to represent your site's content. This layout type is something I'd like to call "conditional layout."

The above design uses CSS3 media queries to scale the design down as required.

Of all the methods of laying out information that have appeared recently, this is by far the one with the most promise (once the browser compatibility issues are ironed out).

Most website designs rely on a single stylesheet. Using CSS3 media queries (especially with mobile and desktop experiences) can bring conditional layouts to best meet the user agent.

The downside of this is that it means you will need to develop and maintain stylesheets for particular devices — much like how you, in the past, maintained IE-specific stylesheets.

Hybrid Layout

Of course, while mentioning all of these layout types, we can't forget to mention the most popular layout method of all — the hybrid layout pretty much stands by its name in that the design ends up using a mixture of various layout types. This includes mixing and matching various units and concepts to ensure that the design adapts to the browser's viewport only when it needs to and still be able to retains a certain level of control over parts of a website that need more fixed structures.

While it requires you to be more thoughtful over your work, it's possibly the smartest way to design and develop.

Most sites don't stick to one measurement type, they hybridize based on needs.

Most websites make use of a hybrid layout because certain measurement units are useful for certain situations. While many people still cling to the idea that there is one perfect layout method

waiting to be found, I think that the hybrid will overcome situational issues by blending together the best of all worlds.

Perhaps you might end up with an absolute layout in your print stylesheet, and maybe you might have fixed widths using a liquid body with elastic content and a fluid control for the outside edges with scaled and flexible support for certain devices — the combinations are bountiful!

The Bigger Picture

Clearly, there are many options to consider when laying out your web pages, and thus it makes sense — both pragmatically and theoretically — to pay close attention to the details and scope of any design project you undertake.

Which layout type you utilize to produce your website is something that deserves as much attention as the fonts you use or the color theme you put together.

It's also worth highlighting that there's no perfect way to deal with every situation and therefore there's no one type that is universally the best for all situations.

There's no right or wrong way to design, but careful thought can improve some situations.

Design is one of the most fundamental skills that any web professional must get to grips with. The way the Internet is being consumed is rapidly evolving, with wide disparities in both the devices we employ and the tools we take advantage of.

There's more to contend with than good usability, accessibility, web copy, color contrast, and so forth. A good website must meet an ever increasing number of needs and thus the search for the perfect layout has become a Holy Grail quest of sorts for web designers.

While times are changing (as do situations), picking the right layout right now should be done methodically.

Sources:

- http://www.braillenet.org/colloques/Bnet2000/access.pdf
- http://www.csszengarden.com/063/
- http://www.csszengarden.com/001/
- https://www.w3.org/TR/css-values-3/#calc-notation
- http://www.positioniseverything.net/articles/jello-expo.html
- https://www.w3.org/TR/css3-mediaqueries/
- https://jonikorpi.com/
- https://www.apple.com/iphone/
- http://green-beast.com/blog/?p=199

Reductionism in Web Design

In the field of design, the phrase "complexity is the enemy" speaks to how keeping things simple makes our work more functional.

With the modern crop of technologies that dole out increasing amounts of functionality, it's important that we take the time to ensure a balanced level between oversimplification to the level that insults our visitor's sense of competency and extreme complexity which endangers their experience.

In this article, I want to talk about the idea of reductionism — a process that improves the efficacy of our designs as well as the time we spend making and maintaining them.

Going "back to basics" and challenging the way we design, write code and produce content will de-clutter our interfaces, improve the readability of our web copy, speed up deployment, make things easier to use, and reduce our maintenance requirements.

Reductionism in Web Design

It's important to define what reductionism is in the context of web design. While ideas towards reductionism vary depending on who you ask, a simple definition is that reductionist methods boil down complex things to simpler things, which might include modularizing the system into more digestible components; all of this while avoiding losses in value (fidelity) and usefulness.

Essentially, it means that if you have something that's bloated, heavy or complex — removing some bulk will improve your work.

Reductionism doesn't mean minimalism – but they can work hand in hand.

Understanding the complexity of things by reducing them into smaller components allow us as "web scientists" to better maintain and organize what we produce.

While reductionism allows us to objectively strip away the complexity and see the fundamental principles that guide our work, it specifically highlights the importance of knowing what is beneficial to the end user and to us as the makers of these products.

As a practice, we can save ourselves time and money (by not undertaking unnecessary work) and free our visitors from distractions.

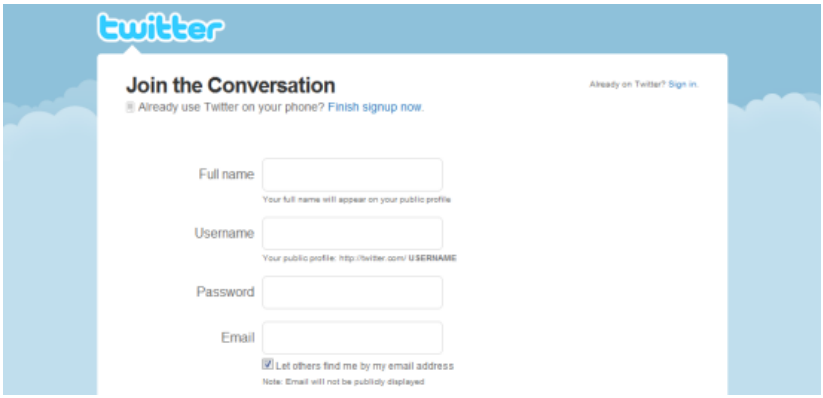
Benefits for the User Experience

Sometimes, we as web professionals spend a great deal of time trying to plan and "pre-react" to situations that will unlikely happen.

We want to give all users every single function that they want/may want. This mentality — though well-intentioned — usually backfires and we end up with something over-engineered and scaled to epic proportions.

But if we just provide our users the things they really need (and nothing more), it reduces the amount of thinking and cognitive processing we subject them to.

If we think about the concept of reductionism in this way, having more features, spending more development time in things practically no one will use, and pre-planning every single potential situation actually makes it worse for our users.



Reducing the amount of content a visitor needs to give to sign up for a web service is an example of reductionism.

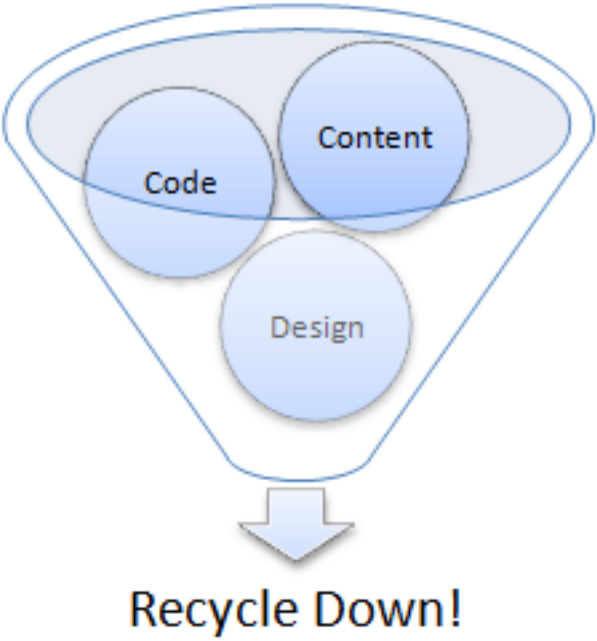
The key point to take away is that reductionism is more than just some quick technique to boost your work's quality. It's a way of life and a beautiful ideology for maintaining a tight workflow.

Principles of the Methodology

In web design, there are three main places we can apply reductionism:

- Website content
- Code
- Design

Additionally, you could apply reductionism in the way your web design business works and how you approach developing solutions to a client's problems.



The three reductionist methods relative to building and creating websites.

Whether you're trying to apply a reductionist methodology to your content, code or design, the principle remains the same: You want to ensure that everything in your product is absolutely critical to the people who'll be using it.

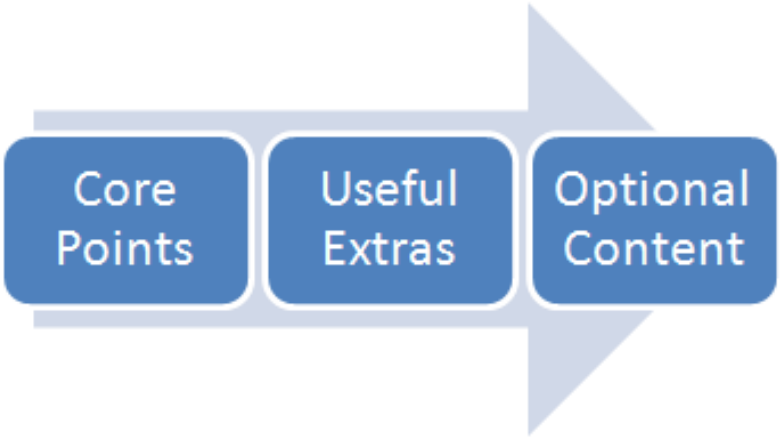
End users hate being confused or feeling like they've lost control; giving them the power of having more time to read your content, achieving a task with your user interface quicker and so forth improves their experience.

The defining characteristic of our work then becomes quality rather than quantity. It's not going to be about having 100 features, it's about having two really great ones.

Content Reductionism

Content reductionism can be approached in many ways. The simplest way is taking your copy, reading through it, and paying attention to ways you can simplify the structure, reduce the word count, remove redundancy, strike out jargon, and just anything that doesn't really add any value to it.

Of course, there's more to content than text. Image or visual sensory reductionism can be performed by taking out gratuitous graphics that simply serve as eye-candy and page bloat, but doesn't really help drive your points across to the reader. Remember that a picture should be worth a thousand words. Using an image should reduce the amount of stuff you have to write, otherwise, it doesn't belong in your copy.



Breaking content into what's necessary and what can be omitted increases readability.

Code Reductionism

Code reductionism is all about simplifying your code and making sure it can't be written any better.

However, this process isn't that simple, and we also have to make micro-decisions in certain cases and go in favor of semantic and web-accessible, yet lengthier code.

Code reductionism can also take place in the amount of web technologies we use. If you can produce a satisfactory effect using CSS (such as this CSS-only hover effect), there's no point in over-engineering the effect by making it dependent on JavaScript or Flash.

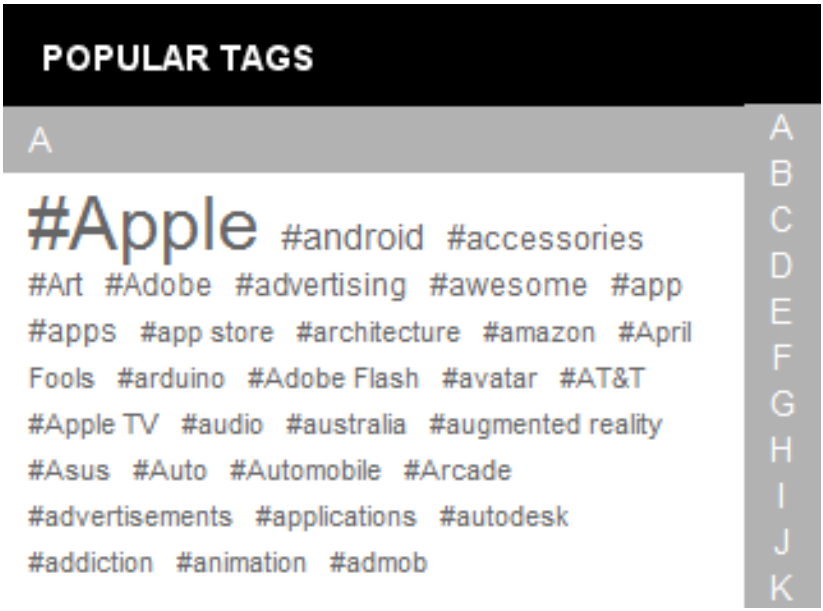
Semantic and minimal code will increase the speed at which your website loads.

Design Reductionism

Design reductionism can be established through the ideals of simplicity, usability and stating the obvious.

Steve Krug's book, Don't Make Me Think!, has a title that summarizes the concept well. We ought to pay attention to the intuitiveness in our designs and by reducing design elements, our designs require less thinking and processing on our user's behalf.

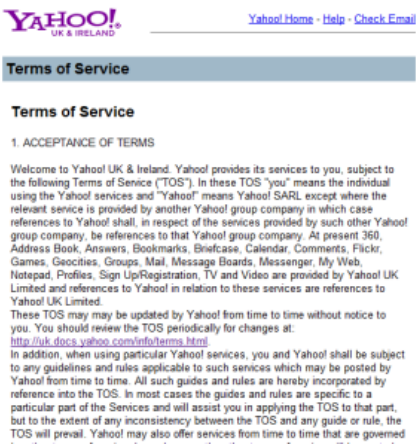
If you have doubts about a particular design element — it probably means you can take it out.



Increasing the simplicity through functional design will boost end-user satisfaction.

Reductionism Tips

The key point to maintain is that when you apply reductionism to your work, the final product should be better or equal to your current state. It's worth noting that reductionism doesn't preach arbitrarily taking things out just to reduce bloat, but rather, encouraging careful and thorough thinking to see if we can make things better by way of simplification. Sometimes things need to be complex or complex things are already as simple as they'll get.



YAHOO! UK & IRELAND

Yahoo! Home - Help - Check Email

Terms of Service

Terms of Service

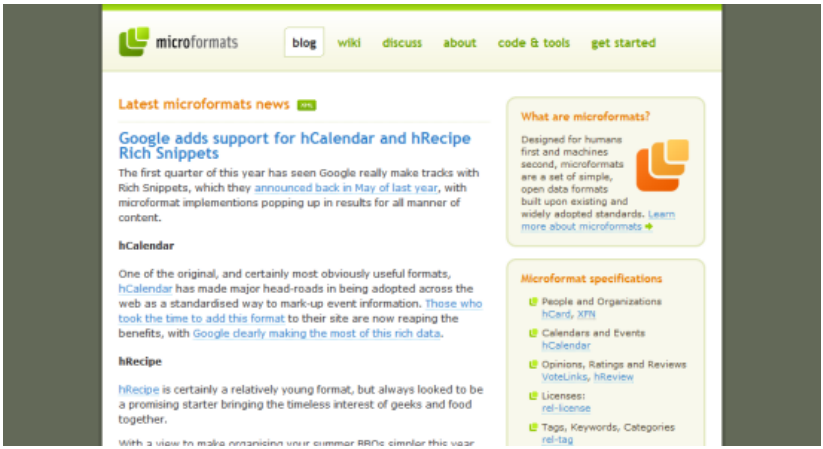
1. ACCEPTANCE OF TERMS

Welcome to Yahoo! UK & Ireland. Yahoo! provides its services to you, subject to the following Terms of Service ("TOS"). In these TOS "you" means the individual using the Yahoo! services and "Yahoo!" means Yahoo! SARL except where the relevant service is provided by another Yahoo! group company in which case references to Yahoo! shall, in respect of the services provided by such other Yahoo! group company, be references to that Yahoo! group company. At present 360, Address Book, Answers, Bookmarks, Briefcase, Calendar, Comments, Flickr, Games, Geocities, Groups, Mail, Message Boards, Messenger, My Web, Notepad, Profiles, Sign Up/Registration, TV and Video are provided by Yahoo! UK Limited and references to Yahoo! in relation to these services are references to Yahoo! UK Limited.
These TOS may may be updated by Yahoo! from time to time without notice to you. You should review the TOS periodically for changes at:
http://uk.docs.yahoo.com/info/terms.html
In addition, when using particular Yahoo! services, you and Yahoo! shall be subject to any guidelines and rules applicable to such services which may be posted by Yahoo! from time to time. All such guides and rules are hereby incorporated by reference into the TOS. In most cases the guides and rules are specific to a particular part of the Services and will assist you in applying the TOS to that part, but to the extent of any inconsistency between the TOS and any guide or rule, the TOS will prevail. Yahoo! may also offer services from time to time that are governed

Remove that waffle! End users don't want lengthy complex documents to read.

To Achieve Content Reductionism

- Focus on the quality of what you produce, not how long it is
- If you can say it in fewer words while still getting your point across, go for it
- Reduce at the end so that you can see how taking something away will affect the entire picture
- Avoid technical language and jargon, it makes content convoluted and exclusionary
- Know who you're writing for and learn what they need/want to know
- Use visuals to reduce the amount of text you have to write and to improve comprehension
- Use headings to modularize your content into logical groupings
- Make things easier to read by using bulleted points instead of paragraphs



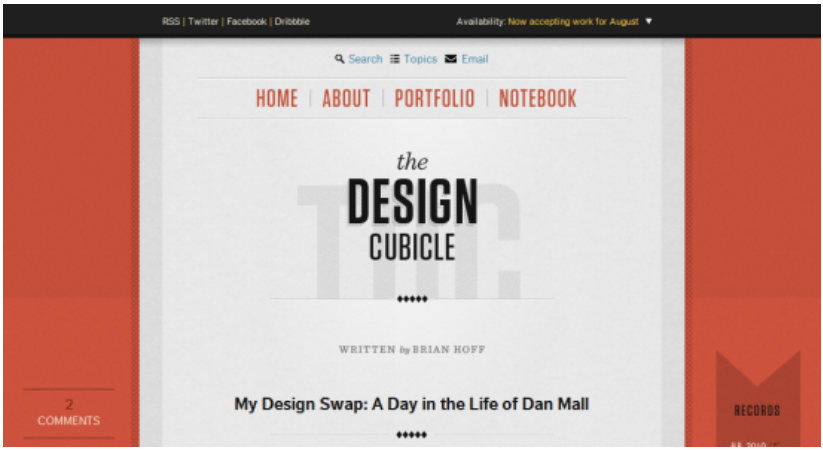
Using the right element for the right job makes website maintenance a lot easier.

To Achieve Code Reductionism

- Have a solid plan and idea of what you're going to develop
- Examine your code frequently and be vigilant against redundancy when you spot it
- Decide which technology will do the job you require with the least amount of code
- Look at your specifications and think of ways it can be done better with less code
- Visit your code regularly and eliminate the zombies
- Experiment with your code and see if you can simplify the structure
- Minify your code to reduce file size
- Try to write code natively before using an abstracted layer (like MooTools or jQuery)

To Achieve Design Reductionism

- Reduce the number of clicks and mouse movement required to find content
- Whitespace gives breathing room for the eyes and for the content of a website — avoid cramping things together
- Simplicity is beautiful: reduce how much information is thrown onto the screen
- Don't use unnecessary flourishes and widgets
- Split test and see if people are accessing things optimally
- Reuse design elements to avoid redundant objects
- Ask yourself what the value of a design element is and if it deserves to be included in your canvas



Keep it simple, stupid (like in Occam's razor)! Complex solutions are much harder to use.

To the Power of 50%

One reductionist method I follow is the "power of 50%" concept. In essence, it's about taking whatever you have right now and then breaking it down until you eliminate 50% it.

Whether you're reducing your web copy in half, cutting down your code base to 50%, or taking out half of the design elements you've plugged in — the theory is you reduce the dilation of your product and enhance the quality of what's left.

While this may seem difficult, the guideline holds true in that, in many cases, the amount of excess that exists in all manners of things is far too disproportionate.

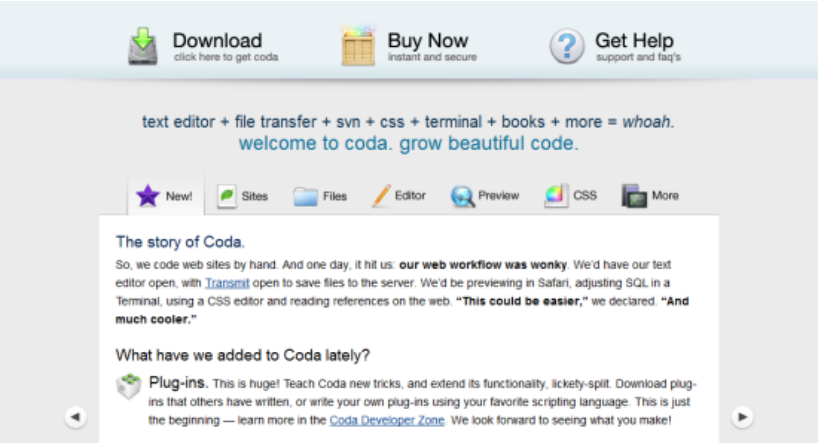
Removing 50% of a website's excess can have profound effects on its usability.

Keep in mind, though, that going over the top with reductionism is possible. If you keep squeezing the juice out of what you create, you may not have enough to drink. Therefore, it makes sense to be thoughtful when applying reductionism.

Final Thoughts

Determining the best way to apply reductionism to your work will differ on a project-to-project basis.

It takes time and effort to get into this mindset of being proactive in creating less stuff, but the ideology it pertains to is grounded in a simple truth: people hate complexity and unnecessary stress. There's nothing worse than being confused or feeling like you've lost control.



Simplified information architecture is just one way to succeed in reductionism.

Reductionism benefits you in the long run and applying its principles to web design is simple. While we accept that over-thinking solutions occurs regularly and we can get a bit sloppy as a result — the reductionist method stands as a way of improving the overall quality of work, which results in us gaining a greater appreciation for refining our thoughts and our products.

In your next project, think of ways to drain away some of the excess.

Sources:

- http://productiveblog.tumblr.com/
- https://twitter.com/signup
- https://getfirebug.com/
- https://www.sensible.com/dmmt.html
- https://policies.yahoo.com/ie/en/yahoo/terms/utos/
- http://microformats.org/
- https://panic.com/coda/

The Art of Distinction in Web Design

One of the hardest tasks we undertake in the user experience field is trying to gain and hold a visitor's attention in the right way. Distinctive design and the ability to focus eyes where they are needed in our web designs is a tricky task, but is something that we should have a firm grasp of.

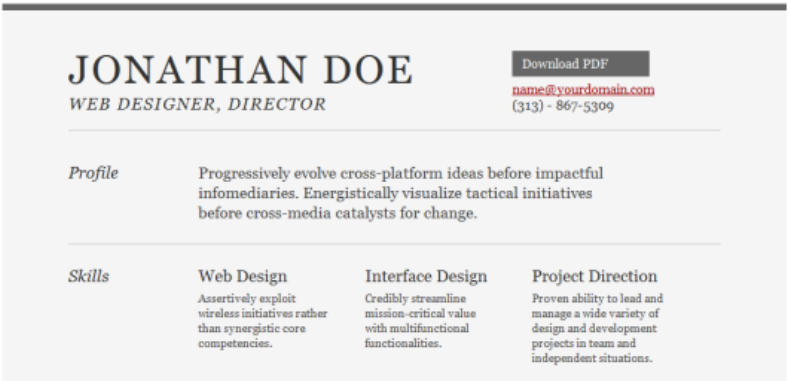
Understanding the artistic traits of influence and distinction allow us to balance important details over our regular content and thus gives us the opportunity to have a great impact and influence on our consumers.

This article aims to highlight various factors you should account for when using distinction in your designs.

Distinctive Design

Distractions in a design lead to a breakdown in communication and can confuse users, paralyzing their ability to quickly determine what to focus on or where to go next.

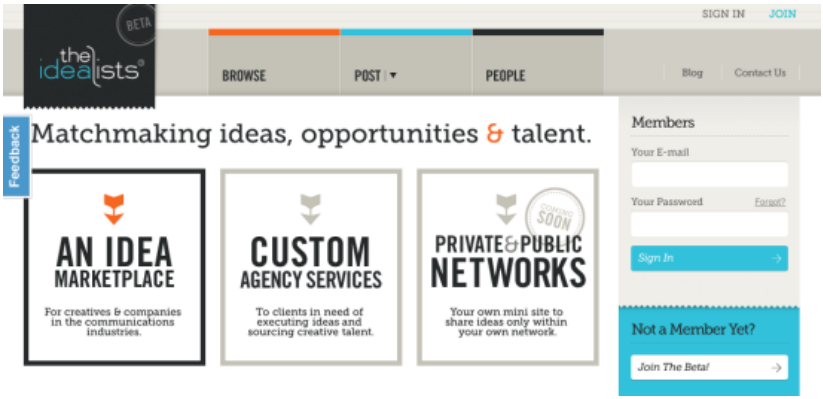
Distinctive design alleviates this by putting forward a few fundamental principles which appeal to the user's needs. Effectively at their core, it underlines the ideal that highlighting content based on importance rather than its position is beneficial and worthwhile.



What is distinctive design? It's a simple goal to make important information visible.

Principles of distinctive design include:

- Not giving prominence to objects unless it has a real need to attract attention
- Limiting the importance you give to all content within the page to avoid diluting the strength of the important content
- Deemphasizing less important content
- Taking the time to help guide the user's eyes through the page to ensure content is read in the right order
- Avoiding too much information on the page to reduce the noise
- Ensuring that what you display fits the ideas you wish to convey

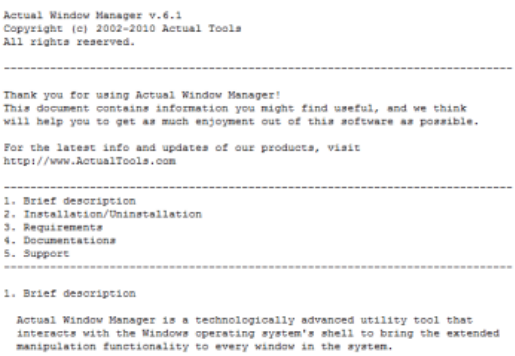


Distinctive design is important to ensure the end-user can find what they require.

While too much focus being demanded on a page (with violent levels of distinction) can be damaging to the user experience, bland data with no focus or selective highlighting can be equally disruptive.

It can make unassuming content (even well written material) appear dry, dull or even hard to read.

While the majority of your content should be neutrally styled (it should have no emphasize), there will always be content in a site that needs reinforcing. And whether you use images, stylistic effects, color, appearance, animation or something else, appropriate distinction is extremely important.



```
Actual Window Manager v.6.1
Copyright (c) 2002-2010 Actual Tools
All rights reserved.

--------------------------------------------------------------------------------

Thank you for using Actual Window Manager!
This document contains information you might find useful, and we think
will help you to get as much enjoyment out of this software as possible.

For the latest info and updates of our products, visit
http://www.ActualTools.com

--------------------------------------------------------------------------------
1. Brief description
2. Installation/Uninstallation
3. Requirements
4. Documentations
5. Support
--------------------------------------------------------------------------------

1. Brief description

  Actual Window Manager is a technologically advanced utility tool that
  interacts with the Windows operating system's shell to bring the extended
  manipulation functionality to every window in the system.
```

Plain text files have no distinctive flourishes and is often hard to read.

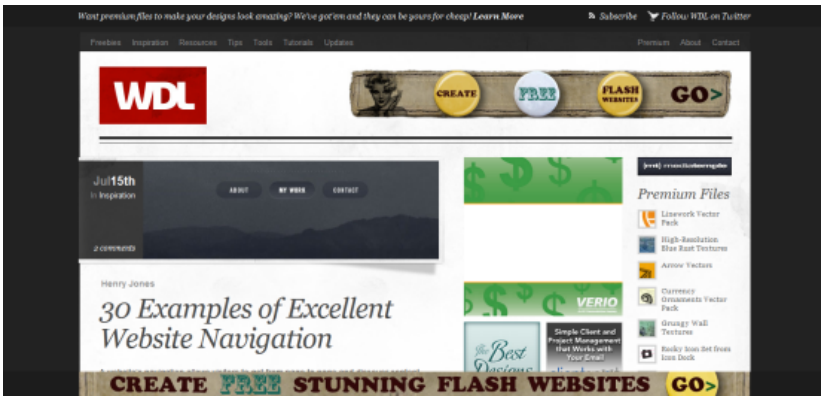
Noise Margins

One key principle in maintaining a distinctive design is the function of noise margins. Have you ever visited a website that has far too much going on? That's the kind of problem that can make it next to impossible for users to find what they need.

By simply reducing the amount of information that appears within the website design, or by breaking it down further into several pages (or sections using headings), you can increase the way people interact with your work because relevant information will be more contextually visible and easier to locate.

Noise can distort the overall message of a website in unnaturally large quantities.

Within a user interface, noise can occur in many formats. Anything intended to grab the user's attention in the design can be a key component of drawing focus away from the content. Ideally, you should have as little of these devices or interface quirks as possible.



Interface noise can result from too many elements appearing within a website.

Even sites with little in the way of noise-inducing components can have noise pollution problems. Text content can suffer issues if it's not laid out effectively. In this case, breaking large clumps of data into tables, bulleted lists, sections and styled content can aid readability.

Images with too much on display can also become like an optical illusion (like a "Where's Waldo?" book) as the search for any relevant points may be lost, which brings attention to the task of trying to simplify diagrams, graphs and infographics to maintain a level of neutrality and visibility.

[Note: At 93 K in size, this "short" help file is anything but short, and may take over a minute to load if you have a slow modem. Don't access anything in the table of contents until the whole thing is loaded. --Jolo]

A Short IRC Primer

Written by:

Nicolas Pioch, (Nap on IRC)
< Nicolas.Pioch@grasp.insa-lyon.fr >

Text conversion by:

Owe Rasmussen, (Sorg on IRC)
< d1rasmus@dtek.chalmers.se >

HTML conversion & update by:

Michelle A. Hoyle, (Eingang on IRC)
< hoyle@ifi.unizh.ch >

Joseph Lo, (Jolo on IRC)
email form

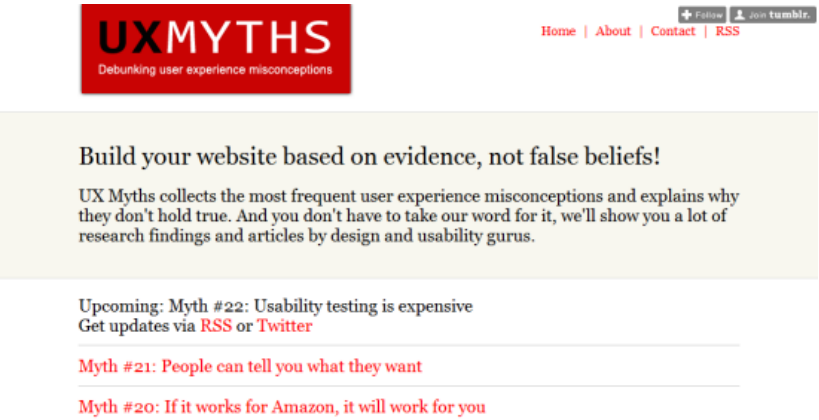
Edition 1.2, January 1, 1997

Content isn't free of noise either! Huge blocks of text can frustrate end-users.

Spatial Awareness

Another principle tied closely to the aspect of noise is spatial awareness. While it's important to know how much content will be displayed in a page, the knowledge of how we represent content can help end-users identify their own surroundings.

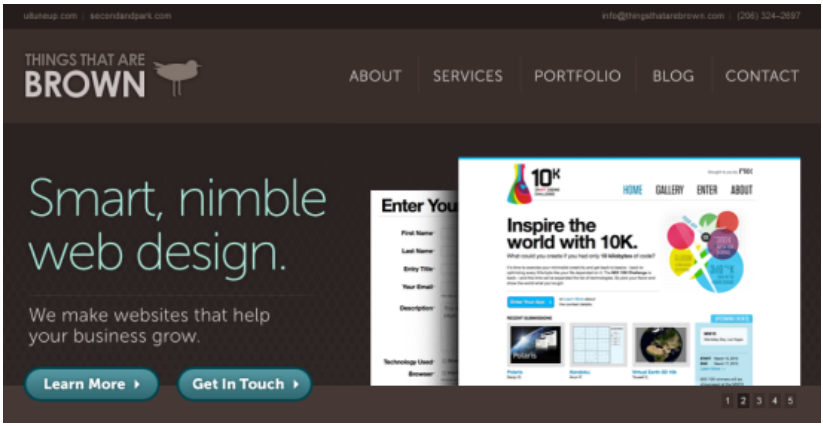
Whilst designing with an emphasis on simplicity — introducing much needed whitespace within even the most lengthy of documents can allow the user to focus within those sections that hold meaningful content. In addition, complex background images or patterns can reduce the effectiveness of added whitespace.



Giving a website plenty of breathing room increases the awareness of space.

Organization is the key to any goal, and conventions have evolved within web design for just such a reason. The complexity of any object can be determined by both how easy it is to not only know what function or purpose it has, but also the way it is presented can help that old friend of ours, readability.

Whitespace has a key function in not only breaking sections apart (through space) but also in increasing awareness of how those sections are maintained, thus making the design a more pleasant experience which can help elements meet the impact needs of their environment.



A well structured and organized layout will be easy to navigate around.

While the idea of flexible layouts showcase that the amount of visible space may differ drastically for the end-user based on the device they have, small screens have highlighted the way in which a web browser and it's viewport with scrollbars dictates the cascade and priority information gets.

It's a well-established and notable fact that information at the top of a website holds more priority than what is at the base of the page (due to the order in which it's likely to be seen). Therefore ensuring your content is provided in a balanced manner which follows the convention is worth considering.

Conventions show that content should be given priority based on importance.

Drawing Focus

One of the key principles of distinctive and attentive design is based on the idea of drawing focus to content and design elements of importance. While making good use of space and reducing noise levels can be beneficial to drawing focus naturally, there are ways to grab the attention of visitors by focusing or drawing the eyes to something onscreen.

While using such elements can be beneficial to the end-user, it's worth highlighting that if you fail to balance the need for attention with the need for neutrality, you might end up counteracting your efforts to showcase importance.

The dConstruct website draws focus towards the famous speakers taking part.

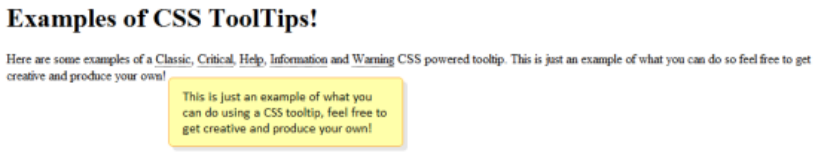
As with many aspects of design, the idea of drawing attention can be achieved through a general understanding of psychology (so getting that book on the subject may pay dividends).

There are a great number of different ways people can be drawn to content. Methods to draw focus are usually based around clever wording (humor is effective), figures (like prices or dates), relevant terms which are easily recognized (like the word contact), pictures denoting what's being mentioned (infographics), and animation (which naturally focuses to understand an effect).

By using words that the reader can relate to, you're more likely to draw interest.

While natural attention can be drawn through psychology and relating data in ways which attract the end-user, visual attention is usually more obvious and easy to snatch focus.

An example of this can be seen through people use Flash animations that naturally blur the relationship between grabbing attention for the content and attracting the attention to the user interface (rather than its content value). While forcing the attention of something through visual effects can be a quick (if not crude) way to ensure a visitor reads something, it should be done with some restraint and care.

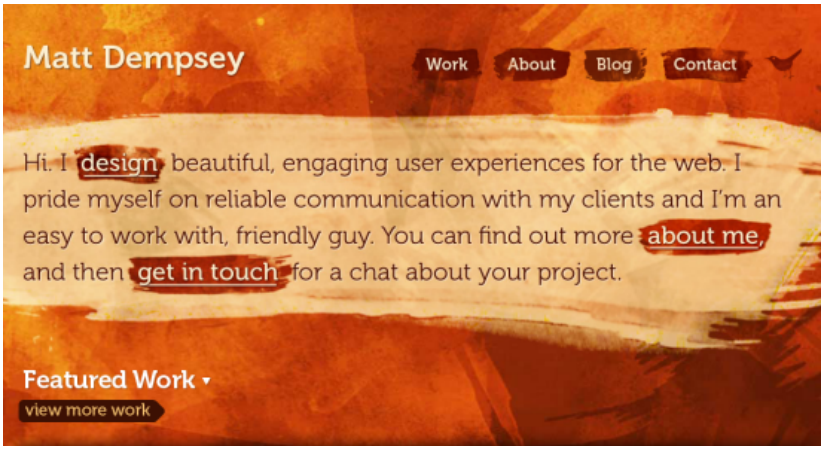


Visual attention-seeking can result from animated effects that result from actions like hovering.

Contrasting Mediums

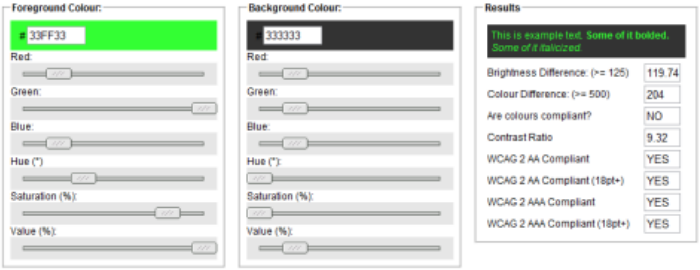
Another key principle of distinctive design is the idea of contrast. Color has become one of the vital components of many website designs and the way in which shades and variants interact with each other can determine not only the visibility of the information, but in worse case scenarios, they could even be an indicator of underlying problems of readability.

One of the simplest ways to measure contrast is to examine how visible the foreground object (like text) is from background content such as a block color or image.



The most common contrast you'll encounter is background and foreground colors.

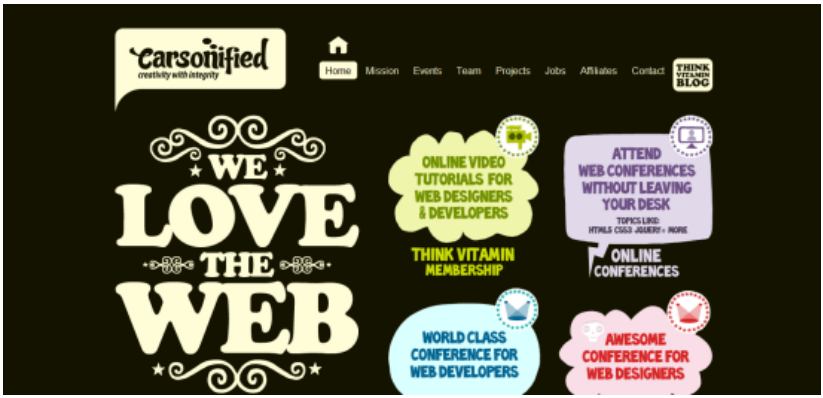
As mentioned, web accessibility plays a common role in contrast as the visibility of information in general can have consequences to who can access the information. While low contrasting combinations may give the impression of subtlety and softness, if contrast is insufficient, or if two shades which are known to conflict in certain vision conditions like color blindness are used, you may well end up with a negative visibility issue that not only makes text less distinctive but impossible to read. Visibility disorders are the most common conditions that suffer from poor color contrast choices.



Accessibility is closely tied in with how well content contrasts in its surroundings.

The benefits of contrast and the use of color psychology can draw user attention in some unusual ways. A primary example of this is what is known as color symbolism, where people visit a website rich with certain shades of colors that remind them of certain things and emotions. This richness which goes unstated can draw their focus.

Depending on the audience type and the culture of the crowd, the interpretations of color may differ, but contrasting and complementing the harmony of your palette will give added emphasis and targeted emotional triggers to make use of.



Color not only evokes emotion (and attention) but it also can make things distinct.

Highlight for Impact

The final principle I want to highlight (no pun intended) is the idea of manually adding distinctive impact to your website through selective effects.

So far, we have focused on taking away objects of unnecessary value and pulling eyes to a section through emphasis and contrast. This method takes a different approach entirely by taking something that already has a user's attention (like a block of text) and then boosts certain parts of that content to increase its visibility around elements which surround it. Of course, this is aimed at natural highlighting in context to an existing range of content.

Drawar

A web design and development community for people with a thirst for knowledge. Follow the Community on Twitter right here.

Just highlighting passages of text can give the content a boost of emphasis.

Highlighting a segment of text may seem subtle, but it can be surprisingly effective. In browsers like Google Chrome, you can actually change the color of highlighted text which may give the manual selection of content a better contrasting (and easier to read) platform to work from.

Of course, apart from using a background color there's a whole range of ways in CSS to draw attention, such as the use of icons besides text, coloring links of unique segments or simply changing the font, size, color or style (bold, italics, underlines, strength and emphasis) of the information needing added visibility.

Common stylistic traits that are anything but the default can represent content well.

The use of color in highlighting for impact and giving distinction go through psychology and design with ease and it can be said that only using color selectively will give your work added impact (like how more whitespace gives impact to less content).

However in certain cases (like children's sites), using a sharp array of bold colors and vibrant shades can actually draw attention well enough for the subtle restriction of color to be a non-issue. While in many sites aimed at adults, it's better to keep a firm focus on color usage, sites aimed at a younger audience can get away with such vivid use of shading.

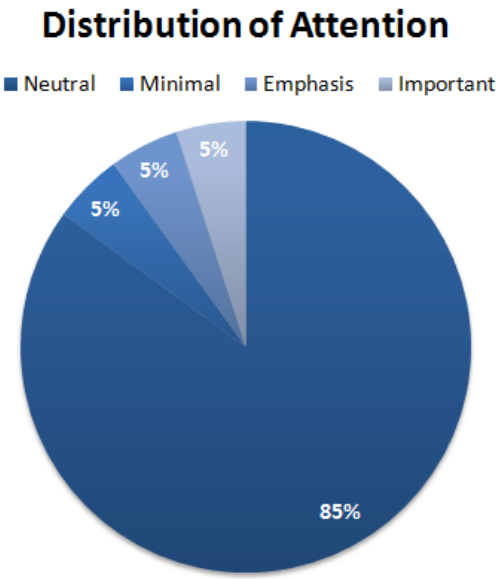
Using an appropriate sharp group of colors can highlight sections uniquely.

Attention to Detail

While there aren't any strict rules to dictate how much strength you should give content, I find that a good balance is as follows:

- 85% of the design is neutral (with no focus)
- 5% of the design has minimal highlights (like banners)
- 5% of the design is emphasized (such as bold, italics, link colors, and other strength)
- 5% of the design being very important content which requires immediate priority

Though, factors which affect how the distribution balances out should be accounted for, such as its position within the viewport, if the content is "fixed" and the amount of strength given (like size and color).



The above graph may help you decide how to breakdown the distinction barrier.

JavaScript and advanced CSS selectors have improved the situation of designing with consideration towards distinction. With the ability to simply hide content (or restrict how much content is visible) at any one time, and the ability to animate or control how information will appear depending on viewport availability or device type, we can put ever increasing efforts into making the most out of the little flourishes we choose to have on display. While such complexity can make us feel like we need to take advantage of it all at once, a controlled approach can simplify our designs.

Scripting and cool CSS techniques aside, single page designs have boomed in popularity.

The key principle of attention is that of the designers own eye for detail, there is little excuse to make things either bland or boring even when the content arrives in beauty and simplicity. The modern advances of scalable designs, flexible layouts and minimalistic ethos have laid down some great methodologies for making information easier to access, read and use.

Taking time to consider how each event or effect may be interpreted, your visitors' ability to read what you produce and assigning yourself the goal of attempting "less is more" (in context) will boost your content's value.

Everything in moderation is the best policy as there's no point wasting effort!

The art of distinction is in the way we portray information and how the important and relevant bits manage to push their way past the flourishes and useful engaging elements which give that content added substance.

While the web evolves (like our tastes and abilities) it's a fact of life that portraying information effectively will become (as it remains) one of the most important philosophies that the design community has to offer.

In reflection, distinctive design gives your content the best chance of being read, understood and enjoyed in the future — which is fantastic if you want to be noticed!

Sources:

- http://sampleresumetemplate.net/srt-resume.html
- http://www.angelfire.com/super/badwebs/
- http://uxmyths.com/
- http://thingsthatarebrown.com/
- http://2010.dconstruct.org/
- https://shop.theoatmeal.com/
- http://www.mattdempsey.com/

- https://snook.ca/technical/colour_contrast/colour.html#fg=33FF33,bg=333333
- https://www.creativehunt.com/shanghai
- http://builtbybuffalo.com/

A Comprehensive Guide Inside Your <head>

As web designers and developers, we pay so much attention to what's directly on the screen (or in our code) that the <head> of a document and what's inside is often considered as an afterthought.

While in many cases it's true that what appears on the screen is the most important part of a website (the content is what people visit a site for), the "thinking code" inside the <head> of our documents plays an important role.

This article will examine exactly what can fit inside a website's head.

Mastering the Mind

The head of an HTML document is a busy area, and while it may not have the range of elements that the <body> can flex, it can actually engineer a range of its own elements to play vital roles in how a site will operate or how it can interoperate with other sites.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="en-US
<head profile="http://gmpg.org/xfn/11">
<script src="http://widgets.digg.com/buttons.js" type="text/java
<meta http-equiv="Content-Type" content="text/html; charset=UTF-
<title>Six Revisions - Web Development and Design Information</t
<meta name="tweetmeme-title" content="" />
<link rel="stylesheet" href="http://sixrevisions.com/wp-content/
<link rel="alternate" type="application/rss+xml" title="Six Revi
<link rel="pingback" href="http://sixrevisions.com/xmlrpc.php" /
<link rel="icon" href="http://sixrevisions.com/favicon.ico" type
<link rel="EditURI" type="application/rsd+xml" title="RSD" href=
<link rel="wlwmanifest" type="application/wlwmanifest+xml" href=
<link rel='index' title='Six Revisions' href='http://sixrevision
<meta name="generator" content="WordPress 2.8.4" />
```

Depending on the website, there might be plenty going on inside its head.

So what are your options and how can they benefit your website? Well there's quite a lot actually!

There are ways to add useful metadata into your documents (for search engines and other web robots to find), icons that you can supply web browsers for extra visuals (like favicons or device-specific icons for the iPad/iPhone), ways to allow the syndication of your content, and even stylistic and behavioral references that include external stylesheets and scripts.

In essence, the <head> of our HTML documents give the markup below it extra meaning.

Thinking code has many purposes, and not all of it is regulated by an official W3C specification!

Head Elements Gone Rogue

One thing that is unique to the head of our HTML is that it's adoption and usefulness is determined on convention and popularity. Metadata, for example, relies on search engines and social networks to acknowledge and use the data, and link references require browsers to take advantage of them. Some things that can be included inside the head of our documents aren't even officially supported by W3C specifications — such as <meta name="googlebot"> that is a proprietary meta tag for Google's spiders — but because of the nature of the elements, they are valid markup.

Independent Elements

The first elements we should talk about are those which are independent in the sense that they just serve a single purpose.

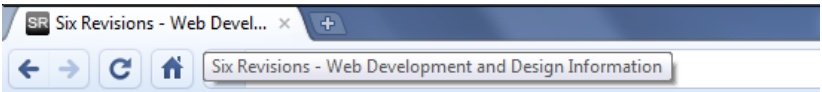
One of these elements is so mission-critical that you are required by HTML standards to include it: the <title> tag.

Below are five of the seven (excluding meta and link, since we will talk about them later) head elements:

<title>	This tag describes your document in the browser's title bar.
<script>	This tag allows you to either embed or link to JavaScript files.
<style>	This tag embeds CSS inside the document.
<base>	This tag states the base directory of files (for relative links).
<object>	This tag can be used in the HTML head; typically it is used inside the <body> to include page assets such as Flash components.

Example:

```
<title>Six Revisions - Web Development and Design Information</
title>
```



A document's title should be unique on every page and should adequately describe its content.

While having a title in your document is required (if you follow W3C standards), the other tags might be of limited use to you.

For example, external stylesheets (instead of <style> tags) and an htaccess file (as opposed to <base>) are better options for declaring CSS style rules and declaring the base directory.

Mighty <meta> Tags

Of all the elements that appear in a document's head, none is as ubiquitous as <meta> elements.

These elements — while being highly desired for search engine optimization– are more of a pseudoscience affair, as the influence they have on search engines aren't publicly disclosed.

With the exception of the <meta="http-equiv" content-type=""> that specifies the document's MIME type, they are all optional.

Getting the page's encoding correct (through content-type) is important.

IE6 can take advantage of a special http-equiv value called imagetoolbar

Below are some examples of http-equiv values:

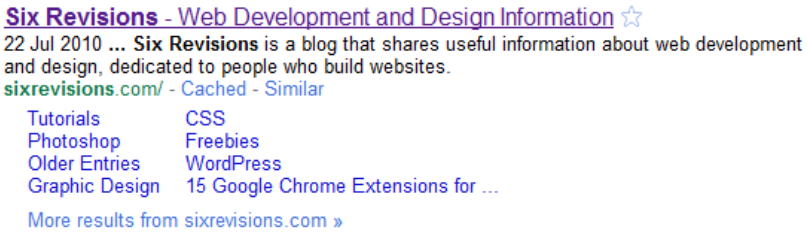
<meta http-equiv="Cache-control">	Gives you greater control over browser caching.
<meta http-equiv="Content-type">	States the MIME type for layout engines.
<meta http-equiv="Content-language">	Dictates the primary spoken/ written language used in the document.
<meta http-equiv="Expires">	States the expiration date of the document.
<meta http-equiv="Imagetoolbar">	Forces IE6 to either disable or enable the image toolbar on hover.
<meta http-equiv="Last-modified">	Allows you to specify when the document was last modified.

<meta http-equiv="MSThemeCompatible">	Disables (or restores) the default theme for form components in Microsoft Internet Explorer 6 and above.
<meta http-equiv="Refresh">	Redirects the page at a specified time (web spiders don't like this one).
<meta http-equiv="X-UA-Compatible">	Microsoft extension to dictate compatibility mode triggering.

Example:

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
```

Of course, http-equiv isn't the only attribute that can provide useful information. And even though keywords are presumed to be ignored by Google and description is only used to represent SERP listings (not acting as a ranking factor), there are no set standards for what values should be used.



Search engines like Google can use a <meta description=""> in its results.

Below are a few more types of <meta> tags:

<meta name="Abstract" >	A quick summary of the main points of the content.
<meta name="Author" >	Who authored the HTML document.
<meta name="Contact" >	An email address, phone number or physical address for the site.
<meta name="Copyright" >	Copyright information.
<meta name="Description" >	Describes the HTML document.
<meta name="Designer" >	The producer of the site.
<meta name="Generator" >	What was used to generate the HTML document.
<meta name="Geo.placename" >	Indicates the town or city of which the site is regionally based.
<meta name="Geo.position" >	Gives longitude and latitude (semicolon separated) position.
<meta name="Geo.region" >	Uses two digit region codes to indicate location (such as GB).
<meta name="Googlebot">	Can tell Google to Noarchive, Nosnippet, Noindex and Nofollow.
<meta name="Keywords" >	This element lists comma separated words linking to content.
<meta name="Language" >	States the primary language used within the sites content.

<meta name="Publisher" >	Lists the name and version of the product that built a site.
<meta name="Robots" >	Replaced by robots.txt, this specifies page by page indexing.
<meta name="Subject" >	Indicates the subject matter to which the content relates.
<meta name="Viewport" >	Used by Apple devices to indicate the content window size.

Code Example:

```
<meta name="description" content="Six Revisions is a blog that shares useful information about web development and design, dedicated to people who build websites." />
```

While it's true that meta tags don't have a strict set of standards, conventions have come into existence to help decide which tags and values should be used.

Search engines (who pick what they feel are worth recognizing), social networks (that use the data to categorize a site) and organizations like the DCMI (trying to reduce our confusion) are responsible for such efforts to help in such situations.

Metadata isn't a science, but the geographical positioning elements can seem like one.

Below is a list of the most widely adopted Dublin core metadata elements:

DC.Contributor	Gives names of individuals who contributed to the resource.
DC.Coverage	The scope covered by content (like place names or co-ordinates).
DC.Creator	An individual responsible for the construction of the content.
DC.Date	The point of time of which the page or content was created.
DC.Description	Explains the resource and can include links to a table of contents.
DC.Format	Provides the file format of the document (for interpretation).
DC.Identifier	Links to a URL, DOI, ISSN or ISBN which references the content.
DC.Language	Provides details of the language or dialect used for content.
DC.Publisher	Identifies the person or group who made the resource available.
DC.Relation	Links to a resource which relates to the published information.
DC.Rights	Gives license data such as a link to your terms of usage policy.
DC.Source	Highlights the source or URL of origin for the pages cited content.

DC.Subject	Lists semicolon separated strings that identify a pages subject.
DC.Title	The name which indicates what the document is known as.
DC.Type	Shows content categories, functions, genres or aggregation levels.

Example:

```
<meta name="DC.Creator" content="Jacob Gube" />
```

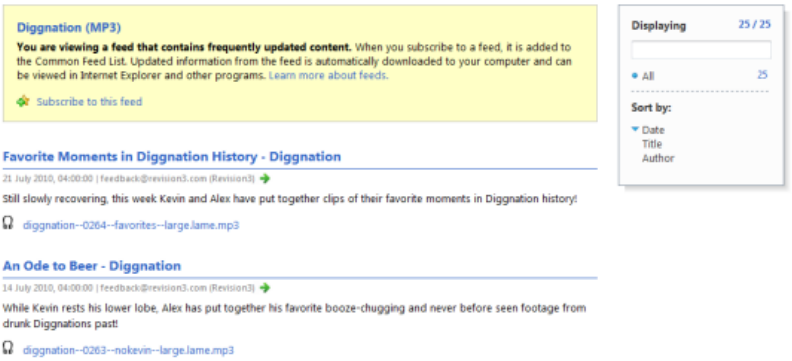
Note: As denoted in an article I wrote called "5 Web Files That Will Improve Your Website," you can actually place all of the relevant DCMI terms and metadata references into a handy, cacheable RDF file. While having meta tags outside the HTML head may sound strange, it's a recognized way of serving repeating information (such as the site's creator or site's spoken language).

The effects meta tags have is of having semantic richness in your HTML documents, making your site all the more interoperable and meaningful to user agents, web robots, and web services.

Luxury <link> Tags

The simple purpose of the link element is (exactly as it sounds) to provide the browser with a location to an external resource that relates to the document in some way.

This element has so many purposes — from providing links to external stylesheets to indicating that RSS feeds are available, licensing of a web page, right across to referencing a favicon for your lovely web design.



Without the link element, Internet Explorer wouldn't immediately know this feed exists.

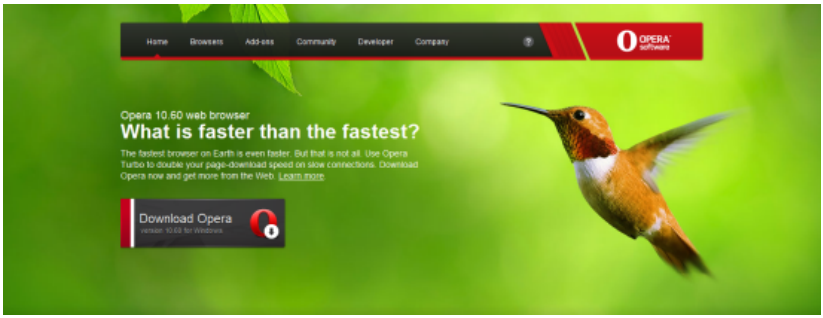
Whilst many link conventions have been established, none are more grounded in the rich, textured history of HTML as the stylesheet. As you are already aware, if you have an external CSS file that contains your site's CSS, you use the <link> element to reference that file within the document's head. This allows the web browser to load the file, process its contents, and then apply your styles to the document's objects.

Below are the two primary ways you refer to a CSS file:

Stylesheet	This identifies a stylesheet to apply to the immediate document.
Alternative Stylesheet	This lists a stylesheet that can be triggered upon a user's request.

Example:

```
<link rel="stylesheet" href="style.css" type="text/css"
media="screen" />
```



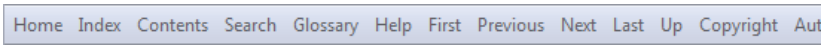
None of the unique style in this design would work without the CSS file it uses.

Since the evolution of CSS3 (and the way Internet Explorer works), a unique level of complexity for stylesheet references have been forged. Not only can you wrap this element within conditional comments, but with media queries, you can apply the code only if certain conditions are met by the user agent.

It's important that while stylesheet references take much of the glory for the link element, we also mention additional ways to make use of this flexible method of enhancing a site.

A classic example of how such elements can be used is defined by the Opera browser's "navigation bar" which actively seeks out link references with certain rel values like index, author and home to provide a site-wide method of linking to specific places which may be of use to the end-user (via the browser).

While it's only found in Opera, or through a plug-in/add-on/extension in other browsers, it can be really helpful in the right hands.



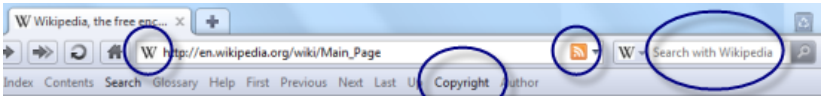
Opera has a navigation menu that links to important document or site sections.

Below are the various navigation links that Opera supports:

Author	Usually links to a page with author details such as "about us".
Contents	Links to a page or sitemap which structures the contents layout.
Copyright	References a sites copyright, privacy or terms of usage policy.
First	Links to the first page or fragment section of a long document.
Glossary	Sends the user to a page of terms or related external resources.
Help	Usually links to pages where support or contact can be gained.
Home	Links to the homepage or the primary document of a website.
Index	Used to link to a page indexing where certain content occurs.
Last	Links to the last page or fragment section of a long document.
Next	Used when content is split over several pages to forward the user.
Prev	Refers to a previous page when content is split over multiple pages.
Search	Identifies a specific page used to perform an advanced search.
Up	Can either link to a pages parent or go to the top of a long page.

Example:

```
<link rel="index" title="Six Revisions" href="http://sixrevisions.com" />
>
```



Web browsers can do so much with link elements. It's extraordinary.

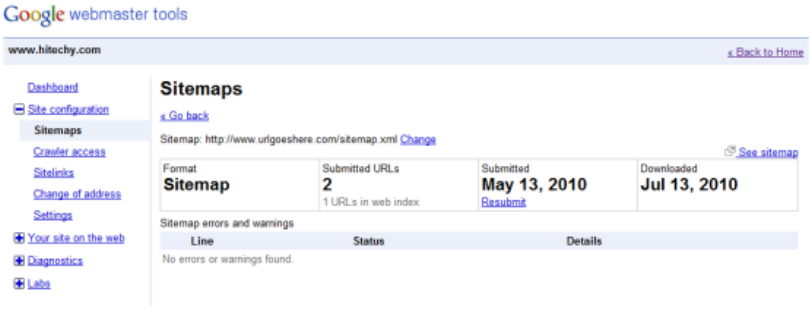
Opera isn't the only browser that can take additional advantage of the link element. In fact, all of the below have been adopted by browsers to reference external files which can impact either the way your website looks or functions.

Alternate	Refers to files that link to the site such as RSS and Atom feeds.
Apple-touch-icon	Identifies an icon image which can be used by Apple devices.
Dublin (Or Meta)	Primarily used to indicate an instance of the DCMI RDF document.
P3P	Links to a policy file which relays information about user privacy.
PICS-label	Assigns a content rating based on a sites family friendliness.
Search	Identifies a web browser compatible OpenSearch document.
Shortcut Icon	Links to a Favicon that is shown within the browser address bar.
Sitemap	Refers to a XML sitemap for search engines to begin indexing.

Example:

```
<link rel="shortcut icon" type="image/vnd.microsoft.icon"
href="favicon.ico" />
```

Examples of what can be included can range from OpenID (a universal authentication system) right through to referencing a document or profile setup by the microformat community (such as XFN) which relay semantic information about your pages.



Just because their effects are not visible doesn't mean they don't exist.

Below you will find a range of non-standard link references:

appendix	Links to a file providing supplementary details for your content.
bookmark	Identifies a key fragment within the document like #important.
chapter	Indicates a specific chapter or volume of a multi-page document.
edit	Used by some CMS software to indicate a location to edit the file.
license	References the documents license (such as Creative Commons).
openid.delegate	The website address used to identify an OpenID users account.

openid.server	The server where verification and validation of the details occur.
pingback	Allows blogs to determine when someone links to a resource.
section	Identifies a section of the document within a given chapter.
subsection	Links to a specific subsection of a section within a given chapter.
Start	Refers to the initial title page within a collection of documents.

Example:

```
<link rel="license" title="Creative Commons Attribution 2.0 UK
Licence" href="http://creativecommons.org/licenses/by/2.0/uk/" /
>
```

As a best practice you should always consider your link options carefully as referencing external files may increase the HTTP requests your web pages make to render a web page, thus slowing down page response times. Use only what you need and know why you need them.

Wrapping Up

While the contents of a document's head could technically be limitless, it's worth knowing what parts will give you a genuine benefit and which parts will simply be extra bloat you don't need.

As a general rule, the best principle you can follow is to try and keep the head clean and well structured, but to also strike a balance between keeping things simple and doing what is best for your visitors.

While many people leave their heads as empty as possible, a better plan is to weigh up your options and use whatever enhances the user's experience.

Behind the scenes in the head of a web page, there are a lot of cool things going on. Don't overlook it.

Sources:

- https://technet.microsoft.com/en-us/windows/cc288325(v=vs.60)
- https://en.wikipedia.org/wiki/MIME
- https://en.wikipedia.org/wiki/Web_browser_engine
- https://en.wikipedia.org/wiki/Search_engine_results_page
- http://www.geo-tag.de/generator/en.html
- https://www.opera.com/
- https://www.w3.org/TR/css3-mediaqueries/
- https://creativecommons.org/share-your-work/

Mobile Web Design: Best Practices

The explosion in user adoption of mobile devices has revolutionized the web. Though designing for the Mobile Web follow similar principles to designing websites, we must consider some notable differences.

For one, current mobile device networks don't run in the same speed as broadband devices.

In addition, there are also a myriad of ways our mobile web designs are displayed in, from touch screens to netbooks, which make even the smallest desktop monitors look like giants.

Some might argue that going mobile isn't necessary yet, however, what no one will disagree with is that it's an inevitable turn in the profession of people who make and run websites.

If you're considering developing mobile web designs (or pushing an existing one onto the Mobile Web), this article should help you get to grips with the growing trend of mobile design.

Delivering the Design

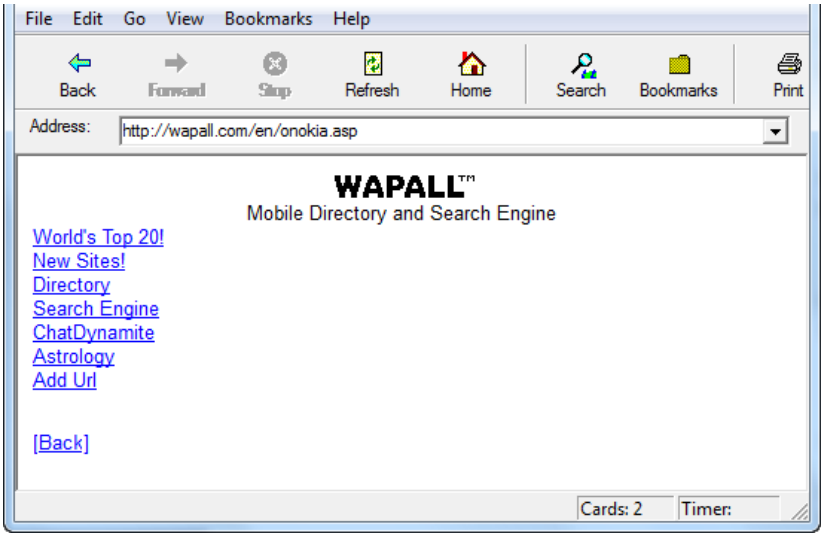
One of the early elements that need to be considered for producing a mobile-device-friendly site is the way the experience will be delivered.

Complications in Delivery Method

The ideal scenario would be that each device simply scales and adapts to your existing website — and some devices, such as the iPhone, are able to because of their built-in web browser. But because of so many devices out there, a cross-device mobile design is difficult to make.

If you thought that developing sites that work on most web browsers such as IE, Firefox, Chrome, and Safari was tough — try developing one for iPhones, BlackBerrys, Palm Pre's, Androids, Motorola devices, Nokia devices, and — the list appears bottomless!

For desktop-based web designs, you only had one markup language to deal with: HTML. But on the Mobile Web, there is also WML and then platforms such as iOS for Apple devices and Android for Android devices.



WML used to limit our design creativity, but we have more flexibility these days.

Adapting a Web Design to Support Mobile Devices

One option to pushing a site to the Mobile Web is to simply create or modify your existing code and design to work well on mobile devices, or building from scratch with mobile devices in mind.

With a bit of CSS3 (using media queries), for example, you can rescale the dimensions of your layout depending on the user's device.

The problem is — you guessed it — not all devices support CSS3. So you may have to resort to using server-side device detection (e.g. HTTP headers to sniff out the user agent) or using JavaScript (e.g. modifying the DOM to rescale your layouts). But again, some devices might not support these techniques.

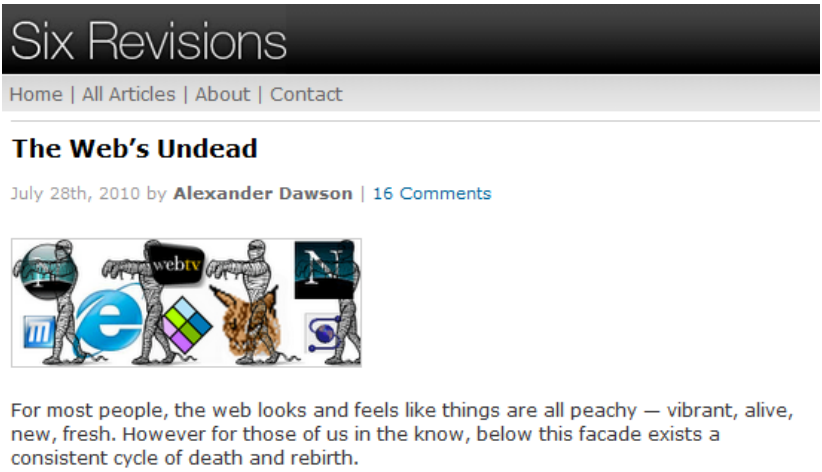
Making an existing layout scale depending on the viewport is as simple as a few lines of CSS3.

Redirect Mobile Users to a Mobile Version of the Site

Another method for delivering a mobile design is to build an especially optimized layout for handheld devices. You can build this yourself or use a web service such as Mobify.

Compared to the first method, this is the better format for delivery as you can create an experience specifically for your mobile users without taking away from the experience of desktop users.

For this to work, you will have to route traffic on your site depending on the user's browser agent. For example, if a mobile device user visits your site (yousitename.com), then they will automatically be redirected to, say, mobile.yoursitename.com or m.yoursitesname.com.



A separate mobile site can mean squeezing out the extra bytes for faster rendering.

Tips on Redirection

Whichever route you decide to go down, it's important that:

- Visitors know that a mobile-friendly version of your site is available
- Visitors can have the choice between a mobile version or the normal version

While forcefully redirecting or changing the layout for your end-users may seem like a good idea, it can lead to frustrations, so there should be ways in which a mobile device user can view the normal site design, and vice versa.

A simple solution would be to provide a link that goes to either version of a website. For example, on Six Revisions, you can find a link to the mobile version (m.sixrevisions.com) in the footer of the regular website, and conversely, a link to the regular website is provided at the footer on the mobile version. Whether you're a mobile device user or a desktop user, you have access to both sites.

Structure and Code

The next thing that we need to consider is the structural code (markup and styles) that goes on behind the scenes.

- Do you go with a mobile-friendly language like WML or the XHTML mobile profiles?
- Do you build an app for iPhones, and then one for the Android?
- How does the cost and speed associated with mobile device web browsing affect the way you should develop your design?
- What about modern standards like HTML5 and CSS3?

These are just some of the questions that we all have in this relatively uncharted and undeveloped territory.

New devices may not support the same code as older mobile handsets.

Choices

Choosing the right language for a mobile-friendly website is paramount; while older devices before the smartphone revolution only support WML (which is pretty basic) the W3C produced a mobile-friendly version of XHTML (referred to as the XHTML Mobile Profiles).

Luckily, due to the speed in which mobile device manufacturers have taken to giving a complete and robust web experience, you can often simply use regular HTML or XHTML — if you don't want to be held back by mobile profiles or WML.

However, it's important to underscore the fact that it's still worth considering WML if you feel your visitors have old phones. Remember, though, you'll be adding more web zombies that we'll all have to deal with some day.

Use your site statistics and carry out some website analytics to help you come up with an educated decision.

Every browser will have its own level of support (it's not all about the device itself).

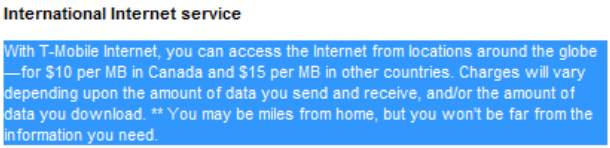
Speed and Cost (to the User)

Ultimately, whichever language you choose, the primary considerations you need to think about is speed and user cost.

It's well known that most mobile internet providers cap connections, and therefore bandwidth has now become a limited and valuable resource.

Even worse is the issue that roaming charges outside of the country you reside in can be expensive, which is a reason to keep the sizes of everything on your site as low as possible.

With caps, costs and speed issues, the need to keep markup as clean, small and standards-based as possible is important.



Roaming charges from mobile web providers can get pretty expensive.

Because of the speed in which adoption is occurring for new technologies, the ability and future of using languages like HTML5 and CSS3 is not out of the question — taking into account that your code degrades gracefully, of course.

Many providers such as Apple provide firmware upgrades that improve how the device will function, which means older devices may be able to take advantage of modern standards. But this situation is analogous to IE6 users refusing to upgrade to modern versions like IE8, therefore, ensure you always research before you implement — then test, test and test again!

Layout Essentials

If there's one issue that mobile devices have in spades, it's the issue of how to lay out your web pages. A design's layout in mobile devices is problematic because:

- Mobile devices come in all shapes and sizes
- Mobile devices have different levels of quality and resolutions
- Mobile devices may or may not support zooming, others scroll content
- Scrolling in mobile devices is more difficult because of their small screen

The goal of a mobile web design's layout is to allow the least amount of burden to the user's ability to find (and quickly read) what they're looking.

Essentially, your layout will be important to making your mobile presence a success.

Due to the lack of space on many screens, single column designs may be required.

Simplicity

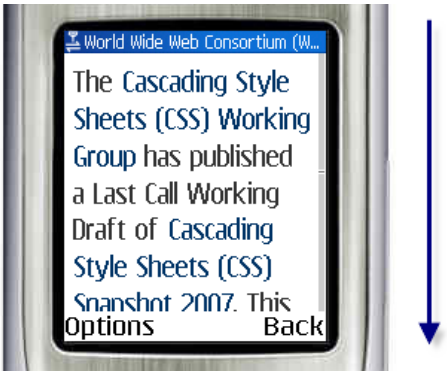
One of the main concepts to an effective mobile web layout is simplicity. It goes without saying that the more information you pile into a small space, the harder it becomes to read and the more scrolling that will be required.

With such limited space to contend with, multi-column layouts often break because the required space to meet the needs of the content cannot span beyond the physical space of the viewport unless passive zooming and scaling comes into play.

Therefore, it pays to use a single column layout.

Avoid Scrolling

Some mobile devices, like the iPhone and iPad, have the ability to adjust a web page's zoom depending on the orientation of the device (portrait or landscape mode), which reduces the need for scrolling; but not all devices have this ability.



Having to scroll down a content-heavy page isn't a fun experience on cell phones.

We all know that horizontal scrolling isn't a good idea — especially on the iPad where scrollbars don't show up until you attempt to scroll — so avoid this situation in your mobile web designs.

ALA:*mobile* #310

ARTICLES TOPICS ABOUT FEED

JULY 20, 2010

Learn how to create inline SVG with Raphaël and take JavaScript minification to the next level.

SVG with a little help from Raphaël

by BRIAN SUDA

Want to make fancy, interactive, scalable vector graphics (SVGs) that look beautiful at any resolution *and* degrade with grace? Brian Suda urges you to consider Raphaël for your SVG heavy lifting.

A good mobile design should have a clean layout with simple navigation options.

Size of Navigation and Clickable Objects

Another key component is the issue of navigation and clickable regions, which is predominantly a problem with touchscreen mobile devices.

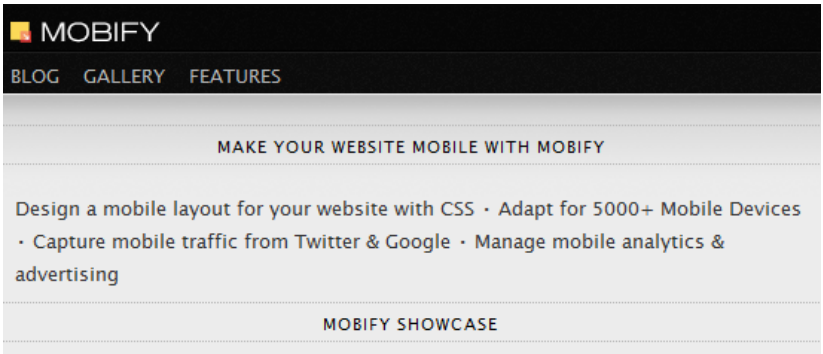
I'm sure if any of you have big hands, you are well aware of what a pain typing on a handheld keyboard can be or how hard it is to click on something small on the screen without having to zoom into it.

Ensuring that your mobile layout has large and easy-to-press links and clickable objects will be essential in streamlining the experience.

Reducing the amount of clicks required to achieve an action — which is a good practice regardless of whether or not you're designing a mobile site — is all the more important in mobile web designs.

Content Design

With the cost of browsing the web and caps on data allowances being put in place (along with speed issues), the most costly component of a website is the content. Knowing how to reduce excess images, text and media can be the difference between a 50KB design and a 2MB layout of crippling intensity.



Less is more on a mobile device; less content equals more likely to read.

Text Content

Of all the components of a site, none plays a more vital role than the text.

But while content is king even on handheld devices — the need for scrolling, small file sizes, quick readability and bandwidth restraints means that we have to reengineer our copy to ensure that it's useful on such devices.

If your design is simply a modification/adaptation of your existing layout (the first method of delivery), you could decide to hide unnecessary text, images or media (even though they'll still download, it will improve readability). The real advantages come from a separate design where you can purge the marketing talk and excessive content.

Images

When working with a small screen, large CSS background images or byte-heavy infographics can be problematic. While some handheld devices have larger displays where this is not an issue, and while the ability to zoom into graphics on devices like the iPhone shouldn't go without some credit, unnecessary bulk for visual embellishments certainly needs to have a good clean up.

Large images sap a lot of bandwidth; consider scaling them back for smaller screens.

Reducing the resolution and dimensions of your images can literally be the difference between 50KB and 500KB — bandwidth consumption that's worth saving.

Video/Audio

It's inevitable in the modern web that utilizing audio and video will be needed. Even with the bandwidth issues that exist, you shouldn't stop using these richer forms of content, as they can be great, especially on handheld mobile devices that have excellent video/audio quality such as the iPhone or iPod Touch. But just like with everything else, moderation and smart usage is key.

There are a few considerations you should make when utilizing video and audio:

- The format you use: Beware of Flash and other closed formats that aren't compatible on some devices
- The file size of video and audio: Optimize your files
- Don't automatically download video/audio files until requested: For bandwidth savings
- Don't auto-play: It's annoying; in fact, don't do this even outside of mobile devices

Other Issues to Consider

Finally, it's important we address the best practices when it comes to scripting, plugins (like Flash and Silverlight), and developing web apps.

Knowing what to cut and what to keep will help enhance your mobile user's experience and also ensure that your mobile website is functional across most/all devices.

BBC © MMX The BBC is not responsible for the content of external sites. Read more.

This page is best viewed in an up-to-date web browser with style sheets (CSS) enabled. While you will be able to view the content of this page in your current browser, you will not be able to get the full visual experience. Please consider upgrading your browser software or enabling style sheets (CSS) if you are able to do so.

Allow your code to degrade gracefully as you can't rely on any technologies success rate.

Interaction in Mobile Devices vs. Personal Computers

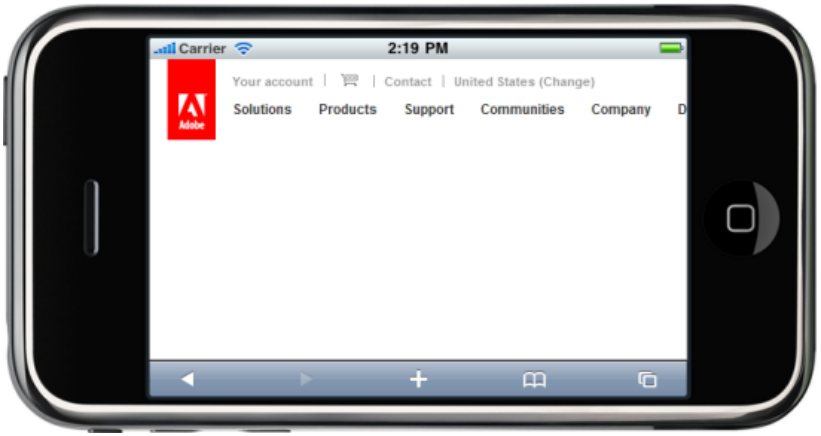
An important point to make is that we interact differently with a mobile design screen versus a regular computer screen.

With the lack of a mouse and the way our hands gesture to instigate actions and reactions, the traditional interaction patterns we're accustomed to, such as hovering over a link (for example), is different.

Consideration must be given to how such functions are affected by a change in interfaces.

Proprietary Technologies and Plugins

The move of Apple to block Flash from their devices underscores the problems of becoming dependent on proprietary plug-in components — closed technologies that other companies can't or won't support.



Adobe Flash isn't supported on the iPhone, iPad or iPod Touch, which is problematic.

Apple's decision against Flash can be a harbinger of things to come, setting a precedence about the way mobile device manufacturers welcome third-party technologies into their own. Other technologies

like Silverlight or Java may not work as intended — or may later be blocked as well.

While many developers may use this as an excuse not to develop on these platforms, the best course of action is simply to ensure that their mobile websites degrades gracefully.

Building an app may be useful if internet access is unavailable (for any reason).

Web Services with Persistent Internet Connections

Even though the availability of web-based services are fantastic, I do worry that the dependence on a constant and reliable (always on) web connection is very much going to be a problem for web apps at the current state of mobile device networks.

While there have been moves towards local storage mechanisms, for now, web apps that rely on persistent internet connections could affect mobile device users due to the capabilities of their networks.

For example, the fact that there are still "dead zones" — places where mobile phones don't have service — can affect the user's interaction, such as in cases where his or her signal suddenly drops in the middle of performing a task.

It's worth considering the idea of developing an app for your service which can function both offline and online (learn how by reading this offline HTML5 iPhone app tutorial).

Testing Your Mobile Website

If you've ever been into a store that sells phones, it can be downright shocking how diverse the screens, devices and contract plans can be.

There are a wide range of emulators for simulating your designs.

With the future set to bring even more mobile devices into the fray, and because we are at the mercy of corporations that want to gain or maintain their competitive edge, the standardization of these web-enabled devices is unlikely to occur.

Therefore, it's up to our common senses to do what we can to ensure that the widest possible audience can access and use our site in a way that's functional and enjoyable.

Testing with Mobile Device Emulators

With such diversity in the mobile device landscape, it goes without question that you should test your designs on as many platforms as you can manage. Below is a list of emulators that will simulate certain devices for you to be able to test your work.

- Android emulator - https://developer.android.com/studio/index.html
- Blackberry emulator - http://www.blackberry.com/developers/downloads/simulators/
- Dot Mobi emulator
- Firefox Mobile emulator
- iPhone / iPad / iPod Touch emulator (xCode via App Store)
- WML emulator - https://www.winwap.com/desktop_applications/winwap_for_windows.php
- LG emulator
- Microsoft Devices emulator - https://www.microsoft.com/en-us/download/details.aspx?id=25191
- Motorola emulator
- Mozilla Fennec emulator
- NetFront emulator
- Nokia emulator
- OpenWave emulator (archive) - http://wapreview.com//?p=3733
- Opera Mini emulator - https://dev.opera.com/articles/installing-opera-mini-on-your-computer/
- Opera Mobile emulator - https://www.opera.com/developer/mobile-emulator
- Palm emulator - http://www.osnews.com/story/29602/The_elusive_Palm_OS_5_5_Garnet_emulator_for_Windows_Linux
- Palm Pre / iPhone emulator
- Samsung Java emulator
- Samsung Platform emulator (Android Studio)
- Windows Mobile emulator - https://docs.microsoft.com/en-us/windows/uwp/debug-test-perf/test-with-the-emulator

Simple, Small and Speedy

While much of what I discussed in this article is straightforward advice, common sense is a major factor that dictates how we build interfaces.

Back in the 56k modem days, we had speed issues to contend with. We also had monitors that were limited in resolution and color. Many ISPs capped our bandwidth and internet access. Some internet connections would drop if you have an incoming phone call to your house. So for those of you older-generation developers — you should be in familiar territory.

For now — and until mobile network infrastructure improves and connectivity is widely available — simple, small and speedy are the three main principles we should abide by.

Sources:

- http://wapall.com/en/main.asp
- https://www.w3.org/TR/css3-mediaqueries/
- https://en.wikipedia.org/wiki/User_agent#User_agent_identification
- http://2010.dconstruct.org/
- https://www.mobify.com/
- https://en.wikipedia.org/wiki/XHTML_Mobile_Profile
- https://www.opera.com/mobile/mini
- https://www.nngroup.com/articles/scrolling-and-scrollbars/
- https://en.wikipedia.org/wiki/Three-click_rule
- https://www.apple.com/iphone-x/
- https://www.bbc.co.uk/
- https://html.spec.whatwg.org/multipage/webstorage.html
- https://gigaom.com/2010/04/09/419-10-q-watch-adobe-admits-apples-anti-flash-strategy-could-be-damaging/

CSS3 Card Trick: A Fun CSS3 Experiment

This tutorial is based on a simple animated experiment that showcases just one of the amazing things you can create using CSS. I've used no images and no scripting; everything's done using HTML and CSS.

It goes without saying that since CSS3 is still not supported by all browsers, it might not work as intended; but I've coded this in such a way that it will degrade gracefully on non-CSS3 browsers, including IE (of course).

Experimenting on cutting-edge standards for the sake of innovation is an attribute that helps us learn, and perhaps by pushing the boundaries, we can improve our knowledge further.

A Primer on Innovation

If you ask any self-taught professional, motivation and willingness to experiment are the two primary skills that feed their passion to becoming better designers and developers.

While memorizing every single element, property, function and attribute for every language is an option, knowing what they can do is even better.

Whether you are working out how to fix a browser bug, achieving a complex layout, or if you simply want to play around and see what you can come up with (like this example), the ability to hone your skills to match the situation puts you in a fantastic position for whatever the world may throw at you.

Now that the inspiring speech about the justification behind this fun experiment is over, it's time we begin examining and reconstructing this example to display how everything came together to produce the final result.

Within this experiment we will be taking advantage of some cool CSS3 code (which you can use in other projects) such as border-

radius, box-shadow along with the target and checked pseudo classes.

Oh, and if that wasn't enough, we're even going to use a few WebKit transformations to give Chrome and Safari (they support animations) a progressively-enhanced experience.

So what I hope will happen is that by going over the process of creating these CSS3 playing cards, you'll have fun learning about these new CSS3 capabilities.

Fragments and Fieldsets

Like with all websites, we need to begin right at the beginning with some HTML that lays down the tracks for the experiment.

For the purpose of this example, we'll keep the CSS and JavaScript used inline inside the document so that you can easily use the View Source feature in your web browser while studying the demo. But in production, I hope you will keep your CSS and JavaScript external.

In the code block, you will find:

- An unordered list that will feature the various card suits
- A paragraph of text which simply describes the experiment on the page
- A (very complex) web form that will hold the stylistic variables which make up each of the cards

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;
        charset=utf-8" />
        <title>Playing Cards with CSS3!</title>
        <style type="text/css">

        </style>
        <script type="text/javascript">
```

```
    </script>
</head>
<body>
    <ul>
        <li><a title="Select Spades" href="#spades">&#9824;</
        a></li>
        <li><a class="fire" title="Select Hearts"
        href="#hearts">&#9829;</a></li>
        <li><a title="Select Clubs" href="#clubs">&#9827;</a></li>
        <li><a class="fire" title="Select Diamonds"
        href="#diamonds">&#9830;</a></li>
    </ul>
    <form action="">
        <fieldset id="spades">
            <input class="card" id="spade" type="radio"
            name="spade" value="spade" />
            <label class="base" for="spade" title="This is the Ace of
            Spades!">
            <span><em>A</em>&#9824;</
            span><strong>&#9824;</strong><em>A</
            em>&#9824;
            </label>
            <input id="cancel1" type="reset" name="spade"
            value="cancel" checked="checked" />
            <label class="close" for="cancel1">Spades</label>
        </fieldset>
        <fieldset id="hearts">
            <input class="card" id="heart" type="radio"
            name="heart" value="heart" />
            <label class="base fire" for="heart" title="This is the
            Ace of Hearts!">
            <span><em>A</em>&#9829;</
            span><strong>&#9829;</strong><em>A</
            em>&#9829;
            </label>
```

```
            <input id="cancel2" type="reset" name="heart"
            value="cancel" checked="checked" />
            <label class="close" for="cancel2">Hearts</label>
        </fieldset>
        <fieldset id="clubs">
            <input class="card" id="club" type="radio"
            name="club" value="club" />
            <label class="base" for="club" title="This is the Ace of
            Clubs!">
            <span><em>A</em>&#9827;</
            span><strong>&#9827;</strong><em>A</
            em>&#9827;
            </label>
            <input id="cancel3" type="reset" name="club"
            value="cancel" checked="checked" />
            <label class="close" for="cancel3">Clubs</label>
        </fieldset>
        <fieldset id="diamonds">
            <input class="card" id="diamond" type="radio"
            name="diamond" value="diamond" />
            <label class="base fire" for="diamond" title="This is the
            Ace of Diamonds!">
            <span><em>A</em>&#9830;</
            span><strong>&#9830;</strong><em>A</
            em>&#9830;
            </label>
            <input id="cancel4" type="reset" name="diamond"
            value="cancel" checked="checked" />
            <label class="close" for="cancel4">Diamonds</label>
        </fieldset>
    </form>
    <p>Select an option above to change the suit displayed!</
    p>
  </body>
</html>
```

Unicode Characters for the Card Suits

While much of the HTML above is straightforward, there is something worth mentioning for its value in our discussion. Within both the list and the form, you will notice that there are a lot of strange Unicode escape characters (such as ♦) which are embedded at various points.

When you view the page in your browser, you will notice that these characters represent each suit (spades, hearts, clubs and diamonds) and therefore we can give the cards their appearance without any images.

The list has one mention for each icon, and each form has 3 for each (for the corners and centerpiece).

With the HTML finished, you should see the list, table and text (plus those cool characters).

What's With The Web Form?

You may be curious about the monstrous HTML web form. The justification for such code is simply due to the way the CSS3 will work with the elements. While the list element uses simple fragment links (attached to the frameset container for each card), each card — when it appears — contains two input elements with associated labels.

The first label contains a mixture of span and em elements (to give the card value in the corners) and a strong element (for the big central character).

The second label simply acts as a reset mechanism for the special effects, i.e. when a WebKit browser is being used.

Laying the Foundations

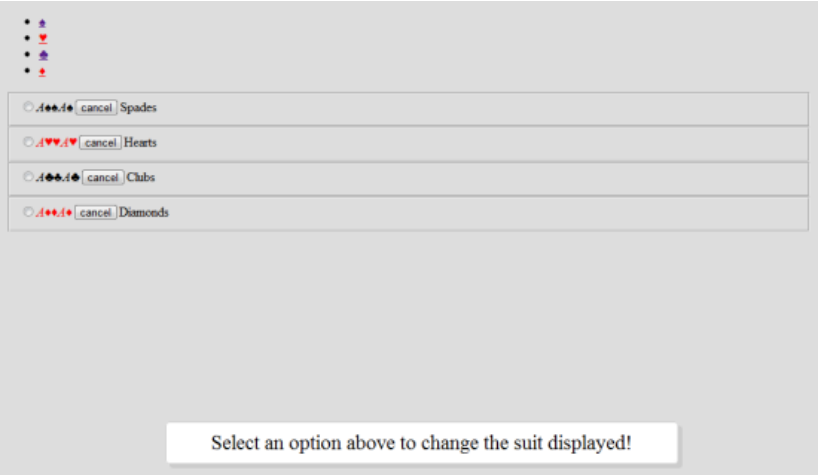
Now that we have completed the HTML, it's time to move onto the CSS. To begin, we will add in all the CSS content for the web page; and we also need to give our cards a bit of color.

The below code begins our journey into CSS3 by taking advantage of both the border-radius and box-shadow CSS properties (with a few compatibility tweaks).

IE Support

It's worth noting that the filter property will not validate under W3C CSS standards due to it being a proprietary extension for Microsoft's Internet Explorer; but as this is just for fun, the validation of the code is less important. This allows us to support Internet Explorer; that's a good enough reason to break auto-validation in this case.

```
body { background: #DDDDDD; overflow: hidden; }
body .fire { color: #FF0000; }
p {
    background: #FFFFFF; border: 1px solid #CCCCCC;
    border-radius: 5px 5px; -moz-border-radius: 5px; -webkit-
    border-radius: 5px;
    box-shadow: 5px 5px 5px #CCCCCC; -webkit-box-shadow:
    5px 5px #CCCCCC; -moz-box-shadow: 5px 5px #CCCCCC;
    filter:
    progid:DXImageTransform.Microsoft.Shadow(color='#CCCCCC'
    , Direction=135, Strength=5);
    font-size: 25px; text-align: center;
    height: 30px; width: 600px;
    margin: -35px -300px; padding: 10px 15px;
    position: absolute;
    bottom: 50px; left: 50%;
    z-index: 99;
}
```



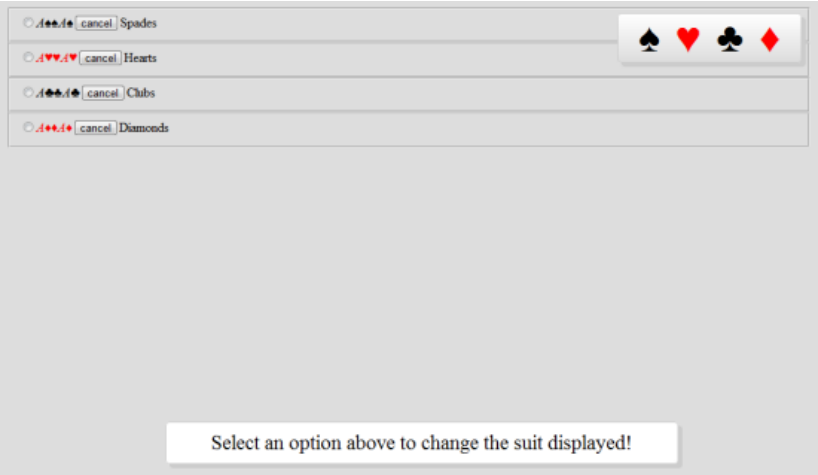
Using the above code, you should see a lovely looking paragraph on your page!

Navigation, Simplified!

Next on the list is the navigation. For our playing card experiment, we shall be making use of anchor fragments that will show and hide each of the cards (as required).

While the code you'll see next doesn't contain the CSS3 that will activate the navigation (we'll get to that later), you will notice the code that not only floats the navigation to the top-right hand side of the screen, but also adds the same cool border-radius and box-shadow effects. When you refresh the page after plugging in this code, you should see that the links are now highly visible and make use of our awesome Unicode characters.

```
ul {
    background: #FFFFFF; border: 1px solid #CCCCCC;
    background:-moz-linear-gradient(top, #FFFFFF, #DDDDDD);
    background:-webkit-gradient(linear,0 0, 0 100%, from(#FFFFFF),
    to(#DDDDDD));
    border-radius: 5px 5px; -moz-border-radius: 5px; -webkit-
    border-radius: 5px;
    box-shadow: 5px 5px 5px #CCCCCC; -webkit-box-shadow:
    5px 5px #CCCCCC; -moz-box-shadow: 5px 5px #CCCCCC;
    filter:
    progid:DXImageTransform.Microsoft.Shadow(color='#CCCCCC'
    , Direction=135, Strength=5);
    font-size: 50px;
    margin: 0;
    padding: 0 15px;
    position: absolute;
    right: 20px;
    top: 15px;
    z-index: 99;
}
ul li {
    display: inline;
    list-style-type: none;
}
ul li a {
    color: #000000;
    display: block;
    float: left;
    padding: 0 10px;
    text-decoration: none;
}
```



Now we have the list hovering and ready for action (along with that paragraph).

Styling the Aces

The most complicated part of the CSS is making the cards look like, well, like playing cards. Using the code that follows will get you the perfect and progressively enhancing design elements that make the final look possible.

Code to pay attention to includes:

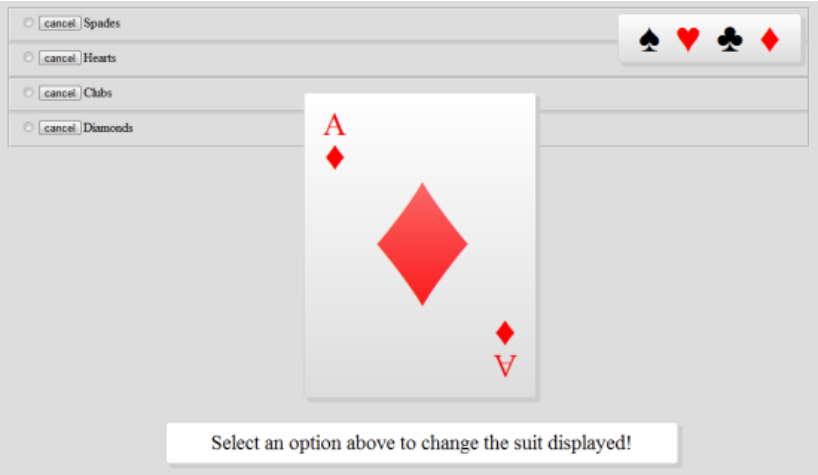
- The CSS3 linear gradient that fade the cards
- The reversed gradient using the mask-image property on the center character (providing an even softer feel)
- A span label effect which rotates the bottom right reference to make it upside down (just like real cards)

All simple but very effective.

```
.base {
    background: #FFFFFF;
    border: 1px solid #CCCCCC;
    color: #000000;
    background:-moz-linear-gradient(top, #FFFFFF, #DDDDDD);
    background:-webkit-gradient(linear,0 0, 0 100%, from(#FFFFFF),
    to(#DDDDDD));
    border-radius: 5px 5px; -moz-border-radius: 5px; -webkit-
    border-radius: 5px;
    box-shadow: 5px 5px 5px #CCCCCC; -webkit-box-shadow:
    5px 5px #CCCCCC; -moz-box-shadow: 5px 5px #CCCCCC;
    filter:
    progid:DXImageTransform.Microsoft.Shadow(color='#CCCCCC'
    , Direction=135, Strength=5);
    height: 360px;
    top: 50%;
    margin-top: -180px;
    width: 260px;
    left: 50%;
    margin-left: -130px;
    z-index: 9;
    cursor: pointer;
    font-size: 50px;
    text-decoration: none;
    padding: 15px 0 0 25px;
    position: absolute;
}
strong {
    font-size: 250px;
    position: absolute;
    left: 50%;
    top: 50%;
    margin-top: -160px;
```

```
    -webkit-mask-image: -webkit-gradient(linear, left top, left
    bottom, from(rgba(0,0,0,0.4)), to(rgba(0,0,0,1)));
}
em {
    font-size: 40px;
    font-style: normal;
    display: block;
    margin-bottom: -15px;
}
label span {
    -webkit-transform: rotate(-180deg); -moz-transform:
    rotate(-180deg); -o-transform: rotate(-180deg); filter:
    progid:DXImageTransform.Microsoft.BasicImage(rotation=2);
    position: absolute;
    bottom: 15px;
    right: 25px;
}
#spades strong { margin-left: -68px; } #spades em { margin-left:
0; }
#hearts strong { margin-left: -70px; } #hearts em { margin-left:
1px; }
#clubs strong { margin-left: -80px; } #clubs em { margin-left: 3px; }
#diamonds strong { margin-left: -60px; } #diamonds em { margin-
left: -2px; }
```

Note: Depending on the browser used, the effects may look different. Chrome, Safari both support all the fancy stuff, Firefox will have the background gradient (but the center character will look strong), and Opera and IE will have no "worn" effects!



With some clever coding, the playing cards should now look easy on the eyes.

Navigation and Animation

Using the code above, your cards should appear like they come from a proper deck of playing cards. Now that we have the cards, navigation, and paragraphs looking all pretty, it's important that we get each of the cards appearing on demand. And we should also do something with the cancel and radio buttons that are hovering around behind the scenes.

What happens next is where things get really interesting for you WebKit users.

```
fieldset { display: none; }
fieldset:target { display: block; }
fieldset:target .card+label { -webkit-animation-name: scaler; -
webkit-animation-duration: 1.75s; -webkit-animation-iteration-
count: 1; }
fieldset:target .card:checked+label { -webkit-animation-name:
effectx; -webkit-animation-duration: 3s; -webkit-transform: scale(0);
}
.close {
    background: #DDDDDD; cursor: default;
    left: 0;
    top: 0;
    position: absolute;
    height: 100%;
    width: 100%;
    z-index: 1;
    text-indent: -999em;
}
@-webkit-keyframes scaler { from { -webkit-transform: scale(0); } to
{ -webkit-transform: scale(1); } }
@-webkit-keyframes effectx {
from { -webkit-transform: rotateX(0deg); }
to { -webkit-transform: scale3d(1.2, 1.2, 1.2) rotateX(-90deg)
translateZ(500px) rotate(180deg); -webkit-animation-duration:
30s; }
}
```

Explaining the above code is very simple. With the anchor links, we are using fragment URLs to navigate between cards in the deck and therefore the target pseudo class hide and show the fieldset as its ID appears.

The .close class takes its place behind the card covering the full browser viewport so that upon clicking the page (once the card is hidden), you can restore the item back to its original position.

This is relative to the usage of radio buttons as we shall be using the CSS3 :checked pseudo to animate the content for WebKit browsers depending on the check state.

The animation and effects may not show, depending on your browser.

The main features that WebKit users will be excited about are the two card classes that trigger either the scaler keyframes which will make the card zoom in (from nowhere) or the effectx keyframes that will rotate and flip the card which makes it appear as if it's being pushed over (and falling off the page).

We can also add some simple effects that work alongside the checked property to give some lovely experimental fun stuff that you can interact with (if you use Chrome or Safari). Other browsers will simply ignore the actions.

Note: With most WebKit animations, once the effect has run its course, it will be reset to a default point. Because this example required the effect to remain permanent (until directed by the user otherwise), I made use of the checked state pseudo and radio buttons to ensure the effects were held firmly in place.

Dealing with Internet Explorer

OK, so remember in the beginning when I said I've used no scripting? Well that's only partially true.

Because IE doesn't yet support all these things, we need to use JavaScript so that at least our IE users will have a decent experience.

Of course, if you don't care about IE — after all, this is an experiment for cutting-edge browsers that support CSS3 — then, yes, this CSS3 experiment uses no scripting whatsoever.

Luckily, because IE — like other non-WebKit browsers — don't support the CSS3 animated effects, the need to deal with the checked state (in CSS3) is moot since the effect will be non-functional anyway.

However, one thing that should be available is the target pseudo so that IE users can change the suit of the playing card. Using a small bit of conditional JavaScript, we can replicate the effect with little impact on the browser. There's also a fragment redirect in the script to help browsers load the first card.

```
function bootup(){
    if (location.hash == "") { location.hash="#spades"; } var tds =
    document.getElementsByTagName("a"); direct();
    for( var x=0; x < tds.length; x++ ){tds[x].onclick = function()
    {setTimeout(direct, 1);};}
}
function direct(){
/*@cc_on @if (@_jscript_version > 5.6)
    var counted = document.getElementsByTagName("fieldset");
    for( var x=0; x < counted.length; x++ ){ counted[x].style.display
    = "none" }
    document.getElementById(location.hash.substr(1)).style.display
    = "block";
@end @*/
}
window.onload=bootup;
```

The example should now be complete and have some neat special effects!

While the above code should be enough to get the basic effect working on the latest version of IE, its worth highlighting that I didn't spend much time attempting to get the effect functional on really old browsers like IE6.

Because this is a proof-of-concept demonstration — and because it exists just to play with what's available in up and coming standards — there was no incentive to try and fire the effects onto every browser.

It's worth saying, however, that if you do intend on using any of this in a production site, you should support the browsers your visitors use.

Playful Innovation

Just a few years ago, the ability to produce such a textured playing card that looks like an image would not have been possible without the use of images. The animation effects, without scripting or Flash, would have been beyond most people's dreams.

Our proof-of-concept shows the benefits of modern standards, and what the future is for us web designers and web developers.

The need to keep innovating and pushing our browsers to the limit is an important part of the web's evolution.

As a web professional, I always try to spend time learning and examining my own abilities to see what unique solutions I can produce. Ideally, after reading this, you will be inspired to experiment on your own.

Sources:

- https://en.wikipedia.org/wiki/List_of_Unicode_characters

Designing By Numbers: Data Analysis for Web Designers

Judging what's best for an audience is never far from the web designer's mind. The ability to predict whether a web design will soar like an eagle or sink like the Titanic is among the most subjective and complex measurements you will encounter.

While resources that explain best practices exist, and your visitors contacting you about serious issues and offering you feedback relating to your site will occur if you have the proper mechanisms in place — it's ultimately your responsibility to be proactive and research, investigate, and determine the what, why and how to ensure widespread usability.

Designing by Numbers

Before we examine the types of statistical information you should be looking at — and the relevance they have to your web design projects — we first need to go over the 3 single-word questions that relate directly to all the design decisions you will make.

These 3 questions are ultimately at the heart of your research, analytics and motivation behind designing by the numbers.

What, why, and how is a simple design process that:

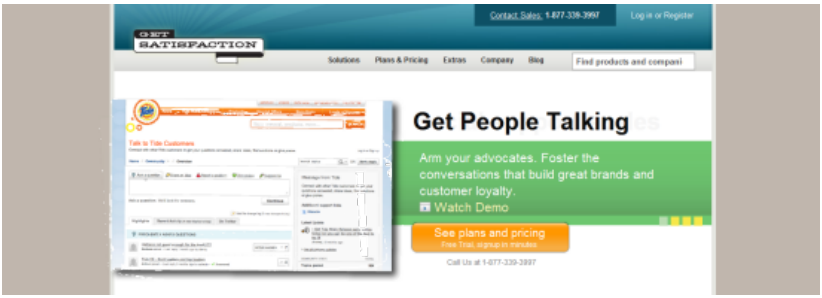
1. Defines what the issue is
2. Proves why it is an issue
3. Determines how to fix the issue with the optimal solution (if it is an issue)

What?

Of all the questions that may enter the mind of a web designer, "What?" is probably the word that relates to the task at hand. The process of understanding relevance and the usefulness of information explicitly relates to the decisions we undertake.

- What do site users need?
- What things frustrate site users?
- What can I do in this design to accomplish the site's objectives?
- What's wrong with the site?
- What's right about the site?
- What can be made better?

Asking "What?" will yield to a lot of information that will help you make optimal design decisions.



What your audience requires is a fundamental principle of designing by numbers. Get Satisfaction is a feedback tool you can use to help you design by the numbers.

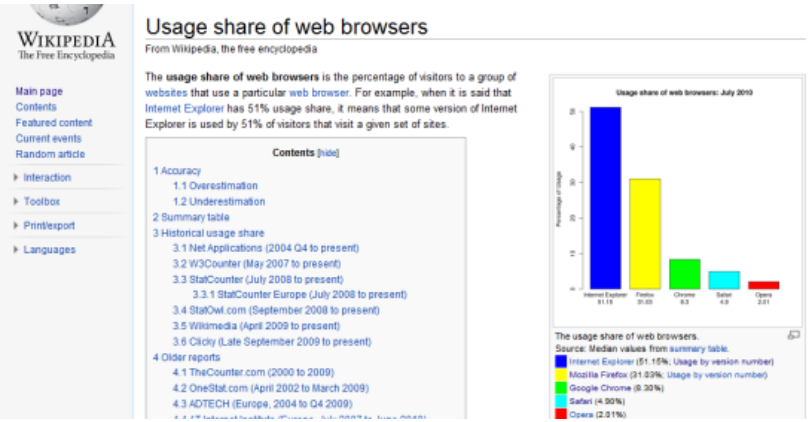
Why?

Next on the list of list of determining factors is the question of "Why?"

Because making changes or implementations beyond what you initially set out to achieve may cost time, money or resources — the ability to back up your ideas with hard data and facts will be enough to even make the bean-counting bosses go weak at the knees and take your professional guidance and ideas more seriously.

- Why are people not using the comments?
- Why is the community participation on the site low?
- Why are users having trouble finding what they need?
- Why do we need to support Internet Explorer 6?

Knowing what needs to be done is one thing — knowing the justification to why it needs to be done is another.



Reasons for why cross-browser support should be implemented can easily be seen when you calculate the percentages of users that use certain browsers.

How?

The last single-word question is "How?" which makes sense in that once you know what needs doing and why it's required, the method of actualizing the "What" is important.

- How should I go about increasing user engagement?
- How can this design improve community participation?
- How can I fix the issue of users not finding the product they need?
- How can I create a design that works in Internet Explorer 6?

Statistical/Gathering Methods

When determining the best course of action for your visitors, there are 3 essential statistic types that will come into play in helping to answer the "what" question. (We shall come to the "how" and the "why" later on, so don't worry!).

Each of these data gathering techniques have their own benefits and pitfalls so there isn't ultimately a perfect solution.

However, designers wanting a well-rounded experience would be better off using a mixture of all 3 as they not only give you a range of quantitative results (raw numbers) but also qualitative research (such as open-ended responses and feedback).

On-Site Data

On-site data are the kind of information you obtain from website analytics software and monitoring user activity on your website.

While this type of data is often ideal in that they relate directly to your visitors, it often takes a while for activity on a new website to build up — and as such, depending on these alone may leave you in the dark as to your visitor's basic primary needs upon launching the service.

In sites with limited or no traffic, or sites that are still in development — analytics software fails because there is no (or limited) data sources; you're pretty much in the dark.

Most websites have some kind of visitor tracking mechanism installed, such as Google Analytics.

Third-Party/Generalized Data

Independent data are often the most useful to new websites, usually produced by large firms who provide demographic services like Net Applications or W3Counter.

These third-party data-gathering sites offer a glimpse at the general population, and by that, it will include useful details such as the browsers and devices they use, their country of origin, and so forth.

On-Site data gathering methods is going to be more accurate and will reflect your particular situation much better — for example, a web development blog will have a different audience than a cooking blog) — but accounting for independent statistics can aid you by providing a baseline to work from, especially if you have no user base.

There are plenty of statistics on the web, you just need to look! The figure above shows statistics from A List Apart's survey of web designers.

Social Data

Socially-sourced data are a relatively new concept that has come out of the rise of networking sites like Facebook or Twitter, where people can promote or discuss your creation through an external site.

While there are still a large number of people who aren't interested in the "social" aspect of social networking, the importance of leveraging these statistics of what visitors like, dislike and their comments attributing to such information can actually be more useful (in different ways) to the conventional number-based statistics from analytics packages.

Social networking can provide you with useful feedback to work with.

Designers Demographics

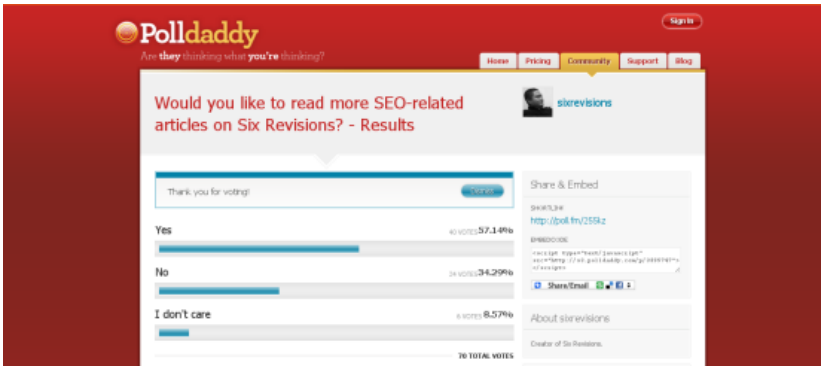
Now that we have covered the "what", we need to examine the "why" (and by association) the need to focus our attention on all the pretty percentages, pie charts and graphics that appear everywhere.

Ensuring your visitors can use and enjoy their experience with your web design is important and determining how we can provide that experience will all be down to using the statistics methods above and then narrowing the focus down onto what is most relevant for your audience.

While pretty numbers may seem impressive on their own, they're not worth anything if they don't speak to your niche, so successful sourcing of your data is critical.

Review websites are notorious for having subjective criteria of questionable validity.

If you've exhausted local statistics and have a general idea of the visitors you're getting (and perhaps where they found you), and if you've gone further afield to seek out related demographics relating to research on an area which affects your niche, its worth going beyond the number crunching and seeking out "intelligent hits" that may help guide your decision making. Asking your community (or perhaps your competitions if you don't have one!) what would enhance the experience can be great, just don't try to please (or annoy) everyone and only implement what will benefit your users!



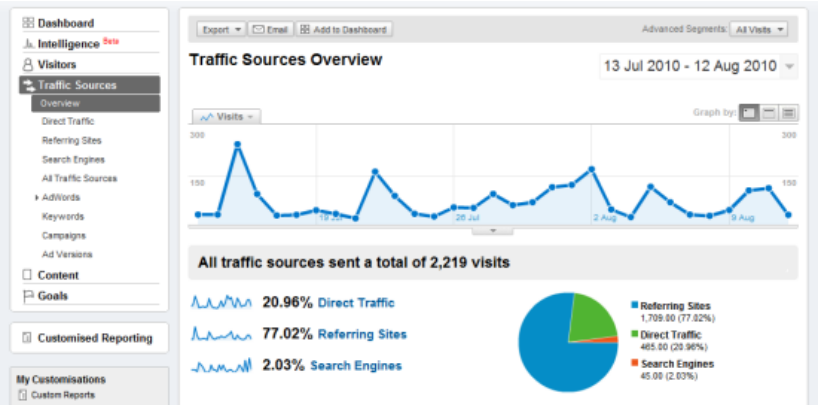
Getting to know your visitors can simply be a matter of knowing how to communicate.

With all of this information in regards to what you're investigating the "why" (as in why changes need to be made) will become quite apparent. While it may seem natural, it's quite easy to become so fixated on the number of visitors or re-tweets we get, that we actually ignore the most important thing a statistics package (or some solid research) can tell us – that our visitors will have their own specific set of needs and requirements that need addressing. As a final point on the matter of "why", if we don't actively seek out ways to improve ourselves, we can't hope to gain new customers.

Visitors may have JavaScript disabled which could leave them excluded from statistics.

A Quick Measurement

The next thing to take into account is how to filter the information once you've collected it (which meets the "how" element). Having lots of statistics and ideas may help, but filtering the stream of data will be critical to making sense of the best route to take in fixing a common problem or deciding the next step. The simplest way to prioritize your data is to follow the below, the higher up on the list the item is, the better and more potentially useful and reliable your research will be. Once the best information is extracted, you can refer to the numbers when making decisions for the design.



Determining the quality of your information is a mission critical part of the process.

Importance of Location:

- Local
- Independent
- Social

Importance of Type:

- Statistics
- Research

Importance of Reliability:

- Proven
- Trends
- Unproven

Importance of Margin:

- Significant
- Proportionate
- Insignificant

Note: Using the above, a locally sourced bunch of statistics that are proven (by a significant margin) to be the best course of action would ultimately be the peak of what you can gather. Though as your research will lead to talking with customers, individual needs should be accounted for as well.

Variable Considerations

Before rounding up this article, it's important that we consider the variables which may impact your statistics. While it's great that there are plenty of studies that may assist you in decision making (like how to build a perfect font stack or what browsers you should support), it's very important that we highlight the issues that will break down the cold harsh numbers and give you a little more to work with. Without making this article particularly heavy going (which isn't the intention), the two types of variables you want to consider are mechanical and personal, and both relative to the visitor.

The first of these (mechanical) will directly affect the way in which your visitor interacts with your site, this isn't as a result of their physical being, but more of their circumstances and equipment. In web design it's obvious that the device used, the OS installed, the browser used, the scripting or plug-ins available or something else will affect their experience. While these are usually listed as independent statistics in packages, they are often related to each other in that a single user will contribute to a number of these breakdown listings, thereby it may directly affect the results.

Logo	Browser Name & Developer	Platform
	**Internet Explorer** Microsoft Corporation Download version 9	
	**Firefox (also called Mozilla Firefox)** Mozilla Corporation Download version 3.6.3	
	**Chrome** Google Download version 4.1.249.1064	
	**Safari** Apple Inc. Download version 4.0.5	
	**Opera** Opera Software ASA Download version 10.53	

Nothing forces greater demands on a website than the range of browsers that exist.

The second and probably one of the more important factors are the personal variables. The reason why these variables are so important is because they will often not appear in statistics packages and require you to undertake independent research to get the numbers or determine the viability of catering to their needs. Such factors include the accessibility level being required, the usability of a site (which won't be a number) and the findability of information. While harder to pin down, it still makes sense to account for such variables as they directly and quite dramatically affect visitors.

Note: Error Margins also play a part in statistics, research made by a human rather than a computer can be subject to biases, errors and omissions – some of which may go unnoticed. The significance of information can also fluctuate depending on the audience who visit the site at any given time.

Research Matters

While this article is not a comprehensive guide to research and statistics (there are entire books on the subject), the importance of knowing your visitors is showcased. When you come to build a site or implement a new feature, it's important that you do your homework to avoid falling into a pitfall that could have otherwise been foreseen earlier. Taking the time to understand how products like Google Analytics work, what their weaknesses are and how to get a well rounded overview and an intimate knowledge of your visitors gives you the best possible chance of hosting a great experience.

It's also worth noting that while this article does indeed focus on the numbers and opinions that lead to decision making, it's very important not to forget the individual as a person who visits your realm (no person should be directly treated as a statistic, they are all just as important to the full equation as each other) and while numbers are great for measurements, opinions often lead to the most amount of innovation. With this article highlighting the benefits of research and accounting for more than a personalized view of a site, hopefully you will go on to target a loyal audience in the future!

Sources:

- https://getsatisfaction.com/corp/
- https://en.wikipedia.org/wiki/Usage_share_of_web_browsers
- http://www.netapplications.com/
- https://www.w3counter.com/globalstats.php
- http://alistapart.com/articles/2007surveyresults
- https://webhostinggeeks.com/
- https://polldaddy.com/
- https://www.google.com/analytics/

The Science Behind a Single Page Website

We have all come across them whilst browsing the web, and many of the examples that exist are quite awe-inspiring, the single page website is a paradigm of the modern web in which everything that needs saying can be placed in a single document.

Whilst the single page layout option can lead to overwhelmingly large documents of endless scrolling, a series of clever mechanisms using modern standards and techniques such as CSS3 and Ajax have burst onto the scenes, offering a method of simply giving information as they're required.

This article is on single page websites that use HTML, CSS and JavaScript; we are skipping the discussion of Flash-only websites, which can technically be classified as a single page website as well.

Once Is Enough for Me

It's understandable that not every type of website will be well suited to having "one page to rule them all," however, a common trend that's seen especially in portfolio websites shows that certain sites can benefit from a simple, yet still multi-faceted, single page.

The idea that a website can be created with just one page seems crazy, but with our industry shifting towards advocating simplicity for ease of use, single page web designs have become a viable and effective option.

CSS Zen Garden is a classic example of a single page with multiple layers of interesting bits.

Trends and Tribulations

While traditional designs with multiple pages will always have its place, there are a number of advantages that give the single page website some potential uses for your own projects.

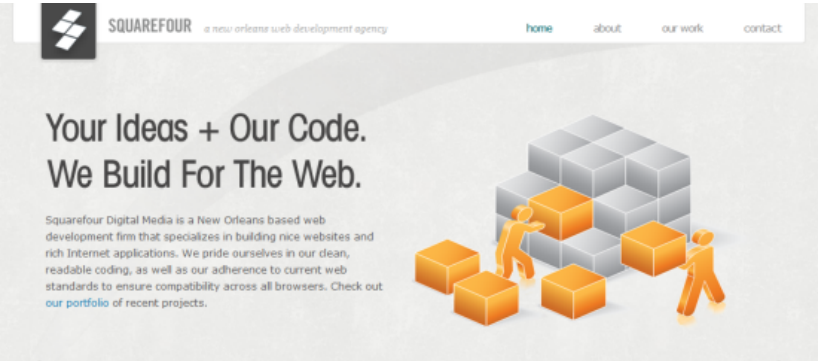
The ability to construct a site that is entirely self-contained gets a bit of getting used to, and involves a lot more thought and planning. Some questions you have to answer are:

- Will a single page meet the project's requirements or will multiple pages be better?
- How do you organize the content?
- How does the navigation work?
- What content do I need and what can I leave out?

Benefits of Single Page Websites

Single page designs have the following advantages over multi-page sites:

- No page refresh when navigating the site (content is either in the page or loaded using Ajax)
- User experience can be improved because navigating through content is quicker and more responsive than having to go to a new web page
- Easier maintenance because you only have to maintain one web page
- You can design for quality over quantity — instead of having to design multiple page layouts for different types of site content, you can focus on just one solid and high-quality design
- Your Google PageRank applies to the whole site
- Higher core content density for search engine spiders
- Distinction from most other websites; single page websites are less common, and can thus leave an impression on your site visitors (and that's why they are popular on portfolio sites)
- Easy solution for simple "brochure" sites that serve one product (i.e. iPhone app) or one purpose (i.e. a designer's work)
- Preferred solution for web apps designed for the Mobile Web



Once the page has loaded, there's nothing else to download.

Disadvantages of Single Page Websites

Single page designs have the following disadvantages against multi-page sites:

- Potentially large file size of the page
- A requirement for scripting or CSS3 support if you want to stand out
- Tabbing through elements can become trickier (for accessibility) because there might be plenty of content on one page (though this wouldn't be a big problem for well-structured markup that use headings and other best practices)
- Producing the design is more time-intensive because it involves a lot more thought and creativity to be able to fit everything in one page and to devise a great interaction design
- The page can take much longer to load if you have a lot of content

File size is an important issue to contend with, especially where Flash is concerned.

The truth is that whenever you implement a specific design pattern, chances are that you will not be able to please everyone. While single page sites can be made to be 100% accessible and highly usable, there will be situations where a single page site is not a good option for you. For example, an e-commerce site such as Amazon.com wouldn't be able to pull off a single page web design successfully because of its vast amount of content — and that's fine because it's better when these types of sites are multiple pages.

Production Theory

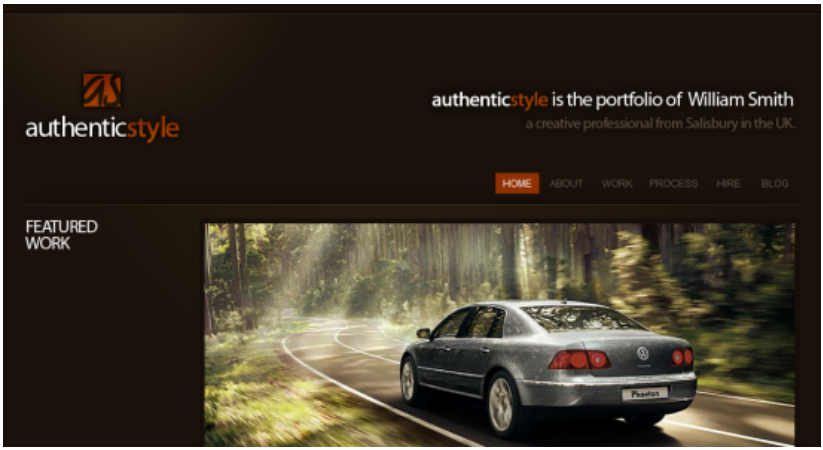
Before we look at some lovely single page designs, it's worth taking a few moments to explain the various mechanisms used to produce such a site. Your emphasis should be on keeping file sizes as slim as possible and about a thoughtful way of presenting and structuring your web page. Think about user flow and interaction design — how does a user move from section to section of the page?

Here are some techniques that are used in single page websites. It's important to note that they are not mutually exclusive, so you might find yourself using them in combination.

Manual Scrolling

The first mechanism implemented by conventional single page designs is to display all of the content on the page, structured logically and laid-out in sections.

The way people navigate through the content is simply by using the native scrollbars in their web browser. While this method is simple to implement because it's just a regular web page with no special interaction, it's also probably the most boring of the options.

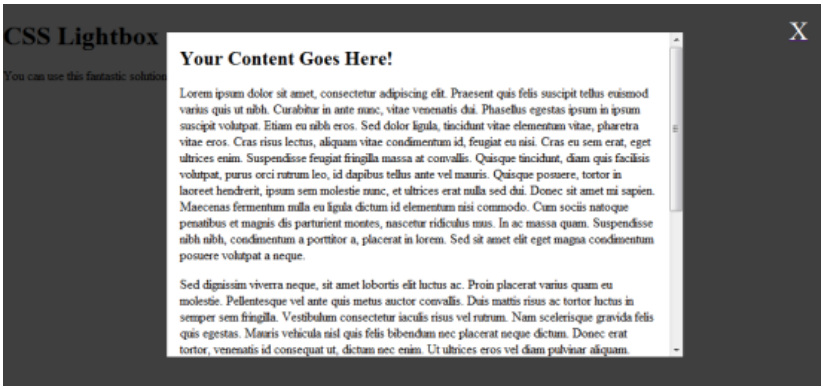


Sites that have no need for fancy effects could easily produce a simple and beautiful single page layout.

CSS3 Interaction

The next mechanism for navigating through content on a single page website worth mentioning is CSS3. With the latest version of the CSS specification, the ability to go beyond existing CSS2 selectors allow for a more unique single page experience. Most notably, you can do interesting, interactive things that deal with content by using CSS transition properties for animation and messing about with the :target and :checked pseudo-classes.

For example, the ability to use the :target pseudo-class (combined with anchor links) gives you the option to make the targeted section a different color or to give it a different background-image.



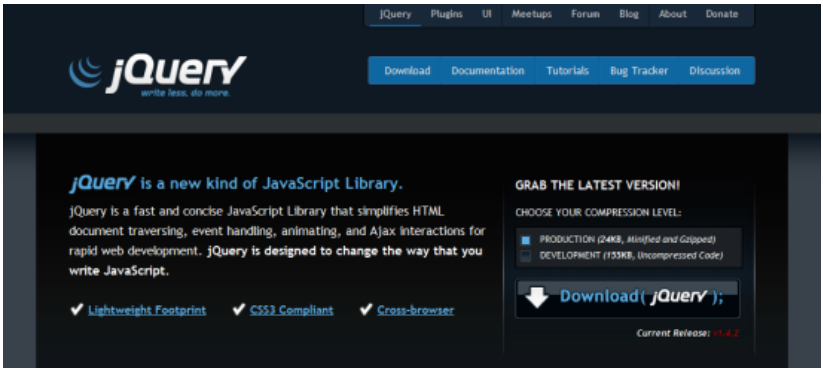
Using CSS3 pseudo selectors, we could form a powerful cross browser "paneling" system.

JavaScript

Finally, we have good old JavaScript, which has been serving us a widespread range of functionality since the web's early days. With the popularity of web development JS frameworks like jQuery, the ability to swap out existing on-page content has never been easier, and with the rise in Ajax, calling content as it's required has an even greater potential for eliminating the need for page refreshes. Take note, though, that there are accessibility and SEO concerns with content that is remotely loaded.

You can also use animated scrolling to sections of the web page using JavaScript — a step up from manual scrolling and using anchor links. For example, check out the jQuery ScrollTo plugin. You can see smooth scrolling in action via Laco Janic's portfolio (click on the primary navigation links such as "identity & print" or "about").

While it's not an option for the poor souls with no scripting knowledge, using JavaScript is certainly the most flexible and robust method out of all three.



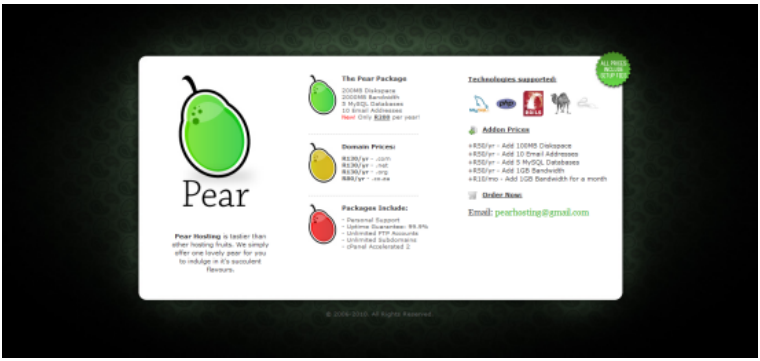
jQuery amongst other scripting frameworks offer easy to implement content swapping.

A Showcase of Single Page Web Designs

As we have now finished examining the general ways that designers and developers go about creating a single page site, it's worth looking at some great single page web designs for inspiration. Perhaps these designs will give you some ideas and inspiration!

Camera+

Webdots

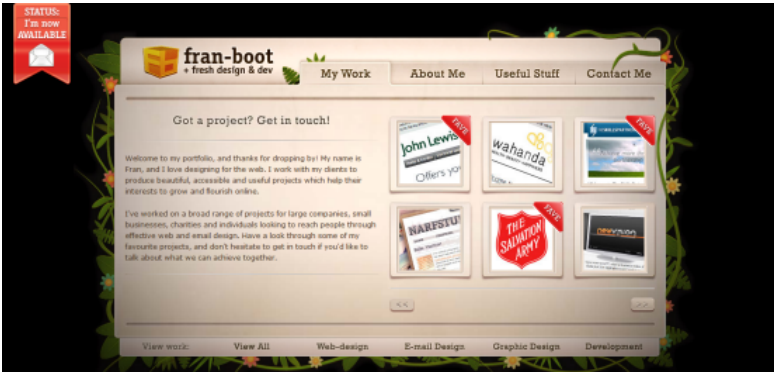


Pear Hosting

Playmation

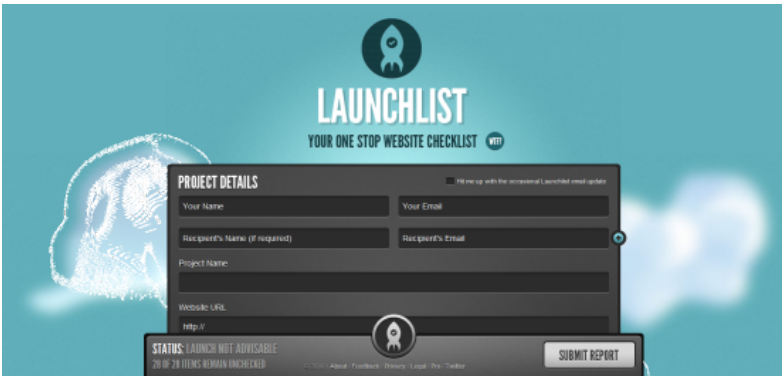
Enrichmint

Basil Gloo

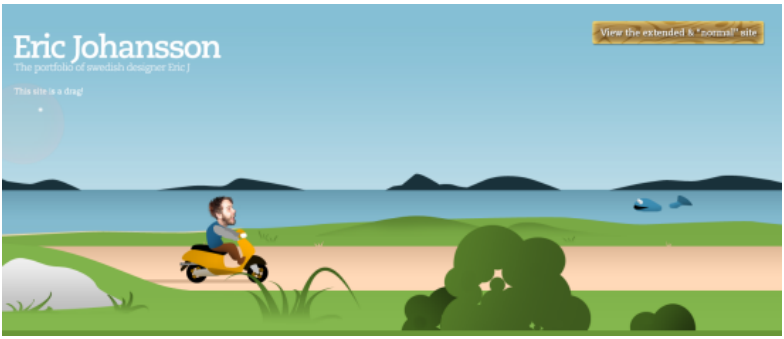


Fran-boot

Milk 'n Honey



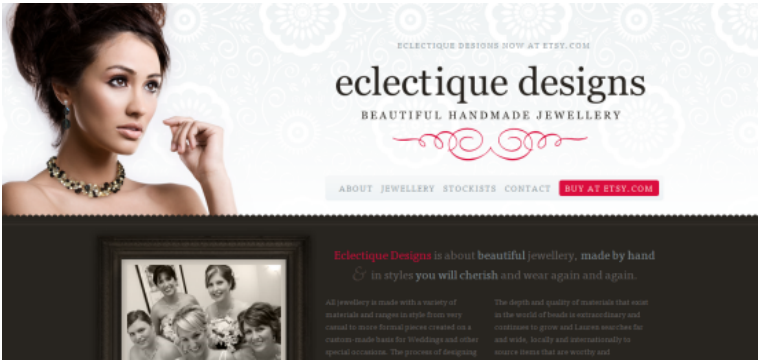
Launch List



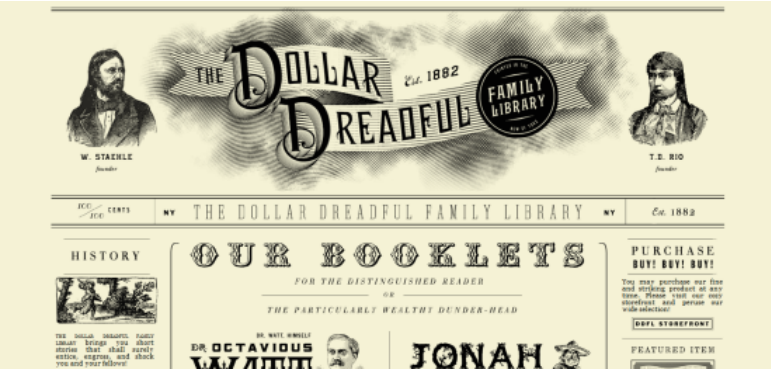
Eric Johansson

Hello Kavita

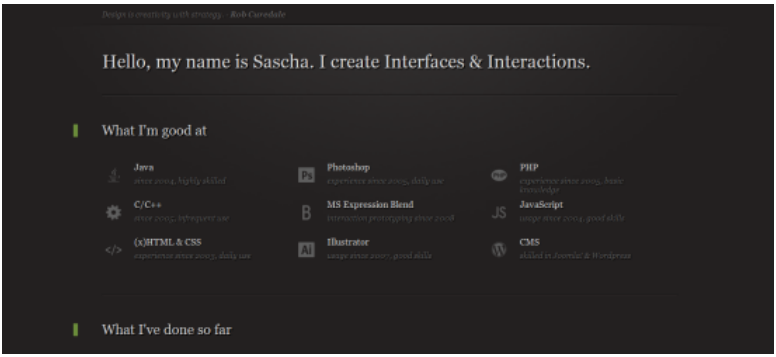
Made By Sofa



Eclectique Designs



The Dollar Dreadful



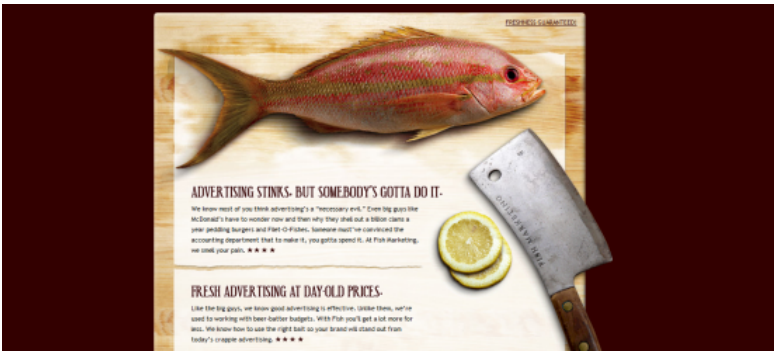
Elementic Interactions

Richard Turnbull Design

James Lai Creative

Tim Van Damme

10 20 Concepts



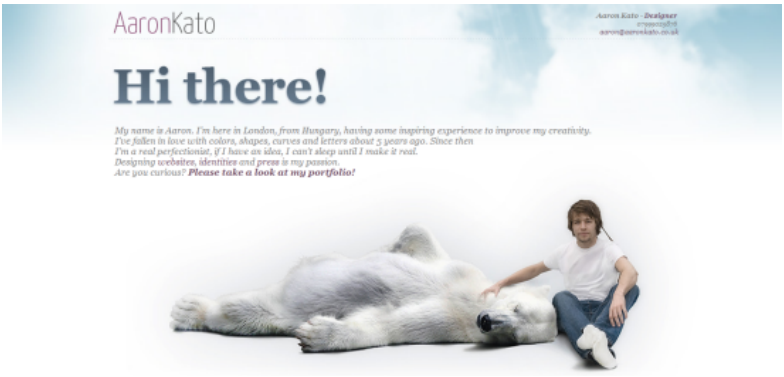
Fish Marketing

Banjax

Silverback

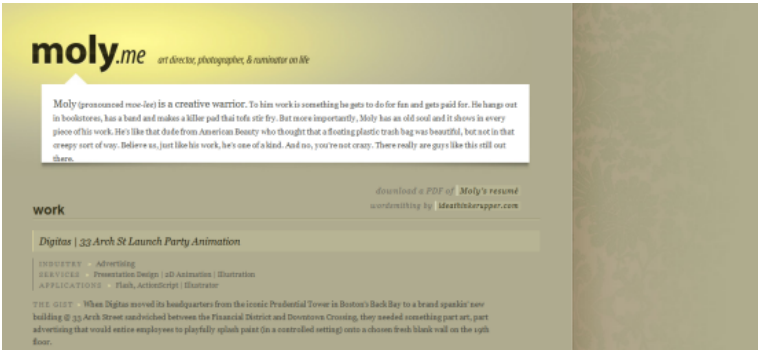
Zee the Designer

Tyler Termini



Aaron Kato

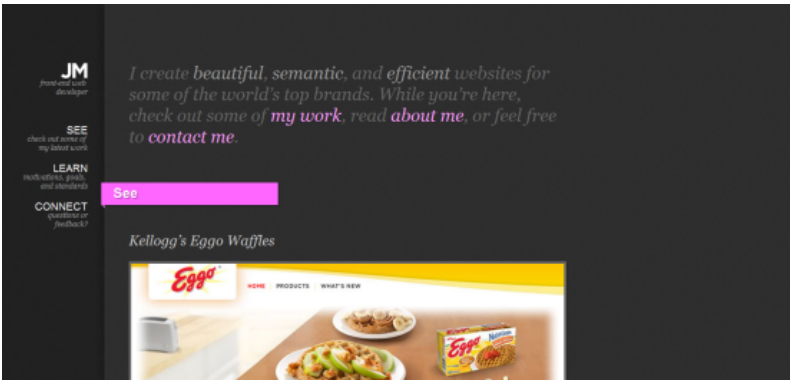
Joni Korpi



Molly Yim

Lataka

Jason Reed



Josh Minnich

Is Single Page Websites for You?

The great thing about the web is that it's constantly evolving, and by principle, the way we build our designs will mould itself to these shifts in our audience's tastes and needs.

The need for lightweight, compact, self-contained websites and web applications due to the Mobile Web will only increase the deployment of single page websites. All single page layouts can be made to look totally unique. While not to everyone's taste, they are well worth a consideration in your web design projects.

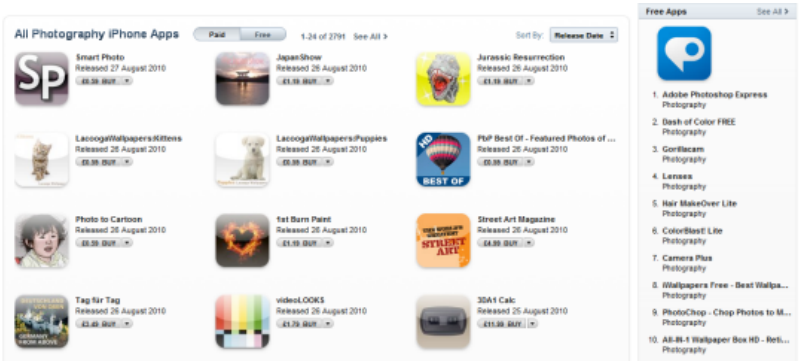
Sources:

- http://www.csszengarden.com/001/
- https://authenticstyle.co.uk/
- https://www.w3.org/TR/selectors-3/#target-pseudo
- https://www.w3.org/TR/REC-html40/intro/intro.html#h-2.1.2
- http://jquery.com/
- https://github.com/flesler/jquery.scrollTo
- http://camera.plus/
- http://basilgloo.com/
- http://www.narfstuff.co.uk/portfolio/
- http://humaan.com/checklist/
- http://madebysofa.com/
- http://www.dollardreadful.com/
- https://1020concepts.nl/
- https://silverbackapp.com/
- http://www.jonikorpi.com/

Improve Site Usability by Studying Museums

Using a website should be easy. It should be intuitive. We should know what button or link to click to get to where we need to be.

But sometimes websites can be insanely confusing. Just look at the Apple app store, for example. When seeking out apps to install on my iPhone — which as an open source advocate and proponent, I feel incredibly guilty with, by the way — I find myself endlessly frustrated at the general lack of good navigation in iTunes, making the process of discovering the apps I want more difficult than necessary.

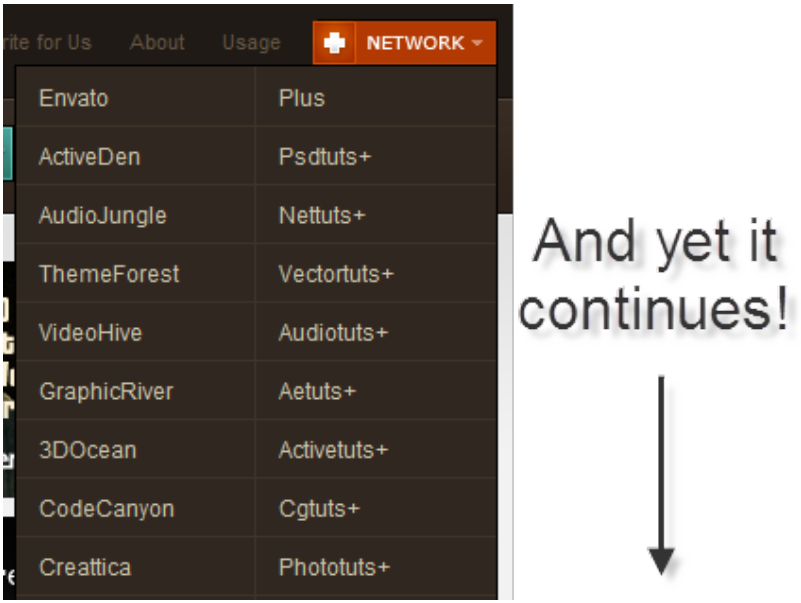


Trying to browse all of the apps on iTunes is next to impossible!

This article aims to underscore lessons we can learn from museums and art galleries in relation to website usability.

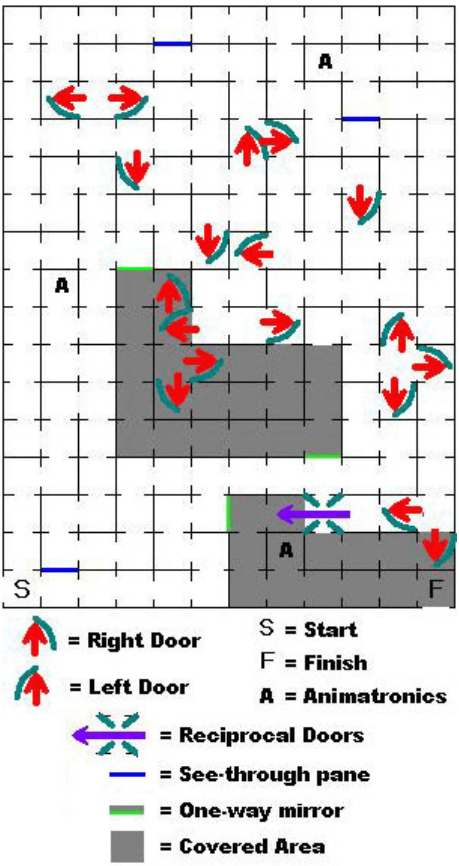
A-Maze-ingly Unfriendly

Many websites are a labyrinth (or maze) of endless tunnels and pathways that have no clear direction — and getting lost or confused is the resulting outcome. Simply put, things really need to change if your website visitor feels like your content organization is like traversing a maze.



Navigation menus for visitors can turn from helpful aids to monstrous mazes; try to keep dropdown menus short.

Perhaps it seems a little unfair or extreme to compare most site navigation designs to a maze — but the problem exists in many sites and is something we need to tackle. It's quite normal to see site navigation that is confusing, leading to dead ends or paths that lead you away from the exit.



Don't make your navigation complex like this. Navigation isn't a puzzle game!

What We Can Learn from Museums and Art Galleries

I remember going to the Natural History Museum (in London) on a school trip. While the endless exhibits were fascinating — what teenager wouldn't get a kick out of weapons, naked statues and dead people on display — the one thing that impressed me was how the museum, for such a huge building, was incredibly easy to navigate.

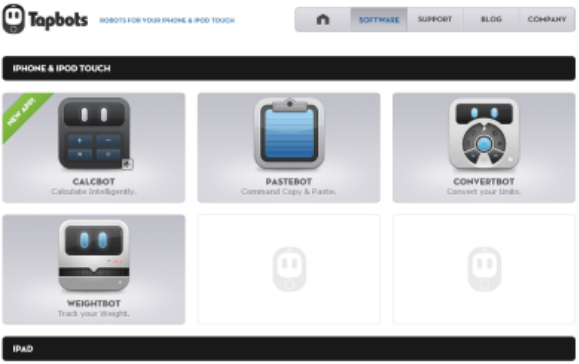
There's nothing like having enough space to appreciate art and websites alike.

Whilst the brick and mortar foundations of a museum or art gallery may not seem, on the surface, like ideal candidates for usability, navigation, and layout design inspiration, indulge me as I try to make the connection.

The Connection

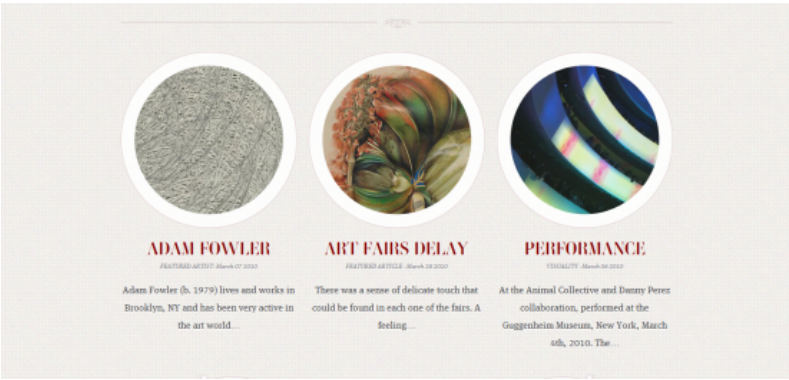
What is it about museums and galleries that encourage people to spend time exploring the many rooms and walkways in these humongous buildings? It's nothing more or less simple than the exhibits.

In terms of the web: our content, site features and service offerings are the core exhibits that we offer.



Giving each of your products its own "room" can aid you in giving them enough distinction.

Content appears in many forms such as text, images, diagrams, videos and audio. While most art galleries focus on showcasing images and museums on historic objects, the idea that we can feature, display, organize and lay out experiences to be browsed at the user's leisure is a central point of design theory and the content management process.

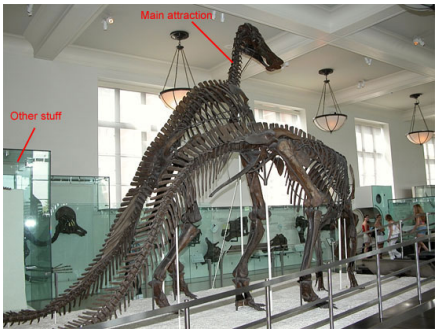


Showcasing images effectively will result in a more streamlined experience.

Components of Museums and Art Galleries

So what is the secret behind the success of a museum's navigability?

Featured Exhibits

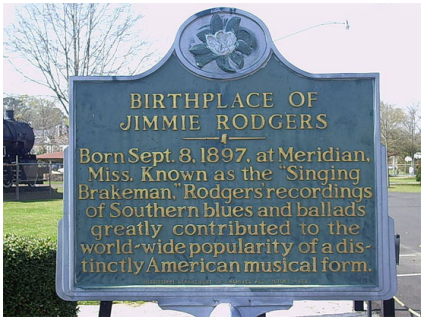


Source: Wikipedia

Museums highlight special exhibitions and actively promote them by ensuring that display items are in an accessible and easy-to-find region of the building.

They might have special posters outside the building, they may even advertise these exhibits off-site (such as in the local newspapers or in their website). As these are of special interest, they are publicized quite heavily.

Signs and Directions



Jimmie Rodgers Museum Sign. Source: Wikipedia

By having signs around the establishment, visitors can accomplish the most fundamental tasks — from locating that exhibit you wanted to see, to finding the closest public restroom.

Brochures and Reference Booklets

Having signs in a gallery or museum is great if you just want to provide simple directions, but many institutions have booklets or leaflets that are more content-heavy for individuals requiring more information.

Maps

Estonian Open Air Museum map. Source: Wikipedia.

A map is useful for providing directions visually towards certain locations. Some museums and galleries offer paper maps and other aids (sometimes even apps for mobile devices) to guide new visitors around.

Human Assistance

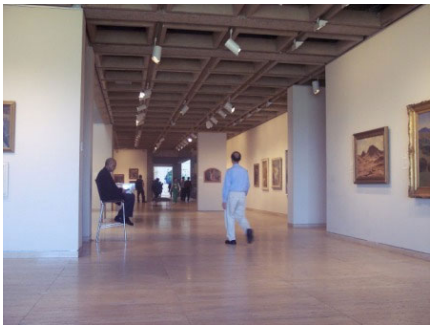
Every now and again, visitors will need more than what's offered by default; when they have inquiries that require a tailored response. It is at this stage where human assistance comes in. Most places you visit have members of staff roaming around to help patrons or stationary informational booths in case someone needs help through human interaction.

Interactive Exhibits

Science Centre Port Blair Biotechnology Gallery. Source: Wikipedia

An increasing amount of museums offer interactive displays to make the things being presented to the visitor engaging and fun.

Space and Clarity



Source: Wikipedia

There is nothing more annoying than an overcrowded location. Whether it's too many items put too closely together or too many visitors — ensuring that every item gets the space it warrants so that it is distinct from other pieces is an effective way of laying out museums and art galleries.

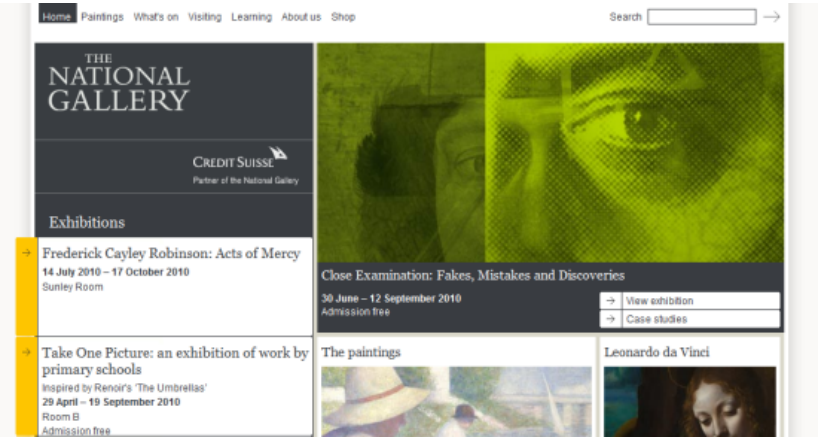
Souvenirs

Souvenirs from USCG Museum Northwest, Seattle, Washington.
Source: Wikipedia

Most museums have some kind of store that sells memorabilia and trinkets related to the museum. These items create a tangibility to the visit, and may even prompt future visits.

Producing Digital Equivalents of Museum Components

Digital equivalents for these museum/art gallery components may feel like an imaginative stretch, but let's give it a shot.



Website Equivalent of Featured Exhibits

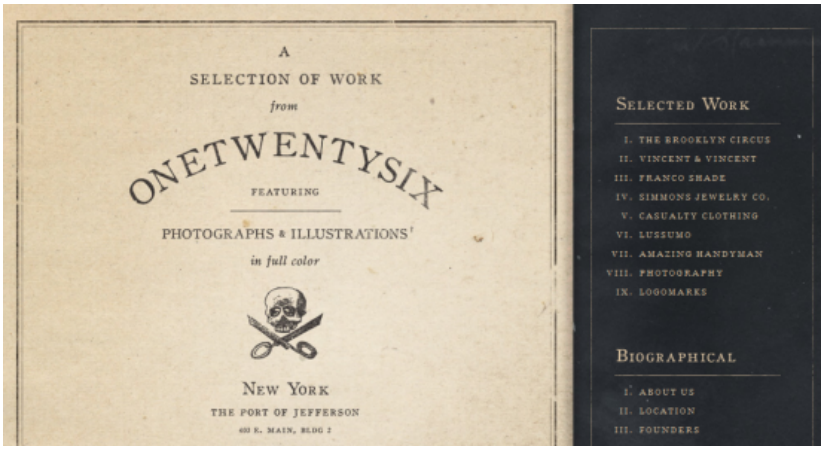
Featured exhibits are the first item on the list, and it won't surprise you that feature sections can be effective in design because you see it quite often. From content sliders and module tabs, to special limited-time offers and displaying fresh site content prominently, these various site components can draw attention to your featured items.

The above image shows precisely how you can feature prominent content.

Remember how I said museums sometimes even promote featured exhibits outside of their physical location? The digital equivalent of this is advertising your "exhibits" in other sites and services, such as Twitter, Facebook, and AllTop, all of which can help enhance findability.

Website Equivalent of Signs and Directions

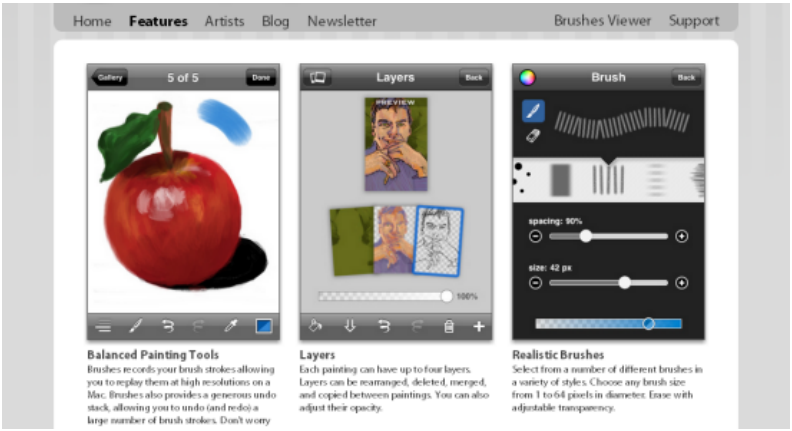
Signposts and directions have their own digital equivalents too. Many websites have help documentation and navigation aids like breadcrumbs to help you locate the things you need. Great designs will also use the art of distinction and call-to-action buttons to draw attention and guide the user's eyes towards what they might be looking for.



Chapters indicate how far you've travelled through a large document.

Website Equivalent of Brochures and Reference Booklets

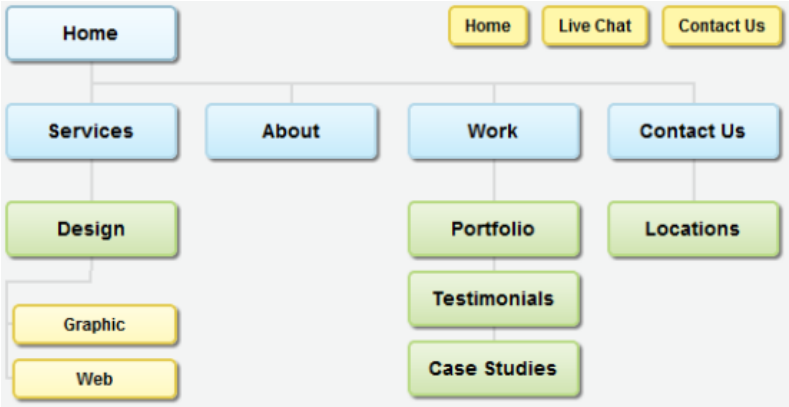
The equivalent of reference booklets and brochures on the web are help doc pages, FAQ pages, video tours, and so forth.



Offering guided tours or quick start tutorials can help engage visitor understanding.

Website Equivalent of Maps

The most obvious manifestations of physical maps on a website are sitemaps and, to a large extent, your primary navigation menu. Creating a visually illustrative diagram of your site's layout could also be components that help your users navigate around your site.

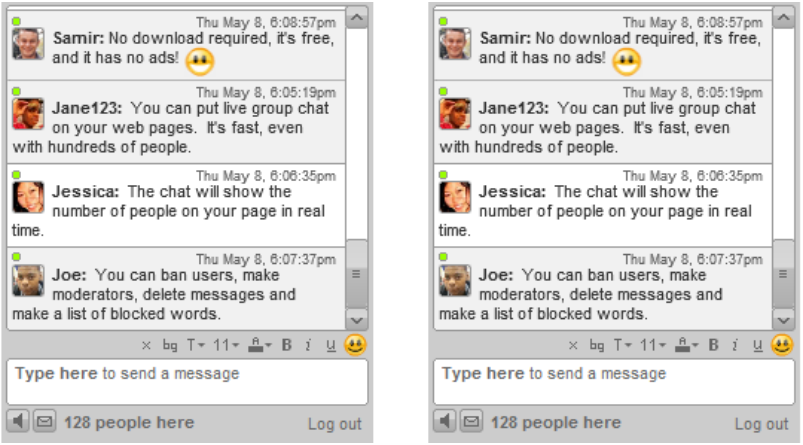


Providing a site map showcases the amazing array of content your site holds.

Website Equivalent of Human Assistance

A lot of businesses may say that it isn't cost-effective to use a member of their team for live support, but they could have fixed hours for when live chat support is available to site visitors.

Otherwise, asynchronous communication methods such as a contact web form or a help desk support system can also be the website equivalent of informational booths that are driven by people.



Live chat software can be useful as a form of human assistance to site visitors.

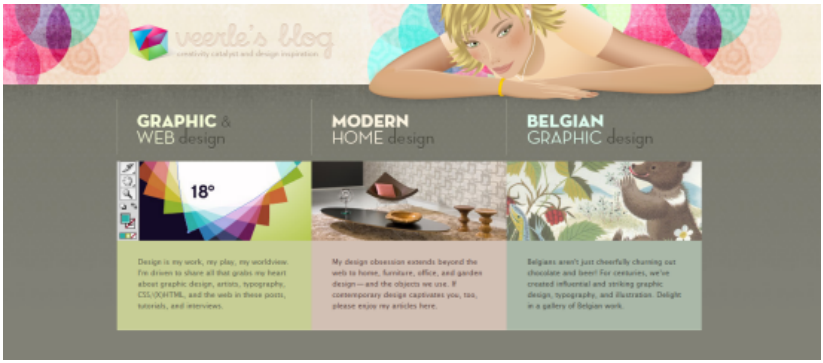
Website Equivalent of Interactive Exhibits

Site interactivity can, just like their offline-counterparts, engage and provide more value to the experience of the user. You can have game mechanisms to enrich the user experience, live chat widgets so that visitors can interact with one another, and other simple strategies for creating a richer and more engaging experience.

Giving your users something to interact with provides a more engaging experience.

Website Equivalent of Space and Clarity

Space and clarity on websites require a solid understanding of design principles. Thinking about Gestalt principles (and more specifically, the concept of proximity), visual hierarchy, the use of negative space, reductionism, all the way down to the basics of clear and legible web typography can lead to effective use of space and clarity that, in turn, can lead to a better and more user-friendly experience.



Breaking information down into manageable segments help progressive navigation.

Website Equivalent of Souvenirs

People like and recommend free stuff to others, and if these "freebies" are branded or help promote your site, then in principle, the site's visibility and illustriousness increases.

Souvenirs on websites could be custom browser extensions (for example, Mashable, a social media blog, has a Google Chrome extension), digital downloads (Six Revisions has freebies that are useful to its audience), or anything that allows your site visitors to retain something from their visit and reminds them to come back.

Giving some branded freebies away will help trigger the user's memory of your website.

Structure from the Chaos

What all of this really boils down to is that it's important to pay attention to the site visitor's experience. All these concepts and components really lead to a single idea: A usable website is effective at providing what the visitor needs in a pleasant and near-effortless manner.

Why not visit a museum or art gallery yourself and see what other things you can apply to your profession? Maybe look at how they lay out their exhibits or observe how visitors interact with the various components provided to them.

The journey of an experience should be less about getting from points A to B (even with people in a hurry), and more about the things in between that make the experience pleasantly memorable.

Sources:

- https://tapbots.com/
- http://advuli.com/
- https://www.smashingmagazine.com/2009/06/module-tabs-in-web-design-best-practices-and-solutions/
- https://www.smashingmagazine.com/2009/03/breadcrumbs-in-web-design-examples-and-best-practices/
- http://veerle.duoh.com/
- https://mashable.com/2010/06/03/mashable-google-chrome-extension/#GlhqYalj1Pq1

Human Behavior Theories That Can be Applied to Web Design

Humans are logical creatures, and as surprising as this might be, when we visit a website our minds make a series of decisions that affect the actions we take. The ability to reason enables us to form judgments, reach conclusions and make decisions. If, on the web, we weren't able to think on the spot and then take action, we would trap ourselves in crippling situations of mindless clicking.

Behavioral psychology is an advancing field, and we web ninjas need to understand something about psychology in order to make usable websites. If we understand human needs and emotions — how we interpret what we see and how we choose to act — then we will better understand our site users. We'll be able to choose and create meaningful layouts, typography and colors.

This article is no substitute for a degree in psychology (so don't give yourself an honorary Ph.D. after reading this). Also, the items mentioned here don't account for every circumstance, because no two people are the same. Yet by understanding the theories outlined below (there are no hard facts in psychology, just theories), you can better understand how your design work will be perceived and used.

Empowerment and Maslow's Hierarchy

Giving your visitors a sense of growth and increasing their self-esteem is essential. One of the main problems faced by Facebook, YouTube, Twitter and many other social networks is how to get visitors to participate and feel that they are welcome and safe.

The "silent visitor" (or lurker) has existed for many years and makes up the majority of people who visit the average website. This type of visitor usually isn't socially inclined; a number of your audience members will not go any further than to visit and read, which isn't very encouraging if you want to establish a community.

StumbleUpon encourages people to make friends and invest themselves socially.

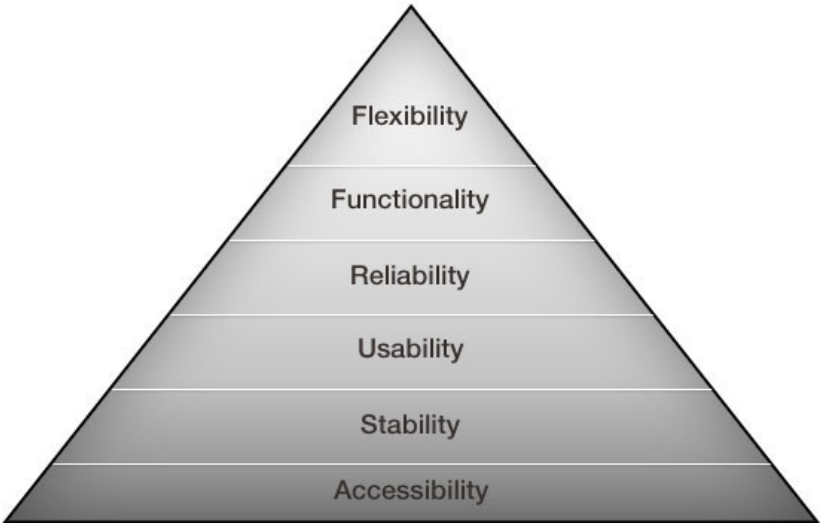
For a design to work, it must meet the needs of visitors–although what's important to visitors is up for debate and could drastically change from project to project. Some needs cannot be met before other needs have been addressed.

The most famous theory about what humans require in order to reach their "pinnacle" — the point where they decide to participate — was posited by the humanist psychologist Abraham Maslow in the form of a "hierarchy of needs."

Maslow defined levels of importance that reflect how and what humans prioritize, as well as what they require in order to appreciate their surroundings and achieve personal growth (or "self-actualization").

Based on his concept, I've created one related to web development: The hierarchy of website user needs. They're listed below in order of importance.

1. Accessibility: The website can be found and used by all people.
2. Stability: The website is consistent and trustworthy.
3. Usability: The website is user-friendly.
4. Reliability: The website is consistently available, without downtime.
5. Functionality: The website offers content, tools and services users value.
6. Flexibility: The website adapts to needs and wants of users.



The hierarchy of website user needs.

Fundamental expectations of visitors include being able to find the website, to browse around it effectively, to return to it easily and to use the available services and content.

Adapt to the needs of your users whenever possible. If the hierarchy of website user needs fails, it will severely hinder the user experience.

Examine, then, the factors that affect experiences, and learn how to reduce the risk that primary needs will go unserved.

Attractiveness Bias

Web design — and specifically the visual aesthetics of a web design — is not a case of "fatal attraction," nor is it the be-all and end-all of a website's success (as shown by big websites that have mediocre visual aesthetics). But beauty is attractive to humans. Psychologists surmise that humans have a cognitive bias to attractive people and things.

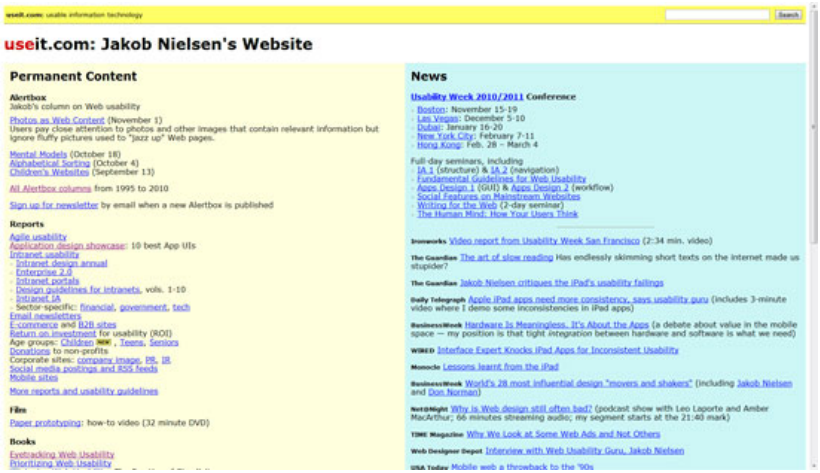
Content is, of course, the most important part of the website; it's the personality that lasts after the design has worn off.

The attractiveness bias theory simply states that a good-looking design will draw more attention than a poor design. Once viewers get over their first impression (which is always important), they'll be content and comfortable, and that will increase the likelihood that they'll visit again.

The principle behind this theory is that humans are naturally attracted to beauty. And yet, content (the personality) is what matters — it's the foundation of your long-term relationship with visitors — but the attractiveness of the design (the exterior) is what will get you noticed.

Beauty becomes less important when people learn what's underneath, and with websites, what should be underneath is quality content.

Status, though, is an exception to this rule: If you're famous, you can make the website as ugly as you like and visitors will still flock in droves. This is called brand recognition. Look at celebrated usability expert Jakob Nielsen's website, for example.



Jakob Nielsen's website.

Some believe that design does not affect the overall impression made by a website, but attractiveness bias theory indicates that, while the content of a website is important to regular visitors, the "wow" factor is responsible for creating initial appeal.

Make your website look professional and beautiful, but make sure that it doesn't overrule the hierarchy of website user needs. If you accomplish this, then your website should naturally attract even those visitors who are quick to judge quality by appearance (which happens a lot).

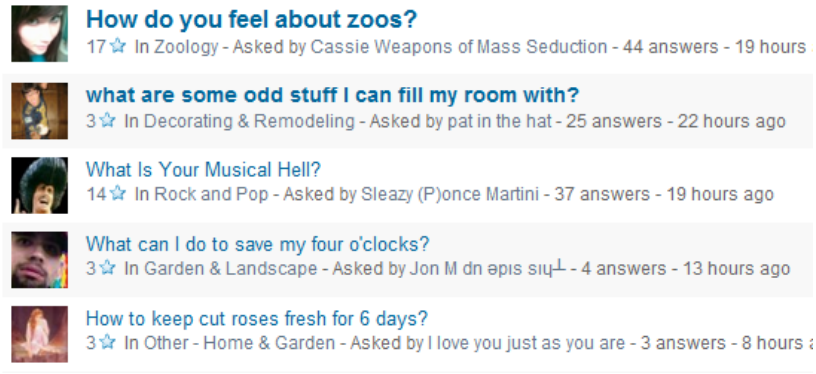
Once you've convinced people that the website is visually solid, then the content's integrity will shine through.

Serial Positioning Effect

The placement of information affects how well it's remembered. Lists, charts and tables are useful because they break large clusters of information into manageable, comprehensible chunks. Organized information stands out from blocks of heavy text on a website.

In lists, the first and last pieces of information will be most easily recalled. The beginning and end are, in our minds, naturally

significant, and therefore the text in those locations will appear to be more important than other text on the page.



Yahoo Answers uses small content blocks to facilitate memorization.

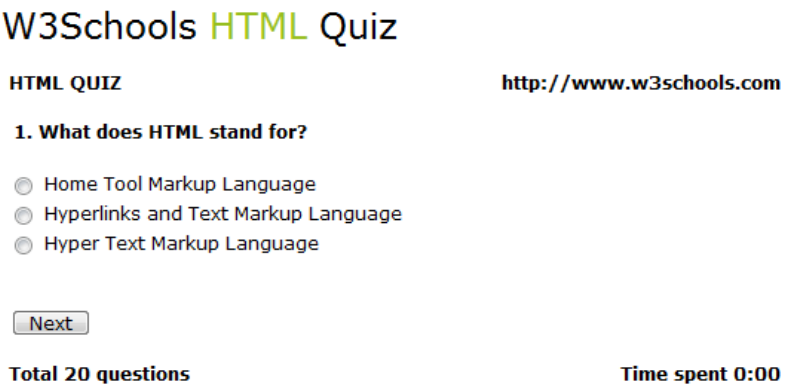
The serial positioning effect, proposed by Hermann Ebbinghaus, proposes that the ability of people to remember something accurately varies with the item's position in a list. In web design, this most closely relates to visual hierarchy.

When people browse the web, looking through pages and pages of information, they typically commit less than 10% to memory (and only 1% if they are looking for a key phrase or definition). Find helpful ways to present your information, and you might make your website and its content memorable.

Depth of Processing

Depth of processing (or levels-of-processing) is a term that refers to the level to which information has to be processed in order to be committed to memory.

Engage your users, and ensure that they read important details carefully by asking them about it afterwards. Remember those exams in school that tested whether you had retained the course material? Tests are feared by many, but there is science — go figure — behind the methodology.



At the end of a W3Schools tutorial, a basic quiz is offered to reinforce knowledge.

Users do not want to be hassled with questions or have their browsing interfered with (which is perfectly understandable) so put simple mechanisms in place (such as an "I accept" button in a license agreement). Find a balance between subtle awareness and intrusiveness.

Use checkpoints and reviewing mechanisms to ensure your visitors are learning what they need to know. For example, asking questions at the end of an e-store checkout process and repeating important warnings can help users process the information you are providing.

If your website provides educational information, a quiz or method of testing knowledge could serve as an interactive memory aid. Even asking a pointed question at the end of a blog post (such as Do you have any tips to share about the subject?) encourages users to critically think about what they just read.

Finding the balance between helping someone remember key pieces of information and annoying them or getting in their way can be tricky. Use your best judgment to decide at which points images, interactive elements or other processing aids would be useful. From quizzes to images to humor — so many options are at your fingertips.

Fitts's Law

Fitts's Law models human interaction with computers. It states that the time and effort required to reach a target depends on the distance and size of the target. In web design, it is most relevant when designing the degree of usability of user interfaces.

To a great extent, websites determine how easy it is for users to achieve their goals. Certain layout structures can become an obstruction. A classic example of an obstruction is a clickable element that is so small that it requires precise movements and targeting to click (such as a small hyperlink text viewed on a mobile device).

The issue of objects with small clickable surface areas is significant especially in mobile devices.

Fitts's Law states that the smaller the clickable area, the longer it will take to activate. The longer something takes to activate and get to, the lower its usability is.

Make the surface area of interactive items on your website sufficient in size. Take advantage of gesture movements on touchscreen devices, and ensure that pages can be zoomed and text enlarged. Removing barriers to access, especially for users with impaired motor skills, is important. Anything that detracts from the experience could cause visitors to run away.

Cognitive Load

The amount of time it takes to accomplish a task increases with the amount of tasks given to a person. Cognitive load is a term that describes how our learning performance is reduced when we have many things we have to do at once.

To put it simply: the more tasks we give users, the slower they are able to finish a task and the more confused they will become. Usability consultant Steve Krug, in his book Don't Make Me Think!, illustrates this theory by using the idea of "goodwill" and gain and loss.

Keep things quick and easy to follow and your visitors will get what they want faster. Smartly laid-out designs are among the easiest to use and receive the most positive feedback. Using your website should be effortless.

The Zombie Browsing Effect

We're not talking about characters from a George Romero movie. Zombie visitors target what they seek and don't get distracted by other items on display at a store or on a website.

As people get used to a website, the zombie effect becomes more likely; as in a supermarket, once you know what goods you want and their location, you don't spend much time looking elsewhere. This explains why stores sometimes change their layouts; it exposes existing customers to new goods.

You don't want to be a one-hit wonder; you want to encourage users to explore your wares so that you can increase sales. Zombies avoid exploration by ignoring their surroundings and merely following their primal instincts. That's what I call brainless buying habits!

Overcoming the zombie problem can be quite a challenge. You don't need to whack them on the head like in the movies (although that might help); rather try getting their attention and engaging them with distinctive design. For example, thoughtfully-placed and distinctive elements could garner the user's attention.

While searching a website like Amazon, our field of vision is limited to specifics.

If everything on the page is somehow important, decide which sections you most want to promote–streamlining content will streamline traffic. Too many advertisements and too much content and clutter will cause the zombies to raise their defenses and ignore all messages; their brains will shut down completely.

Single points of interest attract attention, and if the item is relevant to visitors, they might spend time exploring it. Allowing people to customize your website according to how they prefer to browse (as iGoogle and the BBC do) can discourage the zombie effect, because people will be attracted to the relevance of the content.

Conditioning Models

In psychology, the term conditioning refers to the process of instilling predictable behavior. Classical conditioning modifies involuntary reactions, whereas operant conditioning modifies choices (or the likelihood that the subject will make a certain choice). Both are relevant.

Classical conditioning becomes relevant to web design when we think about visitors closing pop-up windows or turning the volume down on a website's background music. Some reactions are natural, but many of them are conditioned by experience. These are called learned behaviors.

Behaviors are learned when people experience the same thing or similar things repeatedly. In web design, conventions, trends and patterns are responsible for learned behaviors and help users associate events with likely outcomes. Thus, a range of possibilities emerges — emotions can be leveraged to improve conversions, for example.

The learned behavior to a "Page not found" error message is to click the "Back" button.

Instead of trying to override natural behavior, streamline your website by adapting to them. A natural or learned response requires less cognitive power than a response to a new experience — say, when the user faces a change in the layout. In the latter case, the learning curve is increased significantly, primarily because training oneself to suppress a natural response is a lengthy process and can take years — and it might not be worth it. Trying to classically condition Internet users is like swimming against a fierce tidal wave.

Now let's consider operant conditioning. Operant behavior is the behavior of an agent — that is, someone who chooses rather than reacts. The agent learns by trial and error, and patterns of behavior are created by learning to anticipate outcomes.

One method you can use to affect operant behavior is called priming. Here's simple example: if you invite website visitors to "Contact us if you need help," a visitor will feel the need to contact you once they have exhausted their own means of accomplishing their task. If you hadn't suggested it, they'd be less likely to think of it as an option.

By showing off important features, Get Glue primes visitors to use its services.

Operant conditioning employs methods of positive and negative reinforcement (reward and punishment). What does this have to do with web design? Ensure that actions that benefit your website, like visitor comments, are rewarded and that negative ones, like spamming, are punished. Loyalty is in short supply and can fizzle quickly if participants feel they are being treated unfairly, so be fair and make sure that users know you're trying.

When it comes to the underlying features on a website, conditioning is a lot subtler and requires less explanation. Websites that have calls to action — such as buttons that encourage users to download a piece of software or a friendly message that asks people for information — will prime users to perform the relevant action. If the results are beneficial, then users are likely to repeat that behavior.

Conditioning takes time, and people will comply with fair rules if they feel they have reason to. By using operant conditioning you can encourage visitors to change their perception of your website and even go along with the behavior of the majority (although some people might defy conditioning for other reasons).

Coca-Cola uses positive reinforcement: a reward system is in place for customers.

A person's behavior has so many contributing factors that trying to understand it can be a real challenge. Being aware that behavior can be altered (with the right incentive or by addressing certain issues) might be key to keeping your community alive.

Ultimately, you can't force people to do your bidding, but you can give them reasons to visit and instill in them the desire to return. The encouragement of a community is a powerful psychological force.

Conclusion

In almost everything we undertake as web designers, psychology plays an important role. Whether the influence is subliminal or explicit, and whether you leverage common trends and conditions, understanding the experience of the audience and their perception of your web design gives you a great advantage.

Sources:

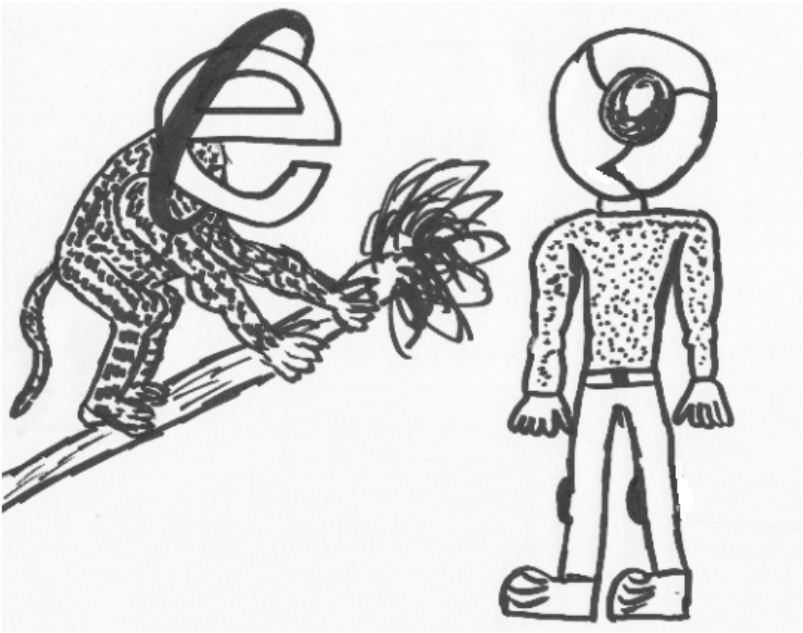
- http://www.stumbleupon.com/home/
- https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs
- https://en.wikipedia.org/wiki/Physical_attractiveness_stereotype
- http://www3.psych.purdue.edu/~willia55/392F/Langlois.pdf
- https://en.wikipedia.org/wiki/Serial-position_effect
- https://en.wikipedia.org/wiki/Levels-of-processing_effect
- https://www.w3schools.com/quiztest/quiztest.asp?qtest=HTML
- https://en.wikipedia.org/wiki/Fitts%27s_law
- https://en.wikipedia.org/wiki/Cognitive_load
- https://us.coca-cola.com/mycokerewards/
- http://alistapart.com/article/visual-decision-making
- http://allpsych.com/psychology101/conditioning/
- http://changingminds.org/explanations/theories/cognitive_dissonance.htm
- http://particletree.com/features/visualizing-fittss-law/

Evolution of Websites: A Darwinian Tale

The web is constantly evolving. It doesn't take a rocket scientist to see how quickly new technologies are being adopted and how fragile design trends are. While the web is still an infant relative to other mediums such as print, TV and radio, and still has fair amount of growing up to do, it has already amassed a rich history. Let's take a look at how the medium has evolved throughout the years.

A Matter of Carbon Dating

Evolution is inevitable. As British philosopher Herbert Spencer put it — inspired by Charles Darwin's theory on natural selection — it's "survival of the fittest." If we examine any aspect of web design, we can see that trends and technologies being discarded, improved on, or superseded by something better is common. Evolve or die, pick one of the two options. And if we delve deeper, we can see three core elements that dictate this natural selection and evolution.



Certain web browsers tend to be more evolved than others!

Code

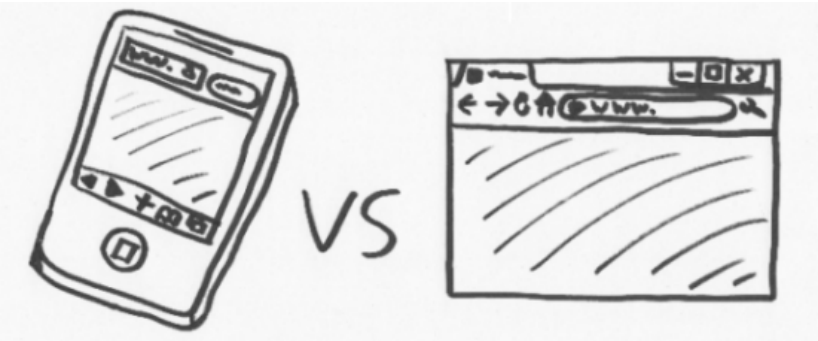
One of the core elements of the web is code. As web designers and web developers, the success of a particular language largely depends on how much value it brings to our work. I'm sure only a handful of you remember VRML with the fondness of the concept that we could soon be browsing the web using the same virtual reality as used in the movie Tron. Alas, virtual reality didn't take off.

The idea of virtual reality and 3D objects fascinated developers.

Web Browsing Devices

The second core element that dictates the web's progress are web-browsing devices. As technologies evolve, as browsers are improved, as browsing methods change, so do we as an industry.

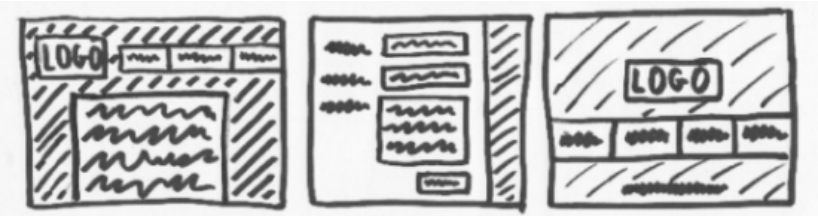
Things have come a long way from IE3 on an i486 computer with a 56k modem that made a screeching sound like a harpy as it connected to the "Net" through AOL. We can see the fossils of our past exist in computers that still have IE6 and Lynx, poking around our websites on occasion (much to our dismay).



Devices come in all shapes and sizes, from cell phones to browsers.

Trends and Conventions

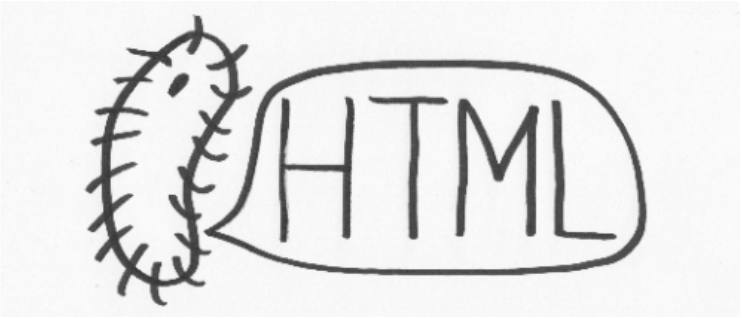
Finally, we have trends and conventions. No, I'm not talking about the latest fashion according to Lady Gaga, but of design trends, design patterns and conventions that have gained acceptance as the norm because they are proven to work well. Things that are known to enhance the user experience and conventions that aren't so annoying that you have to avert your eyes to avoid suffering a brain hemorrhage (that's why the neon pink against a yellow background color scheme never quite took off).



Trends and conventions come in a whole range of shapes and sizes.

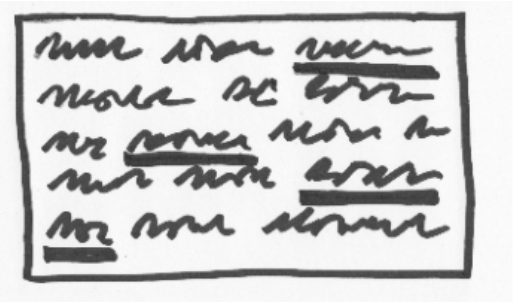
Single-Celled Organisms

When the web first started, it was much like a single-celled organism — strong, resilient yet simple. We didn't have the need (or ability to have) dynamic behavior, database-driven pages and sophisticated web layouts; the web was about sharing text (and some images) and nothing more. Things needed to progress further to gain mass adoption and appeal.



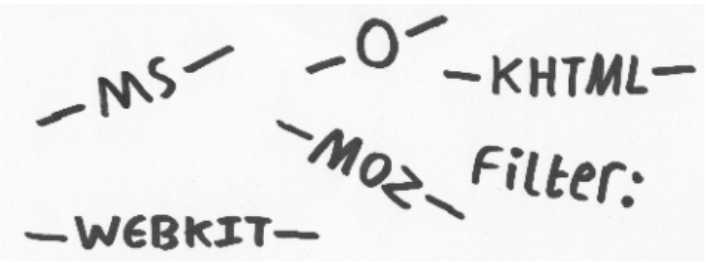
Back in the early days, things were much simpler. Just like microorganisms!

If you've ever seen the first pages created by Tim Berners-Lee, you would see that the original idea was simply to be able to connect and relate disparate text pages by way of hyperlinks.



The web started with text and hyperlinks, but now it's more complicated.

Post the success of the early internet, thoughts of actually doing something more visually intriguing with code (to make the text look extra awesome) started to emerge. While the HTML standard kept evolving to push the envelope further, web browsers began breeding their own webs of "unique" (read as proprietary) evolutions. And as with most children, it was inevitable that tantrums kicked off in the face of adversity.

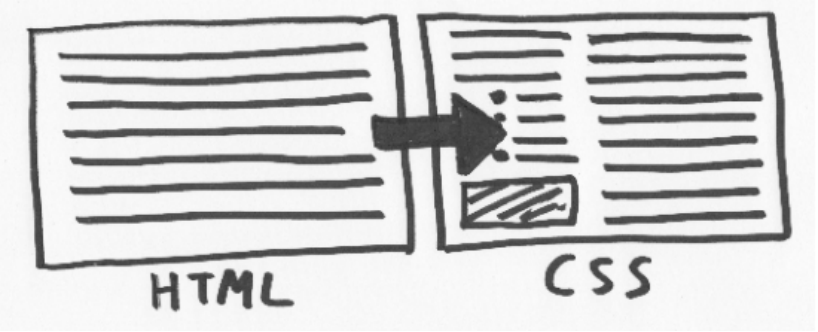


Vendor prefixes exist to this day, though they're much better organized.

Biodiversity in Design

Moving away from the lackluster ideals of just having text and images, the HTML specification grew to include all sorts of wonderful opportunities to begin laying out content. It started with table-based designs and simple multi-column web pages. Then we got CSS, which enabled us to style our content so that they looked nothing like

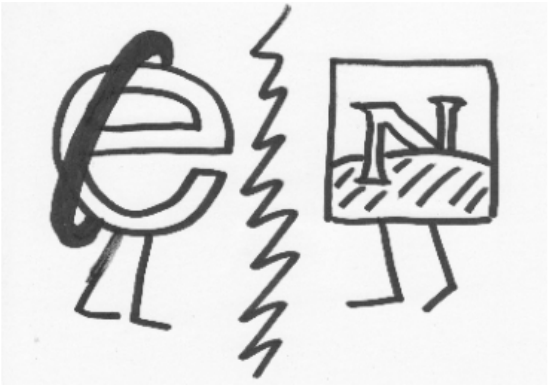
the typical white background, black text, and blue underlined hyperlinks cliché. And finally, we saw an evolution towards more complex organisms.



We went from an era of blocks of text to stuff that can be positioned with CSS.

As the Borg said, "resistance is futile," and the fertile soil that grew the early seeds of the web blossomed to increase our capabilities. Perl, PHP, applets, client-side scripting allowed us to do more than just display content, but also to give site visitors ("web surfers") a way to interact with our pages.

While all of this growing and adoption was taking place, the browser-makers of the time (IE and Netscape), in an attempt to out-do each other and gain supreme dominance over the market, decided to start making things up as they went along. They created proprietary specifications, languages and custom code.



Internet Explorer and Netscape went at each other like Alien versus Predator!

As people began to experiment with this thing called a "website," many enthusiastic developers saw themselves able to achieve some of the most wonderfully annoying techniques to exist in our craft. Often coded with little attention to detail, the amount of alert boxes asking for your name, popup windows, right-click-crippling events, JavaScript clocks, site counters, and other strange things appeared in the boatloads. Many did it for the novelty or marketing agenda, most never thought about the user.

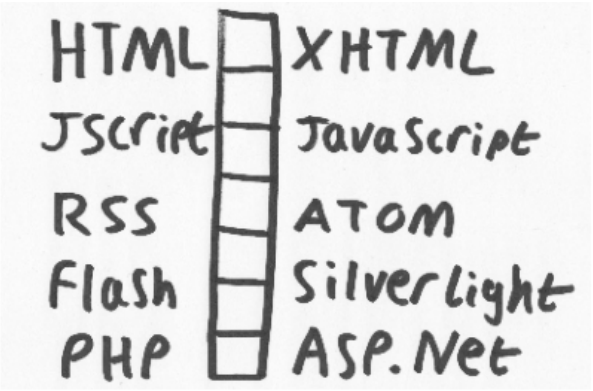
Anti-right-click scripts got thrown about in a futile attempt to curb digital piracy.

Luckily, something changed in the ethos toward the web.

With the birth of style and scripts, the adoption of standards became important. With browsers telling you to code in different ways, things began to conflict — and that wasn't fun for anyone.

Some browsers like Opera have remained standards-focused since their inception.

Part of the natural evolutionary process is based around the concept of natural selection: The organism that is better suited for the environment wins by outcompeting weaker organisms. The evolution of web browsers and competition certainly represented that. In code, multiple competing languages were produced for specific uses (i.e., RSS versus Atom) and this healthy competition led to the aftershock of innovation from developers and an imaginative array of new uses for the web from those pushing the boundaries.



Plenty of languages compete against each other, but this is healthy.

Survival of the Fittest

As we all know, the remnants of the browser wars still exist to this day — JScript, <marquee> and hasLayout immediately come to mind, as do some lost night's sleep — but this simply highlights the evolution that the web has gone under. We started with a very raw text-sharing and hyperlinking format and within 10 years ended up with a browser bidding war (spoiler alert: Microsoft's IE triumphed over Netscape).

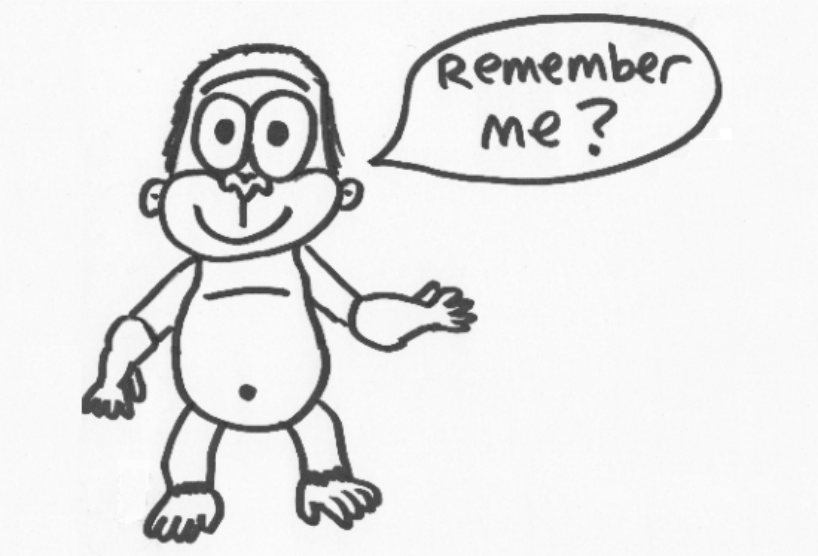
Microsoft won the browser wars by integrating IE within their OS (Windows).

While the idea of the web's old sites make many of us cringe and reflect upon our own first web portfolios with questions like "Why did

I do that?" and "Where did I find that animated GIF?," the change from what we needed the web to do to what we wanted it to do was bridging ever closer.

Sometimes, the extinction of a proprietary element has been a good thing!

The Wild West philosophy where "everything goes" failed toward the early 2000s as evidenced by the "dot-com bubble" bursting.



Bonzi Buddy was a classic example of people taking the web too far.

But this scenario was almost like a purging of sorts. Out with the old, in with the new. In its place, an age of enlightenment began, with the catalyst simply being that more people were using the internet. This surge of users outside of geeks meant a reform would be needed in the way we designed and developed our websites.

From Primate to People

After surviving the horrors of the 90s web and the early failure to monetize anything online, we breathed a huge sigh of relief that the Y2K bug hadn't wiped out mankind, and then we moved forward to a more thoughtful process in designing sites.

Gone were the days where the focus was on funneling as much junk to the user as possible. The idea of designing for the user's experience became interesting. User-centered design. Usability. Accessibility. These were the words being thrown around.

People thought computers and the web would collapse under the Y2K bug!

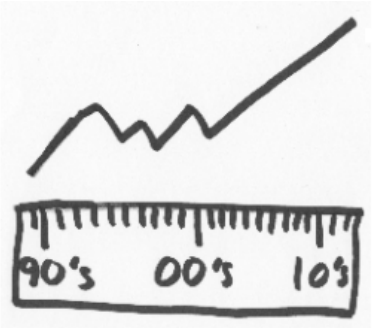
The old, clumsier days of the web in which design decisions were based around which scrolling statusbar script to use or which dancing Homer Simpson GIF would make you feel better were gone, and a focus toward professional user-centered design, data-driven design decisions, accessibility and usability gained importance. This was partly due to the evolution in our collective mindset as to what made good business sense.

Content and navigational structures became less erratic and more organized.

Looking to the Future

Examining things from a Darwinian perspective displays the web's change from its various stages of life. The early days where things were much simpler (and less stressful), the later years when many of us cried ourselves to sleep at the thought of making a design for Netscape and IE, the post browser war depression where design became focused on experimentation, and now with user-centered design — it's been a heck of a ride!

No one knows how this evolutionary story concludes. While the web already has an illustrious history, it's safe to say that we've only just begun.



Usage of the web has increased dramatically, as has the number of websites.

Looking back at the Internet's past, I find it interesting that we can "carbon date" our sites based on the techniques and technologies they use, and even scarier is that much of the web today remains fossilized within the bedrock of our servers.

While the evolution of the web is going on as we speak, and it's fascinating to see how trends, code and devices have changed, looking to the future of the web leaves us with limitless possibilities.

Sources:

- https://en.wikipedia.org/wiki/VRML
- https://en.wikipedia.org/wiki/Lynx_(web_browser)
- http://info.cern.ch/NextBrowser.html
- https://en.wikipedia.org/wiki/JScript
- https://msdn.microsoft.com/en-us/ie/ms530764%28v=vs.94%29?f=255&MSPPError=-2147217396
- https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/hh772379(v=vs.85)
- https://en.wikipedia.org/wiki/BonziBuddy
- https://www.google.com/publicdata/explore?ds=wb-wdi&met=it_net_user_p2&idim=country:USA&dl=en&hl=en&q=internet+usage#met=it_net_user_p2&idim=country:USA&tdim=true

Privacy and the User Experience

The privacy issue is an often-neglected aspect of designing user experiences. All too often in today's information-driven society, we who work on the web sacrifice privacy and submit our users to violation or intrusion. In this article, we'll discuss certain privacy-related concerns — in particular, how asking for too much information can degrade the overall user experience.

Our Thirst for Information

Why is privacy such a hot topic? Look at social networks such as Facebook (whose privacy settings are notoriously complex and ambiguous): the amount of user data that is either being made available publicly, sold without the user's knowledge or is visible because of a security breach is increasing. We as site owners and site builders are responsible for the transactions and activities that occur on our sites. We're the "guardians" of our users, and respecting their privacy is important.

Many people use PayPal to ensure that any breach of their website does not compromise their users' data.

The predominant concern about privacy is that websites often ask for more information than they need. How many times have you been

forced to sign up for an account just to access certain information? How many times have you been asked for personal details when the transactions don't require it? Websites of all scales and sizes are guilty of this, and it's time to address it.

Twitter doesn't make its users submit a ton of information. Excellent work, guys!

In addition to the concerns about the amount of information being harvested by websites, there are concerns about storage and how websites deal with information once they get it. A user's experience of a business and its services will only be as pleasant as the business is trustworthy. Treat visitors with respect and remove barriers to access (such as multiple data requests and spam), and you'll improve usability — and empower your audience in the process.

The Value of Knowledge

We, as users of websites, typically "sell" our personal information to whoever asks for it, whenever they ask. What's your shipping address? We'll also grab your IP address while we're at it (We'll do it secretly after you submit this web form). What's your date of birth? How much do you make a year?

One could argue that we, as a society, are devaluing identity. Knowledge is power, and anyone who knows details about someone

else — details from which they could benefit or profit from — has a leg up on the competition.

We certainly shouldn't manipulate our users or cash in on their data without their explicit consent and knowledge (e.g. Check this box if you're OK with us selling your data to anyone; we'll make $7.99 from selling your data). Quite the contrary: visitors will value our website if we ensure that their information is secure. Trustworthiness is rare and, for that reason, a valuable asset.

Tracking visitors' habits is a debatable practice, but it can help us enhance the experience.

While the data that we harvest from users allows us to target them much more purposefully and give them a better user experience, we can still reap long-term value despite restricting ourselves to minimal data (i.e. personal details). Analytic tools, detection scripts and the logging of IP addresses all hold great benefit to site owners, but they must respect the privacy of users if they want to maintain that experience.

We're discussing value and trust here, and you're probably wondering how this relates to user experience (UX). The answer is simple: trust and confidence are essential components of the experience that users have on your website and with your brand. Trust and confidence are critical to turning one-time visitors into long-term

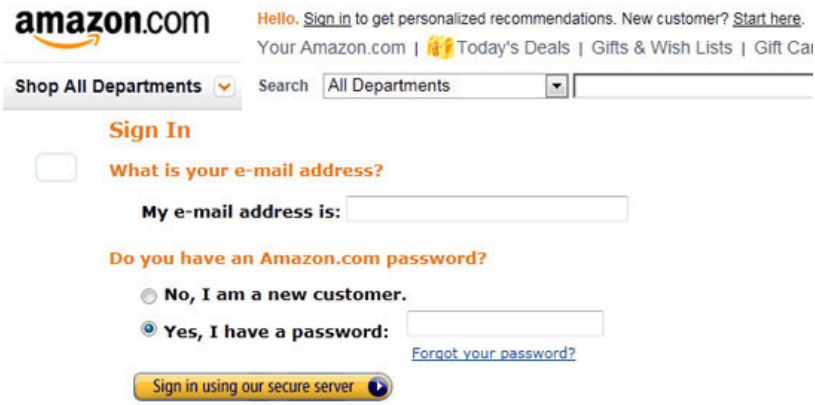
customers. If your business lacks the trust and confidence of users, then they will be reluctant to use your website.

Progressive Disclosure

If privacy problems can be so detrimental, what can we do about them? Presumably, you want to offer visitors a hassle-free experience, one in which they feel safe. A simple way to satisfy privacy concerns and remove barriers to access is by following the principle of progressive disclosure; that is, asking for and using information only when absolutely necessary.

The basic goal of progressive disclosure is to ask for the minimum amount of information. As users interact with the site and encounter something that requires them to divulge more information, that's the only time the site should ask for it. Users should have the choice not to provide the requested information (and thus may not use that feature of the site).

Take for example, Amazon.com. First-time visitors can browse the entire site without giving out any information. (A bit of an awkward example, just because Amazon.com drops cookies to track users that aren't signed in — but that's a conversation for another day.) If the visitor finds an item she likes and would like to put it on her wish list to bookmark for later, that's the only time Amazon.com will ask her to sign up for an Amazon.com account. When a new customer signs up, all they need to provide is an email address.



The new or existing customer signup form on Amazon.com.

Finally, some months later, the user comes back to Amazon.com, ready to buy the item she placed on her wish list — this is the point where Amazon.com will ask for her shipping address and payment information.

The key concept to remember in the Amazon.com example is the progressive disclosure model for acquiring user data: A website should not ask for all the data up front. Let users progressively disclose their information as they use the site.

If a visitor is registering an account on your forum, don't ask for their phone number or home address. If they're paying for goods online, you don't need to know their sex, tax bracket or marital status. Online stores commonly make the mistake of asking for credit card details even when the visitor is just window-shopping. People want to fill their cart with items before checking out and entering their credit card information.

Be sensible about when you ask for information: request it progressively, and only when it becomes necessary.

In addition to restricting your private information requests, consider how you present the requests you make, which could lower barriers. People waste a lot of time fumbling through complex forms that

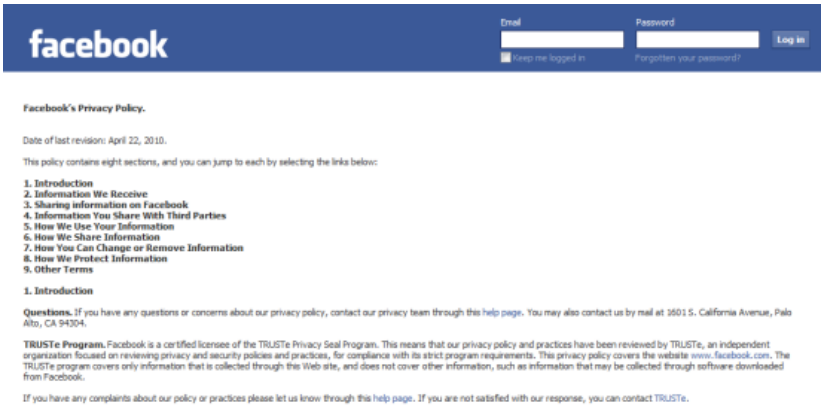
annoy them to no end; our job as web designers is to make such tasks simple. If you need users to fill out a huge form, break it down into progressive (and thus less daunting) goals to improve readability and reduce anxiety.

Breaking Down Barriers

The key to success is removing a website's barriers to access — all barriers, whether related to accessibility, usability or function. Make your website glide, not grind. Two core principles come into play here; principles by which we can satisfy our own thirst for data while still being responsive to our users' needs. The principles also suggest methods for helping visitors find what they're looking for on our websites.

The first principle is more choice, fewer options. While you'll want to avoid extremes, minimalism and reductionism are powerful in their ability to give shape to information and to remove excess from a visitor's line of sight, thus improving the company-customer relationship. Offer clear choices and remove ambiguous input fields, and you'll increase the likelihood of getting responses.

The second principle is education. The need to be transparent and sensible with users has never been greater. Privacy laws exist so that websites take steps to protect the safety of visitors and promote awareness of how user data is handled (data protection laws serve the same purpose in some countries). Posting clearly written and comprehensible (i.e. not too technical) policies in a visible place on your website can put visitors at ease, as can explaining the measures you've put in place to enact those policies.

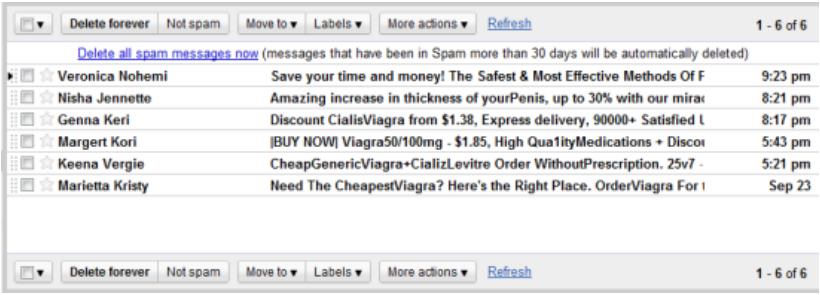


Educate users about what they'll be "giving up," and help them avoid nasty surprises.

It never ceases to amaze me how we web designers — who would never trust a web host that doesn't explain how it stores our sensitive data (user records, registration information, etc.) — are so quick to ask our own users to hand everything over with a mere "Trust me!"

Invisible Data-Mining

The last topic we should discuss is the issue of invisible data-mining (which includes recording IP addresses, using cookies, storing sessions, even using analytics software). Invisible data-mining might seem harmless enough to us professionals, but that doesn't allay the concerns of users.



Spam is a serious issue; intruding on an inbox won't win the person over.

Invisible data-mining encroaches on ethically questionable territory. I don't want to preach about what one should or shouldn't do with respect to procuring and using data; education and awareness solve most problems. In the end, though, more websites and designers should allow anonymous browsing (where sensible) and make cookies and usage-tracking optional: leave it up to the visitor.

Many people will immediately retort, "The data is harmless" or "They can easily delete the cookies." The point is that, while such tools can improve a website's UX through site improvements resulting from analysis of site activity and traffic, they shouldn't be used against the visitor's wishes, and the onus shouldn't be on users to opt out (as is the case with spam).

Value Your Users' Data and Privacy

My purpose was to highlight the importance of trust, which gets compromised when user privacy is handled poorly. Know your visitors' expectations of privacy, as well as the most current methods of handling data and the lawful ways in which data can be collected and used. You might help to dispel some of the anxiety and contention that currently afflicts users and governments. The future of the web almost certainly depends on our methods of dealing with privacy, so taking the issue seriously right now is crucial.

"User experience" is a funny term, and it can be looked at in a number of ways. The lesson to remember, though, is "Value your users." If an

element doesn't enrich the experience or encourage users to continue, your efforts will have been wasted. If your website breeds distrust, then you will certainly lose customers and possibly erode the public's regard of the web as a safe place to store data. As web professionals, we must value our users, recognize their worth and treat them with respect.

Sources:

- https://www.eff.org/deeplinks/2010/04/facebook-timeline
- https://www.paypal.com/
- https://analytics.google.com/

100 Exceedingly Useful CSS Tips and Tricks

You can never have too much of a good thing–and two good things we rely on in our work are tips and tricks. Nuggets of information, presented clearly and succinctly, help us build solutions and learn best practices. In a previous article, we shared a jam-packed list of 250 quick web design tips. It seems only right to continue the trend by showcasing 100 fresh–and hopefully useful–CSS tips and tricks.

General

Not everything in this list was easy to categorize. All of the tips that are relevant and worthy of mention but don't cleanly fit into a category are listed in this section.

```
<!--[if IE 5]>
According to the conditional comment this is Internet Explorer 5<br />
<![endif]-->
<!--[if IE 5.0]>
According to the conditional comment this is Internet Explorer 5.0<br />
<![endif]-->
<!--[if IE 5.5]>
According to the conditional comment this is Internet Explorer 5.5<br />
<![endif]-->
<!--[if IE 6]>
According to the conditional comment this is Internet Explorer 6<br />
<![endif]-->
<!--[if IE 7]>
According to the conditional comment this is Internet Explorer 7<br />
<![endif]-->
```

Conditional comments have been a godsend for resolving Internet Explorer inconsistencies.

1 It's critical when working with CSS to be aware of the various properties at your disposal and how to use them correctly.

2 Using a good editor can increase productivity. Syntax highlighting and auto-complete functionality (plus validation and code references) make life easy. Check out Notepad++, Coda, and don't discount Dreamweaver CS's code view until you try it.

3 In many ways, experimentation is the mother of innovation. Give yourself time to play; trial and error will help you learn and memorize techniques quickly.

4 Enable Gzip compression server-side whenever possible; it shrinks the size of files such as CSS and JavaScript without removing any of the content.

5 Caching will conserve bandwidth for visitors and account for much faster speeds. Take advantage of it whenever you can. Learn about optimizing browser caching.

6 Whitespace is important for CSS readability. Using whitespace to format your stylesheet adds bytes to the file size, but it's made up for in increased readability.

7 Avoid using inline code (in either elements using the style attribute or in the HTML document within <style> tags), and put them instead in your external stylesheets. It'll be easier to maintain and also takes advantage of browser caching.

8 Whatever method you use to lay code out, be consistent. You'll avoid potential problems such as misinterpretation.

9 Conditional comments can help you target versions of Internet Explorer for style. Filtering vendor-specific code isn't ideal, and comments are safer than ugly hacks.

10 This tip is slightly controversial, but I recommend using a single stylesheet rather than multiple ones. It reduces the number of HTTP requests and improves maintainability, giving your site a performance gain. This is a tip supported by Google Page Speed guidelines.

11 When there are conflicting style rules, style rules that are later down the stylesheet supersedes the ones that come before it. Thus, put mission-critical declarations at the end, where they won't be in danger of being overridden by other styles.

12 If you encounter a bug and can't determine its cause, disable CSS (using something like Firebug or the Web Developer add-on)

or simply comment out all of the styles, and then bring selectors back one by one until the fault is replicated (and thus identified).

13 Make sure your code works in various browsers. Check it against the latest versions of the top five: Internet Explorer, Firefox, Chrome, Safari and Opera.

14 Additionally, ensure that your code will degrade gracefully when CSS is disabled in the user's browser. To test this, either turn styles off in every browser or use a text browser such as Lynx.

15 Ensuring that your code degrades gracefully is obviously important, but many people forget that some visitors will have a dodgy browser or have styles turned off, so check that your fallbacks work.

16 Every browser has a built-in debugger. In IE and Firefox you can get to the inspector by hitting F12; for Chrome, Safari and Opera, press Ctrl + Shift + I.

17 Browser emulators can't replace the real thing, so use a real or virtualized edition of a browser or device.

18 Did you know that PHP code can create dynamic CSS files? Here's a tutorial on that. Just give the file a .php extension and ensure that the file declares the document header with a text/css content type.

19 Naming CSS files can be tricky. One of the best ways to approach the task is to keep file names short and descriptive, like screen.css, styles.css or print.css.

20 Any code or process you create is valuable, and recycling what you've produced for other projects is not a bad thing: pre-existing code is a great timesaver, and this is how JavaScript and CSS frameworks are born.

21 While comments in CSS files can assist other people who read or maintain them, avoid writing comments unless they are required. Comments consume bandwidth. Write CSS in a self-explanatory manner by organizing them intuitively and using good naming conventions.

22 If you're struggling to remember what's available in CSS (or even CSS3), get some cheat sheets. They're simple and can help you get used to the syntax. Here are some excellent CSS cheat sheets: CSS Cheat Sheet (Added Bytes), CSS Shorthand Cheat Sheet (Michael Leigeber), CSS 2.1 and CSS 3 Help Cheat Sheets (PDF) (Smashing Magazine).

23 Bad code breaks more websites than anything else. Validate your code by using the free, web-based W3C CSS Validation Service to reduce potential faults.

24 Vendor-specific CSS won't validate under the current W3C specifications (CSS2.1), but could give your design some useful enhancements. Plus, if you'd like to use some CSS3 for progressive enhancement, there's no way around using them in some instances. For example, the -webkit-transform and -moz-transform property was used to progressively enhance these CSS3-animated cards for users using Safari, Chrome, and Mozilla Firefox.

25 Keep multiple CSS files in a single directory, with an intuitive name such as css/. If your website has hundreds of pages, maintainability and information architecture are vital.

At-rules, Selectors, Pseudo-classes, and Pseudo-elements

Targeting your code for styling is one of the primary functions of CSS. Whether you're trying to make your code mobile-friendly, printer-friendly or just good old screen-friendly, you'll be following certain conventions. Ensuring that styles aren't in conflict, using CSS inheritance correctly and triggering actions in response to events (such as hovering) are all part of the CSS package. This section is dedicated to useful tips related to conventions.

With CSS3 media queries, designing for non-standard experiences has become easier.

26 Be careful when using the media attribute in your HTML declaration for an external CSS file. You might be unable to use media queries to better provide pre-cached alternative visuals.

27 If you find elements that use the same properties and values, group them together by using a comma [,] to separate each selector. This will save you from repetition.

For example, if you have the following:

```
h1 { color:#000; }
h2 { color:#000; }
```

Combine them as such:

```
h1, h2 { color:#000; }
```

28 Printer-friendly stylesheets are very important if you want to save your visitors' ink and paper. Use the @media print at-rule, and remove anything that isn't necessary on the printed page.

29 Accessibility also has to do with how the written word is spoken. The aural [now deprecated in CSS] and speech media queries can improve usability for screen readers.

30 Unfortunately, the handheld media query in CSS isn't widely supported. If you want your website to work on mobile devices, don't depend on it to serve the correct visuals to mobile devices.

31 Take the time to eliminate duplicate references and conflicts in your CSS. It will take some time, but you'll get a more streamlined render and fewer bytes in your files.

32 When working with mouse hover events, deal with the (1) :link pseudo-class, then (2) :visited, then (3) :hover and finally (4) :active — in that order. This is necessary because of the cascade.

33 Making a website work well on Apple iOS devices is straightforward: to scale your design, just use CSS3 media queries with the appropriate min-width and max-width values. Here's a tutorial for that.

34 Make the most of CSS inheritance by applying required styles to parent elements before applying them to the children. You could hit several birds with one stone.

35 You can apply multiple classes to an element with space separation, which is great if you want to share properties with numerous elements without overwriting other styles.

36 If you don't want to deal with IE6's conditional comment quirks–they require a separate CSS file–then you can use the star hack (* html) selector, which is clean and validates.

37 HTML tooltips are fine for plain text, but the :hover pseudo-class for CSS tooltips gives you more options for showing styled content. Check out this tutorial on how to create CSS-only tooltips.

38 Using attribute selectors, you can add icons or unique styles depending on the file type you link to in an anchor. Here's an example with which you can add style to a PDF link: a[href$='.pdf].

39 You can also use attribute selectors to target a specific pseudo-protocol such as mailto or skype: [href^="skype:"].

40 Rendering CSS takes time, and listing selectors adds bytes. Reduce the workload of a renderer by using only the selectors you require (an id is better than many child references).

41 Not everyone will agree with this, but I recommend writing every "custom" selector down as a class (before making it an id) to help eliminate duplicate entries.

42 When structuring your CSS file by selectors, the convention is to list elements, then classes (for common components) and finally any ids (for specific, unique styles).

43 Naming conventions for selectors are important. Never use names that describe physical attributes (such as redLink), because changing the value later could render the name inappropriate.

44 Case sensitivity relates to naming conventions. Some people use dashes (e.g. content-wrapper) or underscores (i.e. content_wrapper), but I highly recommend using camel case (e.g. contentWrapper) for simplicity.

45 The universal selector (*) is used widely in CSS reset mechanisms to apply specific properties to everything. Avoid it if you can; it increases the rendering load.

46 With CSS3 media queries, you can easily target the orientation of a viewport with portrait or landscape mode. This way, handheld devices make the most of their display area.

47 Apple's devices are unique in that they support a <meta name="viewport"> tag, which has stylistic value attached to it. You can use this to force the screen to zoom at a fixed rate of 100%.

48 The two CSS3 pseudo-elements, :target and :checked have great potential. They apply their designated style only to certain events and can be useful as hover event alternatives.

49 Embedding content in CSS is a unique way to give anchor links some description in printer-friendly stylesheets. Try them with the ::before or ::after pseudo-elements.

50 IDs can serve a dual purpose in CSS. They can be used to apply styling to a single element and to act as an anchoring fragment identifier to a particular point on the page.

Layout and the Box Model

When we're not selecting elements for styling, we spend a lot of time figuring out how things should appear on the page. CSS resets and frameworks help us with consistency, but we should know how to correctly apply styles such as positioning and spacing. This cluster of useful tips relates to the aspects of CSS that fundamentally affect how the components of a website appear and are positioned.

Positioning plays a critical role in the readability of information and should not be ignored.

51 Many designs are focused on grids and the rectangular regions of the viewport. Instead, create the illusion of breaking out of the box for added effect.

52 If margin: auto; on the left and right sides isn't getting you the central point you desire, try using left: 50%; position: absolute; and a negative margin half the width of the item.

53 Remember that the width of an item constitutes the specified width as well as the border width and any padding and margins. It's basically a counting game!

54 Another controversial tip here: don't use CSS resets. They are generally not needed if you take the time to code well.

55 A CSS framework such as Blueprint or YUI Grids CSS might assist you speed up development time. It's a bit of extra code for the user to download, but it can save you time.

56 Remember that Internet Explorer 6 does not support fixed positioning. If you want a menu (or something else) to be sticky, it'll require some hacks or hasLayout trickery.

57 Whitespace in web designs is amazing; don't forget it. Use margins and padding to give your layout and its content some breathing room. You'll create a better user experience.

58 If one thing overcomplicates the task of scaling a design the way you want, it's using inconsistent measurements. Standardize the way you style.

59 Different browsers have different implementations; visually impaired users may want to zoom in, for example. Give your layout a quick zoom-test in web browsers to ensure the style doesn't break!

60 Most browsers can use box-shadow without extra HTML. IE can do the same with filter: progid:DXImageTransform.Microsoft.Shadow(color='#CCCCCC', Direction=135, Strength=5);

61 Rounded corners aren't as difficult to make as they used to be. CSS3 lets you use the border-radius property to declare the curvature of each corner without surplus mark-up and the use of images.

62 One disadvantage of liquid layouts is that viewing on a large screen makes content spill across the viewport. Retain your desired layout and its limits by using min-width and max-width.

63 WebKit animations and transitions might work only in Safari and Chrome, but they do add a few extra unique, graceful flourishes worthy of inclusion in interactive content.

64 If you want to layer certain elements over one another, use the z-index property; the index you assign will pull an element to the front or push it behind an item with a higher value.

65 Viewport sizes aren't a matter of resolution. Your visitors may have any number of toolbars, sidebars or window sizes (e.g. they don't use their browsers maximized) that alter the amount of available space.

66 Removing clutter from an interface using display:none might seem like a good idea, but screen-reader users won't be able to read the content at all.

67 Be careful with the overflow CSS property when catering to touch-screen mobile devices. The iPhone, for example, requires two fingers (not one) to scroll an overflowed region, which can be tricky.

68 Have you ever come across CSS expressions? They were Microsoft's proprietary method of inserting DOM scripts into CSS. Don't bother with them now; they're totally deprecated.

69 While the CSS cursor property can be useful in certain circumstances, don't manipulate it in such a way as to make finding the cursor on the screen more difficult.

70 Horizontal scrolling might seem like a unique way to position and style content, but most people aren't used to it. Decide carefully when such conventions should be used.

71 Until Internet Explorer 9 is final, CSS3 will have some critical compatibility issues. Don't rely too heavily on it for stable layouts. Use progressive enhancement concepts.

72 CSS makes it possible to provide information on demand. If you can give people information in small chunks, they'll be more likely to read it.

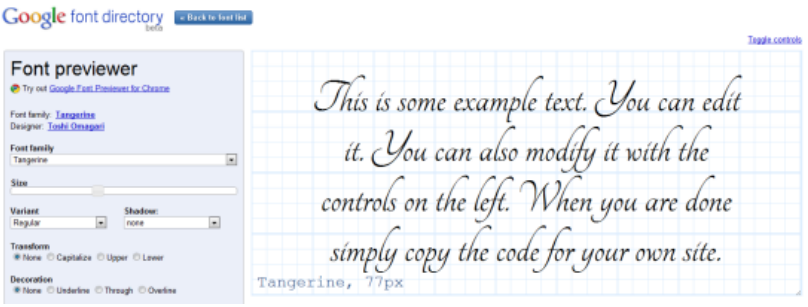
73 When showcasing a menu in CSS, be consistent in implementation. People want to know about the relationship between objects, and it's important to avoid dissonance.

74 CSS isn't a solution to all of your layout woes–it's a tool to aid your visual journey. Whatever you produce should be usable and logically designed for users (not search engines).

75 Once your layout is marked up with CSS, get feedback on how usable it really is; ask friends, family, co-workers, visitors or strangers for their opinions.

Typography and Color

If one thing deserves its own set of tips, it's the complex matter of adding typography, color and imagery to CSS. We want readable content and we want it in a consistent layout. In this section, we'll learn to take advantage of typography and color, which are powerful conventions in design. I'll talk about "web-safe" and share tips relating to the latest craze of embedding fonts.



"Web-safe" concepts have changed over time and could soon become a non-issue.

76 Squeezing content too close together can decrease overall readability. Use the line-height property to space lines of text appropriately.

77 Be cautious about letter-spacing: too much space and words will become illegible, too little and the letters will overlap.

78 Unless you have good reason, don't uppercase {i.e. text-transform:uppercase; } long passages of text {e.g. HEY GUYS AND GALS WHAT'S UP?}. People hate reading what comes off as shouting.

79 Accessible websites have good contrasting colors. Tools exist to measure foreground colors against background colors and give

you a contrast ratio. Check out this list of tools for checking your design's colors. Be sure your text is legible.

80 Remember that default styles might differ greatly from browser to browser. If you want stylistic flourish, reinforce the behavior in the stylesheet.

81 In the old days, the number of colors that a screen could display was rather limited. Some people still use old monitors, but the need for web-safe colors has drastically reduced.

82 Building a font stack is essential when making a design degrade gracefully. Make sure that fallback typefaces exist in case the one you request isn't available.

83 With Vista, Windows 7 and MS Office 07â€“10 (and their free viewers), a number of cool new web-safe fonts have become available, such as Candara, Calibri and Constantina. Read about Windows fonts.

84 Plenty of smartphone apps can boost your ability to build a stylesheet, but Typefaces for the iPhone and other iOS4 devices is particularly useful because it shows you every font installed.

85 Web-safe fonts are no guarantee; people could quite possibly uninstall a font even as ubiquitous as Arial (or Comic Sans!). Then their browsers wouldn't be able to render it.

86 Avoid underlining content with the text-decoration property unless it's a real link. People associate underlined text with hyperlinks, and an underlined word could give the impression that a link is broken.

87 Avoid the temptation to use symbolic typefaces like Wingdings or Webdings in the layout. It might save KBs on imagery, but it'll serve nonsensical letters to screen-reader users.

88 Remember that @font-face has different file format requirements depending on which browser the user is on, such as EOT, WOFF, TTF and OTF (as you would find on a PC). Serve the appropriate format to each browser.

89 The outline property is seriously underused as an aid to web accessibility. Rather than leaving it set to the default value, use border styles to enhance an active event's visibility.

90 Smartphones do not have the same level of core support for web typography that desktop browsers do, which means fewer web-safe fonts and no conventional @font-face embedding.

91 Cross-browser opacity that degrades gracefully can be applied using the -ms-, -moz-, -khtml- vendor prefixes and the filter: alpha(opacity=75); property for Internet Explorer.

92 You can make background-image bullets by using list-style-type:none;, adding a padding-left indent and positioning the background-image to the left (with the required pixel offset).

93 Helping users identify an input box is easy; just add a background image of the name (like "Search" or "Password") and set it so that the image disappears when the box is clicked on by using the :focus pseudo-class and then setting the background property to none.

94 Large and visible click regions enhance the usefulness and usability of anchor links, which in turn guide people among pages. Be attentive when you style these.

95 Remember that background images do not replace textual content. If a negative indent is applied, for example, and images are disabled, then the title will be invisible.

96 Navigation items need to be labeled appropriately. If you use a call-to-action button or an image-based toolbar, make sure a textual description is available for screen readers.

97 If the idea of applying opacity with a bunch of proprietary elements doesn't appeal to you, try RGBA transparency (in CSS3) instead of background-color: rgba(0,0,0,0.5);.

98 If your visitors use IE6, avoid using px as the unit of measurement for text. IE6 users have no zoom functionality; they rely on text resizing, which doesn't work with px. Use a relative unit instead, such as em.

99 Providing alternative stylesheets for supported browsers such as Firefox and Opera can enhance readability, as in the case of high-contrast alternatives.

100 If you find choosing colors difficult, web-based tools like Adobe Kuler, desktop tools like ColorSchemer Studio and mobile apps like Palettes Pro might help.

Style Can Be Stylish!

CSS has come a long way in recent years. With browser makers implementing the CSS3 specification even before it's finalized, adding unique proprietary styles (such as WebKit transformations) to the mix and increasingly supporting web standards, there has never been a better time to be a web designer. In the past, we could only hope that our styles would be correctly applied, but it seems that desktop and mobile platforms are improving like never before.

Web designs have come a long way since the '90s, and that's a good thing.

Implementing CSS can be frustrating, what with ongoing web-browser issues, but it's still one of the most fun web languages you can engage with. Rather than laying out the structure of objects or fiddling with complex mechanisms, you can dictate how content should appear. It's like being given a blank piece of paper and a pack

of crayons. And designers are experimenting with the available styles to create beautiful experiences for audiences.

Consider the implications of every property and style you declare. CSS can turn the simplest or most minimalist layout into a complex structure of interactivity that would terrify all but the most dedicated individuals. As the capabilities and options in CSS grow and devices are updated to support them, a new wave of unique layouts will appear. Hopefully a number of them will be yours.

Sources:

- https://notepad-plus-plus.org/
- https://panic.com/coda/
- https://helpx.adobe.com/dreamweaver/tutorials.html
- https://en.wikipedia.org/wiki/Gzip
- https://en.wikipedia.org/wiki/Web_cache
- https://developers.google.com/speed/docs/insights/LeverageBrowserCaching?csw=1
- https://www.quirksmode.org/css/condcom.html
- https://developers.google.com/speed/docs/insights/mobile?csw=1
- https://getfirebug.com/
- https://www.cheatography.com/
- https://www.smashingmagazine.com/2010/05/css-2-1-and-css-3-help-cheat-sheets-pdf/
- http://jigsaw.w3.org/css-validator/
- https://jonikorpi.com/
- https://www.w3.org/TR/CSS2/media.html#at-media-rule
- https://www.w3.org/TR/css3-mediaqueries/
- https://www.smashingmagazine.com/2010/07/how-to-use-css3-media-queries-to-create-a-mobile-version-of-your-website/
- https://en.wikipedia.org/wiki/Camel_case

- https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag
- https://www.w3.org/TR/selectors-3/
- https://www.w3.org/DesignIssues/Fragment.html
- https://24ways.org/
- http://www.blueprintcss.org/
- https://yuilibrary.com/
- http://webfx.eae.net/dhtml/cssexpr/cssexpr.html
- https://fonts.google.com/
- https://daringfireball.net/2007/07/iphone_fonts
- https://en.wikipedia.org/wiki/Web_Open_Font_Format
- https://color.adobe.com/
- http://www.maddysoft.com/iphone/palettes/

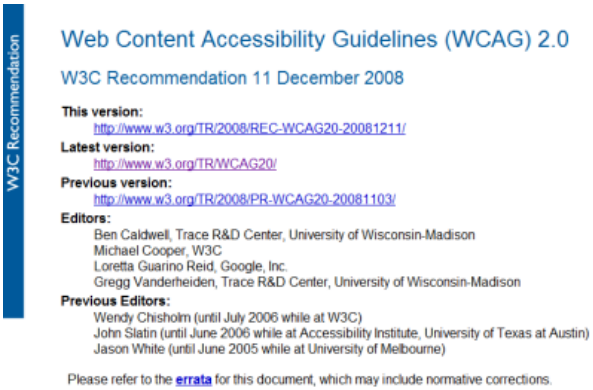
The A-Z List for Web Designers

There are so many technical aspects of web design and development that it can be pretty hard work getting to grips with all the intricacies that have become a part of our ever-growing industry. This A-Z list attempts to assigns each letter of the alphabet to an important aspect of our work as professionals that make websites.

In this list, you may come across terms you're already aware of, or things you might not have ever heard of, but in any case, I recommend that everyone should have at least a basic working knowledge of these particular items. Some things are more orientated toward design, others are aimed toward front-end development, but all of them provide their own unique benefits to the web pages you build.

A is for Accessibility

Accessibility is one of the most critical aspects of our job, as many individuals browse the web with impairments that require non-traditional means of website access, such as screen readers and input-assistive devices. Accessibility, though, is also about universal design; designs that can be used through various situations such as mobile devices or older browsers. While there is no clear definition as to how far accessibility extends, a range of best practices to help certain conditions (such as visual, aural, and motor impairments) have been produced. If you're not already aware of such issues, it's well worth investigating further.



W3C Recommendation

Web Content Accessibility Guidelines (WCAG) 2.0

W3C Recommendation 11 December 2008

This version:
http://www.w3.org/TR/2008/REC-WCAG20-20081211/
Latest version:
http://www.w3.org/TR/WCAG20/
Previous version:
http://www.w3.org/TR/2008/PR-WCAG20-20081103/
Editors:
Ben Caldwell, Trace R&D Center, University of Wisconsin-Madison
Michael Cooper, W3C
Loretta Guarino Reid, Google, Inc.
Gregg Vanderheiden, Trace R&D Center, University of Wisconsin-Madison
Previous Editors:
Wendy Chisholm (until July 2006 while at W3C)
John Slatin (until June 2006 while at Accessibility Institute, University of Texas at Austin)
Jason White (until June 2005 while at University of Melbourne)

Please refer to the **errata** for this document, which may include normative corrections.

The Web Content Accessibility Guidelines are a great place to begin looking at the subject of accessibility.

Laws in many countries influence the need for web-accessible websites as a result of governments seeking to give its citizens equal access to information technology. Most web designers can implement accessibility standards at a basic level just by following web standards and best practices, and these implementations often improve the quality of the site produced, even for able-bodied site visitors. Web accessibility is quite an intricate subject, and will require time and experience to learn fully, but knowing that your products provide universal access makes it worth the extra effort.

B is for Browsers

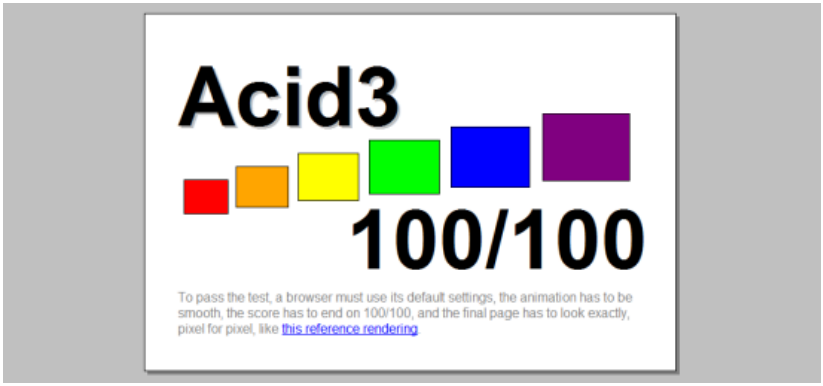
Every internet-enabled device has software that makes those lovely pages of yours viewable by users. The browser is that software, and it is among the most vital elements of the web experience. Most browsers these days render sites uniformly, but old browsers like IE6 may give you issues.

If you see a person using Internet Explorer 6, you may as well abandon all hope!

While hundreds of browsers exist, there are at least five major browsers that you should be concerned about: Internet Explorer, Mozilla Firefox, Google Chrome, Apple Safari and Opera. A web designer having all of these browsers installed on their work machine for testing purposes is not a bad idea. In addition, testing on mobile devices (using their native browsers) is also recommended.

C is for CSS

CSS is the primary method of styling HTML elements. It's what makes a boring page look visually stunning.



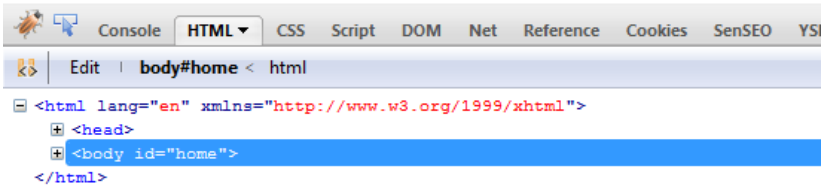
The Acid tests show CSS-standards-compliance of a browser.

Cross-browser compatibility is an issue with CSS — so it's important that you validate your code, know all of the selectors, properties and

values you can work with, and consider the browsers' needs. If you're thinking of expanding your knowledge of CSS for future-standards-compliant browsers like Chrome, Safari and Firefox, now is a good time to learn about how you can progressively enhance your web designs with CSS3.

D is for Debugging

Sometimes your code doesn't work in the way that you intended. The need for debugging has only increased with the range of web languages we now use, the more complex styles of designs we produce, and the chaotic amount of browsers — that now includes the Mobile Web — we need to support. Ensuring that your code works (and works well) has become a skill that all designers should possess.

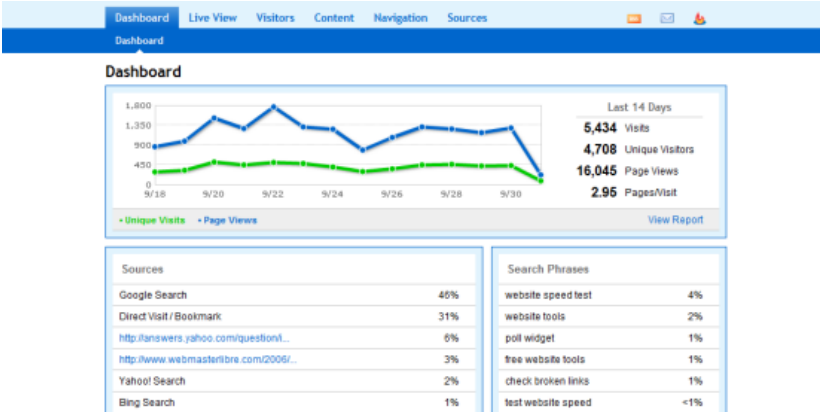


Firebug is an awesome browser extension for debugging front-end source code.

Being able to surgically resolve rendering issues requires a deep and insightful knowledge of the languages you work with, patience, experience, and critical thinking skills.

E is for Ethnography

Let's face it, your site's most important component are the visitors who spend their time browsing your pages. The subjects of Ethnography and, to a greater extent, sociology, are based around the need to understand your audience. It involves collecting meaningful data through studies and research in order to determine the optimal design for a site.



Statistics, such as those from W3Counter, help you understand an audience.

While you may think people are all relatively the same, the cultural differences within us have effects in the way our designs are perceived. It's worth learning about sociology, even in just a fundamental level.

F is for Flash

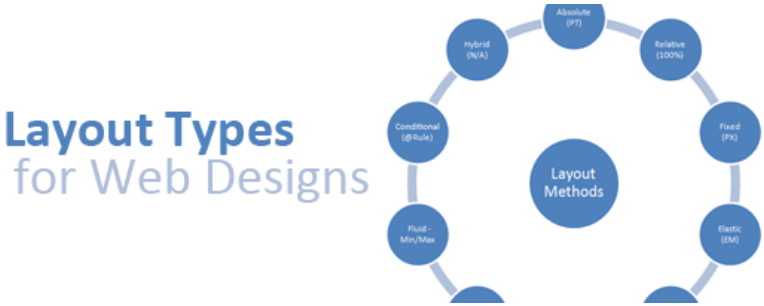
Love it or hate it, Adobe's Flash is a widely implemented web technology that has been around for years. While HTML5 won't kill its purpose as a tool for rapidly building rich, responsive and interactive content (e.g. web-based games and animation sequences) anytime soon (if at all), its worth knowing how the technology works if you are considering the use of highly interactive web media in your web designs.

While Flash is well supported, it can be easily disabled or uninstalled.

It's important to ensure that your site works if Flash is not supported by the user's browser. Many people choose to have Flash disabled, and Apple just recently undid its ban of the technology in its devices. Flash is an excellent web technology when developing with design best practices in mind.

G is for Graphics

Websites without images and graphics would be boring, and knowing how to create useful and captivating images is an imperative skill to have as a web designer.



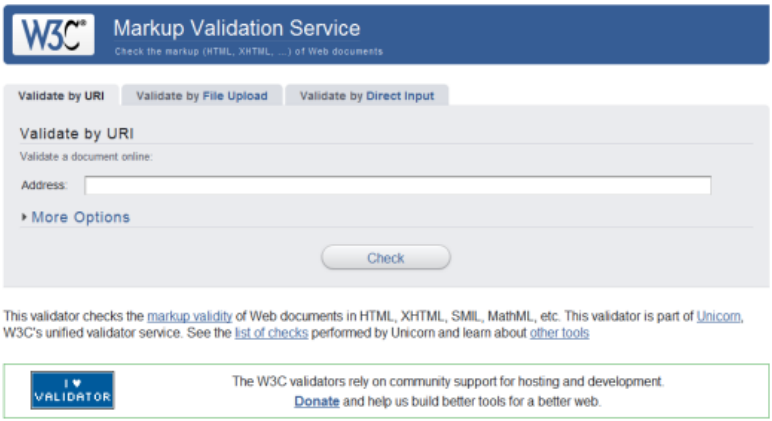
Images are a huge part of the web.

Since the early days of the web, the ability to embed images in an HTML document has enhanced the level of visual aesthetic that can

be applied to a page layout. Graphics are often used for things that can best be presented in visual form rather than text form. For example, charts and graphs can enhance the reader's ability to understand data being presented to them.

H is for HTML

If you build websites, you need to know HTML. It's pretty much as simple as that! The structural language has been around since the web came into being. HTML is meant for marking up your content in such a way that it emphasizes the semantic meaning of it.

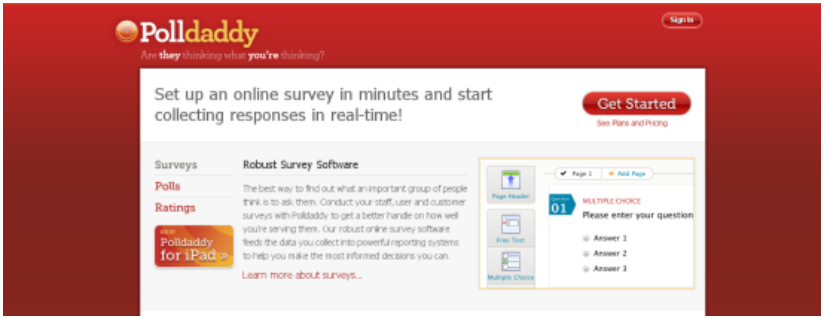


Validating your code using services such as W3C's Markup Validation Service is a good idea.

While cross-browser compatibility isn't so much of an issue with HTML (unless you're branching out into HTML5 or using the proper MIME-type for XHTML), it's still important to validate your code and use the right HTML elements for the right job. HTML is often the starting point for web designers and developers alike and is the most fundamental component of our websites.

I is for Interaction Design

Back in the early days of the internet, web pages were static. We have evolved past that boring era, and now we have pages filled with stuff that moves, responds to user actions, and provides rich components for our visitors that improve their experience (e.g. web forms and real-time information widgets). Interaction design focuses on the philosophy that websites should be utilitarian.

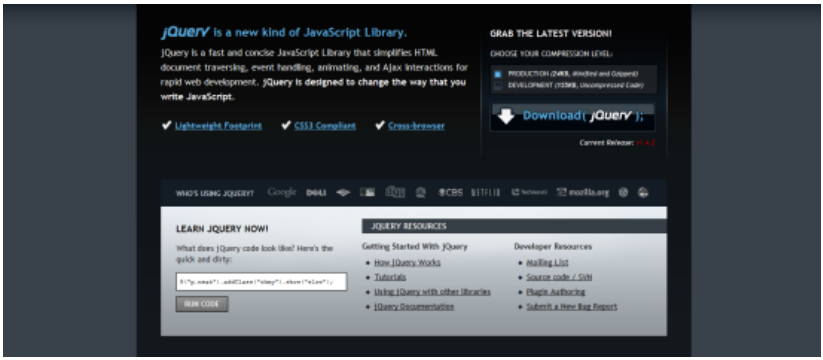


Getting visitors to interact with your site requires an engaging and useful experience.

Whether you use polls, have contact forms, or host a forum — knowing solid interaction design principles is a worthwhile pursuit.

J is for JavaScript

JavaScript tends to scare beginning web designers off due to it being more complex than the simple markup language they're likely more familiar with (HTML). However, JavaScript does play an instrumental part in making modern websites, and especially web applications. JavaScript enhances the user experience through asynchronous, real-time updating of web pages when an event is triggered (such as a click) through a technique collectively known as Ajax. JavaScript also provides slick interactivity and smooth effects that, by design, is aimed to improve interaction tasks.

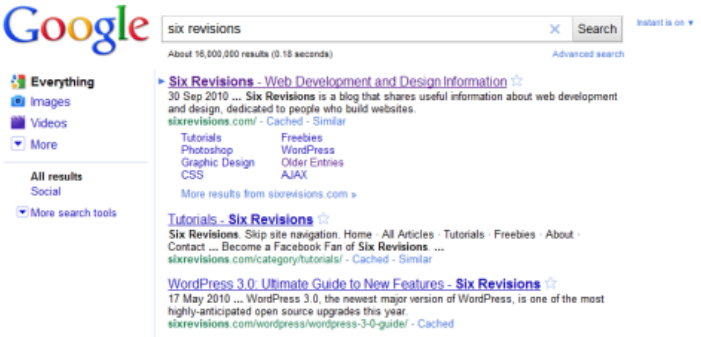


JS frameworks like jQuery aid agile development.

JavaScript does have its limits (and for a good reason, as it could be used to exploit vulnerabilities in a user's computer) and it may not always be available on a user's browser because it can be disabled. Just like Flash, the best practice for JavaScript is that a web page should be usable and accessible without JavaScript.

K is for Keywords

OK, so this one is stretching our web design A-Z list a bit. However, keywords do play their part in search engine optimization and marketing, which relates to web design. Marketing has an unbelievable level of influence on the success of the website projects you build.



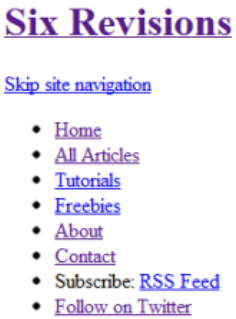
Marketing your website can be done on-page, off-page and even totally offline.

While marketing may appear like child's play, there's a lot more to it than meets the eye. Advertising, SEO, conferences, meet-ups, sponsorship, social networking, viral promotions, PPC, optimizing your site's markup, distributing freebies that provide a link back to your site, and many other methods exist. Knowing about keywords, and about SEO and marketing, gives you a leg up. These things, though, are intrinsically going to be a part of the sites you build if you follow web design and web development best practices.

L is for Limitations

Though web technologies quickly evolve in a relatively short period of time, web designers will always have limitations in the things they can do on a web page. Limiting factors can also be non-technology-related, such as a person's skill and a project's budget.

While the web is evolving, it's still relatively young and can't do everything we'd like it to (even when we push the boundaries quite often).

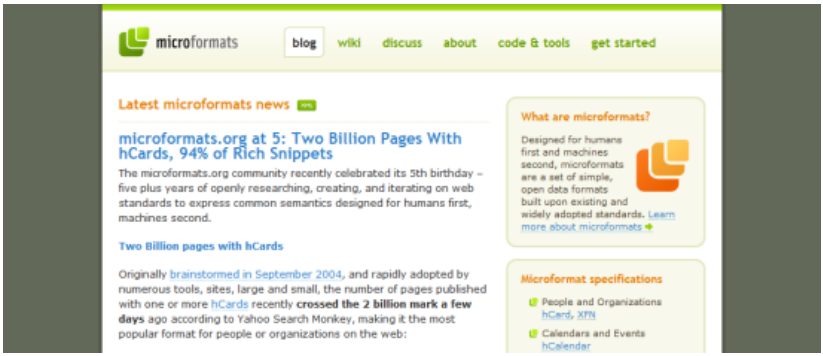


It's important to make sure everything degrades gracefully, with no exceptions.

Dealing with limitations, of how to overcome constraints, is an essential part of being a web designer. Education, experimentation, and experience play a vital role to the advancement of your career. Even the most established professionals require an open mind and the seed of doubt in what they already know so that they can keep pushing forward and stay ahead of the curve.

M is for Metadata

One of the most underestimated elements of web design (which few people give much thought to) is the notion of metadata. Essentially, the concept behind the subject is to provide materials that describe the information's content (data about data). Within the constructs of the web, some of the most well known uses for metadata are Meta tags, RDF data and purpose-built microformats.



There is more to life than HTML, you can play with microformats too.

By including metadata in your documents, you can not only describe your pages in such a way that they can be better indexed by web spiders (like how libraries have indexes to find the book you're looking for) but you can also markup the information on a page in a way that gives them added meaning and utility. An example of this is how a vCard can be coded for browsers, apps (such as an email client) and services that are set to recognize the convention.

N is for Navigation

Another essential component of web design is the concept of navigation and information architecture. A website may contain hundreds, if not thousands, of pages and this presents a challenge of how to ensure that people can find what they are looking for. Knowing how to organize and structure information is a critical skill to learn.

Information architecture is useful in aiding the ability of users to find what they want.

Evaluating the value of (and using) things like breadcrumbs, search features, and content categorization is a significant part of the web design process.

O is for Objectivity

Often, web designers make the mistake of creating experiences that reflect their preferences rather than the users'. Who can blame us? It's hard to be empathic toward your audience if you're not one of them. Remaining objective and dealing with the jobs you undertake professionally is a skill that doesn't come with a manual, but nonetheless needs to be learnt.

Your design isn't based on what you think is pretty (or else things could get ugly)!

Before you begin producing a website, you need to know what you're producing, why you're producing it, and who's going to use it. Research, designing by numbers, knowing general usability guidelines and understanding how to collaborate effectively are but a few things that are involved in creating web designs objectively. Not letting your own biases take control of a project will allow you to make logical considerations that empathize with the greater needs of your clients and users.

P is for Psychology

Of all the areas related to user experience design, psychology is probably amongst the most interesting and useful. Everything a person does on the web relates to the way they behave and think. Knowing how to engage with that behavior will allow you to maximize the success of your web designs.

Human behavior can help you identify emotional attachment to color.

If you haven't really thought much about psychology and you're designing sites for a living, it's really worth learning about psychology, at least at a basic level and in ways that relate to Design. For example, you could learn about how Gestalt psychology can be applied to Web Design. Most aspects of design and what we define as beauty is depicted through psychology, as are the ways we can "train" visitors to become accustomed to a particular interface.

Q is for Quality

An important part of any business is the concept of quality control. Being a successful designer means that you need to be passionate and caring of your craft. In such a highly competitive market, the need to maintain a set of standards, do what's in the users' best interests and ensure what you produce meets requirements is vital. Whether you are a designer or coder, the ability to not just follow specifications, but also to set your own ideals, is worthy of attention.

Why would anyone release substandard work? Make sure you set high standards for your projects.

Whenever I produce a web design, I have a whole compendium of checklists and requirements that it needs to meet before the project is given the final "all clear." It takes a bit of time to develop some decent standards for your own work — and it has to be a personal initiative. However, once you have, you can continue to evolve and refine your processes, not only saving time, but also ensuring that you build a solid reputation for yourself as someone who cares about their work's quality.

R is for Readability

If there's one thing that is repeated to you Buzz-Lightyear-style (i.e. "To infinity and beyond!"), it's the notion that "Content is King." Without content, your website is worthless. And content isn't just text and articles such as the one you're reading now. It's a web app's marketing copy, it's a call-to-action button's choice of words, it's the perfectly placed video demo, or that awesome infographic that takes advantage of visual learning. As such, ensuring that you make your content as interesting, readable and as user-friendly as possible is a sure way to encourage regular visits.

Content is king and ensuring it's easy to understand and visible is part of your job.

Part of readability is the subject of information design, laying out your content in a manner that will appeal to your visitors. Being able to design and code a website are important, but so is usable content.

S is for Server-Side

JavaScript's a great client-side language, but server-side scripting is also essential to modern websites. It does everything else that client-side scripting can't (or shouldn't) do. Saving user input, retrieving and processing database data, and page templating are but a few fundamental examples of how much server-side scripting is a part of our products.

There are plenty of languages and server-side frameworks, such as Rails, to choose from.

Obviously most people will be aware of server-side scripting in some form, but for beginners to the whole backend business-logic arena, it's valuable knowing even just the basics of what server-side scripting can do in order to know the possibilities and limitations.

T is for Typography

With @font-face gaining more widespread support, and standards for web fonts such as WOFF being drawn up, the past restraints on web typography are slowly fading away. But as Peter Parker's uncle warned, "with great power comes great responsibility." With the increasing number of ways we can design type on a web page, the need to understand typography has become a much more significant part of a web designer's job.

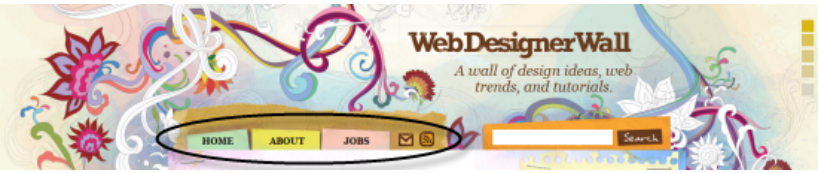
My name is Comic Sans and nobody loves me! :(

People use many fonts on the web, but no one should be using Comic Sans!

If you don't already have a working knowledge of fonts and typography, now is the time to begin. Not only does it have important implications in design and how content is portrayed visually, but also it will ultimately give you added levels of control over page aesthetics.

U is for Usability

Ensuring that your web designs are user-centered and user-friendly is quite a challenge, especially when designing for a diverse spectrum of web users. Usability focuses around the notion that your users don't want more problems, they want solutions — and it's your job to give them what they want as effortlessly as possible.

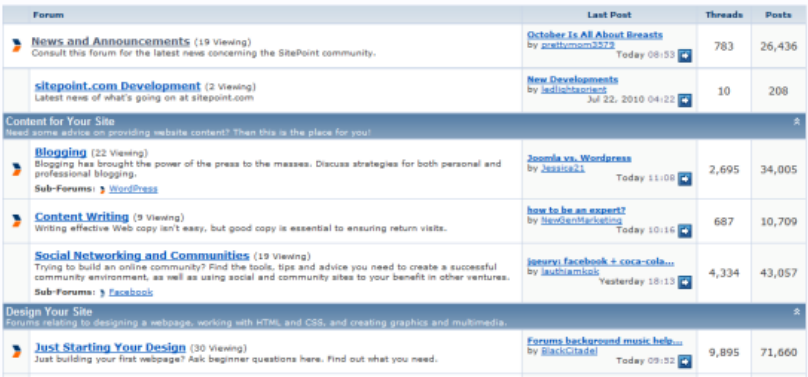


Giving your visitors a smooth experience is part of the package of a successful site.

With ties to interaction design, accessibility, information architecture, user experience, human factors and more, the expansiveness and significance of usability in modern web design needs not be underscored further.

V is for Visitors

Without site visitors, what's the point of a website? You can have the most awesome website in the universe with mind-blowing content that unravels the secrets of time traveling, but if there's no one reading it, your site may as well not exist. While not to detract from the importance of content, a critical skill of design is to get eyeballs on your site (and keep them there).



Forum	Last Post	Threads	Posts
News and Announcements (19 Viewing) Consult this forum for the latest news concerning the SitePoint community.	October Is All About Breasts by prettymom2579 Today 08:53	783	26,436
sitepoint.com Development (2 Viewing) Latest news of what's going on at sitepoint.com	New Developments by ladlightorient Jul 22, 2010 04:22	10	208
Content for Your Site Need some advice on providing website content? Then this is the place for you!			
Blogging (22 Viewing) Blogging has brought the power of the press to the masses. Discuss strategies for both personal and professional blogging. Sub-Forums: WordPress	Joomla vs. Wordpress by Jessica21 Today 11:08	2,695	34,005
Content Writing (9 Viewing) Writing effective Web copy isn't easy, but good copy is essential to ensuring return visits.	how to be an expert? by NewGenMarketing Today 10:16	687	10,709
Social Networking and Communities (19 Viewing) Trying to build an online community? Find the tools, tips and advice you need to create a successful community environment, as well as using social and community sites to your benefit in other ventures. Sub-Forums: Facebook	jquery: facebook + coca-cola... by lauthiamkok Yesterday 18:13	4,334	43,057
Design Your Site Forums relating to designing a webpage, working with HTML and CSS, and creating graphics and multimedia.			
Just Starting Your Design (30 Viewing) Just building your first webpage? Ask beginner questions here. Find out what you need.	Forums background music help... by BlackCitadel Today 09:52	9,895	71,660

A social community will encourage people to continue visiting a site.

Many aspects of user-centered design focus on encouraging community involvement as it identifies the need to emotionally tie people to a site. However, beyond the social aspects, it's also useful to identify your visitors and work with them to make a website better. This ties in quite nicely with the subject of sociology, ethnography and empathy.

W is for Web Standards

The letter "W" could stand for so many things on the World Wide Web that it became a hard choice to decide which item to discuss. However, the idea of web standards underlines a core principle that many in our industry hold dear. Knowing the various languages and standards (and how they work) is something that everyone in the field professionally should be familiar with.

W3C Working Draft

Introduction to CSS3

W3C Working Draft, 23 May 2001

This version:

•http://www.w3.org/TR/2001/WD-css3-roadmap-20010523

Latest version:

•http://www.w3.org/TR/css3-roadmap

Previous version:

•http://www.w3.org/TR/2001/WD-css3-roadmap-20010406

Editors:

•Eric A. Meyer <eric@meyerweb.com>

•Bert Bos (W3C) <bert@w3.org>

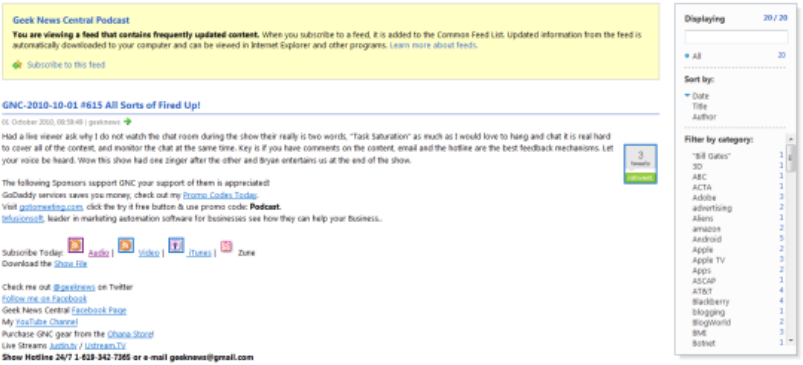
Abstract

Following specs strictly can be time-consuming.

Every language used on the web, from HTML to Python, has its own set of specifications that outlines recommendations and best practices as to how the language should be used (and in what context). Following these standards can be tricky because it requires a lot of reading, patience, and motivation, but if you want to master a language like HTML, CSS, or Ruby (on Rails), you'll need get used to following standards and keeping your knowledge current.

X is for XML

A language that has so many purposes, yet gets pushed to the wayside regularly in our industry, is XML. Being more flexible than HTML in syntax, it's almost like a Swiss Army knife in its unique and multi-functional purpose. Used in so many aspects of the web — from software UI settings, non-SQL local databases, public APIs of third-party services like Twitter's, Sitemaps, and syndication formats like RSS — it's become a mainstay of web development.



RSS is a content syndication language written with XML.

Oddly enough, while XML is so durable, it's not generally considered a core requirement to know. However, remember that with the web constantly evolving, the need for XML is only likely to increase to cope with our diversifying data needs.

Y is for "Yes!"

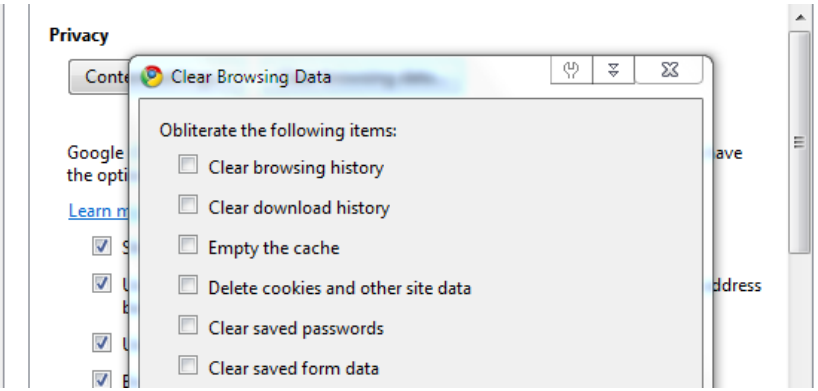
Web design and development is often made up of a series of choices. Knowing how to say "no" is quite an important ability to gain. Knowing when to say "yes" is, too. While this may seem a rather woolly inclusion on our list, if you think about when you produce a website, you constantly deal with micro-decisions, and thus, your ability as a web designer is largely based on how you make judgments and decisions.

Knowing when to say "yes" and "no" is critical.

While it remains an important part of our job — Do we take that client? Should I use PHP? Does it degrade gracefully? — learning about decision-making techniques and incorporating a sound judgment process with your projects is as critical of a skill to learn as code, content, design or theory. Bad decisions lead to a poorly built sites and good decisions save you time, money and stress.

Z is for Zipping

Finally, we come to the last letter of the alphabet and what other choice would fit into this section except zipping (or file compression). The ability to reduce bandwidth consumption has many benefits, including reduced data transfer costs, faster page loads for the visitor and, when combined with caching, reductions in HTTP requests.



Caching is another useful method of reducing the amount of bandwidth used.

Knowing how to optimize your content and code is one thing, but being able to squeeze every last unnecessary byte from your images and knowing how to carefully balance the quality and file weight is critical as well. Learning to optimize website assets is an essential skill which no budding web designer wants to be without.

Alphabet Soup

The web is an ever-evolving platform, and new techniques, technologies and paradigms seem to appear quite regularly. But luckily, we can pick our battles carefully and learn only what's really important to us and to our projects.

So now that you know your ABC's, it's time to get out there and perhaps delve deeper into subjects that you feel will be of benefit to you as a web designer.

Sources:

- https://www.w3.org/TR/WCAG20/
- http://acid3.acidtests.org/
- https://getfirebug.com/
- https://en.wikipedia.org/wiki/Ethnography
- https://www.w3counter.com/globalstats.php

- https://www.adobe.com/products/flash/
- https://venturebeat.com/2010/09/09/apple-loses-game-of-chicken-allows-flash-and-other-conversion-tools-for-ios-apps/
- https://validator.w3.org/
- https://polldaddy.com/
- http://jquery.com/
- http://microformats.org/
- https://www.astuteo.com/slickmap/
- https://www.euromost.info/
- http://sibagraphics.com/utilities/the-meaning-of-colours/
- https://en.wikipedia.org/wiki/Web_content#Content_is_king
- http://rubyonrails.org/
- https://www.w3.org/TR/2012/REC-WOFF-20121213/
- https://www.sitepoint.com/community/
- https://www.sitemaps.org/protocol.html
- http://humaan.com/checklist/

Ultimate Guide to Microformats: Reference and Examples

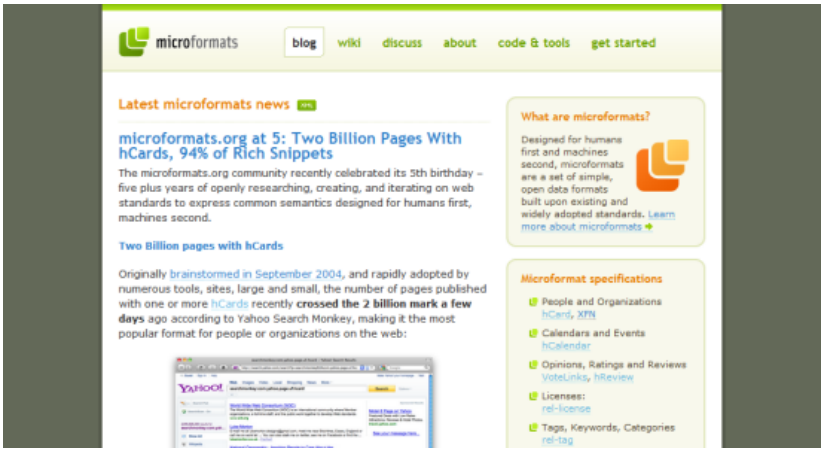
If you're not familiar with the concept of POSH (plain old semantic HTML), the first thing to know is that producing semantic code that reflects content contextually (rather than stylistically) is a critical component of the web design process. While HTML has a whole bunch of awesome elements by which to convey meaning, a slew of purpose-built microformats (conventions) have been created to better represent the kind of content that exists on the page. This guide discusses popular microformats that can enhance the semantics and interoperability of your website.

What Are Microformats?

Microformats are pretty interesting if you give them a chance. While they aren't a component of the W3C HTML spec, they do offer a valuable and useful set of naming conventions (using class, id, rel and rev attribute values) that identify points of interest on the page, such as calendar events, links to the content's license agreement, and even quirky things such as cooking recipes.

While microformats are not a W3C standard yet–though many microformats either have been recommended to the W3C as standards or are in draft form–the level of support browsers and web services have for them explains their utility.

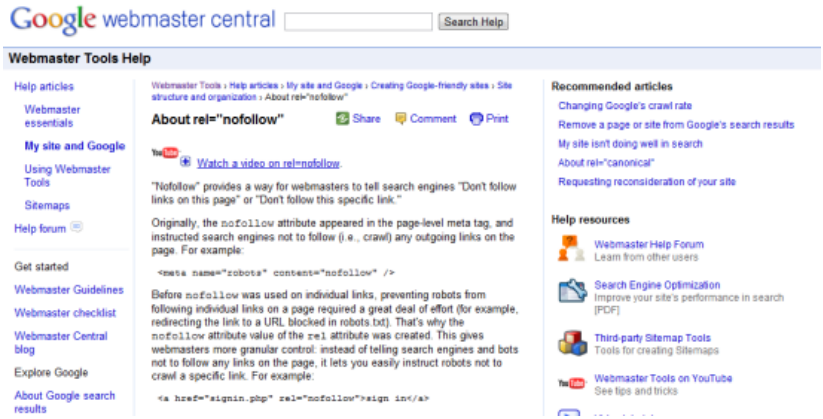
Simply put: microformats are worth learning about and implementing into the websites you build.



The official microformats website has a community wiki, discussion board, and tools for you to use.

You might already be using microformats if you use a CMS like WordPress, because it has built-in support for some simpler forms of data, such as the rel attribute.

If you're new to microformats, then you're probably wondering why you should bother using them. Well, they have a number of pros and cons, but anything that would help our websites be better understood by external machines–such as web spiders that index our web pages–is worth the extra effort.



Of the many microformats, rel="nofollow" is probably the best known.

Because microformats use conventional HTML syntax and attributes, you can use them in XHTML. Even XML pages (such as in RSS and Atom feeds) can leverage microformats (and they do, if you use a service like Feedburner). This dramatically increases their potential use. They also combine well with RDFa and other meta data.

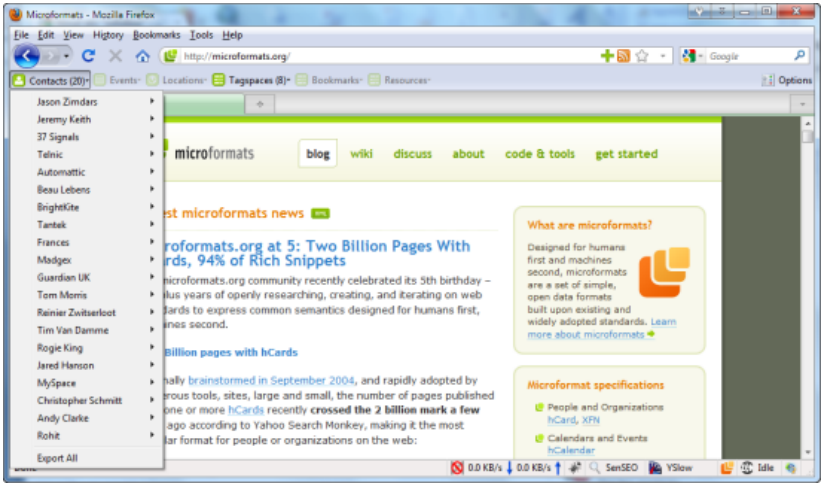
Benefits of Using Microformats

- They will improve the semantic value of your content.
- Web apps can use them to discover data about your website; they can use them to interface with data on your site.
- Social networks are implementing them in user profiles so third-party web services can interoperate with them.
- Browser extensions exist to give users access to microformat data. For example, Michromeformats is a Google Chrome extension that discovers embedded microformats on a web page.
- Web spiders like Googlebot make use of them in site indexing.

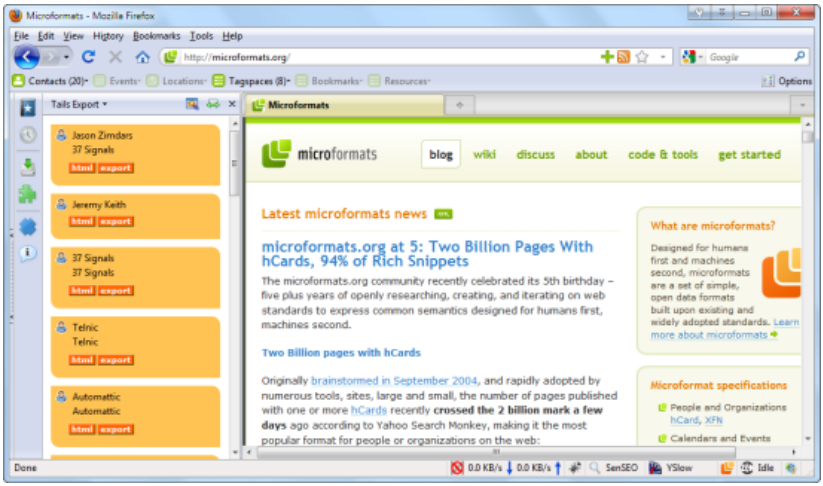
Drawbacks of Microformats

- They require additional HTML markup.
- They're yet another thing you'll have to learn and maintain.

- Microformats exist for relatively few data types.
- They draw attention to your data (which can be mined).
- Web browsers do not support them uniformly.



The Operator add-on for Firefox detects microformats code and makes them human-readable.



The hCard microformat allows Firefox add-ons like Tails Export to discover and interface with a person's virtual business card.

Microformats Reference Table

Each microformat has a unique purpose for presenting a certain type of information, and they could all be potentially useful depending on your needs.

While extensive details can be found in the specifications on the microformats website, below is a quick reference listing of what exists.

ADR	Marks a street address
FOAF	Describes a relationship to another website
Geo	Marks a geographic location
hAtom	Adds syndication-friendly content
hAudio	Describes audio or a podcast
hCalendar	Marks up event or date-based content

hCard	For business and personal contacts
hListing	Listings of goods and services
hMedia	Lists media references
hNews	Uses hAtom for journalistic news
hProduct	Embeds extensive product details
hRecipe	Marks up recipes and cooking data
hResume	To showcase a CV or resume
hReview	Reviews and ratings of products and services
hSlice	Pops up internal or external subscription windows in IE8
rel	The rel attribute is a microformat for HTML elements; some popular examples: • rel="license" • rel="nofollow" • rel="tag" • rel="directory" • rel="enclosure" • rel="home" • rel="payment"
Robot Exclusion Profile	Gives web crawlers instructions
VoteLinks	Provides options to like or dislike a link
XFN	Describes a relationship to a website
XFolk	Lists favorite links

XMDP	Adds resources to the page's profile
XOXO	Outlines a document or list of items

rel Attribute Values

To expound on the table above, here are descriptions of the rel (which is short for "relationship") attribute values:

- **License:** identifies a license agreement (such as Creative Commons or GPL) on a page.
- **Nofollow:** tells search engines not to add weight or value to the linked resource.
- **Tag:** applies keywords to anchors in order to build tag clouds or categories.
- **Directory:** indicates a listing in a directory (such as a folder) on the current website.
- **Enclosure:** for anchors that link to downloadable files and other non-web documents.
- **Home:** produces a permalink to the home page of a website.
- **Payment:** to be included in anchors that point to a purchasing or payments page.

Using Microformats: Examples

Recommending microformats does little good without providing illustrations on how to use them. So, here we'll go over examples of each microformat that can be implemented into your website.

First, the key concept to understand is that a microformat is identified by a piece of data contained in the class or id value of an HTML element.

The element could play a role in the type of data being displayed (such as with anchor links), but if no semantic alternative exists, you could use a div or span to wrap the name around the content. Although using span might seem inelegant, it adds special meaning in this case.

Adr

```
<ul class="adr">
    <li class="street-address">123 North Street</li>
    <li class="locality">Manchester</li>
    <li class="postal-code">MX43 991</li>
    <li class="country-name">UK</li>
</ul>
```

Root name: adr

Attribute values:

- post-office-box
- extended-address
- street-address
- locality
- region
- postal-code
- country-name

FOAF

How to create a FOAF profile:

1. Visit FOAF-o-Matic and create your basic profile.
2. Save the document as foaf.rdf (so that you know what it's for) in a directory (perhaps named something like misc).
3. Use the link tag to reference your FOAF profile inside the <head> of your HTML documents, for example: <link rel="meta" type="application/rdf+xml" title="FOAF" href="foaf.rdf" />
4. Upload all of the FOAF-related files to your website. It's now ready to be used and indexed!

Geo

```
<p class="geo">
    <abbr class="latitude" title="37.408183">N 37° 24.491</abbr> -
    <abbr class="longitude" title="-122.13855">W 122° 08.313</abbr>
</p>
```

Root name: geo

Required attribute values:

- latitude
- longitude

hAtom

```
<div class="hAtom">
    <div class="hentry">
        <h3 class="entry-title">I Love Microformats</h3>
        <abbr class="published"
        title="2010-08-28T13:14:37-07:00">Aug 28, 2010</abbr>
        <p class="category"><a href="/category/rdf" rel="tag">RDF</
        a></p>
        <p><a href="#" title="Post a comment">What do you think of
        this post?</a></p>
        <div class="entry-content">
            <p>Place your content right here for maximum impact!</
            p>
        </div>
        <dl>
            <dt>Tags:</dt>
            <dd><a href="/tag/standards/" rel="tag">standards</a></
            dd>
            <dd><a href=" /tag/microformats/"
            rel="tag">microformats</a></dd>
        </dl>
    </div>
</div>
```

Root name: hAtom, hFeed

Attribute values:

- hentry
- entry-title
- entry-content
- entry-summary
- bookmark
- published
- updated
- author

hAudio

```
<p class="haudio">
    <em class="fn">Bohemian Rhapsody</em>
    by <span class="contributor vcard">
    <em class="fn org">Queen</em></span>
    found on <em class="album">A Night at the Opera</em>
</p>
```

Root name: hAudio

Required attribute values:

- fn
- album

Other attribute values:

- contributor
- duration
- item
- position
- category
- published

- photo
- description
- sample
- enclosure
- payment
- price (currency, amount)

hCalendar

You can use the hCalendar Creator instead of writing the code manually.

```
<p class="vEvent">
    <a class="url" href="http://www.yoursitehere.com/">MySite</a>
    <span class="summary">New website launch</span>:
    <abbr class="dtstart" title="20091202">December  2</abbr>-
    <abbr class="dtend" title="20091204">4</abbr>, at
    <span class="location">Google College, London, UK</span>
</p>
```

Root name: vCalendar, vEvent

Required attribute values:

- dtstart
- summary

Other attribute values:

- location
- url
- dtend
- duration
- rdate
- rrule
- category
- description

- uid
- geo (latitude, longitude)
- attendee (partstat, role
- contact
- organizer
- attach
- status

hCard

You can use the hCard creator instead of writing the code manually.

```
<ul id="hCard-John-Doe" class="vcard">
    <li class="fn">John Doe</li>
    <li class="org">Special Stores</li>
    <li><a class="email"
    href="mailto:John@doe.org">John@doe.org</a></li>
    <li class="adr">
        <ul>
            <li class="street-address">44 Semantic Drive</li>,
            <li class="locality">Markup City</li>,
            <li class="region">World Wide Web</li>,
            <li class="postal-code">BP33 9HQ</li>
            <li class="country-name">Internet</li>
        </ul>
    </li>
    <li class="tel">01234 56789</li>
</ul>
```

Root name: hCard

Required attribute values:

- fn
- n (family-name, given-name, additional-name, honorific-prefix, honorific-suffix)

Other attribute values:

- adr (post-office-box, extended-address, street-address, locality, region, postal-code, country-name, type, value)
- agent
- bday
- category
- class
- email (type, value)
- geo (latitude, longitude)
- key
- label
- logo
- mailer
- nickname
- note
- org (organization-name, organization-unit)
- photo
- rev
- role
- sort-string
- sound
- tel (type, value)
- title
- tz
- uid
- url

hListing

```
<div class="hlisting">
    <p>
        <span class="item fn">Office space</span>
        <span class="offer rent">to rent</span>(<abbr
        class="dtlisted" title="20100202">2/2/10</abbr>)
    </p>
    <p class="description">50-square-foot space available in local
    tech office at:
        <div class="location adr">
            <span class="street-address">123 Microland Road.</span>
            <span class="locality">Cyberspace</span>, <span
            class="region">XD</span>
            <span class="postal-code">12345</span>
            <span class="country">Mars</span>
        </div>
        Available during <abbr class="dtexpired"
        title="20100401">April 2010</abbr>
        for <span class="price">$1500/qtr</span>
    </p>
    <div class="lister vcard">Contact:
        <span class="fn">John Doe</span> at <span
        class="tel"><span class="value">(01) 12345-678900</
        span>(<abbr class="type" title="cell">C</abbr>)</span>
    </div>
</div>
```

Root name: hlisting

Required attribute values:

- description
- lister (fn, email, url, tel)
- action (sell, rent, trade, meet, announce, offer, wanted, event, service)

Other attribute values:

- version
- dtlisted
- dtexpired
- price
- item (fn, url, photo, geo, adr)
- summary
- tag
- permalink

hMedia

```
<div class="hmedia">
    <h3 class="fn">Introduction to the Open Media Web</h3>
    <object class="player" type="application/x-shockwave-flash"
    data="http://www.exampleurl.com/video.swf">
        <param name="movie" value="http://www.exampleurl.com/
        video.swf"/>
        <param name="allowScriptAccess" value="always"/>
        <param name="allowFullScreen" value="true"/>
    </object>
    <ul>
        <li><a rel="enclosure" type="video/mp4" title="Download
        the movie" href="http://www.exampleurl.com/
        video.mp4">Video.mp4</a></li>
    </ul>
</div>
```

Root name: hMedia

Attribute values:

- fn
- contributor
- photo
- player
- enclosure

hNews

```
<div class="hnews hentry item">
    <h4 class="entry-title">Microformats are awesome</h4>
    <p class="author vcard">
    By <span class="fn" >John Doe</span>,
    <span class="source-org vcard dateline"><span class="org
    fn">Associated Press</span></span> -
    <span class="updated" title="2010-04-19">19 April 2010</p>
    <p>News story</p>
</div>
```

Root name: hNews

Required attribute values:

- hentry
- item
- entry-title
- author
- source-org
- vcard
- updated

Other attribute values:

- dateline
- geo (latitude, longitude)
- item-license
- principles

hProduct

```
<ul class="hproduct">
    <li class="brand">MySite!</li>
    <li class="category">Software</li>
    <li class="fn">Microsoft Office 2007</li>
    <li class="description">The world's most popular office suite.</
    li>
    <li class="url">http://office.microsoft.com</li>
</ul>
```

Root name: hProduct

Required attribute value:

- fn

Other attribute values:

- brand
- category
- price
- description
- photo
- url
- review
- listing
- identifier (type (model, mpn, upc, isbn, issn, ean, jan, sn, vin, sku), value)

hRecipe

```
<div class="hrecipe">
    <h3 class="fn">Quick noodles</h3>
    <p class="summary">Noodles are quick and easy, like this
    example!</p>
    <p class="ingredient hcard"><span class="value">2.5</
    span><span class="type">kilogram</span>bag of instant
    noodles.</p>
    <ul class="instructions">
        <li>Put water on to boil,</li>
        <li>Add the powder for the sauce,</li>
        <li>Add the noodles, and stir till ready.</li>
    </ul>
    <p>Enough for <span class="yield">1 adult</span>.</p>
    <p>Preparation time is approximately <span class="duration">5
    <abbr title="minutes">mins</abbr></span>.</p>
    <p class="nutrition hcard">Noodles have more than <span
    class="value">500</span> <span class="type">joules</span> of
    energy.</p>
</div>
```

Root name: hRecipe

Required attribute values:

- fn
- ingredient (value, type)

Other attribute values:

- yield
- instructions
- duration
- photo
- summary
- author
- published
- nutrition (value, type), tag

hResume

You can use the hResume creator instead of writing the code manually.

```
<div id="hResume">
    <p class="summary">I have been producing microformatted
    data for years</p>
    <ul class="vcard">
        <li class="fn">Jane Doe</li>
        <li class="adr">
        <span class="street-address">44 Broadband Street</span>
        <span class="locality">Microland</span>, <span
        class="region">Internet</span>
        <span class="postal-code">QW11 ER4</span></li>
        <li>Email: <a class="email"
        href="mailto:jane@doe.org">jane@doe.org</a></li>
        <li>Homepage: <a class="url" href="http://
        www.yoursitehere.com/">www.yoursitehere.com</a></li>
        <li>Phone: <span class="tel">+44 12345 67890</span></li>
    </ul>
    <ol class="vcalendar">
        <li class="education vevent"><a class="url summary"
        href="http://example/">Example</a>(<abbr class="dtstart"
```

```
            title="2007-02-11">2007</abbr> - <abbr class="dtend"
            title="2009-03-22">2009</abbr>)</li>
        </ol>
        <ol class="vcalendar">
            <li class="experience vevent"><span
            class="summary">CEO</span>, <span
            class="location">Microland</span>, <abbr class="dtstart"
            title="2006-09-01">May 2006</abbr> - <abbr
            title="2009-05-22">present</abbr></li>
        </ol>
        <ul class="vcard">
            <li><a href="/jdoe/index.php" class="include" title="Jane
            Doe"></a></li>
            <li class="org">MicroLand</li>
            <li class="title">CEO</li>
        </ul>
        <p>I have skills in
            <a class="skill" rel="tag" href="http://en.wikipedia.org/wiki/
            HTML">HTML</a> and
            <a class="skill" rel="tag" href="http://en.wikipedia.org/wiki/
            CSS">CSS</a>.
        </p>
    </div>
```

Root name: hResume

Required attribute value:

- contact (hCard + adr)

Other attribute values:

- summary
- education (hCard + vEvent)
- experience (hCard + vEvent)
- affiliation (hCard)
- skills
- publications

hReview

You can use the hReview creator instead of writing the code manually.

```
<div class="hreview">
    <p><span class="rating">5</span> out of 5 stars</p>
    <h4 class="summary">Noodle Hut</h4>
    <span class="reviewer vcard">Reviewer:
    <span class="fn">John Doe</span> - <abbr class="dtreviewed"
    title="20070418T2300-0700">April 18, 2007</abbr>
    </span>
    <p class="description item vcard">
        <span class="fn org">Noodles Hut</span> is one of the best
        little places out there!
    </p>
    <ul>
        <li>Visit date: April 2007</li>
        <li>Food eaten: Instant noodles</li>
    </ul>
</div>
```

Root name: hReview

Required attribute value:

- item (type (product, business, event, person, place, website, url), hCard / hCalendar)

Other attribute values:

- reviewer (hCard)
- version
- summary
- dtreviewed
- rating
- description
- tags
- permalink
- license

hSlice

```
<div class="hslice" id="news">
    <h2 class="entry-title">Recent News</h2>
    ...
</div>
```

Root name: hSlice

Required attribute values:

- ID
- entry-title

Other attribute values:

- entry-content
- end-time
- ttl
- feedurl

Rel

```
<a rel="license" href="http://creativecommons.org/licenses/by/
2.0/">Some rights reserved.</a>
<a rel="nofollow" href="http://www.w3.org/">World Wide Web
consortium</a>
```

Attribute values:

- license
- nofollow
- tag
- directory
- enclosure
- home
- payment

Robot Exclusion Profile

```
<head profile="http://example.org/xmdp/robots-profile#">
</head>
...
<p><img src="exclusive.png" class="robots-noarchive"
alt="Exclusive image" /></p>
```

Attribute values:

- robots-nofollow
- robots-follow
- robots-noindex
- robots-index
- robots-noanchortext
- robots-anchortext
- robots-noarchive
- robots-archive

VoteLinks

```
<a rev="vote-for" href="http://www.yoursitehere.com/vote.php?
id=yes" title="Vote yes!">Vote Yes!</a>
<a rev="vote-abstain" href="http://www.yoursitehere.com/
vote.php?id=maybe" title="Vote maybe!">Vote Maybe!</a>
<a rev="vote-against" href="http://www.yoursitehere.com/
vote.php?id=no" title="Vote no!">Vote No!</a>
```

Attribute values:

- vote-for
- vote-abstain
- vote-against

XFN

You can use the XFN creator instead of writing the code manually.

```
<a href="http://www.yoursitehere.com" rel="me">My Site!</a>
```

Attribute values:

- Friendship (contact, acquaintance, friend)
- Physical (met)
- Professional (co-worker, colleague)
- Geographical (co-resident, neighbor)
- Family (child, parent, sibling, spouse, kin)
- Romantic (muse, crush, date, sweetheart)
- Identity (me)

xFolk

```
<ul>
    <li>
        <ul class="xfolkentry">
            <li><a class="taggedlink" href="http://www.google.com"
            title="Google">Google</a></li>
            <li class="description">The home page of the world's
            biggest search engine</li>
            <li class="meta">Tags:<a rel="tag" href="http://del.icio.us/
            tag/google">google</a><a rel="tag" href="http://
            del.icio.us/tag/search">search</a></li>
        </ul>
    </li>
</ul>
```

Root name: xFolkEntry

Required attribute values:

- description
- taggedlink
- title

Other attribute value:

- meta (tag)

XMDP

```
<head profile="http://www.mysitehere.com/profilename">
```

Root reference: profile

XOXO

```
<ol class="xoxo">
    <li>Subject 1
        <ol>
            <li>item a</li>
            <li>item b</li>
        </ol>
    </li>
    <li>Subject 2
        <ol>
            <li>item a</li>
            <li>item b</li>
        </ol>
    </li>
</ol>
```

Root name: XOXO

Conclusion

Plenty of microformats already exist, and the community is always looking for ways to use existing elements to convey more information about our web pages. They benefit not only search engines and social networks, but the users that traffic our site.

Sources:

- https://en.wikipedia.org/wiki/Semantic_HTML
- http://microformats.org/
- https://support.google.com/webmasters/answer/96569?hl=en
- https://chrome.google.com/webstore/detail/microformats/oalbifknmclbnmjlljdemhjjlkmppjjl
- https://mike.kaply.com/operator/
- https://addons.mozilla.org/en-US/firefox/addon/tails-export/
- http://www.foaf-project.org/

- https://msdn.microsoft.com/en-us/library/cc304073%28VS.85%29.aspx
- http://gmpg.org/
- http://www.ldodds.com/foaf/foaf-a-matic
- http://microformats.org/code/hcalendar/creator
- http://microformats.org/code/hcard/creator
- http://microformats.org/code/hreview/creator
- http://gmpg.org/xfn/creator

Becoming a Better Web Designer

Whenever a student wanting to work in the design industry asks me for advice, the first thing that comes to my mind is the importance of maintaining a current and up-to-date skill set. Often, we spend so much time focusing on the actual jobs at hand that we neglect to nourish and refresh our knowledge.

This article aims to highlight the importance of setting aside time for self-improvement, with recommendations on how you can keep learning to stay ahead of the curve.

A Need to Improve

While many people may feel that they can't afford (or have the time) to put aside some time in order to learn and keep up with industry developments, I would argue that they also can't afford the losses in competitiveness due to having outdated knowledge.

Often, the individuals who stick with what they know are the ones finding themselves out of jobs because the nature of the profession, the technologies, and the medium is such that it evolves quickly. Conventions change, web languages are updated or become obsolete, deployment methods shift from technology to technology. It's survival of the fittest, with "fit" being defined as someone who keeps up with current demands.

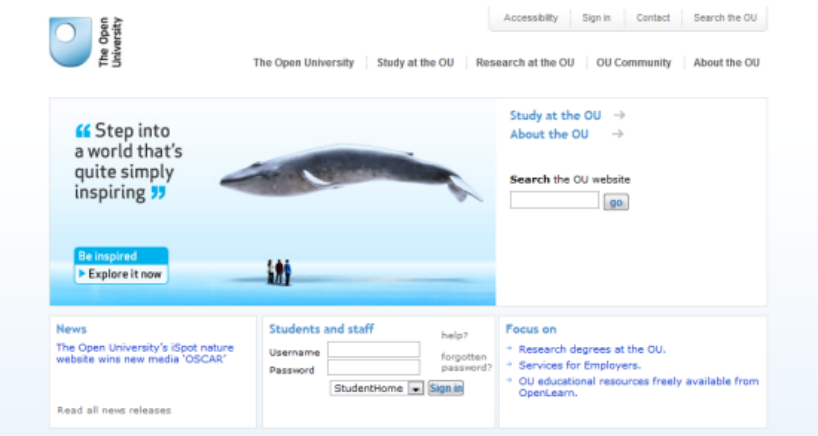
Imagine if Yahoo! had rested on its laurels.

Learning new skills doesn't have to be complicated, and it doesn't even have to be expensive. And in an industry that is always changing — and where becoming stale means being unable to perform — the need for continual self-education and self-training is very important.

With all of that in mind, I'd like to suggest a few ways we as web designers can keep ourselves current.

Education

The first thing I'll mention is probably the most expensive in terms of money and time. However, it's worth discussing because it is the traditional way of learning a particular profession. Education is probably the most obvious route to self-improvement. You can find courses on pretty much anything — from graphic design to ancient Mediterranean studies — these days; courses tailored to your exact interests and needs.



The Open University is based in the UK and offers plenty of structured courses.

The benefits of sitting in a classroom are quite apparent if you thrive on structured learning plans. The experience of being in a classroom also means you'll get to hang out with peers who'll be in the same position.

The problems with education, especially in very traditional colleges, is that courses might be outdated and can cost quite a lot.

Education Resources:

- O'Reilly School - http://www.oreillyschool.com/
- The Top Online Web Design Degrees - http://www.webdesignschoolsguide.com/
- Design Schools and Design Degrees Online Directory
- CIW Certified - https://www.ciwcertified.com/
- WaSP InterAct
- Full Sail University Web Design & Development Bachelor's Degree – Online - https://www.fullsail.edu/degrees/web-design-and-development-bachelor

Internships

In terms of getting an internship at a design agency, the learning you get is often practical. You can get to see how professionals handle their job tasks, and learn through hands on experience.

Even well known agencies might have an intern position available to you.

Often, we learn a lot through observation. If you're a beginning freelancer or a student just finding his way through the world, getting into a proper business as an intern will give you not only the guiding hands and experience of professionals, but also potential job offers if you prove yourself to be a good fit for the company.

Potential downsides are that internships may be unpaid (voluntary). An internship is also a real commitment in both time and effort. However, the experience may be truly rewarding if you are a committed individual.

Also, with an internship, the value you get depends on who you'll be working for and what they'll task you with. If you get stuck in an internship that only deals with activities such as running to Starbucks to get the design team their coffee or sorting out mail, then you would get less value, learning-wise, than in an internship that gets you right in the mix of things and interacting with professional designers. That's one thing to keep in mind when searching for an internship.

Job Boards for Finding an Internship Resources:

- Sensational Jobs - https://www.sensationaljobs.com/
- Smashing Jobs - https://www.smashingmagazine.com/jobs/
- Authentic Jobs - https://authenticjobs.com/
- Krop Jobs - https://www.krop.com/creative-jobs/

Conferences and Workshops

Conferences and workshops can be found all over the world, covering topics ranging from broader topics such as Web Design to more specialized fields like UX and IA. There are even dedicated events and meetups to particular technologies such as JavaScript or Adobe Flash.

All over the world, you'll find a range of conferences at different budgets and topics (such as The Future of Web Design).

A conference is a time-honored tradition for many designers and developers in which a bunch of caffeine-intoxicated geeks gets together to learn from industry experts and each other. While these events can be particularly expensive, they usually cover a wide range of subjects and can allow you the time to speak to people you respect, make a few friends and, in some cases, get some extra business. If you're socially inclined, it can turn into a great few days of learning, sharing and socializing.

Conference Resources:

- Upcoming Conferences 2010 - http://www.smashingmagazine.com/2010/09/09/upcoming-web-design-and-development-conferences-in-2010/
- Web Events - http://www.d.umn.edu/itss/support/Training/Online/webdesign/events.html
- List of conferences - http://djdesignerlab.com/2010/03/24/web-conferences-and-events-every-professional-must-attend/

Over recent years, workshops have really evolved into something quite special. While many exist in which you visit a particular venue and learn about a subject from an industry guru, plenty of websites

now also offer web-based workshops that allow you to learn from the comforts of your own workspace.

While you won't get a certificate or qualification (as you would through a college), the benefits from these workshop sessions are that you'll get a ton of knowledge on a specialist subject in a short period of time and in a structured manner.

Workshop Resources:

- IWA-HWG eClasses - http://iwa-hwg.eclasses.org/
- Workshops for the Web - http://workshopsfortheweb.com/

Networking and Mentoring

Getting to know your fellow designers and developers is a big part of being involved in the industry. It's surprising to see how many people isolate themselves from the design and development community and thereby fail to get the benefits of networking (and mentoring).

Getting your name out there socially will give you a range of benefits, not just to your social life, but also in terms of boosting your own knowledge and being able to assist others in improving their own work.

Social networking is quite a simple idea. You join a community — whether it's offline such as a chamber of commerce group or online like a forum or social networking service like Twitter — and then you interact with other people!

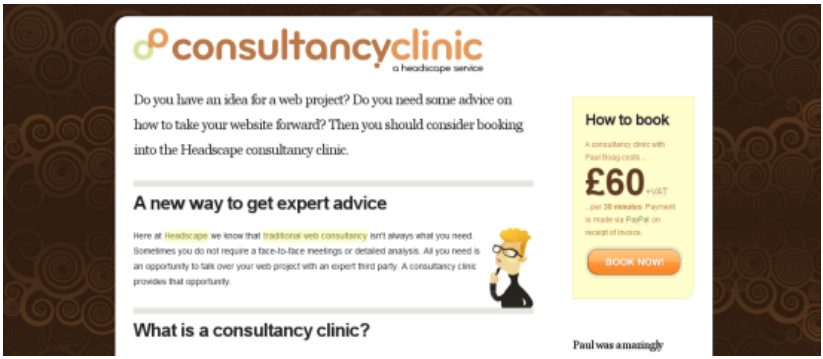
Not only will you make friends and gain useful industry contacts, but also, you can learn plenty from other people's experiences. Even better is that you aren't forced to dedicate a set amount of time to networking; you can participate in networking on your own time.

Networking Resources:

- Web Standards Group - http://www.webstandardsgroup.org/
- SitePoint Forums - https://www.sitepoint.com/community/
- CSS Discuss - http://www.css-discuss.org/
- Designers Talk - http://www.designerstalk.com/forums/

- How Social Media Works

Mentoring is much like networking in terms of the social interaction with other people, but it does give you a slightly different route to learning.



Some charge for consultancy services and other people mentor for free... both work!

Finding someone to mentor may seem beneficial just to the individual being mentored, but when you think about it, the student is going to be testing your knowledge and sharing their thoughts with you. This gives you motivation to keep your knowledge up to date, as well as discover things you might not have thought about. As they say, the best way to learn is to teach.

Mentoring Resources:

- Find a business mentor
- Finding mentors

Books, Blogs, Articles, Podcasts and Videos

If you're a bit shy and don't really like the idea of going back to school or being in a room with other people, consuming books, articles, eBooks, videos and slideshows may be a perfect method for self-betterment.

This method of gaining new knowledge is by far the most prevalent. There are literally thousands of cool titles and resources sitting out there waiting to be purchased, read and watched, as well as a lot of free content too!

You are here... and while Six Revisions is not a book, it'll still teach you oodles of useful information.

Books are probably the most widely recognized resource when it comes to learning about a specific technology, and there are plenty of articles and resources online. With millions of titles to choose from on just about every subject you can imagine, they can be a reasonably priced, go-at-your-own-pace alternative of learning.

Blog and Website Resources:

- A List Apart - http://alistapart.com/
- Smashing Magazine - https://www.smashingmagazine.com/
- Six Revisions (of course)
- Web Designer Deport - https://www.webdesignerdepot.com/

- UX Booth - http://www.uxbooth.com/
- UX Magazine - http://uxmag.com/
- Onextra Pixel - http://www.onextrapixel.com/
- FreelanceSwitch - https://studio.envato.com/freelance-switch/

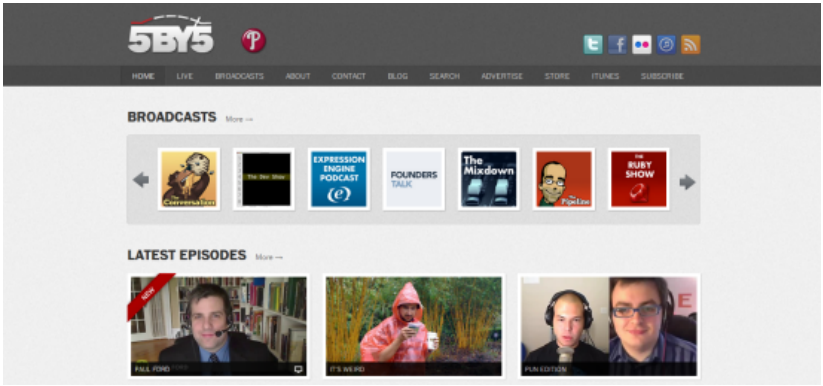
eBook and Article Resources:

- Adobe Typography Primer - https://blogs.adobe.com/typblography/tag/typography-primer
- Art and Science of Web Design, The - http://www.veen.com/jeff/archives/000747.html
- Building Accessible Websites - https://joeclark.org/book/sashay/serialization/
- Designing For The Web - https://designingfortheweb.co.uk/
- Designing Interfaces - http://designinginterfaces.com/
- Dive Into Accessibility - https://github.com/nfreear/diveintoaccessibility
- Elegant Web Typography - https://www.slideshare.net/jeff_croft/elegant-web-typography-presentation
- Elements of Typographic Style For The Web, The - http://webtypography.net/
- Eloquent JavaScript - http://eloquentjavascript.net/
- Getting Real - https://basecamp.com/books/getting-real
- Google SEO Guide - https://support.google.com/webmasters/answer/7451184?hl=en
- Guide to Guerrilla Freelancing, A - http://temza.com/e-books/Guerrilla-Freelancing.pdf
- How Do You Design (Beta)? - http://www.dubberly.com/articles/how-do-you-design.html
- Introduction to Good Usability - https://temza.com/e-books/introduction-to-good-usability.pdf
- jQuery Fundamentals - http://jqfundamentals.com/legacy/

- KnockKnock - http://sethgodin.typepad.com/seths_blog/2005/09/free_ebook_1_no.html
- Meet Your Type - http://www.optimiced.com/wp-uploads/2010/10/meet-your-type-guide-by-fontshop.pdf
- Mobile Web Developers Guide (dotMobi) - https://www.networksolutions.com/help/mobi-guide.pdf
- Opera Standards Curriculum - https://webplatform.github.io/
- PeachPit WDRG
- Programmers Intro to PHP4 - http://web.archive.org/web/20070412121056/http:/apress.com/free/content/ProgrammersIntroductionToPHP4.pdf
- Programming VB.NET - http://computer-books.us/vb_0004.php
- Search User Interfaces - http://searchuserinterfaces.com/book/
- Software Engineering for Internet Apps - http://philip.greenspun.com/seia/
- Task-Centered UI Design - http://hcibib.org/tcuid/
- Teach Yourself JavaScript in 24 Hours (Sams) - https://web.archive.org/web/20160314103602/http://www.informit.com/library/library.aspx?b=sty_javascript_24_hours
- Teach Yourself HTML 4 in 24 Hours (Sams) - https://web.archive.org/web/20160310185016/http://www.informit.com/library/library.aspx?b=STY_html_24hours
- Time Management for Creative People - https://www.wishfulthinking.co.uk/2007/12/03/time-management-for-creative-people-free-e-book/
- Type Classification E-Book - http://justcreative.com/featured-articles/type-classification-ebook/
- UIAccess: Just Ask! Integrating Accessibility Through Design - http://uiaccess.com/accessucd/contents.html
- Usability.gov – The Guidelines - https://guidelines.usability.gov/

- Vignelli Canon, The - http://www.vignelli.com/home/bookmagazine/canon.html
- Web Book, The
- Web Designers Success Guide - http://airgid.com/wp-content/uploads/2012/06/wdsg_fitc.pdf
- Web Style Guide - http://webstyleguide.com/wsg3/
- Website Migration Handbook, The - https://davidhobbsconsulting.com/migration_handbook
- Why Design? (AIGA)
- Woork Handbook, The - https://protuts.net/wp-content/uploads/wordpress-ebook-the-woork-handbook.pdf
- XML Programming - http://web.archive.org/web/20070411121620/http:/www.apress.com/free/content/xmlprogramming.pdf

If you're not one for going through pages of text in order to learn, and you prefer something that engages many of your senses, perhaps you might consider the range of audio and video podcasts that exist for designers and developers.



Podcasts like those provided by the 5×5 network offer design and development fun!

While not on many people's radars, these shows give you industry news and great advice for free! In addition, there's plenty of paid and free slideshows and videos which can train you in a particular craft while visually orientated and worthy of consideration.

What's convenient about podcasts, especially audio podcasts, is that you can listen to them while commuting to work.

Podcast Resources:

- Web Hosting Show - https://www.webhostingshow.com/
- Web Axe - http://www.webaxe.org/category/podcast/
- Web Dev Radio - http://webdevradio.com/
- Web 2.0 Show - http://web20show.com/
- Web Designer Magazine - https://www.listennotes.com/bg/podcasts/web-designer-podcasts-imagine-publishing-epvIRL54wDX/
- Think Vitamin Radio - https://podcasts.apple.com/gb/podcast/think-vitamin-radio/id351340191
- User Interface Engineering - https://uie.fm/shows
- This Week in Start-ups - http://feeds2.feedburner.com/twist-audio
- SitePoint Podcast - https://www.sitepoint.com/web/podcast/
- Boagworld - https://boagworld.com/show/
- 5×5 Network - http://5by5.tv/
- TWiT.TV - https://twit.tv/
- CNET Podcasts - https://www.cnet.com/g00/cnet-podcasts/?i10c.encReferrer=&i10c.ua=1&i10c.dv=14

Video Resources:

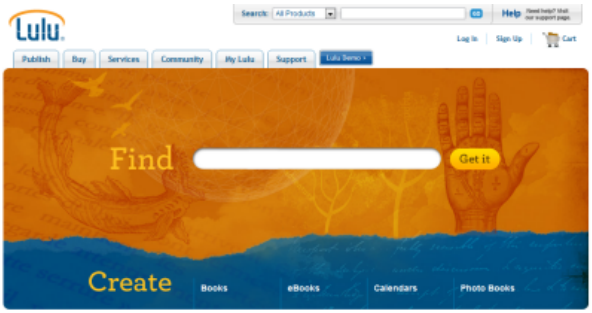
- Lynda Training - https://www.lynda.com/
- Slideshare Presentations - https://www.slideshare.net/

- Must See Design Videos - https://www.smashingmagazine.com/2016/03/never-stop-learning-with-live-streams-and-conference-videos/
- Think Vitamin Videos - https://teamtreehouse.com/library
- SitePoint Videos - https://www.sitepoint.com/premium/

Engaging in Side Projects

A final route worthy of consideration is learning through side projects. Taking on a side project allows you to learn and sharpen your skills by doing something fun. It's not a secret that personal projects are good for you.

Consider having a side project to help you learn at your own pace. Your side project can be pro bono design work for an organization you're passionate about or building a web app that solves a problem (and if it's good, who knows, it might become profitable). You could start a blog, create a podcast, self-publish your own books or eBooks, provide tutorials — all of which can reinforce the things you learn day-by-day.



The ability to publish your own book through services such as Lulu has become quite popular!

While side projects may arguably be among the more ambiguous, indirect ways to learn, it's also true that theory you learn from books and classrooms will only take you so far and that practical experience has its own benefits.

Never Stop Learning

Surviving and succeeding in this industry requires constant upkeep of the things you already know. If there's one thing that I admire above everything else, it's the enthusiasm and craving for knowledge that many of us still manage to maintain after working in this fast-paced environment for years. If you only dedicate a couple of hours a week to self-improvement, that's still better than nothing.

In any business, you don't want to become the weak link in the chain, and setting yourself up with some long-term goals will help you avoid becoming the individual whose work has the reputation of being outmoded.

There is always something new to learn, and those awesome new skills can bring more value to you and the people you work with/for. So go out and buy that book you saw on Amazon.com, learn that new web language you've been putting off for too long, read that new blog everyone keeps talking about, sign up for that workshop, or watch that video — whatever you do, just never stop learning!

Sources:

- http://www.open.ac.uk/
- https://thisibelieve.org/essay/12278/
- https://www.lulu.com/

Ways to Horrify Website Designers

Most people love a good scare. That moment where you almost jump out of your skin can pump you full of adrenaline and get your senses heightened.

Unfortunately, while zooming through a theme park ride at epic speeds or watching Michael Myers chase Jamie Lee Curtis with a knife will give us a "fun" type of scare, the web — whether by design or sadism — tends to be full of the kind of scary traps that would make the Jigsaw Killer's creepy puppet giggle with glee.

Horrors on the Web

As you may have guessed, I'm a fan of horror movies. I could happily spend a few hours watching Sadako from The Ring franchise scare people to death. However, as we all know, the real world can be just as terrifying (or more so) than the world of movies.

While I love the medium of the web, and the awesome things we can do with it, a few things about it chill me to the bone.

Internet Explorer 6 or Michael Myers: both shorten your lifespan!

Much of the web's horrors usually arises through no fault of the site designer and are often misguided attempts at solving a particular requirement for a site. In many cases, people implement such spooky

site features to draw the attention of site visitors or to add something that the site owner thinks is cool.

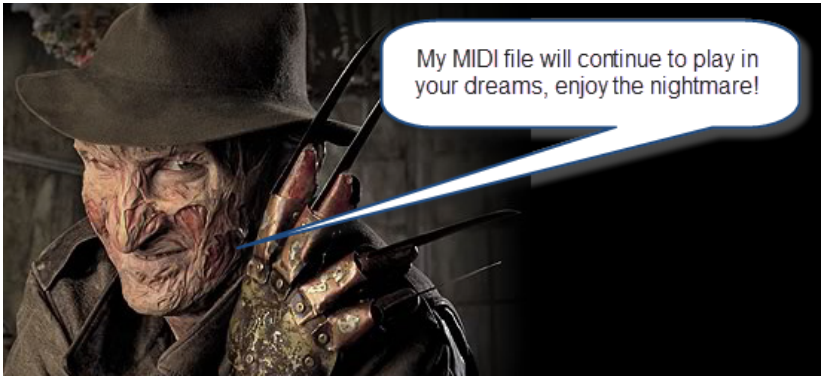
Let's take a look at some scary things the web has to offer.

Automatically Playing Music

Of the web's many horrors, automatically playing music has to be high on the list of terrifying web crimes.

Imagine that you're sitting in the office and have forgotten that your speakers were cranked up from showing co-workers a funny YouTube video.

A few hours later, you wonder onto a site you've never visited before. Suddenly, Bohemian Rhapsody blasts through your ears, nearly shattering your eardrums. That incident might have sent coffee into your keyboard or given you a heart attack.



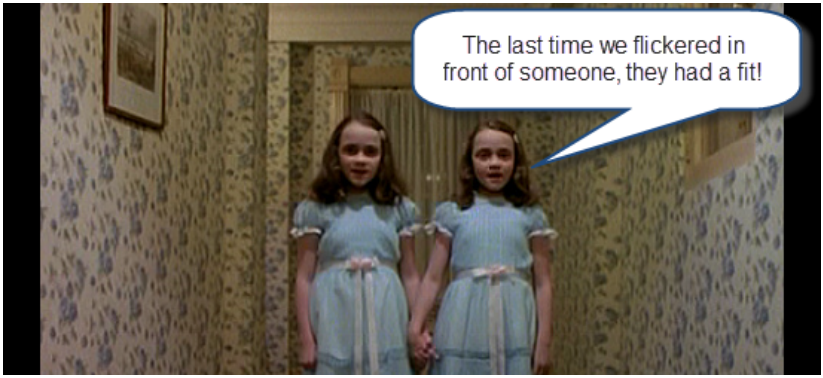
One, two, Freddy's coming for you, three, four better... <insert MIDI sound file here>.

Automatically playing music has been popular for many years but it's still not a very good idea. Even with an "off" button provided, it won't resolve the initial scare of the noise pollution.

Flashing Content

While scaring people with audio is effective, an equally annoying method to creep people out is by using rapidly flashing content.

Have you ever been on a site where tons of things start occurring on the page all at once and your eyes are forced to bounce around the browser to get a handle on it? Yes, this isn't something that'll make you jump, but it's still a worthy horror!



Like the twins in The Shining, we really don't want random things jumping in front of us.

Animated GIFs have mainly been ostracized for their ability to turn your professional-looking site into a kid's TV show on acid (with dancing hamsters and Homer Simpson avatars). It's scary stuff to see in modern sites.

Hideous Source Code

Our next horror is dedicated to designers and developers who deeply care about quality and best practices. Yes people, there's nothing like seeing a website that seems OK on the surface only to look under the hood and be reduced to tears at the sight of the source code.

Obtrusive JavaScript smeared everywhere, copy/paste scripts, divitis, table-based layouts, deprecated code — a site appearing to be resurrected from the 90s.

If you've seen the mummy, he's not so pretty once you've taken off the wrapping!

Beyond the issues of not following web standards and best practices, the scale of the terror is most felt when you realize that the site owner of the unfortunate site probably paid good money for it.

With the next generation of web designers being more aware of how important quality code is, this horror should eventually be reduced to obscurity.

But for the moment, the shock of seeing some gnarly, bloated code is enough to freak a web designer out.

Sudden Client Deadlines

Here's another way to terrify a designer and, in this case, it's the result of something our clients do rather than what we do to ourselves or each other.

You've got a contract in place, and you have a friendly client who's a bit lacking in clarity on the project. Suddenly, from nowhere, you receive an email in which the individual says they need to push the deadline up to tomorrow!

As you don't want to lose your client, you work through the night to finish the work. Scary thought, right?

Working with clients can quickly become like an Alien vs. Predator movie.

Communication is important in any project but, unfortunately, it's something that seems to degrade regularly between designers and clients.

While contracts can avoid scope creep and while project planning can help limit these types of problems, it's both sad and frightening to hear the stories of those who have suffered at the hands of their client's needs (some clients may be the most terrifying creatures in existence).

Outdated Technology

Our next horror has given many designers and developers nightmares. Outmoded IT can be scary: anyone who still fights against the tide in developing for Internet Exploder 6 will testify to the torture that can ensue from watching your beautiful design be reduced into a hideous Frankenstein-like creature.

In addition, the memory of Microsoft FrontPage code sends a shiver through our spines.

Vampires have a habit of living for long periods of time, just like Internet Explorer 6.

It goes without saying that like a Stephen King style horror novel, the ravages of the web's aging leave us being forced to endure the problems of compatibility in a manner we cannot easily escape.

Just like Freddy Krueger, IE6 keeps returning to stalk us even though we've tried to kill it off more times than Rasputin.

Continuing to test for compatibility and ensuring that our code is built using standards and best practices is part of the job. For the sake of our site users, we need to be pragmatic and embrace this terror!

Obnoxious Scripts

Nobody likes feeling as if they've lost all control of a situation. Obnoxious scripts that automatically resizes a window or disables right-click functionality is just plain horrific. Obnoxious scripts are among the most inflictive terrors online.

Werewolves lose control at the full moon. Internet users suffer it on a regular basis.

It's only natural that site owners may want to protect their assets from being stolen; but this should never be done at the user's expense. Crippling the right-click functionality, for example, may seem like you're preserving your content, but it's also going to hurt the visitor who isn't there to do bad things. In addition, unscrupulous individuals that do want to steal content already have workarounds to these things anyways.

Exploitative Site Activities

Beyond losing control of your browser (thanks to some ugly JavaScript abuse), there's some other scripts which are worthy of inclusion within this horror gallery, most notably the redirects and sudden popup windows that occur without permission.

Nothing surprises your visitors more than unexpected navigational events, and I am sure that after the 90s, many of us fear the unexpected redirect on the premise that we may end up in some malware-infested site.

Don't be like Jigsaw, avoid trapping your visitors into situations they don't want!

If that wasn't enough horror to contend with, the levels of malware seems to regularly be on the rise, privacy has become a serious issue and there's always the questionable use of redirection and page refreshes. As a site designer, we need to tackle these many adversities in order to gain our visitor's trust.

Get The Oxygen, Nurse!

As you think about the web's many problems, it may seem like a never-ending battle between good and evil (and unfortunately, it's one that may never end).

From the outright scary (that will literally startle you and make you jump) to the factually scary (which could make you feel a little depressed), the web is filled with monsters and creatures that keep us on our toes. Sometimes it's comical, sometimes it's quite annoying, many times it's genuinely wrong — but like with any good horror film it's full of twists.

As we become more digitally dominated — as our lives increasingly revolve around the internet — the need to fight off these horrors is in our best interests because horror really isn't much fun when it directly affects us and our users.

60 Questions to Consider When Designing a Website

We spend a lot of time asking ourselves, our clients and other people questions. Whether it's choosing the perfect shade of green for our latest web layout or figuring out how to implement a complex typographical solution, the ability to ask the right questions is among the most critical of skills for a web designer. In this article, we'll go over 60 specific questions that web professionals should ask before taking their website public.

Why Asking Yourself Questions Is Important

Many professionals work with the aid of checklists, while others routinely check for certain issues as the design evolves. While there isn't a sure-fire way to avoid the embarrassment of forgetting something post-launch, the habit of continually questioning your work as you develop a website is critical. Sometimes it can be as simple as "Does this work?"; in other cases, more technical questions need to be asked (and answered).

25-point Website Usability Checklist

Accessibility	Rating	Comments
1. Site load-time is reasonable	✓ ✓ ✗	
2. Adequate text-to-background contrast	✓ ✓ ✗	
3. Font size/spacing is easy to read	✓ ✓ ✗	
4. Flash & add-ons are used sparingly	✓ ✓ ✗	
5. Images have appropriate ALT tags	✓ ✓ ✗	
6. Site has custom not-found/404 page	✓ ✓ ✗	

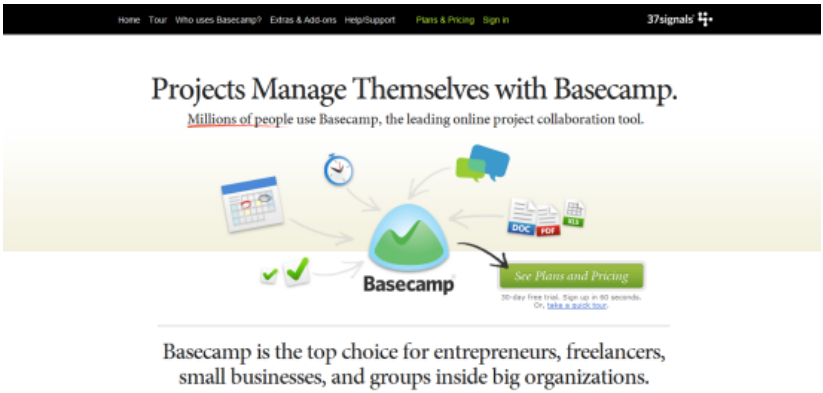
The 25-point Website Usability Checklist (PDF) can be a helpful aid to your workflow.

It doesn't make the job any easier to second-guess yourself into a state of neurosis (something perfectionists do quite often) or to make

blind decisions. There's no perfect method for gauging a project's needs or the decisions we make, but asking difficult questions during the process helps us avoid issues later on.

15 Questions for Project Management

One of the central tasks of web design is project management. Building a new website is like setting the foundation for a house. With so many details to deal with, planning ahead and managing the ongoing tasks is essential.



Basecamp is a popular and effective project management app.

1 Has the client signed the contract? Working without a contract is extremely risky.

2 Do you know what the final product should look like? Having a solid plan of action, including a few diagrams, wireframes, prototypes or mock-ups, can enhance clarity.

3 Has all of the content been written? A website without content is like a painting without a canvas; ideally, a website should be built around the content, not vice versa.

4 Does the website require any pre-built solutions? Life can be made easier with tools such as content management systems (e.g.

WordPress) and scripts, so determine what you need before you start coding.

5 Do you know what the competition offers? Your rivals are often the best source of ideas, and knowing what they offer can help you meet visitors' expectations.

6 Have you set appropriate deadlines? Setting realistic deadlines and tracking your progress towards those deadlines is always important.

7 Will you need to factor in additional costs? Websites are relatively inexpensive, and you can build a good one using free software, but still, you must be on top of any expenses you might incur.

8 Do you have the necessary skills? Some websites are more complex than others; consider which technologies you will need to work with and whether your knowledge of them is current.

9 Have you thought about marketing? A website without visitors is useless. Look into your options for social networking, SEO, advertising and more.

10 Will the website actually be useful (or even necessary)? There is no point wasting your energy on a project that will have no value for end users, so start by weeding out bad ideas.

11 Is a target audience mapped out? Knowing what kind of people you hope will visit the website will help you not only write appropriate content but design effectively, too.

12 Do you have a checklist or criteria? Even a set of basic criteria to maintain quality control or a checklist for larger projects would help.

13 Can your host cope with the demand? Getting the right type of hosting is important; there's no point in having shared hosting if you're going to be streaming gigabyte-heavy video.

14 Have you got the media? Some websites require video, audio and special file types such as PDF documents. Accounting for assets early on lessens the risk of launch delays.

15 What features do you hope to include? Perhaps you need to accept payment, or maybe you want a photo gallery. Whatever you need, plan ahead prior to designing the layout.

15 Questions for Code-Authoring

Next up are questions to ask regarding writing code. If you design or develop websites, you'll find yourself working with HTML, CSS, and JavaScript. Every language has a range of best practices and guidelines to follow, which is great if you want to standardize your end-product. However, there are a lot of other things to consider besides being standards-compliant.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
<html xmlns="http://w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>Project management software, online collaboration: Base
  <meta name="description" content="Trusted by millions, Basecam
schedules, and milestones." />

<meta http-equiv="content-type" content="text/html;charset=UTF-8
<meta name="viewport" content="width=1065" />
<link rel="Stylesheet" href="/bchg.css?1288102059" type="text/cs
<script type="text/javascript" src="https://use.typekit.com/jty7
<script type="text/javascript">try{Typekit.load();}catch(e){}</s
<script type="text/javascript" src="/sprockets.js?1288102059"></
<script type="text/javascript">var thirtySevenSiloTrackerId = "U
<script src="http://37signals.com/ga.js" type="text/javascript">
```

The impact of source code on the effectiveness of your content is often overlooked yet very real.

16 Does the code validate? While validation isn't a complete testament to code quality, it does help to make sure that your code follows recommended standards and can show you errors in your markup, CSS, and JavaScript.

17 Have you considered using CSS3 and HTML5? Though many users still don't use browsers that have CSS3/HTML5 support, if implemented with progressive enhancement in mind, taking advantage of these future W3C recommended standards gives your products added value and improves the craftsmanship of your web designs.

18 How semantic is the code? Using the right tag for the job is essential, and search engines love semantic code. Use <p> for paragraphs, <ul> for listed items that have no ranking, <ol> for items that have a sequential relationship, <a> for hyperlinks and <button> for clickable UI components that perform an action/user task.

19 Are you taking advantage of optional files and site add-ons? Whether in the form of using the Sitemaps XML protocol or including a favicon, these optional files can enhance your website. See five files that can improve your website.

20 Do you need an RSS feed? If your website is content-heavy and is updated frequently (e.g. a blog or news site), having an RSS feed will be a necessary site feature for keeping your users up-to-date with fresh content. If you don't use a CMS with this feature built-in, check out SimplePie, a PHP class for building your own RSS feed.

21 Will the code degrade gracefully? Graceful degradation (also known as fault-tolerance) — the notion that a system (in this case, a website) will still function under sub-optimal situations — is essential for flexible and web accessible site builds. Learn how to pragmatically apply graceful degradation when using CSS3.

22 Have you considered SEO? While search engine optimization should not dictate your design decisions, it wouldn't hurt to consider how your website could be more visible in search engine results. Read some SEO tips to remember when building your site and ways to improve the SEO of your web designs.

23 Do you provide a printer-friendly style sheet? Designing a print CSS file is worth the time investment as many users still do print out web pages.

24 Is any of your code deprecated or non-standard? Using "dead code" such as the <font> tag that was deprecated in W3C HTML 4 specifications as well as non-standard code such as the <blink> tag is frowned upon and won't allow your work to validate against W3C web standards recommendations. Double-check that you're

not using any by taking advantage of free validation tools such as the W3C Markup Validation Service.

25 Do you need to use conditional comments? IE6 isn't going to go away completely, and if your project requires you to support IE and to use browser-specific code, use conditional comments to serve IE-specific stylesheets instead of using hacks. This does two things: It gives you the ability to get your code to validate under W3C standards, and when you decide to stop using browser-specific code, you only need to remove the conditional comments in your site template. Leveraging JavaScript techniques for fixing IE6 and projects such as HTML5 Boilerplate to solve deprecated-browser rendering issues could be another option, but you'll be stuck in scenarios where the user uses an old browser with JavaScript disabled — a scenario that is not as uncommon as you might think.

26 Are structure (HTML), presentation (CSS), and behavior (JavaScript) separated? This is important not only because it's best practice, but also leads to more manageable and maintainable code.

27 Is your site navigation laid out in a practical way? The navigation menu is the most important part of your website. Getting it right is an integral part of an effective site information architecture.

28 Have you checked for unnecessary HTML elements and redundant CSS style rules? Code bloats easily, so strip away any non-essential and repeated bits of code for more maintainable and leaner (and thus, higher performance) website builds. For HTML/CSS optimizing purposes, check out HTML Tidy and CSSTidy.

29 Is the code organized and maintainable? Put care and attention into your code. Lay it out so that it's easy to read, update and manage.

30 Would a framework enhance the site? These days, open source Ajax/web development frameworks such as jQuery and MooTools can speed up code-authoring and ensure fewer headaches due to

cross-browser issues. If you suspect these frameworks might help, why not investigate and learn about them?

15 Questions for Web Designing

The process of creating a layout is full of questions related to color, typography and even distinctiveness. While your project management style may be superb and your coding technique beyond measure, design comes with its own set of questions. Web design calls for endless decisions, and that's what these following questions are supposed to help you resolve.

Using a wireframing tool like Balsamiq Mockups ensures a solid layout foundation.

31 Have you optimized your media? Images, videos and audio take up more bandwidth and space than anything else. Consider compressing and optimizing them with tools such as Smush.it.

32 Is the user interface overcrowded? If there's one thing no one likes, it's a stuffy and bloated design. Determine whether reductionism can help you design better websites.

33 Is the design distinctive and unique? With site templates in abundance, having a layout that's fresh and eye-catching is a must. Breaking the mould may improve your brand's identity.

34 When should I redesign? Are you able to produce something totally different or enhance what you have?

35 Does the layout make sense? Whether you pick one column or three, a lot of scrolling or none, decisions on your pages' visual hierarchy will directly affect readability.

36 Do the colors give off the right feeling? Color is closely linked to emotion; a palette can be the difference between a fun-looking website and professional-looking one.

37 What typography is best? As with color, typography affects the feel of the website. Build your font stacks wisely and attentively and take time to craft a masterful typography design.

38 How visible are links? Links have no purpose if they cannot be seen. Make sure you take the time to design your hyperlinks well.

39 Are you using enough white space? Too many websites squeeze everything too close together. If you add some breathing room, the result could be improved readability.

40 Have you considered content on-demand? With the rise of Ajax and fast content switching techniques, packing more data onto the page is easy. Consider doing this with very long web pages.

41 Is the design aesthetically pleasing? While this process is subjective, it's still a good idea to get feedback from your friends, co-workers and perhaps a stranger or two to determine whether your work is visually appealing or not.

42 Is the content readable? Nothing is more important than content; if it's legible and coherent, then your site users will be happy.

43 Does the design scale at various resolutions? Displays are getting bigger (bigger desktop monitors) and smaller (mobile devices) at the same time; make sure your work renders in all web-enabled devices. For mobile devices, take advantage of free tools for testing designs in mobile devices.

44 Are important site features emphasized? Some things are more important than others; consider the various ways that relevant content can be highlighted so that visitors can easily find it.

45 Does the website feel complete? This is probably among the most important yet difficult questions to answer. Recognizing when it's ready requires a lot of care and thought.

15 Questions for the User Experience

The user experience is perhaps the most important factor for determining the success of a website. Here are questions related to UX, usability, and accessibility.

Accessibility, usability and compatibility: few things are more important. Silverback is a popular usability testing app.

46 Does the website work equally well across different browsers? There are plenty of browsers out there — make sure your website works well in the major ones. You can use a web service like Browsershots to preview your work in various operating systems and web browsers.

47 Is the website mobile-friendly? While desktop browsers are pretty straightforward (with the exception of IE), mobile devices require an extra bit of care and attention; read about best practices for the mobile web design.

48 Have you tested the website in a screen reader? Unfortunately, even with free screen readers out there like Fire Vox, a screen-reading add-on for Firefox, few web designers consider testing

their designs for screen-reader web accessibility. You might want to.

49 What happens when JavaScript is turned off? Not every experience is the same and we can't control the visitor's browsing environment, so try to make sure your website gracefully degrades when JS is turned off.

50 Do you offer alternatives to Flash content? Following on the previous point, if your website is particularly Flash-dependent, you might want to make sure that your use of Flash is accessible.

51 Did you remember alt attributes? One of the simplest accessibility aids to implement is using descriptive and useful alt attribute for images.

52 Have you evaluated your website against Web Content Accessibility Guidelines? Complying with web accessibility best practices is important for users who have disabilities that affect their capability to browse the web. Fulfilling the recommendations in the W3C Web Content Accessibility Guidelines (WCAG) is the perfect place to start.

53 Has the website been tested by other people? Usability testing is quite easy and inexpensive to carry out nowadays. Performing tests could give you ideas for improvements. Check out web services such as Concept Feedback and Feedback Army.

54 Do your URLs make sense? URLs that are easy to read will give potential visitors the chance to predict where they're headed (and is good for SEO to boot). Using pretty URLs (example.com/about-us) instead of system URLs (example.com/?p=655) can enhance the experience of visitors. If you're using a content management system or a custom-built app, learn about rewrite engines.

55 How quickly does your site load? Speed is an important factor of usability. Consider how your website will affect visitors, particularly ones on slow connections.

56 Is the search functionality easy to use? Most websites need a search box to help visitors locate the information they need. Ensure that yours is easy to use and that the results are accurate.

57 Will there be any potentially obnoxious behavior? Whether it's pop-ups and modal windows that won't close, or scripts that cripple right-clicking, make sure your site doesn't have behavior that annoys users.

58 Are your web forms usable? If you're asking for too much information or your forms are too hard to complete, people will enter fake details or simply refuse to submit the data.

59 Can the site owner be contacted without difficulty? While you might get spammed in the process, allowing visitors to send you an email or to initiate a Skype call could be a great way to connect with them.

60 Have you checked for broken links? Root out dead links in every nook and cranny. Tools such as Xenu's Link Sleuth can automate the process; learn how to discover broken links in your website.

Conclusion

As we learn and grow, our competency increases, which changes our perspective and workflow. Designers and developers who regularly question their methods and ideas are usually the ones who get the job done right and are the ones who consistently improve their processes and products.

Sources:

- http://drpete.co/pdf/checklist.pdf
- https://basecamp.com/
- http://alistapart.com/article/understandingprogressiveenhancement
- https://www.sitemaps.org/protocol.html
- http://simplepie.org/

- https://en.wikipedia.org/wiki/Fault_tolerance
- http://jonraasch.com/blog/graceful-degradation-with-css3
- https://validator.w3.org/
- https://www.quirksmode.org/css/condcom.html
- https://html5boilerplate.com/
- http://www.html-tidy.org/
- http://csstidy.sourceforge.net/
- https://balsamiq.com/products/
- http://www.imgopt.com/
- https://silverbackapp.com/
- http://browsershots.org/
- http://firevox.clcworld.net/
- https://www.w3.org/TR/WCAG20/
- https://en.wikipedia.org/wiki/Rewrite_engine
- https://www.nngroup.com/articles/the-need-for-speed/
- http://home.snafu.de/tilman/xenulink.html

Situational Design for the Web

During the browser wars, interesting problems presented themselves to the web design community. Many web professionals resorted to drastic measures and built separate websites for IE and Netscape — and later we had wireless markup language (WML) for mobile phones. This was because of inconsistent rendering and poorly implemented standards, and it was a means of avoiding the ugly hacking that was otherwise necessary.

This practice has evolved over the years (take print-friendly pages, for example), but the modern web almost shuns the practice entirely.

Web standards are being developed and adopted at a rapid pace (especially as older browsers retire) but the dawn of a new device-friendly era in which mobile phones, cars, televisions and household appliances can access the web has resulted in an uncontrollable situation.

As a web designer, I'm always on the lookout for good solutions, and as strange as it may sound, the case for having separate, situational websites to deal with different devices is getting stronger by the day.

Here's why.

Problems with Singular Solutions

The initial problem that forced us to adopt a one-size-fits-all model was discrimination. During the browser wars, our main complaint was that deliberate actions by browser makers to gain dominance ended up hurting web standards. Even the humble print-only page was an act of "discrimination"; it drew users away from the standard "Print" button in browsers.

Every case of separation led to some kind of web design anxiety, which made life extra hard.

We no longer have two devices and five renderers; now it's 100+ devices and 20+ renderers!

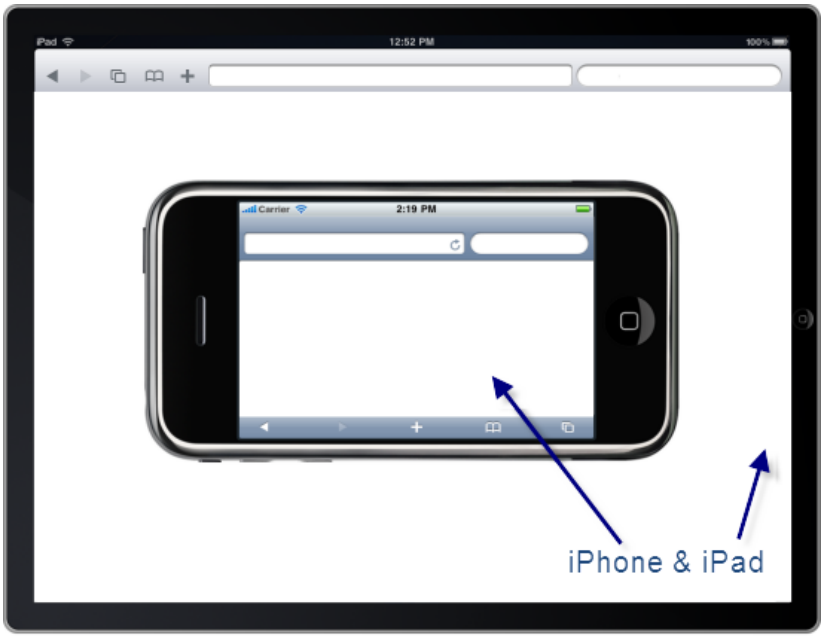
A single website for every situation: what a wonderful concept. Such a website would be easier to maintain (unless the other designs are automatically generated), and they would cut down on domain confusion by doing away with forced URL redirects and not burdening users with having to remember the top-level domain (TLD) for their specific version.

If browsers and devices rendered things equally well and treated our wonderful layouts as they were intended, we could continue to rely on one-for-all websites.

But things are no longer that simple.

Years ago, we worried only about screen resolutions, web-safe typography and whether browsers could handle our code.

Now, we deal with a range of unit measurements that boggle the mind. How big is the screen? Does it support zooming? What input hardware is available? How fast can it render? Do users have bandwidth limits? How can one layout be fluid enough to deal with it all?



Don't confuse the iPhone and the iPad; there are two different screen sizes to design for!

With single page websites, it isn't so much about the visuals that our users have to settle for (though usability is important); it's about the measure of the background processes involved, and about making our pages as agile as possible for a platform's needs (while taking advantage of its hardware).

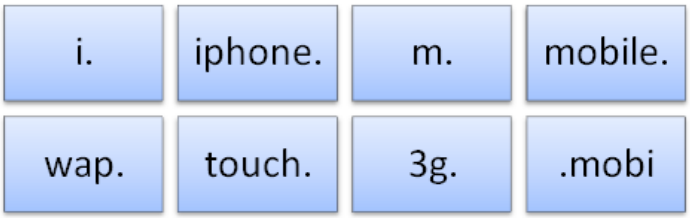
We talk a lot about optimizing our web designs for usability, user experience, and front-end performance, so (putting aside time constraints for the moment) why would we give users a "second-hand" layout — built for one platform and then hacked into a new shape for another?

We Need Dynamic Designs

The Wireless Markup Language (WML) is a depreciated language, but I have to admit that web designers could learn a lot from the short-

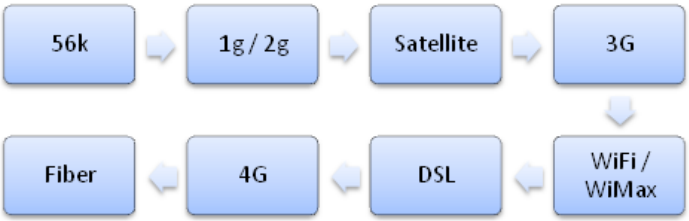
lived benefits that it brought to the table. It forced us to create a separate layout and structure for mobile phones — which were the only alternative devices, apart from the unpopular WebTV product that MSN put out in the '90s.

The new layouts brought new advantages. Most pages were small and agile, and they were guaranteed to work on mobile devices. Plus, we weren't left with countless "what if" questions.



Many designers recognize the appeal of being platform-specific, so TLD conventions for them now exist.

Standards have evolved, naturally, but the need for dynamic, situational websites that cater to certain device types is evident. Bandwidth and speed are major issues on mobile platforms, and support for technologies that address those issues could change the game. Too many designers look at the mobile web simply as "the desktop, but smaller" — and that is a fatal mistake.



Your visitors could be using one of a number of connection types that affect speed.

The problems we face now result from the mysterious issue of device diversity. Following the principles of progressive enhancement would curtail the quirks of designing for certain devices, but they wouldn't help with the redundancy of adapting desktop websites to a mobile platform.

It's not just a matter of having separate websites for each device; it's about creating one version for all web-enabled platforms and using the available tools to enhance the experience on each.

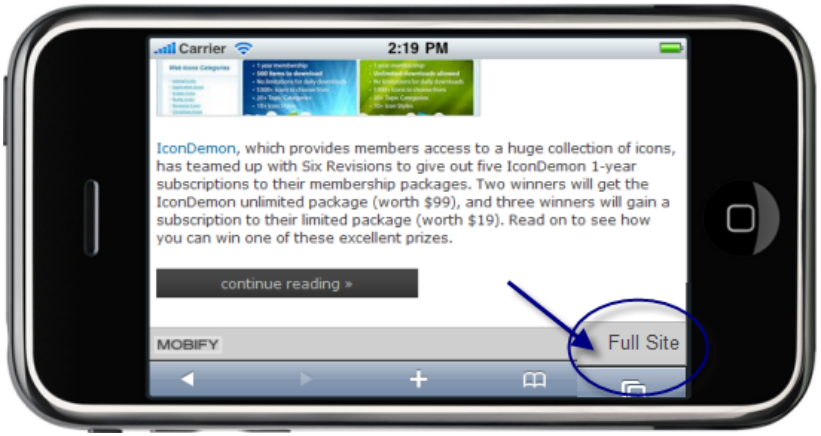
As it currently stands, the major platforms include:

- Desktops (including notebooks, netbooks and traditional computers)
- Mobile devices (not just mobile phones, but tablets and other touch devices)
- Televisions (both web-enabled TVs and game consoles on large displays)

Appliances (don't forget those quirky devices that don't fit in other categories, like refrigerators)

The number of people using each of these platforms is increasing, and although the numbers are difficult to crunch, it stands to reason that we should design only as our audience calls for it.

If you're building a mobile website, why not tear it away from the desktop and gain the benefit of a clean surface to work with — and, more importantly, the opportunity to adapt your content to the platform? If your visitors want the mobile experience, they can choose it; leave the full desktop website untarnished and accessible to others.



Allowing users to revert to the desktop experience is important. Don't forget it!

The Benefits of Separating Structure

Apart from offering two ways to browse, what are the main reasons to build separate websites rather than make a desktop design "mobile-friendly"?

The primary benefit is greater compatibility. With a desktop "conversion," one relies entirely on browser detection to ensure that the right design is being served. Instead, make a generic mobile layout that can work on all devices of that type but that can still be customized for specific platforms. There are plenty of tools for testing your mobile layout on various devices.

Let me justify separate, situational web designs:

- Even unknown mobile devices will be able to access the mobile experience instead of the desktop one.
- Maintaining "themes" for particular devices (i.e. styling the mobile structure) is easier.
- Users can quickly and easily switch between designs, with no extra scripting.

- You can identify key markets (phones, TVs, etc.) and design unique layouts for them.
- Screen real estate will not be wasted on clutter that forces users to zoom around.

A separate website could have a standardized mobile theme and be further enhanced with feature detection, perhaps for iOS or Android. Of course, this is also possible with a normal layout. The unfortunate side effect of detection is that website overhead increases. Rather than designing for smart phones and focusing on a couple of models for custom styling, we instead have to account for the hundreds of thousands of variations. It's a scary thought.

Remember that old devices deserve as much respect as advanced platforms.

The benefits of separate website designs can also be seen in code. By offering options to your visitors, you'll avoid the need for feature detection (unless you want to auto-forward them). Keep your desktop website clear of mobile code and the mobile website free of desktop code; this is crucial. It may not seem significant, but redundancy on the mobile web is a serious issue that desktop websites can't solve alone.

Here are some more reasons:

- Users only have to download the code that is useful for their device type.

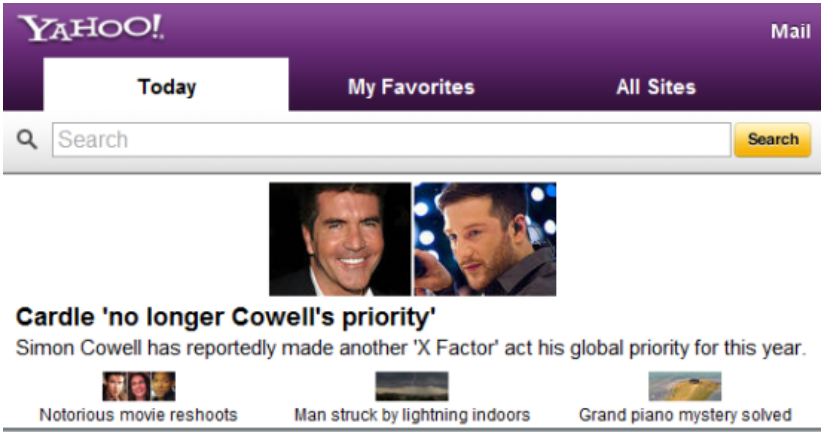
- The overhead for detection scripts is greatly reduced to just special cases (if any).
- You can craft an alternative experience that takes advantage of hardware requirements.
- Printers have browser support, but device capabilities require feature detection.
- You save time with less CSS and JavaScript or having an alternative WML structure.

Content Design Optimization

Of course, the biggest justification for separate web designs — and it goes against the current trend — is content, and the opportunity to tailor it to individual platforms.

If your website is for mobile phones, you know that screen sizes will be small and, as such, images and video should be scaled down to match (which will save loads on bandwidth).

Content can be reduced in density, too (to reduce heavy scrolling), and longer documents can be broken down and served in more digestible chunks.



Scrolling a mile of content is no fun, so consider breaking it into small clusters.

If you still need reasons, here are a few more:

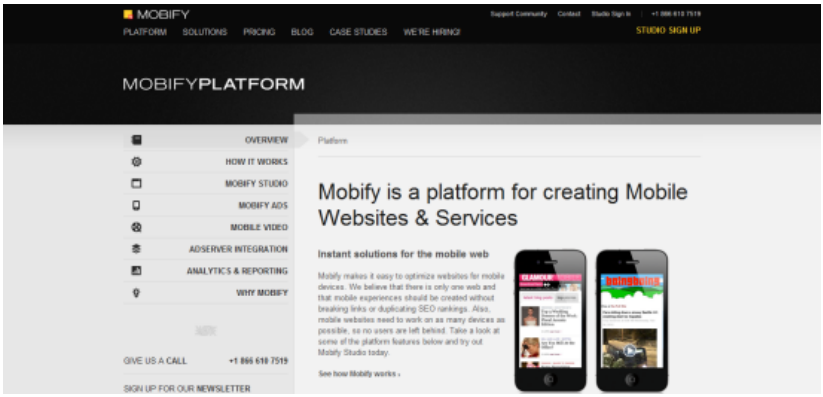
- Lengthy content may not adapt well from one layout to another.
- You can treat media-capable platforms different to text-only devices.
- Images and media can be scaled up or down to match screen size (saving bandwidth).
- Connectivity issues can be addressed (such as by displaying content on demand on one-page websites).
- You can serve relevant content, discard the rest and reduce costly interactions more easily.

The Web's Wild West

Providing a device-oriented experience ensures that dissonance with interfaces is as low as possible. Still, there will be occasions when a separate website is not needed. Whatever your decision about building separate websites, your visitors' needs should be the guiding factor.

Complex websites with multiple columns, lengthy scrolling, large file sizes and a lot of images and media are ideal for separate websites in most cases.

A simply designed or single-page website might not benefit much from dynamic, situational behavior. Often, you'll want to drain away excess material, to be left with a design that merely requires some optimization. If you're producing something complex, like a web app or an elegant use-every-inch-of-the-screen layout, that's obviously another matter; but simplicity is diversity's best friend.



Automated tools like Mobify help, but they don't optimize content — which is 90% of the website.

Situational web design isn't just about seeing the web in a new light: it's about recognizing that the diversity of ways of interacting with the web need to be addressed. Designers put so much effort into building desktop websites that it becomes second nature to think that variations should match the original design as much as possible. In the '90s, we realized that serving the exact same layout to two browsers (and have it be optimal and usable) isn't possible; the same is now true with the device revolution.

Examine your website and how it interacts with different devices. Perhaps you could reduce data usage by employing device-oriented graphics (based on screen size). Maybe you could serve less content per page for small screens, or more content for large one. Whatever you do, design for each platform independently. Design around the content in order to give users on a particular device the opportunity to enjoy it.

There is no longer just one web to design for; let's make the best of it!

Sources:

- https://en.wikipedia.org/wiki/Wireless_Markup_Language
- https://en.wikipedia.org/wiki/MSN_TV#The_WebTV_set-top_box
- https://www.mobify.com/

The Importance of Historiography on the Web

The topic of history immediately draws to mind a dusty classroom in which professors tell stories of war, royalty and civilizations lost to the sands of time.

While traditional history is expressed as a vibrant tapestry of events, dates, people and places, we often forget that the web has its own rich history and a legacy to leave future generations that needs both preservation and recognition.

By examining current problems in how we preserve our digital heritage and through a significant change in our attitude towards web content, we can hope to leave future Internet users with something tangible and useful.

History doesn't exist separate from our actions; it is built up over time in what we write and record, allowing those in the future to analyze and improve upon our work. Addressing our current perspective of web content as "disposable data" is critical at this time.

Evolution of Knowledge-Sharing

The passing on and recording of knowledge is a time-honored tradition. This practice has spanned generations, ranging from mankind's early cave paintings to the industrious storage of information, such as can be found in the Library of Alexandria in Egypt.

Yet, while so much emphasis is placed on preserving our rich analog history, our digital past seems to be rapidly disappearing around us; and like the great Egyptian library that was lost to fire, we are now left with large gaps in our understanding of how much of our present web culture came to be.

Creative Commons, freeing content from the claws of copyright and loss.

The average website exists in a single form for a period of time before being reinvented when it goes through a website redesign, but if we value published content, then the need to preserve it should be immeasurable.

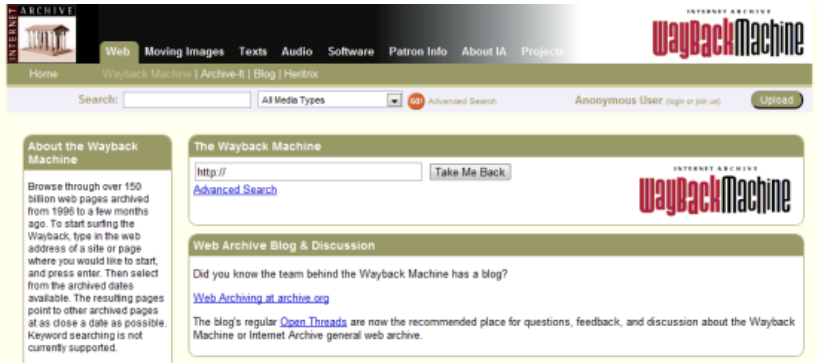
Today, many argue that if it does not appear in search engines or if there's no clear link to it to the web, then it no longer exists. By law, the famous Library of Alexandra preserved any scroll it acquired, yet with the web, we throw out useful old content if visitor numbers are not high enough or because certain copyright laws prohibit us from preserving knowledge soon to be lost.

Digital Curators and Librarians

It would be unfair not to mention some present-day schemes to protect valuable web content from being discarded, such as the Internet Archive and, to a point, Wikipedia and Google.

However, if we look to Geocities being shut down, we can see the damage that a web service's disappearance can cause to our web culture. With the bookmarking service Delicious being threatened into extinction, we might see valuable bookmarks of its users lost as well.

The knowledge and imprint of our history serves to teach others about the development of the web, and we must accept our role as curators and librarians of this modern digital world.



The Internet Archive is like a museum of old websites, but it hasn't saved everything.

As a web professional, seeing web content disappear makes me sad, even if its relevancy and accuracy changes over time.

As we produce more and more content, finding true treasures on the web is becoming increasingly difficult. While the average blog may only have fragments of gold, it still reflects the diverse world we inhabit. The purpose of owning a website is to increase visibility, but so many still leave their creations unattended, creating dead, broken links, orphan pages and poor navigation and archival systems.

We should do more to improve our websites and showcase the good content in our archives, even if only to revisit an article or subject from a present-day perspective.

The Danger of Disposable Data

We have bred a society in which information and opinion aren't valued over the long term. As a result, we are left with no infrastructure to ensure the sustainability of that content.

For example, little exists from websites created in the 1990s, and what can be found is often disjointed and scattered; imagine the state of our web content 20 years from now!

Quite paradoxically, we value our own work so highly, often spending hours upon hours creating beautiful websites and amazing content, and yet we forget them after their 15 minutes of fame.

The BBC archives old information, but even that's not safe forever.

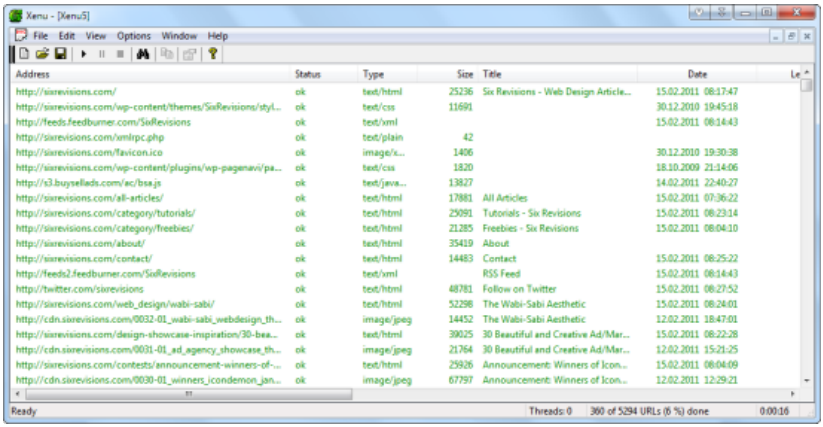
To wake up from this mentality of waste, we have to be critical in our evaluation of modern web history. We must recognize that as the web changes, formats will shift and media consumption will alter, and we will need to maintain some level of control over the effects of popular trends.

Missing Links and Lost Empires

As a web professional or website owner, you can do plenty to reduce the disappearance of your content on your website. Certain practices have profound benefits and could give users additional reasons to return to your website. If we can expose a website's history, we provide a more enriching experience full of quality content.

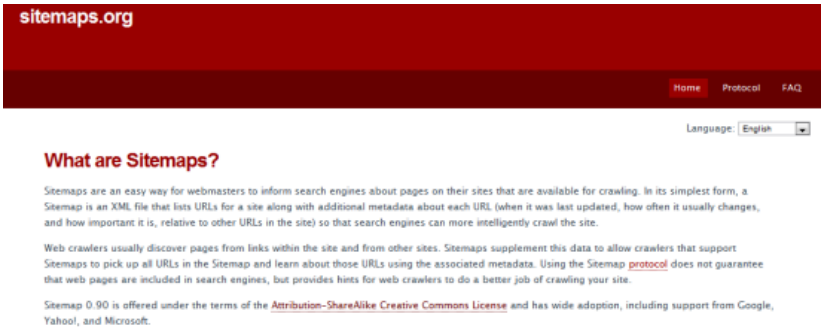
The first thing to do in promoting a healthy archive is to weed out the dead links that accumulate over time. This waste is easy to spot with tools like Xenu Link Sleuth (freeware), which scans every piece of content on a website.

Chronicling a website with thousands of pages can be quite complex, but maintaining an effective website navigation scheme and an up-to-date site map is central to good information architecture.



Automated tools can poke around your website and report dead links.

Next, we can ensure the survival of our content by connecting every page to the rest of the website and listing them all on a site map (as well as creating a robot-readable Sitemaps XML file). Disconnected pages — pages that have no remaining active links to them — are orphans. Orphan pages rarely index well, which negatively impacts the findability of your content and its usefulness to future generations.



Site maps can be easily produced using an app, but you might want to code one yourself.

If you're the kind of person who shudders at that thought of their earlier work, consider making active revisions of your old content

visible on your website; do, however, preserve the original using some form of version control system.

If you redesign your website regularly and change content often, keep older pages available for nostalgic users or those interested in seeing previous versions. Exposing revisions (like a version history file) can also be very beneficial in tracking the progress and evolution of websites.

Some people allow users to revisit previous versions of their website's layout as a way to showcase their evolving talent.

Donating published content is another practice to consider. While republishing old material ad infinitum is not sensible (because it would create duplicate content for search engines to index and perhaps be regarded as spam), there may come a time when you close down, change direction or overhaul your website.

In such cases, consider donating your posts to other websites (think of it as content acquisition); you could make a bit of money from it and free up useful data.

Certain interest pieces might be useful for Wikipedia or could be donated elsewhere.

While donating outdated or useless content could help you clean up your web presence, the websites or services that receive this old material might not be able to manage it effectively, especially if it comes in at an unsustainable rate; this could, in turn, lead to dead image links, stale content and an increase in 404 errors. Instead, try to use your archived material in other ways, perhaps by promoting earlier articles, as a retrospective of sorts.

This is the first article published on Six Revisions, and yet it could still help a number of readers today.

Finally, the most important precaution for ensuring the survival of your content is to back it up. To this day, many websites still have no substantial process for archiving and backing up their content. What happens if your website is hacked? What if your computer crashes? If the history of computing has shown us anything, it's that data increasingly disappears as a result of both computer failure and a website or web technology aging.

Historiography for the People

The web is ever-changing, and its history is being erased, written over and lost in poorly maintained archives.

High-quality content — even an individual's personal reflection on the world on his or her blog — never loses value. Of course, drowning out the spam and fluff helps, but if we value a healthy digital ecosystem, then we will focus on producing things that contribute to our evolving worldwide virtual library.

The content we produce will give future generations a fascinating look at how the web has evolved over time and how web professionals and ordinary folks have carried out their daily tasks.

We should not build websites solely for the here and now, forgetting the mistakes and successes of those who have come before us. By

preserving the past and documenting the development of the web, we are immortalizing ourselves, ensuring that we don't become yet another people who simply fade into oblivion.

Sources:

- https://archive.org/
- https://mashable.com/2009/04/23/geocities-shutdown/#I5iWgzGu6EqM
- https://techcrunch.com/2010/12/16/is-yahoo-shutting-down-delicio-us/
- https://en.wikipedia.org/wiki/Main_Page

Why IE9 is a Web Designer's Nightmare

Web professionals have been getting pretty excited lately, and it's no surprise why. The latest spawn of Microsoft's browser, Internet Explorer 9, has just been released. Many people have been talking about the changes and whether the latest version is a solid step forward, or if it's too little, too late.

In a previous article, Jacob Gube (this site's founder) had a more positive view of IE9. I'm here to play devil's advocate and present the other side of the coin.

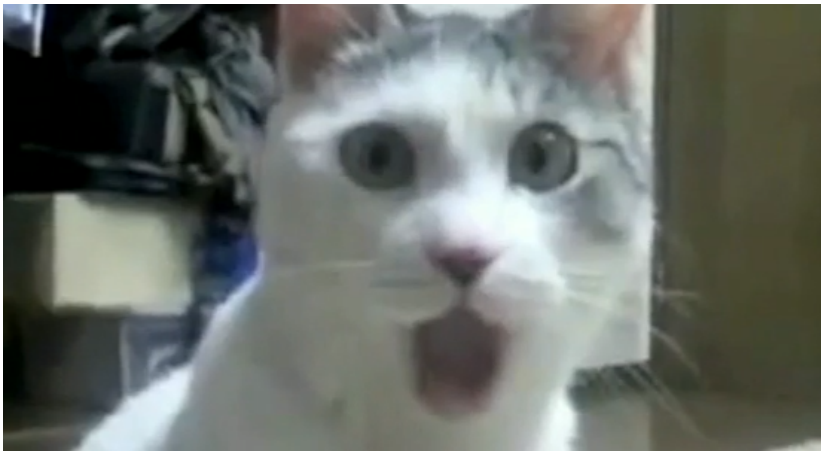
My Rocky Relationship with IE9

After waking up one morning and checking out my Twitter feed, I spotted a tweet that got me pretty excited: The first release candidate for IE9 had launched!

Now, among web designers, I am probably one of the most skeptical of IE — we'll skip the IE6 jokes for now — but this time around, I had a great beta experience and saw so much good work. IE9 was different.

I quickly downloaded it and began the installation process. That went well. I then performed the obligatory Microsoft reboot — it's 2011 and the software still needs a computer restart, but whatever, I'll live — and then I opened up the browser.

So far, things were turning out better than I'd hoped. It was at this early point, though, that my first negative experience with IE9 occurred. I visited my own website to see how it looked, and what I saw left me with a facial expression that only this image can accurately depict:



If my webcam had been turned on, this is the expression you would have seen.

My website's perfectly formed layout was broken. Not only was it rendering badly, but the rendering defects were so great that no amount of IE conditional comments could resolve them. (Dear IE9, gray hair from stress on a 27-year-old is not cool.)

At this point, I did what many in my situation would do: I screamed "Nnnooooooooo!!" so loudly that a dog in the street barked in response.

Luckily, I was able to patch the issue using the XUA Meta hack (so much for semantics and well-formed markup).

Perhaps the bugs will be ironed out before the full release. But I'm a realist, and I didn't feel that lucky.

Internet Explorer Is Improving

Before anyone dives down to the comments and retorts by pointing out all of the great new stuff worth defending in IE9, I should state that Microsoft has done a lot to improve its browser. And truth be told, we need this update more than any other.

So, before we focus on the issues that have made me an IE9 cynic, it's important to note a few good things about it.

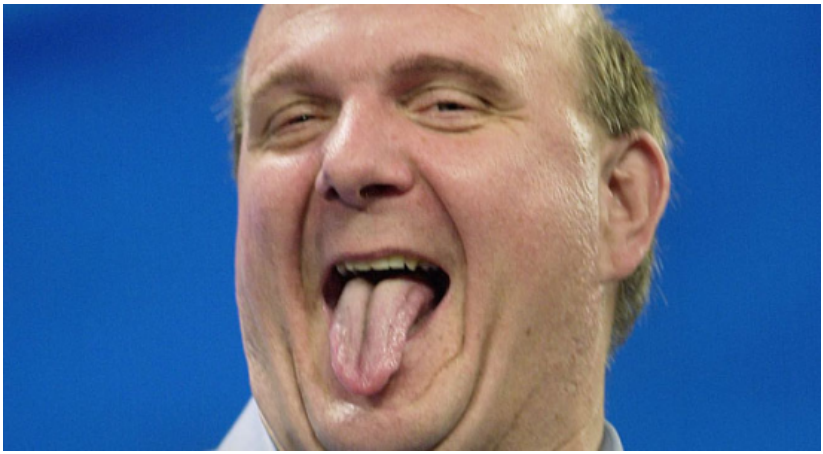
First, although many would like to forget it, IE has been a pioneer in what we now refer to as modern web standards. They were innovators. How we see the Web today is a direct result of their early work.

Granted, Microsoft's push for change hasn't always hit the mark, but without browsers like IE6, we may not have seen such CSS3 properties as the overflow-x and overflow-y properties, web fonts, Ajax, and the ever-useful conditional comments (our savior in the development process). New features in version 9 have impressed me, and they're worth having.

In addition, I'm particularly pleased with the way Microsoft is embracing HTML5 and CSS3. While not perfect by any stretch, the fact that we're seeing current standards being supported by the browser shows that Microsoft is making at least some effort to ensure a better all-around browsing experience. Also, surprisingly, Microsoft's transparency during the testing process (letting everyone get the beta and listening to feedback) is really bringing their browser line back on the path to success.

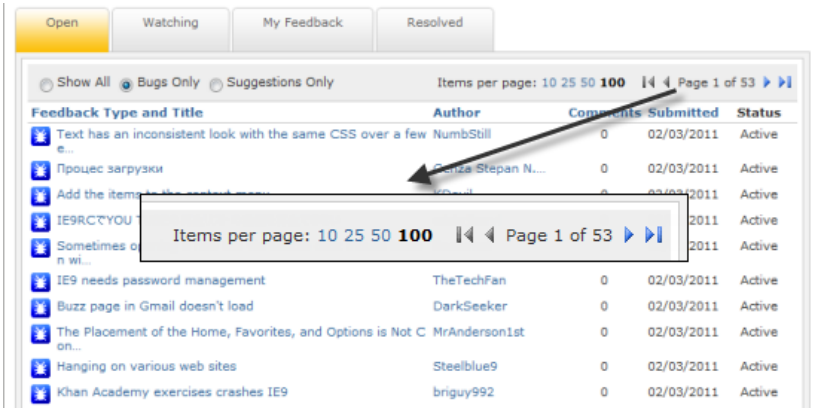
Will IE9 Be the New IE6?

The early warning signs were there from the start, and people have criticized Microsoft's choice to include HTML5 and CSS3 (both unfinished specifications), arguing that poor rendering (which does exist) and future changes could leave the browser in an IE6-like situation when it gets outdated. The frequent release cycles and automatic updates (by default) of other browsers will minimize this problem, but given how slow Internet Explorer has always been with major versions, it may well become the IE6 of 2020!



If Steve Ballmer can't save us from this, who can?

Moreover, the number of bug reports in the feedback program — over 5,000 of them — and the IE team's announcement thanking the jQuery team for updating their popular JavaScript library to be IE9-friendly makes me suspect that this will indeed be our bleak future.



Microsoft Connect: approximately 5,300 open reports (and mine is among them). Oh dear!

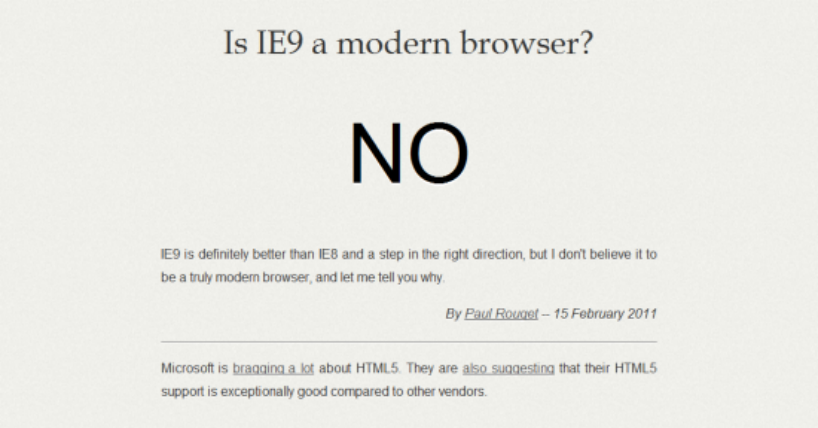
The notion that coders, designers and service providers should patch their code for new browsers is scary. If browsers followed the standards, then bugs wouldn't be there in the first place.

If the issues are severe enough to need patching before a new browser version comes out, how bad will things really get?

Recalling the days of hasLayout, these problems seem to be the same ones we have seen before. Yes, Microsoft has come a long way since IE6, and yes, it is doing its best to keep the browser up to date. But remarking on how well it's doing — and going so far as proclaiming it is better than the competition when known issues are left for the rest of us to work around — seems inexcusable.

Marketing Machine

In a quirky letter to Microsoft, Mozilla proclaimed that Microsoft's boasting of IE9's high level of support for standards is inaccurate. Microsoft has always had an effective marketing strategy with IE with its substantially biased claims and inaccurate research.



Mozilla posted this to show how inaccurate the claims are about IE.

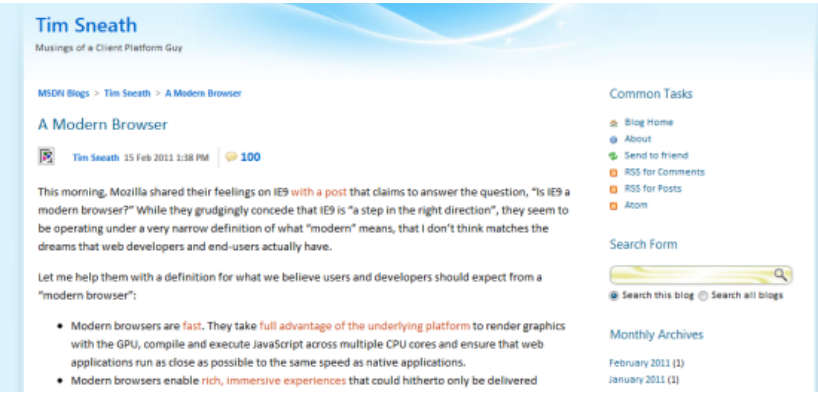
While marketing doesn't affect the browser itself, this misleading treatment of consumers, both past and present, only rubs salt in the wounds of those who make websites for IE.

It's true that while Microsoft's new browser appears far from perfect, no other browser gets it quite right either. All of the other browsers have their share of flaws and bugs, missing technologies, and incomplete spec implementations.

However, the problem with Microsoft is partly due to how it portrays itself and the frustrating way it sometimes goes one step forward, two steps back.

Is IE9 a Modern Browser?

To see what we're all in for, we need only examine a blog post by Tim Sneath of Microsoft, who opines on what makes a modern browser. The substance of his retort to Mozilla's open letter is what concerns me.



The "modern browser" earns points for effort but is rather short on substance.

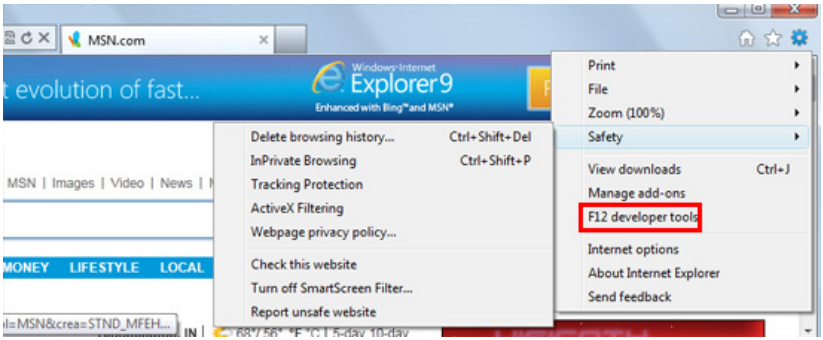
First, Sneath proclaims that Mozilla is narrowly defining the "modern browser." While I agree with him in that Mozilla defines "modern browser" in its own terms and agendas, if Mozilla had broadened its definition, Microsoft would still have come off looking even worse.

Below I've paired up points that Sneath's made with my own view of why the term "modern browser" doesn't apply to IE9.

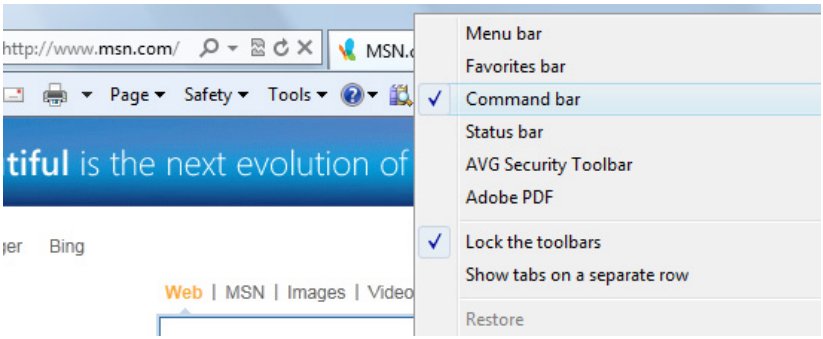
"Modern browsers are fast. They take full advantage of the underlying platform to render graphics with the GPU, compile and execute JavaScript across multiple CPU cores and ensure that web applications run as close as possible to the same speed as native applications."

The new version of IE has improved with regard to the interface's overall speed and usability, and the rendering speed of IE9 does clock well against the other browsers. But in terms of the overall speed of the browser itself (and the intuitiveness of the interface compared to Chrome and Opera), the differences are still quite staggering. The loading times of windows and tabs are not favorable, and regardless of the rendering engine, the interface is not as refined as the ones in other browsers.

The settings menu is a simple illustration of where IE9's interface shows its inconsistencies. For instance, F12 developer tools is about the worst label I've seen in a product; for consistency, it should have been laid out like the View downloads option.



The RSS and Atom feed notification feature has inexplicably disappeared from the address bar, so instead, we must use the less obvious (and hidden by default) command bar. Guess how you turn the command bar on without Googling it. If it's hard for you to figure out, being the tech-savvy individual that you are, imagine how much more difficult it would be for the average Internet user.



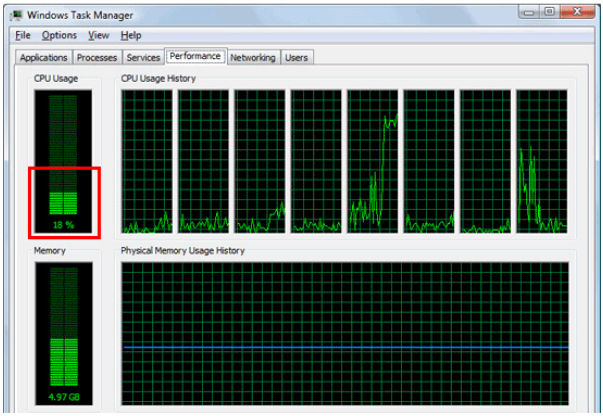
Making the "Refresh" and "Stop" commands as separate buttons is rather redundant; these browser commands are similar to light switches — you either need to refresh or stop the page from rendering.

These are just a few rather simple examples in IE9 that makes the browser's UI feel unpolished.

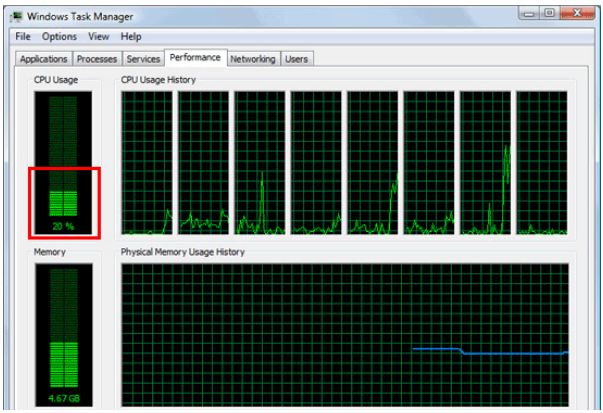
> *"Modern browsers enable rich, immersive experiences that could hitherto only be delivered through a plug-in or native application. They can blend video, vector and raster graphics, audio and text seamlessly without sacrificing performance."*

This seems to be pointing to technologies such as Flash and HTML5, which IE9 does rather well, especially with hardware acceleration and within the Windows platform where it can outdo even Chrome in CPU utilization (as shown below).

Internet Explorer 9



Google Chrome 10



CPU utilization of IE9 versus Chrome in Windows Vista while SunSpider is running.

> *"Modern browsers implement features when they are ready, providing predictable patterns that developers can rely on rather than suddenly breaking or removing specifications. They don't check off support based on a half-completed implementation written to pass a synthetic test, but validate against a test suite that confirms interoperability."*

While keeping older specifications in the rendering engine could be deemed a useful compatibility feature that some developers can rely on, I can't see how maintaining such old standards for the sake of those who couldn't be bothered upgrading is the mark of a modern browser.

The web has changed, as should standards. Continuing to support old specifications (just like old browsers) will only make compatibility harder to achieve in the long run.

If old code remains supported, then designers will have little incentive to innovate (unless they have a need or interest). As it stands, the industry already has major problems with poor-quality code, semantics and standards. The prospect of having to cater to an old generation of code could discourage designers from staying up to date in their knowledge. While this may not be a problem now, we could do without this stagnation, especially in proprietary technologies.

Another point is that — unlike IE8 — IE9 does not support Windows XP (an operating system that seems to have staying power of its own), which might keep the benefits of this modern browser out of the hands of many users (people on Microsoft's own platform).

As for Sneath's reference to the test suite that confirms interoperability, as we have seen throughout IE's life cycle, the number of test suite entries has never made a difference to the rendering stability expected of the browser. So why would it now?

> *"Modern browsers do adopt standards at an early stage of readiness so developers can experiment and validate the specification, but clearly delineate unstable prototypes as such."*

Finally, this rather non-committal statement underpins the root of my cynicism toward IE9 and why web designers will suffer for years to come. With every new version of Internet Explorer, trying to get users to upgrade remains a struggle. IE6 is still in use to this day. It's all well and good to allow "experimentation," but as these living specifications gain more adoption by mainstream Internet users,

browsers (like IE9) that have intermittent upgrade cycles will undoubtedly complicate the average web designer's testing process.

Why It Matters

My criticisms of the browser do not stem from some discontent about how it's turned on me; far from it. The browser has improved, and (like its predecessors) it will help us bring some modern features to a willing audience.

Is IE9 a modern browser? In my humble opinion, no, simply because of these issues of life cycles, patchy support and the fact that other browsers (while not perfect) are making innovative progress.

Web designers should be cynical of any new browser out there, and Internet Explorer feels like just another catch-up release with the same issues we've seen before. The Web is evolving, as are the tools and technologies that people use to access it. Only the browsers that meet such needs will survive.

Is IE9 a good browser? Yes. Is it at the same level as its competitors? Maybe. Will it remain at this level for its lifespan, and can IE survive to version 10? I'm not too sure.

For IE9, the future is up for grabs.

As for me, it's back to business as usual, patching and hacking my work for another Microsoft browser.

Sources:

- http://fu2k.org/alex/css/cssjunk/ie8/xua/xua-ieEmulateIE7
- http://blogs.msdn.com/b/ie/archive/2011/03/02/jquery-1-5-1-supports-ie9.aspx
- https://www.zdnet.com/article/mozilla-is-ie9-a-modern-browser-no/
- https://blogs.msdn.microsoft.com/tims/2011/02/15/a-modern-browser/

Progressive Disclosure in User Interfaces

As designers, we're always trying to get the most out of our interfaces and maximize whatever space is made available to us. While many solutions have been devised over the years, one above all others has consistently influenced the way visitors access the content they seek.

From simple techniques, such as tooltips and drop-down menus, to complex single-page websites powered by Ajax, progressive disclosure has become a formidable force.

This article explores the methodology of progressive disclosure and its impact on our interface design work.

What Is Progressive Disclosure?

Before we begin, let's quickly define progressive disclosure. We all know how damaging clutter can be on usability, so simplifying interfaces whenever we can makes sense.

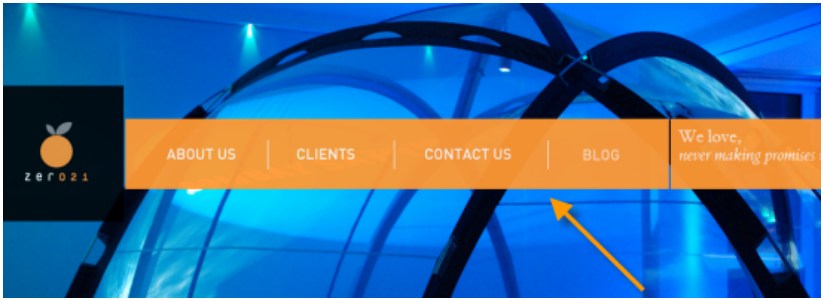
Progressive disclosure breaks content into smaller, more digestible blocks (showing them only when required). HTML elements, for example, can be hidden from users using the visibility or display properties in CSS; and scripting for events such as onclick can make things appear when requested.

By using these content presentation design patterns, we can keep our interfaces simple and help the user accomplish their tasks with greater ease.

The Good, Bad and Indifferent

The thing is, users can be quite exacting. Most demand power and flexibility in the tools they use to accomplish their everyday tasks — which we provide to them on the web with increasingly complex methods of delivery — yet users also want that power to be delivered with simplicity. For instance, many users may want feature-rich apps without needing to learn anything more than they already know (e.g., not having to read an instruction manual).

This dual need of complexity and ease of use has to be dealt with, even though it seems that both things are diametrically opposed.



Progressive disclosure can bring simplicity and feature-richness to a website.

The idea for progressive disclosure originated as a guideline in software design. When you click on an item in a menu, a dialog box usually appears with an array of carefully organized options. Disclosure happens not just in the dialog becoming visible, but also in the tabs used to move between categories.

On the web, this technique has come to be relied on because of the increased demands of users. Web designers will disclose parts of a page, offering secondary "advanced" options as required.

The benefits of progressive disclosure are these:

- Clean, simpler, more productive interfaces (a godsend for small displays)
- Important content is prioritized by giving the initial focus to the most common features
- Less important content is hidden from view so as not to confuse visitors
- Time is saved if scrolling is reduced and fewer refreshes are needed
- Fewer errors occur because novice users will take easier, more manageable steps

There are also a few disadvantages to consider:

- Accessibility can be tricky, as in the case of helping screen readers navigate to page sections
- Page-loading times could increase because of the size of preloading content
- Client-side technologies like JavaScript (Ajax), CSS3 or Flash could be disabled by the user (so you must take into account graceful degradation, which could add to the development time)
- Too many choices on a navigation path could be confusing
- Indexing by search engines and social networks could be affected by progressive disclosure methods (such as in the case of using the display:none; CSS property)

Progressive Disclosure Basics

In designing a site with progressive disclosure in mind, we should prioritize content into primary or secondary categories.

Primary content appears immediately in the normal flow of the page (and is highly visible).

Secondary content makes space for itself by doing one of three things:

- Taking up a part of the browser window when requested (as we see with content that slides down when requested)
- Replacing visible content in the layout (such as in tabbed interfaces)
- Overlaying over the primary content (such as in lightboxes/ modal windows)

Primary and secondary disclosure methods accomplish their purpose in different ways.

Progressive Disclosure Design Patterns

Let's look at some of the ways progressive disclosure shows up in design work. Because this technique has existed for quite some time, designers have come up with innovative ways to give content added depth and substance, while making readability and the learning curve easier on users.

As with any pattern, every website will have its own requirements, and you should investigate your options before renovating your layout.

Primary Progressive Disclosure Methods

Below are some primary disclosure methods.

Hyperlinks

Clicking on a link redirects us to another page (or we scroll to the target content in the case of fragment links). This might entail losing the visible content, but it's the most basic and recognizable way of navigating content.

Scrolling

Scrolling provides a way of disclosing content in order of priority. It could be as simple as ensuring that important content appears higher in the page.

CSS Media Queries

Different browsing devices will involve different requirements for content. A print stylesheet, for example, can remove unnecessary bulk (such as menus) and avoid wasted ink. Media queries, which was discussed in the article on responsive web design, can also help with this disclosure effect by taking away content from plain view for screen sizes that are smaller (which is the case with netbooks and smartphones).

Server-side Techniques

Languages such as PHP enable us to make requests to databases, external files and even other websites. As users request this content, we can mould an experience around them, rather than just showing just generic, static content.

Secondary Progressive Disclosure Methods

Below are some secondary disclosure methods.

Mouse and Focus Events

If you've made use of the :hover pseudo-class, you'll know that we can dynamically style content based on mouse events such as a mouse hover. A reaction to a hover event could involve displaying a tooltip, presenting tab content or presenting menus in dropdown menus.

A focus event (which can be captured using the :focus dynamic pseudo-class) can also help with progressive disclosure in a similar manner.

Conditional CSS

Two CSS3 pseudo-classes in particular (:target and :checked) can be used for progressive disclosure (see a proof-of-concept here).

Ajax

The technique of content-on-demand has seen a significant rise in popularity through the use of Ajax. Ajax allows us to request new content based on user decisions and then embed it without needing

to refresh the page. However, JavaScript is essential to this process (which can be problematic for a few users).

Pop-up windows

While many designers do not recommend them, pop-ups are another form of progressive disclosure. They bring up a new window or new content through some interaction with the current page. This method is less cleaner and more invasive than the others.

Modal windows

Progressive disclosure doesn't always involve swapping content (either temporarily or long term). You could display a lightbox, a step-by-step wizard or a dialog box, thereby putting the focus on the requested content.

Some Things to Keep In Mind

With all of these techniques available to us, we should highlight a few guidelines for making use of this dynamic form of content presentation. We've already covered the advantages and disadvantages of this method. Below, you'll find some practical advice on how best to eliminate or minimize any impairment of the user experience.

Keeping it brief, first decide whether a single- or multi-page layout suits your content (there is no harm in extending long articles and tutorials over multiple pages).

You'll want to ensure that the disclosed content gets the full attention of the user (for example, lightboxes fade the background to reduce noise).

Limit choice so that users are not overwhelmed with options and content, but give enough options for them to make an informed decision.

Never force an option on users.

One page or many? The question often faces designers.

As always, compatibility is critical to the process. The method you adopt should have good solid browser support. If you opt for a dropdown menu with hover events, then you may need to use anchor links to ensure that IE6 users see the dropdown menu. If you use Flash or JavaScript, then make sure alternatives are available for when they are not, which many web designers forget about.

All basic points, but important nonetheless!

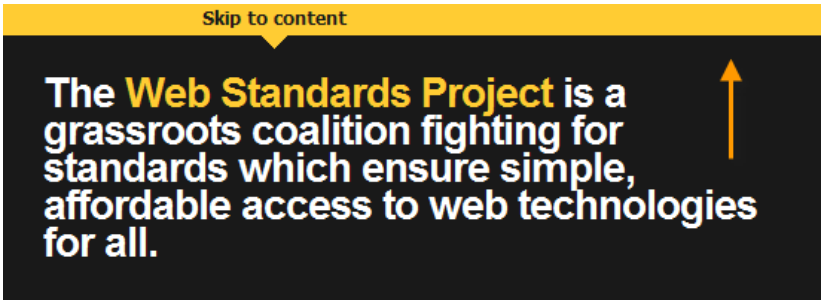
Users can disable scripting and Flash, so ensure that alternatives are available.

To avoid quirks with visibility, label your links so that users know what action is coming.

Structure and organize your mechanisms logically (having copy appear on the screen at random isn't helpful), and use calls to action to point users to key data (since the material won't be immediately visible, users need to know where they can locate the content).

Mystery-meat navigation is like playing guess who, without the fun.

Also, with regard to accessibility, ensure that screen readers can navigate around disclosed content effectively. Use skip links and label fragment identifiers clearly (because most screen readers will take this as an instruction to read part of the page, rather than just repeat everything).



Embedding skip links in the content and body helps orient screen reader users.

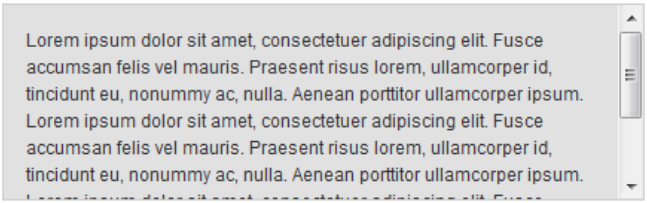
Finally, a note on performance. Technically speaking, progressive disclosure allows us to keep content hidden or off the page until it's required (which reduces page requests and inter-sectional downloads). But performance can take a hit on single-page layouts if content is merely hidden until needed by the user, because content that is not in use is being loaded. While Ajax alleviates this (via asynchronous requests), it's still a concern (Ajax can cause high amounts of persistent network traffic, which can slow down a website).

Showcase of Progressive Disclosure Examples

Here are a few examples of progressive disclosure in the real-world.

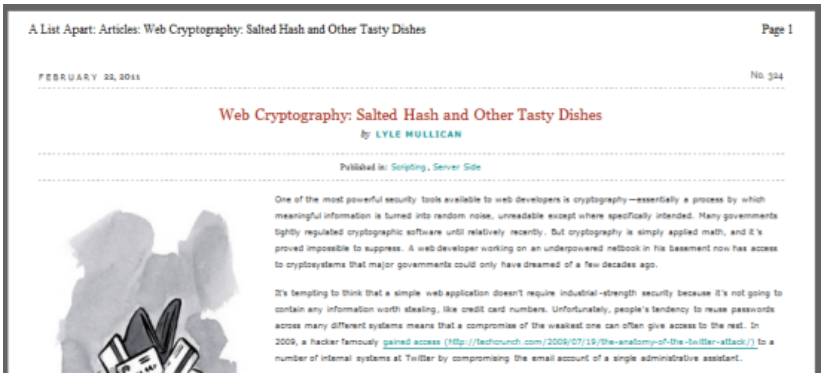
Hyperlinks are straightforward. Users simply click and go to a page with more content.

Traditional scrolling makes content below the fold of the canvas visible.



Mid-window scrolling can be initiated through iframes and the CSS overflow property.

Positional scrolling, to push users down the page, can be triggered with scripting.


A List Apart: Articles: Web Cryptography: Salted Hash and Other Tasty Dishes Page 1

FEBRUARY 22, 2011 No. 324

Web Cryptography: Salted Hash and Other Tasty Dishes

by LYLE MULLICAN

Published in: Scripting, Server Side

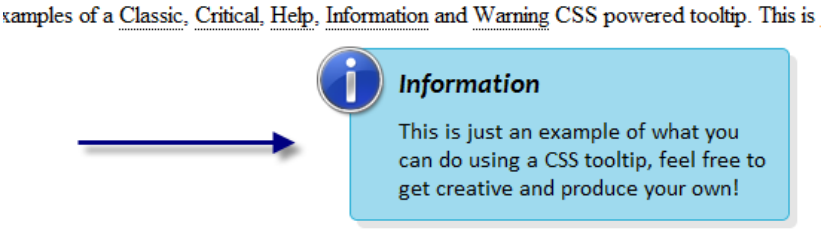
One of the most powerful security tools available to web developers is cryptography—essentially a process by which meaningful information is turned into random noise, unreadable except where specifically intended. Many governments tightly regulated cryptographic software until relatively recently. But cryptography is simply applied math, and it's proved impossible to suppress. A web developer working on an underpowered netbook in his basement now has access to cryptosystems that major governments could only have dreamed of a few decades ago.

It's tempting to think that a simple web application doesn't require industrial-strength security because it's not going to contain any information worth stealing, like credit card numbers. Unfortunately, people's tendency to reuse passwords across many different systems means that a compromise of the weakest one can often give access to the rest. In 2009, a hacker famously gained access (http://techcrunch.com/2009/07/19/the-anatomy-of-the-twitter-attack/) to a number of internal systems at Twitter by compromising the email account of a single administrative assistant.

Print style sheets disclose content, hopefully without the junk!

Media queries can contract and expand content, catering to different device

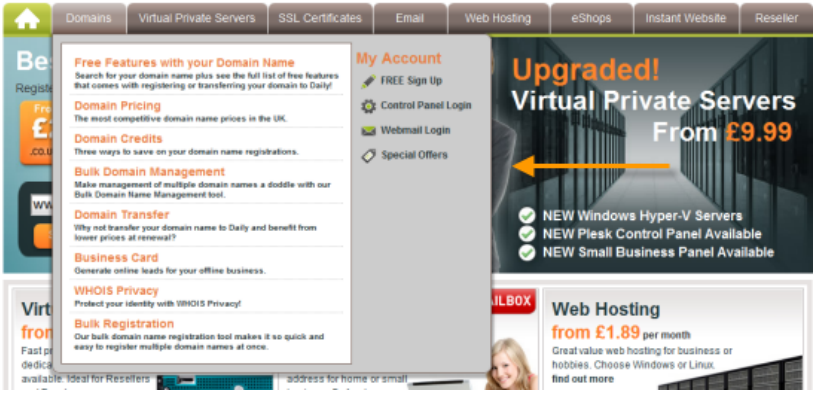
Amazon has a database of products and discloses them via server-side requests.



Tooltips disclose content temporarily and can be quite elegant.

Navigation tabs can be used to group and break sections of a page into categories.

Dropdown menus in web forms can be implemented not only via lists, but with option inputs.



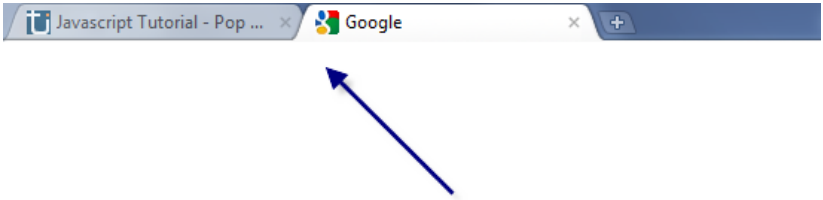
Breakout boxes (slide-out menus) can present a variety of options in a structured way.

Using the target or checked pseudo-class gives the impression of multi-page depth.

JavaScript methods of altering visibility create effects similar to CSS3 but with the advantage of compatibility.

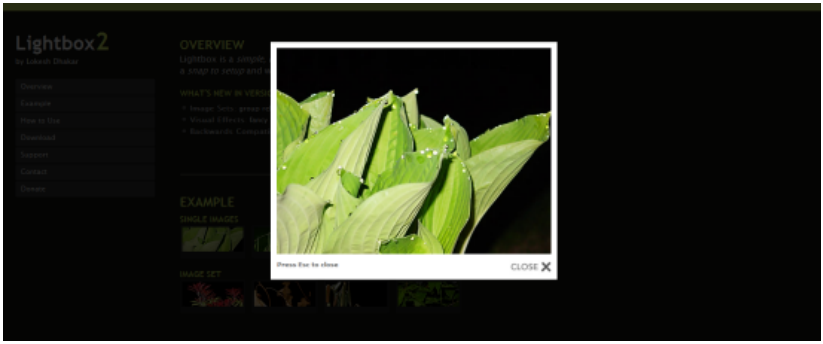
Panels can be produced by swapping content. (The event can even be a timed.)

Some websites make use of AJAX requests to speed up the appearance of content.

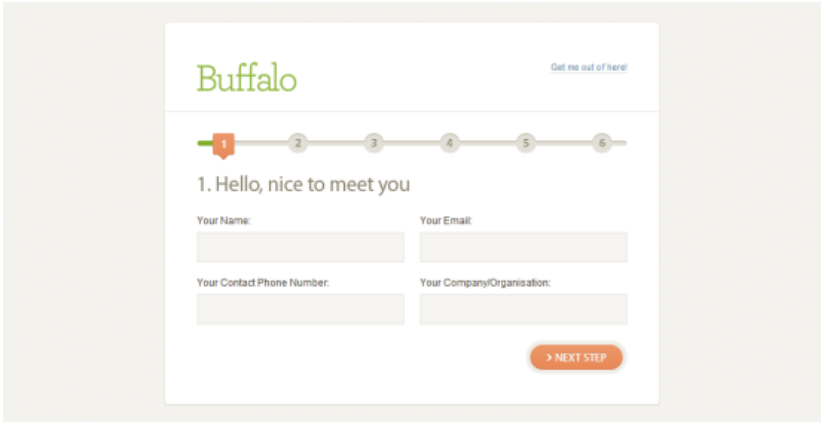


Pop-up windows aren't usually welcome by users, but they are a form of disclosure.

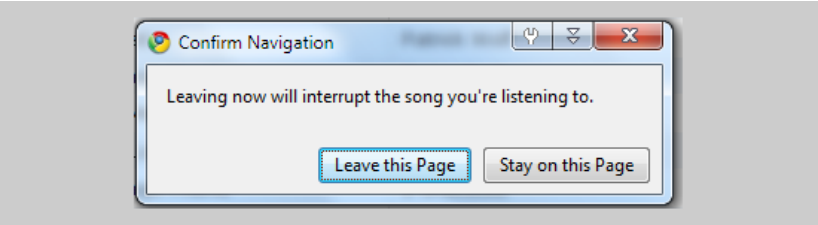
Websites can do amazing things with Flash. Just a shame about all the disadvantages.



Lightbox effects exist all over the web. They hold small individual items beautifully.



Step-by-step wizards can break up progressively disclosed content into digestible chunks.



Dialog boxes and alert windows provide a basic level of detail for decision-making.

Summary

There are many different ways to display content on the screen, and progressive disclosure is one of the most exciting. It increasingly frees us from having to refresh pages and from navigating to a unique location just to see something change slightly; it even allows us to simplify complex aspects of a website.

Enhancing a website is straightforward, but maximizing the available space can be quite a task. As you produce designs for clients or yourself, consider the impact that disclosure could have on your workflow. Perhaps a lengthy web form could be made easier; perhaps you could use lightboxes to better distinguish useful content on the page; and there is always the option of just giving more (or less) choice to users in order to reduce information overload.

However you disclose your wonderful content to visitors, give it some consideration and be creative!

Sources:

- https://en.wikipedia.org/wiki/Progressive_disclosure
- https://www.webstandards.org/
- https://alistapart.com/article/goingtoprint
- https://2018.ampersandconf.com/
- https://www.amazon.com/books-used-books-textbooks/b/ref=nav_shopall_bo_t3?ie=UTF8&node=283155
- https://www.daily.co.uk/

Effective Communication Tips for Web Designers

We lay increasing importance on doing things in the user's best interest and meeting their expectations, but we often forget that content and design is the window to a website's soul.

Our designs tell visitors something about us and build emotional bonds to brands through first impressions and reputation.

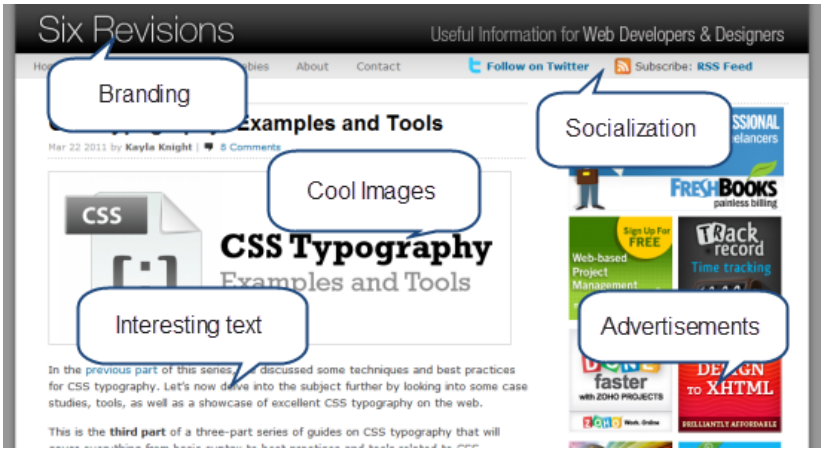
By taking advantage of communicative design, we can better engage audiences and more effectively serve user needs.

Reactive Design: Beyond Simple Objects

In digital communication, computers act as a portal between our input and output devices. How they display websites to users and how those users respond indicate exactly how communicative our work really is.

I refer to this process as reactive design. We produce something that is presented to our users, and users react or respond to that.

Forget email: your website speaks volumes by itself, and as you progress through the iterations of a website (more on this below), you'll find that an improved layout can make all the difference.



If you examine a website closely, you'll see that it attracts the user's attention in multiple ways.

Of the many factors in website design, content is a big component. If we have a block of text that explains the service we offer, that is communication. If we stream useful audio content to visitors, that is communication. If we provide interactive training or videos, that is communication.

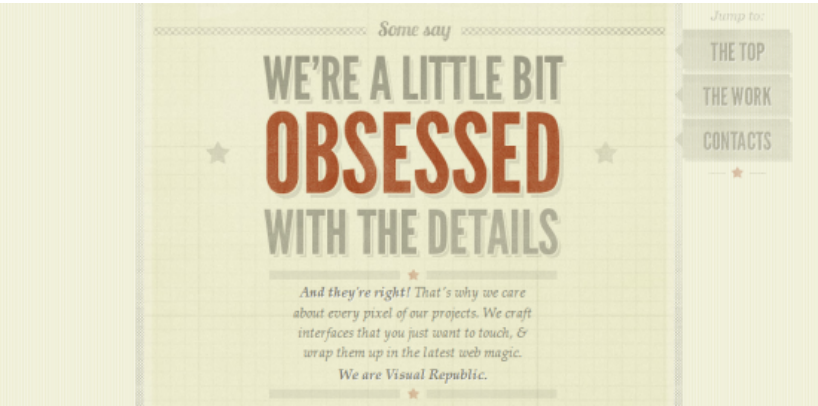
Website content is a repository of words, visuals and sounds that trigger the senses. Don't let all of that power go to waste.

Tips for Optimizing Content

To help you optimize your content, here are a few tips:

- If applicable, inject some humor into your work; it can liven up dry boring discussions.
- Break long pieces of content into smaller segments (users can handle only so much data).
- Beware of sarcasm; it can be quite hard to interpret.
- Images can support text very well, so don't be afraid to use them to get a point across.

- Don't condescend; produce technical content at a level that visitors will understand.
- Always label menus and links; confusion often begins when choices aren't defined.
- Personalize what you write; use testimonials to give perspective.
- Be direct and to the point.
- Moderate all forums and blog comments, because spam will reduce credibility.
- Always keep your content up to date, and sustain the conversation with users.
- Having one active voice on a website is okay, but user involvement adds depth.
- Video is a wonderful communication aid, but don't forget about accessibility.
- Don't just tell people what to do: show them with video (especially good for visual and practical learners).



Inject some fun into your content. Stale, boring material is often skipped over!

Enhancing Visual Communication

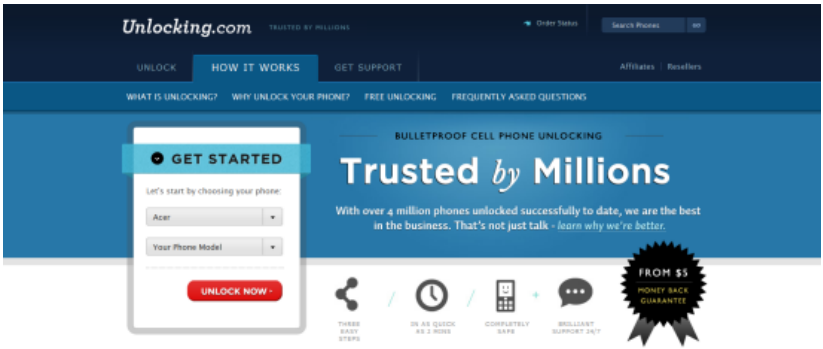
Next up, we have the visual design or layout of a page. While more abstract and open to user interpretation, we do know that design affects usability.

The very fact that usability plays a part in our layouts says something about its value and importance. If visitors can learn to use an interface and find some easier to use of than others, it stands to reason that design speaks to users through association, recognition and other types of non-verbal communication; they can see something on the screen and either know what it does or can find out.

These tips can enhance your website's visual communication:

- Use color to enhance a website's appeal; color association can be quite strong.
- Your logo represents your identity; it should be professional and high quality.
- Icons are your best friend. They provide recognizable points of reference.
- If you want your website to look professional, keep visual clutter to a minimum.
- Your website layout should be compatible with browsers and devices.
- It takes less than a second for a user to make a judgment; try to impress rather than shock.
- A design should represent your content and services; don't use templates without customizing them.
- Size matters! Large font can draw attention to important bits of content.
- Don't be afraid to make the website fun and quirky; just be sure it's not too distracting.
- Design patterns put users at ease because users are able to recall features from memory.

- Visibility is central to communication; give users the cues they need to scroll.
- If you redesign the site, keep the things that work! Users hate having to relearn skills.
- An attractive layout will get a user's attention, but make sure there is substance to maintain it.
- Advertising is a classic form of communication, but too many websites overdo link placement.
- If your website promotes a physical location, match the designs so that visitors know what to expect.



A well-structured, clean, organized website gives a feeling of professionalism.

Enhancing User Interaction

While thinking mainly about content and appearance are fine for the average static website, we also have the big factor of interactivity, which many modern designers offer these days in order to create dynamic, engaging experiences.

Consider JavaScript or PHP: in their own way, they encourage users to talk to the website through clicks, key presses and data entry.

In fact, users actively speak to websites whenever they fill out their profiles and user other personalization features. Giving the user

attention, recognition and a record of their contributions helps to create a sense of community.

These tips will help you engage with audiences and harness the power of interactivity:

- List other content that readers might enjoy; visitors like to window shop.
- If you have user profiles, have your website address users by name, to add a sense of personalization.
- Quizzes and games reinforce memory, and they could be popular tools for content.
- Letting users customize the theme or layout could help them digest content.
- Every action should have a reaction: if a user enters incorrect data, inform them immediately.
- Clickable regions should be friendly; make sure they are large enough to accommodate interaction.
- Take readers on a journey by progressively disclosing content to them.
- RSS feeds help infrequent visitors stay in touch with a website.
- Exposing old content to users can lengthen their visit; offer related (but useful) content.
- Web forms that are quick to fill in and non-invasive discourage users from entering fake data.
- Supplementary content can be distracting; allow users to turn comments on and off at will.
- Mobile device users interact with websites differently, so adjust your design to match, perhaps by using responsive web designs.
- Don't include widgets (such as clocks) unless your website really needs them.

- Use confirmation dialogs for critical actions, so that users have a chance to rectify errors.
- For complicated steps, provide a wizard to guide users to their destination.

Good functionality not only helps users achieve their tasks, but strengthens the message.

Going Beyond Protocol

While subtle improvements to the message of your website will of course have a dramatic impact on how engaging and appealing your content is (especially if the subject is a bit dry), we can't forget the other form of communication that our websites facilitates, and that's the person-to-person (or direct) method. Skype, instant messaging, email and even IPTV all present an opportunity to both target an entire audience and provide something a bit more personal to each user's needs. Service providers often fall short of satisfying the latter.

Below are some tips to improve direct communication with users, whatever the medium:

- Confirm that the user's message was received (and provide a time frame for response).
- Always respond promptly; the longer users wait, the less professional you'll appear.

- Address frequent issues on your website to avoid answering the same questions.
- If something can be automated (like account closures), do it; it will usually make things easier for everyone involved.
- Provide live assistance only if you can handle the workload; don't make users wait for hours.
- Chatting in person over Skype avoids the back and forth of email, and it saves time!
- Never ignore users, even if the request seems silly. Just thank them and note their issue.
- Critics are worth listening to because they want to help you do better. Just don't start flame wars with them.
- Transparency is critical to communication; admit your faults, deal with the issue, and tell users.
- Problems occur, but what matters is how you deal with the issue, not the problem itself.
- Don't send users information that doesn't apply to them or you'll be labeled a spammer!
- Give users a simple point of contact; helping them helps you be a better provider.
- Don't be afraid to bring the community together; forums allow users to assist each other.
- Comments add to discussions; they bring out unique perspectives and useful content.
- Allow users to remain anonymous if they prefer to; communication should not be mandatory!

Offering live support will help you appear more responsive and attentive.

Golden Rules for Communication

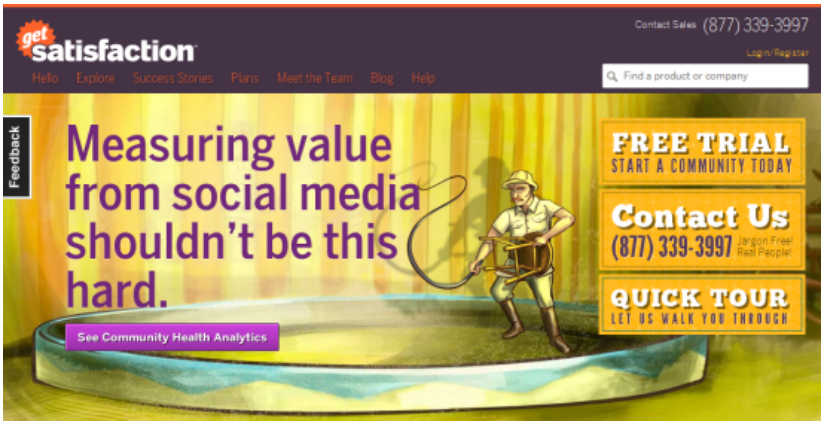
there are some golden rules that can help us maintain the highest level of communication with users. While there are probably more, I've narrowed them down to the top five best practices:

- Be considerate
- Be prompt
- Be helpful
- Be inclusive
- Be friendly

Improving Communication with Customers and Clients

Because we are first and foremost designers (rather than consumers), our communication must cater to two types of individuals: our visitors (or customers) are the ones whose attention we have to get, while our clients are the ones whose goals we are serving.

Be under no illusion: the visitors should always come first (because without them, a website is pointless). But as designers, we have a duty to those we design for!



Visitors and clients are different beasts, and we're responsible for connecting them to each other.

Communication between visitors and clients comes with different variables to consider. When we build a website for a client, our focus is often on their business objectives (things like cost, scope, etc.).

Ensuring that both the client and visitors end up with something that encourages the client to retain our services is a bit of a balancing act. If you want to be a web designer, you must be able to communicate through your portfolio and other direct methods.

Below are some tips and tricks for working with clients, bosses and other third parties:

- Tell people who you are; individuals posing as faceless corporations are impersonal.
- Clients want to know what you know; always state outright what you provide.
- Price is everything! Most things in life come down to numbers, and so estimates are important.
- Your portfolio does not need to be big; it just needs to make a positive impact.
- Avoid jargon because other professionals will see through it.

- Maintain confidentiality; it's not the world's business to know how your last meeting went.
- If you can't convince others to design how you think best, make a better case.
- Be confident, and be willing to compromise; business is all about give and take.
- Never be afraid to say "No" to offers that look suspect; you might save yourself some trouble.
- Opportunities come to those who work hard; do your very best with every project!

Here are tips for communicating with customers:

- Customers want to reduce the risk of purchase, so offer trials if possible.
- Money-back guarantees inspire confidence and are an incentive to purchase.
- Provide frequent customers with offers, discounts and benefits for using your service.
- If the website is for someone else, make sure the experience matches user needs.
- Every audience has different needs, so make the layout fit them.
- Visitors want to be gratified; your website should be well-rounded and purposeful.
- A blog can be useful for keeping users up to date, but only if it's updated frequently.
- Version history logs are useful for tracking a service's evolution; maintain one online.
- If your service relies on a third-party product, consider the consequences if that product disappears.
- Downtime is a big issue on the web; do your best to keep the website going during traffic spikes.

Final Reflection

The key to successful communication is to think beyond the items on the page. Our users want to be engaged and feel that they are welcome and special.

Our ability to communicate through content, code, visuals, interactivity and just about anything that can be crammed onto the page speaks volumes. What does your own website's content say about you? How could you better converse with users? It really does matter!

How a website reacts and interacts with users and caters to their needs has repercussions. Consider how poor websites abuse users' trust (through privacy violations), confuse their message (through poor usability) or simply neglect users (through inadequate accessibility); the damage will foster a negative attitude toward the services.

As designers, our job is to make our websites communicate with attitude, professionalism and (perhaps) a sense of humor. Anything that keeps a reader's energy level up is more than welcome.

And as we try to out-do our competitors, we need to realize that what matters is not always what we say, but the way we say it!

Sources:

- https://www.unlocking.com/
- https://www.liveperson.com/

Designing for Different Age Groups

Diversity is one of the things that make the web great, and every audience has its own needs and requirements. But what happens if that audience is comprised of a specific age group? Are you providing something fun and interactive for kids, or are you strictly an adult-only website (such as one that sells alcohol)?

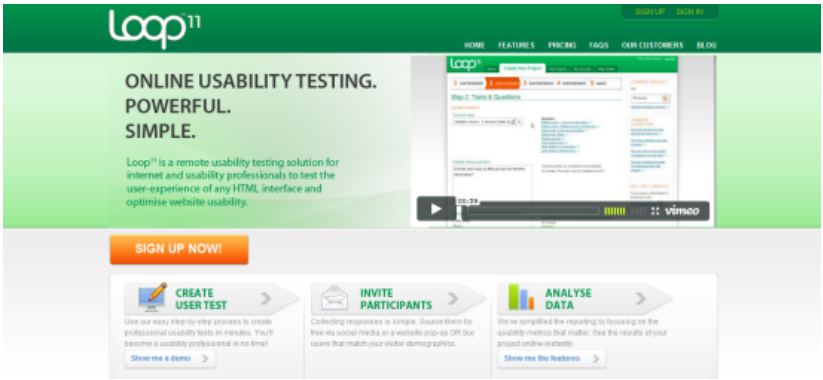
Age is an influential factor on the web in terms of not only psychology, but also accessibility, usability, and user interface design. Many other variables can affect your designs, but we'll focus on the difference that age can make in creating a website.

From 0 to 80 in Under 5 Seconds

The differences in how various ages use the web have never been starker than they are today. Because the web has become so integral to many people's lives, a sort of age gap has arisen where different generations of users have developed different abilities on the web.

While much of our work depends on generalizing about age groups (and not everyone will fall neatly into one of them), our understanding is based largely on sensible, logical guesswork.

So, as long as you take the time to know how your target age group is affected, you should be able to dodge the pitfalls of catering to fringe users.



As usual, research is the order of the day. Study and analytics to the rescue!

Consider today's older generation, many of whom are only now logging on for the first time. There are so many books that teach senior citizens how computers and the web work; there is almost a small society of people playing catch-up with this newfangled invention that we design for daily.

At the other end of the spectrum, we have infants and young children who will have never known a world without the web and who are learning the concepts and gaining dexterity online as we speak.

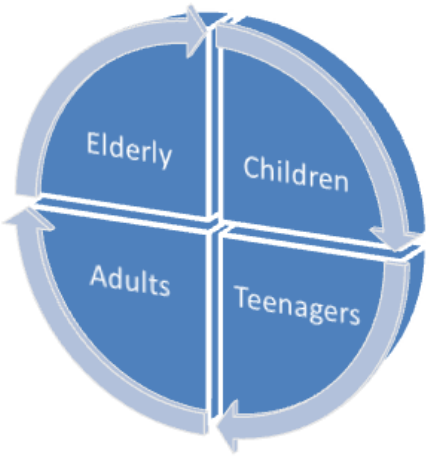
These people are real, and you have to make sure your website serves their needs.

Why It Matters

Designing for different age groups is important for two reasons. First, ignoring an entire user base such as the older generation alienates them from the experience. And let's face it: all of us will get old, and we wouldn't want to be treated like that.

Secondly, younger users will be tomorrow's designers (or, from your client's point of view, tomorrow's customers). If they stumble upon your website and have an awful experience, that will likely stick with them and could shape their perception of the website or service.

(However, access to some websites, such as ones related to alcohol and adult content, should be restricted.)



The four age brackets for web design (although you should expand them as required).

While this article distinguishes between children, teenagers, adults and the older folk, it's worth noting that the difference in computing ability between a six and a ten year old will be dramatically different, so you can't take anything for granted.

The best approach is to define your age bracket (and the younger the audience, the narrower it should be). With this in mind, let's look at the first age group and its implications for your website's design.

Designing For Early Years

The impact of websites is most heightened with children. When the web was young, the education system saw computer skills as a luxury. Times have changed, and the skills have become central to our society.

Ten years ago, the average 10 year old would have quite limited computer skills; this is no longer the case. Through early interaction

with the web, children as young as five and six (even younger) are gaining rudimentary experience with devices and websites.

But there is a variable that affects their experience more than education: physical development.

Very young children may find computers challenging at first. Image source: creactions

While computers have become ubiquitous in early education, children's bodies and brains are still developing. Knowledge that adults take for granted may be limited. Their motor skills and ability to use mice and keyboards don't match ours. Our layouts need to account for such limitations.

While adults may have some patience for errors, young children have less (or in many cases, lack the knowledge to overcome them).

Nevertheless, designing for children has a different advantages to designing for adults. Young children want to be entertained and don't necessarily have a direct goal in mind, which gives us the opportunity

to engage them through exploration and interaction (rather than just putting them on the fastest route to a solution).

If the journey is colorful and educational and engaging, then it will likely be a successful visit.

Children tend to play minesweeper with links, just clicking them to see what happens. But when they find a route that works for them, they are more likely than adults to stick with it; a trait referred to as learned path bias. But they are also (paradoxically more flexible in their ability to adjust to new environments).

Children's websites should be educational, entertaining and clutter-free.

So, how do we make our websites child-friendly? Best practices include:

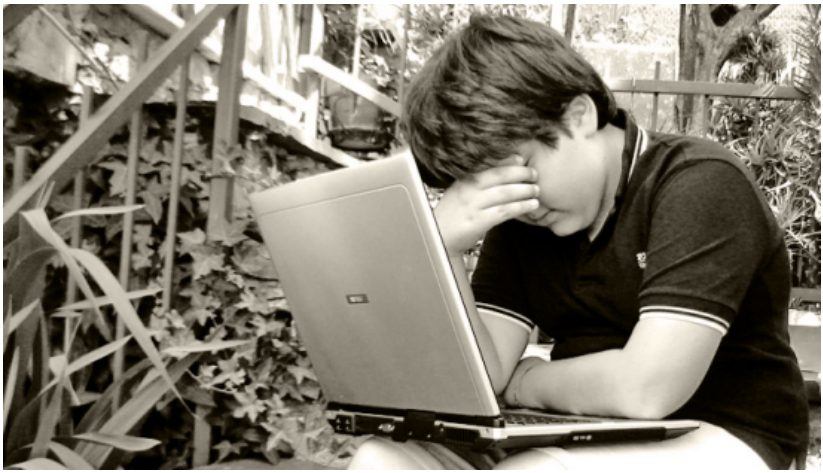
- Keeping the UI clean (children get distracted by visual clutter)
- Using iconography (they identify with experiences that are recognizable)
- Using vivid, exciting colors
- Avoiding integrated advertisements (kids find it harder than us to distinguish content from ad banners, which quickly lead them away)

- Consider using animation and sound (this is the only age group for which video seems ideal)
- Relate content to characters they know (like from TV)
- Provide games that educate and attract their attention
- Reinforce their actions through emotion (telling them that they did a good job encourages repetition)

Designing For Tweens and Teens

As children grow up, motor skills and comprehension become less of a limiting factor. Older children and teenagers often gain experience with computers through school homework and recreation (for example, on social networks), although this doesn't necessarily mean they know how computers work fully.

Technology is generally more prevalent with teenagers than with children; although even very young children now have mobile phones and laptops, albeit they may monitored by parents. Patience levels also increase.



Teenagers rely on technology to keep up with friends and for homework. Image source: duchesssa

Teenagers (and tweens) tend to be more resilient to targeted advertising and are less willing to explore websites (adopting a more methodical approach: seeking rather than discovering information). Though still engaged in reading information of interest.

Research indicates that the major difference between this age group and adults (and children) is that teenagers are more socially focused. While adults tend to use technology to achieve set goals, teens are preoccupied with interacting socially, being heard and partaking in group activities (such as in online forums and social networks). This gives designers an opportunity to engage with this audience directly.

While not all teens are the same (and despite some adults believing they are an entirely different species), design choices have more of a chance of affecting a much greater portion of this user base. Consider how these socially inclined users can contribute to your website and how they might spend their free time using your service and promoting it to their friends. Also, think about how open they might be to new experiences, not being so tied to learned behavior.

Following popular culture is key to attracting teens.

Making a website teenager-friendly means:

- Keeping the UI clean (a factor common to all ages)
- Favoring graphical content to textual content (teens tend to read less online)
- Using animation and sound (moderately, though — not as much as for young children)
- Ensuring that the content isn't so simplistic that it appears childish

In addition, research shows that teens have the same learned path bias as children, are more easily distracted by interactivity, are more social online, and are more driven by social trends (fashion and peer interests hold sway in web usage — there is power in numbers).

Designing For Adults of All Ages

Of course, people who have been alive since the web first reached a critical mass comprise a large proportion of our user base today. Many designers look to them for usability testing and when assessing whether their work serves the audience's needs.

This can be problem, though, with increasingly diverse audiences needs going unaccounted for.

Adults are seen as the average web user. Image source: rajsun22

As with younger and older users, most adults have at least moderate experience using computers.

But that isn't to say that all adults are computer literate. While most adults have computer experience, only those who are very interested in technology tend to understand how it works. For example, a Google survey showed that 90% of people didn't know what a browser is, despite being able to use one.

Adults tend to be at their peak in dexterity and motor skills. Accessibility is still an issue, but most adults are at a stage when they aren't so dependent on instruction, have little trouble making choices and don't need advocates for their needs.

Adults in general are goal-oriented and tend to visit websites with explicit objectives (relying on search more than discovery), and they are usually more accustomed to (and forgiving of) quirks in the user experience.

But this comes at the cost of being less focused on social interaction and being averse to advertising (they filter out noise while scanning).

Usability and accessibility are critical, but that doesn't mean the website can't be clever!

Tailoring a website to adults is generally straightforward. If it is accessible and usable by modern standards, then it will likely be useful to them.

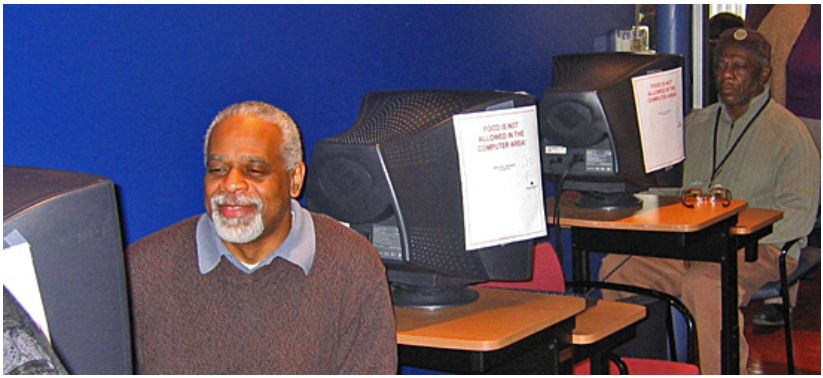
Unlike younger users, adults are much less drawn to animation and sound (favoring text over visuals).

Unlike older users, they put less value on research and study and more on getting answers as quickly as possible.

Ironically, then, it is more difficult to engage this age group than others.

Designing for Later Years

Older folks get it the worst with regard to targeted design. While much research has been done into human behavior and HCI for children (not to mention the investment to educate children in computers), the expanding age group of seniors — who are more familiar with a world without the web than with it — seem to be less catered to.



Older users tend to be the ignored. Image source: EPA Smart Growth

Seniors tend to experience a decline in dexterity and motor skills, which affects website usage. Many of them may be using the web for the first time, and because their developmental years were at a time when computers and the Internet weren't part of mainstream society, they're less likely to take on the technology as fast as other generations. In addition, the aging process means a decline in health (both mental and physical), which can affect online interaction.

This isn't always the css of course, many older individuals are competent with technology to revel their younger generations. It's important to take variation into account with statistics.

Being older does have its advantages, though. Unlike adults, seniors are often focused on achieving set tasks, while still being open to explore websites and sometimes having more patience than children and adults.

In addition, they tend to be more focused on interaction and are more willing to research, read, learn and involve themselves in communities.

And having more life experience, they may have an advantage in solving problems and parsing technical content. Designers will certainly appreciate having an audience that is more likely to appreciate the nuances of what they provide.

Some older users can find adaptation to change difficult, so make sure everything is visible, clutter-free and well labeled.

Best practices for designing for elderly users include:

- Making websites highly visible and highly memorable
- Text should be large and easy to read
- Links should be easy to click
- There should be little animation or movement that might be distracting
- Website navigation should be straightforward

Older users are open to having an emotional connection to a website; they are more likely to form strong opinions, and they are more susceptible to the effects of a negative experience.

When designing for this group, avoid putting the onus on them to correct errors, cut down on confusion as much as possible, and encourage social interaction through an engaging UI.

Age Matters in Web Design

Whether you are building a website for children, adults or the whole family, age affects how it will be used and perceived. Young children are still developing in mind and body, and seniors encounter issues of their own. Teenagers and adults have particular objectives when browsing the web and interpret information differently.

Only by looking objectively at who will use your website can you hope to attract the widest possible audience.

One of the central principles of web design is usability, and while it would be incorrect to assume that all of your visitors have the same ideas, goals and perceptions, we still have to generalize to some extent so that we can make timely decisions.

Keep your website accessible to the silver surfers, meet the criteria for adults, keep teenagers engaged, and make your work child-friendly.

Each internet user is unique, but several generations of users may want what you offer.

Sources:

- https://www.loop11.com/

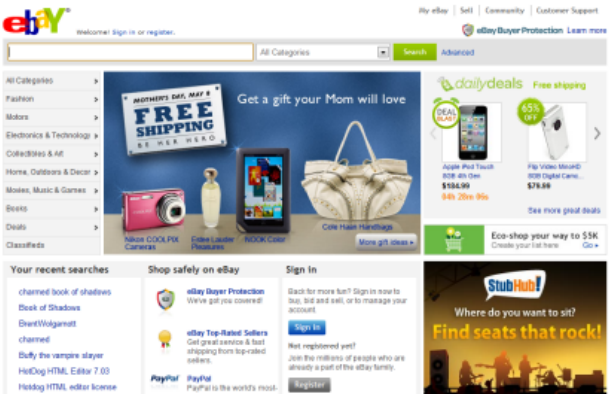
Smarter Web Designs: Responsive and Customizable

The way we design websites has changed profoundly in recent years. We have more information on how web users interact with user interfaces, we have developed many testing methods for evaluating usability, and we now build sites with great emphasis on user-centered design. In addition, research in the fields of psychology, sociology and usability has enriched our understanding of our site visitors.

Yet, while methods of designing for devices, the choice of browsers, and user demands all have increased, designers still tend to shy away from providing users with an experience that suits the ideology of truly responsive design — smart designs that fits the unique personal traits and preferences of a user.

Intuitiveness: The Double-Edged Sword

If you've ever conducted a usability research study, you'll be well aware of the issues people face when presented by an out-of-the-box UI that don't abide by conventions or trends. The truth is that knowledge — and more specifically, the ability to recognize common objects and their function (such as a button or hyperlink) — is the root of the issue. Knowledge is easy to come by, but it takes time to instill. Every experience comes with a learning curve.



Designing for navigation, links and content is a non-stop juggling act, as shown at eBay.

On the other hand, an intuitive design also empowers users. The user feels that they have the ability to understand the environment and deduce things for themselves. Empowering users is a critical part of the web designer's role. We can only hope that users know which links are clickable and what button to press to submit a web form.

So, here we are in this position: Many users struggle with the usability of our websites, and we do all we can, generally, to accommodate everyone's needs.

Nevertheless — and all too often — little is done to level with individual users because of the up-front costs of building flexible, customizable interfaces.

At least, that's what used to happen.

These days, we invest time and money in building smartphone apps, we build dedicated websites for portable devices, and some of us still provide support for Internet Explorer 6.

Yet, when it comes to interaction, we let our standards slide; we don't seem to put enough effort into crafting them for the site audience's needs.

Design Smart, Not Static

If we're trying to learn what our users want, we should leave customization in their hands.

There are all sorts of general usability tests, but we need to consider which users to include, what to improve, how to implement it, and much more.

To put it simply, users know when something isn't right about their experience, and even if they have the time to bother telling you about it, they don't always know what they want or how to express it in a way that would make sense to others.

They know there's a problem, but they may not know how to solve it.

Visitors idealize. They aren't trained designers and are liable to request solutions for what may not be problems in the first place, yet we are the ones who make the final decisions.

Design is less about fighting fires and more about empowerment; and smarter web designs are becoming increasingly aware of that.

Usability testing isn't bullet-proof, but it helps users help us to help them! Shown above is the Silverback usability testing software.

In the movie 10 Things I Hate About You, there is a wonderful quote that sums up our frequent attempts to use our psychic powers to build the perfect website for the default user: "You're 18. You don't

know what you want. And you won't know what you want 'til you're 45. And even if you get it, you'll be too old to use it."

If we base design decisions on assumed knowledge of what users want, we can't empathetically suit our websites to each user's needs.

However, add responsiveness and customization to the mix, and our designs will become easier to use. They'll become smarter.

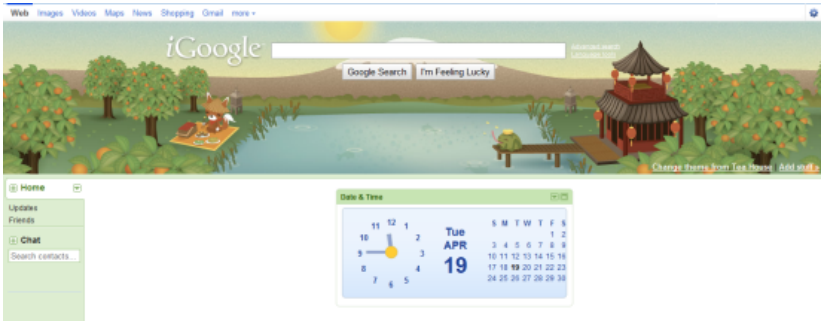
Responsive and Suggestive Methods

Users don't necessarily understand what they need, and designers typically aim to please the masses. We need to remedy that.

Design patterns and techniques exist to help us interact with our users and make our work truly responsive to an individual's needs.

We know that behavioral engineering is powerful and that user-centric design aids usability.

We also know that every user has different requirements or preferences for an interface. We already create responsive web designs that change depending on whether they're being viewed in smaller screens (like mobile devices) or bigger screens. The next logical evolution of Web Design is being able to predict what users want and to allow users the ability to customize their experience.



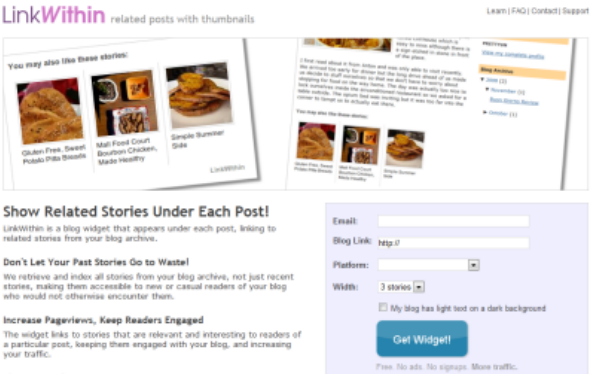
Smart designs are customizable and useful to visitors and they're built with efficiency in mind, as shown by iGoogle.

Building smart web designs — in concept, but not necessarily in implementation — is simple. Look beyond aesthetics, and think about how users want to interact with your site.

Personalize the content. Enhance the user experience based on user activity. Observe what users do on the website, and try to provide useful signposts and suggestions in the interface.

Presenting Related Content

Knowing what interests the user is half the battle in design. If visitors read and enjoy an article, they might be open to visiting something else that's equally good.



Links to related material serve as incentives for users to continue their journey on your website. LinkWithin is a tool for showing related posts.

Provide calls to action that direct visitors to the latest content, and provide links to useful material on the same subject. Show keyword tags (or categories and keywords) to identify similar places of interest. Every little bit helps.

Relative Navigation

This is important for users entering your website from another website (such as a search engine or social news aggregator); it helps them find their way around with as few clicks as possible.

Offer human-friendly site maps, with helpful directions. Provide useful error messages, and offer breadcrumb navigation for content categories.

Content Customization

Many users don't want to view our websites in the way we present them. Users expect to be able to format your work to their taste and, in that way, they can have a personalized site experience. Allowing your content to be viewed through RSS, for example, is a simplistic example of allowing your users to access content through their preferred means.

Smarter Search Features

The search box is the most recognizable way to help refine what users want to see. Allowing advanced searches could help users find what they're looking for quickly. Don't be afraid to include auto-completion or to suggest corrections of search terms to respond to errors.

Customization Methods

The second type of enhancement we can provide helps to create dynamic websites. With it, we can offer user-specific customizations. When someone signs up for a service, they expect it to be personable and meet their needs. Each of us consumes web content in our own way.

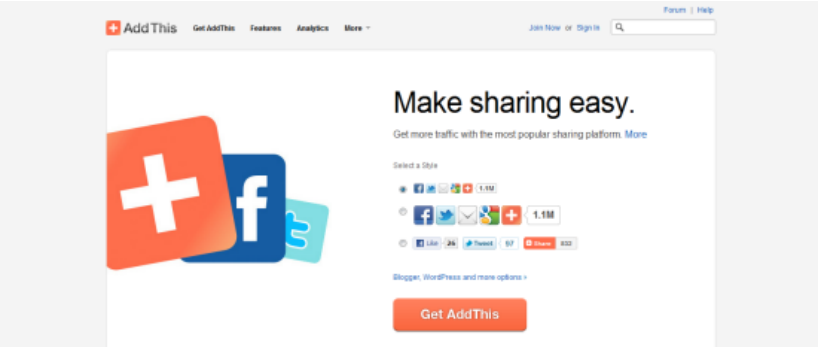
Basic ways to create smarter designs include offering custom themes or multilingual translations and allowing panels to be moved around the page (or added or removed entirely).

Profile Personalization

If you have a system in place that allows for user accounts, allow them to go wild with customization. Greet users by name, show them what has been updated since their last visit, offer subscription-based email or RSS notifications, and allow them to modify the site layout (as above) in their account as a "saved" state.

Provide Opportunities for Contributing

This technique is widely used in content management systems and social networks. Get users to contribute by recommending articles to them, inviting them to post follow-ups or comments, and offering customizable widgets or visual shortcuts (based on their preferences).



AddThis lets users recommend your work on the social network of their choosing.

Some Responsiveness and Customizability Ideas

Finally, there are "in the future" solutions that could increase the chances of our websites becoming more responsive (even to the point of being self-correcting). By taking advantage of scripting and by monitoring our users' actions autonomously, we could make websites smarter and more malleable.

Variable Designs

We could build a responsive framework to analyze what devices visitors use and then provide suitable environments for them. We are seeing it begin with CSS media queries and platform-unique layouts. But have you considered blending the techniques by scripting "detect and redraw" (rather than "redirect")?

Layout Profiles

Another technique for personalizing the experience is to allow users to build a visual profile. That is, let them choose what appears on the

screen, and where. Then let them save it, and save it publicly, so that all can see — which will enable users to share and apply one another's themes and gain unique tailor-made layouts.

Customize Content Presentation Based on User Activity

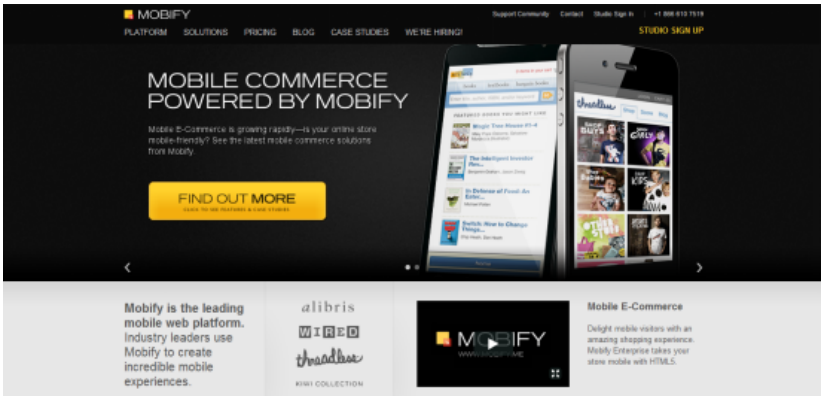
Here's the most radical idea for smartening up your designs: you could write a script that monitors user interactions — such as what links they click on and how they flow through the site — but use that data to identify what content to show the user, and redraw the layout so that useful content becomes more visible.

In effect, the website will train itself to become more useful for its users.

The BBC offers a powerful "launch pad" platform for visitors who want to read its content.

The Smart Website

Behavior is a powerful indicator of the user's process and progress of negotiating through a website. We place increasing emphasis on studying patterns and adjusting our pages for a variety of devices. Strangely enough, for all the energy we put into testing and site improvements, we don't create scripts and frameworks that empower users, whether empowering by recognizing their habits and adapting accordingly to make the experience easier or to increase returning visits, or by letting them change things themselves.



Mobile browser detection isn't the limit of our ability to improve designs for users. Shown above is Mobify, a mobile web platform.

We live in a paradoxical environment: every user is different (and wants something specific from an interface), yet often, the changes we implement resort to cycles of iterations (human-powered tweaks). No silver bullet exists to get our websites to do all of the hard work.

But isn't this odd: social networks are among the most highly customizable and flexible infrastructures, yet few websites follow their lead and let users decide what works best for them.

As we move into the future, user expectations will only increase. We've had it easy as an industry so far — users have just accepted the way things are.

But as users gain knowledge of the Web and what it's capable of, they'll question the validity of the one-size-fits-all model. They'll want compatibility and the ability to customize.

Personalizing designs is a challenge, but one worth taking on.

Sources:

- https://www.ebay.com/
- https://silverbackapp.com/
- http://www.linkwithin.com/learn
- http://www.addthis.com/
- https://www.bbc.co.uk/
- https://www.mobify.com/

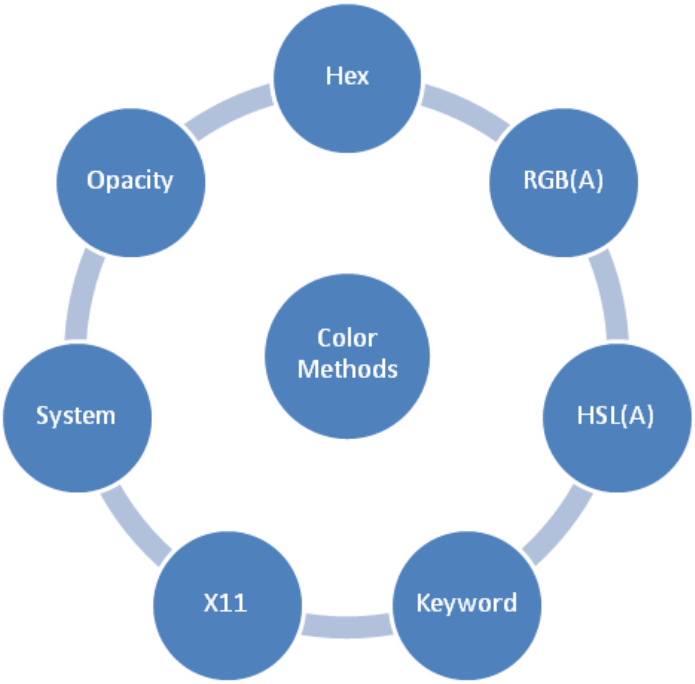
A Guide to CSS Colors in Web Design

With the exceptions of typography and layout, nothing has a more profound impact on the way we design and craft websites than color — from the visuals we showcase through images and media to the simple choice of whether to use salmon pink or neon green to give a website that ’90s "Help, I’m going blind!" appeal. This simple guide will look into CSS colors. You’ll also find excellent color charts and tools to help you work with color values.

Let’s set our objectives:

- Examine the variety of options that exist
- Analyze basic code examples for each color type
- Examine opacity, transparency and more
- Explore simple color theory and psychology

Color Value Notation



There are many types of color notation in CSS, and the lesson to take away is that you need to decide for yourself which method is best for you after taking into account cross-browser compatibility and other limitations.

In many ways, color is limited only by our imagination. Our choices for shades and contrasts have changed over the years with the introduction of displays that are capable of displaying many millions of colors (a far cry from the old 256-color limit).

When we identify that perfect shade in our code, the format we use to describe and reference it can appear as one of many color values that help the browser determine exactly what we want it to render (and where).

For every color, there's a hex, RGB and HSL equivalent. You just need to know the value.

Hexadecimal Value Notation

Of the many methods to describe color in CSS, the most popular is the hexadecimal value. It's been around since the beginning and has retained maximum browser support. A hexadecimal color value starts with the hash (#) character, followed by three or six hexadecimal digits in RGB notation format. For example, the hexadecimal (often unofficially abbreviated as hex) color value for white is #FFFFFF, or #FFF in shorthand.

Because hexadecimal value notation supports a wider range of colors than the equally old keyword system (e.g. red, green, blue, black), it remains highly used for its preciseness.

Below are simple examples of longhand and shorthand hex value notation:

```
a { color: #AABBCC; }
a { color: #ABC; }
```

RGB Value Notation

Of course, while hex codes remain highly popular in CSS, many designers don't realize that the other convention for RGB color notation has been around just as long (and we can trace its roots back to the earliest CSS specification). Using 0 to 255 numerals for each of

the red, green and blue values that comprise our RGB colors (or using percentages instead), we can present our choices in an easier-to-read format.

Below are simple examples of RGB representations of colors:

```
a { color: rgb(170, 187, 204); }
a { color: rgb(67%, 73%, 80%); }
```

HSL Value Notation

We also have HSL, which is the new kid on the block. Built to complement the existing RGB system, HSL allows us to combine hue, saturation and brightness into a triple declaration.

HSL makes guessing colors easier because you are declaring shades that are either lighter or darker by increasing or decreasing other values (against the base hue).

Hues are declared in degrees of the color circle (a value of 0 to 360), and both saturation and brightness are declared in percentages.

Below is a simple example of an HSL representation of a color:

```
a { color: hsl(210, 13.3%, 73.3%); }
```

Color Keywords and X11

So far, CSS notation may seem quite difficult for beginners, especially if hexadecimal values and color calculations (using percentages, degrees of the color circle or light values from the RGB spectrum) aren't your thing.

Fear not. A bunch of explicitly worded alternatives, referred to as color keywords, can help.

Unlike the other methods, the supported values are limited, but they do what they do well.

Using a color keyword is one of the simplest methods of describing color in CSS. Rather than using some weird notation that is all but alien to the average end user, simply provide one of a base set of values that describe the color by a recognizable name.

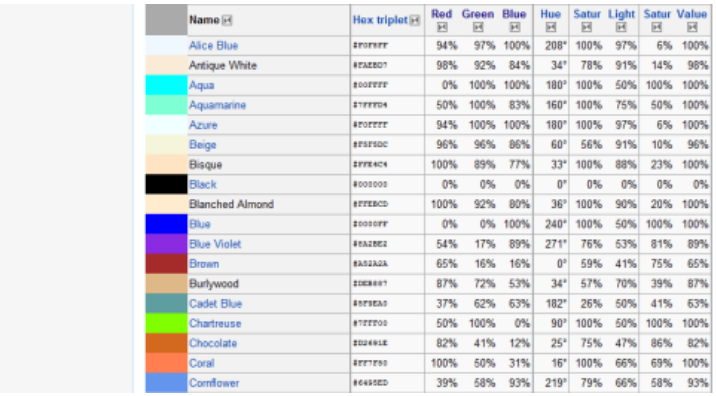
The color names are limited to 16 values but, as we'll find out, can be expanded using an extended palette. Below is a simple example:

```
a { color: green; }
```

Basic color keyword notation has been around for a long time, but given the limitations of the 16 base colors mentioned above, browser makers (and now the CSS3 spec, which has adopted the values from the SVG specification) have expanded the range of colors by taking advantage of the X11 extended color palette (which gives us over 100 named colors).

We can describe colors using keywords (as long as they are supported by browsers).

The only potential problem with X11 is that browser support and consistency aren't a guarantee (some color clashes can occur, for example).



Name	Hex triplet	Red	Green	Blue	Hue	Satur	Light	Satur	Value
Alice Blue	#F0F8FF	94%	97%	100%	208°	100%	97%	6%	100%
Antique White	#FAEBD7	98%	92%	84%	34°	78%	91%	14%	98%
Aqua	#00FFFF	0%	100%	100%	180°	100%	50%	100%	100%
Aquamarine	#7FFFD4	50%	100%	83%	160°	100%	75%	50%	100%
Azure	#F0FFFF	94%	100%	100%	180°	100%	97%	6%	100%
Beige	#F5F5DC	96%	96%	86%	60°	56%	91%	10%	96%
Bisque	#FFE4C4	100%	89%	77%	33°	100%	88%	23%	100%
Black	#000000	0%	0%	0%	0°	0%	0%	0%	0%
Blanched Almond	#FFEBCD	100%	92%	80%	36°	100%	90%	20%	100%
Blue	#0000FF	0%	0%	100%	240°	100%	50%	100%	100%
Blue Violet	#8A2BE2	54%	17%	89%	271°	76%	53%	81%	89%
Brown	#A52A2A	65%	16%	16%	0°	59%	41%	75%	65%
Burlywood	#DEB887	87%	72%	53%	34°	57%	70%	39%	87%
Cadet Blue	#5F9EA0	37%	62%	63%	182°	26%	50%	41%	63%
Chartreuse	#7FFF00	50%	100%	0%	90°	100%	50%	100%	100%
Chocolate	#D2691E	82%	41%	12%	25°	75%	47%	86%	82%
Coral	#FF7F50	100%	50%	31%	16°	100%	66%	69%	100%
Cornflower	#6495ED	39%	58%	93%	219°	79%	66%	58%	93%

The X11 palette gives us an extended, supported list of color keyword names.

System Colors

Finally, one other group of color keywords needs to be mentioned. System colors allow designers to match certain color choices to those of the operating system's default scheme.

Think of the display options in Windows and how you can alter the color of system text, the background, title bars and such; the keywords ask the browser to emulate them.

However, system color keywords are deprecated, poorly supported and dangerous for your website's accessibility because you leave it up to the user's OS to decide on colors and native appearance.

In addition to being able to assign pre-defined color values to text, backgrounds, etc., CSS2 introduced a set of named color values that allows authors to specify colors in a manner that integrates them into the operating system's graphic environment.

For systems that do not have a corresponding value, the specified value should be mapped to the nearest system value, or to a default color.

The following lists additional values for color-related CSS properties and their general meaning. Any color property (e.g., 'color' or 'background-color') can take one of the following names. Although these are case-insensitive, it is recommended that the mixed capitalization shown below be used, to make the names more legible.

ActiveBorder
Active window border.
ActiveCaption
Active window caption.
AppWorkspace
Background color of multiple document interface.
Background
Desktop background.
ButtonFace
Face color for three-dimensional display elements.
ButtonHighlight
Highlight color for three-dimensional display elements (for edges facing away from the light source).
ButtonShadow
Shadow color for three-dimensional display elements.

System colors were an attempt to make websites look and feel more like an operating system.

Color Opacity and Transparency

So far, we have outlined the basics of color value notations, and you could be forgiven for thinking that simply telling the browser what color you want is all there is to it. But with the advent of CSS3 comes an additional level of controlling colors, bringing us the ability to layer semi-transparent or opaque effects onto objects.

Opacity and transparency effects on websites can be subtle and quite beautiful.

As with the usual color notation, there are many ways to achieve this. Below are a few of the most common.

Opacity

First, let's discuss opacity, which is perhaps the trickiest to make cross-compatible due to all of the browsers and versions that need supporting. While opacity appeared only in CSS3, it has quite a history of early browser support, especially in Internet Explorer (which used the proprietary DirectX filter property). Getting the mix exactly right is a challenge, and some people use different blends to maintain the effect.

Below is a solid formula that includes everything you need for bulletproof opacity:

```
div {
    opacity: 0.5;
    -moz-opacity: 0.5;
    -khtml-opacity: 0.5;
    -ms-filter:
    "progid:DXImageTransform.Microsoft.Alpha(Opacity=50)";
    filter: alpha(opacity=50);
}
```

Transparency

Transparency has also been revitalized with CSS3. No longer are you limited to image-based alpha transparency (although that's still an option) or single-color transparency effects on images. Modern browsers take advantage of both RGBa and HSLa, which, as you can guess, use conventional RGB and HSL notation but with a percentage-based alpha transparency value attached to the end.

Below are simple examples of RGB and HSL with alpha-transparency of 0.5 (or 50%):

```
a { color: rgba(170, 187, 204, 0.5); }
a { color: rgba(67%, 73%, 80%, 0.5); }
a { Color: hsla(210, 13.3%, 73.3%, 0.5); }
```

Safe Colors

With something as important as color, you'd expect compatibility to be a given. But the situation in reality is far from perfect. The X11 color system isn't supported equally across web browsers (and has only just been added to the CSS specification). HSL, HSLa and RGBa are new additions to CSS3 and so cannot be relied on in older browsers; system colors have been deprecated; and both opacity and alpha-transparency support are messy!

If you think things are limited now, just think back to when colors were restrictive.

Even the base colors we depend on could potentially be a problem on older hardware, because back when displays could render only 256 colors (or even further back with 8-bit displays), compatibility issues were serious enough for us to limit the power behind color notation.

To ensure that all devices got a consistent and usable look and feel, three levels of safe color use were formed.

Really-safe Colors

This existed back when 8-bit displays were still fairly common, because on those displays, only 22 of the colors defined as "web-safe" displayed reliably.

Web-safe Colors

This is the one that most designers know (and some still use). 216 colors were available (of the 256 that included dithering), mainly for older displays.

Web-smart Colors

This was 4,096 colors and was produced to remove the limitations of the web-safe palette, while keeping color use reasonably compatible.

Even if you look beyond compatibility, web designers have a few additional complications with color use, mainly as a result of web accessibility. Color blindness affects many individuals worldwide, and the condition has varying degrees of severity, which makes the situation especially hard to deal with.

Web Accessibility Considerations

Some people may be deficient in a single color, others completely blind to an entire spectrum, still others able to see only in black and white (monochromes), and some have little or no vision at all.

Here are the various forms of color blindness:

Monochromacy

The inability to see colors outside of black and gray shades, or, in rarer cases, the inability to see outside of certain chroma shades like mild browns or blue variants.

Dichromacy

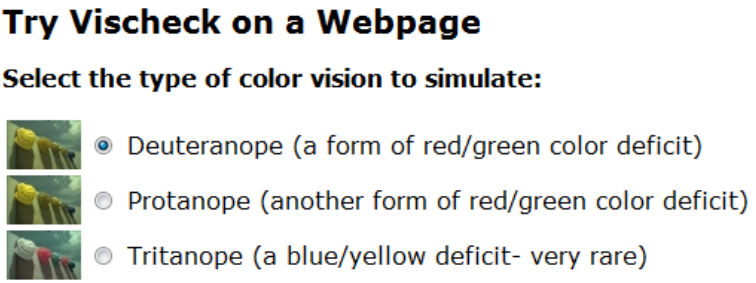
This is when one of the three basic color mechanisms fails to work and is commonly referred to as protanopia (blind to red), deuteranopia (blind to green) and tritanopia (blind to blue).

Trichromacy

This condition is when mechanisms are present but defective (thus causing confusion). Trichromacy includes protanomaly (reduced red), deuteranomaly (reduced green, which is common) and tritanomaly (reduced blue).

This combination of issues affecting web accessibility requires us to be more sensitive to how we use color (and the colors we choose), so that we make our websites easy to use, engaging and highly readable.

While some colors look perfectly fine to one set of eyes, certain conditions could render your work useless or invisible to less fortunate users.



Color blindness is hard to emulate, but there are tools to help us out.

If you don't currently consider color blindness in your designs, you should start right away. To help you on your way, read 10 Tools for Evaluating Web Design Accessibility.

Color Associations

Let's just briefly go over some basic color associations. As you can see, there is more to color than meets the eye. Color can have a dramatic impact on how users interact with the page, and studies on the effects of color on buying habits and behavior are well known among designers and psychologists. Plenty of articles about color theory exist, so don't shy away from pursuing the subject further, because it could benefit your decision-making process.

When we see certain colors, many of us subconsciously associate them with things we know from the world around us. For example, when we see the color red, we may think of love, blood or passion depending on various factors such as cultural upbringing and

personal experiences. We can use these associations to impress our identity on users, just as painters use color to elicit emotion. For example, a website about vampires would be better served by a red and black palette than a yellow and green one because those are the colors traditionally associated with the genre.

Many different color associations arise from culture, and some are quite obvious.

Not only does this psychological phenomenon affect our perceptions of the environment (though to what extent is still up for debate), different cultures associate colors with different things; we mustn't assume that Western standards hold true worldwide.

Another color association is with temperature: we often associate blue with cool things and red with hot things, and this can be seen in our visuals.

Color psychology can help you better connect with your audience.

Color Tools and Resources

Below is a list of links related to color in web design that you might want to explore.

Color Charts

- CSS 2.1 specification: Colors - https://www.w3.org/TR/CSS21/syndata.html#color-units
- CSS3 color specification - https://www.w3.org/TR/css-color-3/
- Color names - http://coloria.net/bonus/colornames.htm
- Color Codes - http://www.december.com/html/spec/colorcodes.html
- Really-safe colors - http://web.archive.org/web/20001218100700/http://hotwired.lycos.com:80/webmonkey/00/37/stuff2a/complete_websafe_216/reallysafe_palette.html
- Web-safe colors - http://www.elizabethcastro.com/html/colors/websafecolors.html
- Web-smart colors - http://cloford.com/resources/colours/websmart1.htm
- Colorblind Web Page Filter - https://www.toptal.com/designers/colorfilter
- The Meaning of Colors - http://sibagraphics.com/utilities/the-meaning-of-colours/

Color Pickers

- ColorSchemer Studio 2
- Color Cop (free) - http://colorcop.net/
- Adobe Kuler (web-based) - https://color.adobe.com/
- ColorToy (iOS)
- Magic Color Picker (Android)

Conclusion

Color plays a huge role in web design, and our implementation of it matters. Whether we're playing with contrast, shades, hues, transparency and so forth, our goal is to produce websites that are aesthetically pleasing and relevant to our audience.

Our choices should avoid clashes and blurring, and we should choose a palette that is compatible and accessible to our users.

CSS has given us the groundwork to use any color we desire, and CSS3 offers additional tools, such as opacity.

But with great flexibility comes great responsibility. Color covers every square inch of our websites, and we should never underestimate its value to our work. Review the color system you are currently using, and be creative in your effort to build increasingly beautiful, vivid experiences. Your colors should leave a positive, lasting impression.

5 Little-Known Web Files That Can Enhance Your Website

Previously, I wrote about 5 web files that will improve your website and discussed files that, while small in size, pack a solid punch and make our work that little bit better. In this article, we'll look at five more web files that can improve and your website.

Quick Overview

Here are the files we will cover:

- P3P.xml – for user privacy
- Geo.kml (and Geo.rdf) – for geolocation
- Humans.txt – for attribution
- vCard.vcf – digital business card
- PICS.rdf – declares a website to be safe (or not safe) for young web users

P3P.xml

Issues related to user privacy on the web are of paramount concern both to those who store information (site owners) and to those who submit the information (site users).

The Platform for Privacy Preferences Project (P3P) encourages website owners to declare all details relating to their privacy measures in a standardized XML document (or via meta tags), so that browsers can pick up the information, display warnings and details, and empower users to take whatever action they feel necessary.

On the surface, this system sounds wonderful: it gives users control over their data, it helps site owners keep them informed, and the browsers let users choose sites to trust.

Platform for Privacy Preferences or Pretty Poor Privacy? The truth is out there!

Unfortunately, creating the file can be quite complex, with all of the variables involved (it's like filing a tax return).

Worse, Internet Explorer is the only major browser that does much with the file. It grants greater control over cookie-blocking in IE6+, while letting you display an on-screen privacy policy.

In addition, the issuer of the file is responsible for complying with its own guidelines, and there is no enforcement.

The reality is that, with all of the critics and compatibility issues, no alternative has gained as broad of a support as P3P.

Website owners need to deal with these increasingly prevalent privacy issues, and because P3P offers a workable solution right now, adopting it for the sake of IE, privacy search engines, and other user-focused tools is justifiable.

Creating a P3P.xml File

Once you've created the file, it requires little maintenance.

There are two ways to go about creating a P3P.xml file.

One way is you could follow the W3C Platform for Privacy Preferences 1.0 (P3P1.0) specification and build the file by hand using your favorite text editor.

Alternatively, a few useful apps will do the work for you.

P3P.xml tools:

- IBM P3P Policy Editor (freeware)
- JRC Policy Workbench (open source) - https://sourceforge.net/projects/jrc-policy-api/
- P3PEdit (web-based, $39)

Other useful resources:

- P3P policy validator - https://www.w3.org/P3P/validator.html
- Internet Explorer and P3P - https://blogs.msdn.microsoft.com/ieinternals/2013/09/17/a-quick-look-at-p3p/
- P3P privacy verification - http://www.privacybird.org/

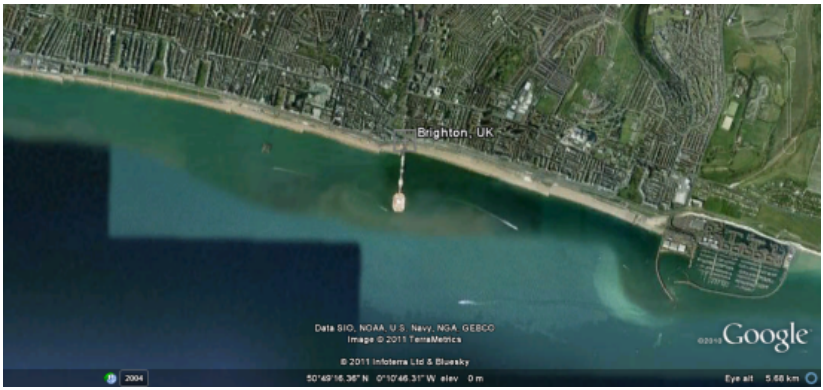
Once you've created the file, name it P3P.xml, put it in either the site's root directory or a directory named w3c.

Next, you'll need to add a reference in the <head> of your HTML document. Below is a sample reference:

```
<head>
    <link rel="P3Pv1" type=" text/xml" href="/w3c/p3p.xml" />
</head>
```

Geo.kml and Geo.rdf

Geotagging has taken the web by storm. Disclosing your geographic location to site visitors can build trust, especially in e-commerce websites.



Using Google Earth or Maps, we can guide visitors to our office.

The benefits of geotagging are quite evident. You let users see where your offices are (great if you need to arrange meetings with clients at your headquarters). Also, showing that there are real people behind the website makes you seem less like an anonymous corporation. Not to mention, mapping services will be able to index you in their listings.

There are several possible approaches to geotagging your site, including using microformats. We'll look at how to build two different solutions: one for Google Earth (Geo.kml) and a helpful RDF fallback (Geo.rdf) for other tools.

Creating a Geo.kml file

You can create this file using any text editor. You could name the file after the website or place that you're mapping. For example, if we made a file for Six Revisions, it could be named SixRevisions.kml.

Put your Geo.kml file in the root directory of your website.

Below is a basic example of what the code should include:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://
www.google.com/kml/ext/2.2" xmlns:kml="http://
www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/
2005/Atom">
```

```
    <Document>
        <name>Brighton</name>
        <description>The place that I call home!</description>
        <Style id="pin"><IconStyle><Icon><href>http://
        maps.google.com/mapfiles/kml/pushpin/ylw-
        pushpin.png</href></Icon></IconStyle></Style>
        <Placemark>
            <LookAt>
            <longitude>-0.13642</longitude>
            <latitude>50.819522</latitude>
            <altitude>0</altitude>
            <tilt>0</tilt>
            <range>5500</range>
            </LookAt>
            <styleUrl>#pin</styleUrl>
            <Point><coordinates>-0.13642,50.819522,0</
            coordinates></Point>
        </Placemark>
    </Document>
</kml>
```

Every KML file begins with a document type declaration (DTD), which states that this is an XML file that follows the KML specification.

Inside the KML element, there should be a <Document> tag (just as an HTML document has a <body> tag), and in it, you put the details of your address.

Briefly, here are explanations for tags to include:

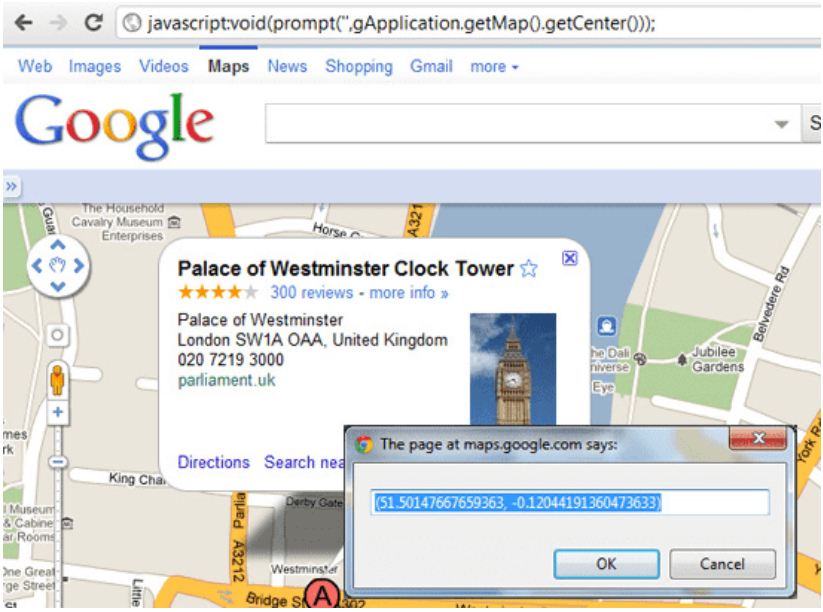
- <name> and <description> lets users know what is being shown
- <style> offers an image to pinpoint the location on the globe
- <Placemark> pinpoints the object
- <Point> contains <coordinates> to the latitude and longitude of your location
- <LookAt> data about the coordinates, such as altitude and tilt
- <range> tells how far to zoom in

Most of these are easy to declare. The only thing to find are the coordinates.

Finding your coordinates is rather easy. If you visit maps.google.com and type in the place or address, half the job is done.

Of the many methods of extracting these details, my favorite requires the least amount of work. Simply using the script below into the address bar after you've found your location will yield the information.

```
javascript:void(prompt('',gApplication.getMap().getCenter()));
```



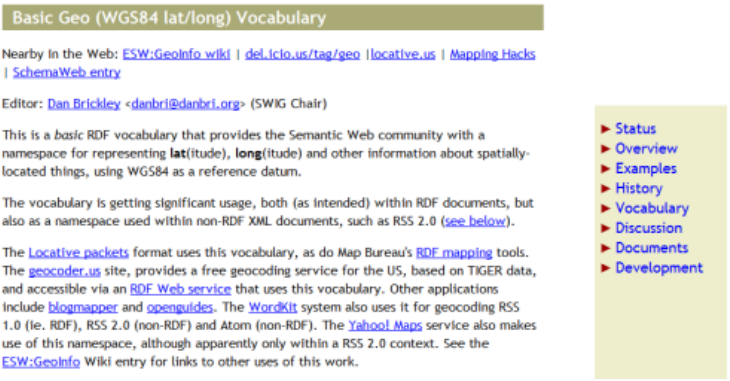
To reference it in HTML, add the following inside your <head> tag:

```
<link rel="alternate" type="application/vnd.google-earth.kml+xml"
href="SixRevisions.kml" />
```

Of course, you can do more with your KML file than what is described here. If you'd like to explore further, Google has a KML Reference documentation.

Creating a Geo.rdf File

Of course, not everyone uses Google Earth, and many other web services exist (such as search engines) that gather geodata. So, we should also produce an RDF file that works some geo magic on the semantic web.



Geodata has a lot of uses, and it's very easy to create.

If you have the coordinates, the file is actually a lot more straightforward to create than the Google Earth KML file, because we're not worried about visual representation; we simply want to get the coordinates out there for other services to make use of them (whether social networks or search engines).

To build the file, create a new document named Geo.rdf, and in it, just use the code below, replacing details such as your website (rdf:about), the place or website name (dc:title), and your coordinates (geo:lat and geo:long tags).

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:geo="http://
www.w3.org/2003/01/geo/wgs84_pos#" xmlns:rdf="http://
www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://www.mysite.com">
    <dc:title>Brighton</dc:title>
    <foaf:topic rdf:parseType="Resource">
```

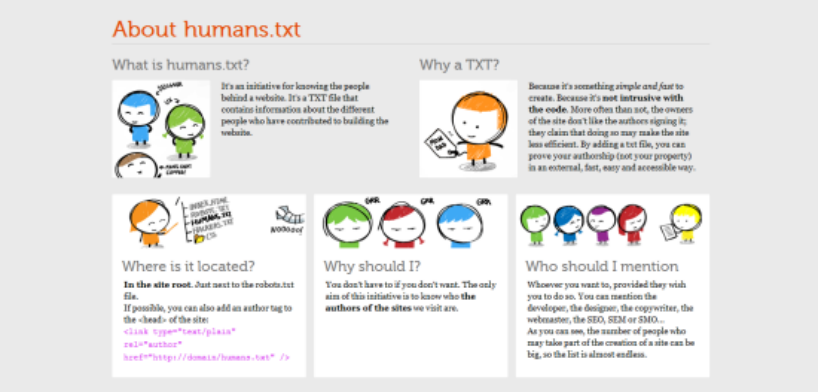
```
    <geo:lat>50.819522</geo:lat>
    <geo:long>-0.13642</geo:long>
    </foaf:topic>
    </rdf:Description>
</rdf:RDF>
```

Once you've created the file, add the link reference to your website:

```
<link rel="alternate" type="application/rdf+xml" title="Geo"
href="Geo.rdf" />
```

Humans.txt

The proper way to attribute work has been up for debate since the Internet went mainstream. As professionals, we want to act in our clients' best interests, but as creatives, we want people to know who is behind the wonderful work we put together (which could lead to new clients).



Websites are built by people, so why not credit them?

humans.txt is a standard format not unlike a robots.txt, but with the intention of providing information about the people behind a particular website.

Creating a Humans.txt File

The great thing about Humans.txt is its simplicity. While there is no formal standard for what (or who) to include, there are some best

practices to ensure that the file is as human-readable (and possibly machine-readable, for web spiders) as possible.

To begin, create a humans.txt file and put it in your website's root directory. In that file, you will be enclosing three primary categories in comments. You could add and remove categories as you see fit

The first category, TEAM, can include directives such as title, position, website, Twitter profile and location. The purpose is to provide information about the individuals responsible for creating the site.

The second category, THANKS, attributes the project's contributors by name (or URL).

The final category, SITE, provides information about the standards, components and software used in the website's production, along with a timestamp for the last update and language details.

Below I've adapted a humans.txt template, illustrating this format:

```
/* TEAM */
Title:      YourName.
Position:   Job Role
Site:       http://yoursite.com
Twitter:    @YourSite
Location:   City, Country.
/* THANKS */
Name: TheirName
/* SITE */
Updated:     YYYY/MM/DD
Language:    English (US)
Standards:   HTML5, CSS3, JavaScript
Components:  jQuery, etc.
Software:    Adobe Photoshop, Notepad++
```

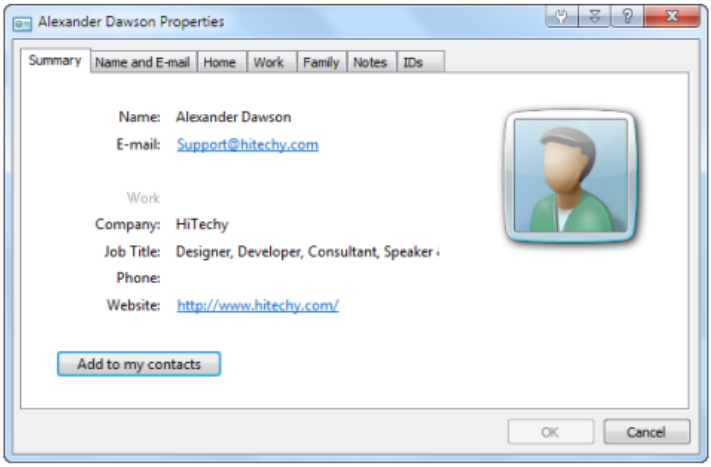
Once you've filled in as much detail as you'd like (remember that this file is primarily for humans, so keep it simple), you just need to link to the <head> (as always).

```
<link type="text/plain" rel="author" href="humans.txt" />
```

It's a great way to credit the entire team unobtrusively.

vCard.vcf

Next on the list is a personal favorite of mine. In this era of communication, enabling clients and visitors to get (and stay) in touch is essential. Our contact pages are often fragmented by social network icons, email forms and lists of instant messaging and VoIP accounts. Visitors and clients just want to get in touch with us, so make the process as easy as possible.



With a few lines of code, we can produce a useful importable contact reference.

vCard is a standardized format for digital business cards. One file lists all of the applications, services and social networks that people can use to connect with you. It functions as an index of meta data about you or your business, and people can import all of the data into their favorite address book or email client. Microsoft Outlook and other clients support vCards, as do the Windows and Mac address books.

vCard has its own microformat that semantically marks up any related information on our pages. You could offer just the microformat version, but for compatibility and ease of access, vCard (or even both formats) is preferred.

Creating a vCard.vcf File

The first thing to do is create the vCard.vcf file (which is case-insensitive). Inside are a few things every vCard must have, according to the specifications:

- BEGIN:VCARD and END:VCARD (case-sensitive) to map the start and end of the line (the same way that we open and close the <html> tag in HTML documents)
- VERSION: with a value of 3.0 (the latest edition)
- N: (Lastname;Firstname) and FN: (Full Name) directives

Here is an example:

```
BEGIN:VCARD
VERSION:3.0
N:LastName;FirstName
FN:FirstName SecondNames LastName
END:VCARD
```

You can add a bunch of other useful directives to declare things about yourself; if supported, these details can be used by other apps and services.

The general syntax for vCard files is the directive in uppercase, followed by a colon character, except where a variable is required (like TYPE=HOME or EMAIL), in which case the colon becomes a semi-colon. TYPE= becomes the variable identifier, multiple variables are comma-separated (like TYPE=HOME, WORK), and new lines for values are identified by more semi-colons.

Below are some examples of the various directives:

```
NICKNAME:Name
X-GENDER:Male
BDAY:YYYY-MM-DDT
ORG:Company
TITLE:Web Designer
URL;TYPE=WORK:http://www.yoursite.com/
EMAIL;INTERNET:Hello@yoursite.com
```

```
EMAIL;TYPE=PREF,INTERNET:Support@yoursite.com
X-MSN;TYPE=HOME:You@hotmail.com
X-SKYPE;TYPE=WORK:MySkypeID
X-GOOGLE-TALK;TYPE=WORK:MyGoogleID
```

The variable TYPE=PREF indicates a preferred contact type (if the destination program recognizes it).

For more details about directives and extensions, please check the links below. You can add all sorts of awesome things into vCards, like images, links and even sometimes audio!

- Basic directives - https://en.wikipedia.org/wiki/VCard#Properties
- Known extensions - https://en.wikipedia.org/wiki/VCard#vCard_extensions

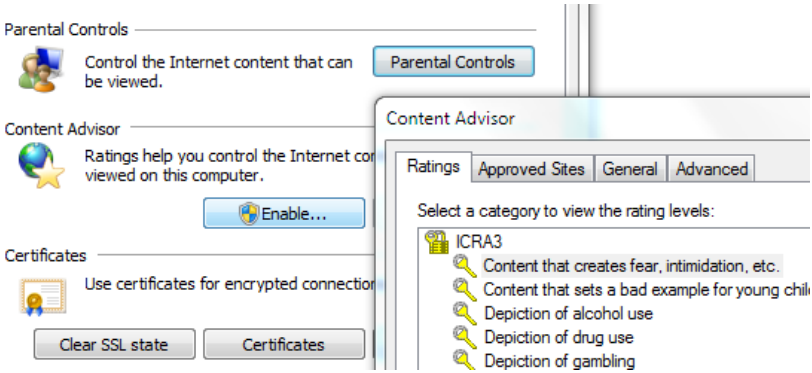
There are many more directives than are mentioned here. Another reason vCards are great is that they are extensible, supporting proprietary extensions. vCard extensions are prefixed with X- (the way we use vendor prefixes like -moz- in CSS). The only downside is that, as with CSS, support isn't a given, so you'll have to figure out the best semantic route.

Once you have your vCard.vcf ready, reference it in your HTML documents like so:

```
<link rel="alternate" type="text/directory" title"vCard"
href="vCard.vcf" />
```

PICS.rdf

Not everything on the web is child-friendly, and this last file helps with that issue. Many software providers (even Windows with IE6+) provide specialized tools to filter out objectionable content for young audiences (i.e. generic content filtering). While some providers use human-based filtering, PICS (Platform for Internet Content Selection) helps any automated product gauge whether your content is age-appropriate. And it's well supported.



The content advisor in Internet Explorer is just one system that integrates PICS tags.

Getting your content blocked automatically reduces your ability to reach users, and while some newer tools rely on human screening (because of a distrust of self-regulation), we can at least certify our content to aid with such decisions. It's pretty much like how the music and film industries work, and there are several ratings systems, too.

Creating a PICS.rdf file

Because there are several ratings systems, each with its own methodology, I've adapted one of the most popular formats (ICRA's RDF method) to accept other types of labeling. Even though the ICRA has ceased developing its PICS labeling system, the system is still widely used by content filters, so it still has an important role to play, until something better comes along.

Start by creating a PICS.rdf file, and use the following code in it:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dcterms="http://
purl.org/dc/terms/" xmlns:label="http://www.w3.org/2004/12/q/
contentlabel#" xmlns:icra="http://www.icra.org/rdfs/
vocabularyv03#" xmlns:rsac="http://www.icra.org/rdfs/
```

```
vocabularyv01#"  xmlns:ss="http://www.safesurf.com/ssplan/"
xmlns:sfk="http://www.weburbia.com/safe/ratings/ ">
    <rdf:Description rdf:about="">
        <dc:creator rdf:resource="http://www.icra.org" />
        <dc:creator rdf:resource="http://www.safesurf.com" />
        <dc:creator rdf:resource="http://www.weburbia.com/safe" /
        >
        <dcterms:issued>2011-04-15</dcterms:issued>
        <label:authorityFor>http://www.icra.org/rdfs/
        vocabularyv03#</label:authorityFor>
    </rdf:Description>
    <label:Ruleset>
        <label:hasHostRestrictions><label:Hosts><label:hostRestrictio
        n>www.yoursite.com</label:hostRestriction></
        label:Hosts></label:hasHostRestrictions>
        <label:hasDefaultLabel rdf:resource="#label_1" />
    </label:Ruleset>
    <label:ContentLabel rdf:ID="label_1">
    <rdfs:comment>ICRA Ratings</rdfs:comment>

    </label:ContentLabel>
    <label:ContentLabel rdf:ID="label_2">
    <rdfs:comment>RSACi Ratings</rdfs:comment>

    </label:ContentLabel>
    <label:ContentLabel rdf:ID="label_3">
    <rdfs:comment>SafeSurf Ratings</rdfs:comment>

    </label:ContentLabel>
    <label:ContentLabel rdf:ID="label_4">
    <rdfs:comment>WebUrbia Ratings</rdfs:comment>

    </label:ContentLabel>
</rdf:RDF>
```

Below each rdfs:comment element, you'll notice some empty spaces. Your job is to fill them in according to the specifications of the various groups. Guides exist to help you determine what code to put in each category (mentioned below).

In each section, create a tag that begins with the letters of the system you're using there, and then (separated by a colon) define the relevant rating by its unique abbreviation. Once you have the tag, simply enter the appropriate digit, 0 or 1, inside it.

ICRA, RSAC, SafeSurf and Safe for Kids represent the four most popular PICS systems.

For example, if you used ICRA, your tags would be <icra:x></icra:x>, with the x's being replaced by the rating, and the 1 or 0 value going between the tags.

For RSAC, it would be <rsac:x>, for SafeSurf it would be <ss:x>, and for Weburbia's "Safe for Kids" system it would be <sfk:x>.

To determine exactly what sections you'll need, visit these websites:

- ICRA - http://web.archive.org/web/20090228211023/http:/www.icra.org/decode/
- RSAC - https://en.wikipedia.org/wiki/Recreational_Software_Advisory_Council
- SafeSurf - http://www.safesurf.com/ssplan.htm
- WebUrbia - https://web.archive.org/web/20061031134823/http://www.weburbia.com/safe/ratings.htm

To see how this code might look, I've pre-built some example labels. ICRA labels have a two-letter code (for example, NZ declares that there is no nudity on the website). For RSAC, it's a letter followed by a number (indicating severity).

For SafeSurf, it's a slightly longer value (SS~~, followed by two zeros and the number or letter). The easiest of all (with only one declaration) is Weburbia's "Safe for Kids" scheme, with an S to represent the safety level, and a value of a 0, 1 or 2 to match the PICS scheme.

Below are some basic examples of PICS labels from the four providers:

```
<label:ContentLabel rdf:ID="label_1">
    <rdfs:comment>ICRA Ratings</rdfs:comment>
    <icra:nz>1</icra:nz>
    <icra:sz>1</icra:sz>
    <icra:vz>1</icra:vz>
    <icra:lz>1</icra:lz>
    <icra:oz>1</icra:oz>
    <icra:cz>1</icra:cz>
    <icra:xz>1</icra:xz>
</label:ContentLabel>
<label:ContentLabel rdf:ID="label_2">
    <rdfs:comment>RSACi Ratings</rdfs:comment>
    <rsac:L>0</rsac:L>
    <rsac:N>0</rsac:N>
    <rsac:S>0</rsac:S>
    <rsac:V>0</rsac:V>
</label:ContentLabel>
<label:ContentLabel rdf:ID="label_3">
    <rdfs:comment>SafeSurf Ratings</rdfs:comment>
    <ss:ss000>1</ss:ss000>
    <ss:ss001>1</ss:ss001>
    <ss:ss002>1</ss:ss002>
    <ss:ss003>1</ss:ss003>
    <ss:ss004>1</ss:ss004>
```

```
        <ss:ss005>1</ss:ss005>
        <ss:ss006>1</ss:ss006>
        <ss:ss007>1</ss:ss007>
        <ss:ss008>1</ss:ss008>
        <ss:ss009>1</ss:ss009>
        <ss:ss00A>1</ss:ss00A>
    </label:ContentLabel>
    <label:ContentLabel rdf:ID="label_4">
        <rdfs:comment>WebUrbia Ratings</rdfs:comment>
        <SFK:S>0</SFK:S>
    </label:ContentLabel>
```

Once you've created the PICS label and determined your content's suitability for younger audiences (basically by filling out the RDF file like a questionnaire), all that's left to do is save the file, put it in the root directory of your website, and declare it in the <head> via a link tag:

```
<link rel="meta" href="PICS.rdf" type="application/rdf+xml"
title="PICS labels" />
```

Summary

With these five easy to use files, you should find yourself at the pinnacle of modern Web Standards.

Sources:

- https://www.w3.org/TR/P3P/
- https://en.wikipedia.org/wiki/Geotagging
- https://www.google.com/earth/index.html
- https://maps.google.com/
- https://developers.google.com/kml/documentation/kmlreference?csw=1
- https://www.w3.org/2003/01/geo/
- http://humanstxt.org/
- https://en.wikipedia.org/wiki/VCard
- http://microformats.org/wiki/h-card

The Evolution of Internet-Enabled Devices

The Internet is a wondrous thing. It's an unrivaled source of knowledge for its users, and as web designers and web developers, it keeps many of us from becoming homeless with "Will code for food" signs hanging around our necks!

As the Web matures, the devices that provide access to it have evolved along with it. No longer are we limited to "surfing the 'net" on a 28.8 kbps dial-up modem. These days, we don't even require a computer to go online — we have smartphones, tablets, e-book readers like the Kindle, and more. Let's look at the evolution of the hardware that gives us access to the Internet.

Computers and Laptops

We can trace our digital ancestry back to devices we still use today: traditional desktop and laptop computers.

In the formative years, designing and developing websites wasn't much to write home about. Web standards were in their infancy, browsers were firing bazookas at one another from the rooftops of the digital infrastructure, and a "feature" of many websites was a counter that told everyone that the website had "000002" visitors (one of those being you, the person who built the site). Isn't nostalgia fun?

They've been on earth longer than some geeks, and we still love 'em! Source: TehBoris

Screens started in the 800×600 range, grew to 1024×768, and then a few others fell into the mix as resolutions became something of a "whatever works" issue for users.

That, and browser windows used their fair share of monitor screen real estate, what with the unnecessary Ask.com and Alexa.com toolbars and sidebars people installed.

Smartphones

Then everything changed. There were rumors of cell phones gaining Internet access, whispers about how this would change everything.

One little smartphone, powered by hearsay, Steve Jobs and unicorn blood, broke open the doors on a new range of devices to design for. Smartphones came onto the scene.

What made things even more fun was that other rendering engines joined the fray, too!

Some are fully featured, and others... not so much. Source: Guy Schmidt

No longer were cell phones powered by WML (a special markup language intended for portable devices). No longer did we need to access the Web on scaled-down user experiences through PDAs if we wanted to browse on the go. Touchscreens were the next big interface movement. Displays got smaller, and imaginations got bigger.

For the cell phone makers that didn't put radioactive glowing apples on the backs of their products, the need to build something competitive grew — as did the number of mobile-friendly browsers for us to test our stuff on.

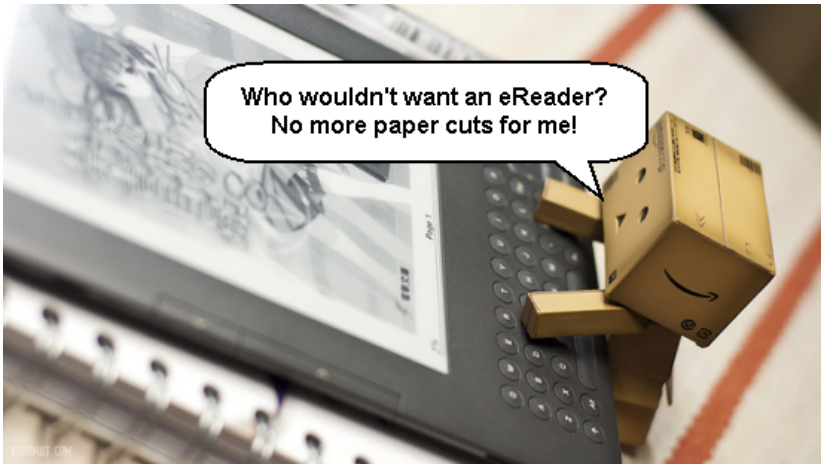
Tablets, Netbooks and eReaders

The switch-to-mobile excitement was a giant leap for mankind, but the third Internet device evolution was a small step for many of us.

Sure, mobile devices were cool; we could research Angry Birds cheats on some website while sitting on a train. We got a taste for the Mobile Web and wanted more.

So touchscreen smartphones scaled up to become tablets, PCs scaled down to become netbooks, and printed books are being replaced with Internet-enabled reading devices.

Thus came the next wave of devices!



They need to boot, but you're less likely to have them thrown at you. Source: kodomut

Tablets aren't exactly new. But when they became popular, and then ubiquitous, they gave us designers a reason to pay attention. The Web was no longer bound by one of two device types; we all wanted to pretend we were in Minority Report, with thin yet full-featured gadgets. Netbooks, being cheap, also became popular, and resolution consistency became something designers could only dream about.

Television and Game Consoles

So many things are becoming web-enabled that we've arrived at the next era of device types — one that is set to blow the wheels off the tablet surge in terms of widespread appeal. Our old friend, the television, and its time-eating sidekick, the game console, are becoming web-enabled faster than you can say "Oh no, not something else to design for!"

They present a new range of issues to address. (Try navigating a website with a TV remote.)

There are adaptors for everything, so users can browse with almost anything! Source: Plinkk

A whole new type of web browsing could arise: the ability to browse websites by re-enacting the YMCA song by way of a HAL-9000-style geek-mocking device called the Kinect (i.e. with gestures and movement).

And the ways we code, design and use the Web could change as well. Personally, I'm all for learning to code for all kinds of devices. Nothing seems cooler than waving a Wii-mote like a ninja in front of a webcam that's attached to a 100-inch LCD display while telling clients, "Yes sir, it's all part of my job."

Vehicles and Home Appliances

Another advancement in our consumption of the Web, beyond TV, has been the inclusion of web-enabled devices in trains, planes and automobiles. The ability to watch your favorite YouTube clips while on the move is quickly becoming standard — though we still need safety devices bolted on to ensure that we don't get distracted by the hamster dance video while driving and crashing into a lake.

Having web access in your car makes you wonder whether you're addicted. Source: battlecreekcvb

Yet with this next area of expansion, the web-enabled movement gets still more bizarre.

You may think the kitchen is one place that will remain Web-free, but I'm sure you have household appliances with some degree of cleverness.

In the past, we had glimpses of Internet-enabled appliances, but at the Consumer Electronics Show and other technology expos this year, companies announced that ovens, refrigerators, dishwashers and microwaves would get "webified."

The prospect can be terrifying as well as exciting. I'm sure your immediate thought, like mine, is of an oven crashing like IE6 and setting your house on fire.

Conclusion

From the early days of computers and laptops to the Mobile Web movement to the feature-filled tablets, watches, MP3 players, netbooks, e-book readers, radios and other digital goods becoming Web-friendly, the way we access the Internet has come a long way.

Yet, even as you read this, the gap between devices is increasing, the range of platforms is increasing, and, before we know it, we might be living in houses of the future powered entirely by Rickrolling and troll-hunting.

As web designers and web developers, we need to account for a wide range of platforms in order to make our users' lives convenient.

The Internet is a grand social platform with nigh-endless possibilities for applications and services. We need to rethink what we choose to (or choose not to) support.

Luckily, there will always be cool new stuff for designers and developers to wrestle with, and thankfully, few of us will go out of business!

The Proxority Principle in Web Design

In web design, the position of design elements and the layout of web pages is everything. So many cool, exciting techniques are available to help us lay out our designs (especially with CSS3 at our disposal) that we often forget that structure is as important as aesthetics.

How do you determine where content should appear, and how can a well-oiled interface increase website readability? This is what we'll aim to uncover in this article.

We're going to examine a basic technique that could help you improve your general content flow, and, for lack of a better term, I'm going to call that technique the proxority principle (a portmanteau word that combines "proximity" and "priority").

Communication through Design

Designers already understand that the relationships between objects on a page matter. That's why when we create a design, we think about visual hierarchy, visual weight, Gestalt psychology, the distinctiveness of important elements and other principles that affect relationships between the various components of a web page. It's one of the reasons why we tend to get neurotic when it comes to navigation menus, headers or footers.

With so much going on inside the average web page, getting the right content to users in the right place and at the right time is quite an achievement!

If your content isn't structured suitably, there can be a number of downsides, such as:

- Critical information getting lost or skipped
- User interaction issues like error-proneness or confusion
- Reduced web accessibility for screen readers

The proxority principle, in a nutshell, puts forth the idea that if we can prioritize our content to ensure that the most relevant material is visible and appealing, users will immediately be drawn to it.

This principle asserts that all related, important content should be grouped or joined together whenever possible to allow flow and feedback.

The art of this technique isn't in the theory, because we often lay out content logically as we write it (headings, subheadings, bullet points, etc.); instead, it's in the planning stage.

Proxority Principle in Site Navigation

Consider something like a navigation menu. One of the first things we do when producing the information architecture of a site is to organize pages and links into one cohesive structure, after which we add categories or subsections if appropriate.

This technique leads to the development of drop-down menus and other unique browsing aids which help us to further bind content that lacks proxority.

When planning navigation menus, one must pay particularly close attention to the value of pages and their connections, in order to make them a perfect example of proxority in action.

This particular technique works for any style of website, so whether you're scaling a huge service-heavy but content-light layout (like Amazon) or a content-heavy but feature-light design (like a blog), the technique should be of use.

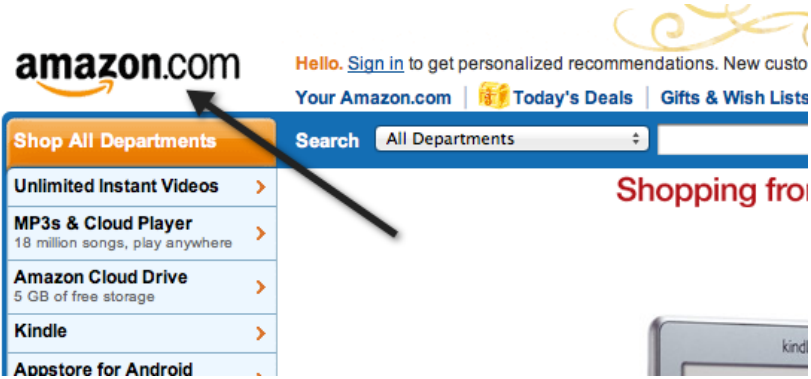
Better yet, the principle can help you organize your website's information architecture; the proxority between pages is as important as what exists upon individual pages.

If you find yourself struggling to determine where stuff should be placed, this strategic guide will help you.

Priority: Boost the Best, Weed out the Worst

Many of us know only too well the benefits of prioritizing. The priority we give our content plays a huge part in the perceived value it has upon a page.

A site's logo/name, for example, is recognizable because of its critically high placement, usually in the top left-hand corner, and it maintains visibility in that position on every page of a website.



A logo should be dominant over all other objects on a page. That's its proxority.

Rate Each Element's Value

To identify which pieces of a web page are most critical and important, we need to begin by examining every object in a layout, no matter how small.

Rate them based on their perceived value (according to what your visitors need to know or are likely to want to know) and their functional value (according to what contributions they make to the website, such as functionality or advertisements).

You can do this either by taking a screen capture (or printout) of the entire page and annotating it, or by producing a list of everything that appears on a page. This exercise will help you reassess the value of your website's content.

Rate images, media, content (at a paragraph level), and everything else according to this numbering system:

Rating	Description of Element
1	The website cannot function without this.
2	This adds benefits but is non-essential.
3	This supplements or reiterates content.
4	This is redundant or wastes space in some other way.

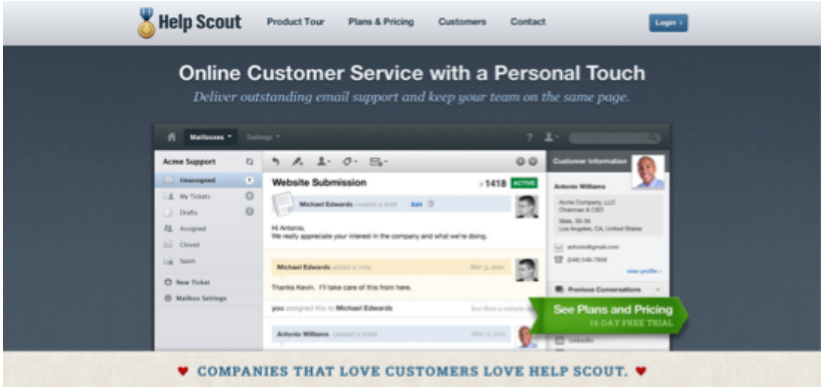
Eliminate Unneeded Elements

When you've gone through everything, review the results.

Before we go any further, it's probably worth mentioning that if you find content, links or objects that are no longer useful or don't contribute anything, remove them.

Eliminating clutter from an interface is tough, but reductionism improves the general user experience of a website.

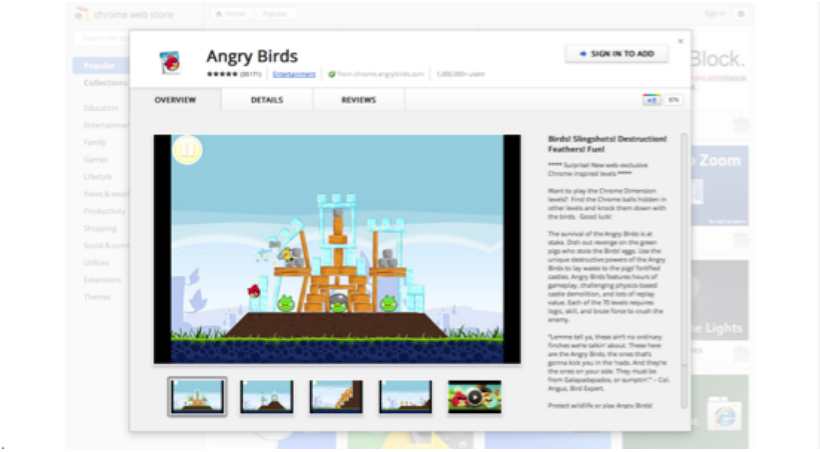
For elements rated at a 3, removal might be a bit harsh, so consider rewording or attaching things together to add value.



This website arranges its critical information into clearly defined segments for readability.

After you've identified the stuff that can be deleted, merged or moved, look at everything you've given a rating of 2. These elements can be the toughest to deal with because you want visitors to benefit from them, but you don't want to overburden them.

One solution to giving these these needed, but non-essential elements an appropriate proxority is to use progressive disclosure: make content appear on demand with drop-down menus or tooltips, or display it further down the page so that it's still available but less prominent.



Many websites use progressive disclosure to avoid swamping their users with details.

Proximity: Flow, Feedback and Functionality

We've considered the importance of prioritizing every asset on your page, identifying which bits have more sway than others, eliminating the fluff that has accumulated, combining weak material into a strong structure and pushing the less critical data out of the field of vision.

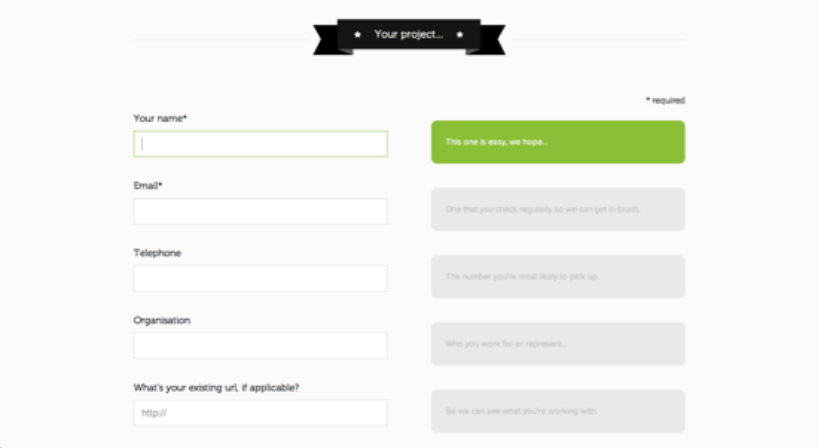
We now need to take all the remaining content and follow through on the second part of the process: to connect everything logically and put it all back together, as if it were a jigsaw puzzle — or a storybook, wherein the plot develops at consecutive points.

You should be left with everything that needs to be on your website, in its most diluted form, with supplementary content either hidden down the page or waiting in tooltips and extensible data boxes.

Rate Important Elements in Relation to Each Other

Go back over everything to which you have assigned ranks of 1 and 2, and rank them again; number everything according to what order you believe readers need to know about it.

If everything ends up where it should end up, it will all make perfect sense when you read it aloud.



The developers and designers behind this website clearly understand the need for organization and feedback.

Redesign

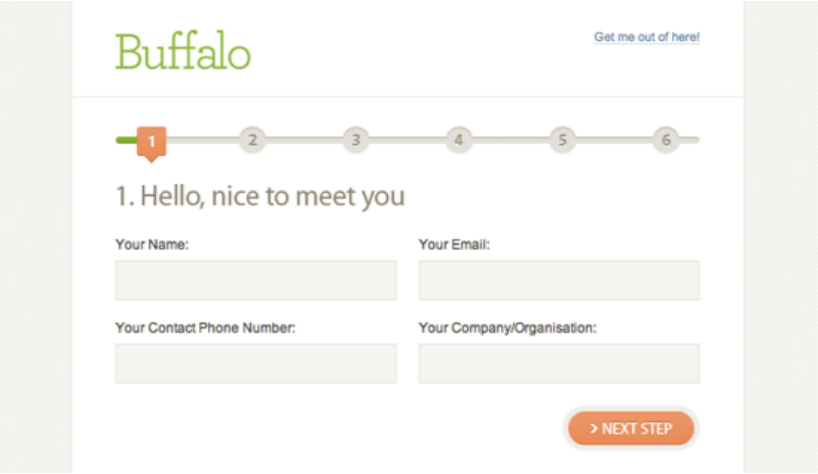
Once you've got everything labeled, re-shuffle your source code to match the new reading order. Pay special attention to bits of content that connect to or depend on other pieces of content (such as image captions), and put them as close together as possible.

Then, make the necessary changes to your CSS and JavaScript.

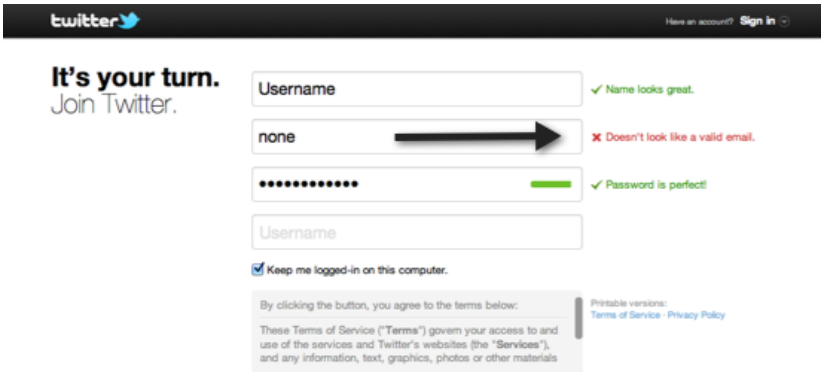
Proxority: Examples and Patterns in Action

Many websites already exhibit what I'd define as high proxority in that they take great care to use techniques that account for both priority (bringing attention to certain elements) and proximity (making reactions happen directly next to or above the objects being interacted with).

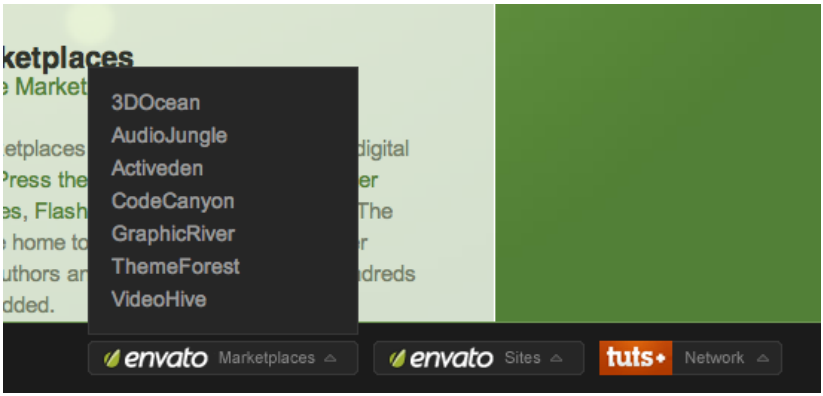
Below is a showcase of a few examples of best practices in use. Some we've briefly mentioned before, and others are being introduced here. By following similar practices, you can avoid user confusion and increase reading efficiency.



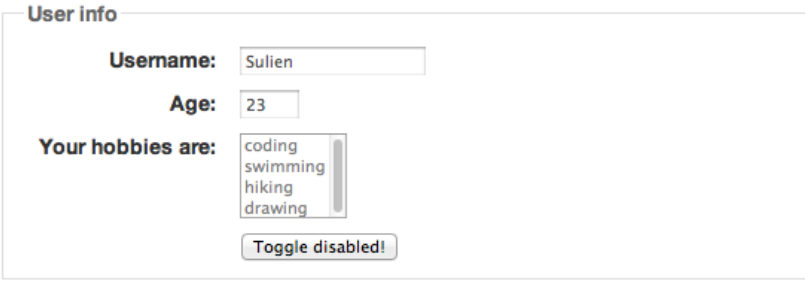
Using progressive disclosure to track the progress of a form.



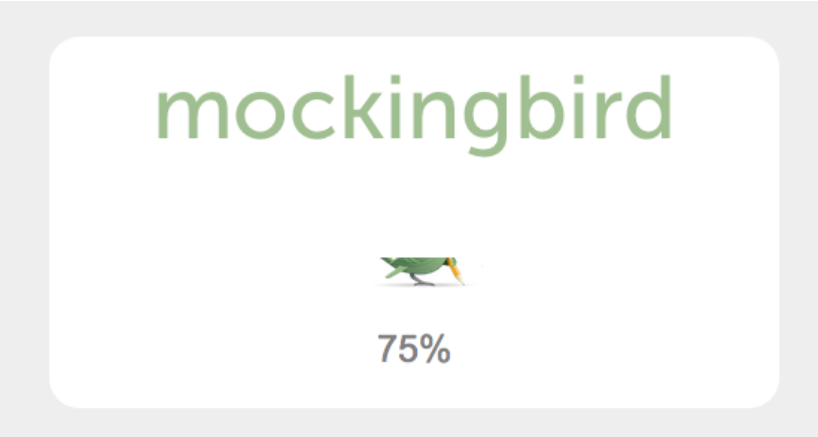
Informing visitors of errors as they enter data.



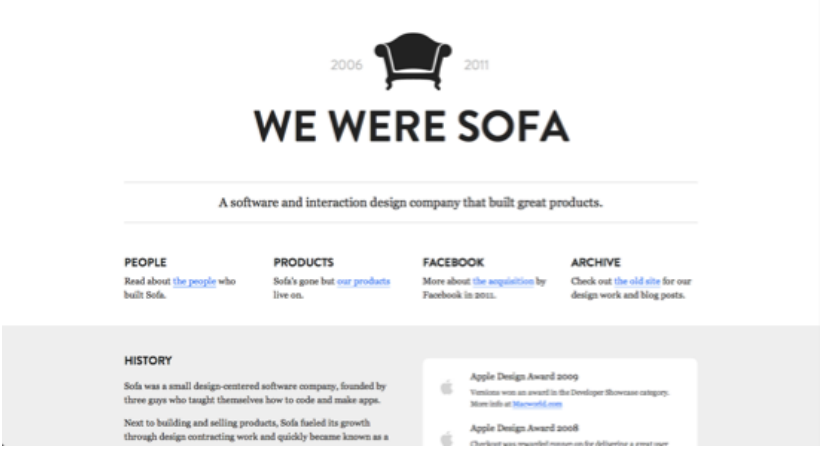
Drop-down menus expanding close to the cursor icon.



Input objects disabling once they've been submitted.



Progress bars showing loading progress.

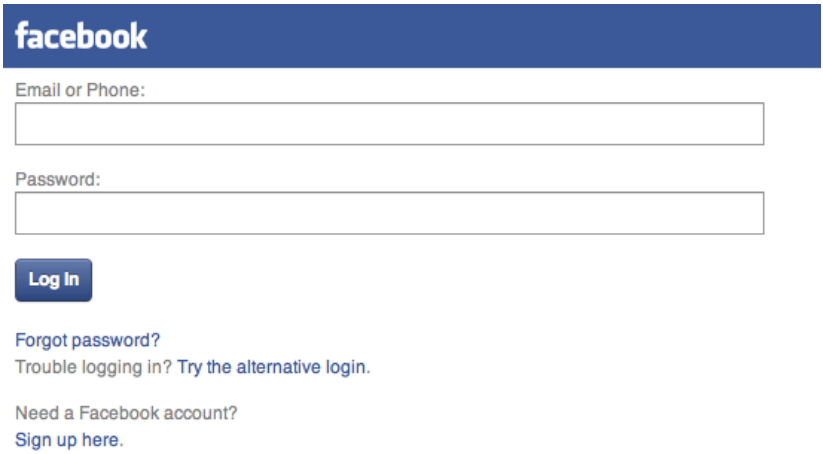


Content or light boxes expanding upon user interaction.

Proxority: Origami for the Web

The proxority principle posits that everything you find on a web page can be assigned a value and a place in sequence, in relation to the objects that surround it. This idea has existed since the early days of the Web, but too few designers pay enough attention to it. Think through what is actually needed, where it is needed and when it

should appear (as opposed to simply putting all of the content on the screen, in its entirety, in an order that "looks pretty"). The need for such techniques is increasing, especially given the proliferation of handheld devices and the idea of designing with a "mobile-first" philosophy.



If content isn't worthy of a restrictive mobile layout, why is it needed in the desktop layout?

If you have ten spare minutes, give this simple activity a try. Go through your website and weed out anything that isn't offering what it should. Make existing objects provide greater value to users (or use less space), and don't be afraid to reorganize your code and its content to ensure that what's needed is what appears. Oh, and if you do feel tempted to make actions elicit responses, ensure that users know that your website is responding; after all, you don't want them clicking "submit" ten times in a row, only to fail.

Sources:

- https://chrome.google.com/webstore/category/extensions
- http://builtbybuffalo.com/planner
- http://madebysofa.com/

Getting the Most Out of QR Codes Using URI Schemes

Lately, everyone has been talking about the potential of the QR code. It has become the Internet's equivalent of traditional barcodes (like those you'd find on physical goods at your favorite retail store).

Someone can take a quick snapshot of a QR code with their smartphone and immediately have a website up and loaded, so we could print QR codes on paper and physical goods such as business cards, magazine ads and posters in order to lead people to our site.

But, more often than not, web developers don't use QR codes to their fullest potential.

In this article, we'll discuss a technique that will unlock the full potential of QR codes through URI schemes.

How QR Codes Work in a Nutshell

Let's learn how the QR code mechanism actually works. QR codes, at their core, are specially generated images that work like barcodes.

Certain commands (known as responses) are built in, and when a QR code is captured by a camera (usually one on a smartphone or tablet), the image of the QR code is processed and then the built-in response is carried out.

A common QR code response is to open a certain website in a web browser on the device that captures the QR code.

QR codes look like the above and contain information that can be captured by a camera and interpreted by a smartphone, tablet or computer.

QR-Code Readers

This is where things get interesting. QR-code readers — the apps that scan the codes and perform the actions — are not equal when it comes to functionality. Some are able to recognize QR codes and do all sorts of amazing things, and others can simply open URLs or display text.

Most smartphones have the same set of basic features (address book, calendar, texting functionality, alerts and Internet browsing), and a good QR-code reader can cater to all of those.

With low-functionality apps in the mix, though, URL-opening is likely the only reliable function that we can expect most QR-code readers will have.

Here's a list of features that QR-code readers are generally equipped with:

- Displaying text
- Setting up system alerts
- Adding events to calendars
- Opening URLs in a browser
- Collecting contact information (including vCards)
- Sending email
- Sending text messages (SMS)
- Geolocation
- Calling other phones
- Connecting to WiFi

The Idea: Using URI Schemes

Since browsers can already launch default email clients via a simple mailto URL (browsers have supported URI schemes like these for years), could the same technique be used to offer in-app functionality for other products that have non-native URI schemes and that a QR reader wouldn't support by default?

The answer is yes!

If the application supported the scheme, and if the user had the app the QR represented installed, it would work beautifully — and it wouldn't matter which QR-code-reader was used.

The Possibilities

Here's a list of some cool things that are possible:

- Launching native Apple apps (Mail, Phone, FaceTime, Text, Map, YouTube and iTunes)
- Running JavaScript bookmarklets
- Opening certain IM clients (Skype, AIM, MSN, GTalk, ICQ and Yahoo)
- Opening special applications (IRC tools, feed readers, FTP clients and SVN repositories)
- Opening any application that registers a URI scheme when it installs on a platform

What makes this particular technique so amazing is that it doesn't rely on the QR-code reader being very advanced; all it needs is for the app creator (be it Skype, Evernote or Angry Birds) you want to launch and interface with to use the system development API to register a URI scheme.

In the case of Apple, it's the Cocoa Touch OpenURL method of the shared UIApplication object, and for other platforms (like Android), there will be an equivalent somewhere in the documentation.

From scanner to browser to application — the possibilities are endless!

Below is a list of some common non-http URI schemes:

Application	**URI Scheme or Protocol**	**Query Strings**
Default e-mail application	mailto:<email>?query	Subject
		CC
		BCC
		Body
Default phone application	tel:<number>	N/A
Default SMS application	sms:<number>	N/A
Chat Room client	irc://<url>:query	port
		channel
		password
Syndication feed reader	feed:<url>	N/A
Apple FaceTime	facetime:<number>	N/A
Skype client	skype:<username|number>?query	add
		call
		chat
		sendfile
		userinfo
Google Talk client	gtalk:query?<email>	chat
		call
Windows Live Messenger client	msnim:query?<email>	add
		chat
		voice
		video

Yahoo! Messenger client	ymsgr:query?<email|number>	sendim
		addfriend
		sendfile
		call
		callPhone
		chat
		im
		customstatus
		getimv
AOL Instant Messenger client	aim:query?<username>	goim
		goaway
		addbuddy

What does this mean for us? As it stands, there are actually quite a number of applications for the desktop (as well as mobile), which already use URI schemes.

Take Skype, for instance. Do you want people to be able to capture a QR code that causes their phones to call you via Skype on their handsets? That can be done (at least on iOS) right now.

It can also be used to run several native apps, launch JavaScript bookmarklets (perhaps including apps that run within browsers) and access the usual FTP and IRC protocols (if associated with something).

Here are some useful links for developers:

- Overview of Existing URI Schemes - https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml
- Apple iPhone URI Scheme Reference - https://developer.apple.com/library/content/featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html
- Android Custom URI Scheme Code - https://stackoverflow.com/questions/3471503/how-to-listen-for-a-custom-uri

Now that you know a bit more about QR codes and have gotten a brief overview of things you need to know to get started, I'll outline the steps to the technique below.

Note: The technique I'll be describing below can even be used to execute JavaScript code, so under the wrong set of code-writing fingers, this could be exploited.

Step 1: Choose the Response You Want to Perform

First, decide what you'd like the QR code to accomplish. Obviously, you can use the regular QR-code generator stuff — like sending text messages to phones or sending emails — but let's try something trickier.

We'll create a Skype QR code that will automatically launch the app and make a voice call to a Skype contact. (This will only work if the user that scans the QR code has Skype installed).

Step 2: Create a URL Using a URL-shortening Service

Many QR-code generators suffer from a particular problem: if you try to put a mailto or other non-http link into them, they sometimes think that an error occurred (this also affects some reader apps).

This, of course, won't get our Skype launcher anywhere near being cross-QR-code-reader-compatible, so we need another solution — and, luckily, a common one exists that can help get this URL into the browser (and help web developers monitor click stats too)!

We're going to use a URL-shortening service. These services are rarely fussy about what you insert into them, and they push out well-formed URLs with the HTTP header required for best-possible compatibility (most QR readers know how to open basic URLs).

A URL-shortening service such as TinyURL will turn that unconventional URL into a generally acceptable one.

For this QR, we'll call the Skype contact named echo123 — this contact is the free voice-quality testing contact that Skype offers.

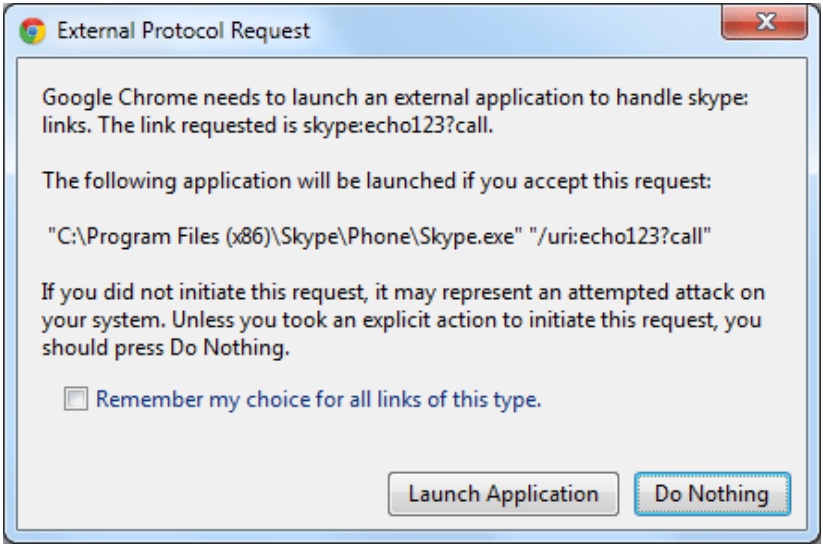
For our URL-shortening service, I'll use TinyURL, but you can use one of the many other URL-shortening services out there such as bit.ly, is.gd and so on.

Just enter your chosen URI scheme into the shortener.

This is the result of my TinyURL link conversion:

URL	skype:echo123?call
To	http://tinyurl.com/echo123skype

If you click on the URL above, you might see something like the following (if you're using Google Chrome on Windows):



Step 3: Generate the QR Code

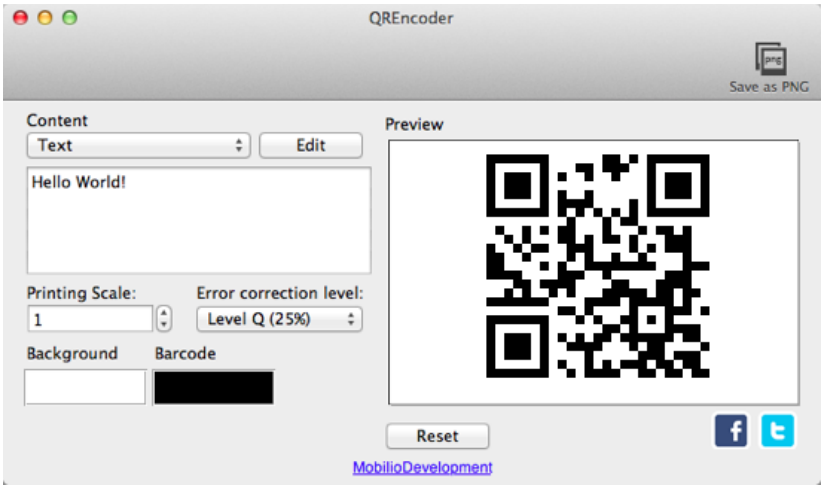
Now take that generated URL — which will actually work if you have the desktop version of Skype installed for Windows or Mac — and generate a working QR code from it.

Check that the shortened URL works in your web browser; the link should launch Skype.

There is an abundance of choice when it comes to ways of generating QR codes; free tools exist all over the Web.

Below are a couple of free generators:

- QR Stuff (online tool) - https://www.qrstuff.com/
- QREncoder (Mac App) - https://itunes.apple.com/us/app/qrencoder/id452695239?mt=12



After you've generated your QR code, test it. Download one of the available applications for your mobile device (if you don't already have one) and take a snapshot of the QR code you've produced.

Below are some free QR-reader products for mobile platforms:

- Scan (iOS) - https://itunes.apple.com/us/app/scan/id411206394?mt=8
- QR Droid (Android) - https://play.google.com/store/apps/details?id=la.droid.qr&hl=en
- QR Code Scanner Pro (Blackberry)

What should happen, if you followed the instructions correctly and if you've got the Skype app installed, is this:

1. Your QR code will be identified by the QR reader app
2. The QR reader app will notice that the QR code contains a URL (thus it will skip the compatibility quirks of the readers)
3. The TinyURL link will open in the default browser
4. It will redirect to the Skype-call URI scheme and launch Skype

Pretty amazing stuff for one clever image!

The only thing left to do with your wonderful creation is to upload it to the Web (as I have with the finished example below). I recommend, as a best practice, that you consider providing a link to the URL being snapped in the QR image itself (for desktop users).

Beyond that, you could spruce up your QR codes with a bit of color or some limited artwork if you're feeling extra creative.

The Finished QR Code

The above QR code will call echo123 on Skype if you have the app installed.

Conclusion

The power of this technique rests in the opportunity it gives designers and developers. A URI scheme exists between app developers and the browsers that support them (luckily, that's all of them!).

Using a simple redirect (provided by you or a third party) and these already existing and well-supported features, you can launch applications, interact via query codes, execute JavaScript and do all sorts of wonderfully creative things.

QR codes are being increasingly adopted and have never been more widely used. With this single technique you can break past most existing limitations.

Sources:

- https://en.wikipedia.org/wiki/QR_code
- http://web.archive.org/web/20110724083725/http:/www.ianr.unl.edu/internet/mailto.html
- https://developer.apple.com/library/content/featuredarticles/iPhoneURLScheme_Reference/PhoneLinks/PhoneLinks.html
- https://developer.apple.com/library/content/featuredarticles/iPhoneURLScheme_Reference/SMSLinks/SMSLinks.html
- https://en.wikipedia.org/wiki/Internet_Relay_Chat#URI_scheme
- https://en.wikipedia.org/wiki/Feed_URI_scheme
- https://www.skype.com/en/developer/create-contactme-buttons/
- http://juberti.blogspot.co.uk/2006/11/gtalk-uri.html
- https://en.wikipedia.org/wiki/Yahoo!_Messenger#URI_scheme
- https://en.wikipedia.org/wiki/AIM_(software)#URI_scheme
- https://tinyurl.com/
- https://bitly.com/
- https://is.gd/

Why We Still Need Web-safe Fonts

During the early days of the Web, there wasn't a standard font that could be rendered across all platforms. However, some fonts — like Arial, Helvetica, and Times New Roman — were more likely to be installed in a person's computer. These popular system fonts are called Web-safe fonts. It was the best practice in web design to stick to them.

Things have changed. It's now safe to use Web fonts — a technique for rendering any remote font file in a web page using @font-face. This give us more creative freedom and a much wider range of font options.

The @font-face rule has been around for close to 13 years, first seen and supported in Internet Explorer 5.5[1].

And @font-face has been formally included in the most recent version of W3C's CSS standards (CSS3) — it was taken out in CSS2.1 — and so modern browsers (e.g. Chrome, Safari, IE, and Firefox) support the rule.

And with services like the free Google Fonts API, implementing Web fonts is easy. For instance, using the Google Fonts web service, it takes two lines of CSS to render even a relatively unknown font such as Bigelow Rules in the vast majority of browsers:

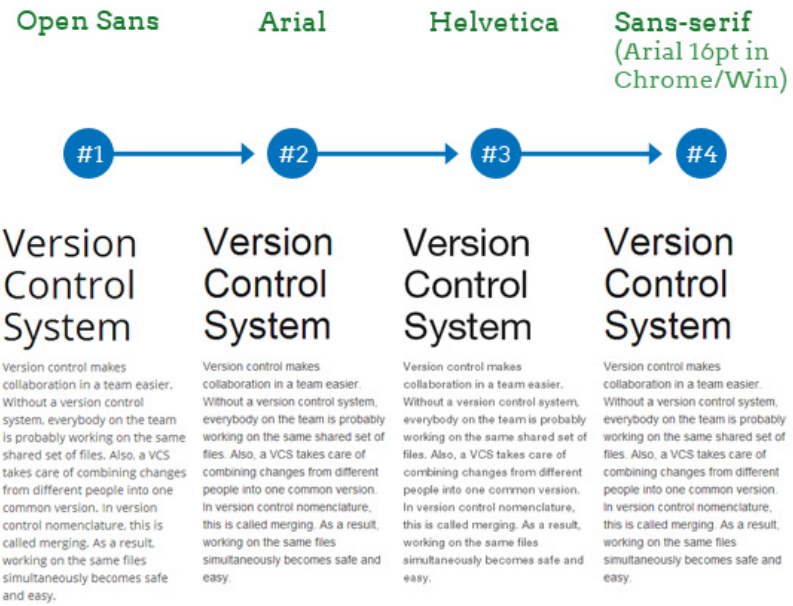
```
@import url(http://fonts.googleapis.com/css?
family=Bigelow+Rules);
body { font-family: 'Bigelow Rules'; }
```

But even with near-perfect support of the @font-face rule, as a good practice, we should still use CSS font stacks. Moreover, our CSS font stacks should still include Web-safe fonts as well as generic font families like sans-serif and serif.

For example, even if we were to use the most popular font on Google's Web font service (which is currently Open Sans), we should still use a CSS font stack that includes similar Web-safe fonts, with the last declaration in the stack being a generic font family (sans-serif in the case of Open Sans):

```
body {font-family: "Open Sans", "Arial", "Helvetica", sans-serif; }
```

In Chrome, this is the fallback-rendering sequence:



As shown above, using a good font stack assures us that we are gracefully degrading our HTML text in case our chosen font is for some reason unable to load.

Why We Still Need a Web-safe Font Stack

Web-safe fonts and CSS font stacks seem like outdated Web design practices, especially since @font-face has great support.

Right now close to 90% of the Internet's users use a browser that supports @font-face[2].

But if you've ever considered dropping your CSS font stacks, below are a few reasons that might change your mind.

Incomplete Fonts

If certain characters in your font are not available, the browser will attempt to render those unavailable characters using the next font in the stack. But if you don't have a font stack, it will use the browser's default standard font.

For instance, the Libre Baskerville font doesn't have the ™ character.

The first example below shows how the missing character renders in Chrome without a font stack, while the second example stacks "Times New Roman" and serif in the style rule:

```
font-family: "Libre
Baskerville";
```

Logo is TradeMarked™

```
font-family: "Libre
Baskerville", " Times New
Roman", serif;
```

Logo is TradeMarked™

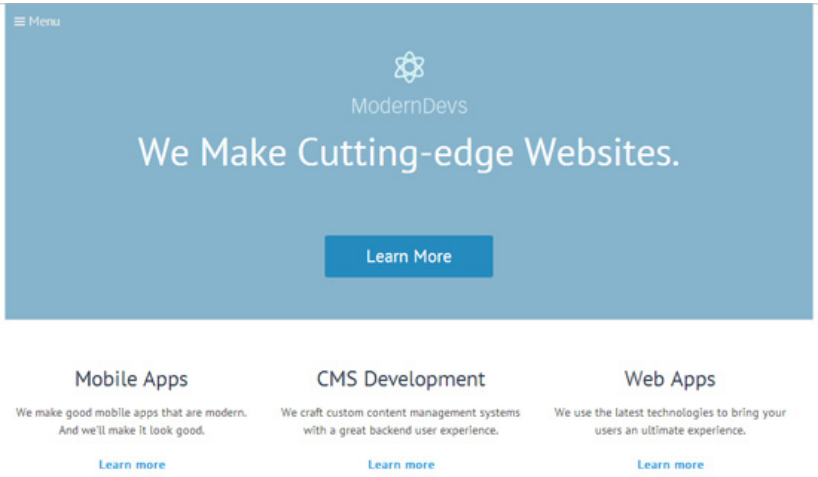
At least with a good font stack, the ™ character looks similar to the preferred Libre Baskerville font.

Network Issues

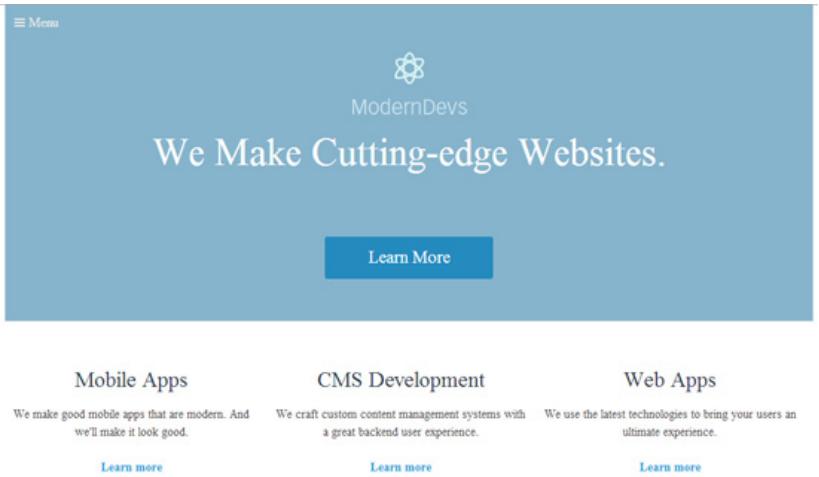
Loading a remote font using @font-face requires an Internet connection. If the web server responsible for serving the font is unavailable or is down for maintenance, the browser will use its default standard text, unless you specify a Web-safe font in your CSS font stack.

For Google Chrome, the standard text (on Windows) is "Times New Roman". This can be bad news if we were using a sans-serif font, and the web server or the content delivery network where the font file is located goes down.

For example, check out this mockup of a web design that uses the PT Sans font:



If our font stack didn't have Web-safe fonts in the stack and a network interruption happened, our web page would render like this (in Chrome):



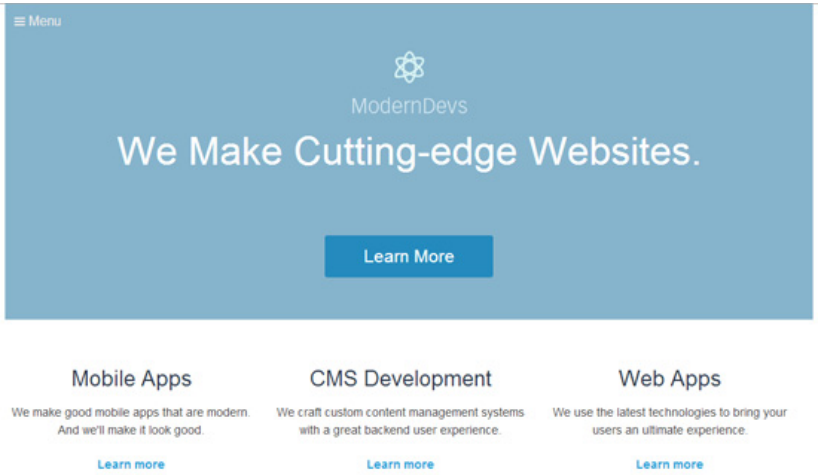
The web page looks completely different because Times New Roman greatly affects the visual message of the design.

But if we used a font stack that included Web-safe fonts, we can mitigate some of the aesthetic issues that come with network problems.

Using the font stack:

```
font-family: "PT Sans", "Helvetica", "Arial", sans-serif;
```

We are able to lower the visual impact caused by a network interruption:



@font-face Can Be Turned Off Client-Side

Some web browsers provide the option to disable the downloading of font files. In most cases, disabling remote font files in a web browser is obfuscated, but it's possible. Here's a forum post for turning off remote font downloading in Firefox and Chrome, and here's one for Opera.

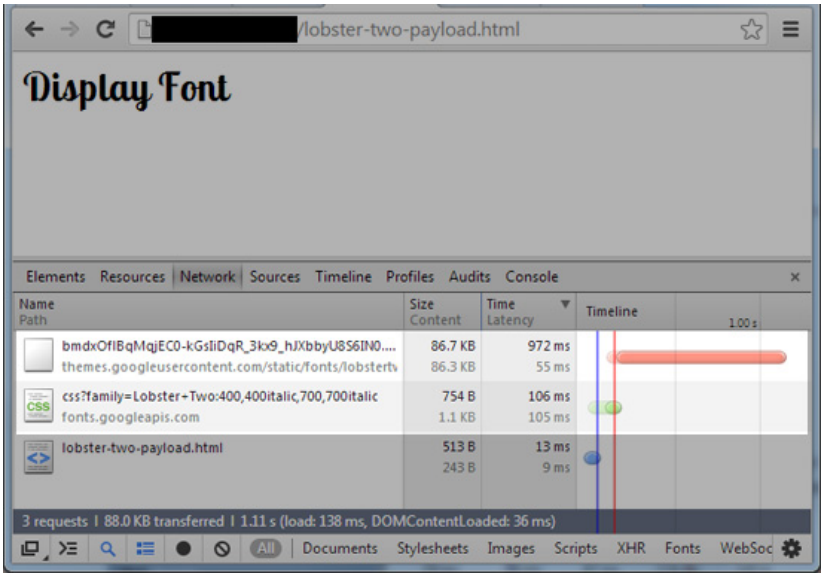
Why would someone want to disable remote fonts? To speed up web page load times, which is especially desirable for Internet users who have slow Internet connections.

To illustrate how Web fonts impact Web performance, let's use a display font such as "Lobster Two".

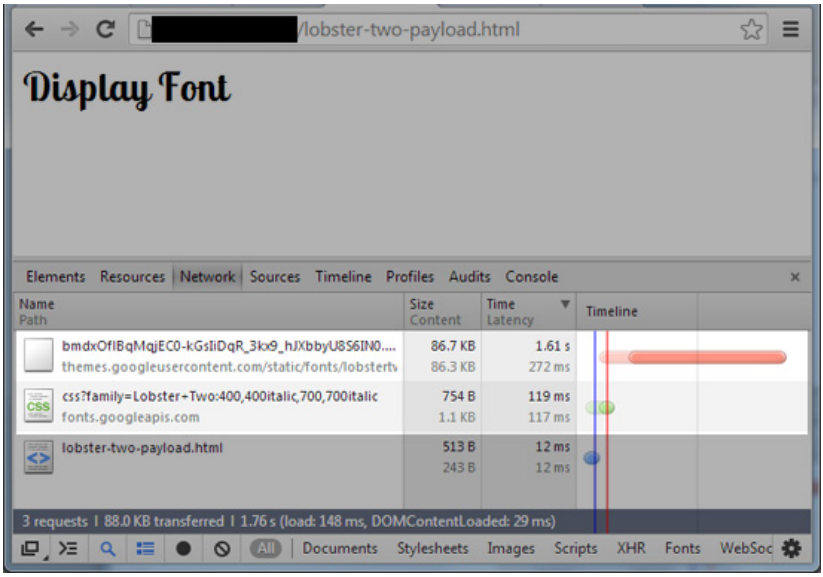
The following markup was used to test the Lobster Two payload locally using an HTML document called "lobster-two-payload.html":

```
<html>
    <head>
        <link href='http://fonts.googleapis.com/css?
        family=Lobster+Two:400,400italic,700,700italic'
        rel='stylesheet' type='text/css'>
        <style>
        body { font-family:"Lobster Two"; }
        </style>
    </head>
    <h1>Display Font</h1>
</html>
```

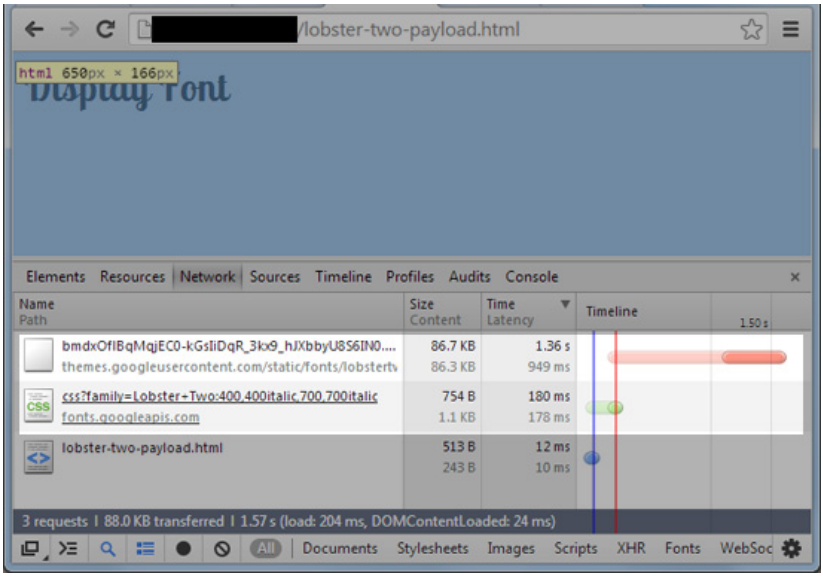
The test conditions are a broadband Internet connection and an unprimed cache (meaning that the browser cache was cleared for each test).



#1 result. Total time: 972ms



#2 result. Total time: 1.61s



#3 result. Total time: 1.36s

In the condition tested, it would take an average of 1.3 seconds to render the <h1> tag fully for someone who has an unprimed cache.

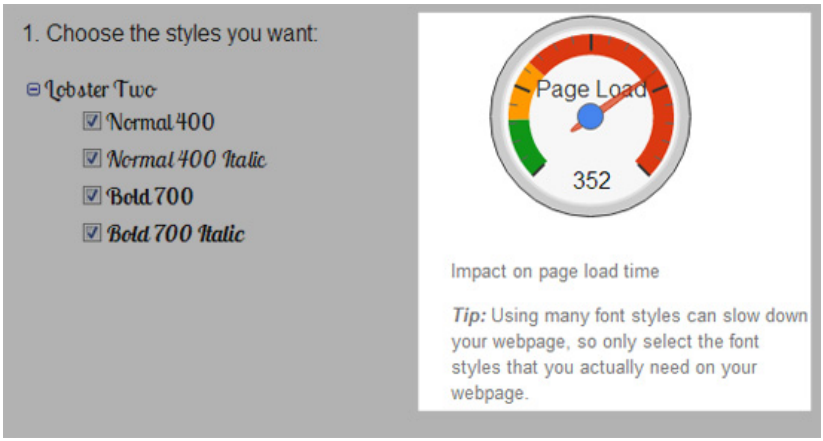
This means staring at a blank page for 1.3 seconds before the user can read the text, because Chrome's default behavior is to show no text while the font is still loading[3].

Consider that 1.3 seconds is the average load time using a broadband connection to download the font files from Google's servers. Imagine how long it would take in less-than-ideal situations, such as slow mobile networks and using a shared-hosting server to serve the font.

Without the Web font, the same document would finish rendering in an average of only 0.012 seconds, which means the font increased the load time by 10,733%. That is a hefty price to pay for rendering a novelty display font that's not functional enough to be used for critical website content. That's why some users choose to disable remote font files from loading.

If we want to gracefully degrade our web design in situations where the user has chosen to disable remote font files, we should use Web-safe fonts in our font stack.

It must be noted here that you don't have to load the entire font family. And if you use the Google Font service, you can selectively load just the font variations you need.



Web-safe fonts = Cheap and Easy Graceful Degradation

Though it's rare, @font-face isn't supported in a few web browsers, especially old ones.

Earlier, I mentioned that 90% of Internet users use a browser that supports @font-face.

Not having a font stack that includes Web-safe fonts and a generic font family means that we aren't controlling how our web design degrades for at least 10% of the Internet's users.

The time and effort required to use a simple CSS font stack that includes Web-safe fonts is very small, so there's little reason not to continue doing it.

References

1. The SitePoint CSS reference for @font-face has a table showing @font-face as being supported in Internet Explorer 5.5+.
2. According to The Can I use @font-face Web fonts compatibility table, 89.84% of the Internet population uses a browser that has full or partial @font-face support. This data is from StatCounter Global Stats, November 2013.
3. See the description of the browser-specific Web font rendering behaviors on the Google Developers site. This is outdated because as of Firefox 4 (officially released March 2011), loading Web fonts now behave similar to WebKit-based browsers.

Sources:

- https://en.wikipedia.org/wiki/Web_typography#Web-safe_fonts
- https://developers.google.com/fonts/
- https://www.techpowerup.com/forums/threads/disabling-web-fonts-in-mozilla-firefox-and-google-chrome.184198/
- https://caniuse.com/#feat=fontface
- http://gs.statcounter.com/
- https://developers.google.com/fonts/docs/technical_considerations
- https://www.paulirish.com/2009/fighting-the-font-face-fout/#update2011

15 HTML Questions for Testing Your Knowledge

Think you know HTML? Most developers probably feel they have a good handle on modern Web standards. It's with this in mind that I offer you a challenge.

Below you'll find some HTML questions that will test your familiarity and understanding of the markup language. And if you're conducting job interviews, you can use these questions to gauge a candidate's knowledge of the Web's standard markup language.

You will also find questions about closely-related markup languages and standards such as XML, XHTML, and microformats in order to really test your mettle.

There isn't a time limit or score-tracking for this "quiz", so take your time. Then, share your results in the comments.

How well do you know HTML?

Beginner Questions

Question 1

What's the name of the main international standards body that publishes HTML specifications?

> Answer: World Wide Web Consortium (W3C).
>
> The W3C was formed in October 1994. You can find the current recommended HTML specs at w3.org, the official site of the W3C.

Question 2

How many HTML heading levels are there?

> Answer: 6.
>
> The HTML heading elements are h1, h2, h3, h4, h5, and h6. You can use these elements to create a hierarchical outline of the contents of an HTML document.

Question 3

What's wrong with the following HTML markup?

<p style"font-size:10px;">Copyright 2015

Answer: The style attribute is missing an equals (=) sign.

But, did you know that under the current HTML standards, the double-quotes (") above are optional? As long as the attribute value has no spaces, you can drop the double-quotes. Thus, the following is valid HTML markup:

Copyright 2015

Learn more about the unquoted attribute value syntax in section 8.1.2.3 Attributes of the HTML5 specs.

Question 4

Which version of Internet Explorer was the first to natively support new HTML5 elements?

Answer: Internet Explorer 9 (IE9).

IE9 was released in March 2011. IE9 was the first to support new semantic HTML elements such as article and section, as well as canvas and inline SVG support, and more.

Question 5

What is the name of the metadata that allows you to set a value of initial-scale=2, causing a page to zoom to twice its natural size?

Answer: viewport.

In the following example, the viewport meta tag (as it's commonly called) is used to specify that the zoom level must be twice the device's width:

<meta name="viewport" content="width=device-width, initial-scale=2">

The viewport meta tag is not a standard metadata name for the meta element. However, it is well-supported by browsers. Quirksmode has a good guide on the viewport meta tag.

Question 6

What's the name of the microformat in the following example? Fill in the blank (???).

52.48, -1.89

Answer: geo.

geo is a microformat designed for semantically marking up geographic coordinates in HTML and other standard markup languages. Read more about the geo format in the Geo Microformats Wiki.

Question 7

What markup language do RSS, Atom and OpenSearch use?

Answer: Extensible Markup Language (XML).

XML is short for Extensible Markup Language. Learn more about the language by reading XML's W3C specs.

Question 8

What's the name of the new HTML5 element that begins with the letter K?

Answer: keygen.

The keygen element is for marking up a control element that generates a public-private key pair, which is typically used for encryption.

Intermediate Questions

Question 9

If a hyperlink points to a resource containing copyright information about the current web page's main content, what link type can you specify on the hyperlink?

Answer: license.

The license link type can be used as follows:

<main>

```
    <img src="freephoto.gif" />
    <a rel="license" href="copyright.html">Copyright info.</a>
</main>
```

Learn more about the license link type (and other hyperlink link types) by reading this guide.

Question 10

According to Microdata and Schema.org vocabulary, what's the name of the microdata boolean attribute which indicates that the element's descendants may contain information about the element?

Answer: itemscope.

When the itemscope attribute is specified on an HTML element, it informs search engines and web browsers that descendants of the HTML element may be carrying machine-readable information about the HTML element. Search engine like Google, Microsoft, and Yahoo! use microdata markup to improve their search results.

Question 11

What is the HTML5 element that represents a line break opportunity?

Answer: wbr.

The wbr element indicates a location in the document where there's a good opportunity to render a line break if needed. Quirksmode describes the utility of the wbr element if you are curious about this obscure HTML tag.

Question 12

What attribute can you use to specify a regular expression which describes a valid value for an input element?

Answer: pattern.

The pattern attribute exposes a client-side, machine-readable description of how an input element is being validated. This, in turn, can be used by software such as assistive technologies to help its users understand why the form submission was not successful. The attribute can also be used for client-side input validation logic in conjunction with JavaScript.

Expert Questions

Question 13

According to the HTML5 W3C Recommendation, how many states/values does the type attribute have?

Answer: 18.

You can see all states of the type attribute in section 4.10.5.1 States of the type attribute.

Question 14

Which HTML element can be used to express and annotate the pronunciation of East Asian characters?

Answer: ruby.

Ruby characters are annotative characters typically associated with East Asian (e.g. Japanese and Chinese) typography.

Here is an example from the HTML5 specs which uses the ruby element:

<ruby>日<rt>に</rt></ruby><ruby>本<rt>ほん</rt></ruby>

Question 15

Which ARIA landmark role does the HTML5 footer element default to, if the footer element isn't inside an article or section element?

Answer: contentInfo.

ARIA landmark roles are regions of a web page that contain navigational aids. Navigational aids are things such as a website's navigation menu or breadcrumb navigation. The contentInfo role is one of these standard landmark roles.

The contentInfo role states that the element contains information about the web page. As defined in section 3.2.7.3 Strong Native Semantics, if the footer element isn't inside an article or a section element, its role is implicitly set to contentinfo.

How Did You Do?

Don't feel bad if you didn't get them all right. I'd be quite surprised if anyone got the right answer for every single question. For the questions you managed to answer correctly, congratulations!

Sources:

- https://www.w3.org/TR/html5/
- https://en.wikipedia.org/wiki/Internet_Explorer_9
- https://www.quirksmode.org/mobile/metaviewport/
- http://microformats.org/wiki/h-geo
- https://www.w3.org/TR/REC-xml/
- https://schema.org/docs/gs.html#microdata_itemscope_itemtype
- https://www.quirksmode.org/oddsandends/wbr.html

15 CSS Questions to Test Your Knowledge

How well do you know CSS? Test your knowledge by trying to answer the CSS questions in this post.

The questions are divided into three categories:

- Basic CSS questions
- Intermediate CSS questions
- Advanced CSS questions

This set of CSS questions is a follow-up to our previous collection of HTML questions.

Basic CSS Questions

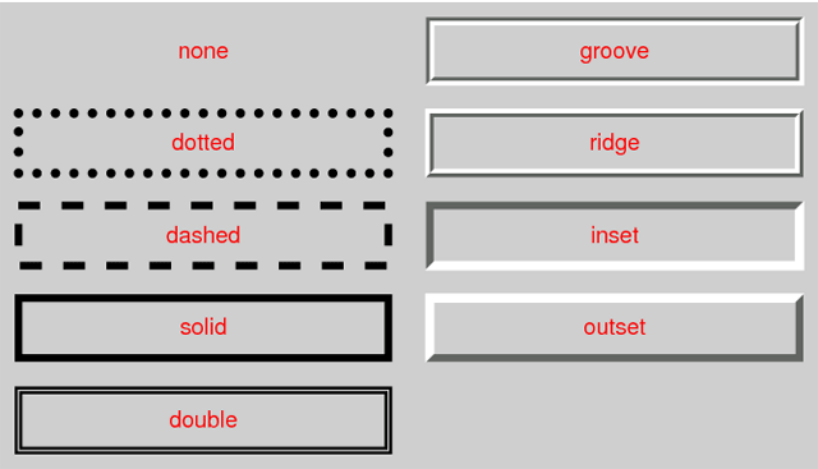
Question 1

Which of the following is NOT a valid border-style property value?

- dotted
- inset
- glazed
- groove
- solid

Answer: glazed

You can see all the border-style property values by reading the "4.2. Line Patterns: the 'border-style' properties" section in W3C CSS Backgrounds and Borders Module Level 3 specs.



Visual guide of border-style property values. Source: w3.org

Question 2

Which of the following is NOT a valid CSS length unit?

- cm
- dm
- em
- mm

Answer: dm

cm and mm are absolute length units. em is a font-relative length.

Question 3

What is the CSS selector which allows you to target every element in a web page?

Answer: The universal selector (*).

An example: The following style rule uses the universal selector to set the margin and padding of all HTML elements to zero:

```
* {
    margin: 0;
    padding: 0;
```

}

Question 4

Which two CSS properties allows you to hide an element but still maintain the space it occupies in the web page?

Answer: visibility or opacity

There are several ways to hide an HTML element with CSS.

Setting the visibility property of the element to hidden will hide the element. The element will still occupy space equal to its geometric size in the web page. For example, if the hidden element's dimensions are 100x100px, you will see an empty 100x100px space in the area where the element is located. Hiding an element can also be accomplished by assigning opacity: 0 to an element.

Hiding an element without maintaining the space it occupies in the web page can be done by setting the element's display property to none. Setting display to none renders the element as though it doesn't exist.

Question 5

There are 16 basic color keywords in CSS. Which of the following are NOT basic color keywords?

- olive
- fuchsia
- cyan
- aqua
- maroon

Answer: cyan

cyan is a valid color keyword. But it's not one of the basic color keywords.

The cyan color keyword is documented as being part of the extended color keywords.

Question 6

The font-style CSS property has four different valid values. Three of these values are inherit, normal, and italic. What is one other valid value?

Answer: oblique

Read the font-style docs on MDN to learn more about this CSS property.

Question 7

Which of the following two selectors has a higher CSS specificity?

- Selector 1: #object h2::first-letter
- Selector 2: body .item div h2::first-letter:hover

Answer: Selector 1: #object h2:first-letter

The specificity value of Selector 1 is 102. The specificity value of Selector 2 is 24.

Intermediate CSS Questions

Question 8

What is the ideal order of the following pseudo-class selectors in a stylesheet?

- :active
- :hover
- :link
- :visited

Answer

- :link
- :visited
- :hover
- :active

An element can match multiple pseudo-class selectors at the same exact time. That is the reason why the order of the pseudo-classes

above is crucial. We know that if two selectors are equal in specificity, by default, the selector farther down the stylesheet wins.

One situation where you can clearly see this issue is via a hyperlink element. Suppose that you hover your mouse pointer on the link, and then click on the link without moving your mouse afterwards. This situation means the link matches both :hover and :active selectors.

So if the :active style rule is above the :hover style rule — for instance — users will never get to see the :active style rule applied. This is because the :hover style rule will always overwrite it.

You can remember the ideal order by memorizing the acronym, LVHA.

Link → Visited → Hover → Active

Question 9

Which of the following CSS properties DOES NOT influence the box model?

- content
- padding
- margin
- outline
- border

Answer: outline

Here's a portion of the outline property's specifications:

"The outline created with the outline properties is drawn "over" a box, i.e., the outline is always on top, and does not influence the position or size of the box, or of any other boxes. Therefore, displaying or suppressing outlines does not cause reflow or overflow."

Question 10

When using media queries, which of the following is NOT a valid media type?

- tv
- all
- voice
- print
- braille
- tty
- embossed

Answer: voice

You can find all the valid media types in the Media Queries W3C specs. voice is not a valid media type. Though there is a speech media type.

Question 11

There are five generic font family values that can be assigned to the font-family property. Three of them are listed below. What are the other two generic font family values?

- serif
- sans-serif
- monospace

Answer: cursive and fantasy.

Question 12

What is the color keyword that will always be equal to the calculated color property value of the selected element/elements?

Answer: currentColor

Below is an example where the background-color and the border color will be equal to the color property value of .box elements:

.box {

```
    color: green;
    background-color: currentColor;
    border: 1px dashed currentColor;
}
```

The benefit of using the currentColor keyword is that we only need to change the color value in one place. We can just change the value of the color property, and the change will cascade to the other properties. This keyword works much the same way as CSS variables.

Advanced CSS Questions

Question 13

Which of the following is NOT a valid CSS unit?

- ch
- turn
- px
- ems
- dpcm
- s
- hz
- rem

Answer: ems

ch and rem are font-relative length units.

turn is an angle unit.

px is an absolute length unit.

dpcm is a resolution unit.

s is a time unit.

hz is a frequency unit.

Question 14

Which of the following color keywords has NOT yet been proposed in a W3C specification?

- blanchedalmond
- dodgerblue
- peachpuff
- orchidblack
- navajowhite
- tomato

Answer: orchidblack

Question 15

What is the CSS at-rule that can allow you to define the character encoding of a stylesheet?

Answer: @charset

UTF-8 should always be used as your CSS file's character encoding. If this is the case, then you don't need to declare a @charset rule.

Sources:

- https://www.w3.org/TR/css-backgrounds-3/#the-border-style
- https://www.w3.org/TR/css3-values/
- https://www.w3.org/TR/css-color-3/
- https://www.w3.org/TR/css3-mediaqueries/
- https://www.w3.org/TR/css-fonts-3/#generic-font-families
- https://www.w3.org/International/questions/qa-css-charset

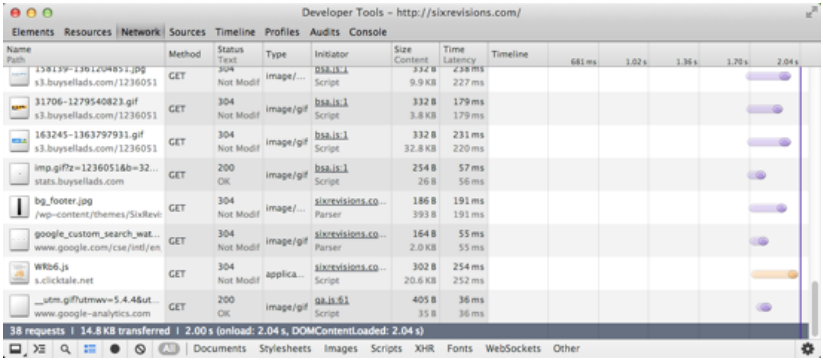
Web Agility: Pushing for Performance

In a situation that resembles “life imitating art, and art imitating life”, the Web is suffering an obesity epidemic. Despite knowing that performance is critical to design, pages are getting more HTTP requests, file sizes are expanding beyond Internet speed averages; and with the rise of frameworks, we’re often finding ourselves using a 30kb JavaScript file to justify 1kb of effects.

With this in mind, we’re going to examine the usefulness of performance budgets, how they could benefit you, how to set one up, what variables you should target, and why you should consider using them in your future projects. If you design or develop websites for clients, the results could win new customers or sales. If you run your own site, you’ll get many benefits too, especially if you run a popular site that handles millions of page views.

Performance Budgets

The idea behind setting performance budgets is a sound one. Because small, agile sites load faster than heavy ones, trimming the fat and preventing harmful waste has noticeable benefits to a sites agility, and potentially even it’s running costs. Being able to show clients the numbers to back the findings up could really help you justify the value you bring to client work. Therefore, having such a process could well be worth the added effort.



Of course, setting such targets and restrictions is easier said than done. Anyone who has been on a diet will tell you that temptation is never far away, and it's likely that because of this, few designers have felt comfortable enough to tighten their belts or commit to such changes. Despite constant warnings that surplus frameworks, web fonts, and other luxuries can clog up a site; our lust for aesthetics often result in sites that look minimalistic and free of bloat, but suffer the same problems as more feature rich designs.

Setting Goals and Targets

To create a performance budget, you first need to identify the areas which can affect a sites ability to function, one obvious example of this is file size. While some sites could cope with only 50KB available to them, larger sites will of course require scaling up. If you are trying to slim down an existing site, trying to get a 10-25% reduction is entirely possible, and could make a huge difference.

- Small site: Aim for 100KB total size or 25KB reduction.
- Medium site: Aim for 250KB total size or 50KB reduction.
- Large site: Aim for 500KB total size or 100KB reduction.

Another variable that can ultimately affect performance is a sites loading time. Users will often give up if nothing has happened within 3-5 seconds, so you should aim to boost the page speed. However, with connection speeds varying consistently, you need to be careful not to assume that what is fast for you will translate to your users.

- Basic HTML & CSS: Load within 1 second.
- JavaScript and animations: Load within 2 seconds.
- Images and media: Load within 3 seconds.

You set performance budgets to a range of scenarios. Depending on the work you're being asked todo, or how it will best fit into your workflow, you can use one of the following methodologies to set your limits or reduction ratios. If you're going to be setting budgets on a continual basis, it might be worth setting up a spreadsheet to track

your progress, and potentially see where the majority of your savings originate.

- Per site: Best used for calculating file size averages, say the ratio of images to content.
- Per page: Best used for calculating page loading times, and individual file size offenders.
- Per asset: Best used for setting goals relative to the weight or loading time of a single file.

Going on A Data Diet

When calculating performance goals, try to be realistic as there is only so much you can do before a sites quality and content will suffer. Once you have worked out what needs to be done, feel free to use the below tips and tricks to help you shift some of the excess weight. Keep in mind that certain actions will yield bigger results than others, for example, removing a couple of large images will do more than most markup tweaking.

File Size Variables:

- Bloated markup: Can you reduce the amount of code required to achieve a goal? Cleaning your HTML, CSS, JavaScript, and server-side scripts will make a difference. Minifying your code will also reduce the weight.
- Redundant code: If you have duplicate CSS properties, unused code snippets, code comments, pages of pointless content, or lots of frameworks that aren't necessary, remove them to save additional kilobytes.
- CSS and JavaScript: This tip dates back to the web standards movement. Get your CSS and JavaScript out of the HTML, and into separate files. It will save reloading the resources on every page visited by users.
- Feature creep: Does the site have any functionality that is rarely used? If you don't need it, get rid of it. Just remember to ask

your customers if they object beforehand, otherwise you could upset some visitors.

- Code Compression: If you don't have GZIP enabled on your site, ensure that you do it right away. While it will increase the pressure on your servers, the file size and speed savings are well worth it.
- Images and media: This is the area where much of a sites bloat comes from. Unfortunate as it may be; audio, video and images are very bandwidth hungry and data costly. If it's not necessary dispose of it.
- Conditional Code: Using media queries, JavaScript and CSS selectors, you can lazy load resources. Retina images can be swapped within stylesheets, and video quality can be adjusted using conditional JavaScript.
- Resource Optimization: For images you can't do without, use ImageOptim or SmushIt to squeeze every redundant byte from your visuals. For media, increase the compression levels for data savings.

Time Based Variables:

- Server speed: Picking a host that has good performance reviews and buying the right hardware to match visitor levels matters. Most hosts allow you to add or remove resources to cope with spikes in demand.
- HTTP requests: Putting raster graphics in sprites, using icon fonts over vector images, and bundling all of your CSS or JavaScript into a single cacheable files will reduce the queue for resource requests.
- Inlining resources: Do you have any tiny assets that you don't want to sprite as it only has a single page or instance use? Rather than making another HTTP request, embed it in the HTML or CSS using base64.
- DNS requests: When you first request a resource, the browser has to perform a domain lookup. Reduce the number of sites

hosting your content, and use HTML prefetching to help reduce the wait.

- Load latency: Get your JavaScript to the bottom of the page, and keep your CSS in the head. You want the layout to be immediately visible, and scripts shouldn't actively prevent content from loading up.
- Superficial speed: If you want to make your site visually look like it's loading content quickly, ensure your JPEG's are set to "progressive" loading, and your PNG and GIF files should be set to "interlaced".
- Code rendering: Ensure the HTML and CSS validates, in worst case scenarios it could affect page rendering speeds. Additionally, keep CSS selectors succinct and direct to avoid excessive repainting and overwrites.
- Animation issues: While animated effects via CSS3 and JS can be pretty, constant DOM redraws, lagging from heavy transitions and the code bloat can cause performance to suffer. Use at your own risk.
- Script bottlenecks: htaccess, JavaScript, and server-side code can cause issues. Investigate where delays occur using tools like YSlow, Google PageSpeed and Firebug, it could reduce lagging and outages.
- Caching and Offline storage: If content can be cached, it should be. Set expiry headers in htaccess and reference things like jQuery from their site. If users can load data locally, it saves further downloads.
- Geographical delivery: Using a CDN (content delivery network) will reduce the distance data has to travel to a users computer, and it will also spread the load from your sites server. Both will reduce loading times.
- Progressive loading: Sites feel like they load faster if something reaches the user other than a blank screen. Try to

asynchronously load resources like fonts and progressively disclose content post the initial page load.

Real Business Benefits

After reading through the list of performance tips, and trying to work out how much time and data you should aim to save, you may be wondering about the real world benefits of such extreme budgeting techniques.

For clients or site owners, a more agile design places less strain upon the hardware and bandwidth allowances of a host, thus huge financial savings could be made on popular sites. With reduced running costs, an additional "green" sustainability benefit exists as less time will be spent by users loading pages and thus energy waste will be reduced. The improvement in load times may also encourage visitors to use the site more frequently.

For customers, the reduced waiting time will satisfy those on low quality or throttled connections, but for those who suffer bandwidth caps or expensive roaming charges or data rates, it could save visitors a fair amount of money along with electricity savings and perhaps a few less charge cycles of their smartphone battery.

The Sustainable Future

As the possibilities of the Web expand, so has it's waistline. By fighting the trend of page bloating and by taking care only to include what a site really needs, and eliminating the wasteful aspects of your design, you can help yourself, your clients and your customers. With performance remaining high on the user-experience agenda, it will be those sites that reduce the burden upon visitors that gain a competitive edge against similar services.

A Guide to Styling with SVG

SVG has a lot going for it - not only as an image format, but as an important part of any responsive design workflow. It compresses well, it's accessible, it's got decent browser support (supporting graceful fallbacks), it flexes to changes in size without losing clarity, and (importantly for this article) it can be manipulated with CSS and JavaScript.

The application of CSS to SVG images has a lot in common with how you'd style an HTML document, this article will aim to address the differences and similarities that exist. In addition, I'll outline a few best practices and use-cases to make the most of your sites vector graphics.

Declaring and Editing Styles

Before we get started, it's important to note that SVG wasn't designed with a requirement to separate structure and style. Because of this, there are still occasions when CSS must be added to the image using attributes or stylistic elements. Until SVG 2.0 is complete, this is something we have to put up with (like table based layouts in HTML emails). With this in mind, let's examine the four methods used to style SVG images.

Method #1 (Elements) - You probably remember the days of using <B>, <I>, and <U> in HTML for presentational purposes, this practice still exists within SVG - one example being for SVG animation (using the SMIL specification). You can use either CSS animations or JavaScript in it's place, but it's still worth checking out SMIL as it's well supported and more powerful than CSS animation and transitions alone.

```
<animate />
```

Method #2 (Attributes) - Just like HTML4, attributes can be used to declare element styles like width="200px" (or style="property: value;" for compatible CSS). While SVG 2.0 will allow more attributes to be globally controlled using CSS, the specification isn't at recommended

stage (and browser support isn't complete). Because of this, attributes (using case-sensitive name-value pairs) will still be required on some elements.

```
<circle cx="30" cy="50" fill="black" height="120" r="20"
width="100%" />
```

Method #3 (Style tags) - Grouping compatible CSS in <style> tags is supported both within and outside of the SVG file. Outside it's treated the same as regular CSS, though it will only be applied if the SVG image is embedded into the HTML (not if you use IMG elements or CSS background images). Inside the SVG file, you'll need to wrap the style with CDATA declarations to ensure it's parsed correctly (due to SVG being XML).

```
<style type="text/css">
<![CDATA[
    polygon { fill:blue; stroke:purple; stroke-width:1; }
]]>
</style>
```

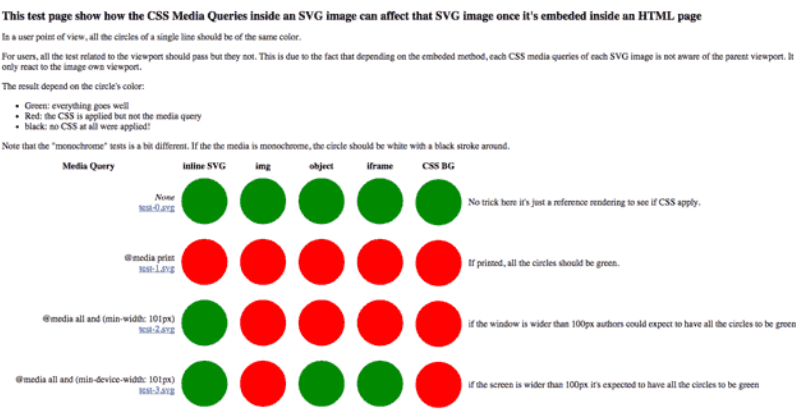
Method #4 (Link tags) - As with <style> tags, you can link to external CSS from outside and inside the SVG. Outside, you use normal <link> tags in the HTML head or add the styles into your existing stylesheet. As SVG has some non-standard styling properties, this will fail the W3C CSS validator (browsers will render the contents fine). Inside, you need to use a special declaration just below the DTD as we're dealing with XML.

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="style.css"?>
```

At-Rules, Selectors and Properties

CSS 2.1 At-rules like @media, @font-face and @import are widely supported in SVG images (like "@media print" friendly graphics). What's really cool is that despite SVG1 predating CSS3, modern browsers (and IE9+) already support embedded media queries. When used in SVG, they use the dimensions of the image (not the page),

which is useful if you want an image to change (rather than scale) when its physical dimensions are altered.



SVG also supports the same general selectors you'd find in CSS 2.1. While it also supports pseudo-classes like :first-child, :visited, :link and :lang; the pseudo classes :hover, :active, and :focus will only work if the SVG is directly embedded within the HTML (not as an <IMG> or background-image). Regarding CSS3 selectors and pseudo classes, only :target (useful in SVG image sprites) has a mention in SVG2 (and browser support is minimal).

Beyond selectors, the W3C provides a really handy index of properties that are supported in SVG 1.1 and 2.0. Many of these are the same as you'd find in CSS 2.1. However, while SVG 1.1 predates CSS3, some SVG properties were later adopted by the CSS3 specification, and are therefore already supported by browsers. SVG was quite ahead of it's time, and because of this, we have more flexibility than we'd otherwise expect.

W3C Recommendation

W3C

Scalable Vector Graphics (SVG) 1.1 (Second Edition)

W3C Recommendation *16 August 2011*

This version:
http://www.w3.org/TR/2011/REC-SVG11-20110816/
Latest version:
http://www.w3.org/TR/SVG11/
Previous version:
http://www.w3.org/TR/2011/PR-SVG11-20110609/
Public comments:
www-svg@w3.org (archive)

Editors:
Erik Dahlström, Opera Software <ed@opera.com>
Patrick Dengler, Microsoft Corporation <patd@microsoft.com>
Anthony Grasso, Canon Inc. <anthony.grasso@cisra.canon.com.au>
Chris Lilley, W3C <chris@w3.org>
Cameron McCormack, Mozilla Corporation <cam@mcc.id.au>
Doug Schepers, W3C <schepers@w3.org>
Jonathan Watt, Mozilla Corporation <jwatt@jwatt.org>

Jon Ferraiolo, ex Adobe Systems <jferrai@us.ibm.com> (Versions 1.0 and 1.1 First Edition; until 10 May 2006)
藤沢 淳 (FUJISAWA Jun), Canon Inc. <fujisawa.jun@canon.co.jp> (Version 1.1 First Edition)
Dean Jackson, ex W3C <dean@w3.org> (Version 1.1 First Edition; until February 2007)

Please refer to the errata for this document, which may include some normative corrections.

This document is also available in these non-normative formats: a single-page version, a zip archive of HTML (without external dependencies), and a PDF. See also translations, noting that the English version of this specification is the only normative version.

SVG 1.1 shares the following property/value pairs with the CSS specification: clip (CSS3), clip-path (CSS3), color-profile (CSS3), cursor, direction, display, font, font-family, font-size, font-size-adjust (CSS3), font-stretch (CSS3), font-style, font-variant and font-weight, isolation (CSS3), letter-spacing, mask (CSS3), opacity (CSS3), overflow, text-decoration, text-rendering, unicode-bidi, visibility, word-spacing and writing-mode (CSS3).

In SVG 2.0 height and width will also be declarable in CSS (rather than as a SVG attribute), as will the SVG exclusive geometry attributes cx, cy, r, rx, ry, x and y.

Some SVG properties (or elements) have names you likely won't recognize, but in actuality serve a similar or identical purpose to an existing CSS entity. I've included a handy list of these properties below, explaining the element or property, and what CSS equivalent it imitates. It can get a little confusing, so hopefully this list will help you align your current CSS knowledge to the convention differences.

- <animate> = animation (CSS3)
- fill, flood-color, stop-color = background-color
- <pattern> = background-repeat
- stroke, stroke-linejoin, stroke-miterlimit = border
- stroke-width = border-width

- stroke-linecap = border-radius (CSS3) for lines
- stroke-dasharray, stroke-dashoffset = border-style
- <feOffset> (child of <filter>) = box-shadow (CSS3)
- <filter> & filter = filter (CSS3)
- kerning = font-kerning
- <linearGradient> = linear-gradient (CSS3)
- <radialGradient> = radial-gradient (CSS3)
- text-anchor = text-align
- glyph-orientation-horizontal and glyph-orientation-vertical = text-orientation
- alignment-baseline, baseline-shift, dominant-baseline = vertical-align
- Element order (first to last) = z-index (as there is no box model like with CSS)

The following have no CSS equivalent, so you should acquaint yourself with their usefulness by reading the specification: clip-rule, color-interpolation, color-rendering, color-interpolation-filters, enable-background (filter), fill-rule, lighting-color (filter).

Notable others with no CSS equivalent, but remain incredibly useful include:

- image-rendering and shape-rendering (like text-rendering but for the SVG image data).
- fill-opacity, flood-opacity, stop-opacity and stroke-opacity (control alpha transparency on individual parts of a shape and it's color).
- marker (like marker-end, marker-mid and marker-start) - for arrowheads in diagrams.
- pointer-events (control interaction effects like drag and drop)

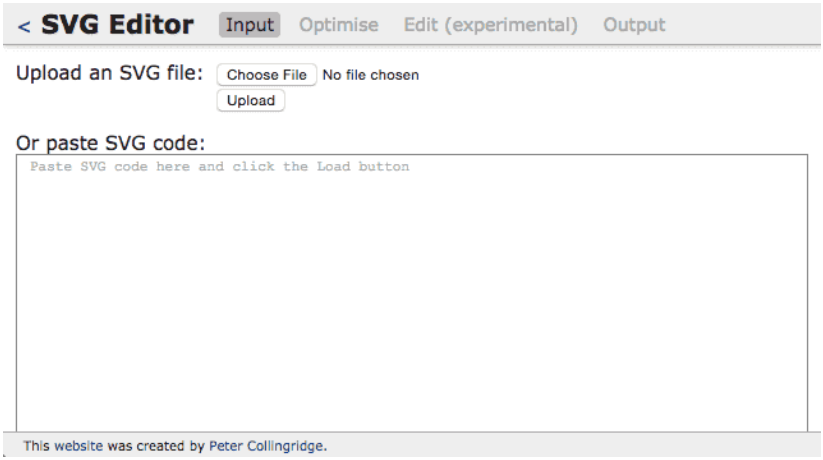
It's also worth noting that when SVG2 is finalized, SVG will be all caught up with the CSS3 color module. Until then, SVG supports hex,

RGBA, the x11 “named colors” list, and the useful currentColor value which inherits the value of the CSS color property within the SVG fill property (if the SVG is embedded in your HTML), thereby eliminating the need to duplicate color values in CSS (like the deprecated CSS2 system color palette).

Best Practices for Maintainable Code

Check your SVG files for redundancy, just like with HTML and CSS there are probably improvements to be made. Like other Web languages you can lint the file by removing whitespace and comments. Ensure that caching and gzip are enabled on your server for further compression based savings. Furthermore, run your SVG images and sprites through the W3C validator, it supports SVG 1.1 and will help identify common errors.

More space savings can be made by trying to separate style and structure as much as is possible. With SVG2 still in development, you'll be restricted as to how much can be pushed into stylesheets, but it's worth doing what you can and revisiting the files later when it is supported.



I would also avoid linking to external stylesheets from within the SVG file as it could block the render path while it waits for the file to load

(the same reason most designers avoid @import in CSS). Better options include grouping CSS within <style> tags in the SVG file (using sprites when browser support is solid) or linking to the stylesheet from within your HTML (if the SVG image is embedded).

Finally, a note on compatibility. SVG's do have some issues between browsers which can be easier to spot if you work in co-ordination with a site like Can I Use. If you're thinking of using SVG animations, polyfills like Snap.svg and Raphael may be useful. Whatever you do though, make sure your SVG images have a fallback, after all Internet Explorer 8 will probably still be around for a little while longer (even with Spartan around the corner).

Sources:

- https://www.w3.org/TR/SMIL/
- https://css-tricks.com/guide-svg-animations-smil/
- http://jeremie.patonnier.net/experiences/svg/media-queries/test.html
- https://www.w3.org/TR/SVG/propidx.html
- https://www.w3.org/TR/SVG2/propidx.html
- http://petercollingridge.appspot.com/svg-editor

Another 6+ Web Files to Help Improve Your Website

I've previously examined web assets that while small in size can improve your sites performance or user-experience. In this concluding part to the trilogy, we'll once again examine some files that don't take long to create, but can positively impact your visitors.

Quick Overview

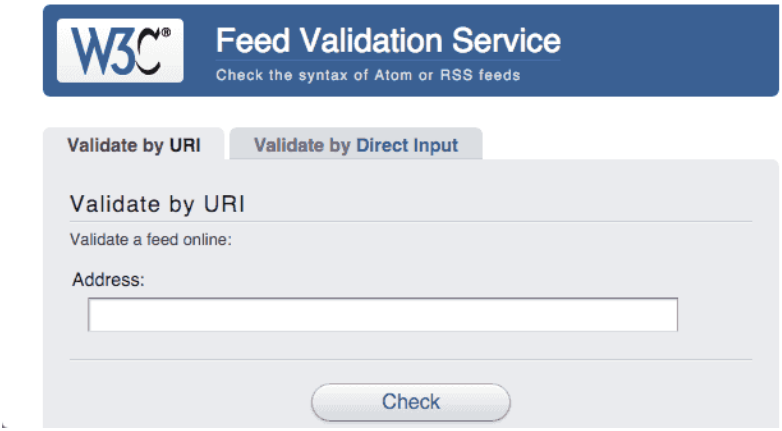
Here is what we shall cover:

- RSS.xml, Atom.xml and Feed.opml
- iCal.ics
- Subtitles.vtt
- Manifest.json and Browserconfig.xml
- .htaccess
- Crossdomain.xml

Syndication XML Feeds

Everyone likes to stay up-to-date, whether it be with their favorite blog, podcast or the news. Syndication feeds allow individuals to get push notifications of updated content as it happens in the same way that new tweets or instant messages are received.

All web browsers support at least one of the two syndication formats; RSS and Atom (though both use XML as their publishing language). They use very similar syntax and while neither is 100% dominant, RSS tends to be the most used web-wide due to it's ubiquitous auto-generation in CMS systems like Wordpress.



You can even group multiple syndication feeds into a bundle using a specially formatted XML based outline language called OPML. Which is great to introduce newbies to a network of sites you operate.

Creating the RSS.xml and Atom.xml files

Because RSS and Atom have very similar syntax, we are going to create some basic syndication files in both formats at the same time. So to begin, open a couple of blank documents in your favorite IDE or text editor and save one as RSS.xml and the other as Atom.xml

Below is an example of a basic RSS container:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/terms"
xmlns:atom="http://www.w3.org/2005/Atom">
    <channel>
        <title>Six Revisions Feed</title>
        <link>http://sixrevisions.com/</link>
        <description>Web Design Tutorials, News and Articles</
        description>
        <copyright>Some Rights Reserved</copyright>
        <dc:creator>Jacob Gube</dc:creator>
        <dc:date>2015-10-24T00:00:00Z</dc:date>
```

```
        <atom:link rel="self" href="http://sixrevisions.com/feed/
        rss.xml" type="application/rss+xml" />

    </channel>
</rss>
```

Below is an example of a basic Atom container:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title>Six Revisions Feed</title>
    <id>http://sixrevisions.com/</id>
    <subtitle>Web Design Tutorials, News and Articles</subtitle>
    <rights>Some Rights Reserved</rights>
    <author><name>Jacob Gube</name></author>
    <updated>2015-10-24T00:00:00Z</updated>
    <link rel="self" href="http://sixrevisions.com/feed/atom.xml"
    type="application/atom+xml" />

</feed>
```

In order to keep things simple, I've used the Dublin Core within RSS to give it equal footing with Atom in terms of descriptive elements. The above (of course) is a very basic example and theres a LOT more that you can add but if you just want to get up and running, it's a solid boilerplate. There is space for The feeds title, a URL and description of your site, copyright information, site author details, and when the feed was last updated.

Once you've filled in the general information, we now need to add entries for both feeds. The newest item should be at the top of the list, placed directly after the self URL reference. You can repeat the below entry and item tags as needed to build up your feeds.

An example of an RSS item:

```
<item>
    <title>RSS Secrets</title>
    <dc:creator>Jacob Gube</dc:creator>
    <dc:date>2015-10-24T00:00:00Z</dc:date>
    <guid>http://sixrevisions.com/?p=0000</guid>
    <category>Development</category>
    <link>http://sixrevisions.com/dev/article-000.html</link>
    <description>Everything you wanted to know about RSS but
    were afraid to ask.</description>
</item>
```

An example of an Atom entry:

```
<entry>
    <title>Atom Secrets</title>
    <author><name>Jacob Gube</name></author>
    <updated>2015-10-24T00:00:00Z</updated>
    <id>http://sixrevisions.com/?p=0000</id>
    <category term="Development"/>
    <link rel="alternate" href="http://sixrevisions.com/dev/
    article-000.html" />
    <content type="xhtml" xml:lang="en"><div xmlns="http://
    www.w3.org/1999/xhtml"><p>Everything you wanted to know
    about Atom but were afraid to ask.</p></div></content>
</entry>
```

There are tools which can build RSS and Atom files if you want a more automated or user friendly route to syndication, though some of these cost money or rely on third parties:

- Feedburner - http://feedburner.com/
- RSS Generator - http://rssgen.com/
- Feed Validator - http://feedvalidator.org/
- Feeder - http://reinventedsoftware.com/feeder/
- RSS Builder - http://www.softpedia.com/get/Internet/News-Newsgroups-Blog-Tools/RSS-Builder.shtml

Once you've got your completed RSS and Atom files, you just need to link to them from the head of your HTML documents:

```
<link rel="alternate" type="application/rss+xml" title="RSS Feed"
href="cache/rss.xml">
<link rel="alternate" type="application/atom+xml" title="Atom Feed"
href="cache/atom.xml">
```

Creating a Feed.opml file

If you own multiple websites, you probably have thought about the possibility of building multiple RSS and Atom feeds at this point. With this in mind I'd like to introduce you to the OPML format. OPML is built upon XML and is intended for outlining lists of data, but it's especially popular for aggregating lists of feeds, which is why I'm including it here.

Below is an example of what the code should include:

```
<?xml version="1.0" encoding="utf-8"?>
<opml version="2.0">
    <head>
        <title>Six Revisions Network</title>
        <docs>http://dev.opml.org/spec2.html</docs>
    </head>
    <body>
        <outline type="rss" text="Six Revisions" xmlUrl="http://
        feeds.feedburner.com/sixrevisions" htmlUrl="http://
        sixrevisions.com/" title="Six Revisions" />
        <outline type="rss" text="Design Instruct" xmlUrl="http://
        feeds.feedburner.com/designinstruct" htmlUrl="http://
        designinstruct.com/" title="Design Instruct" />
    </body>
</opml>
```

As with all the files, you can build it using any text editor; though with this you can give it any filename, put it in any location (preferably alongside your feeds), and as with all XML files it contains a DTD declaring it as such. Just like HTML it contains head and body elements. Below the DTD you'll notice an OPML tag with a version number stating the version of the specification it conforms to (this URL is referenced in the docs tag in the head of the document).

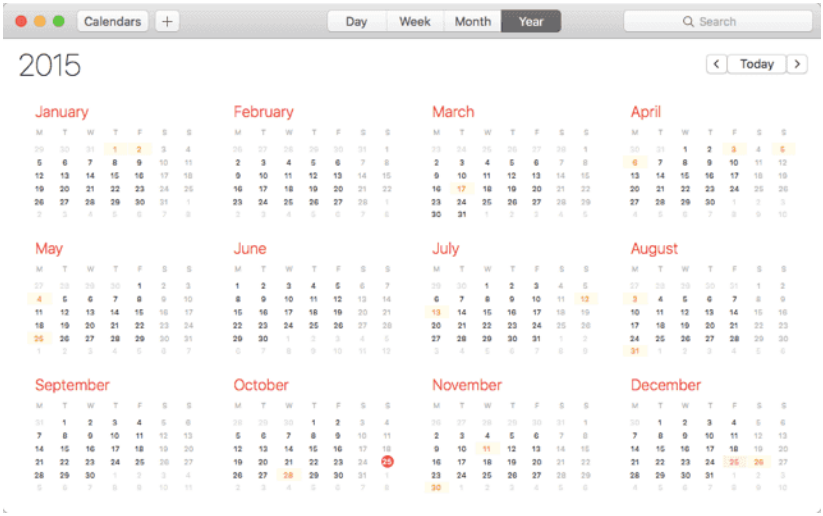
Within the head you have space to include a title for your batch of feeds, and within the body you have outline elements which represent each feed being linked too. Because we're referencing RSS feeds, there is an XML url and a HTML url for maximum compatibility. If you're offering other types of content, only a URL attribute may be necessary (instead of htmlUrl).

As with RSS and Atom, add a link to the head of your document:

```
<link rel="outline" type="text/x-opml" title="Network Feed"
href="http://sixrevisions.com/feed.opml">
```

iCal.ics

In a previous article I mentioned the benefits of vCard.vcf, the little file which allows you to markup and produce business cards that work in a variety of environments. The great thing is that there's also another little file that does the same thing except for events. This file (iCal.ics) allows you to set appointments and reminders via your default calendar app.



iCal.ics comprises meta-data about events, dates, times; and can include descriptions and comments which can be imported into calendars on mobile and desktop platforms (which can be in-turn read by personal assistants like Siri). While not supported in Web browsers; when downloaded - calendar apps, email apps, and cellphones can take advantage of the event information.

iCal also has it's own microformat (just alike vCard) which can be used to semantically enhance pages, and with a browser extension, this data can be either exported, used by web-apps, or imported into third party services.

Creating a iCal.ics file

The first thing you need todo is create the iCal.ics file within your favorite plain text editor (the filename is case insensitive). Inside there's a few things every iCalendar must have, according to the specifications:

- BEGIN:VCALENDAR and END:VCALENDAR to map the start and end of the document (the same way we open and close HTML files).
- VERSION: with a value of 2.0 (the latest edition).
- BEGIN:VEVENT and END:VEVENT to map the start and end of the event that the calendar is going to make a note of.

Note: You can also add Journal and todo entries to your calendar using BEGIN:VJOURNAL and BEGIN:VTODO.

Here is an example of a working iCal event:

```
BEGIN:VCALENDAR
VERSION:2.0
BEGIN:VEVENT
DTSTART:20151024T190000Z
DTEND:20151024T190000Z
SUMMARY:Six Revisions Meetup
END:VEVENT
END:VCALENDAR
```

The general syntax for iCal files is the same as for vCard files. Directives and element references are all in uppercase followed by a colon character.

DSTART and DTEND give the start and end dates of events. End dates are optional (for ongoing events), times are optional too, and you can make events repeat using the RRULE property.

It's also worth giving your event a description using the summary property as apps will use this to alert users when the event is going to take place.

Below are some of the additional directives you can use:

```
LOCATION: Six Revisions HQ
URL: http://www.sixrevisions.com/
ORGANIZER:MAILTO:jacob@sixrevisions.com
```

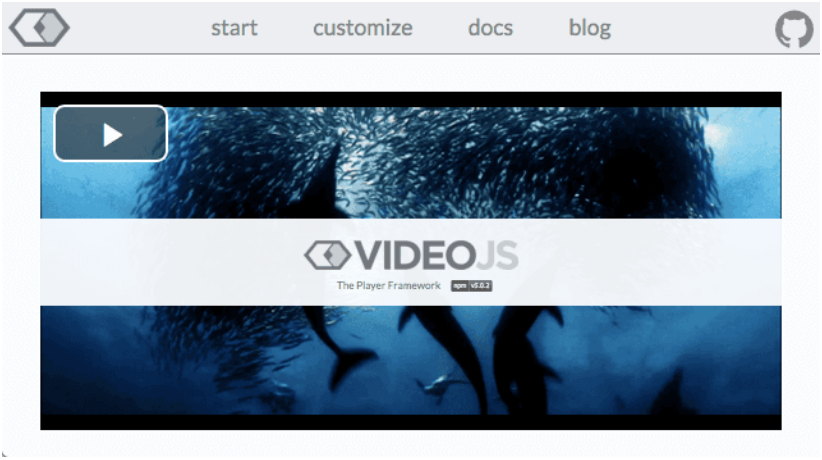
While iCal does have many additional directives and features, the number of them in use or compatible with third party software is limited (unlike vCard). Therefore I've limited the above to just the most important ones that are frequently used.

Once you've finished your file, save it and add the below reference to the <head> (and add a regular link to the .ics in the body) of your html documents:

```
<link rel="alternate" type="text/calendar" title"iCal" href="iCal.ics">
```

Subtitles.vtt

Accessibility is important. We all know this, but sometimes we find it difficult to ensure our work is visible to the widest possible audience. How many of us consider offering fallbacks, text transcripts, subtitles, or audio descriptions for those with visual impairments?



Building a subtitles.vtt file is extremely easy. It might take a little bit of effort to get the timings correct (as you have to sync the text to the

speech manually), however, at the end, you'll have a more accessible website. It also has pretty good browser support.

Subtitle files can also be generated in multiple languages and offered alongside your videos to provide international visitors with a better experience. The information within these files can also be read with Javascript and turned into human readable transcripts on the website.

Creating a Subtitles.vtt file

You can create a subtitle file using any text editor and call each track what you'd like, however if you intend on offering multiple videos and languages, it might be worth labelling them in a format such as 01-EN.vtt (using two digit language codes) to avoid mix-ups.

If you don't want to build the files manually, there are some nifty tools that can help with the process:

- Microsoft Caption Maker - https://dev.modern.ie/testdrive/demos/captionmaker/
- VTT Captions - http://www.vttcaptions.com/
- Subtitle Edit - http://www.nikse.dk/SubtitleEdit
- Subtitle Workshop - http://www.uruworks.net/index.html
- AegiSub - http://www.aegisub.org/
- Jubler - http://jubler.org/

Some of these tools will not output subtitles in a compatible VTT format. If this is the case (they may just support SRT), you can use this great converter or if you prefer command line only use this instead.

If you do want to build the file by hand instead, it's really straight forward. Below is the sort of code you might find in a subtitle file:

```
WEBVTT

1
00:00:01.000 --> 00:00:10.000
This is the first ten seconds of the video.
```

```
2
00:00:10.100 --> 00:00:14.900
Milliseconds count too!

3
00:00:15.000 --> 00:00:27.000
You can break <00:00:18.000> section down into <00:00:21.000>
parts for occasions <00:00:24.000> like karaoke songs!

4
00:00:28.000 --> 00:00:30.000 A:end
You can <b>style</b> text too!

5
00:00:31.000 --> 00:00:38.000 D:vertical
<00:00:31.000><v tom>You can also give the speaker names too.</
v>
<00:00:35.000><v rebecca>so they can have a conversation.</v>
```

Each WebVTT file is comprised of a declaration that states it's a WebVTT file, followed by equally spaced cues that break down the segments of the video. You can use cues as chapter markers by compiling a VTT file of the most important ones and using kind="chapters" in the below track element code example.

Each element of the subtitle is comprised of a timeframe; made up of the hour, minutes, seconds and milliseconds from and to the event being spoken. The format is HH:MM:SS.TTT —-> HH:MM:SS.TTT.

You can also change how your subtitles appear visually. More options are available but I'll cover the coolest below:

- Make the subtitle box appear vertical on the left by using D:vertical or on the right D:vertical-lr (place the code after the timescale).
- Align the text left (A:start), center (A:middle), or right (A:end) within the subtitle box (place the code after the timescale).
- You can also use <B>, <I>, and <U> within the VTT file to draw attention to important subtitled words.
- Even better, you can even use <c.myClass>text</c> to allow CSS styling within VTT (browser support is limited).

When you have conversations with multiple people, it helps to keep track by assigning identities. In the example above (Chapter 5) you'll notice I've used <v tom> and <v rebecca> as an example of how you can really give your subtitles context.

Once you've built your VTT file and are ready to add it into your video, it should be a matter of linking it using the HTML5 track element (just remember to test that it all works and the timings are fine). Below I've included a full example:

```
<video width="640" height="480" controls preload="metadata">
    <source src="media/01.mp4" type="video/mp4">
    <track src="media/01-EN.vtt" kind="subtitles" srclang="en"
    label="English" default>
</video>
```

Manifest.json & Browserconfig.xml

Who doesn't love a favicon? These 16x16 graphics have taken the web by storm and have become one of the smallest but most identifiable aspects of a sites identity. We've gone from having icons so small that you can barely see them to graphics large enough to fit as a speed-dial or television set icon - and a wide range of use-cases in between.

With this in mind, Google's Android and Microsoft's Windows Phone have separated from Apple's iOS in their approach to favicon

presentation. It used to simple in the past: you had a favicon.ico for Internet Explorer and an apple-touch-icon.png file for iOS (and compatible mobile) devices at various sizes.

In a previous article I covered the apple-touch-icon so now it's time to cover Android's Manifest.json file and Microsoft's Browserconfig.xml file. Each has it's own method of showcasing home screen icons, each has it's own size requirements, each has it's own syntax rules.

For both of these formats you'll need images, I recommend PNG format and optimizing them using tools like ImageOptim to get the file sizes as small as possible. For the file dimensions needed, check the code samples for "sizes" (manifest.json) and the numbers in the square and wide (browserconfig.xml) tags.

Creating a Manifest.json file

Just like many of the small files in this list, you can create the file using any text editor, but you must keep the file name as-is (manifest.json) and the file should be kept in the base directory as that's where devices tend to request it by default.

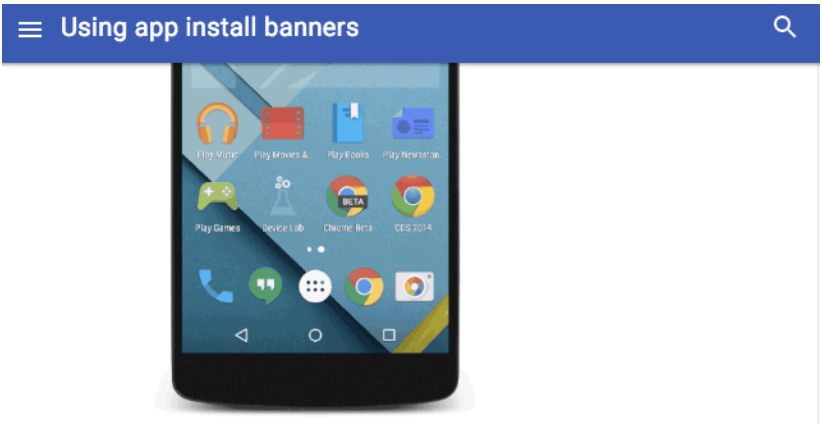
Below is an example of what the code should include:

```
{
    "short_name": "Six Revisions",
    "name": "Six Revisions - Web Design Tutorials, News and
    Articles",
    {
        "src": "/android-36.png",
        "sizes": "36x36",
        "type": "image/png",
        "density": "0.75"
    },
    {
        "src": "/android-48.png",
        "sizes": "48x48",
        "type": "image/png",
        "density": "1.0"
```

```
        },
        {
            "src": "/android-72.png",
            "sizes": "72x72",
            "type": "image/png",
            "density": "1.5"
        },
        {
            "src": "/android-96.png",
            "sizes": "96x96",
            "type": "image/png",
            "density": "2.0"
        },
        {
            "src": "/android-144.png",
            "sizes": "144x144",
            "type": "image/png",
            "density": "3.0"
        },
        {
            "src": "/android-192.png",
            "sizes": "192x192",
            "type": "image/png",
            "density": "4.0"
        }
    }
```

The file contains a short title which is what the app will be called on the home screen (as opposed to what is in the title bar of the browser window or the name).

Each property contains sizing information (the pixel size and density level), the file MIME type, and of course a link to the image which you can change to match the location on your server.

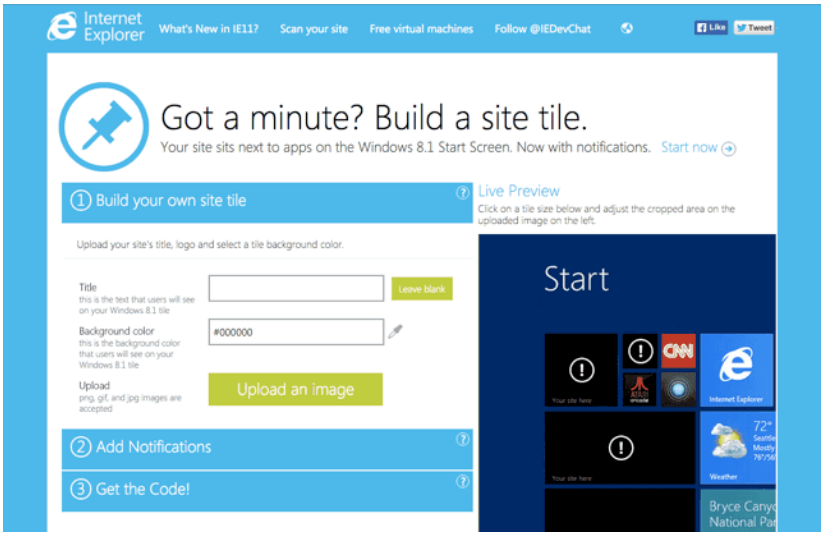


Once you've completed your file, you just need to add a link within the <head> of your html. Below is an example you can use:

```
<link rel="manifest" href="manifest.json">
```

Creating a Browserconfig.xml file

You can create this file using any text editor however the filename must match browserconfig.xml and it must be placed in the base directory of your website (as devices will explicitly make http requests for it).



Below is an example of what the code should include:

```
<?xml version="1.0" encoding="utf-8"?>
<browserconfig>
    <msapplication>
        <tile>
            <square70x70logo src="http://www.sixrevisions.com/
            images/70s.png"/>
            <square150x150logo src="http://www.sixrevisions.com/
            images/150s.png"/>
            <square310x310logo src="http://www.sixrevisions.com/
            images/310s.png"/>
            <wide310x150logo src="http://www.sixrevisions.com/
            images/tile-wide.png"/>
            <TileColor>#FFFFFF</TileColor>
        </tile>
    </msapplication>
</browserconfig>
```

The file contains an XML DTD with a browserconfig declaration tag and a tag stating that the the home-screen icon will be a web-app (it'll run in a browser shell on the phone).

Inside the tile tag you'll see four icons each of which conform to the requirements of the Windows phone resolution for static and live tiles and a hex color code for background transparency.

Now, just like with the manifest, add a link to it within the head of your HTML:

```
<meta name="msapplication-config"
content="browserconfig.xml">
```

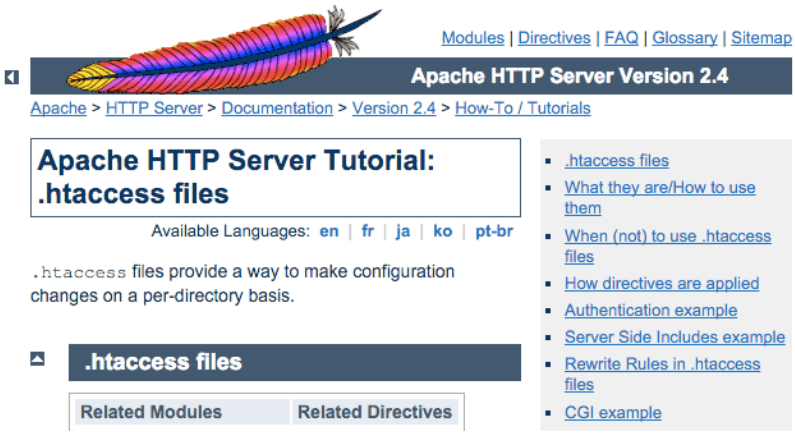
If you also want to offer legacy support for older versions of IE mobile, you can add a non-live tile which doesn't use the browserconfig.xml file by referencing the below in the head of your HTML:

```
<meta name="msapplication-TileColor" content="#FFFFFF">
<meta name="msapplication-TileImage" content="images/
tile-144.png">
```

.htaccess

One of the most complicated small files out there has always been the server configuration file. If you make a mistake, your site goes down. If you really make a mistake, your server goes down. Therefore, people have been timid to tweak it in the past.

The most common server software is Apache, nGenx, and IIS; though Mac OS X server, Node, and others have fans. While Apache is the one I'll discuss as it's the server I primarily use, I'll also include documentation for a few others in the notes.



The server configuration file is where many of the really substantial benefits to a site can be made. Custom error pages, GZIP compression, caching, url rewrites, general performance tweaks, and more can be configured and maintained. Being able to manage these settings is a very useful skill for any developer.

It's worth noting that the more you pack into the htaccess file, the more impact it will have on your servers processor - so try to keep it as efficient as possible. If you do have root access to your server, you can place repeat rules within the httpd.conf file (instead) to improve performance further.

Creating a .htaccess file

If you're going to tinker with your servers configuration files, you should first check your hosts documentation to review any special requirements or details that may affect their location, implementation or usage.

If you're going to be making the changes to httpd.conf, you won't need to create any files as the file should already exist on your server. Just track it down and open it up (though be sure to make a backup in case you make a mistake).

If you're working with .htaccess, you can use any text editor, create a new empty file and save it with the .htaccess extension (this means

there is no filename and may show on your OS as a hidden file). When you upload the file it will usually go in the base directory of your domain.

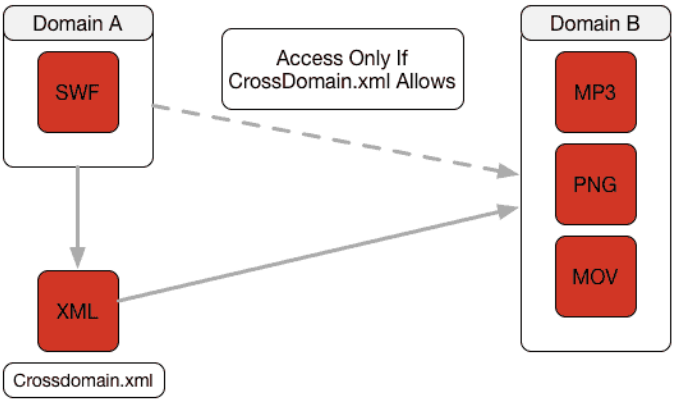
Once you're ready to begin crafting your configuration files contents, the best advice I can offer is to use the Apache server configs from the HTML5 boilerplate as a template. It's always kept current, has lots of useful material to get started with and you can trim away the bits you don't require.

Boilerplates & Docs:

- Apache Server Configs - https://github.com/h5bp/server-configs-apache/
- Apache Documentation - https://httpd.apache.org/docs/
- IIS Server Configs - https://github.com/h5bp/server-configs-iis/
- IIS Documentation - http://www.iis.net/configreference/
- nGenx Server Configs - https://github.com/h5bp/server-configs-nginx/
- nGenx Documentation - http://nginx.org/en/docs/
- Node.js Server Configs - https://github.com/h5bp/server-configs-node/
- Node.js Documentation - https://nodejs.org/en/docs/

Crossdomain.xml

With the introduction of AJAX in the Web 2.0 movement, a central component of the process was CORS (Cross-Origin-Resource-Sharing). This allowed websites to request data (and permission to use that resource) from other sites. While fairly straight forward and needing little more than an HTTP header for most media types, Adobe Flash - the archaic web format requires something a bit special.



Adding a crossdomain.xml file to your website allows your flash file to allow CORS requests to and from external sites. If all your media is embedded directly within your SWF files then this file isn't necessary. However, if your file downloads or streams any resource hosted on a third party server (like an image or video), you'll need this important file to give it permission to proceed (otherwise external files won't load).

Creating a Crossdomain.xml file

You can create a this file using any text editor. However, it must be placed in the root directory of your website and the filename must be "crossdomain.xml".

Below is an example of what the code should include:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="www.sixrevisions.com" />
    <site-control permitted-cross-domain-policies="master-only" />
</cross-domain-policy>
```

Every file begins with the DTD and a tag which declares it as a cross domain policy file.

Inside this, you can edit allow-access-from elements in a number of ways:

- Edit the domain attribute to match the URL (or IP address) where the resource(s) are located.
- Include additional allow-access-from elements if you want to use resources from multiple domain names.
- Use the secure="false" or secure="true" attribute to allow switching between HTTP and HTTPS servers.

The site-control tag sets permissions for cross-domain requests - by default it's set to master-only. If you don't have flash files or don't want flash talking to third party sites, set it to none (the most restrictive policy).

Sources:

- https://validator.w3.org/feed/
- http://cyber.law.harvard.edu/rss/rss.html
- https://tools.ietf.org/html/rfc4287
- http://dublincore.org/metadata-basics/
- http://dev.opml.org/spec2.html
- http://www.opmlicons.com/
- http://microformats.org/wiki/hcalendar
- https://en.wikipedia.org/wiki/ICalendar
- http://www.grokkingandroid.com/recurrence-rule-and-duration-formats/
- http://videojs.com/
- http://www.w3.org/TR/webvtt1/
- https://atelier.u-sub.net/srt2vtt/
- https://github.com/mafintosh/srt-to-vtt
- https://w3c.github.io/webvtt/
- https://developer.mozilla.org/en/docs/Web/API/Web_Video_Text_Tracks_Format

- https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/IconMatrix.html
- http://www.w3.org/TR/appmanifest/
- https://developers.google.com/web/fundamentals/engage-and-retain/simplified-app-installs/?hl=en
- https://msdn.microsoft.com/library/dn455106.aspx
- http://www.buildmypinnedsite.com/en
- https://msdn.microsoft.com/en-us/library/dn320426(v=vs.85).aspx
- http://httpd.apache.org/docs/current/howto/htaccess.html
- https://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html
- http://www.adobe.com/devnet/adobe-media-server/articles/cross-domain-xml-for-streaming.html

Six Revisions

45 Articles for Web Developers

Alexander Dawson
(Author)

Through my six years as Six Revisions most prolific contributing author, I wrote 45 articles on a range of subjects including Web Design, UX Design and Full-Stack Web Development. Now that the Six Revisions website is defunct, I've taken the time to carefully package all of my articles into a book.

While some of the best practices and code may be dated, holding only nostalgic value; a lot of the information is surpassingly still relevant. It's for this reason I decided to restore my collective work.

There's tonnes of code, plus hundreds of tips, links and topics for everyone - from beginner to expert alike. There's a few exclusive (unseen) articles included in this book too. I hope you enjoy this compilation.

"Excellent post Alexander!"
James Royal-Lawson (UX Podcast)

"Love your illustrations!"
Amber Weinberg

"Great post for educating people in the industry."
Dainis Graveris (1st Web Designer)

"Fantastic article Alex! You must have put some work into it!! Well done."
Sheena Oosten

"I really enjoy your articles Alex. They are never irrelevant. Thanks for another great read."
Young J Yoon

"Alex, you're really contributing a huge amount of very useful articles for SR This is terrific."
Michael Tuck

"A picture says a thousand words (but the words are great too!)."
Carolyn King

Visit: https://alexanderdawson.com