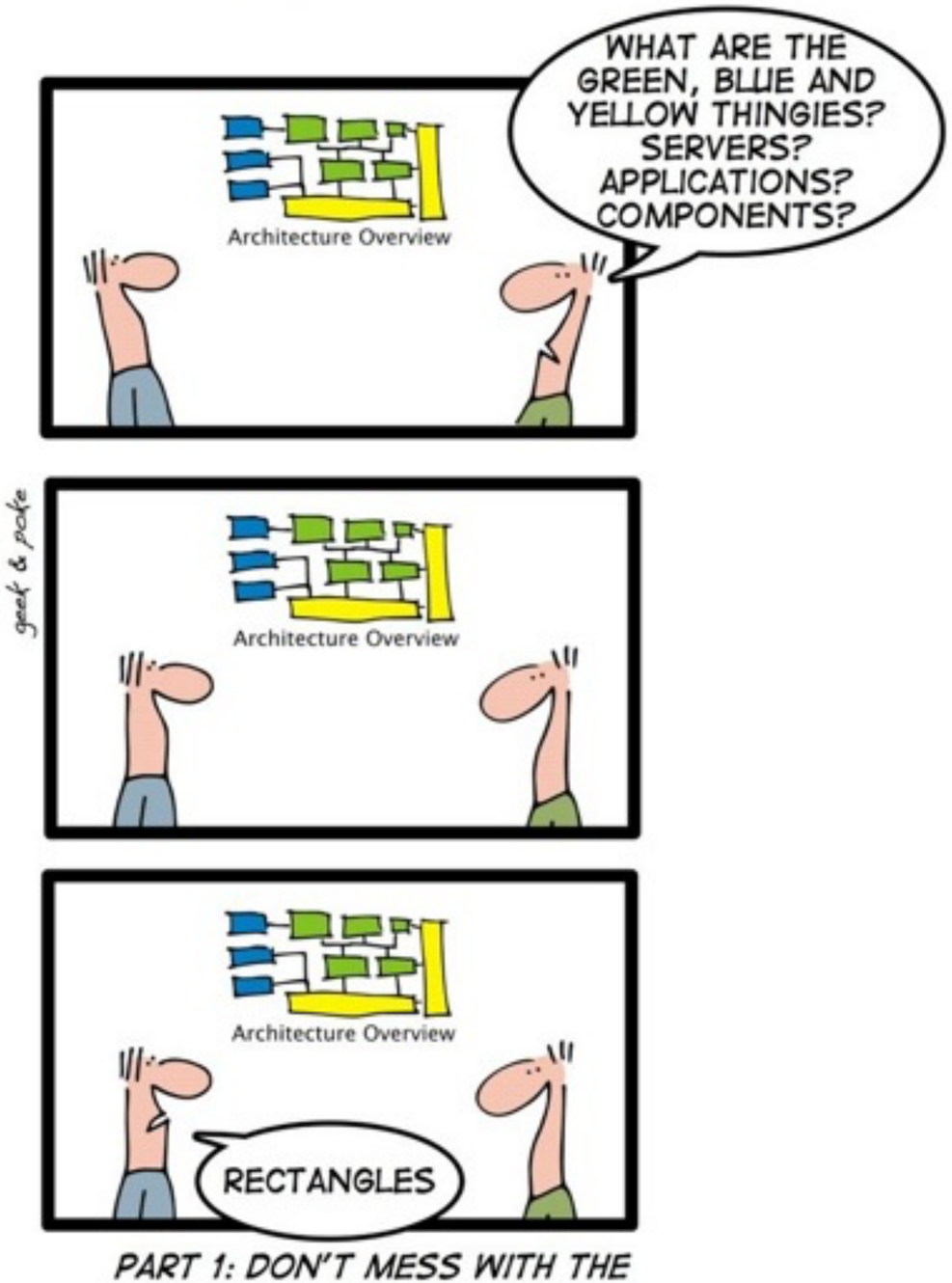
Introduction to Software Architecture: Views

Michael Coblenz

ENTEPRISE ARCHITECTURE MADE EASY

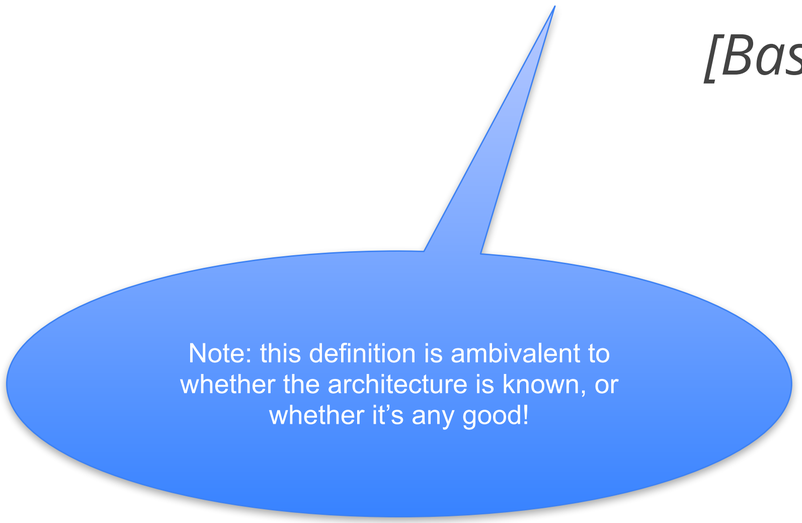


PART 1: DON'T MESS WITH THE GORY DETAILS

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Bass et al. 2003]



Reminder: Two Kinds of Requirements

- Functional requirements: what the system should do
  - "The system shall enable the user to read email."
  - Generally, these are either met or not met (if not met, the system is unacceptable)
- Quality attributes: the degree to which the software works as needed
  - "The system shall fetch 1 GB of email in under 1 minute."
  - Sometimes called "non-functional requirements"
  - Maintainability, modifiability, performance, reliability, security
  - Generally, these can be achieved in degrees

Goal: Meet Quality Requirements

- Maintainability / Modifiability
- Performance
- Scalability
- Availability
- Usability

Key lesson: software architecture is about selecting a design that meets the desired quality attributes.

Abstraction

- Goal: Reason without understanding implementation details
- Approach:
  - Divide enormous system into smaller pieces
  - Define what those pieces do and how they relate to each other

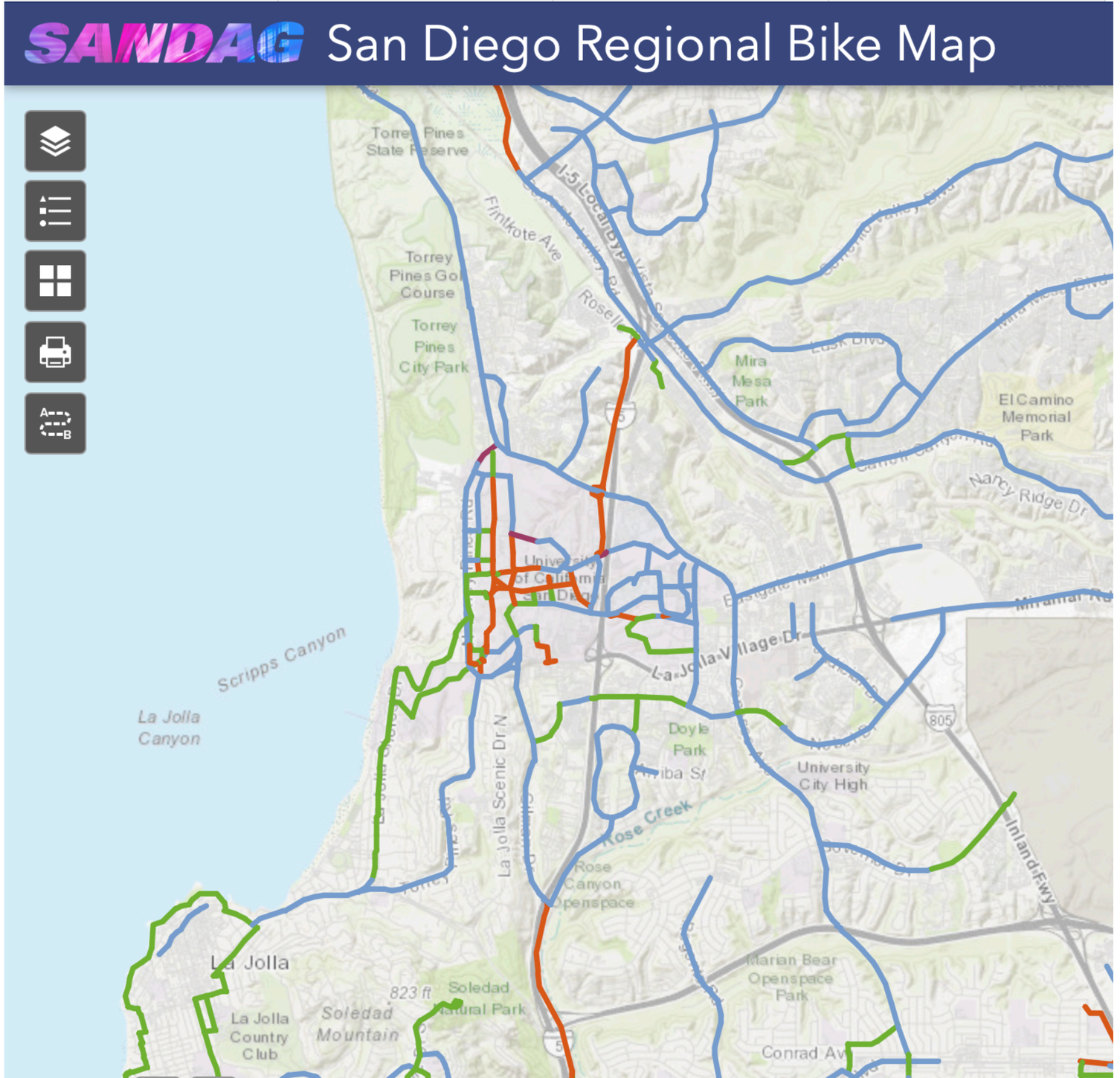
Considerations for Decomposition

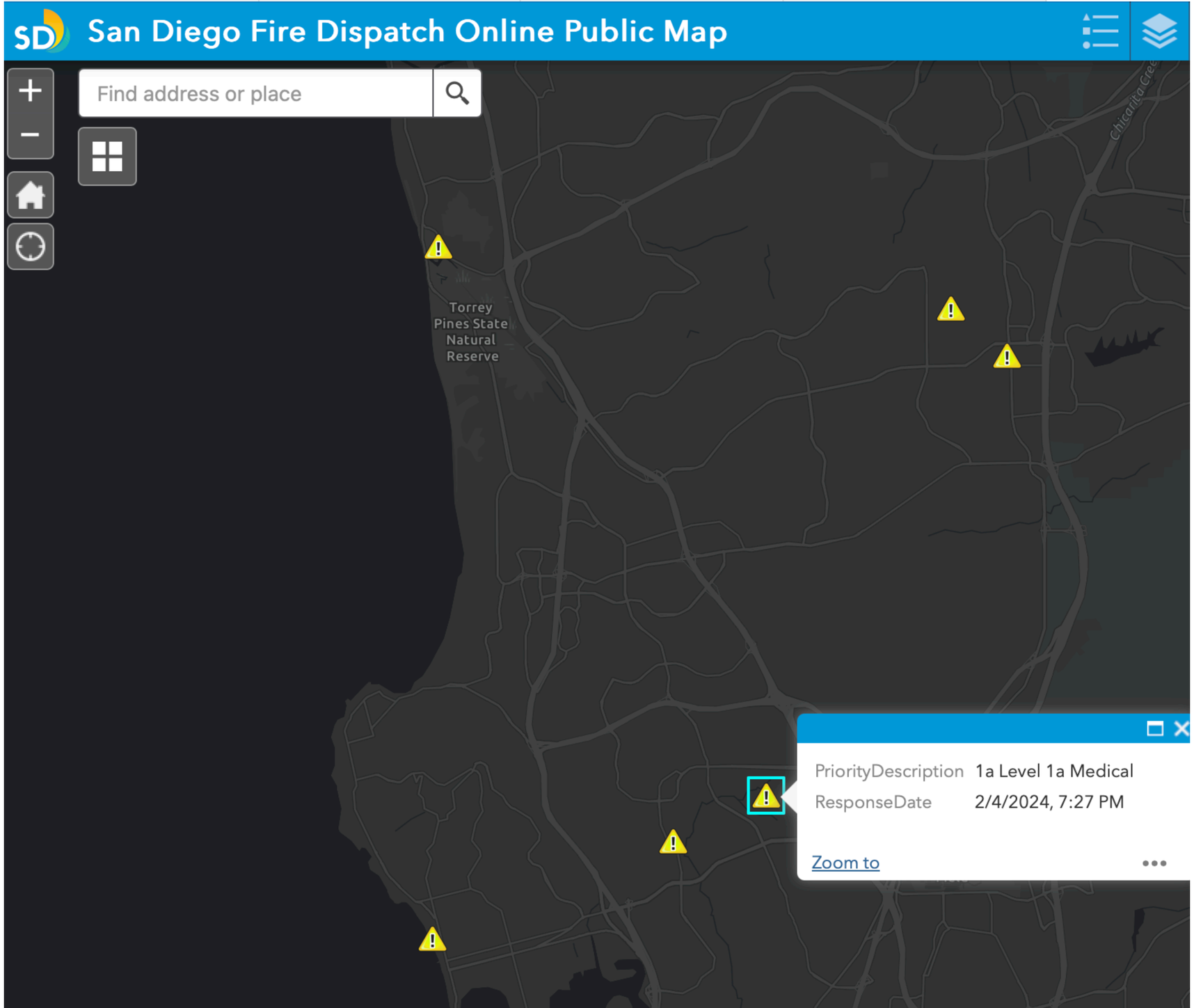
- Conceptual integrity (each piece is responsible for one "thing")
- Conway's Law (organizations inevitably produce copies of their org charts)
- Minimize coupling (avoid entangling separate modules)
- Maximize cohesion (everything in a module fits the theme)
- Use known-good solutions for prioritized quality attributes (see book)

Views

- Once you have a design, how do you draw it?

SANDAG San Diego Regional Bike Map




San Diego Fire Dispatch Online Public Map
Find address or place
Torrey Pines State Natural Reserve
PriorityDescription 1a Level 1a Medical
ResponseDate 2/4/2024, 7:27 PM
Zoom to

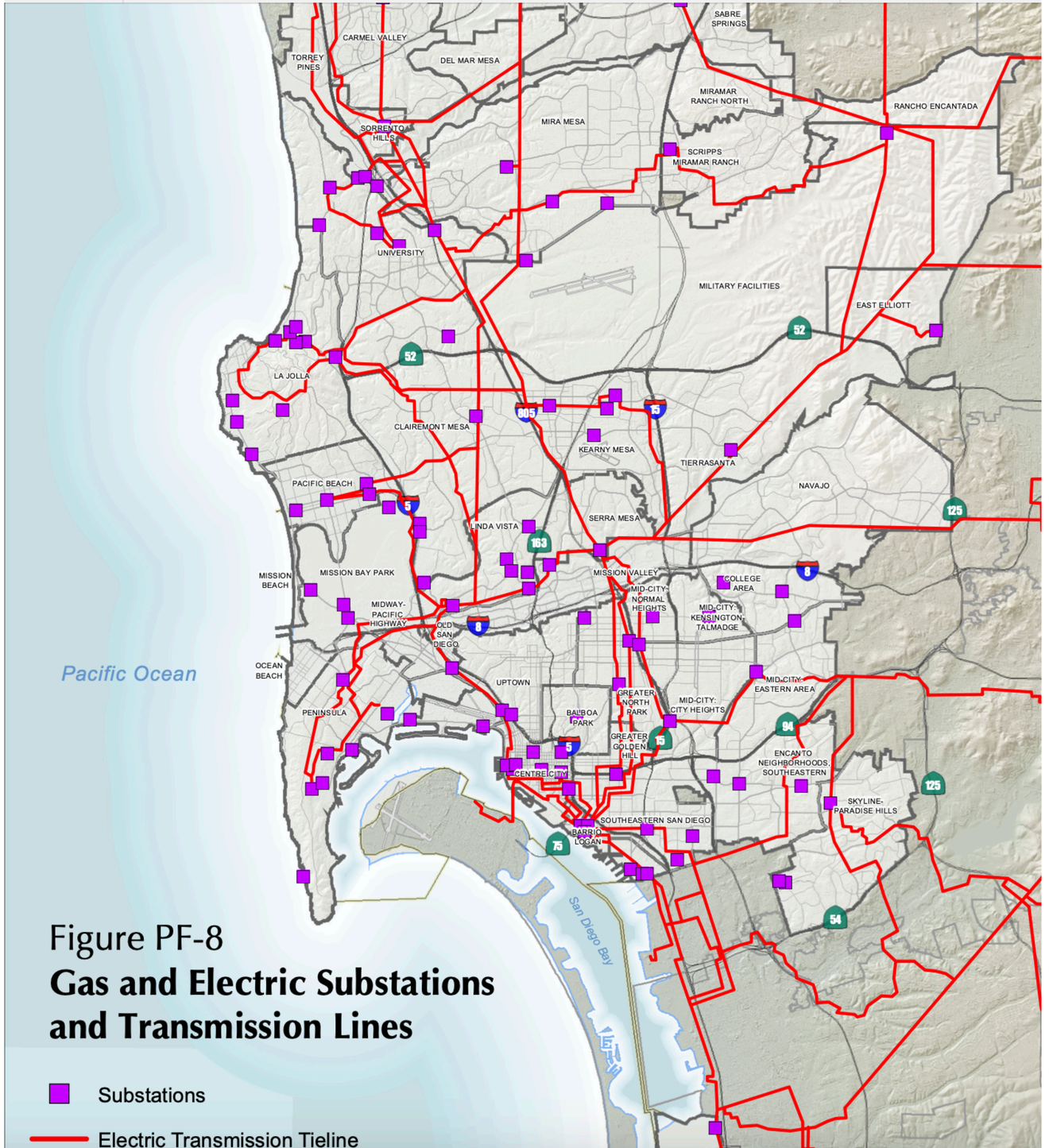


Figure PF-8

Gas and Electric Substations and Transmission Lines

Views and Purposes

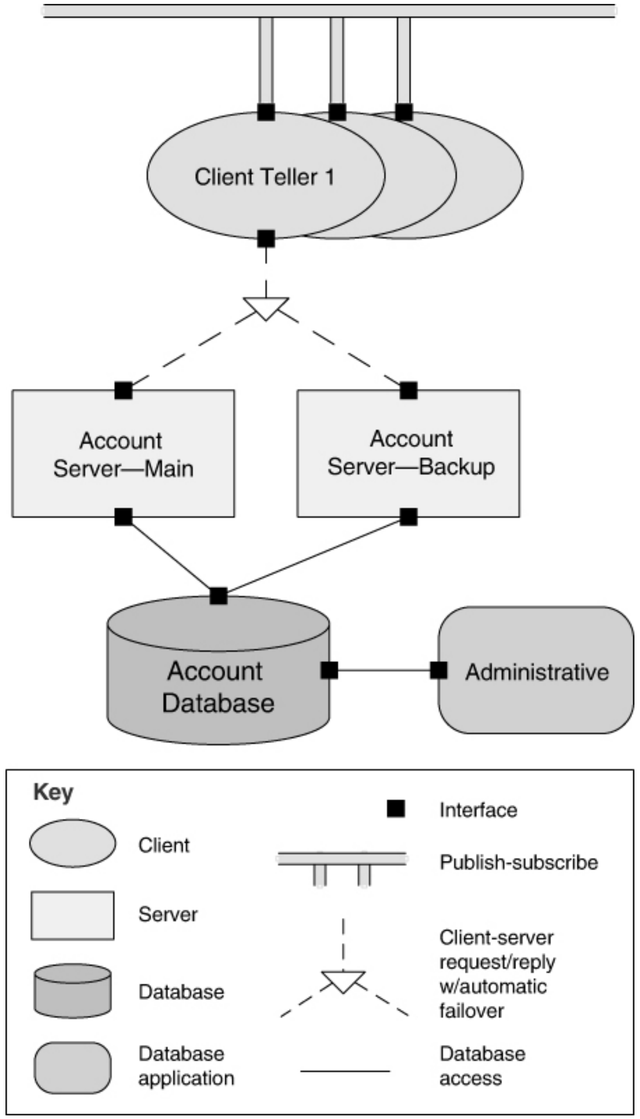
- Every view should align with a purpose
- Views should only represent information relevant to that purpose
  - Abstract away other details
  - Annotate view to guide understanding where needed
- Different views are suitable for different reasoning aspects (different quality goals), e.g.,
  - Performance
  - Extensibility
  - Security
  - Scalability
  - ...

Architectural Structures

- Three kinds of structures:
  - Components and connectors (runtime entities)
  - Modules (static entities)
  - Allocations (mapping of software to the real world)
- Each type has its own kind of *view*

Components and Connectors (for *run time* entities)

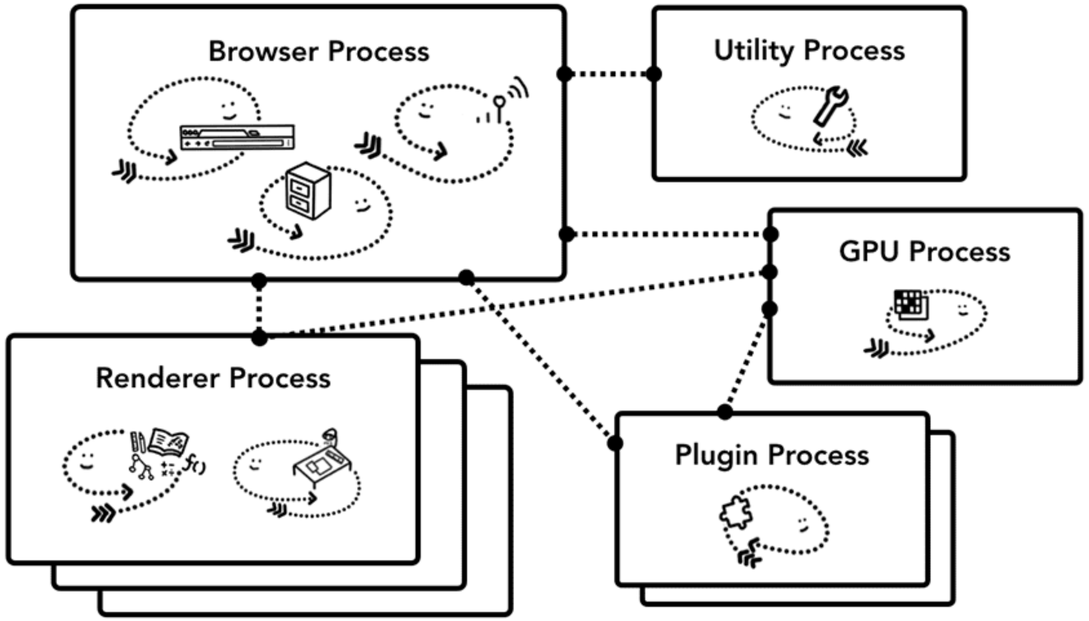
- These show:
  - Major executing components and interactions
  - Major shared data stores
  - Replicas
  - How data progresses through system
  - Which parts run in parallel
  - How structure can change at run time



Credit: *Software Architecture in Practice*

Component views (dynamic)

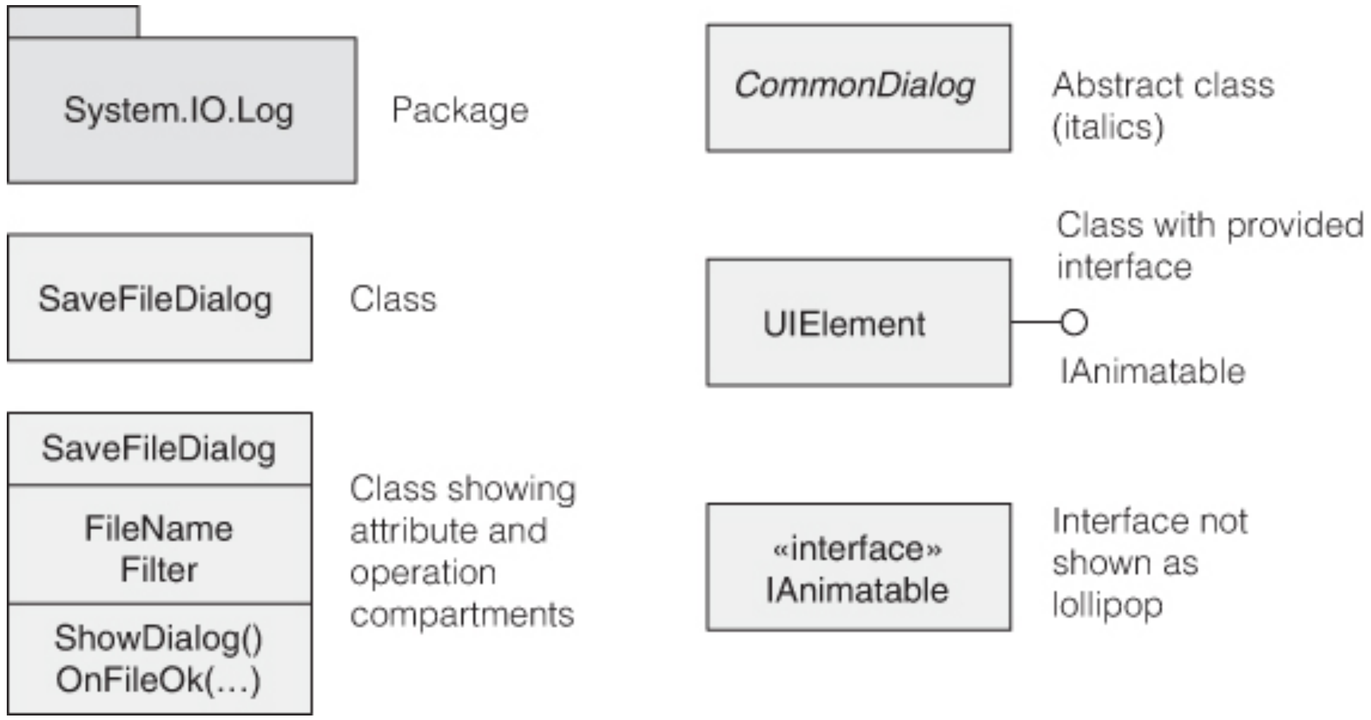
- Shows entities that exist at *run time*
- Components (processes, runnable entities) and connectors (messages, data flow, ...)
- These do not exist until the program runs; cannot be shown in a static view



https://developer.chrome.com/blog/inside-browser-part1

Module Structures (for *static* entities)

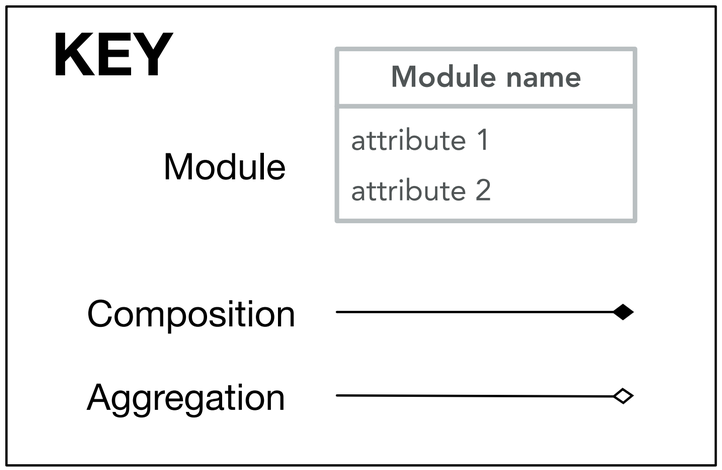
- Show how responsibilities are held by *code* structures
- Packages, classes, layers

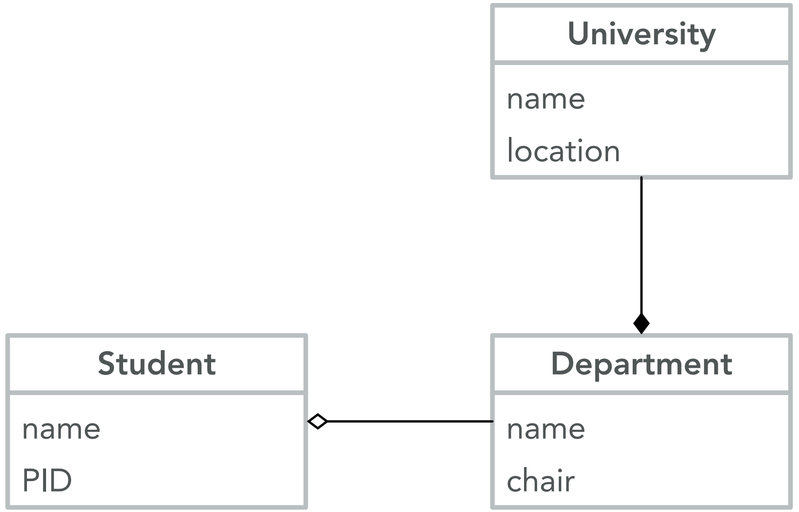


Credit: *Software Architecture in Practice*

Module views (*static*)

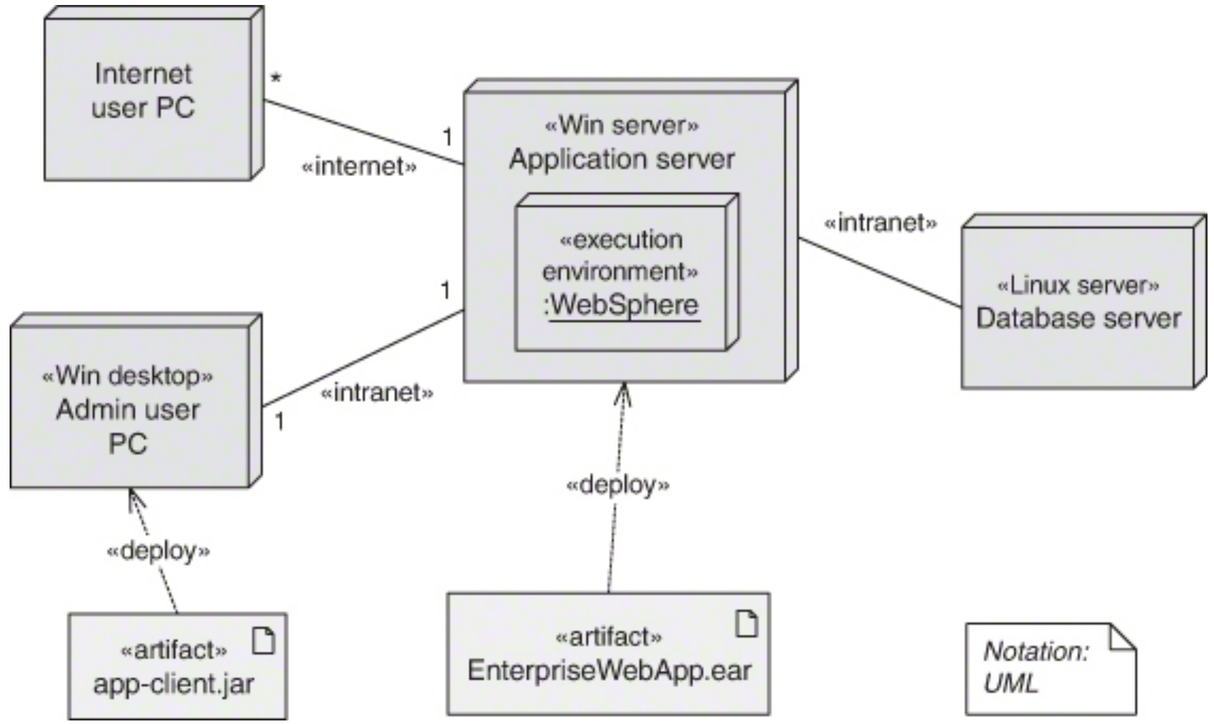
- Shows structures that are defined by the code
- Modules (subsystems, structures) and their relations (dependencies, ...)
- Often shows *decompositions* (a University consists of Departments) and *uses* (a Course uses a Classroom)





Allocation Views (relate different kinds of dynamic components)

- Example: deployment view shows how software artifacts are deployed on servers



Credit: *Software Architecture in Practice*

Physical view (deployment)

- Hardware structures and their connections
- Which parts of the system run on which physical machines?
- How do those machines connect?

Why Document Architecture?

- Blueprint for the system
  - Artifact for early analysis
  - Primary carrier of quality attributes
  - Key to post-deployment maintenance and enhancement
- Documentation speaks for the architect, today and 20 years from today
  - As long as the system is built, maintained, and evolved according to its documented architecture
- Support traceability.

Software Architectural Styles

- A style describes a family of architectures
- Each style promotes some quality attributes and inhibits others
- Learning these patterns can enable you to make good architectural choices
- Important: "pure" styles rarely occur in practice
- But we can study them as if they were pure so we can focus on them individually
- Each style includes:
  - **Components** or **modules**
  - **Connectors** that describe relationships between components or modules

Styles for your assignment

- You will choose some quality attributes to prioritize.
- Look in the book (see link in assignment) for guidance on what techniques to use for the things you're focusing on.
- Examples:
  - To promote *modifiability*, divide into modules so that the changes you have in mind will affect a small number of modules
  - To promote *availability,* use replicas (if one goes down, others will take the load)
    - I'd be surprised if this was a priority for your project!