JULY 13, 2018 / #SOFTWARE DEVELOPMENT

How to write a good software design doc

As a software engineer, I spend a lot of time reading and writing design documents. After having gone through hundreds of these docs, I've seen first hand a strong correlation between good design docs and the ultimate success of the project.

This article is my attempt at describing **what makes a design document great**.

The article is split into 4 sections:

- **Why** write a design document
- **What** to include in a design document
- **How** to write it
- The **process** around it

Why write a design document?

A design doc — also known as a technical spec — is a description of how you plan to solve a problem.

There are lots of writings already on why it's important to write a design doc before diving into coding. So all I'll say here is:

A design doc is the most useful tool for making sure the right work gets done.

The main goal of a design doc is to make you more effective by forcing you to think through the design and gather feedback from others.

As a general rule of thumb, if you are working on a project that might take 1 engineer-month or more, you should write a design doc. But don't stop there — a lot of smaller projects could benefit from a mini design doc too.

Great! If you are still reading, you believe in the importance of design docs. However, different engineering teams, and even engineers within the same team, often write design docs very differently. So let's talk about the content, style, and process of a good design doc.

Photo by Todd Quackenbush on Unsplash

What to include in a design doc?

design doc differently.

To start, the following is a list of sections that you should at least *consider* including in your next design doc:

Title and People

The title of your design doc, the author(s) (should be the same as the list of people planning to work on this project), the reviewer(s) of the doc (we'll talk more about that in the Process section below), and the date this document was last updated.

Overview

A high level summary that every engineer at the company should understand and use to decide if it's useful for them to read the rest of the doc. It should be 3 paragraphs max.

Context

A description of the problem at hand, why this project is necessary, what people need to know to assess this project, and how it fits into the technical strategy, product strategy, or the team's quarterly goals.

Goals and Non-Goals

The Goals section should:

- describe the user-driven impact of your project — where your user might be another engineering team or even another technical system

Non-Goals are equally important to describe which problems you **won't** be fixing so everyone is on the same page.

Milestones

A list of measurable checkpoints, so your PM and your manager's manager can skim it and know roughly when different parts of the project will be done. I encourage you to break the project down into major user-facing milestones if the project is more than 1 month long.

Use calendar dates so you take into account unrelated delays, vacations, meetings, and so on. It should look something like this:

```
Start Date: June 7, 2018
Milestone 1 — New system MVP running in dark-mode: June 28, 2018
Milestone 2 - Retire old system: July 4th, 2018
End Date: Add feature X, Y, Z to new system: July 14th, 2018
```

Add an `[Update]` subsection here if the ETA of some of these milestone changes, so the stakeholders can easily see the most up-to-date estimates.

Existing Solution

In addition to describing the current implementation, you should also walk through a high level example flow to illustrate how users interact with this system and/or how data flow through it.

Proposed Solution

Some people call this the **Technical Architecture** section. Again, try to walk through a user story to concretize this. Feel free to include many sub-sections and diagrams.

Provide a big picture first, then fill in *lots* of details. Aim for a world where you can write this, then take a vacation on some deserted island, and another engineer on the team can just read it and implement the solution as you described.

Alternative Solutions

What else did you consider when coming up with the solution above? What are the pros and cons of the alternatives? Have you considered buying a 3rd-party solution — or using an open source one — that solves this problem as opposed to building your own?

Testability, Monitoring and Alerting

I like including this section, because people often treat this as an afterthought or skip it all together, and it almost always comes back to bite them later when things break and they have no idea how or why.

Cross-Team Impact

How will this increase on call and dev-ops burden?
How much money will it cost?
Does it cause any latency regression to the system?
Does it expose any security vulnerabilities?

Open Questions

Any open issues that you aren't sure about, contentious decisions that you'd like readers to weigh in on, suggested future work, and so on. A tongue-in-cheek name for this section is the "known unknowns".

Detailed Scoping and Timeline

This section is mostly going to be read only by the engineers working on this project, their tech leads, and their managers. Hence this section is at the end of the doc.

Essentially, this is the breakdown of how and when you plan on executing each part of the project. There's a lot that goes into scoping accurately, so you can read this post to learn more about scoping.

I tend to also treat this section of the design doc as an ongoing project task tracker, so I update this whenever my scoping estimate changes. But that's more of a personal preference.

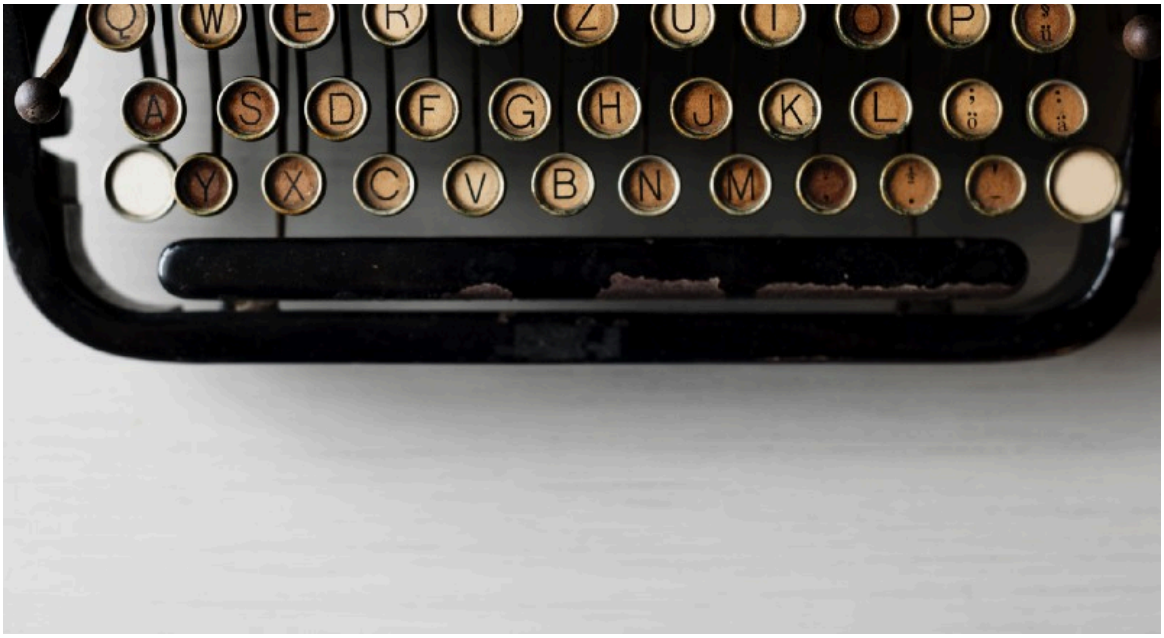

Photo by rawpixel on Unsplash

How to write it

Now that we've talked about **what** goes into a good design doc, let's talk about the style of writing. I promise this is different than your high school English class.

Write as simply as possible

Don't try to write like the academic papers you've read. They are written to impress journal reviewers. Your doc is written to describe your solution and get feedback from your teammates. You can achieve clarity by using:

- Simple words
- Short sentences
- Bulleted lists and/or numbered lists

Add lots of charts and diagrams

Charts can often be useful to compare several potential options, and diagrams are generally easier to parse than text. I've had good luck with Google Drawing for creating diagrams.

Pro Tip: remember to add a link to the editable version of the diagram under the screenshot, so you can easily update it later when things inevitably change.

Include numbers

The scale of the problem often determines the solution. To help reviewers get a sense of the state of the world, include real numbers like # of DB rows, # of user errors, latency — and how these scale with usage. Remember your Big-O notations?

Try to be funny

A spec is not an academic paper. Also, people like reading funny things, so this is a good way to keep the reader engaged. Don't overdo this to the point of taking away from the core idea though.

If you, like me, have trouble being funny, Joel Spolsky (*obviously* known for his comedic talents…) has this tip:

> One of the easiest ways to be funny is to be *specific* when it's not called for [… Example:] Instead of saying "special interests," say "left-handed avacado farmers."

pretending to be the reviewer. What questions and doubts might you have about this design? Then address them preemptively.

Do the Vacation Test

If you go on a long vacation now with no internet access, can someone on your team read the doc and implement it as you intended?

The main goal of a design doc is not knowledge sharing, but this is a good way to evaluate for clarity so that others can actually give you useful feedback.

Photo by SpaceX on Unsplash

Process

solution to the wrong problem. There's a lot of art to getting good feedback, but that's for a later article. For now, let's just talk specifically about how to write the design doc and get feedback for it.

First of all, everyone working on the project should be a part of the design process. It's okay if the tech lead ends up driving a lot of the decisions, but everyone should be involved in the discussion and buy into the design. So the "you" throughout this article is a really plural "you" that includes all the people on the project.

Secondly, the design process doesn't mean you staring at the whiteboard theorizing ideas. Feel free to get your hands dirty and prototype potential solutions. This is not the same as starting to write production code for the project before writing a design doc. Don't do that. But you absolutely *should* feel free to write some hacky throwaway code to validate an idea. To ensure that you only write exploratory code, make it a rule that **none of this prototype code gets merged to master**.

After that, as you start to have some idea of how to go about your project, do the following:

1. Ask an experienced engineer or tech lead on your team to be your reviewer. Ideally this would be someone who's well respected and/or familiar with the edge cases of the problem. Bribe them with boba if necessary.
2. Go into a conference room with a whiteboard.
3. Describe the **problem** that you are tackling to this engineer (this is a very important step, don't skip it!).

Doing all of this **before** you even start writing your design doc lets you get feedback as soon as possible, before you invest more time and get attached to any specific solution. Often, even if the implementation stays the same, your reviewer is able to point out corner cases you need to cover, indicate any potential areas of confusion, and anticipate difficulties you might encounter later on.

Then, after you've written a rough draft of your design doc, get the same reviewer to read through it again, and rubber stamp it by adding their name as the reviewer in the **Title and People** section of the design doc. This creates additional incentive and accountability for the reviewer.

On that note, consider adding specialized reviewers (such as SREs and security engineers) for specific aspects of the design.

Once you and the reviewer(s) sign off, feel free to send the design doc to your team for additional feedback and knowledge sharing. I suggest time-bounding this feedback gathering process to about 1 week to avoid extended delays. Commit to addressing all questions and comments people leave within that week. **Leaving comments hanging = bad karma.**

Lastly, if there's a lot of contention between you, your reviewer, and other engineers reading the doc, I strongly recommend consolidating all the points of contention in the **Discussion** section of your doc. Then, set up a meeting with the different parties to talk about these disagreements in person.

that you are still responsible for making the final call, even if everyone can't come to a consensus.

In talking to Shrey Banga recently about this, I learned that Quip has a similar process, except in addition to having an experienced engineer or tech lead on your team as a reviewer, they also suggest having an engineer on a *different* team review the doc. I haven't tried this, but I can certainly see this helping get feedback from people with different perspectives and improve the general readability of the doc.

Once you've done all the above, time to get going on the implementation! For extra brownie points, **treat this design doc as a living document as you implement the design**. Update the doc every time you learn something that leads to you making changes to the original solution or update your scoping. You'll thank me later when you don't have to explain things over and over again to all your stakeholders.

Finally, let's get *really* meta for a second: How do we evaluate the success of a design doc?

My coworker Kent Rakip has a good answer to this: **A design doc is successful if the right ROI of work is done.** That means a successful design doc might actually lead to an outcome like this:

1. You spend 5 days writing the design doc, this forces you to think through different parts of the technical architecture
2. You get feedback from reviewers that `X` is the riskiest part of the proposed architecture

more difficult than you originally intended

5. You decide to stop working on this project and prioritize other work instead

At the beginning of this article, we said the goal of a design doc is to **make sure the right work gets done.** In the example above, thanks to this design doc, instead of wasting potentially months only to abort this project later, you've only spent 8 days. Seems like a pretty successful outcome to me.

Please leave a comment below if you have any questions or feedback! I'd also love to hear about how you do design docs differently in your team.

Giving credit where credit is due, I learned a lot of the above by working alongside some *incredible* engineers at Plaid (we are hiring! Come design and build some sweet technical systems with us) and Quora.

If you like this post, follow me on Twitter for more posts on engineering, processes, and backend systems.
