第7章 设计工程

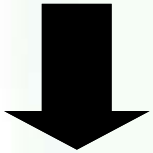
王美红

主要内容

- 软件设计
- 设计过程和设计质量
- 设计概念
- 设计模型
- 基于模式的软件设计

7.1 软件设计

- 软件设计的目标
  - 软件需求：解决“做什么”



  - 软件设计：解决“怎么做”

7.1 软件设计

- 软件设计的任务
  - 问题结构(软件需求)  软件结构
  - 从软件需求规格说明书出发，形成软件的具体设计方案。

软件设计的目标和任务

- 根据用信息域表示的软件需求，以及功能和性能需求，进行：
  - 数据设计
  - 系统结构设计
  - 过程设计
  - 接口设计

软件设计的目标和任务

- 数据设计将类模型转化为设计类的实现以及软件实现所需要的数据结构。
- 体系结构设计定义了软件的主要结构化元素之间的关系，可满足系统需求的体系结构和模式以及影响体系结构实现方式的约束。
- 接口设计描述了软件和协作系统之间，软件和使用人员之间是如何通信的。
- 过程设计/构件设计则是把结构成份转换成软件的过程性描述。在编码步骤，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件。

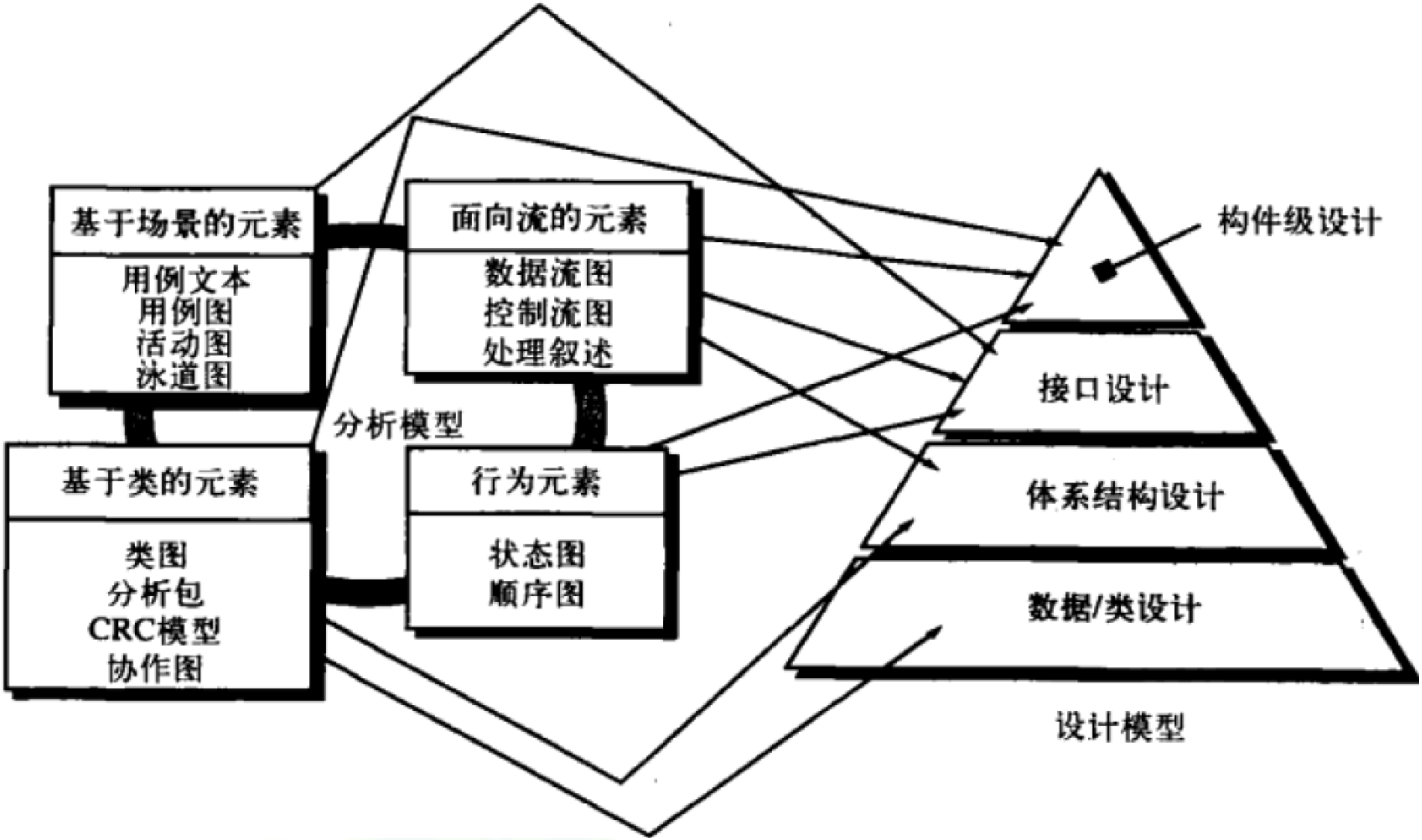
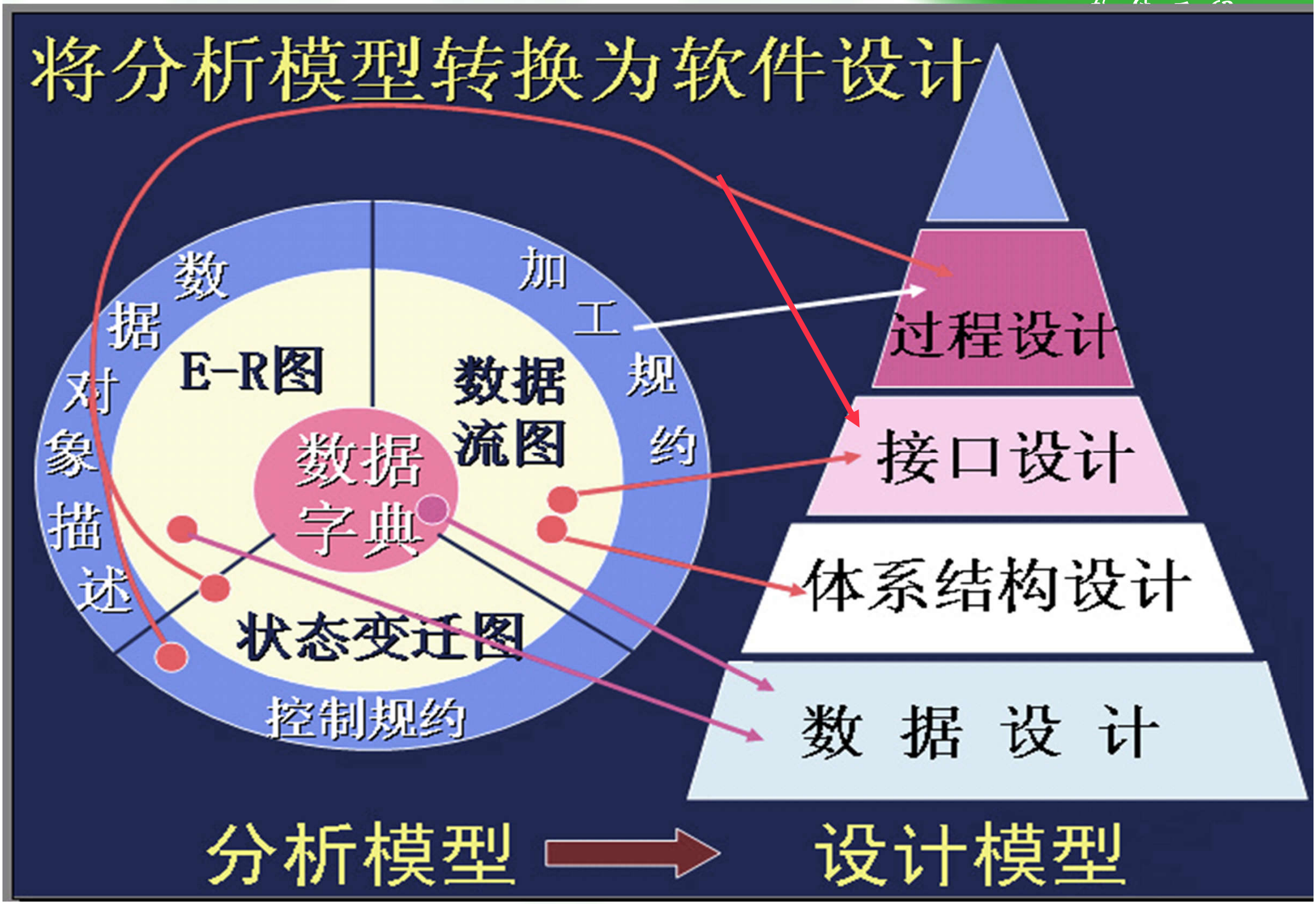
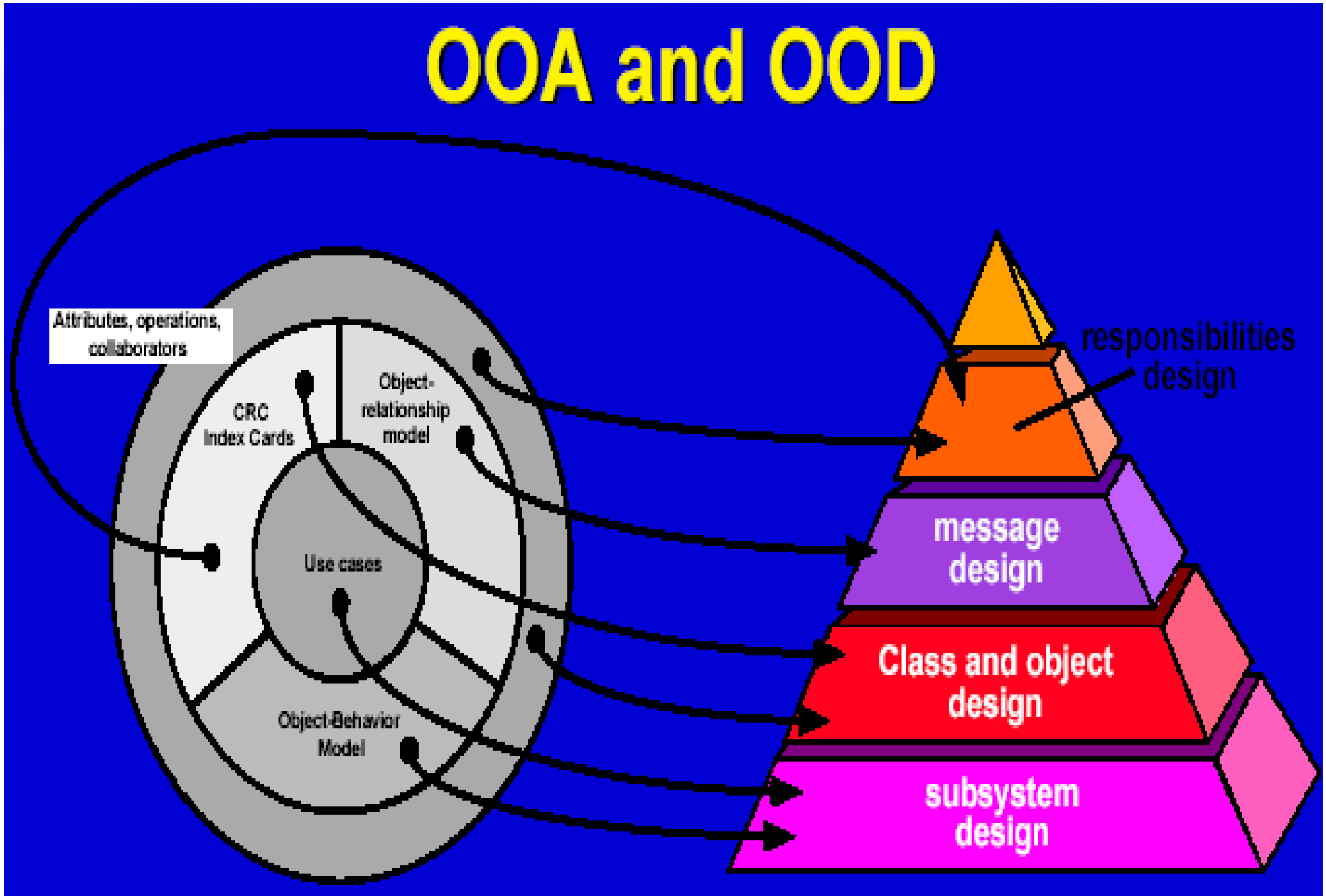


图 从需求模型到设计模型的转换

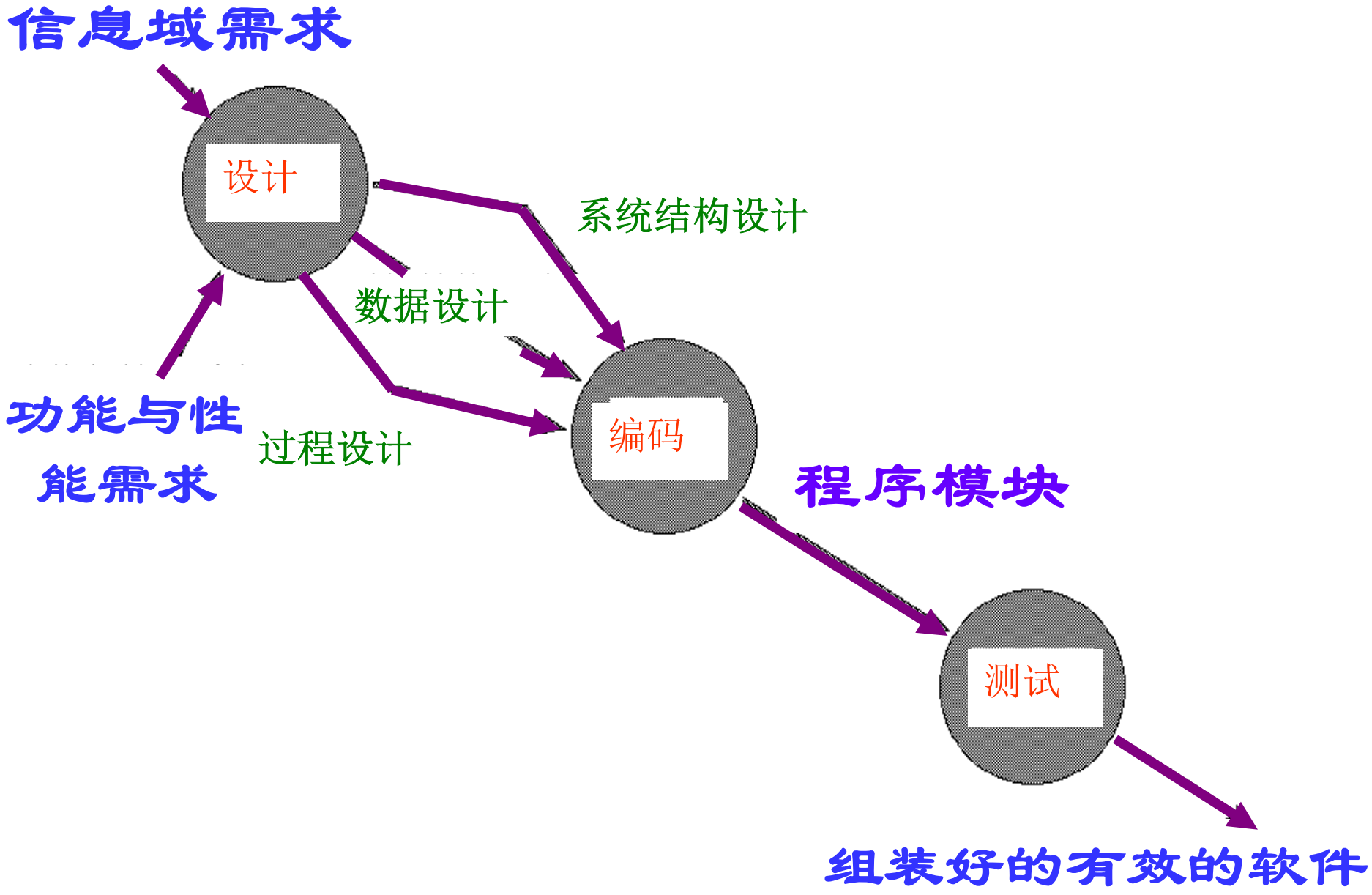

将分析模型转换为软件设计
数据对象描述
E-R图
加工规约
数据流图
数据字典
状态变迁图
控制规约
过程设计
接口设计
体系结构设计
数 据 设 计
分析模型
设计模型


OOA and OOD
Attributes, operations, collaborators
CRC Index Cards
Object-relationship model
Use cases
Object-Behavior Model
responsibilities design
message design
Class and object design
subsystem design

OOD在两个不同抽象级别上进行设计

- 系统设计(子系统设计):
  - 着重于构造一个完全的产品或系统所需的构件的布局。即软件体系结构.
- 对象设计（类设计）:
  - 强调个体对象的详细布局,及其相互交互的描述。

开发阶段的信息流

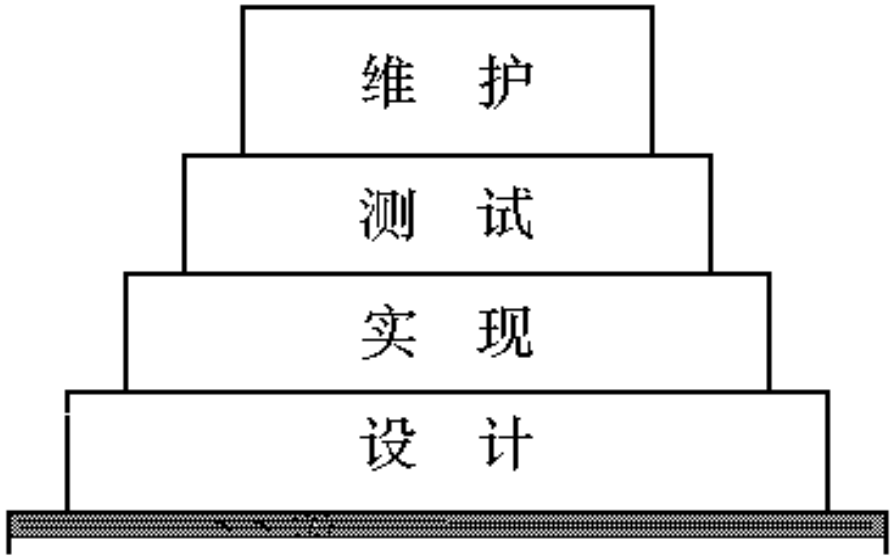


为什么软件设计很重要?

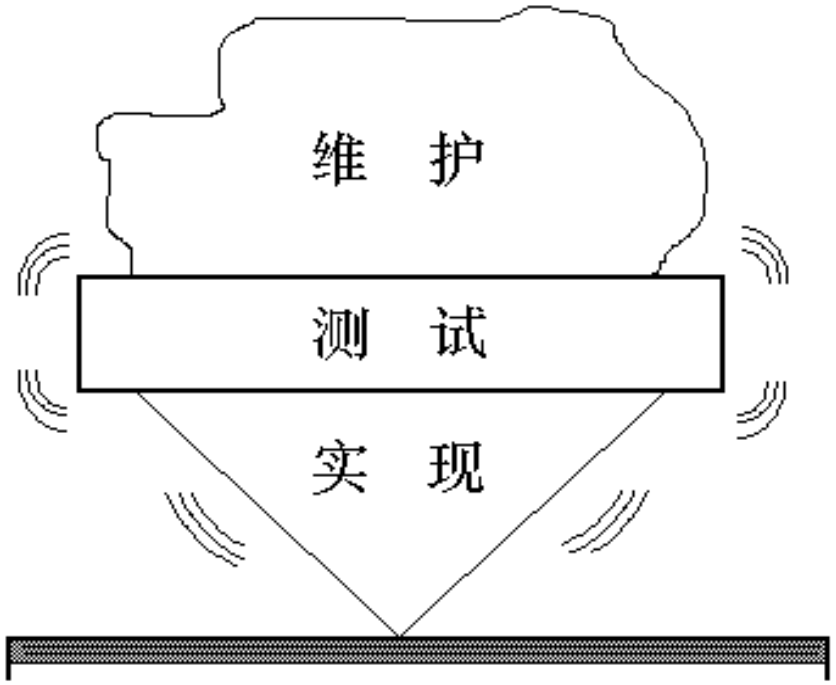
软件设计是开发阶段中最重要的步骤，它是软件开发过程中质量得以保证的关键步骤。设计提供了软件的表示，使得软件的质量评价成为可能。

同时，软件设计又是将用户要求准确地转化成为最终的软件产品的唯一途径。另一方面，软件设计是后续开发步骤及软件维护工作的基础。如果没有设计，只能建立一个不稳定的系统。

- 软件设计是后续开发步骤及软件维护工作的基础。如果没有设计，只能建立一个不稳定的系统结构



有软件设计



没有软件设计

只要出现一些小小的变动，就会使得软件垮掉，而且难于测试。

软件设计方法

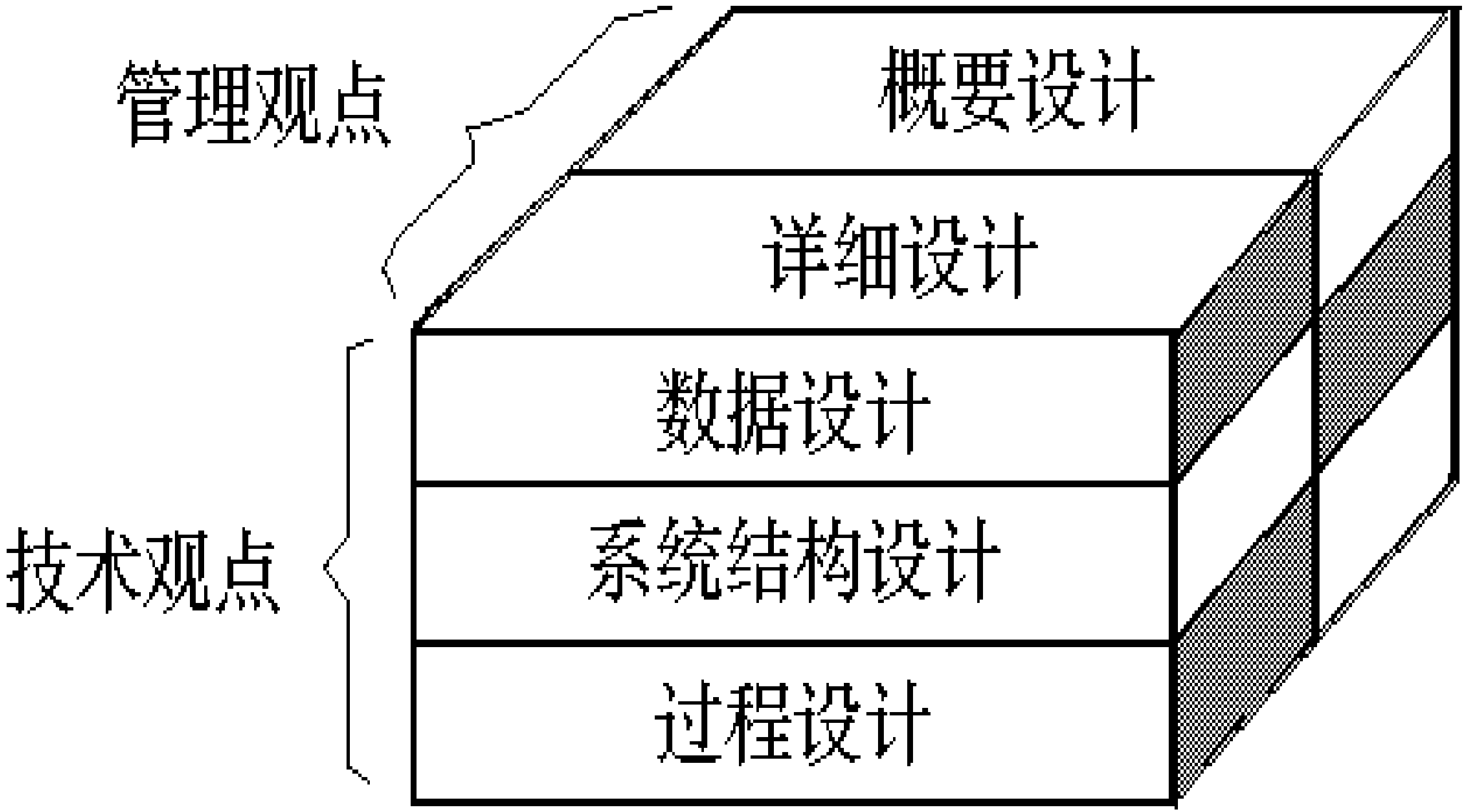
结构化设计方法(SD)

面向数据结构的设计方法(JSD方法)

面向对象的设计方法(OOD)

软件设计任务

- 从工程管理的角度来看，软件设计分两步完成：
  - 概要设计（总体设计）：
    - 将软件需求转化为数据结构和软件的系统结构。
  - 详细设计（过程设计，模块设计）：
    - 通过对结构表示进行细化，得到软件的详细的数据结构和算法。


管理观点
概要设计
详细设计
技术观点
数据设计
系统结构设计
过程设计

7.2 设计过程和设计质量

- 软件设计是一个迭代的过程
- 初始时，设计描述了软件的整体视图，是高抽象层次上的表达
- 迭代中，后续的精化导致更低抽象层次上的设计表示

7.2 设计过程和设计质量 （续）

- 评价良好设计的三个特征：
  1. 设计必须实现所有包含在分析模型中的明确需求，而且满足客户期望的所有隐含需求
  2. 对于那些生成代码的人和那些进行测试以及随后维护软件的人而言，设计必须是可读的，可理解的指南
  3. 设计必须提供软件的全貌，从实现的角度说明数据域、功能域和行为域。

7.2 设计过程和设计质量 （续）

- 优秀设计的技术标准：

1. 设计应展示出这样的一种结构

a） 已经使用可识别的体系结构风格或模式创建

b） 由展示出良好设计特征的构件构成；

c） 能够以演化的方式实现，从而便于实现和测试

2. 设计应该模块化

7.2 设计过程和设计质量 （续）

3. 设计应该包括数据、体系结构、接口和构件的清楚的表示
4. 设计应导出数据结构，这些数据结构适于要实现的类，并由可识别的数据模式提取
5. 设计应导出显示独立功能特征的构件

7.2 设计过程和设计质量 （续）

6. 设计应导出接口，这些接口降低了构件之间以及与外部环境连接的复杂性
7. 设计的导出应根据软件需求分析过程中获取的信息采用可重复使用的方法进行
8. 应使用能有效传达其意义的表示法来表达设计

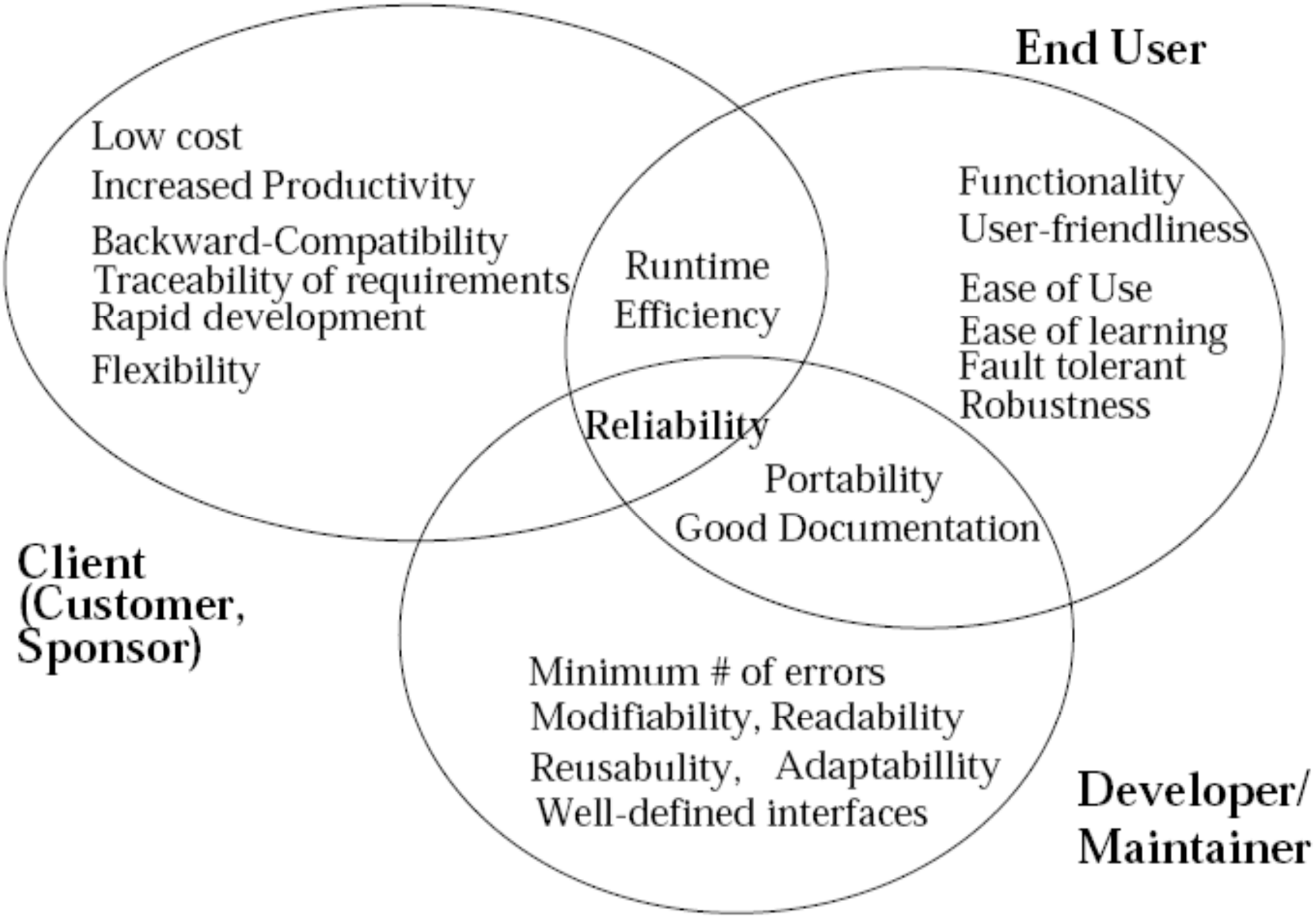
7.2 设计过程和设计质量 （续）

- 在设计阶段，通过开展一系列的正式技术评审（Formal Technical Review, FTR）来评估质量
  - 会议形式，多角色参与，阅读找问题。
  - 在20章介绍

7.2 设计过程和设计质量 （续）

- FURPS质量属性体现了所有软件设计的目标：
  - 功能性(Functionality)：
    - 通过评估程序的特征集和能力、所提交功能的通用性以及整个系统的安全性来评估
  - 易用性（Usability)
    - 通过考虑人员因素、整体美感、一致性和文档来评估。
  - 可靠性（Reliability）
    - 通过测量故障的频率和严重性、输出结果的精确性、平均故障时间、故障恢复能力和程序的可预见性来评估
  - 性能（Performance)
    - 通过处理速度、响应时间、资源消耗、吞吐量和效率来度量
  - 可支持性(Supportability)
    - 综合了可扩展性、可适应性和可用性。此外，还包括可测试性、兼容性、可配置性、系统安装的简易性和问题定位的容易性。

Relationship Between Design Goals



典型设计权衡

- 功能性与可用性
  - 客户与开发商
- 成本与鲁棒性
  - 客户与用户
- 效率与便携性
  - 快速开发与功能
- 成本与可重用性
  - 向后兼容性与可读性

7.2 设计过程和设计质量 （续）

- 通常设计任务集（面向对象）：
  1. 检查信息域模型，并为数据对象及其属性设计恰当的数据结构
  2. 使用分析模型，选择一个适于软件的体系结构类型

通常设计任务集

3. 将分析模型分割成若干个设计子系统，并在体系结构内分配这些子系统。要确定每个子系统是功能内聚的。

   设计子系统接口

   为每个子系统分配分析类或功能

通常设计任务集

4. 创建一系列的设计类或构件。将每个分析类说明转化为设计类

据设计标准检查每个设计类

定义与每个设计类相关的方法和消息

评估设计类或子系统并为这些类或子系统选择设计模式

评审设计类，并在需要时修改

通常设计任务集

5. 设计外部系统或设备所需要的所有接口

6. 设计用户接口

   评审任务分析的结构

   基于用户场景详细说明活动序列

   创建接口的行为模型

   定义接口对象、控制机制

   评审接口设计，并在需要时修改

通常设计任务集

7. 进行构件级设计

   在相对较低的抽象层次上详细地说明所有算法

   精化每个构件的接口

   定义构件级的数据结构

   评审每个构件并修改所有已发现的错误

8. 开发部署模型

7.3 设计概念

- 抽象
  - 软件系统进行模块设计时，可有不同的抽象层次。
  - 在最高的抽象层次上，可以使用问题所处环境的语言概括地描述问题的解法。
  - 在较低的抽象层次上，则采用过程化的方法。
  - 数据抽象：描述数据对象的冠名数据集合，如“门”
  - 过程抽象：具有明确和有限功能的指令序列，如“开”门

(1) 过程的抽象

在软件工程中，从系统定义到实现，每进展一步都可以看做是对软件解决方法的抽象化过程的一次细化。

- 在软件需求分析阶段，用“问题所处环境的、为大家所熟悉的术语”来描述软件的解决方法。
- 在从概要设计到详细设计的过程中，抽象化的层次逐次降低。产生源程序时到达最低抽象层次。

例：开发一个CAD软件的三层抽象

抽象层次Ⅰ. 用问题所处环境的术语来描述这个软件：

该软件包括一个计算机绘图界面，向绘图员显示图形，以及一个数字化仪界面，用以代替绘图板和丁字尺。所有直线、折线、矩形、圆及曲线的描画、所有的几何计算、所有的剖面图和辅助视图都可以用这个CAD软件实现……。

抽象层次Ⅱ. 任务需求的描述。

CAD SOFTWARE TASKS

user interaction task;

2-D drawing creation task;

graphics display task;

drawing file management task;

end.

在这个抽象层次上，未给出“怎样做”的信息，不能直接实现。

抽象层次III. 程序过程表示。以2-D（二维）绘图生成任务为例：

PROCEDURE：2-D drawing creation

REPEAT UNTIL (drawing creation task terminates)

DO WHILE (digitizer〔数字转换器〕interaction occurs)

digitizer interface task;

DETERMINE drawing request CASE;

line：line drawing task;

rectangle：rectangle drawing task;

circle：circle drawing task;

……

(2)数据抽象

在不同层次上描述数据对象的细节，定义与该数据对象相关的操作。

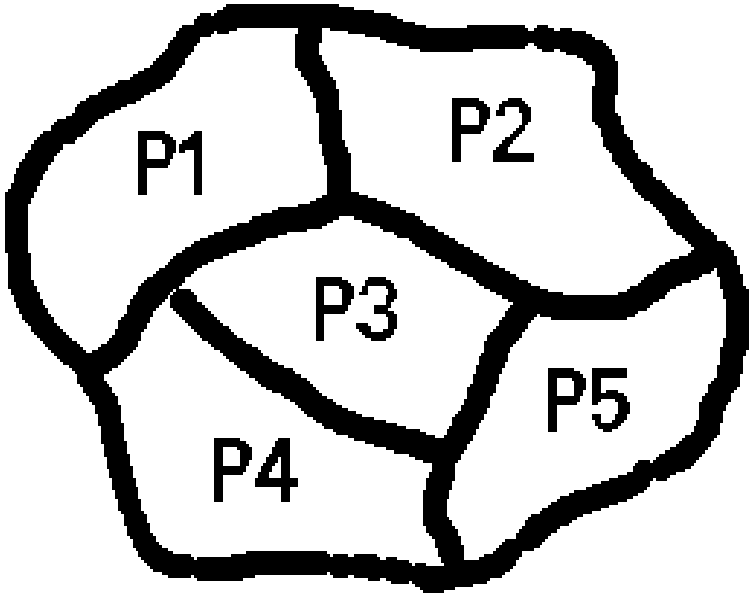
例如，在CAD软件中，定义一个叫做drawing的数据对象。可将drawing规定为一个抽象数据类型，定义它的内部细节为:

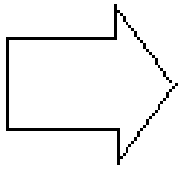
```
TYPE drawing IS STRUCTURE

  DEFIND
      number IS STRING LENGTH(12);
      geometry DEFIND ……

      notes IS STRING LENGTH(256);
      BOM   DEFIND ……

END drawing TYPE;
```

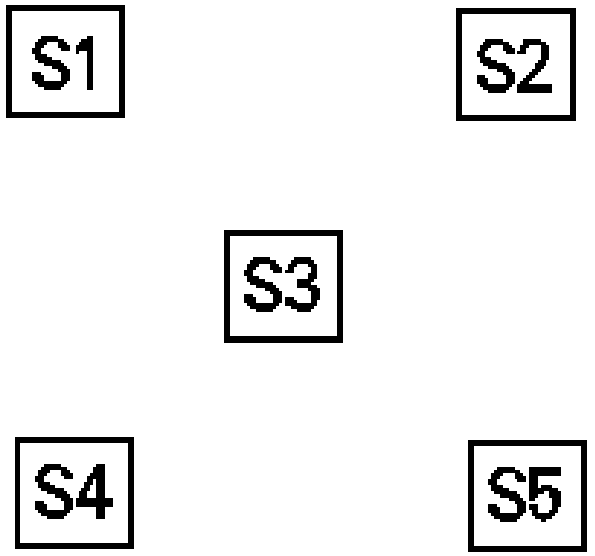
7.3 设计概念（续）

- 体系结构
  - 软件的整体结构和这种结构为系统提供概念上的完整性的方式
  - 程序各构件（模块）的结构或组织、这些构件交互的方式以及这些构件所使用数据的结构。
  - 软件结构包括两部分：程序结构和数据结构
  - 软件设计的目标之一是导出系统体系结示意图，其作为一个框架，将指导更详细的设计活动。

一．软件体系结构





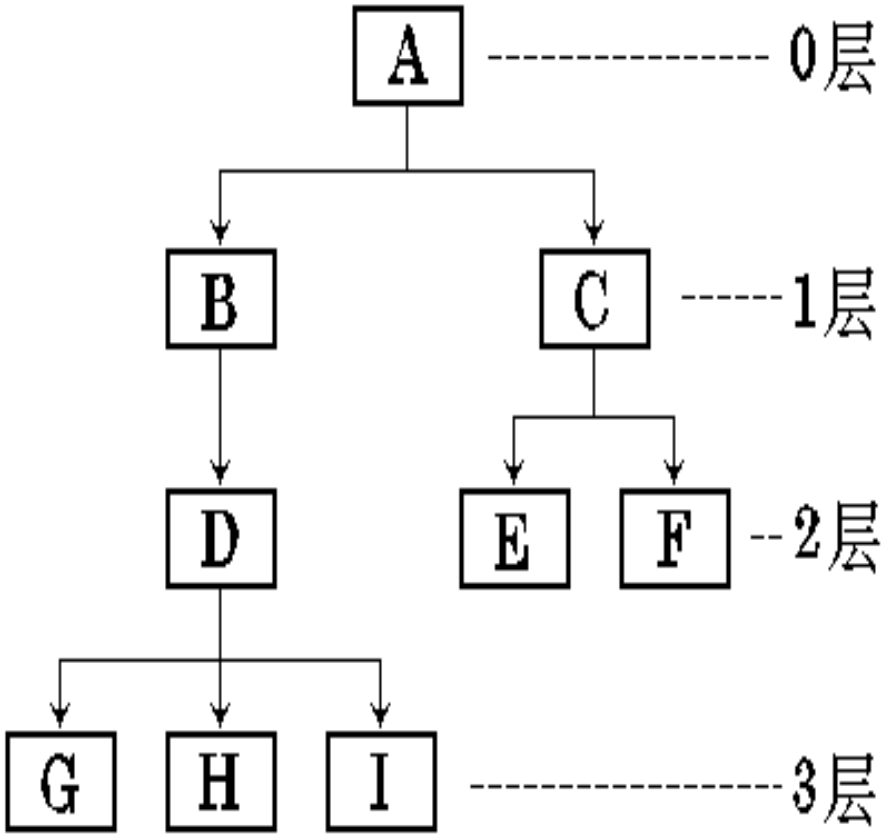


需要通过软件解决的"问题"

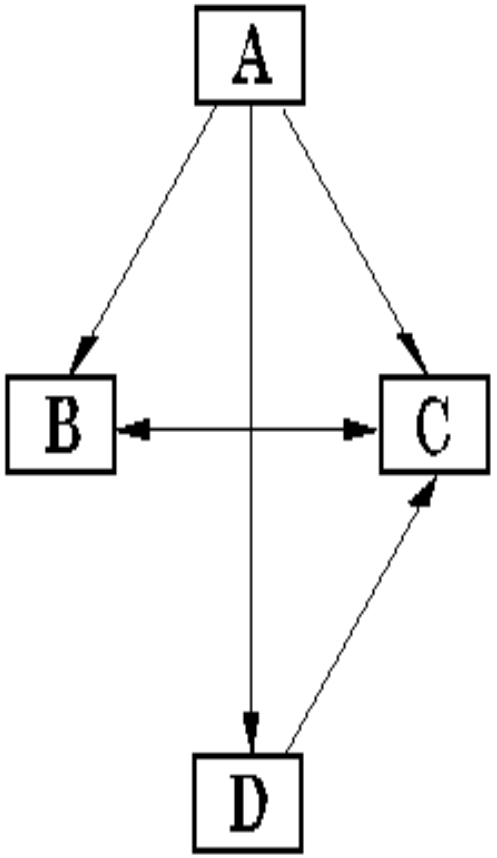
软件的"解决方案"

二. 程序结构

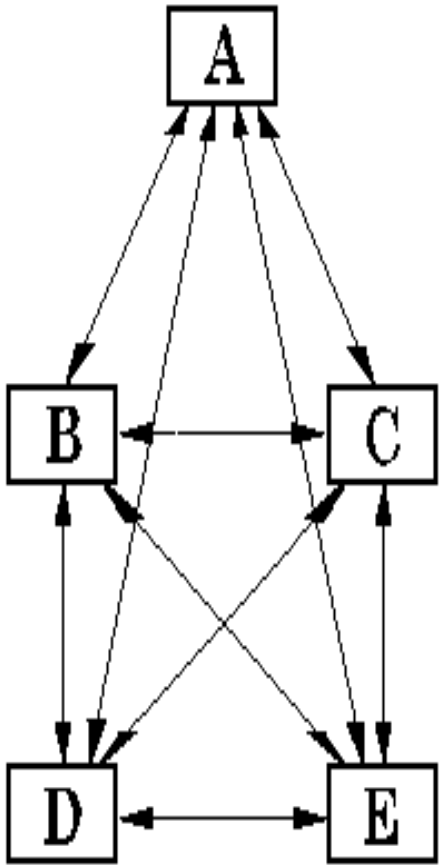
程序结构（控制层次）表明了程序各个部件(模块)的组织情况。



(a) 树状结构

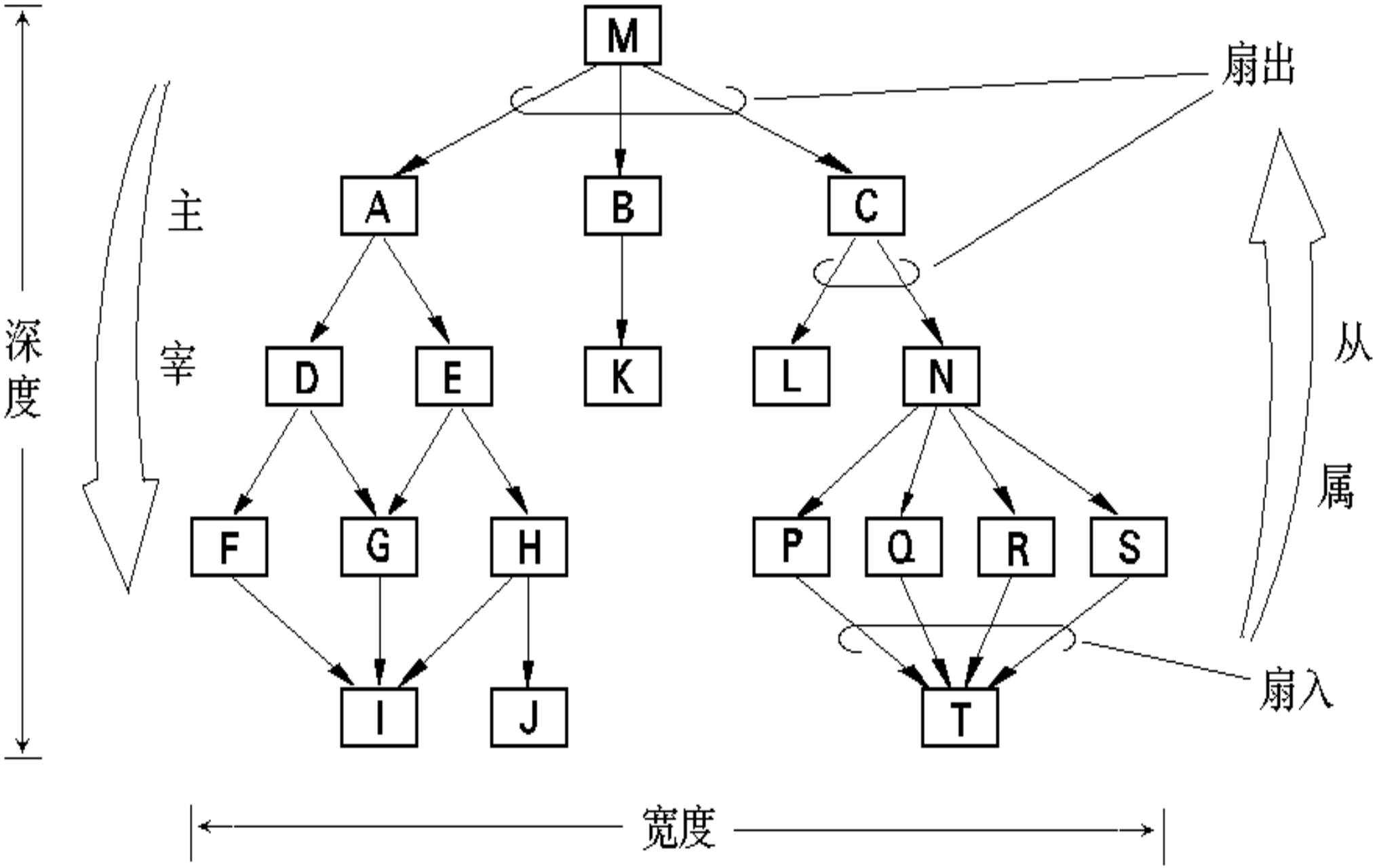


(b) 网状结构



(c) 网状结构

程序的结构图例子

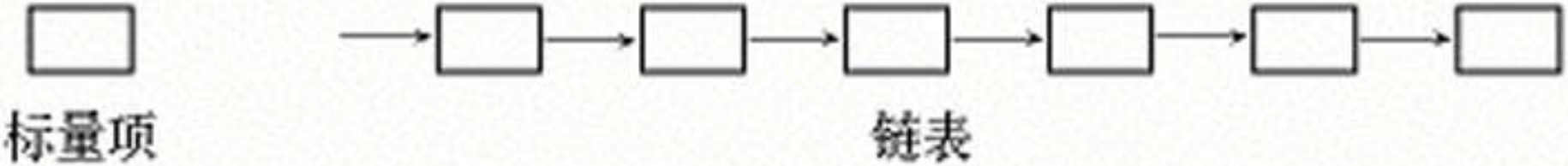


三. 数据结构

数据结构是数据的各个元素之间的逻辑关系的一种表示。数据结构设计应确定数据的组织、存取方式、相关程度以及信息的不同处理方法。

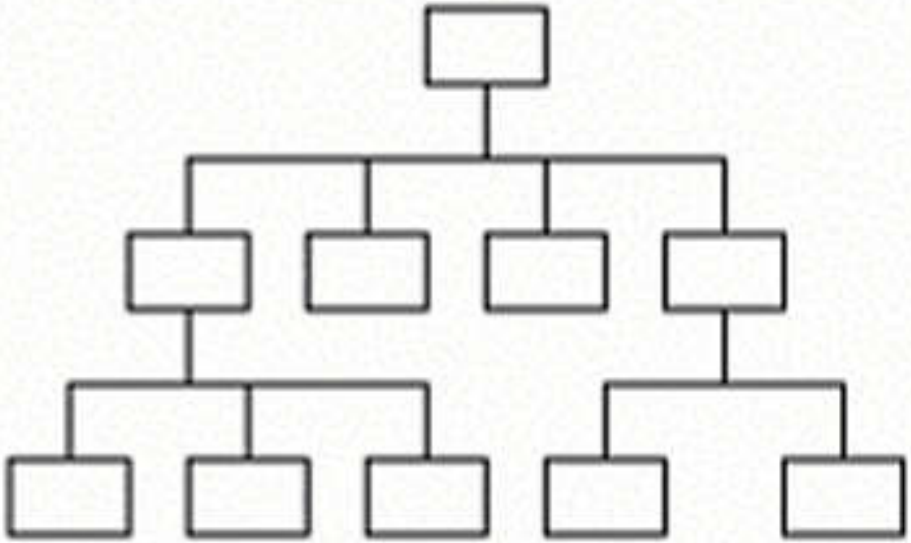
数据结构的组织方法和复杂程度可以灵活多样，但典型的数据结构种类是有限的，它们是构成一些更复杂结构的基本构件块。

三. 数据结构

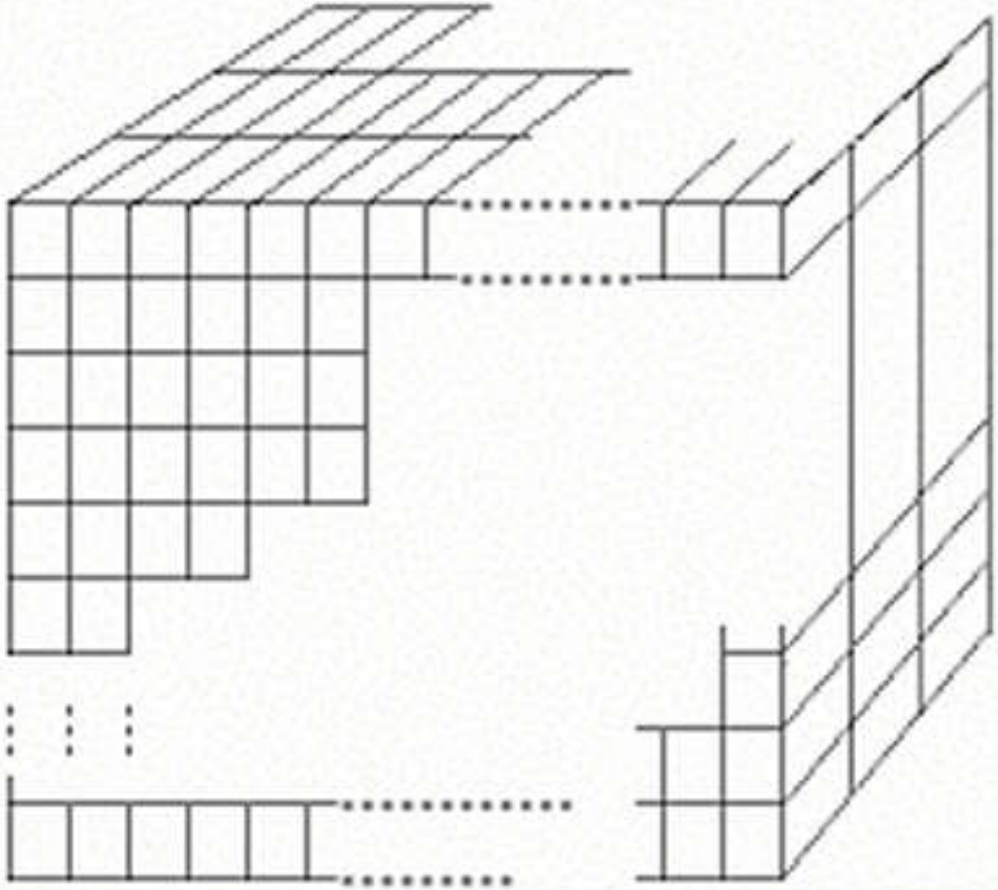


标量项　　链表

顺序向量

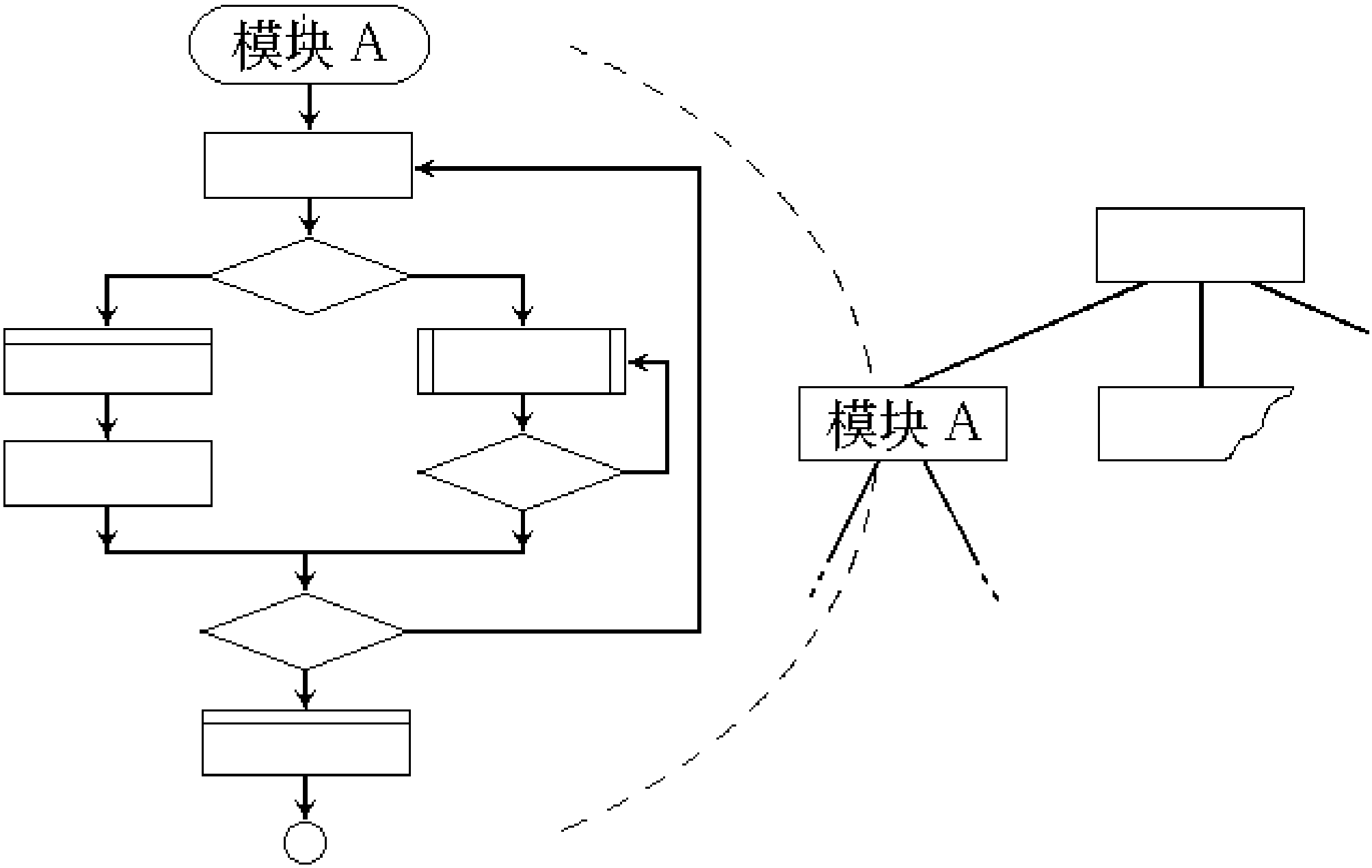


分层树



n维空间

四．软件过程



四．软件过程

软件过程则着重描述各个模块的处理细节。软件过程必须提供精确的处理说明，包括事件的顺序、正确的判定点、重复的操作等等。

五. 自顶向下，逐步细化

将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。

7.3 设计概念（续）

- 模式
  - 设计模式描述了在某个特定场景与可能影响模式应用和使用方式的“影响力”中解决某个特定的设计问题的设计结构。

7.3 设计概念（续）

- 关注点分离
  - 任何复杂问题如果被分解为可以独立解决和（或）优化的若干块，该复杂问题能够更容易被处理。
  - 一个关注点是一个特征或行为，被指定为软件需求模型的一部分。

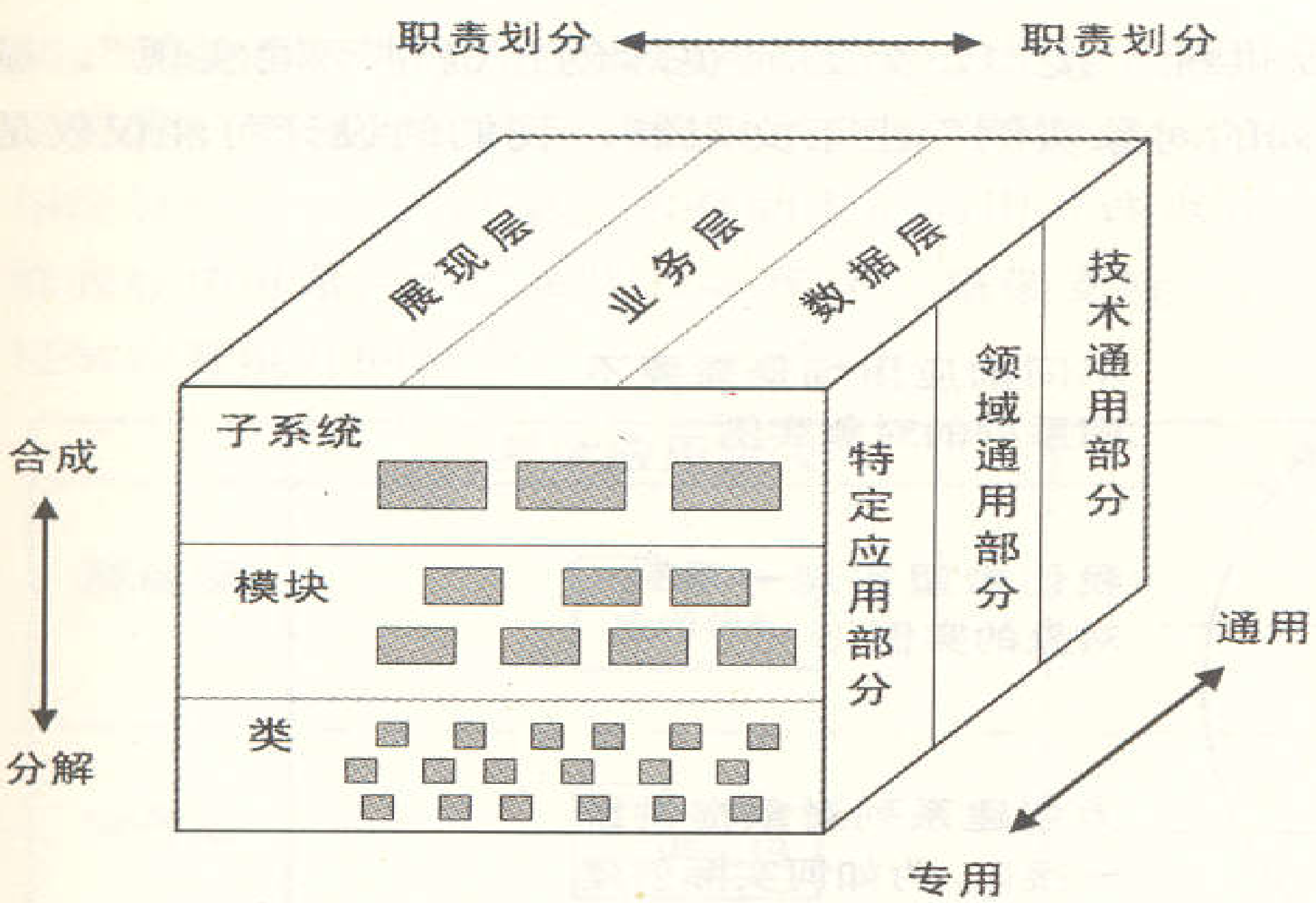
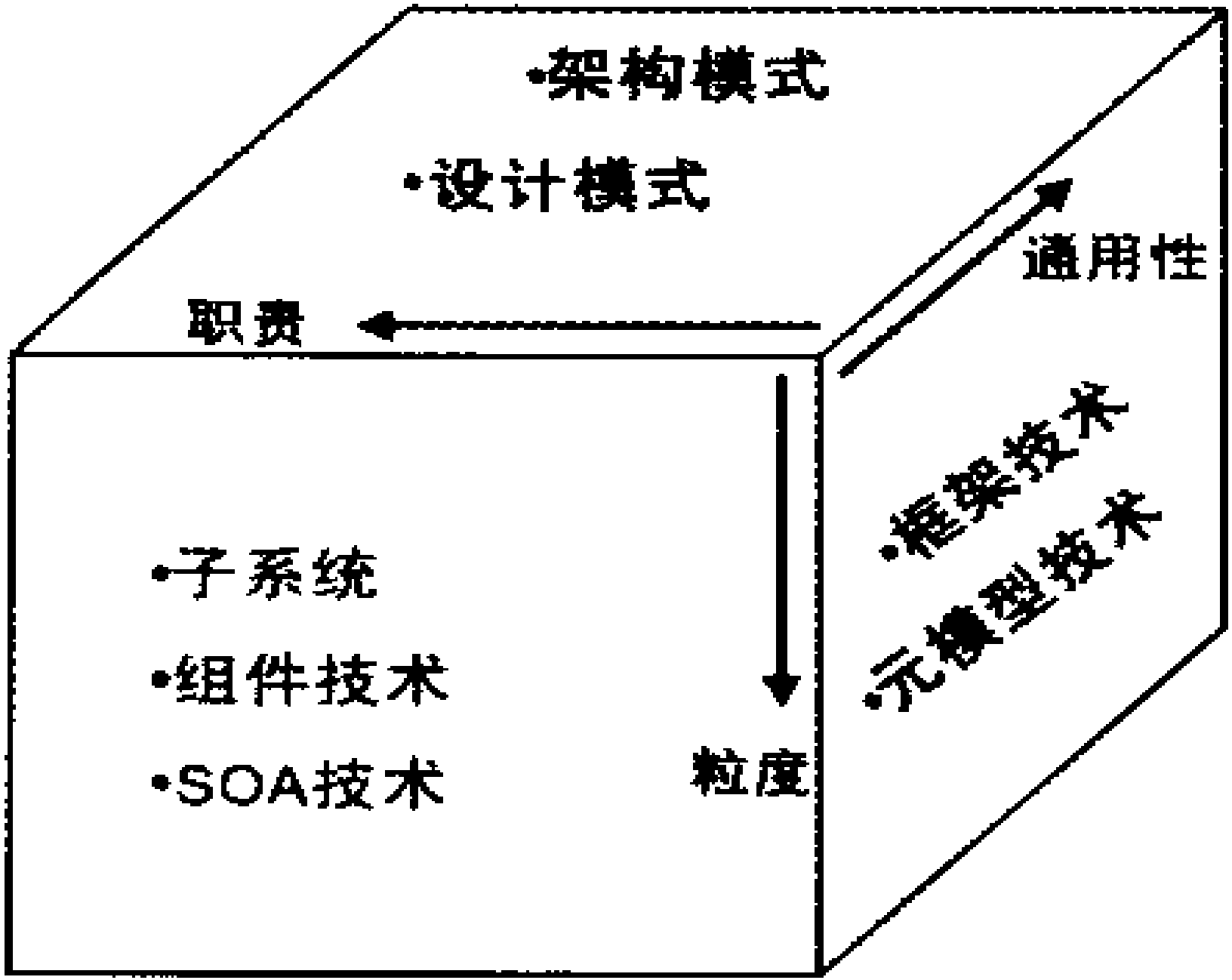


图 2-1　架构设计关注点分离原理

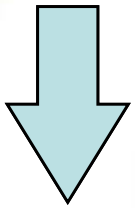

•架构模式
•设计模式
职责
通用性
•子系统
•组件技术
•SOA技术
粒度
•框架技术
•元模型技术

7.3 设计概念（续）

- 模块化
  - 软件被划分为可以独立命名的、可寻址的构件，有时被称为模块
  - 两个问题综合时的理解复杂度通常要大于每个问题各自的理解复杂度之和
  - “分而治之”，将一个策略问题分解成可以管理的若干块，这样更容易解决问题。

模块化是好的软件设计的一个基本准则

高层模块 —— 从整体上把握问题, 隐蔽细节



分解

复杂问题 → 较小问题

分解

可减小解题所需的总的工作

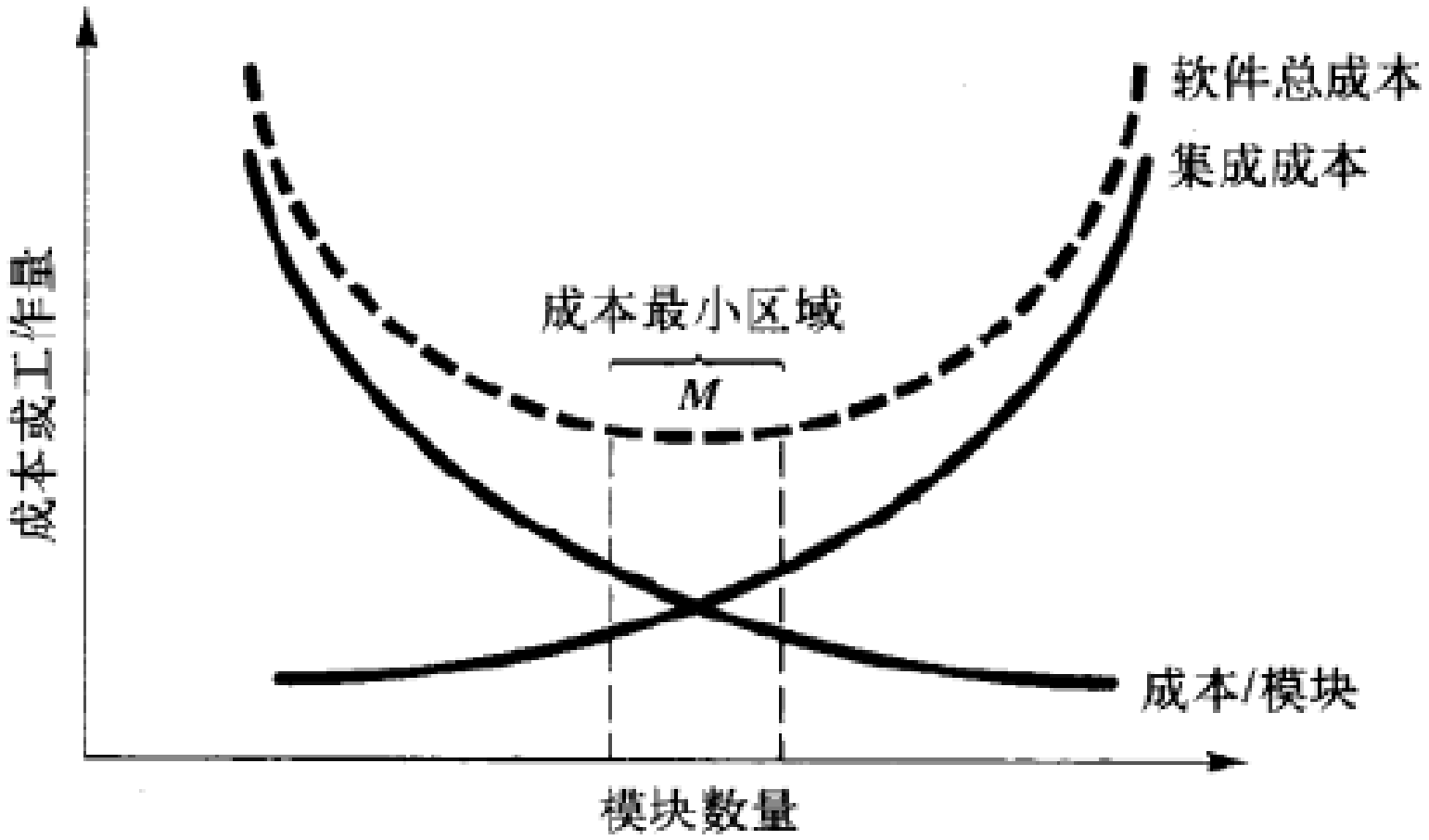


图　　模块化和软件成本

7.3 设计概念（续）

- 信息隐蔽
  - 每个模块对其他所有模块都隐蔽自己的设计策略
  - 隐蔽意味着通过定义一系列独立的模块可以得到有效的模块化，模块间只交流必须信息
  - 为修改提供很大的益处

7.3 设计概念（续）

- 功能独立
  - 模块化、抽象概念和信息隐蔽的直接结果
  - 通过开发具有“专一”功能和“避免”与其他模块过多交互的模块
  - 具有独立模块的功能更易开发

功能独立

- 模块的独立程度可以由两个定性标准来度量：
  - 内聚——模块之间的互相连接的紧密程度的度量
  - 耦合——模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量

模块间的耦合

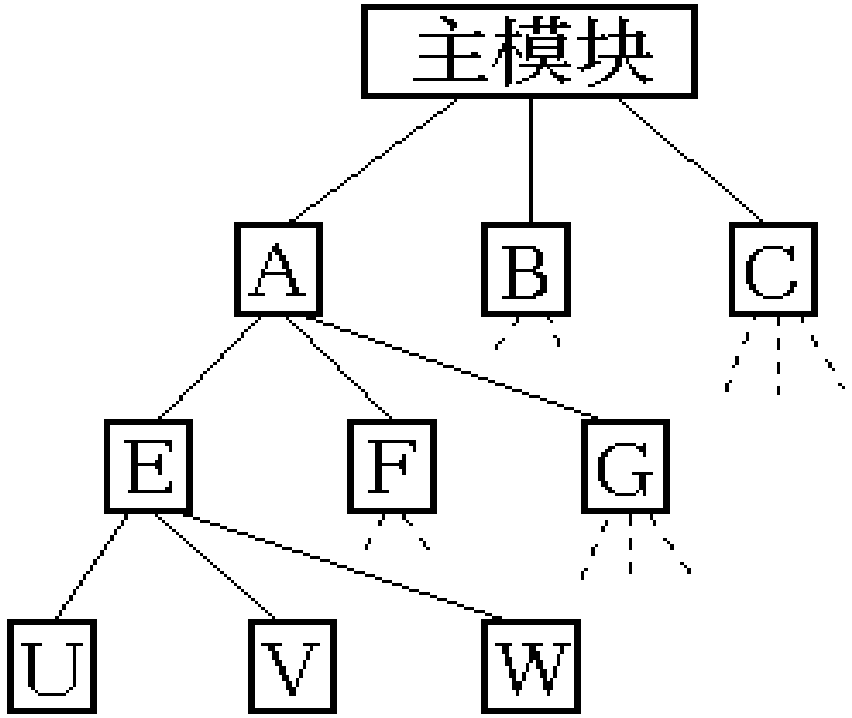
低 ——— 耦合性 ———→ 高

非直接耦合	数据耦合	标记耦合	控制耦合	外部耦合	公共耦合	内容耦合

强 ←——— 模块独立性 ——— 弱

非直接耦合(Nondirect Coupling)

- 两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。



非直接耦合的模块独立性最强。

数据耦合（Data Coupling）

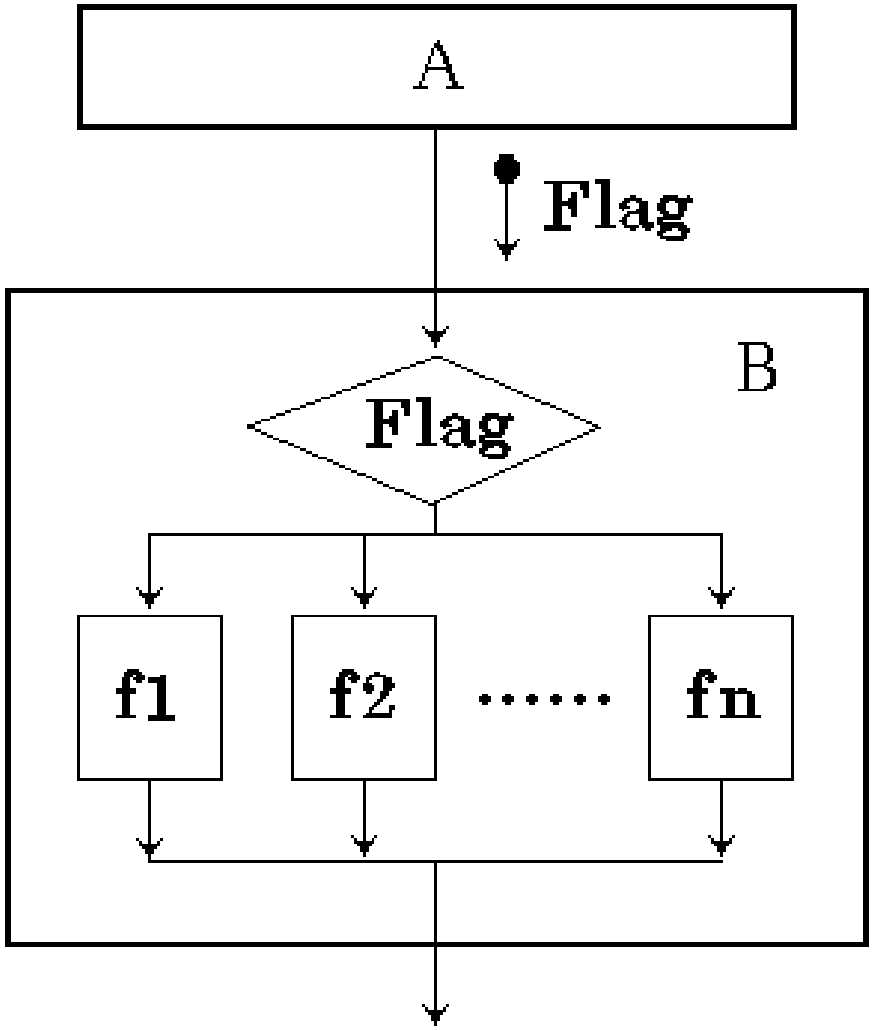
- 一个模块访问另一个模块时，彼此之间是通过简单数据参数（不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的。

标记耦合（Stamp Coupling）

- 一组模块通过参数表传递记录信息，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量

控制耦合（Control Coupling）

- 如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。

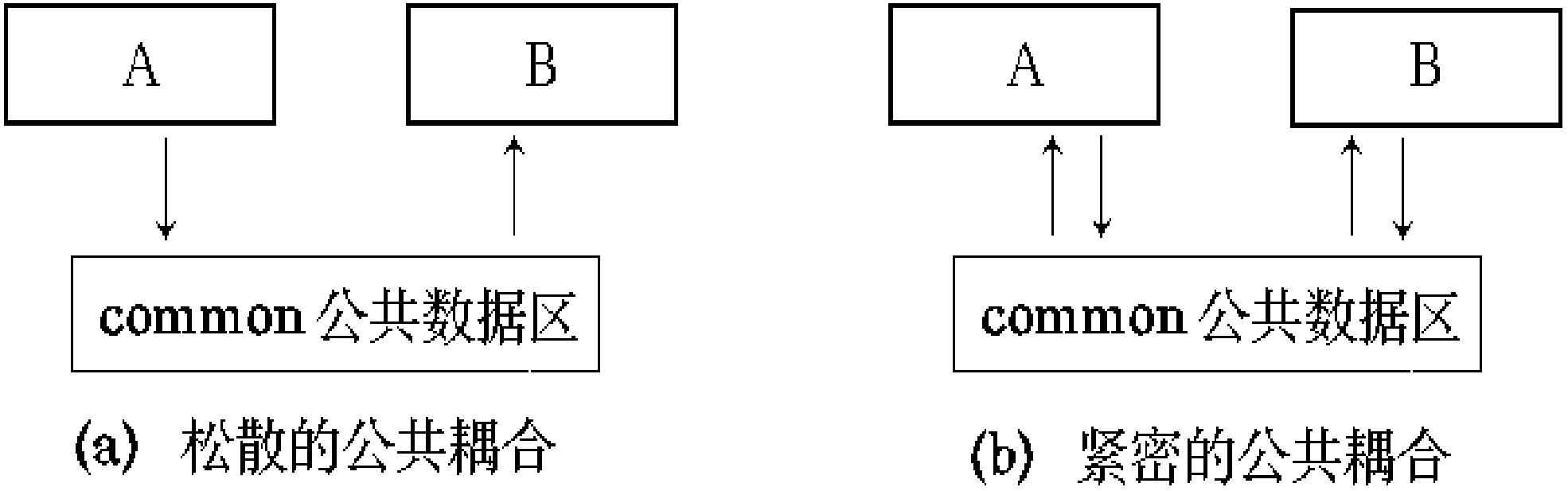


外部耦合（External Coupling）

- 一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。

公共耦合（Common Coupling）

- 若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的

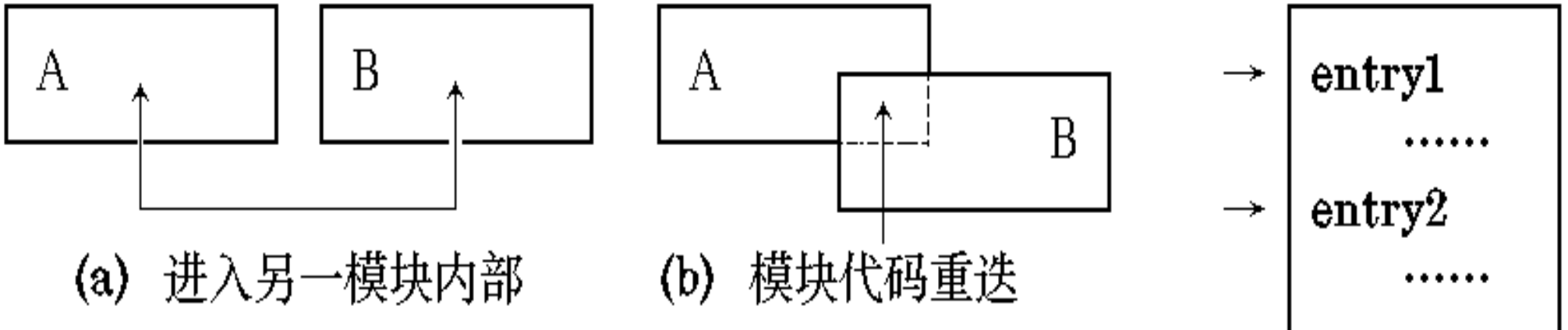


(a) 松散的公共耦合　　(b) 紧密的公共耦合

公共耦合的复杂程度随耦合模块的个数增加而显著增加。

内容耦合（Content Coupling）

- 如果发生下列情形，两个模块之间就发生了内容耦合
  1. 一个模块直接访问另一个模块的内部数据；
  2. 一个模块不通过正常入口转到另一模块内部；
  3. 两个模块有一部分程序代码重迭(只可能出现在汇编语言中)；
  4. 一个模块有多个入口。



(a) 进入另一模块内部　　(b) 模块代码重迭

实际上，模块之间是混合式的耦合

原则

尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。

模块内聚

高 ← 内聚性 → 低

功能内聚	信息内聚	通信内聚	过程内聚	时间内聚	逻辑内聚	巧合内聚

强 ← 模块独立性 → 弱

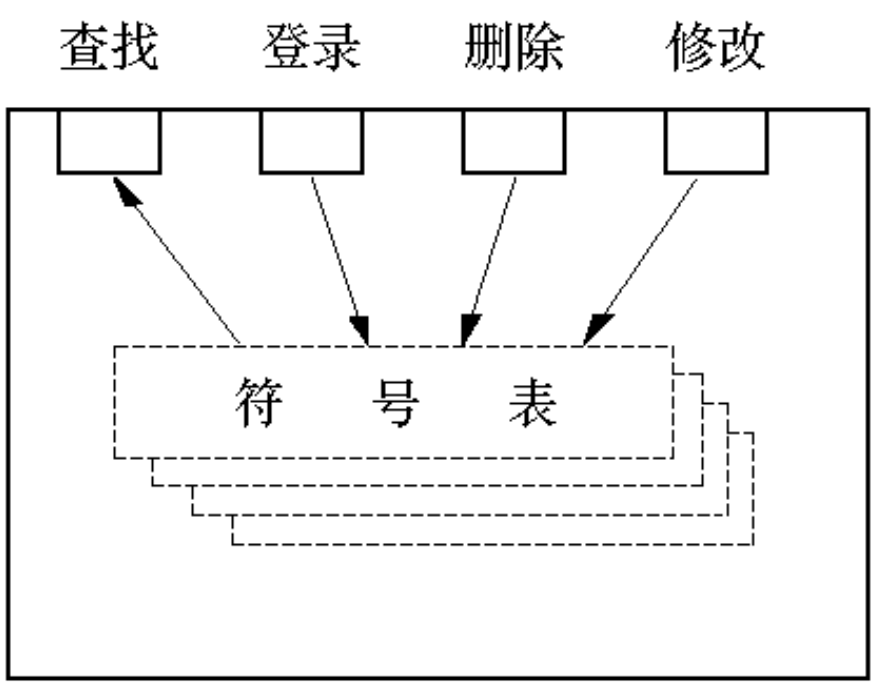
功能单一　　功能分散

功能内聚（Functional Cohesion）

- 一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的。则称该模块为功能内聚模块。

信息内聚(Informational Cohesion)

- 这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表），因此，它是一个信息内聚的模块。

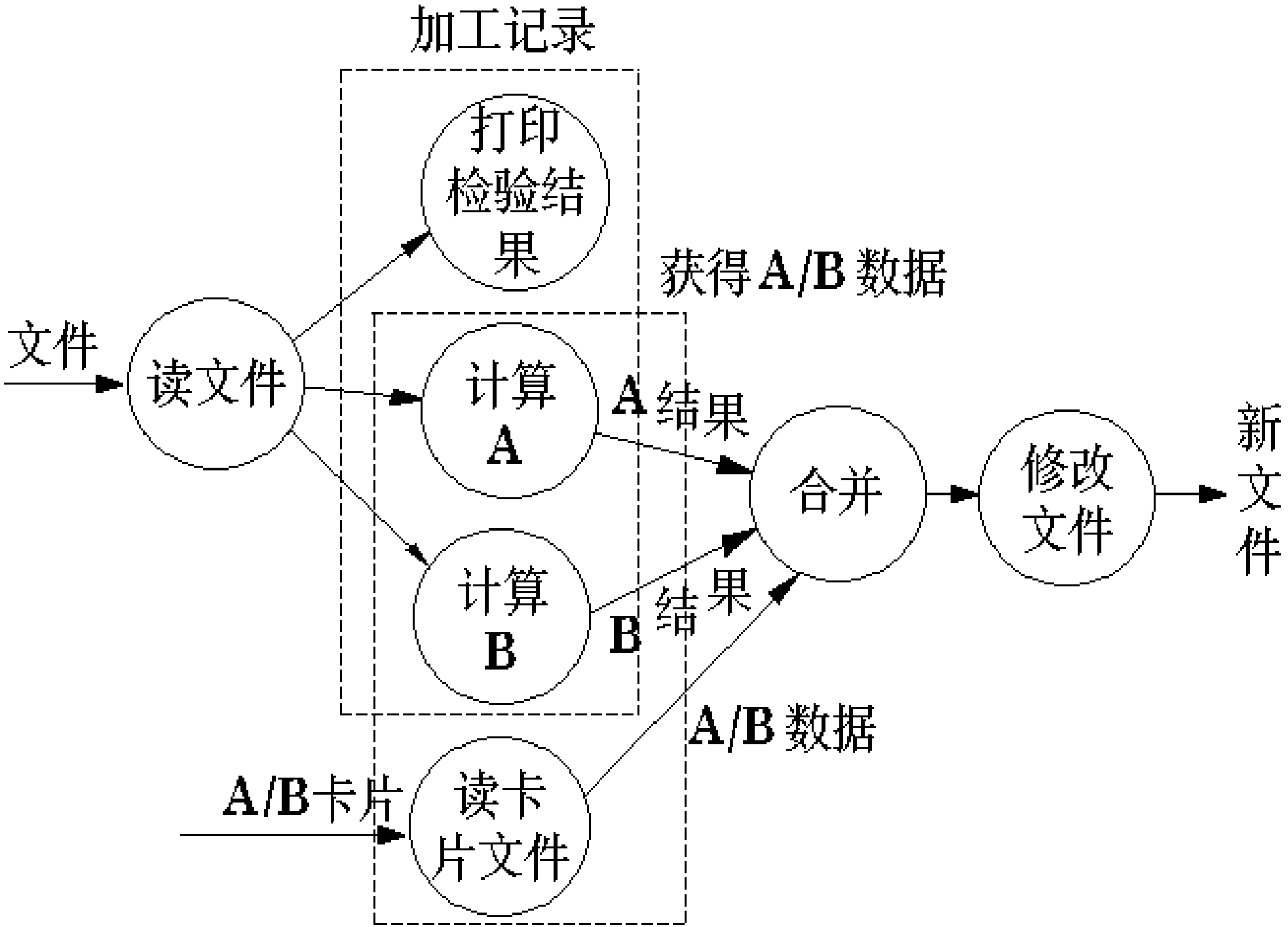


信息内聚

- 信息内聚模块可以看成是多个功能内聚模块的组合，并且达到信息的隐蔽。即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。

通信内聚 (Communication Cohesion)

- 如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称之为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。


加工记录
打印检验结果
获得A/B数据
文件
读文件
计算A
A结果
合并
修改文件
新文件
计算B
B结果
A/B数据
A/B卡片
读卡片文件

过程内聚（Procedural Cohesion）

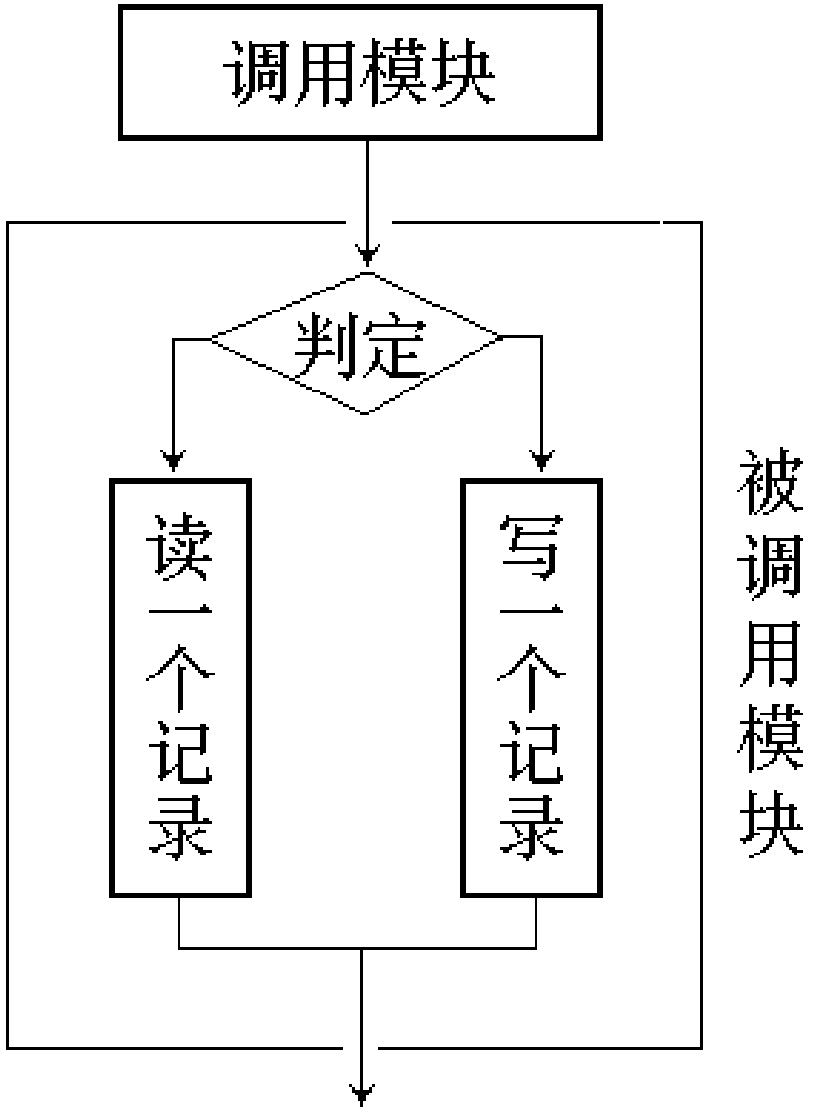
- 使用流程图做为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。

时间内聚 (Classical Cohesion)

- 时间内聚又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。例如初始化模块和终止模块。

逻辑内聚（Logical Cohesion）

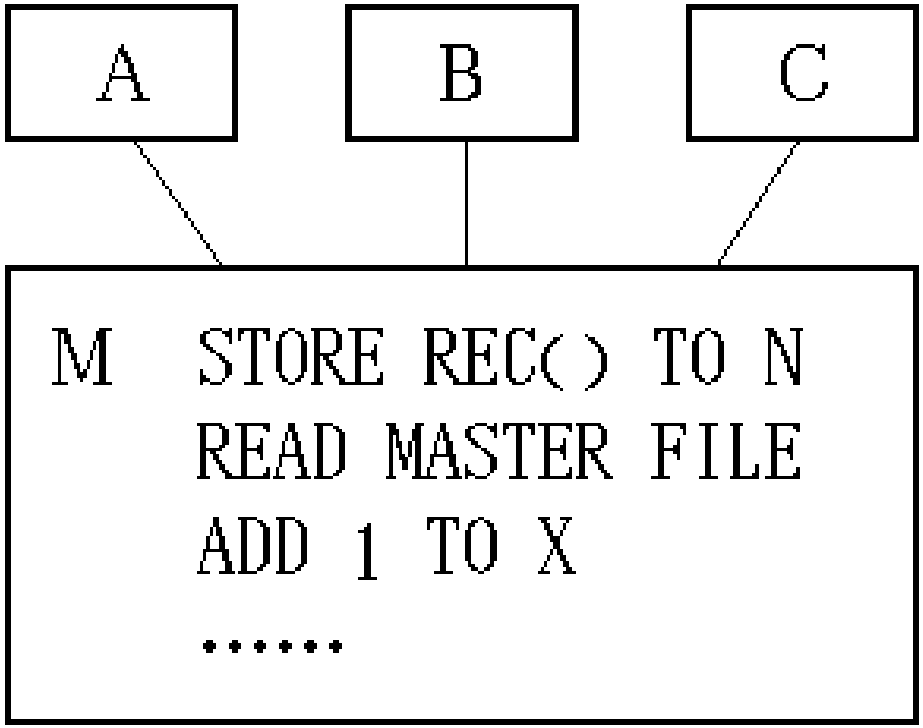
- 这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。



巧合内聚（Coincidental Cohesion）

- 巧合内聚（偶然内聚）。当模块内各部分之间没有联系，或者即使有联系，这种联

  系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。



7.3 设计概念（续）

- 求精
  - 层次结构的开发将通过逐步分解功能的宏观陈述（过程抽象）直至形成程序设计语言的语句。
  - 精化促使设计者在原始陈述上细化，并随着每个精化（细化）的持续进行提供越来越多的细节。

7.3 设计概念（续）

- 重构
  - 重构是使用这样的一种方式改变软件系统的过程：不改变代码的外部行为而是改进其内部结构
  - 检查冗余、未使用的设计元素、低效或不必要的算法、低劣或不恰当的数据结构以及设计不足，修改以获取更好设计

7.3 设计概念（续）

- 设计类
  - 分析类关注用户或客户可见的问题，描述问题域中的某些元素
  - 设计类
    1. 通过提供设计细节精化分析类
    2. 创建一组新的设计类，实现软件的基础设施以支持业务解决方案

7.3 设计概念（续）

- 五种设计类：
  - 用户接口类：定义人-机交互所必须的所有抽象
  - 业务域类：分析类的精化
  - 过程类：实现完整管理业务域所必须的低层次业务抽象
  - 持久类：代表将在软件管理业务域类中所必须的低层业务抽象
  - 系统类：实现软件管理和控制功能

7.3 设计概念（续）

- 组织良好的设计类的四个特征：
  - 完整性与充分性
  - 原始性
  - 高内聚性
  - 低耦合性

7.4 设计模型

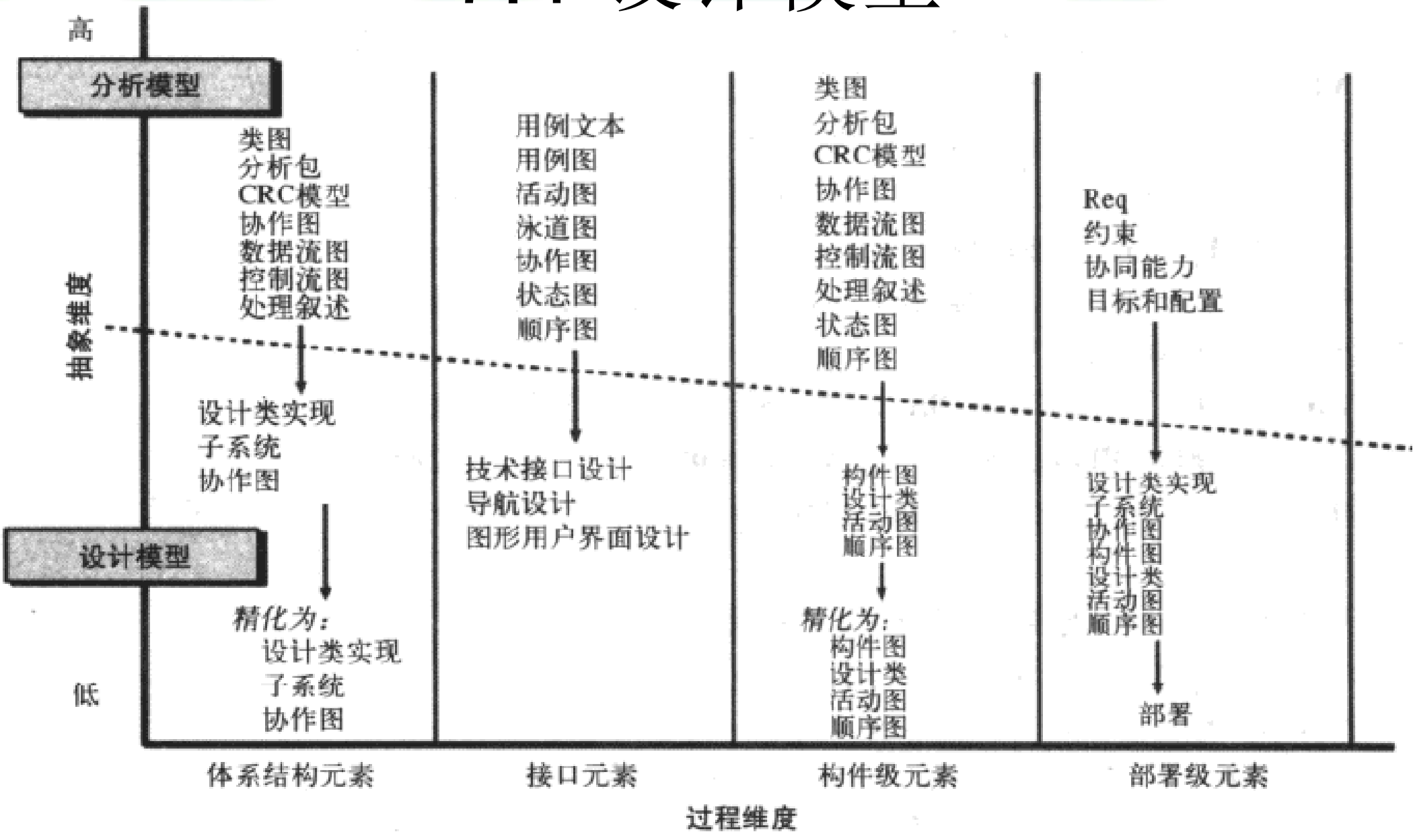


图　　设计模型的维度

7.4 设计模型（续）

- 设计建模原则
  - 原则1:设计应可追溯到需求模型
  - 原则2:始终考虑要构建系统的体系结构
  - 原则3:数据设计与处理功能设计同等重要
  - 原则4:接口（内部和外部）的设计必须谨慎
  - 原则5:用户界面设计应适应最终用户的需求

7.4 设计模型（续）

- 设计建模原则
  - 原则6:构件级设计应在功能上独立
  - 原则7:构件应彼此松耦合，并应与外部环境松耦合。
  - 原则8:设计表示（模型）应易于理解
  - 原则9:设计应迭代式开发
  - 原则10:设计模型的创建并不排除采用敏捷方法的可能性

7.4 设计模型（续）

- 数据结构元素

以客户/用户的数据观点表述的数据模型和/或信息模型

计算机的系统能够处理的表示

7.4 设计模型（续）

- 体系结构设计元素
  - 为我们提供软件的整体视图
  - 体系结构的三个来源：
    - 将要构建的软件应用域信息
    - 特定的分析模型元素，例如数据流图或分析类、现有问题中的关系和协作
    - 体系结构模式和风格的可获得性

7.4 设计模型（续）

- 接口设计元素
  - 告诉我们信息如何流入和流出系统以及被定义的体系结构一部分的构件之间是如何通信的
  - 接口设计的三个重要元素：用户界面；和其他系统、设备、网络或其他的信息生产者或使用者的外部接口；各种设计构件之间的内部接口。

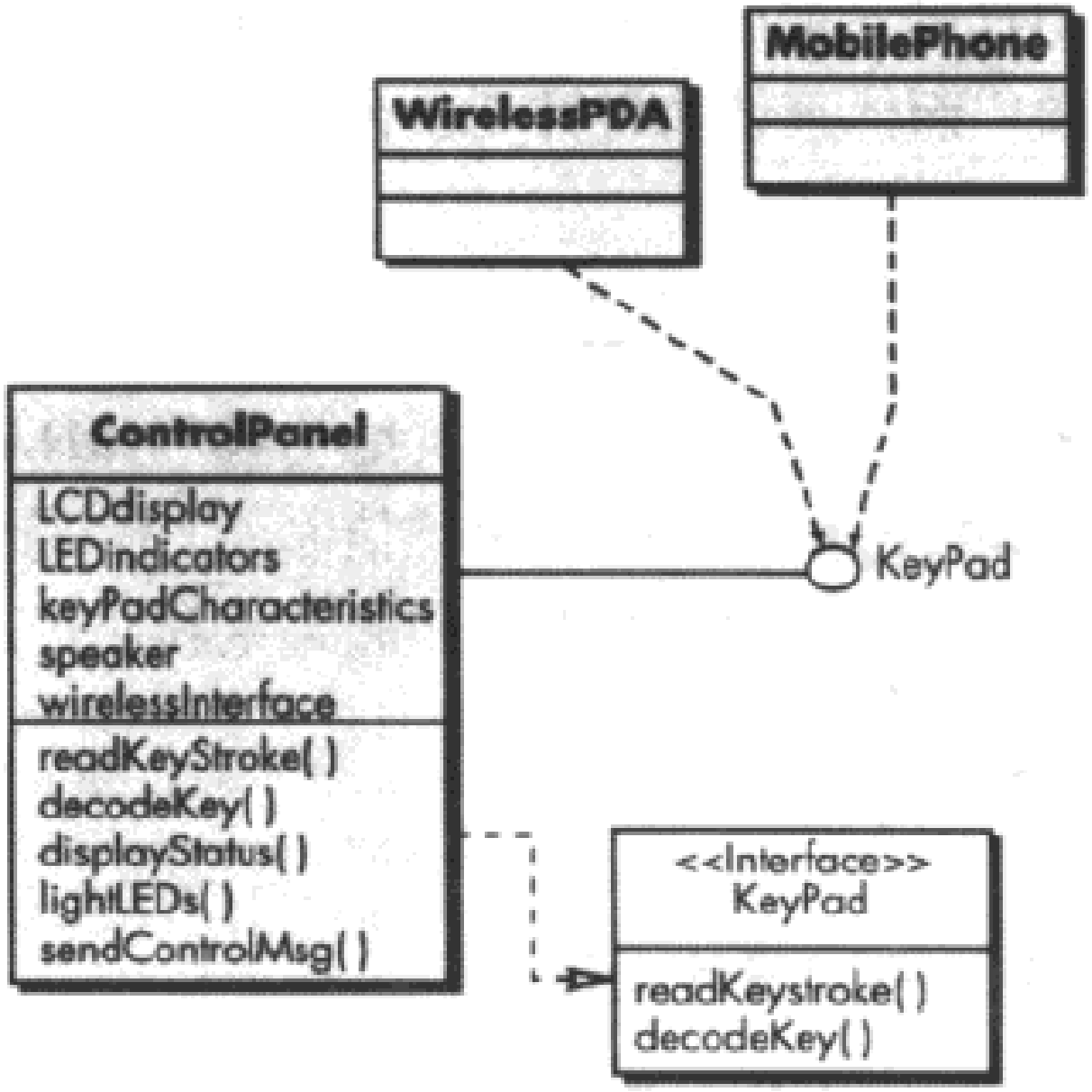


图　　ControlPanel（控制面板）的UML接口设计表示

接口是一组描述类的部分行为的操作，并提供了那些操作的访问方法

7.4 设计模型（续）

- 构件级设计元素
  - 描述了软件构件的内部细节。
  - 为所有本地数据对象定义数据结构、为所有在构件内发生的处理定义算法细节，并定义允许访问所有构件操作（行为）的接口

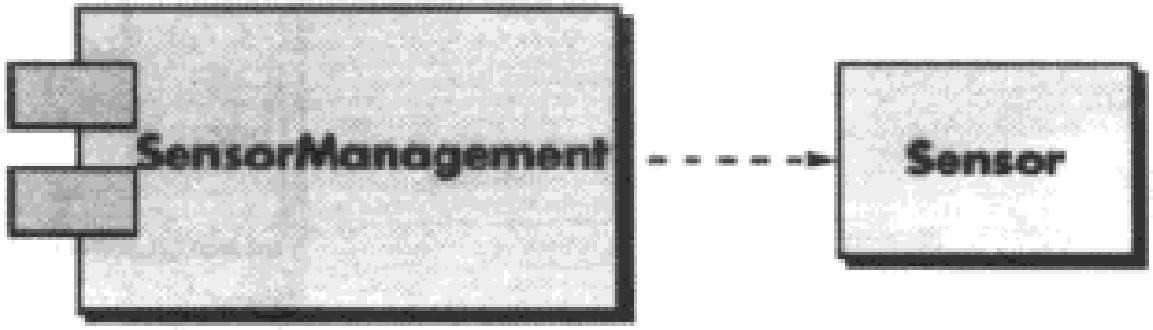
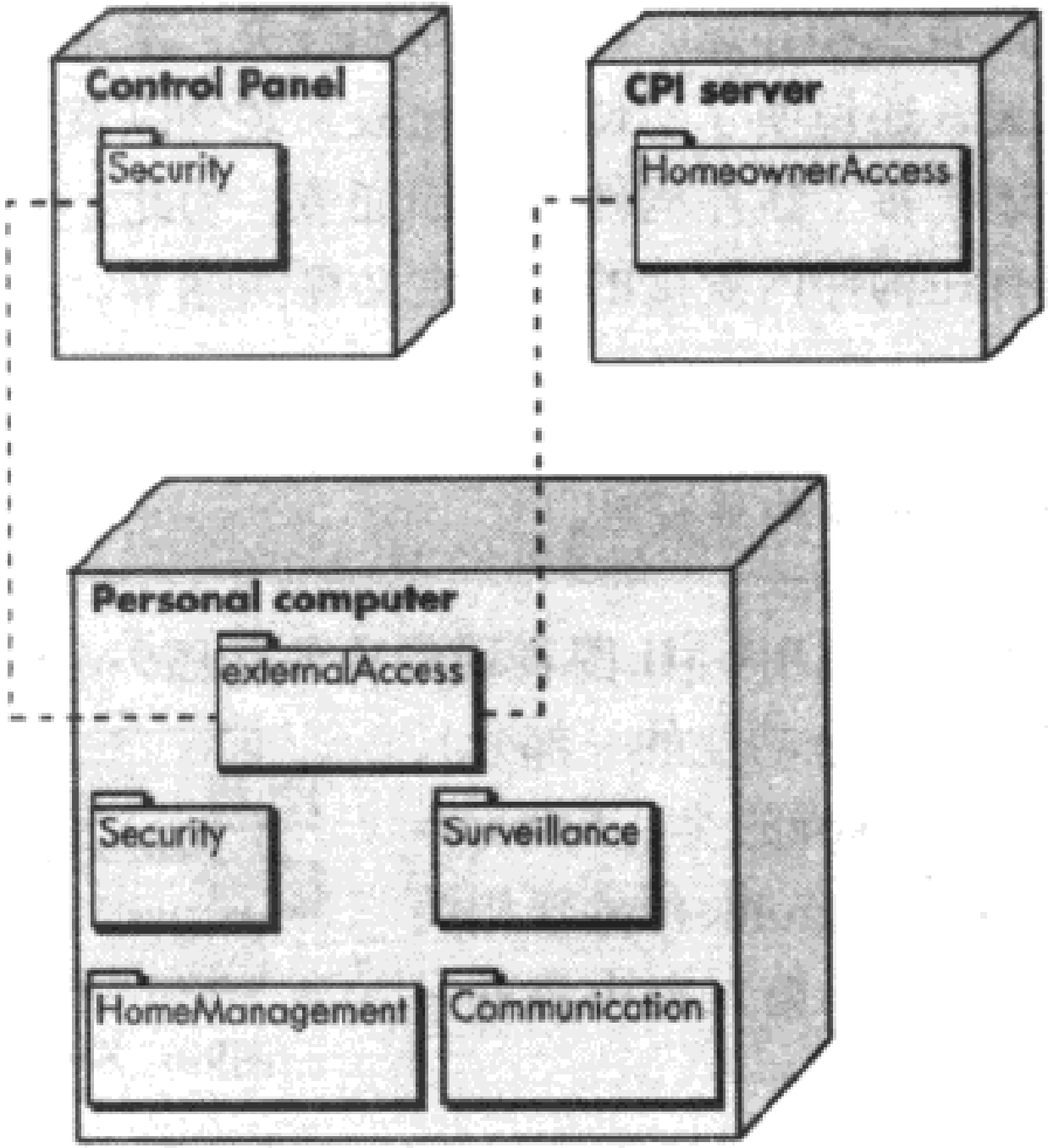


图　　SensorManagement的UML构件图

7.4 设计模型（续）

- 部署级设计元素
  - 指明软件功能和子系统将如何在支持软件的物理计算环境内分布。



图' SafeHome的UML部署图

7.5 基于模式的软件设计

- 贯穿整个设计过程，软件设计人员都应寻找每个机会重用现有的设计模型（当它们满足设计需要时）而不是创建一个新的

7.5.1 描述设计模式

设计模式模板：

- 模式名
- 含义
- 其他名称
- 动机（问题示例）
- 适用性
- 结构（所必须的类）
- 参与者（必须类的职责）
- 协作（参与者如何协作完成职责）
- 因果关系（设计的影响因素）
- 相关模式

7.5.2 在设计中使用模式

- 在整个设计中都可以使用设计模式
  - 体系结构模式
  - 设计模式
  - 习惯语法

7.5.3 框架

- 框架不是设计模式，而是一个带有“插入点”集合的骨架，插入点使得体系结构能够适应特定的问题域。

构件—模式—框架

- 构件通常是代码重用
- 设计模式是设计重用
- 框架则介于两者之间，部分代码重用，部分设计重用，有时分析也可重用。