基于面向对象的架构设计
Architectural Design

Yang YI, Ph.D

issyy@mail.sysu.edu.cn

Architectural Design

Agenda

- Objectives(本章教学目标)

- Context in the RUP(在整个设计流程中的位置)

- Architectural Design steps(步骤)

- Exercises (实验任务)

Objectives: Architectural Design

- Explain the purpose of Architectural Design and where it is performed in the lifecycle.
  - (架构设计的目标及其在软件生命周期中的位置)
- Describe a representative architectural pattern and set of analysis mechanisms, and how they affect the architecture.(一些相关的概念)
- Describe the rationale and considerations that support the architectural decisions.(架构设计中常见的错误)
- Show how to read and interpret the results of Architectural Design:(步骤)
  - Architectural layers and their relationships
  - Key abstractions

Architectural Design

Agenda

- Objectives

- Context in the RUP

- Architectural Design steps

- Exercises

架构设计这个概念的简单解释

- 凡事皆有章法，做软件系统，怎么分工、怎么编码、怎样分配工作等等，需要若干解决策略，统称为架构

- 架构这个事情，在很多领域都有，例如
  - 我们大学是一个本硕博培养体系的架构
  - 中国大学本科的学制是四年制，大一、大二、大三、大四都有一些他一般性的学习和实践安排

What is Software Architecture?

- The process of defining a solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability.

- It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

- Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman derived and refined a definition of architecture based on work by Mary Shaw and David Garlan (Shaw and Garlan 1996).

- Their definition is:

- "Software architecture encompasses the set of significant decisions about the organization of a software system, including

  – the selection of the structural elements and their interfaces by which the system is composed;

  – behavior as specified in collaboration among those elements;

  – composition of these structural and behavioral elements into larger subsystems;

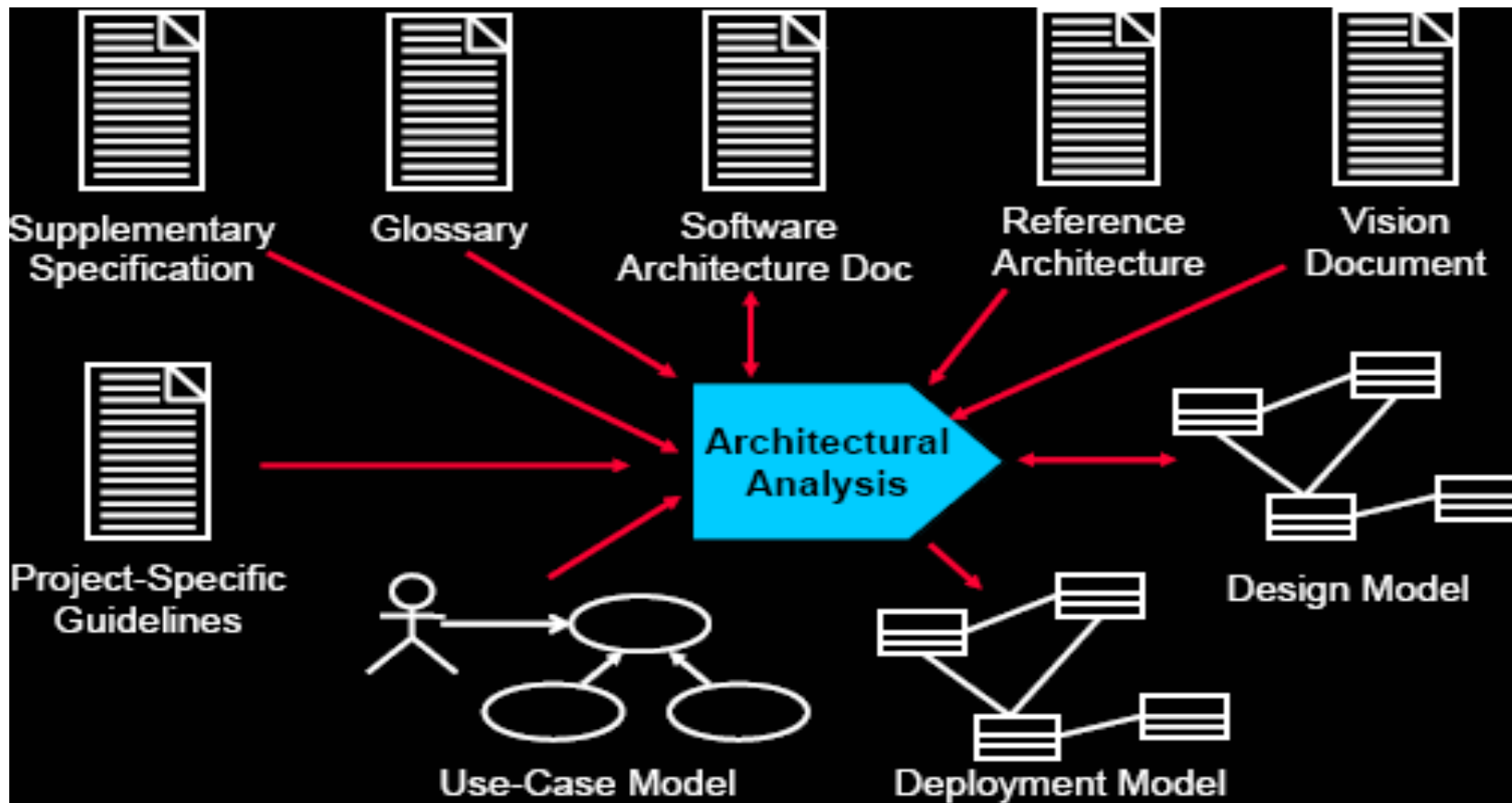 – an architectural style that guides this organization.

- Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns

Architectural Design in Context of System Design
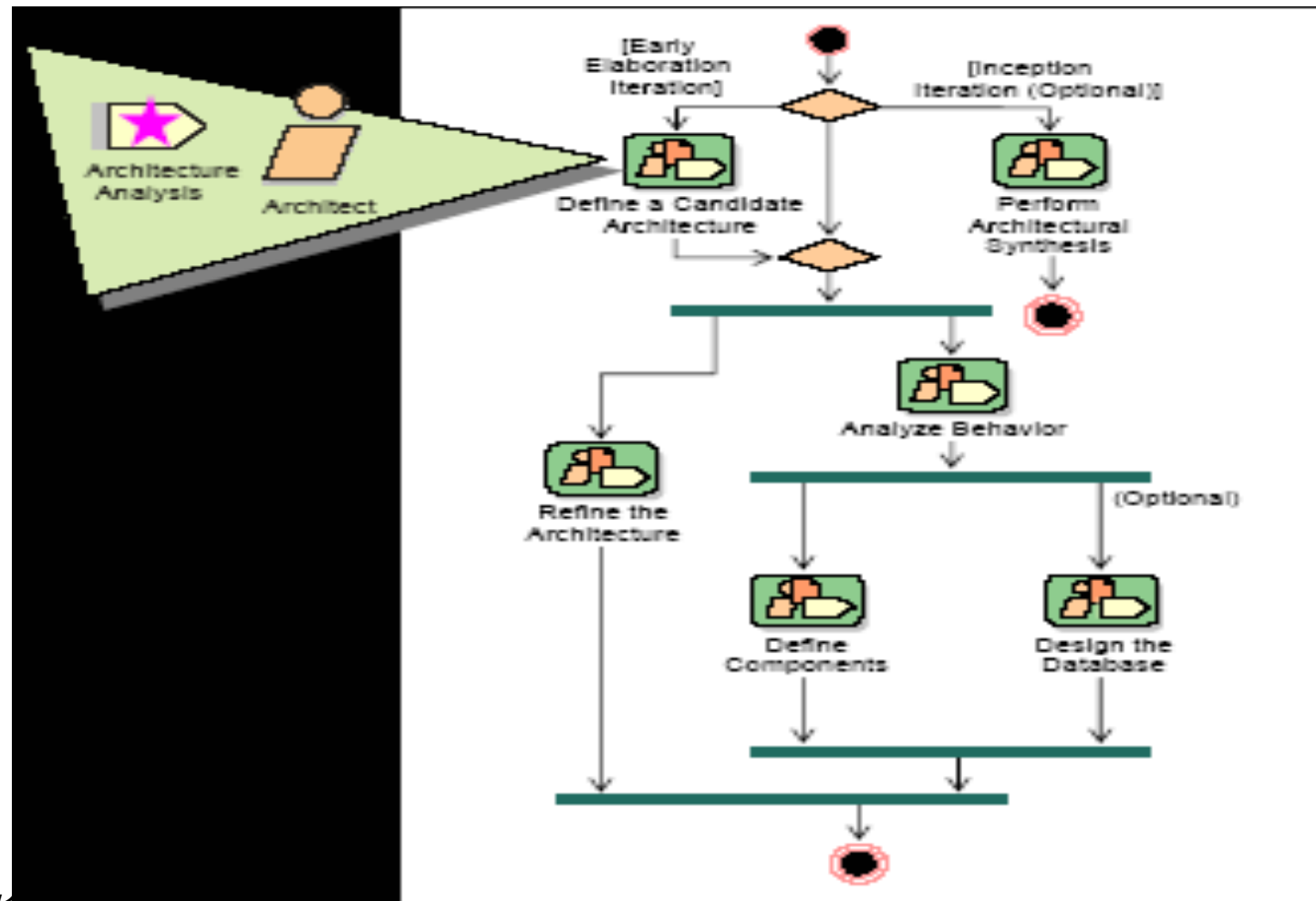
Architectural Design Overview

架构设计涉及的制品

Architectural Design

Agenda

- Objectives

- Context in the RUP

- Architectural Design steps

- Exercises

Architectural Design Steps（步骤）

1. **Key Concepts**

2. **Define the High-Level Organization of Subsystems**

3. **Identify Key Abstractions**

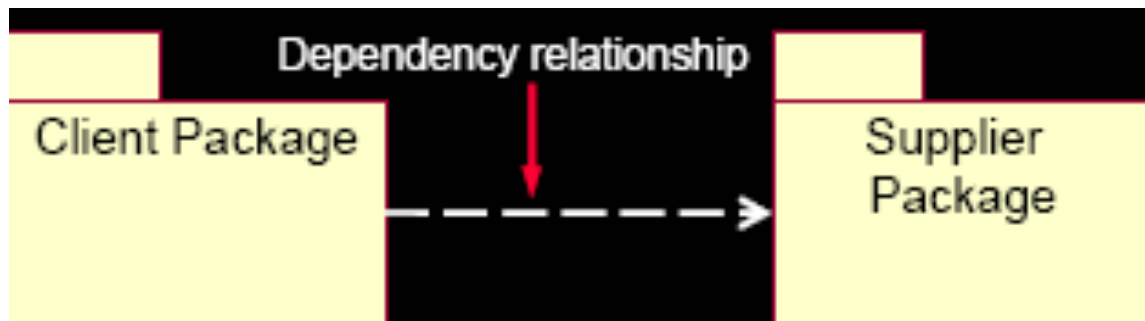4. **Checkpoints**

Review: What Is a Package?

- **A package is a general-purpose mechanism for organizing elements into groups.**

- **It is a model element that can contain other model elements.**

- **A package can be used**

  - **To organize the model under development.**
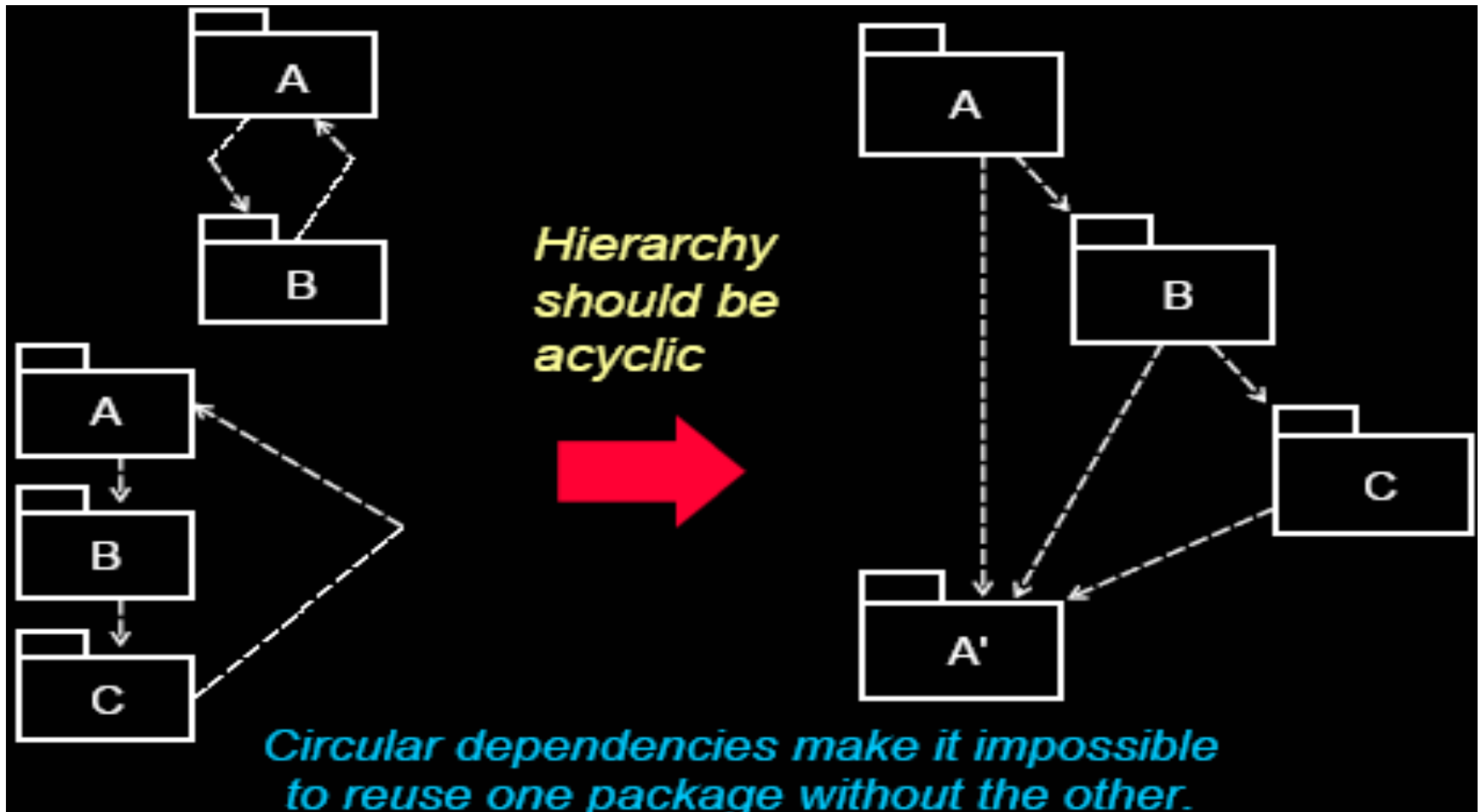
  - **As a unit of configuration management.**

University Artifacts

Package Relationships: Dependency（*依赖关系*）

- **Packages can be related to one another using a dependency relationship.**

- **Dependency Implications**
  - **Changes to the Supplier package may affect the Client package.**
  - **The Client package cannot be reused independently because it depends on the Supplier package.**

Avoiding Circular Dependencies

Hierarchy should be acyclic

Circular dependencies make it impossible to reuse one package without the other.

Architectural Design Steps

1. **Key Concepts**

2. **Define the High-Level Organization of Subsystems**

3. Identify Key Abstractions

4. Checkpoints

Patterns and Frameworks

- Pattern（模式）

    - Provides a common solution to a common problem in a context

    - Analysis/Design pattern

    - Provides a solution to a narrowly-scoped technical problem

    - Provides a fragment of a solution, or a piece of the puzzle

- Framework（框架）

    - Defines the general approach to solving the problem

    - Provides a skeletal solution, whose details may be analysis/Design patterns

Architecture/Pattern/Framework

名词	中文释义	功能	范围	是否有代码实现？
Architecture	架构	说明 方案	全局	无
Pattern	模式		局部	无
Framework	框架		局部	有部分

What Is a Design Pattern?

- **A design pattern is a solution to a common design problem.**
  - **Describes a common design problem**
  - **Describes the solution to the problem**
  - **Discusses the results and trade-offs of applying the pattern**
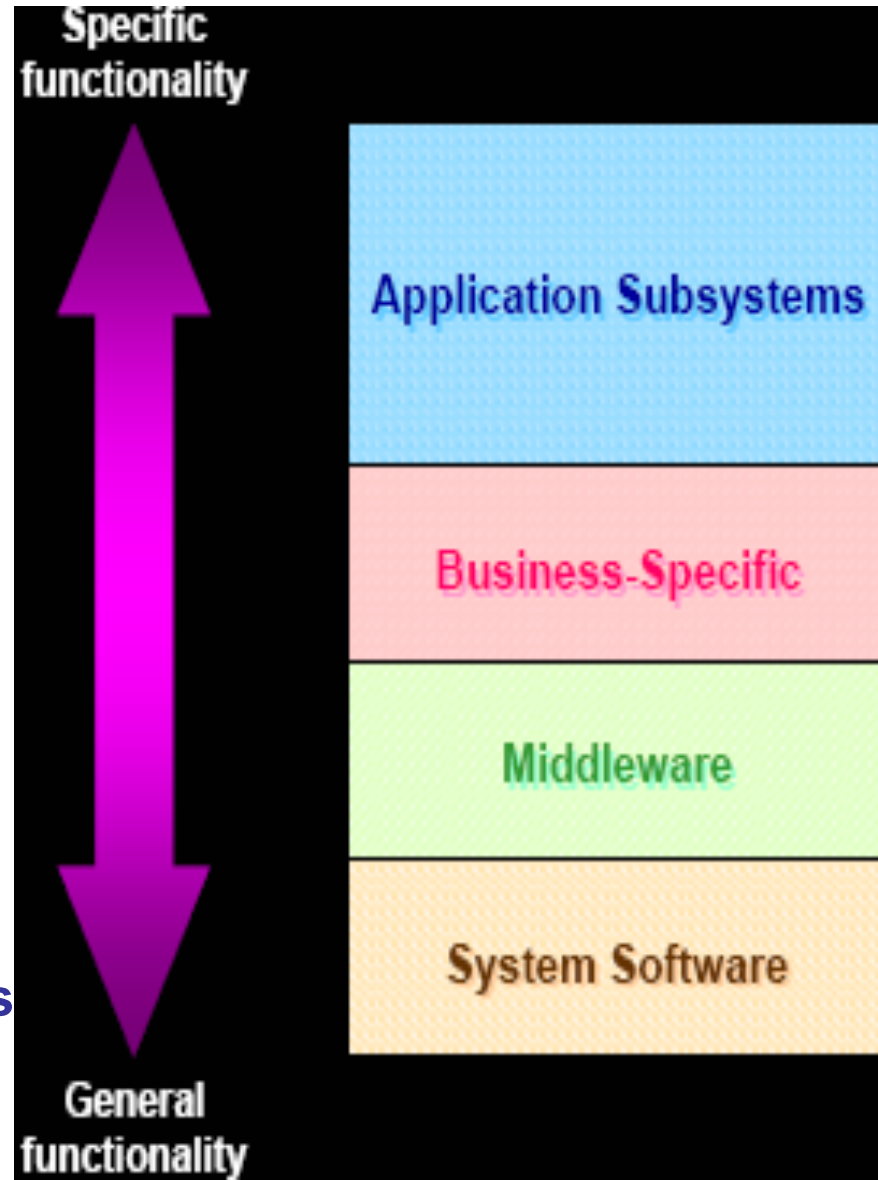- **Design patterns provide the capability to reuse successful designs.**

Template Parameters

Pattern Name

Parameterized Collaboration

Structural Aspect

Behavioral As...

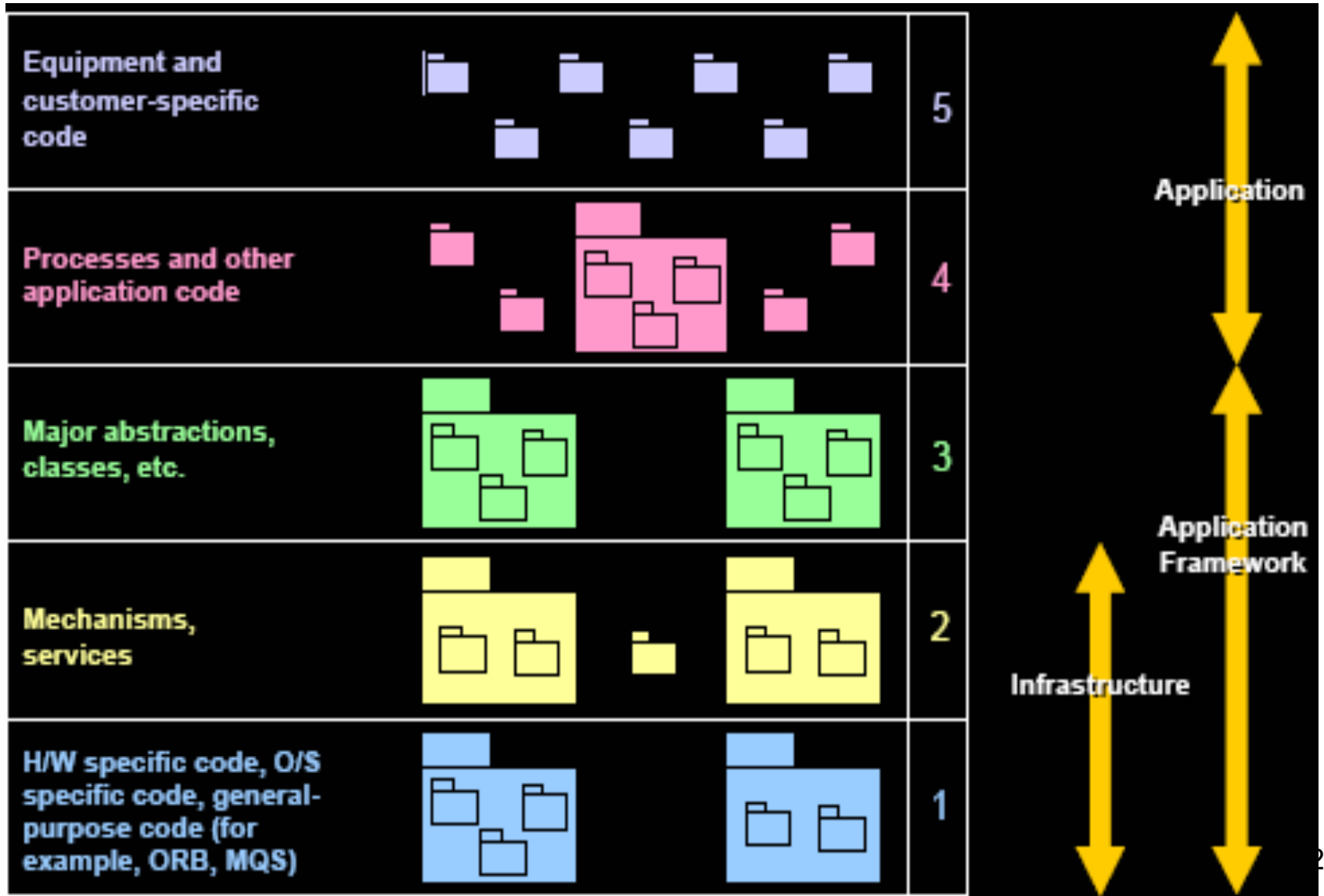# What Is an Architectural Pattern?

- An architectural pattern expresses a fundamental structural organization schema for software systems.

- It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them
    - – *Buschman et al, "Pattern-Oriented Software Architecture — A System of Patterns"*

➢ Layers

➢ Model-View-Controller (M-V-C)

➢ Pipes and filters

➢ Blackboard

Case: Typical Layering Approach

- **Distinct application subsystems that make up an application — contains the value adding software developed by the organization.**

- **Business specific — contains a number of reusable subsystems specific to the type of business.**

- **Middleware — offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.**

- **System software — contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers, and so on.**
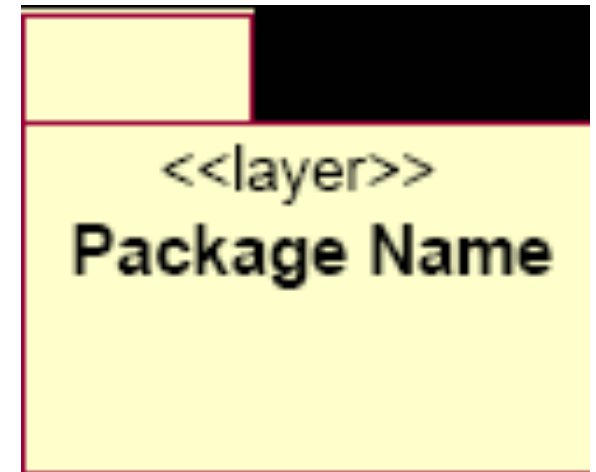
Specific functionality

Application Subsystems

Business-Specific

Middleware

System Software

General functionality

Architectural Pattern: Layers

Layer 5 — Equipment and customer-specific code

Layer 4 — Processes and other application code

Layer 3 — Major abstractions, classes, etc.

Layer 2 — Mechanisms, services

Layer 1 — H/W specific code, O/S specific code, general-purpose code (for example, ORB, MQS)

Application

Application Framework

Infrastructure

Layering Considerations

- Level of abstraction（抽象级别）（每层的抽象度一致。从逻辑上，级别一样）

  - Group elements at the same level of abstraction

- Separation of concerns（关心分离）（每一层有每一层自己的关注点，例如，V关注与外部世界的交互，C关注服务）

  - Group like things together

  - Separate disparate things

  - Application vs. domain model elements

- Resiliency（弹性）（鲁棒性，适用范围比较广）

  - Loose coupling

  - Concentrate on encapsulating change

  - User interface, business rules, and retained data tend to have a high potential for change

Modeling Architectural Layers
(层次架构的建模）

- **Architectural layers can be modeled using**

stereotyped packages.

- **<<layer>> stereotype**

<<layer>>
Package Name

View(GUI)

注册课程界面（学生）

receive or
send infor

注册课程用例

Controller(业务控制层)

注册课程操作

business
processing

Model(数据层)

srudent

courseOffering

some entity
classes
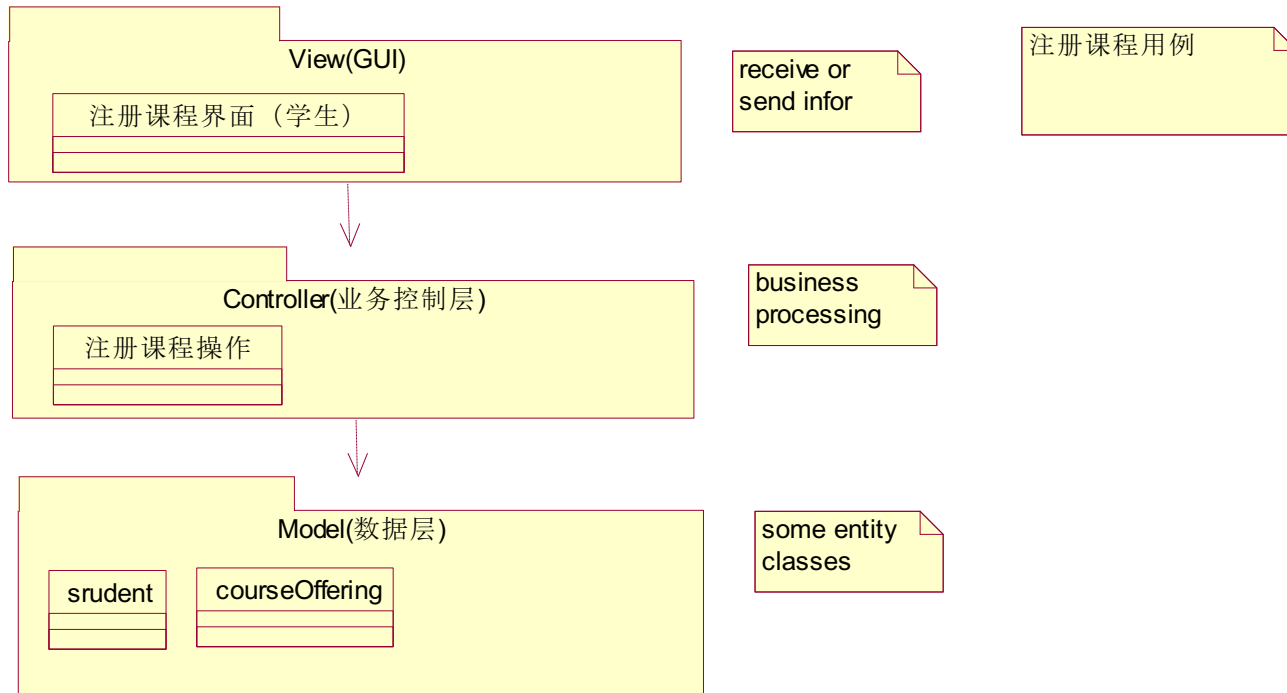
Example: High-Level Organization of the Model
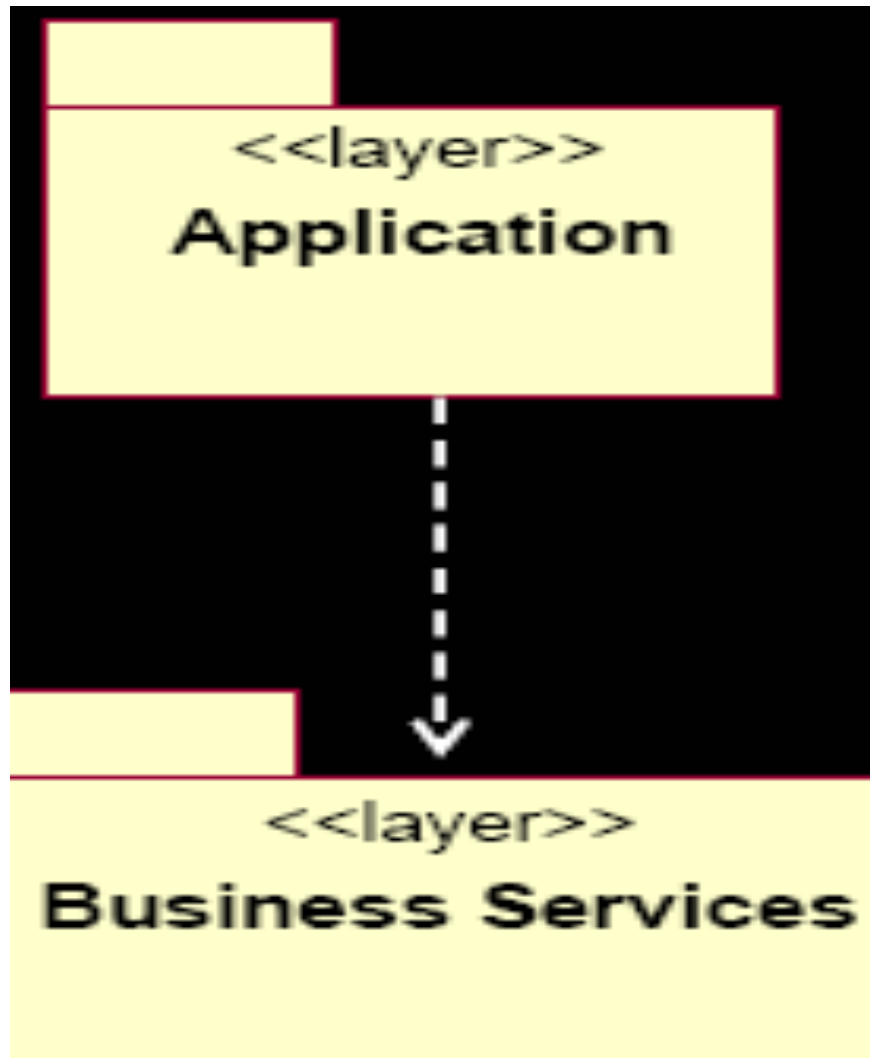
设计的四大内容

- 概要设计：**架构设计**

- 详细设计

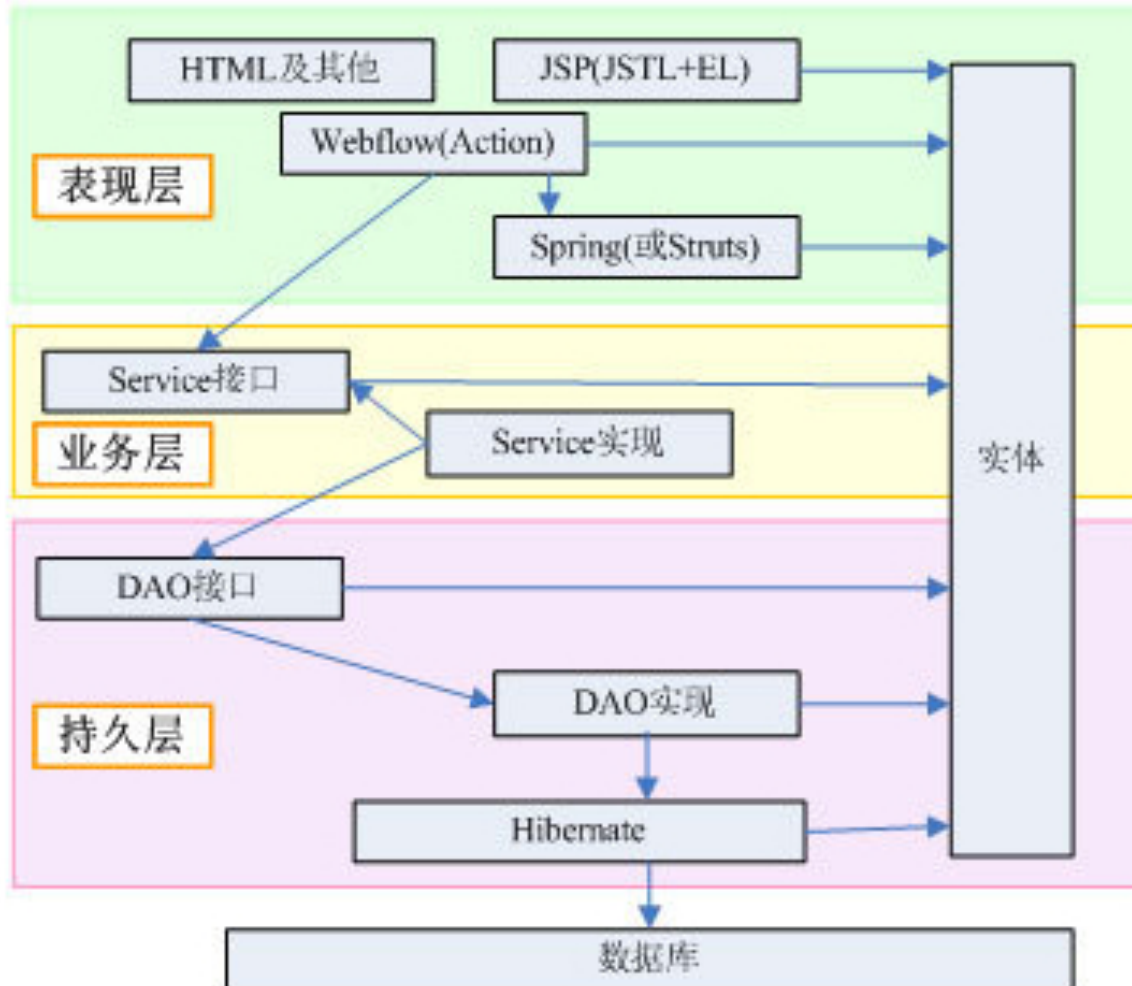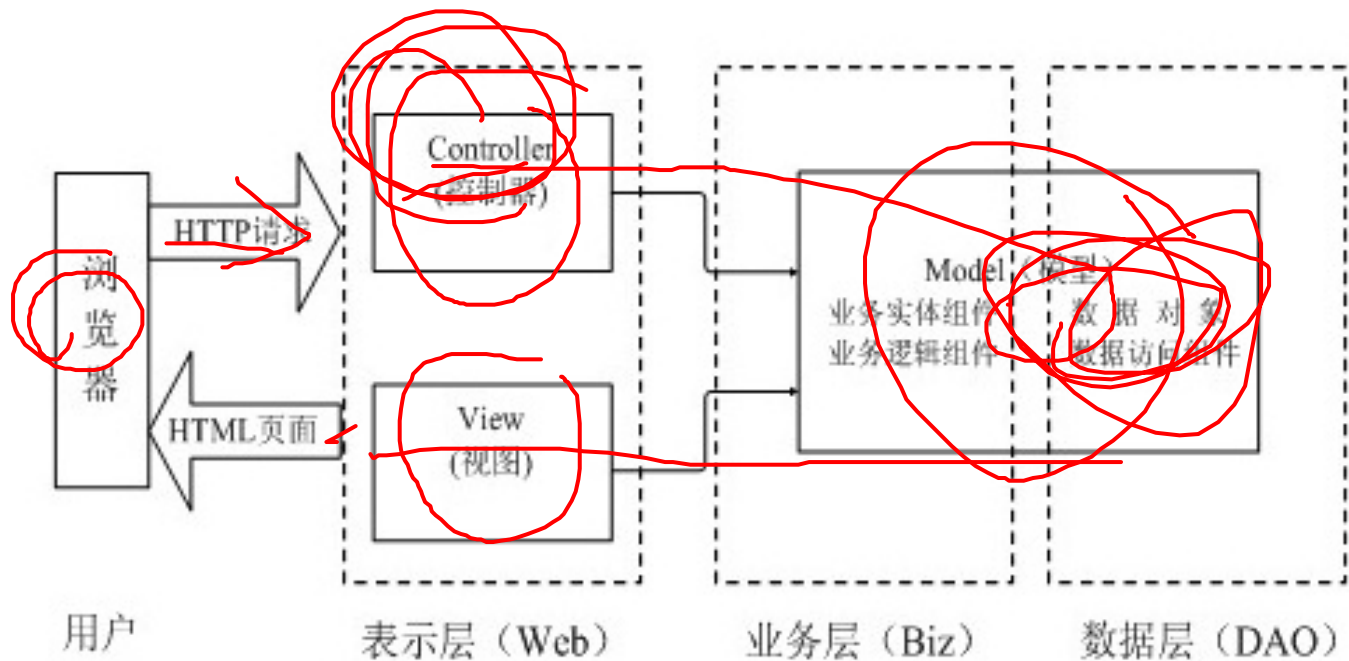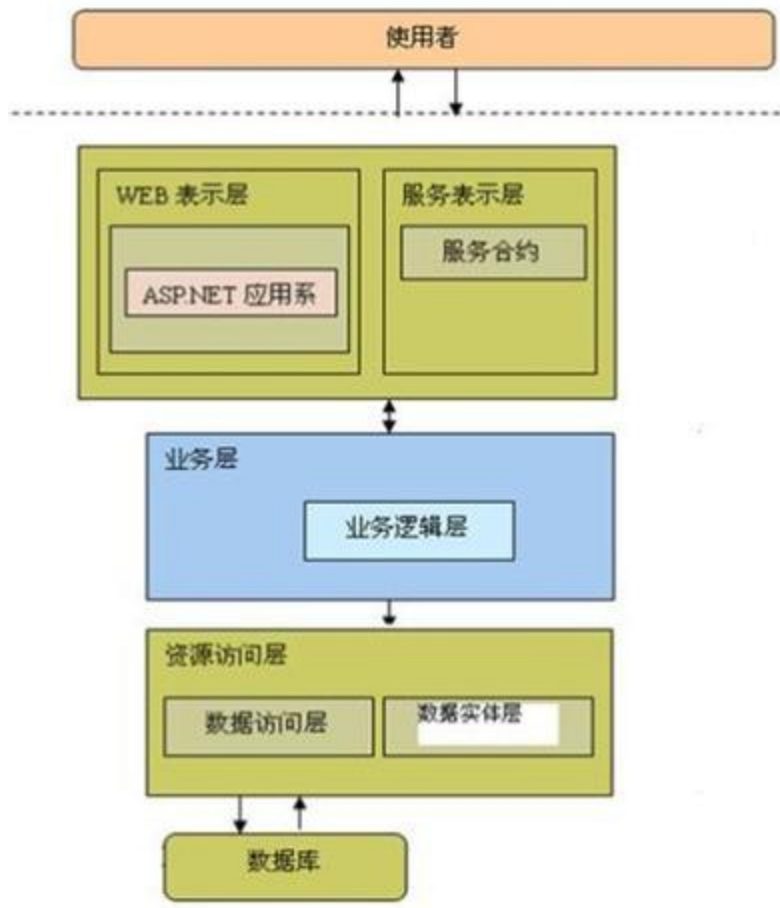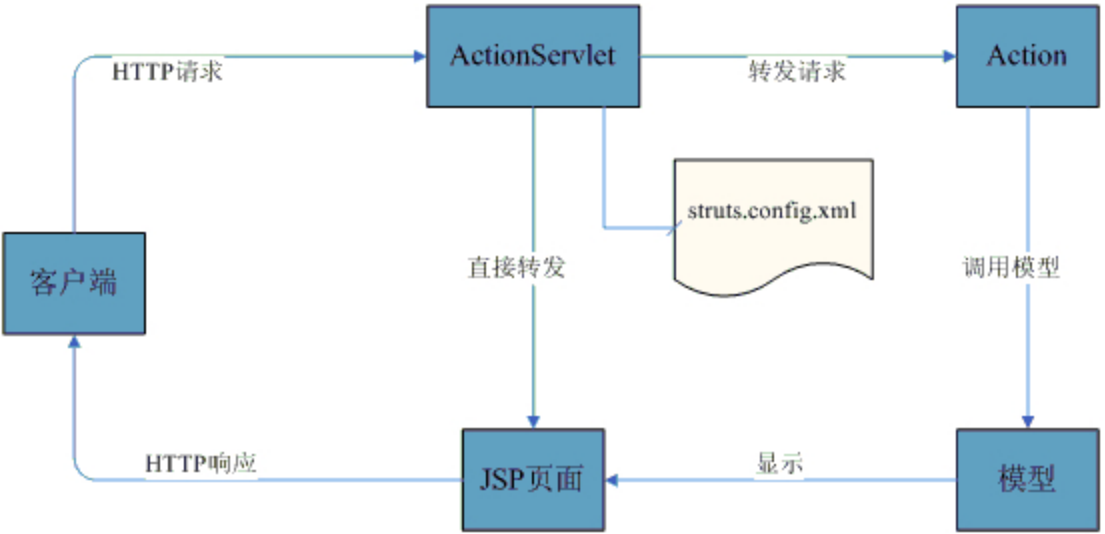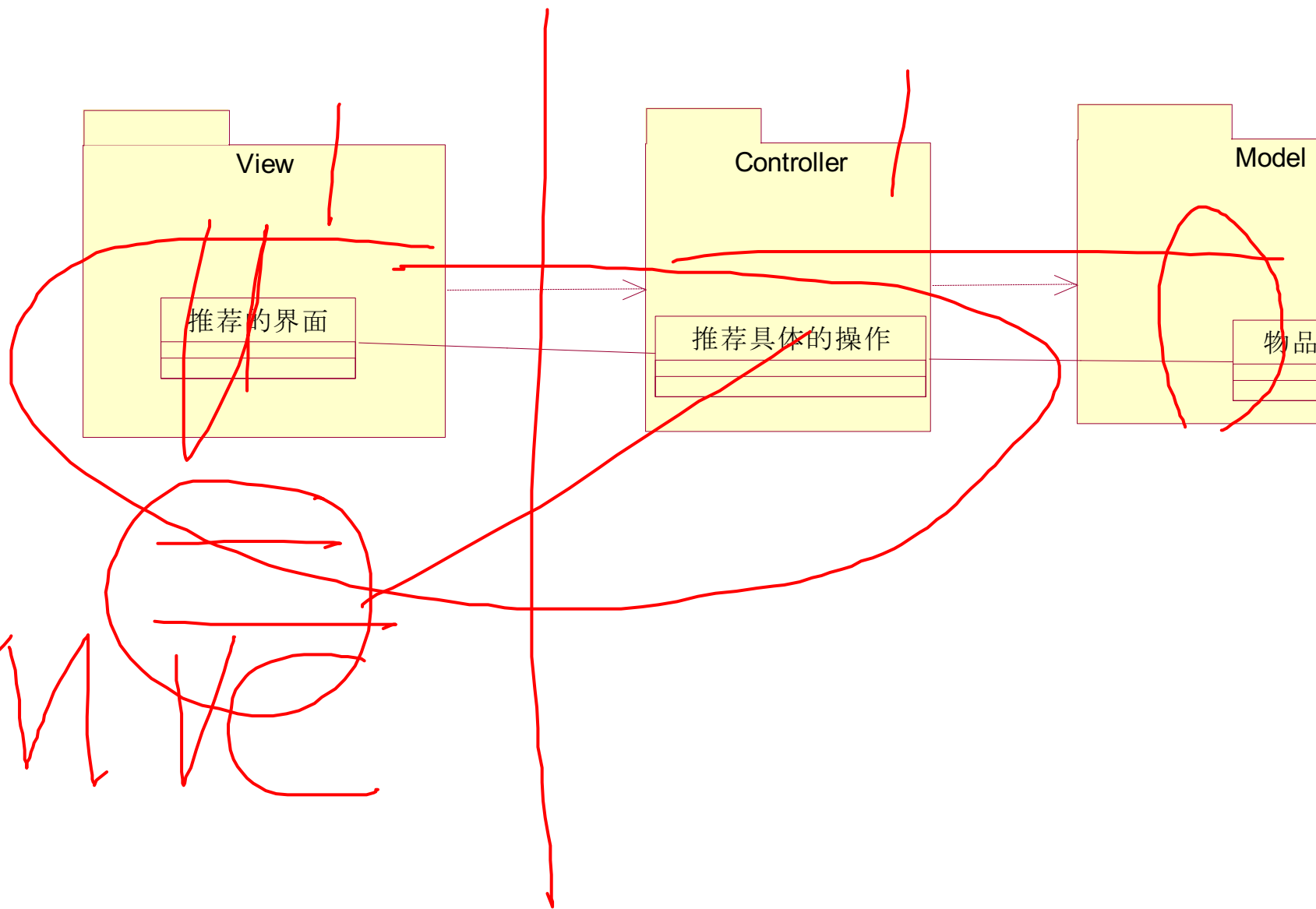 - **数据设计**

  - 子系统及其接口设计

  - 部件设计

MVC
Model-View-Controller

MVC
Model-View-Controller

login
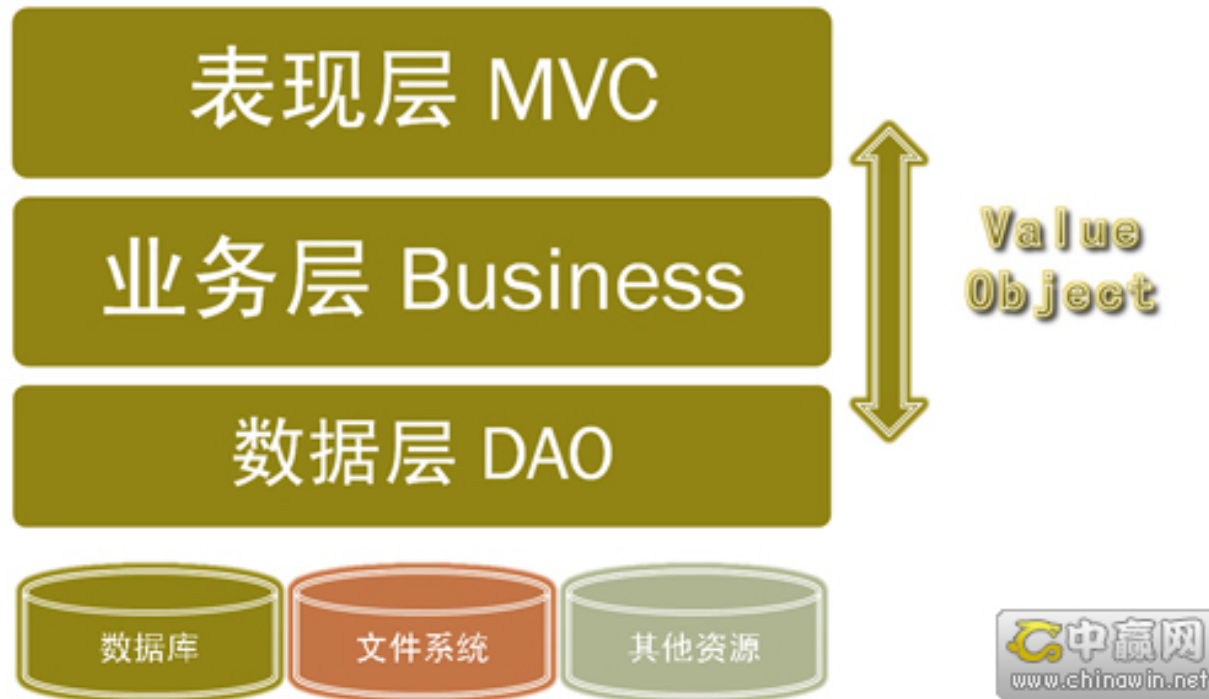
Actor
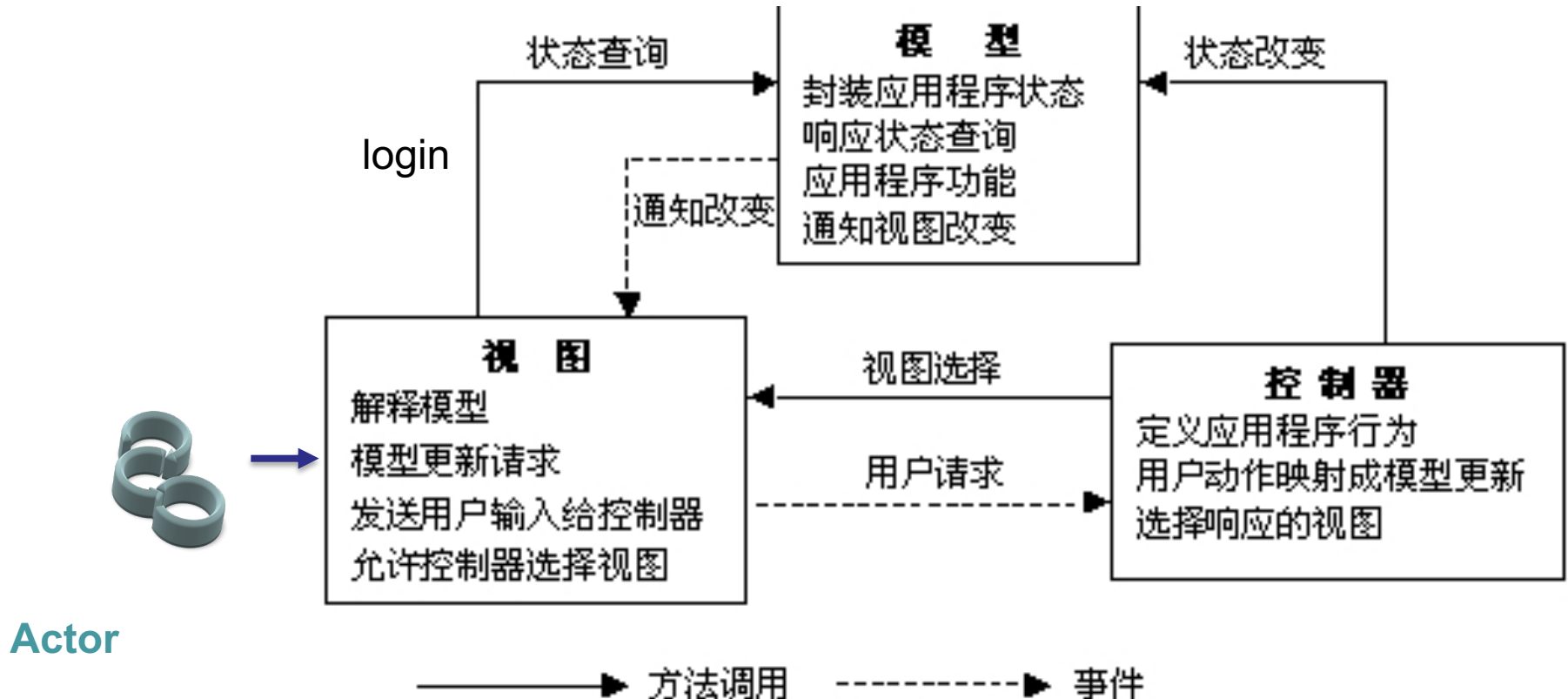
图1 MVC组件类型的关系和功能

MVC
Model-View-Controller
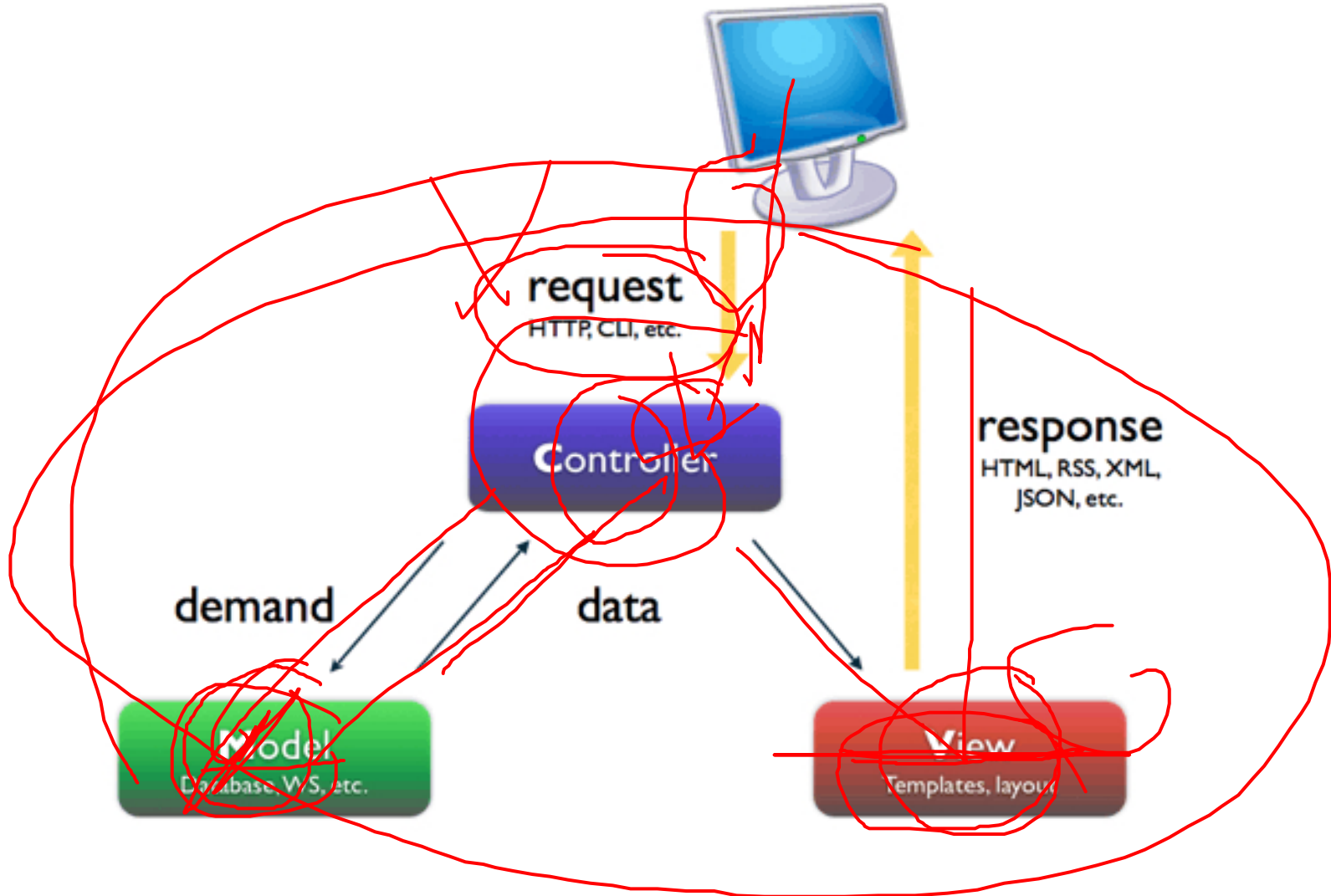
MVC
Model-View-Controller

MVC
Model-View-Controller

用户　　　　　　表示层（Web）　　　　业务层（Biz）　　　数据层（DAO）

Controller
(控制器)

View
(视图)

Model (模型)
业务实体组件　　数据对象
业务逻辑组件　　数据访问组件

HTTP请求

HTML页面

浏览器

左图流程：

客户端 →(HTTP请求)→ ActionServlet →(转发请求)→ Action

ActionServlet → struts.config.xml

ActionServlet →(直接转发)→ JSP页面

Action →(调用模型)→ 模型

模型 →(显示)→ JSP页面

JSP页面 →(HTTP响应)→ 客户端

右图分层：

使用者

WEB 表示层
　　ASP.NET 应用系

服务表示层
　　服务合约

业务层
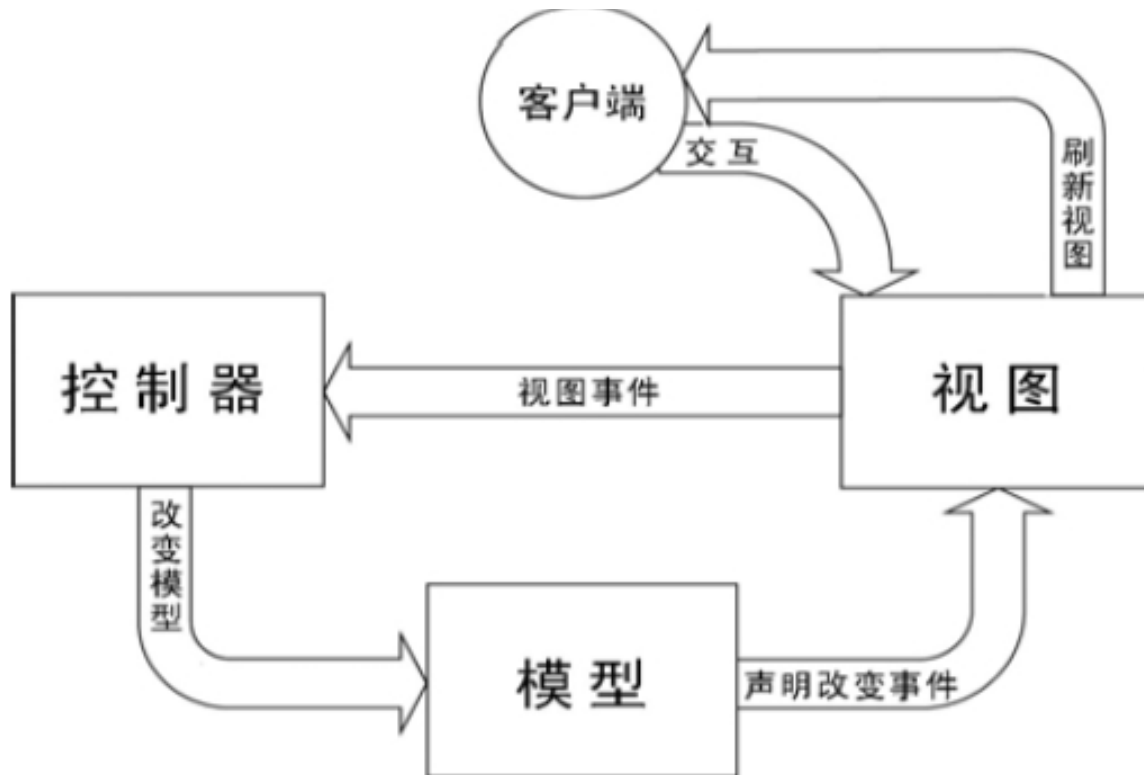　　业务逻辑层

资源访问层
　　数据访问层　　数据实体层
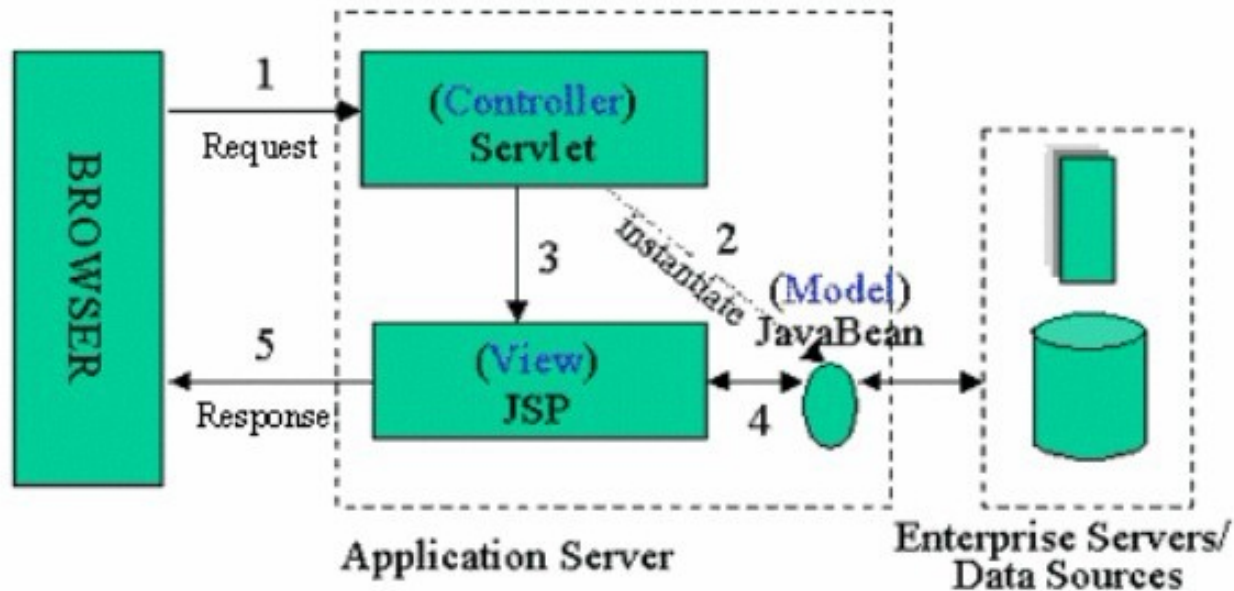
数据库

View

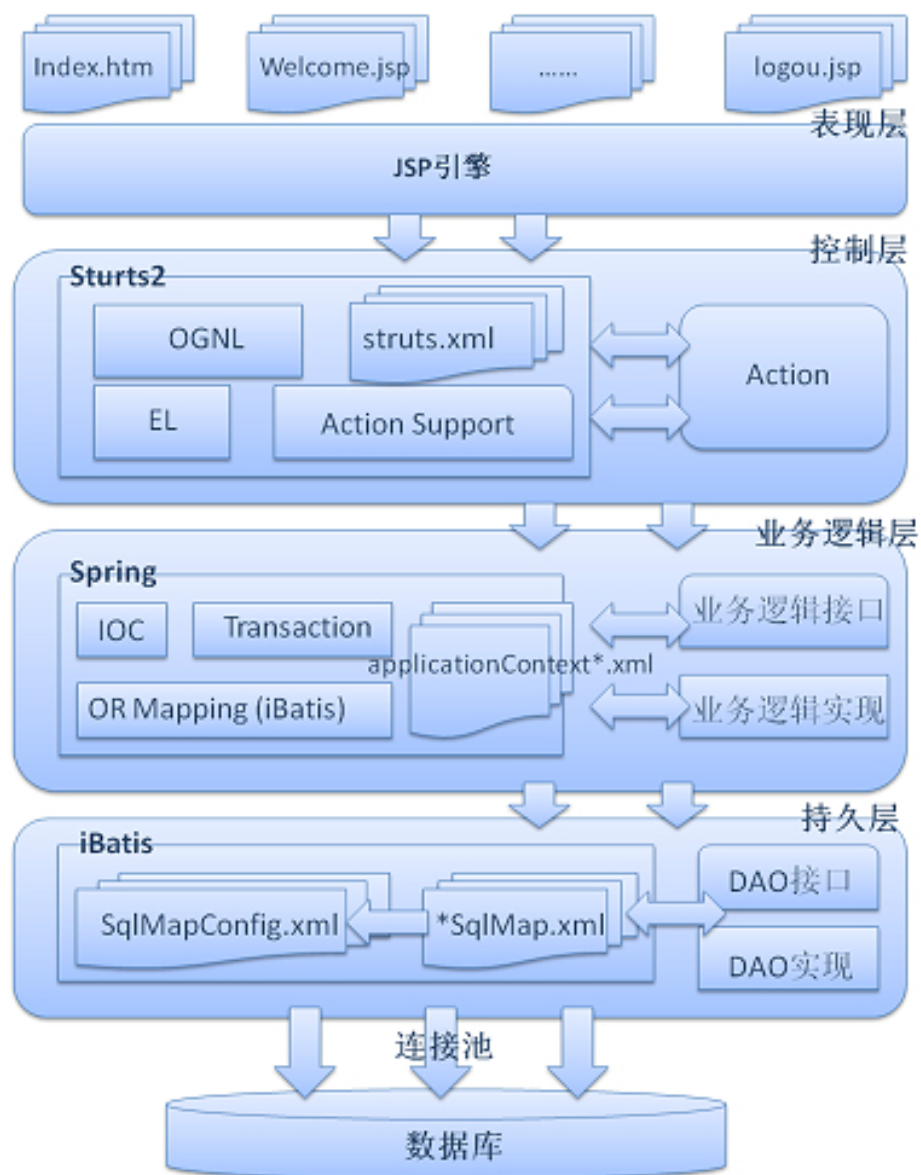Controller

Model

推荐的界面

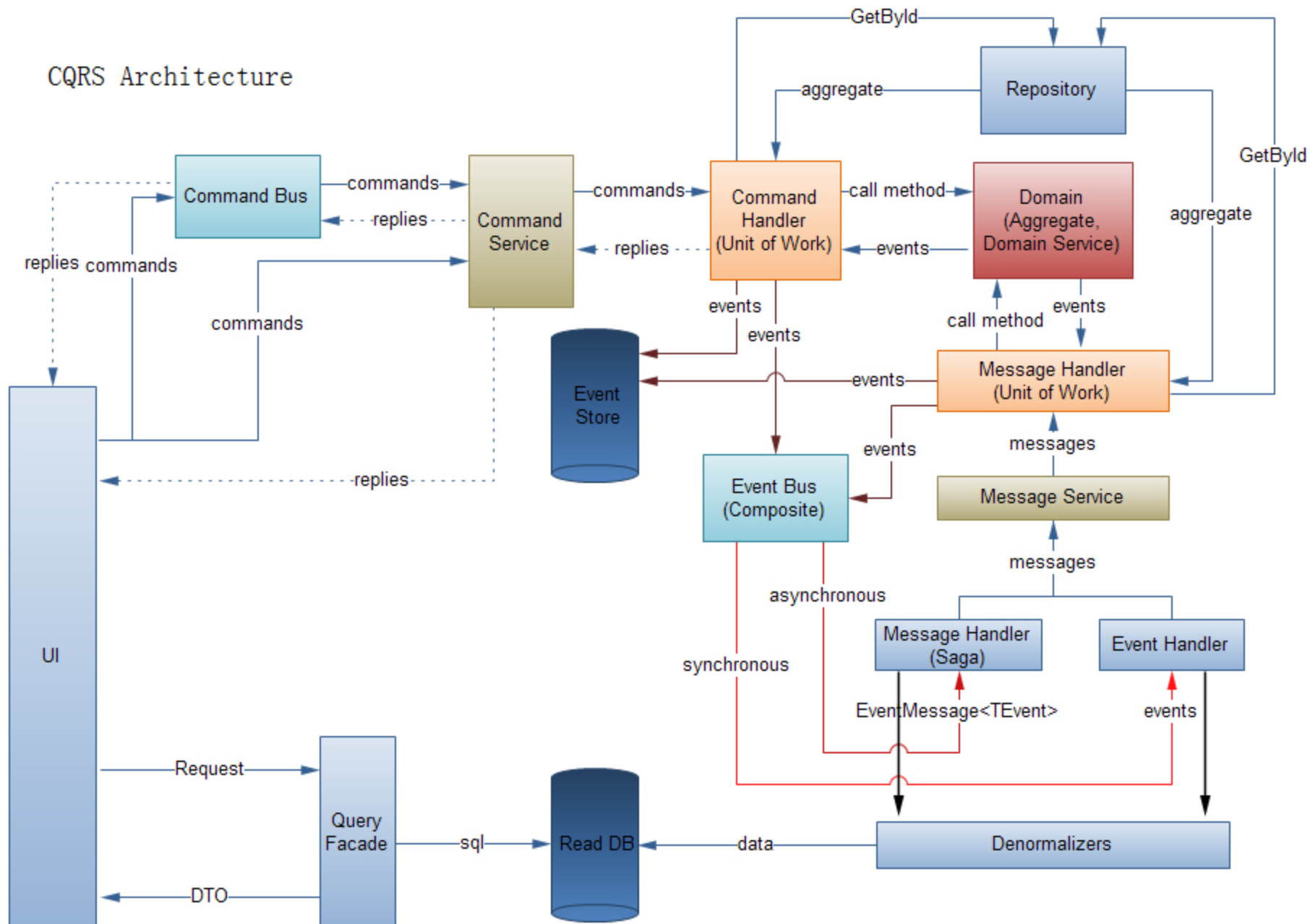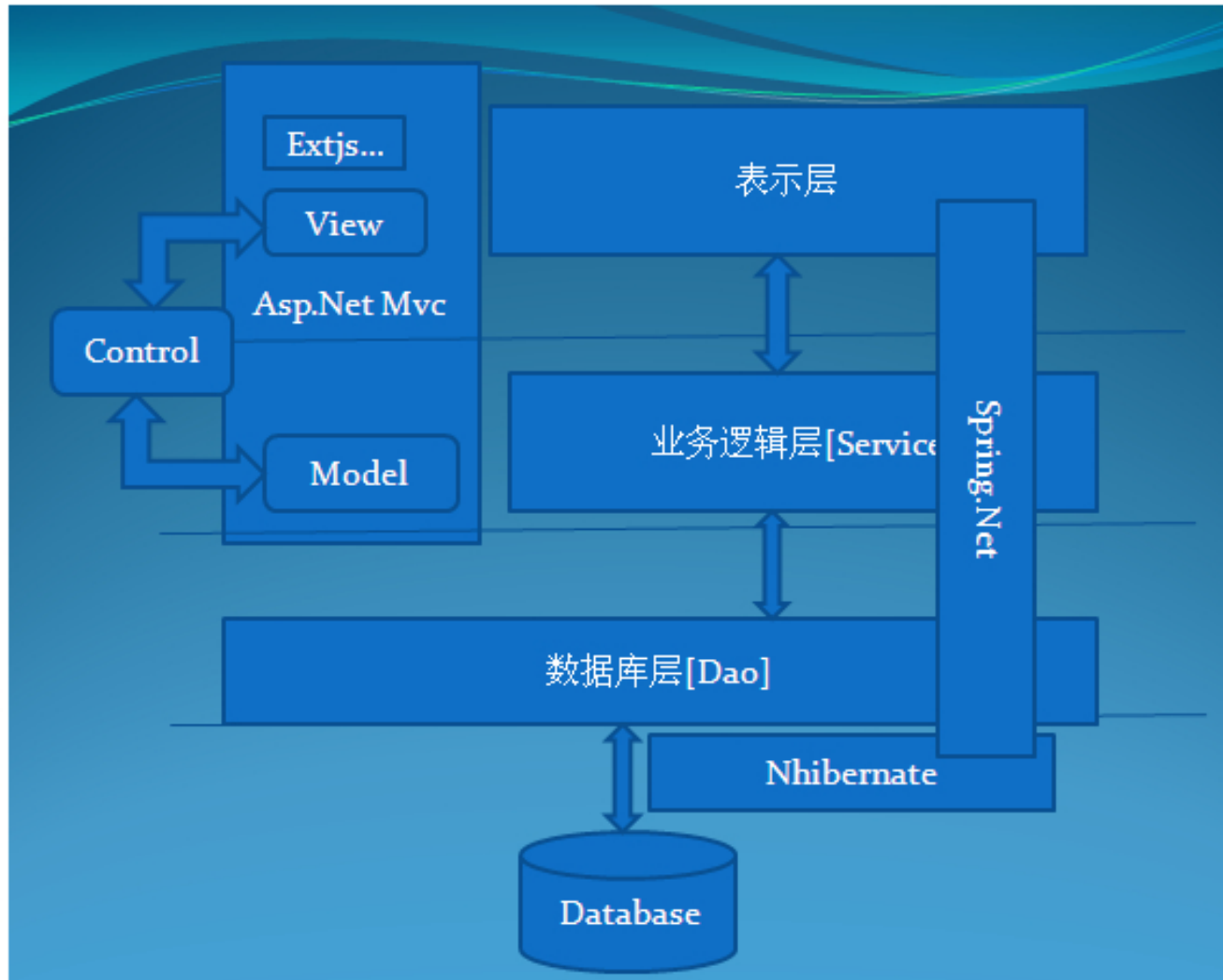推荐具体的操作

物品

MVC
Model-View-Controller
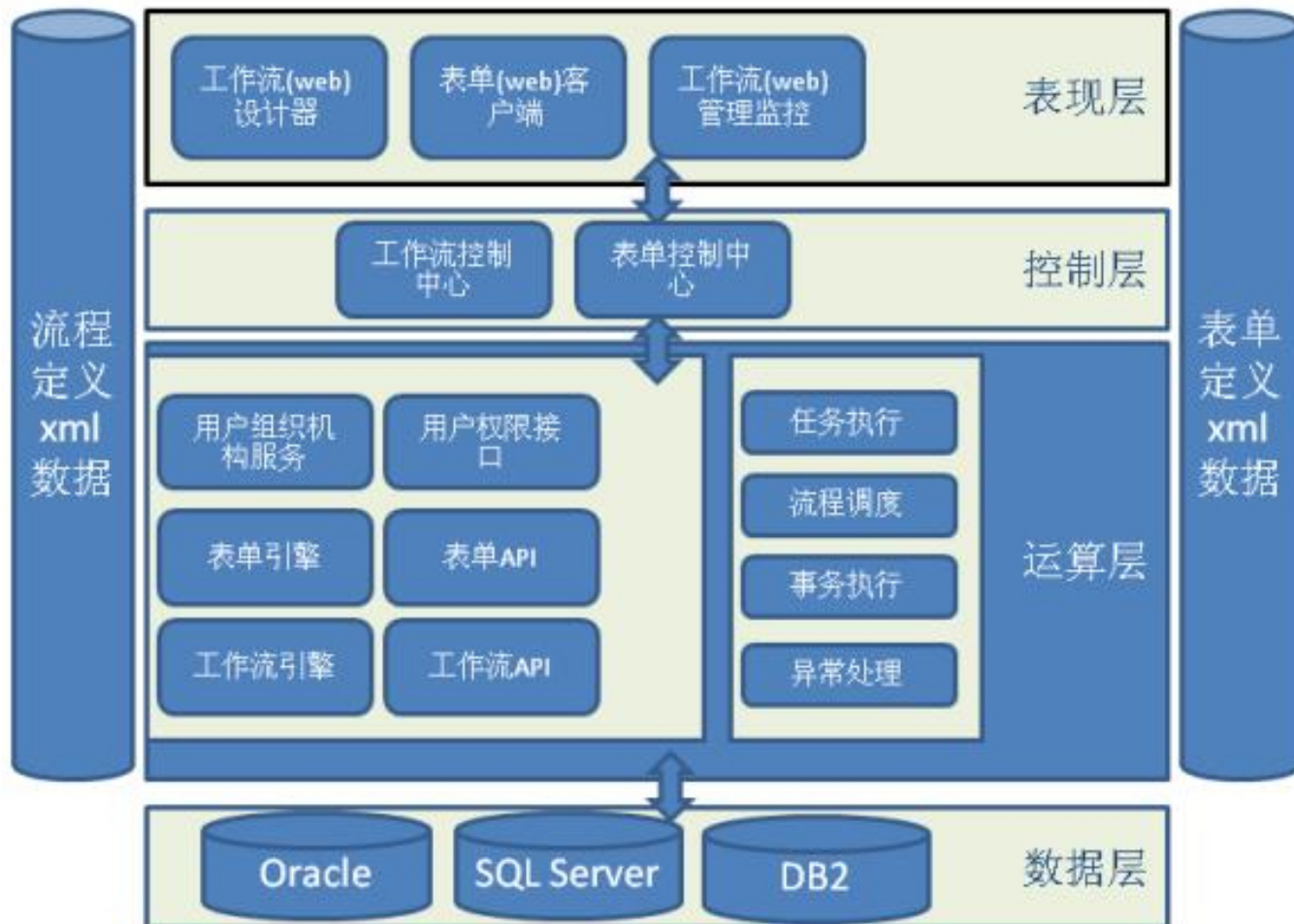
MVC
Model-View-Controller

CQRS Architecture

MVC
Model-View-Controller

MVC
Model-View-Controller

MVC
Model-View-Controller

软件三层架构(MVC)

Web 层	业务处理层 service	数据访问层

Service

dao

```
IE
```

```
Servlet
```

```
Service
```

```
Dao
Jdbc
…
Hibernate
```

db

```
Jsp

Jstl+el
```

```
Javabean
```

Com.lich.domain
Com.lich.dao
Com.lich.dao.impl
Com.lich.service
Com.lich.service.impl
Com.lich.web.controller
Com.lich.web.filter
Com.lich.web.listener
Com.lich.utils
Web-inf/jsp 目录保存 jsp 文件

Architectural Design Steps

1. **Key Concepts**
2. **Define the High-Level Organization of Subsystems**
3. **Identify Key Abstractions**
4. **Checkpoints**

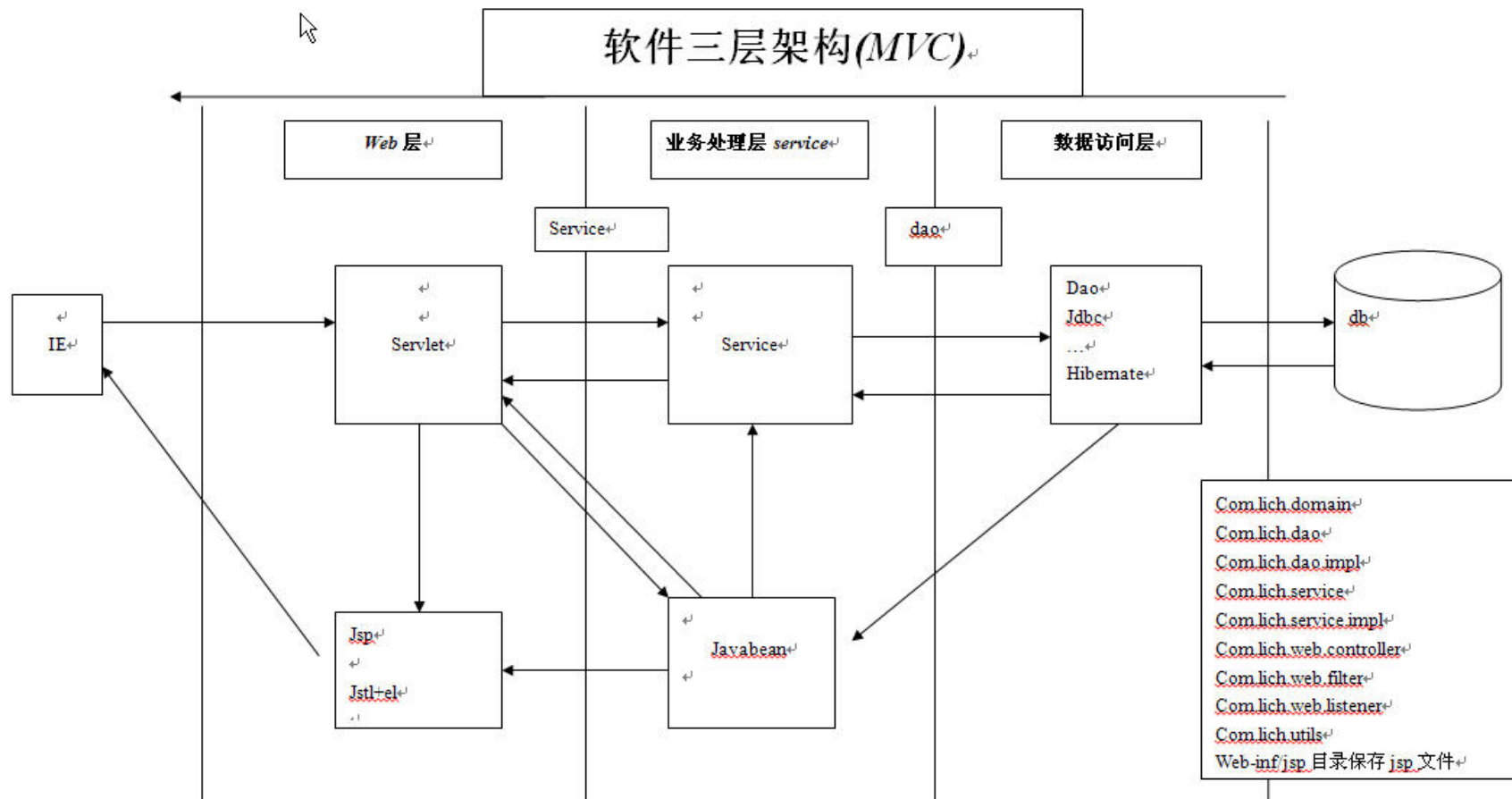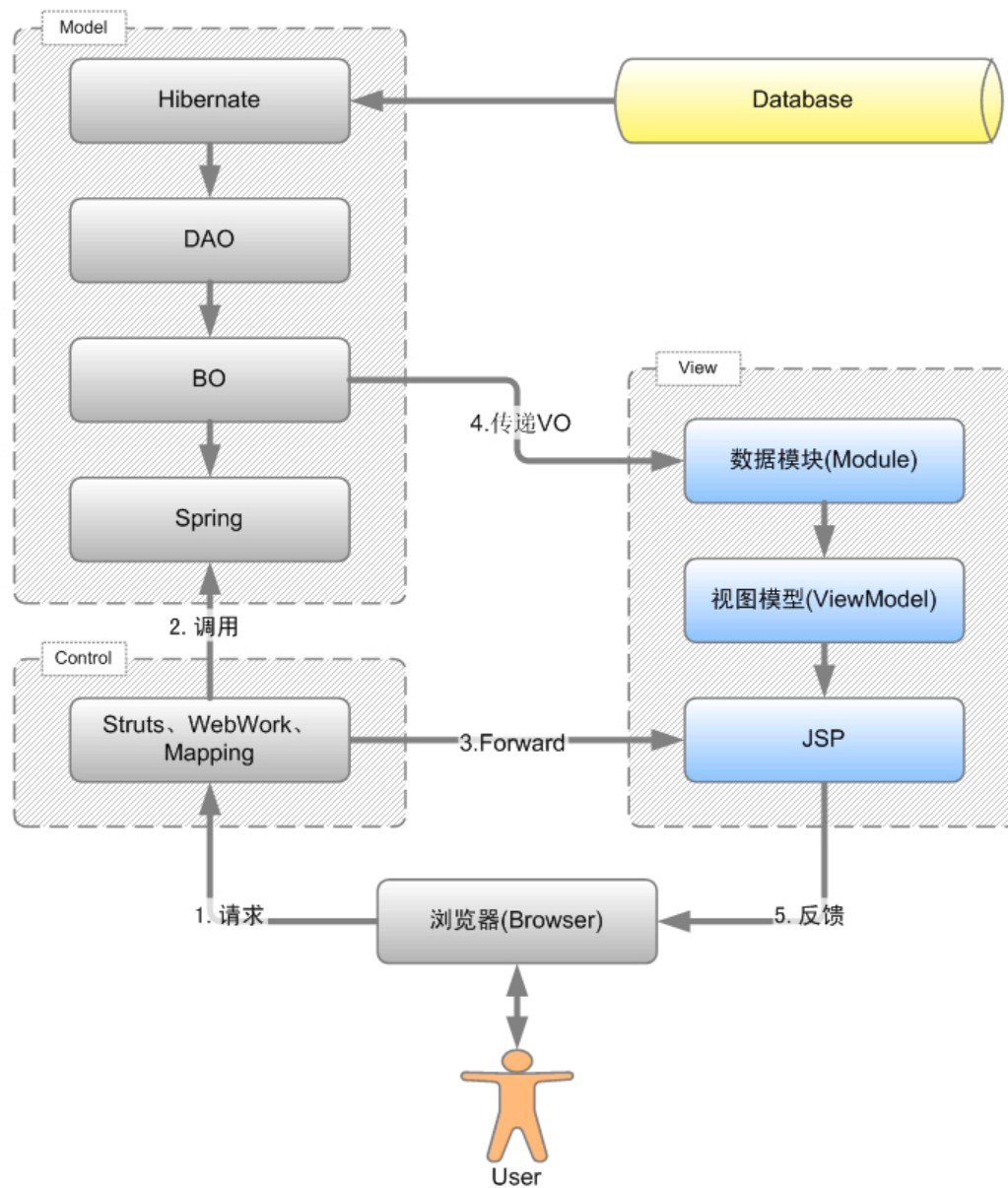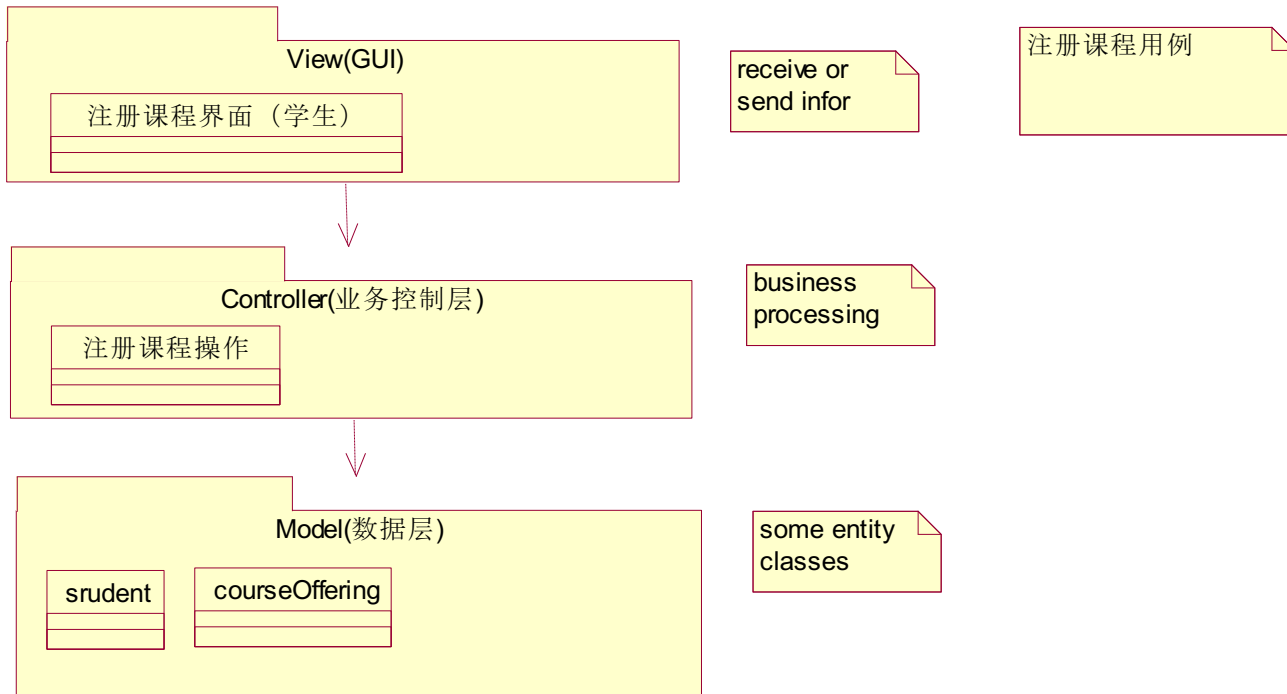# What Are Key Abstractions?

何为关键抽象

- A key abstraction is a concept, normally uncovered in Requirements, that the system must be able to handle

- Sources for key abstractions

    - Domain knowledge

    - Requirements

    - Glossary

    - Domain Model, or the Business Model (if one exists)

是否记得，在大家写需求分析的术语表的时候，老师要求你把系统中所有需要保存的信息都纳入术语表？
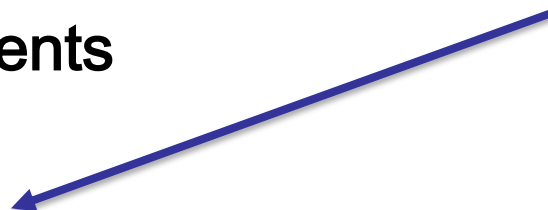
Defining Key Abstractions *Steps*

- Define analysis class relationships

- Model analysis classes and relationships on class diagrams

  – Include brief description of analysis class

- Map analysis classes to necessary analysis mechanisms

Example: Key Abstractions
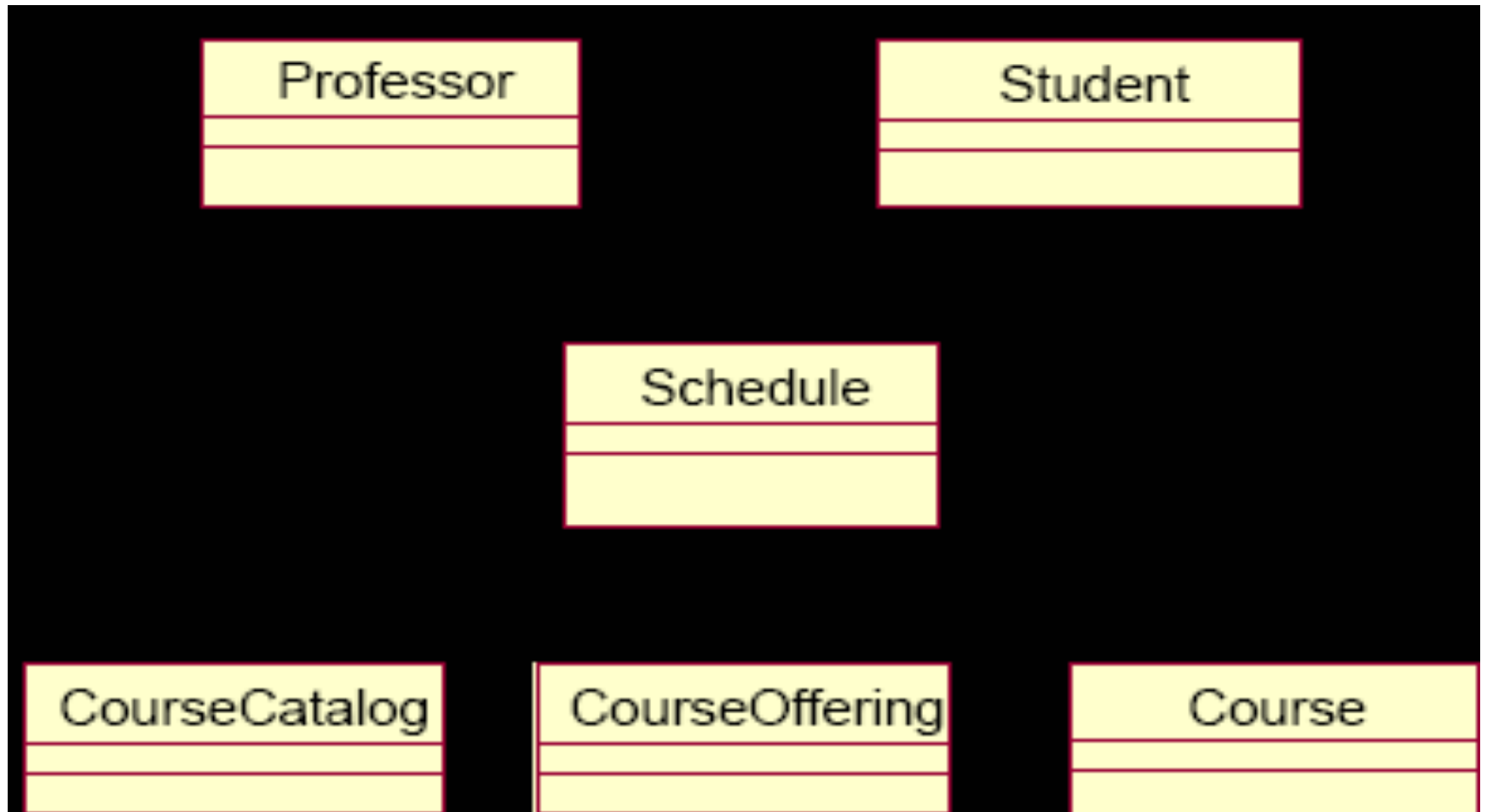
下面来自一组学生的项目

2.1 架构描述

2. 架构设计

2.1. 架构描述

本系统基于thinkphp框架，采用MVC（Model View Controller）三层架构。该架构的使用实现了应用程序的分层管理，简化了后续对程序的修改和扩展，并且使程序某一部分的重复利用成为可能。

View层又称视图层，是用户看到并与之交互的页面。在本系统中，视图层由众多HTML文件组成，它们负责获取用户的输入及显示控制层处理的结果。

Controller层又称控制层，它收到来自视图层的请求并对实体类进行增删改查。当单击Web页面中的超链接和发送HTML表单时，控制层接收请求并处理请求，然后确定用哪个视图来显示处理返回的数据。

Model层又称数据层，主要作用是存储和访问数据。

2.2 架构图

● **根据2.1的描述，图文结合，画出架构图**

● **（补充每一种架构的图示）**

2.2.架构图
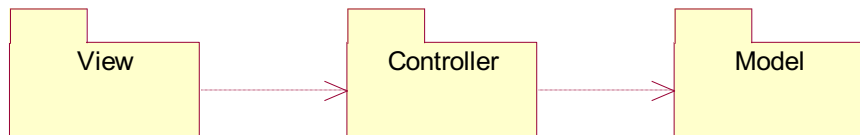
图2-1 志愿者管理系统架构图

● Tips
● 此处架构图中使用的是空包，不需要写出类。View, Controller, Model的顺序不能随便放，同时要注意箭头方向

2.3 关键抽象

● 关键抽象就是找到系统实体类的过程

● 根据第一章的问题陈述以及术语表, 可初步找出系

统的实体类

● 实体类是存储和管理的信息（数据）

2.3 关键抽象

2.3. 关键抽象

关键抽象即为找到系统实体类的过程。

实体类为系统中存储和改动的数据，可以从需求分析中的术语表中得到。经过分析，本系统有三个实体类，分别为用户表，活动表和报名表，用户表存储了与用户相关的一切信息，包含用户名、密码、入学年份等，活动表包含了活动名称、活动时间、活动人数等信息，报名表包含了活动名称、报名情况等信息。

ActivityModel

UserModel

ApplicationModel
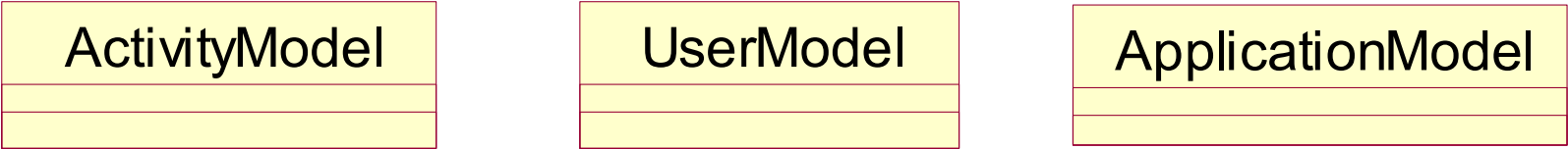
图2-2 志愿者管理系统实体类图

基于面向对象的软件架构设计的步骤和制品

1. **架构描述**

   – 用一段文字，描述你的项目将采用哪种架构，例如MVC，SringMVC,….也可多做些解释。

2. **架构图**

   – 基于上面1）画出你系统的架构图，就是用包来描述，包里面是空的。

3. **关键抽象**

   – 找出系统的数据类（entity classes)，并且用类图描述出来

Architectural Design Steps

1. **Key Concepts**
2. **Define the High-Level Organization of Subsystems**
3. **Identify Key Abstractions**
4. **Checkpoints**

Architectural Design Steps

- Key Concepts
- Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- Checkpoints

Checkpoints

- General

  – Is the package partitioning and layering done in a logically consistent way?

  – Have the necessary analysis mechanisms been identified?

- Packages

  – Have we provided a comprehensive picture of the services of the packages in upper-level layers?

Checkpoints (cont.)

- Classes

  - Have the key entity classes and their relationships been identified and accurately modeled?

  - Does the name of each class clearly reflect the role it plays?

  - Are the key abstractions/classes and their relationships consistent with the Business Model, Domain Model, Requirements, Glossary, etc.?

作业-7th

1. What is the purpose of Architectural Design?

2. What is used in architecture diagram?

3. What key abstractions are identified during Architectural Design? Why are they identified here?

4. What is Design Pattern, and Framework?

5. How many Design Pattern can you describe?

6. 请描述架构设计的步骤及其制品

7. 请描述，采用基于结构化与基于面向对象的软件架构设计的相同点及不同点(建议对比步骤和制品）(选做）

作业（20201007）

◆ **每位同学自己写的作业（学号最后一位是5的同学把作业发给TA）**

1. What is the purpose of Architectural Design?

2. What key abstractions are identified during Architectural Design?

3. What is Design Pattern, and Framework?

4. How many Design Pattern can you describe?

5. 请描述架构设计的步骤及其制品

6. 请描述，采用基于结构化与基于面向对象的软件架构设计的相同点及不同点(建议对比步骤和制品）(选做）

◆ **小组项目作业（10月14日之前各小组长邮件给TA）**

完成各组的架构设计，把项目规格说明书向前推进

Lab: Architectural Design

- **Given the following:**

  - **Some results from the Requirements discipline:**

- **Problem statement**

  - **Use-Case Model main diagram**

  - **Glossary**

  - **Some architectural decisions:**

  - **(textually) The upper-level architectural layers and their dependencies**

Lab: Architectural Design (cont.)

- **Identify the following:**

- **The key abstractions**

Lab: Architectural Design (cont.)

- **Produce the following:**

  - **Architecture Doc**

  - **Architecture Diagram （Class diagram containing the upper-level architectural layers and their dependencies（the architecture design of your project)**

  - **Class diagram containing the key abstractions**

  - **Other DOC if necessary, describing the above artifacts**
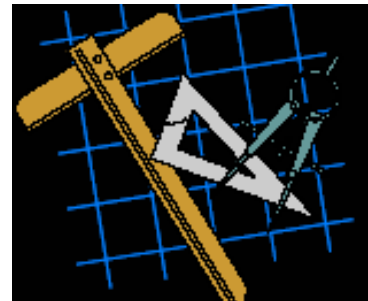
Lab: Review

- **Compare your key abstractions with the rest of the class**

  - **Have the key concepts been identified?**

  - **Does the name of each class reflect the role it plays?**

- **Compare your class diagram showing the upper-level layers**

  - **Do the package relationships support the System architecture?**
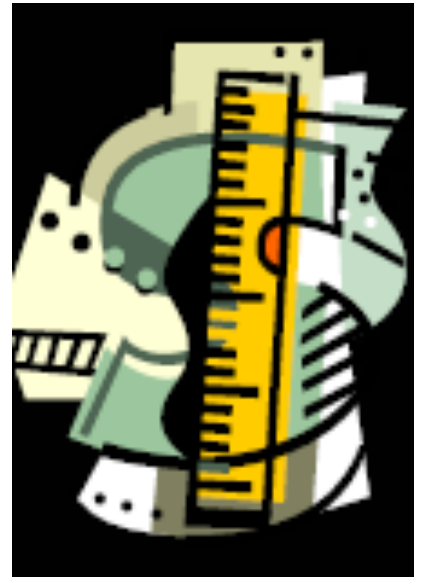
- **Architectural Design focuses on defining a candidate architecture and constraining the architectural techniques to be used in the system.**

- **It relies on gathering experience gained in similar systems or problem domains to constrain and focus the architecture so that effort is not wasted in architectural rediscovery.**

- **In systems where there is already a well-defined architecture, Architectural Design might be omitted; Architectural Design is primarily beneficial when developing new and unprecedented systems.**

Activity diagram

- **The workflow of a use case describes what needs to be done by the system to provide the value that the served actor is looking for.**

- **It consists of a sequence of activities that, together, produce something for the actor.**

- **The workflow often consists of a basic flow and one or several alternative flows.**

- **The structure of the workflow can be described graphically with the help of an activity diagram.**

Patterns

- offer a way to capture, reuse, and share solutions to common problems and a common language for describing problems and solutions.

- implemented in a development tool like Rational Software Architect help automate access to, and the application of, patterns.

- Design patterns and transformations in Rational Software Architect help automate routine modeling and coding tasks.

Continue……

- Grady Booch's definition of a pattern is still one of the best: "a solution to a recurring problem in a given context."

- Patterns provide a standard way of capturing and naming solutions, programming idioms, and best practices.

- As more developers have researched and understood patterns, patterns have become a standard way for practitioners to communicate and share what they know with each other.

Continue……

- For an architect leading a large team of developers, with tools to share and apply patterns automatically, patterns can become a way to enforce coding standards and ensure consistency in the way a design for a system is implemented.

- For the developer, a set of carefully selected patterns, customized for a specific project or application, can reduce mundane coding tasks and head off confusion about how to approach the implementation of design elements in the system.

- Most encounters with patterns occur through pattern specifications, descriptions of patterns in documentation, books, articles, and Web sites. An important function for patterns is to enhance communication and education within the community of designers and developers. The efficiency of using patterns is greatly enhanced by documenting them in a standard way. Usually pattern specifications include at least these three main topics:
  - Context: When would you apply this pattern?
  - Problem: A precise statement of the problem solved by the pattern
  - Solution: The description of the solution provided by this pattern
- This list is often extended with other topics such as consequences of applying the pattern, examples, keywords, and classification.
- Pattern specifications are critical, but they provide only part of the promise of patterns.

- Patterns allow you to create artifacts that can be more easily reused and customized for each use.
- The power of patterns is not realized by simply copying and reusing artifacts as-is (unlike cutting and pasting lines of code, or deploying an existing component into a new application). Such procedures are common and do constitute reuse, but patterns take reuse to a higher level by allowing points of variability.
- The key to pattern automation is the use of pattern implementation tools that expose the variability choices but hide the substitution details.

- Design Patterns
- Design patterns allow you to make use of existing solutions developed in the same type of model that you are working on.
- So, for example, the Observer GoF pattern contains design-level UML classes that can be applied in a design-level UML class model.
- Patterns have parameters so that you can customize them for a specific context, but patterns do not automatically translate themselves to work in different model types.
- You cannot, for example, apply a design pattern and get a code-level idiom in Java code without using transformations.

- Transformations
- Transformations take elements from one model and translate them into elements of a different model.
- For example, you can apply a transformation to move from a platform-independent model to a platform-specific model as you add in details about the platform and get closer to the implementation.
- When adding levels of refinement, you can transform from a platform-specific model to another, adding more details without changing the model type.
- Transformations are often applied to whole models, but they can be applied to selections from models as well.