

System Design: Part I

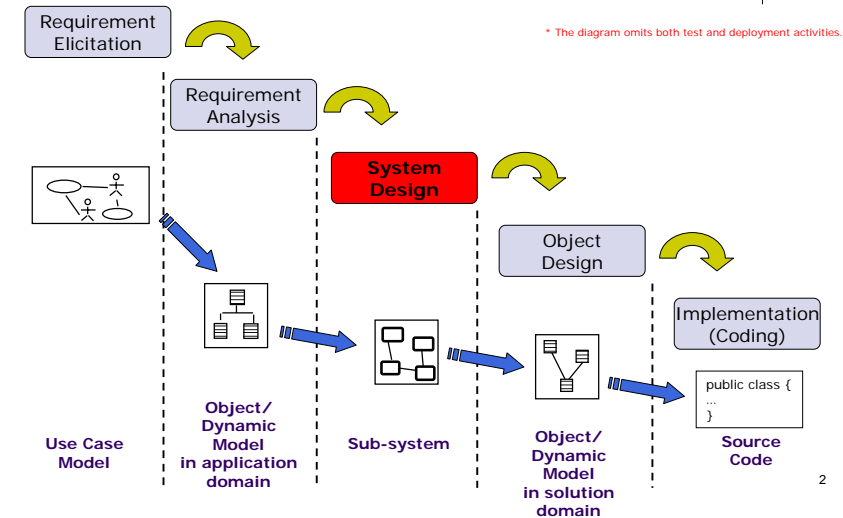
Lecture #07
Software Engineering and
Project Management

Instructed by Steven Choy on Nov 13, 2006



1

Software Development Activities



2

System Design Overview



3

Software Design!

“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.”

- C.A.R. Hoare



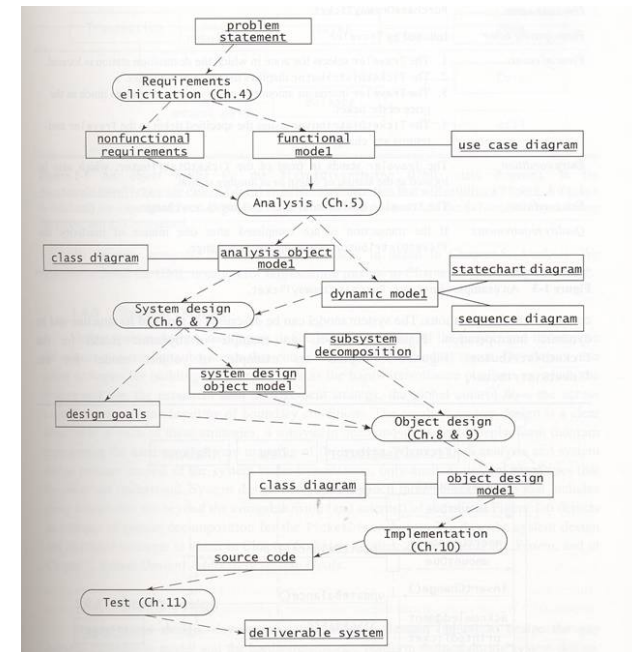
4

Deliverables of Requirement Analysis



- What do we have after the requirement analysis phase?
 - A set of functional and non-functional requirements
 - A set of constraints about the requirement, such as maximum response time
 - A use case model
 - An object model, describing the relationships of objects involved in the system
 - A dynamic model (sequence diagram/statechart)

5



6

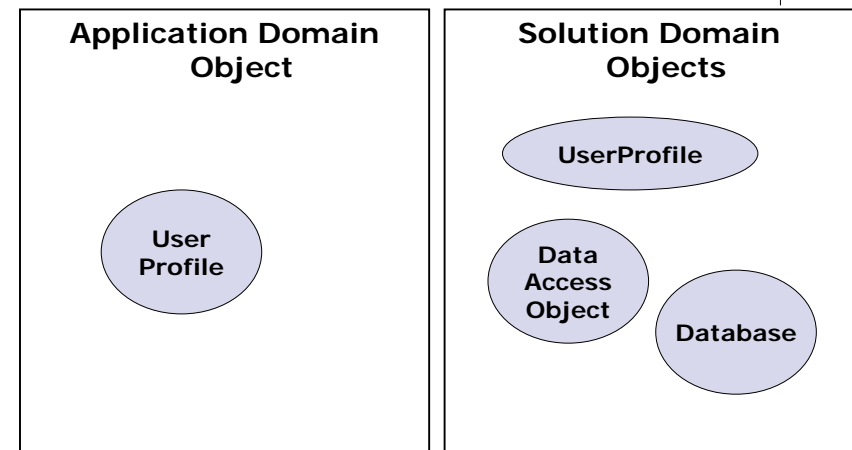
Application Domain Object vs Solution Domain Object



- Application Domain Objects
 - Represent concepts of the domain that are relevant to the system.
 - They are identified by the application domain specialists and by the end users.
- Solution Domain Objects
 - Represent concepts that do not have a counterpart in the application domain
 - They are identified by the developers

7

Application Domain Object vs Solution Domain Object



8

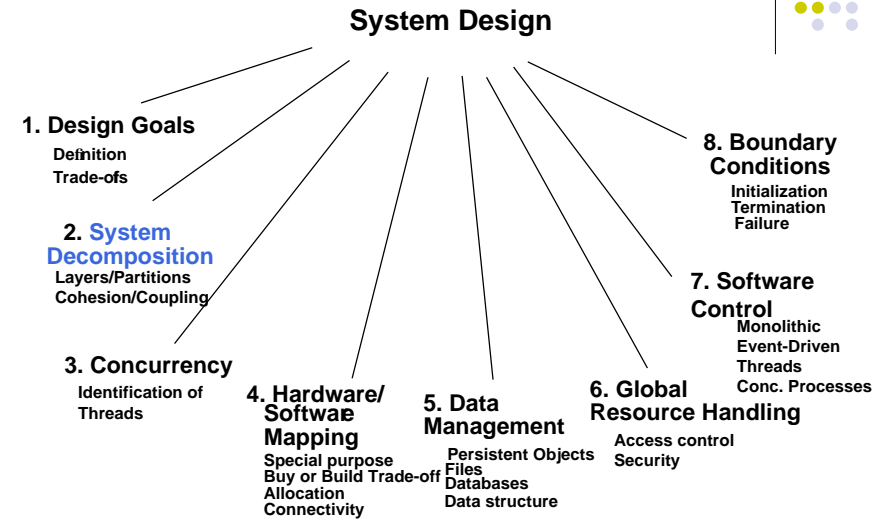
System Design Phrase



- Transform analysis model to system design model
 - Decompose the whole system into sub-systems
 - Software Architectures
 - Identify Design Goals
 - Design strategies for the system include:
 - Hardware/Software strategies
 - Persistent data management strategy
 - Global control flow
 - Exception handling
 - Access control policy

9

System Design Activities



10

Decomposing the System



Identifying Subsystems



- Group functionally related objects into a subsystem
- Subsystem decomposition minimizes **coupling** among objects
- By dividing a system into subsystem, each subsystem can be managed by a developer/small group of developers

11

12

Coupling and Cohesion



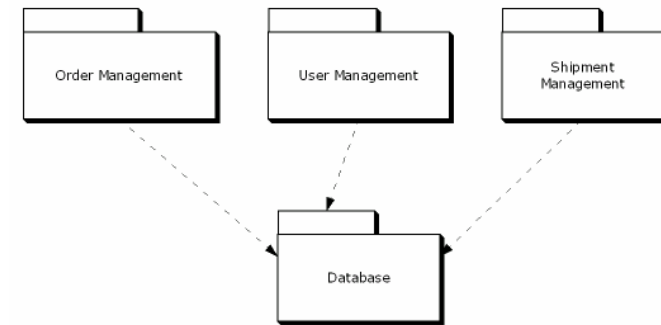
- Coupling refers to the measures of dependencies between sub-systems
 - High Coupling: Changing one sub-system will greatly impact other sub-systems
 - Low Coupling: Changing one sub-system does not/have low impact other sub-systems
- Cohesion refers to measure of the dependencies among classes within a module
 - High Cohesion: Classes are closely related to each other
 - Low Cohesion: A lot of unrelated classes
- Design Goal: High Cohesion and Low Coupling

13

High Coupling Example

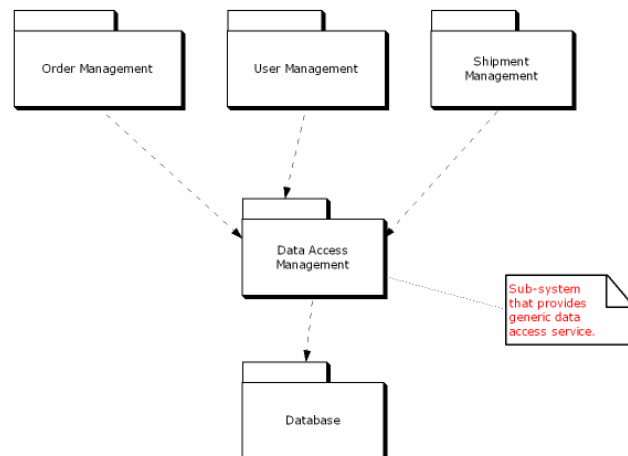


- Think about if we want to change from Oracle database to another database vendor, how many sub-systems are affected?



14

Low Coupling Solution



15

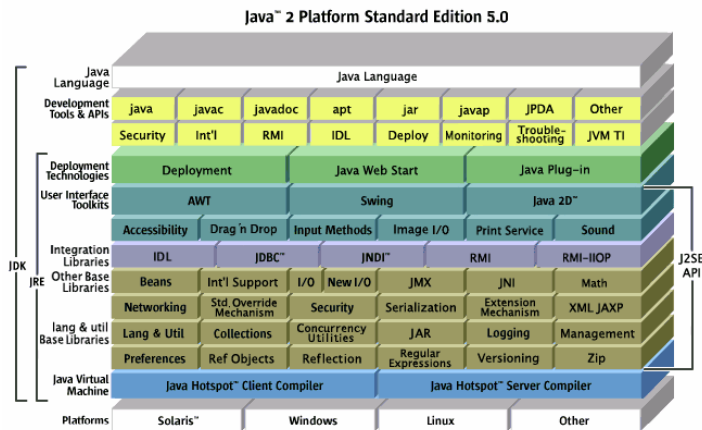
Layer and Partition



- How can we achieve high cohesion and low coupling?
 - Decomposing a system using layer and partition techniques
- Layer vertically divide a system into groups of subsystems providing related services
 - A layer can only depend on its lower layers
 - A layer has no knowledge about its upper layers
- Partition horizontally divide a system into several independent subsystems
 - Sub-systems are freely to use service provided by other sub-systems

16

Layers in Java 2 Platform



17

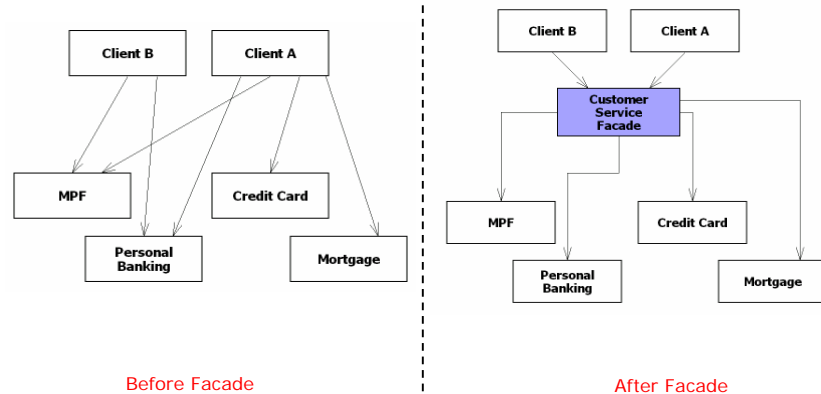
Façade Design Pattern



- One way to reduce complexity and dependencies between subsystems/classes
- What is a pattern?
 - A recurring solution to a common problem in a context
- Façade Design Pattern
 - To provide an unified interface for accessing the underlying subsystems
 - To provide a simpler interface for accessing complex subsystems

18

Façade Design Pattern



Before Façade

After Façade

19

Architectural Styles



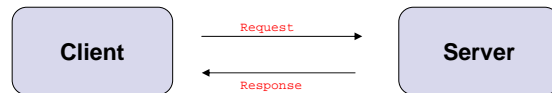
- A "software architectural style" describes how a system is composed and the relationship between its sub-systems
- Common Software Architectural Styles:
 - Client/Server
 - Model/View/Controller (MVC)
 - 3-Tier
 - N-Tier
 - Peer-to-peer
- Adopting an existing and well-known architectural style simplifies the decision of system decomposition

20

Client/Server Architecture



- Server provides service to instances of sub-systems, known as clients
- Clients call on server to perform a task. Server then accomplishes the task and returns the results to client



- The request of a service is done via an interface provided by the server (e.g. Java RMI)
- Often use in database applications:
 - Front-end: User application (client)
 - Back-end: Database (server)

21

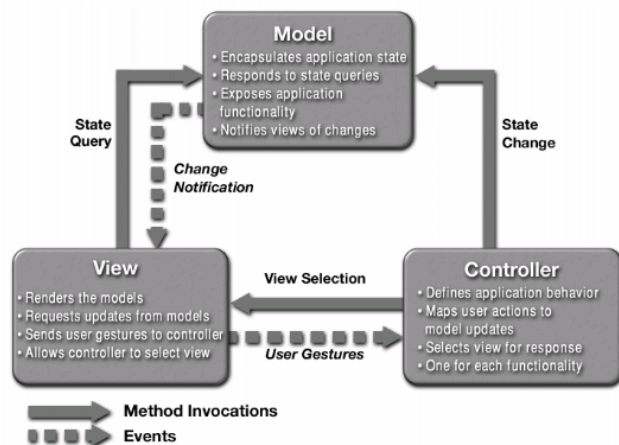
Model/View/Controller (MVC)



- Well-known practice for GUI applications
- For MVC, subsystems are classified into:
 - Model – represents enterprise data and business rules
 - View – represents user interface
 - Controller – controls the interaction between model and view
- MVC allows developer to change the view of the application without impacting the model and vice versa

22

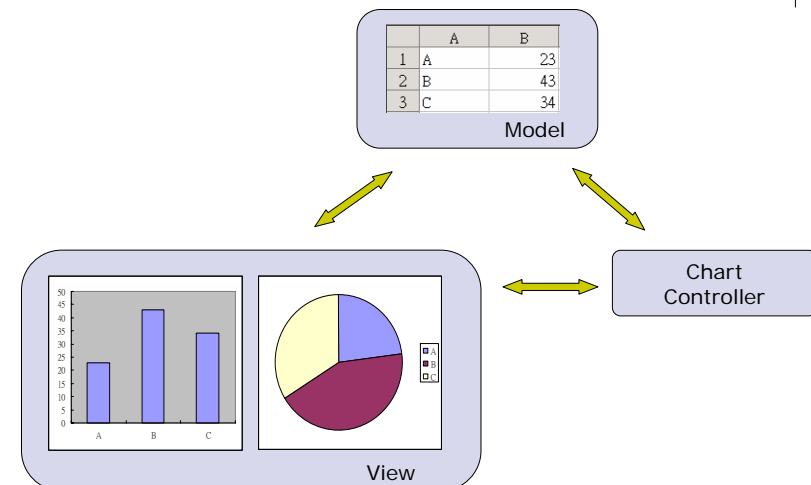
MVC Illustrated



23

Source: java.sun.com

MVC Example

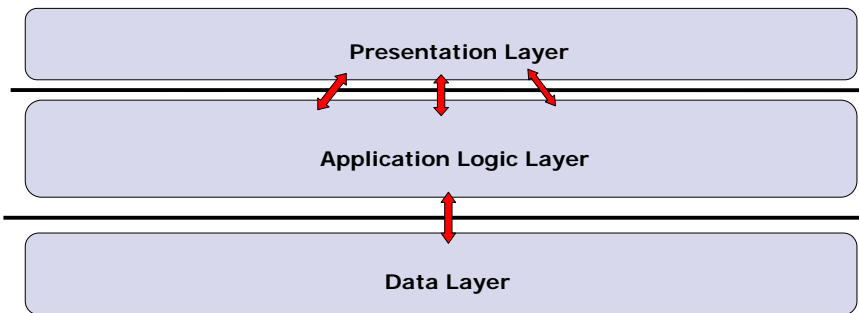


24

3-Tier Architecture

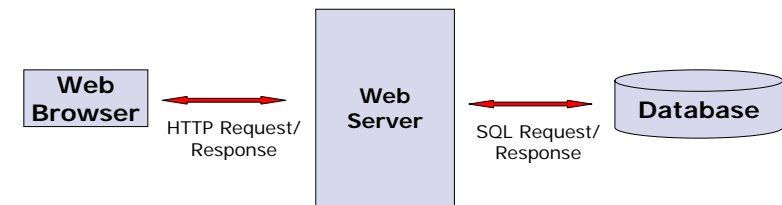


- For 3-tier architecture, the system is divided into three layers: Presentation, Application Logic and data/storage layer



25

Sample 3-Tier Architecture



- Web Browser handles the UI and communicates with web server via HTTP
- Web components (e.g. Servlet) handles business logic and data modeling
- Web components talks to backend database

26

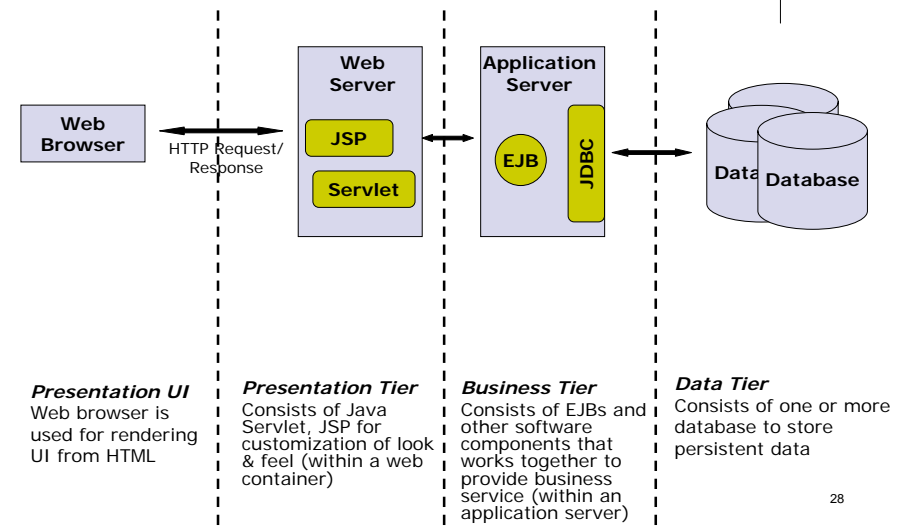
N-Tier Architecture



- An N-Tier architecture adds one or more tiers to three-tier architecture
- Goals:**
 - Allows various tiers to scale independently, without intervene the peer layer
 - Performance problem and errors are localized to a specific tier. Any changes in one tier will not intervene the other layer

27

Sample N-Tier Architecture



28

Peer-to-Peer Architecture



- In client/server architecture, a server is dedicated to provide services
- For peer-to-peer architecture, any subsystem acts as a server (to provide service) and at the same time as a client (to use service)
- E.g. Napster, BitTorrent

29

Design Goals



30

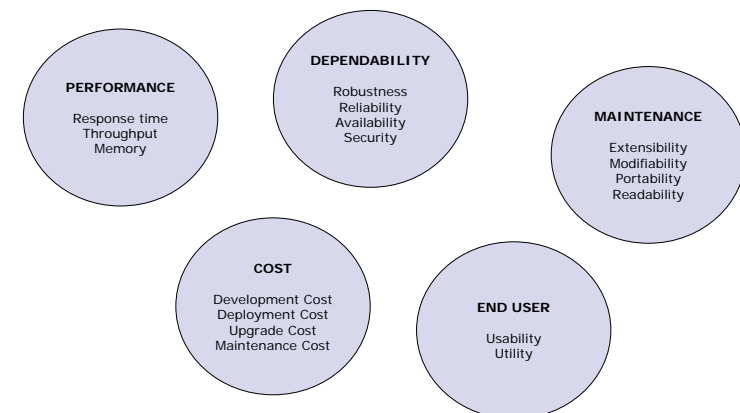
Defining Design Goals



- The very 1st step in system design
- But what does design goal refer to ?
 - A set of **highly desirable** qualities
 - These include performance, dependability, cost, maintenance and end-user criteria
- How do we know about the design goal of the system?
 - From the non-functional requirements
 - From the application domain
 - From the client

31

Quality Criteria



32

Design Goals Trade-offs



- As a software engineer/architect, you are desire to develop a perfect system, that is highly scalable, extensible, ...
- But the real business world does not allow your system to be perfect
- You always need to trade-off

33

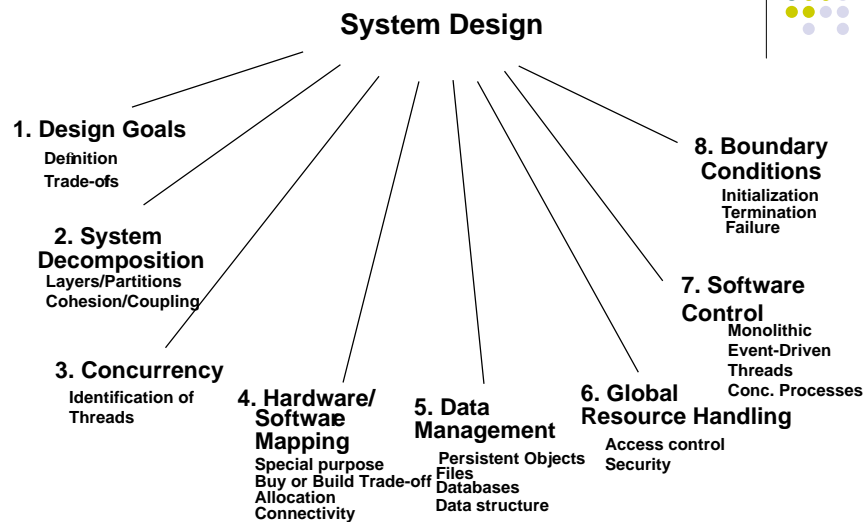
Typical Design Trade-offs



- Functionality vs. Usability
- Cost vs. Robustness
- Efficiency vs. Portability
- Rapid development vs. Functionality
- Cost vs. Reusability
- Backward Compatibility vs. Readability
- Delivery time vs Quality
- Cost vs Performance
- Man Power vs Functionality
- Efficiency vs Adaptability

34

System Design Activities



35

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

36