

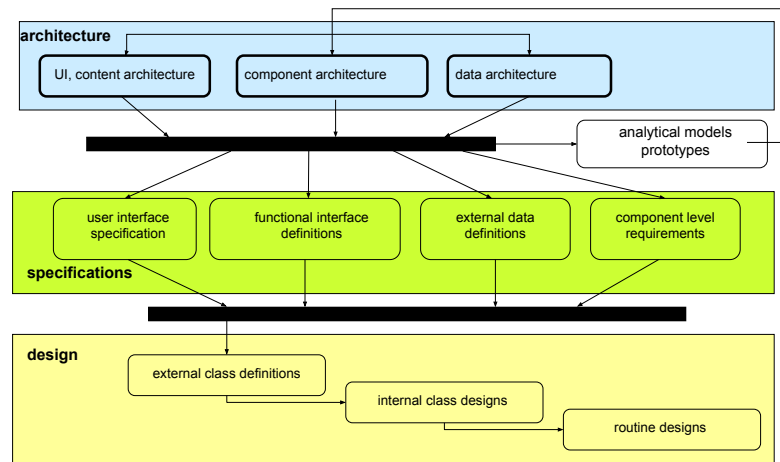
System Architecture

- What is architecture?
 - architecture vs. design
 - criticality of architecture
 - managing system complexity
- Characteristics of good design/architecture
 - simplicity & generality
 - modularity, information hiding, coupling
 - interfaces: criticality, abstraction, stability
 - mechanism policy separation

System Architecture (what)

1

Model Hierarchy/Succession



Models and Prototypes

2

Software Architecture (SEI)

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

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

to which I add:

4. how they cooperate to solve the problem

System Architecture (what)

3

System Architecture Description

- 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

System Architecture (what)

4

Architecture v.s. Functionality

- Functionality must be understood
 - operations that can be performed
 - information that can be presented
 - parameters that must be supplied
 - dialogs and the paths between them
- None of this is architecture
 - these are functional requirements
 - these are user interface proposals
- Architecture describes implementing s/w
 - independent run-time components
 - the classes that implement them

5

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

System Architecture (what)

6

Criticality of Architecture

- It solves hard problems
- It drives performance & robustness
- It drives the development process
 - achievability/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

System Architecture (what)

7

The Goal: “elegance”

“Solving a complex problem, or achieving a difficult goal with a minimum of effort or mechanism”

- approaches that eliminate hard problems
- greatly simplify common operations
- mechanisms that solve multiple problems
- often problems not previously recognized to have been related

System Architecture (what)

8

Complexity is the enemy

- Complex systems
 - many types of components and interfaces
 - likely to have many modes of failure
- Complex components
 - many methods, variants and parameters
 - many interactions, elaborate usage rules
- Complex systems are very difficult to ...
 - design and build ... much to remember
 - test ... many interactions to test
 - maintain ... every change breaks something

System Architecture (what)

9

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

System Architecture (what)

10

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

System Architecture (what)

11

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

System Architecture (what)

12

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

System Architecture (what)

13

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

System Architecture (what)

14

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

System Architecture (what)

15

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

System Architecture (what)

16

Elements of Good Architecture

- understandable
 - simple (relative to the problem)
 - well-described and logical (once it is understood)
- good component modularity
 - well-apportioned responsibility among the components
 - clear and reasonable functionality
 - good independence between components
 - enables independent design, development, testing
- well designed interfaces
 - well abstracted for the intended clients
 - adequate flexibility for implementers
 - mechanism/policy separation
- enables performance, scalability, robustness, ...

System Architecture (what)

17

My thoughts on process

1. Architecture development is a process of constructive decomposition.
2. The fundamental goal is anti-entropic evolution, organizing simple and distinct parts into a system whose functionality is beyond that of its parts.
3. The important decisions are often as much compromise as inspiration
4. Remember the scene near the end of *The Matrix*, where he suddenly stopped all the bullets in mid-air? It can be that obvious when you get it right!

System Architecture (what)

18

In-Class Architectural Review

- Need Two Volunteer Teams
 - architecture to review
 - reviewing team
- Benefits to reviewing team
 - score: better of 100% or +25%
- Cost
 - preparation must be completed sooner
 - other teams get to learn from your experience

For Next Lecture

- McConnell 5.1
 - why design is a “wicked” problem
- McConnell 5.3-4, 34.1, 34.6
 - heuristics for system design
- Kampe: Architecture Does All That?
 - non-functional issues architecture must solve

Supplementary Slides

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