Thoughts on Elm

I recently had the opportunity to build a [frontend application]() using [Elm](). Here are a few thoughts on the experience.

**Sometimes a smaller ecosystem is a good thing.** In a larger ecosystem, I find myself spending most of my time searching for pre-built solutions and trying to make sense of them. Almost everything you could ever want to do has probably already been done in a really robust way by someone else, so it’s often just a matter of finding the existing solution and plugging it in. It can be comforting to have all this figured out for you, but it also takes away opportunities for you to innovate and contribute your own ideas. Additionally, reading docs and adjusting configs is very passive and can result in me feeling more detached from my work than if I was down in the trenches coming up with my own solutions.

**I miss thinking in terms of objects and components.** Over the years, my thought process has become very object oriented. I’m always trying to pull out new objects from my code and organize them into classes with clear separation of responsibilities. In Elm, you’re encouraged to think more loosely about how you organize your code. That opens up a lot of opportunities to try different things but I often just feel lost and end up with modules that look a lot like my classes would look in an object oriented language. I even started building a component hierarchy at one point, but stopped when trying to pass state up and down it became too much of a headache. I think I’m slowly getting the hang of things, but I still worry that a huge refactor is not far off since most of my decisions on where to put things feel very uncertain.

**Elm’s runtime and compiler relieve a lot of my fear and paranoia.** I’ve become accustomed to feeling like my code always has bugs lurking beneath the simple test cases I exercise it with when writing in scripted languages like Ruby. In Elm, the compiler and runtime bring all the potential bugs right to the surface. They manifest as missing `case` statements rather than as runtime exceptions. For example, if I need to make an HTTP request, the runtime takes responsibility for actually executing it and then the compiler makes sure I deal with all the possible outcomes. So you’re forced into doing things in a safe way right from the start, compared to in Ruby where there’s a million different ways to make a HTTP request and its easy to forget to check for a specific error and have an exception sneak through.

So overall, my experience with Elm has been really great! It’s been fun to learn and I’ve gotten a pretty solid application shipped in the process. It’s a nice change of pace from the object oriented and scripted languages that I’m used to working with. I’ve also been really impressed with the thought and care [its creator]() has put into its evolution. He hasn’t been afraid to challenge conventional thinking and push for solutions that really make peoples lives better rather than just trying to get everything working in the quickest way possible. I’m hopeful I’ll find ways to contribute to the Elm community going forward because I think it’s an important alternative to some of the more established solutions that its commonly compared to.