



Blog



Log in

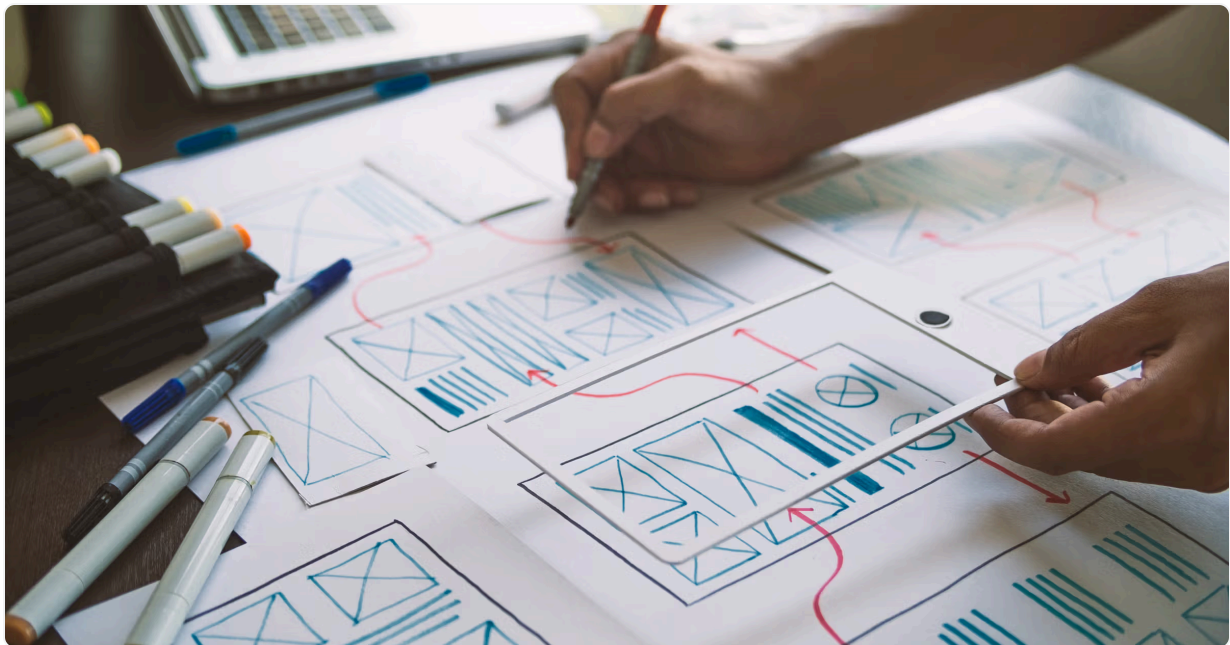
Sign up



APRIL 6, 2020

A practical guide to writing technical specs

Writing a technical spec increases the chances of having a successful project, service, or feature that all stakeholders involved are satisfied with. It decreases the chances of something going horribly wrong during implementation and even after you've launched your product.



As a software engineer, your primary role is to solve technical problems. Your first impulse may be to immediately jump straight into writing code. But that can be a terrible idea if you haven't thought through your solution.

You can think through difficult technical problems by writing a technical spec. Writing one can be frustrating if you feel like you're not a good writer. You may even think that it's an unnecessary chore. But writing a technical spec

increases the chances of having a successful project, service, or feature that all stakeholders involved are satisfied with. It decreases the chances of something going horribly wrong during implementation and even after you've launched your product.

In this article, I'll walk you through how to write a technical spec that ensures a strong product.

What is a technical specification document?

A technical specification document outlines how you're going to address a technical problem by designing and building a solution for it. It's sometimes also referred to as a technical design document, a software design document, or an engineering design document. It's often written by the engineer who will build the solution or be the point person during implementation, but for larger projects, it can be written by technical leads, project leads, or senior engineers. These documents show the engineer's team and other stakeholders what the design, work involved, impact, and timeline of a feature, project, program, or service will be.

Why is writing a technical spec important?

Technical specs have immense benefits to everyone involved in a project: the engineers who write them, the teams that use them, even the projects that are designed off of them. Here are some reasons why you should write one.

Benefits to engineers

By writing a technical spec, engineers are forced to examine a problem before going straight into code, where they may overlook some aspect of the solution. When you break down, organize, and time box all the work you'll have to do during the implementation, you get a better view of the scope of the solution. Technical specs, because they are a thorough view of the proposed solution, they also serve as documentation for the project, both for

the implementation phase and after, to communicate your accomplishments on the project.

With this well-thought out solution, your technical spec saves you from repeatedly explaining your design to multiple teammates and stakeholders. But nobody's perfect; your peers and more seasoned engineers may show you new things from them about design, new technologies, engineering practices, alternative solutions, etc. that you may not have come across or thought of before. They may catch exceptional cases of the solution that you may have neglected, reducing your liability. The more eyes you have on your spec, the better.

Benefits to a team

A technical spec is a straightforward and efficient way to communicate project design ideas between a team and other stakeholders. The whole team can collaboratively solve a problem and create a solution. As more teammates and stakeholders contribute to a spec, it makes them more invested in the project and encourages them to take ownership and responsibility for it. With everyone on the same page, it limits complications that may arise from overlapping work. Newer teammates unfamiliar with the project can onboard themselves and contribute to the implementation earlier.

Benefits to a project

Investing in a technical spec ultimately results in a superior product. Since the team is aligned and in agreement on what needs to be done through the spec, big projects can progress faster. A spec is essential in managing complexity and preventing scope and feature creep by setting project limits. It sets priorities thereby making sure that only the most impactful and urgent parts of a project go out first.

Post implementation, it helps resolve problems that cropped up within the project, as well as provide insight in retrospectives and postmortems. The

best planned specs serve as a great guide for measuring success and return on investment of engineering time.

What to do before writing a technical spec

Gather the existing information in the problem domain before getting started. Read over any product/feature requirements that the product team has produced, as well as technical requirements/standards associated with the project. With this knowledge of the problem history, try to state the problem in detail and brainstorm all kinds of solutions you may think might resolve it. Pick the most reasonable solution out of all the options you have come up with.

Remember that you aren't alone in this task. Ask an experienced engineer who's knowledgeable on the problem to be your sounding board. Invite them to a meeting and explain the problem and the solution you picked. Lay out your ideas and thought process and try to persuade them that your solution is the most appropriate. Gather their feedback and ask them to be a reviewer for your technical spec.

Finally, it's time to actually write the spec. Block off time in your calendar to write the first draft of the technical spec. Use a collaborative document editor that your whole team has access to. Get a technical spec template (see below) and write a rough draft.

End of year wrap up! Programming on your phone and generative AI



Contents of a technical spec

There are a wide range of problems being solved by a vast number of companies today. Each organization is distinct and creates its own unique engineering culture. As a result, technical specs may not be standard even within companies, divisions, teams, and even among engineers on the same team. Every solution has different needs and you should tailor your technical spec based on the project. You do not need to include all the sections mentioned below. Select the sections that work for your design and forego the rest.

From my experience, there are seven essential parts of a technical spec: front matter, introduction, solutions, further considerations, success evaluation, work, deliberation, and end matter.

1. Front matter

- Title
- Author(s)
- Team

- Reviewer(s)
- Created on
- Last updated
- Epic, ticket, issue, or task tracker reference link

2. Introduction

a. Overview, Problem Description, Summary, or Abstract

- Summary of the problem (from the perspective of the user), the context, suggested solution, and the stakeholders.

b. Glossary or Terminology

- New terms you come across as you research your design or terms you may suspect your readers/stakeholders not to know.

c. Context or Background

- Reasons why the problem is worth solving
- Origin of the problem
- How the problem affects users and company goals
- Past efforts made to solve the solution and why they were not effective
- How the product relates to team goals, OKRs
- How the solution fits into the overall product roadmap and strategy
- How the solution fits into the technical strategy

d. Goals or Product and Technical Requirements

- Product requirements in the form of user stories
- Technical requirements

e. Non-Goals or Out of Scope

- Product and technical requirements that will be disregarded

f. Future Goals

- Product and technical requirements slated for a future time

g. Assumptions

- Conditions and resources that need to be present and accessible for the solution to work as described.

3. Solutions

a. Current or Existing Solution / Design

- Current solution description
- Pros and cons of the current solution

b. Suggested or Proposed Solution / Design

- External components that the solution will interact with and that it will alter
- Dependencies of the current solution
- Pros and cons of the proposed solution
- **Data Model / Schema Changes**
 - Schema definitions
 - New data models
 - Modified data models
 - Data validation methods
- **Business Logic**
 - API changes
 - Pseudocode

- Flowcharts
- Error states
- Failure scenarios
- Conditions that lead to errors and failures
- Limitations
- **Presentation Layer**
 - User requirements
 - UX changes
 - UI changes
 - Wireframes with descriptions
 - Links to UI/UX designer's work
 - Mobile concerns
 - Web concerns
 - UI states
 - Error handling
- **Other questions to answer**
 - How will the solution scale?
 - What are the limitations of the solution?
 - How will it recover in the event of a failure?
 - How will it cope with future requirements?

c. Test Plan

- Explanations of how the tests will make sure user requirements are met
- Unit tests
- Integrations tests

- QA

d. Monitoring and Alerting Plan

- Logging plan and tools
- Monitoring plan and tools
- Metrics to be used to measure health
- How to ensure observability
- Alerting plan and tools

e. Release / Roll-out and Deployment Plan

- Deployment architecture
- Deployment environments
- Phased roll-out plan e.g. using feature flags
- Plan outlining how to communicate changes to the users, for example, with release notes

f. Rollback Plan

- Detailed and specific liabilities
- Plan to reduce liabilities
- Plan describing how to prevent other components, services, and systems from being affected

g. Alternate Solutions / Designs

- Short summary statement for each alternative solution
- Pros and cons for each alternative
- Reasons why each solution couldn't work
- Ways in which alternatives were inferior to the proposed solution

- Migration plan to next best alternative in case the proposed solution falls through

4. Further Considerations

a. Impact on other teams

- How will this increase the work of other people?

b. Third-party services and platforms considerations

- Is it really worth it compared to building the service in-house?
- What are some of the security and privacy concerns associated with the services/platforms?
- How much will it cost?
- How will it scale?
- What possible future issues are anticipated?

c. Cost analysis

- What is the cost to run the solution per day?
- What does it cost to roll it out?

d. Security considerations

- What are the potential threats?
- How will they be mitigated?
- How will the solution affect the security of other components, services, and systems?

e. Privacy considerations

- Does the solution follow local laws and legal policies on data privacy?
- How does the solution protect users' data privacy?

- What are some of the tradeoffs between personalization and privacy in the solution?

f. Regional considerations

- What is the impact of internationalization and localization on the solution?
- What are the latency issues?
- What are the legal concerns?
- What is the state of service availability?
- How will data transfer across regions be achieved and what are the concerns here?

g. Accessibility considerations

- How accessible is the solution?
- What tools will you use to evaluate its accessibility?

h. Operational considerations

- Does this solution cause adverse aftereffects?
- How will data be recovered in case of failure?
- How will the solution recover in case of a failure?
- How will operational costs be kept low while delivering increased value to the users?

i. Risks

- What risks are being undertaken with this solution?
- Are there risks that once taken can't be walked back?
- What is the cost-benefit analysis of taking these risks?

j. Support considerations

- How will the support team get across information to users about common issues they may face while interacting with the changes?
- How will we ensure that the users are satisfied with the solution and can interact with it with minimal support?
- Who is responsible for the maintenance of the solution?
- How will knowledge transfer be accomplished if the project owner is unavailable?

5. Success Evaluation

a. Impact

- Security impact
- Performance impact
- Cost impact
- Impact on other components and services

b. Metrics

- List of metrics to capture
- Tools to capture and measure metrics

6. Work

a. Work estimates and timelines

- List of specific, measurable, and time-bound tasks
- Resources needed to finish each task
- Time estimates for how long each task needs to be completed

b. Prioritization

- Categorization of tasks by urgency and impact

c. Milestones

- Dated checkpoints when significant chunks of work will have been completed
- Metrics to indicate the passing of the milestone

d. Future work

- List of tasks that will be completed in the future

7. Deliberation

a. Discussion

- Elements of the solution that members of the team do not agree on and need to be debated further to reach a consensus.

b. Open Questions

- Questions about things you do not know the answers to or are unsure that you pose to the team and stakeholders for their input. These may include aspects of the problem you don't know how to resolve yet.

8. End Matter

a. Related Work

- Any work external to the proposed solution that is similar to it in some way and is worked on by different teams. It's important to know this to enable knowledge sharing between such teams when faced with related problems.

b. References

- Links to documents and resources that you used when coming up with your design and wish to credit.

c. Acknowledgments

- Credit people who have contributed to the design that you wish to recognize.

After you've written your technical spec

Now that you have a spec written, it's time to refine it. Go through your draft as if you were an independent reviewer. Ask yourself what parts of the design are unclear and you are uncertain about. Modify your draft to include these issues. Review the draft a second time as if you were tasked to implement the design just based on the technical spec alone. Make sure the spec is a clear enough implementation guideline that the team can work on if you are unavailable. If you have doubts about the solution and would like to test it out just to make sure it works, create a simple prototype to prove your concept.

When you've thoroughly reviewed it, send the draft out to your team and the stakeholders. Address all comments, questions, and suggestions as soon as possible. Set deadlines to do this for every issue. Schedule meetings to talk through issues that the team is divided on or is having unusually lengthy discussions about on the document. If the team fails to agree on an issue even after having in-person meetings to hash them out, make the final call on it as the buck stops with you. Request engineers on different teams to review your spec so you can get an outsider's perspective which will enhance how it comes across to stakeholders not part of the team. Update the document with any changes in the design, schedule, work estimates, scope, etc. even during implementation.

Conclusion

Writing test specs can be an impactful way to guarantee that your project will be successful. A little planning and a little forethought can make the actual

implementation of a project a whole lot easier.

AUTHORS

Zara Cooper ›

[Bulletin](#)[Bulletin](#)[Code for a Living](#)[Stackoverflow](#)[Stackoverflow](#)[technical specs](#)

RECENT ARTICLES



FEBRUARY 13, 2025

How to harness APIs and AI for intelligent automation



FEBRUARY 10, 2025

Shifting left without slowing down: Q&A with Moti Gindi of Apiiro



FEBRUARY 6, 2025

Investing in the Stack Exchange Network and the future of Stack Overflow



FEBRUARY 3, 2025

Community Products Roadmap Update, January 2025

LATEST PODCAST



FEBRUARY 14, 2025

Solving the data doom loop

Add to the discussion



Login with your **stackoverflow.com** account to take part in the discussion.

Show **65** comments



Light Dark Auto

Stack Overflow for Teams

Pricing

Customers

Our solution

Integrations

Features

Customer Success

Security

Return on Investment (ROI)

OverflowAI NEW

Now available on Enterprise.

Try free

Log in

Use cases

Engineers

Data Scientists

DevOps & SRE

Support

Product Management

Resources

Productivity

AI/ML

Guides and Insights

Customer Academy

FAQ

Help center


Stack Overflow Advertising

Why Stack Overflow?

What to expect

Advertise to developers

Attract tech talent

Post a job 

Powered by Indeed

Use cases

Marketing Teams

Employer Branding Teams

DevRel Teams

Talent Teams

Technology Teams

Agencies

Resources

Product guides & insights

Community insights

Advertising best practices

Talent best practices

Company

OverflowAPI NEW

Stack Overflow's subscription-based API service to train and fine-tune large language models.

About

Leadership

Social Impact

Press

Careers

Open positions

Contact us

Blog

Newsletter

Podcast

Labs

Annual Developer Survey

Site design / logo © 2025 Stack Exchange Inc.
User contributions licensed under CC BY-SA.

[Terms](#) [Privacy policy](#) [Cookie policy](#) [Cookie settings](#) [Go to stackoverflow.com](#)

