Documentation

Watched 2 videos in our breakout rooms:
https://www.youtube.com/watch?v=R6zeikbTgVc – Writing effective documentation

https://www.youtube.com/watch?v=uPMxUnBjGG8 – Top 5 ways to document your code

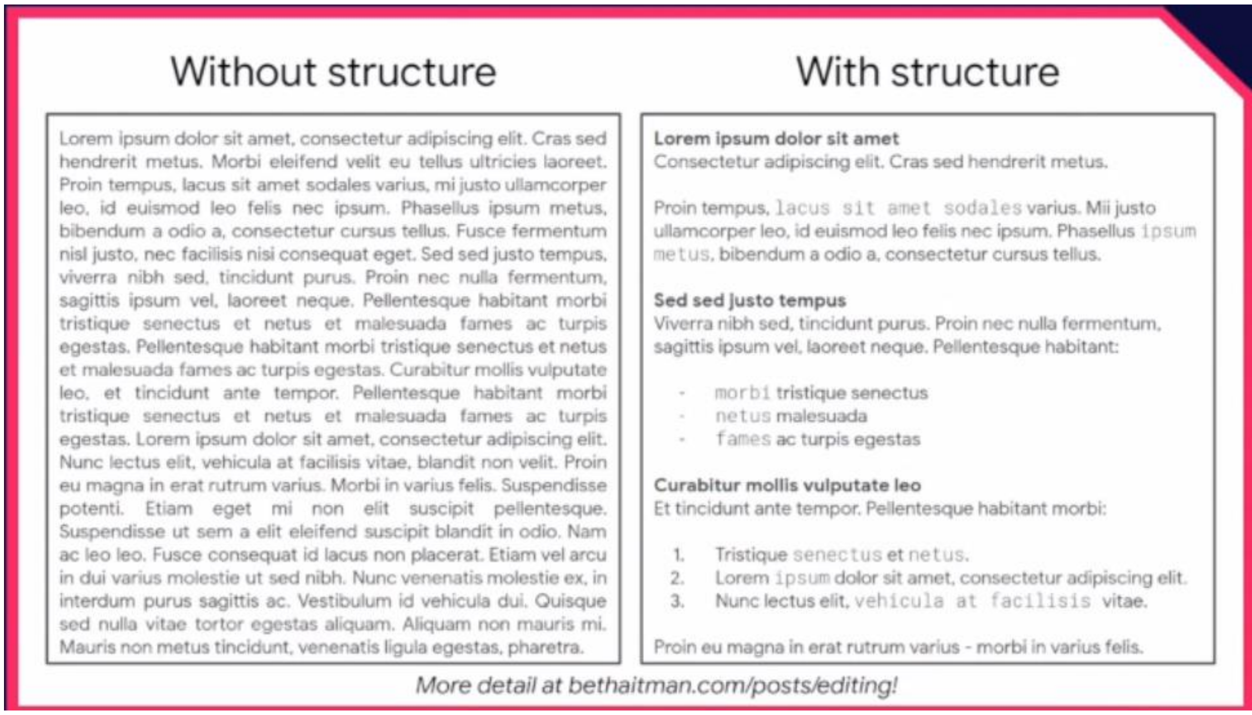
Below are my takeaways from the videos.

Writing effective documentation notes:

5 tips for writing effective documentation:

- Start with what the reader needs
  - Help them achieve a goal
  - Could be narrow, such as "How do I setup this CLI" or broad like "I'm new to the team, how do I use this whole architecture?"
  - For each feature, ask yourself – "Does the read need to know about that?"
- Write less!
  - This makes it easier for the reader to find what they need
  - Remember – you don't need to cover everything!
  - Write documentation that reflects what you can commit to (don't document things you aren't going to maintain)
  - Make sure it adds value
- Write the outline first
  - Think about the questions from point 1 above – use those as headings for your docs and fill them in
  - This will help structure
- Rubber ducking works for docs too
  - (This is talking it through out loud)
  - We talk more than we write, so talking it out loud can help express it better
  - Can help unblock and make better quality docs
  - A friendly, conversational style will help users take it in more
- Write readably
  - Not just for docs writing!

- Give it structure:



Without structure

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras sed hendrerit metus. Morbi eleifend velit eu tellus ultricies laoreet. Proin tempus, lacus sit amet sodales varius, mi justo ullamcorper leo, id euismod leo felis nec ipsum. Phasellus ipsum metus, bibendum a odio a, consectetur cursus tellus. Fusce fermentum nisl justo, nec facilisis nisi consequat eget. Sed sed justo tempus, viverra nibh sed, tincidunt purus. Proin nec nulla fermentum, sagittis ipsum vel, laoreet neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Curabitur mollis vulputate leo, et tincidunt ante tempor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc lectus elit, vehicula at facilisis vitae, blandit non velit. Proin eu magna in erat rutrum varius. Morbi in varius felis. Suspendisse potenti. Etiam eget mi non elit suscipit pellentesque. Suspendisse ut sem a elit eleifend suscipit blandit in odio. Nam ac leo leo. Fusce consequat id lacus non placerat. Etiam vel arcu in dui varius molestie ut sed nibh. Nunc venenatis molestie ex, in interdum purus sagittis ac. Vestibulum id vehicula dui. Quisque sed nulla vitae tortor egestas aliquam. Aliquam non mauris mi. Mauris non metus tincidunt, venenatis ligula egestas, pharetra.

With structure

Lorem ipsum dolor sit amet
Consectetur adipiscing elit. Cras sed hendrerit metus.

Proin tempus, `lacus sit amet sodales` varius. Mii justo ullamcorper leo, id euismod leo felis nec ipsum. Phasellus `ipsum metus`, bibendum a odio a, consectetur cursus tellus.

Sed sed justo tempus
Viverra nibh sed, tincidunt purus. Proin nec nulla fermentum, sagittis ipsum vel, laoreet neque. Pellentesque habitant:

- `morbi` tristique senectus
- `netus` malesuada
- `fames` ac turpis egestas

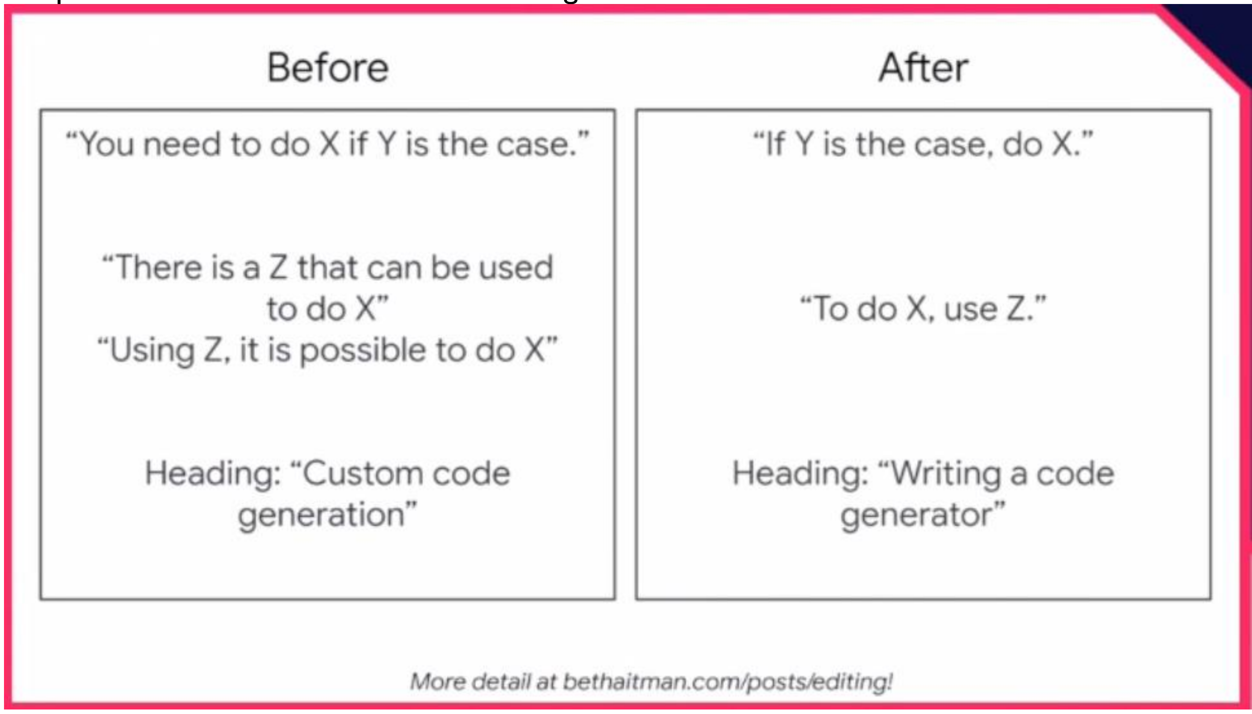
Curabitur mollis vulputate leo
Et tincidunt ante tempor. Pellentesque habitant morbi:

1. Tristique `senectus` et `netus`.
2. Lorem `ipsum` dolor sit amet, consectetur adipiscing elit.
3. Nunc lectus elit, `vehicula at facilisis` vitae.

Proin eu magna in erat rutrum varius - morbi in varius felis.

More detail at bethaitman.com/posts/editing!

- Keep your paragraphs nice and short (1 idea per paragraph)
- Keep sentences to less than 25 words as a guideline

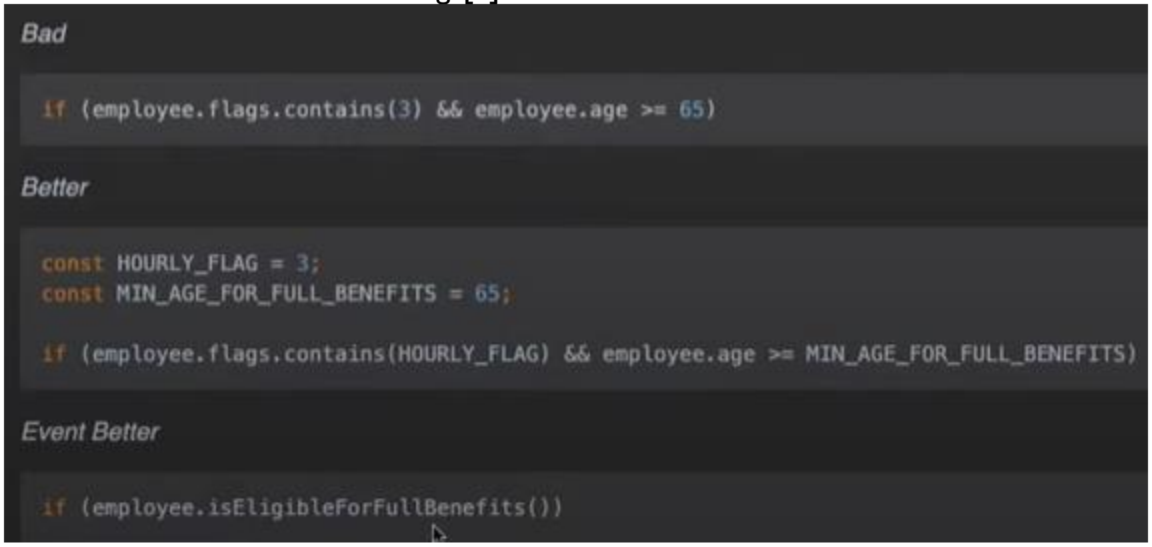


Before	After
"You need to do X if Y is the case."	"If Y is the case, do X."
"There is a Z that can be used to do X" "Using Z, it is possible to do X"	"To do X, use Z."
Heading: "Custom code generation"	Heading: "Writing a code generator"

More detail at bethaitman.com/posts/editing!

- There's a 6th meta point - It's not just about documentation
  - Docs is one part of the process to help the user
  - If you find as you're writing that you're trying to document around a problem, try and fix the thing so the problem isn't there anymore!

5 ways to document your code notes:

- Use the tools your language provides
    - A.k.a Self-documented code (code describes itself)
    - Using the language to express the intent of the code clearly. For example using meaningful:
        - Directory structure
        - File/class/method/variables names
    - Use common standards/patterns that are familiar
    - The main goal for this is to allow devs to understand the code at a glance
    - Examples:
        - Bad: var n = args[0]
            - N means nothing to anyone
        - Better: var username = args[0]
    - 

      Bad

      ```
      if (employee.flags.contains(3) && employee.age >= 65)
      ```

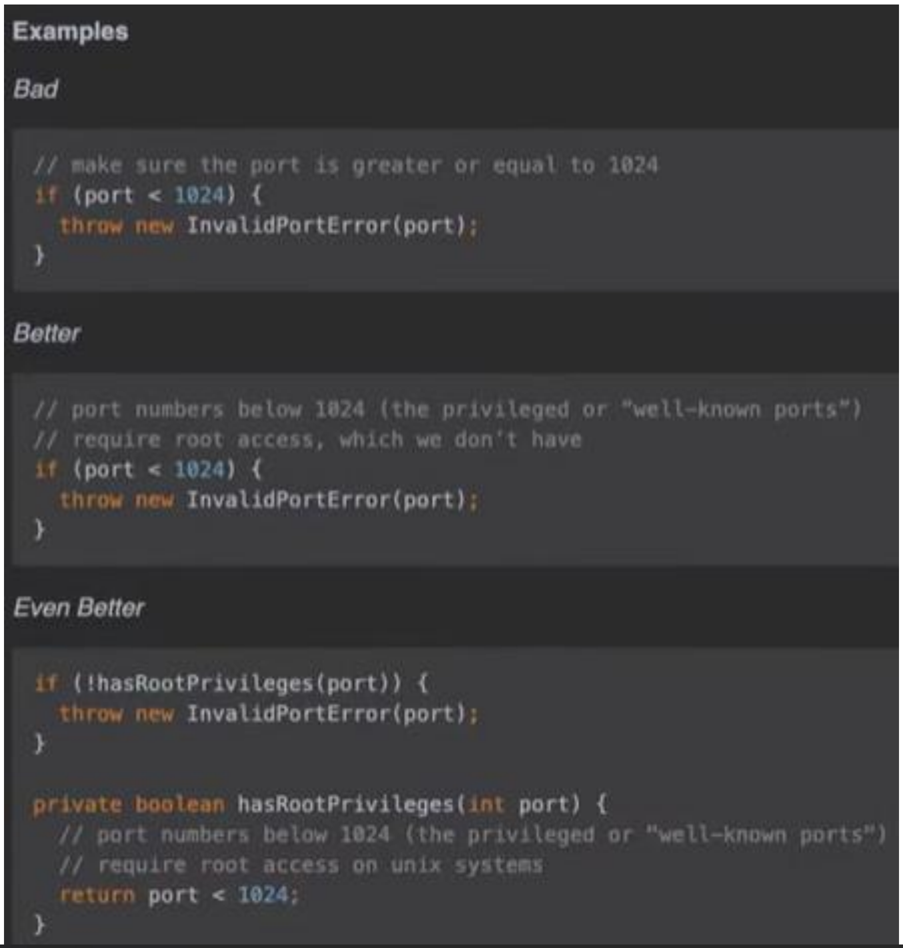
      Better

      ```
      const HOURLY_FLAG = 3;
      const MIN_AGE_FOR_FULL_BENEFITS = 65;

      if (employee.flags.contains(HOURLY_FLAG) && employee.age >= MIN_AGE_FOR_FULL_BENEFITS)
      ```

      Event Better

      ```
      if (employee.isEligibleForFullBenefits())
      ```
- Comments
    - Comments should be used only to explain the intent behind code that cannot be refactored to explain itself
    - Mostly used for providing additional context, i.e. answering the WHY not the WHAT
    - Example:

        - 

Examples

Bad

```
// make sure the port is greater or equal to 1024
if (port < 1024) {
  throw new InvalidPortError(port);
}
```

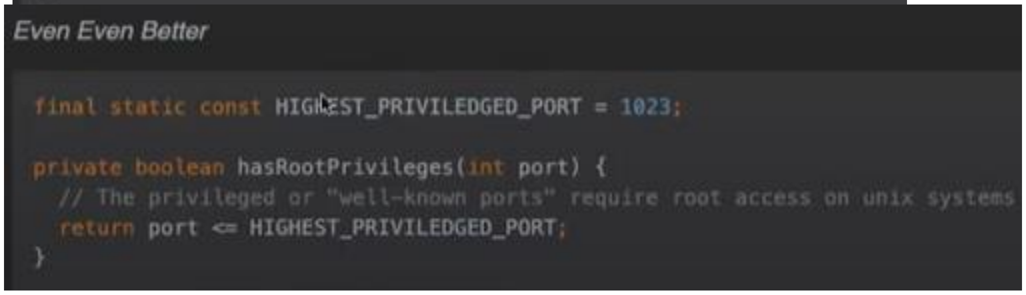
Better

```
// port numbers below 1024 (the privileged or "well-known ports")
// require root access, which we don't have
if (port < 1024) {
  throw new InvalidPortError(port);
}
```

Even Better

```
if (!hasRootPrivileges(port)) {
  throw new InvalidPortError(port);
}

private boolean hasRootPrivileges(int port) {
  // port numbers below 1024 (the privileged or "well-known ports")
  // require root access on unix systems
  return port < 1024;
}
```

        - 

Even Even Better

```
final static const HIGHEST_PRIVILEDGED_PORT = 1023;

private boolean hasRootPrivileges(int port) {
  // The privileged or "well-known ports" require root access on unix systems
  return port <= HIGHEST_PRIVILEDGED_PORT;
}
```

- Tests
    - Tests document how the code should behave, not just that it “does what it does”
    - Test are examples of how to use an API, with assertions
    - Tests usually also give examples of edge case scenarios
- Git Commit Log
    - Most misunderstood
    - Gives more context about why a change is going into the codebase
    - Answers the “why is this done like that??” question
    - Flexible in terms of the amount of content, lives with the code with no clutter
    - Commit message can never get out of date
    - How to write a git commit: https://chris.beams.io/posts/git-commit/
- Readme and Wikis

    - Everything that is not directly code-related lives here (e.g. architecture, structure, how user can use it etc.)
    - Higher level context such as structure of code, coding standards, design choices etc.
    - This can help you better navigate the different components of the codebase

We were then tasked with documenting our projects from last week.

Other resources provided by Arshi in Slack that we haven't visited:

- Atlassian - Building Better Documentation
- Write the Docs Guide
- README Editor with Template Sections
- How to Write a Good Documentation: Home
- Best Practices When Documenting Your Code for Software Engineers
- Mastering Markdown