

# System Design: Part II

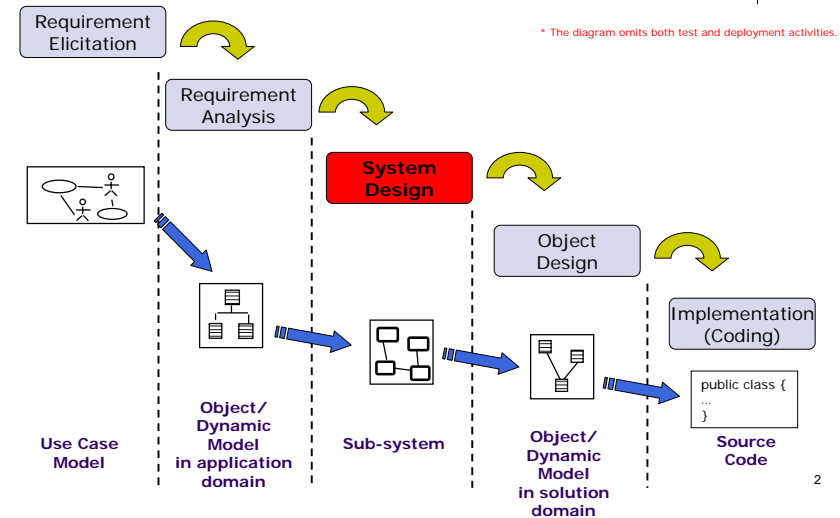
Lecture #08  
Software Engineering and  
Project Management

Instructed by Steven Choy on Nov 20, 2006



1

# Software Development Activities



2

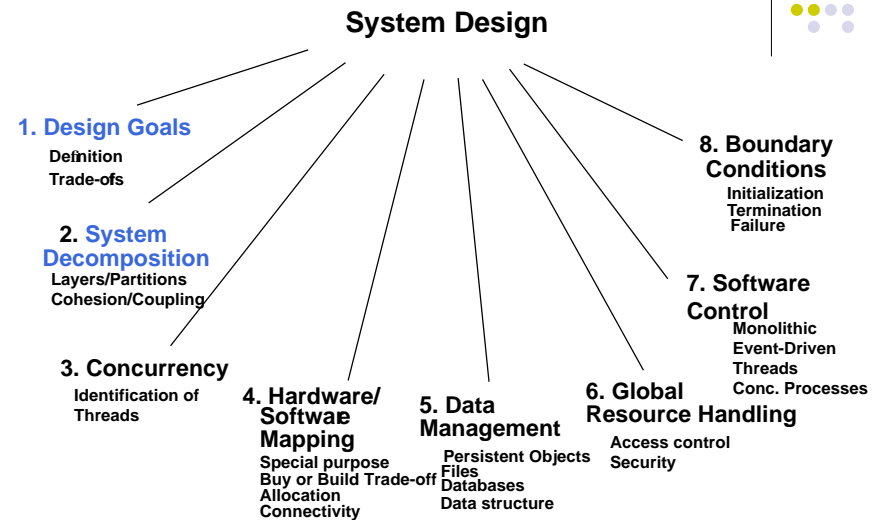
# System Design Activities

- Subsystems are managed by individual developers/groups of developer, but that doesn't mean they are independent
- Global design issues must be addressed:
  - Selection of off-the-shelf components and framework
  - Mapping of subsystems to hardware
  - Design of persistent data management
  - Design of access control policy
  - Design of global control flow
  - Handling of exceptions/boundary conditions



3

# System Design Activities



4

## How to use the results from the Requirements Analysis for System Design



- Nonfunctional requirements =>
  - Activity 1: Design Goals Definition
- Functional model =>
  - Activity 2: System decomposition (Selection of subsystems based on functional requirements, **cohesion**, and **coupling**)
- Object model =>
  - Activity 4: Hardware/software mapping
  - Activity 5: Persistent data management
- Dynamic model =>
  - Activity 3: Concurrency
  - Activity 6: Global resource handling
  - Activity 7: Software control

5

## Selection of off-the-shelf Components/Framework



- Any existing components/framework that can help realize the system. Examples:
  - The system involves asynchronous communication between subsystems. Any messaging products? → IBM MQ Series, JMS
  - I'm developing a Java web application and consider a framework for web-tier → Struts
  - The web-based system needs to provide caching of JSP. Any off-the-shelf products? → OSCache
  - The system is a Java-based standalone application. Any GUI frameworks? → SWING, AWT, SWT

6

## Hardware/Software Mapping



7

## Software/Hardware Mapping



- This activity addresses two issues:
  - You have come to an object model during analysis phase, but how can it be realized using hardware/software?
    - (Q: How shall we realize the subsystems: Hardware or Software?)
  - How does your object model map to the chosen hardware/software?
    - Mapping Objects onto Reality: Processor, Memory, Input/Output
    - Mapping Associations onto Reality: Connectivity

8

## Architectural Decision



- What is the hardware configuration?
  - Intel or Sun Sparc or AS400?
  - Window or Linux or Unix?
- How do the hardware nodes connect and communicate?
  - What's the network topology?
  - What's the protocol for communication? TCP/IP?
- What architecture do we use?
  - N-Tier or Client/Server?
- Which node is responsible for which functionality?
  - E.g. For 3-tier design,
    - Presentation/UI component is on client PC
    - Business logic components reside on web/application server
    - Storages are done on database server

9

## Architectural Challenges



- Processor Issues
  - Do we need a single or multi processor machine?
  - A cluster of machines or a multi-processor machine would be better for performance?
  - How many processors or machines are required?
  - How about scalability? Can we scale up the machines by just adding processors?
- Memory Issues
  - How much memory is enough?
  - Is there enough memory to handle burst of requests?

10

## Architectural Challenges



- Disk I/O Issues
  - Is disk I/O a bottle-neck of the system?
  - Is a single hard disk enough? Do we need a RAID system? Or even a network storage?
- Network I/O Issues
  - Is bandwidth enough for handling a burst of requests?
  - How much is enough? 10Mbs/100Mbs/1Gbs?

11

## Presenting System Architecture using UML



- We have learnt about UML for modeling classes/objects
  - Class Diagram
  - Sequence/Collaboration Diagram
  - StateChart
  - Activity Diagram
- How about UML diagrams for modeling subsystem relationship?
  - Component Diagram
  - Deployment Diagram

12

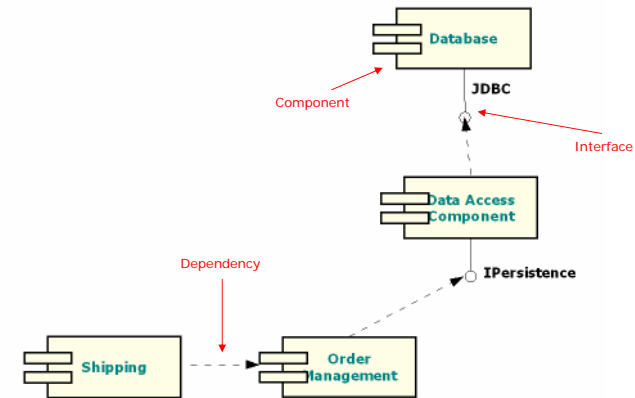
## Component Diagram



- Component diagram depicts the components, that compose the system and the relationship between them
- A component represents a code module

13

## Sample Component Diagram



14

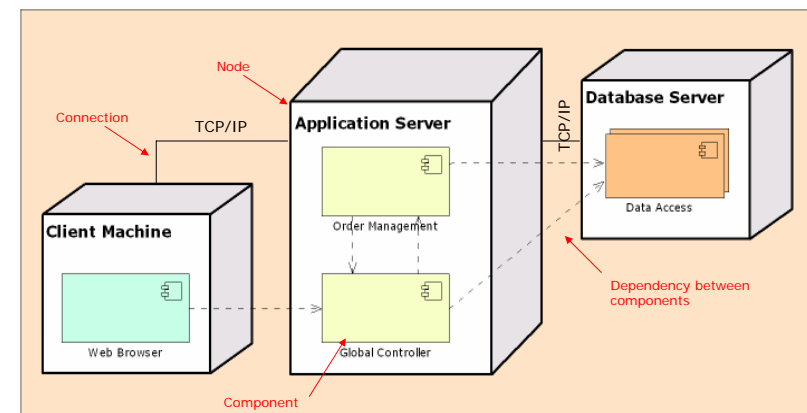
## Deployment Diagram



- Use to depict the **physical** view of software and hardware components
- What does Deployment Diagram show you?
  - Hardware of your system and its relationship
  - Software components that are installed on the hardware
  - Application components that run on the node
- Deployment diagram can combine with component diagram.

15

## Sample Deployment Diagram



16



## Data Management

17



## Data Management

- Persistent Objects
  - An object that has been assigned a permanent storage where it will be stored. When you commit the transaction in which you create a persistent object, the object is saved in the storage. A persistent object continues to exist and retains its data beyond the duration of the process that created it.
- What objects need to be persistent?
- How persistent objects be saved?
  - File system
  - Database

18



## Identify Persistent Objects

- Entity objects that you come up from analysis phrase are usually persistent objects
- Generally, those objects that must survive system shutdown (either normal shutdown or system crash) are considered as “Persistent Objects”

19



## File vs Database

- **Is the data searchable? Do you need a complex search? What's the expected data size?**
  - Normally database provides better search capability
- **Is the data just for archiving purpose, such as system log?**
  - Flat file may be good enough
- **Do you need concurrent access of data? E.g. multiple processes update the data simultaneously**
  - You can use file lock to control concurrent access
  - But database already comes with sophisticated table/record locking
- **Do you need to port the data across platforms?**
  - Flat file can work across platforms
  - Database data can be ported across different vendors but it takes much efforts (Setting up tables, export/import data, create indexes, etc...)

20

## File or Database?



- When should you choose a file?
  - Are the data voluminous (bit maps)?
  - Do you have lots of raw data (core dump, event trace)?
  - Do you need to keep the data only for a short time?
  - Is the information density low (archival files, history logs)?
- When should you choose a database?
  - Do the data require access at fine levels of details by multiple users?
  - Must the data be ported across multiple platforms (heterogeneous systems)?
  - Do multiple application programs access the data?
  - Does the data management require a lot of infrastructure?

21

## Issues To Consider When Selecting a Database



- Storage space
  - Database require about triple the storage space of actual data
- Response time
  - Most databases are I/O or communication bound (distributed databases). Response time is also affected by CPU time, locking contention and delays from frequent screen displays

22

## Issues To Consider When Selecting a Database



- Locking modes
  - Pessimistic locking: Lock before accessing object and release when object access is complete
  - Optimistic locking: Reads and writes may freely occur (high concurrency!) When activity has been completed, database checks if contention has occurred. If yes, all work has been lost.
- Administration
  - Large databases require specially trained support staff to set up security policies, manage the disk space, prepare backups, monitor performance, adjust tuning.

23

## Access Control



24

# Access Control



- Authorization
  - Describe the access rights of an actor for different objects/classes
  - In a multi-user system, different actors have different access rights for system functionalities
  - Examples:
    - System administrators have access to disable a user account
    - Accounting staffs have access to view all users' payment histories. But a normal user can only view its own one
- Authentication
  - Describe how the system determines authorized access
- Architectural Access Control
  - Network/Firewall/Encryption

# Access Control from Analysis to System Design



- During analysis, we model different access rights by associating actors and **use cases**
- In System Design phrase, we dive deeper. We model different access right by associating actors with **objects**

# Access Control Matrix



- A common way to describe access control of objects is to create a **Access Control Matrix**
- Example:

Objects/Actors	UserDB	PaymentDB
<b>SysAdmin</b>	deleteUser disableUser enableUser	
<b>Customer</b>	saveProfile getProfile	viewHistory
<b>AccountStaff</b>		viewAllUserHistories

# Implementing Authentication



- Is the actor the "Real Actor"?
- How do you implement authentication?
  - Simply using user ID and password
  - Client certificate? (e.g. Smart ID Card)
  - Biometric authentication? (e.g. Finger Print)



## Global Control Flow

29

## Global Control Flow

- Control Flow
  - Sequencing of actions in a system
  - Sequencing action includes
    - Decide which operations should be executed and its order
    - The decision may be determined by external event, result of the precedent action or the passage of time
- Three types of control flow mechanisms:
  - Procedure-driven control
  - Event-driven control
  - Threads

30

## Types of Control Flow

- Procedure-Driven Control
  - Operations wait for input whenever they need data from an actor
- Event-Driven Control
  - A main loop waiting for an event
  - Whenever there is an event, it's dispatched to the appropriate objects and takes action
  - What action to take is specified in the event
- Threads
  - Concurrent variation of procedure-driven control

31

## Boundary Conditions

32

## Exception Handling



- The real world is not ideal, so remember to consider the exceptions and boundary conditions:
  - What if the system shuts down abnormally
  - What if there is a network outage
  - What if the OS hangs
  - What if the database is down, will it bring down the whole system?
  - What if the required file does not exist in the file system
  - What if the user input an extremely lengthy user ID
- Think about exceptional case, make your system more reliable!

33

## Boundary Conditions



- Most of the system design effort is concerned with steady-state behavior.
- However, the system design phase must also address the initiation and finalization of the system.
- This is addressed by a set of new uses cases called **administration use cases**

34

## Modeling Boundary Conditions



- Boundary conditions are best modeled as use cases with actors and objects.
- Actor: often the system administrator
- Interesting use cases:
  - Start up of a subsystem
  - Start up of the full system
  - Termination of a subsystem
  - Error in a subsystem or component, failure of a subsystem or component

35

## Administration Use Cases



- Initialization
  - Describes how the system is brought from a non initialized state to steady-state ("startup use cases").
- Termination
  - Describes what resources are cleaned up and which systems are notified upon termination ("termination use cases").
- Failure
  - Many possible causes: Bugs, errors, external problems (power supply).
  - Good system design foresees fatal failures ("failure use cases").

36

## Boundary Condition Questions



- Initialization
  - How does the system start up?
    - What data need to be accessed at startup time?
    - What services have to registered?
  - What does the user interface do at start up time?
    - How does it present itself to the user?

37

## Boundary Condition Questions (continue)



- Termination
  - Are single subsystems allowed to terminate?
  - Are other subsystems notified if a single subsystem terminates?
  - How are local updates communicated to the database?
- Failure
  - How does the system behave when a node or communication link fails? Are there backup communication links?
  - How does the system recover from failure? Is this different from initialization?

38

## Documenting System Design



- System design is documented in Software Design Document (SDD)
- It serves as a common reference for the whole development team

39

## Software Design Document – Sample Outline



1. Introduction
  - 1.1 Purpose of the system
  - 1.2 Design Goals
  - 1.3 Definition, acronyms and abbreviations
  - 1.4 References
  - 1.5 Overview
2. Software Architecture
  - 2.1 Overview
  - 2.2 Subsystem Decomposition
  - 2.3 Hardware/software mapping
  - 2.4 Persistent data management
  - 2.5 Access control and security
  - 2.6 Global Software Control
  - 2.7 Exception Handling Strategies
  - 2.8 Boundary Conditions
3. Subsystem
4. Glossary

40