What is Architecture?

- **Software System Architecture** = defines:
  - (1) A collection of software and system components, connectors, and constraints
  - (2) A collection of system stakeholders’ need statements
  - Rationale which demonstrates that (1) will satisfy (2)

- **System stakeholders**
  - An individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system
    - *Concerns* = interest pertaining to system’s development, operation, or any other critical aspect (performance, reliability, security, etc.)
  - Includes customers/users, architects, developers, and evaluators
What is Architecture?

In other words, Software Architecture is...

- **High-level** part of software design
- **Frame** that holds the more detailed parts of the design
- Important because:
  - Conceptual integrity
  - Partitioning of work

“Architecture specification” / “Top-level Design”
- Document that describes architecture
Architecture vs. High-Level Design

- Some make a distinction between:
  - *Architecture* ➔ system-wide constraints
  - *High-level design* ➔ subsystem constraints
What is a System?

**System**
- A collection of components organized to accomplish a specific function or set of functions
- Can loosely refer to individual applications, product lines/families, enterprises, etc.

**Every system has an architecture**
- May not be documented
- May be good, mediocre, or terrible, but still has one
General Architectural Quality

Good architecture:

- Discusses:
  - Classes (and info hidden in classes)
  - Rationale/justification for major decisions

- Attributes:
  - Internally consistent
  - Complete (w.r.t. requirements)
  - Matches problem
  - Machine- and language-independent
  - Balances over- and under-designing
  - Identifies risky areas

- Has multiple views
  - View = a representation of whole system from perspective of a related set of concerns

- Something YOU understand and are comfortable with
Structure, Components, and Connections

- Structure, Components, and Connections
  - Program Organization
    - Major building blocks
    - How block communicate
  - Major Classes
    - Includes:
      - Responsibilities, interactions, hierarchies, state transitions, persistence
    - **80/20 rule**
      - Specify 20% of classes that make up 80% of system’s behavior
  - Data Design
    - Files, tables, etc.
  - User Interface Design
    - Must specify now (if not at requirements time)
    - Should be modularized (separate from other subsystems)
Constraints

- Constraints
  - Business Rules
    - Rules and impact on design
  - Resource Management
    - How resources managed
    - Normal and maximum cases
Quality

- Quality Attributes
  - Security
    - Design and code level
    - Build threat model
  - Performance
    - Goals and priorities
    - Budgets and estimates
    - Feasibility and risks
    - Algorithms/data types needs
  - Scalability
    - How system will deal with growth
      - If NOT to be expanded → state explicitly
  - Interoperability
    - Sharing data/resources with other software/hardware
  - Internationalization/Localization
    - Internationalization ("I18n") = support multiple locales
    - Localization ("L10n") = support specific local language
    - Important → buckets of strings in any given interactive program
Error Handling and Robustness

- **Error Handling and Robustness**
  - **Input/Output**
    - Describe level where I/O errors detected
  - **Error Processing**
    - Estimated that 90% of code is for error checking/processing
    - Should have error processing strategy in architecture
  - **Fault Tolerance**
    - Techniques to increase system reliability in the face of errors
      - Detection, recovery, and containment of bad effects
  - **Overengineering**
    - Determine system-wide level of robustness desired
    - Don’t want mix of hyper-robust and barely-adequate classes
Management, Planning, and Business Decisions

- Architectural Feasibility
  - Is it possible to build this at all?
- Buy-vs.-Build Decisions
- Reuse Decisions
- Change Strategy
  - Anticipate possible changes