Engagement Engineering Consultant Guidelines

Release 0.1

Mozilla

July 06, 2016

Contents

These guidelines are meant to ensure consultants working for Engagement Engineering have clear instructions to follow when delivering code. Whenever possible these guidelines should be integrated into contracts and statements of work.

Contents:

CHAPTER 1

Getting Started

Prior to entering into a contract with a consulting company, please read through and familiarize yourself with this documentation.

There are 3 relationships EE can have with consultants.

1. Fully Outsourced
2. Fully Outsourced With Handover
3. Partially Outsourced

It is important to define the type of relationship before creating a contract and SOW. The type of relationship informs what the constultants and Mozilla are responsible for.

Fully Outsourced

1. All code written by consultant.
2. Site not hosted on Mozilla resources.
3. No expectations of EE resources providing maintenance or feature devleopment.

Fully Outsourced With Handover

1. All code written by consultant.
2. Site may be hosted on Mozilla resources.
3. Expectations of EE resources providing maintenance or feature devleopment post launch.

Partially Outsourced

1. Non production ready code is written by consultant.
2. Site may be hosted on Mozilla resources.
3. EE resources finish development and are responsible for feature development post launch.

CHAPTER 2

Questions For RFPs

It is a challenge to vet agencies during the RFP stage. You may be including the development of the 'what' in the RFP so it's entirely possible that very few technical details will be known at the RFP stage.

However there are a few questions you can include in the RFP to tease out potential risks. Any risks should be mitigated by addressing them specifically in the SOW/Contract.

Questions To Include In An RFP

- Have they launched projects similar to what you are asking for? If so, include examples and include links to source code if available.
- Do they work on projects similar in scale to what you are asking for? If so, include examples and include links to source code if available?
- What will be the composition of the team working on this project? What are each persons expertise and experience?
- Do they have engineers on staff who will be responsible for this work? If sub-contracted, who are the sub-contractors?
- What is their preferred/proposed tech stack for this application?

CHAPTER 3

Communication

Establish and agree upon communication channels in your kickoff meeting.

- **Single point of contact:**
  - Establish a single point of contact at Mozilla responsible for communicating with the consultants for the duration of the project.
- **Define how the team will communicate including:**
  - Meeting cadence.
  - IRC, Email, Slack or other appropriate communication methods.
  - Establish how progress on deliverables will be tracked and communicated.
  - Reporting & tracking bugs.
  - Establish documentation, what and where?

CHAPTER 4

Version Control

All code provided by a third party must reside in version control and be made available to Mozilla.

The following details should be decided upon and outlined in the SOW.

- **Using Version Control:**
  - Project code must be in version control.
  - Code should reside on Github unless another platform has been specifically agreed to.
  - Github usage and worfklow should follow WebDev guidelines .
  - Private repositories are allowed but must be mutually agreed upon.
  - Access to repository should be granted to Mozilla at kickoff.
  - At project completion Mozilla must have full admin to the repository or the repository must be transfered to Mozilla.

CHAPTER 5

Code Guidelines

Code you deliver to Mozilla as a consultant should be written according to the same standards as code written by any other Mozilla developer. Plan for your code to be maintained by other coders and comment liberally.

5.1 Code Style

Standardizing a common style for writing and formatting code helps a team or organization maintain a sensible code base. The goal is consistency and predictability so that any coder can view the work of another coder and quickly follow along. We have documented style guides elsewhere that go into greater depth, but we'll also summarize the main points here for some common languages.

We urge you to read through the Mozilla Webdev Bootcamp docs for more detailed guidelines.

5.1.1 General

- Use spaces for indentation. Never use tabs – history has shown that we cannot handle them.
- Use four spaces to indent most languages, but two spaces in HTML. HTML lends itself to a lot of nested elements and indenting each level four spaces can lead to long lines and messy formatting.
- Wrap lines at 80 characters when possible. HTML is often the exception here as well.
- Eliminate trailing whitespace at the end of lines. Blank lines should have no spaces.
- Include a single blank line at the end of files.

5.1.2 Python

- Follow PEP8.
- Follow Pocoo's extensions to PEP8, although these are a little less strictly enforced across Mozilla projects.
- Check your code against a linting tool. We highly recommend Flake8 for this.
- Use single quotes unless double (or triple) quotes would be an improvement.
- To continue a new line use a `()` not `\`.
- Indent code with either a hanging indent or a 4 space indent on the next line.

5.1.3 HTML

- Use HTML5, with the appropriate doctype: `<!DOCTYPE html>`
- Omit XML-style trailing slashes: `<br />`
- Use lowercase for tags and attributes.
- Use double quotes for attribute values.
- Validate your markup.
- Avoid inline or embedded CSS or JavaScript.
- Indent nested elements two spaces; don't use tabs.
- Use the most semantically valuable element suited to the content.

5.1.4 CSS

- Multi-line rules, not single line.
- Four space indentation; don't use tabs.
- Order declarations alphabetically (with some exceptions).
- Use the simplest, least specific selector possible.
- Make meaningful names, not presentational.
- All lowercase for classes and IDs, no camelCase.
- ID selectors are allowed but use them sparingly and appropriately.
- Don't use `!important`.
- Use unitless `line-height`.
- Group related rules into sections.
- Order sections and rules from general to specific.
- Validate!

Avoid stock CSS frameworks such as Twitter's Bootstrap or Zurb's Foundation. These frameworks are great tools for building prototypes, but shouldn't be used in a production website. A stock framework is by necessity a generic, one-size-fits-all, off-the-rack solution, not something designed for a specific purpose and tailored for specific content.

Most CSS frameworks also come with an over-reliance on presentational classes to dictate layout and style, permanently coupling structure to presentation. Want to change the color of a button? You might have to change the class on every button on the site. That isn't how CSS is supposed to work.

Frameworks tend to include a lot of stuff you won't need for a given project since they're built to cover every possible situation. Unless you can customize the build (something the better frameworks offer) you can end up with unnecessarily bloated cruft. It's often feasible to begin with a stock framework and customize it to your needs, renaming classes and eliminating cruft. That may be more efficient than building a system from scratch, depending on the complexity of the system. Choose wisely.

5.1.5 JavaScript

- **Always** use JSHint on your code.
- Assign each variable on a newline, not comma-separated.

- Use `[]` to assign a new array, not `new Array()`.
- Use `{}` for new objects, as well.
- Always use semicolons.
- Always use ===.
- Always use single quotes.
  - Exception: `"Don't escape single quotes in strings.  Use double quotes."`
- Cache regex into a constant.
- Try to avoid ternaries.
- Check for truthiness, not lack of falseness.

5.2 Progressive Enhancement

With few exceptions, Mozilla websites should practice progressive enhancement. Begin with simple, meaningful HTML. Add CSS and JavaScript as progressive layers in a way that still falls back to usable, accessible content in older browsers that don't support the latest features or in the event of JavaScript failing.

Don't rely exclusively on JavaScript to serve static content. JavaScript is a brittle language that can fail to execute in the browser for any number of reasons. This is less about catering to the few people who purposely disable JavaScript and more about ensuring some fault tolerance as a best practice. JavaScript can be a single point of failure. A one-page Ajax website that requires flawless client-side JavaScript to deliver all of its otherwise static content (text and images) becomes completely unusable and inaccessible if there's one error.

5.3 Browser Support

Mozilla websites should be tested in a broad range of browsers, though that doesn't mean a site must look or behave exactly the same in all browsers. Sites should look good and work well in any modern browser, even if it's not perfectly identical.

The current generation of browsers (versions released within the last few years) are all pretty even in their support of most modern web standards, but many people still use older browsers and deserve equal access to information. Consider following a system of graded browser support wherein the oldest or least capable browsers can still access all the content and essential functionality, just without all the bells and whistles better browsers can enjoy. If you use a progressive enhancement methodology, support for older/other browsers may already be a given.

Don't use proprietary features supported only by a single browser unless what you're making is a demo of that specific feature. Be mindful when you use emerging standards supported in some browsers but not others and include fallbacks for less capabale browsers. Check MDN or CanIUse for compatibility info. If you use vendor prefixes in CSS, include the prefixes for all supporting browsers as well as the unprefixed standard.

5.4 Accessibility

Strive to make all Mozilla websites reasonably accessible to people with disabilities and those using assistive technologies. In many cases this isn't a matter of adding accessibility features into a site, but simply *not* adding obstacles that make the site harder to use. Accessible sites are better for everyone, not only those with disabilities.

A few guidelines:

- Include descriptive `alt` text for any meaningful images in HTML.
- Include `<label>` elements in forms. Don't use `placeholder` as a label.
- Navigation, forms, and interactive widgets like dropdown menus and modal windows should be keyboard accessible.
- Ensure sufficient color contrast so text is readable for most users, including those with color vision dificiencies.
- Text should be resizable for those who need larger type. Make reasonable allowances for text to wrap and reflow when its size changes.
- Include ARIA attributes in HTML where appropriate.

CHAPTER 6

Hosting

There are 2 common scenarios for hosting a project developed by consultants.

1. Consultant Hosted
2. Mozilla Hosted

6.1 Consultant Hosted

Consultant hosted means the 3rd party is taking responsibility for hosting the website. The project is not hosted on Mozilla hardware and Mozilla is not responsible for either performance or uptime.

The following responsibilites should be outlined in a SOW:

Consultant Responsible

- **Billing, either direct or pass through.**
  - Include duration of hosting.
  - Provide estimates of all costs (hosting/bandwidth/other) including any possible overages.
- **Ensuring hosting is performant.**
  - Ensure project can handle expected traffic.
  - Ensure uptime for duration of project.

Mozilla Responsible

- **Domain Configuration:**
  - Registration of any domains.
  - DNS & SSL for any domains.

6.2 Mozilla Hosted

Mozilla hosted means the project is hosted on Mozilla hardware. Mozilla is responsible for providing performant hosting.

The following responsibilites should be outlined in a SOW:

Consultant Responsible

- **Delivering Code:**
  - Deploying and configuring code that works properly on Mozilla's Infra.
  - Ensuring project works as intended on Mozilla Infra.

Mozilla Responsible

- **Providing Infrastructure & Documenting Environments & Deployment:**
  - Registration of any domains.
  - DNS & SSL for any domains.
  - Providing (if needed) dev, stage and production hosting.
  - Documentation on how code will be deployed to each environment and by whom.
  - Any specific implementation details specific to Mozilla infra.

CHAPTER 7

QA

Who is doing QA and how should be determined prior to a signing a contract or SOW.

Generally there are 2 approaches for QA used with consutlants, the first being the most common:

1. Combined QA, Mozilla + Consultant
2. Consultant QA Only

7.1 Combined QA

Consultants are expected to deliver working code, however in many cases they do not have the same standards for QA or expertise to fully deliver a project. In most circumstances we will want to have Mozilla's QA providing additional reviews and testing.

In this scenario you should do the following.

- **Request a WebQA Resource:**
  - They should review the contract and SOW and determine their level of involvement.
  - They should develop a QA plan that is reviewed and agreed upon by Mozilla and the consultant.

7.2 Consultant Only QA

If the consulting firm is large enough and the project is fully outsourced they may handle the entirety of the QA process. This must be defined in the SOW along with any project specific areas of concern. These areas will be specific to the project but some areas to think about are listed below.

- Browser support.
- Mobile support.
- Accessability support.
- Verifying proper analytics behavor.
- Testing integration with any 3rd party services.
- Load testing.

Even in circumstances where the consutling firm is handling the entirety of the QA, a method for Mozilla submitting bugs should still be established at Kickoff.