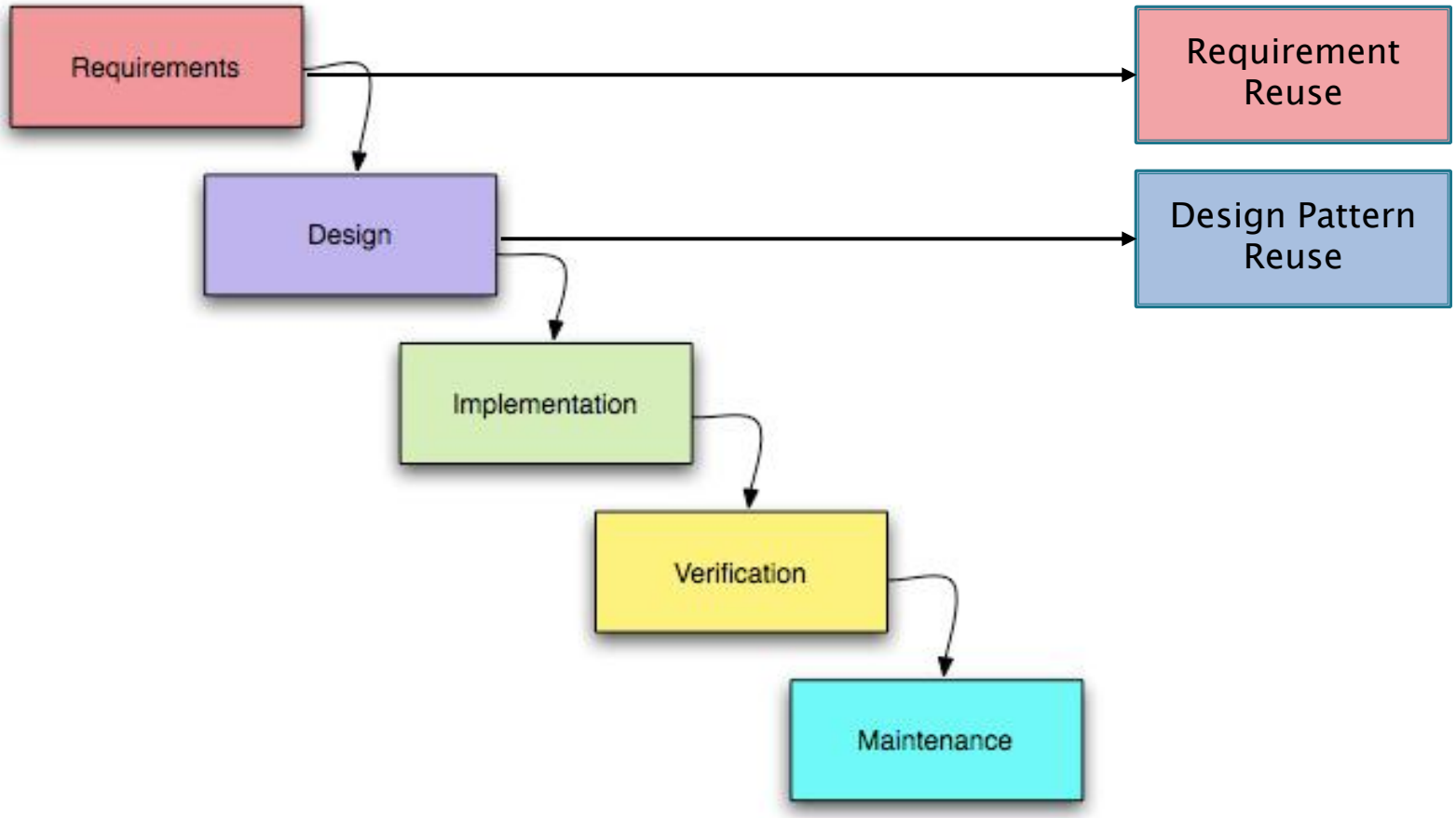
Module 3
Design Pattern Reuse (Part 1)

Dr. Shiping Chen

Where are we?



Outline

- What is Software Architecture?
- Roles of Software Architects
- Architectural Abstraction
- UML Basics
- Common Enterprise Design Patterns
  - The most common/important design pattern: Layering
  - The Model-View-Controller (MVC) Pattern

What is Software Architecture?

- The **software architecture** of a system is the set of structures needed to reason about the system, which comprise *software elements*, *relations* among them, and *properties* of both.

From: Clements, Paul, Felix Bachmann, Len Bass, *et al* (2010). *Documenting Software Architectures: Views and Beyond, Second Edition*. Boston: Addison-Wesley.

Software Element

- Depending on the applications, software elements can be (*but not limited*) as follows:
  - The logic functional components (class, component, services etc.)
  - Operation Systems: Windows vs. Linux
  - Middleware: .NET vs. J2EE
  - Programming Languages: Java, C#, C/C++, Python, PHP
  - Library/Drivers:
    - static lib (*.o) vs. dynamic shared lib (*.so)
  - User interfaces:
    - Web vs. Desk App vs. Mobile App
    - Language-based vs. GUI
  - Security policies: XACML
  - Data Model: structural vs. non-structural,

Relations among them

- Why using these software elements?
- What dependences between them?
- How they interact each other?
- What communication protocol used ? why?
- What are alternatives?
- …

Properties of Both

- For example,
  - Operation Systems……What version
  - Middleware……..Which platform
  - Programming Languages…….Java or C#? what version?
  - Library/Drivers……Which one? What version?
  - Communication protocols……Which protocol used?
  - Classes……..External & International API
  - User interfaces……What looks like? Event?
  - Data……What database? Schema?

Key Architecture Principles

- Build to change instead of building to last
- Model to analyze and reduce risk
- Use models and visualizations as a communication and collaboration tool
- Identify key engineering decisions

Why Software Architecture?

- Building software systems, especially large-scale distributed enterprise systems, induces a lot of complexities
  - Diversity of parameters and unknowns (see next slide)
  - Diversity of objectives (see next slide)

Roles of Software Architects 1/5

- A diversity of parameters and unknowns:
  - Technical team:
    - Profile and evolution of skill sets
    - Actual productivity, learning ability, task switching agility
    - Employment policy of technical team
    - How about outsourcing alternatives?
  - Evolution of technology landscape:
    - Will technologies X and Y be still supported in 3 years?
    - What if next version of Y presents backward compatibility issues, as seen in the past from that same vendor?
  - Enterprise resource planning:
    - What frequency/range/priority of upgrade can be assumed with this company's hardware/software?

Someone has to handle the complexity, i.e. Architects! 2/5

- The role of a project manager is to handle planning-related complexity
- The role of a sales manager is to handle customer-related complexity
- The role of a software/system architect is to handle '*design-related*' complexity by making right decisions!

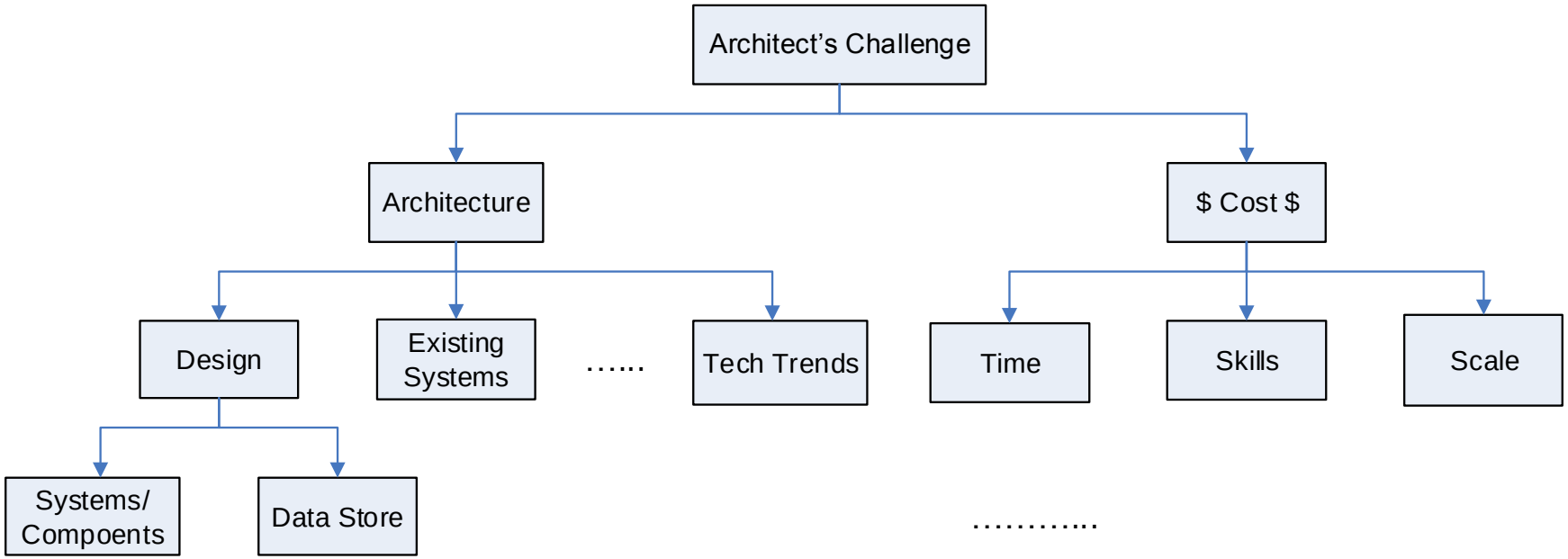
Roles of Software Architects 3/5

Architect's Concerns and Responsibilities:

- A diversity of requirements and objectives
  - Functional requirements:
    - Usage scenarios, business logics (transactions), business processes…
  - Non-functional requirements (Capacities/Qualities of an Architecture):
    - **Security objectives**: Identification, confidentiality, proof of origin, policy enforcement…
    - **Performance/Scalability**: Response time, throughput, fairness…
    - **Availability/Reliability**: 24 hours x 7 days, disaster recovery, robustness to extremely load…
    - **Usability**: Look-and-feel, media and platforms for delivery, localization…
    - **Evolution objectives**:
      - Compatibility/portability, manageability, maintainability, extensibility…

Roles of Software Architect 4/5

- Architect's concerns and responsibilities should NOT be confused with a designer's:



Based on 'SUN Certified Enterprise Architecture for J2EE' by Paul Allen and Joseph Bambara

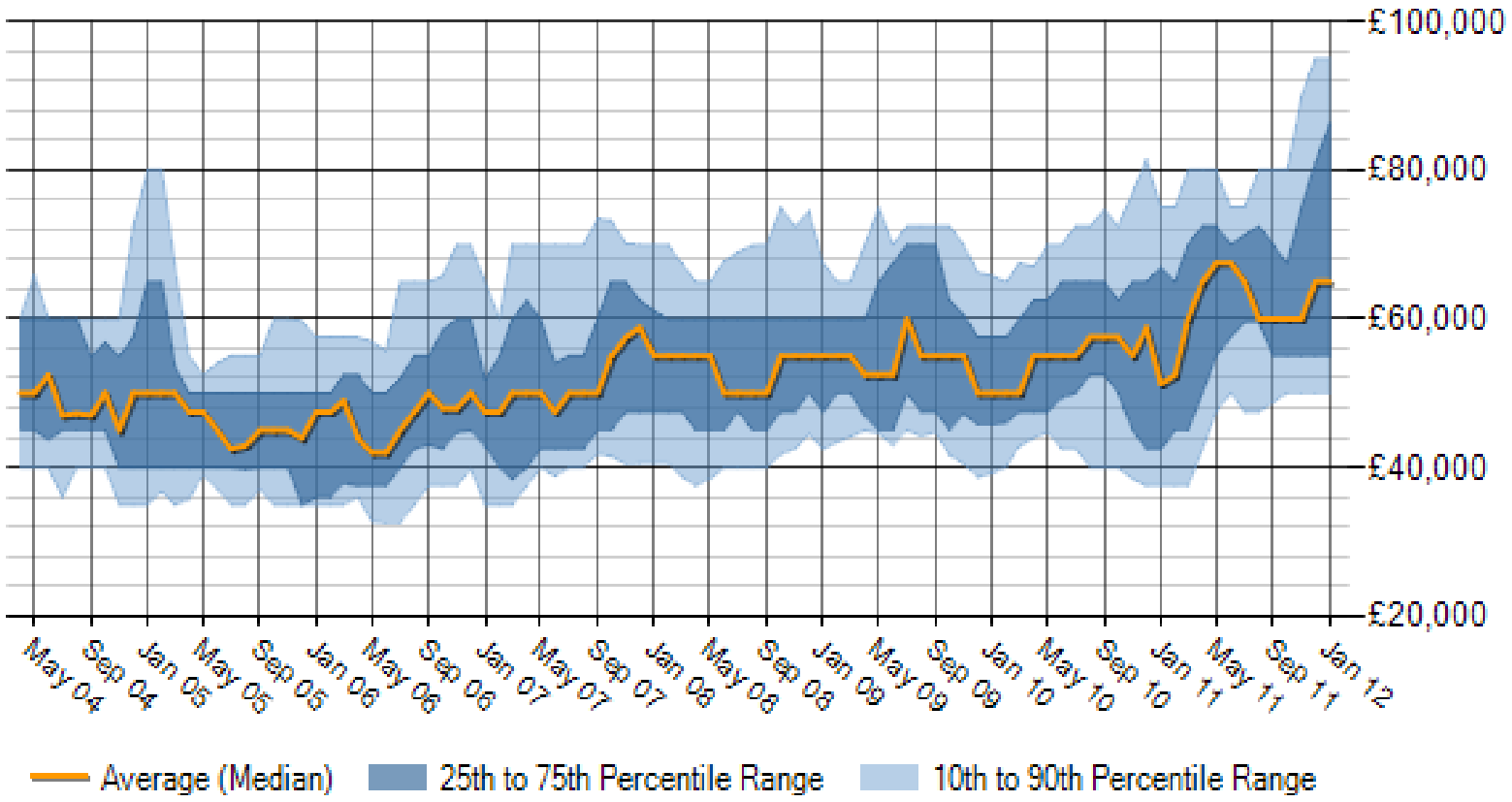
http://www.mhprofessional.com/downloads/products/0072226870/0072226870_ch02.pdf

Roles of Software Architect 5/5

- Architect's concerns and responsibilities should NOT be confused with *the other Roles in a Technical Team:*

Architects who have to make tough technical decisions at early stage of a project to span almost all these aspects...	Business Analysis: Business process analysis, pre-sale prototyping
	Quality management: User Acceptance Test Plan, internal audit plan, …
	Developers: Component design and coding...
	Testers: unit testing, performance testing, integrating testing, etc.
	DBA: Design and manage database...

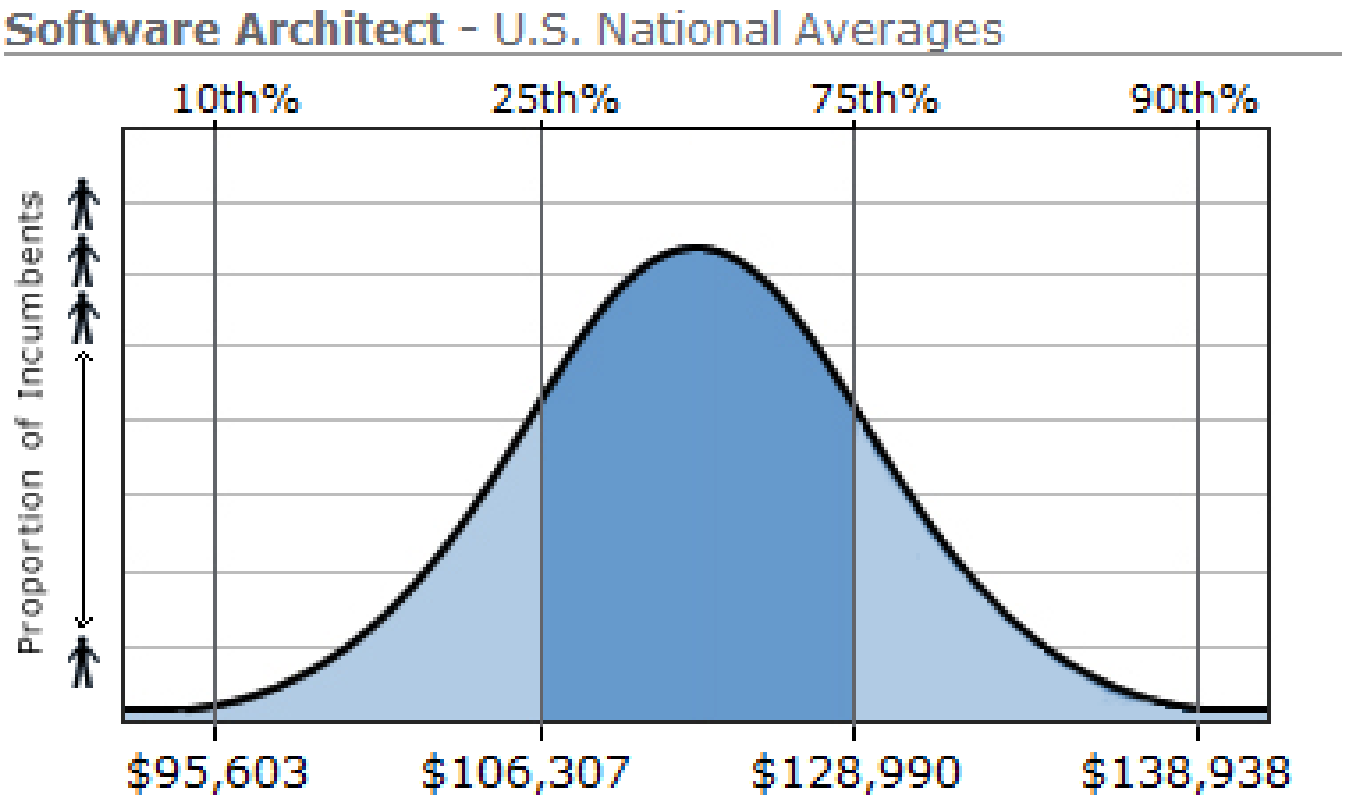
UK Software Architects' Salary Trends



From: http://www.itjobswatch.co.uk/jobs/uk/software%20architect.do

UK£1 ≈ RMB¥9.9762

US Software Architect Salary



Source: http://www1.salary.com/Software-Architect-Salary.html

Abstraction 1/2

- *What is Abstraction?*
  - Abstraction is a process of
    - Observing practical instances …
    - … and from there identifying, extracting and then formalising qualities which seem to apply to all/most instances
  - Abstraction also refers to the outcome of the process itself
    - Generic/formal qualities, properties, principles, …
- *The Benefits of Abstraction?*
  - Remove complicity;
  - Capture keys
  - Enable software reuse

How to be Abstract 2/2

- Rely on abstraction in order to
  - Compare **existing** systems, and development processes of **existing** systems
  - For a new project, determine which port is ***similar to*** other successful projects
  - For a new project, identify potential technical **risk** – i.e. design objectives which have never been encountered or solved successfully
- Select the right level of abstraction
  - A hierarchy of components potentially leads to indefinite refinement …
  - This is where the software/system architect has to be pragmatic! by *choosing the right level of abstraction* according to communication and technical objectives
- Select a suitable style for conveying abstraction
  - The architect should use existing conventions, methodologies or (software) tools, such as UML

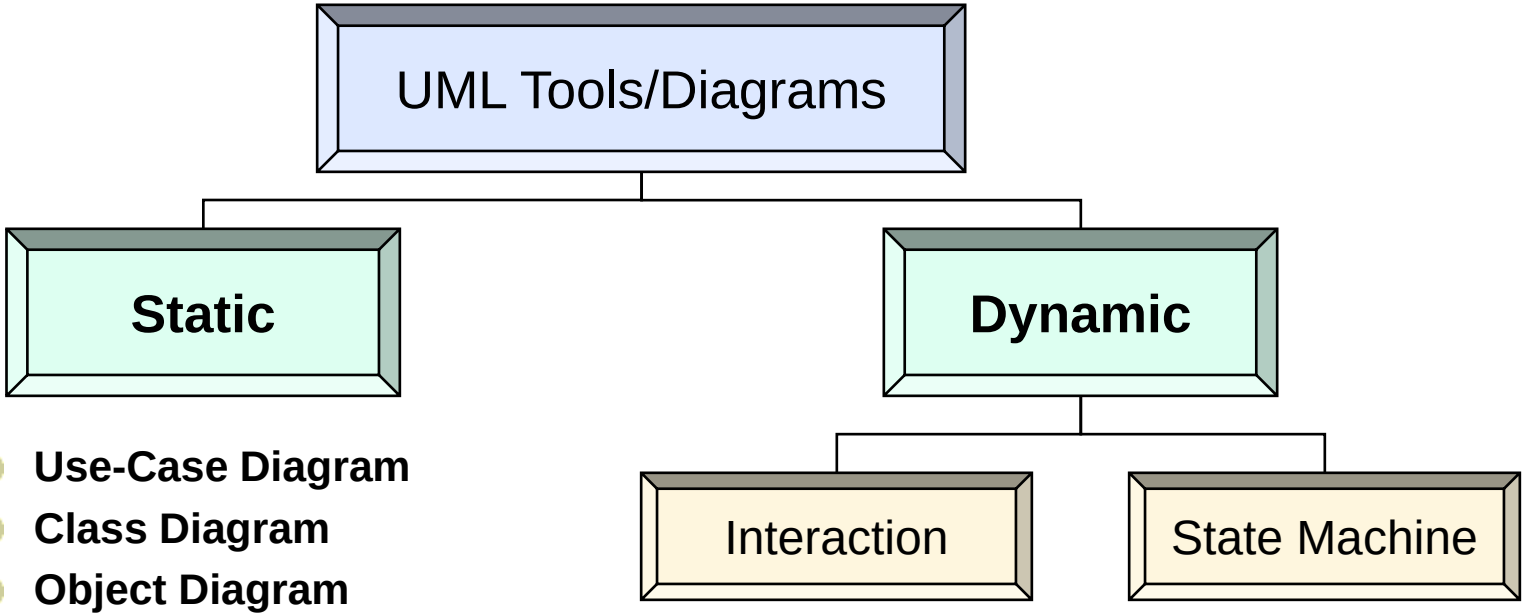
An example of Identifying Reusable Components

	EC-1	EC-2	NC-1	NC-2
R1	√			
R2			√	
R3		√		
R4			√	
R5	√			
R6				√
R7			√	

Where R = Requirement
EC = Existing Component
NC = new Component

UML Basics

The UML Taxonomy



- **Use-Case Diagram**
- **Class Diagram**
- **Object Diagram**
- **Component Diagram**
- **Deployment Diagram**

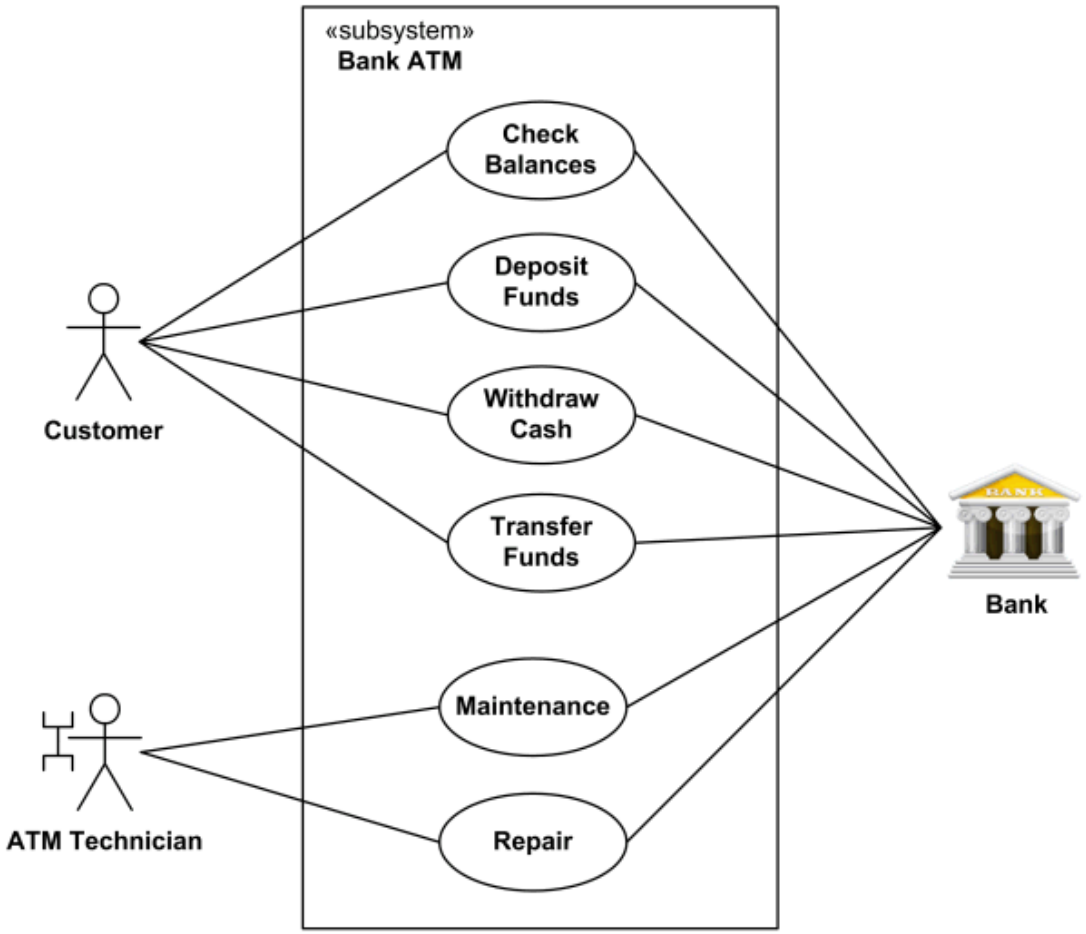
- **Sequence Diagram**
- **Collaboration Diagram**

- **State Chart Diagram**
- **Activity Diagram**

Source: http://people.csa.iisc.ernet.in/chkalyan/notes3.htm

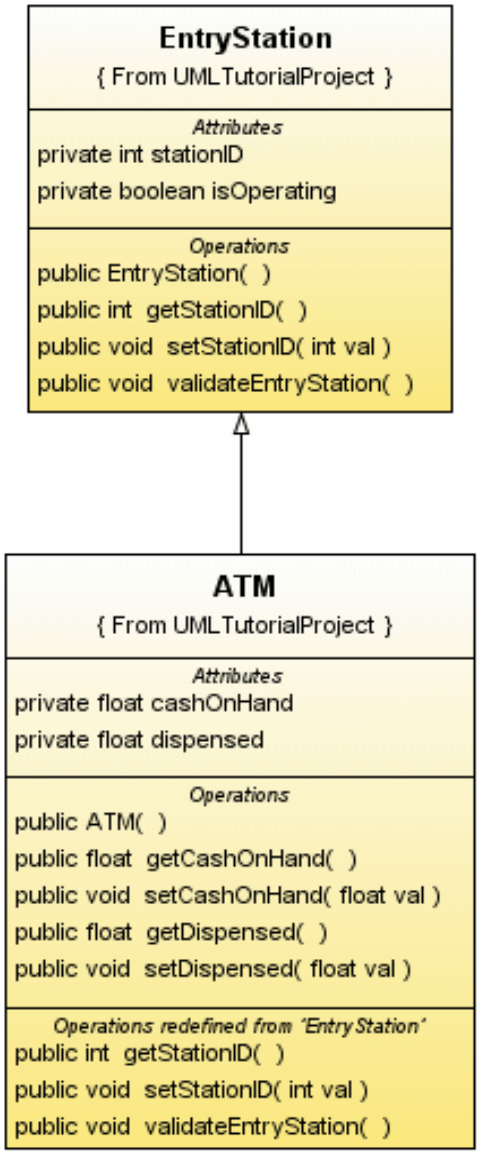
Use-Case Diagram

- Used to capture key components (including users) and their relationships in a system

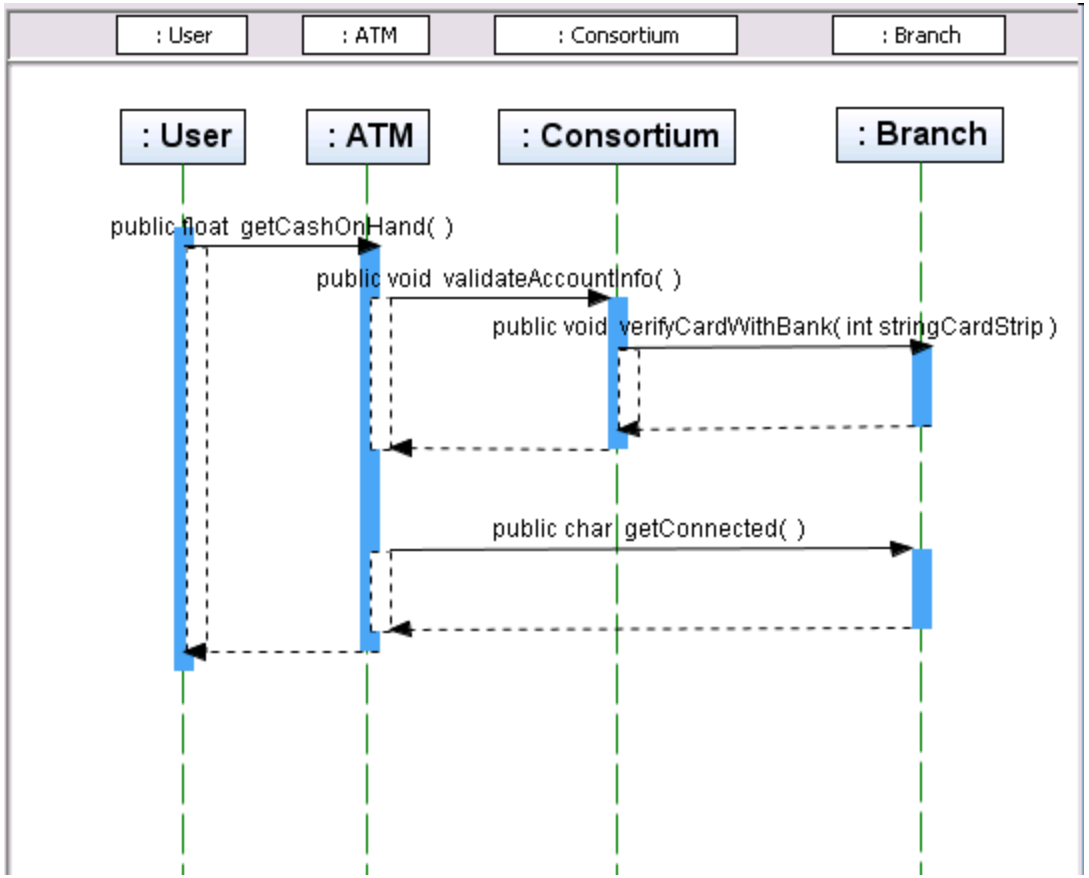


Class Diagram

- To model detailed operations of each class and their relationships

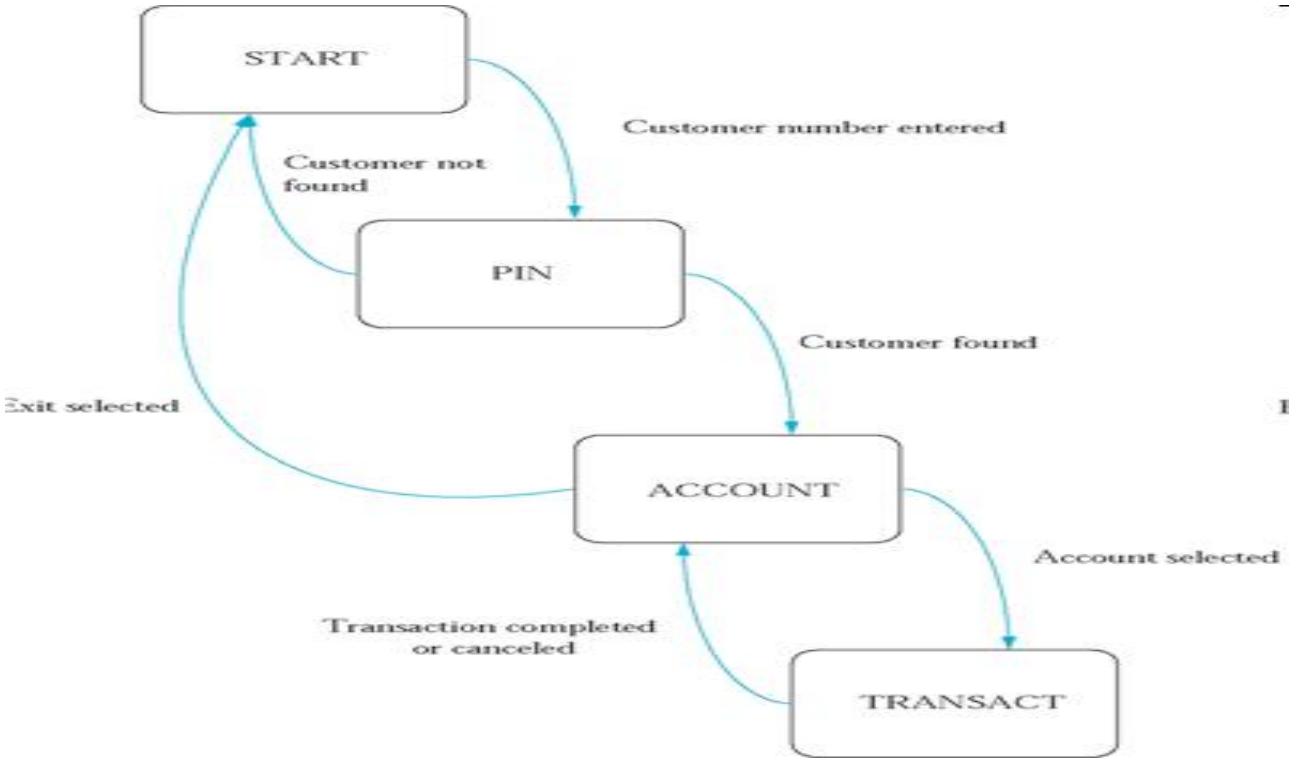


Sequence Diagram



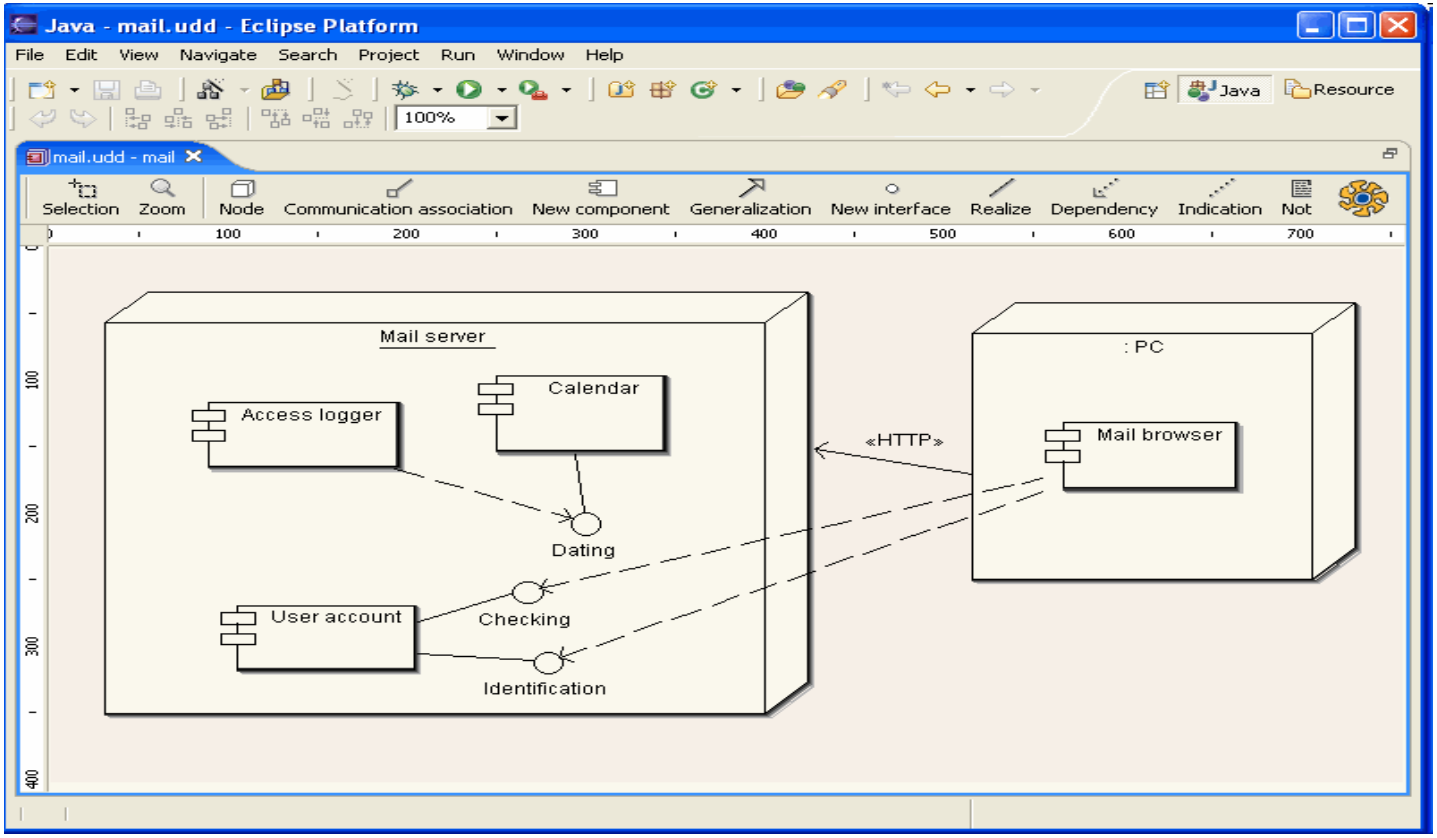
- To capture the detailed interactions between components

State Diagram



- To capture the states in a system

Deployment Diagram



- To model how to deploy a software system into a set of physical hardware

UML Resources

- UML Official Site: http://www.uml.org/
- UML Tutorial by IBM: http://www.ibm.com/developerworks/rational/library/769.html
- UML Tutorial Video: http://www.youtube.com/watch?v=FkRwbVUVFvE
- UML Tools:
  - Commercial: IBM Rational Rose & Microsoft Visio
  - Open sources: ArgoUML http://argouml.tigris.org/

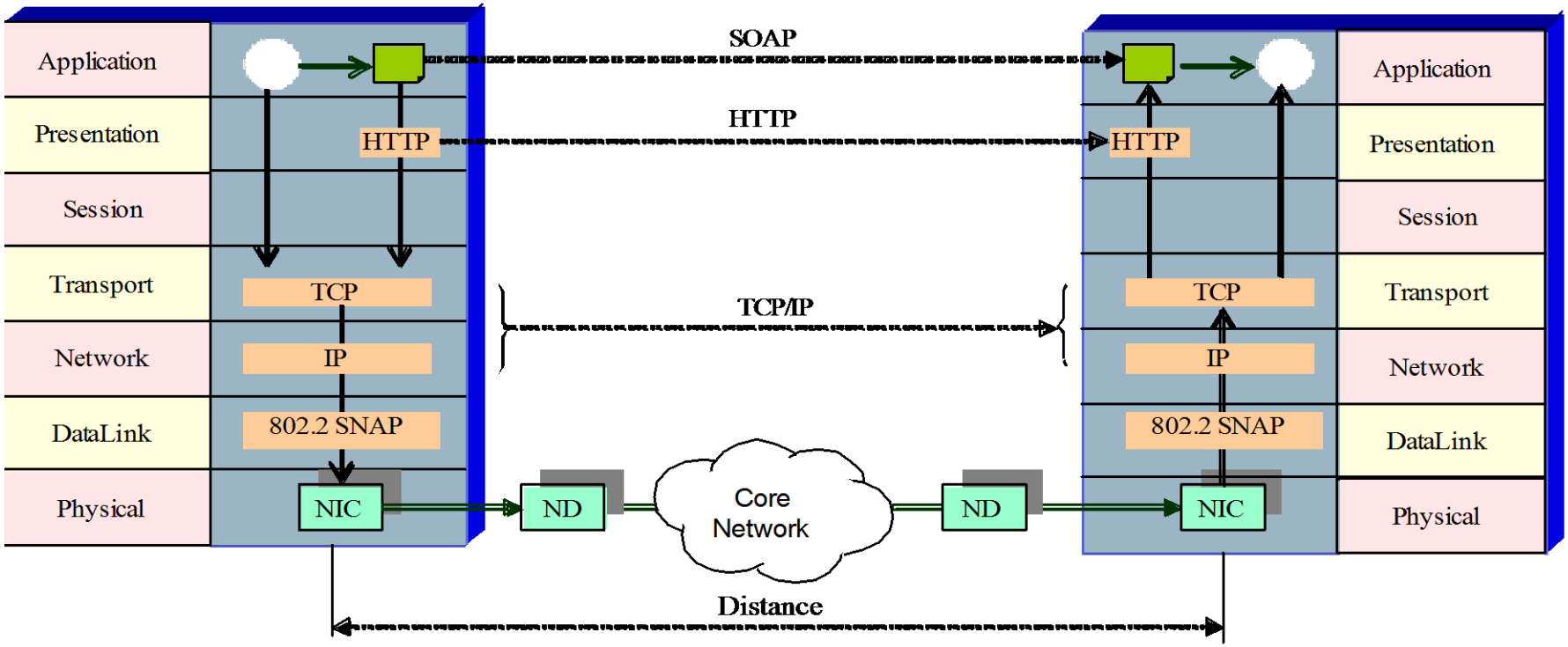
Design Patterns

- What?
  - A general *reusable* solution to a *commonly occurring* problem within a given context in software design
- Why?
  - Common requirements
  - Proved mature solutions
  - Can be reused/repeated!

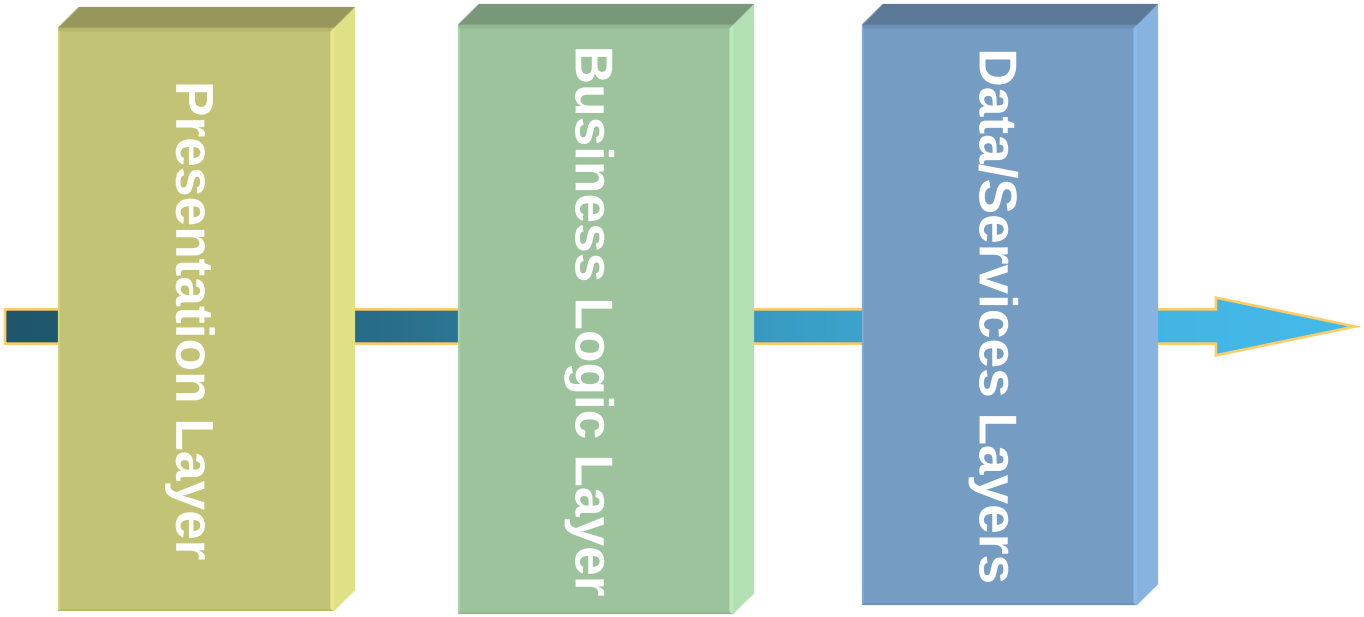
Classification of Design Pattern

- System level design patters, such as:
  - Layering
  - MVC
  - Session State Management
  - Messaging
- Component level design patters, such as:
  - Creational design patterns: How to create objects?
  - Structural design patterns: How to construct a system with objects/components?
  - Behavioural patterns? How to control objects at runtime?

Architecture Pattern 1. Laying Pattern – *From Vertical Viewpoint*



Architecture Pattern 2. Laying Pattern
–*From Horizontal View*

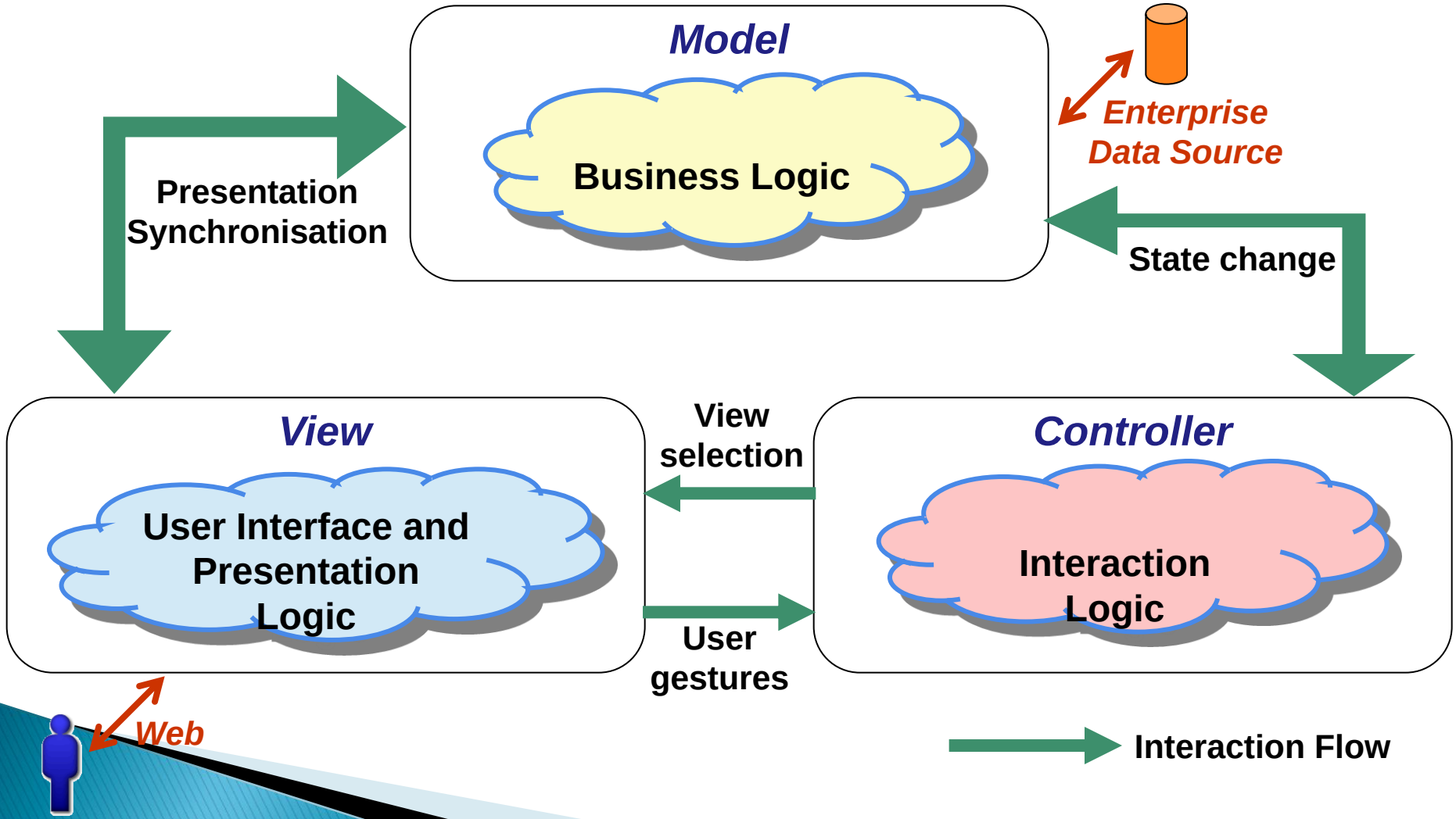


- With the multi-tiers (layers), we can:
  - Separate/isolate complexities from each other
  - Maximize reuse of business components & data
  - Making it easier to manage and scale

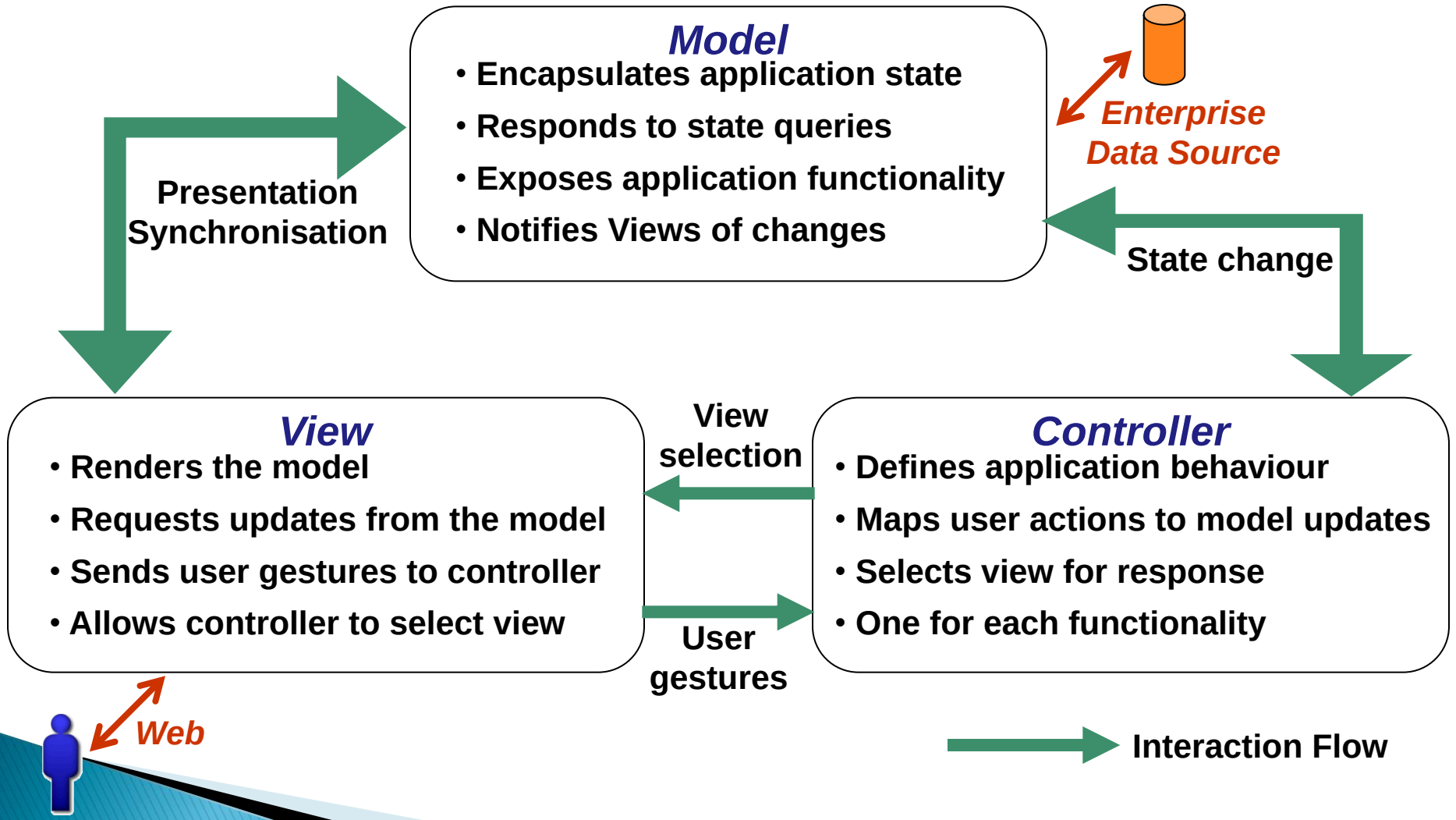
What we can get from laying pattern?

- Understand a single layer as a coherent whole without knowing much about the other layers.
- Substitute layers with alternative implementations of the same basic services.
- Minimize dependencies between layers.
- Standardize each layer's interfaces to its upper layers
- Use the standard interfaces for many/various higher-level services.

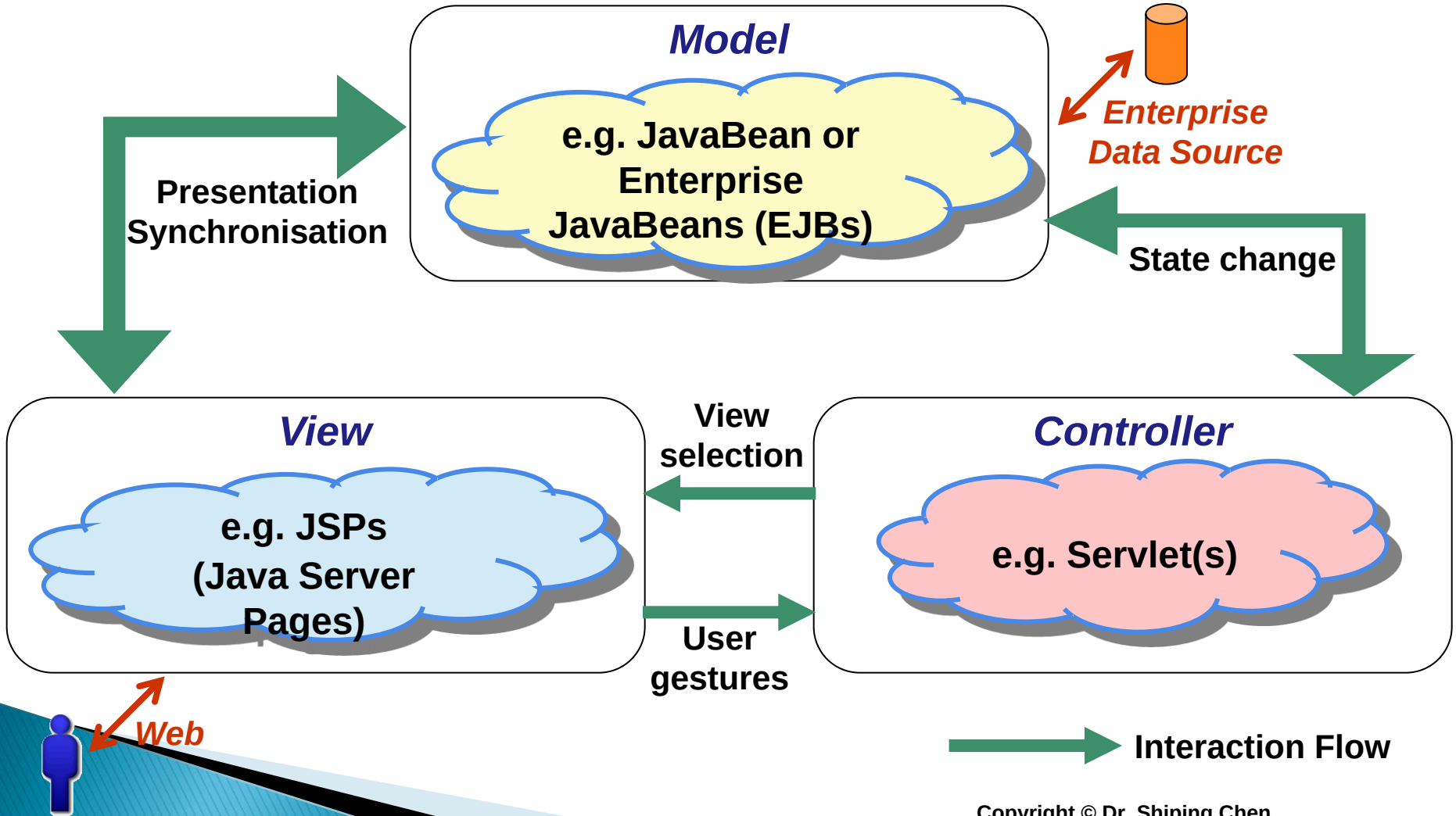
Architecture Pattern 3 - MVC Pattern



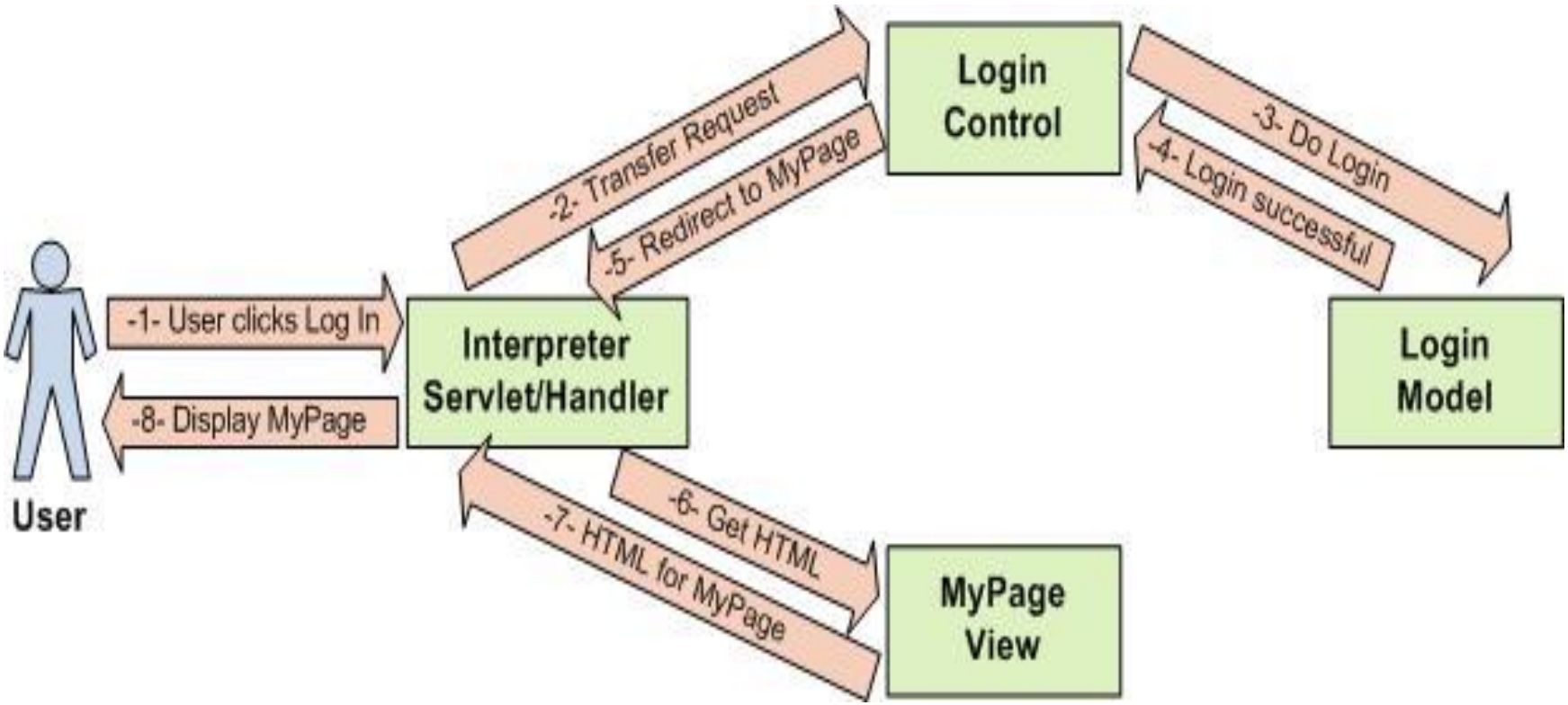
Why MVC?



MVC Pattern in J2EE



An Example of MVC: Login



MVC Resources

- JAVA's MVC
  - Struts: http://jakarta.apache.org/struts/
  - String: http://www.springsource.org/
- Microsoft's MVC
  - http://www.asp.net/mvc
  - http://www.asp.net/mvc/tutorials