Architecture: Design Docs

17-313 Fall 2023

Foundations of Software Engineering

https://cmu-313.github.io

Michael Hilton and Eduardo Feo Flushing

Administrivia

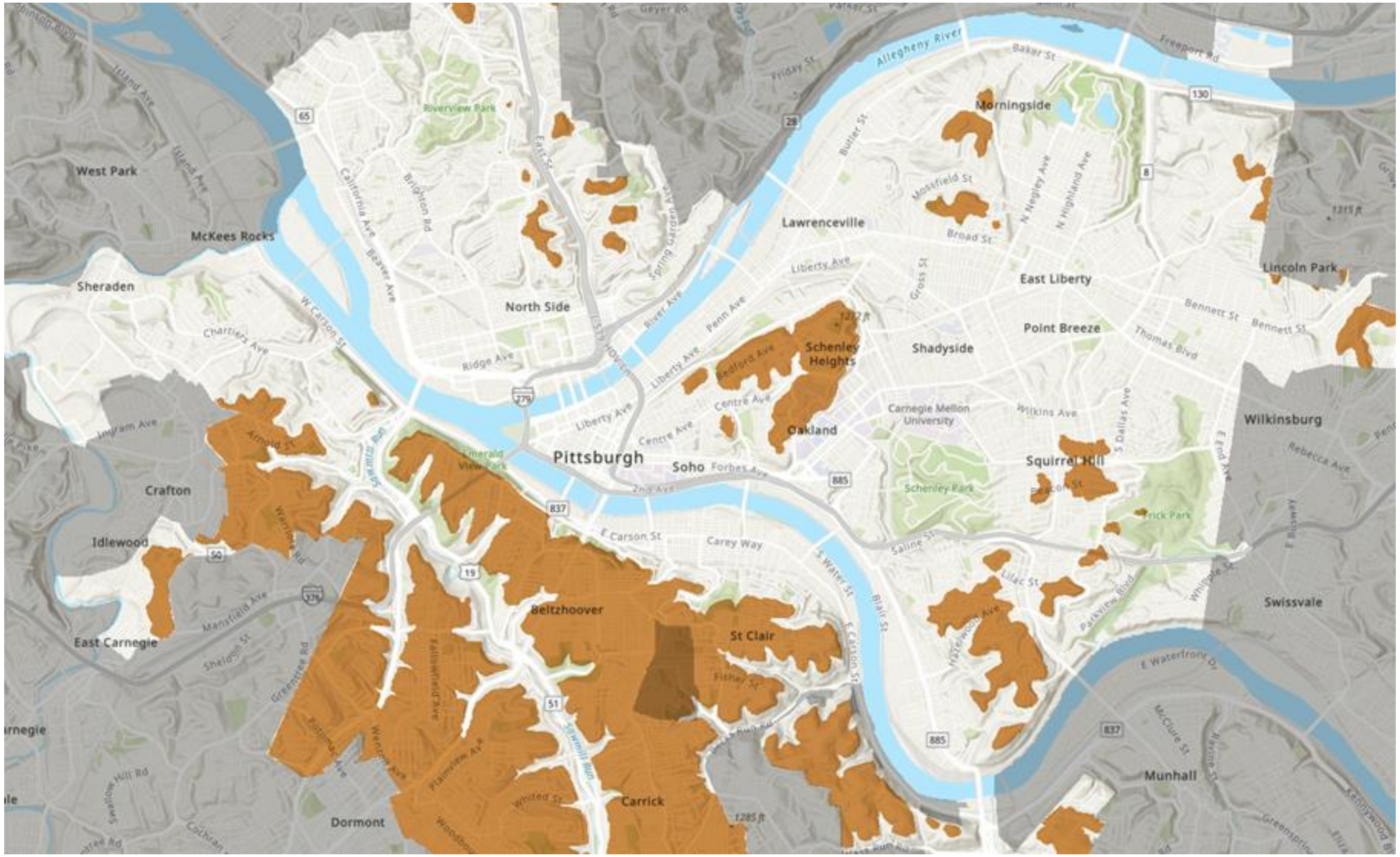
- Teamwork assessments due every Friday.
- Happy Lunar New Year!
- Happy Super Bowl Weekend!

Learning Goals

- Articulate the various purposes of a design document.
- Use design documentation to ensure that the correct thing is being implemented.
- Write useful, clear, high-quality design documentation.

Smoking Section

- Last **two** full rows

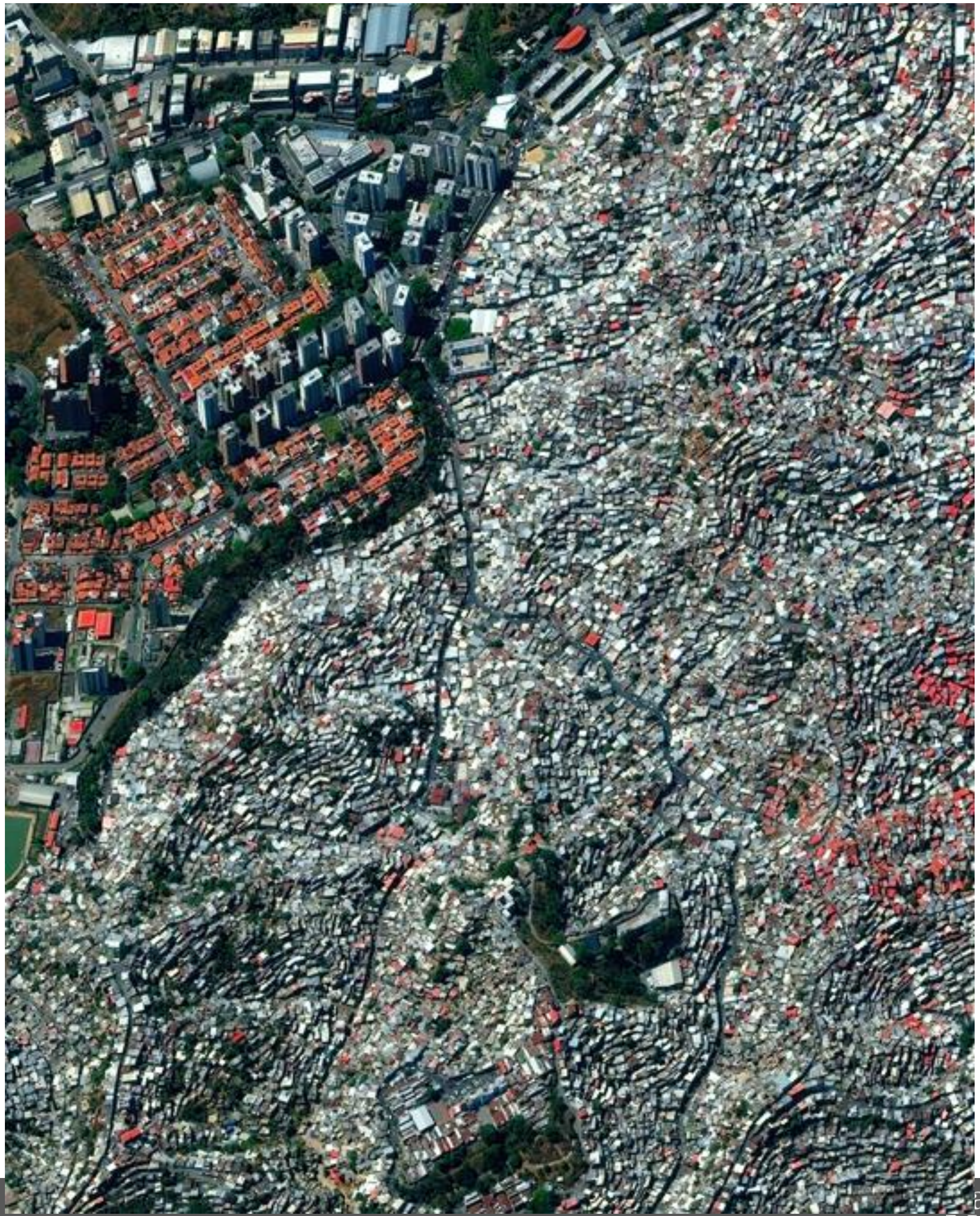


Source: Pittsburgh Zoning Map (https://gis.pittsburghpa.gov/pghzoning/)

https://www.archdaily.com/

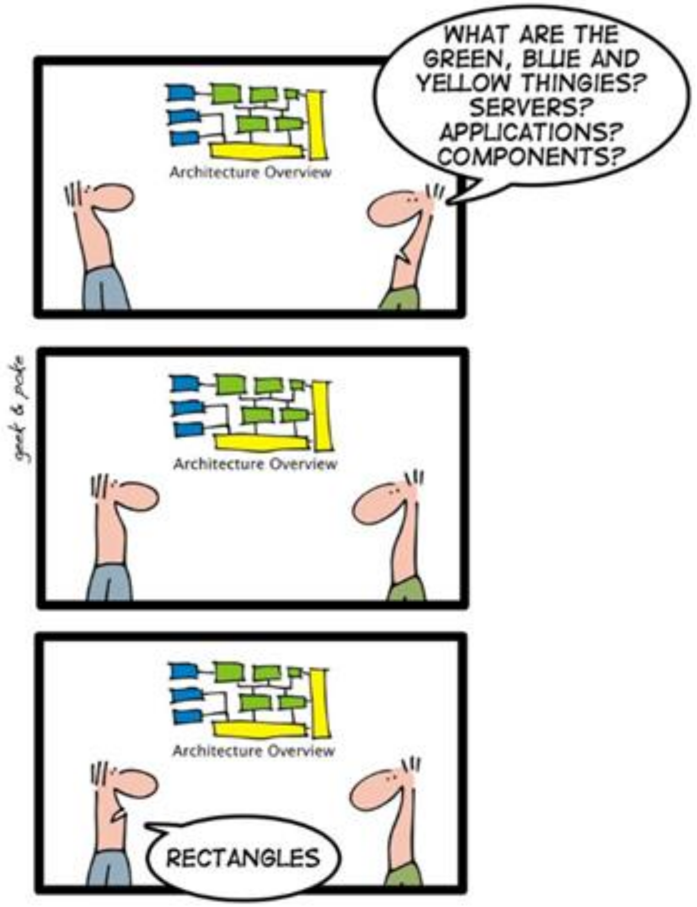
https://www.instagram.com/architectanddesign

https://www.mykonosceramica.com/


www.over-view.com

Guidelines for selecting a notation

- Suitable for purpose
- Often visual for compact representation
- Usually, boxes and arrows
- UML possible (semi-formal), but possibly constraining
  - Note the different abstraction level – Subsystems or processes, not classes or objects
- Formal notations available
- Decompose diagrams hierarchically and in views
- Always include a legend
- Define precisely what the boxes mean
- Define precisely what the lines mean
- Do not try to do too much in one diagram
  - Each view of architecture should fit on a page
  - Use hierarchy


WHAT ARE THE GREEN, BLUE AND YELLOW THINGIES? SERVERS? APPLICATIONS? COMPONENTS?
Architecture Overview
geek & poke
RECTANGLES

Example:

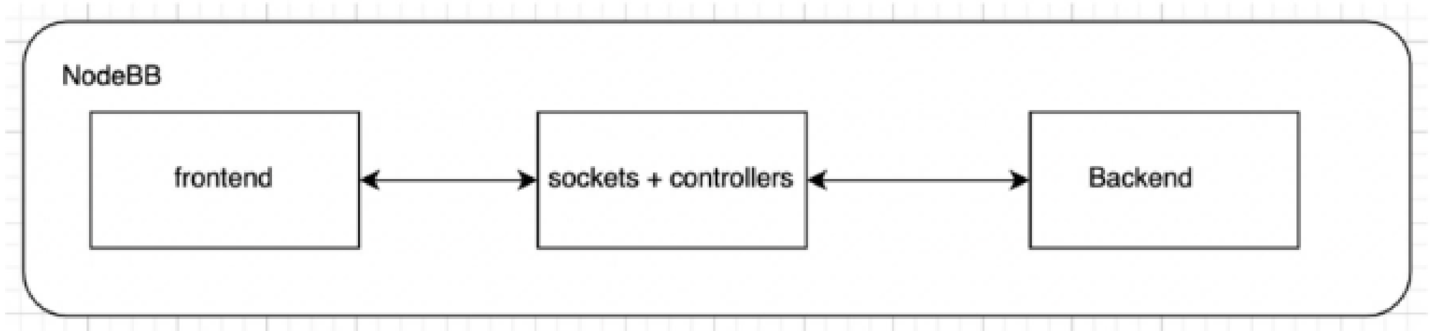
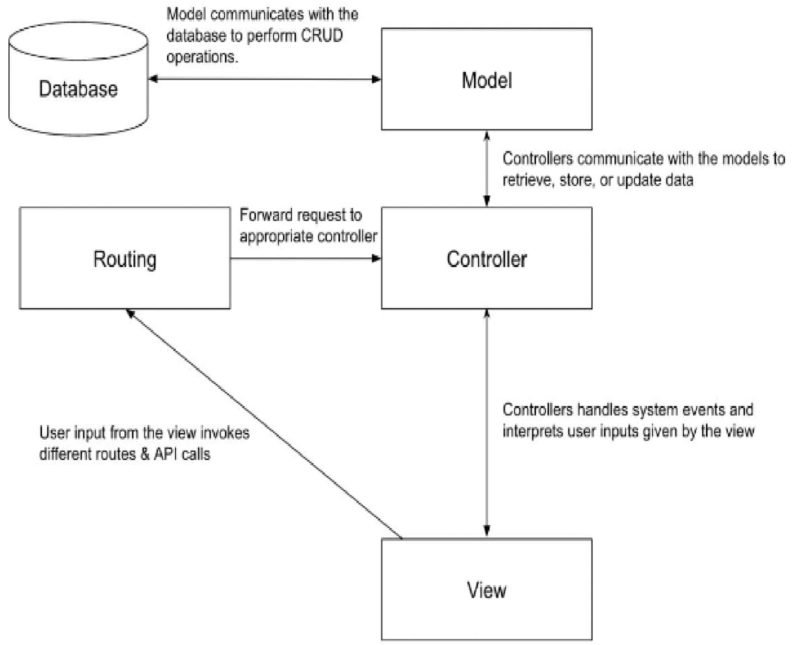
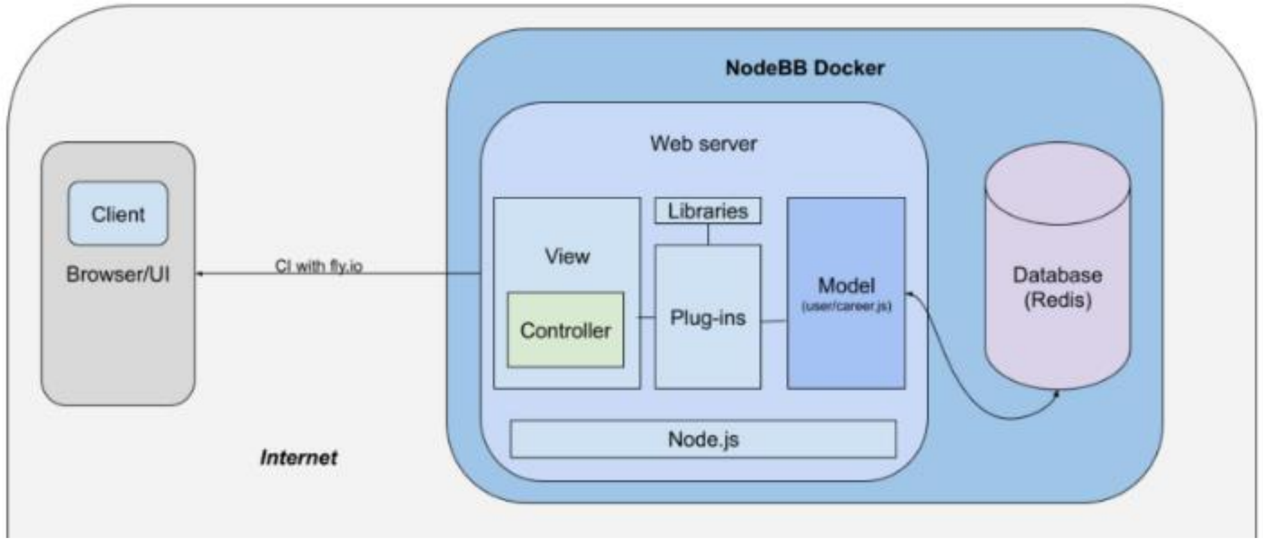


Figure 1: Architecture Diagram of Current State of NodeBB


Database
Model communicates with the database to perform CRUD operations.
Model
Controllers communicate with the models to retrieve, store, or update data
Routing
Forward request to appropriate controller
Controller
User input from the view invokes different routes & API calls
Controllers handles system events and interprets user inputs given by the view
View

VS



Types of documentation

- Reference documentation (incl. code comments)
- Design documents
- Tutorials
- Conceptual documentation
- Landing pages

O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
Curated by Titus Winters, Tom Manshreck & Hyrum Wright

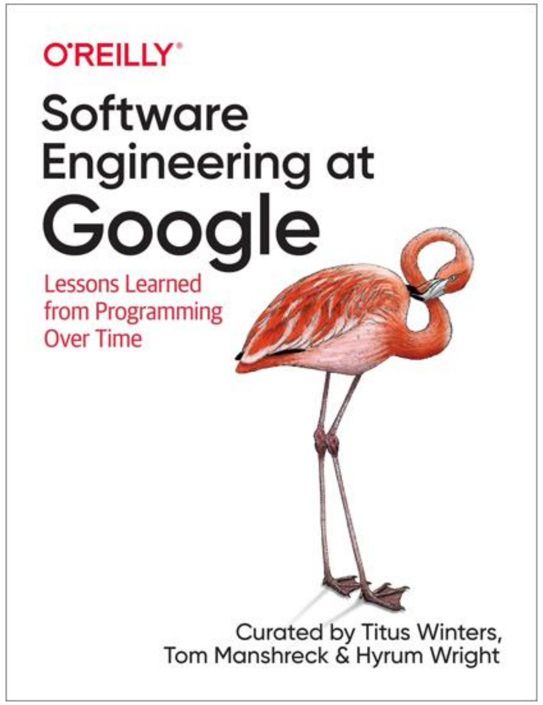
Design documents

- **Code review before there is code!**
- Collaborative (Google Docs)
- Ensure various concerns are covered, such as: security implications, internationalization, storage requirements, and privacy concerns.
- A good design doc should cover
  - Goals and use cases for the design
  - Implementation ideas (not too specific!)
  - Propose key design decisions with an emphasis on their individual tradeoffs

O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
Curated by Titus Winters, Tom Manshreck & Hyrum Wright

Design Documents

- The *best* design docs suggest design goals, and cover alternative designs, documenting the strengths and weaknesses of each.
- The *worst* design docs accidentally embed ambiguities, which cause implementors to develop contradictory solutions that the customer doesn't want.


O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
Curated by Titus Winters, Tom Manshreck & Hyrum Wright

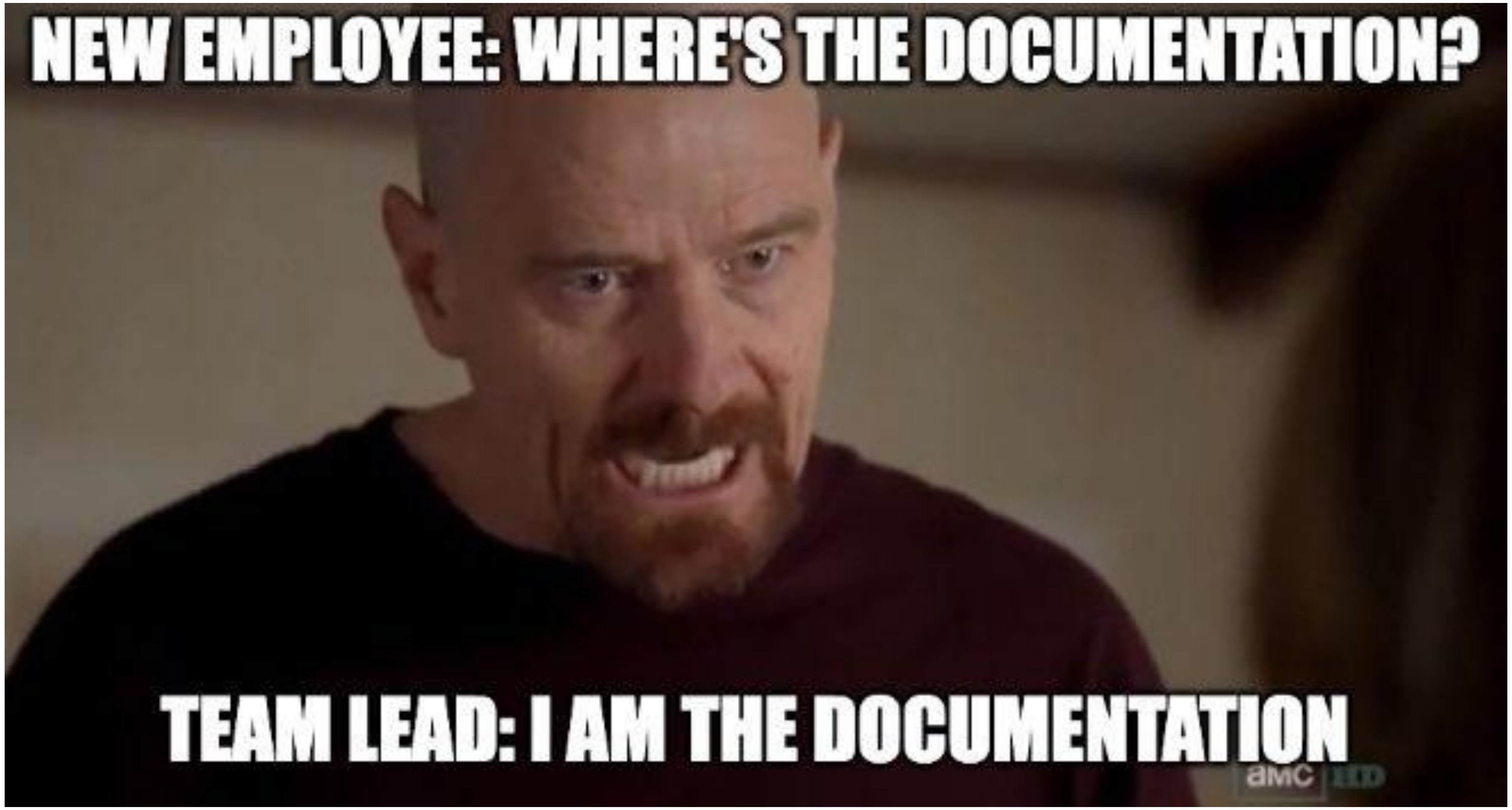
Companies using an RFC-like engineering planning process*

- Airbnb
- Affirm
- Algolia
- Amazon
- AutoScout24
- Asana
- Atlassian
- Blue Apron
- Bitrise
- Booking.com
- Brex
- BrowserStack
- Canonical
- Carousell
- Catawiki
- Cazoo
- Cisco
- CockroachDB
- Coinbase
- Comcast Cable
- Container Solutions
- Contentful
- Couchbase
- Criteo
- Curve
- Daimler
- Delivery Hero
- Doctolib
- DoorDash
- Dune Analytics
- eBay
- Ecosia
- Elastic
- Expedia
- Glovo
- Gojek
- Grab
- Faire
- Flexport
- GitHub
- GitLab
- GoodNotes
- Google
- Grafana Labs
- GrubHub
- HashiCorp
- Hopin
- Hudl
- Indeed
- Intercom
- LinkedIn
- Kiwi.com
- Klarna
- MasterCard
- Mews
- MongoDB
- Monzo
- Mollie
- Miro
- N26
- Netlify
- Nobl9
- Notion
- Nubank
- Oscar Health
- Octopus Deploy
- OLX
- Onfido
- Pave
- Peloton
- Picnic
- PlanGrid
- Preply
- Razorpay
- Reddit
- Red Hat
- SAP
- Salesforce
- Shopify
- Siemens
- Spotify
- Square
- Stripe
- Synopsys
- Skyscanner
- SoundCloud
- Sourcegraph
- Spotify
- Stedi
- Stream
- SumUp
- Thumbtack
- TomTom
- Trainline
- TrueBill
- Trustpilot
- Twitter
- Uber
- VanMoof
- Virta Health
- VMWare
- Wayfair
- Wave
- Wise
- WarnerMedia & HBO
- Zalando
- Zapier
- Zendesk
- Zillow

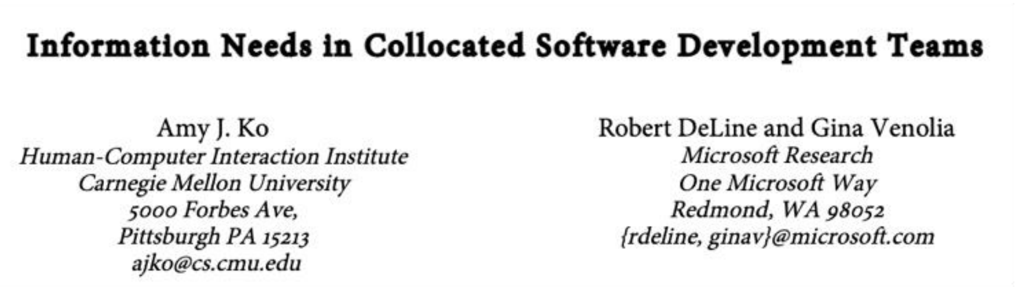
*not a complete list

pragmaticengineer.com

Why is this important?

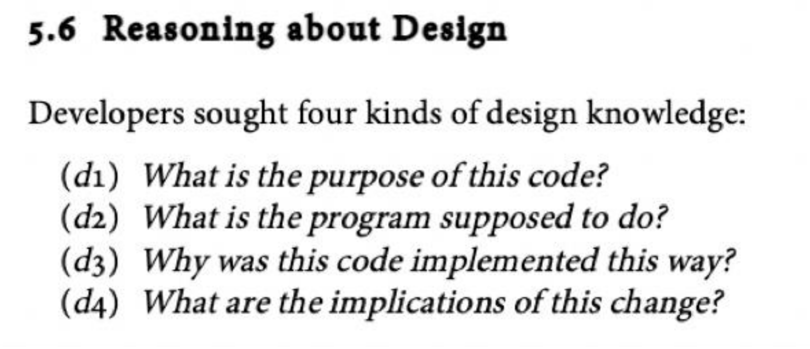

NEW EMPLOYEE: WHERE'S THE DOCUMENTATION?
TEAM LEAD: I AM THE DOCUMENTATION
amc HD

Lots of evidence this is hard


Information Needs in Collocated Software Development Teams

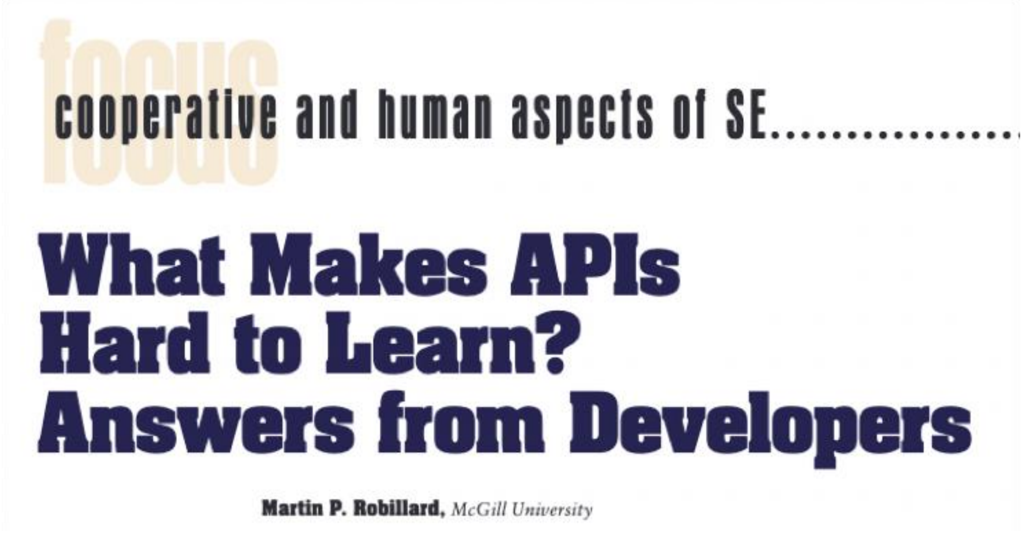
Amy J. Ko
Human-Computer Interaction Institute
Carnegie Mellon University
5000 Forbes Ave,
Pittsburgh PA 15213
ajko@cs.cmu.edu

Robert DeLine and Gina Venolia
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{rdeline, ginav}@microsoft.com


5.6 Reasoning about Design

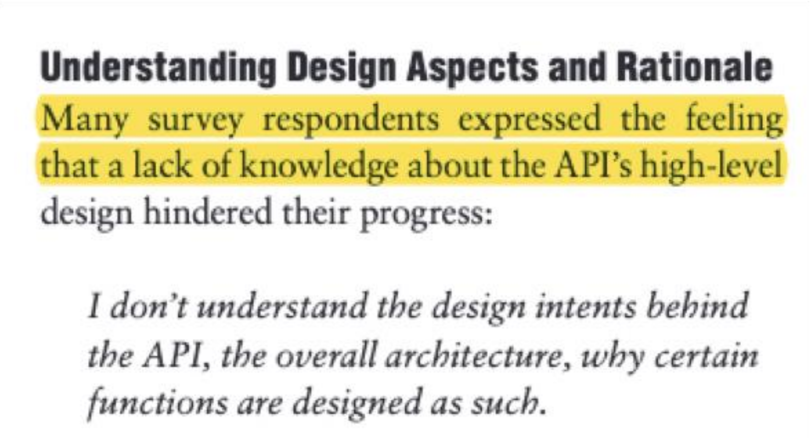
Developers sought four kinds of design knowledge:

(d1) *What is the purpose of this code?*
(d2) *What is the program supposed to do?*
(d3) *Why was this code implemented this way?*
(d4) *What are the implications of this change?*


focus
cooperative and human aspects of SE................

What Makes APIs Hard to Learn? Answers from Developers

Martin P. Robillard, McGill University


Understanding Design Aspects and Rationale

Many survey respondents expressed the feeling that a lack of knowledge about the API's high-level design hindered their progress:

I don't understand the design intents behind the API, the overall architecture, why certain functions are designed as such.

Common parts/templates

1. Metadata: *version, date, authors*
2. Executive Summary: *problem being solved, project mission*
3. Stakeholders (and non-stakeholders)
4. Scenarios / User Stories
5. User Experience

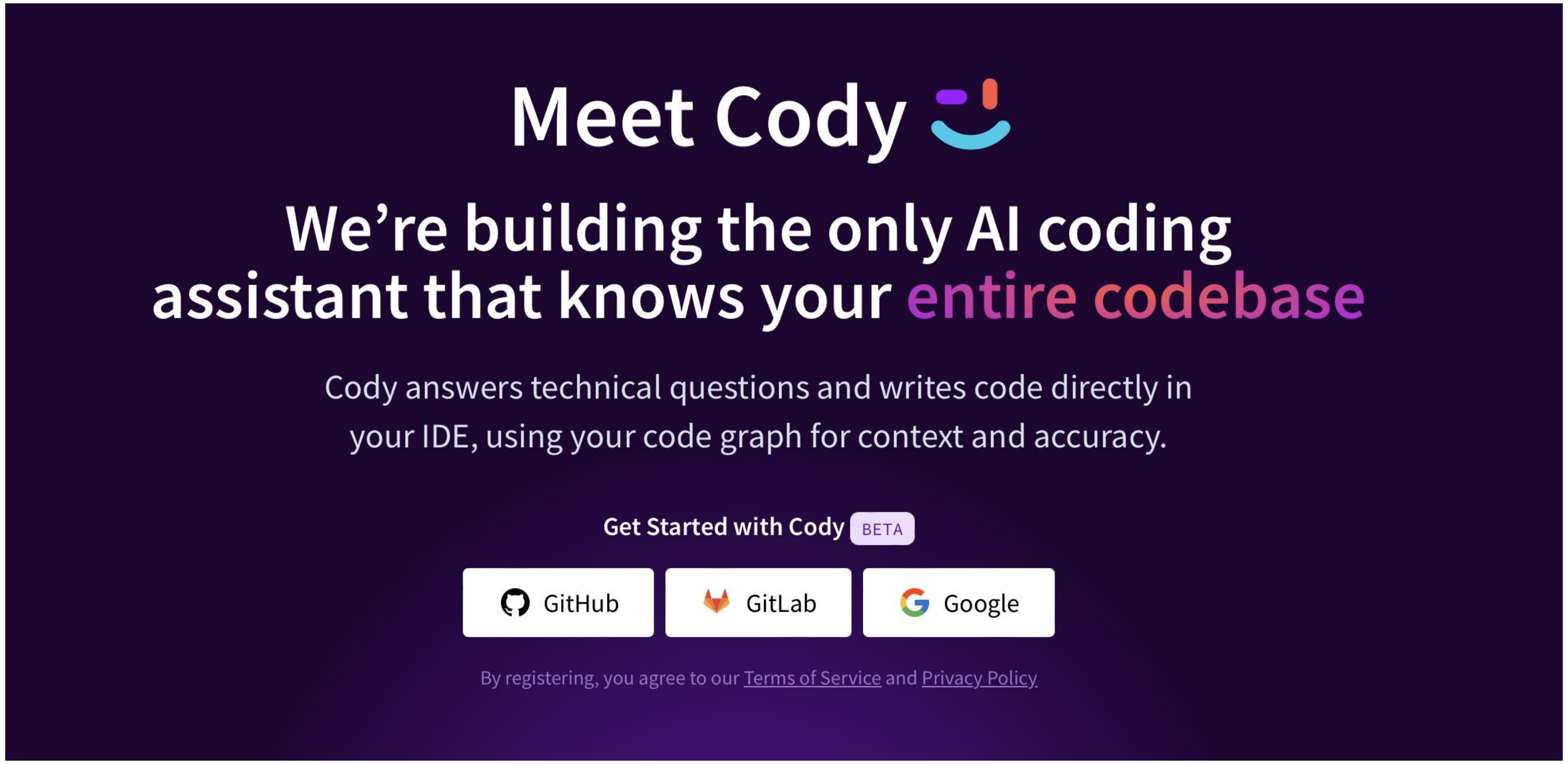
1. High-level Requirements: *Functional*
   - Global Requirements: *Quality, Security, Privacy, Ethics*
2. Features and Operations
3. Design Considerations and Tradeoffs
4. Non-Goals
5. Roadmap / Timeline
6. Open Issues

Examples: SourceGraph RFCs

Requests for Comment

https://about.sourcegraph.com/handbook/communication/rfcs

Sourcegraph


Meet Cody

We're building the only AI coding assistant that knows your entire codebase

Cody answers technical questions and writes code directly in your IDE, using your code graph for context and accuracy.

Get Started with Cody BETA
GitHub
GitLab
Google
By registering, you agree to our Terms of Service and Privacy Policy

When to use an RFC:

Sourcegraph

- You want to frame a problem and propose a solution.
- You want thoughtful feedback from team members on our globally-distributed remote team.
- You want to surface an idea, tension, or feedback.
- You want to define a project or design brief to drive project collaboration.
- You need to surface and communicate around a highly cross-functional decision with our formal decision-making process.

Don’t use an RFC when

Sourcegraph

- You want to discuss personal or sensitive topics one-on-one with another team member.
- You want to make a decision to change something where you are the decider. In the vast majority of cases, creating an RFC to explain yourself will be overkill. RFCs should only be used if a decision explicitly requires one of the bullets in the previous page.

RFC Labels

Sourcegraph

- **WIP**: The author is still drafting the RFC and it's not ready for review.
- **Review**: The Review label is used when the RFC is ready for comments and feedback.
- **Approved**: When the RFC is for the purpose of making a decision, the Approved label indicates that the decision has been made.
- **Implemented**: When the RFC is for the purpose of making a decision, the Implemented label indicates that the RFC's proposal has been implemented.
- **Closed**: When the RFC is for the purpose of collaboration or discussion but not necessarily to make a decision or propose a specific outcome that will eventually become Implemented, the Closed label indicates that the RFC is no longer an active collaborative artifact.
- **Abandoned**: When the RFC is for the purpose of making a decision, and there are no plans to move forward with the RFC's proposal, the Abandoned label indicates that the RFC has been purposefully set aside.

Observe Sourcegraph Design Docs

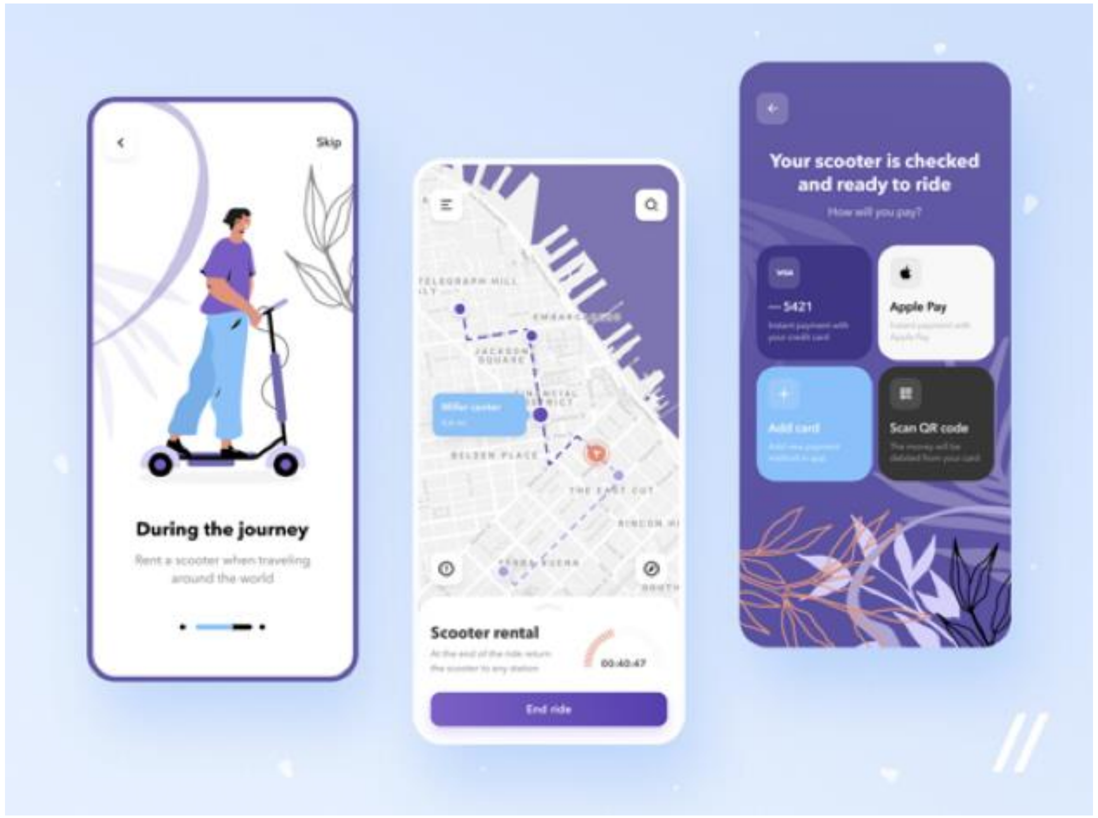
- Docs are publicly available
  https://drive.google.com/drive/folders/1zP3FxdDlcSQGC1qvM9lHZRaHH4I9Jwwa

- Let's take a look at one!

Exercise

4 Proposed features:

- Add Payment Methods
- More Secure Authentication
- Add Android Support
- Internationalization (i18n)



Time to write our own design docs!

- Divide up into 4 teams.
- Your mission:
  - Brainstorm a feature to add to a scooter app and write a design spec, together!
  - Review the design doc, collaborate around text
  - Review another team's design doc, ask questions/leave comments