西南大學

软件工程

Software Engineering

吴 浪

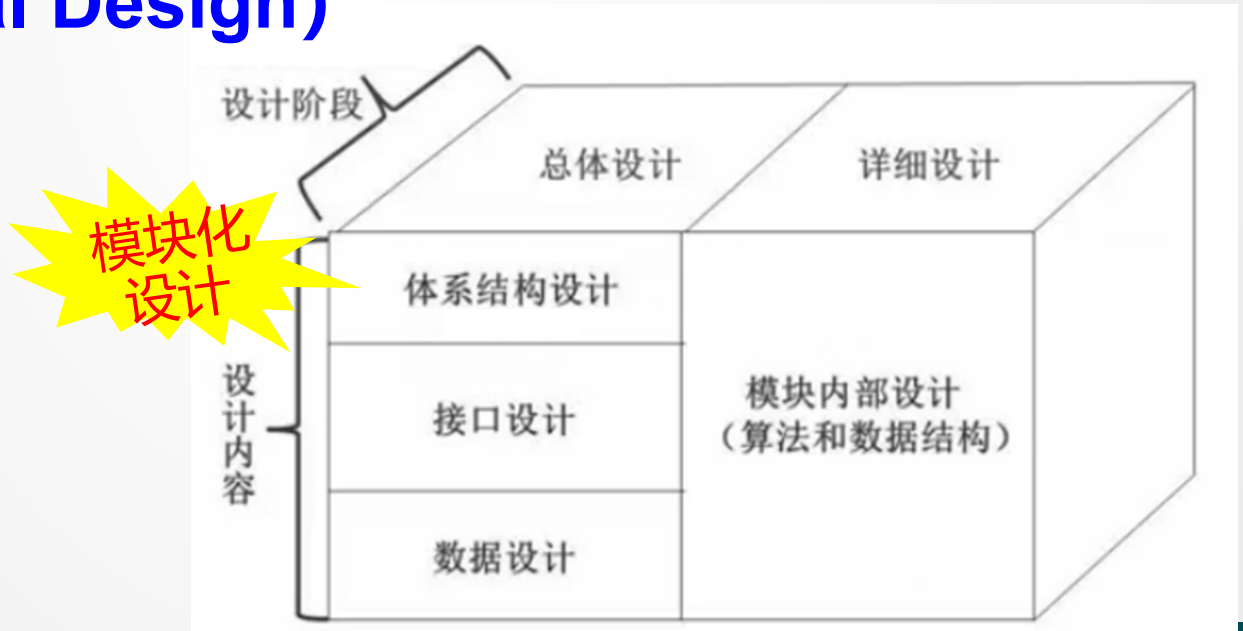
西南大学 计算机与信息科学学院

Email：wl0776@swu.edu.cn

引言 结构化设计（SD）

- 设计：作为**软件开发**的一种活动，定义**实现需求建模（WHAT）**所需要的软件结构——回答如何解决问题——给出**软件解决方案（HOW）**。

- **SD（Structural Design）分为两个阶段**：
  - 总体设计（概要设计/初步设计）
  - 详细设计



SD设计阶段和设计内容

西南大學

第五章 总体设计

教学目标

1. 识记和理解总体设计的主要任务和参与人员，以及阶段性的成果。
2. 理解软件总体设计的基本过程。
3. 识记和理解软件设计若干基本原理的含义，如模块化、抽象、逐步求精、信息隐藏和局部化等。
4. 识记和理解模块独立性的含义，即内聚和耦合是模块独立性的两个衡量指标，内聚度越高、耦合度越低，模块独立性越强。
5. 识记和理解内聚的含义，能够区分不同程度的内聚。
6. 识记和理解耦合的含义，能够区分不同程度的耦合。
7. 识记和理解模块设计的七条启发式规则。
8. 识记和理解扇入、扇出基本概念的含义。
9. 识记和理解模块作用域和模块控制域的含义。
10. 熟练掌握层次图和 HIP0 绘制系统结构的作图方法。
11. 熟练掌握结构图中各种图形符号的含义和基本的作图方法。
12. 识记并理解面向数据流设计的相关基本概念。
13. 熟练掌握面向数据流(变换流和事务流)的系统结构设计方法。
14. 理解数据库设计的基本原理，并能够对小型数据库进行设计。

总体设计任务

- 总体设计的基本目的是回答“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。
- 总体设计阶段的另一项重要任务是，也就是要确定系统中每个程序是由哪些模块组成的，设计软件的结构以及这些模块相互间的关系。

- **系统方案设计**

  划分出组成系统的物理元素——程序、文件、数据库、人工过程和文档等等，但是每个物理元素仍然处于黑盒子级。

- **体系结构设计**

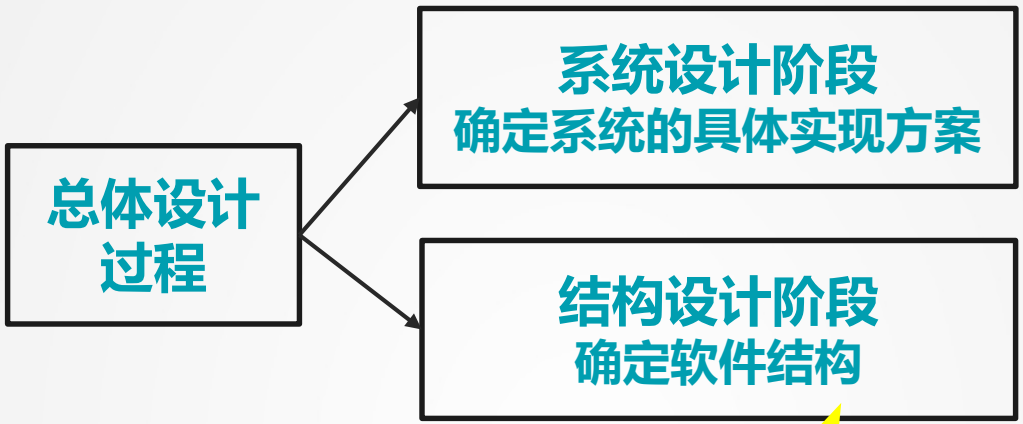
  设计软件的结构，确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系。

西南大學

目录
CONTENTS

西南大學

目录
CONTENTS



总体设计过程的九个步骤

（一）步骤1

- 设想供选择的方案
  - 逻辑模型→各种可能的实现方案
  - 划分数据流图的自动化边界

需求分析阶段得出的数据流图是总体设计的极好的出发点。

设想供选择的方案的一种常用的方法是，设想把数据流图中的处理分组的各种可能的方法，抛弃在技术上行不通的分组方法(例如，组内不同处理的执行时间不相容)，余下的分组方法代表可能的实现策略，并且可以启示供选择的物理系统。

（二）步骤2

- 选取合理的方案
  - 至少选取低成本、中等成本和高成本三种方案
  - 每种方案准备四份资料：
    - 系统流程图
    - 组成系统的物理元素清单
    - 成本/效益分析
    - 实现这个系统的进度计划

（三）步骤3

- 推荐最佳方案
  - 综合分析对比各种合理方案的利弊，推荐一个最佳的方案，并且为推荐的方案制定详细的实现计划

用户和有关的技术专家应该认真审查分析员所推荐的最佳系统，如果该系统确实符合用户的需要，并且是在现有条件下完全能够实现的，则应该提请使用部门负责人进一步审批。审批之后进入下一个阶段——结构设计。

（四）步骤4

- 功能分解
  - 结构设计
    - 确定程序由哪些模块组成，以及这些模块之间的关系
    - 结构设计是总体设计阶段的任务
  - 过程设计
    - 确定每个模块的处理过程
    - 过程设计是详细设计阶段的任务

（五）步骤5

- 设计软件结构
  - 把模块组织成良好的层次系统
  - 模块与其子模块的调用关系
  - 层次图或结构图

软件结构（即由模块组成的层次系统）可以用层次图或结构图来描绘。

——第5.4节 图形工具

如果数据流图已经细化到适当的层次，可以直接从数据流图映射出软件结构。

——第5.5节 面向数据流的设计方法

（六）步骤6、7

- 设计数据库
  - 模式设计
  - 子模式设计
- 制定测试计划
  - 开发早期考虑测试问题，可提高软件的可测试性。

（七）步骤8、9

- 书写文档
  - （1）系统说明
    - 系统流程图、成本 / 效益分析、最佳方案、精化的数据流图、软件结构模块的算法、软件结构、各个模块的算法、模块间的接口关系、……
  - （2）用户手册
    - 修改更正在需求分析阶段产生的初步的用户手册
  - （3）测试计划
    - 测试策略，测试方案，预期的测试结果，测试进度计划等等
  - （4）详细的实现计划
  - （5）数据库设计结果
- 审查和复审
  - 对总体设计的结果进行严格的技术审查，在技术审查通过之后再由客户从管理角度进行复审

西南大学

目录
CONTENTS

（一）设计面临的挑战

- 求解
  - 需求模型主要描述目标软件产品要“做什么”，而设计模型则要回答“如何做”。
- 抉择
  - 对于同一项软件需求，可行的技术实现方案往往有多。明辨各方案优缺点，综合考虑选择。
- 灵巧
  - 软件对需求变化的适应能力主要来源于软件结构的柔韧性，这就要求软件设计师预判未来可能的需求变化，并为此设计软件模块的扩充机制或更新机制。

（二）设计原理

为了应对上述挑战，软件设计必须遵循一些基本的设计原则。

- 模块化
- 抽象
- 逐步求精
- 信息隐藏与局部化
- 模块独立

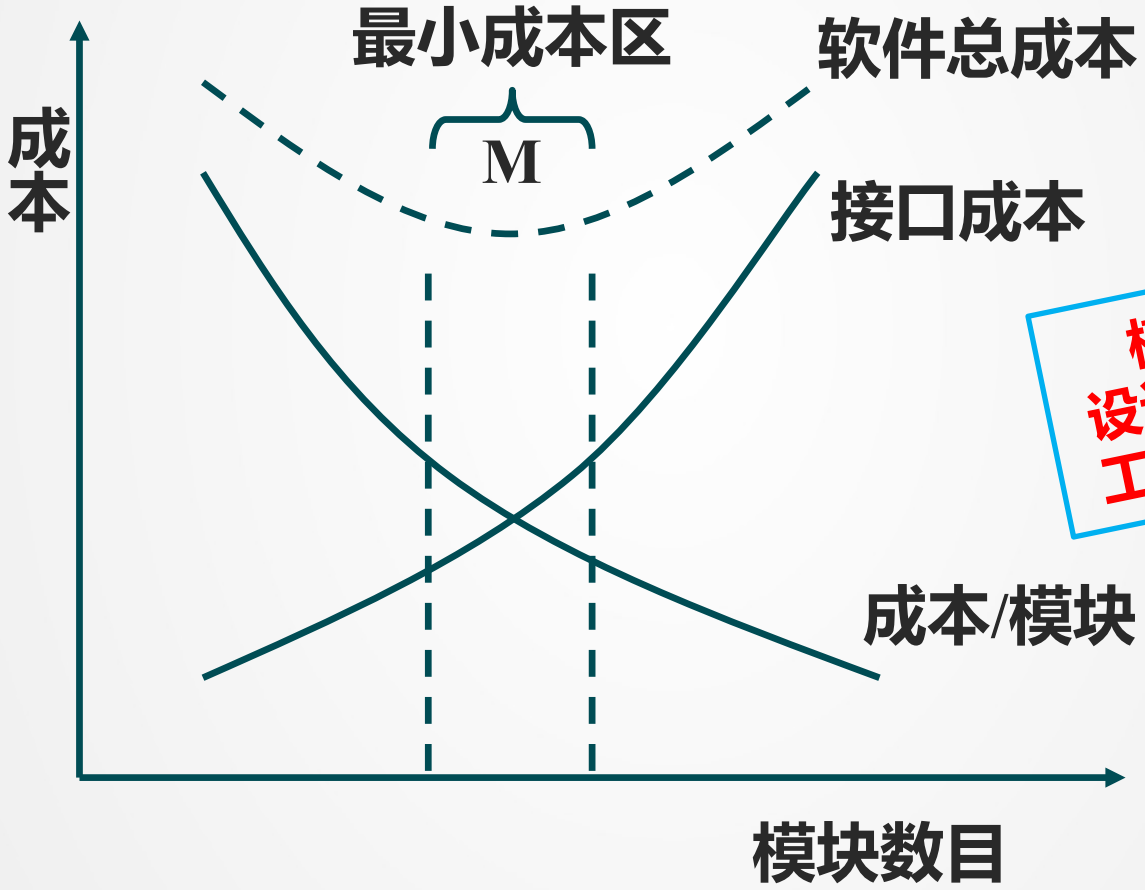
（三）模块化

主要思想是将软件系统划分为若干个相对独立的部件（模块），所有模块组装后可获得满足问题需要的软件解。

- 如果一个大型程序仅由一个模块组成，很难被人理解。
- 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P1和P2，如果：
- C(P1)>C(P2)
- 那么 E(P1)>E(P2)

- 根据解决问题的经验，有一个规律是：
- C(P1+P2)>C(P1)+C(P2)
- 于是有
- E(P1+P2)>E(P1)+E(P2)

（三）模块化



模块数目增加，
设计（模块间接口）
工作量也同步增加！

模块化与软件成本图

（三）模块化——模块和模块化定义

按照模块的定义，过程、函数、子程序和宏等，都可作为模块。
面向对象方法学中的对象是模块，对象内的方法（或称为服务）也是模块。

- **模块（Module）**
  - 由边界元素限定的相邻程序元素（例如，数据说明，可执行的语句）的序列，而且有一个总体标识符代表它。
  - 构成程序的基本构件。
- **模块化（Modularization）**

把系统/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构成一个整体完成指定的功能、
满足用户的需求。

（三）模块化——优点

- 使软件结构清晰，容易设计也容易阅读和理解。
- 提高软件的可靠性。

  程序错误局限在有限的模块中，使软件容易测试和调试。
- 提高了软件的可修改性。

  即使有变动，往往只涉及少数几个模块。
- 有助于软件开发工程的组织管理。

  一个大型程序由许多程序员分工编写，分配技术熟练的程序员编写困难的模块。

（三）抽象

把事务相似的方面集中和概括起来，暂时忽略它们之间的差异。抽象就是抽出事物的本质特性而暂时不考虑它们的细节。

软件设计一般要自顶向下地经历一系列抽象级别从高到低的设计阶段。

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精化。在可行性研究阶段，软件作为系统的一个完整部件；在需求分析期间，软件解法是使用在问题环境内熟悉的方式描述的；当由总体设计向详细设计过渡时，抽象的程度也就随之减少了；最后，当源程序写出来以后，也就达到了抽象的最低层。

（四）逐步求精

逐步求精定义为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。

逐步求精最初是由Niklaus Wirth提出的一种自顶向下的设计策略。按照这种设计策略，程序的体系结构是通过逐步精化处理过程的层次而设计出来的。通过逐步分解对功能的宏观陈述而开发出层次结构，直至最终得出用程序设计语言表达的程序。

- 求精实际上是细化过程。
- 抽象与求精是一对互补的概念。

（五）信息隐藏与局部化

信息隐藏：一个模块内包含的信息(过程和数据)对于不需要这些信息的模块来说，是不能访问的。

局部化：是指把一些关系密切的软件元素物理地放得彼此靠近。

如果在测试期间和以后的软件维护期间需要修改软件，使用信息隐藏原理作为模块化系统设计的标准就会带来极大好处。

（六）模块独立

模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。模块独立：具有独立功能而且和其他模块之间没有过多的相互作用的模块。

模块独立的重要性体现：

①有效的模块化(即具有独立的模块)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。

模块的独立程度可以由两个定性标准度量，这两个标准分别称为内聚和耦合。

（六）模块独立——耦合

耦合是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。

影响耦合强度的因素：

- 一个模块对另一个模块的引用
- 一个模块向另一个模块传递的数据量
- 一个模块施加到另一个模块的控制的数量
- 模块之间接口的复杂程度

（六）模块独立——耦合

耦合的类型	强度
内容耦合	强
公共耦合	↓
控制耦合	↓
标记耦合	↓
数据耦合	弱

在软件设计中应该追求尽可能松散耦合。

对模块的测试或维护时，不需要对系统的其他模块有很多了解。

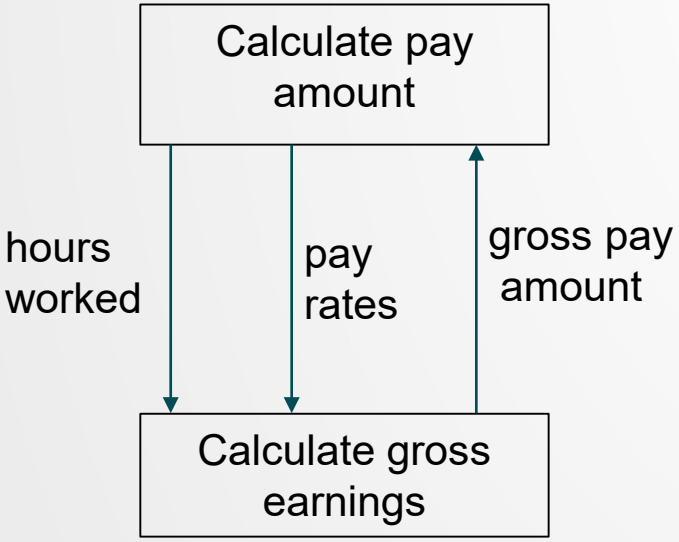
此外，由于模块间联系简单，发生在一处的错误传播到整个系统的可能性就很小。

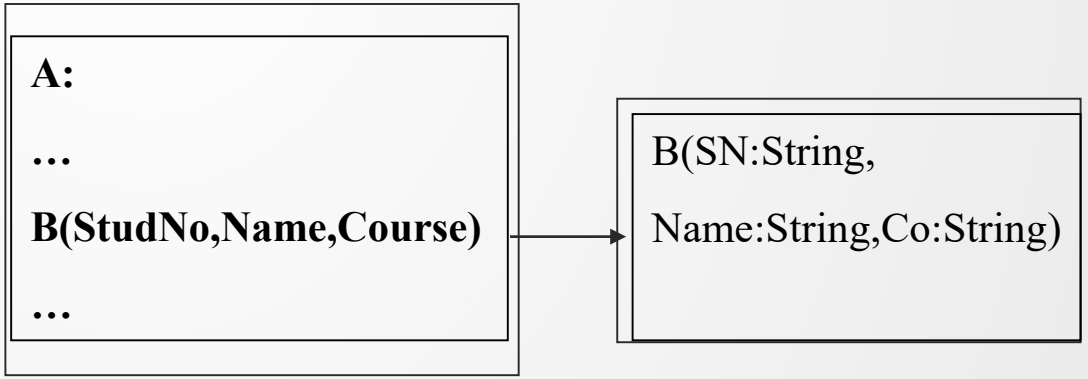
因此，模块间的耦合程度强烈影响系统的可理解性、可测试性、可靠性和可维护性。

（六）模块独立——耦合

①数据耦合

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**，那么这种耦合称为数据耦合。数据耦合是低耦合。系统中至少必须存在这种耦合。

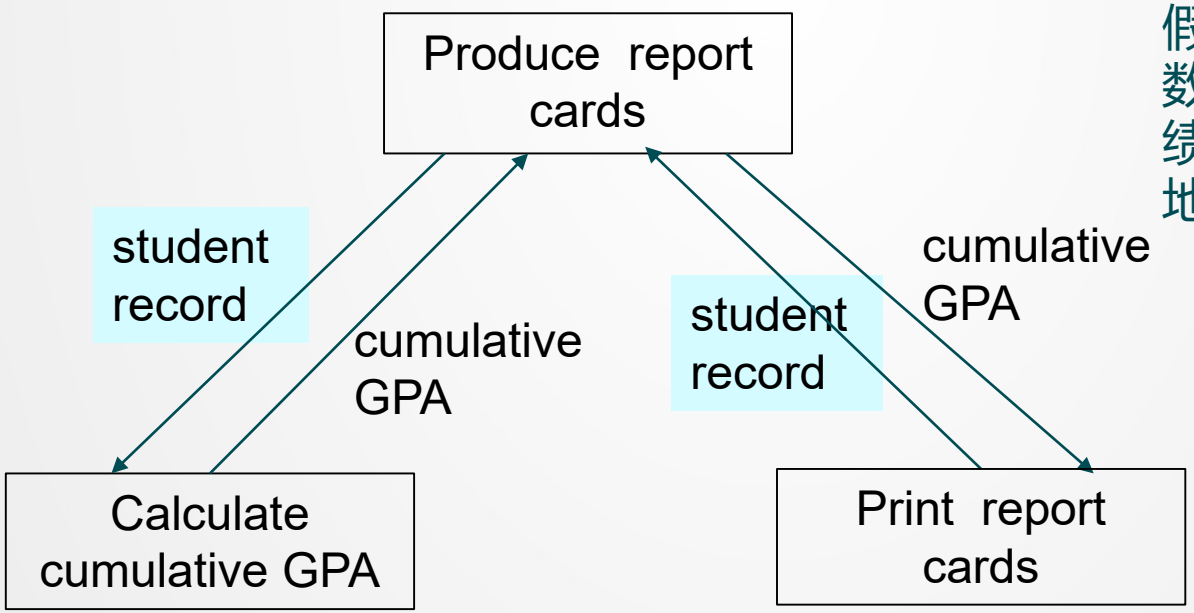




（六）模块独立——耦合

②标记耦合

若两个模块间传递的参数中至少有一个是数据结构，如字符串或记录，并且在模块中仅用到该数据结构中的部分元素，则称这两个模块之间存在标记耦合。



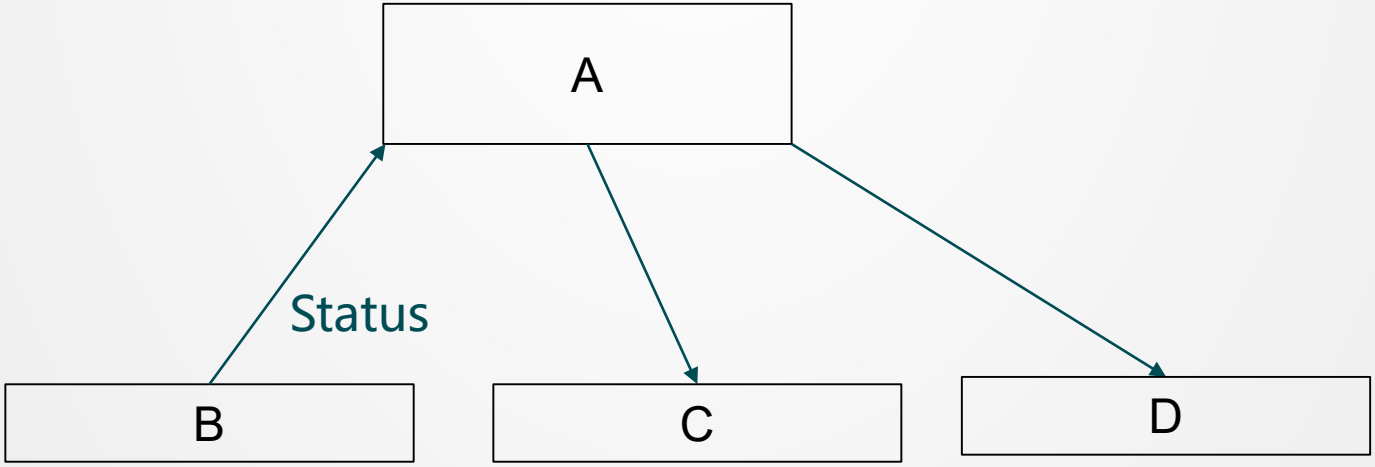
假设“学生记录”是数据结构，除了学习成绩信息，还包含姓名、地址、电话等。

（六）模块独立——耦合

③控制耦合

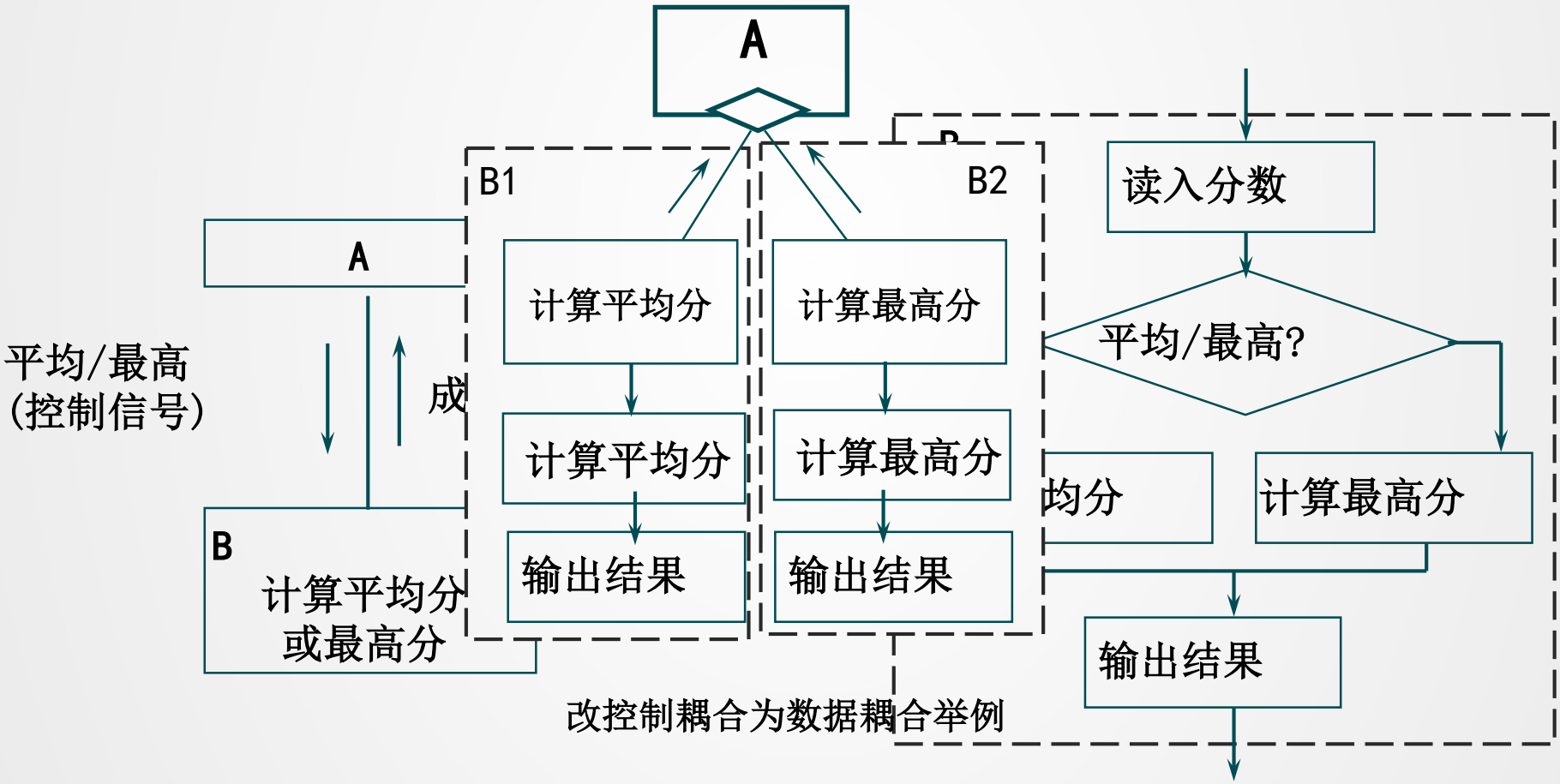
一个模块向另一个模块传递控制信息(尽管有时这种控制信息以数据的形式出现)，接收信息的模块的动作根据信息值进行调整。

控制耦合是中等程度的耦合，它增加了系统的复杂程度。在把模块适当分解之后通常可以用数据耦合代替它。



（六）模块独立——耦合

③控制耦合



（六）模块独立——耦合

④公共耦合

当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。

公共环境可以是全程变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等。

公共环境耦合的复杂程度随耦合的模块个数而变化，当耦合的模块个数增加时复杂程度显著增加。

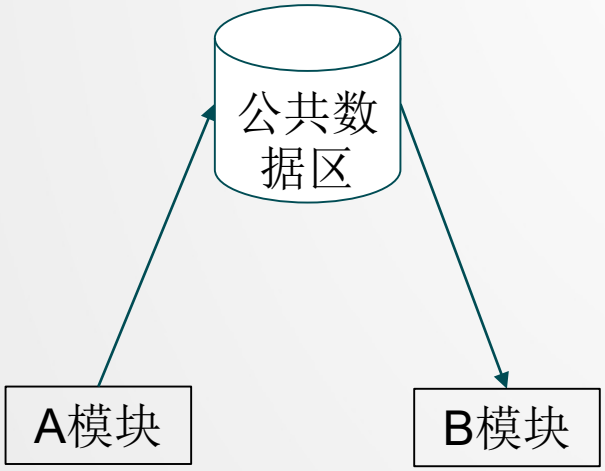
（六）模块独立——耦合

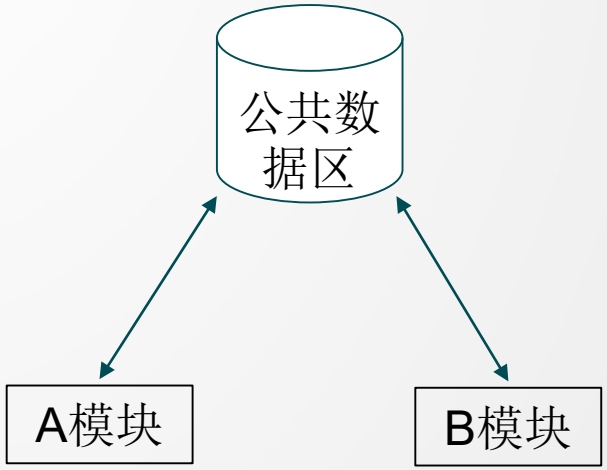
④公共耦合

两个模块的公共耦合有两种可能：

(1) 一个模块往公共环境送数据，另一个模块从公共环境取数据。这是数据耦合的一种形式，是比较松散的耦合。

(2) 两个模块都既往公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。





（六）模块独立——耦合

⑤内容耦合

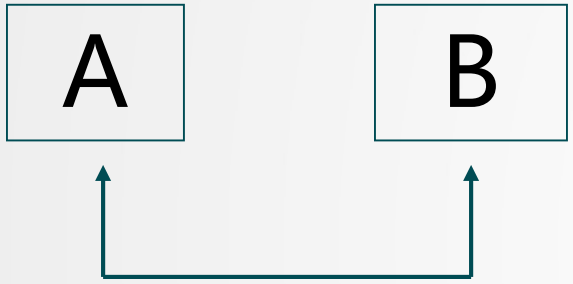
最高程度的耦合是内容耦合。如果出现下列情况之一，两个模块间就发生了内容耦合。

内容耦合情况：

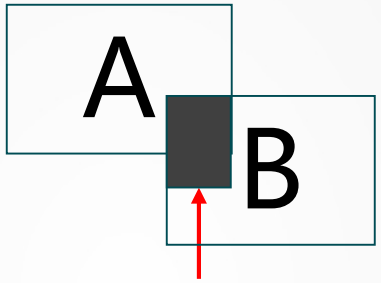
- 一个模块修改另一个模块的语句。（LISP语言可以）
- 一个模块引用或修改另一个模块的内部数据。
- 一个模块不通过正常入口而转到另一个模块的内部。
- 两个模块有一部分程序代码重叠。(只可能出现在汇编程序中)
- 一个模块有多个入口。(这意味着一个模块有几种功能)

应该坚决避免使用内容耦合。

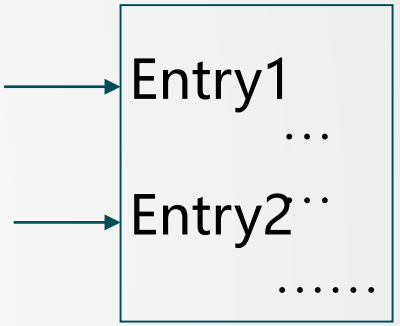
（六）模块独立——耦合



一模块直接访问另一模块的内部信息



模块代码重叠



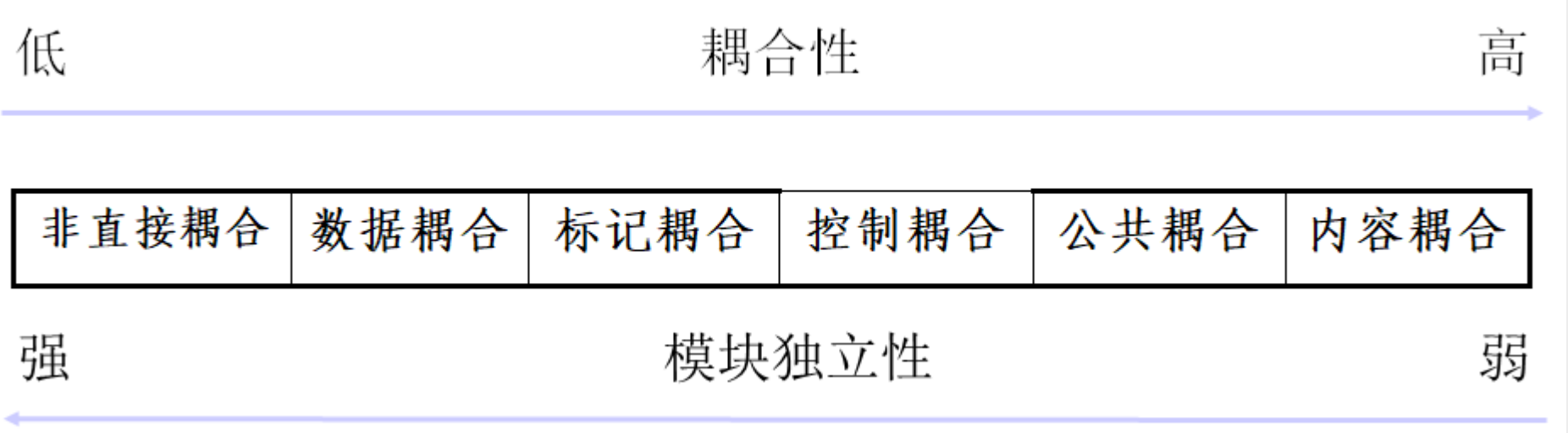
多入口模块

内容耦合的耦合度最高、是病态的模块耦合。

应该坚决避免!!!

（六）模块独立——模块独立与耦合

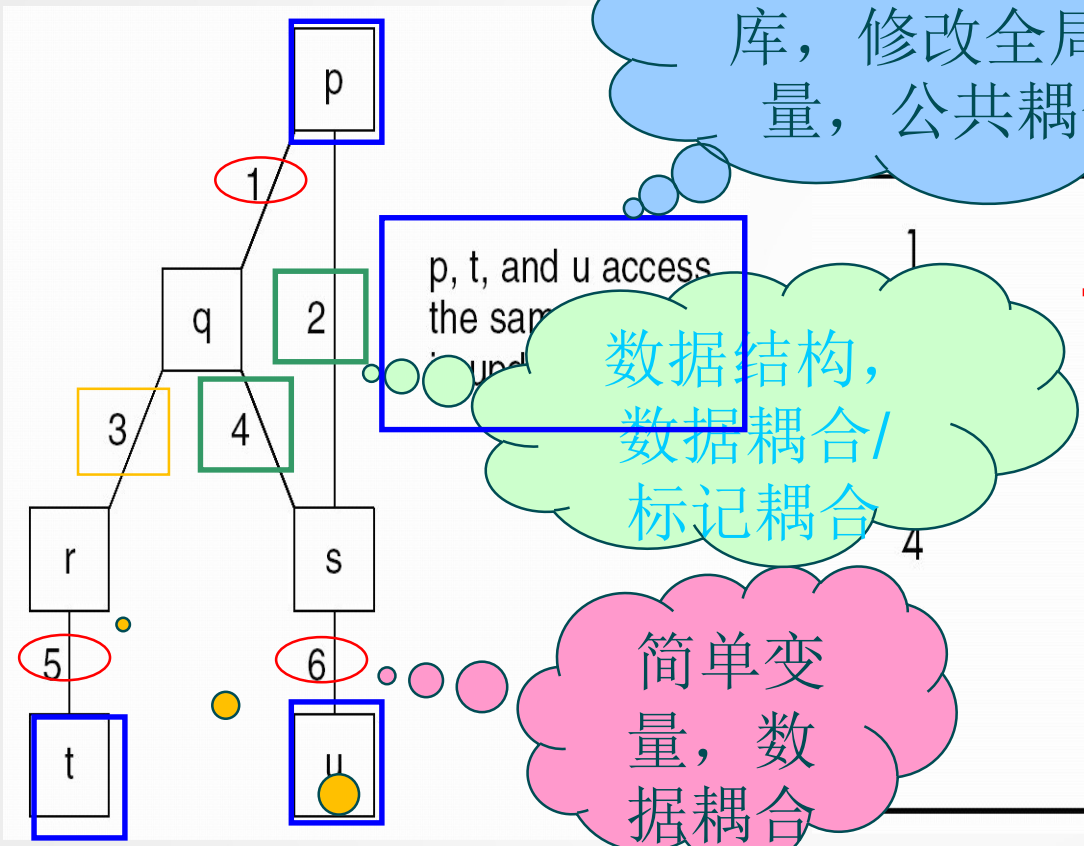
模块独立性与耦合的关系：



*非直接耦合：两模块中任意一个都不依赖另一个而独立工作。

总之，耦合是影响软件复杂程度的一个重要因素。应该采取下述设计原则：**尽量使用数据耦合，少用控制耦合和标记耦合，限制公共环境耦合的范围，完全不用内容耦合。**

（六）模块独立——耦合综合实例



Out	
aircraft type	status flag
list of aircraft parts	–
function code	–
list of aircraft parts	–
part number	part manufacturer
part number	part name

访问相同的数据库，修改全局变量，公共耦合

数据结构，数据耦合/标记耦合

简单变量，数据耦合

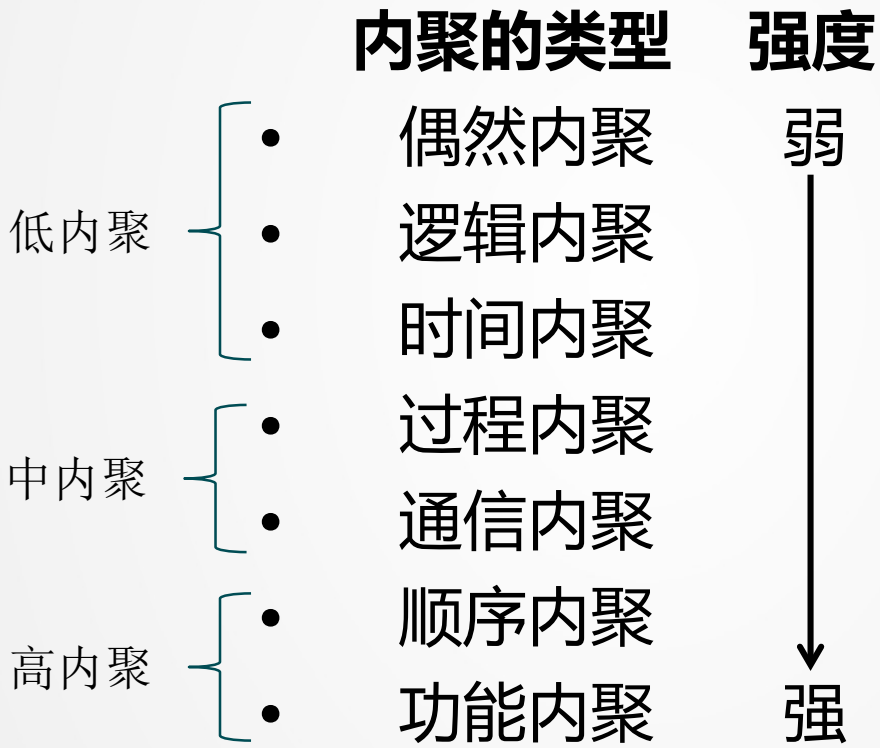
功能代码，控制耦合

（六）模块独立——内聚

内聚衡量一个模块内部各个元素彼此结合的紧密程度。它是信息隐藏和局部化概念的自然扩展。简单地说，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。

（六）模块独立——内聚



（六）模块独立——内聚

①偶然内聚

一个模块由完成若干毫无关系的功能处理元素偶然组合在一起的，叫做偶然内聚。

偶然内聚是最差的一种内聚。

例如一组语句在多处出现，于是为了节省空间而将这些语句作为一个模块设计。

（六）模块独立——内聚

②逻辑内聚

一个模块完成的任务在逻辑上属于相同或相似的一类，则称为逻辑内聚。

几个逻辑上相关的功能被放在同一模块中。

例如对文件记录执行插入、删除和修改的编辑模块

函数A1
函数A2
函数A3

```
function A(opcode: int)
{
  If code = 7  new operation(opcode);
  else new operation (opcode, dummy 1, dummy 2, dummy 3);
  // dummy 1, dummy 2, and dummy 3 are dummy variables,
  // not used if opcode is equal to 7
}
```

（六）模块独立——内聚

③时间内聚

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，也称为瞬时内聚。这种内聚方式通常是因为外部事件（如定时中断）而将任务组织在一起，但任务之间的逻辑关联可能并不强。

（六）模块独立——内聚

④过程内聚

过程内聚指的是一个模块中的处理元素之间存在某种控制流的关系，这些元素必须按照一定的顺序执行，即使它们之间没有数据传递。在过程内聚中，控制流从一个动作流向另一个动作，而不需要数据作为媒介。

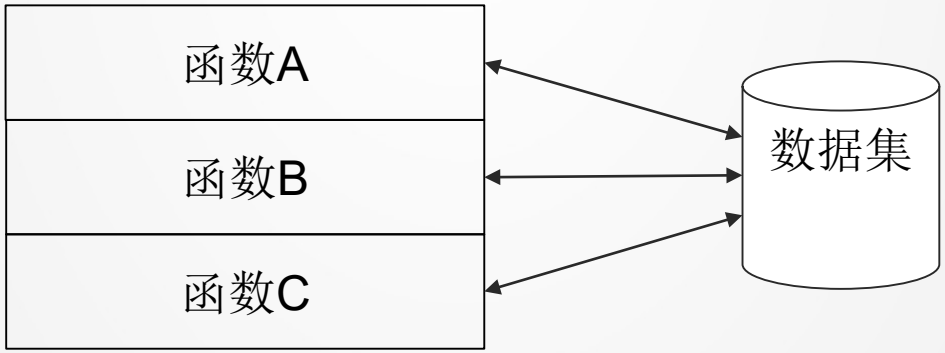

```
Function
{
    read part number from database;
    update repair record on master file;
}
```

处理记录按照特定顺序执行

（六）模块独立——内聚

⑤通信内聚

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。

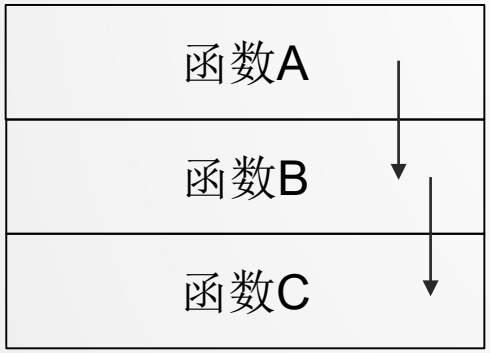


（六）模块独立——内聚

⑥顺序内聚

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(通常一个处理元素的输出数据作为下一个处理元素的输入数据)，则称为顺序内聚。

根据数据流图划分出的模块通常都是顺序内聚的模块。



示例
学生成绩合格率统计模块：
前一个部分根据成绩统计出**及格的学生人数**，
后一个部分根据**及格人数**计算出学生的及格率

注意：
顺序内聚中是数据流从一个处理单元流到另一个处理单元。
过程内聚是控制流从一个动作流向另一个动作。

（六）模块独立——内聚

⑦功能内聚

模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

不是每个模块都能设计成一个功能内聚模块。

（六）模块独立——内聚

耦合和内聚的概念是Constantine，Yourdon，Myers和Stevens等人提出来的。上述7种内聚的优劣评分，将得到如下结果：

事实上，没有必要精确确定内聚的级别。重要的是设计时尽量使用**高内聚，低耦合**模块。

课堂练习

- 耦合、内聚的定义
- 各自的分类

西南大学

目录
CONTENTS

（一）启发式规则

改进软件设计，提高软件质量的途径。

有7条规则：

1. **改进软件结构提高模块独立性**
2. **模块规模应该适中**
3. **深度、宽度、扇出和扇入都应适当**
4. **模块的作用域应该在控制域之内**
5. **力争降低模块接口的复杂程度**
6. **设计单入口单出口的模块**
7. **模块功能应该可以预测**

经验规律
具有参考价值

（一）启发式规则

经验规律
具有参考价值

①改进软件结构提高模块独立性

设计初步结构以后，应进行审查分析，通过模块分解或合并，力求降低耦合提高内聚。

- 例如，多个模块公有的一个子功能可以独立成一个模块，由这些模块调用；
- 有时可以通过分解或合并模块以减少控制信息的传递及对全局数据的引用，并且降低接口的复杂程度。

（一）启发式规则

经验规律
具有参考价值

②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）。

- 过大的模块可能由于分解不充分，但进一步的分解必须符合问题结构。
- 分解不应该降低模块独立性。
- 模块的可理解与接口的复杂性的矛盾统一。

（一）启发式规则

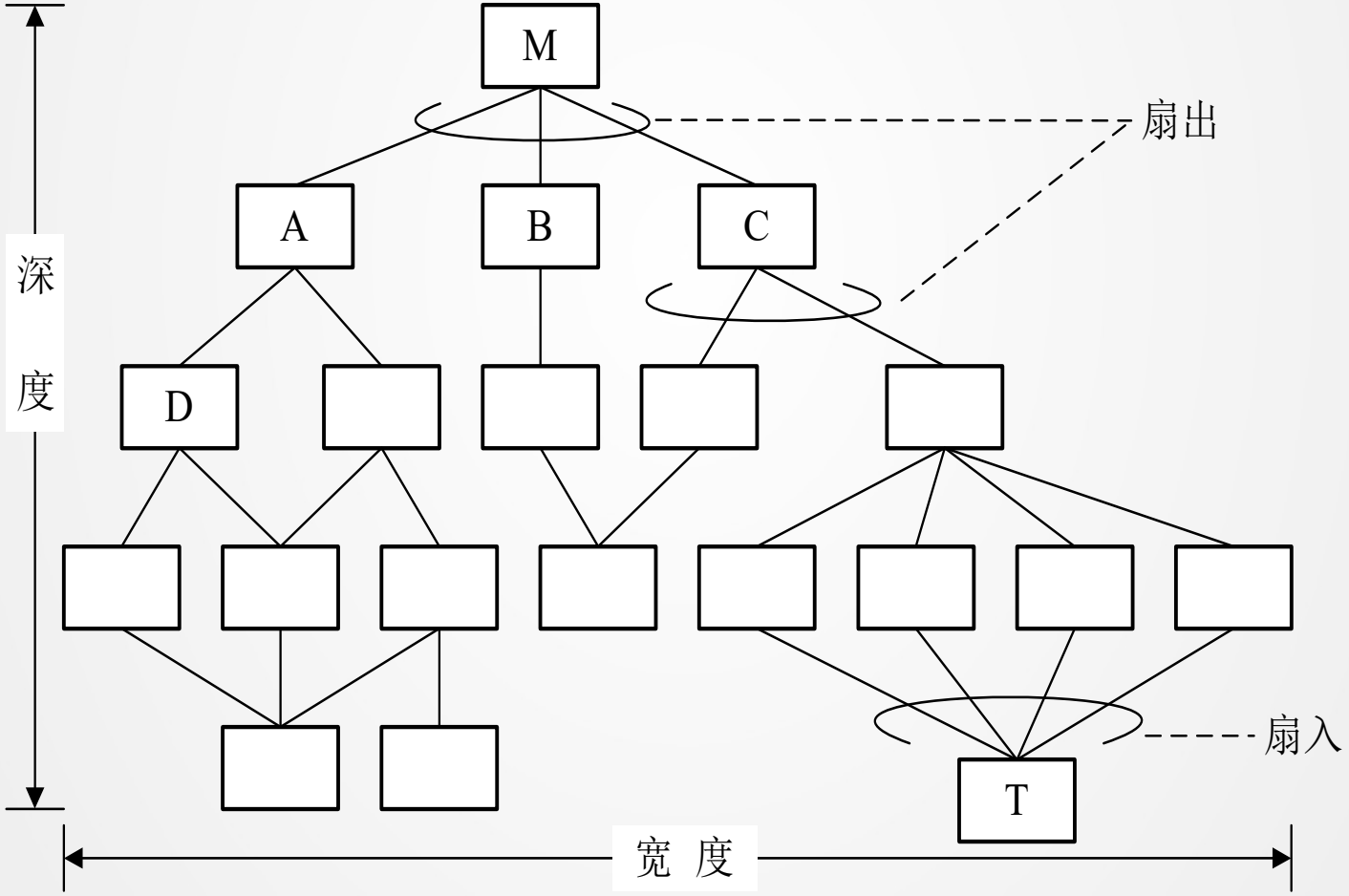
经验规律
具有参考价值

③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，它往往能粗略地标志一个系统的大小和复杂程度。
  - 如果层次过多，则应该考虑是否有许多管理模块过分简单了，能否适当合并。
- 宽度：软件结构内同一个层次上的模块总数的最大值。
  - 一般说来，宽度越大系统越复杂。
    对宽度影响最大的因素是模块的扇出。
- 扇出：一个模块直接控制（调用）的模块数目
- 扇入：表明有多少个上级模块直接调用它

（一）启发式规则

深度、宽度、扇出和扇入（示例）



（一）启发式规则

经验规律
具有参考价值

③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)的模块数目
- 扇入：表明有多少个上级模块直接调用它
  - 经验表明，一个设计得好的典型系统的平均扇出通常是3或4(上限通常是5～9)。扇出太大：缺乏中间层次，应该适当增加中间层次的控制模块。扇出太小：把下级模块进一步分解成若干个子功能模块，或者合并到它的上级模块中去。
  - 扇入越大则共享该模块的上级模块数目越多，这是有好处的。但是，不能违背模块独立原理单纯追求高扇入。

建议：顶层高扇出，底层高扇入

（一）启发式规则

经验规律
具有参考价值

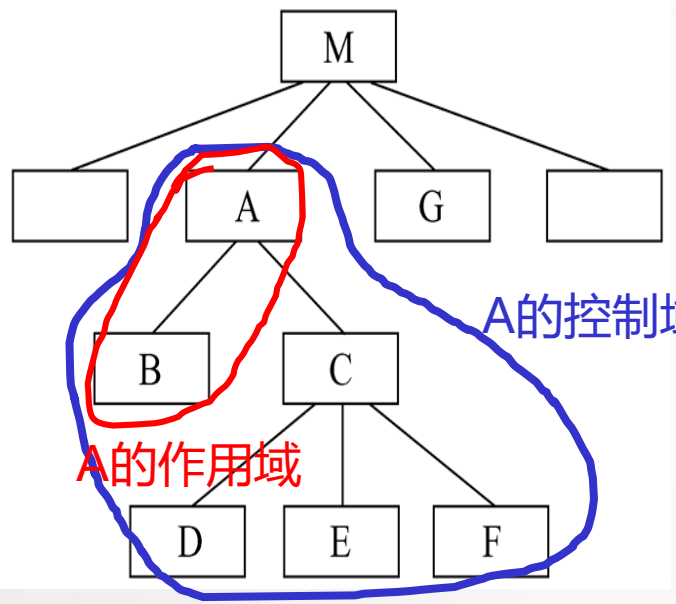
④模块的作用域应该在控制域之内

作用域： 受该模块内一个判定影响的所有模块的集合。

控制域： 模块本身以及所有直接或间接从属于它的模块的集合。

所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块。

（一）启发式规则

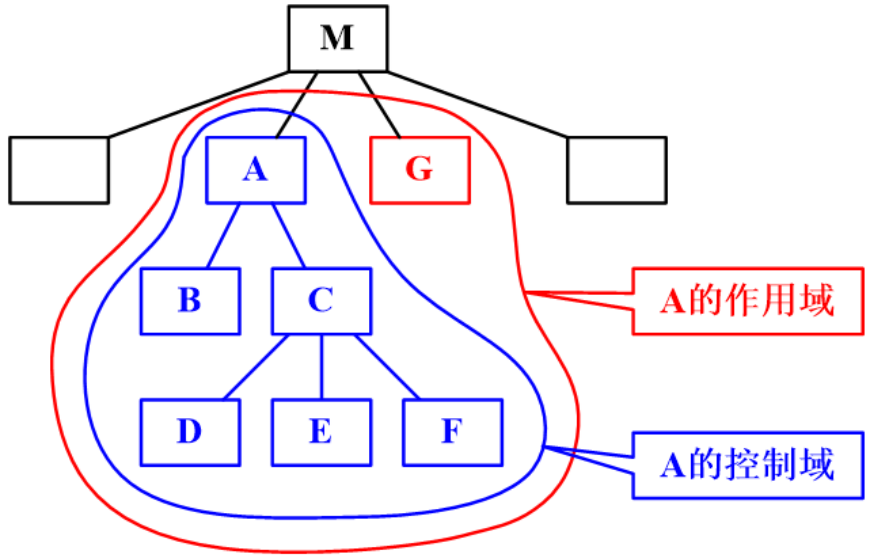


假如A里面只有一个判定影响B

```
Function A
    if  x=10
      goto B
    y=x+1;
    …
    …
```

（一）启发式规则

假如A里面还有一个判定影响了G

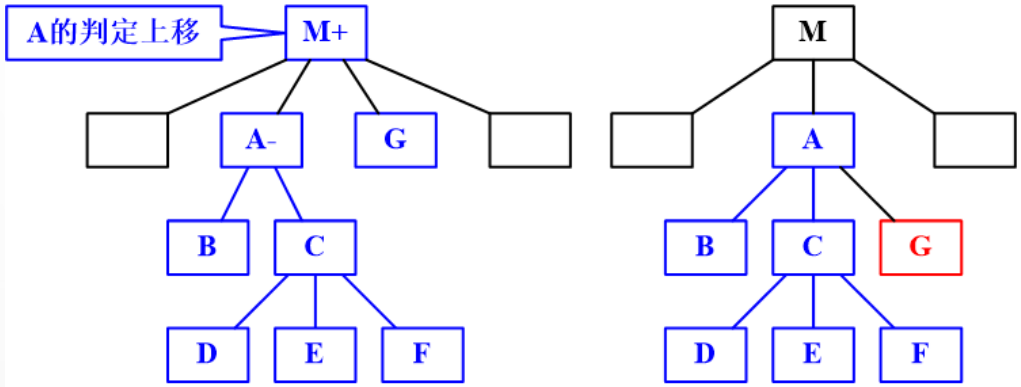


违反规则的情况

修改软件结构使得作用域在控制域之内:

方法1：是把做判定的点往上移;

方法2：把那些在作用域内但不在控制域内的模块移到控制域内。



（一）启发式规则

经验规律
具有参考价值

⑤力争降低模块接口的复杂程度

- 模块接口复杂是软件发生错误的一个主要原因。
- 应仔细设计模块接口，使得信息传递简单并且和模块的功能一致。
- 接口复杂或不一致（即看起来传递的数据之间没有联系），是紧耦合或低内聚的征兆，应该重新分析这个模块的独立性。

（一）启发式规则

力争降低模块接口的复杂程度（示例）

例：设计求一元二次方程的根的模块的接口。

QUAD_ROOT (TBL, X)

其中，数组 TBL传送方程的系数，数组X回送求得的根。

这种传递信息的方法不利于对这个模块的理解，不仅在维护期间容易引起混淆，开发期间也可能容易出错。

QUAD_ROOT (A, B, C, ROOT1, ROOT2)

其中，A、B、C是方程的系数，ROOT1和ROOT2是算出的两个根。

这种接口设计相对比较简单。

（一）启发式规则

经验规律
具有参考价值

⑥设计单入口单出口的模块

- 不要出现内容耦合。
- 从顶部进入模块，从底部退出模块。
- 易理解，易维护。

（一）启发式规则

经验规律
具有参考价值

⑦模块功能应该可以预测

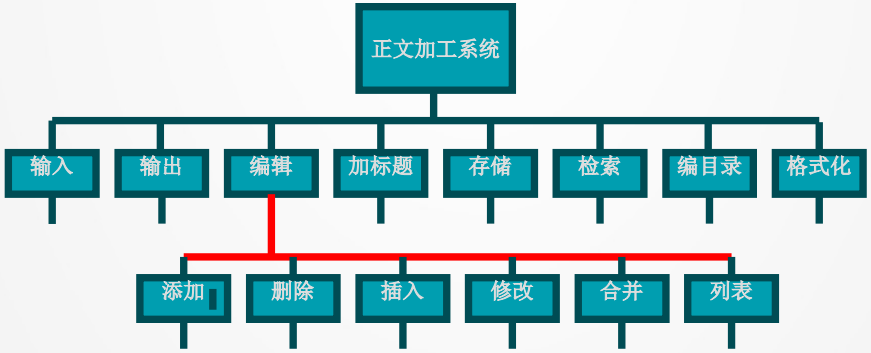
- 即模块的功能应该明确且易于理解，避免模糊不清或难以预测的行为。
- 带有内部“存储器”的模块的功能可能是不可预测的，其输出可能取决于内部存储器（例如某个标记）的状态。
- 内部存储器对于上级模块而言是不可见的，所以这样的模块既不易理解又难于测试和维护。

西南大學

目录
CONTENTS

（一）描绘软件结构的常用图形工具

描绘软件结构常用图形工具：
层次图、HIPO图、结构图

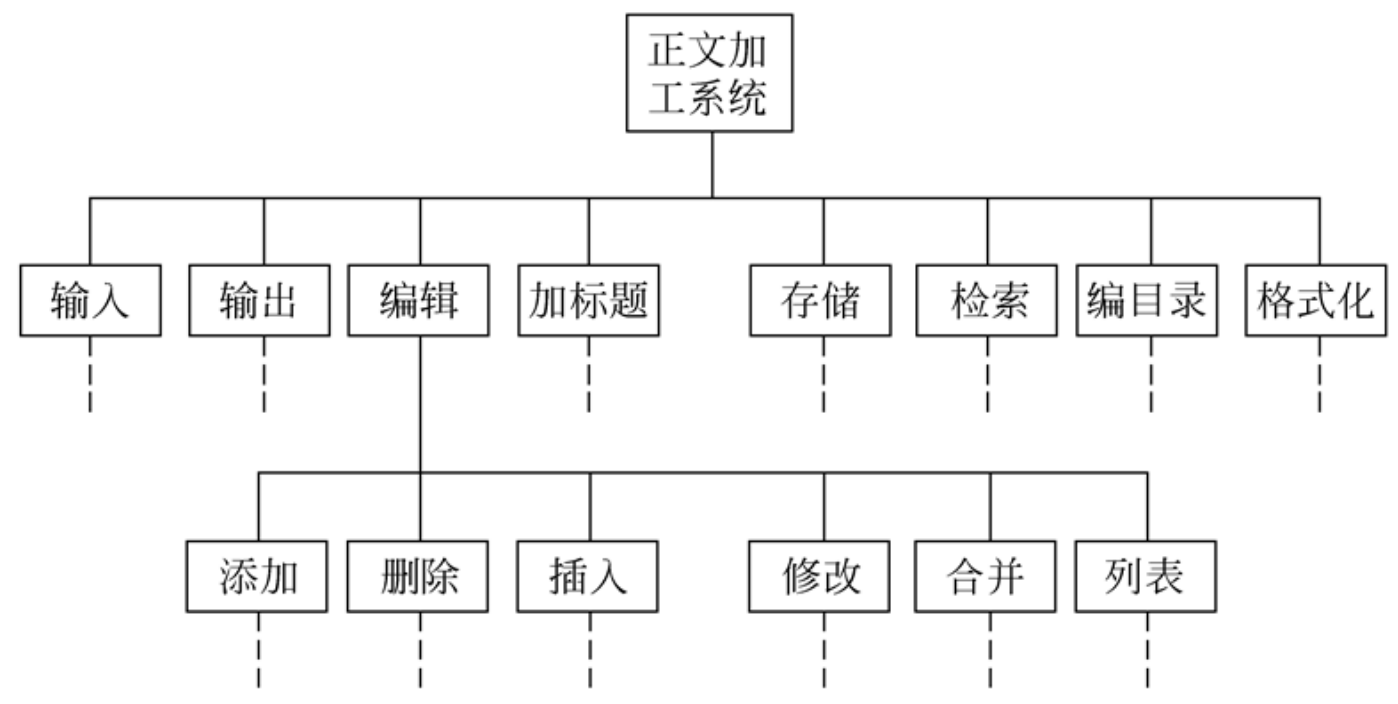


（二）层次图和HIPO图

层次图：用来描绘软件的层次结构。

形式和描绘数据结构的层次方框图相同，但表现的内容却完全不同。

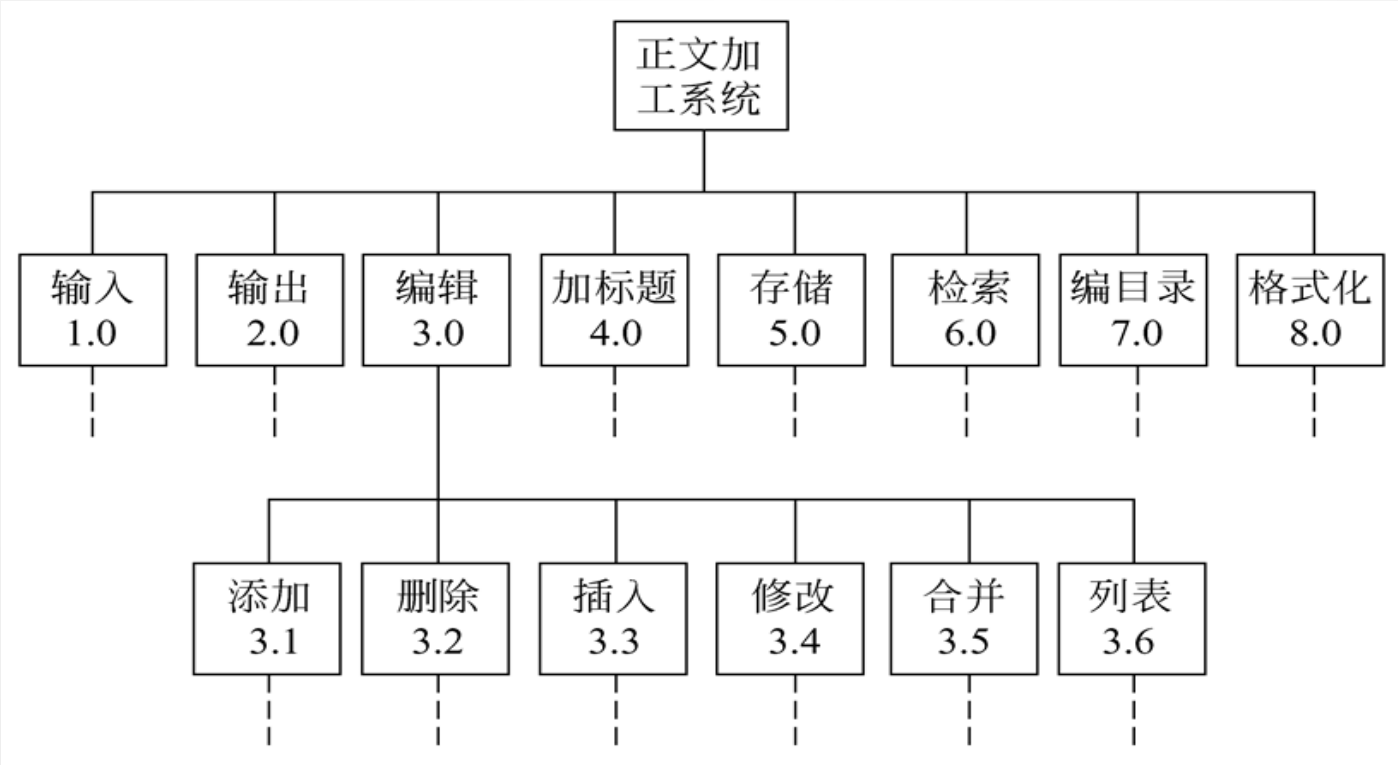
- 层次图中的一个矩形框代表一个模块，方框间的连线表示调用关系
- 层次方框图中的一个矩形框代表数据的子集，方框间的连线表示组成关系。



（二）层次图和HIPO图

HIPO图=层次图+IPO图

为了能使HIPO图具有可追踪性，在H图(层次图)里除了最顶层的方框之外，每个方框都加了编号。编号规则和数据流图的编号规则相同。

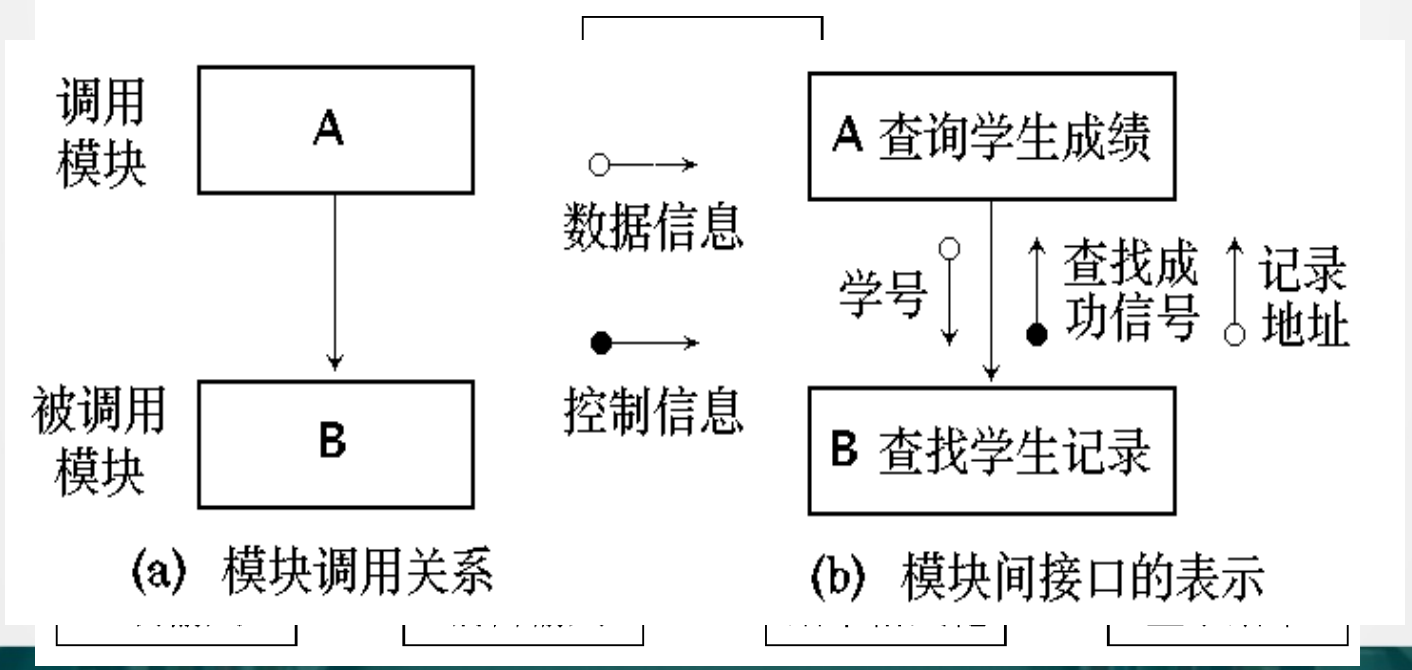


（三）结构图

Edward Yourdon 提出的一个软件结构设计工具

结构图基本符号：

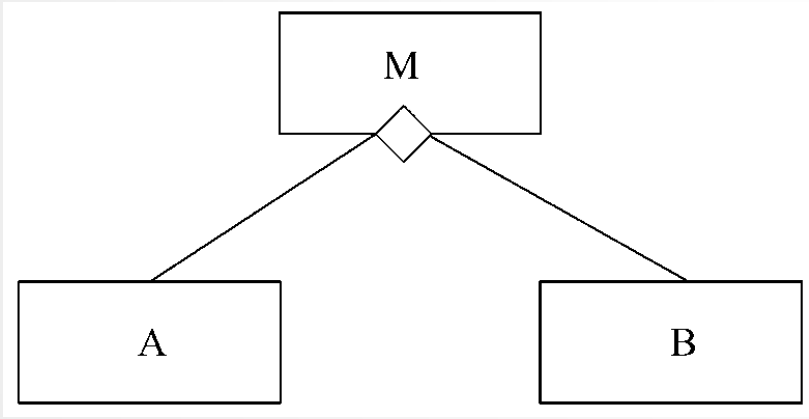
- 方框——模块
- 方框间直线——模块调用关系
- 箭头尾部空心圆——传递数据信息
- 箭头尾部实心圆——传递控制信息



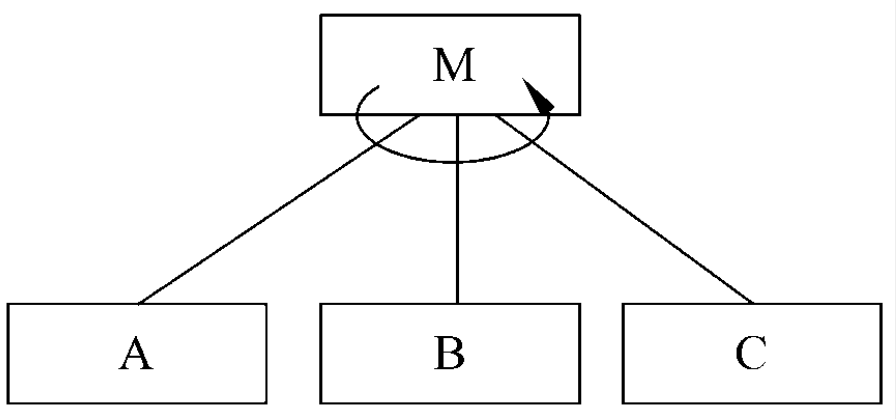
（三）结构图

结构图附加符号：

- 菱形——模块的选择调用
- 环形箭头——循环调用



判定为真时调用A，为假时调用B



模块M循环调用模块A、B、C

（四）层次图和结构图的关系

共同点：

- 都是描绘软件结构的图形工具。
- 都不严格表示模块的调用次序（多数人习惯按调用次序从左到右画模块）。
- 都不指明何时调用下层模块。
- 都只表明一个模块调用哪些模块，而不表示模块内是否还有其他成分。

差异点：

- 软件结构图更侧重于描述软件系统的整体结构和组件间的交互关系，而层次图则更专注于展示软件的层次结构和模块间的调用关系。

西南大學

目录
CONTENTS

（一）面向数据流的设计概念

通常所说的结构化设计方法(简称SD方法)，也就是面向数据流的设计方法，是一种基于数据流图（DFD）来设计软件结构的方法。面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的策略。

信息流有下述两种类型：

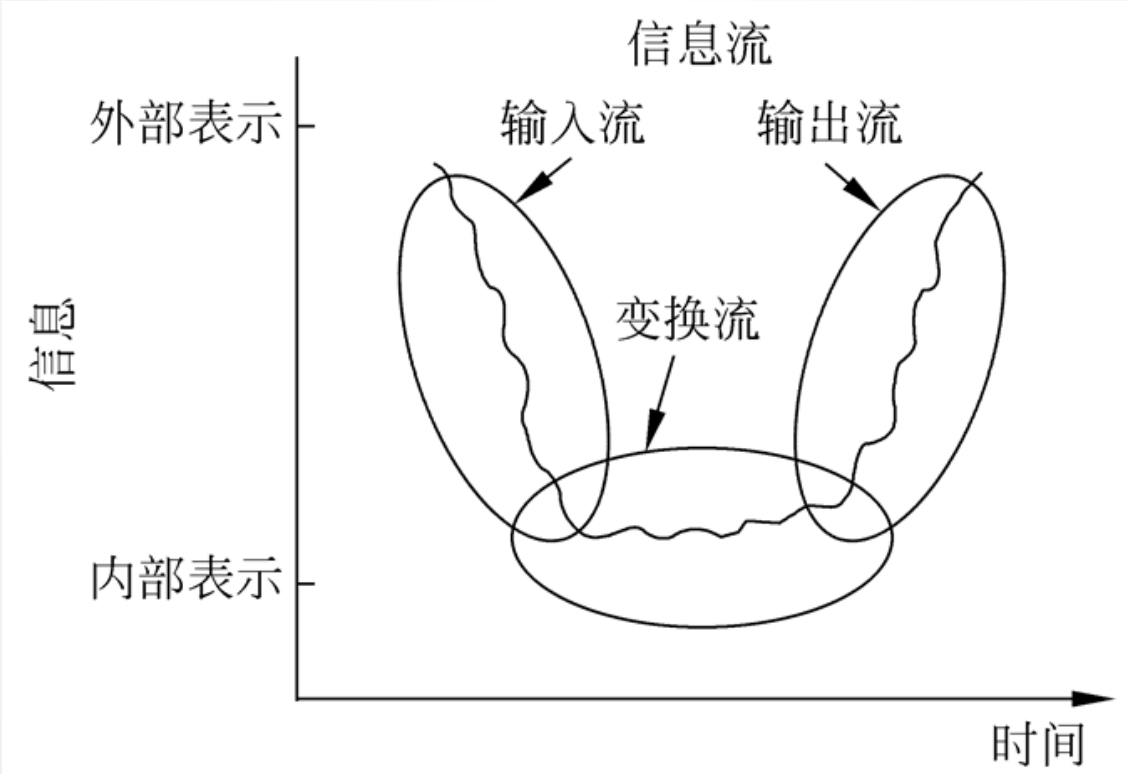
1.变换流（交换流）

2.事务流

任何软件系统都可以用数据流图表示，所以面向数据流的设计方法理论上可以设计任何软件的结构。

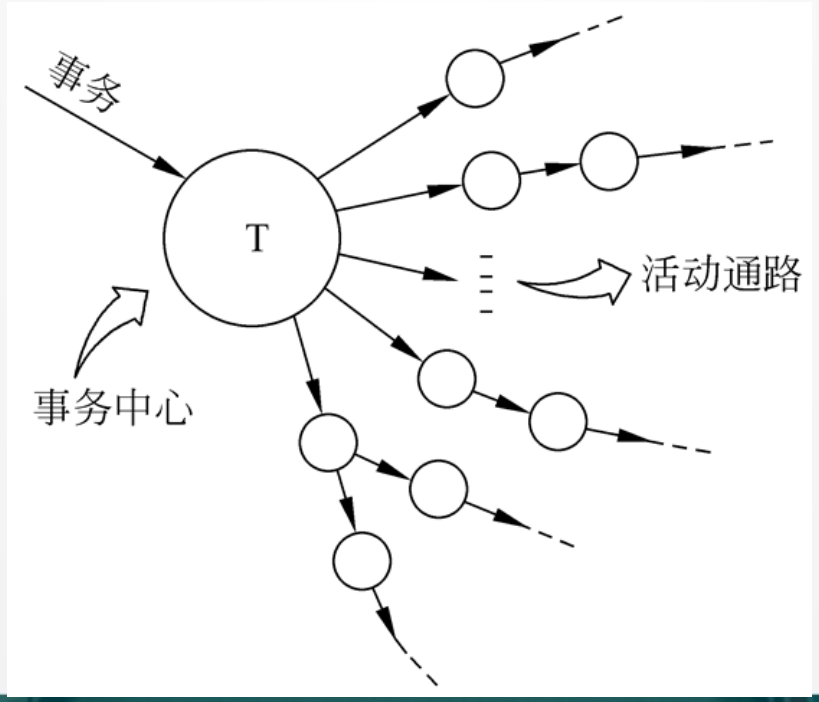
（二）变换流

信息沿输入通路进入系统，同时由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式离开软件系统。



（三）事务流

数据沿输入通路到达一个处理T，T根据输入数据的类型在若干个动作序列中选出一个来执行。处理T称为事务中心，它可以完成这些任务：(1)接收输入数据；(2)分析每个事务以确定它的类型；(3)根据事务类型选取一条活动通路。



（四）总体设计过程

1

精化的DFD

→初始模块结构图

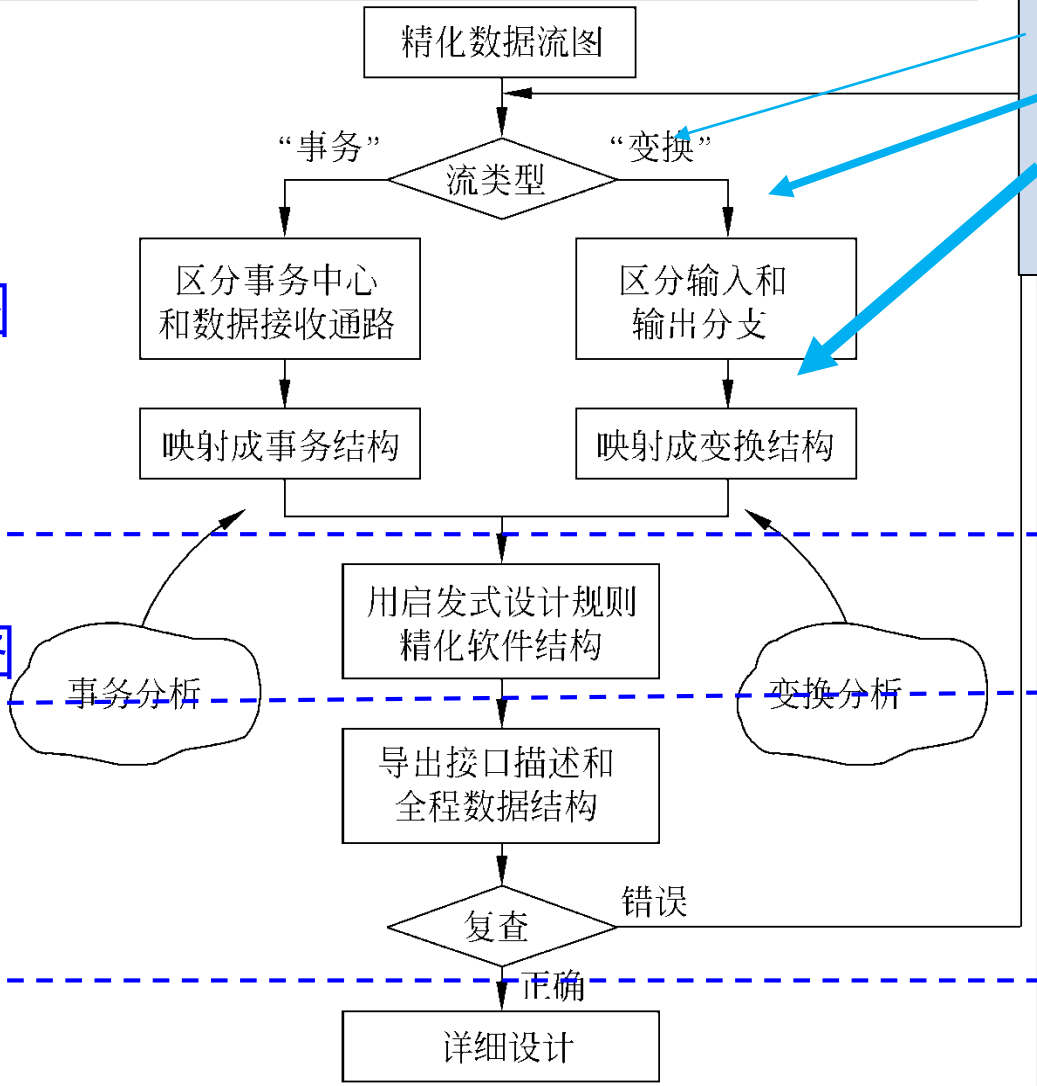
2

精化初始模块结构图

3

导出/复查、优化

注：两个不规则形状及箭头为说明信息。



面向数据流方法的总体设计过程

蓝色箭头:

1）判断类型

2）画出边界

3）变换分析或事务分析

（五）变换分析

变换分析是把具有变换流特点的数据流图按预先确定的模式映射成软件结构的一系列设计步骤的总称。

具体步骤：

第1步复查基本系统模型。

第2步复查并精化数据流图。

第3步确定数据流图具有变换特性还是事务特性。

第4步确定输入流和输出流的边界，从而孤立出变换中心。

第5步完成第一级分解。

第6步完成第二级分解。

第7步使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

（五）变换分析——例子

例子： 汽车数字仪表板的设计

假设的仪表板将完成下述功能：

(1) 通过模数转换实现传感器和微处理机接口。

(2) 在发光二极管面板上显示数据。

(3) 指示每小时英里数(mph)，行驶的里程每加仑油行驶的英里数(mpg)等。

(4) 指示加速或减速。

(5) 超速警告：如果车速超过55英里/小时，则发出超速警告铃声。

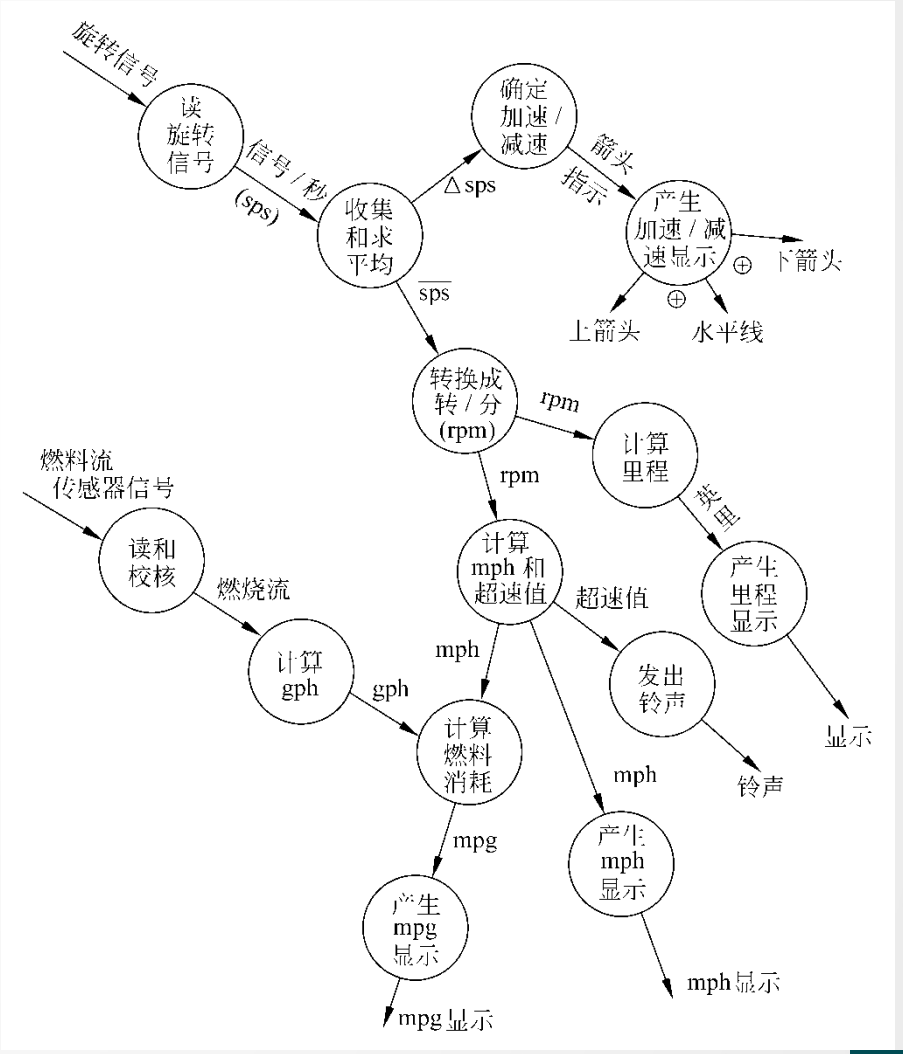
在软件需求分析阶段应该对上述每条要求以及系统的其他特点进行全面的分析评价，建立必要的文档资料，特别是数据流图。

（五）变换分析——例子

第1步复查基本系统模型。

第2步复查并精化数据流图。

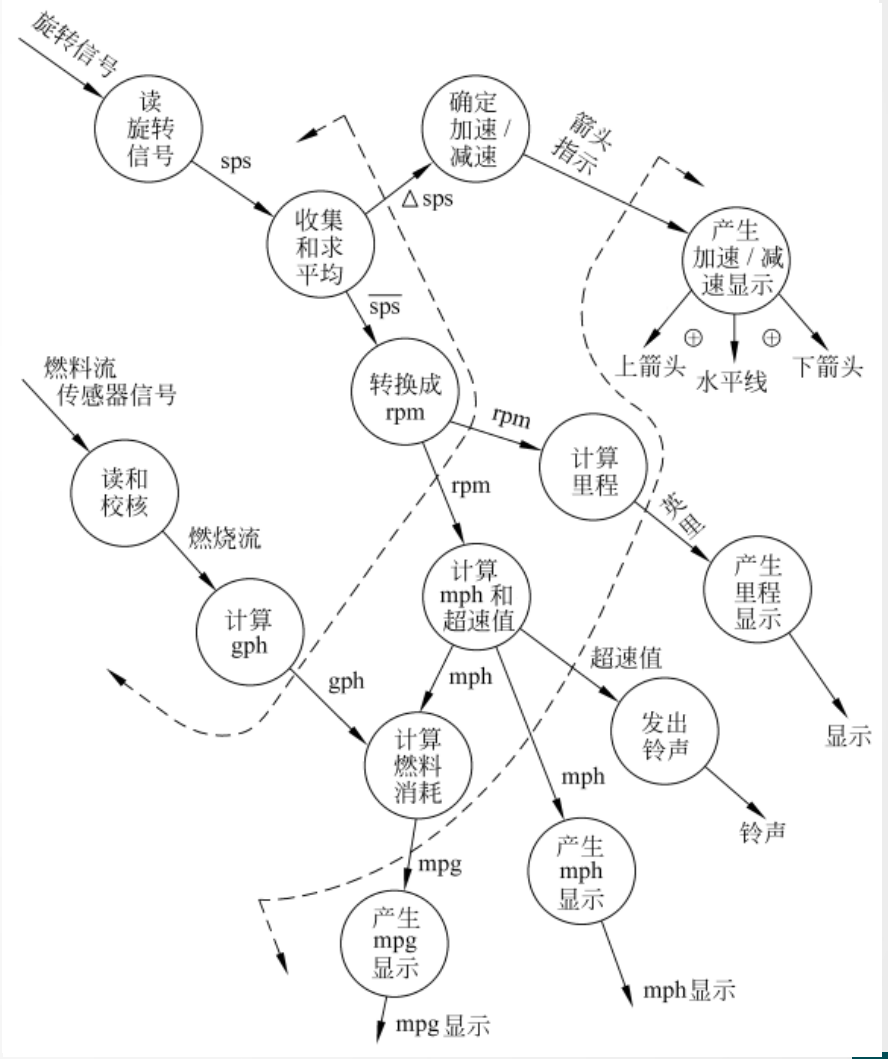
假设在需求分析阶段产生的数字仪表板系统的数据流图如右图所示。



（五）变换分析——例子

第3步确定数据流图具有变换特性还是事务特性。

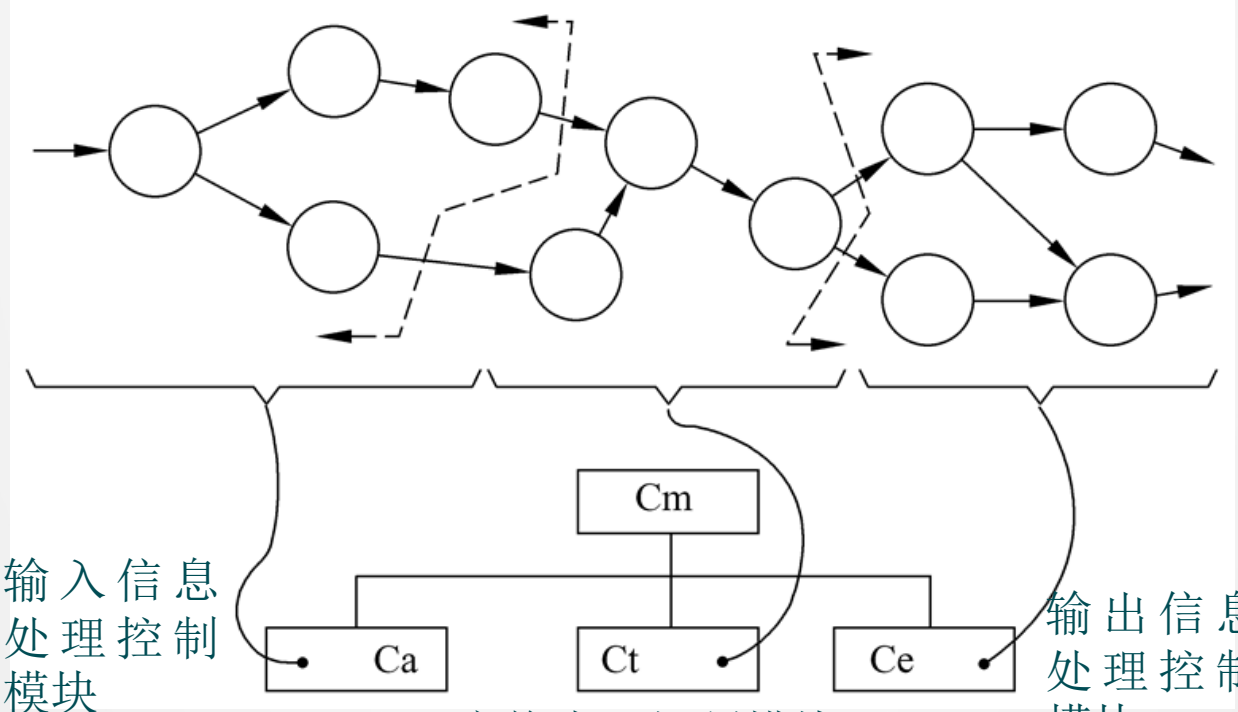
第4步确定输入流和输出流的边界，从而孤立出变换中心。



（五）变换分析——例子

第5步完成第一级分解。

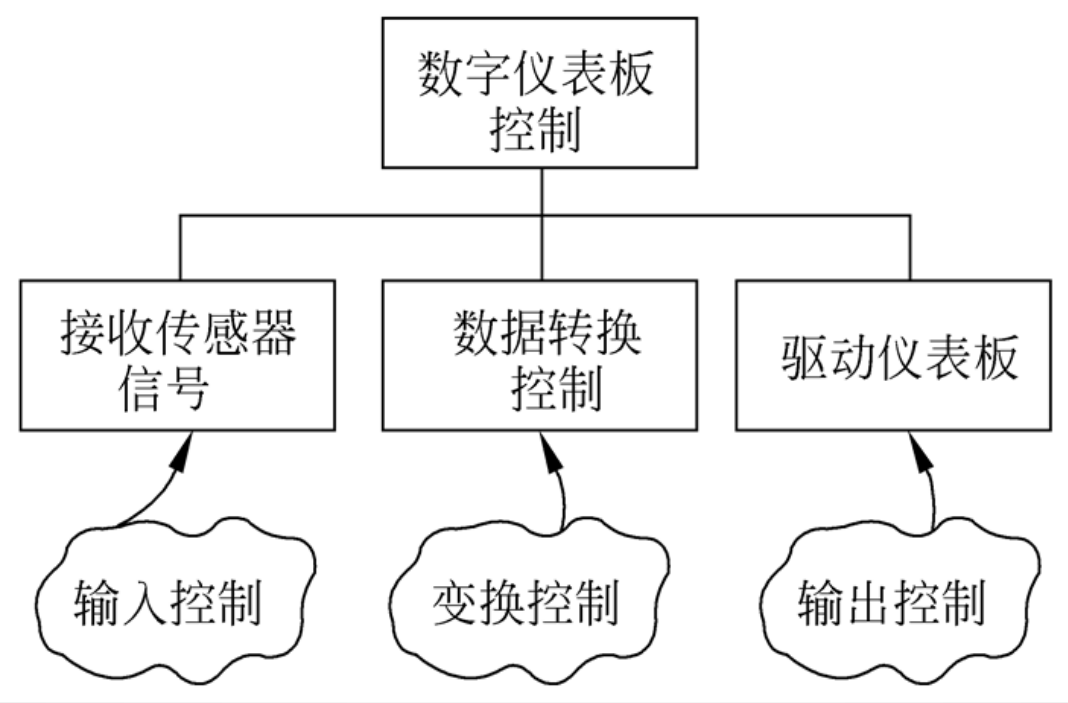
软件结构代表对控制的自顶向下的分配，所谓分解就是分配控制的过程。数据流图被映射成一个特殊的软件结构，这个结构控制输入、变换和输出等信息处理过程。



（五）变换分析——例子

第5步完成第一级分解。

对于数字仪表板的例子，第一级分解得出的结构如下图所示。每个控制模块的名字表明了为它所控制的那些模块的功能。

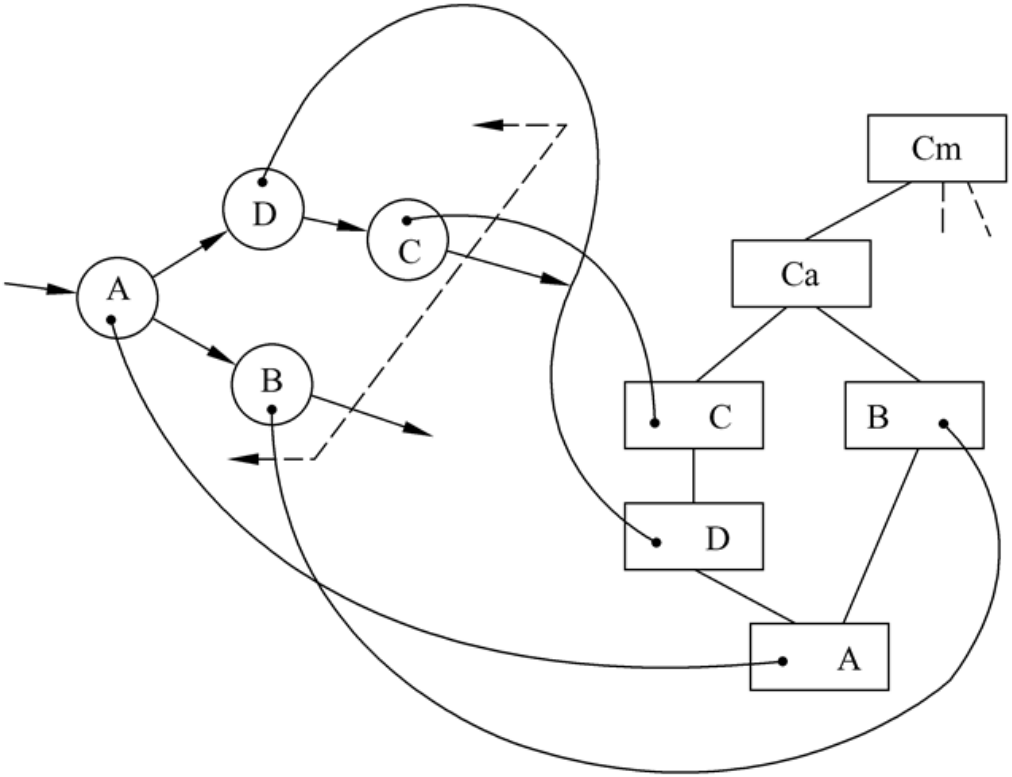


（五）变换分析——例子

第6步完成第二级分解。

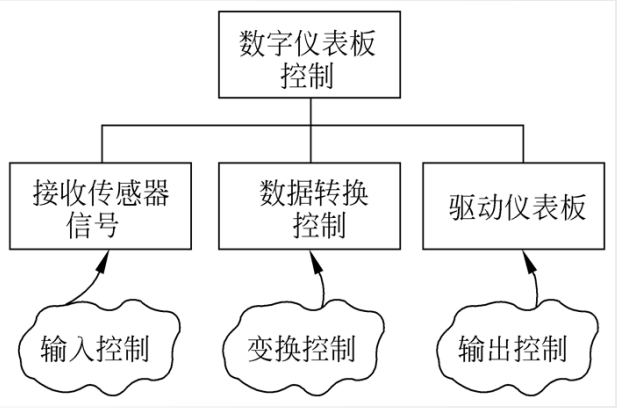
第二级分解就是把数据流图中的每个处理映射成软件结构中一个适当的模块。
完成第二级分解的方法：

- 从变换中心的边界开始逆着输入通路向外移动，把**输入通路**中每个处理映射成软件结构中Ca控制下的一个低层模块；
- 然后沿**输出通路**向外移动，把输出通路中每个处理映射成直接或间接受模块Ce控制的一个低层模块；
- 最后把**变换中心**内的每个处理映射成受Ct控制的一个模块。



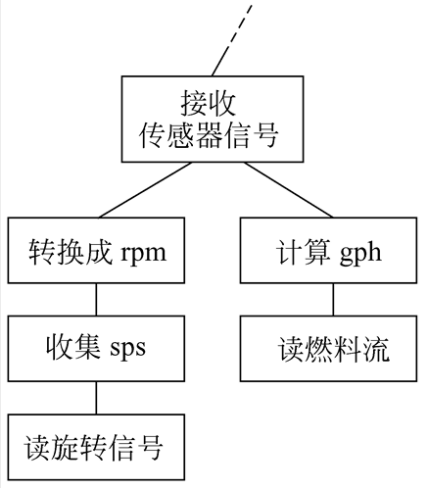
输入信息处理控制模块分解示例

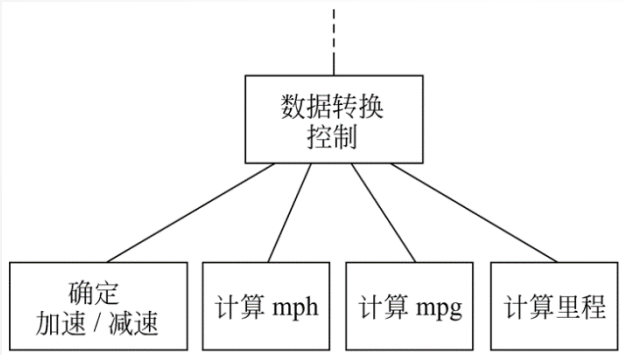
（五）变换分析——例子

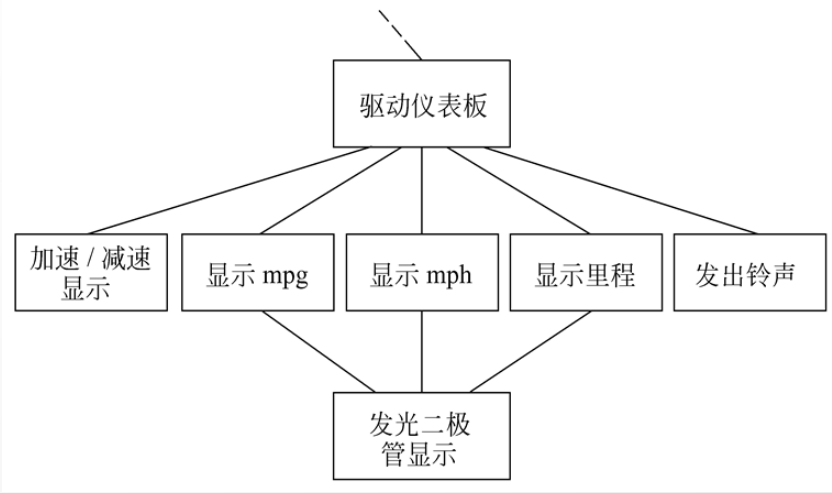


第6步完成第二级分解。

对于数字仪表板的例子，第二级分解的结果分别用图下面三图描绘。



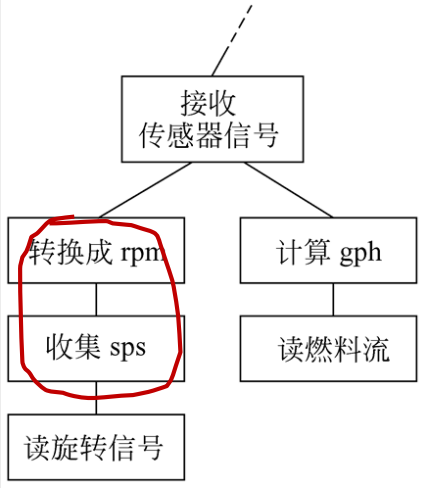


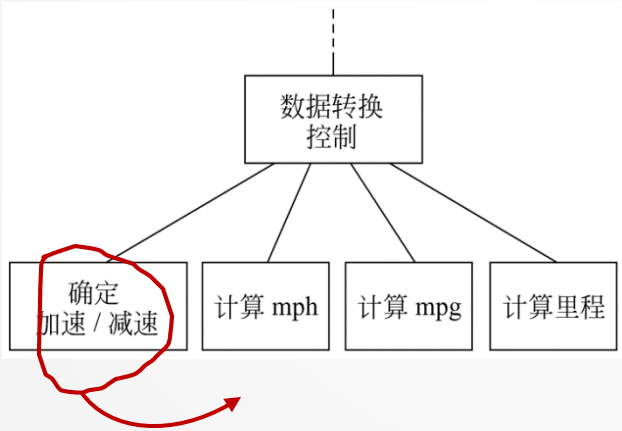


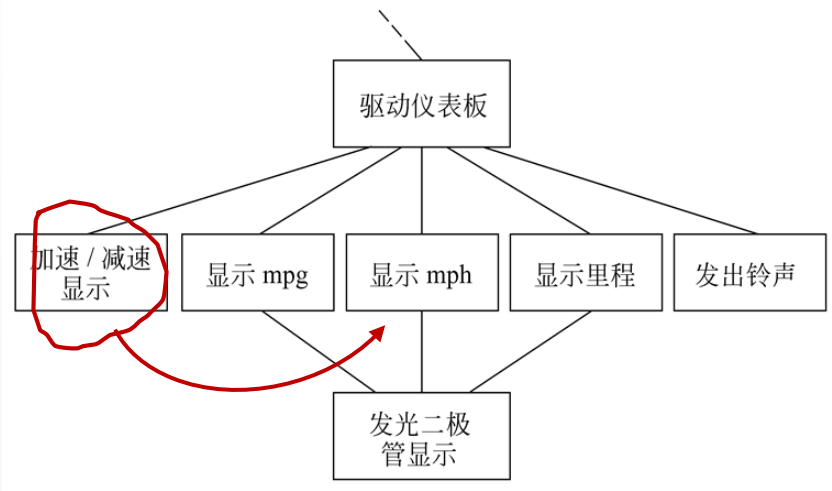
（五）变换分析——例子

第7步使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

具体到数字仪表板的例子，对于从前面的设计步骤得到的软件结构，还可以做许多修改：



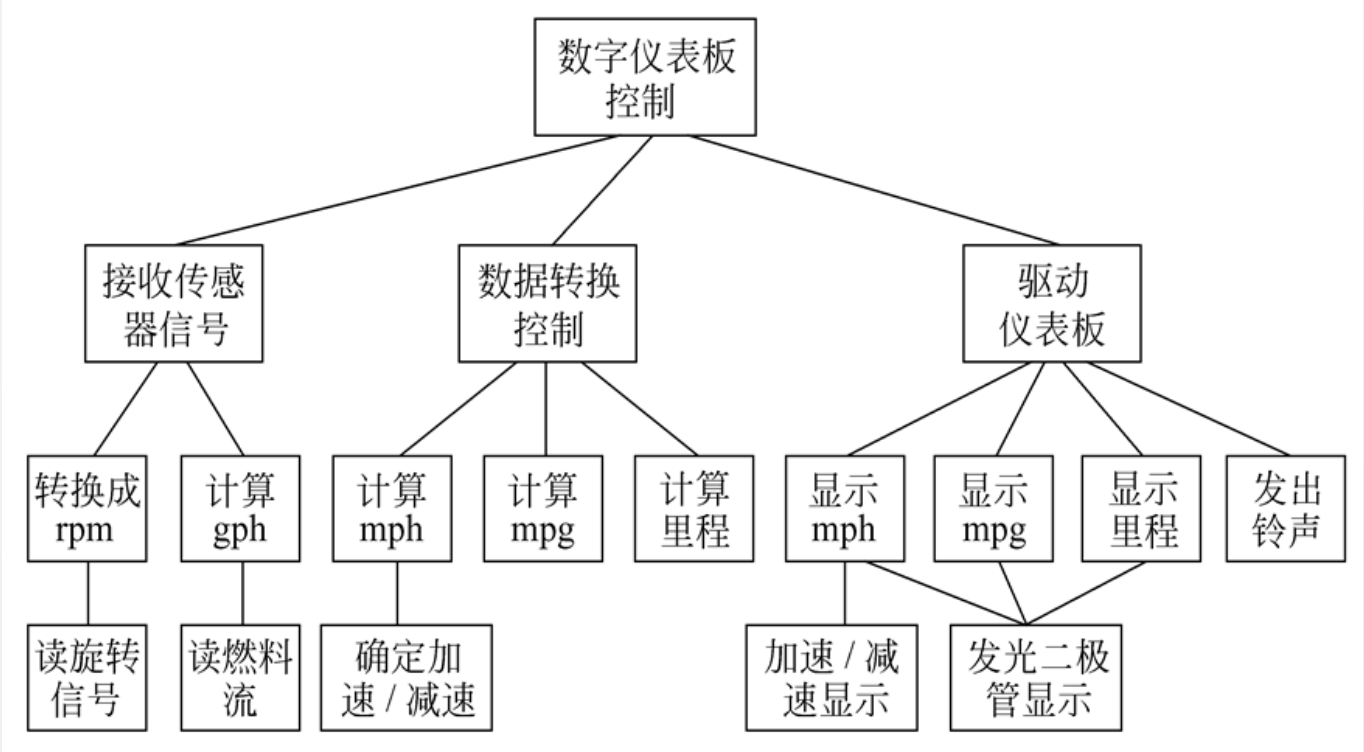




（五）变换分析——例子

第7步使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

经过修改（精化）后的软件结构图如下：



（六）事务分析

在数据流**具有明显的事务特点**时，也就是有一个明显的“发射中心”(事务中心)时，还是以采用事务分析方法为宜。

具体步骤：

第1步复查基本系统模型。

第2步复查并精化数据流图。

第3步确定数据流图具有变换特性还是事务特性。

第4步确定接收部分和发送部分的边界。

第5步完成第一级分解。

第6步完成第二级分解。

第7步使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

（六）事务分析

接收分支结构

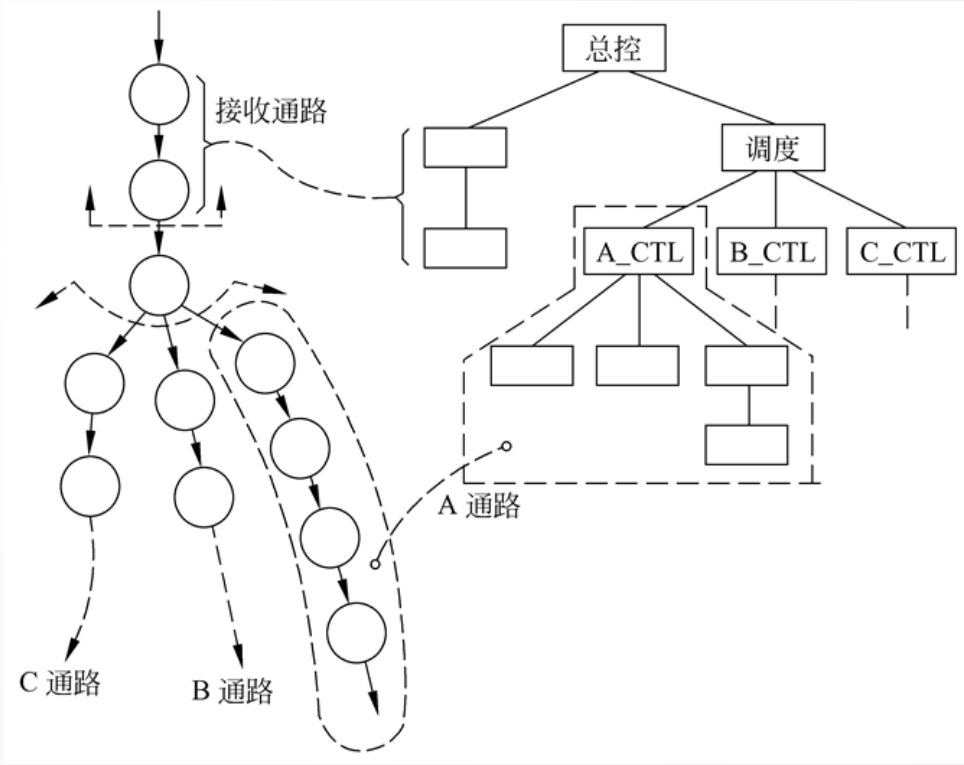
映射方法和变换分析映射出输入结构的方法很相像。

从事务中心的边界开始，把沿着接收流通路的处理映射成模块。

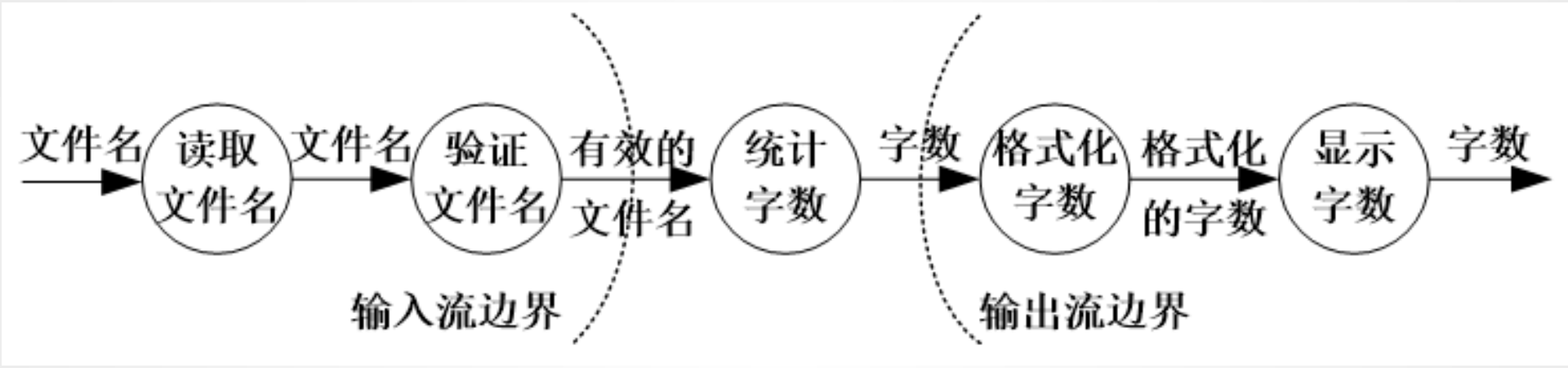
发送分支结构

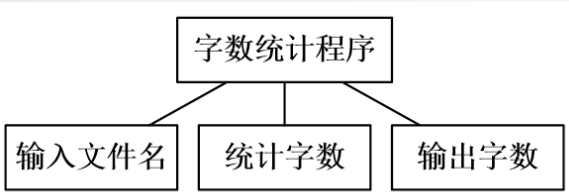
包含一个调度模块，它控制下层的所有活动模块；

然后把数据流图中的每个活动流通路映射成与它的流特征相对应的结构。

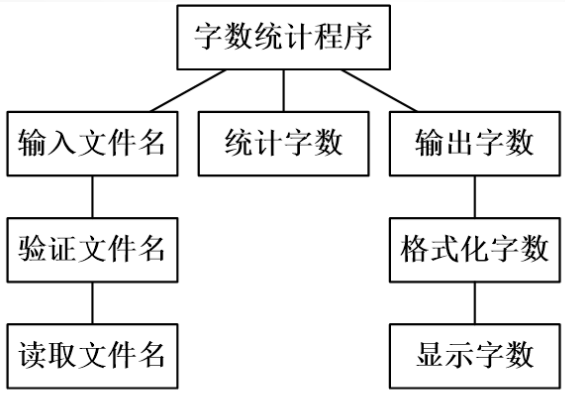


例子：将一个文件名作为输入，并返回文件中的字数。

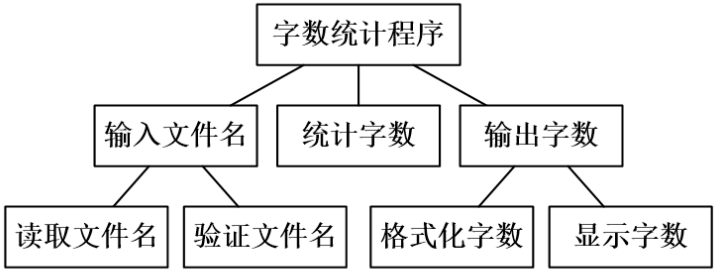




第一级分解



第二级分解



精化

（七）设计优化

对第一次分割得到的软件结构，总可以根据模块独立原理和启发式设计规则进行优化。为了产生合理的分解，得到尽可能高的内聚、尽可能松散的耦合，最重要的是，为了得到一个易于实现、易于测试和易于维护的软件结构，应该对初步分割得到的模块进行再分解或合并。

设计优化应该力求做到在有效的模块化的前提下使用最少量的模块，以及在能够满足信息要求的前提下使用最简单的数据结构。

（七）设计优化

“一个不能工作的‘最佳设计’的价值是值得怀疑的”。

所以，应遵守这样一句格言：“先使它工作起来，然后再使它快起来。”

- **用下述方法对时间起决定性作用的软件进行优化是合理的**：

(1) 在不考虑时间因素的前提下开发并精化软件结构。

(2) 在详细设计阶段选出最耗费时间的那些模块，仔细地设计它们的处理过程(算法)，以求提高效率。

(3) 使用高级程序设计语言编写程序。

(4) 在软件中孤立出那些大量占用处理机资源的模块。

(5) 必要时重新设计或用依赖于机器的语言重写上述大量占用资源的模块的代码，以求提高效率。

西南大學

目录
CONTENTS

小结

1.总体设计阶段主要由系统设计和结构设计两阶段组成。

2.进行软件结构设计时应该遵循的最主要的原理是模块独立原理。

3.在软件开发过程中既要充分重视和利用这些启发式规则，又要从实际情况出发避免生搬硬套。

4.层次图和结构图是描绘软件结构的常用工具。

5.用面向数据流的设计方法由数据流图映射出软件结构。

本章结束