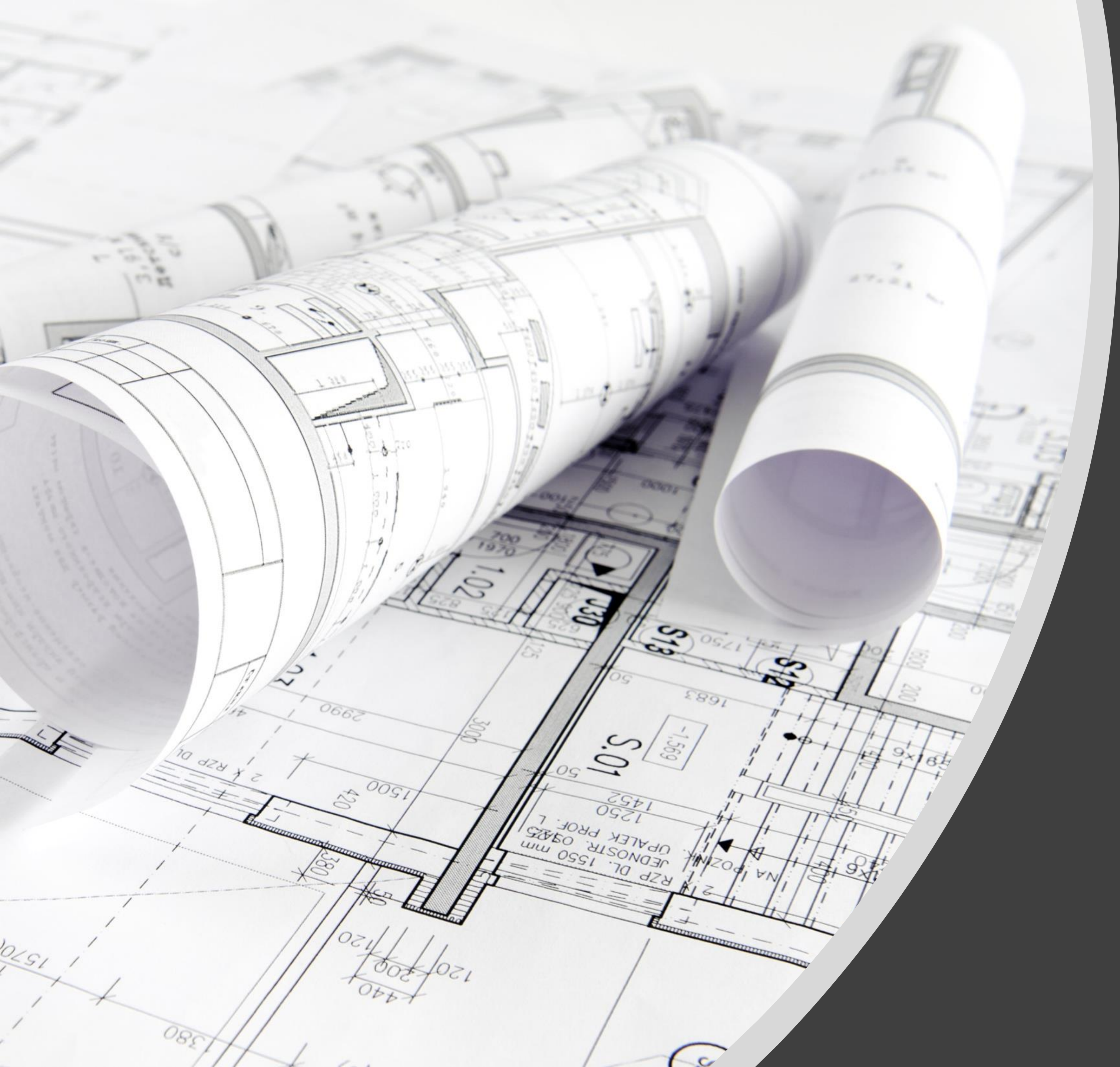


Lecture 6: Architecture

CS 5150, Spring 2026

Lecture goals

- Understand the importance and need for software architectures
- Visualize structural models with deployment and interface diagrams
- Identify common architectural styles

Poll: How are you feeling about the project?

PollEv.com/cs5150sp26

System design

Design steps

- Given requirements, must design a system to meet them
  - System architecture
  - User experience
  - Program design

- **Ideal**: requirements are independent of design (avoid implementation bias)
- **Reality**: working on design clarifies requirements
  - Methodology should allow feedback (strength of iterative & agile methods)

Design principles

- Design is an especially **creative** part of the software development process
  - More a "craft" than a science
  - Many tools are available; must select appropriate ones for a given project

- Strive for simplicity
  - Use modeling, abstraction to (hopefully) find simple ways to achieve complex requirements
  - Designs should be easy to implement, test, and maintain
- Easy to use correctly, hard to use incorrectly
- Low coupling, high cohesion

Software Architecture

Software architecture *is the* ***set of structures*** *needed to reason about a software system and the discipline of creating such structures and systems. Each structure comprises* ***software elements****,* ***relations*** *among them, and* ***properties*** *of both elements and relations.* [Bass et al. 2003]

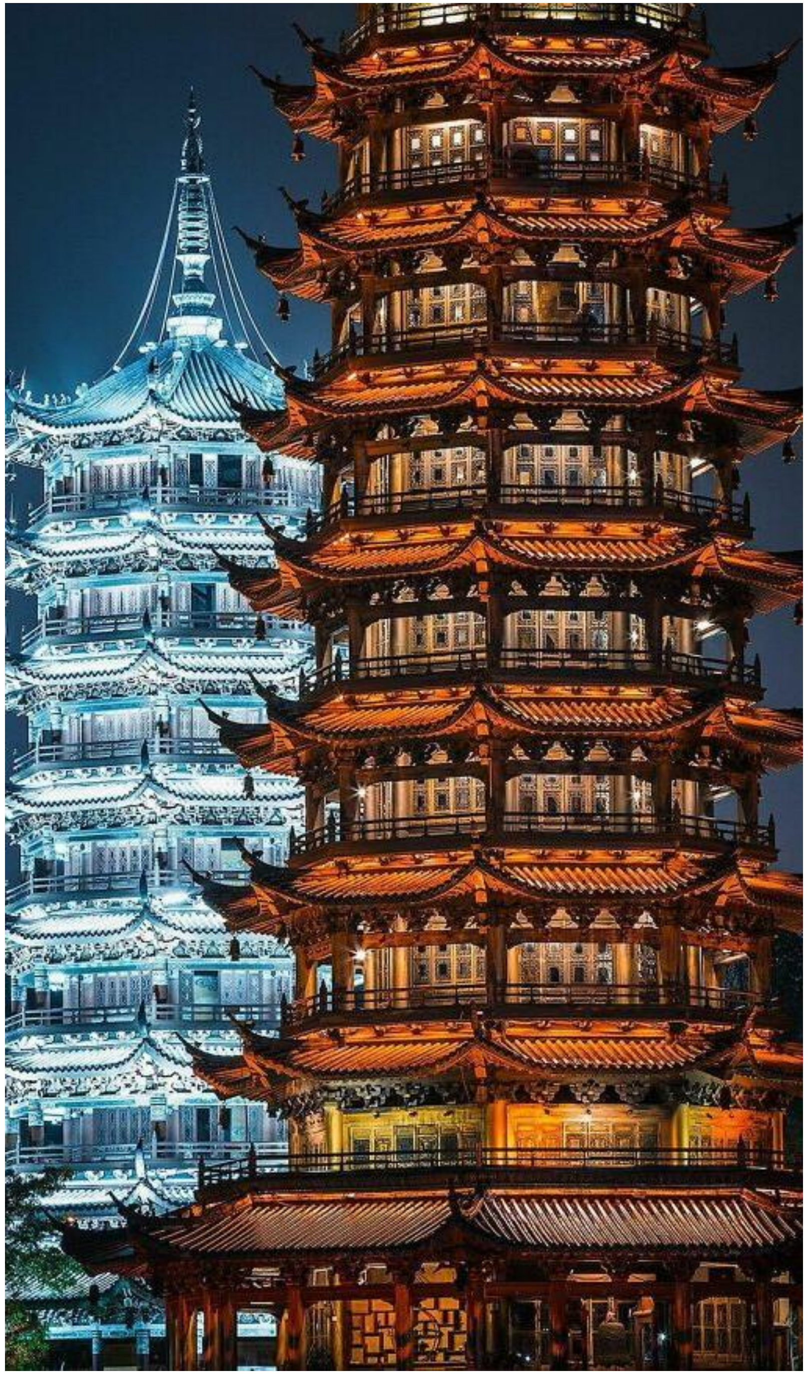
Note: this definition is ambivalent to whether this architecture is known or any good!

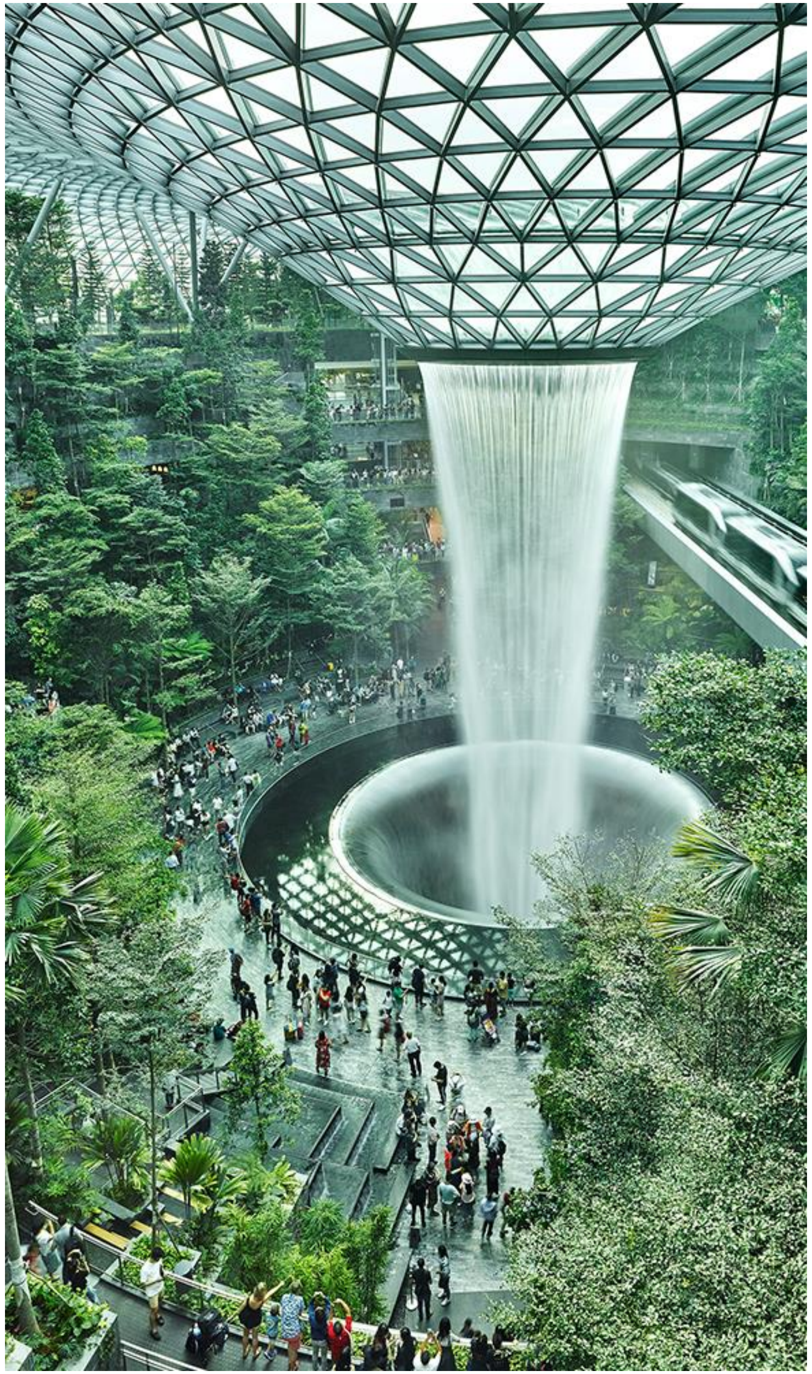
Software Architecture: Other Definitions

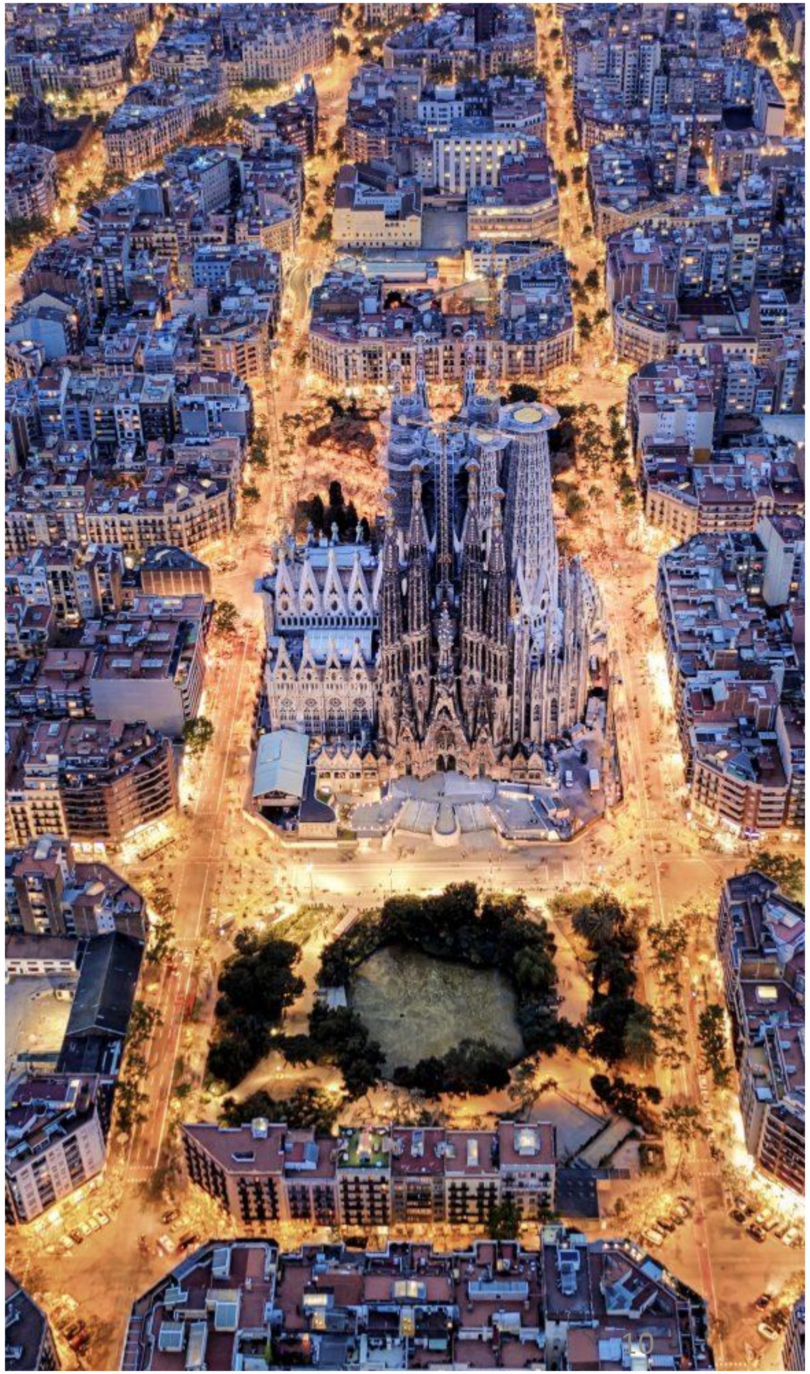
- Brooks: Conceptual integrity is key to usability and maintainability; architect maintains conceptual integrity
- Johnson: *The shared understanding that expert developers have of the system / The decisions you wish you could get right early in a project*
- Sommerville: Dominant influence on non-functional system characteristics
- "Highest level" organization of system

Why document Architecture?

- Blueprint for the system
  - Artifact for early analysis/communication
  - Primary carrier of quality attributes: **performance, robustness, reusability**
  - Key to post-deployment maintenance
- 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







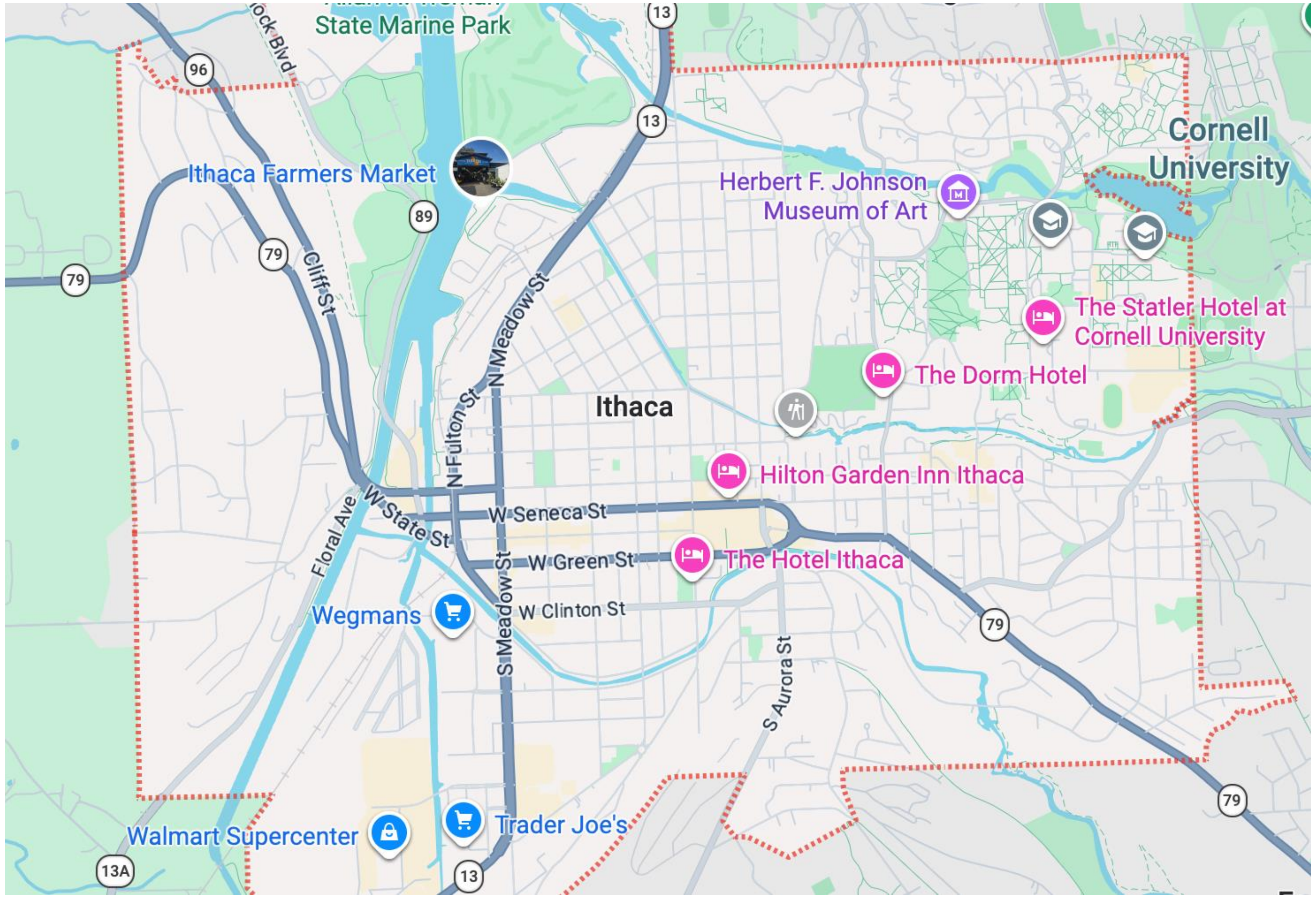
Every software system has an architecture, whether you know it or not!
If you don’t consciously elaborate the architecture, it will evolve by itself!

Levels of abstraction

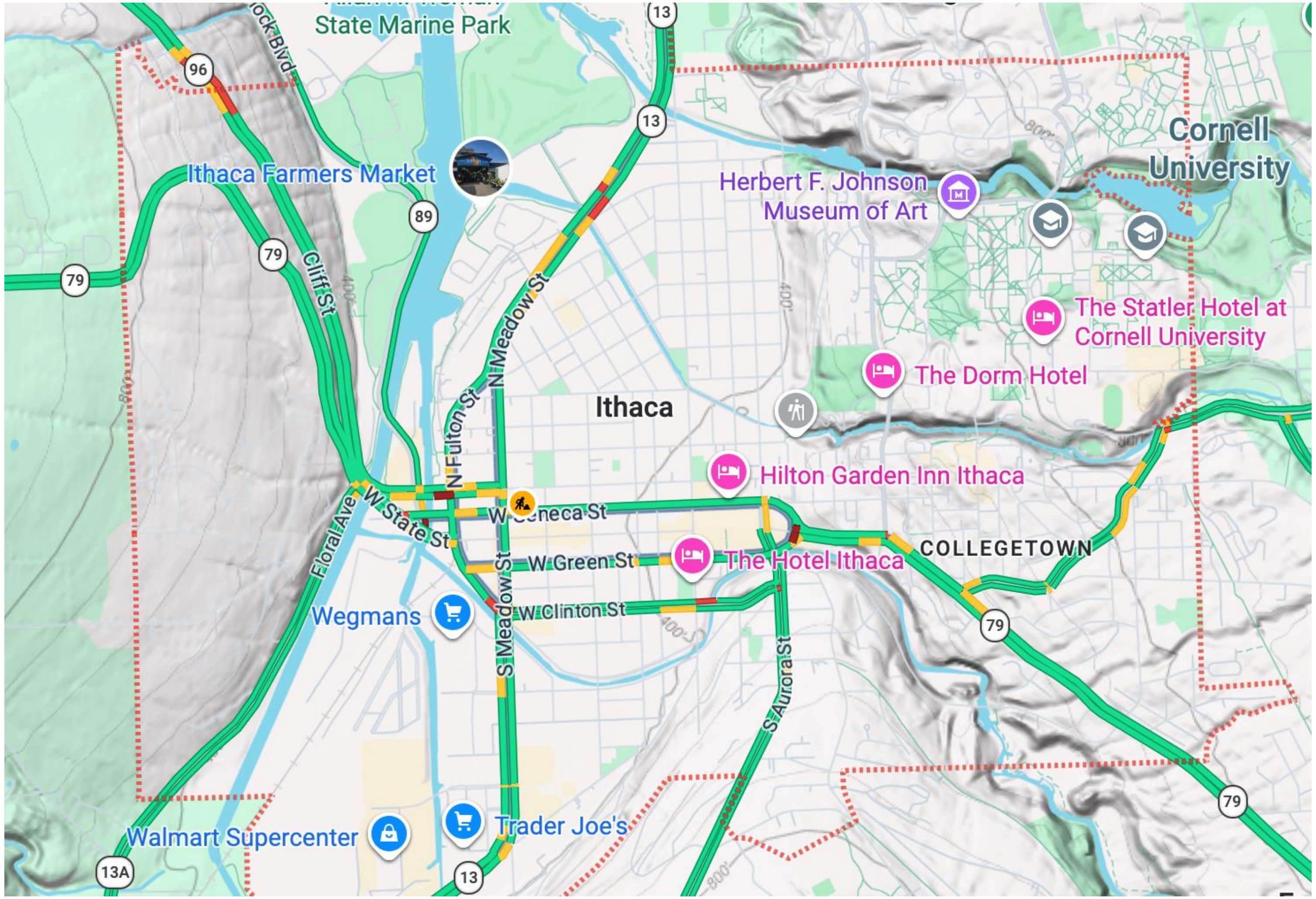
- Requirements
  - High-level "what” needs to be done
- Architecture
  - High-level "how"
  - Mid-level "what"
- Program design (Design patterns)
  - Mid-level "how"
  - Low-level "what"
- Code
  - Low-level "how"

- Documentation for each step should respect its level of abstraction
  - Avoid biasing later steps
  - Avoid redundancy

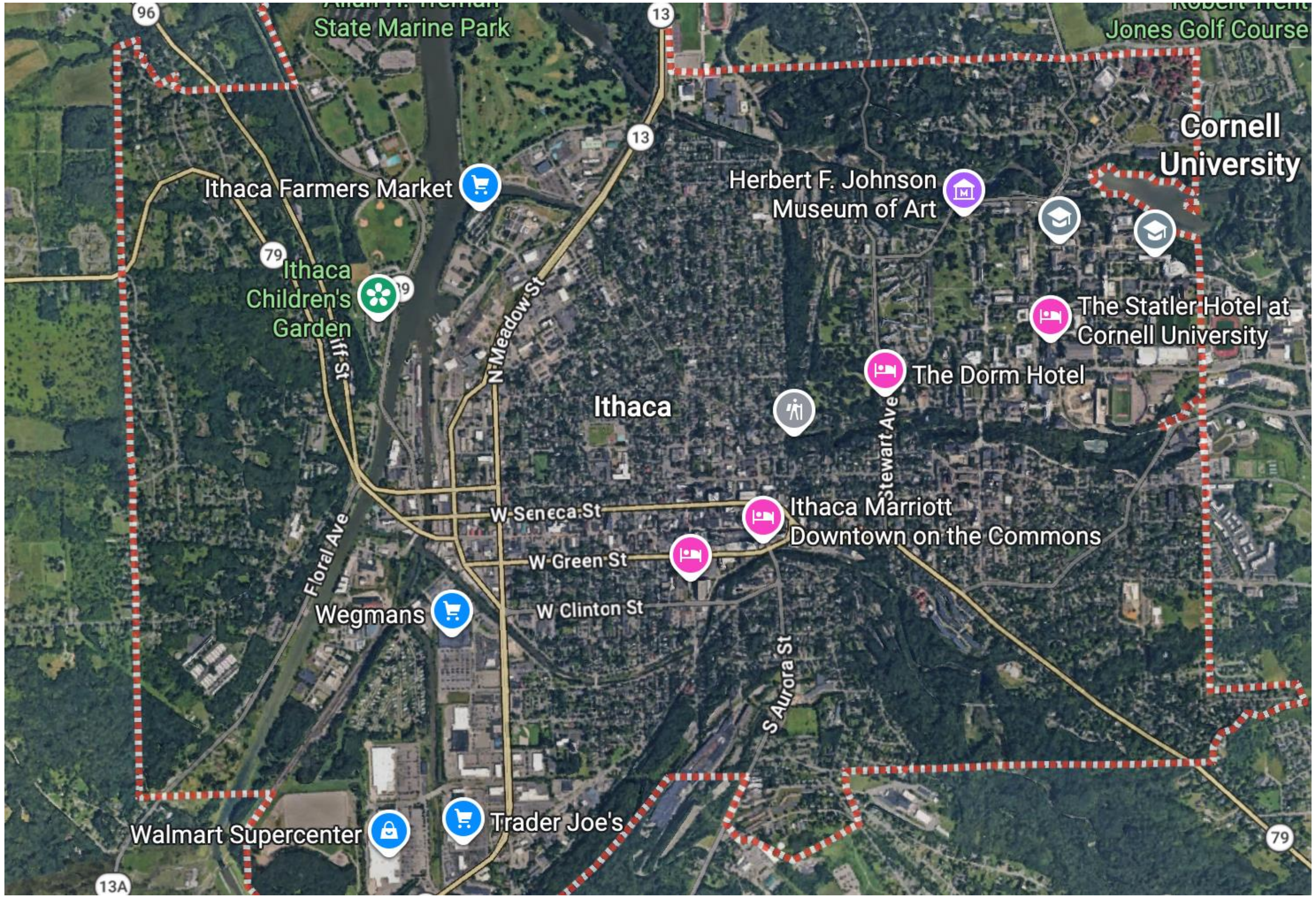
Architectural Views



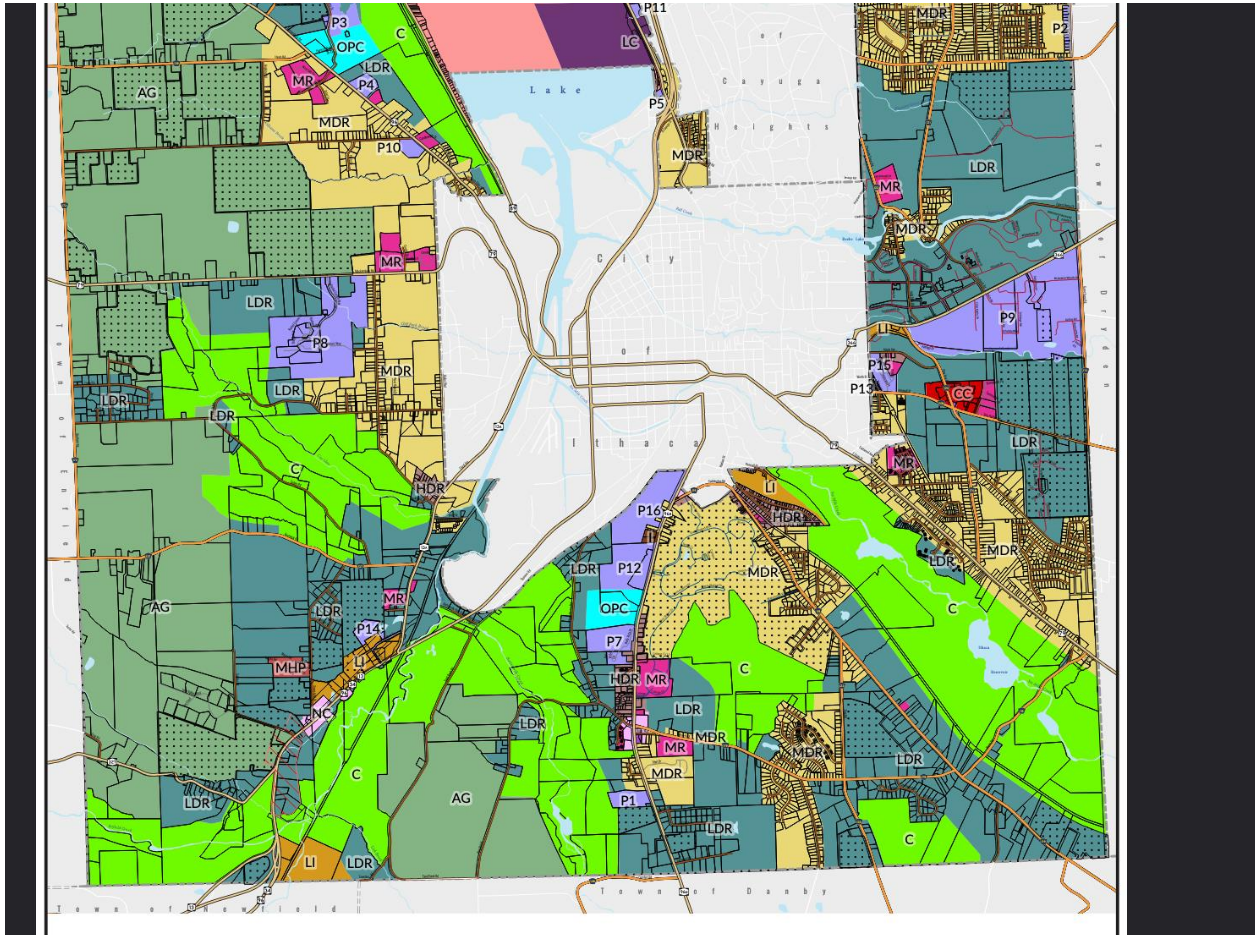
Architectural Views



Architectural Views



Architectural Views



https://townithacany.gov/wp-content/uploads/2025/07/Zoning-Wall-Map-Finished-2025-V7.pdf

Example

- Requirements:
  - Drone should hover stably
- Architecture
  - Sensing → navigation → control → actuation
  - Radio input
- Program design
  - PID controller, low-pass filter
  - Gain registry

- Code
  -
    ```
    def lpf(x1, y0, a):
      """exp filter w/ smoothing factor a"""
      return a*x1 + (1-a)*y0
    ```

Architectural considerations

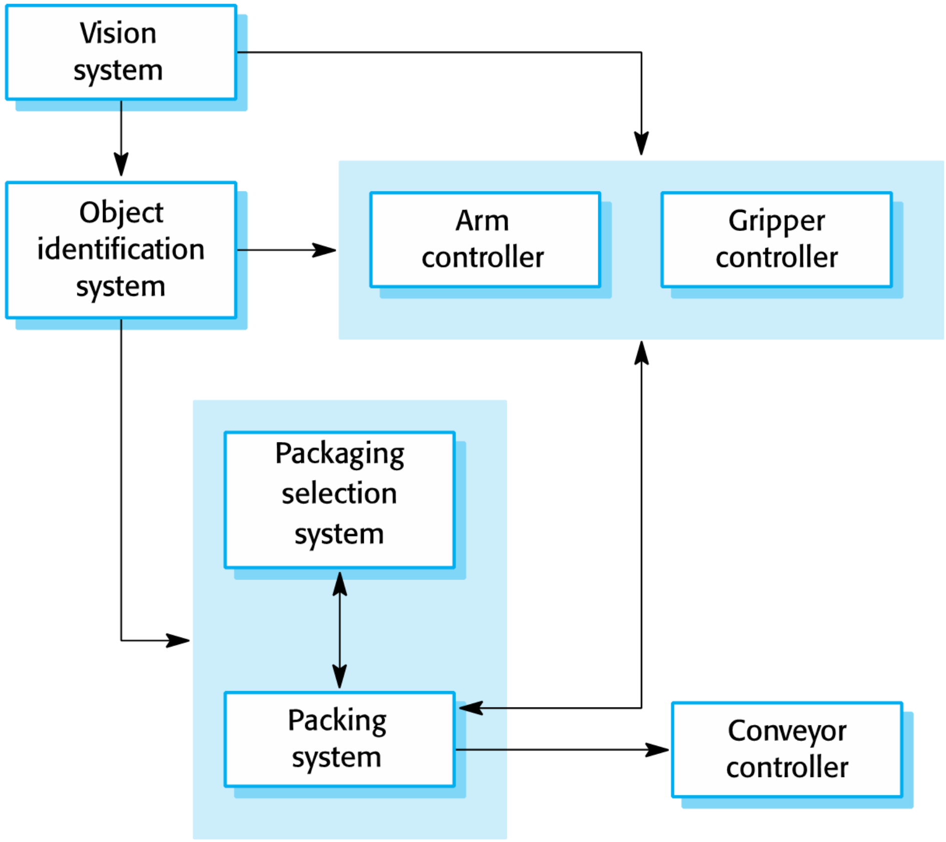
- Infrastructure
  - Hardware
  - Operating systems
  - Virtualization
- Interfaces
  - Networks/buses
  - Protocols
- Services
  - Databases
  - Authentication
- Operations
  - Testing
  - Logging/monitoring
  - Backups
  - Rolling deployment
- Product line

Architectual models

- Diagram **and** supporting specification
  - Be *specific* with notation
- Multiple perspectives
  - Conceptual
  - Static (subsystems)
  - Dynamic (data flow)
  - Physical (deployment)

- Appropriate level of detail
  - A single diagram should fit cleanly on one page
- Distinct from program models
  - Inheritance diagrams don’t show interactions

Examples



Vision system

Object identification system

Arm controller

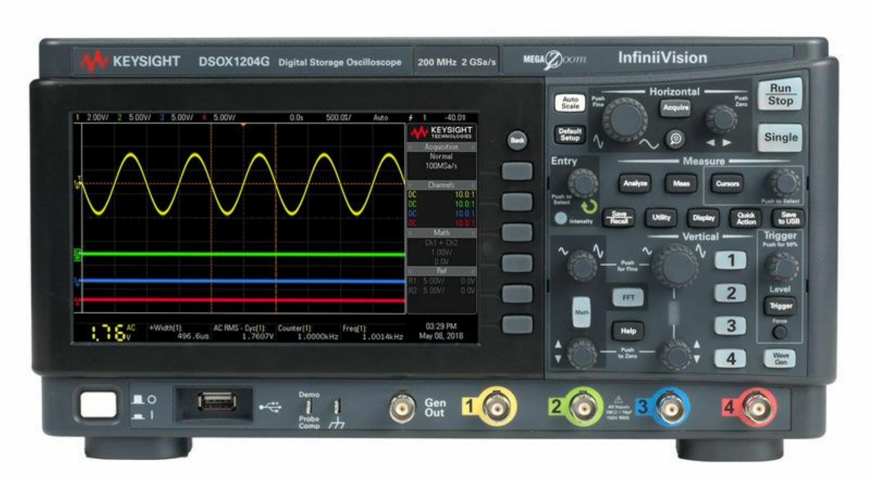
Gripper controller

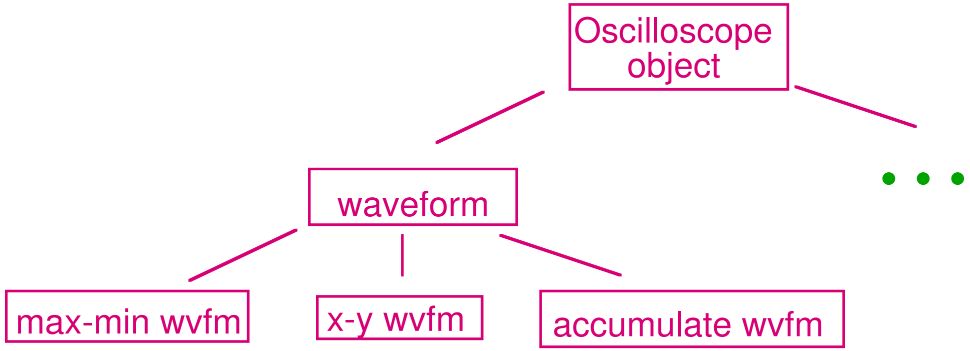
Packaging selection system

Packing system

Conveyor controller

Packing Robot control system





Oscilloscope object

waveform

max-min wvfm

x-y wvfm

accumulate wvfm

• • •

Oscilloscope

Sommerville, *Software Engineering*

Garlan & Shaw, "An Introduction to Software Architecture"

Subsystems

- Improve comprehensibility of system by decomposing into subsystems
- Group elements into subsystems to minimize coupling while maintaining cohesion

- Coupling: Dependencies *between* two subsystems
  - If coupling is high, can't change one without affecting the other
- Cohesion: Dependencies *within* a subsystem
  - High cohesion implies closely-related functionality