Developer frustration and technology

There is a point when we need to decide if technology is a help or a hindrance.

While cognizant that things are different for different developers and that different projects have different requirements I can speak about some of the things I've learned over the years as a developer.

Don't take it personally

Technology can be supremely frustrating.

I've spent hours working on a problem only to figure out days later that I had misnamed a component and that was why it wasn't rendering as I wanted.

Sometimes an evaluation tool will tell you that you need to do "x" because your current result causes "Y" and gives you no ideas or solutions for how to accomplish "x" or do so in the most vague of terms.

It is only through trial and error that we learn how do these things. We've also come to depend on tedious and repetitive processes that are error prone when done manually or that require adopting cumbersome build systems. In later sections we'll explore how to create build systems that can be reused across projects.

When you get a result from an evaluation tool like Lighthouse you're getting an evaluation against the perfect standard. I have seen very few apps reach a perfect 100 score in Lighthouse when working outside of an Intranet. If you get a score of 90 you have to decide if the extra 10 points needed to reach 100 are worth the extra effort.

It's also a good idea to set up a target score for Lighthouse, Page Speed Insight and Pagespeed Test from the beginning. This will help manage expectations and will make developers look at the tool before having to face the tools with in production code.

So, take a deep breath and realize that this is not about you :-)

Clear project definition and priorities

When you start a project, whether personal or for a client, ask yourself the following questions

1. What's the goal for the project?
2. Who is your target audience?
3. Does the project impose a technology stack (like React, Angular or Polymer)?
4. How will you evaluate the project?

These questions help you get an idea of what the project will be like, what, if any, technologies you will have to add to your development stack, how much leeway do you have in terms of performance and, most important, how will you measure your project success.

What is the goal for the project?

The simplest way to do this is to ask **Why are you doing this?**

Are you learning how to use a new technology or a new methodology for working with the web stack? for example are you moving from LESS to SCSS or Stylus for CSS development? Are you teaching yourself how to use web components for a project?

Are you changing the way you address a technology you're already using? Are you implementing BEM, SMACKS or OOCSS for the the first time? Did you switch from SASS to SCSS syntax?

What other questions do you have about your project that are worth including in your project goal and objectives?

For targeted projects or for projects assigned to me at work this may already have been decided for you but it's also a chance to ask for the reasoning behind the decisions made. It may also be a chance to talk to your team and get a unified vision of how the project will work and how the team will work together.

Who is your target audience?

Once you've decided the what and the why you need to decide the who and,

indirectly, the where.

For most of my projects, those I decide to host publicly in Github, the answer is simple. I create the projects for myself and other developers who are interested in the technology.

Other projects may have requirements imposed from clients or management but it's still important to know who they are, what their connections look like and how the use their devices.

Does the project impose a technology stack?

We know that some stacks require you to use a certain language and/or a certain set of technologies. We know that if we're going to use Express that's a Node application with it's associated stack. If you're building for Android that requires Graddle, Android Studio and a fairly good knowledge of Java, both Oracle's implementation and Google's expansions, or Kotlin... The point is that every technology has a set of requirements in terms of languages and tools.

Identify the technology stack as early as possible and decide if this is something you're comfortable with. If this is a work project then, hopefully, your manager will make a decision based on your team's skills and strengths.

> Because I want to learn Go, I've chosen the Go language, its stack and its way to build applications. I can also leverage the ability to build for multiple platforms from the same source when it comes to deployment. I can build the package for a linux server from my Mac with very little extra work.

Netowrk and user analysis

For those projects that have a wider audience then we **must** take into consideration the network on our target users live in and how they use it. For example, users in India sometimes share SIM cards and the applications downloaded to them and a 2MB page will eat 20% of more of the bandwidth allocation.

I did take a look at general capabilities of browsers in China and India earlier in Who are the next billion users? and, even when looking at your personal projects, it's worth considering who is your target audience and where they are

located.

This is not a static process. We should constantly evaluate our logs and analytics against the initial analysis. It may yield unexpected and surprising results.

How will you evaluate the project?

Now comes, for me, the hardest part of the whole process: Evaluation.

How will you measure the success or need for improvement for your project?

Build a set of criteria that combines both object and subjective measurements.

For example: If you include documentation have someone else read it and see if it makes sense to them. Same thing if you include a README file as entry point for your documentation make sure it makes sense to someone who is not familiar with the project.

Objective criteria is easier to test. Either you get a 95 in Lighthouse or you don't.

I measure success in this big buckets:

- Working code without major bugs
- Applications are performant
- PWA scores 90 or higher in lighthouse
- Applications score between 85 and 90 in Google's Page Speed Insight
- Applications score in Pagespeedtest.org tests when using servers from 3 different geographical locations
- Application is properly documented (inline comments that can be used to build documentation and a README file)
- Ease of use

Best tool for the job: research up front

Once you know the technology the project will use you need to make sure you're up to date with best practices.

Many current best practices in front end web development seem to depend

tedious manual processes or having a build system in place. he type of build system you use in your project will depend on the requirements of your framework and the language you've chosen to work with.

As you learn more about best practices in your

Some elements that I've chosen to include in a basic starter set:

- Gulp-based. I started with Grunt and Grunt is still better when creating multiple versions of the same task. Gulp is better in creating tasks programmatically and using tools directly
- Babel transpilation for modern JS (ES2015+) using Babel
- Javascript linting (with ESLint and the Google preset), and documentation (using JSDoc)
- Imagemin image compression for PNG, GIF, JPG and SVG
- A 2-pass Ruby SASS/CSS toolchain. The first pass lints the SASS files using SASSLint and documents them using SASSDoc. The second pass adds prefixes for a predefined set of older browsers using auto-prefixer and minimizes the CSS (currently using CSS Nano, likely to change)
- Critical path CSS generator. It will find what CSS is needed for the critical path at different resolutions and insert it into the HTML pages
- Command line Page Speed Insights reports for both desktop and mobile

Granted, this is just a beginning stage. Some things become aparent when you're researching specific technologies:

- If you're working with JSX you'll have to add a preset to the Babel task and create a new task to process your JSX into something usable in a browser. A good example of current React/JSX/Redux ecosystems is at https://www.toptal.com/react/navigating-the-react-ecosystem
- If you use ES6 modules you will need a bundler like Webpack or Rollup
- Angular has it's own ng-cli to create, manage, compile and work with Angular 2 applications
- Polymer also has a polymer-cli to work with Polymer projects (both elements and applications)

Start small and grow

One thing that caught me for several years in front end development is the notion that everything has to be done in one block and all at once. It wasn't until I read Rebecca Murphey's essay Evolution in Beautiful Javascript that I realized that

Javascript can be as modular as other programming languages but that it can be a gun language to experiment with at the same time.

It’s ok to start with smaller pieces of your project. If you’re building a CMS it may be a good idea to build a static file server first that will work on your file system and then integrate it into a larger CMS file that will serve files from a database.

> For this project I will first build a static file server, compile it and use it as a development tool. I will then integrate it with MongoDB and finally put it all together in a web server.

Reuse code wherever possible

One of the advantages of starting small is that you can reuse the code you write in multiple places. Whether you call it a Go Package, Node Module, ES2015 module or whatever else your languages is there is a notion of smaller reusable components.

Make sure that these components are usable in multiple places. It’s not always possible to reuses components verbatim but the work in adapting an existing module surely is far less than rewriting a similar module from scratch.

Conclusions

Yes, front end development (and software development in general) can be very frustrating but with some preparation and the knowledge that it’s not you the process can be way less painful than it needs to be.