9 Ways to Misunderstand Web Standards

If you're a webmaster, you should definitely read this!

Misunderstanding #1: "We Need Separate Print Pages"

We've all seen this – a separate print page, linked to from a crowded, table-layoutish HTML page, aiming to serve no other need than being printed out (it fails, because bloggers link to print pages – they're mostly easier to read and not split up into multiple pages). The good thing about these pages is that the user gets an instant impression of what the print-out will look like. Of course, the right way to do this would be to serve a separate stylesheet for medium print, and if the browser does it right, it will show the visitor a print preview.

This is old news, but why do I consider it noteworthy? Because it's the #1 application where media-dependent CSS, on top of media-independent HTML, ought to come into play... and yet, and I'm guessing, only 5% of all pages make use of it. You'd think after years of evangelizing done by web developers, the likes of CNN or Wired would have gotten the point.

Misunderstanding #2: "We Need an Alternative Mobile Web on Top of the Existing Desktop Web"

WML, the WAP's Wireless Markup Language, is dead today for a good reason: there was no need for this species in the course of web evolution. Even before WML was invented, plain vanilla HTML was created to be media-independent. Not by accident, but by design, because platform-dependence was one of the main problems Tim Berners-Lee tried to solve when inventing the World Wide Web and HTML.

But the fact that the same document can be used in a variety of circumstances, and that the browser simply requests the stylesheet needed – or uses its own, by default – is not an intuitive one to grasp. Most real world objects don't behave that way, changing their shape and behavior depending on what is needed. (A German joke goes like this: "What is the smartest invention of the 20th century? The thermos flask, as it knows it needs to keep things warm in Winter, and cold in Summer.")

So today, you'll see many companies creating an alternative mobile web. Well, those who can afford it, anyway, like Google (which is releasing products like Search or Gmail in special mobile versions, while often ignoring accessibility guidelines on their "main" sites).

Now, you might argue, it's not so much a technical matter of how to serve the mobile web – it's a matter of finding the right content fitting small devices. However, we simply don't know in which contexts a visitor might want to read a longer text. At my last job I used to drive in the train and bus for about an hour every morning, and during that time, I used my cell phone's browser to read longer articles. When I was at work on a bigger screen, I had much less time to read articles. (People watch movies on their PSP... it's really not all about screen-size.)

What was the original solution to formatting the same document for both the desktop screen, and the mobile device? Media-dependent stylesheets, again, delivered on top of the same HTML. And yet, if you look at different mobile browsers available today, you'll see that they might do any of the following:

1. ignore all your stylesheets, and do something more or less reasonable with the HTML (e.g. breaking up table layouts)
2. use the screen stylesheet, but adjust it here and there to fit the smaller resolution
3. use the handheld stylesheet, but only if you deliver your XHTML using the "XHTML Mobile 1.0" doctype (what is the need for yet another doctype, when plain XHTML claims to serve mobile media already?)
4. use the mobile stylesheet, if available, and use its own default formatting if not

Can we really, truly blame the mobile browser makers for resorting to 2 and 3, which are definitely breaking web standards by choosing the screen stylesheet? After all, any real browser that wants to find a commercial niche by serving user needs must work on the real web and what's available on it. If 95% of all web pages ignore the honorable goals of media-neutrality, mobile stylesheets and so on, why build a browser with focus on the 5%? Conversely, in true chicken and egg fashion, many mobile developers might steer clear from mobile standards done right... 'cause done right, they don't work.

Misunderstanding #3: "Accessibility Means to Always Use Alt Text"

The meaning of alt text is to serve as replacement (hence, "alternative") when the image the HTML intends to serve cannot be seen. There's a variety of reasons for that to happen; the user is blind, the medium doesn't support images, the user disabled images on purpose so she can browse on low-bandwidth, or the Googlebot comes around to visit.

For example, **when your image is a mere illustration to a point you're repeating elsewhere in the text, there's no need at all to use the alt text**. Because the image is not crucial to understanding the point. Most people however at this point heard some rumors that the "alt tag" is increasing accessibility; and possibly, they've heard somewhere else that accessibility is professional these days. And they also might want to see a tool-tip, not knowing that the title attribute would come in handy for that. (Wordpress in typical installations even creates an alt text based on the image's file name; tools like Frontpage have committed similar silliness in the past.)

So, let's say our blog post is titled "New AJAX Flavor Discovered" and the illustration following the headline is a shrink-wrap box with the shiny colorful letters "AJAX," then repeating the letters AJAX in the alt text will result in something like this within alternative browsing contexts:

New AJAX Flavor Discovered
AJAX

Is this useful? No – the point of the illustration was to use a recognizable catchy visual, or to suggest that AJAX is hyped as a shrink-wrapped shiny product; the point was not to clarify (in text) that the article is about AJAX, as the headline already did a good job at that.

In XHTML2, by the way, some of us may be able to get rid of the alt attribute altogether. Why? In XHTML2 you can use the "src" attribute on anything, including e.g. a paragraph. It's a bit like longdesc (today's link to a longer image description) done right.

Misunderstanding #4: "Sites Always Become More Accessible With CSS"

If I'd have to make a bet, I'd say creating an **inaccessible** website with CSS is 10 times as fast as doing the same with table layout. You know, you just need to use an incredibly tiny font in your CSS and you ruined the reading experience for your visitors on thousands of pages (unless they're the 5% of visitors who know which button to press to increase the font size again, of course). Now try that with the nasty font tag... even if you have a great template system, it'll take much longer!

CSS on its own doesn't do much in terms of pushing accessibility. And table layout on its own also doesn't do much in terms of preventing it. Sure, it's better to have full control over the linear flow of the HTML (e.g. to put a long navigation to the left side via CSS, but make it follow the main content in the HTML for search engines, smaller devices, text-to-speech contexts and so on). But CSS is ten times as fast... for better, or worse.

The biggest advantage of CSS is simply that it helps developers create websites much faster by separating layout and content. As far as that's possible, and it's certainly not possible 100% – which brings us to our next point.

Misunderstanding #5: "With CSS We're Completely Separating Content and Layout"

I've completely switched to CSS in around 1998/1999. I'm still happy I was able see "old" HTML (know your enemy!). Back then, when I saw the proverbial light, I figured that CSS enables you to completely separate content from layout. Put the h1 here, the address element there (is anyone still using that?), use <em> instead of <i> and <strong> instead of <b> (and grok the philosophy behind that), spice it up with a div & span & list, and basically markup everything based on its never-changing semantic/ structural role, and you're done.

Boy, was I wrong!

For starters, different browser quirks often need different HTML element wrappers. More importantly, and that one's true even in browser utopia (the place without quirks), once you realize you want to redesign a part of your site you also realize how many **classes are missing**. Next to creating a class in advance for everything imaginable (see the CSS Zen Garden), you're basically back to HTML.

CSS doesn't completely separate content from its layout – only a template system can do that. CSS has its place on top of the template system, and greatly simplifies the HTML (if you think there's div soup today, you've got to see true 1996 table soup... it's worse.)

Misunderstanding #6: "With CSS, You Can Do the Design Later"

Actually, it's true. You can try to do your HTML structure first and then create the CSS as afterthought. You'll probably have a couple if iterations, going back and forth between the two, but you'll learn and adapt your processes in the future

The real problem I'm seeing here, however, is that some people – heck, I was often one of them – tend to forget to add a fine-tuned design at all, happily using valid HTML + CSS as excuse for not having a page layout (relying on user stylesheets alone is *not* an option). Have you ever come across one of those black-on-white sites of nothingness, where the only graphical element is a "valid HTML" badge? (And no, I'm not talking about well-done minimalism a la Joe Clark's blog.)

Misunderstanding #7: "The Web's Becoming More and More Accessible Every Day"

I think the biggest factor making accessible web sites a reality today is that most people simply don't do HTML anymore – they use **ready-made blog templates**. And several blog templates deliver good, strict (X)HTML.

However, there's another movement throwing needles at the high-flying balloon of accessibility, and partly, usability; it's *The Return of JavaScript (Episode III: A New AJAX)*.
Now the rule of thumb should be this; put JavaScript on top of HTML to allow for features you otherwise wouldn't have. Like nifty forms with auto-completion and such... but importantly, *auto-completion where the rest of the application still works as expected*, including all kinds of keyboard shortcuts. Yet, we're seeing the focus turn away from this philosophy; some apps even put the "AJAX" right into their product name.

Google's **Gmail**, for instance, is a guilty pleasure, usability-wise. While you have a lot of features normal HTML could never offer (unless we're talking about some advanced, it's-the-year-2014 XForms browsers), you're also often losing out on simple things like the back button. Or out-of-the box mobile support. Or copy-and-pasting of whatever you see on-screen. (The list goes on.) Now, the Gmail team was wise enough to give us a second, plain-HTML version of their tool. But the fact remains that **many web applications of today destroy a dozen features by introducing a dozen new ones.** Whether or not that's simply the wave of the future and all good remains to be seen, but it's not accessibility in the true sense.

Misunderstanding #8: "The Semantic Web is Just Around the Corner"

Today, the W3C is putting much of its efforts behind the Semantic Web (also described by Tim Berners-Lee in the second part of his book *Weaving the Web*). But Cory Doctorow put it so well back in 2001: *people lie, and people are lazy*. And I got a feeling even the W3C misunderstands why some of their web standards took off (they are relatively simple to apply, and certain complexities can be ignored without dangers), and I also got a feeling lower-case semantic web AIs, e.g. Google's Q&A feature, will be quicker to show results.

Misunderstanding #9: "CSS Hacks Are Always Superior"

Do you know how to separate between Internet Explorer 6 and The Rest? It's simple, just use the "!important" keyword, like this:

```
.content
{
width: 600px !important;
width: 580px;
}
```

Firefox and others will correctly interpret the content layer as to have a width of 600 pixels. 'Cause "!important" overrules. IE6 on the other hand erroneously grabs the last width it sees.

Or take this little hack:

```
.con\tent

{

   width: 500px;

}

.content

{

   width: 450px;

}
```

This one is the g-z backslash trick, supposed to separate between different flavors of MSIE. We used a lot of those in the stylesheets for Porsche.com's relaunch.

Those hacks are fun, sure, but are they superior to HTML table layout hacks? Well, a little, but CSS hacks are still just that: *hacks*. Workarounds that will cause troubles with every new browser. Workarounds the new developer won't get when you're away, unless she's a CSS expert. So instead of thinking that CSS hacks are a great way of developing websites, it's better to use them sparingly and accept them as what they are: an often necessary evil.
