# Design Documents

Sarah Morin

## Agenda

| | |
|---|---|
| 3 | Introduction |
| 4 | Motivation |
| 7 | Structure of Design Docs |
| 9 | Example: Chat Room |
| 10 | High Level Summary |
| 11 | Component Design |
| 12 | Development Plan |
| 15 | Conclusion |

## Motivation

Short-term – **Your writing assignments!**

Long-term – Good design docs are important

Cynical – Fewer annoying questions

## Why do we write design docs?

- Record of ideas (your future self will thank you)
- Identify problems early
- Team consensus
- External Collaboration - professors, mentors, managers, other teams etc.

**Don’t be intimidated by the blank page!**

## Common Pitfalls

### Stream of Consciousness

The freeform word dump

Easy to write, impossible to read

### The Everything Document

Background, Design, API Spec, Test Plan, Task Breakdown, and Schedule, all in one!

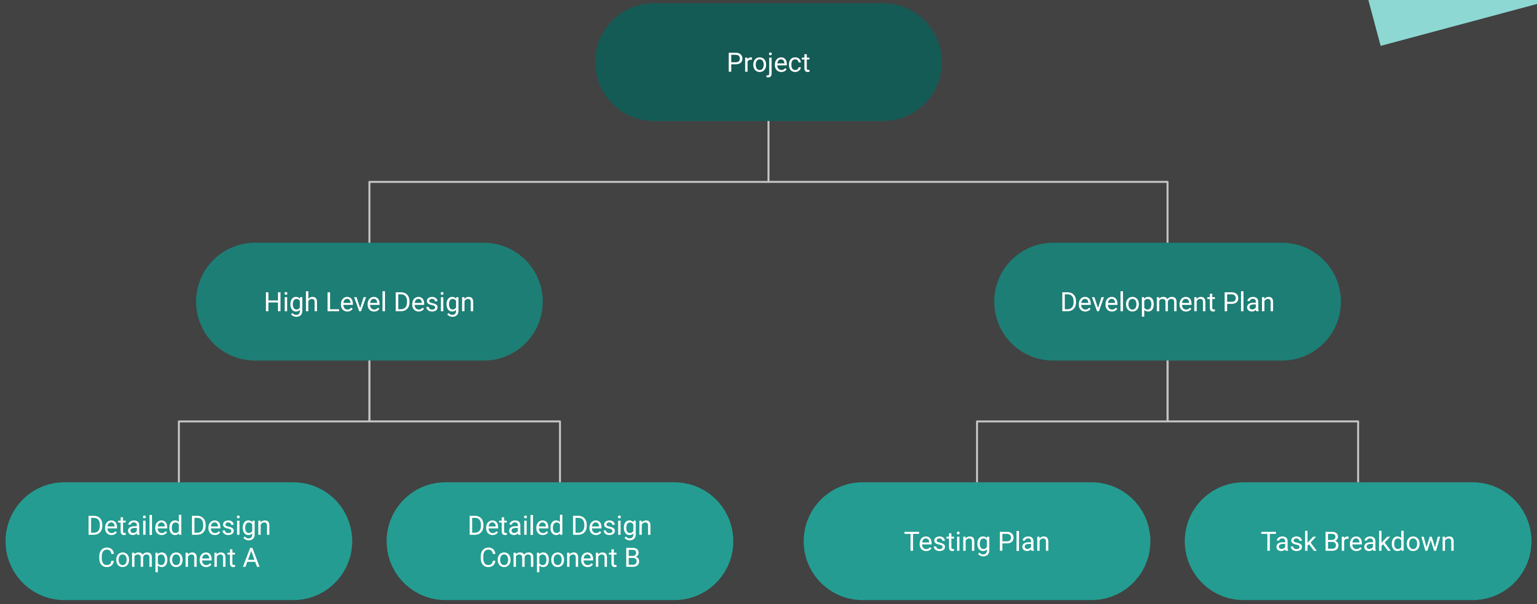
Hard to follow, harder to maintain

### Depth-first Design

Dive into the details, give context later!

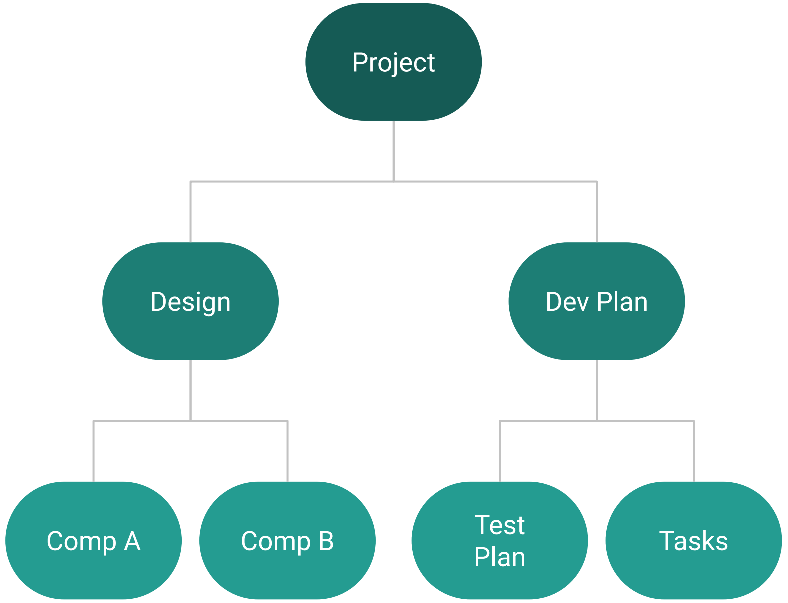
Easy for experts…but what about everyone else?

## Anatomy of Good Design Docs



## Why this structure?

- **Context first, details later**
  - Introduce readers to problem before diving into design
  - Readability without expertise
- **One thing at a time**
  - Separate background, design, testing, and task breakdowns
  - Everything in one monster doc = unreadable
- **Ease of navigation**
  - Common structure means everyone can find information they want quickly
  - Easy to refer to development info (e.g. test plan) without re-reading the entire design



## Design Docs by Example: Chat Room

## Making a Better Chat Room

**What we have:** A super basic chat room

- A single, open room users can freely join and leave
- Message history only persists locally for each user from the time they join to when they leave

**What we want:** Private channels with persisted message history

- User’s can create channels and manage members
- Channels store a complete message history
- Distinguish between “becoming a member” (joining) and “opening the channel” (joining)

## High Level Summary

| | | |
|---|---|---|
| 01 | Problem Summary and Background | "The current chat room is rudimentary. We want to add two features: private channels and persistent message history…" |
| 02 | Requirements, Goals, Non-goals | • **Functional :** "Users can create private channels"<br>• **Performance:** "Load the most recent X messages when users connect to a channel"<br>• **Non-goal:** "Immediate garbage collection of deleted channels, this will be eventual" |
| 03 | Solution Summary | • **New Components:** Persistent Message Store, Membership Database, Multiple Chat-Servers, Load Balancer, etc.<br>• **New Algorithms:** Loading channel history, Load balancing strategy to map clients to chat-servers based on desired channel |
| 04 | Diagrams and Workflows | • Updated system component diagram<br>• Channel creation workflow and components involved |
| 05 | Trade-offs, Performance, and Concerns | • **Trade-offs**: "Membership database will be lock protected, we choose correctness and safety over performance for operation like adding a member."<br>• **Concerns**: "We introduce a lot of coordination requirements between system components. Testing must be aggressive" |

## Component Design | Membership Database

| | | |
|---|---|---|
| 01 | Component Summary | "The membership database stores information about users, channels, and the relationship between them. It is the source of truth when determining if a user can join a channel…." |
| 02 | Requirements | • **Basic Functionality**: "Store User and Channel Information, Membership relationship…"<br>• **Supported Requests:** "create/delete users/channels, add/remove member, …"<br>• **Performance Goals**: "Store X channels and Y users without performance degradation" |
| 03 | Detailed Design | • What type of storage do we use? Specific database type?<br>• Schema for users, channels, and membership records<br>• Sorting strategy for records and reasoning |
| 04 | API Specs | create_user(username, display_name=None, photo=None) -><br>○ Success<br>○ Error: username exists<br>○ Error: Membership DB Unreachable |
| 05 | Performance Analysis and Future Improvements | "In the future we will add different types of users to channels like owner, administrator, read-only, etc." |

## **Development** | Test Plan

|  |  |  |
|---|---|---|
| 01 | Unit Tests | **Basic, per-component tests that are easy to run and verify quickly**<br>create_user(sarah) -> Success, create_user(sarah) -> Error: username exists |
| 02 | Integration Tests | **Test new workflows through multiple components**<br>create_user, create_channel, connect to channel, send messages, disconnect, reconnect -> touches membership DB, chat servers, and persistent message store |
| 03 | Performance Tests | **Test/determine performance limits of the system**<br>Create channel, add 1000 members, spawn 1000 client processes to connect to channel and send messages. |

Do I actually need to write out every test case?

# **Development** | Task Breakdown & Schedule

## 01 Task Breakdown

- Divide design into manageable chunks of work
- Identify dependencies - where can we develop concurrently?
- Do we need to reach out to other teams?

## 02 Schedule

- Estimate work hours for each task
- Divide project into sets of tasks based on # of engineers, work estimates, and dependencies
- Timeline = longest set of tasks + extra time for mistakes

## 03 Assign

- Make tickets, boards, etc. for tasks
- Assign tasks to engineers

## Useful Resources

How to Write an Effective Design Document | Rina Artstain

Design Docs at Google | cramforce

Writing Design Docs | Oppia