

第7讲 软件概要设计

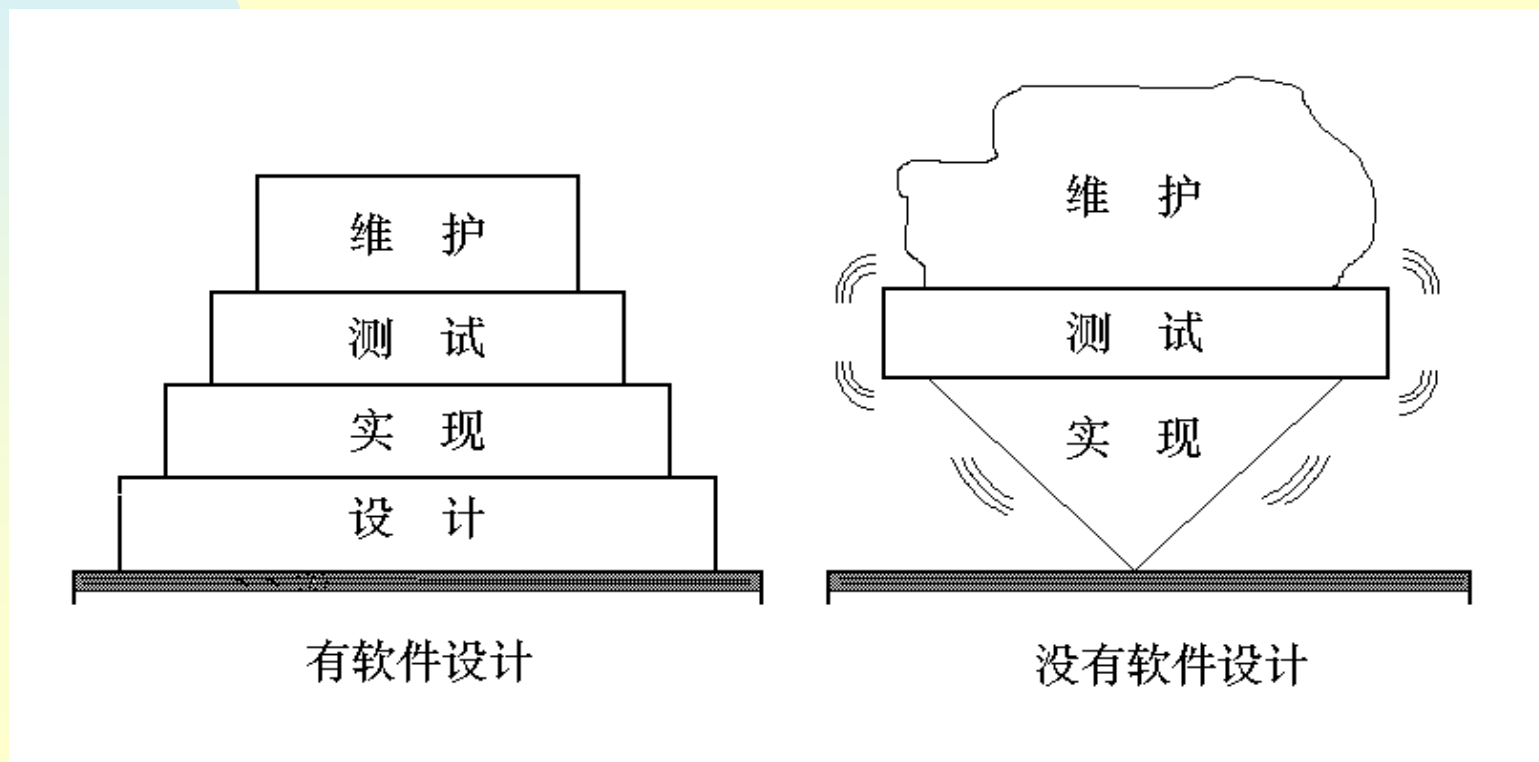
网络空间安全学院

芦效峰

授课内容

- 软件工程概述
- 可行性研究
- 需求分析（UML）
- 概要设计（UML）
- 详细设计
- 人机交互/用户界面
- 编码
- 软件测试
- 软件维护
- 软件项目管理

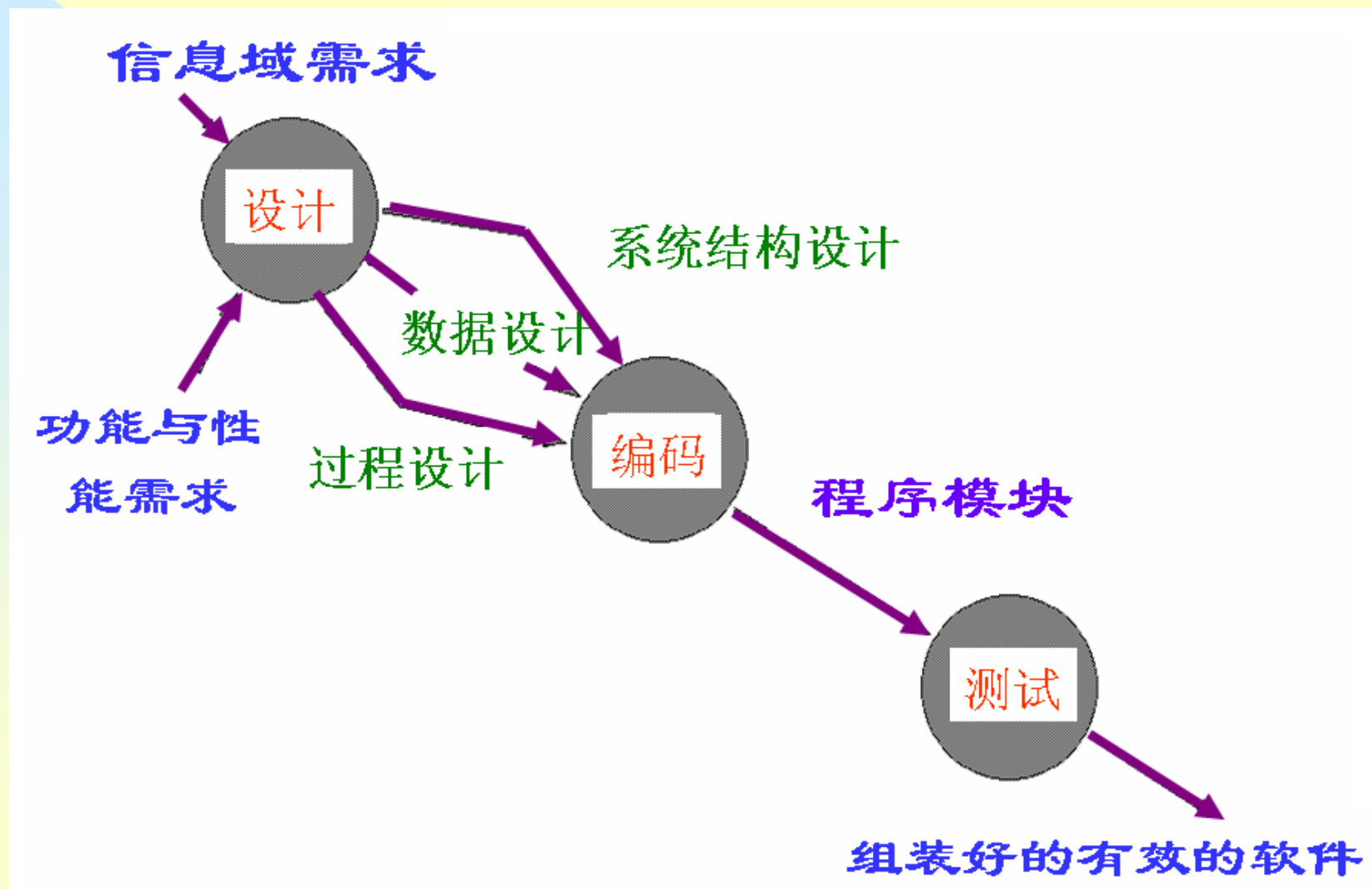
- 软件设计是后续开发步骤及软件维护工作的基础。
- 软件设计直接影响软件质量。如果没有设计，只能建立一个不稳定的系统结构



1. 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
 - 过程设计
 - 系统结构设计

开发阶段的信息流



- 数据设计把分析阶段创建的信息域模型转变成实现软件所需要的数据结构。
- 过程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。
- 系统结构设计定义软件系统各主要成份之间的关系。

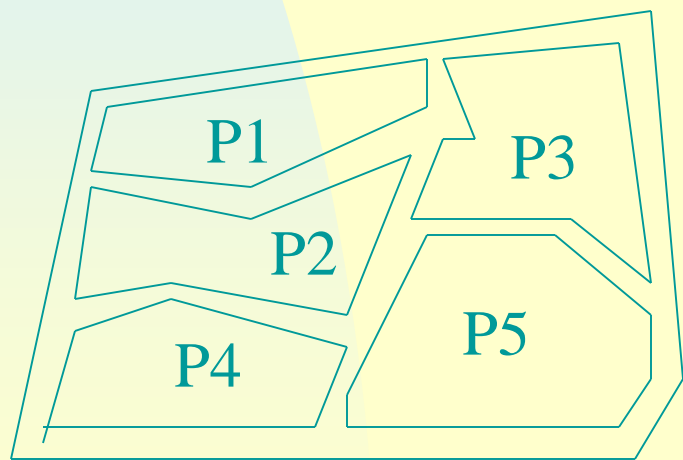
2. 软件设计任务

- 从工程管理的角度来看，软件设计分两步完成。
 - ◆ **概要设计**，将软件需求转化为**数据结构和软件的系统结构**。
 - ◆ **详细设计**，即**过程设计**。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/流程**。

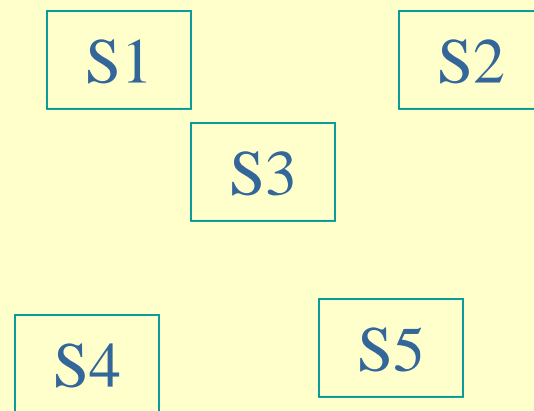
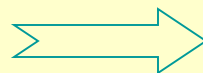
软件系统结构

软件概要（总体）设计的主要任务就是**软件结构的设计**。包含了计算机程序的两个重要特性：模块的层次结构、数据结构。

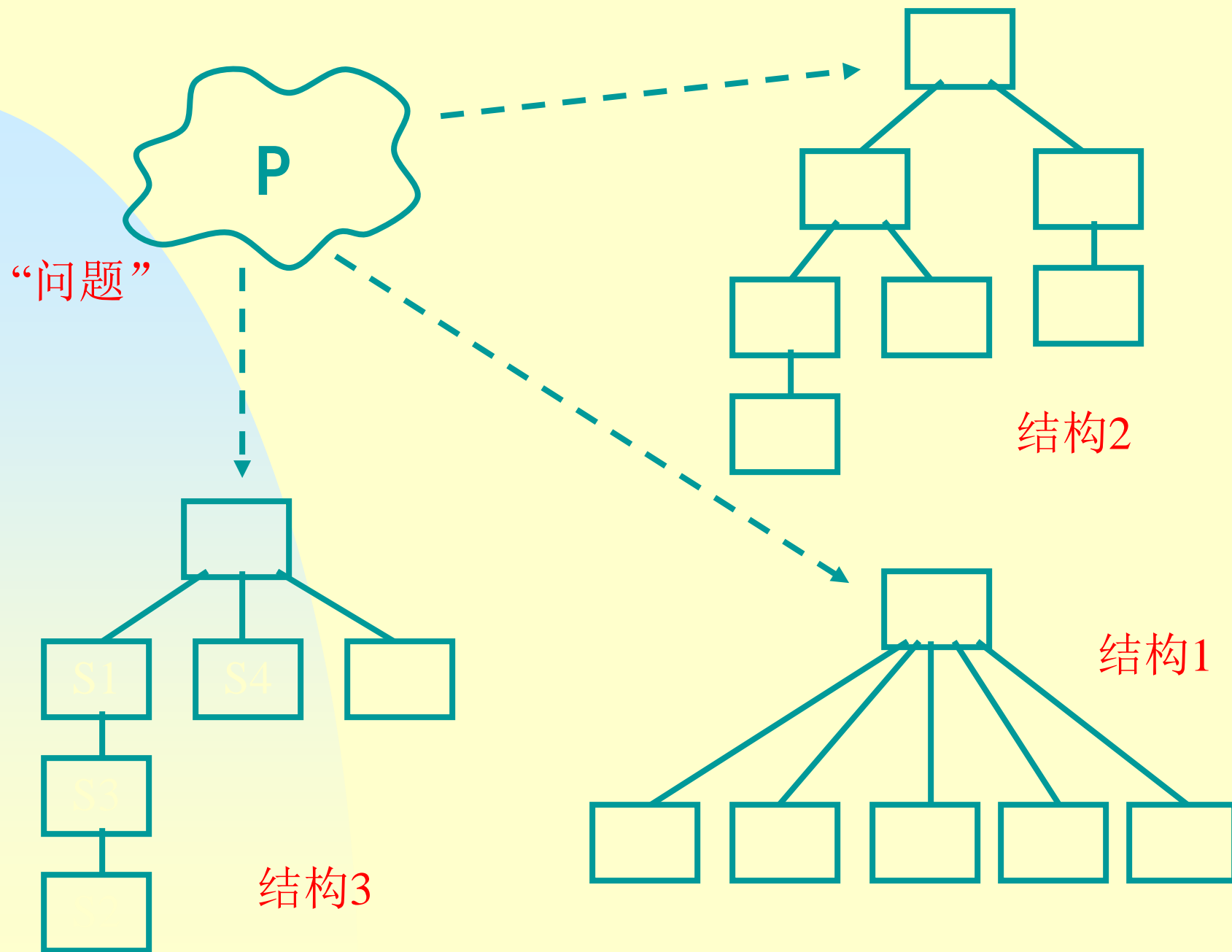
软件结构和数据结构的演化，从问题定义开始。当问题的各部分分别由一个或多个软件元素求解时，问题解也就有了。



软件求解的“问题”



软件的“解法”



3 概要设计的过程

- 设想供选择的方案（低、中、高）
- 选取审查合理的方案（参照可行性与投资）
- 功能分解（使功能显而易见）
- 设计软件结构（模块的调用层次）
- 数据库设计
- 制定测试计划（测试计划从需求阶段开始）
- 书写文档
- 审查和复查

文档

- 系统说明书，包括：
 - ◆ 用系统流程图描绘的系统构成方案
 - ◆ 细化的数据流图
 - ◆ 用层次图或结构图描绘的软件结构
 - ◆ **IPO**图或其他工具简要描绘的模块算法
 - ◆ 模块间的接口关系
- 用户手册
- 测试计划
- 实现计划
- 数据库设计

4 软件设计的基本原理

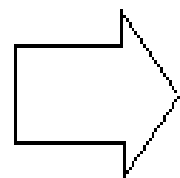
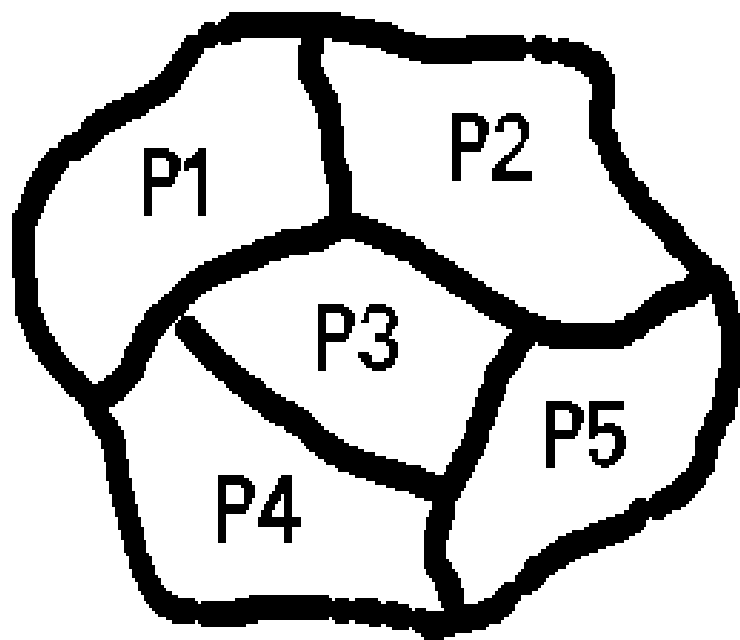
- 自顶向下，逐步细化
- 软件结构
- 模块化
- 抽象化
- 信息隐蔽

1.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。

2.软件结构

- 软件结构包括两部分。程序的模块结构和数据的结构
- 软件的结构通过一个划分过程来完成。该划分过程从需求分析确立的目标系统的模型出发，对整个问题进行分割，使其每个部分用一个或几个软件成份加以解决，整个问题就解决了



S1

S2

S3

S4

S5

需要通过软件解决的“问题”

软件的“解决方案”



3.模块化

■ 模块（Module）

“模块”，又称“组件”。模块是数据说明、可执行语句等程序对象的集合，它是**单独命名的而且可以通过名字来访问**。如：过程，函数，子程序，宏等。它一般具有如下三个基本属性：

- ◆ **功能**：描述该模块实现什么功能
- ◆ **逻辑**：描述模块内部怎么做
- ◆ **状态**：该模块使用时的环境和条件

- 在描述一个模块时，还必须按模块的**外部特性**与**内部特性**分别描述
- 模块的**外部特性**
 - ◆ 模块的模块名、参数表、其中的输入参数和输出参数，以及给程序以至整个系统造成的影响
- 模块的**内部特性**
 - ◆ 完成其功能的程序代码和仅供该模块内部使用的数据

模块化

- 模块化就是把程序划分成若干个模块的过程，每个模块完成一个子功能，把这些模块集合起来组成一个整体，可以完成指定的功能满足问题的要求。

模块化的依据：（各个击破）

设函数 $C(X)$ 定义问题 X 的复杂程度，

函数 $E(X)$ 确定问题 X 需要的工作量。

对于两个问题 $P1$ 和 $P2$ ，

如果： $C(P1) \gg C(P2)$

显然： $E(P1) > E(P2)$

根据人类解决问题的经验，一个有趣的规律是：

$$C(P1+P2) > C(P1) + C(P2)$$

综合考虑得：

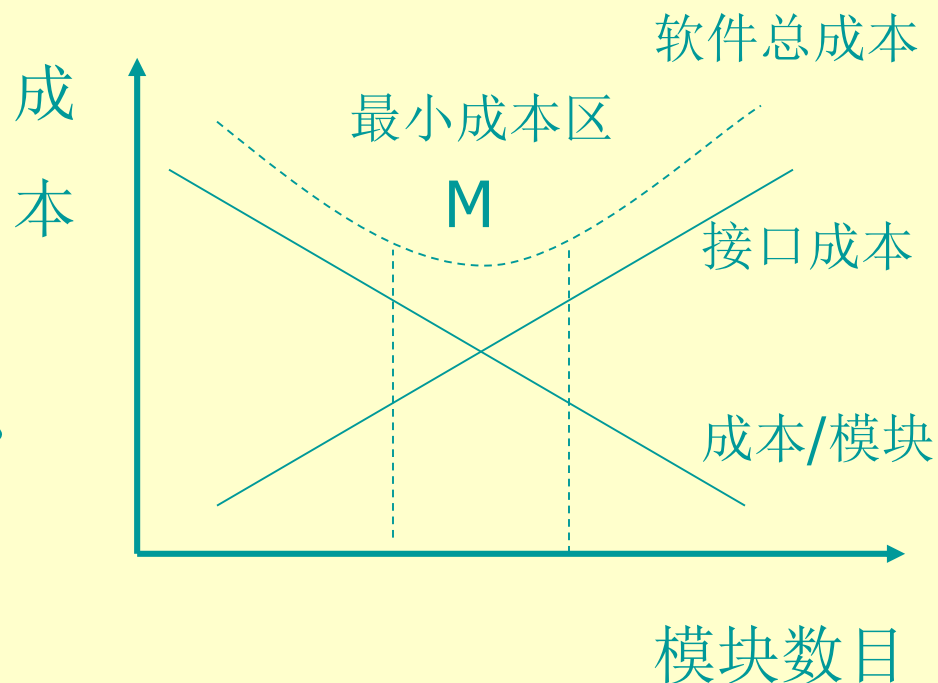
$$E(P1+P2) > E(P1) + E(P2)$$

模块化与软件成本的关系

模块化的好处

- 可以使软件结构清晰；
- 容易测试和调试；
- 提高可修改性；
- 提高代码复用
- 有助于软件开发的组织管理。

- 是不是模块越多越好？



4.抽象与求精

概念：在现实世界中一定事物、状态或过程之间总存在着某些**相似的共性**，把这些相似的方面集中和概括起来，暂时忽略它们之间的差异，这就是抽象。

抽象具有不同的程度：

用分层的方法，自顶向下，逐步求精。**抽象的高层：**使用问题环境语言，以概括的方法叙述问题的解法。**较低抽象层次：**采用更过程化的方法，把面向问题的术语和面向实现的术语结合起来叙述问题的解法。**最低抽象层次：**用可以直接实现的方法叙述问题的解法。

优点 简化了软件的设计与实现，提高了可理解性和可测试性，易于维护。

抽象和求精

- 求精实际上是细化过程。
- 抽象和求精是一对互补的概念。抽象能说明数据和过程，并忽略低层细节。抽象忽略多余的细节同时强调相关的细节。求精在设计过程中揭示出低层细节。
- 原因：一个人在任何时候都只能把注意力集中在 7 ± 2 个知识块上。

(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，采用“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。当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

用户交互任务；

2维绘图生成任务；

图形显示；

绘图文件管理；

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 interaction occurs)

digitizer interface task;

CASE (drawing request) :

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。

5.信息隐藏

信息隐藏原则指出：应该这样设计和确定模块，使得一个模块内包含的信息对于不需这些信息的模块来说，是不能访问的。

隐藏的不是有关模块的一切信息，而是模块的实现细节。隐藏意味着有效的模块化可以通过定义一组独立的模块来实现。模块之间只交换那些为了完成软件功能必须交换的信息。

局部化：是指把一些关系密切的软件元素物理地放地彼此靠近（局部数据元素）。

6.模块独立性

- ◆ 模块独立性,是指软件系统中每个模块只涉及软件要求的具体的子功能,而和软件系统中其它的模块的接口是简单的
- ◆ 例如,若一个模块只具有单一的功能且与其它模块没有太多的联系,则称此模块具有模块独立性

模块独立性重要的原因：

(1) 有效的模块化的软件比较容易开发出来；

(2) 独立的模块比较容易测试和维护。

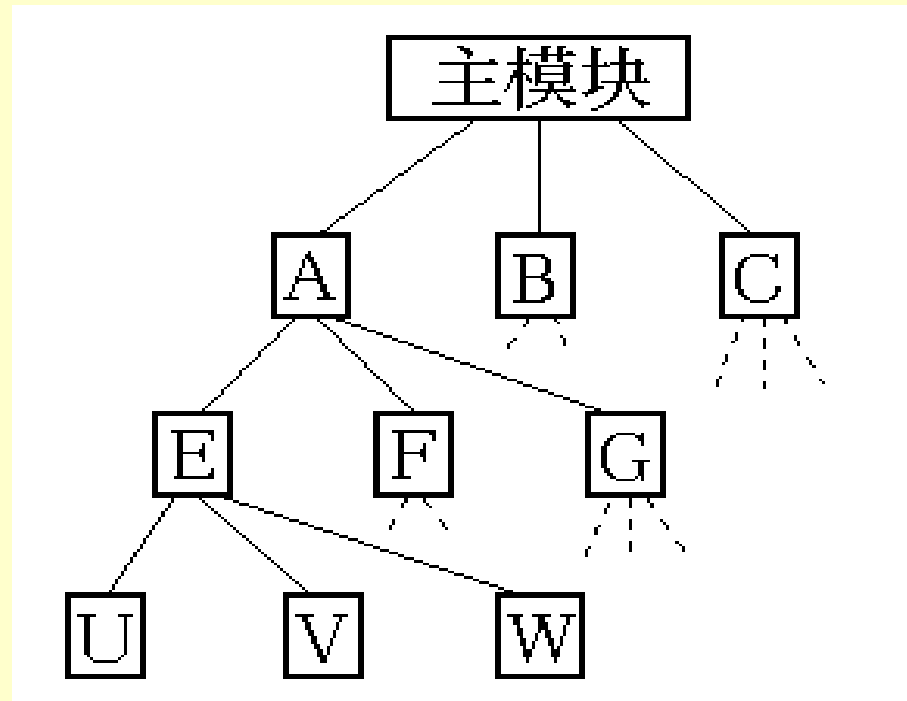
模块独立的度量标准：内聚和耦合。

- ◆ **耦合**是模块之间的互相连接的紧密程度的度量。
- ◆ **内聚**是模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量。
- ◆ 模块独立性比较强的模块应是**高内聚、低耦合**的模块。

非直接耦合(Nondirect Coupling)

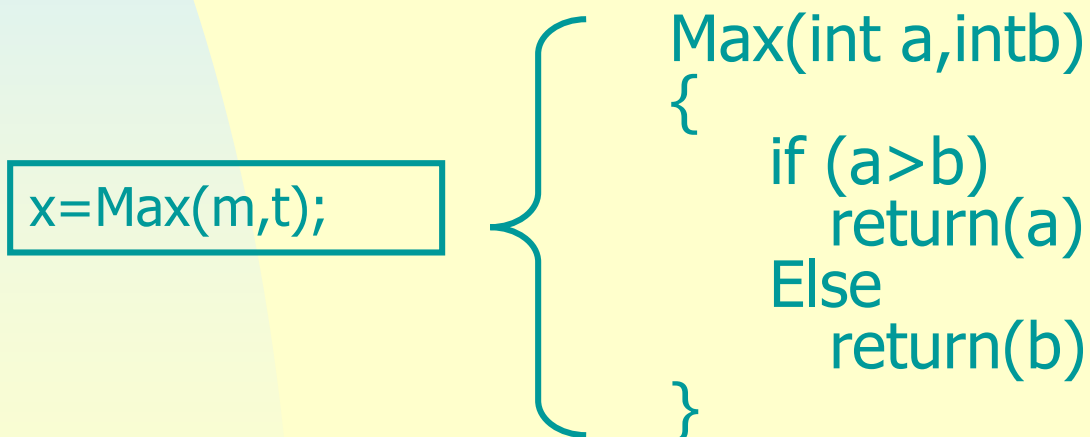
两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。

非直接耦合的
模块独立性最
强。



数据耦合

模块之间只是通过参数交换信息，且交换的信息只是数据。数据耦合是最低程度的耦合。



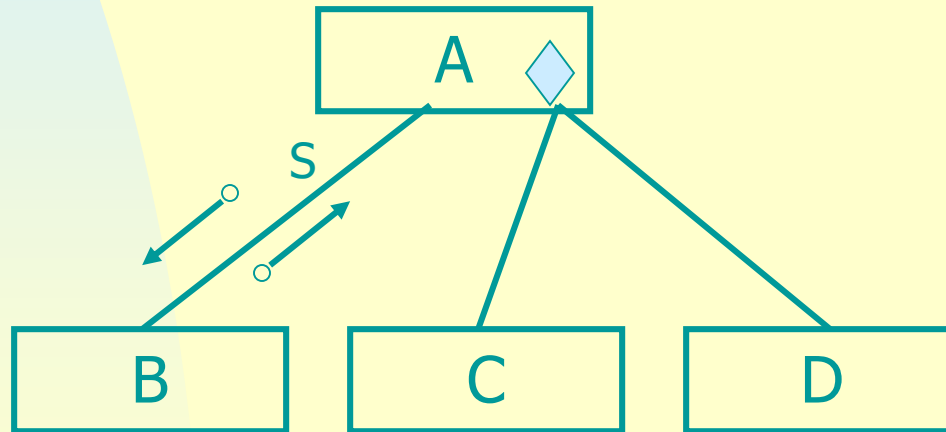
标记耦合 (Stamp Coupling)

模块间通过参数传递复杂的内部数据结构，就是标记耦合。

如高级语言的数组名,记录名,文件名等这些名字即为标记,其实传递的是这个数据结构的地址.

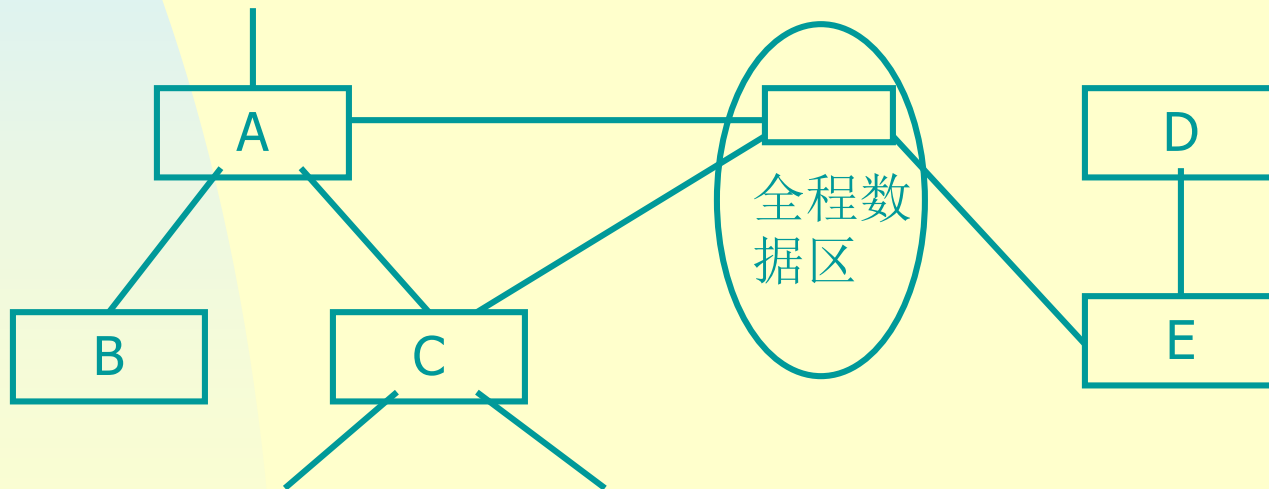
控制耦合

模块之间交换的信息包含控制信息。如：模块A的内部处理逻辑判断是决定执行C还是执行D，决定于模块B传来的信息标志。控制耦合是中等程度的耦合。

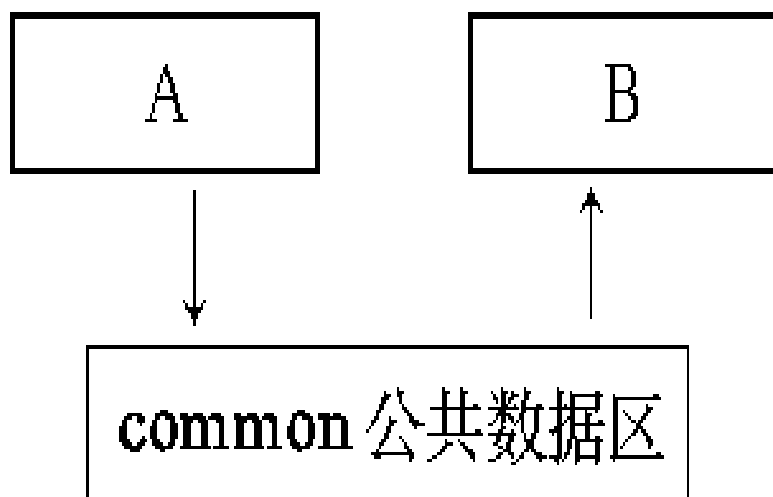


公用耦合（公共环境耦合）

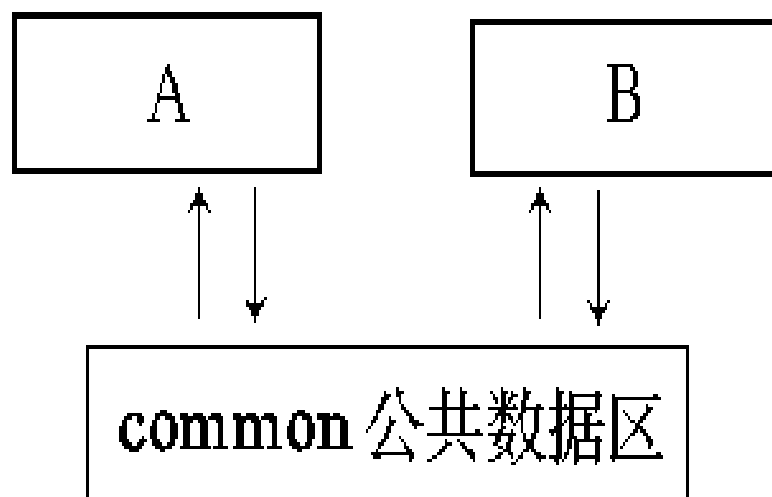
当两个或多个模块通过一个公共区相互作用时，它们之间的耦合称为公共环境耦合或公用耦合。这类公共区可以是全程变量、共享通讯区、内存公共覆盖区、任何介质上的文件、物理设备等。



- 公共耦合的复杂程度随耦合模块的个数增加而显著增加。若只是两模块间有公共数据环境，则公共耦合有两种情况。松散公共耦合和紧密公共耦合。



(a) 松散的公共耦合

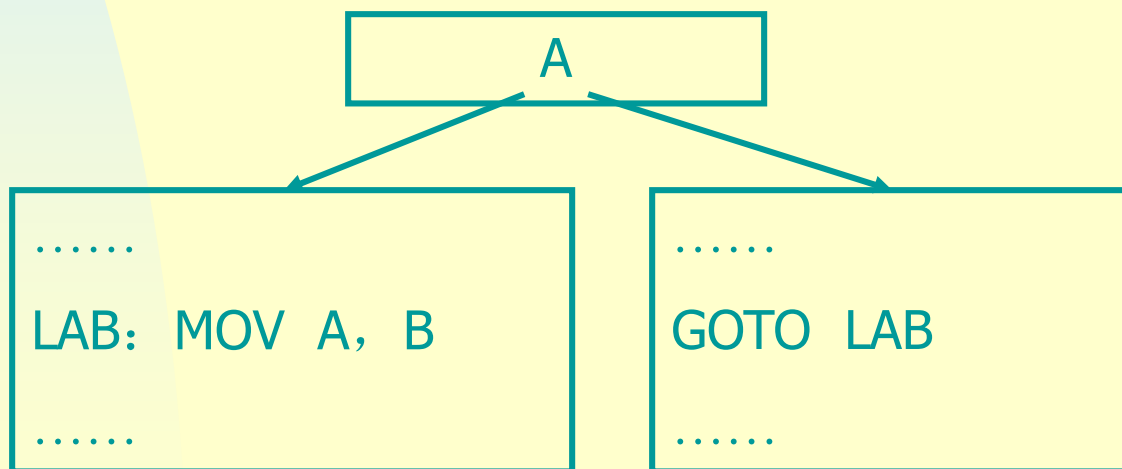


(b) 紧密的公共耦合

内容耦合

一个模块和另一个模块的内容直接发生联系。如：一个模块直接转移到另一个模块内部；一个模块使用另一个模块内部的数据等。

内容耦合是最高程度的耦合。应该避免使用内容耦合。



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。

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

内聚：

功能内聚

如果一个模块内所有处理元素完成一个，且仅完成一个功能，称为功能内聚。功能内聚是最高的内聚。

求a、b的最大值


```
Max(int a,intb)
{
    if (a>b)
        return(a)
    Else
        return(b)
}
```

但是，在软件结构中，并不是每个模块都能归结为完成一个功能而设计成一个功能内聚的模块。

顺序内聚

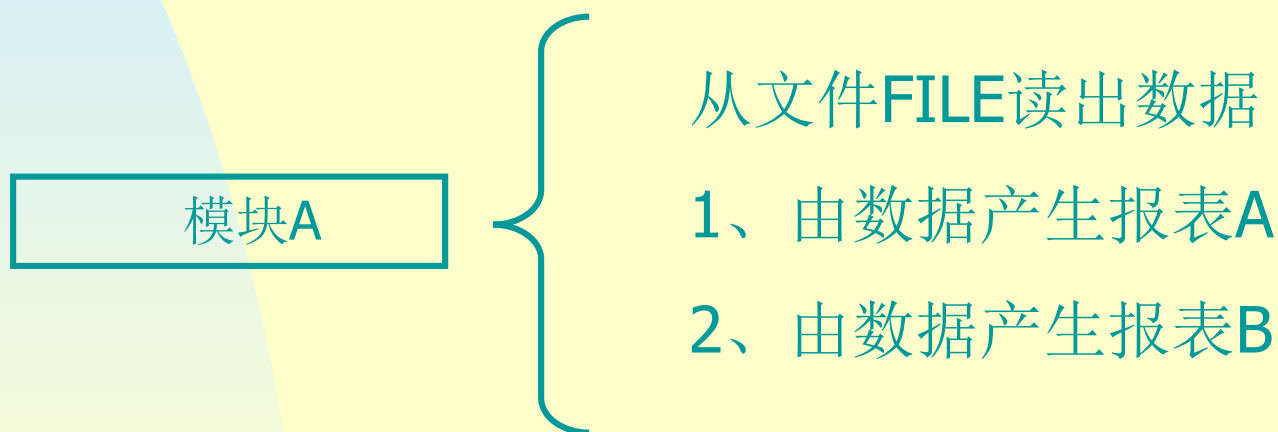
如果一个模块内处理元素和同一功能密切相关，而这些处理元素必须顺序执行，称为顺序内聚。通常一个处理元素的输出是另一个处理元素的输入。

求一元二次方程
根模块

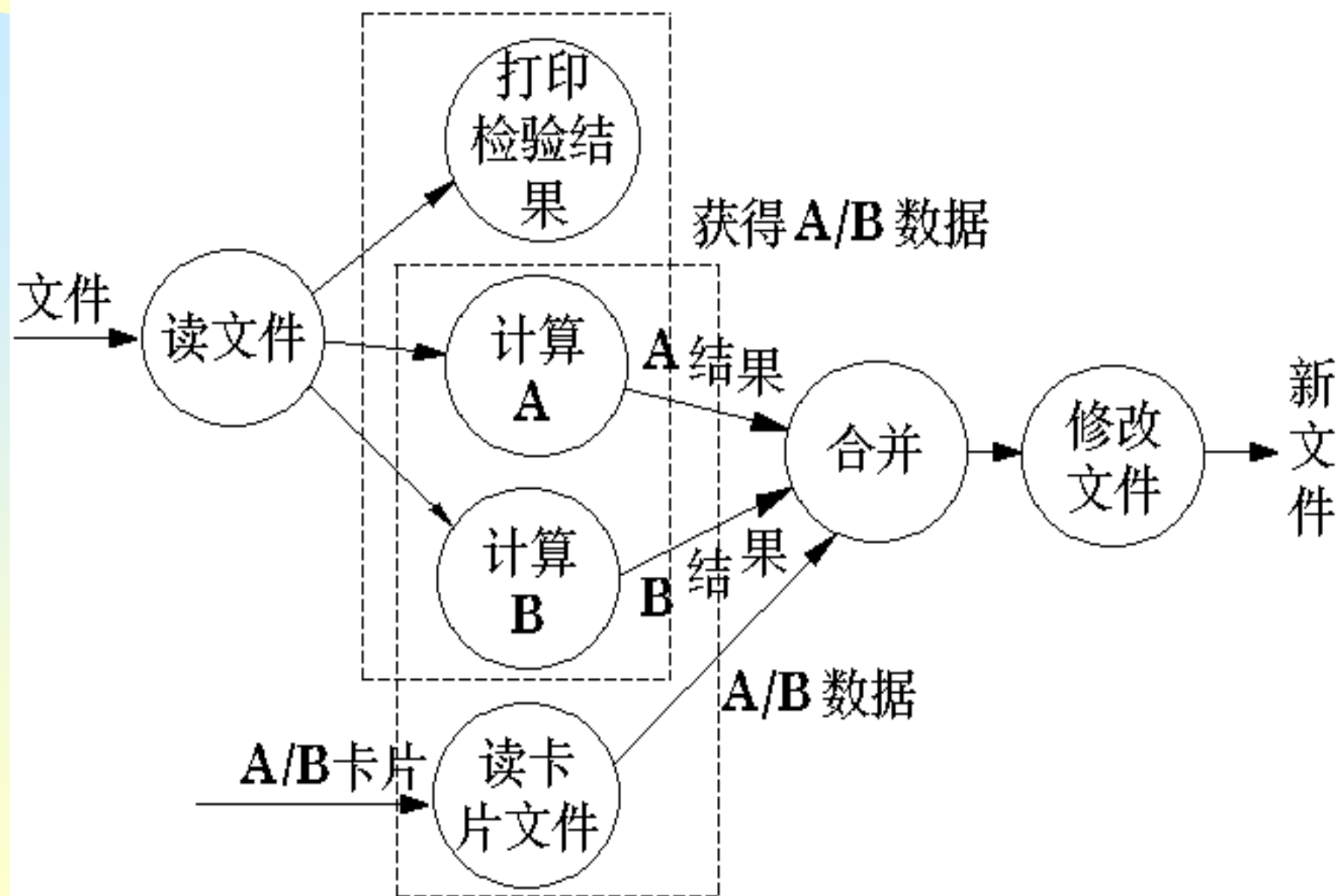
- 
- 1、输入系数
 - 2、求解
 - 3、打印方程的解

通讯内聚

如果一个模块中所有处理元素使用同一输入数据和（或）产生同一输出数据，称为通讯内聚。有时也称为数据内聚。



加工记录



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。

过程内聚内的处理元素必须以特定的次序执行。其各组成功能由控制流联结在一起，实际是若干个处理功能的公共过程单元。

过程内聚和顺序内聚的区别：

顺序内聚中是数据流从一个处理元流到另一个处理元。

过程内聚中是控制流从一个动作流到另一个动作。

时间内聚

(Classical Cohesion)

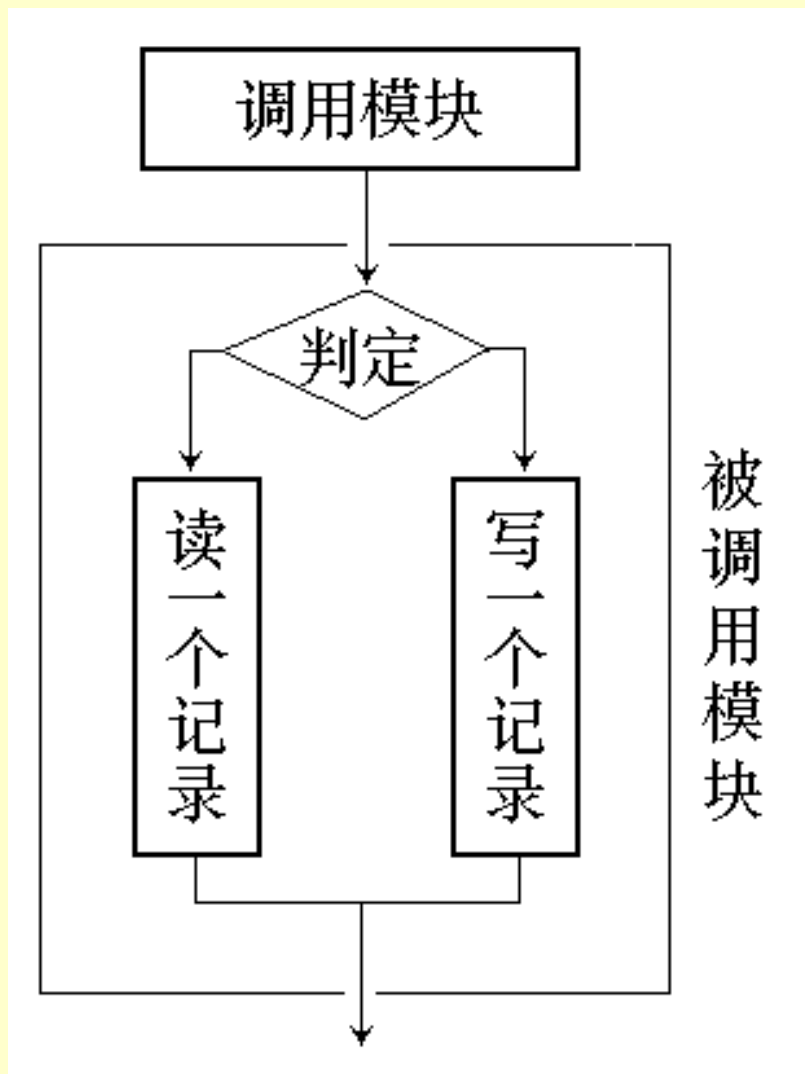
时间内聚又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。例如初始化模块和终止模块，处理故障模块。

紧急故障处理模块

- 1、关闭文件
- 2、报警
- 3、保留现场

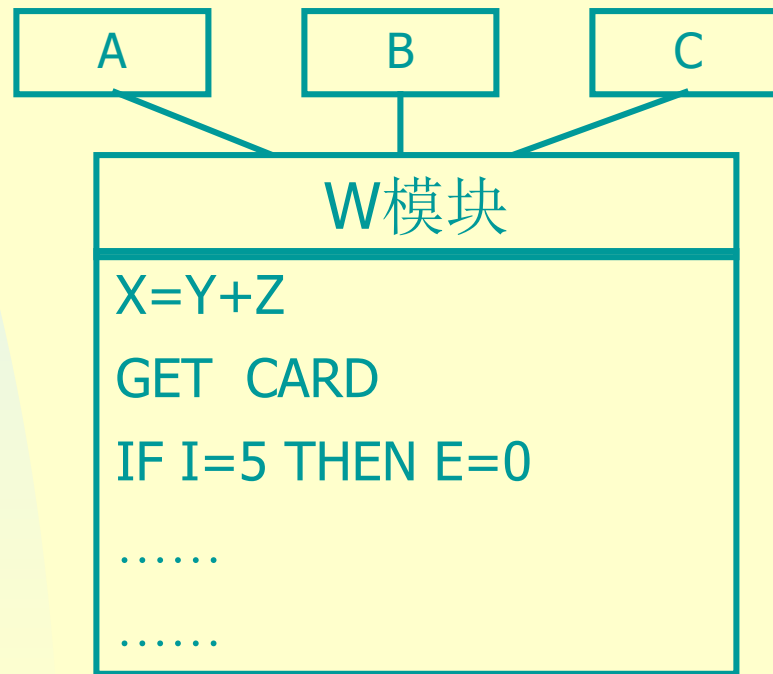
逻辑内聚（Logical Cohesion）

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。

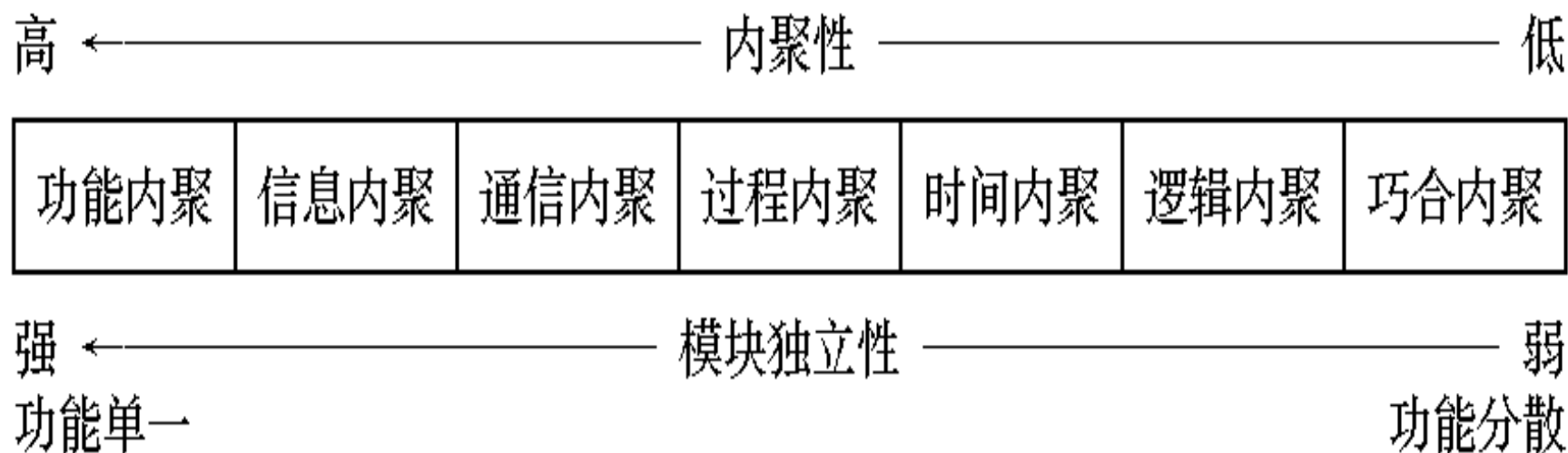


偶然内聚 (Coincidental Cohesion)

如果一个模块由完成若干毫无关系（或关系不大）的功能的处理元素偶然组合在一起的，称为偶然内聚。是最差的一种内聚。



模块内聚



关于模块独立性设计

尽量使用数据耦合，少用控制耦合，限制公共环境耦合，完全不用内容耦合。

力求高内聚，通常中等程度的内聚也是可以的，效果和高内聚差不多，但低内聚不要使用。

软件开发重点的转移

面向对象技术、可视化编程技术、组件技术，以及强大的软件工具的支持，使得原来实现起来很复杂、很耗费精力的软件实现技术变得非常简单，并且在某些环境下，可以不编程序代码，或编很少的程序代码，就可实现一个界面美观、可靠性高的软件。

另一方面，软件的规模和复杂性越来越大，软件运行环境越来越复杂，使得**软件开发的主要精力放在了软件运行基础环境的考虑、软件开发环境的配置、软件开发总体方案的制定、软件架构模式的选择、不同环境中软件元素间通信的实现、软件安装及运行环境的配置等事关全局的问题上。**

即软件设计已经从关心基本的结构和算法转移到了对宏观结构的认识上；从关注功能的实现转移到了对综合性能的要求上，软件架构的设计成了关注的重点。

软件体系结构的概念

软件体系结构和软件架构对应的英文单词都是“**Software architecture**”，即它们是意义完全相同的两个中文单词用语。在使用它们时往往带有一种习惯上的差异，通常学术上用“软件体系结构”较多，在软件系统设计上用“软件架构”较多，如在软件公司里，有“软件架构师”的职位，但很难听到“软件体系结构师”的说法。

软件架构的概念

在软件架构概念的理解上通常分为两大流派：

- **组成派**：软件系统的架构将系统描述为计算组件及组件之间的交互。

- **决策派**：软件架构是一系列重要决策的集合。

与决策派相关的软件架构定义

Booch、Rumbaugh、Jacobson的定义：

架构是一系列重要决策的集合，这些决策与以下内容有关：软件的组织、构成系统的结构元素及其接口的选择，这些元素在相互协作中明确表现出的行为，这些结构元素和行为为进一步组合所构成的更大规模的子系统，以及指导这一组织—包括这些元素及其接口、它们的协作和它们的组合—架构风格。

Woods的观点：

软件架构是一系列设计决策，如果作了不正确的决策，你的项目可能最终会被取消。

与组成派相关的软件架构定义

Garlan和Shaw的定义：软件架构包括**构件、连接件和约束**三大要素。构件可以是一组代码（如程序模块），也可以是独立的程序（如数据库服务器）。连接件可以是过程调用、管道和消息等，用于表示组件之间的相互关系。“约束”一般为组件连接的条件。

Dewayne Perry和Alex Wolf的定义：软件体系结构是一组具有一定形式的体系结构元素。这组元素分为**3类**：负责完成数据加工的处理元素，作为被加工的信息的数据元素和连接元素。连接元素用于把架构的不同部分组合连接在一起。

IEEE610.12-1990软件工程标准词汇定义：体系结构是以**构件、构件之间的关系、构件与环境之间的关系**为内容的某一系统的基本组织结构以及指导上述内容设计与演化的原理。

与软件架构相关的概念—构件

广义上，构件是软件系统的结构块单元，是软件功能和承载体。所以，从系统的构成上看，任何在系统中承担一定功能、发挥一定作用的软件体都可以看成是构件。构件可以分为计算构件、数据构件和连接构件。

狭义上，把构成软件系统的结构块分为构件和连接件，而构件与连接件在一个软件系统中扮演的角色是可区分的。因此，狭义的构件指与系统设计目标（业务功能需求）对应的结构块（处理单元和数据单元），为狭义的构件之间协同运行穿针引线的结构块称为连接件。

构件与连接件的根本区别是：构件有领域业务处理的功能，连接件只起业务构件之间的中介作用

与软件架构相关的概念—构件

构件有不同粒度。一个构件可以小到只有一个过程（或函数），也可以大到包含整个应用程序。函数、例程、对象、类库、数据包、服务器、文件等都可以作为一个构件。

对应一个软件架构实例来说，用什么粒度的构件来描述软件架构呢？应该从软件“系统”层面上可以把软件分成多少个逻辑上相对独立的计算单元（子系统）。每一个相对独立的计算单元就是一个构件。如果每个子系统又是一个高度独立的系统，又可以以它作为观测的“系统”对象，将其分解为若干个逻辑上相对独立的计算单元，每个计算单元就是一个构件。

在不同设计环境中，构件可表现为控件、组件、表、实体、包、设计模式、框架等。

与软件架构相关的概念—连接件

连接是构件和构件之间建立和维持行为关联和信息传递的途径。连接包括：实现机制和信息交换协议。

1、实现机制

硬件层：过程调用、中断、存储、栈等

基础控制描述层：过程调用、中断/事件、流、文件、网络等。

资源管理调度层：进程、线程、共享、同步、并行、事件、异常、远程调用等。

高级抽象层：管道、解释器、转换器、浏览器、组件/中间件、C/S、B/S、ODBC等。

2、信息交换协议

信息交换协议是连接的规约，是实现有意义连接的保证。最简单的连接是过程调用，过程调用的实现也有协议，如接口参数的顺序和类型、如何把参数放到获得数据的一方能够取得到的地方等。

与软件架构相关的概念—连接件

连接件的设计已经成为软件开发的关键技术。

连接件的设计要考虑的因素包括不同涉众的需求、构件的形式、系统的性能、系统的可维护性、系统的安全性、系统的可靠性等。构件可以以多种形式出现。对于集中数据管理的分布式应用系统，共享后台数据库的不同构件之间的连接件。相互独立的应用系统之间的交互可以用Web Service 组件、消息中间件实现连接是目前广泛采用的技术。分布式系统采用选定的机制和协议实现连接。

与第三方认证的连接、与银行交费系统的连接、与税务系统的连接，都与系统的安全性、涉众利益直接相关，技术非常复杂。

C/S、B/S模式的构件之间的连接，都是通过中间件和协议实现。

两种架构设计的区别和联系

组成派和决策派关于软件架构概念的区别在于从不同角度来描述对软件架构概念的认识。**组成派是从软件架构的最终形态角度来描述软件架构，决策派是从软件架构形成过程来描述软件架构。**

软件架构最终要用构件、连接件及其约束来描述，这既是决策的主要内容和决策结果，也是软件架构最终的表现形式。这就是他们的共同点。

决策的内容范围更广，软件架构不仅注重软件本身的结构和行为，还注重可用性、功能性、性能、重用、可理解性、经济和技术等的限制等。任何一项决策都是属于软件架构的内容，有的决策可能与构件、连接件的选择无关。

一个软件项目开发可能涉及到多套方案，对方案的选择可能涉及到复杂的因素，需要高层决策者做出有远见的决策。这样的决策对系统的开发产生根本性的影响，软件系统最终表现的构件、连接件及其约束的选择是这些决策导致的产物。

实例—一个考试系统的设计决策（1）

对于一个考试系统，考试的公正性、可用性、系统性能是考试系统的关键质量属性。

1、对公正性的考虑：避免抄袭是重要措施。

方案：同一份母卷，产生不同的考试试卷。

考生登录后，直接生成考试试卷。

2、可用性考虑：考生年龄差异大、工作岗位特殊、有的考生计算机应用水平很低，可能无法输汉字。

方案：①考生登录只输数字型考号，登录后显示考生信息进行核实；②客观题机考，主观题可机考，也可笔试（通过投影仪显示主观题）

3、系统性能不影响考试进度和考生情绪。

前面**1、2**条的方案属于软件架构的内容，因为它是考试系统设计必须遵循的原则。

性能问题难以估计，将逐步解决。

实例—一个考试系统的设计决策（2）

1、生成试卷存在的问题：当超过**50**人考试时，生成试卷遇到性能瓶颈：等待时间长，甚至产生的试卷不完整。

原因：生成试卷是对母卷进行随机的大题交换、小题交换、备选答案交换等一系列复杂运算实现，运行时间长，并发操作不能太多。

解决方案：将试卷生成功能独立，提前一个时间量先生成考试试卷，考生登录后直接取试卷。

2、考生登录遇到的问题：当一次考试超过**300**人后，考生登录输入考号后不能如考生所期望的那样立即显示相关信息，需要等待一会儿，尽管只有几秒钟，但会影响考生情绪。

原因：尽管考生身份验证简单，但并发操作太多同样影响性能。

解决方案：将考生分成不同的逻辑班，不同逻辑班将考试时间错开一个小的时间间隔（如**5**分钟）。

实例—一个考试系统的架构设计（3）

3、网络环境对考试的影响：有的地区网络环境差，难以保证考试顺利进行；考生范围局限在某个地区，将考生集中到某个地方参加考试的成本大。

解决方案：通过考试系统下载与安装、考试数据下载、上传、装载等功能，辅之其它措施保证考试的有效性。

该方案将影响考试系统的物理架构，即系统是一个分布式架构，交互机制是通过网络进行文件下载、上传进行。



一个软件系统的设计，首先要对需求中提出的各种问题提出合理的解决方案。这些解决方案反映的是设计过程，代表的是高层设计决策。对这些设计决策具体化，综合化，就构成了以构件、连接件等形式表示的软件架构。

软件架构设计的关注点

软件设计的首要任务是实现系统的功能需求，功能不能满足用户需求的软件无论如何也算不上好软件。同时软件设计必须对软件的非功能要求提出解决方案，使开发的软件满足那些非功能要求，不能满足非功能要求的软件也不是一个好软件。简单地说，不能满足功能要求的软件是一个不能用的软件（不能解决用户关注的业务问题），不能满足非功能要求的软件是一个不好用的软件。

在以计算机网络和**Internet**为计算环境的今天，仅仅满足软件的功能要求是远远不够的。可以设想，如果学生学籍管理系统只由一台微机来实现，或者只能由连接相邻几个房间的局域网来实现，将是什么场景？

软件体系结构设计的关注点是用户的非功能需求。软件的非功能需求是软件结构设计的驱动力。

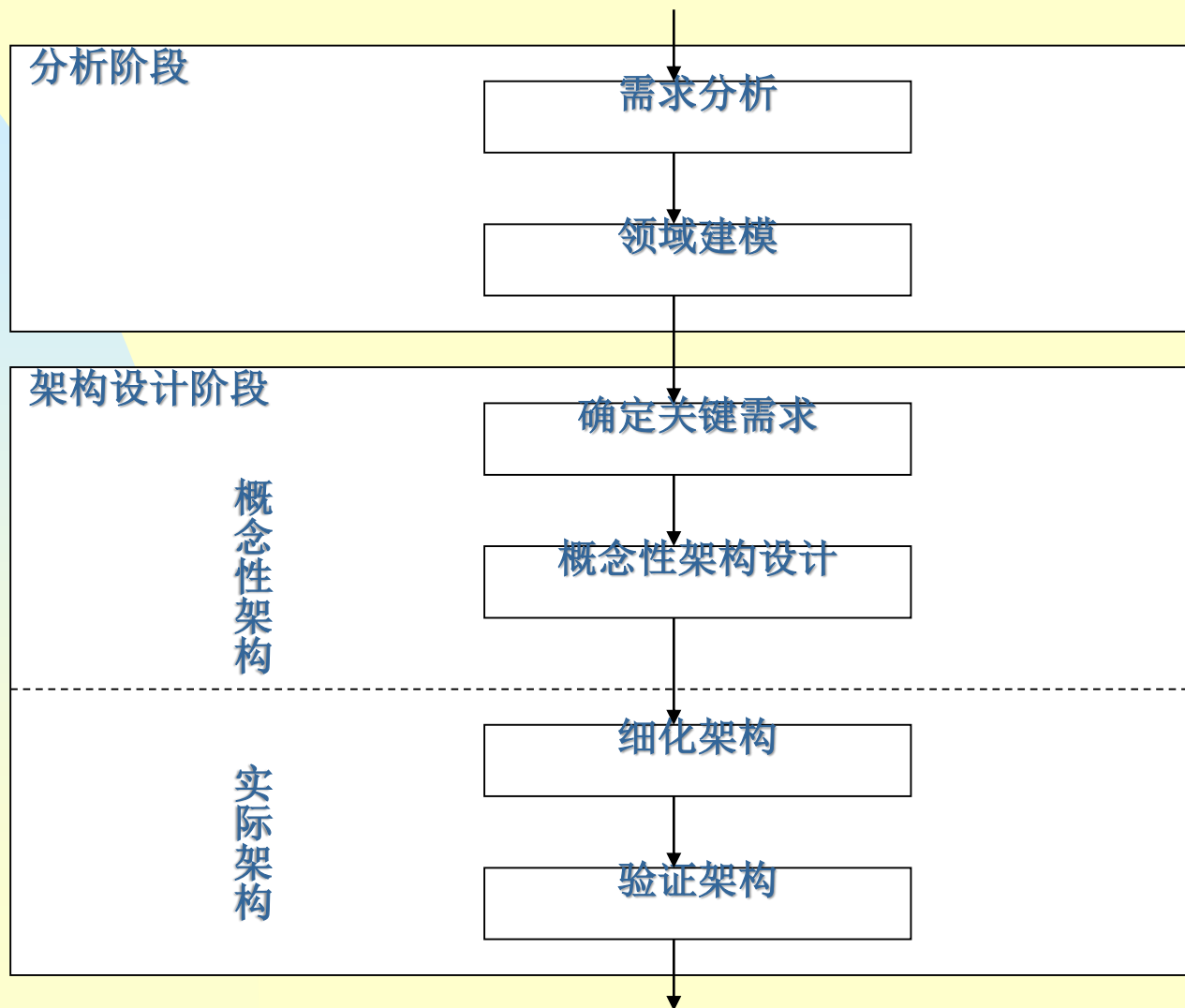
软件架构设计的内容

始终记住：

- 1、软件架构是一系列重要决策的集合。**
- 2、软件架构=构件+连接件+约束**
- 3、基于视图的架构模型具有指导作用。**

凡是与软件设计有关的重要决策都是软件架构设计的内容，可能不一定用构件或连接件表现出来，但会影响最终产生的构件或连接件的内容。

架构设计过程



关键需求决定架构

关键需求决定架构，其余需求验证架构，是架构设计应遵循的基本策略。

影响软件架构设计的关键需求包括功能需求、质量（属性）需求、商业需求三类。

关键的功能指影响系统成败的功能。如考试系统的在线考试功能是最关键的功能，其它功能无论多么完善，如果在线考试功能过不了关，就是一个不合格的考试系统。

有的系统围绕关键性能进行架构设计。电信系统、考试系统的性能是最关键的质量特性。**7*24**小时的不间断运行的系统，短暂停机就可能对生产造成重大影响。

商业需求指**软件系统开发和应用方面的商业考虑**，它关注从客户群、企业现状、未来发展、预算、立项、开发、运营、维护在内的整个软件生命周期涉及到的**商业因素**，包括了商业层面的目标、期望和限制。系统开发预算会影响对技术的选择。客户群对架构的影响，法律法规会影响系统的可扩展性、可修改性、可维护性。