Evolving Complexity: Conclusion

Over the years the sheer number of tools that are available has become staggering; at least it was for me. I had to consciously make a choice about what I really wanted to work with and in what areas. These were my choices:

- HTML
- CSS
- Javascript
    - Current Version
- Data Visualization
- WebGL

Much to my surprise, each one of these areas had a much bigger footprint than I thought it did.

The list has evolved over time to something like this:

- HTML
    - Hand-coded
    - Templates
        - Mustache
        - Handlebars
- CSS
    - Vanilla CSS (features are added all the time)
    - SCSS (Superset of CSS with programming-like constructs)
- Javascript
    - Current Version at the time (not including HTML5 APIs)
        - ES5
    - ES Next (not including HTML5 APIs)
        - ES6
        - ES7 + whatever is next
    - Typescript
- Data Visualization
    - D3 V3
    - D3 V4
- WebGL
    - Three.js

- Node.js
    - Expres.js
- Tooling
    - Transpilers
        - Babel
    - Script Runners
        - Grunt
        - Gulp
        - NPM
    - Module Bundlers
        - Rollup
        - Webpack

Where there is more than one option on the list they are listed from the ones I learned first and moved to different tools either because they performed a task better or because a framework or library I wanted to work with used those libraries or build tools (looking at you Polymer).

The thing is: none of the items on either list need a full framework to work with them; rather, we need to ask ourselves what do we need for the project and how not to go overboard. No more JS frameworks presents a view of why we should carefully evaluate the needs of the projects.

> For a long time there was a whole lot of inconsistency between browsers and we, as an industry, had to write frameworks to paper over them. The problem is that there was disagreement even on the fundamental issues among browsers, like how events propagate, or what tags to support, so every framework not only papered over the holes, but designed their own model of how the browser should work. Actually their own models, plural, because you got to invent a model for how events propagate, a model for how to interact with the DOM, etc. A lot of inventing went on. So frameworks were written, each one a snowflake, a thousand flowers bloomed and gave us the likes of jQuery and Dojo and MochiKit and Ext JS and AngularJS and Backbone and Ember and React. For the past ten years we've been churning out a steady parade of JS frameworks.
>
> — No more JS frameworks

We've spent the best part of 2 decades working over how to make the web work

well, work consistently and work closer to the way we want them to.

In the beginning frameworks were necessary because browsers were incompatible and you had to make your code work the across browsers and, more recently, across form factors... but browsers have gotten better at implementing the same specs and in the same way. So the need to have normalization frameworks like jQuery, MooTools, Dojo, and others has lessened from this point of view.

You Might Not Need jQuery and (Now More Than Ever) You Might Not Need jQuery but, depending on your the browsers you must support, frameworks like jQuery might still be needed as John-David Dalton and Paul Irish wrote in jQuery's browser bug workaround.

Another thing worth remembering is that when you use a framework you're not only buying into the functionality you need; it's an all or nothing proposition. Either you use the full set of capabilities the framework offers or you are wasting bandwidth sending over unnecessary bytes if you don't slim your framework down to what you're actually using.

One last thing about the complexity curve. There may come a time when no one on your team will understand all of the code in your application and when the documentation will not be enough to do proper maintenance. What happens when you decide to refactor the code or more to a different framework that has become the latest and greatest?

A separate issue, for me, is when to use a library versus when to use a framework versus when vanilla javascript will work, and how to tell the difference.

In When Does a Project Need React? Chris Coyier presents a list of items to consider when implemeting an application and whether to use React and its ecosystem or other less complex alternatives. I think we need to ask the same type of question about any framework... is React overkill? Angular? Polymer?

I'm not against frameworks and libraries, they have their place. What worries me is that we're reaching for frameworks when we don't neccessarily have to and by not doing so we save ourselves from fatigue.

References

- Simplify the stack

        - Everything Easy Is Hard Again
        - Complexity
        - CSS Tricks Newsletter #83
        - The increasing nature of frontend complexity
        - I finally made sense of front end build tools. You can, too.
        - What is the Future of Front End Web Development?
        - The What and Why of Javascript Frameworks
        - Owning the Role of the Front-End Developer
    - Build Tools And Task Runners
        - JS Task Runners Comparison: Grunt vs Cake vs Gulp vs Broccoli
        - Comparison of Build Tools
    - JS Fatigue
        - Why I'm Thankful for JS Fatigue. I know you're sick of those words, but this is different.
        - What is JavaScript Fatigue?
        - The Ultimate Guide to JavaScript Fatigue: Realities of our industry
        - JavaScript fatigue fatigue
    - CSS In Javascript
        - CSS in JavaScript: The future of component-based styling
        - Stop using CSS in JavaScript for web development
    - Simplify the design
        - What Screens Want
        - The Web's Grain
        - Designing in The Borderlands
        - A Simpler Page
    - The Web Standard Project
        - The Web Standards Project
        - WaSP History
        - WaSP Mission
        - Our Work Here is Done
    - Responsive Web Design
        - Responsive Web Design
        - Responsive Web Design Examples
    - Bootcamps
        - Complete guide to the top 24 coding bootcamps
        - Bootcamps won't make you a coder. Here's what will
    - Conclusion
        - When Does a Project Need React?
        - Loading Third-Party JavaScript
        - Is React library or a framework?