第五章 总体设计

第五章　总体设计

5.1　总体设计的过程

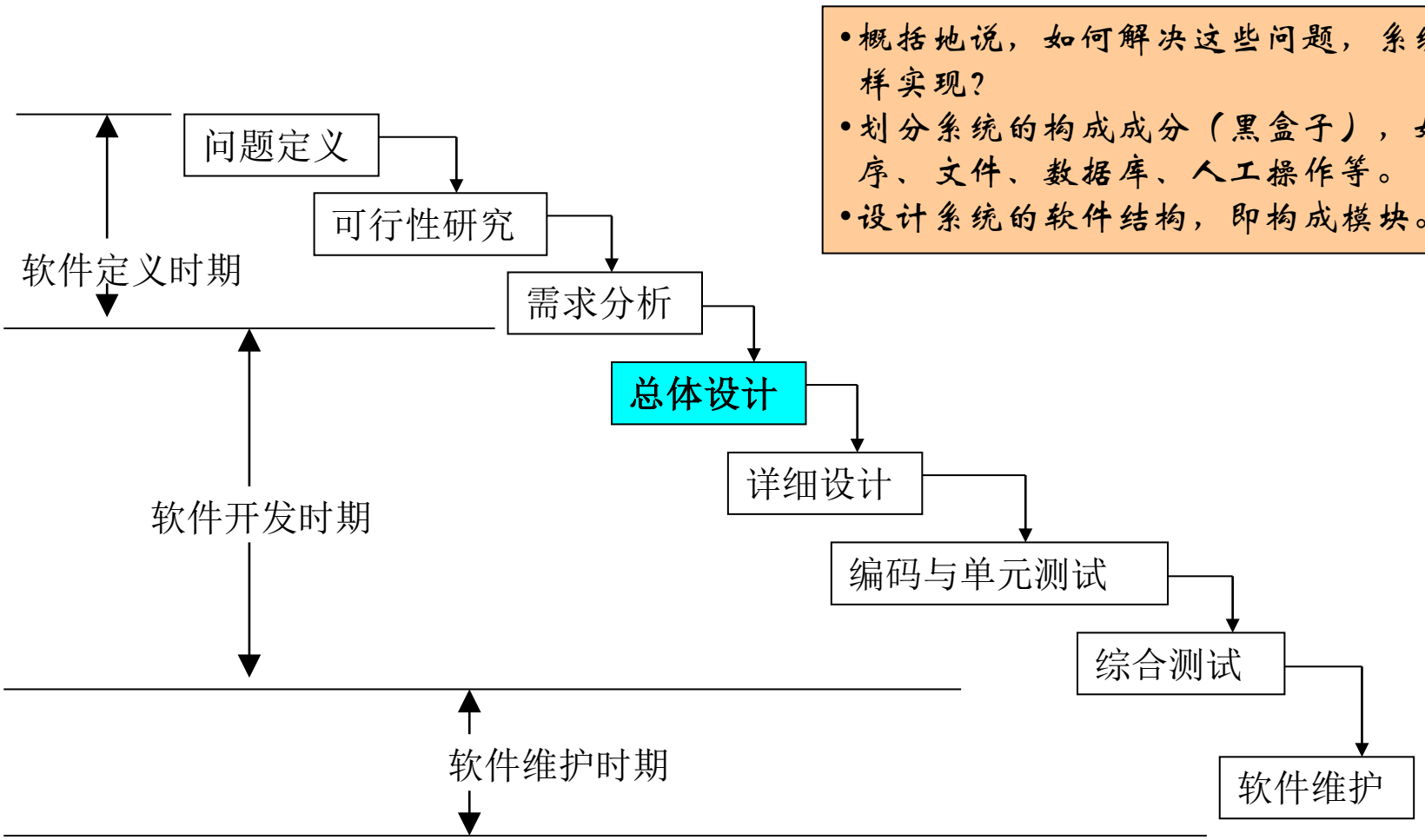
5.2　软件设计基本原理

5.3　设计准则

5.4　总体设计的图形描述工具

5.5　结构化设计方法

总体设计



- 概括地说，如何解决这些问题，系统怎样实现？
- 划分系统的构成成分（黑盒子），如程序、文件、数据库、人工操作等。
- 设计系统的软件结构，即构成模块。

第五章　总体设计

5.1　总体设计的过程

5.2　软件设计基本原理

5.3　设计准则

5.4　总体设计的图形描述工具

5.5　结构化设计方法

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

设想方案

- 需求分析阶段得出的数据流图是总体设计的根本出发点。用不同思路对需求分析中的数据流图的处理进行分组(不同的处理边界和处理方法)。

可供选择的方案

- 从实用的、可操作性的角度，去除不合理的方案，留下合理的方案进入下一步的选取和设计。

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

合理方案的选取：

- 在上一步设计出的方案中选取合理的方案。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。

对每个合理方案要提供以下几方面资料：

（1）系统流程图；
（2）数据字典；
（3）成本／效益分析；
（4）实现这个系统的进度计划。

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

推荐方案

- 选择一个最佳方案，并制定详细的实现计划。

审查方案

1. 对于最佳方案，先提请使用部门负责人审批。
2. 在使用部门负责人接受了分析员所推荐的方案之后，方可进入总体设计过程的下一步工作，即结构设计阶段。

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

大型系统的设计分为两个阶段：

- 结构设计：确定系统由哪些模块组成，以及这些模块之间的相互关系。（总体设计）
- 过程设计：确定每个模块的处理过程。（详细设计）

功能的分解：

- 为了确定软件结构（下一步），需要对软件的功能进行分解。功能分解可通过分析和细化数据流图中的处理而实现。

功能的描述：

- 完成功能分解后，应采用适当的工具（如IPO图、IPO表）描述每个处理的算法。

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

软件结构

- 软件结构是指软件各功能模块之间的关系，如层次关系、调用/被调用关系等。
- 层次关系：软件结构设计将软件模块组织成良好的层次关系：顶层、下层、更下层...。
- 调用/被调用关系：软件功能模块通过调用其下层子模块，实现所需要完成的功能。

软件结构的描述

- 层次方框图
- Warnier图

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

数据库设计

- 数据库设计指构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求（信息要求和处理要求）。

数据库设计内容

- 模式设计，子模式设计
- 完整性
- 安全性设计
- 优化处理等(如索引优化)

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

软件测试计划的主要内容：

1. 测试目的
2. 测试范围
3. 被测试/不被测试的特性
4. 测试方法
5. 测试的通过标准
6. 测试环境要求
7. 测试人员要求和职责分配
8. 测试进度要求
9. 测试风险和应急措施

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

总体设计的文档

- 总体设计说明书（包括系统实现方案和软件模块结构）；
- 测试计划（包括测试策略、测试方案、预测的测试结果、测试进度计划等）；
- 用户手册（根据总体设计阶段的结果，编写的初步的用户操作手册）；
- 细化的实现计划；
- 数据库设计。

5.1 总体设计的过程

1. 设计供选择的方案
2. 选取合理的方案
3. 推荐最佳实现方案
4. 功能分解
5. 软件结构设计
6. 数据库设计
7. 制定测试计划
8. 编写文档
9. 审查和复审

技术审查

- 召开项目小组评审会，从技术角度审查总体设计的合理性、正确性、可追溯性等。

管理复审

- 从管理者的角度，审查总体设计中组织管理工作规范性、文档的规范性和完整性等。

第五章　总体设计

5.1　总体设计的过程

5.2　软件设计基本原理

5.3　设计准则

5.4　总体设计的图形描述工具

5.5　结构化设计方法

5.2 软件设计基本原理

1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐蔽
5. 模块独立性

5.2 软件设计基本原理

1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐蔽
5. 模块独立性

5.2.1 模块化—概念

模块

- 模块是由一个标识符所代表的、由边界元素（如Java中的｛…｝）限定的相邻程序元素（数据/变量说明、可执行语句）、实现特定功能的序列。
- 模块又称构件，是能够单独命名并独立地完成一定功能的程序语句的集合。例如高级语言中的过程、函数、子程序等都可作为模块。

模块化是软件的一个重要属性

- 模块化的特性提供了人们处理复杂的问题的一种方法，同时也使得软件能够被有效地管理。

5.2.1 模块化—模块化特性

设函数：

- C(x)表示问题x的复杂程度；
- E(x)表示解决问题x所需要的工作量（时间）。

对于两个问题P1和P2，

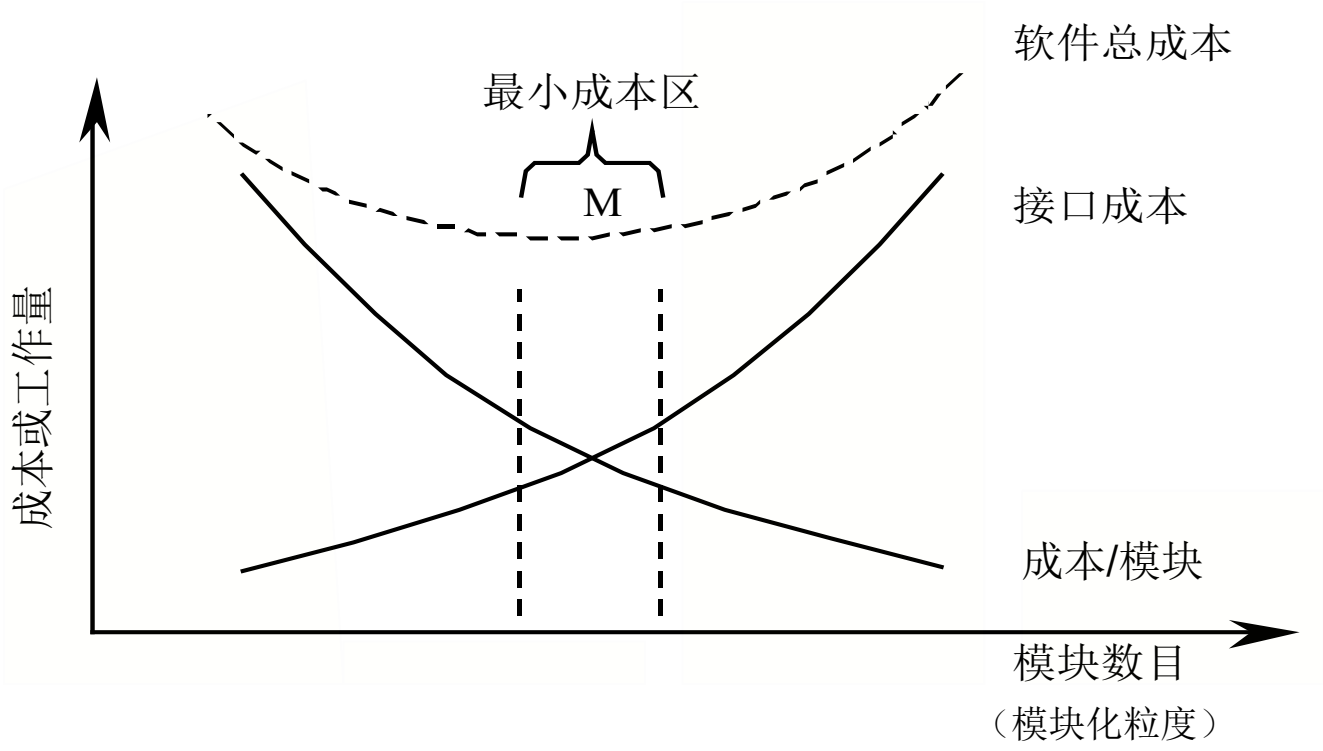
- 如果：C(P1) > C(P2)，则：E(P1) > E(P2)
- 另一个特性是：C(P1 + P2) > C(P1) + C(P2)

根据上面的结论，还有：

- E(P1 + P2) > E(P1) + E(P2)
- 这个不等式表明：单独解决问题P1和P2所需的工作量之和，比把P1和P2合起来作为一个问题来解决时所需的工作量要少。

这种“分而治之”的思想提供了模块化的根据：把复杂的问题分解成许多容易解决的小问题，原来的问题就容易解决了。

5.2.1 模块化—模块化和软件成本的关系



5.2 软件设计基本原理

1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐蔽
5. 模块独立性

5.2.2 抽象—概念

抽象

- 我们在考虑问题时，集中考虑和当前问题有关的方面，而忽略和当前问题无关的方面，这就是抽象。或者说抽象就是抽出事物的本质特性而暂时不考虑它们的细节。

软件工程过程中的抽象

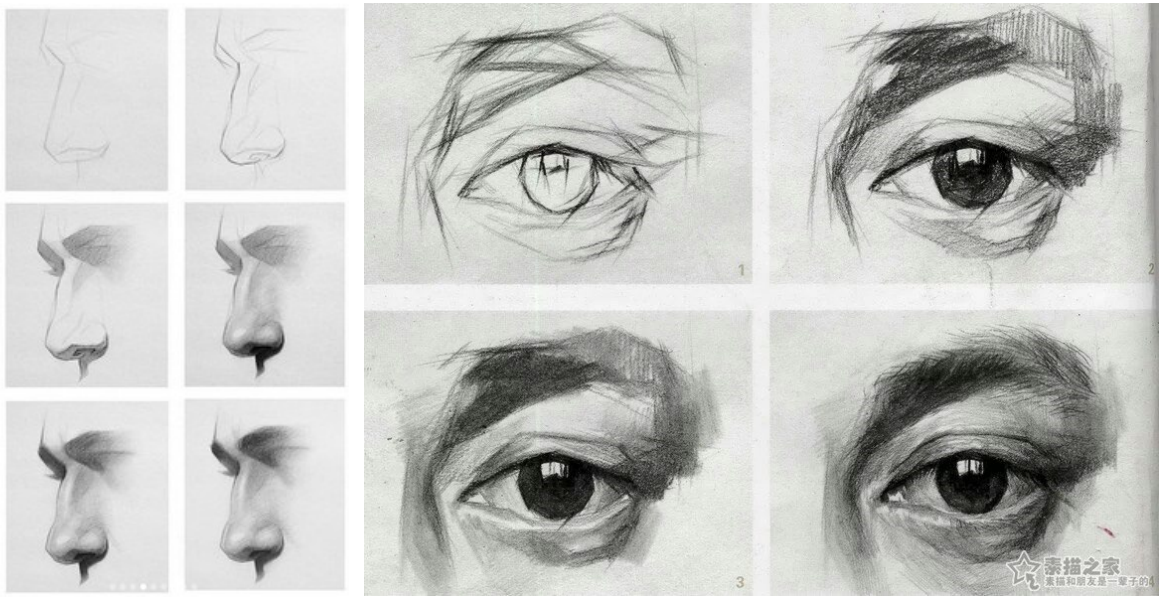
- 软件工程过程的每一步，都是对软件解法的抽象层次的一次细化。在可研阶段，软件被看作是一个完整的系统部分；在需求分析期间，使用在问题环境中熟悉的术语来描述软件的解法；当由总体设计阶段转入详细设计阶段时，抽象的程度进一步减少；最后，当源程序写出来时，也就达到了抽象的最低层。

总体设计中的抽象

- 总体设计中的软件层次结构是一种抽象，自顶向下是软件抽象层次的细化。这种抽象简化了软件的设计与实现，提高了可理解性和可测试性。

5.2 软件设计基本原理

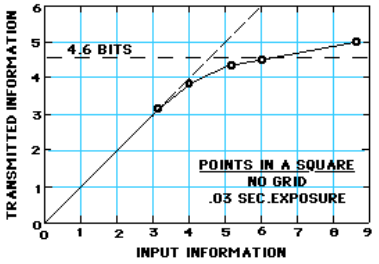
1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐蔽
5. 模块独立性



5.2.3 逐步求精

逐步求精的理论基础： Miller法则

- 一个人在任何时候都只能把注意力集中在5到9个知识块上。



逐步求精的好处：

- 帮助软件工程师把精力集中在与当前开发阶段最相关的那些方面上，忽略那些目前还不需要的细节，把这些细节将留到以后考虑。

软件设计中“抽象”和“求精”的互补关系

- 抽象是在软件设计中关注系统/模块的特征、功能和数据的，忽略底层细节；
- 求精是在设计过程中逐步揭示和实现底层细节。

5.2 软件设计基本原理

1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐蔽
5. 模块独立性

5.2.4 信息隐蔽

信息隐蔽

- 模块所包含的信息（过程和数据）对于其他模块来说应该是隐蔽的。
- 包含在模块内部的信息对于其它不需要这些信息的模块来说，是不能访问的，或者说是“不可见”的。

局部化（做法）

- 局部化是指把关系密切的软件元素物理地放得彼此靠近。

信息隐蔽的好处

- 信息隐蔽给软件测试与维护带来便利。
- 数据和过程的隐蔽，引入的错误所造成的影响就可以只局限在一个或几个模块内部，不波及到软件的其他部分（局部化）。

5.2 软件设计基本原理

1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐蔽
5. 模块独立性

5.2.5 模块独立性

模块独立性

- 模块独立是指开发具有独立功能且和其他模块没有过多的相互作用的模块
- 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。

模块独立性的好处

- 有效模块化的软件比较容易开发，而且适于团队分工。
- 模块化程度较高的软件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

- 耦合：指不同模块彼此间互相依赖的紧密程度；
- 内聚：指在模块内部各个元素彼此结合的紧密程度。

5.2.5 模块独立性——耦合

耦合：软件结构中各个模块之间相互关联程度的度量。

5.2.5 模块独立性——耦合

无耦合/非直接耦合

- 两个模块之间没有或没有直接的数据或调用关系。

数据耦合：

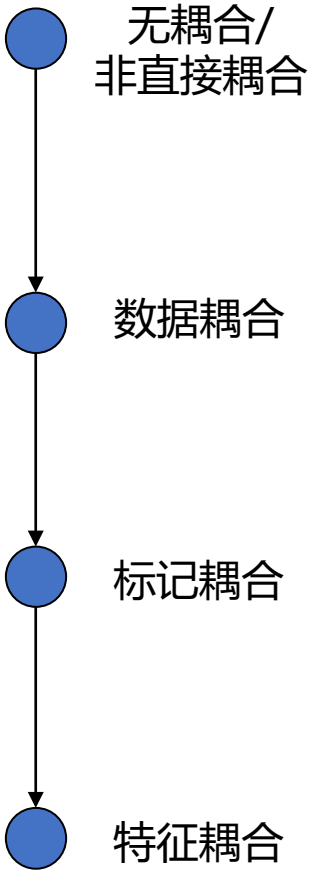
- 如果两个模块通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合就是数据耦合。

标记耦合

- 两个模块应用了（访问了）同名的外部标识符。

特征耦合

- 当把整个数据结构作为参数传递而使用其中一部分数据元素时，就出现了特征耦合。在这种情况下，被调用的模块可以使用的数据多于它确实需要的数据，这将导致对数据的访问失去控制。



5.2.5 模块独立性——耦合

控制耦合

- 如果两个模块通过参数交换信息，交换的信息有控制信息，那么这种耦合就是控制耦合。

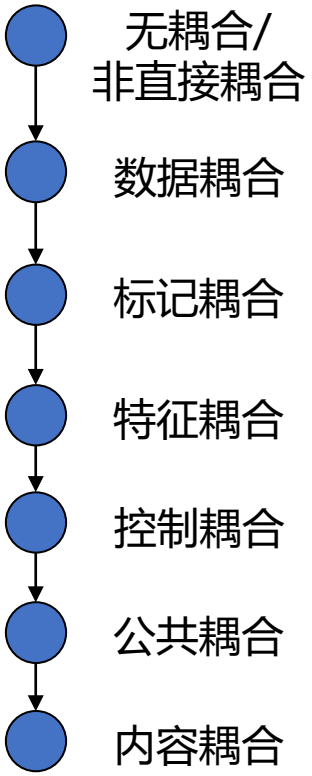
公共耦合

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

内容耦合

有下列情形之一，两个模块就发生了内容耦合：

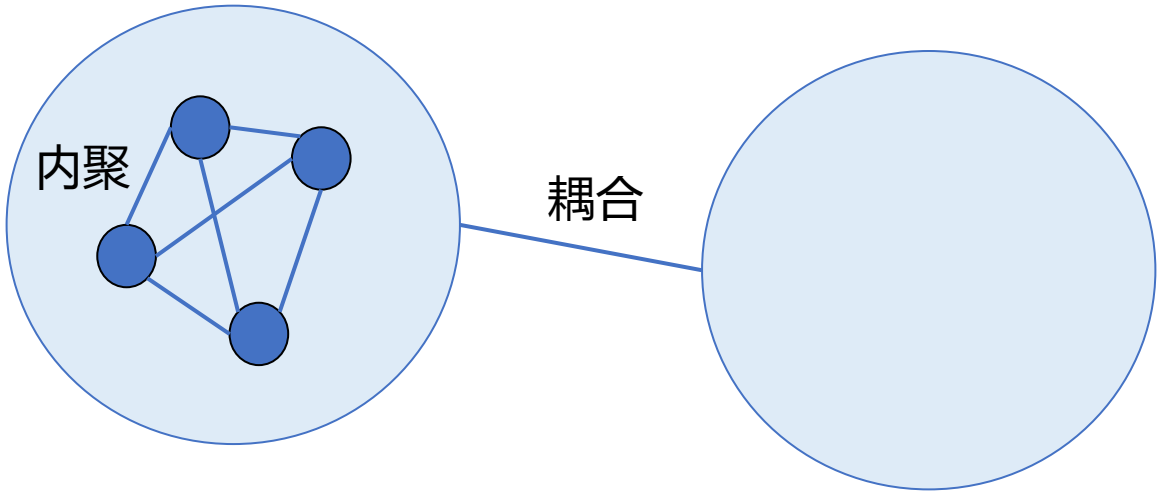
- 一个模块访问另一个模块的内部数据；
- 一个模块不通过正常入口而转到另一个模块的内部；
- 一个模块有多个入口。



设计原则：尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，避免使用内容耦合。

5.2.5 模块独立性——内聚

内聚：模块内部各个元素彼此结合的紧密程度的度量。



5.2.5 模块独立性——内聚

偶然内聚（低，0分）

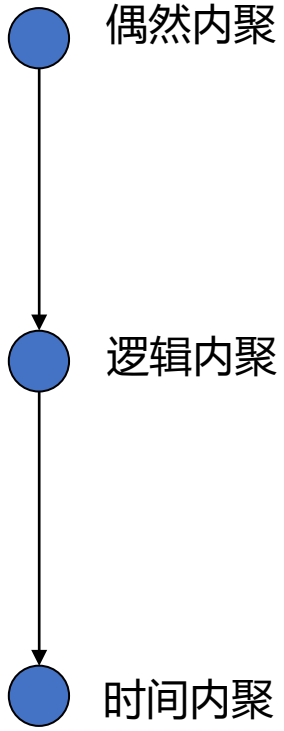
- 如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也比较松散，就叫做偶然内聚。

逻辑内聚（低，1分）

- 如果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。

时间内聚（低，3分）

- 如果一个模块包含的任务必须在同一段时间内执行，称为时间内聚。



5.2.5 模块独立性——内聚

过程内聚（中，5分）

- 如果一个模块内的元素是相关的，且必须以特定次序执行。

通信内聚（中，7分）

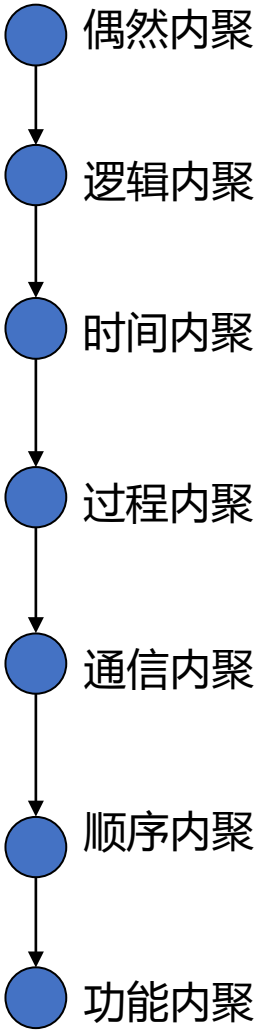
- 如果模块中所有元素都使用同一个输入数据，产生一个输出数据。

顺序内聚（高，9分）

- 如果一个模块内的元素与一个功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。

功能内聚（高，10分）

- 如果模块内所有元素属于一个整体，完成一个单一的功能，则称为功能内聚。



第五章　总体设计

5.1　总体设计的过程

5.2　软件设计基本原理

5.3　设计准则

5.4　总体设计的图形描述工具

5.5　结构化设计方法

5.3 设计准则

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

5.3 设计准则

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

5.3 设计准则

1. 改进软件结构提高模块独立性

- 初步的软件结构，通过模块的合并和分解，达到降低耦合、提高内聚的目的。

- 提高独立性的主要方法：
  - 提取公用子功能、
  - 减少模块间控制信息的传递、
  - 减少模块对全局变量的访问、
  - 降低接口的复杂性等。

5.3 设计准则

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

5.3 设计准则

2. 模块规模应该适中

- 规模过大的模块逻辑复杂，难以理解。可以进一步分解。
- 规模过小的模块会使系统的模块数增加很多，造成模块接口开销大。对于非公共调用的小模块，可合并在其父模块中。
- 推荐模块的大小为一页纸，大约不超过60条语句。

5.3 设计准则

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

5.3 设计准则

3. 深度、宽度、扇出和扇入都应适当

深度

- 软件结构的控制层数（数的深度），反映了系统的逻辑复杂程度。

宽度

- 软件结构的同一层上最多模块数，反映了系统的功能复杂程度。

扇出

- 一个模块直接调用（控制）的下级模块数，反映了该模块的复杂程度。

扇入

- 一个模块被直接调用的上级模块数，反映了该模块的共享程度。

5.3 设计准则

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

设计准则

4. 模块的作用域应该在控制域之内

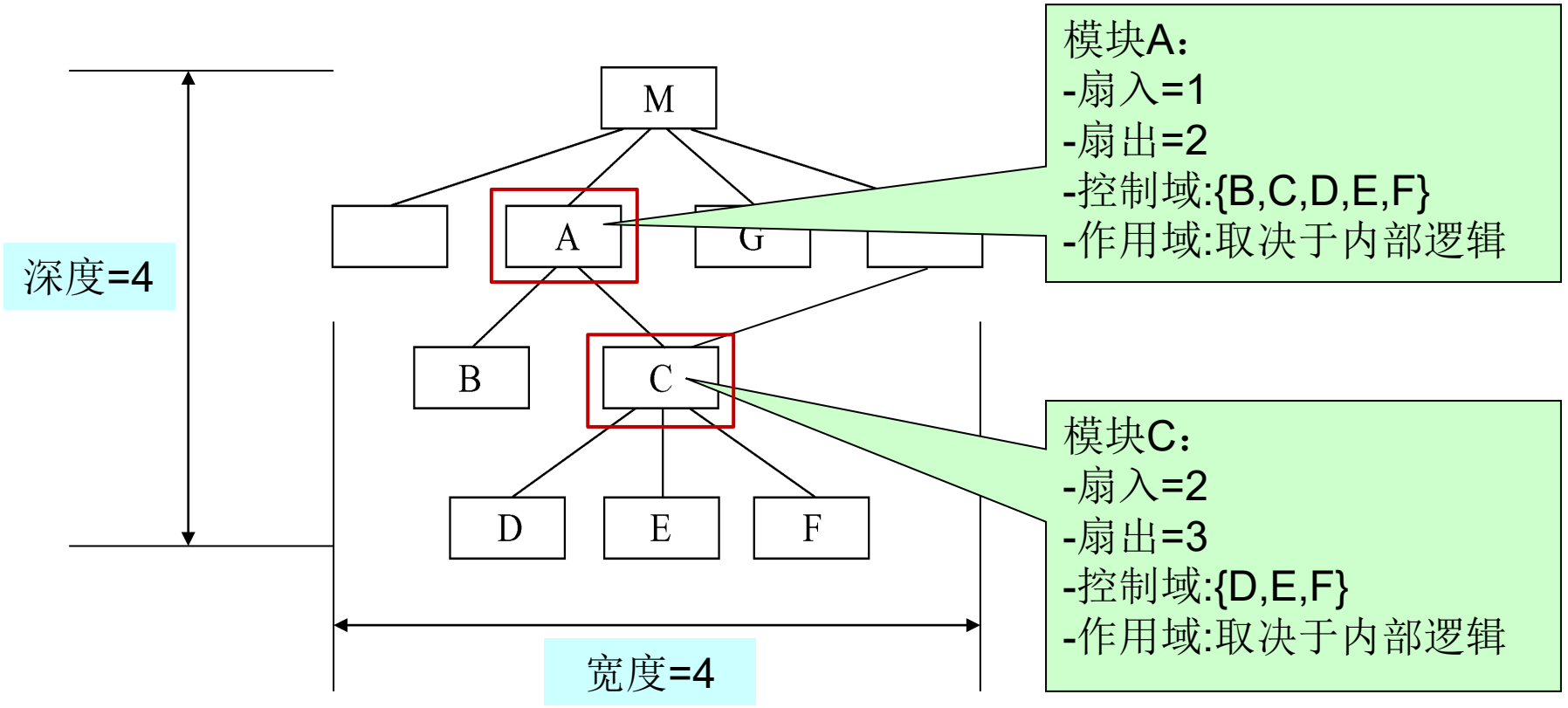
控制域：

- 该模块直接或间接从属于它的所有模块的集合
- （结构上受影响）

作用域：

- 受该模块内的一个判定影响的所有模块的集合
- （逻辑上受影响）

5.3 设计准则—软件结构的度量



“好的”软件结构：

深度和宽度适中，上层模块高扇出,下层模块高扇入；作用域在控制域内。

5.3 设计准则

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

设计准则

5. 力争降低模块接口的复杂程度

- 模块接口复杂是容易引发软件错误的重要原因。
- 模块接口复杂是模块紧耦合、低内聚的征兆。
- 模块接口的设计应使信息传递简单清晰，与模块功能一致。

5.3 设计准则

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

5.3 设计准则

6. 设计单入口单出口的模块

- 使模块只有一个入口、一个出口，不出现中间跳出、跳入。
- 模块之间不出现内容耦合。

7. 模块功能应该可以预测，且功能不过份局限

- 模块的功能是确定的、可重现的（对于同样的输入产生同样的输出）。
- 在功能确定的基础上，模块处理能力应有较广的适应性，满足不同数据结构和处理规模。

第五章　总体设计

5.1　总体设计的过程

5.2　软件设计基本原理

5.3　设计准则

5.4　总体设计的图形描述工具

5.5　结构化设计方法

5.4 总体设计的图形描述工具

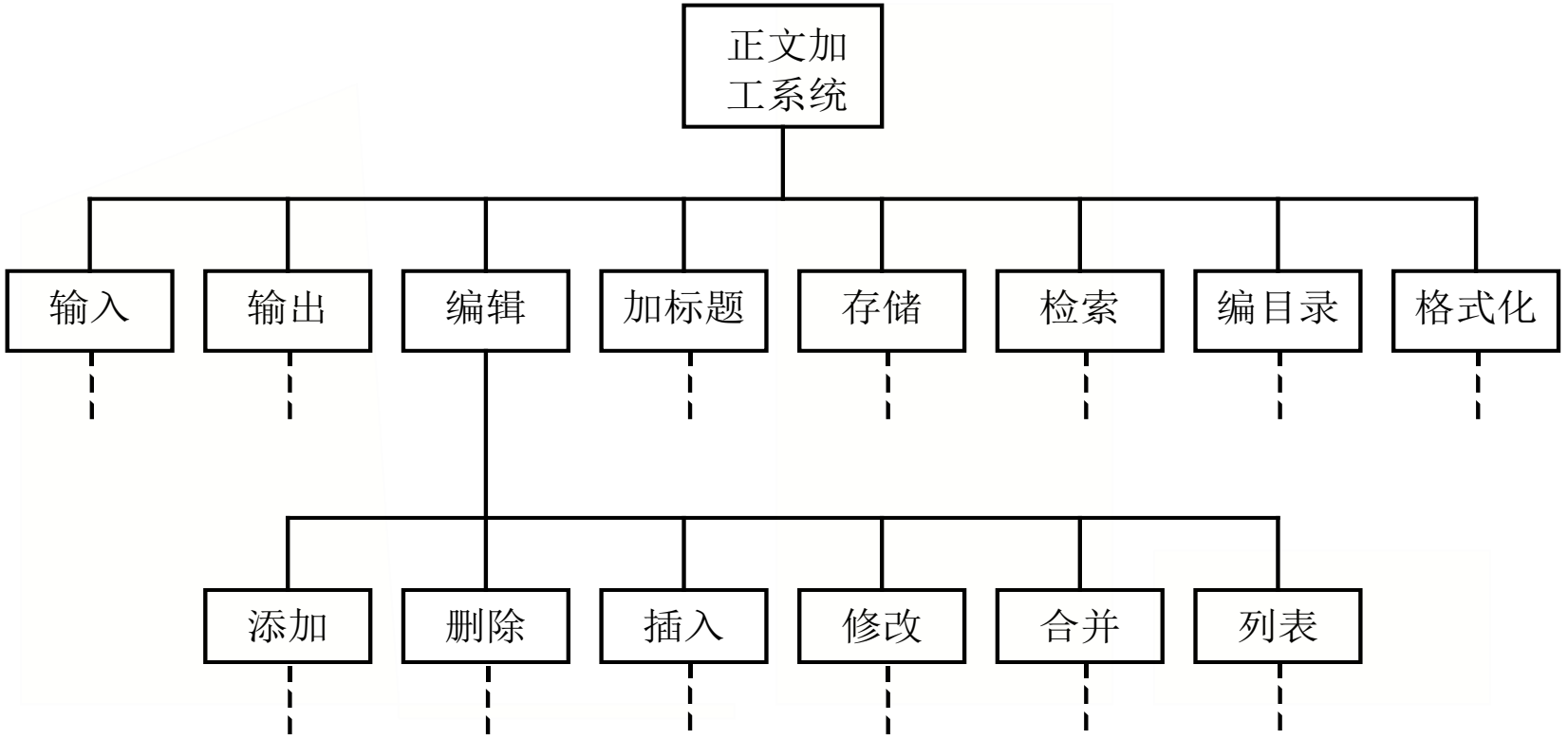
5.4.1 层次图

5.4.2 HIPO图

5.4.3 结构图

5.4.1 层次图

层次化表示功能模块的调用关系，上层模块调用下层模块完成所需的功能。

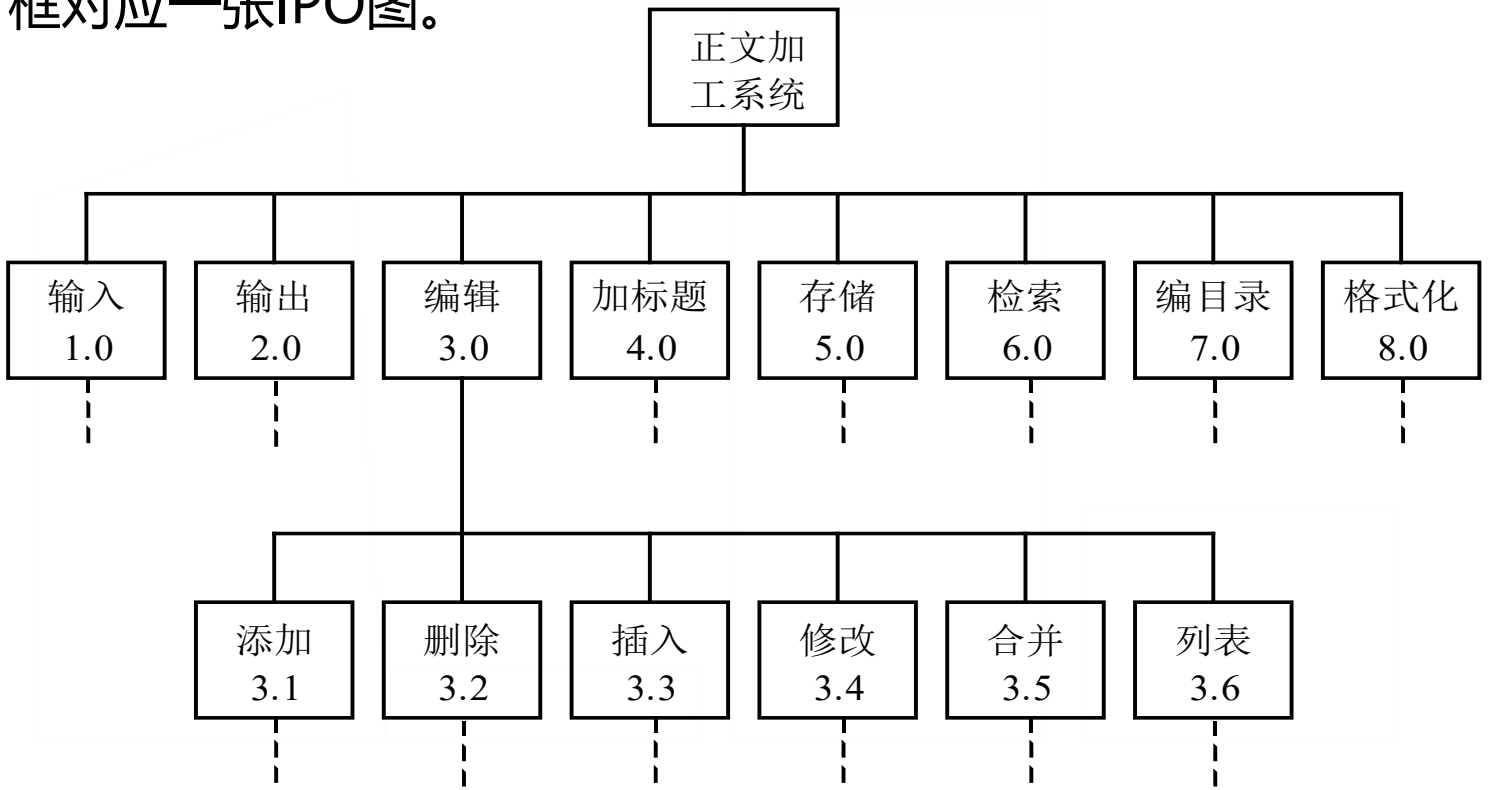


正文加工系统的层次图

5.4.2 HIPO图

对层次图改进：

- 每一个方框有编号；
- 每一个方框对应一张IPO图。

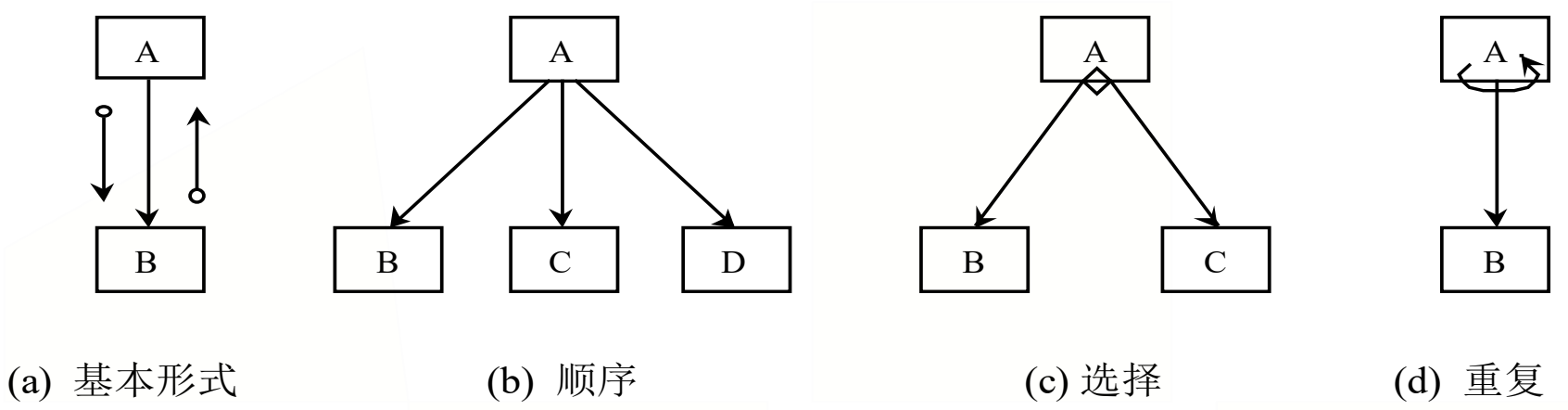


带编号的层次图(H图)

5.4.3 结构图

对层次图改进：

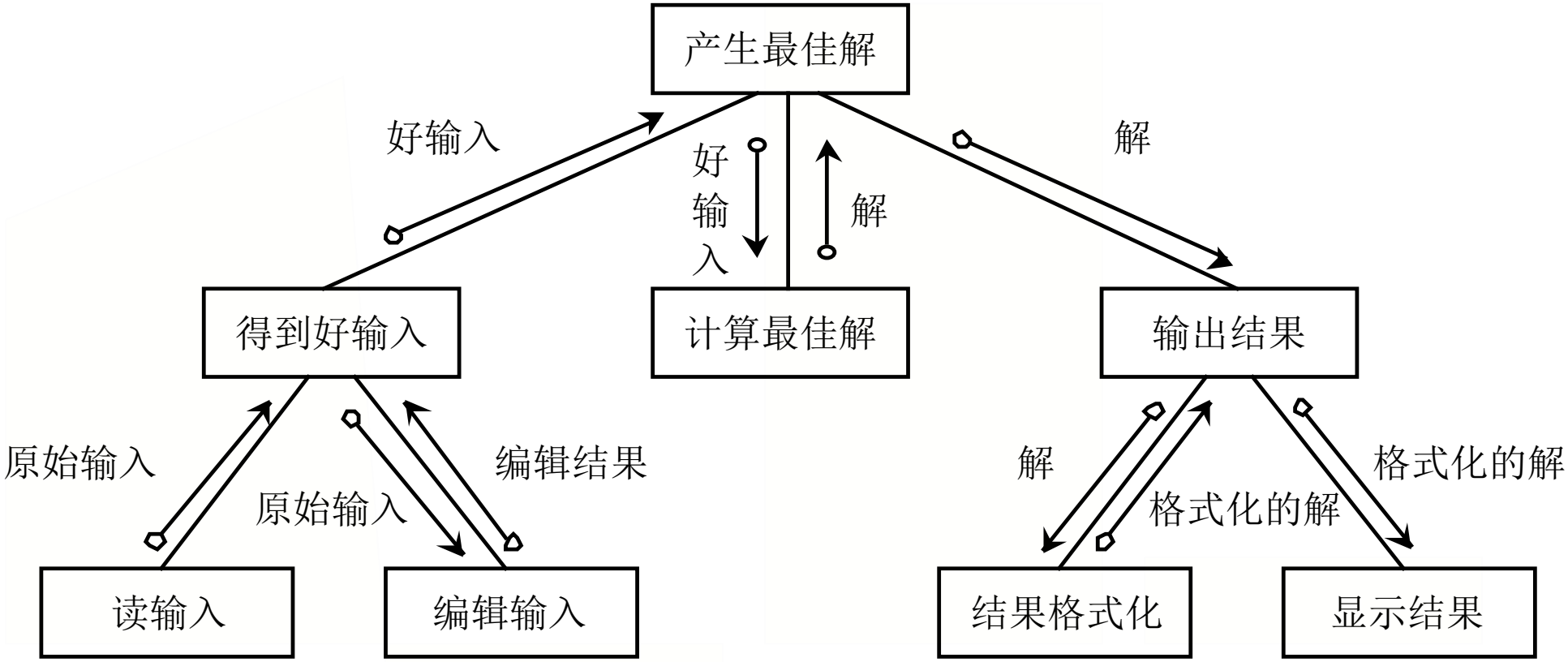
- 采用带有注释的箭头表示模块调用时参数的传递；
- 实心圆 注释箭头表示传递控制信息，空心圆注释箭头表示传递数据；
- 菱形◇表示条件分支选择；
- 箭头圆弧表示循环。



(a) 基本形式　(b) 顺序　(c) 选择　(d) 重复

软件结构图的基本符号

5.4.3 结构图——例



产生最佳解的结构图

第五章　总体设计

5.1　总体设计的过程

5.2　软件设计基本原理

5.3　设计准则

5.4　总体设计的图形描述工具

5.5　结构化设计方法

5.5 结构化设计方法

5.5.1 数据流图的类型

5.5.2 设计步骤

5.5.3 变换设计

5.5.4 事务设计

5.5.5 设计的后处理

结构化设计从数据流图入手，分析DFD的类型，找出DFD的处理中心，映射成初步的软件结构，按照设计准则细化和优化结构，描述模块的接口和全局数据，经过评审后，提交下一步进行详细设计。

5.5 结构化设计方法

5.5.1 数据流图的类型

5.5.2 设计步骤

5.5.3 变换设计

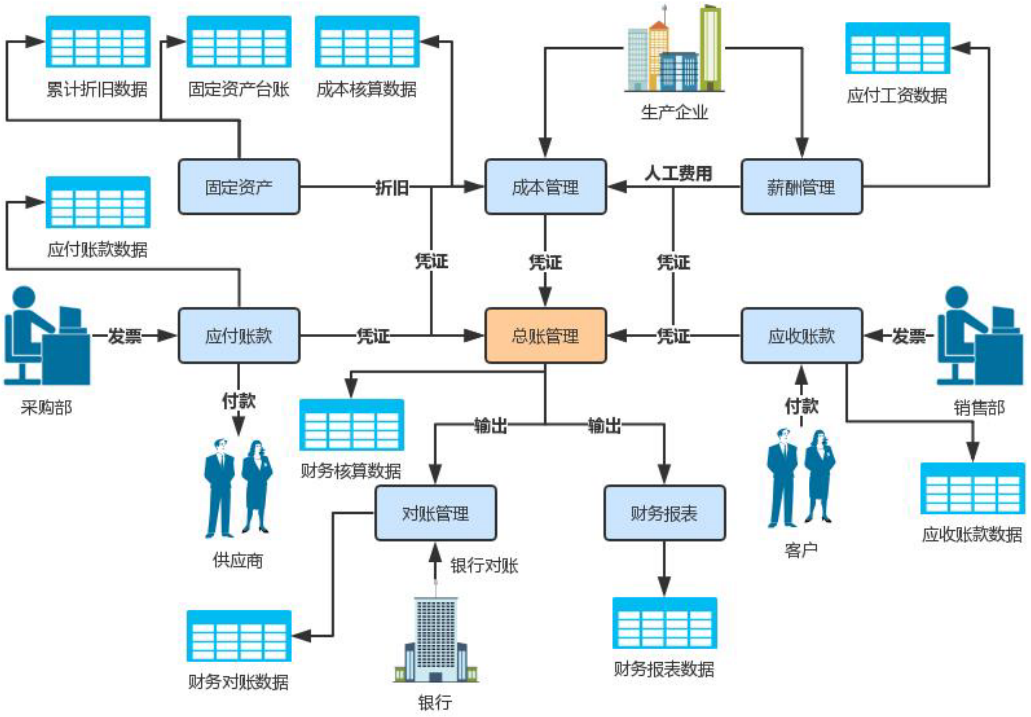
5.5.4 事务设计

5.5.5 设计的后处理

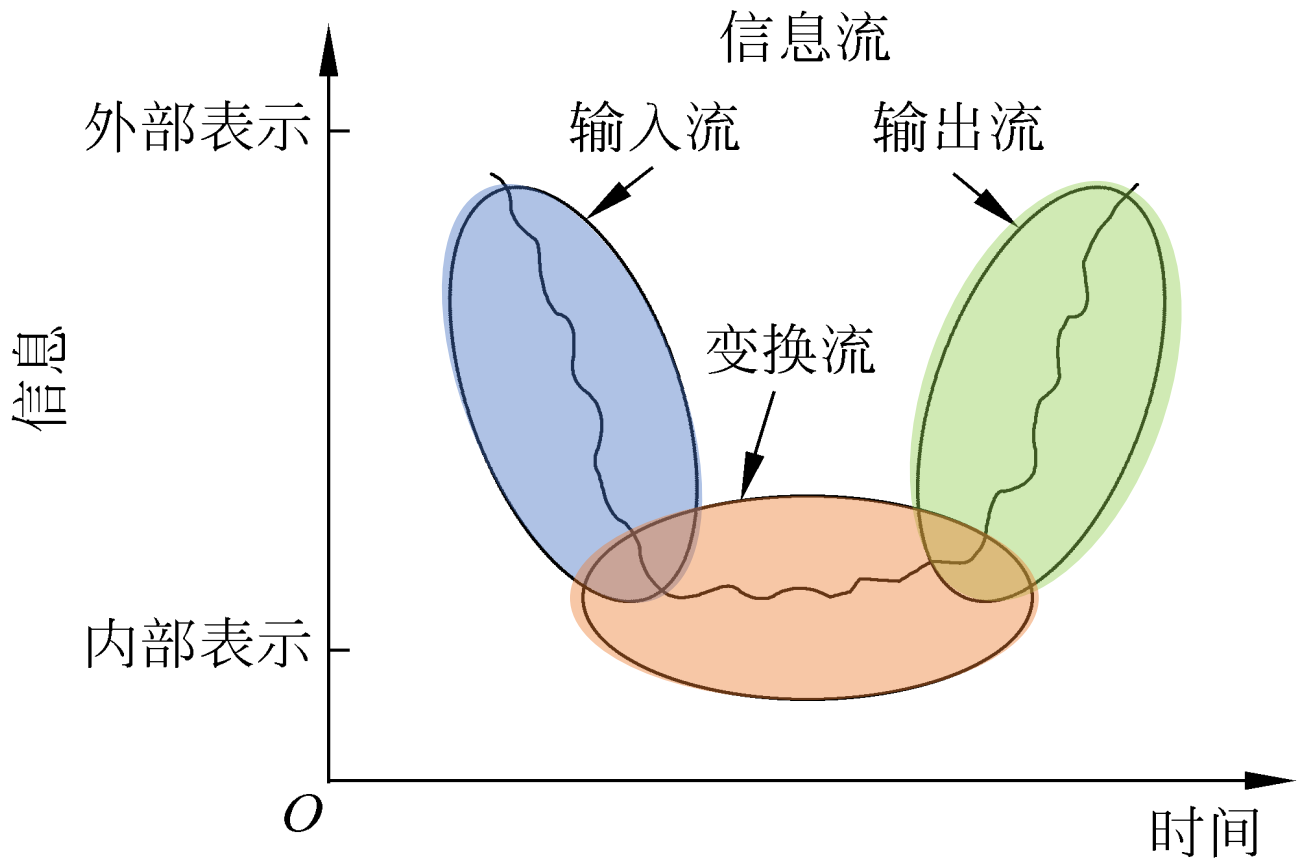
5.5.1 数据流图的类型

1、变换型数据流图

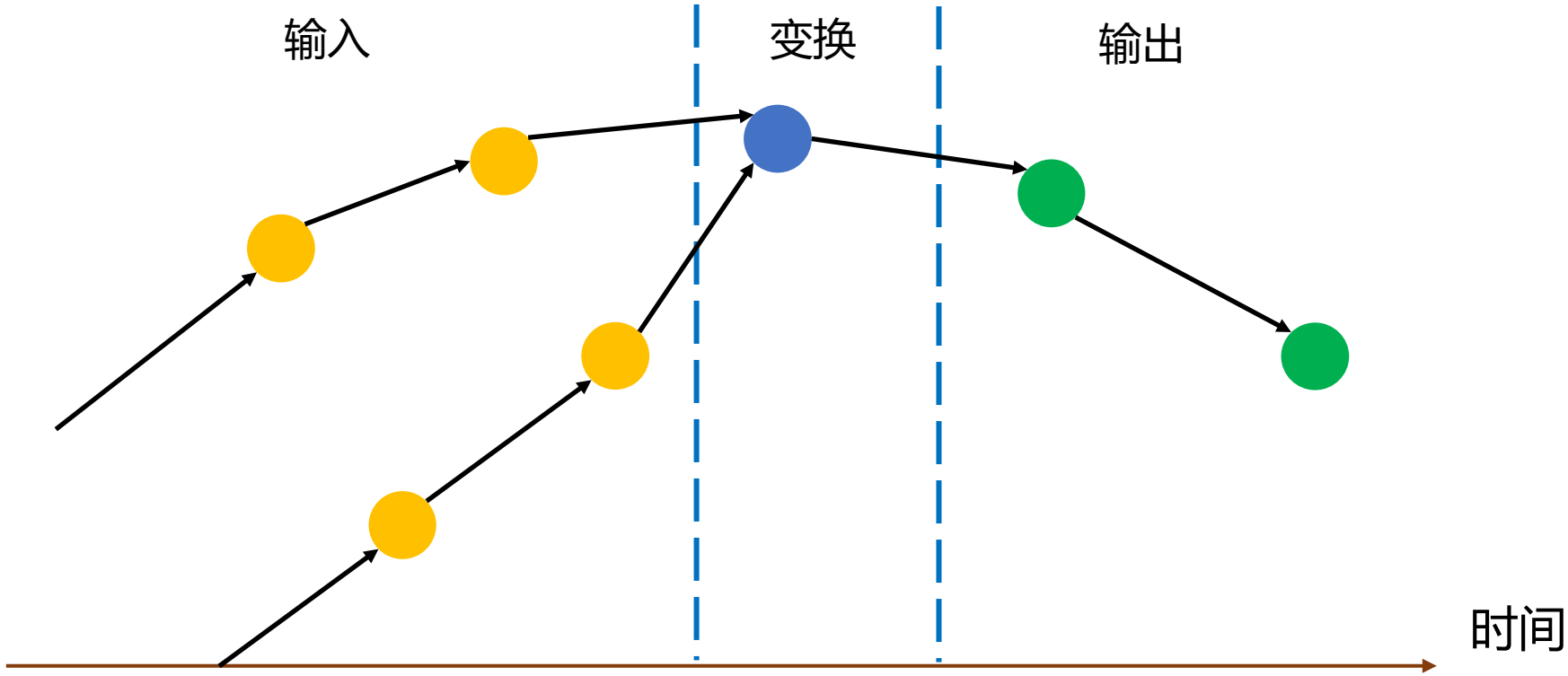
2、事务型数据流图



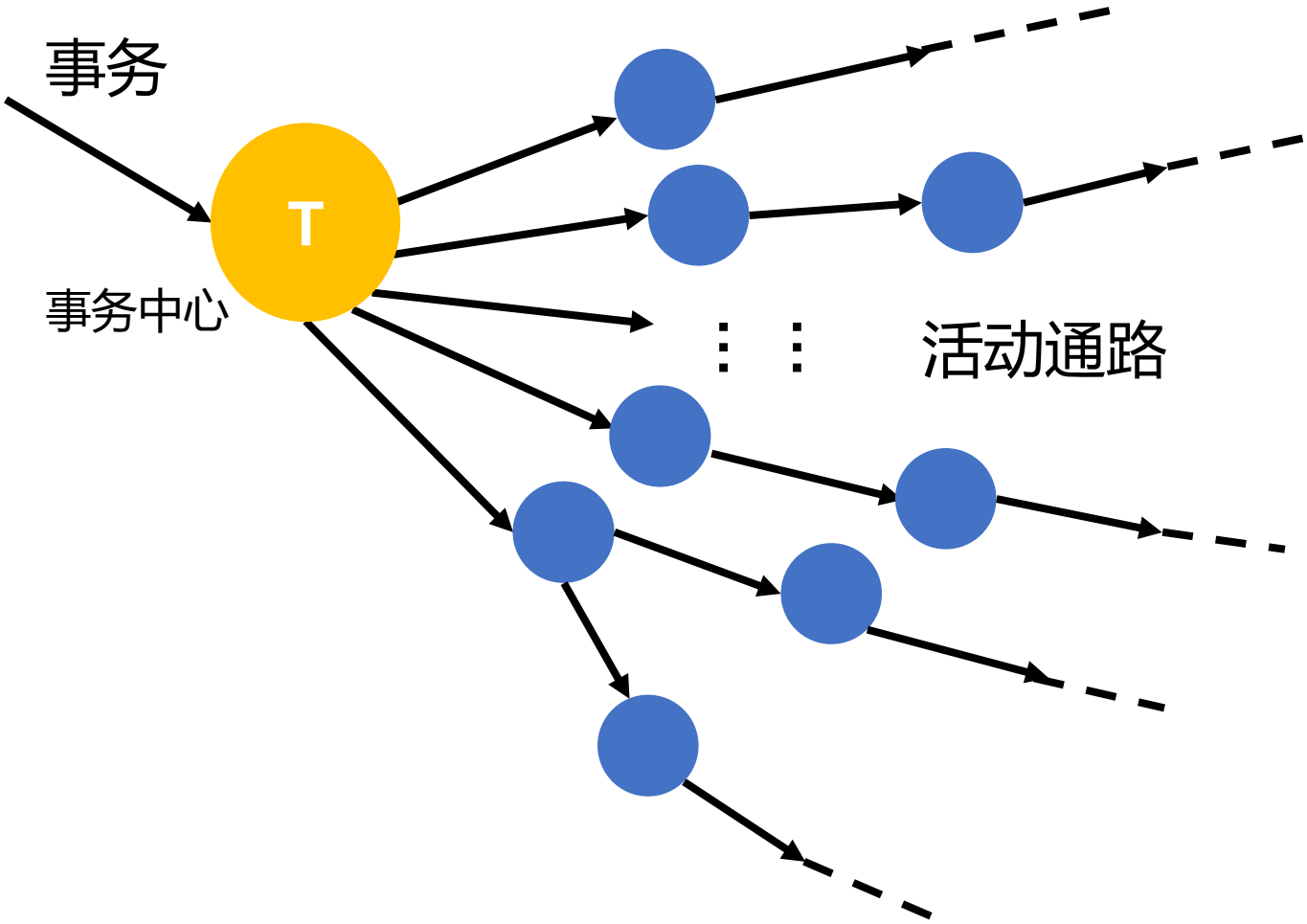
5.5.1 数据流图的类型——变换型数据流图



5.5.1 数据流图的类型——变换型数据流图



5.5.1 数据流图的类型——事务型数据流图



5.5 结构化设计方法

5.5.1 数据流图的类型

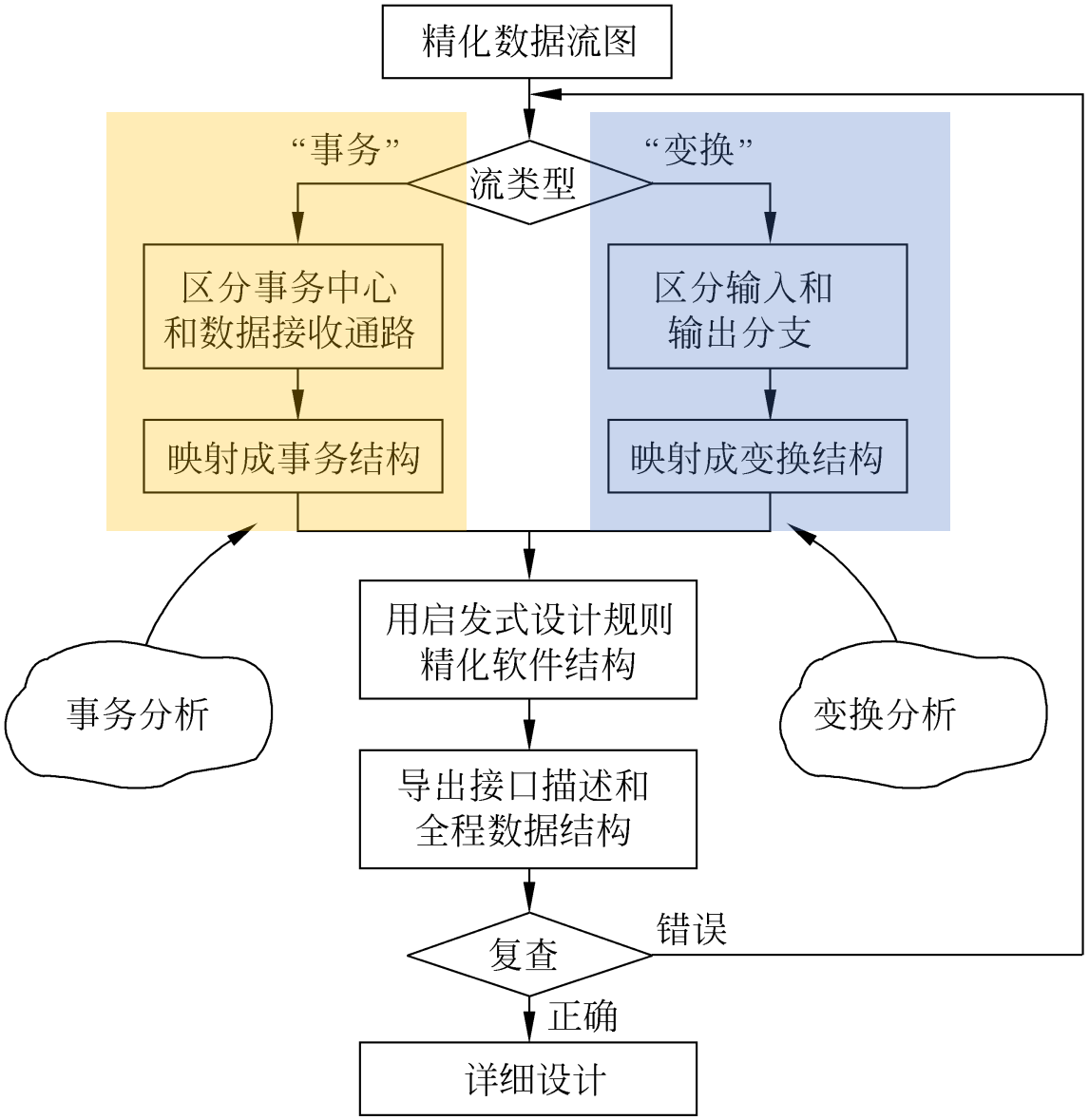
5.5.2 设计步骤

5.5.3 变换设计

5.5.4 事务设计

5.5.5 设计的后处理

5.5.2 设计步骤



面向数据流方法的设计过程

5.5 结构化设计方法

5.5.1 数据流图的类型

5.5.2 设计步骤

5.5.3 变换设计

5.5.4 事务设计

5.5.5 设计的后处理

5.5.3 变换设计

1. 复查基本系统模型
2. 复查并精化数据流图
3. 确定数据流图的类型
4. 确定输入流和输出流的边界，从而孤立出变换中心
5. 进行“第一级分解”
6. 进行“第二级分解”
7. 对第一次分割得到的软件结构进一步精化

5.5.3 变换设计——案例：汽车仪表盘

我们通过一个汽车数字仪表板的设计来介绍变换分析的过程。假设仪表板的功能如下：

- （1）通过模 / 数（A / D）转换实现传感器和微处理机接口；
- （2）在发光二极管（LCD）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。

需求分析阶段，对每项性能和其它要求进行分析，得出数据流图。

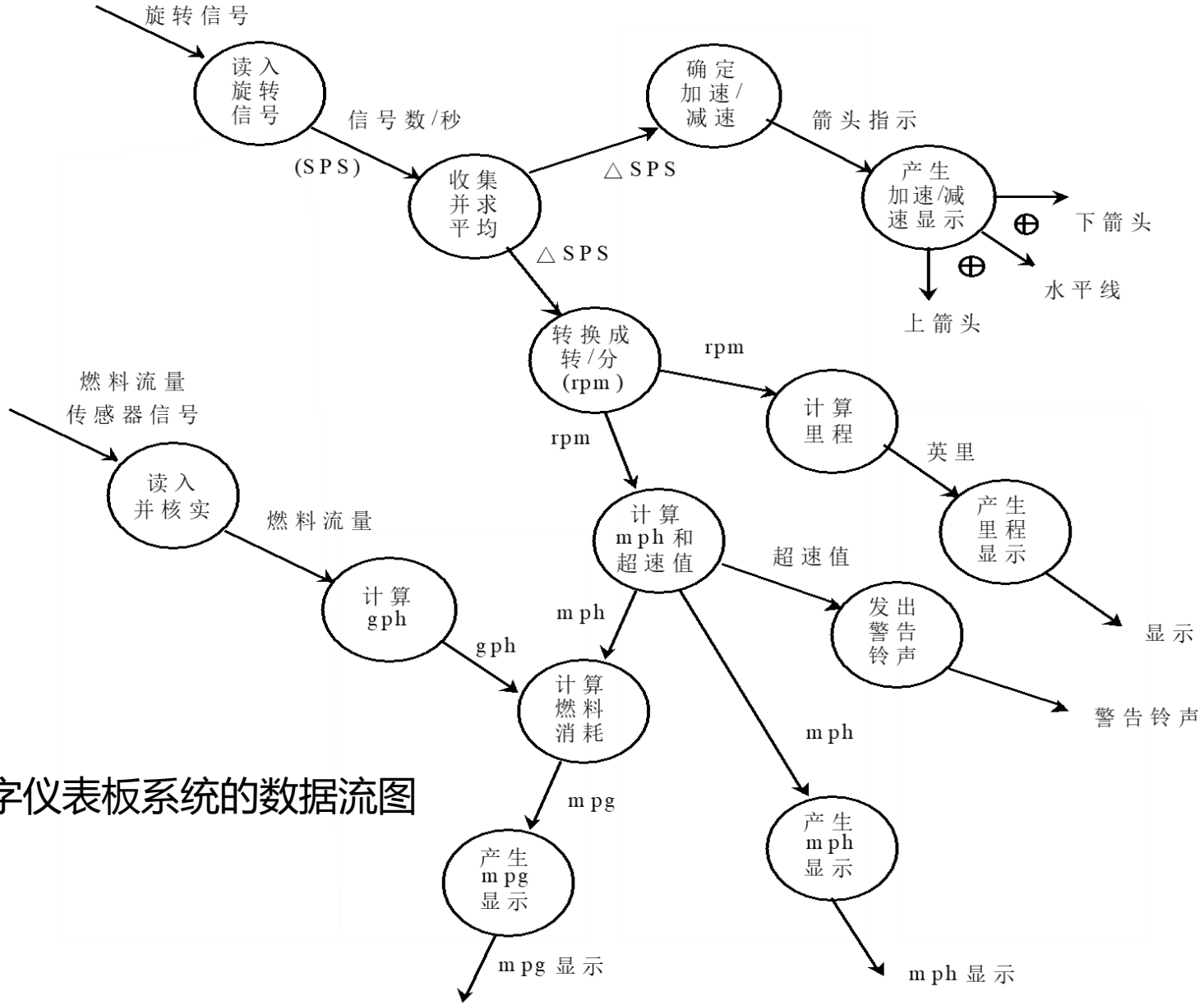
1、复查基本系统模型

仔细阅读和理解需求规格说明书

复查基本系统模型

- 初步的数据流图
- 数据字典
- IPO图
- ……

2、复查并精化数据流图



数字仪表板系统的数据流图

3、确定数据流图的类型

变换型

事务型

从上图中可以看出：

- 数据沿着两条输入通路（旋转信号和燃料流量传感器信号）进入系统，
- 沿着五条通路（4个显示，一个警告铃声）离开，
- 没有明显的事务中心，可以认为这个数据流图的类型是变换型数据流图。

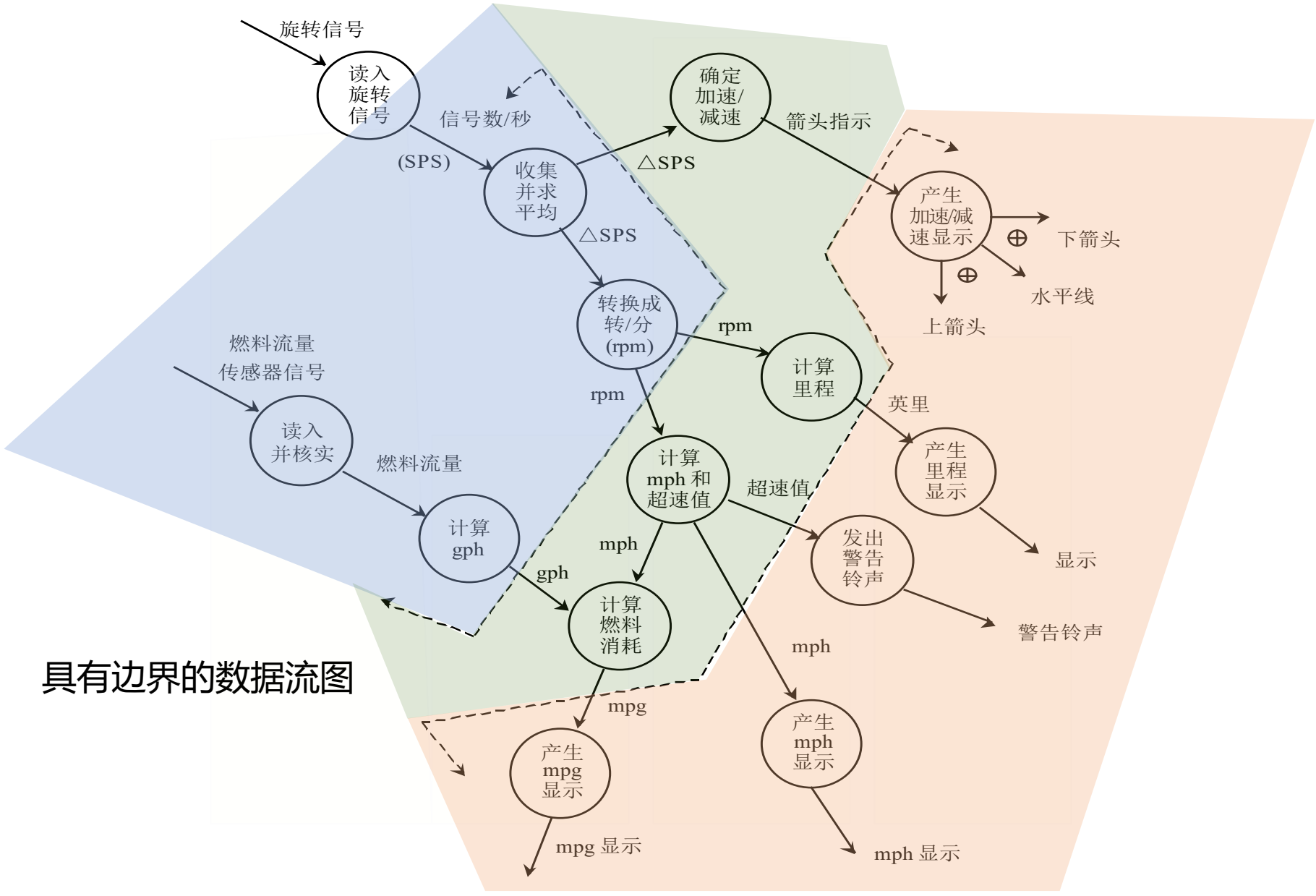
3、确定数据流图的类型

事务型

从上图中可以看出：

- 数据沿着两条输入通路（旋转信号和燃料流量传感器信号）进入系统，
- 沿着五条通路（4个显示，一个警告铃声）离开，
- 没有明显的事务中心，可以认为这个数据流图的类型是变换型数据流图。

4、确定输入流和输出流的边界，从而孤立出变换中心

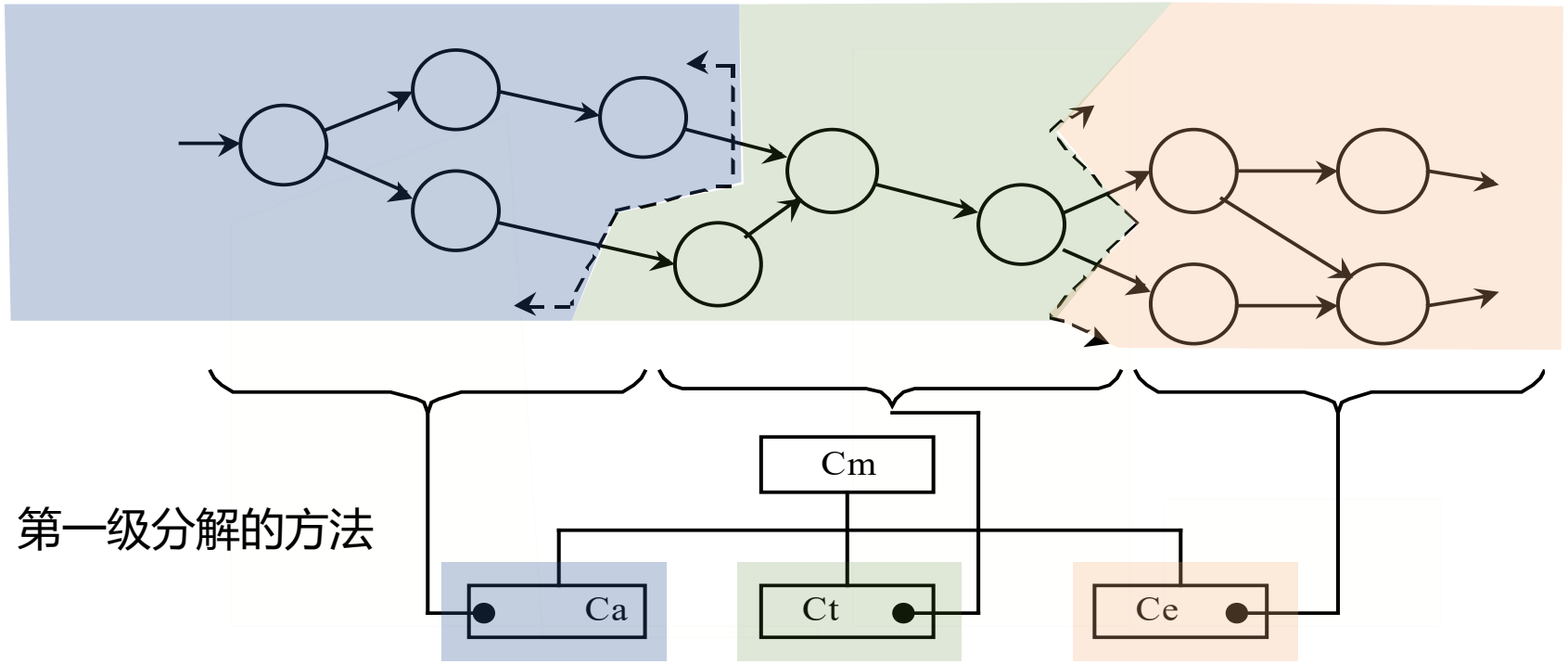


具有边界的数据流图

5、进行“第一级分解”

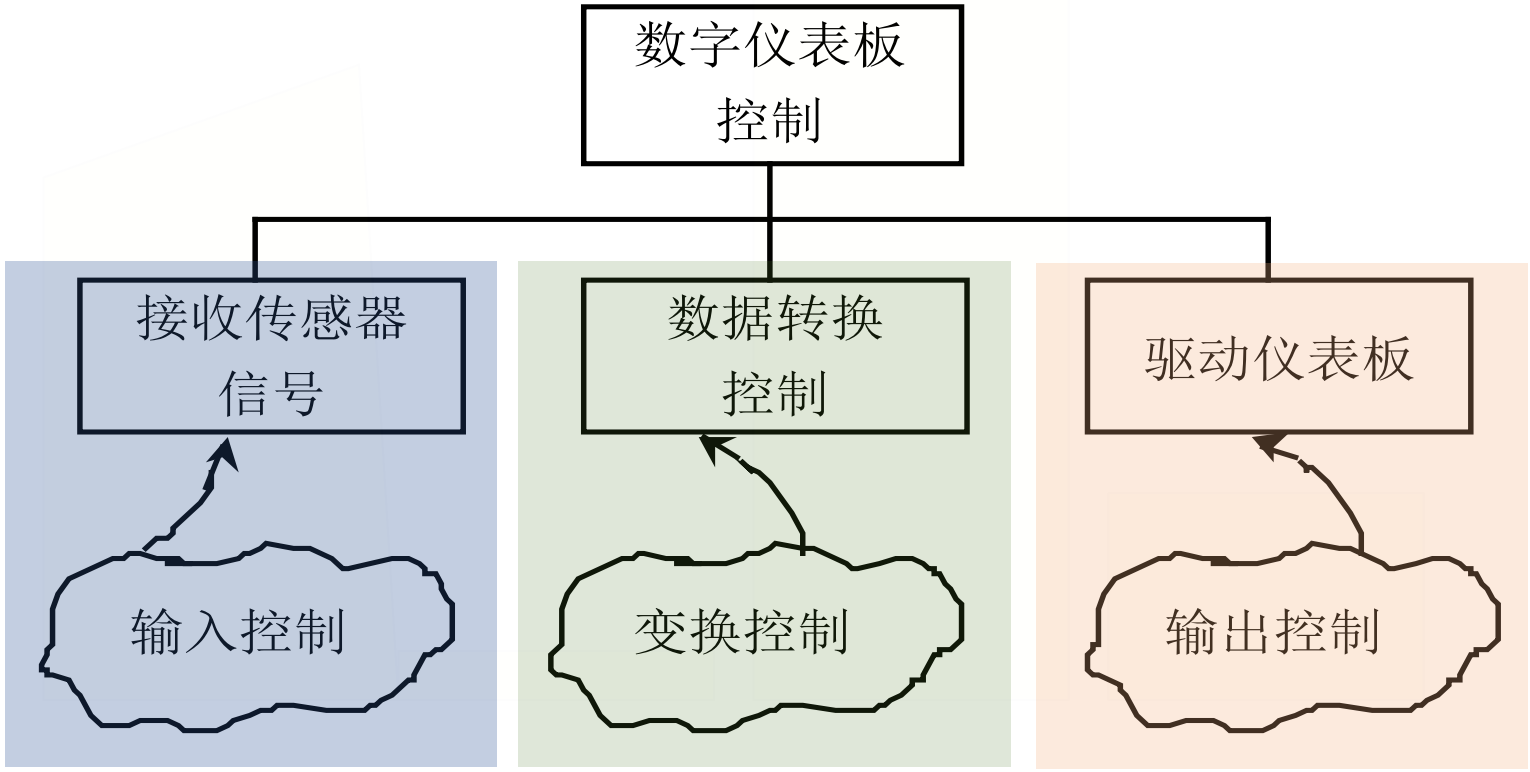
一个总控模块，若干个二级模块

- 对“变换型”DFD：一个输入模块、一个变换模块、一个输出模块。
- 对“事务型”DFD：一个事务接收模块、每个事务流对应一个模块。



第一级分解的方法

第一级分解的结果

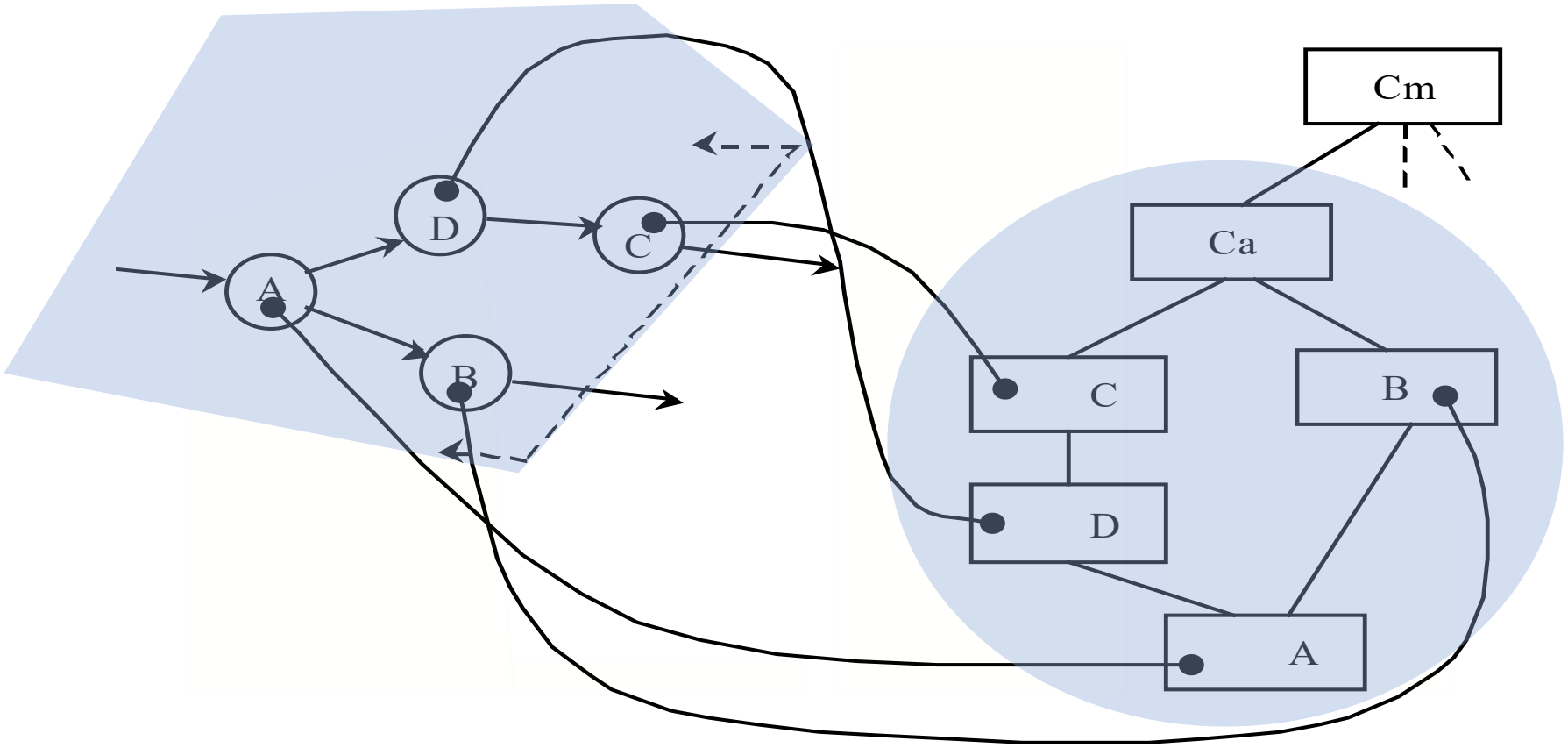


数字仪表板系统的第一级分解

6、进行“第二级分解”

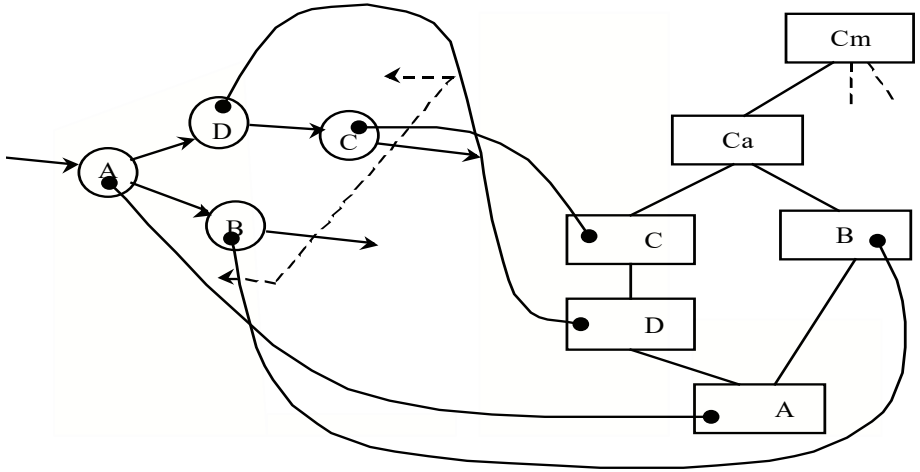
DFD中的每一个加工对应一个模块

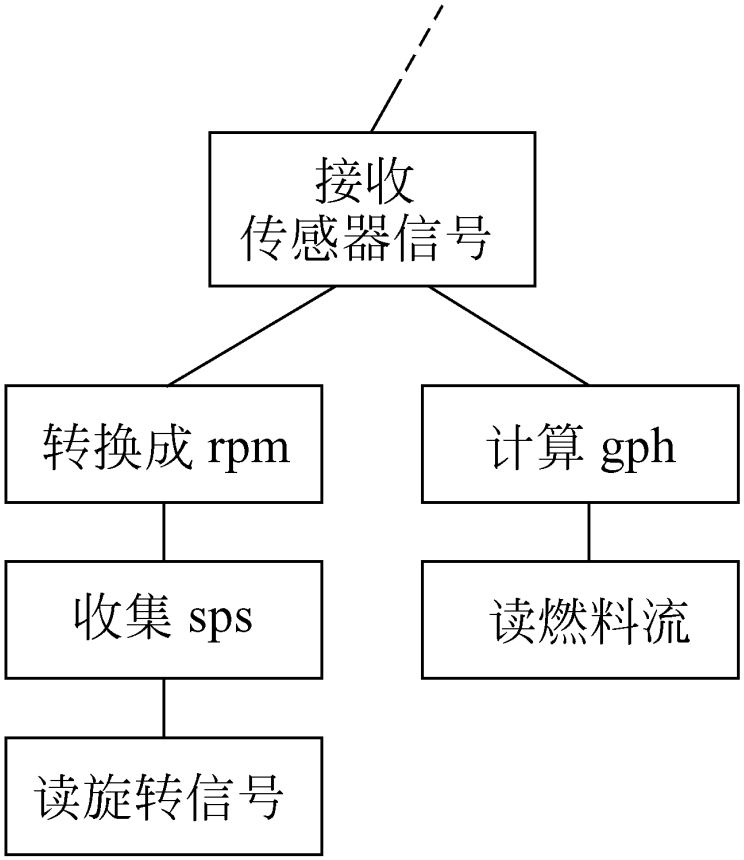
从“变换中心”向外延伸，构造模块间的层次（调用）关系



第二级分解的方法

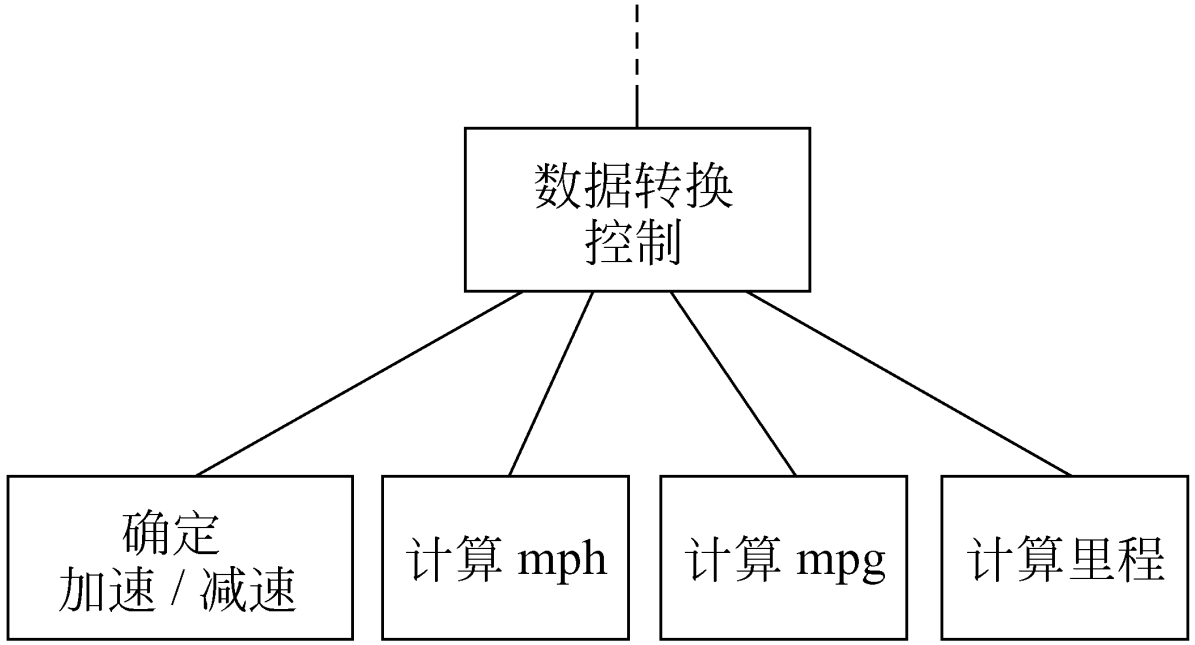
第二级分解的结果—输入结构





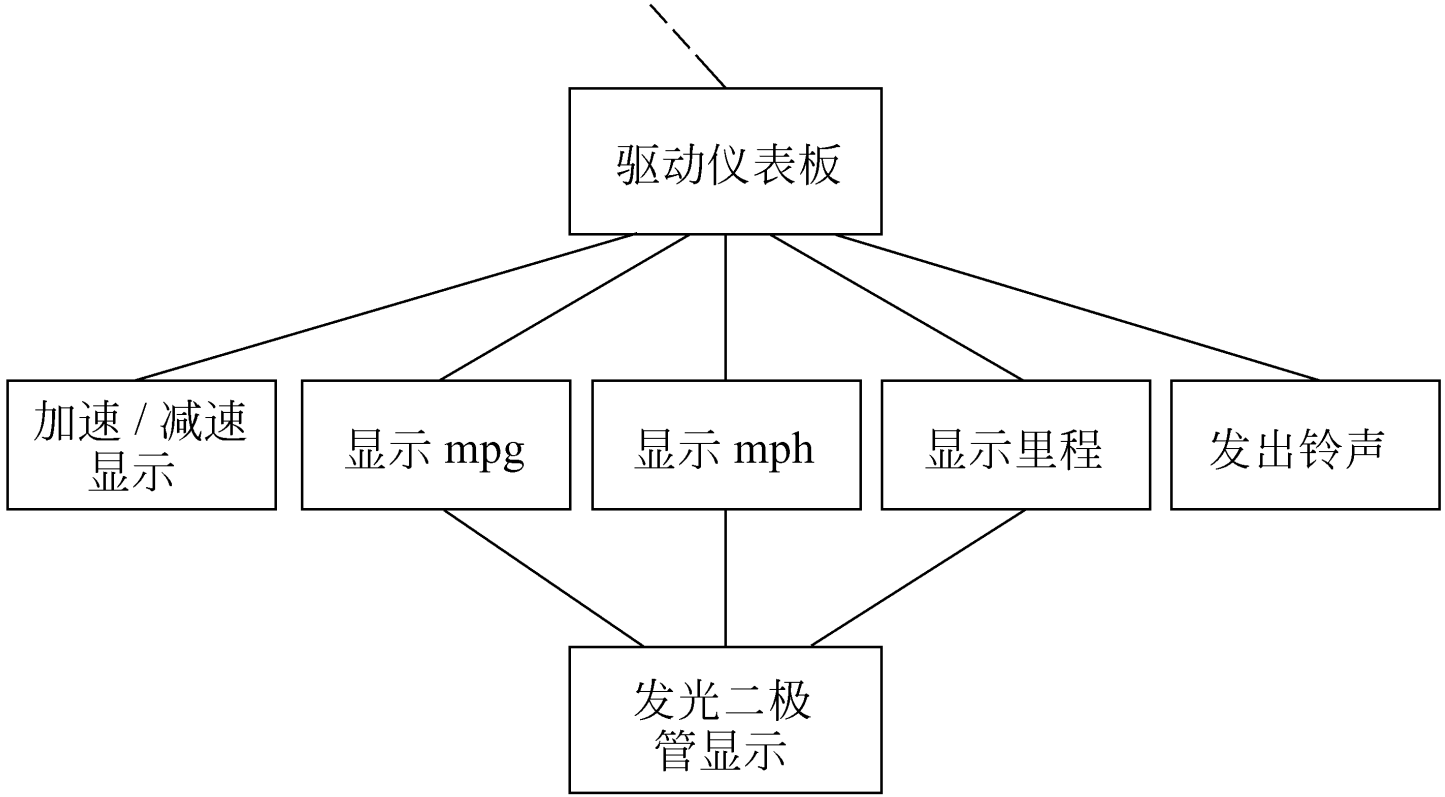
未经精化的输入结构

第二级分解的结果—变换结构



未经精化的变换结构

第二级分解的结果—输出结构

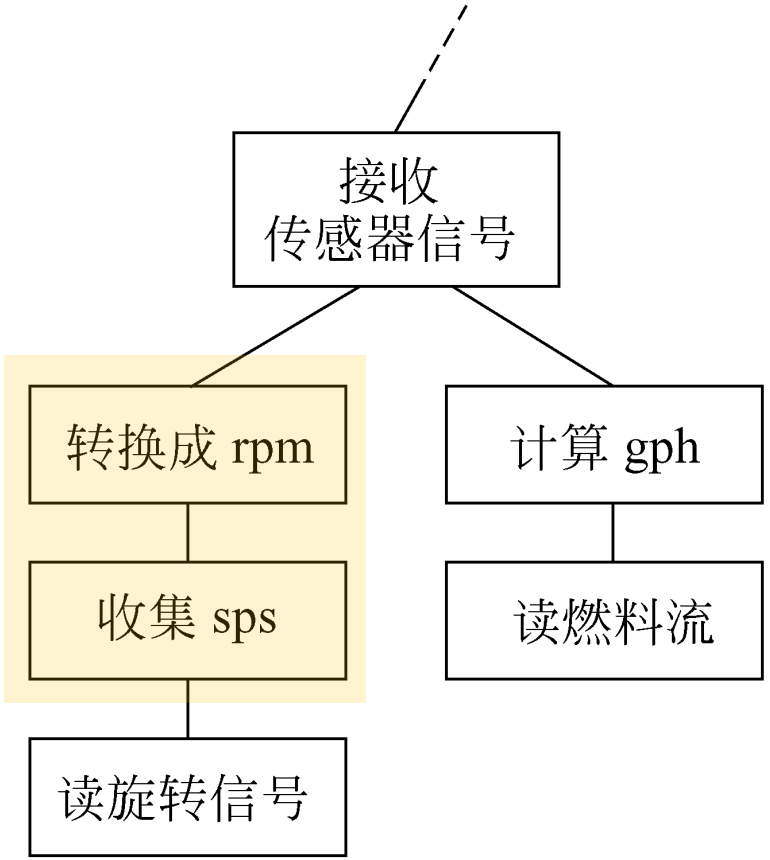


未经精化的输出结构

7、对第一次分割得到的软件结构进一步精化

对于从前面的设计步骤得到的软件结构，使用设计度量和设计准则，还可以再修改：

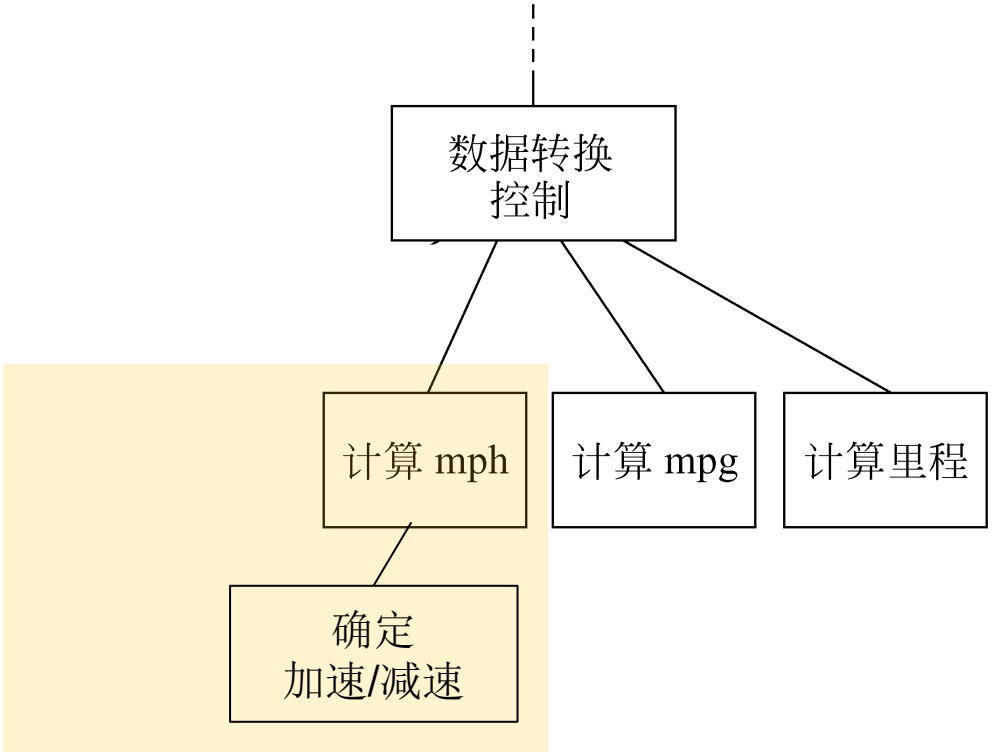
- 输入结构中的模块“转换成rpm”和“收集sps”，模块太小，功能简单，可以合并；



7、对第一次分割得到的软件结构进一步精化

对于从前面的设计步骤得到的软件结构，使用设计度量和设计准则，还可以再修改：

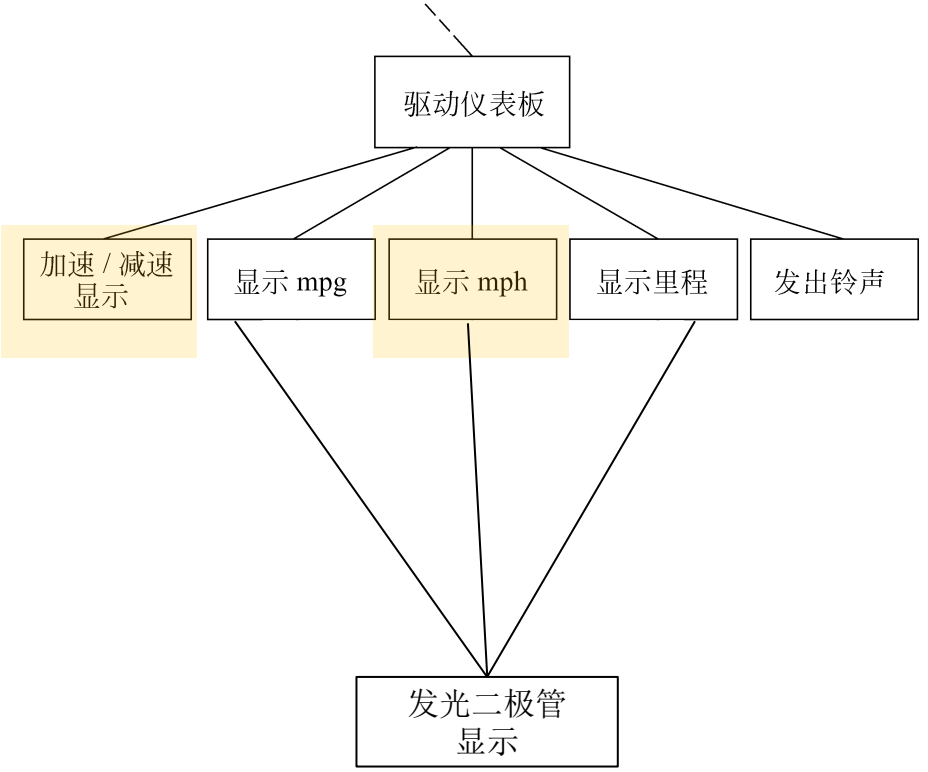
- 输入结构中的模块“转换成rpm”和“收集sps”，模块太小，功能简单，可以合并；
- 模块“确定加速 / 减速”可以放在模块“计算mph”下面，以减少耦合；
- 模块“加速 / 减速显示”可以相应地放在模块“显示mph”的下面。



7、对第一次分割得到的软件结构进一步精化

对于从前面的设计步骤得到的软件结构，使用设计度量和设计准则，还可以再修改：

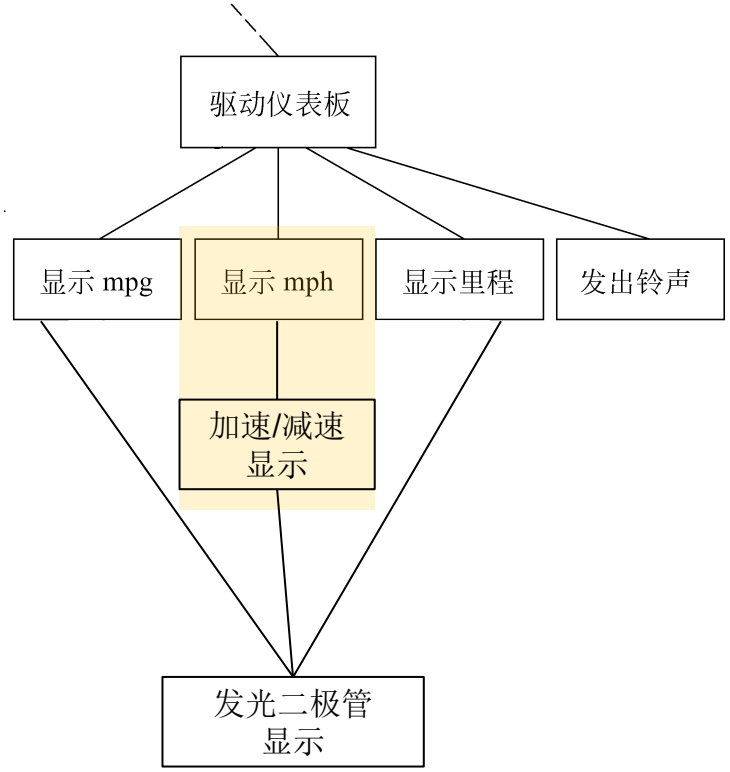
- 输入结构中的模块“转换成rpm”和“收集sps”，模块太小，功能简单，可以合并；
- 模块“确定加速 / 减速”可以放在模块“计算mph”下面，以减少耦合；
- 模块“加速 / 减速显示”可以相应地放在模块“显示mph”的下面。



7、对第一次分割得到的软件结构进一步精化

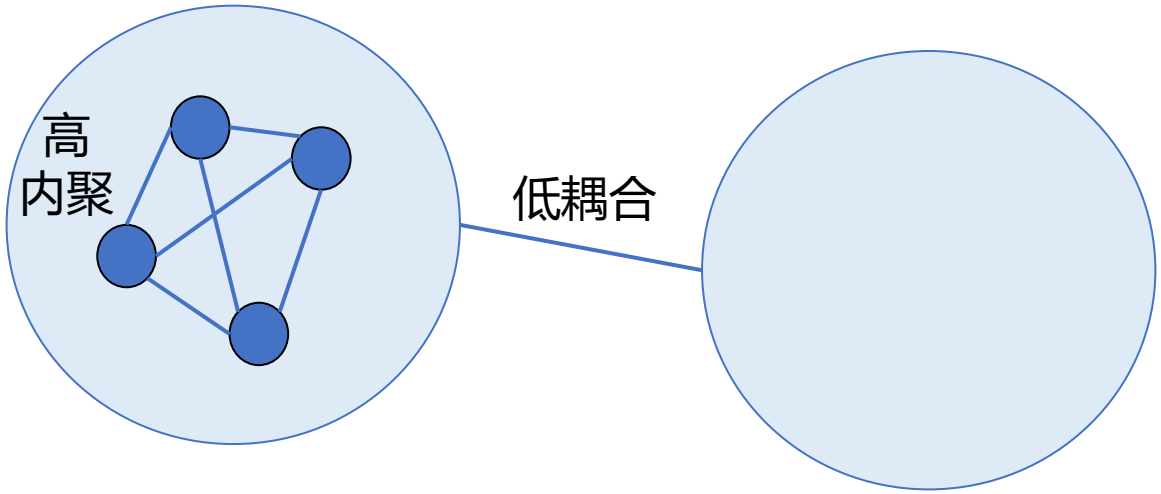
对于从前面的设计步骤得到的软件结构，使用设计度量和设计准则，还可以再修改：

- 输入结构中的模块“转换成rpm”和“收集sps”，模块太小，功能简单，可以合并；
- 模块“确定加速 / 减速”可以放在模块“计算mph”下面，以减少耦合；
- 模块“加速 / 减速显示”可以相应地放在模块“显示mph”的下面。

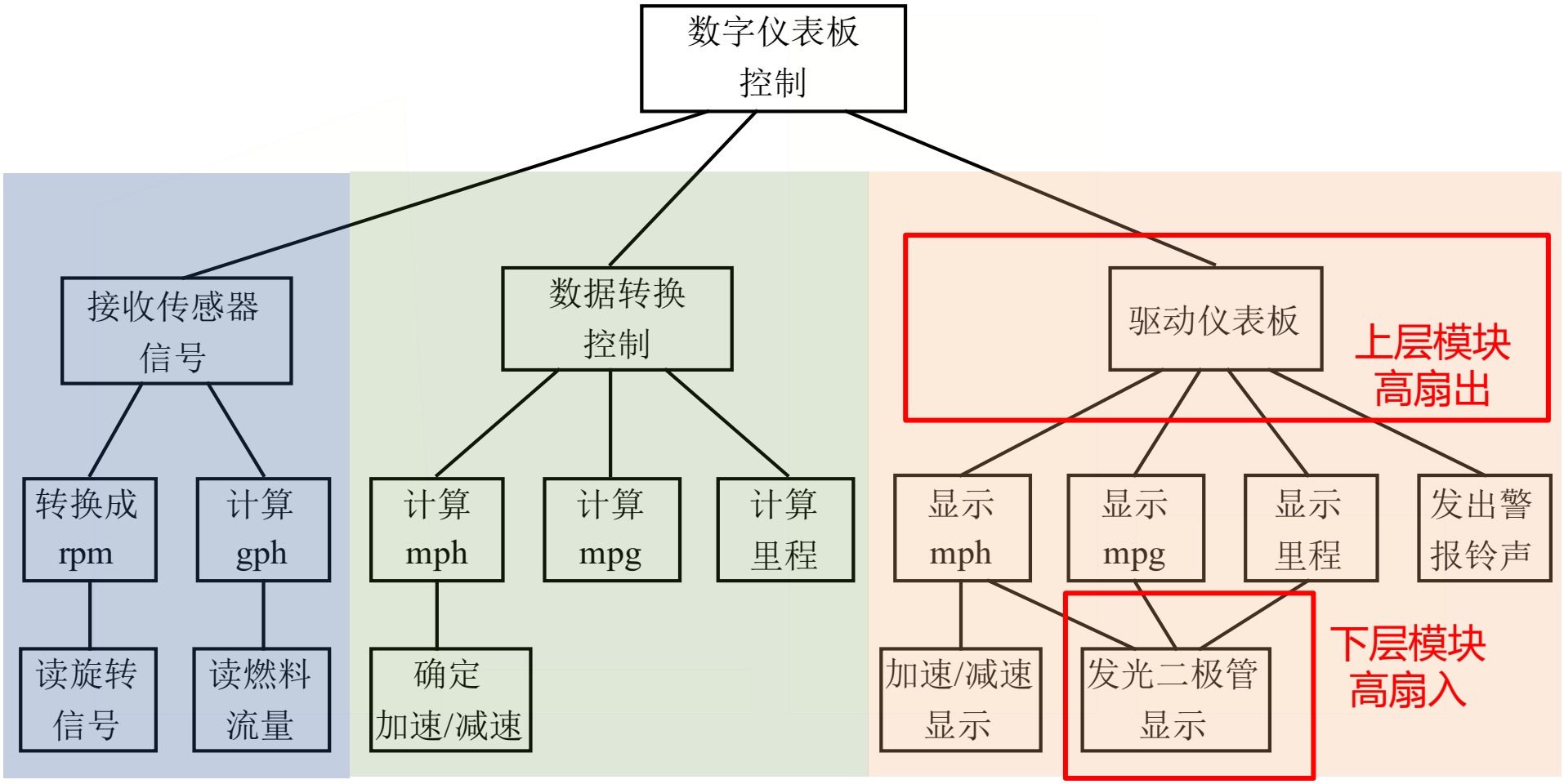


7、对第一次分割得到的软件结构进一步精化

精化的基本思路：模块化、高内聚、低耦合、上层模块高扇出、下层模块高扇入。



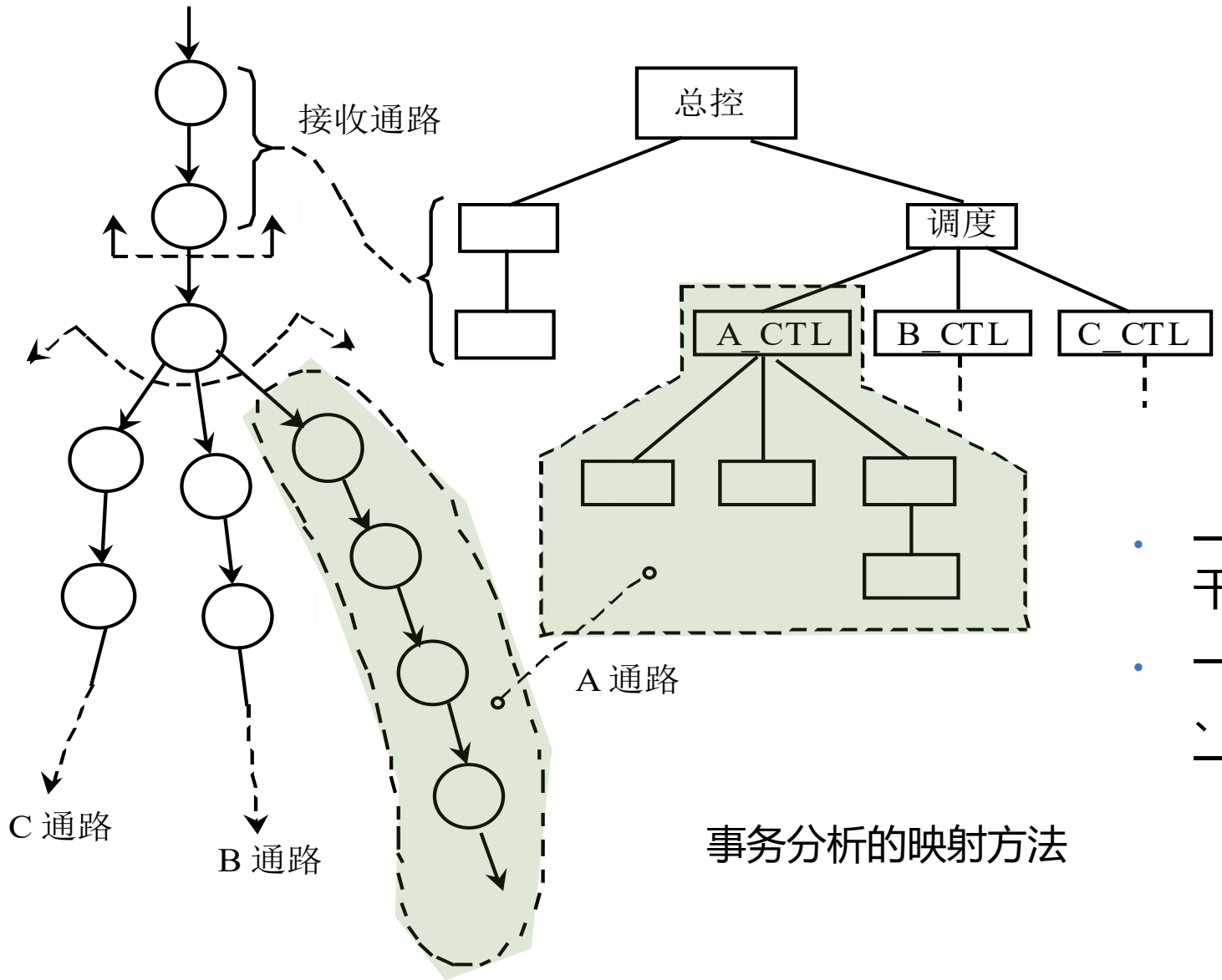
精化的数字仪表板系统的软件结构



精化后的数字仪表板系统的软件结构

5.5.3 变换设计

5.5.4 事务设计



事务分析的映射方法

- 一个总控模块，若干个二级模块
- 一个事务接收模块、每个事务流对应一个模块。

5.5.3 变换设计

5.5.5 设计的后处理

在确定系统的软件结构以后，还必须做好下述工作：

- 为每个模块开发一份功能说明；
- 为每个模块提供一份接口说明；
- 定义局部的和全程的数据结构；
- 给出所有的设计限制或约束；
- 进行总体设计评审；
- 如果需要和可能的话，进行设计“优化”。

第五章　总体设计

总体设计的目的

- 用比较抽象的方法确定系统概要地是如何实现的（How to do generally !）。
- 从初步的数据流图导出（设计出）软件结构。

总体设计的过程——9步

- 设想可能方案、选取合理方案、推荐最佳方案、功能分解细化DFD、设计软件结构、设计数据库、制定测试计划、编写文档、审查和复审。

总体设计的原理

- 模块化、抽象vs逐步求精、信息隐蔽/局部化、模块独立性（耦合、内聚）

总体设计的启发规则——7条原则

- 保证设计出良好的软件结构。

总体设计的图形工具

- 数据流图、层次图、HIPO图、结构图。

结构化设计方法（SD）

- 区分数据流图的类型（变换型、事务型）；
- 结构化设计方法的步骤（7步）