# 课程章节安排

- 第一部分 软件工程概论
- 第二部分 软件项目开发过程与管理
- 第三部分 软件需求工程
- 第四部分 软件设计
- 第五部分 软件编码、测试与质量保障
- 第六部分 软件实施、维护与演化

# 第四部分 软件设计

- 4.1 软件工程开发方法与软件设计
- 4.2 体系结构设计
- 4.3 类/数据建模与设计
- 4.4 行为建模与设计
- 4.5 物理建模与设计

# 4.1 软件工程开发方法与软件设计

- 软件工程开发方法
  - 传统开发方法
  - 面向对象方法
- 软件设计
  - 设计的概念
  - 设计的原则

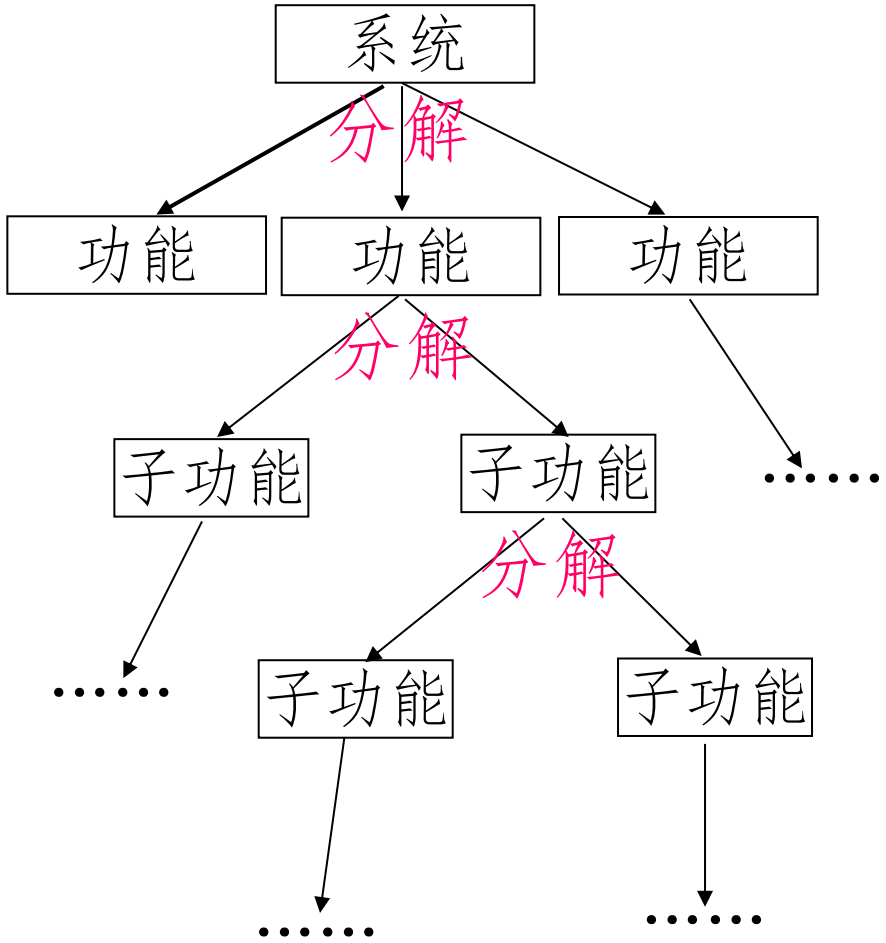
# 软件工程方法

建模过程：
层层进行功能分解

## 功能分解法

以系统需要提供的功能为中心来组织系统

1. 首先定义各种功能，然后把功能分解为子功能
2. 对较大的子功能进一步分解，直到可给出明确的定义
3. 设计功能/子功能所需要的数据结构
4. 定义功能/子功能之间的接口



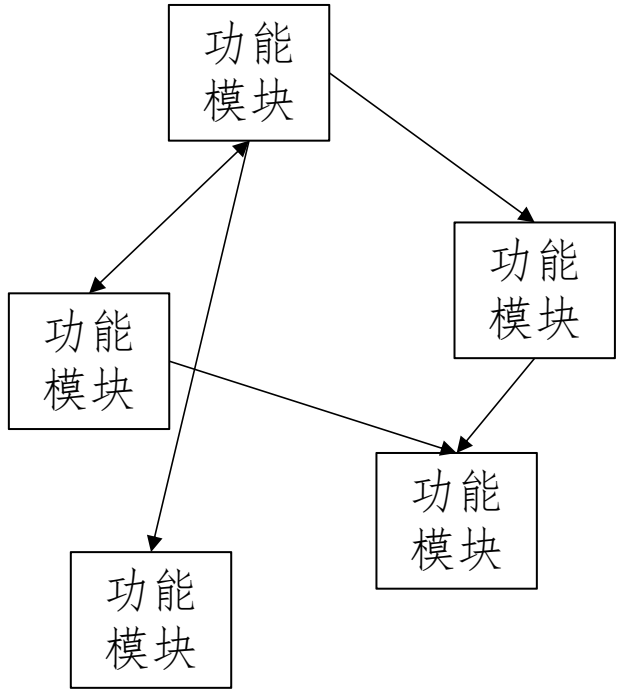

作为一种早期的建模方法，没有明确地区分分析与设计

# 软件工程方法

## 功能分解法的优缺点

- 直接地反映用户的需求所以工作很容易开始
- 不能直接地映射问题域很难检验结果的正确性
- 对需求变化的适应能力很差
- 局部的错误和修改很容易产生全局性的影响

功能分解法得到的系统模型：由模块及其接口构成



# 软件工程方法

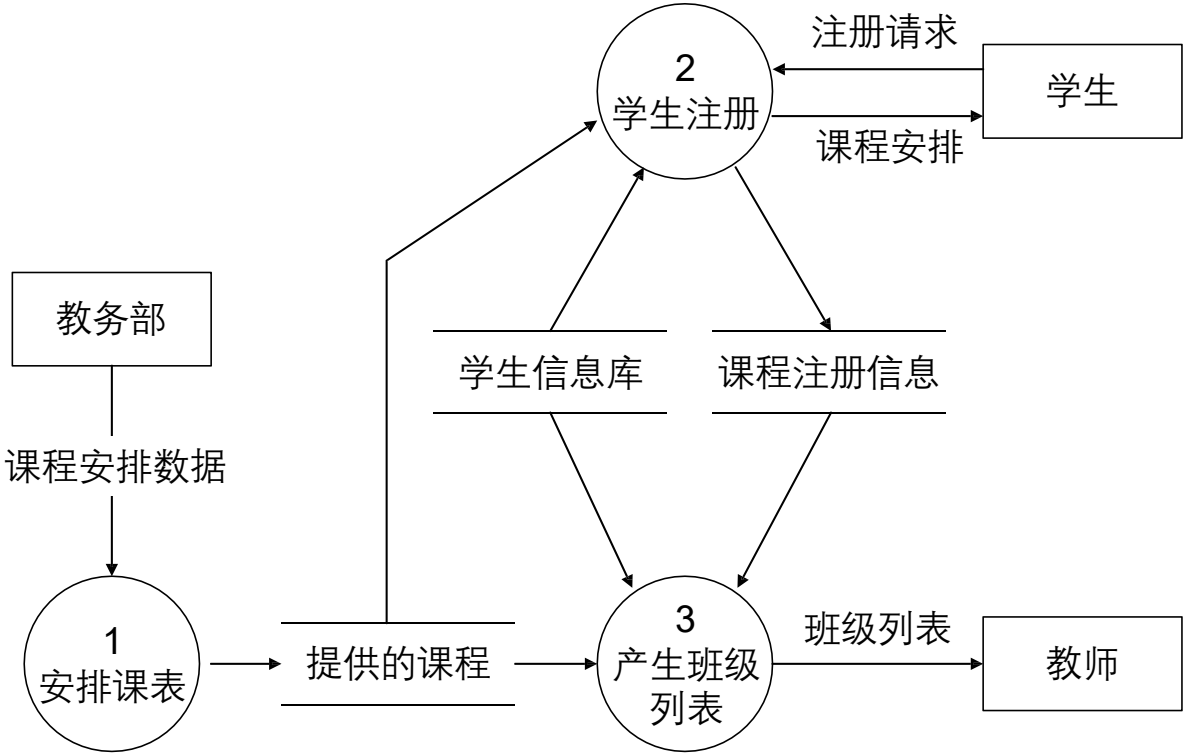
- **结构化方法**

使用结构化编程、结构化分析和结构化设计技术的系统开发方法

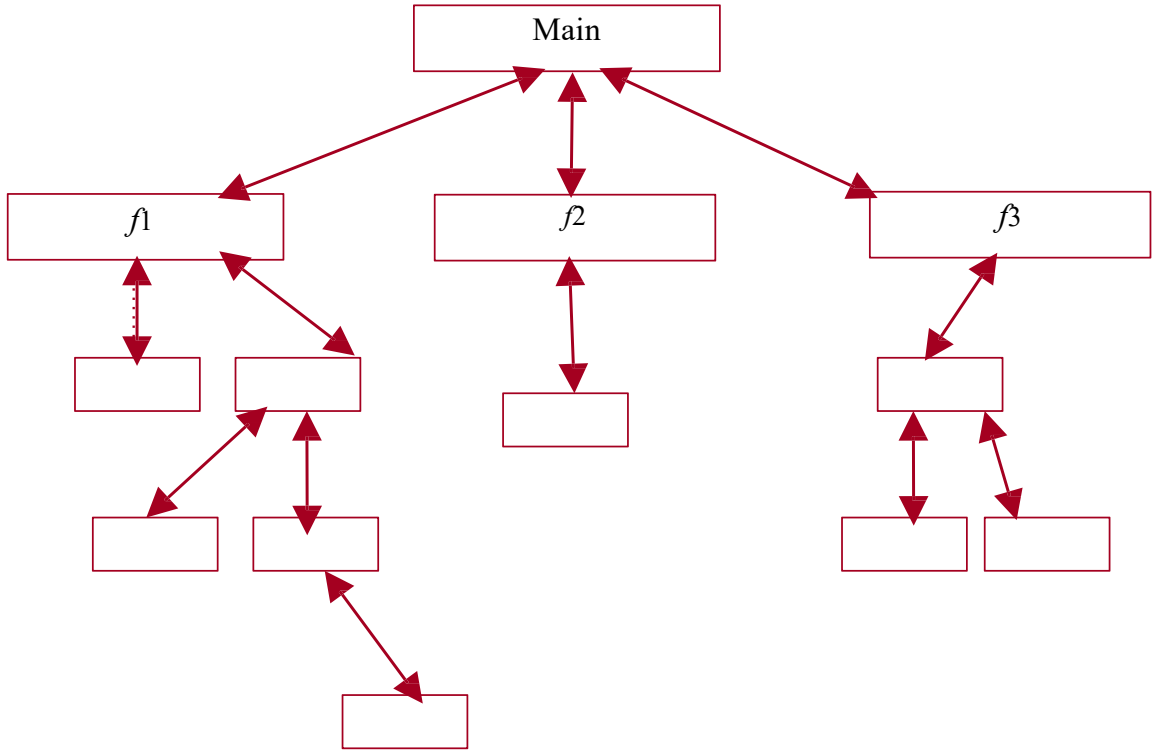
- 结构化分析
- 结构化设计
- 结构化编程

# 结构化分析

- 结构化分析又称数据流法，其基本策略是跟踪数据流，即研究问题域中数据如何流动，以及在各个环节上进行何种处理，从而发现数据流和加工。得到的分析模型是数据流图，主要模型元素是数据流、加工、文件及端点，外加处理说明和数据字典。



# 结构化设计

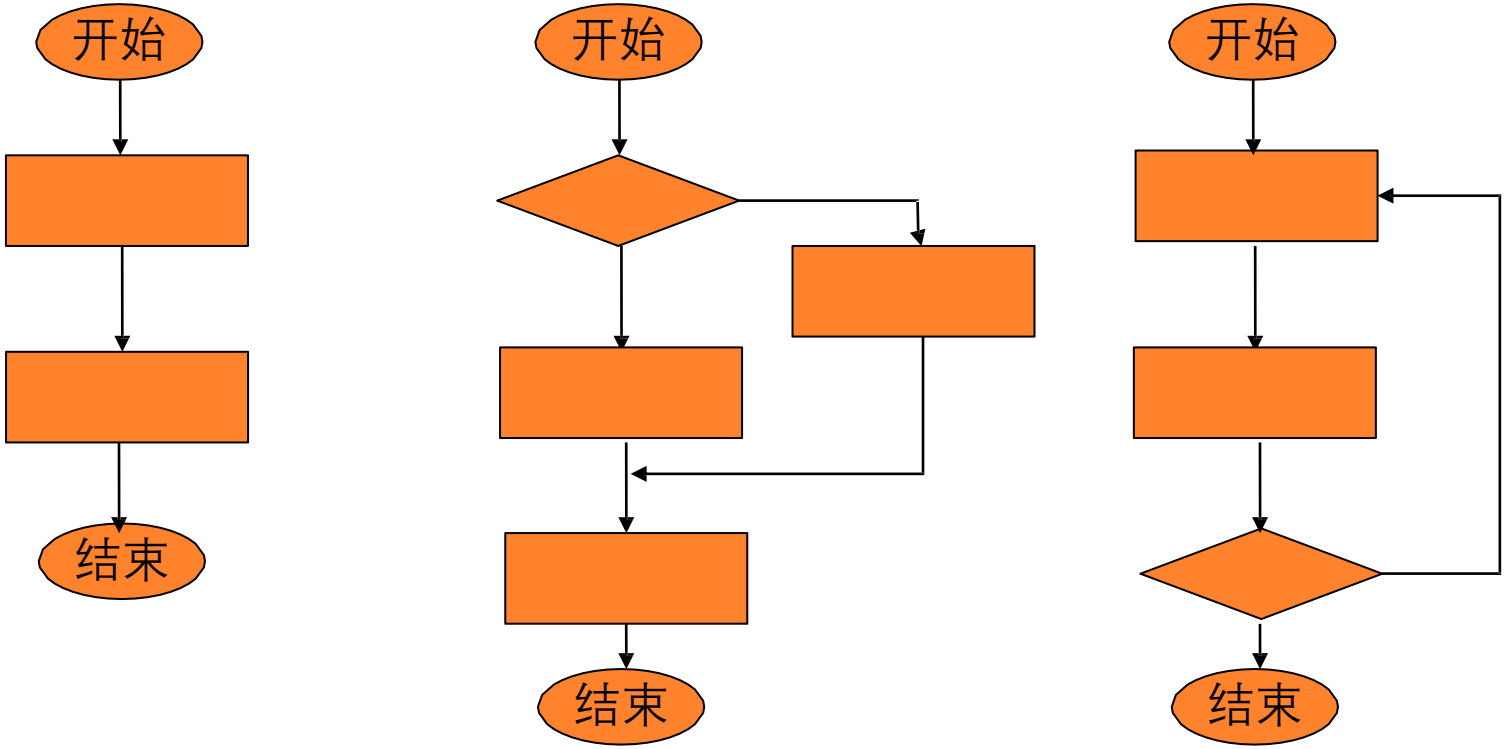



结构图

- 与功能分解法基本相同，基于模块的概念建立设计模型，分为概要设计和详细设计
- 从功能的观点设计系统
- 自顶向下，逐步分解和细化
- 将大系统分解为若干模块，主程序调用这些模块实现完整的系统功能

# 结构化编程

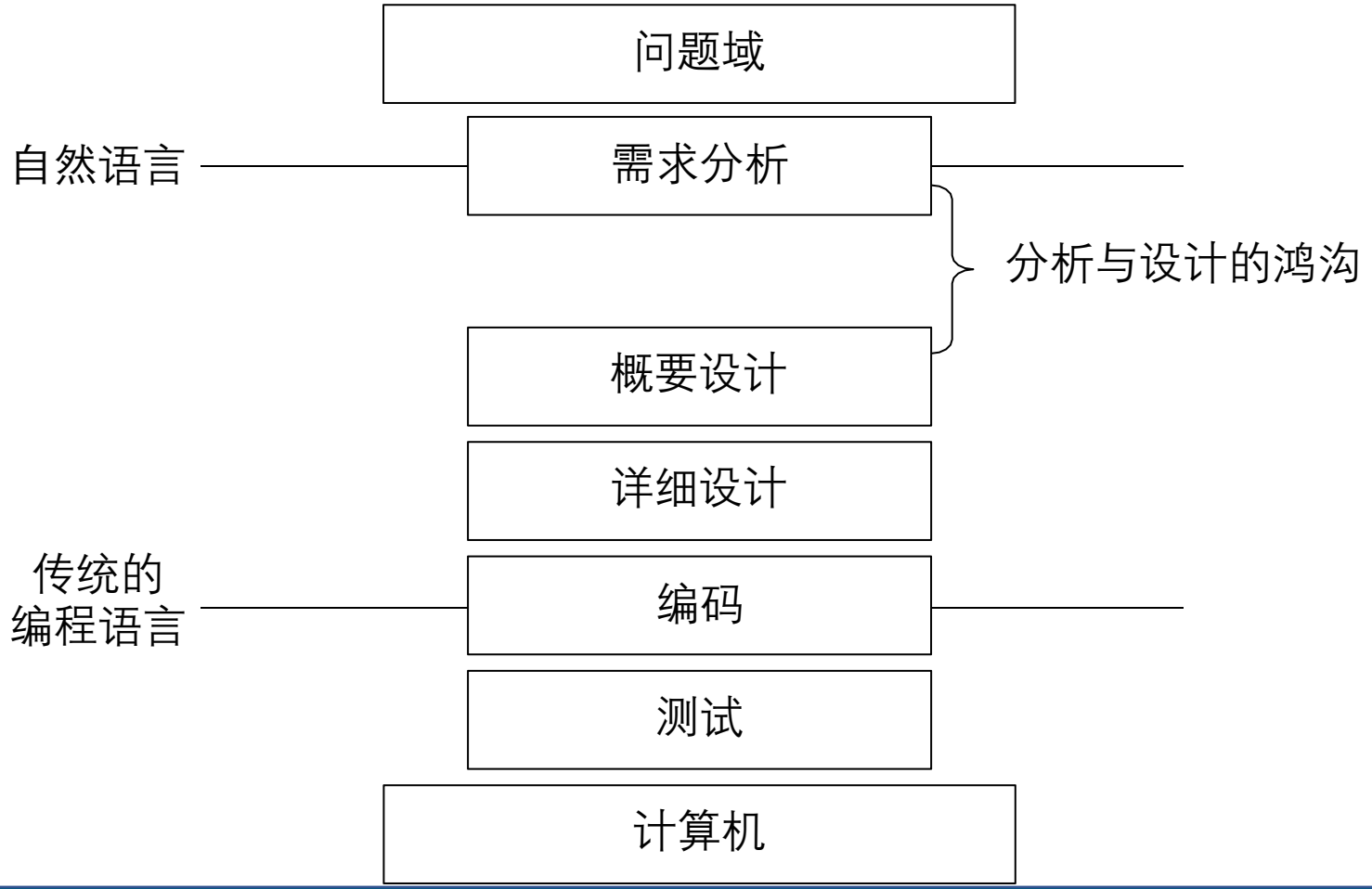
- 具有一个开始和一个结束的程序或程序模块，并且在程序执行中的每一步都由三个部分之一组成：顺序、选择或循环结构



# 结构化方法的模型

- **结构化需求分析方法通常需建立以下模型：**
  - 数据流图(Data Flow Diagram, DFD)
    - 描述系统由哪些部分组成、各部分之间有什么联系等
  - 数据字典(Data Dictionary, DD)
    - 定义了数据流图中每一个数据元素
  - 结构化语言(Structured Language)
  - 判定表或判定树(Decision Table/Tree)
    - 详细描述数据流图中不能被再分解的每一个加工的内部处理逻辑
  - 实体联系图(Entity-Relationship Diagram, E-R)
  - 状态转换图(State Transition Diagram, STD)

# 传统软件工程方法：结构化方法

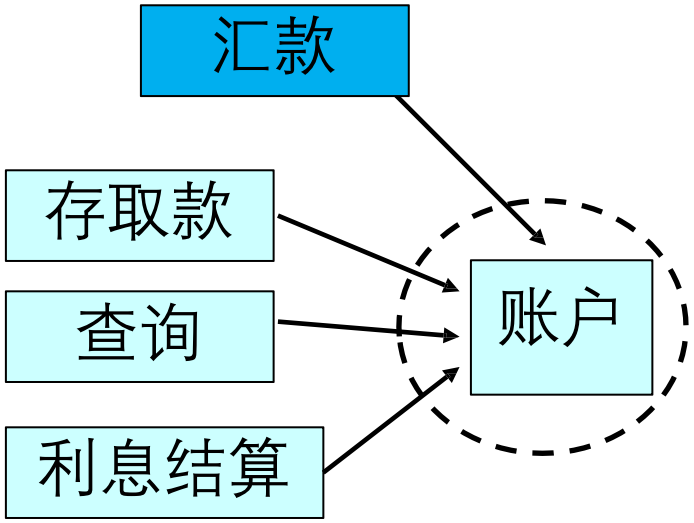


# 结构化方法的常见问题

- **需求的错误**
  - 不完整、不一致、不明确
  - 开发人员和用户无法以同样的方式说明需求
  - 需求分析方法与设计方法不一致，分析的结果不能平滑过渡到设计
- **需求的变化**
  - 需求在整个项目过程中始终发生变化
  - 系统功能不断变化
  - 许多变化出现在项目后期
  - 维护过程中发生许多变化
- **系统结构的崩溃**
  - 系统在不断的变化中最终变得不可用

# 造成上述问题的根本原因...

- **结构化方法以功能分解和数据流为核心，但是...系统功能和数据表示极有可能发生变化；**
  - 以ATM银行系统为例：帐户的可选项、利率的不同计算方式、**ATM**的不同界面；



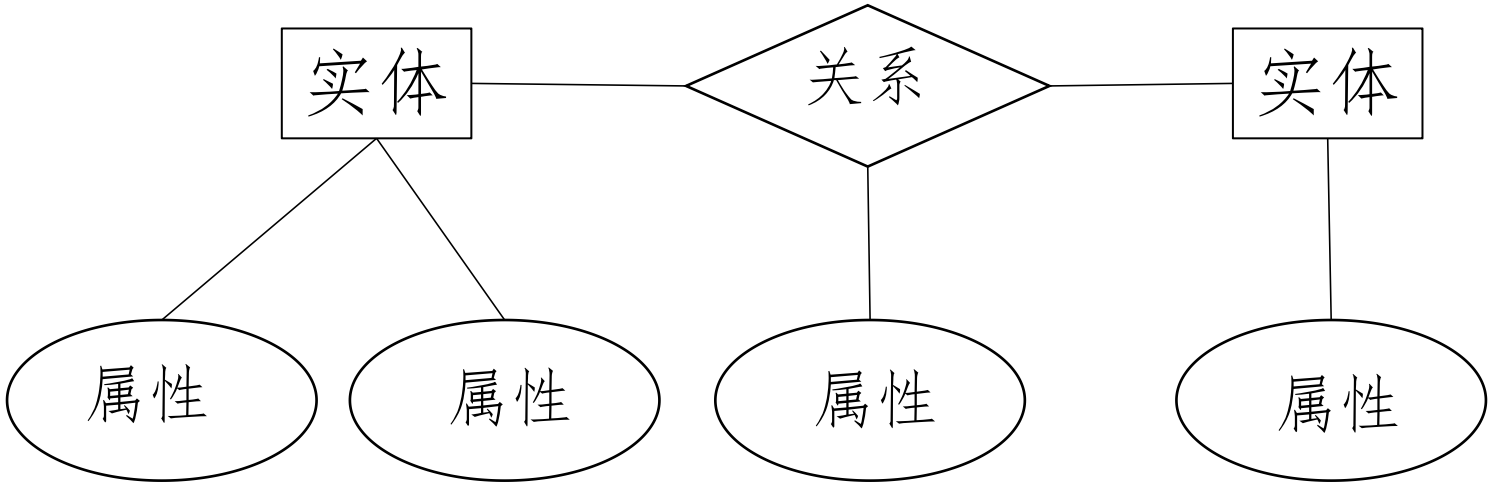
# 软件工程方法

## 信息建模法

由实体-关系法（E-R方法）发展而来，核心概念是实体和关系。实体描述问题域中的事物，关系描述事物之间在数据方面的联系，都可以带有属性。

发展之后的方法也把实体称为对象，并使用类型和子类型的概念，作为实体（对象）的抽象描述。

E-R 图



# 软件工程方法

## 信息建模法

信息建模法已经很接近面向对象方法，因此有的文献也把它称为一种面向对象方法，但有以下差别：

1. 强调的重点是信息建模和状态建模，而不是对象建模
2. 实体中只有属性没有操作
3. 只有属性的继承，不支持操作的继承
4. 没有采用消息通讯

# 4.1 软件工程开发方法与软件设计

- 软件工程开发方法
  - 传统开发方法
  - 面向对象方法
- 软件设计
  - 设计的概念
  - 设计的原则

# 软件工程方法

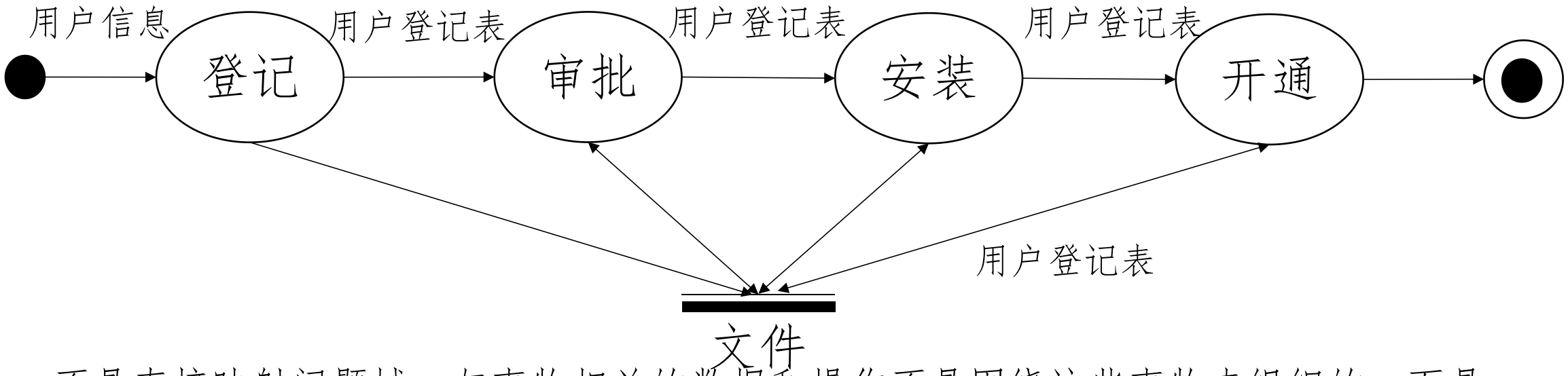
- **结构化**
  - 复杂世界—>复杂处理过程（事情的发生发展）
  - 设计一系列功能（或算法）以解决某一问题
  - 寻找适当的方法存储数据
- **面向对象**
  - 任何系统都是由能够完成一组相关任务的对象构成
  - 如果对象依赖于一个不属于它负责的任务，那么就需要访问负责此任务的另一个对象（调用其他对象的方法）
  - 一个对象不能直接操作另一个对象内部的数据，它也不能使其它对象直接访问自己的数据
  - 所有的交流都必须通过方法调用

# 举例：五子棋游戏

- **面向过程（事件）的设计思路就是首先分析问题的步骤：**
  1. 开始游戏，初始化画面
  2. 黑子走，绘制画面，
  3. 判断输赢，如分出输赢，跳至步骤6
  4. 白子走，绘制画面，
  5. 判断输赢，如未分出输赢，返回步骤2，
  6. 输出最后结果。
- **面向对象的设计思路是分析与问题有关的实体：**
  1. 玩家：黑白双方，这两方的行为是一模一样的，
  2. 棋盘：负责绘制画面
  3. 规则：负责判定诸如犯规、输赢等。

# 举例：电话安装业务系统

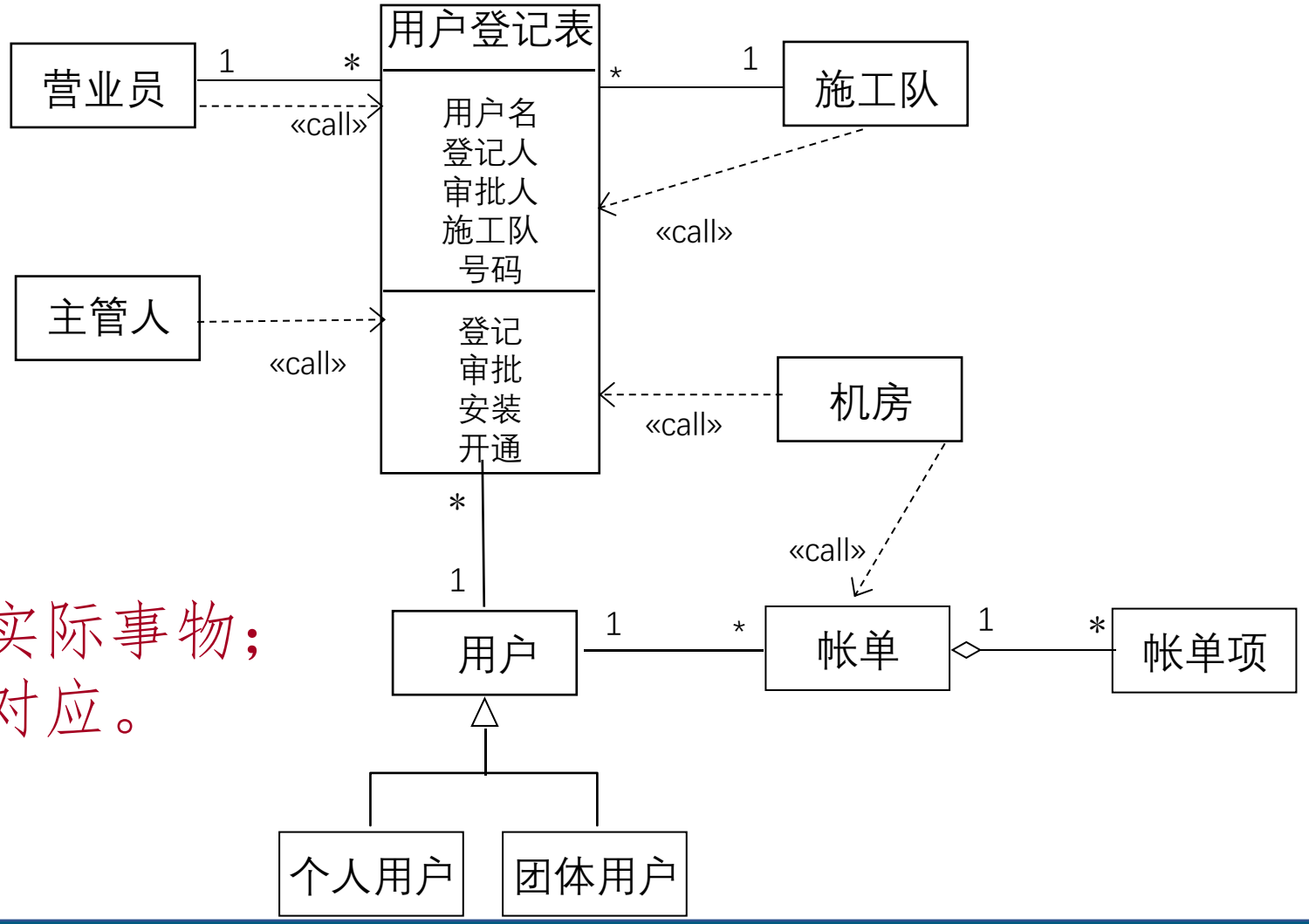
## 结构化分析-数据流和加工




- 不是直接映射问题域，与事物相关的数据和操作不是围绕这些事物来组织的，而是分散在数据流和加工中。
- 经常发生信息膨胀-模型中的多个数据流，实现中其实只是一项数据。
- 分析模型难以与设计模型及源程序对应。

# 举例：电话安装业务系统

- 面向对象方法—对象及其关系



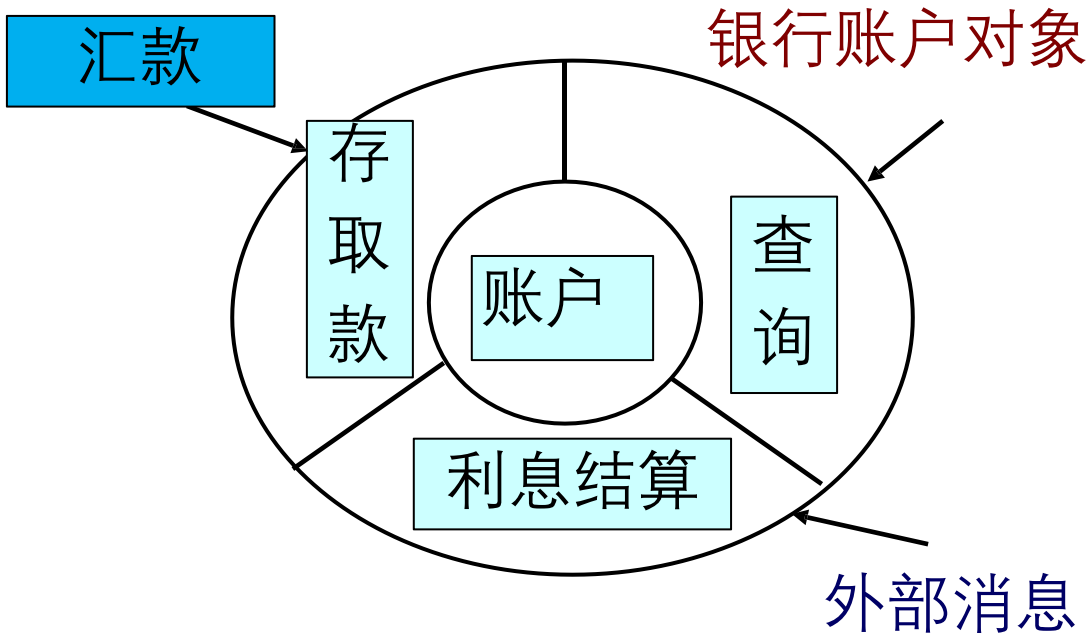

直接映射了问题域中的实际事物；
能够与程序形成良好的对应。

# 结构化和面向对象分析

- 什么情况下使用结构化的分析方法？
- 什么情况下使用面向对象的分析方法？

# 面向对象的程序开发...

- 软件设计应尽可能去描述那些极少发生变化的稳定要素：**对象**
  - 银行客户、帐户

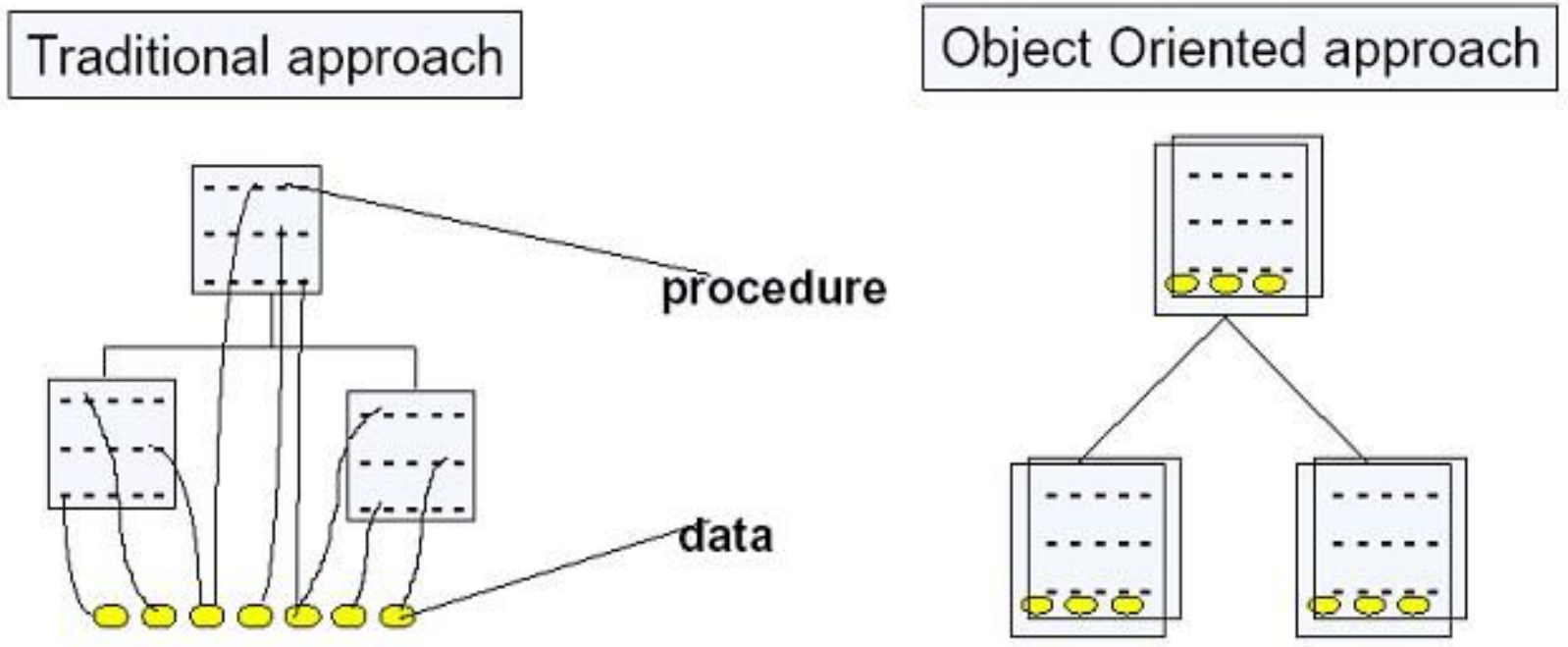


# 面向对象的程序开发

- 在结构化程序开发模式中优先考虑的是过程抽象，在面向对象开发模式中优先考虑的是实体(问题论域的对象)；
- 主要考虑对象的行为而不是必须执行的一系列动作；
  - 对象是数据抽象与过程抽象的综合；
  - 算法被分布到各种实体中；
  - 消息从一个对象传送到另一个对象；
  - 控制流包含在各个对象的操作内；
  - 系统的状态保存在各个对象所定义的数据抽象中；

# 二者的本质区别

- 面向过程的结构化系统 = 功能 + 数据
- 面向对象的系统 = 对象 + 消息



# 面向对象方法的优势

- **面向对象模型更接近于问题域(尽可能模拟人类习惯的思维方式)**
  - 以问题域中的对象为基础建模
  - 以对象、属性和操作对问题进行建模
- **反复细化高层模型直到可以实现的程度**
  - 努力避免在开发过程中出现大的概念跳变
- **将模型组织成对象的集合**
  - 真实世界中的具体事物
    - 售货员、商品、仓库、顾客
    - 飞机、机场等
  - 逻辑概念
    - 商品目录、生产计划、销售
    - 操作系统中的分时策略、军事训练中的冲突解决规则等

# 面向对象方法

- 把系统看做是一起工作来完成某项任务的相互作用的对象的集合
  - 面向对象分析
  - 面向对象设计
  - 面向对象编程

# 面向对象方法

- **面向对象分析(Object Oriented Analysis, OOA)**
  - 分析和理解问题域，找出描述问题域和系统责任所需的类及对象，分析它们的内部构成和外部关系，建立OOA 模型。
- **面向对象设计(Object Oriented Design, OOD)**
  - 将OOA 模型细化，描述对象间交互，变成OOD 模型，并且补充与一些实现有关的部分，如人机界面、数据存储、操作细节等。
- **面向对象编程(Object Oriented Programming, OOP)**
  - 用一种面向对象的编程语言将OOD 模型中的各个成分编写成程序，由于从OOA→OOD→OOP实现了无缝连接和平滑过渡，因此提高了开发工作的效率和质量。

# OOA/D

- 分析：强调的是对问题和需求的调查研究，而不是解决方案
  - 面向对象分析过程中，强调的是在问题领域内发现和描述对象
- 设计：强调的是满足需求的概念上的解决方案（在软件方面和硬件方面），而不是其实现。
  - 面向对象设计过程中，强调的是定义软件对象以及它们如何协作以实现需求。
- 有价值的分析和设计可以概括为：做正确的事（分析）和正确地做事（设计）

# 面向对象的基本概念

- 对象(Object)
- 类(Class)
- 继承(Inheritance)
- 多态(Polymorphism)
- 消息(Message)

# 对象(Object)

- 对象(Object)：具有责任的实体。一个特殊的，自成一体的容器，对象的数据对于外部对象是受保护的。
- 特性：标识符（区别其他对象）、属性（状态）和操作（ 行为）。
  - 属性(Attribute)：与对象关联的数据，描述对象静态特性；
  - 操作(Operation)：与对象关联的函数，描述对象动态特性；
- 示例
  - 学生：王洪波
  - 属性：姓名，性别，年级，院系；年级 = “2019”
  - 操作：选课、撤销、支付

对象是具体、有意义的个体

# 对象(Object)

- 对象是类的实例，表示为：

```
  Fred_Bloggs:Employee
name: Fred Bloggs
Employee #: 234609234
Department: Marketing
```

两个不同的对象可以有相同的属性取值(正如两个同名的人，或者住在同一栋楼的人)

- 对象与其他对象之间发生关联关系

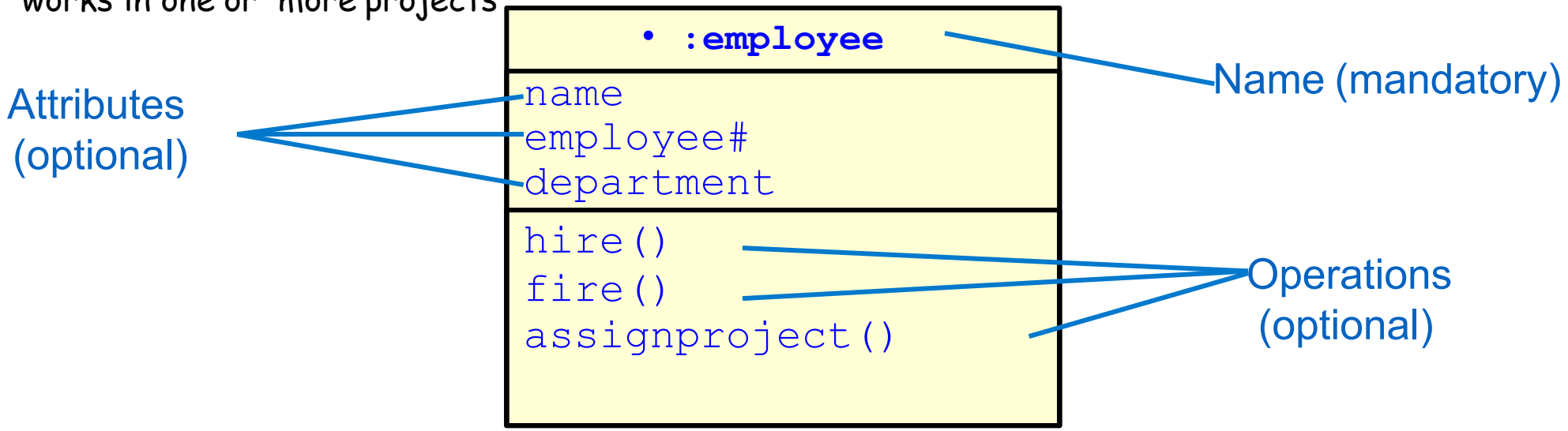
例如：**Fred_Bloggs:employee** 对象与 **KillerApp:project** 对象相关联

- 但这个关联关系要定义为类之间的关系注意将属性划归正确的类

例如：不要将经理姓名和员工编号同时定义为project类的属性

# 类(Class)

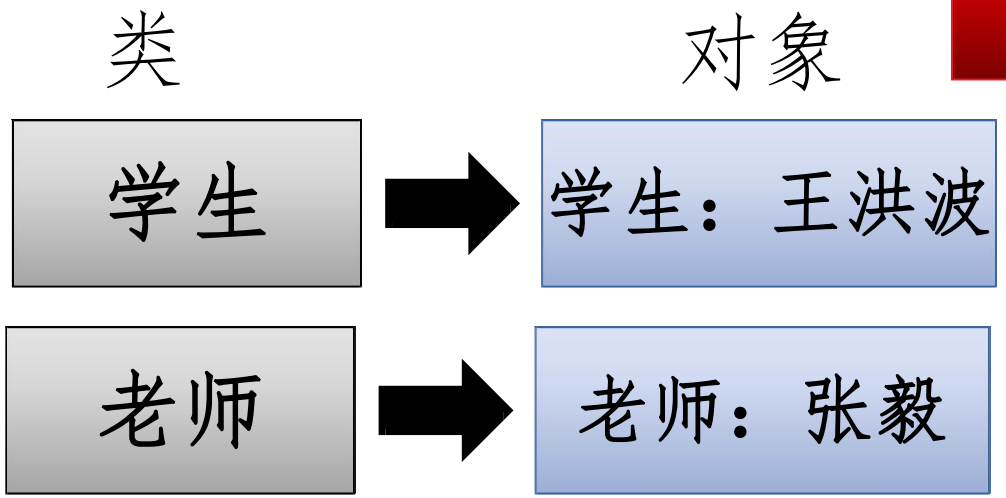
- 类是具有以下特征的对象集合
  - 相同性质(attributes)
  - 相同行为(operations)
  - 相同的对象关系
  - 相同语义(“semantics”)
- 举例
  - employee: has a name, employee# and department; an employee is hired, and fired; an employee works in one or more projects

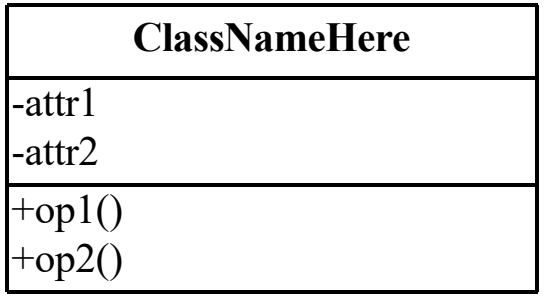


# 类(Class)

- 类(Class)：具有相同属性和操作的一组对象的抽象，它为属于该类的全部对象提供了统一的抽象描述。
  - 类是概念定义，抽象了同类对象共同的属性和操作
  - 对象是类的一个实例
- 示例

类是一种分类和组织机制及方法



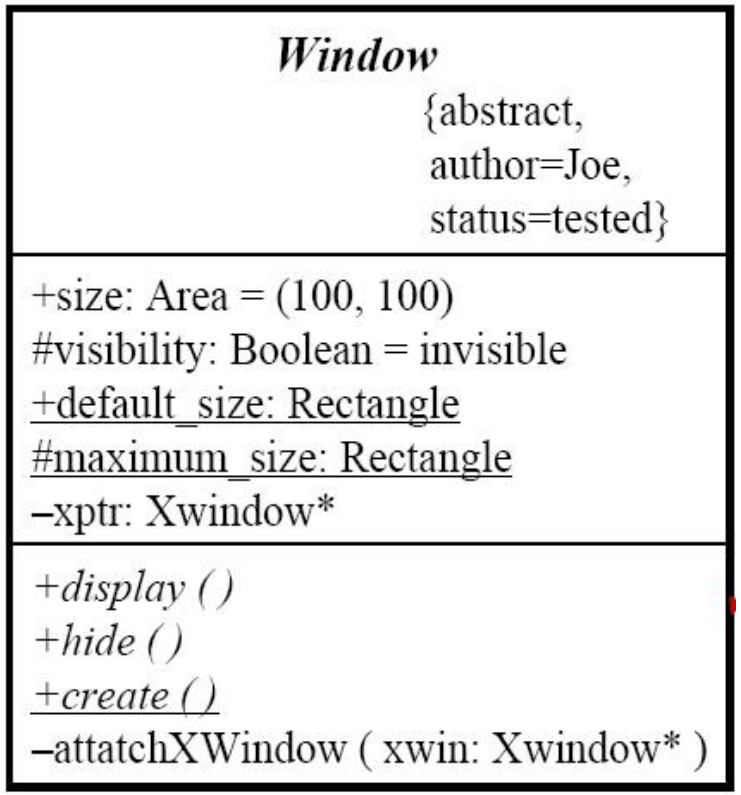
# 类与对象的对比

- **类与对象的比较**
  - “同类对象具有相同的属性和操作”是指它们的定义形式相同，而不是说每个对象的属性值都相同。
  - 类是静态的，类的存在、语义和关系在程序执行前就已经定义好了。
  - 对象是动态的，对象在程序执行时可以被创建和删除。

- **在面向对象的系统分析和设计中，并不需要逐个对对象进行说明，而是着重描述代表一批对象共性的类。**

# 类的属性(Attribute)

- **类的属性：描述对象“静态”(结构)特征的一个数据项；**
- **属性的“可见性”(Visibility)分类：**
  - 公有属性(public)　　+
  - 私有属性(private)　　-
  - 保护属性(protected)　#
- **属性的表达方式：**
  - 可见性属性名：数据类型=初始值



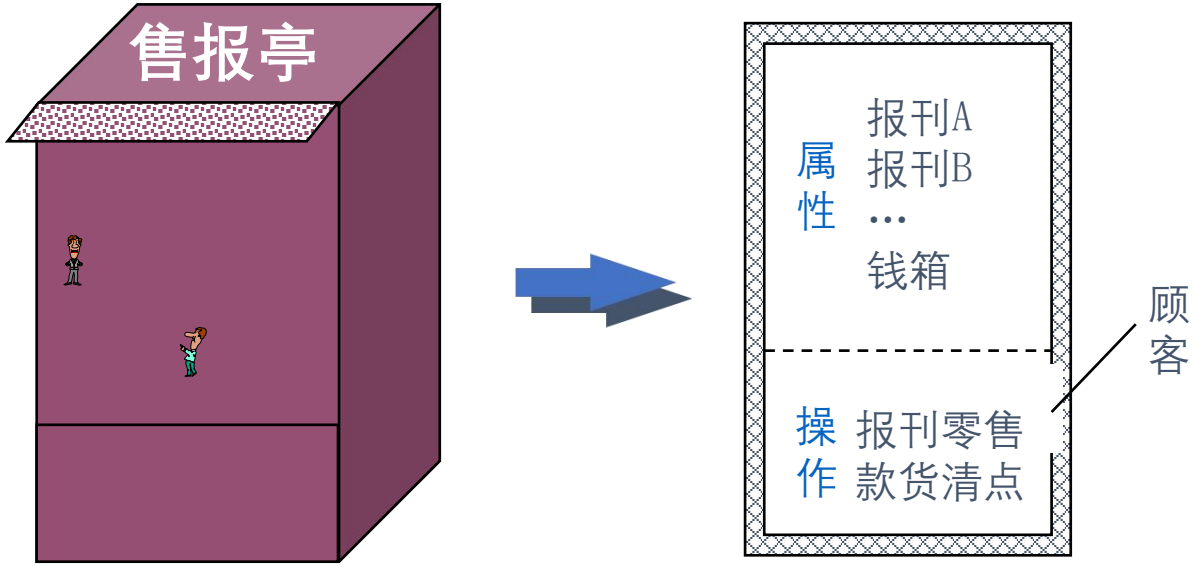
# 封装(Encapsulation)

- 封装(Encapsulation)：把对象的属性和操作结合成一个独立的单元，并尽可能对外界隐藏数据的实现过程，隐藏实现细节；
- 一个对象不能直接操作另一个对象内部数据，它也不能使其他对象直接访问自己的数据，所有的交流必须通过 方法调用。
  - getXXX()
  - setXXX()
- 例如：对“汽车”对象来说，“司机”对象只能通过方向盘和仪表来操作“汽车”，而“汽车”的内部实现机制则被隐藏起来。

# 封装(Encapsulation)

- 封装的重要意义：
  - 保护对象，避免用户误用；
  - 保护客户端(调用程序)，其实现过程的改变 不会影响到相应客户端的改变。

- 封装带来的问题：
  - 编程的麻烦
  - 执行效率的损失
- 解决办法：
  - 不强调严格封装，
  - 实行可见性控制。



# 消息(Message)

**消息是实现对象间交互和协同的方法**

- **消息传递是对象间通信的手段**
  - 通过消息请求/提供服务
- **一个对象向另一个对象发送消息来请求其服务**
  - 消息描述：接收对象名、操作名和参数
- **消息类型**
  - 同步消息，请求者需要等待响应者返回
  - 异步消息，请求者发出消息后继续自己工作，无需等待响应者返回
- **示例**

# 泛化（Generalization）/继承（Inheritance）

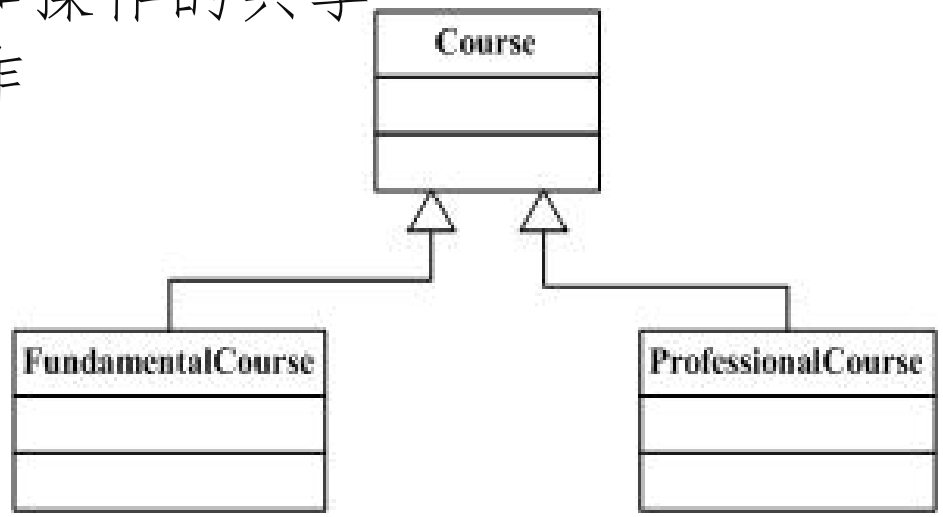
- 泛化关系是类元的一般描述和具体描述之间的关系，具体描述建立在一般描述的基础之上，并对其进行了扩展。
- 在共享祖先所定义的成分的前提下允许它自身定义增加的描述，这被称作继承。

# 继承(Inheritance)

**继承可形成层次化的类结构**

- **表示类与类之间的一般与特殊关系**
  - 何为“一般与特殊”关系?
  - 模拟现实世界的遗传关系
- **子 (特殊)类可以继承共享父 (一般)类的属性和操作**
  - 刻画类间的内在联系以及对属性和操作的共享
  - 子类可以有自己的独特属性和操作
- **示例**

“课程”、“公共课”、“专业课”

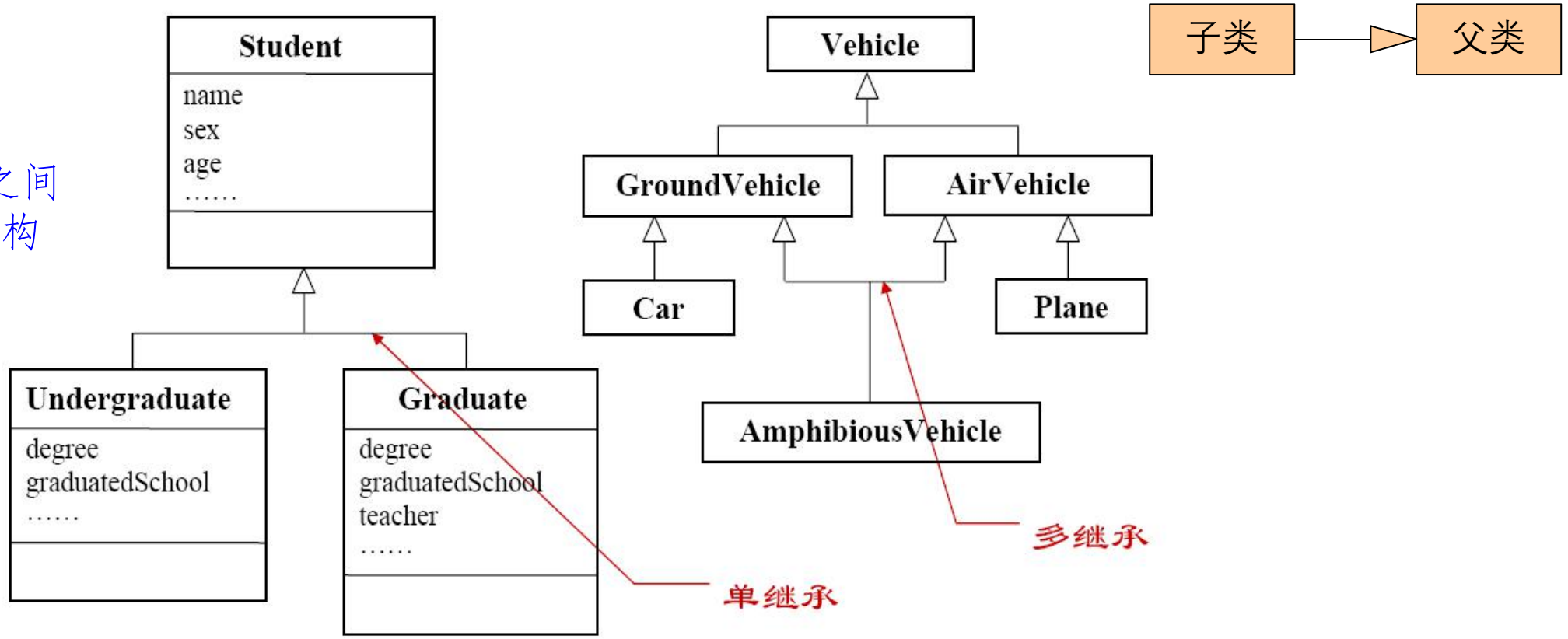



**举例说明继承关系**

# 继承(Inheritance)

**继承(Inheritance)：** 子类可自动拥有父类的全部属性和操作。

继承关系使类之间形成层次化结构



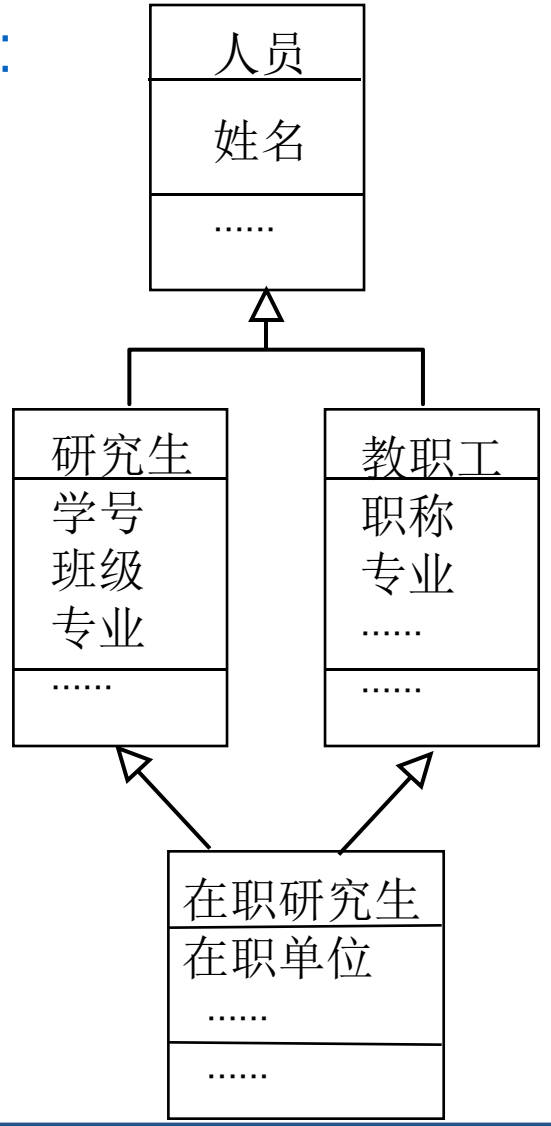
# 继承(Inheritance)

- 单一继承：一个子类只有唯一的一个父类
- 多重继承：一个子类有一个以上的父类

子类可以继承父类的属性和方法

Java不支持多重继承

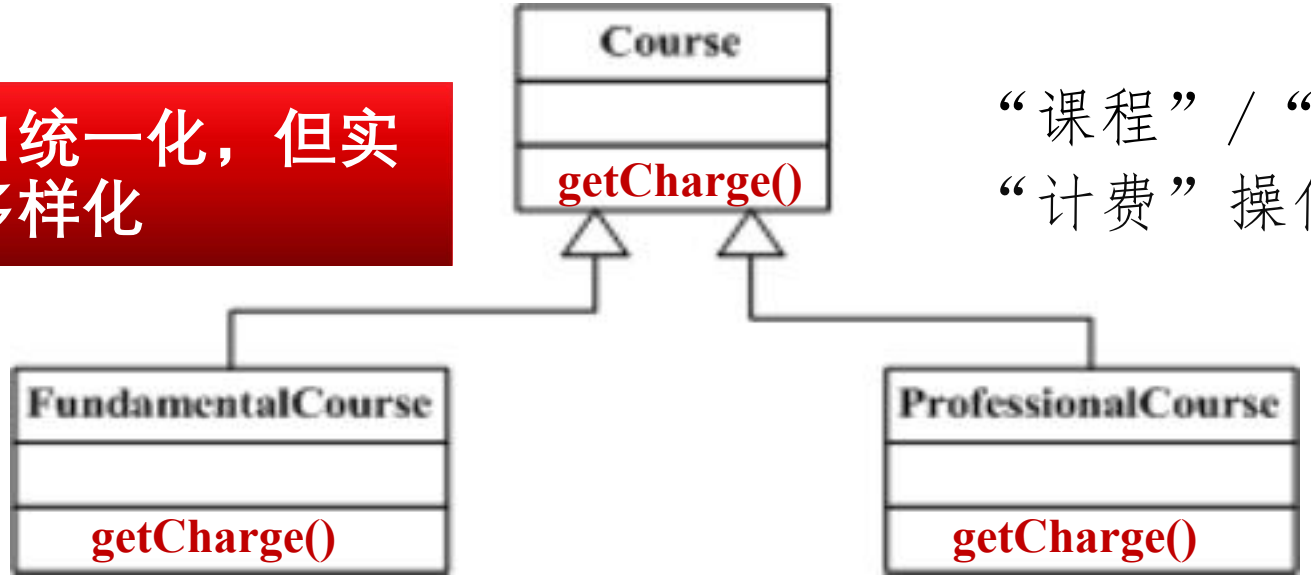
例:



# 多态(Polymorphism)

- 同一个操作作用于不同的对象上可以有不同的解释，并产生不同的执行结果。
  - 接口定义形式相同，但是实现形态不一样
  - 相同的消息发送给不同的对象时，每个对象将根据自己所属类中定义的这个操作去执行，从而产生不同的结果

**多态支持对外接口统一化，但实现方式多样化**




“课程”/“公共课”/“专业课”“计费”操作实现不尽相同

# 小结：面向对象技术

- **面向对象技术(Object Oriented Technology)**
  - 客观世界是由对象组成的，任何客观事物或实体都是对象；复杂对象可以由简单对象构成；
  - 具有相同数据和相同操作的对象可以归并为一个统一的“类”，对象是类的实例；
  - 类可以派生出子类，子类继承父类的全部特性(数据和操作)，同时加入了自己的新特性；子类和父类形成层次结构；
  - 对象之间通过消息传递相互关联；
  - 类具有封装性，其数据和操作对外是不可见的，外界只能通过消息请求某些操作；
  - 具体的计算则是通过新对象的建立和对象之间的通信来执行的。

# 4.1 软件工程开发方法与软件设计

- 软件工程开发方法
  - 传统开发方法
  - 面向对象方法
- 软件设计
  - 设计的概念
  - 设计的原则

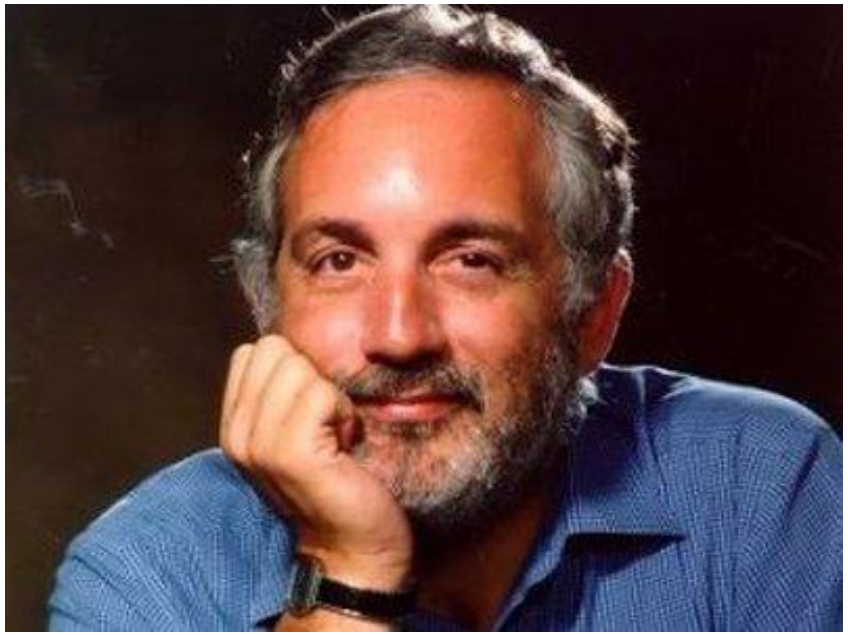
# 设计=天才+创造力

- 每个工程师都希望做设计工作，因为这里有“创造性”——客户需求、业务要求和技术限制都在某个产品或系统中得到集中的体现。
- “设计”是最充分展现工程师个人价值的工作。

# “设计”的本质

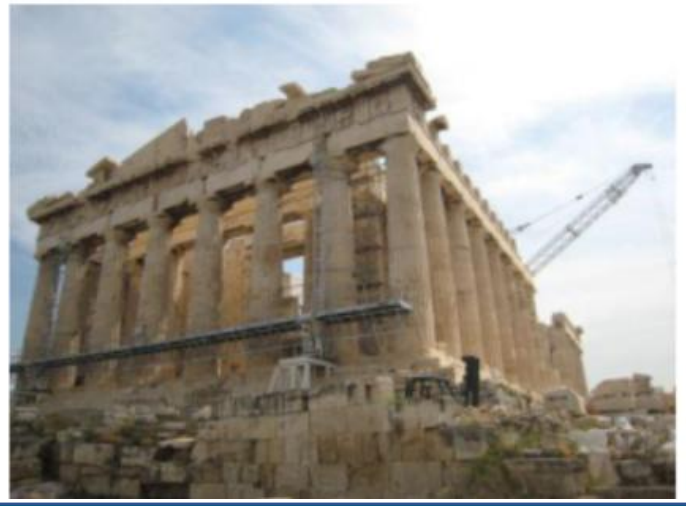
- 什么是设计？设计是你身处两个世界，技术世界和人类的目标世界，而你尝试将这两个世界结合在一起。

--- Mitch Kapor，《软件设计宣言》



# 从建筑设计看软件设计

- “设计良好的建筑应该展示出坚固、适用和令人赏心悦目的特点”。
- 对好的软件设计来说也是如此
  - 适用：软件要符合开发的目标，满足用户的需求
  - 坚固：软件应该不含任何妨碍其功能的缺陷
  - 赏心悦目：使用软件的体验应该是愉快的

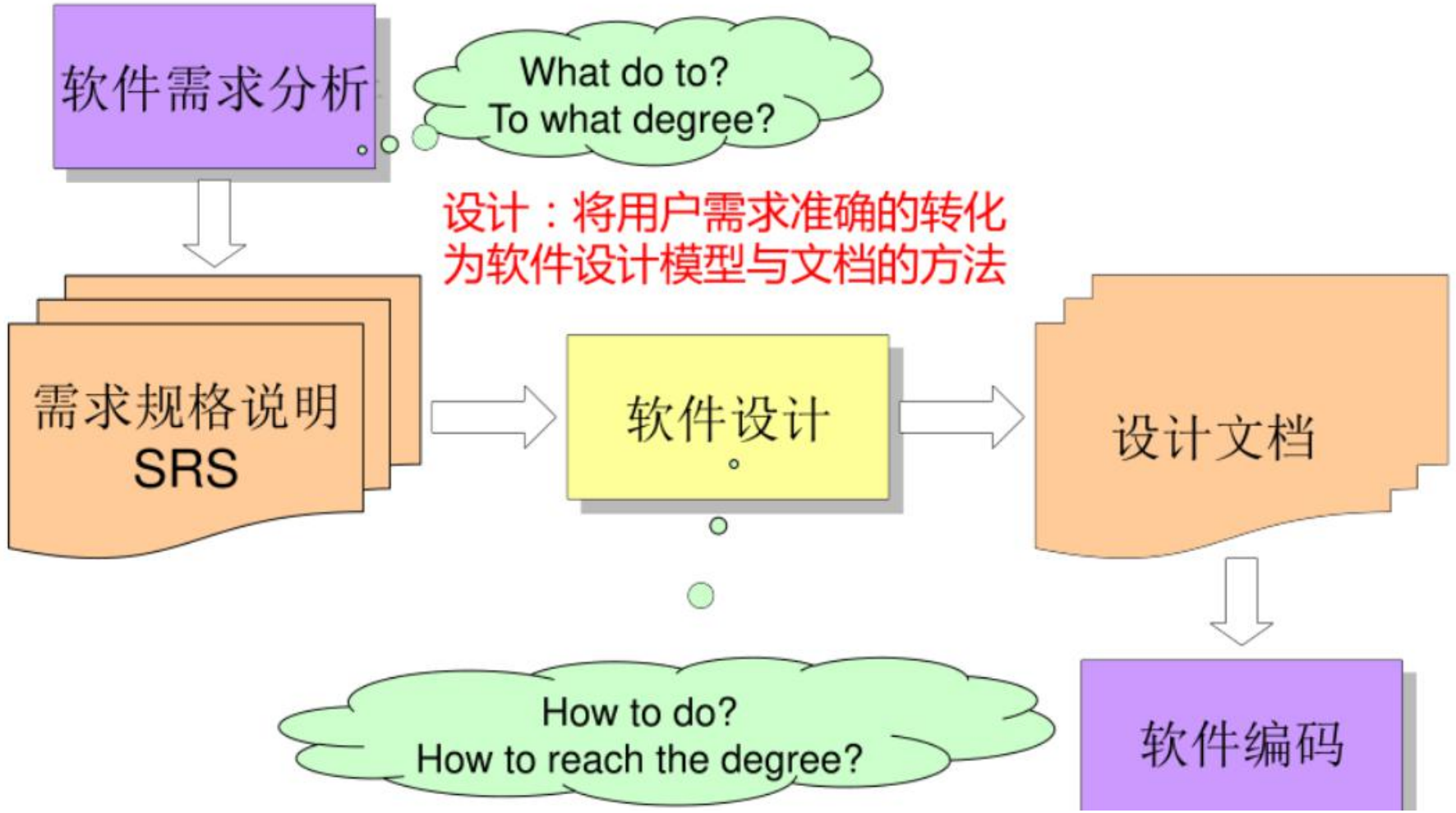

# 良好的软件设计的三个特征

- 目标：设计必须是实现所有包含在分析模型中的明确需求、以及客户期望的所有隐含需求
- 形态：对开发、测试和维护人员来说，设计必须是可读的、可理解的、可操作的指南
- 内容：设计必须提供软件的全貌，从实现的角度去说明功能、数据、行为等各个方面。

# “软件设计”的定义

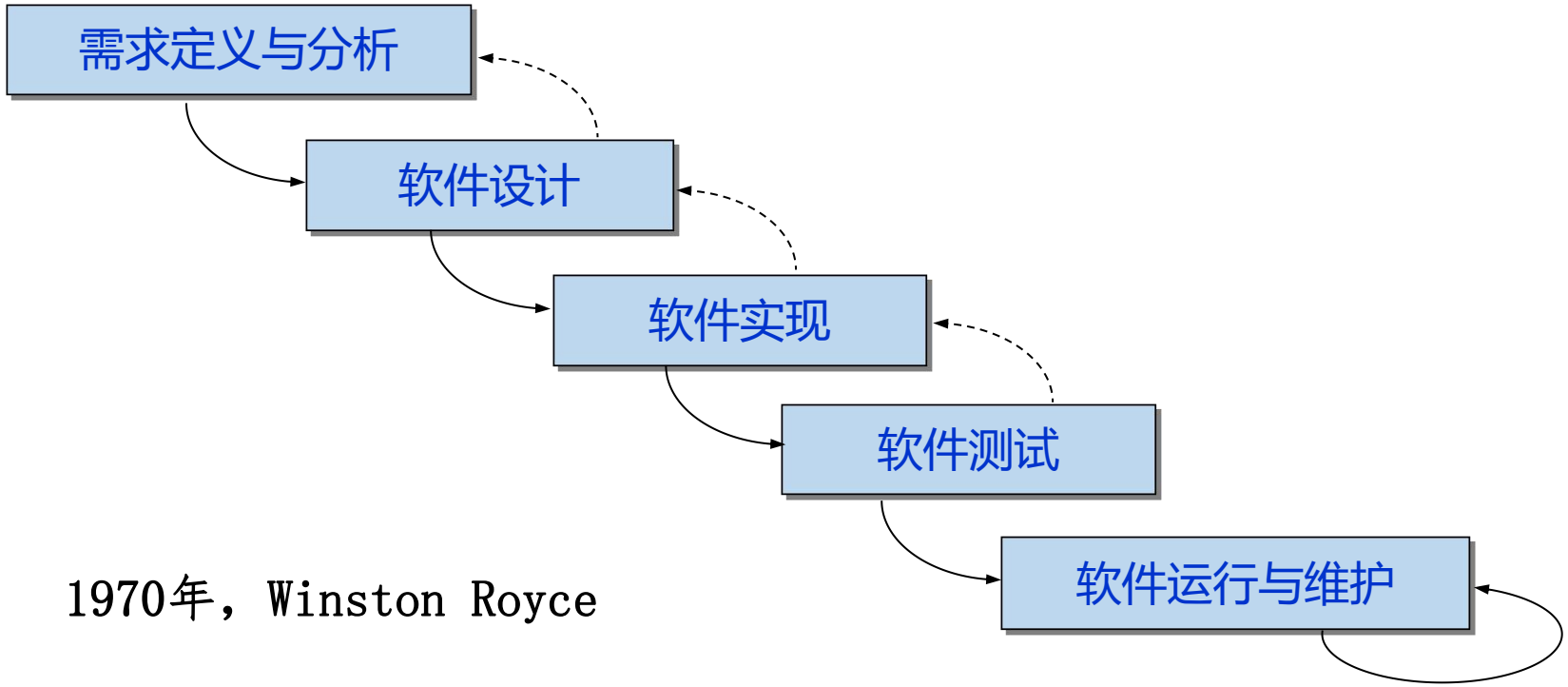
- **设计：为问题域的外部可见行为的规约增添实际的计算机系统实现所需的细节，包括关于人机交互、任务管理和数据管理的细节。**
  **——Coad/Yourdon**
- 关于“软件设计”的几个小例子：
  - 需求1：教学秘书需要将学生的综合成绩按高到低进行排序
  - 设计1：void OrderScores ( struct * scores[ ]) { 冒泡排序算法, step1; step2;… }
  - 需求2：“销售订单”
  - 设计2：关系数据表Order(ID, Date, Customer, …);
    OrderItem(No, PROD, QUANTITY, ..)
  - 需求3：“查询满足条件的图书”
  - 设计3：图形化web用户界面

# 软件设计在SE中所处的位置



# 软件设计在SE中所处的位置

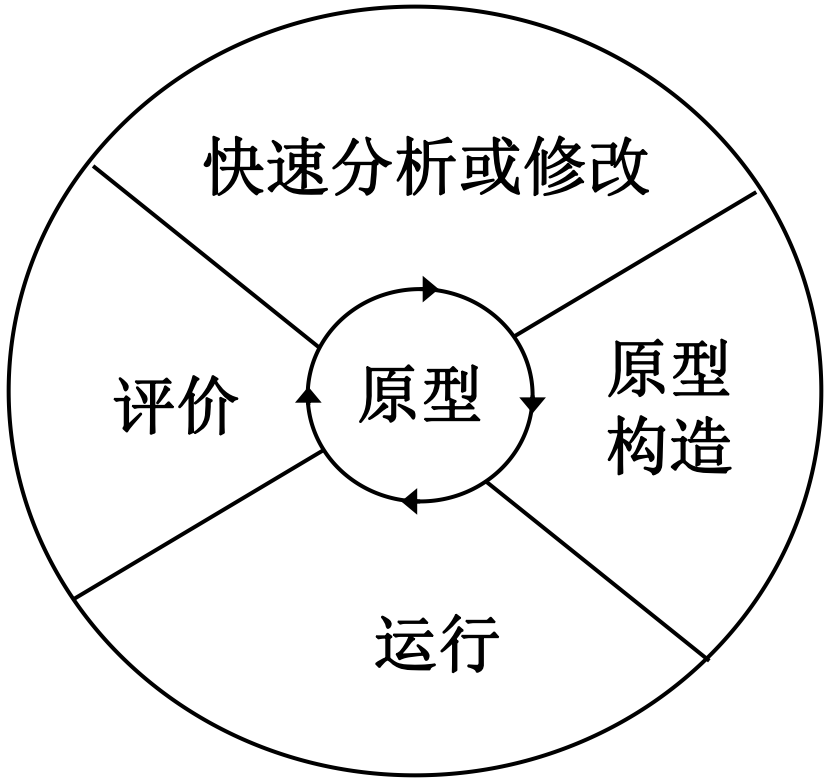
- 设计所处的位置？




1970年，Winston Royce

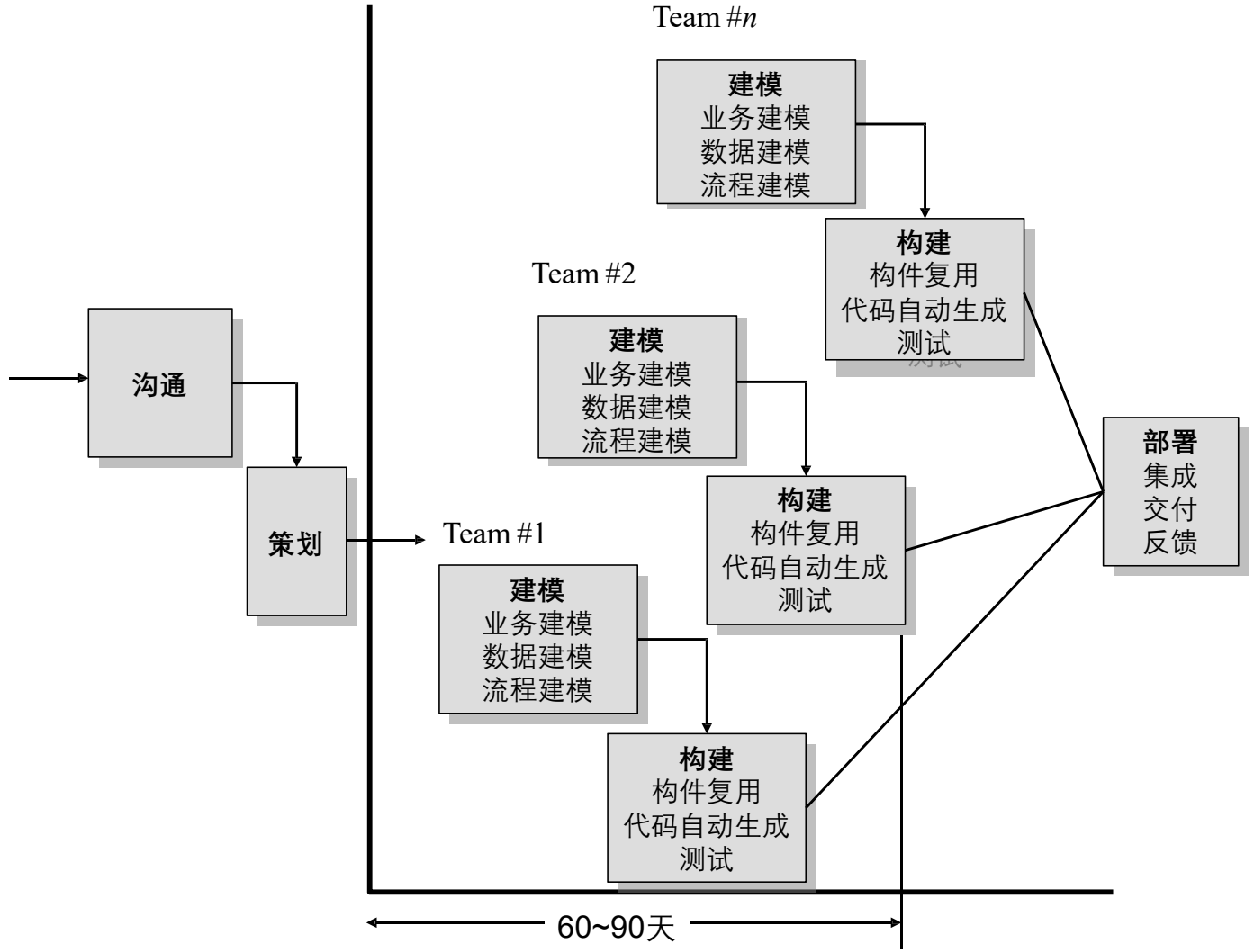
# 软件设计在SE中所处的位置

- 设计所处的位置？



# 软件设计在SE中所处的位置

- 设计所处的位置？



# 设计的目标：质量

# 设计的目标：质量

- “设计阶段”是软件工程中形成质量的关键阶段，其后所有阶段的活动都要依赖于设计的结果。
- “编写一段能工作的灵巧的代码是一回事，而设计能支持某个长久业务的东西则完全是另一回事。”

# 软件质量

- **外部质量：面向最终用户**
  - 如易用性、效率、可靠性、正确性、完整性等
- **内部质量：面向软件工程师，技术的角度**
  - 如可维护性、灵活性、可重用性、可测试性等

# 4.1 软件工程开发方法与软件设计

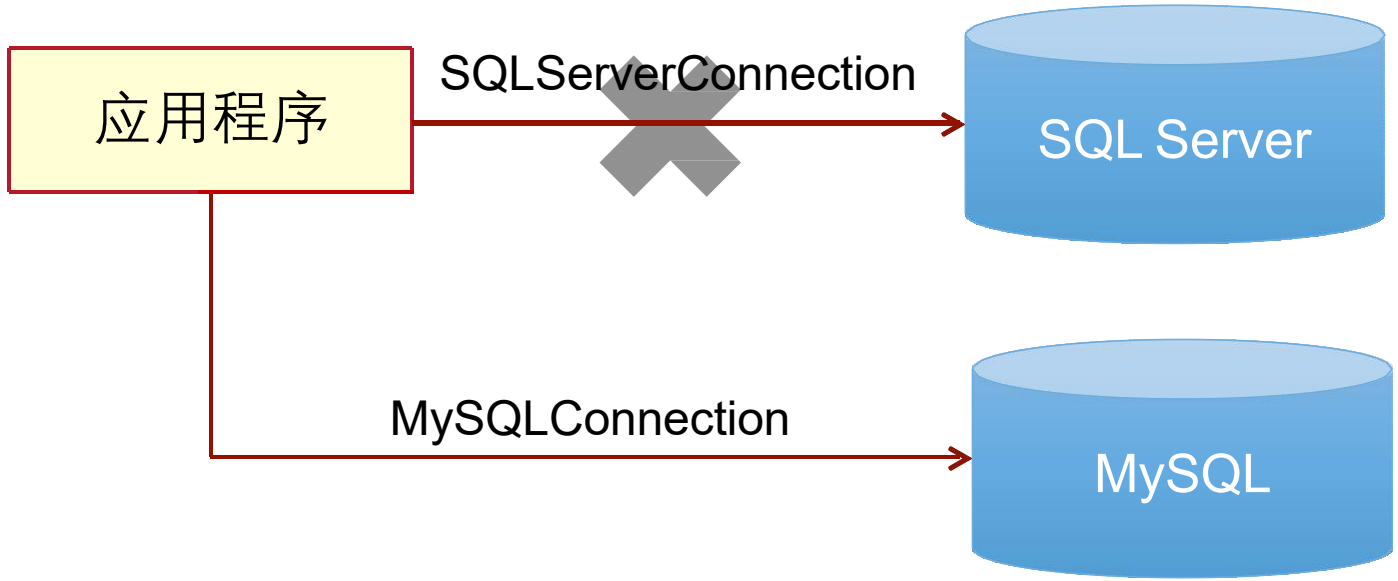
- 软件工程开发方法
  - 传统开发方法
  - 面向对象方法
- 软件设计
  - 设计的概念
  - 设计的原则

# 软件设计的原则

- 设计原则是系统分解和模块设计的基本标准，应用这些原则可以使代码更加灵活、易于维护和扩展。

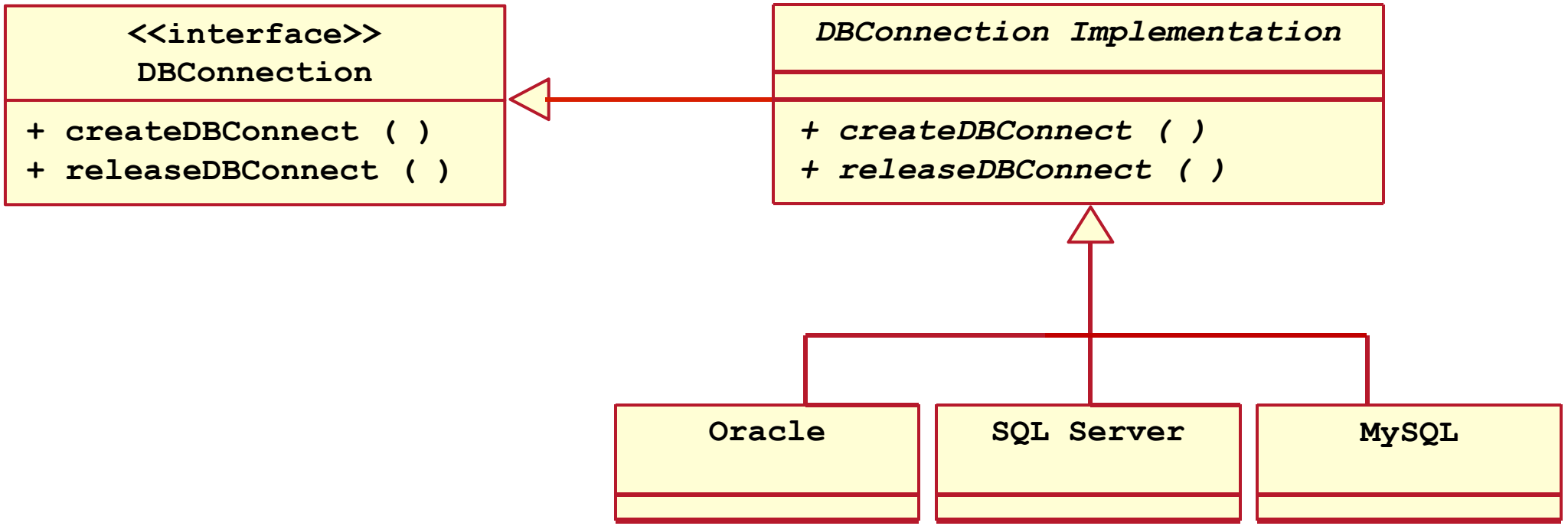
# 抽象

- 抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。



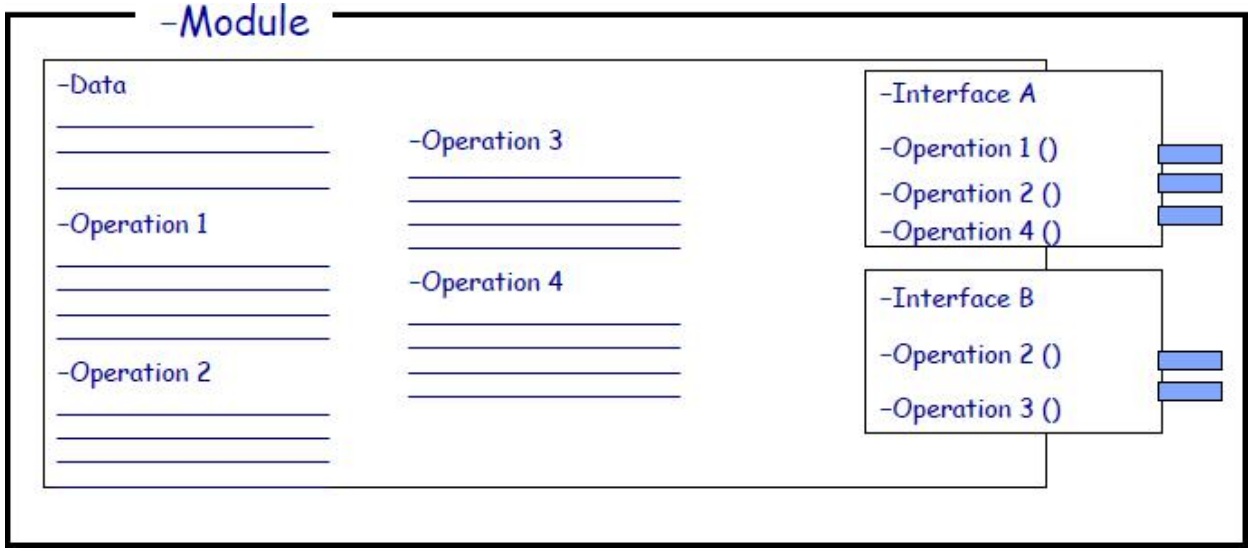
# 抽象

- 抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。



# 封装

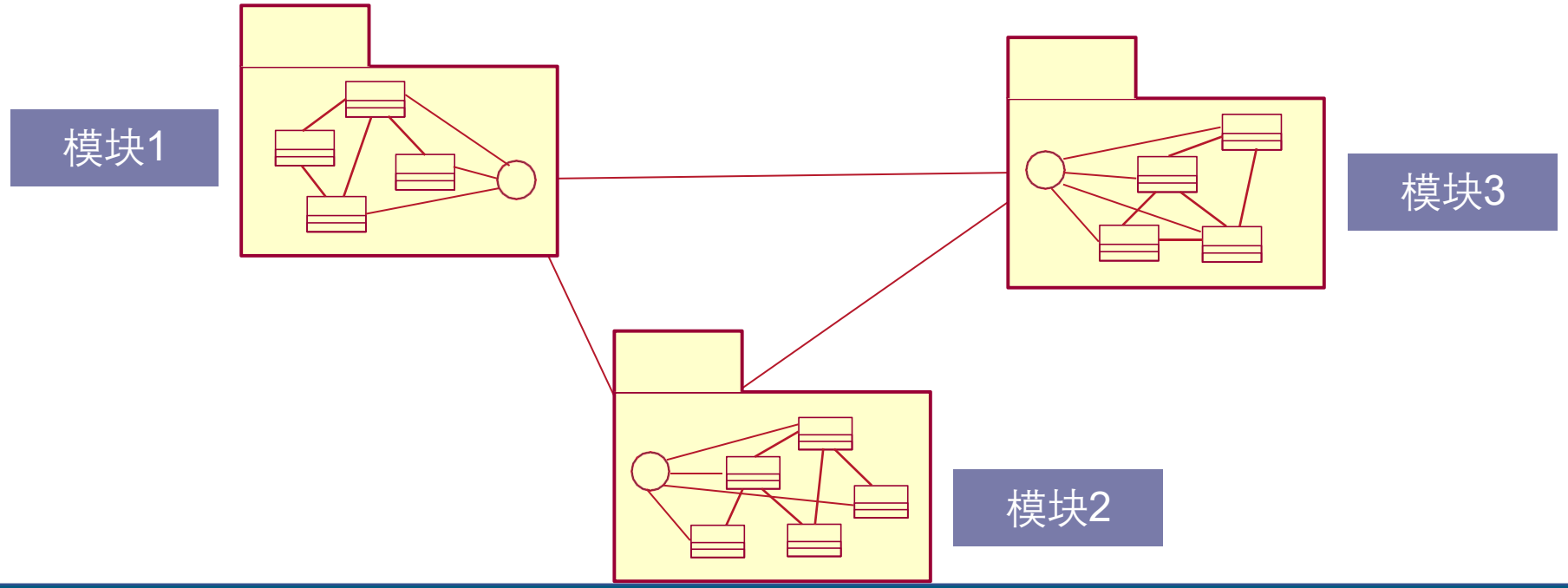
- 封装和信息隐藏是指每个软件单元对其他所有单元都隐藏自己的设计决策，各个单元的特性通过其外部可见的接口来描述。




要求：应将单元接口设计得尽可能简单，并将单元对于环境的假设和要求降至最低。

# 模块化

- 模块化是在逻辑和物理上将整个系统分解成多个更小的部分，其实质是“分而治之”，即将一个复杂问题分解成若干个简单问题，然后逐个解决。



# 系统分解原则

- 内聚性尽量将同一个功能的模块彼此之间的通信都能放在模块内部封装起来。
  - 如果一个模块或子系统含有许多彼此相关的元素，并且它们执行类似任务，那么其内聚性比较高；如果一个模块或子系统含有许多彼此不相关的元素，其内聚性就比较低。

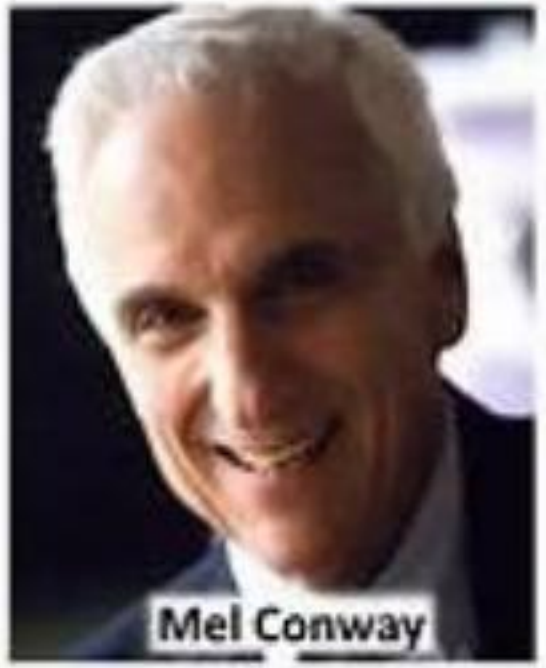
系统分解的目标：高内聚、低耦合

- 耦合性两个模块或子系统之间依赖关系的强度。
  - 如果两个模块或子系统是松散耦合的，二者相互独立，那么当其中一个发生变化时对 另一个产生的影响就很小；如果两个模块或子系统是紧密耦合的，其中一个发生变化就可能对另一个产生较大影响。

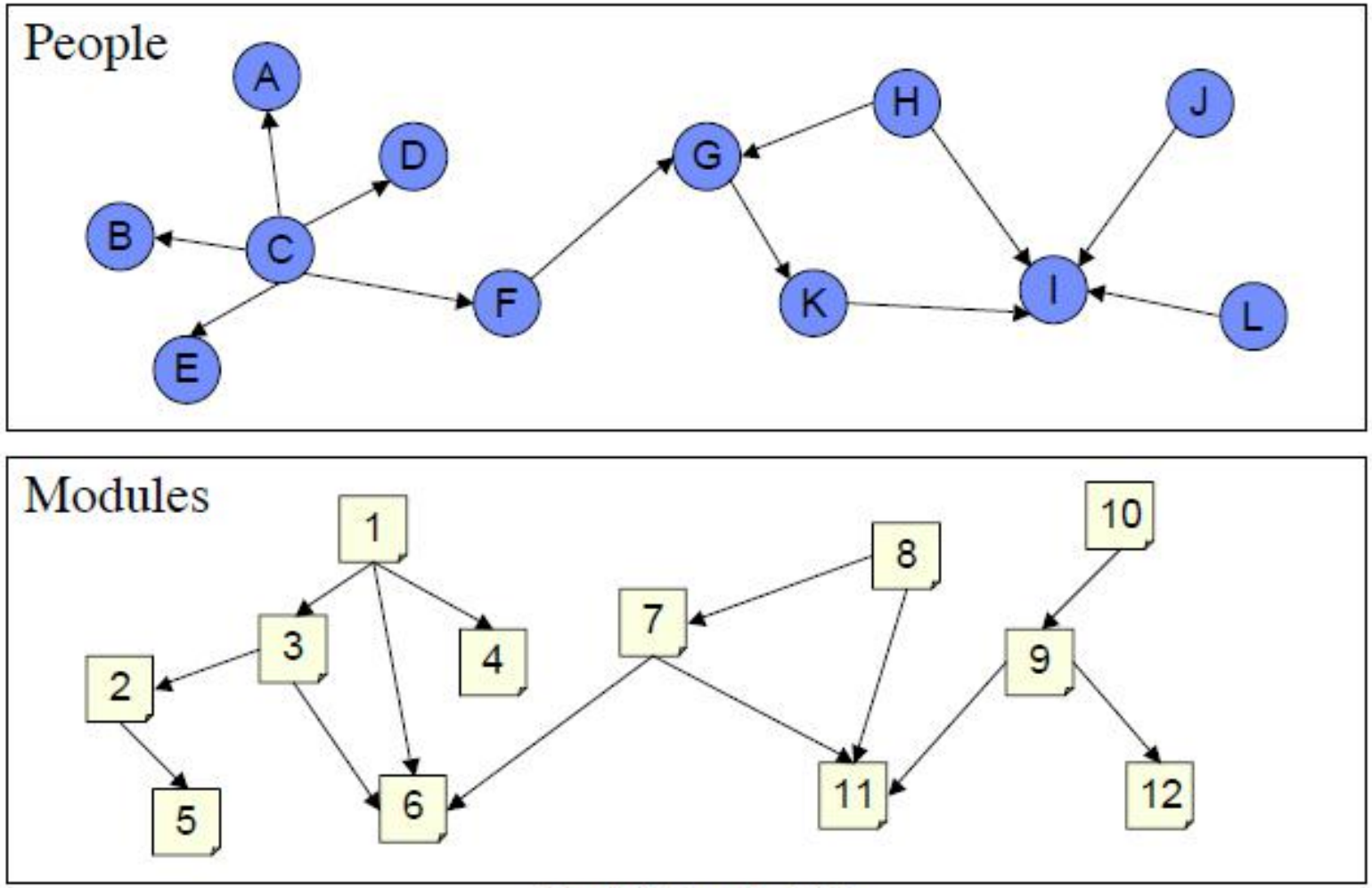
# 系统分解原则

"The structure of a software system reflects the structure of the organization that built it"

--Conway's Law



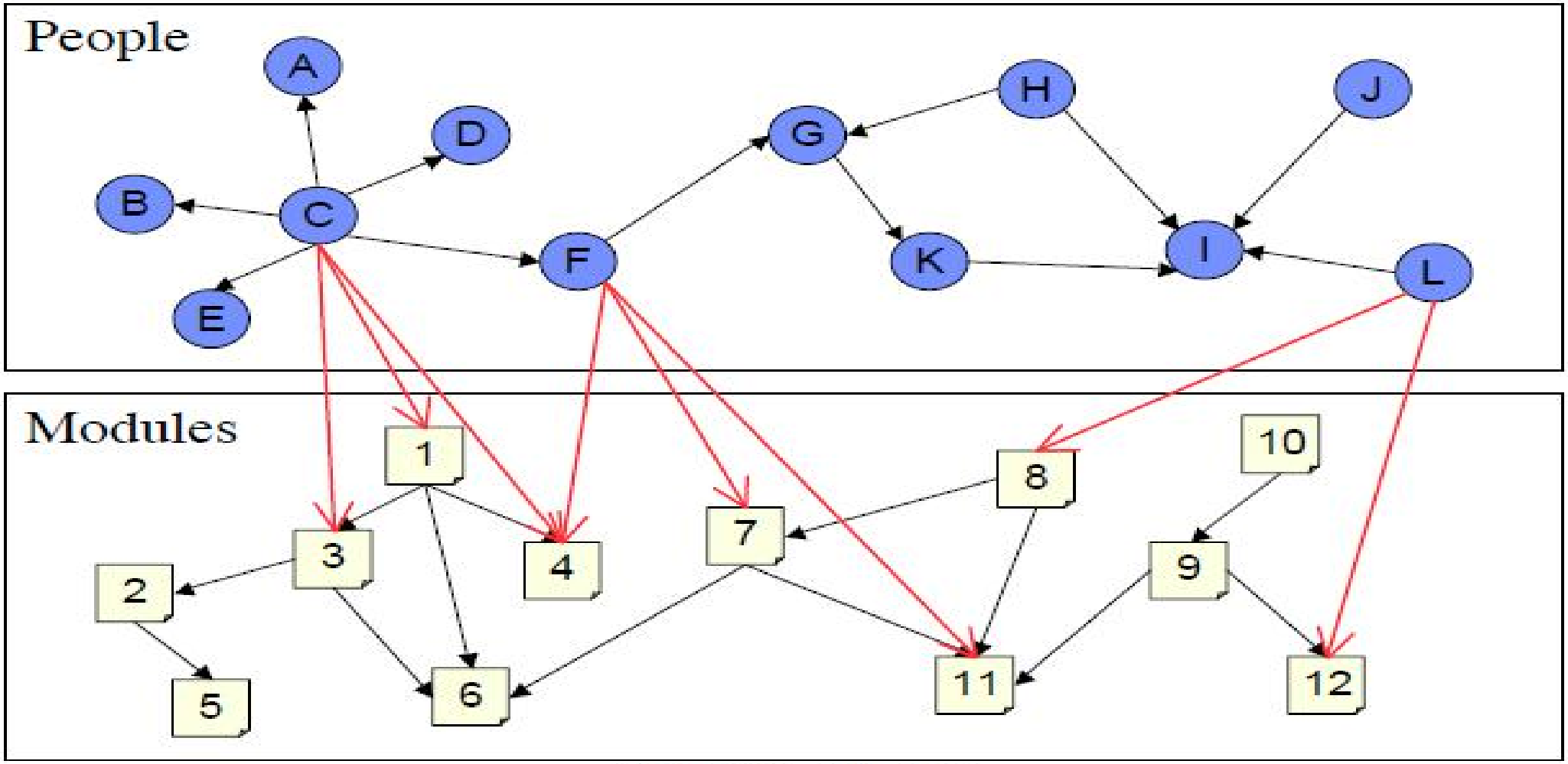
# 系统分解原则




See: Valetto, et al., 2007.

Socio-Technical Congruence

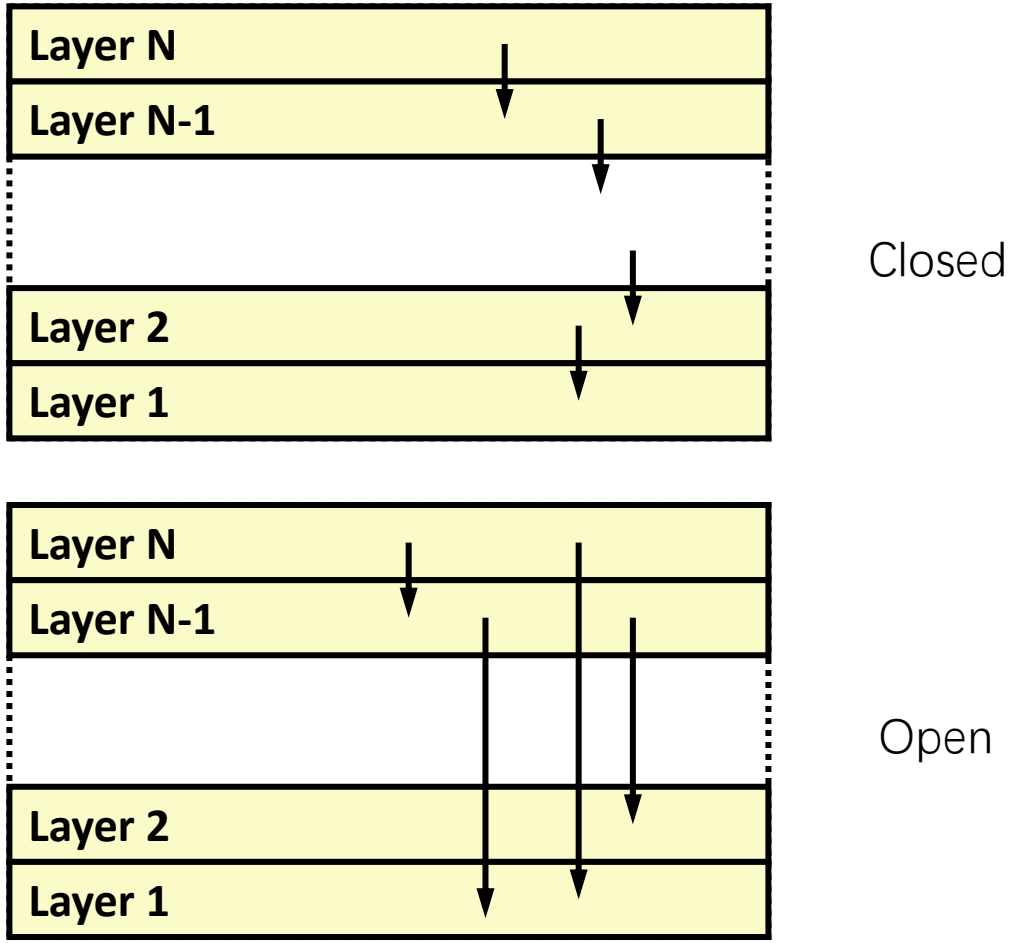
# 系统分解原则



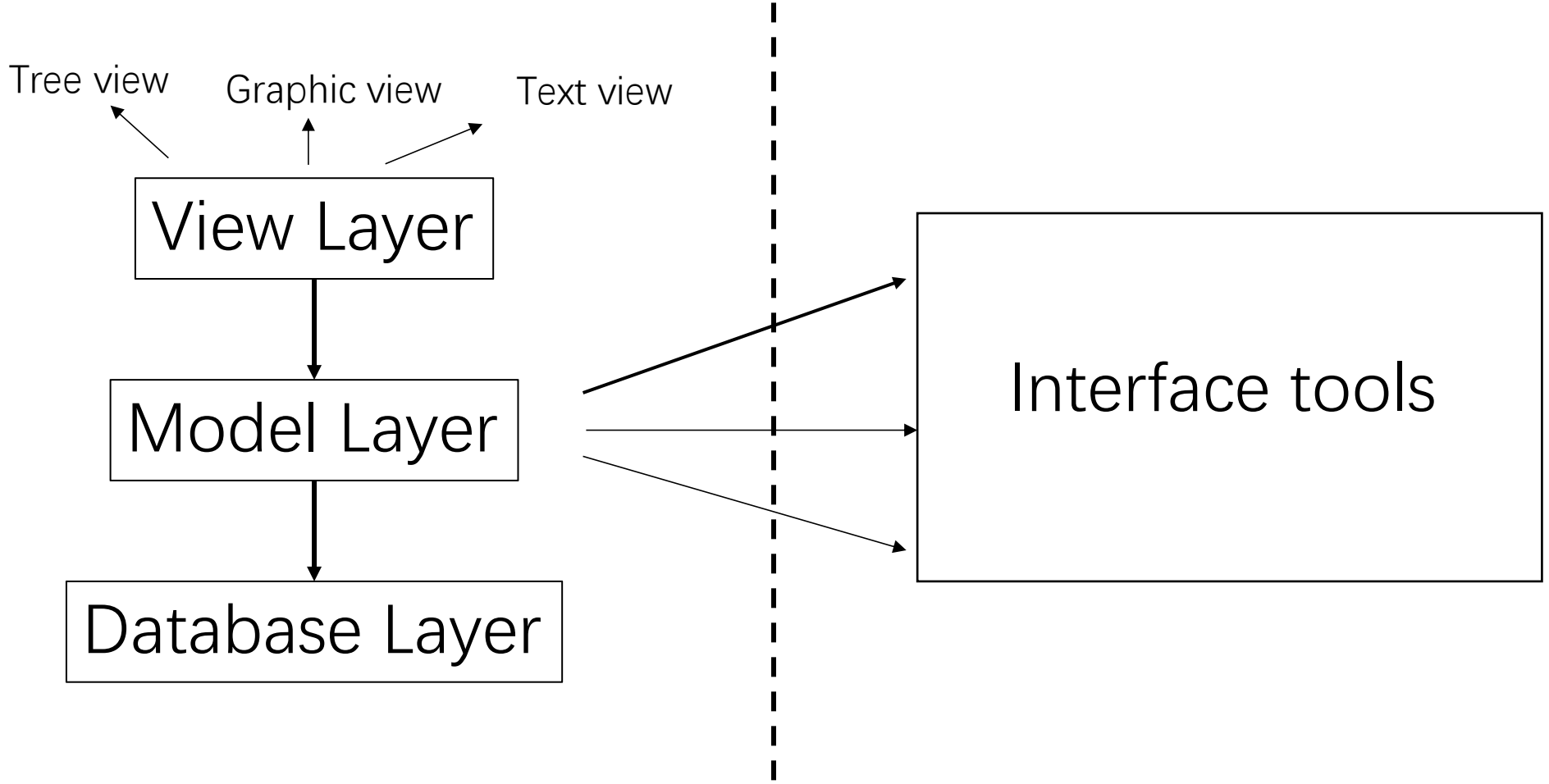

See: Valetto, et al., 2007.

# 层次化

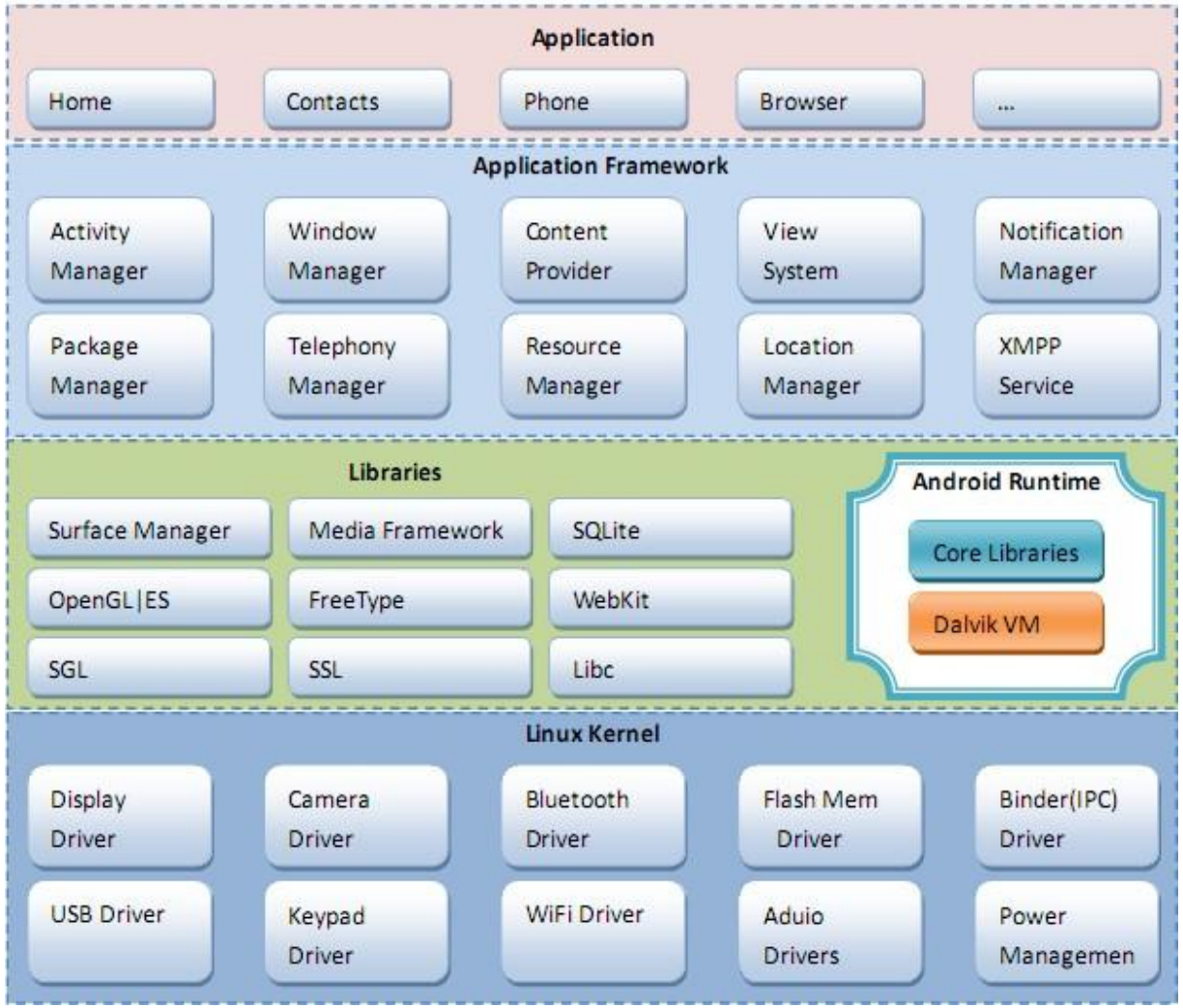
- 分层（Layering）
  - 每一层可以访问下层，不能访问上层
  - 封闭式结构：每一层只能访问与其相邻的下一层
  - 开放式结构：每一层还可以访问下面更低的层次
  - 层次数目不应超过 7±2 层



# 举例：画图软件


Tree view
Graphic view
Text view
View Layer
Model Layer
Database Layer
Interface tools

# 举例：安卓操作系统层次结构




应用层：运行在虚拟机上的Java应用程序。

应用框架层：支持第三方开发者之间的交互，使其能够通过抽象方式访问所开发的应用程序需要的关键资源。

系统运行库层：为开发者和类似终端设备拥有者提供需要的核心功能。

Linux内核层：提供启动和管理硬件以及Android应用程序的最基本的软件。

# 复用

- 复用（Reuse）是利用某些已开发的、对建立新系统有用的软件元素来生成新的软件系统，其好处在于提高生产效率，提高软件质量。
  - 源代码复用：对构件库中的源代码构件进行复用
  - 软件体系结构复用：对已有的软件体系结构进行复用
  - 框架复用：对特定领域中存在的一个公共体系结构及其构件进行复用
  - 设计模式：通过为对象协作提供思想和范例来强调方法的复用