中国石油大学（华东）
CHINA UNIVERSITY OF PETROLEUM

软件工程

主要内容

第一章 软件工程学概述
第二章 可行性研究
第三章 需求分析
第四章 总体设计
第五章 详细设计
第六章 编码与测试
第七章 软件维护
第八章 面向对象方法学
第九章 面向对象分析设计与实现
第十章 软件项目管理

第四章　总体设计

第一节　总体设计过程

第二节　设计原理

第三节　总体设计准则

第四节　描绘软件结构的图形工具

第五节　面向数据流的设计方法

第四章　总体设计

- 在软件需求分析阶段确定了要让所开发的软件“做什么”的问题，接下来就是实现软件的需求，解决“怎样做”的问题。
- 软件设计就是处理“怎样做”的问题，它被定义为“应用各种技术和原理，对设备、过程或系统作出足够详细的描述，使之能够在物理上得以实现”。
- 从工程管理的角度看来，软件设计可划分为总体设计和详细设计两个阶段。总体设计也称为概要设计，其基本目的是：

——— 概括地说“系统是如何实现的”。

总体设计的任务：

- 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；
- 设计软件结构。即确定系统中由那些模块组成，以及这些模块之间的相互关系。

第四章　总体设计

第一节　总体设计过程

典型的总体设计过程包括下述 9 个步骤：

1. 设想供选择的方案
2. 选取合理的方案
3. 推荐最佳方案
4. 功能分解
5. 设计软件结构
6. 设计数据库
7. 制定测试计划
8. 书写文档
9. 审查和复审

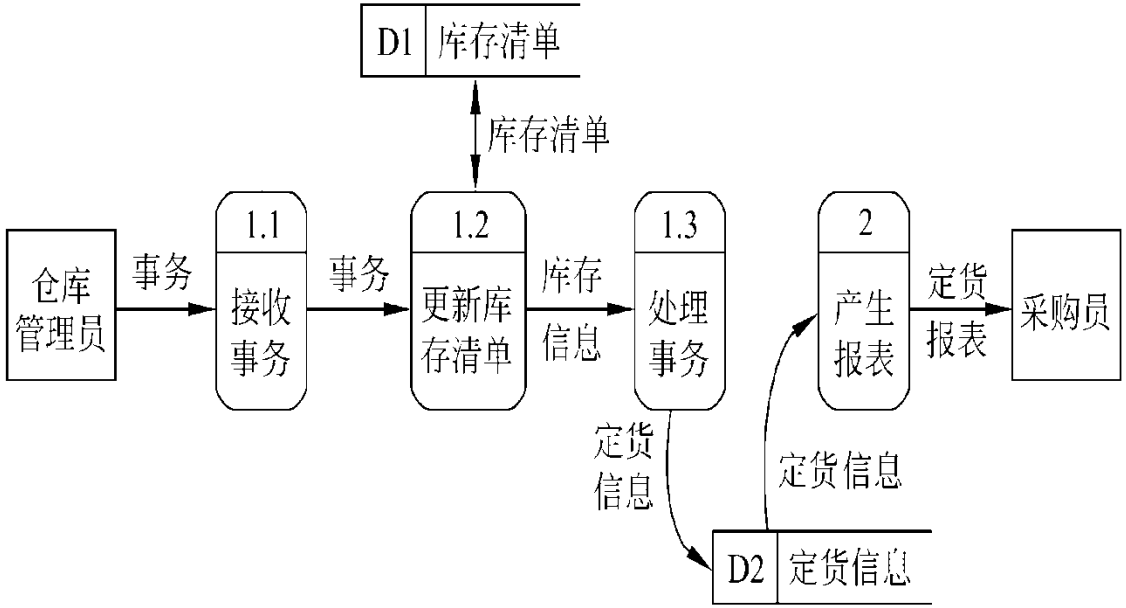
1．设想供选择的方案

（1）在数据流图的基础上，一个边界一个边界设想并列出供选择的方案。通常，选取的这些方案中至少应包括低成本、中成本和高成本的三种方案类型；

（2）对每个合理方案要提供以下几方面资料：

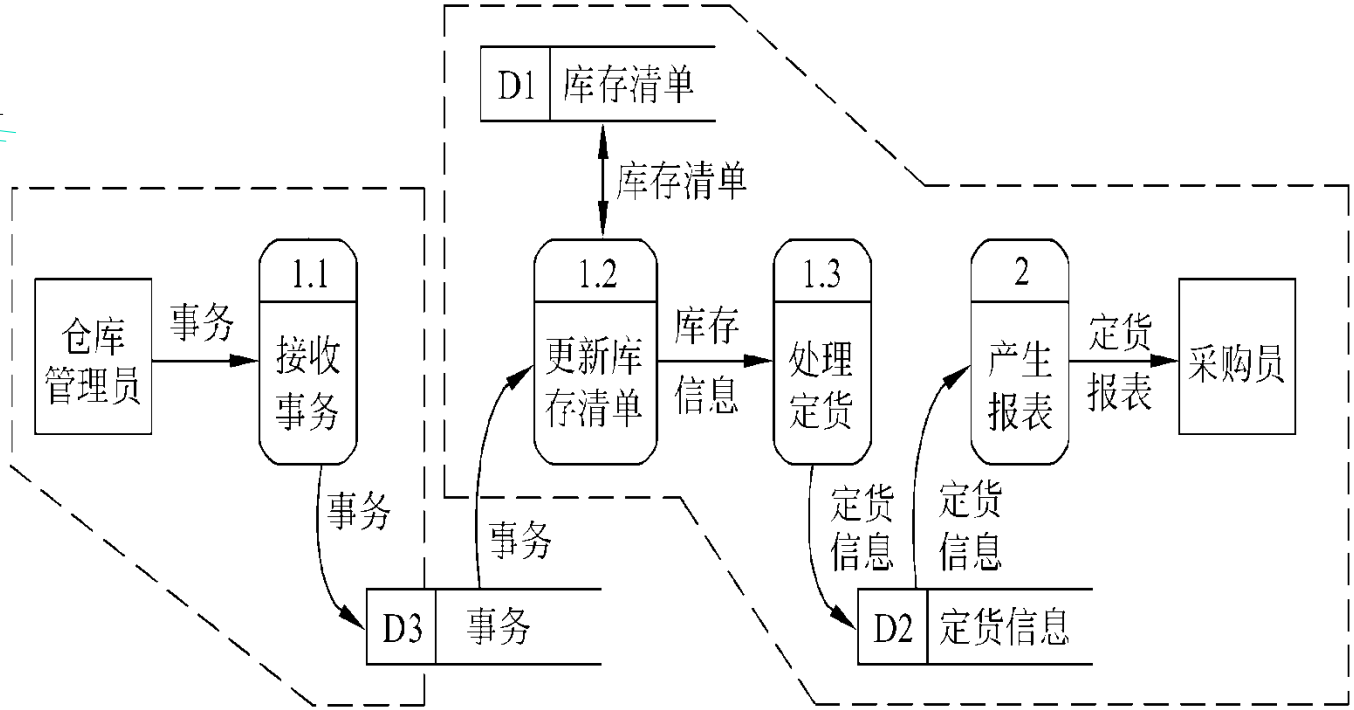
- 系统流程图；
- 数据字典；
- 成本／效益分析；
- 实现这个系统的进度计划。

实例分析：订货系统的数据流图



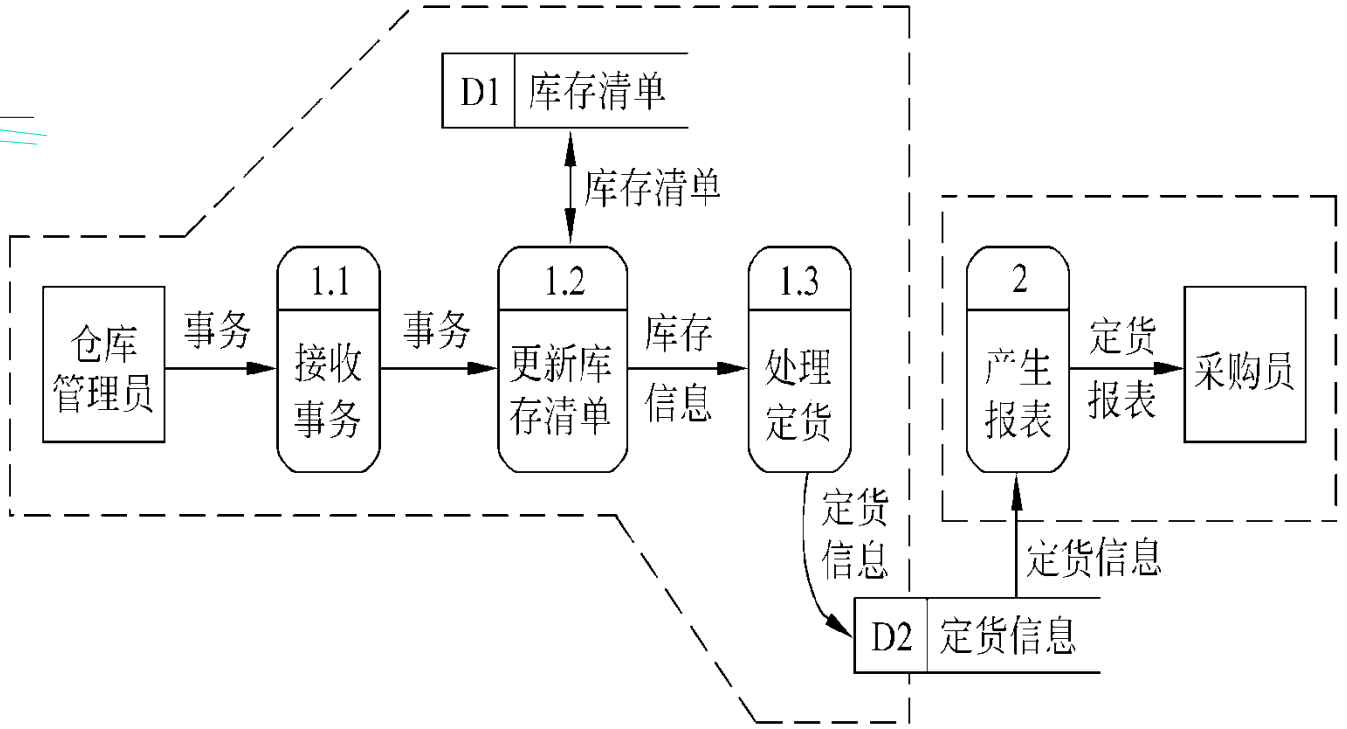
- 当用数据流图辅助物理系统的设计时，以图中不同处理的定时要求为指南，能够在数据流图上画出许多组自动化边界，每组自动化边界可能意味着一个不同的物理系统，因此可以根据系统的逻辑模型考虑系统的物理实现。

思考：从更新库存清单的角度看，下述设计方式的含义是什么？



这种划分自动化边界的方法暗示以批量方式更新库存清单

思考：从更新库存清单的角度看，下述设计方式的含义又是什么？



这种划分自动化边界的方法建议以联机方式更新库存清单

2. 选取合理的方案

从上一步得到的一系列供选择的方案中选取若干个合理的方案，通常至少选取低成本、中等成本和高成本的三种方案；

根据系统分析确定的目标，来判断哪些方案是合理的；

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，并为最佳方案制定详细的实现计划。

4. 功能分解

需要从实现的角度把复杂的功能进一步分解。

5. 设计软件结构

软件结构反映系统中模块的相互调用关系：顶层模块调用它的下层模块以实现程序的完整功能，每个下层模块再调用更下层的模块，从而完成程序的一个子功能，最下层的模块完成最具体的功能；

软件结构通过层次图或结构图来描绘，也可以直接从数据流图映射出软件结构。

6. 设计数据库

数据库设计是一项专门的技术，包括模式设计、子模式设计、完整性和安全性设计和优化处理等。

7. 制定测试计划

在软件开发的早期阶段提前考虑软件的测试计划是很有必要的。这样能促使软件设计人员在设计时注意到软件的测试问题，从而有利于提高软件的可测试性。

8. 书写文档

系统说明、用户手册、测试计划、详细的实现计划、数据库设计结果。

8. 审查和复审

- 先技术审查
- 后管理复查

第二节　设计原理

一、模块化

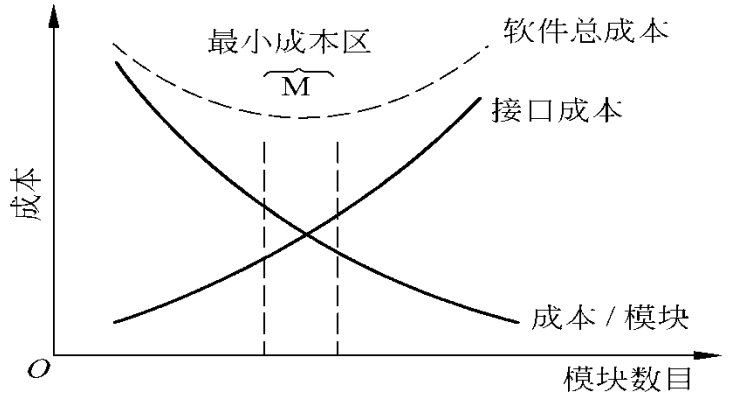
模块是数据说明、可执行语句等程序对象的集合，它是单独命名的而且可通过名字来访问。例如，过程、函数、子程序、宏等等都可作为模块。

1. 理想模块的特点

- 每个理想模块只解决一个问题；
- 每个理想模块的功能都应该明确，使人容易理解；
- 理想模块之间的联结关系简单，具有独立性；

- 由理想模块构成的系统，容易使人理解，易于编程，易于测试，易于修改和维护。

2. 模块化和软件成本的关系



3. 采用模块化原理的优点：

- 可以使软件结构清晰，容易设计、容易阅读和理解、容易测试和调试；
  - 提高软件的可靠性；
  - 有助于软件开发工程的组织管理。

二、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，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；
- 在最低的抽象层次上，用可直接实现的方式叙述问题的解法。

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。

三、逐步求精

- 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。

四、信息隐藏和局部化

- 信息隐藏是指每个模块的实现细节对于其它模块来说是隐藏的。也就是说，模块中所包含的信息（包括数据和过程）不允许其它不需要这些信息的模块访问。
- 局部化是指把一些关系密切的软件元素物理地放的彼此靠近。

五、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， 而且和软件系统中其它的模块的接口是简单的。

- 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。
- 一般采用两个准则度量模块独立性。即模块间耦合和模块内聚。

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。

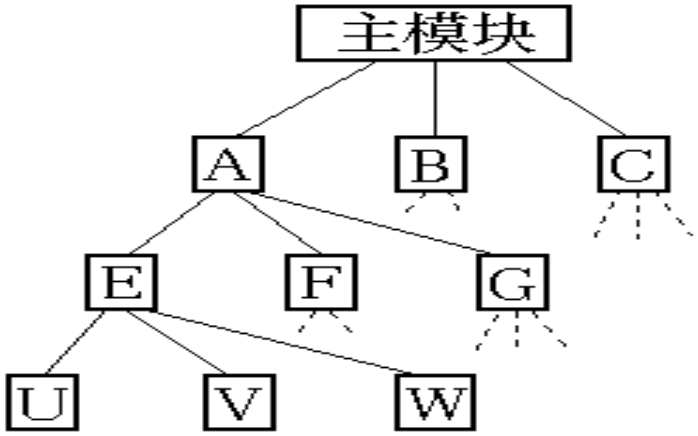
1. 耦合

- 耦合是对一个软件结构内不同模块之间的互相连接的紧密程度的度量。
- 耦合强弱取决于各个模块之间接口的复杂程度、调用模块的方式以及哪些信息通过接口。
- 一般模块间可能的连接方式有 7 种，构成耦合的 7 种类型：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。

1. 耦合

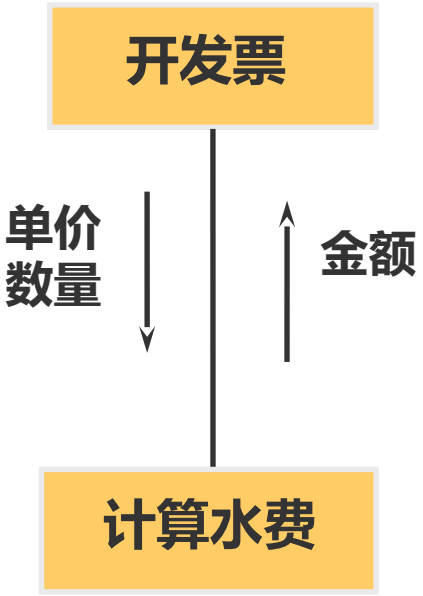
(1) 非直接耦合

- 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。



(2) 数据耦合

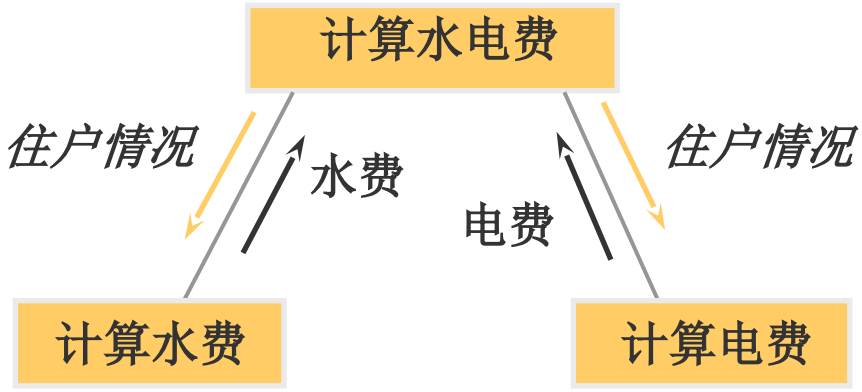
- 一个模块访问另一个模块时，彼此之间是通过简单数据参数 （不是控制参数、公共数据结构或外部变量） 来交换输入、输出信息的，称此为数据耦合。
- 数据耦合是松散的耦合，模块间的独立性比较强，在软件程序结构中至少必须有这种耦合。



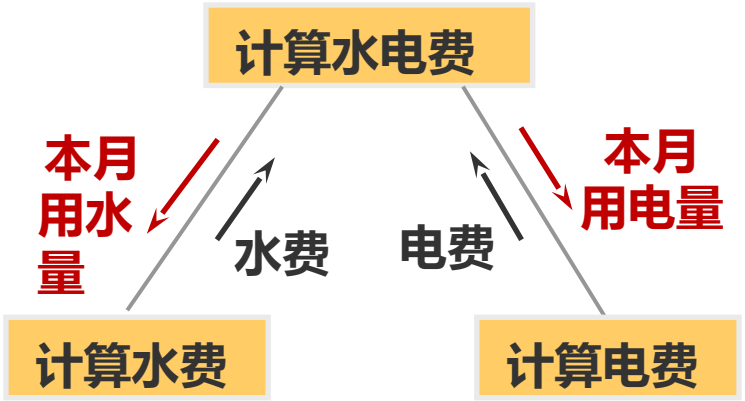
(3) 标记耦合（特征耦合）

- 一组模块通过参数表传递记录信息，就是标记耦合。这组模块共享了这个记录，它是某一数据结构的子结构，而不是简单变量。
- 在设计中应尽力避免这种耦合（但允许使用），它使在数据结构上的操作复杂化了。

标记耦合举例：



将标记耦合修改为数据耦合：

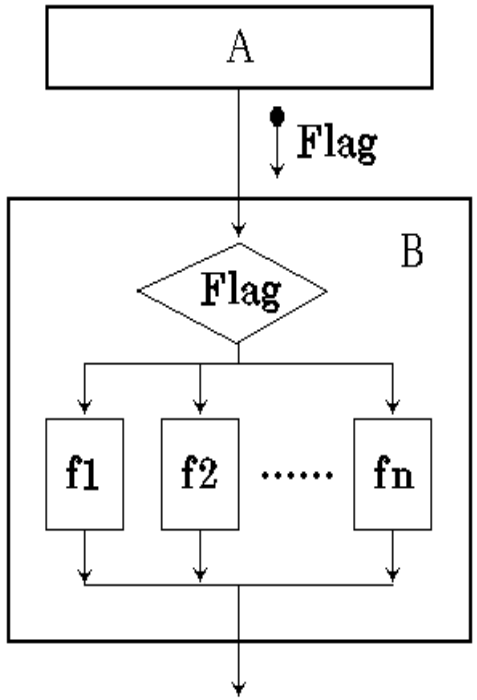


“住户情况”是一个数据结构，图中模块都与此数据结构有关。“计算水费”和“计算电费”本无关，由于引用了此数据结构产生依赖关系，它们之间也是标记耦合。

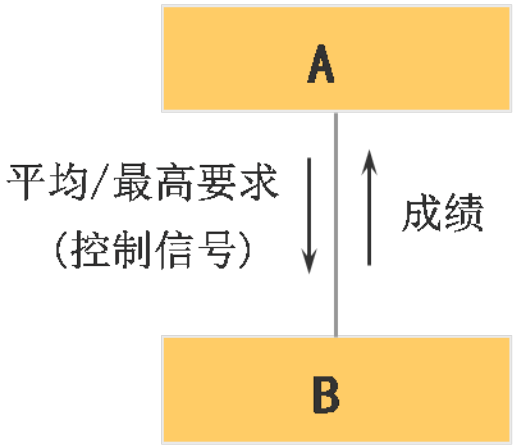
如何将标记耦合修改为数据耦合？

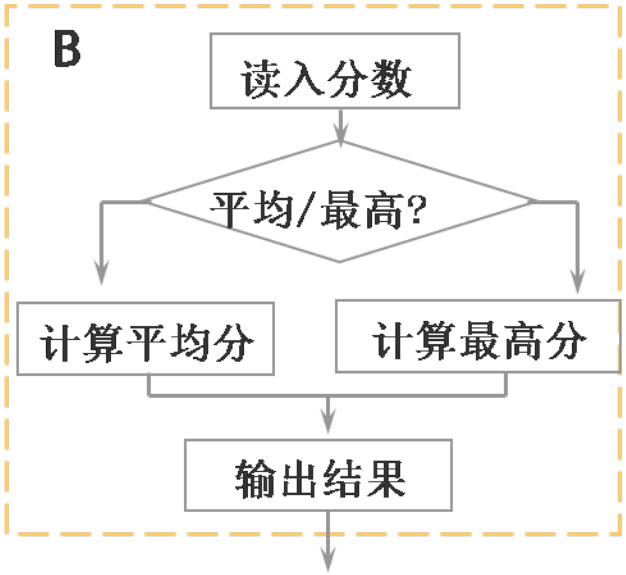
(4) 控制耦合

- 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。
- 控制耦合意味着控制模块必须知道所控制模块内的一些逻辑关系，这样会降低模块的独立性。



控制耦合举例：





（B 模块：计算平均分或最高分）

- 控制耦合增加了理解和编程的复杂性，调用模块必须知道被调模块的内部逻辑，增加了相互依赖。

去除模块间控制耦合的方法：

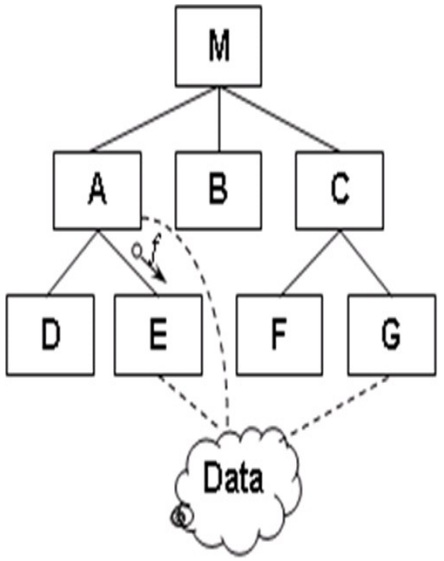
- 将被调用模块内的判定上移到调用模块中进行；
- 被调用模块分解成若干单一功能模块。

(5) 外部耦合

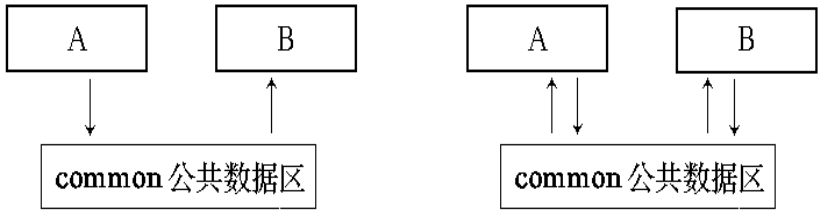
- 一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。
- 例如： C 语言程序中各个模块都访问被说明为 extern 类型的外部变量。

(6) 公共耦合

- 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、共享的通信区、内存的公共覆盖区等。
- 只有在模块之间共享的数据很多，而且通过参数表传递不方便时，才使用公共耦合，否则，还是使用模块独立性比较高的数据耦合较好些。



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，即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

公共耦合会引起下列问题：

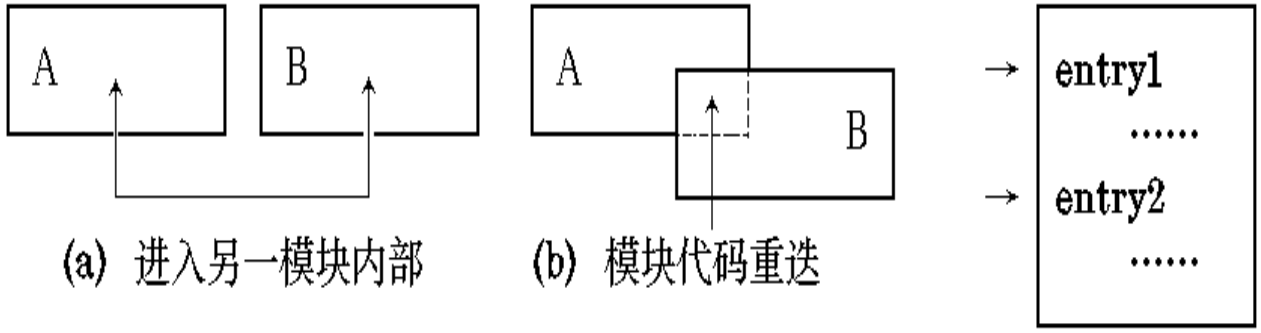
- 所有公共耦合模块都与某一个公共数据环境内部各项的物理安排有关，若修改某个数据的大小，将会影响到所有的模块；
- 无法控制各个模块对公共数据的存取，严重影响软件模块的可靠性和适应性；
- 公共数据名的使用，明显降低了程序的可读性。

请慎用公共数据区和全程变量！！！

(7) 内容耦合

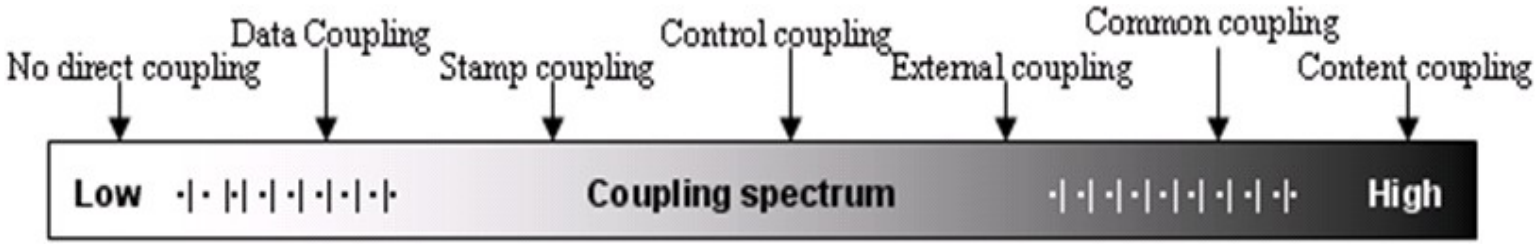
如果发生下列情形，两个模块之间就发生了内容耦合：

- 一个模块直接访问另一个模块的内部数据；
- 一个模块不通过正常入口转到另一模块内部；
- 两个模块有一部分程序代码重迭；
- 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

耦合谱系：



模块独立性比较强的模块应是低耦合的模块。

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，限制公共耦合的范围，完全不用内容耦合。

2. 内聚

- 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；
- 内聚是信息隐藏和局部化概念的自然扩展，它标志一个模块内部各成分彼此结合的紧密程度；
- 一般模块的内聚性分为七个类型：

  功能内聚、顺序内聚、通信内聚、过程内聚、时间内聚、逻辑内聚、偶然内聚。

(1) 功能内聚

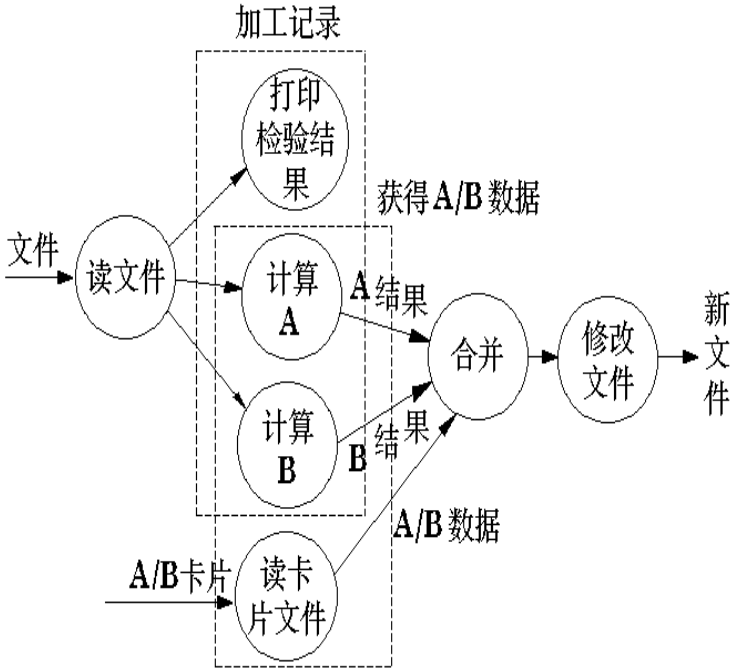
- 一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的。则称该模块为功能内聚模块。
- 功能内聚模块的优点是模块的内聚程度最高，容易修改和维护，模块间的耦合是简单的。

(2) 顺序内聚

亦称信息内聚，模块中所有处理元素和同一个功能密切相关，且这些处理必须顺序执行。通常，顺序内聚的模块，其前一处理元素的输出数据是后一处理元素的输入数据。

（3）通信内聚

如果一个模块内各功能部分都使用了相同的输入数据，和（或）产生了相同的输出数据，则称之为通信内聚。



(4) 过程内聚

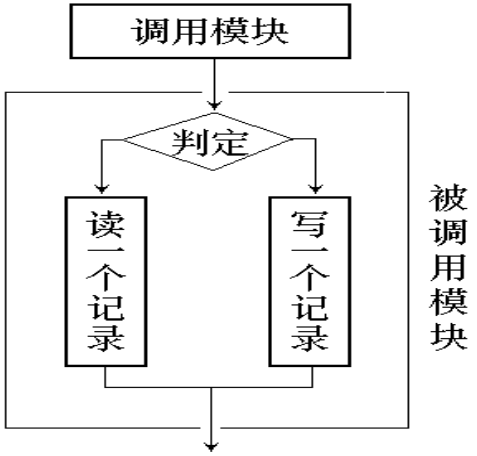
- 如果一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。
- 使用流程图做为工具设计软件时，常常通过研究流程图确定模块的划分，这样得到的往往是过程内聚的模块。

(5) 时间内聚

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、终止模块、紧急故障处理模块等均是时间性聚合模块。

(6) 逻辑内聚

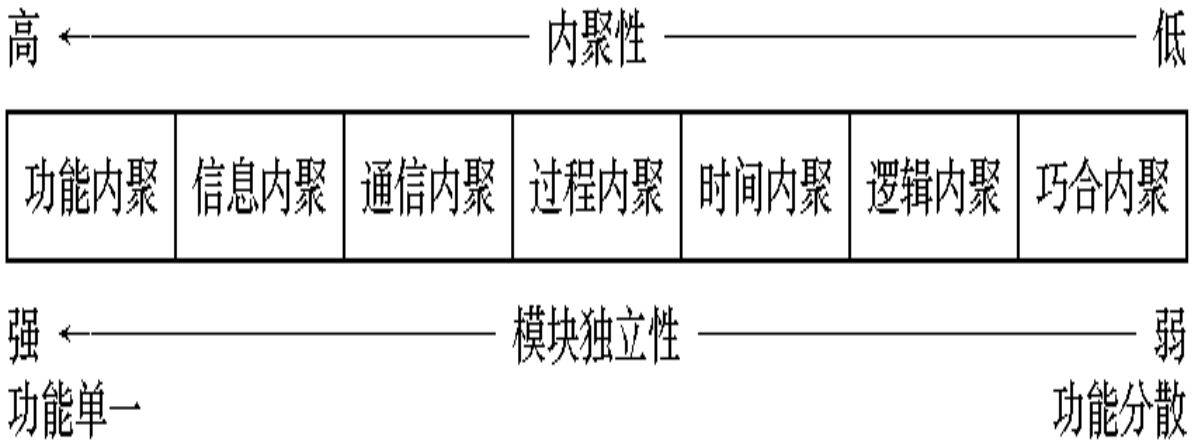
这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。



(7) 巧合内聚

亦称偶然内聚，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。

内聚谱系：



模块独立性比较强的模块应是高内聚的模块。

第三节　总体设计准则

一、改进软件结构，提高模块独立性

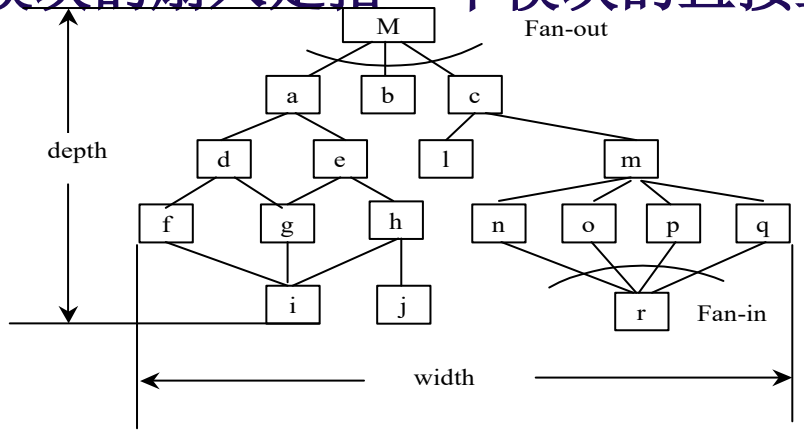
设计出软件的初步结构以后，应该审查分析这个结构，通过模块分解或合并，力求降低耦合提高内聚。

二、模块规模适中，功能单一

规模过大不易理解；规模过小，数量增加，系统接口复杂。一般一个模块内包含的语句在 30-50 条左右较好（指高级语言）。

三、模块的深度、宽度、扇出和扇入

- 模块的深度表示软件结构中控制的层数；
- 模块的宽度是软件结构内同一个层数上的模块总数的最大值；
- 模块的扇出指一个模块直接控制（调用）的模块数目。经验表明，设计得好的系统平均扇出通常是 3 或 4（扇出上限通常是 5-9）；
- 模块的扇入是指一个模块的直接上级模块的个数。



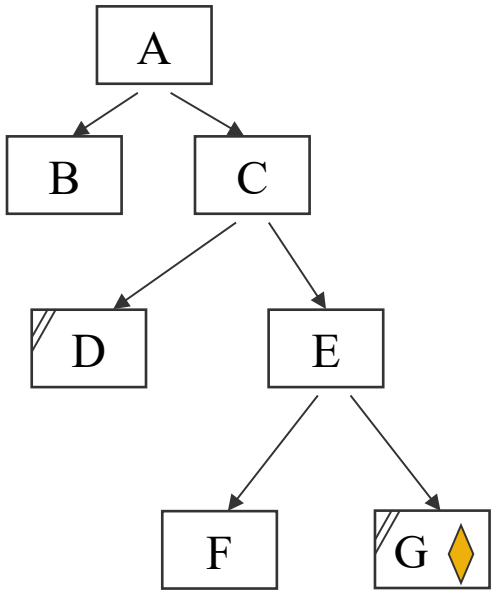
经验证明，一个设计得很好的软件模块结构，通常顶层扇出比较高，中层扇出比较少，底层模块有高扇入。

四、模块的作用域应该在控制域之内

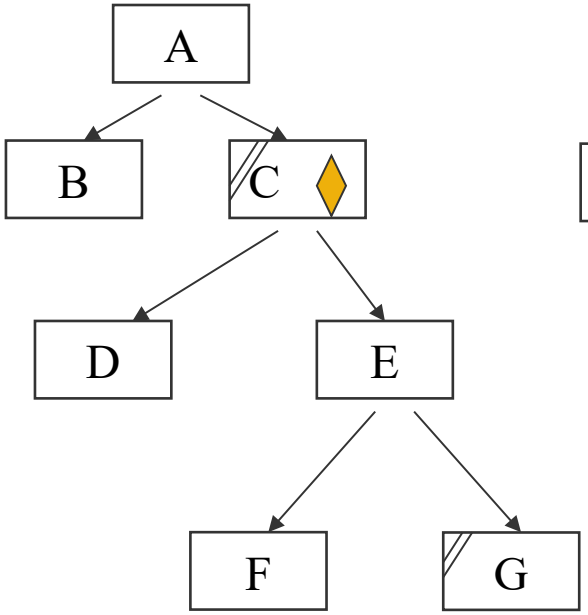
- 模块的控制域：模块本身及其所有直接或间接从属于它的下级模块。
  - 模块的作用域：受该模块内一个判断影响的所有模块的集合。
  - 原则：

A. 对于任何一个内部存在判断调用的模块，它的判断作用的范围应该是其控制范围的一个子集；

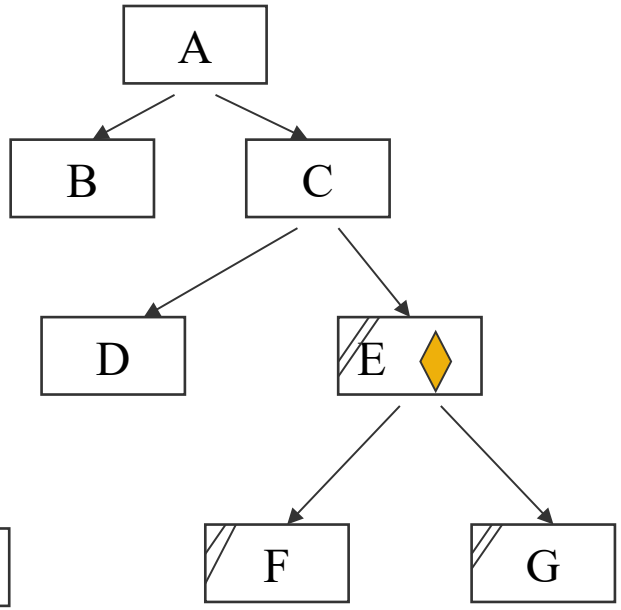
B. 存在判断调用的模块，所在层次不要与那些属于判断作用范围的模块所在的层次相隔太远。



模块 G中有一条判断调用D的语句，违反第 1 条原则



违反第 2 条原则



正确

五、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。应该仔细设计模块接口，使得信息传递简单并且和模块的功能一致。

六、设计单入口单出口的模块

不要使模块间出现内容耦合。

七、模块功能应该可以预测

输入的数据相同就会产生同样的输出，这个模块的功能就是可以预测的。

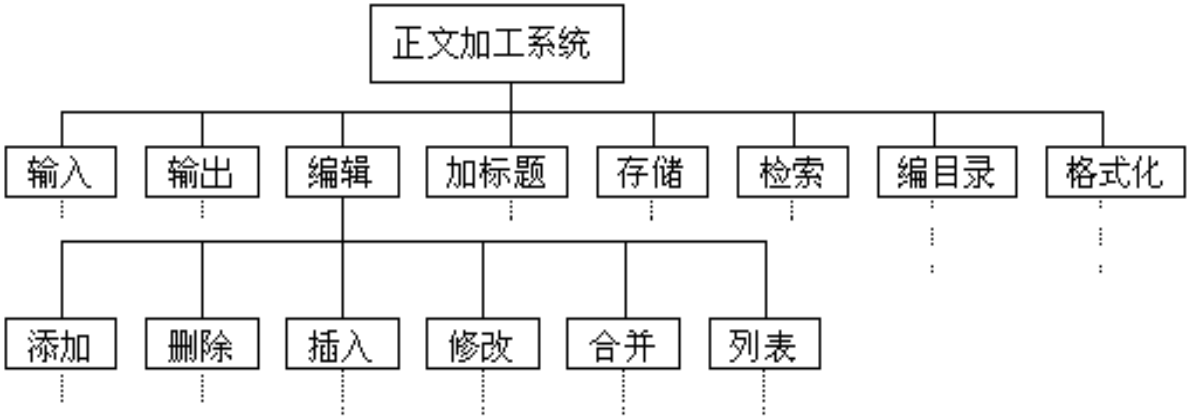
以上列出的启发式规则多数是经验规律，对改进设计，提高软件质量，往往有重要的参考价值；但是，它们既不是设计的目标，也不是设计时必须遵循的原则。

第四节　描绘软件结构的图形工具

一、层次图（H 图）和 HIPO 图

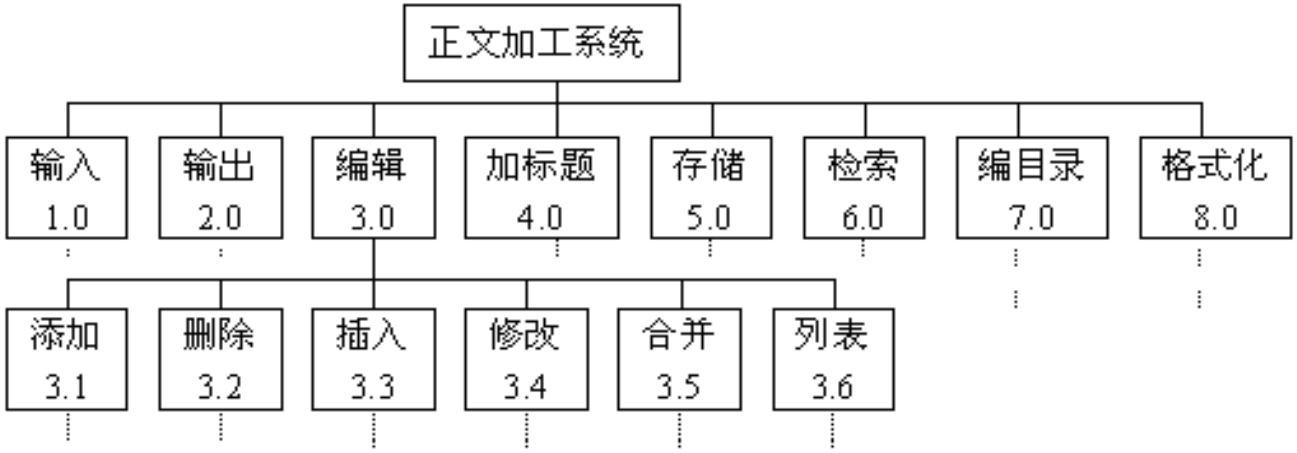
1. 层次图

层次图用来描绘软件的层次结构。层次图与前面学习的描绘数据结构的层次方框图在形式上相同，但表现的内容却完全不同：层次图中的一个矩形块代表一个模块，方框间的连线表示调用关系而不像层次方框图那样表示组成关系。



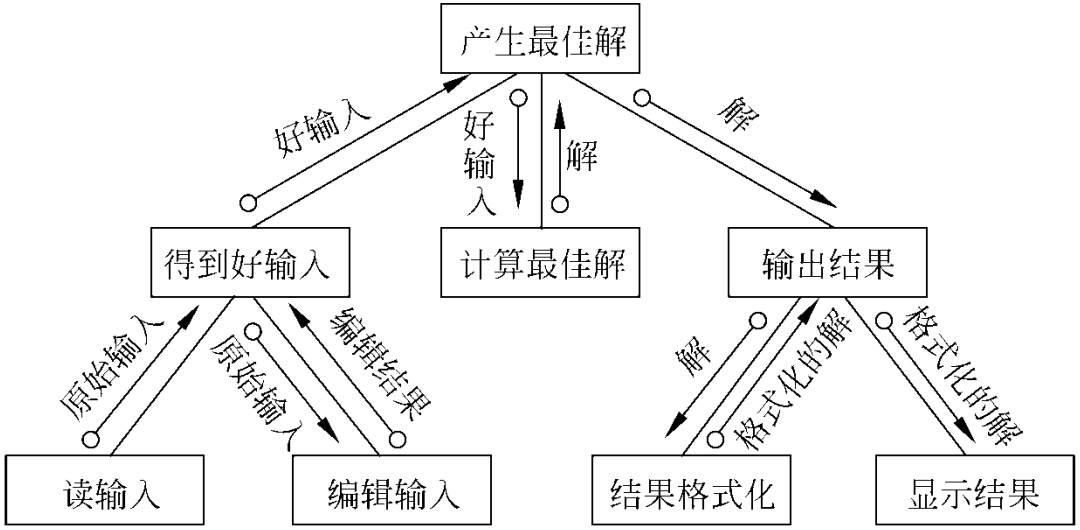
2. HIPO 图

HIPO 图是“层次 + 输入 / 处理 / 输出”的英文缩写。它在层次图的基础上，为除顶层外的每个方框都加上编号，以便于追踪模块在软件结构中的位置。与 H 图中每个方框相对应，有一张 IPO 图描绘该模块的处理过程。



二、结构图

- 一个方框代表一个模块，框内注明模块的名字或主要功能；
- 方框之间的箭头表示模块的调用关系；
- 尾端带有空心圆的短箭头表示数据信息，尾端带有实心圆的短箭头表示控制信息。



第五节　面向数据流的设计方法

面向数据流的设计方法的目标是给出设计软件结构的一个系统化的途径。

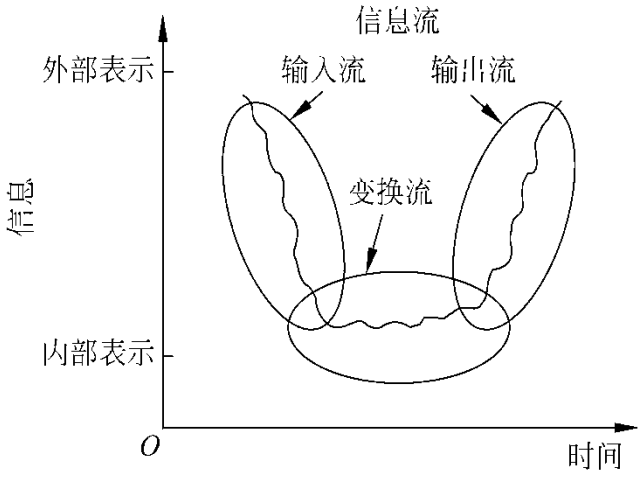
在软件工程的需求分析阶段，信息流是一个关键考虑，通常用数据流图描绘信息在系统中加工和流动的情况。面向数据流的设计方法定义了一些不同的“映射”，利用这些映射可以把数据流图变换成软件结构。因为任何软件系统都可以用数据流图表示，所以面向数据流的设计方法理论上可以设计任何软件的结构。

通常所说的结构化设计方法（简称 SD 方法），也就是基于数据流的设计方法。

一、概念

1. 变换流

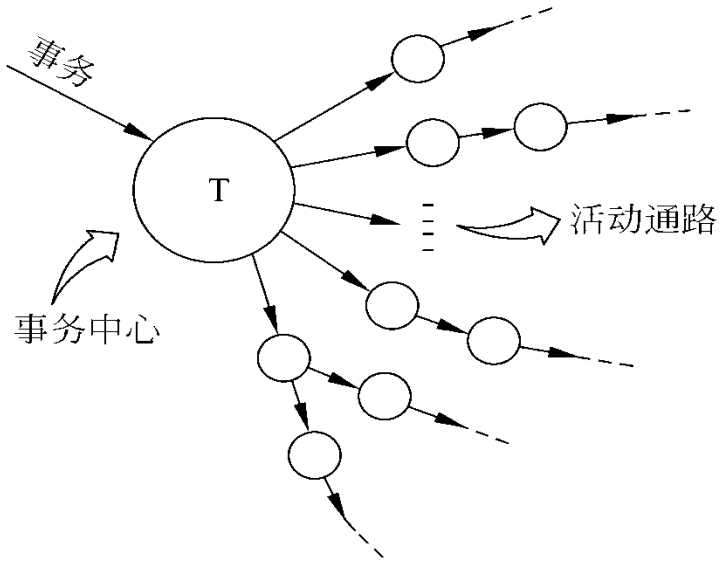
信息沿输入通路进入系统，同时由外部形式变换成内部形式，进入系统的信息通过变换中心，经过加工处理以后再沿输出通路变换成外部形式离开软件系统。当数据流图具有这些特征时，这种信息流称为变换流。



特点：从同一数据源进入系统的数据，它在数据流图中流动的逻辑路径是相同的。

2. 事务流

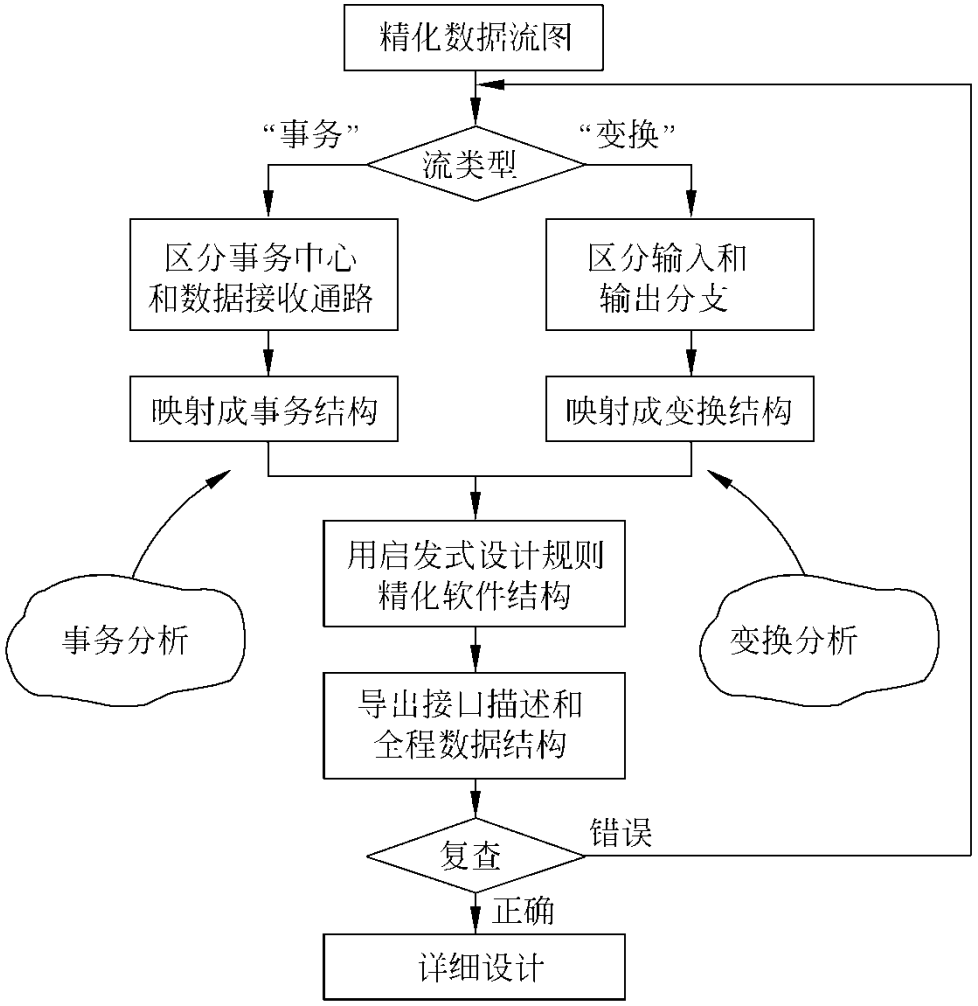
数据沿输入通路到达一个处理 T，这个处理根据输入数据的类型在若干个动作序列中选出一个来执行。这种“以事务为中心的”的数据流，称为“事务流”。



3. 设计过程

右图说明了使用面向数据流方法逐步设计的过程。

注意：任何设计过程都不是机械地一成不变的，设计首先需要人的判断力和创造精神，这更重要。



二、变换分析

变换分析是一系列设计步骤的总称，经过这些步骤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。变换分析方法如下：

1. 复查基本系统模型

复查的目的是确保系统的输入数据和输出数据符合实际。

2. 复查并精化数据流图

应该对需求分析阶段得出的数据流图认真复查，并且在必要时进行精化。不仅要确保数据流图给出了目标系统的正确的逻辑模型，而且应该使数据流图中每个处理都代表一个规模适中相对独立的子功能。

3. 确定数据流图具有变换特性还是事务特性

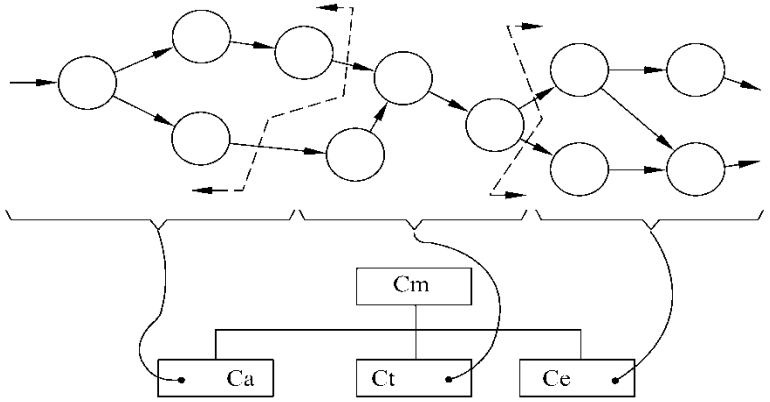
一般地说，一个系统中的所有信息流都可以认为是变换流。但是，当遇到有明显事务特性的信息流时，建议采用事务分析方法进行设计。

4. 确定输入流和输出流的边界，从而孤立出变换中心

- 检查“输入流”的边界；
- 检查“输出流”的边界；
- 得到变换中心。

5. 完成“第一级分解”

对于变换流的情况，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。下图说明了第一级分解的方法。



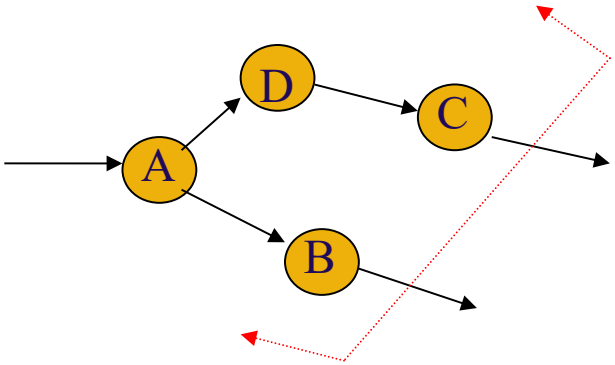
控制模块 Cm 协调下述从属的控制功能：

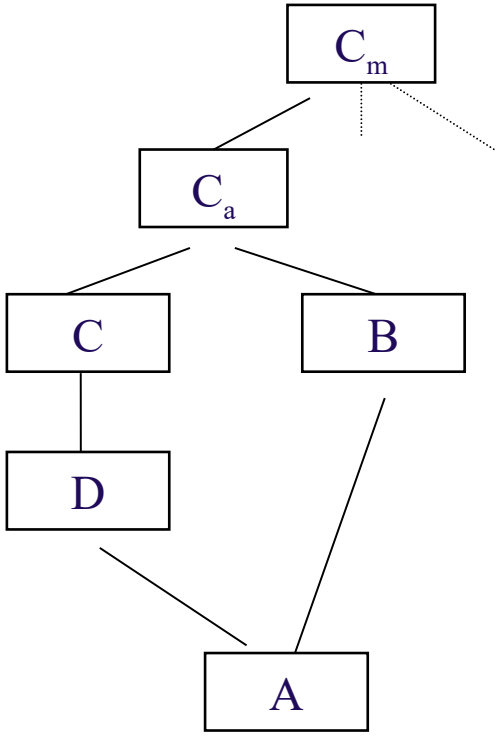
- 输入信息处理控制模块 Ca，协调对所有输入数据的接收；
- 变换中心控制模块 Ct，管理对内部形式的数据的所有操作；
- 输出信息处理控制模块 Ce ，协调输出信息的产生过程。

6. 完成“第二级分解”

所谓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沿着输入通路向外移动，把输入通路中每个处理逻辑映射成软件结构中 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逻辑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。





第二级分解的方法举例

7. 根据设计原理和总体设计准则，对第一次分割得到的软件结构进一步精化

为了得到一个易于实现、易于测试和易于维护的软件结构，根据软件设计的基本原则和其它启发性原则，对初步分割得到的模块进行再分解或合并。

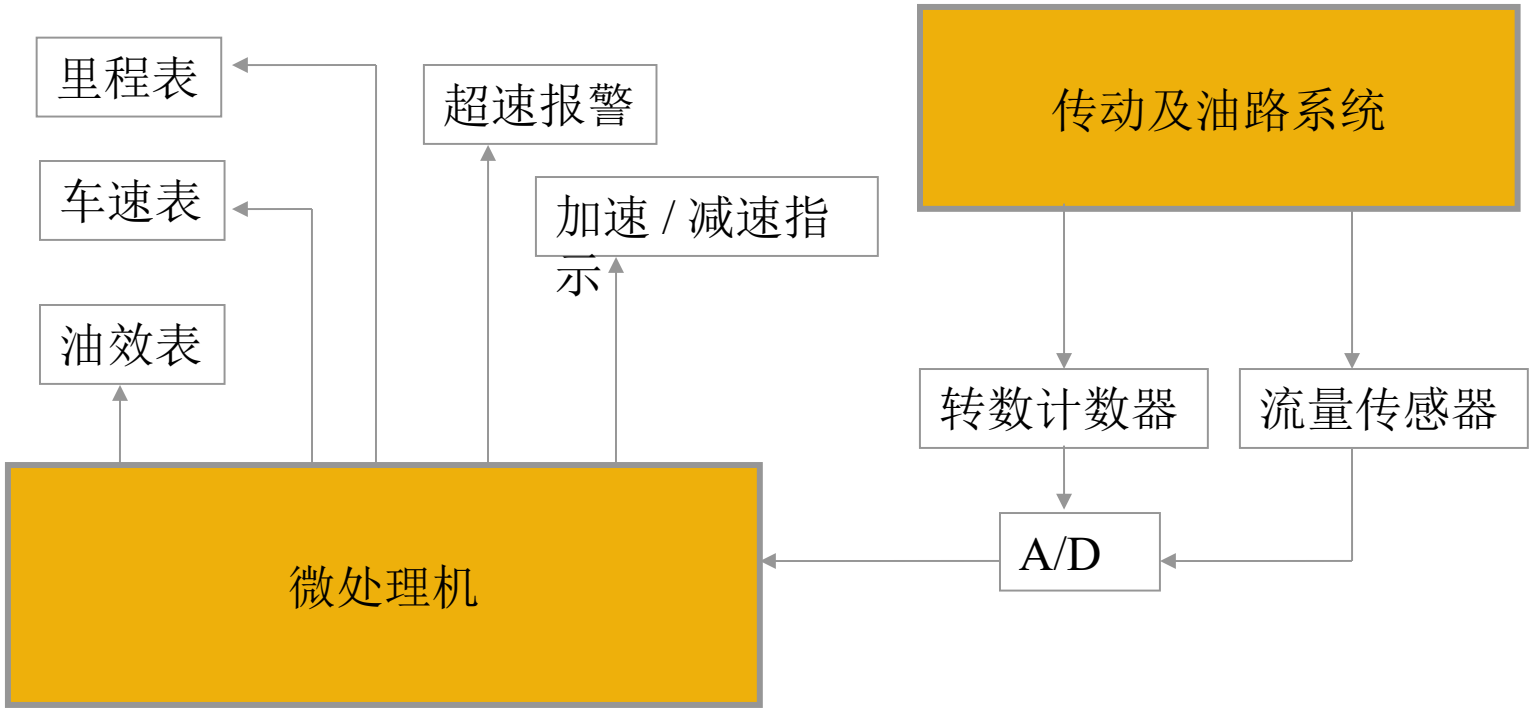
变换分析举例

假设的汽车数字仪表板将完成下述功能：

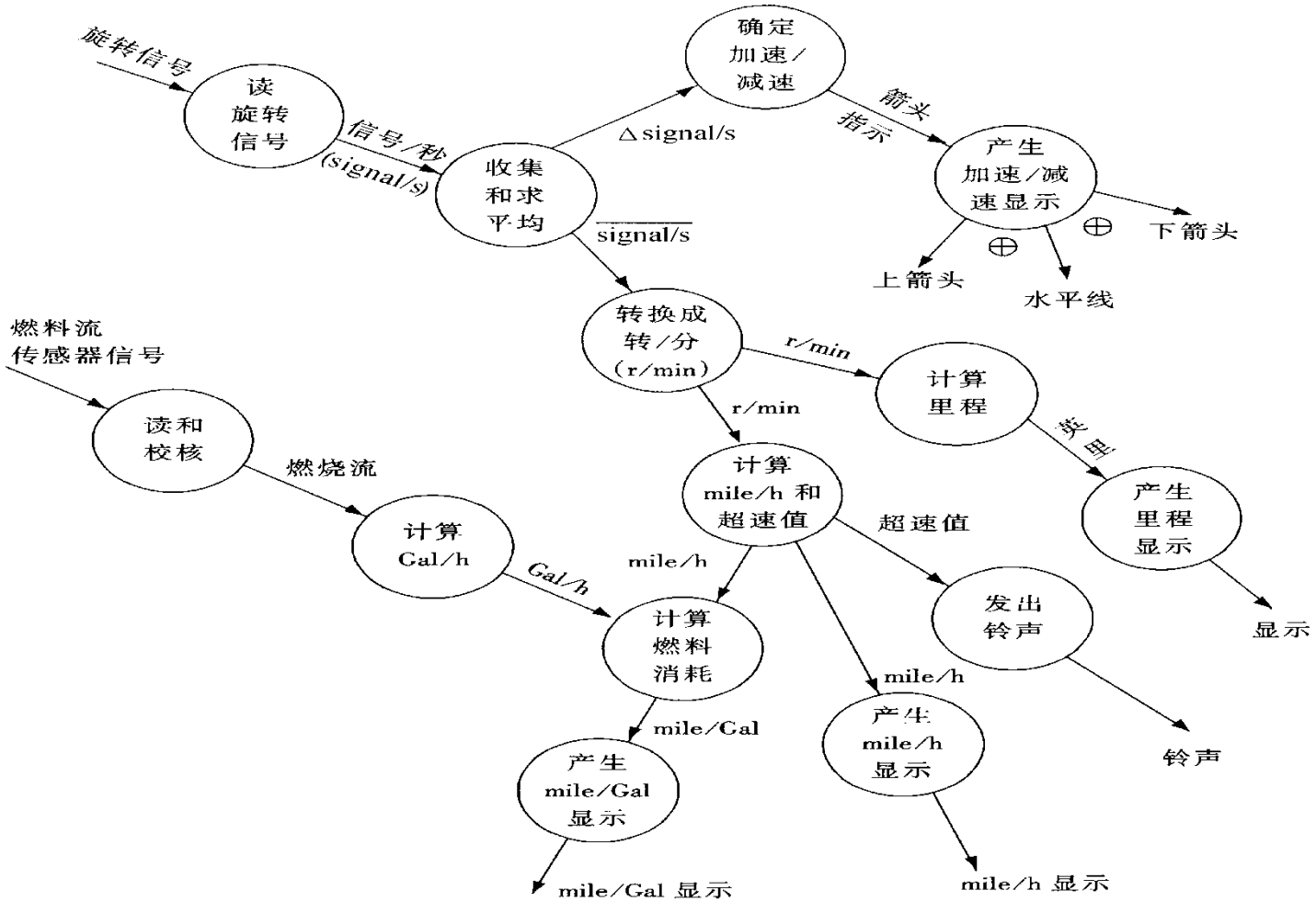
- 通过 A/D 转换实现传感器和微处理机接口；
- 在发光二极管面板上显示数据（如下第 3 条中的数据）；

- 指示每小时英里数（mile / h），行驶的里程，每加仑油行驶的英里数（mile / Gal）等等；
  - 指示加速或减速；
  - 超速警告：如果车速超过 120km / h，则发出超速警告铃声。

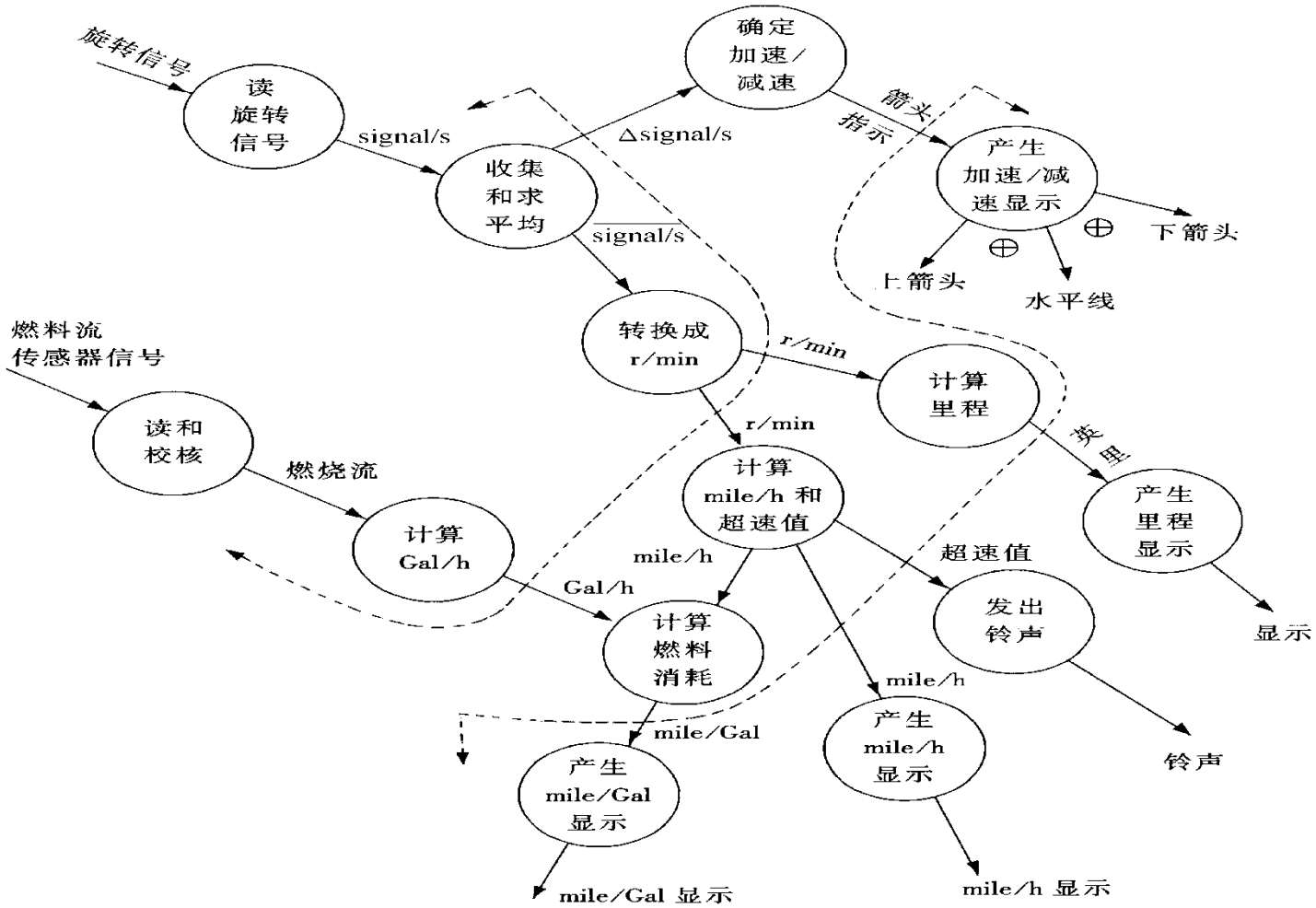
变换分析举例



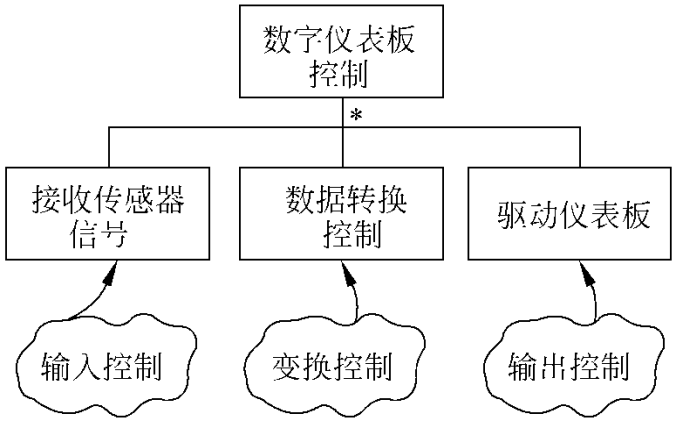
数字仪表板系统的数据流图



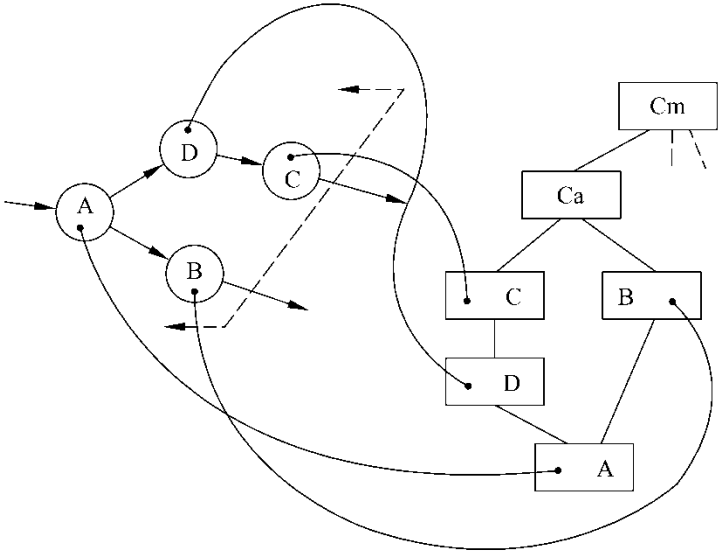
具有边界的数据流图


旋转信号
读旋转信号
signal/s
收集和求平均
△signal/s
确定加速/减速
箭头指示
产生加速/减速显示
下箭头
上箭头
水平线
signal/s
转换成r/min
r/min
计算里程
里程
产生里程显示
显示
燃料流传感器信号
读和校核
燃烧流
计算Gal/h
Gal/h
r/min
计算mile/h和超速值
mile/h
超速值
发出铃声
铃声
计算燃料消耗
mile/Gal
产生mile/Gal显示
mile/Gal 显示
mile/h
产生mile/h显示
mile/h 显示

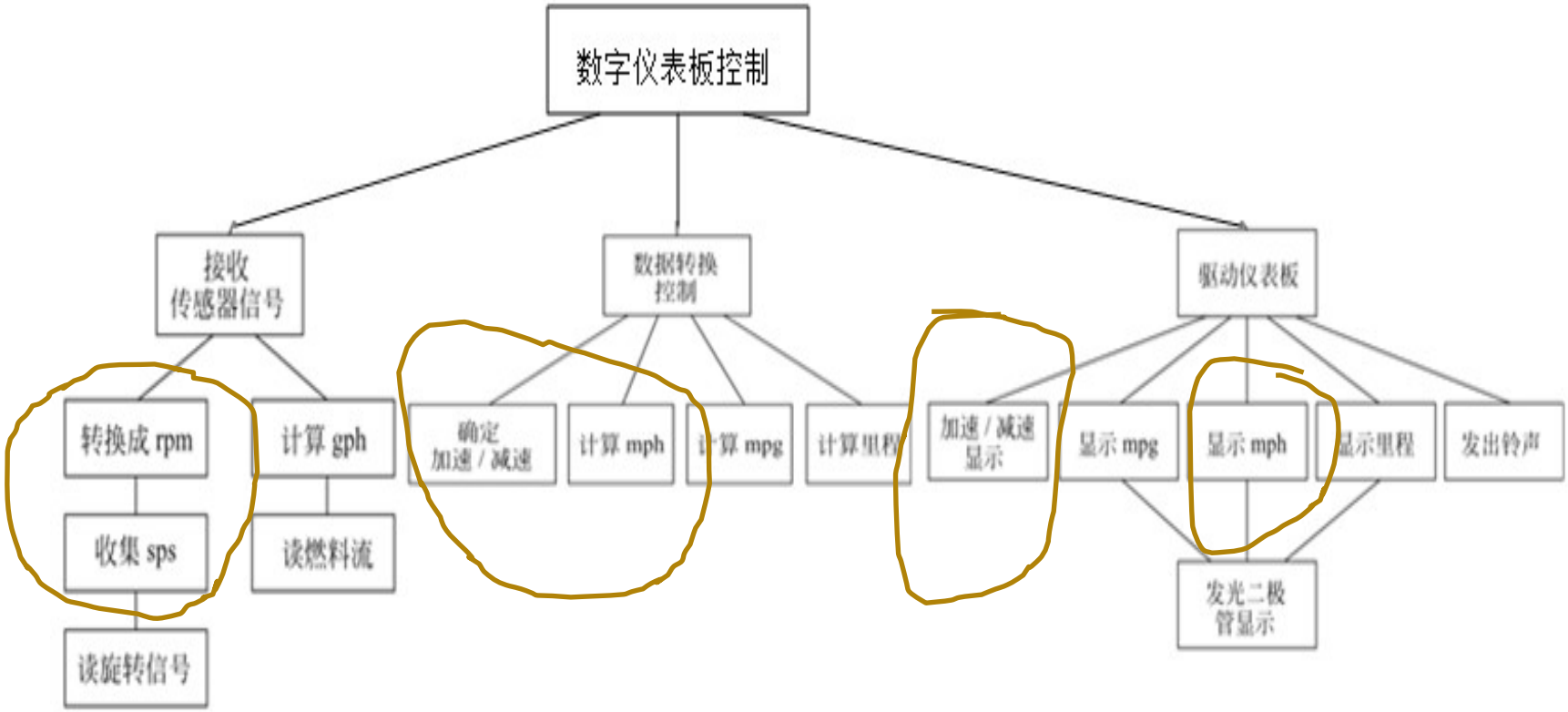
第一级分解



第二级分解的方法

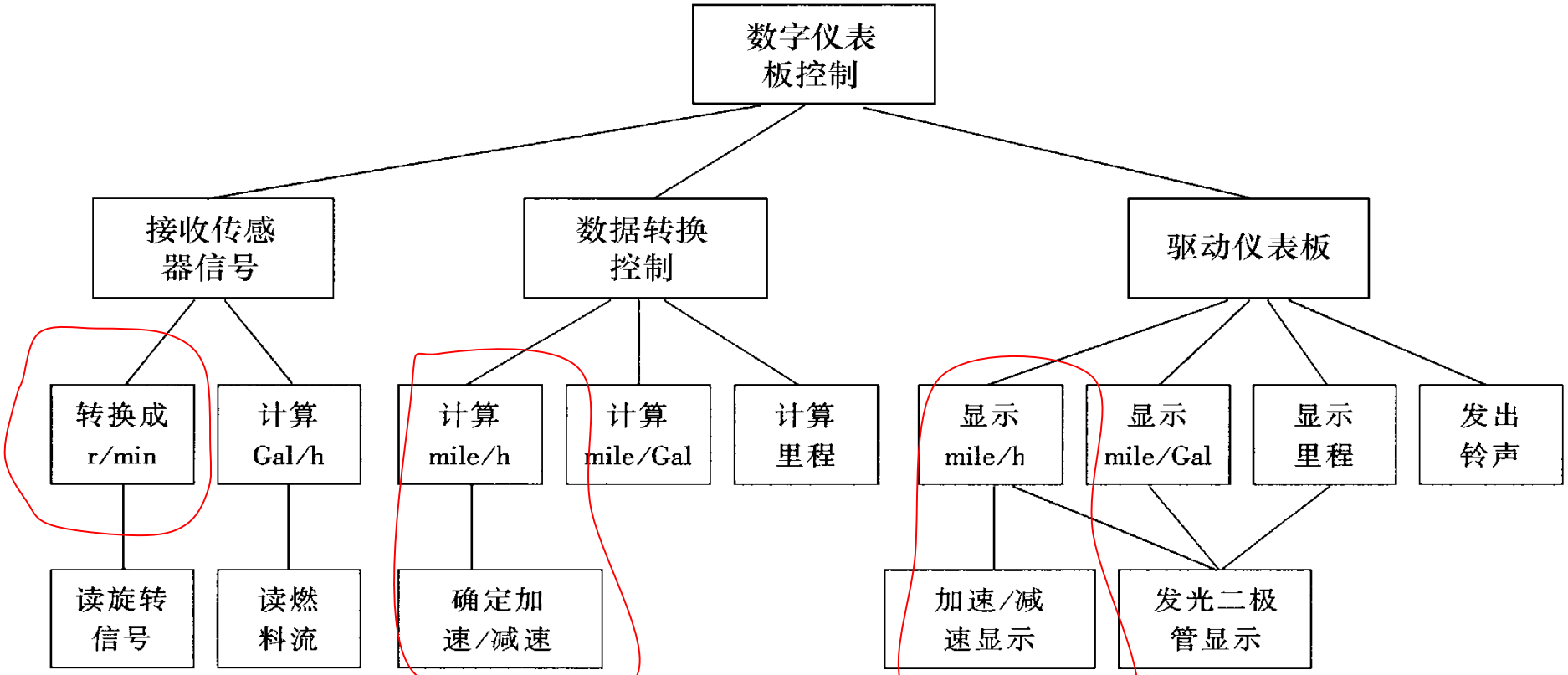


第二级分解：未经精化



输入结构　　变换结构　　输出结构

精化后的数字仪表板系统的软件结构



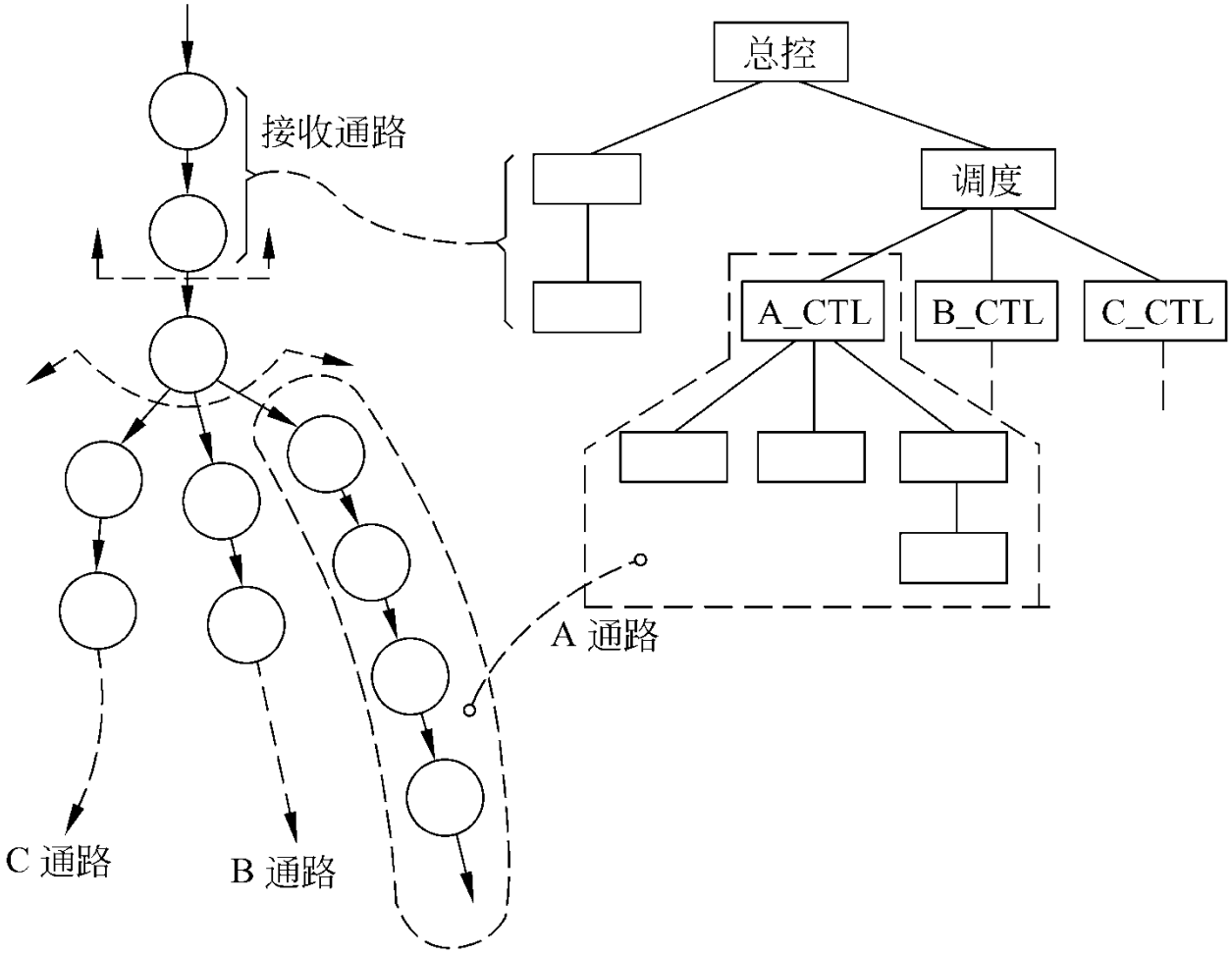
精化后的数字仪表板系统的软件结构

三、事务分析

事务分析的设计步骤和变换分析的设计步骤大部分相同或类似，主要差别仅在由数据流程图到软件结构的映射方法不同。由事务流映射成的软件结构包括一个接收分支和一个发送分支：

- 映射出接收分支结构的方法和变换分析映射出输入结构的方法相似，即从事务中心的边界开始，把沿着接收流通路的处理逻辑映射成模块。
- 发送分支的结构包含一个调度模块，它控制下层的所有活动模块；然后把数据流程图中的每一个活动流通路映射成与它的特征相对应的结构。

事务分析的映射方法



四、设计优化

考虑设计优化问题时应该记住，“一个不能工作的‘最佳设计’的价值是值得怀疑的”。软件设计人员应该致力于开发能够满足所有功能和性能要求，而且按照设计原理和总体设计准则衡量是值得接收的软件。应该在设计的早期阶段尽量对软件结构进行精化。可以导出不同的软件结构，然后对它们进行评价和比较，力求得到“最好”的结果。

五、面向数据流设计小结

- 对于一个大系统，常常把变换分析和事务分析应用到同一个数据流图的不同部分，由此得到的子结构形成“构件”，可以利用它们构造完整的软件结构；
- 应该灵活运用，合并不必要的控制模块，分解功能过分复杂的控制模块。
- 由于任何软件系统都可以用数据流图表示，因此，面向数据流的设计方法理论上可以设计任何软件的结构。通常所说的结构化设计方法（简称 SD 方法）也是基于数据流的设计方法。


Thank
You