Lecture 22: Moving into Design

→ Analysis vs. Design
  % Why the distinction?

→ Design Processes
  % Logical vs. Physical Design
  % System vs. Detailed Design

→ Architectures
  % System Architecture
  % Software Architecture
  % Architectural Patterns (next lecture)

→ Useful Notation
  % UML Packages and Dependencies

Refresher: Lifecycle models

Waterfall model
- Requirements
- Logical design
- Physical design
- Code
- Test
- Integration
- Maintenance

V model
- Requirements
- Logical design
- Physical design
- Code
- Test
- Integration
- Acceptance test
- Maintenance

Spiral model
- Requirements
- Analyze and design
- Test and integrate
- O&M

Refresher: different worlds

Application Domain
- Analysis is all about studying this world

Machine Domain
- Design is all about building this world

But who builds the bridge?
Four design philosophies

Decomposition & Synthesis
- Drivers:
  - Managing complexity
  - Reuse
- Example:
  - Design a car by designing separately the chassis, engine, drivetrain, etc. Use existing components where possible

Search
- Drivers:
  - Transformation
  - Heuristic Evaluation
- Example:
  - Design a car by transforming an initial rough design to get closer and closer to what is desired

Negotiation
- Drivers:
  - Stakeholder Conflicts
  - Dialogue Process
- Example:
  - Design a car by getting each stakeholder to suggest (partial) designs, and then compare and discuss them

Situated Design
- Drivers:
  - Errors in existing designs
  - Evolutionary Change
- Example:
  - Design a car by observing what's wrong with existing cars as they are used, and identifying improvements

Logical vs. Physical Design

→ Logical Design concerns:
  - Anything that is platform-independent:
    - Interactions between objects
    - Layouts of user interfaces
    - Nature of commands/data passed between subsystems
  - Logical designs are usually portable to different platforms

→ Physical Design concerns:
  - Anything that depends on the choice of platform:
    - Distribution of objects/services over networked nodes
    - Choice of programming language and development environment
    - Use of specialized device drivers
    - Choice of database and server technology
    - Services provided by middleware

System Design vs. Detailed Design

→ System Design
  - Choose a System Architecture
    - Networking infrastructure
    - Major computing platforms
    - Roles of each node (e.g. client-server; clients-broker-servers; peer-to-peer, ...)
  - Choose a Software Architecture
    - (see next lecture for details)
  - Identify the subsystems
  - Identify the components and connectors between them
    - Design for modularity to maximize testability and evolveability
    - E.g. Aim for low coupling and high cohesion

→ Detailed Design
  - Decide on the formats for data storage
    - E.g. design a data management layer
  - Design the control functions for each component
    - E.g. design an application logic layer
  - Design the user interfaces
    - E.g. design a presentation layer

Global System Architecture

→ Choices:
  - Allocates users and other external systems to each node
  - Identify appropriate network topology and technologies
  - Identify appropriate computing platform for each node

→ Example:
  - See next slide...
System Architecture Questions

Key questions for choosing platforms:
- What hardware resources are needed?
  - CPU, memory size, memory bandwidth, I/O, disk space, etc.
- What software/OS resources are needed?
  - Application availability, OS scalability
- What networking resources are needed?
  - Network bandwidth, latency, remote access.
- What human resources are needed?
  - OS expertise, hardware expertise, system administration requirements, user training/help desk requirements.
- What other needs are there?
  - Security, reliability, disaster recovery, uptime requirements.

Key questions constraining the choice:
- What funding is available?
- What resources are already available?
  - Existing hardware, software, networking
  - Existing staff and their expertise
  - Existing relationships with vendors, resellers, etc.

Data Management Questions

How is data entry performed?
- E.g. Keyless Data entry
  - Bar codes, Optical Character Recognition (OCR)
  - E.g. Import from other systems
    - Electronic Data Interchange (EDI), Data interchange languages...

What kinds of data persistence is needed?
- Is the operating system's basic file management sufficient?
- Is object persistence important?
- Can we isolate persistence mechanisms from the applications?

Is a Database Management System (DBMS) needed?
- Is data accessed at a fine level of detail
  - E.g. Do users need a query language?
- Is sophisticated indexing required?
- Is there a need to move complex data across multiple platforms?
  - Will a data interchange language suffice?
  - E.g. HTML, SGML, XML
- Is there a need to access the data from multiple platforms?

Software Architecture

A software architecture defines:
- The components of the software system
- How the components use each other's functionality and data
- How control is managed between the components

An example: client-server
- Servers provide some kind of service; clients request and use services
- Applications are located with clients
  - E.g. running on PCs and workstations;
  - Data storage is treated as a server
  - E.g. using a DBMS such as DB2, Ingres, Sybase or Oracle
  - Consistency checking is located with the server
- Advantages:
  - Breaks the system into manageable components
  - Makes the control and data persistence mechanisms cleaner
- Variants:
  - Thick clients have their own services, thin ones get everything from servers
- Note: This is a SOFTWARE architecture
  - Clients and server could be on the same machine or different machines...
Criteria for decomposing a system into packages:

- **Ownership**
  - who is responsible for working on which diagrams
- **Application**
  - each problem has its own obvious partitions:
  - Clusters of classes with strong cohesion:
    - e.g., course, course description, instructor, student,...
- **Or use an architectural pattern to help find a suitable decomposition**

2 alternatives for showing package containment:

- **Use Cases**
  - Campaign Management
  - Use Case Model

- **Sub-system**
  - Package
  - Model

- **Again**
  - Campaigns
  - Staff
  - Campaigns
  - Staff
Package Diagrams

- Dependencies:
  - Similar to compilation dependencies
  - Captures a high-level view of coupling between packages:
    - If you change a class in one package, you may have to change something in packages that depend on it

A good architecture minimizes dependencies:

- Fewer dependencies means lower coupling
- Dependency cycles are especially undesirable

Dependency Cycles

The server sub-system does not depend on the client sub-system and is not affected by changes to the client’s interface.

Each peer sub-system depends on the other and each is affected by changes in the other’s interface.

Architectural Patterns

E.g. 3 layer architecture:

- Presentation Layer Package
  - Java AWT
  - Application Windows

- Application Logic Layer Package
  - Control Objects
  - Business Objects

- Storage Layer Package
  - JDBC
  - Object to Relational
  - Java SQL