Responsibility in web development

I sat on Jeremy Keith's presentation at AEA San Francisco on how to evaluate technologies for the web. It made me think a lot about the current iteration of the `progressive enhancement and what it means` debate and how do we need to accommodate all our users: those at the top end of the spectrum, the people in the middle and the next billion people who will come to our content from much slower connections, using much slower and cheaper devices and for whom data, or more specifically the cost of data, will continue to be the deciding factor on what content they access and how.

Here are some thoughts and ideas triggered by the presentation.

What assumptions does the technology make?

> To put it bluntly, software, like all technologies, is inherently political... Code inevitably reflects the choices, biases and desires of its creators.
>
> Jamais Cascio — Opennes and The Metaverse... Singularity

There is no such thing as neutral software. I love Cascio's quote and I think it fully reflects the issues we face when dealing with software.

What assumptions do we make when we send 2 megabytes of Javascript down the wire to our users?

Applications that rely on Javascript to do everything, will send large payloads of Javascript down the wire and that will work really well on a fast 3G, LTE or 4G networks, what we usually don't do enough of is test in actual mobile devices or, at the very least, test with actual devices in a lab setting.

Chrome Dev Tools provides a network simulator. It is a good starting point but it doesn't provide a fully accurate answer. There are things that we can not test in an emulator. Some of these things include:

- How long does it take the device to wake up and spin up all its cores
- Simulate different speeds in each core. Unlike desktops and laptops with symmetric multi cores, mobile devices use cores with different speeds asymmetric multi cores and once a task is performed the cores power down to save battery
- How the devices manages heat. Mobile devices don't disipate heat as well as desktops or laptops so they seldom work at full power

Webpagetest.org provides a better solution by allowing you to test in actual devices and actual browsers. The mobile devices are all located in a single location (east coast of the United States)

How are users supposed to handle almost 1.5 megabytes of images in a slow connection?

We've allowed our pages to grow fat. We have gotten lazy or, possibly, made the wrong assumptions regarding our users or, more likely, about the networks our users are in. Images are the largest components of our pages... this graphic is from June, 2015 but there is no reason to believe the trend will get better.

Figure 1: Image size in relation to overall page size, May 2015

The graphic below, also from Soasta's blog post, show how the size of our web content has changed over the years... and it shows no signs of decreasing.

Figure 2: How big have our pages become

So how do we fix this?

To start we should move all icons and non photographic images to SVG or other vector formats. Because they are text-based they are smaller than equivalent raster graphic files. Furthermore, they scale with your resolution and make it possible to use a single image for both normal resolution and the multiple high density devices in the market.

Responsive images may also be a response to how we deliver images to our clients. We no longer need to feed megabytes of images to mobile devices and on the other hand we no longer have to serve low resolution images just for the sake of saving bandwidth.

And if everything else fails we can always fall back on our regular image compression tools to reduce the file size for the images on our pages. Tools like Imagemin can be used from either CLI or build systems to compress images.

This may affect quality for our high end / fast connection users but I think it's a price worth paying to get our payloads slim enough

How well it works? How well it fails?

> None of these parts operate independently. No mechanical system can function by itself. Each bit of technology requires the viability and growth of all the rest of technology to keep going. There is no communication without the nerves of electricity. There is no electricity without the veins of coal mining, uranium mining, or damming of rivers, or even the mining of precious metals to make solar panels. There is no metabolism of factories without the ingest of food from domesticated plants and animals, and no circulation of goods without vehicles. This global-scaled network of systems, subsystems, machines, pipes, roads, wires, conveyor belts, automobiles, servers and routers, institutions, laws, calculators, sensors, works of art, archives, activators, collective memory, and power generators – this whole grand system of interrelated and interdependent pieces forms a very primitive organism-like system. Call it the technium.
>
> Kevin Kelly — What Technology Wants

When we work with technology we tend to worry about how well the technology

works but we seldom, if ever, worry about what happens when the technology fails. I've documented elsewhere that we worry about the wrong things in the progressive enhancement debate... it's got less to do with whether Javascript is enabled and more to do with what happens if Javascript is not available or your site is not reachable in a timely manner.

I know you've probably heard Alex Russell, Paul Irish and Paul Lewis speak about these numbers as they relate to performance but I want to take a different approach and use them to talk about networks.

Even for people who, like me, live in bandwidth rich environments (San Francisco Bay Area of California) constant bandwidth is not a given. Many times, particularly when working on a train or traveling the bandwidth fluctuates between 3G, 4G and LTE wireless networks. It's never one reliable connection and whenever the browser in my phone detects the switch it will stop loading the content and will leave the application in whatever state it is, usually not loaded or not fully working.

Who (directly) benefits?

> In case of conflict, consider users over authors over implementors over specifiers over theoretical purity. In other words costs or difficulties to the user should be given more weight than costs to authors; which in turn should be given more weight than costs to implementors; which should be given more weight than costs to authors of the spec itself, which should be given more weight than those proposing changes for theoretical reasons alone.
>
> HTML Design Principles — Priority of Constituencies

Some times we forget who we're building sites and apps for. We think that bloated content is ok and that everyone will have the same network connectivity and access to high end devices as we do.

Particulary when we're beginning to use a technoloy or when we write our first tutorials for beginners it's easy to provide shortcuts and incomplete examples thinking that we can cover it in later chapters or in later sections of the tutorial and don't think that users who stumbled upon our content may not stick around long enough to get to more complete examples.

Take this React example, taken from buildwithreact.com, but found in similar fashion everywhere that you find React tutorials.

```
ReactDOM.render(
  <div>Hello!</div>,
  document.getElementById('container')
);
```

Or a slightly more elaborate version using `createClass` to define the component and then rendering it on the DOM.

```
// Create a component named MessageComponent
var MessageComponent = React.createClass({
  render: function() {
    return <div>{this.props.message}</div>;
  }
});

// Render an instance of MessageComponent into document.body
ReactDOM.render(
  <MessageComponent message="Hello!" />,
  document.body
);
"Hello!"
```

It tells you how to build a React component and how to attach it to the DOM but there is no mention on how to build a page to handle React failure because Javascript is not available, timed out, or had any other kind of failure.

If this is all a users see when looking for code to copy and paste then there is a problem... because this is all the code that will get copied into a page and modified until the next time she gets stuck and then the search will be for the next

I pick on React because it's the latest and greatest tool (for how long it reamins to be see) but any framework, library or script that relies on Javascript to be always on must provide a fallback for when it's not. HTML gives us the `<noscript>` element to handle these kinds of situations but do we really bother using it? I'll be the first one to admit that I don't.

There is also no accessibility baked into the react platform. The only thing I've been able to find is react-a11y that provides accessibility evaluation and cues for fixing these accessibility issues.

I won't go into the details of how React handles accessibility, I'll let Marcy Sutton's CSUN '17 presentation talk about the what and the how.

Figure 3: What will be the next greatest thing? (taken from Marcy Sutton's CSUN 2017 presentation)

React is the latest "Greatest Tool Ever" joining a long line of greatest tools that now exist. In researching accessibility for React components I've found out that this is not a trivial task and one that most beginning developers may or may not use.

We owe users a more complete and accessible web from the ground up. We should teach people to leverage the platform and only use special cases and frameworks when they are needed, not because they are "the best".