

---

# CS6362 Software Architecture and Design

---

Jing Dong

1

---

## Dog House Architecture



---

2

## Architecting a house



3

## Architecting a high rise



4

## Early architecture



### Progress

- Limited knowledge of theory

5

## Modern architecture



### Progress

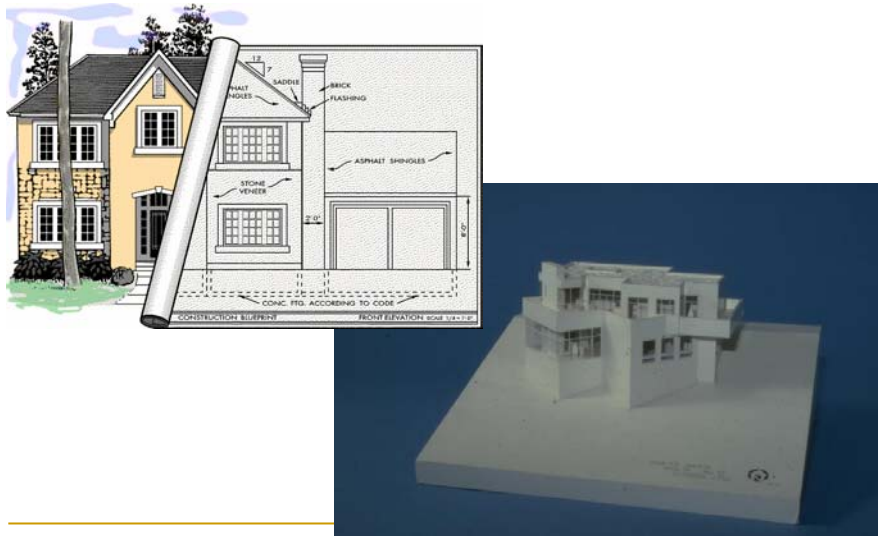
- Advances in materials
- Advances in analysis

### Scale

- 5 times the span of the Pantheon
- 3 times the height of Cheops

6

## Modeling a house



7

## Introduction

- Intuition about architecture
  - Building architecture
  - Network
  - Hardware

8

---

## Building Architecture

- **Multiple Views**: skeleton frames, detailed views of electrical wiring, etc.
  - **Architectural Styles**: Classical, Romanesque, and so on.
  - **Materials**: One does not build a skyscraper using wooden posts and beams.
- 

9

---

## Hardware Architecture

- **RISC** machines emphasize the instruction set as an important feature.
  - **Pipelined** and **multi-processor** machines emphasize the configuration of architectural pieces of the hardware.
- 

10

## Differences & Similarities of SW & HW Architectures

- **Differences:**

- relatively (to software) small number of design elements.
- scale is achieved by replication of design elements.

- **Similarities:**

- we often configure software architectures in ways analogous to hardware architectures. (*e.g.*, we create multi-process software and use pipelined processing).

## Network Architecture

- Networked architectures are achieved by abstracting the design elements of a network into nodes and connections.
- Topology is the most emphasized aspect:
  - Star networks
  - Ring networks
  - Manhattan Street networks
- Unlike software architectures, in network architectures only few topologies are of interest.

## Motivations

- Bridge the gap between requirement and implementation
- Increase level of abstraction
- Provide vocabulary of architectural descriptions and definitions
- Document patterns of interactions
- Help on the decisions of architectural alternatives
- Allow high level analysis

## History

- Early 1950s:
  - Software was written in machine language
  - Programmers placed instructions and data individually and explicitly in the computer's memory
  - Insertion of a new instruction in a program might require hand checking the entire program to update references to data and instructions

## History (cont.)

- Assemble language
- Some machine code programming problems were solved by adding a level of abstraction between the program and the machine:
  - **Symbolic assemblers:**
    - Names used for operation codes and memory addresses.
    - Memory layout and update of references are automated.
  - **Macro processors:**
    - Allow a single symbol to stand for a commonly used sequence of instructions.

15

## History (Cont.)

- **Late 1950s:** programming language
  - The emerging of the first high-level programming languages. Well understood patterns are created from notations that are more like mathematics than machine code.
    - Evaluation of arithmetic expressions,
    - Procedure invocation,
    - Loops and conditionals

16

## History (Cont'd)

- **FORTRAN** becomes the first widely used programming language.
- **Algol** and its successors followed with higher-levels of abstraction for representing data (types).

17

## History (Cont.)

- **Late 1960s and 1970s:** abstract data types
  - Programmers shared an intuition that good data structure design will ease the development of a program.
  - This intuition was converted into theories of modularization and information hiding.
    - Data and related code are encapsulated into modules.
    - Interfaces to modules are made explicit.
  - Various programming languages (e.g., Modula, Ada, Euclid)
  - Module interconnection languages (e.g., MIL75, Intercol)
  - Emerge with implementations of this theory.

18

## Why Software Architecture

- Aren't programming language good enough?
- As the size and complexity of software systems increases, the design problem goes beyond algorithms and data structures.
- Designing and specifying the overall system structure (**Software Architecture**) emerges as a new kind of problem.

## Software Architecture Issues

- Organization and global control structure,
- protocols of communication, synchronization, and data access,
- assignment of functionality to design elements,
- physical distribution,
- selection among design alternatives.

---

## State of Practice

- There is not currently a well-defined terminology or notation to characterize architectural structures.
  - However, good software engineers make common use of architectural principles when designing complex software.
  - These principles represent rules of thumb or idiomatic patterns that have emerged informally over time. Others are more carefully documented as industry standards.
- 

21

---

## Software Architecture

- The architecture of a software system
    - Defines the system in terms of components and interactions among components
    - Shows correspondence between requirements and elements of the constructed system
    - Addresses system-level properties such as scale, capacity, throughput, consistency, compatibility
- 

22

## Software Architecture

- An architectural definition selects
  - Components: define the locus of computation, e.g. filters, databases, objects, ADTs
  - Connectors: mediate interactions of components, e.g. procedure call, pipers, event broadcast
  - Properties: specify information for construction & analysis, e.g. signatures, pre/post conditions, RT specs.

23

## Theme

- Progress in software has results from
  - Abstraction – focus on important commonalities
  - Codification – presentation in operational form
  - Modularity – divide and conquer
- These techniques will help establish an engineering discipline for software
- They will guide our study of software architecture

24

## What Is Engineering?

- Solve practical problems
- Create cost-effective solutions
- Apply scientific knowledge
- Build things, e.g. systems
- Serve mankind
- Engineering enables ordinary people to do things that formerly required experts

25

## Characteristics of Engineering Practice

- Iteration between analysis and design
- Tradeoffs between alternatives
- Heavy use of earlier designs
- Handbooks, manuals
- Pragmatic approach to cost-effectiveness

26

## Engineering Design to Decisions

- Must discriminate among
  - different kinds of problems
  - different kinds of solutions
- Must make informed choices to match problems with solutions
- The decisions with the most impact are the ones that deal with overall system organization

27

## Problems vs Solutions

- Problems
  - Requires context – the subject matter of the system
    - Relevant parts of real world, their properties & relations
    - Often informal and using heterogeneous notations
  - Software designer must be able to learn about domain
    - Phenomenology, technology of description, formalization
  - Problem frame == principal parts + solution task
- Solutions
  - Characteristics of the machine that solves the problem
  - Draw on standard parts, templates, ...
  - Software architecture, patterns, programming clichés
  - Most of the subject matter of this course

28

## Elements of a Complete Software System

User Model	User view of problem
Requirement	Software view of problem
Architecture	Modules and connections
Code	Algorithms & data structures
Executable	Data layouts, memory maps

29

## Observations About Designers

- They freely use informal patterns (idioms)
  - Very informal, imprecise semantics
  - Diagrams as well as prose, but no uniform rules
  - Communication takes place anyhow
- Their vocabulary uses system-level abstraction
  - Overall organization (styles)
  - Kinds of components and interactions among them
- They compose systems from subsystems
  - Tend to think about system structure statically
  - Often select organization by default, not by design

30

## Architectural Design Task

Architectures	Programs
Interactions among parts	Implementations of parts
Structural properties	Computational properties
Declarative	Operational
Mostly static	Mostly dynamic
System-level performance	Algorithmic performance
Outside module boundary	Inside module boundary

31

## System Architecture

- Architecture is the underlying structure of things
- What is good architecture?
  - Principles and techniques consistently applied through all phases of a project
  - Resilient in the face of changes
  - Source of guidance throughout the product lifetime

32

## Architecture Descriptions

- *“Camelot is based on the **client-server model** and uses **remote procedure calls** both locally and remotely to provide communication among applications and servers.”*
- *“We have chosen a **distributed, object-oriented** approach to managing information.”*

33

## Architecture Descriptions

- *“**Abstraction layering** and system decomposition provide the appearance of system uniformity to clients, yet allow Helix to accommodate a diversity of autonomous devices. The architecture encourages a **client-server model** for the structuring of applications.”*

34

## Architecture Descriptions

- *“The easiest way to make a canonical sequential compiler into a concurrent compiler is to **pipeline** the execution of the compiler phases over a number of processors. A more effective way is to split the source code into many segments, which are concurrently processed through the various phases of compilation (by multiple compiler processes) before a final, merging pass recombines the object code into a single program.”*