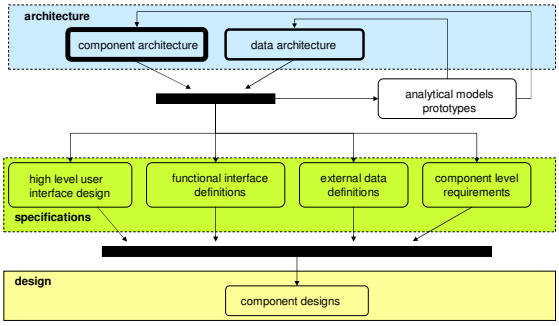
System Architecture

- Overview of architecture
  - managing complexity, criticality of architecture
  - architecture vs. design
- Characteristics of good design/architecture
  - simplicity & generality
  - modularity, information hiding, coupling
  - interfaces: criticality, abstraction, stability
  - mechanism policy separation
- Introduction to Project 3

Model Hierarchy/Succession



Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise:

1. software elements,
2. the externally visible properties of those elements,
3. and the relationships among them.

Complexity and Architecture

- How to solve a complex problem
  - break it down into multiple sub-problems
  - tackle the sub-problems one-at-a-time
- How to design a complex s/w system
  - decompose it into independent components
  - design and build each independently
- Not just any decomposition will do
  - decomposition must be stable and robust
  - each piece must be reasonably build-able
  - components must be truly independent

Independent Components

- have clearly specified external interfaces
  - defined by the architecture
- can be designed independently
  - but dependencies often emerge w/design
- can be built and tested independently
  - this greatly constrains external interfaces
- may have to be Field Replaceable Units
  - replace one component, leaving others alone
  - this further constrains external interfaces

Architecture v.s. Design

- an architecture ...
  - is a technical description of a system
  - enumerates the high-level sub-components
  - describes functionality of each component
  - describes interfaces to and between them
- a design ...
  - is a technical description of a component
  - describes how it is implemented
- the difference is the things described, more than the specificity of description

Architectural Models

- Help us understand the system
  - structure – components it is comprised of
  - behavior – how the components interact
- Basis for project planning
  - project is implementation of specified components
- Basis for analytical models
  - model system components and functions
- Context for component requirements & designs
  - component requirements follow from the role each component plays in satisfying overall system requirements

Criticality of Architecture

- It solves hard problems
- It drives performance & robustness
- It drives the development process
  - complexity of each component
  - how tasks can be divided among groups
  - order of component development
  - how system & components can be tested
  - component integration strategy
- It determines supportability

Complexity = lack of simplicity

- Complex systems
  - many types of components and interfaces
- Complex components
  - many methods, variants and parameters
  - many interactions, elaborate usage rules
- Complex systems are very difficult
  - to design, build, use, support and maintain
- Elegance
  - finding simple and robust solutions to complex and challenging problems

Design Modularity

- High Cohesion
  - consistency - module only does one thing
    - manage one type of object, perform one computation
  - completeness – centralized responsibility
    - all operations on this class are in this module
- Low Coupling (information hiding)
  - well abstracted interfaces to all services
    - provide services required by all clients
  - minimal exposure of internal details
    - clients depend on interfaces, not implementations

Benefits of Modularity

- Better Architecture
  - simpler component specifications
  - results in naturally hierarchical design
- Easier Design and Implementation
  - enable parallel development of different components
  - fewer interactions to manage, simpler code
  - faster and easier to design and code
  - will have fewer errors and be easier to test
- Maintenance less expensive, more effective
  - simpler modules are easier to understand
  - most changes are confined to a single module
  - implementation changes have few side effects

Software Interfaces

- where independent components meet
  - Application Programming Interfaces
    - packages, classes, includes, defines, routines
  - data formats
    - file formats, databases, dynamic data structures
  - network protocols
    - basic communication, higher level services
- interface specifications are contracts
  - they spell out responsibilities of each party
  - if all parties follow them, the system will work

Well Abstracted Interfaces

- Do what the client needs
  - provide all the required functionality
  - in a simple to use fashion
- Without exposing the implementation
  - client view is abstract (what, not how)
  - a simpler view for client
  - greater freedom for the implementer
    - to change implementations in the future
    - to optimize performance, to fix bugs
    - to address future requirements

Slavery and Freedom

- a contract tells us what we have to do
  - we must conform to the interface specification
  - this greatly constrains our design freedom
- well abstracted interfaces don't tell us how
  - we can implement contract any way we want
  - we can change our implementation any time
- we can make changes in the future
  - if they are upwards compatible
  - or if we can find and fix all existing clients

Mechanism/Policy Separation

- Mechanisms should not unduly limit the range of policies that users can employ.
  - Mechanisms
    architecture, algorithms, and data structures
    (all things that are difficult to change in the field)
  - Policies
    how the system should behave in specific situations
- we can't envision all possible situations
  - different users have different needs
  - mechanisms will find new uses in the future

For Next Lecture

We will look at the processes for developing and validating an architecture

- McConnell 5.3-4, 34.1, 34.6
  - good heuristics for system design
- Garlan : Introduction to Architecture (ch 3-4)
  - intro to some basic architectures & their application
- Foote & Yoder: Big Ball of Mud
  (only need to read Intro through "Keep it Working")
  - a long but good (and easily read) treatise on anti-architecture and the forces that drive it