# ARCHITECTURE.md

Feb 6, 2021

If you maintain an open-source project in the range of 10k-200k lines of code, I strongly encourage you to add an `ARCHITECTURE` document next to `README` and `CONTRIBUTING`. Before going into the details of why and how, I want to emphasize that this is not another "docs are good, write more docs" advice. I am pretty sloppy about documentation, and, e.g., I often use just "simplify" as a commit message. Nonetheless, I feel strongly about the issue, even to the point of pestering you :-)

I have experience with both contributing to and maintaining open-source projects. One of the lessons I've learned is that the biggest difference between an occasional contributor and a core developer lies in the knowledge about the physical architecture of the project. Roughly, it takes 2x more time to write a patch if you are unfamiliar with the project, but it takes 10x more time to figure out *where* you should change the code. This difference might be hard to perceive if you've been working with the project for a while. If I am new to a code base, I read each file as a sequence of logical chunks specified in some pseudo-random order. If I've made significant contributions before, the perception is quite different. I have a mental map of the code in my head, so I no longer read sequentially. Instead, I just jump to where the thing should be, and, if it is not there, I move it. One's mental map is the source of truth.

I find the `ARCHITECTURE` file to be a low-effort high-leverage way to bridge this gap. As the name suggests, this file should describe the high-level architecture of the project. Keep it short: every recurring contributor will have to read it. Additionally, the shorter it is, the less likely it will be invalidated by some future change. This is the main rule of thumb for `ARCHITECTURE` — only specify things that are unlikely to frequently change. Don't try to keep it synchronized with code. Instead, revisit it a couple of times a year.

Start with a bird's eye overview of the problem being solved. Then, specify a more-or-less detailed *codemap*. Describe coarse-grained modules and how they relate to each other. The codemap should answer "where's the thing that does X?". It should also answer "what does the thing that I am looking at do?". Avoid going into details of *how* each module works, pull this into separate documents or (better) inline documentation. A codemap is a map of a country, not an atlas of maps of its states. Use this as a chance to reflect on the project structure. Are the things you want to put near each other in the codemap adjacent when you run `tree .`?

*Do* name important files, modules, and types. Do *not* directly link them (links go stale). Instead, encourage the reader to use symbol search to find the mentioned entities by name. This doesn't require maintenance and will help to discover related, similarly named things.

Explicitly call-out architectural invariants. Often, important invariants are expressed as an *absence* of something, and it's pretty hard to divine that from reading the code. Think about a common example from web development: nothing in the model layer specifically doesn't depend on the views.

Point out boundaries between layers and systems as well. A boundary implicitly contains information about the implementation of the system behind it. It even constrains all *possible* implementations. But finding a boundary by just randomly looking at the code is hard — good boundaries have measure zero.

After finishing the codemap, add a separate section on cross-cutting concerns.

A good example of `ARCHITECTURE` document is this one from rust-analyzer: architecture.md.

This post is a part of One Hundred Thousand Lines of Rust series.