Write a Design Doc—even if no one else will read it

3-4 minutes

I often write design documents even if no one will read them.

There are a lot of resources out there on how to write good design documents. There are also many different ways to define what constitutes a design doc—what it includes, how long it is, how formal it is, etc.

For my purposes, a design doc is any document that you write before you begin the actual implementation. It can be long or short, formal or informal, etc. The point is it's something you do independently of the implementation.

Most of the known benefits of writing design docs center around organizational alignment. Design docs can help you plan, help you get input from others on your team or in your company, serve as a record for the future. At larger companies, they're also a great educational channel. While experienced engineers debate pros/cons of different approaches, many other can watch from the stands.

I'm a big fan of design documents on large teams and at large companies, but I still find them tremendously valuable even if no one else reads them.

A good design doc includes, at some level of detail:

- What you're planning to do.
- Why you're doing it.
- How you're going to do it (including discussions of alternative implementations).

Being forced to write those things down (even if it's in a few sentences or paragraphs plus a diagram or two) sets a minimum bar that can help solve a lot of software development problems.

1. **Thinking strategically instead of tactically.** Tactical thinking focuses on the details and on immediate results. Strategic thinking focuses on higher-level concepts (what we'd call "architecture") as well as on the future. Code lends itself to tactical thinking. Design docs force strategic thinking.
2. **Creative thinking.** Complementary to strategic thinking, when writing out a plan, you'll often realize that there are alternative solutions to the problem you're trying to solve (or in some cases, that the problem you're trying to solve isn't worth solving). It's hard to do this when you're bogged down in implementation details.
3. **Avoiding complexity and obscurity.** Being forced to articulate your plan in pain English can often expose complexity. Often, things that are complex tend to be hard to describe, and so, if you think your implementation is simple but are finding that writing out your high-level plan is hard, it's a good indicator you're wrong about how simple it is.

It is, of course, entirely possible to sometimes begin with the implementation first, but in this case, you should treat the implementation as a discovery implementation or a prototype to collect some "on the ground details". But once you have those details, *then* you write your design doc before beginning the real implementation.

Post navigation