3.1 需求分析的任务

3.2 与用户沟通获取需求的方法

3.3 分析建模与规格说明

3.4 实体联系图（不讲）

3.5 数据规范化（不讲）

3.6 状态转换图

3.7 其他图形工具

3.8 验证软件需求

# 3.6 状态转换图

状态转换图(简称状态图)通过描绘系统的状态及引起系统状态转换的事件,来表示系统的行为。状态图还指明了作为特定事件的结果系统将做哪些动作。

## 口 状态:

- 是任何可被观察的系统行为模式。一个状态代表系统的一种行为模式。状态规定了系统对事件的响应方式。
- 在状态图中定义的状态主要有:初态(即初始状态)、终态(即最终状态)和中间状态。

# 3.6 状态转换图

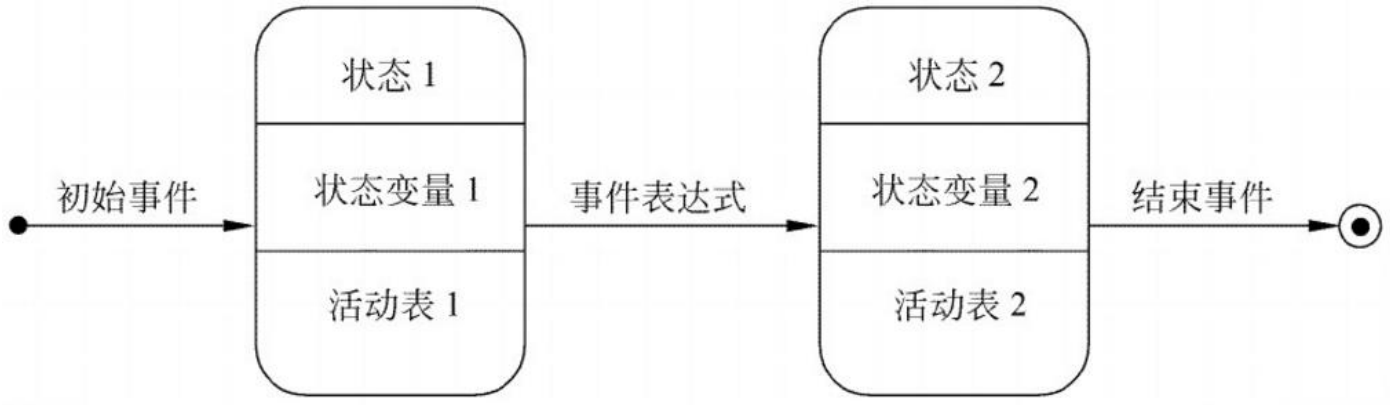
## 口 事件：

- 是在某个特定时刻发生的事，它是对引起系统做动作或(和)从一个状态转换到另一个状态的外界事件的抽象。
- 事件就是引起系统做动作或(和)转换状态的控制信息。

## 口 符号：

- 初态用实心圆表示，终态用一对同心圆(内圆为实心圆)表示。
- 中间状态用圆角矩形表示，可用两条水平横线把它分成上、中、下3个部分。上面部分为状态的名称(必须有)；中间部分为状态变量的名字和值(可选)；下面部分是活动表(可选)。

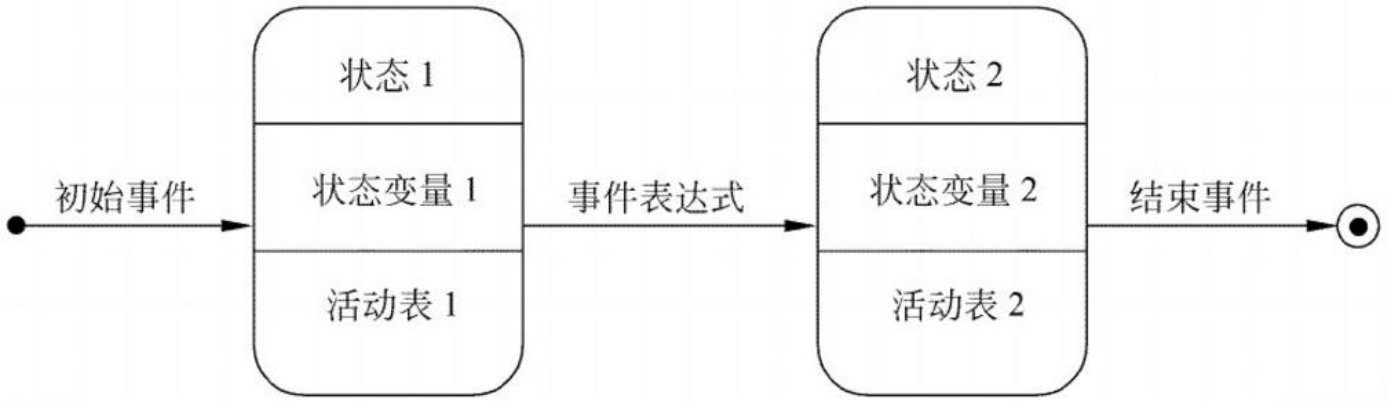
# 3.6 状态转换图




活动表的语法格式：事件名(参数表)/动作表达式

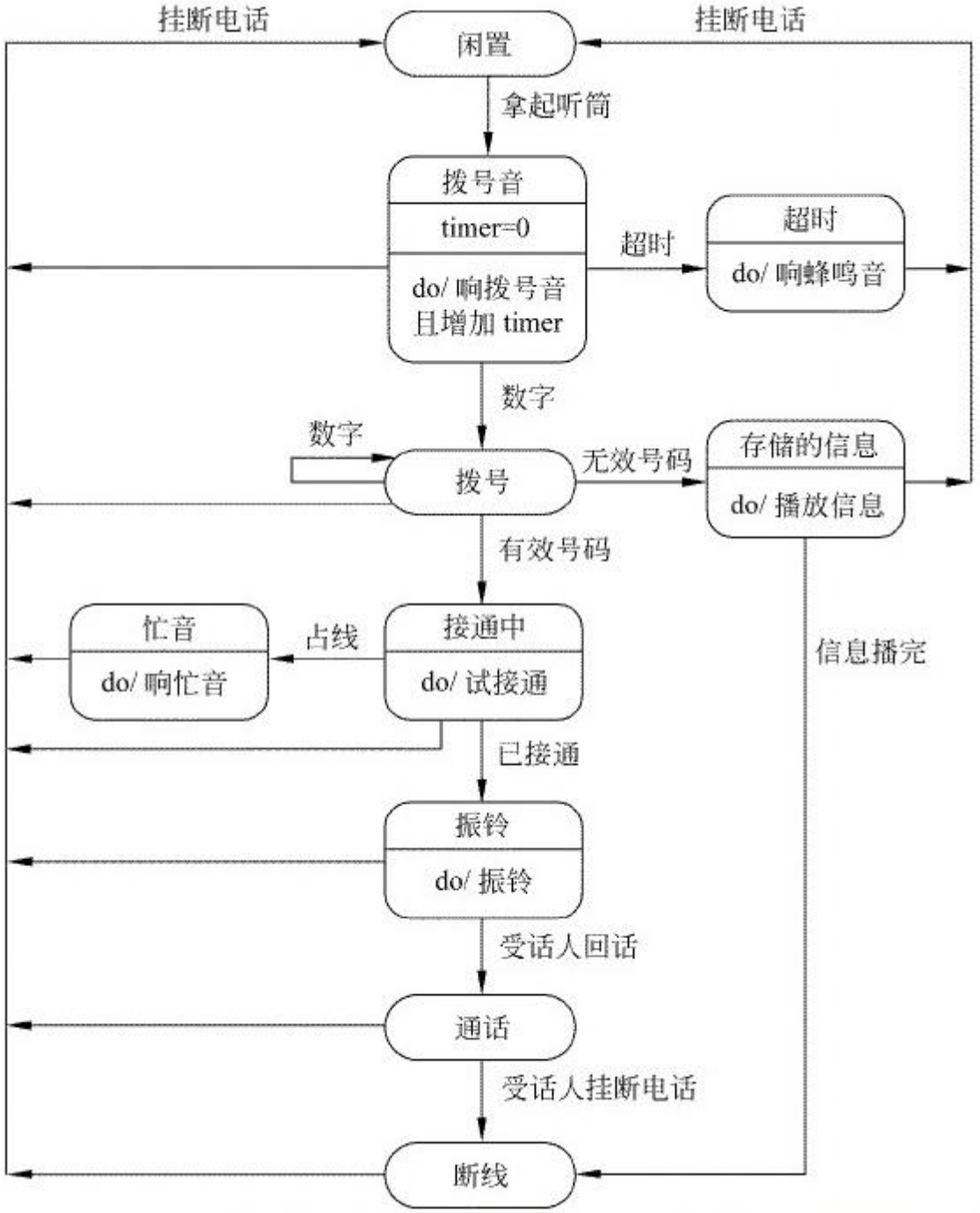
- 在活动表中经常使用下述3种标准事件： entry、exit和do。
  - entry事件指定进入该状态的动作
  - exit事件指定退出该状态的动作
  - do事件指定在该状态下的动作
- 需要时可以为事件指定参数表。
- 动作表达式描述应做的具体动作。

# 3.6 状态转换图




- 两个状态之间带箭头的连线称为**状态转换**。箭头指明转换方向。
- 状态变迁通常是由事件触发的，应在表示状态转换的箭头线上标出触发转换的**事件表达式**。
  - 事件表达式的语法：**事件说明[守卫条件]/动作表达式**
  - 事件说明的语法为：事件名(参数表)
- 如果在箭头线上未标明事件，则表示在源状态的内部活动执行完之后自动触发转换。

# 电话系统的状态图

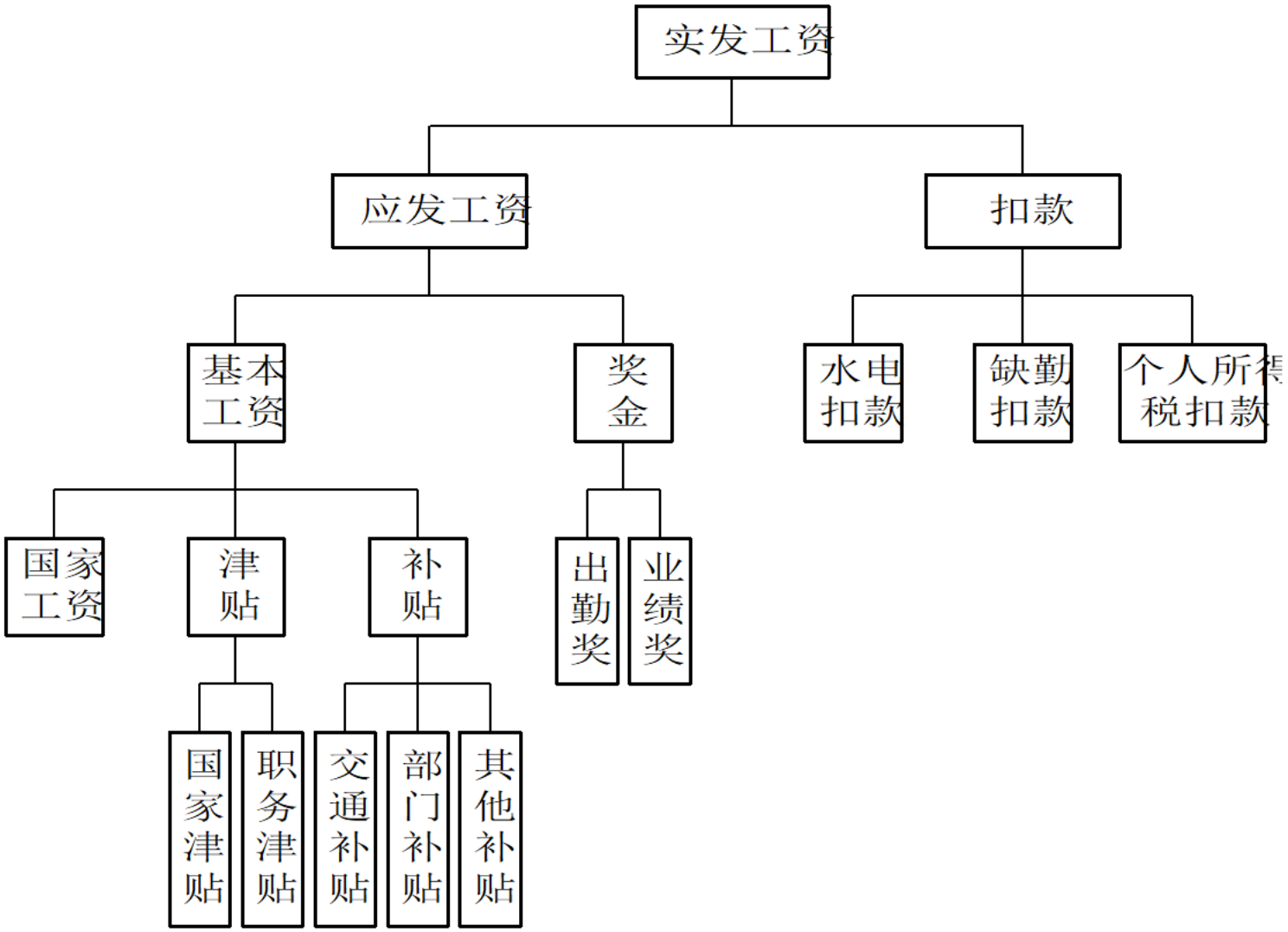

挂断电话
闲置
挂断电话
拿起听筒
拨号音
timer=0
do/ 响拨号音
且增加 timer
超时
超时
do/ 响蜂鸣音
数字
数字
拨号
无效号码
存储的信息
do/ 播放信息
有效号码
忙音
do/ 响忙音
占线
接通中
do/ 试接通
信息播完
已接通
振铃
do/ 振铃
受话人回话
通话
受话人挂断电话
断线

# 3.7 其他图形工具

- **层次方框图**用树形结构的一系列多层次的矩形框描述复杂数据的层次结构。
  - 树型结构**顶端只有一个矩形框**，代表完整的数据结构。
  - 下面各层的矩形框是对完整数据结构的逐步分解和细化得到的数据子集。
  - 最底层的矩形框代表组成该数据结构的基本元素(不能再分割的元素)。

**建立方法：**从对顶层信息的分类开始，沿着层次图中的每条路径逐步细化，直到确定了数据结构的全部细节为止。

# 3.7 其他图形工具




某单位职工实发工资的层次方框图

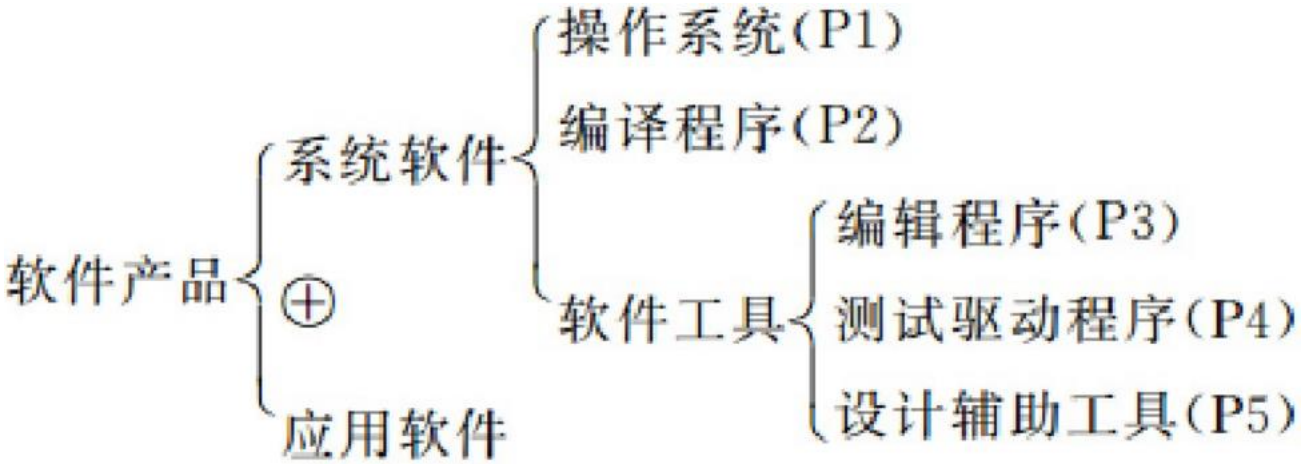
# 3.7 其他图形工具

- Warnier图与层次方框图类似，但提供了更丰富的描绘手段，可更清楚地描述信息的逻辑组织。
  - 可表明数据的逻辑结构中某类信息的重复出现。
  - 表明数据的逻辑结构中某些特定信息出现的条件约束。
- Warnier图的基本符号
  - 大括号“{”：区分数据结构的层次。
    - 一个大括号内的所有名字都属于同一类信息。
  - 异或符号“⊕”
    - 表明一类信息或一个数据元素在一定条件下出现，而且在这个符号上、下方的两个名字所代表的数据只能出现一个。

# 3.7 其他图形工具

## ■ 圆括号“( )”

- 在一个名字下面或右边的圆括号中出现的数字指明了这个名字所代表的信息类或数据元素在该数据结构中重复出现的次数。




**描绘一种软件产品的Warnier图**

(1) 某计算机公司的一种软件产品要么是系统软件，要么是应用软件；

(2) 系统软件中有P1种操作系统、P2种编译程序，还有软件工具；

(3) 软件工具是系统软件的一种，它进一步又可划分为编辑程序、测试驱动程序和设计辅助工具，它们各自的数量分别为P3、P4和P5。

# 3.8 验证软件需求

- 对系统的需求必须严格地进行验证，以保证这些需求的正确性。
  - 大量统计数字表明，软件系统中有大约15%的错误起源于错误的需求。
- 需求验证的目的：
  - 提高软件质量，降低软件开发成本，确保软件开发的顺利进行。

需求验证一般应从下述四个方面进行

# 3.8 验证软件需求

## (1) 验证需求的一致性

- 所有需求必须是一致的，任何一条需求不能和其他需求互相矛盾。
  - 当需求分析的结果是用非形式化的方法，如自然语言书写的时候，除了靠人工审查、验证软件需求规格说明书的正确性之外，目前还没有其他更好的方法。
  - 当软件需求规格说明书是用形式化的需求描述语言书写的时候，可以用软件工具来验证需求的一致性。

# 3.8 验证软件需求

## (2) 验证需求的完整性

- 需求必须是完整的，需求规格说明书中应包括用户需求的每一个功能或性能。

需求的完整性常常难以保证

- 原因：用户并不能清楚地认识到他们的需求，或不能有效地表达他们的需求。大多数用户只有在面对目标软件系统时，才能完整、确切地表述他们的需求。
- 解决：需要开发人员与用户双方的充分配合和沟通，加强用户对需求的确认和评审，尽早发现需求中的遗漏。

# 3.8 验证软件需求

## (3) 验证需求的有效性

- 必须证明需求是正确有效的，确实能够解决用户面对的问题。

需求的有效性只有在用户的密切配合下才能完成

- 原因：只有目标系统的用户才能真正知道软件需求规格说明书是否准确地描述了他们的需求。

# 3.8 验证软件需求

## (4) 验证需求的现实性

- 需求在现有硬件和软件技术水平上应该是能够实现的。

- 应该参照以往开发类似系统的经验，分析采用现有的软、硬件技术实现目标系统的可能性，必要的时候可以通过仿真或性能模拟技术来辅助分析需求的现实性。

设置

结构化分析方法就是面向（　　）自顶向下逐步求精进行需求分析的方法。

A 目标

B 功能

C 对象

D 数据流

提交

设置

数据字典是软件需求分析阶段的最重要工具之一，其最基本的功能是

A 数据通信

B 数据定义

C 数据维护

D 数据库设计

提交

# 单选题 1分

设置

## 下列各组用例之间存在扩展关系的是

A ATM提款与登录

B 借书与还书

C 购买商品与查找商品

D 预订机票与网上预订机票

提交

单选题 1分

设置

（　　）是从用户使用系统的角度描述系统功能的图形表达方法。

A 类图

B 用例图

C 序列图

D 协作图

提交

设置

以下不属于需求分析的工具模型是

A 状态图

B 程序流程图

C 用例图

D 数据流图

提交

设置

需求分析阶段开发人员要从用户那里了解

A 输入的信息

B 用户使用界面

C 软件的规模

D 软件要做什么

提交

单选题 1分

设置

面向对象的类层次结构中，聚合关系是一种

A “一般——特殊”关系

B “整体——部分”关系

C “相互依赖”关系

D “一般——具体”关系

提交

5.1设计过程

5.2设计原理

5.3启发规则

5.4描绘软件结构的图形工具

5.5面向数据流的设计方法

5.6面向对象的设计方法

- 总体设计的基本目的就是回答“概括地说，系统应该如何实现”这个问题，因此，总体设计又称为概要设计或初步设计。
- 总体设计阶段的一项重要任务是设计软件结构，也就是确定系统中每个程序由哪些模块组成的以及这些模块相互间的关系。还有接口设计和数据设计。

**软件需求分析转化为软件的内部结构**

# 5.1 设计过程

## 1.设想供选择的方案

应考虑各种可能的实现方案，从中选出最佳方案。

## 2.选取合理的方案

从前一步得到的方案中选取若干合理的方案，通常选取低成本、中等成本和高成本的3种方案。

## 3.推荐最佳方案

## 4.功能分解

从实现角度把复杂的功能进一步分解。如对DFD进一步细化。

# 5.1 设计过程

## 5.设计软件结构

通常一个模块完成一个适当的子功能。应该把模块组织成良好的层次系统，每个模块调用其下层模块实现完整功能。最下层的模块完成最具体的功能。

## 6.设计数据库

## 7.制定测试计划

## 8.书写文档

## 9.复查和复审

# 5.2 设计原理

## 一、模块化

- **模块**：一组有序操作的总称，通过一个总体标识符代表它。**模块是构成程序的基本构件**。
- **模块化**：把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来，并通过模块间的调用关系把它们组成一个完整的整体，完成指定的功能。

# 5.2 设计原理

## 一、模块化

### □ 模块化的依据

■ **使问题复杂度降低，易实现易理解。**

设函数C(*x*)定义问题*x*的复杂程度，函数E(*x*)定义解决问题*x*需要的工作量(时间)，对于两个问题$P_1$和$P_2$，

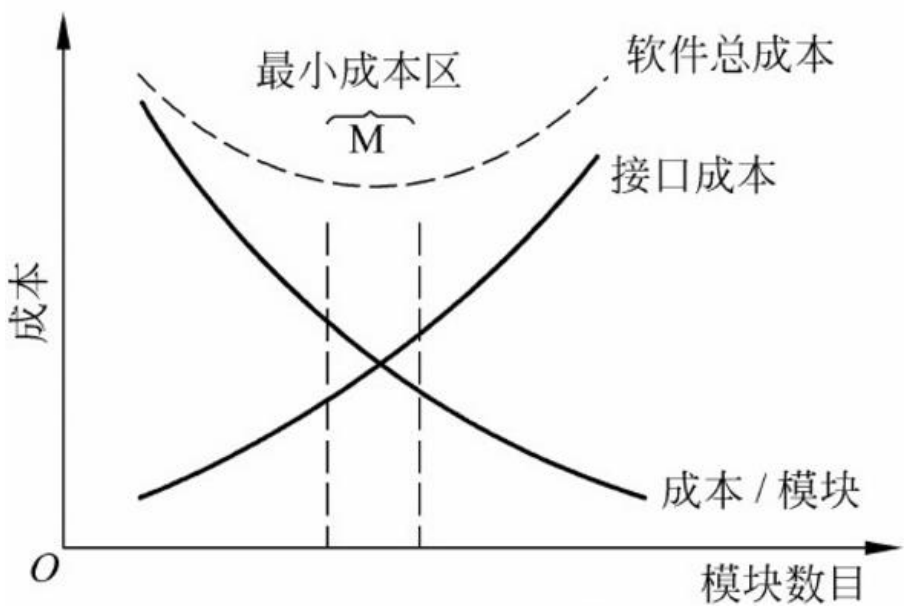
如果：　$C(P_1) > C(P_2)$　　则：　$E(P_1) > E(P_2)$

- 一个有趣的规律：多个问题复合而成的大问题的复杂度大于分别考虑各问题的复杂度之和。即：C(P1+P2)> C(P1)+C(P2)
- 将复杂问题分解成若干小问题，各个击破，所需的工作量小于直接解决复杂问题所需的工作量。即：E(P1+P2)> E(P1)+E(P2)

# 5.2 设计原理

**如果无限制地划分软件，那开发所需的工作量可变的非常小?**

- **模块数目增加时每个模块的规模将减小，开发单个模块的工作量也将减少。但同时增加了设计模块接口的工作量。**




**模块数与软件开发成本的关系图**

- 当划分的模块数处于最小成本区时，开发软件的总成本最低。
- 目前还不能得到模块数M的精确取值

## 二、抽象

- 抽象是指将现实世界中具有共性的一类事物的相似的、本质的方面集中概括起来，而暂时忽略它们之间的细节差异。
- 软件工程过程的每一步都是对软件解法的抽象层次的一次求精。
  - 软件结构中顶层的模块抽象级别最高，控制并协调软件的主要功能且影响全局。
  - 软件结构中位于底层的模块抽象级别最低，具体实现数据的处理过程。

## 三、逐步求精

- 为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。
- 逐步求精最初是自顶向下的设计策略，程序的体系结构通过逐步精化处理过程的层次设计出来。
  - 求精实际上是细化过程。
  - 抽象与求精是一对互补的概念。

## 四、信息隐藏

- **信息隐藏**：模块内部的信息（处理过程和数据）对于不需要这些信息的模块来说是不能访问。
  - **提高模块的独立性**，减少将一个模块中的错误扩散到其他模块的机会。但并不意味着某个模块的内部信息对其他模块来说完全不可见或不能使用，而是说模块之间的信息传递只能通过**合法的调用接口**来实现。

## 五、模块独立

- 模块独立的概念是模块化、抽象、信息隐蔽概念的直接结果。
- 开发具有独立功能且与其他模块之间没有过多的相互作用的模块，就可以做到模块独立。
- 模块独立程度的度量标准:

  ■**耦合**：衡量**不同模块之间**相互依赖程度的度量指标。

  ■**内聚**：衡量一个**模块内部**各元素相互依赖程度的度量指标

# 耦合

- 耦合强弱取决于模块之间接口的复杂程度，调用模块的方式，以及通过接口的数据。实际上，耦合是接口数据对模块独立性的影响。
- 如果两个模块中的每一个都能独立地工作而不需要另一个模块地存在，那么它们彼此完全独立，这意味着模块间无任何连接，耦合程度最低。但一个软件系统中的模块之间是彼此协同工作的，不可能所有模块间没有连接。

# 耦合

## （1）数据耦合

- 如果两个模块彼此间通过**参数**交换信息，而且交换的信息仅仅是**数据**，则称这种耦合为数据耦合。

```
int Sum(int a, int b)
{
    return(a+b);
}
```

```
main( )
{ int x, y;……
 printf ("x+y=%d“ ,sum(x, y));
}
```

数据耦合是低耦合。系统必须存在这种耦合，因为只有当某些模块的输出数据作为另一些模块的输入数据时，系统才能完成有价值的功能。

# 耦合

## (2) 控制耦合

- 若模块之间交换的信息中包含有**控制信息**(尽管有时控制信息是以数据的形式出现的)，则称这种耦合为控制耦合。
- **控制耦合是中等程度的耦合**，它会增加程序的复杂性。

```
void output ( flag )
{ if ( flag ) printf ("OK! ");
   else printf ("NO! ");   }
main()
{ int flag;……
  Output ( flag );
  }
```

# 耦合

## (3) 特征耦合

- 当把整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时，就出现了特征耦合。
- 这种耦合可能会带来一些问题：

① 不该修改的数据可能会被不小心修改了

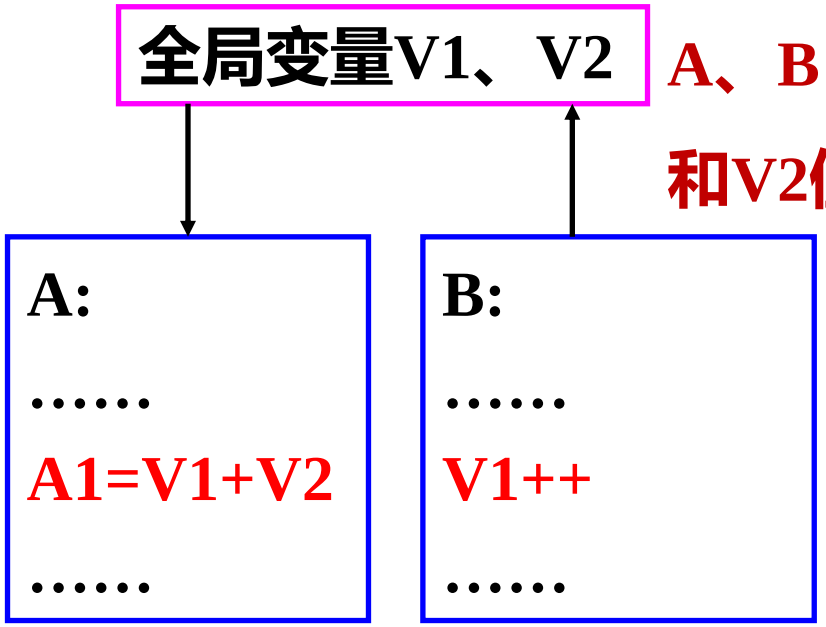
② 没有权利接触某些数据的人也能修改

  - 如计算水电费模块输入参数是住户情况
  - 可将参数修改为用水量和用电量

出于安全考虑，应该只传递那些必须的数据项

## (4) 公共耦合

- 两个或多个模块通过引用公共数据相互联系，则称这种耦合为公共耦合。
  - 例如，在多个模块中对全局变量进行了读写。




A、B 两个模块都能使用和改变V1和V2值，若发生错误，错误定位难

公共耦合的复杂度随耦合的模块个数的增加而显著增加

## (5) 内容耦合

- 若出现下列情况之一，两个模块间就发生了内容耦合
  - 一个模块访问另一模块的内部数据
  - 一个模块不通过正常入口而转到另一模块内部
  - 两个模块有一部分程序代码重叠
  - 一个模块具有多个入口(意味着模块有几种功能)
- **内容耦合是所有耦合关系中程度最高的。**

原则：尽量使用数据耦合，少用控制耦合和特征耦合，限制公共耦合的范围，完全不用内容耦合。

# 内聚

- 内聚标志着模块内各个元素彼此结合的紧密程度。内聚是信息隐藏功能的自然扩展，是模块内部功能独立性的表现。**理想内聚的模块只做一件事情**。
- 内聚和耦合是密切相关的，模块内的**高内聚**往往意味着模块间的**松耦合**。
- 实践表明内聚更重要，应把更多的注意力集中到提高模块的内聚程度上。
- 按程度分类：**低内聚 中内聚 高内聚**。

# 内聚

## (1) 低内聚

- **一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。**
  - 如程序中多处出现一些无联系的语句段序列，为节省内存空间将其组合成为一个模块。

- 偶然内聚的模块由于组成部分之间没有实质的联系，因此难于理解和修改，会给软件开发带来很大的困扰。
- 偶然内聚的模块出错的机率要比其他类型的模块大得多。
- 偶然内聚是内聚程度最低的一种，应尽量避免。

# 内聚

## (1) 低内聚

- **一个模块完成的任务在逻辑上属相同或相似的一类称为逻辑内聚。**
  - 如一个模块根据传入的控制标志打印季、月或日报告。
- **一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。**
  - 如多个变量的初始化放在同一个模块中实现。

# 内聚

## (2) 中内聚

- **一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。**
  - 如一个模块读入并审查成绩单。
- **模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚。**
  - 如读写同一个数据表。

# 内聚

## (2) 高内聚

- 一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行(通常前一部分的输出作为后一部分的输入)，则称为顺序内聚。
- 模块内所有处理元素属于一个整体，完成一个单一的功能，称为功能内聚，是最高程度的内聚。

原则：设计时高内聚，并能辨认低内聚的模块，有能力通过修改设计提高模块的内聚程度并降低模块间的耦合程度，从而获得较高的模块独立性。

# 5.3 启发规则

从长期的软件开发实践中，总结出来的规则。

## 一、改进软件结构提高模块独立性

- 设计出软件的初步结构以后，应审查分析这个结构，通过模块分解或合并，力求**降低耦合提高内聚**。
  - 如多个模块公用的子功能可独立成一个模块，供这些模块调用

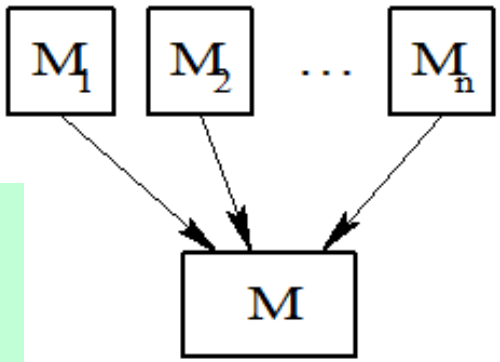
## 二、模块规模应该适中

- 一个模块的规模不应过大，最好能写在一页纸内(通常不超过60行语句)。

# 5.3 启发规则

## 三、深度、宽度、扇出和扇入都应适当

- 深度：软件结构中控制的层数，粗略地标示一个系统的大小和复杂程度。
- 宽度：软件结构内同一个层次上模块总数的最大值，一般来说，宽度越大的系统越复杂。
- 扇入：一个模块被多少个上级模块直接调用的数目。
  - 扇入越大说明共享该模块的上级模块越多
  - 不违背模块独立性的条件下，高扇入好。

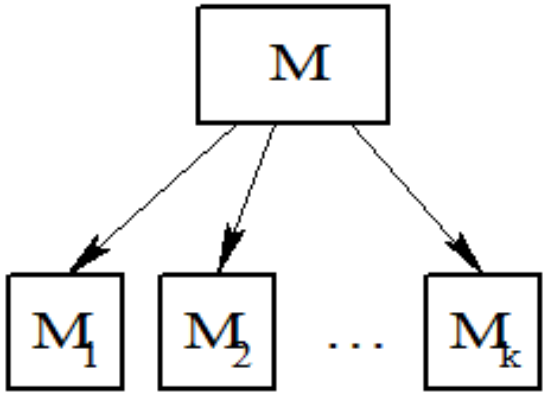



当多个模块具有一部分相同功能时，可将这部分相同功能分离出来，编写成独立的模块供需要的模块调用。

# 5.3 启发规则

## 三、深度、宽度、扇出和扇入都应适当

- 扇出：一个模块直接控制(调用)的下级模块数目
  - 扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块。可适当增加中间层次。
  - 扇出过小也不好，可考虑将其合并到上级模块中



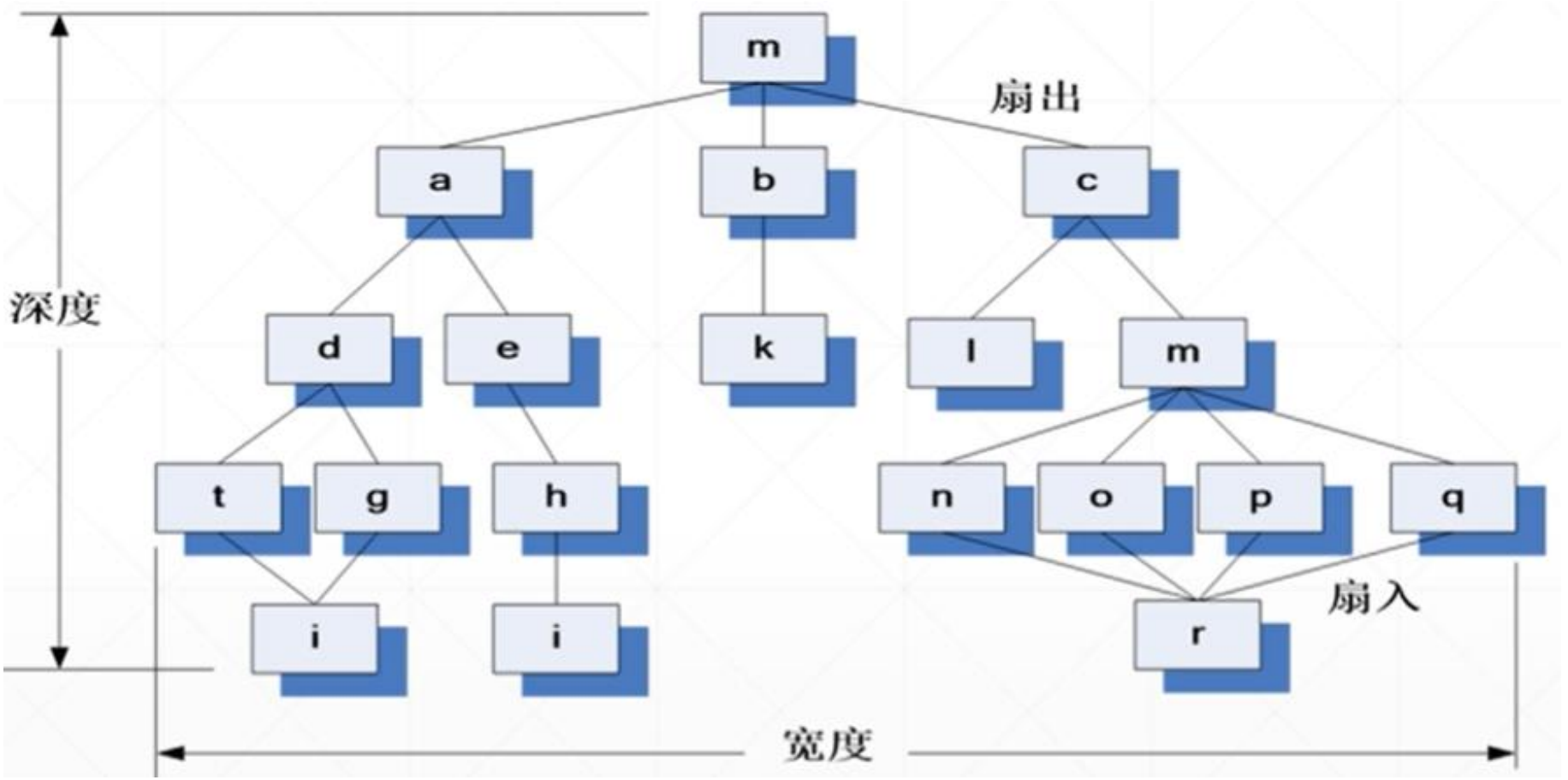

**一个好的系统的平均扇出通常是3或者4**(上限5-9)

## 三、深度、宽度、扇出和扇入都应适当

- 深度宽度分析：
  - **影响深度的因素**：深度在程序中表现为模块的嵌套调用，嵌套层数越多，程序就越复杂，程序的可理解性也随之下降。深度过大可将结构中过于简单的模块与上一级模块合并。
  - **影响宽度的因素**：对宽度影响最大的因素是模块的**扇出**，即模块可以调用的下级模块数越多，软件结构的宽度就越大。宽度过大则可通过增加中间层来解决。
  - **深度和宽度是相互对立的两个方面**，降低深度会引起宽度的增加，而降低宽度又会带来深度的增加。

# 5.3 启发规则

## 三、深度、宽度、扇出和扇入都应适当



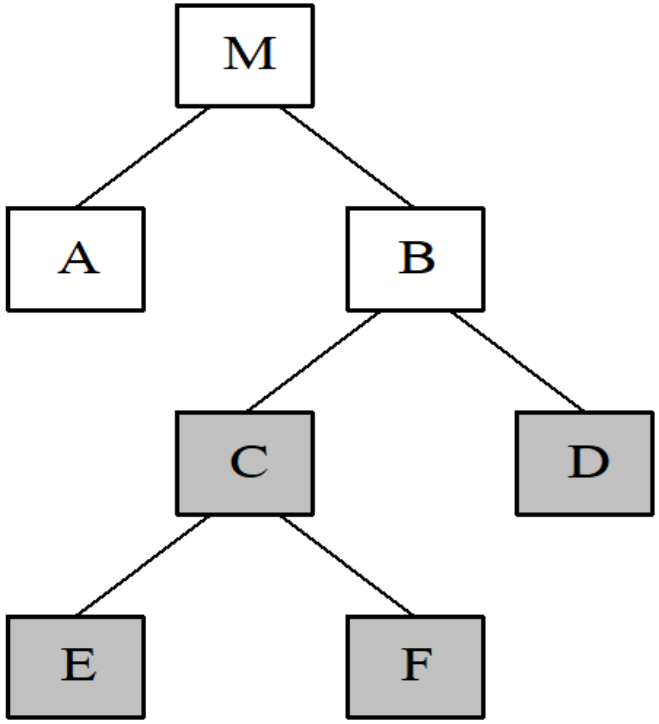

**顶层扇出比较高，中层扇出较少，底层高扇入**

# 5.3 启发规则

## 四、模块的作用域应该在控制域范围之内

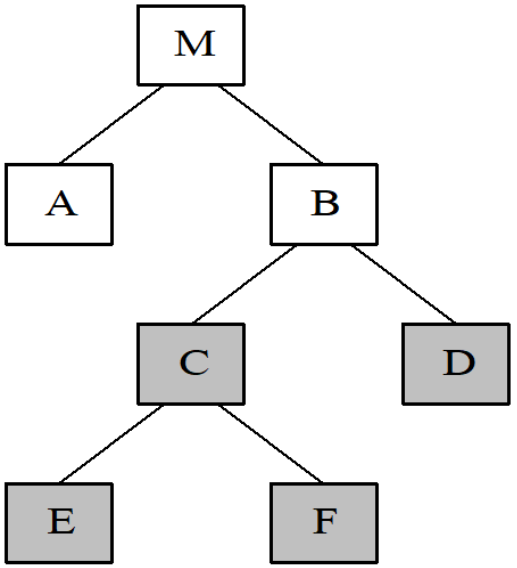
- **作用域**：受该模块内一个判定影响的所有模块范围
- **控制域**：模块本身及所有直接或间接从属于它的模块

- 若在模块C中存在对D、E和F均有影响的判定条件，则模块C的**作用域**：
  - C、D、E、F
- 模块C的**控制域**：
  - C、E、F
- 显然**模块C的作用域超出了其控制域**。



# 5.3 启发规则

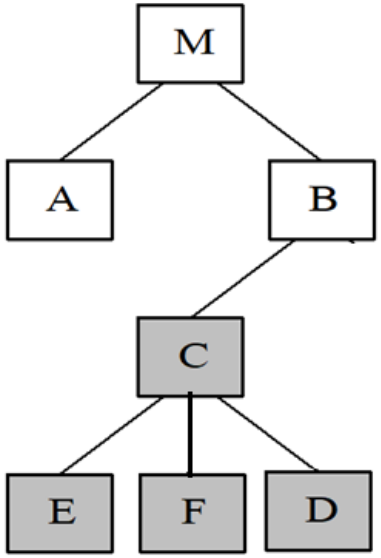
- D在C的作用域中会**增加模块间耦合性**，因为C对D的控制信息必然要通过上级模块B进行传递。




- 若发现不符合此原则的模块，可通过下面的方法改进：

① 将判定位置上移。

② 将超出作用域的模块下移。



## 五、力争降低模块接口的复杂程度

- 模块接口设计原则：易理解，传递信息简单且与模块功能一致。

例：求一元二次方程Ax^2+Bx+C=0的根：

Quad_Root(Tal,x); //复杂，不易理解

Tal--系数数组

x--根数组

Quad_Root(a,b,c,Root1,Root2);//简单，便于理解

## 六、设计单入口单出口的模块

- 不要使模块间出现内容耦合。当从顶部进入模块并且从底部退出来时，软件是比较容易理解的，因此也是比较容易维护的。

## 七、模块功能应该可以预测

- 功能可以预测：相同的输入应产生相同的输出，
- 功能不可预测：模块带有内部状态，输出取决于该状态。