BUPT
TSEG

软件工程 模型与方法
Models & Methods of SE

结构化设计方法

本章内容

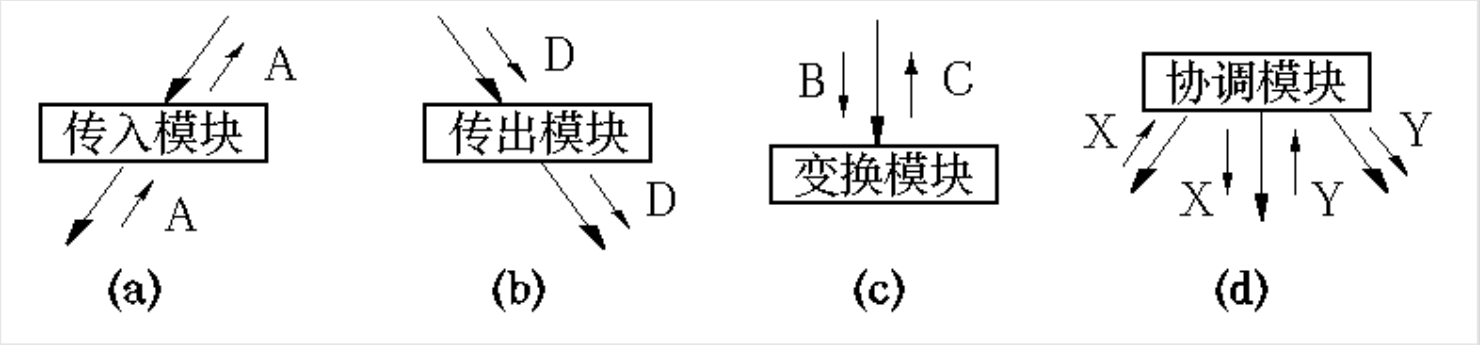
- 系统功能结构图
- 变换映射
- 事务映射
- 优化系统功能结构图
- 设计后处理
- 详细设计

引言

- 结构化设计方法依据需求分析的结果“数据流图”推导出软件的系统功能结构图。其要点是：
  - 建立数据流的类型。
  - 指明数据流的边界。
  - 将数据流图映射到程序结构。
  - 用“因子化”方法定义控制的层次结构。
  - 用设计测量和一些启发式规则对结构进行细化。

功能结构图的基本结构

- 四种基本类型的模块
  - 传入模块： 从下属模块取得数据，经过某些处理，再将其传送给上级模块。
  - 传出模块： 从上级模块获得数据，进行某些处理，再将其传送给下属模块。
  - 变换模块： 即加工模块。它从上级模块取得数据，进行处理，转换成其它形式，再传送回上级模块。
  - 协调模块： 对所有下属模块进行协调和管理的模块。

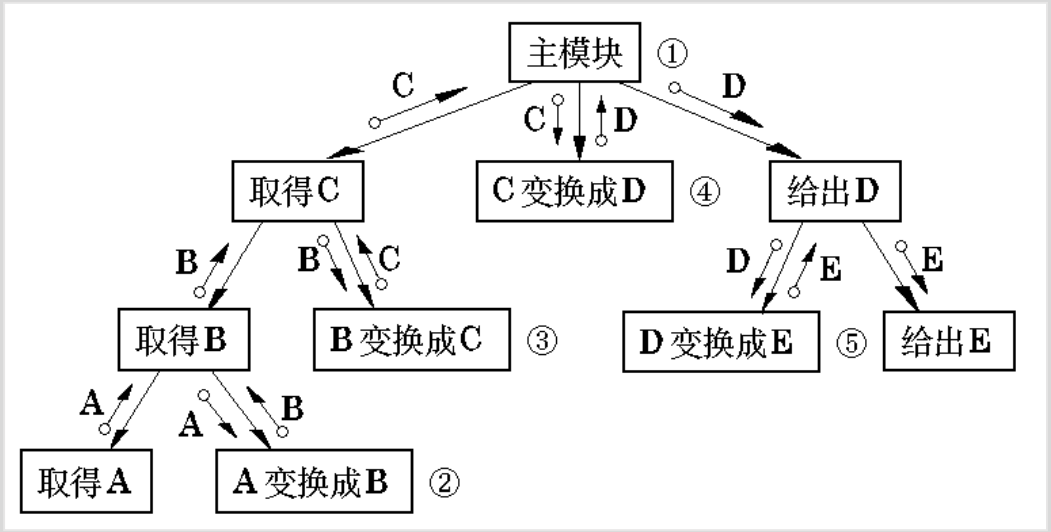


功能结构图的类型和分解

- 通常，系统功能结构图根据数据流图中加工特性分为以下两种结构：
  - 变换处理型
  - 事务处理型
- 原子模块：结构图中不能再分解的底层模块；
- 因子分解系统：
  - 所有系统的加工处理都由原子模块完成；
  - 其它非原子模块仅仅进行控制和协调的功能。

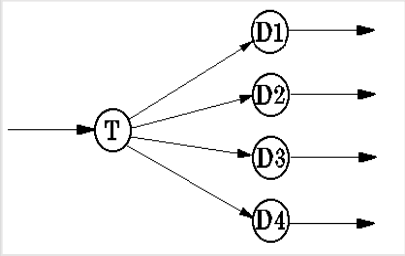
变换型结构

- 变换型数据处理问题的工作过程大致分为三步：
  - 取得数据
  - 变换数据
  - 给出数据

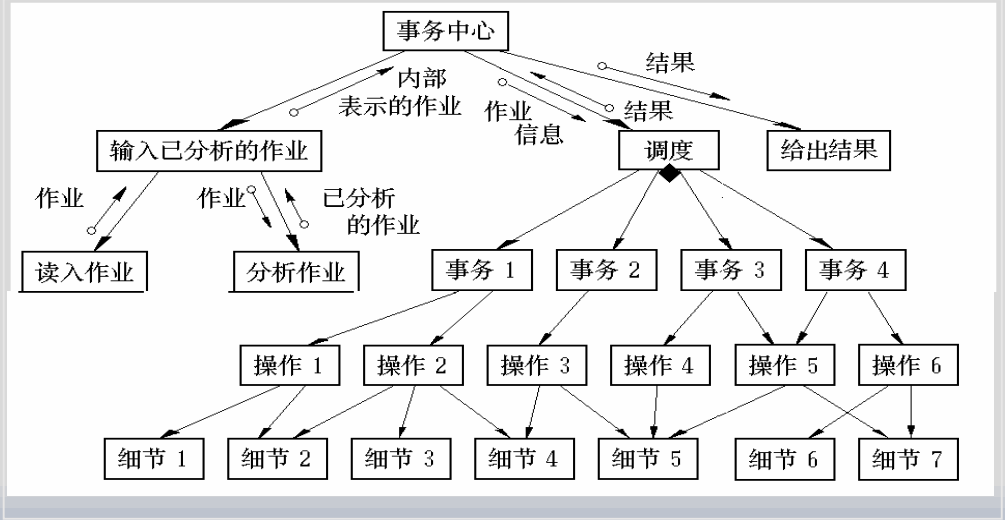


事务型结构

- 存在某一个数据流处理节点，引发一个或多个相同的处理，并将处理结果返回给该节点，则该数据流就叫做事务，该节点称为事务处理中心。
  - 事务处理中心
  - 事务处理加工



- 事务是最小的工作单元，不论成功与否都作为一个整体进行工作。
  - 原子性
  - 一致性
  - 隔离性
  - 持久性



变换映射

- 变换映射是一组设计步骤，将具有变换流特征的数据流图映射为一个预定义的程序结构模版。
- 运用变换映射方法建立初始的系统功能结构图，然后进行多次改进或优化，得到系统的最终结构图。
  - 复审并评估分析模型；
  - 复审并重画数据流图；
  - 确定数据流图中的变换和事务特征；
  - 区分输入流、输出流和中心变换部分，即标明数据流的边界；
  - 进行一级“因子化”分解，设计顶层和第一层模块；
  - 进行二级“因子化”分解，设计中、下层模块；
  - 利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。

重画数据流图

- 其出发点是描述系统中的数据是如何流动的；并根据需要将局部层次的数据流图合并为一层，便于理解和设计；
  - 以需求分析阶段的数据流图为基础，可以从物理输入到物理输出，或者相反；也可以从顶层加工开始，逐层向下；
  - 一般情况下，在图上不要出现控制逻辑（例如判定和循环等），箭头只表示数据流而非控制流；
  - 不用考虑系统的开始和结束；
  - 省略每一个加工的异常处理，只考虑主要加工处理逻辑；
  - 当数据流进入和离开一个加工时，要仔细地标记它们，不要重名。
  - 如有必要，可以使用逻辑运算符“与”和“或”。

确定系统边界

- 中心变换：多股数据流汇集的地方往往是系统的中心变换部分。
- 逻辑输入：可以从数据流图上的物理输入开始，一步一步向系统中间移动，一直到数据流不再被看作是系统的输入为止，则其前一个数据流就是系统的逻辑输入。
  - 可以认为逻辑输入就是离物理输入端最远的，且仍被看作是系统输入的数据流。
- 逻辑输出：从物理输出端开始，一步一步地向系统中间移动，就可以找到离物理输出端最远，且仍被看作是系统输出的数据流。

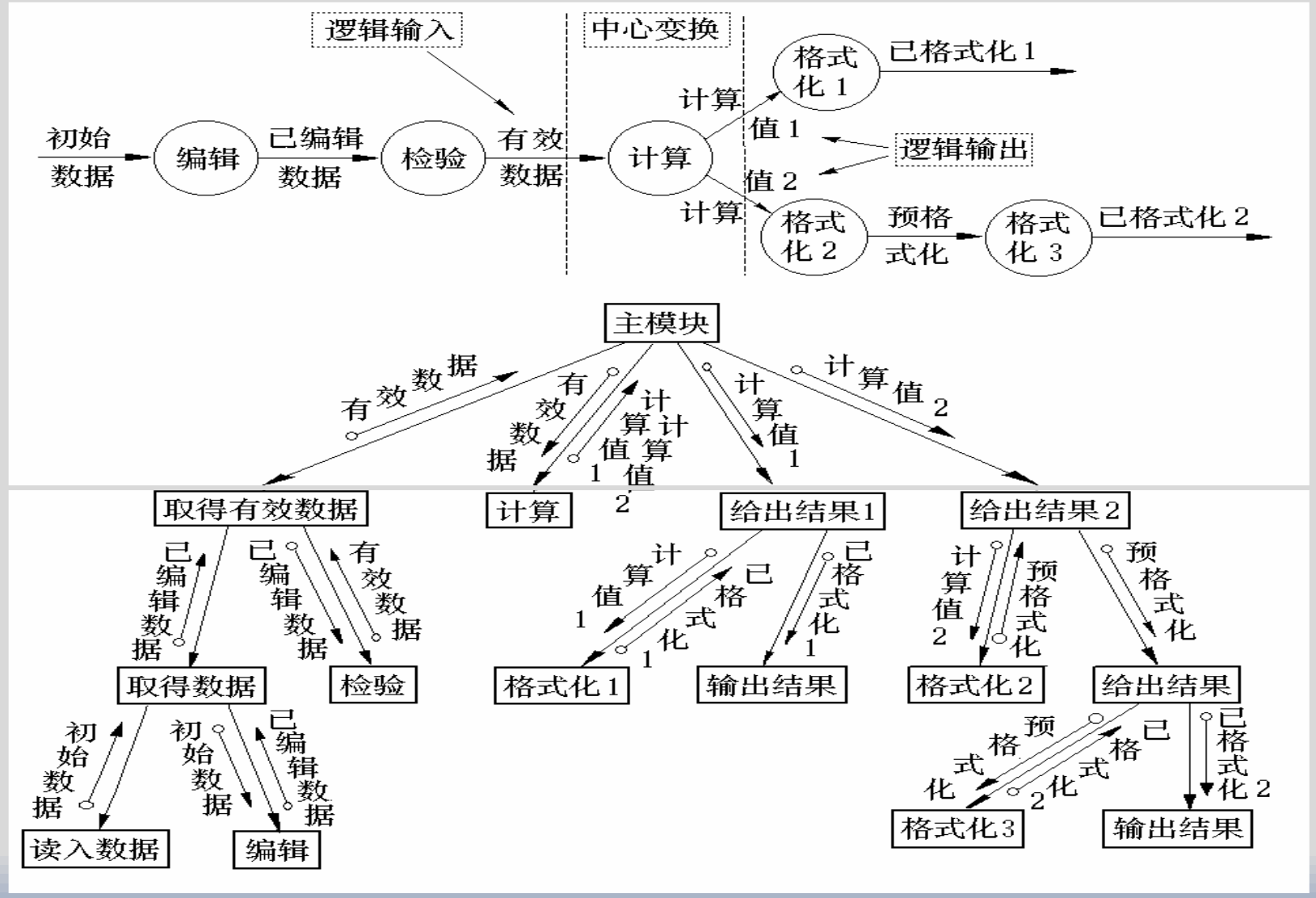
一级因子化分解

- 其主要任务就是设计顶层和第一层模块。
  - 顶层模块：
    - 主要起到控制和协调的作用；
    - 用程序名字命名（或者功能需求/主模块），将它画在与中心变换相对应的位置上；
    - 它调用下层模块，完成系统所要做的各项工作 。
  - 中层模块（第一层）：
    - 既完成一部分控制，又完成适当的变换工作；
    - 为每一个逻辑输入设计一个输入模块，为主模块提供数据；
    - 为每一个逻辑输出设计一个输出模块，它将主模块提供的数据输出；
    - 为中心变换设计一个变换模块，它将逻辑输入转换成逻辑输出；
    - 第一层模块与主模块之间传送的数据应与数据流图相对应。

二级因子化分解

- 其主要任务是设计中下层模块
  - 将数据流图中的每一个变换型加工映射为程序结构中的模块
    - 输入模块：取得模块（左侧），处理模块（右侧）
    - 输出模块：处理模块（左侧），给出模块（右侧）
    - 变换模块：表示功能需求。
  - 从变换中心的边界开始，沿输入路径和输出路径向外，将变换依次映射到低层的软件结构中去（直到外部实体）。
  - 最终给出初步的系统功能结构图

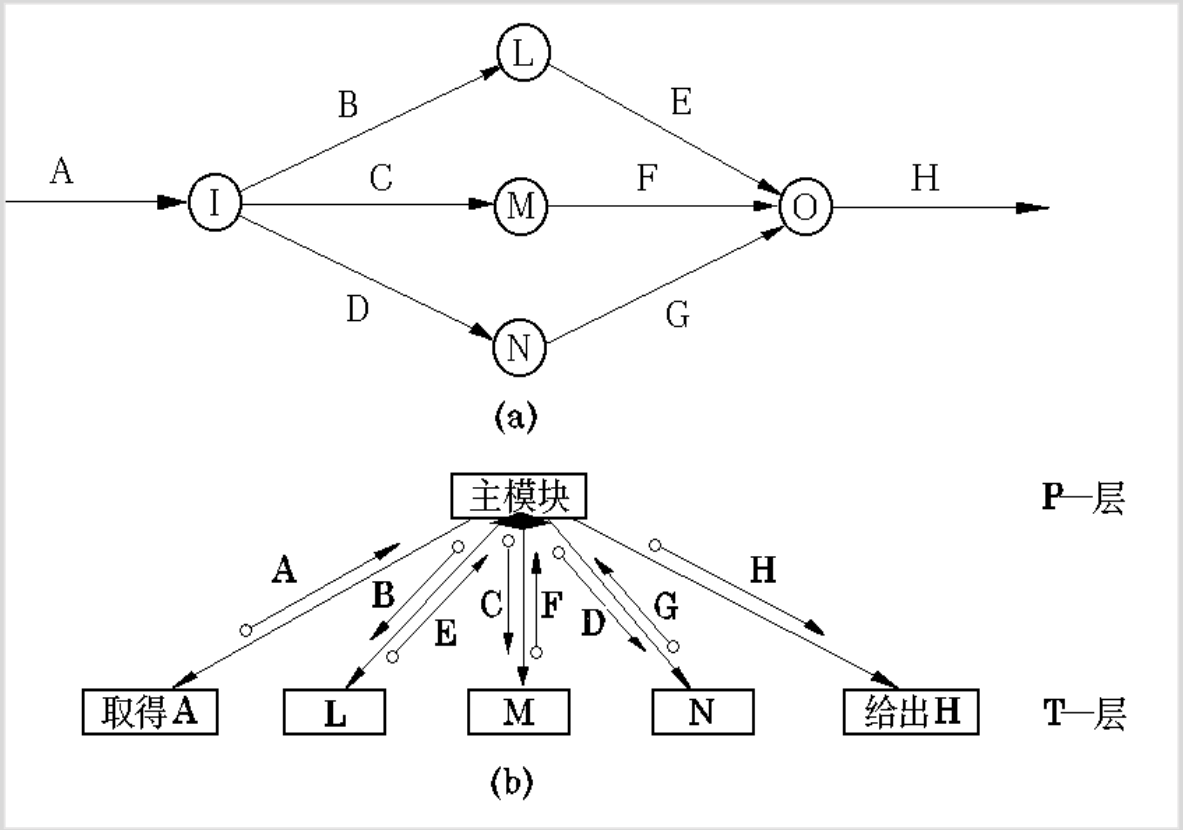
数据流图推导出的系统初始结构图



事务映射

- 事务映射也从分析数据流图开始，自顶向下，逐步分解，建立事务型系统结构图。

1. 复审系统分析模型
2. 重画数据流图
3. 确定是否具有事务流特征
4. 确定事务中心及流特征
5. 进行事务映射
6. 因子化分解和细化
7. 优化系统结构



识别事务中心

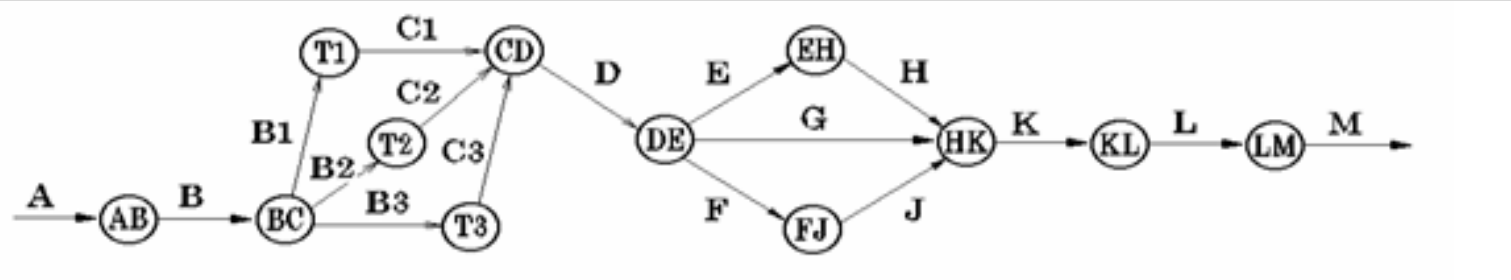
- 事务中心通常位于几条操作路径的起始点上，可以从数据流图上直接找出来。
  - 事务源：一个带有请求性质的数据流；
  - 事务中心：处理事务源的加工，且后继的多个加工必须是并列的并在事务中心的控制下完成不同功能处理。

事务映射

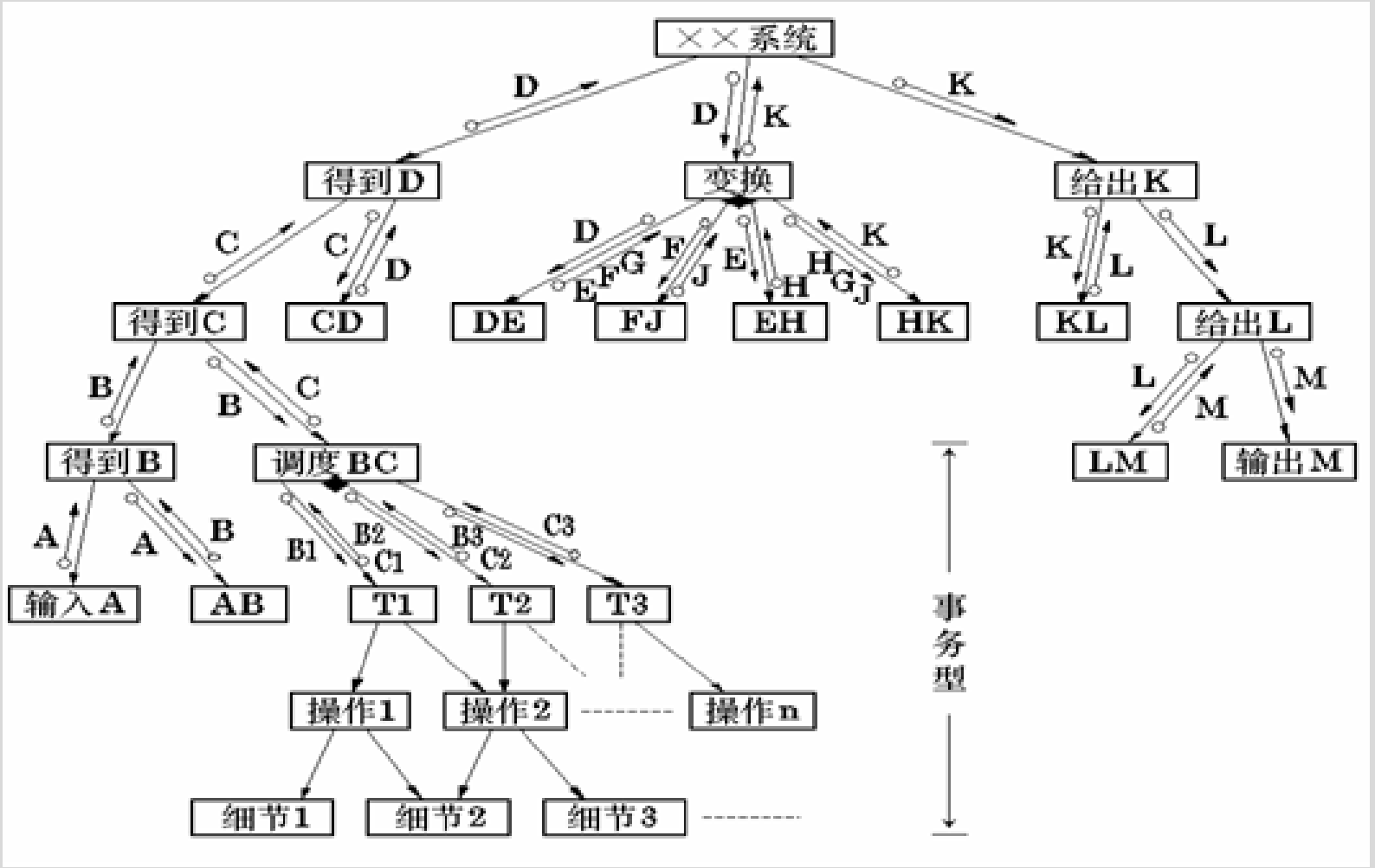
- 事务流应映射到包含一个输入分支和一个分类事务处理分支的程序结构上。
  - 输入分支结构的开发与变换流的方法类似
  - 分类事务处理分支结构包含一个调度模块，它调度和控制下属的事务处理模块。
    - 建立一个主模块用以代表整个加工，P层
    - 然后考虑被称为事务层的第二层模块，T层
    - 第二层模块只能是三类：取得事务、处理事务和给出结果。
    - 处理事务模块的下层为操作模块，A层
    - 操作模块之下为细节模块，D层

变换-事务混和型结构

- 一般来讲，一个大型的软件系统不可能是单一的数据变换型，也不可能是单一的事务型，通常是变换型结构和事务型结构的混合体。
- 在具体的应用中一般以变换型为主，事务型为辅的方式进行软件结构设计。



变换-事务混和型结构


××系统
D
D K
K
得到D
变换
给出K
C
C D
D E F G F J E H G J K
K L
L
得到C
CD
DE
FJ
EH
HK
KL
给出L
B
B C
L M
M
得到B
调度BC
LM
输出M
A
A B
B1 B2 C1
B3 C2
C3
输入A
AB
T1
T2
T3
事务型
操作1
操作2
操作n
细节1
细节2
细节3

软件模块的优化原则

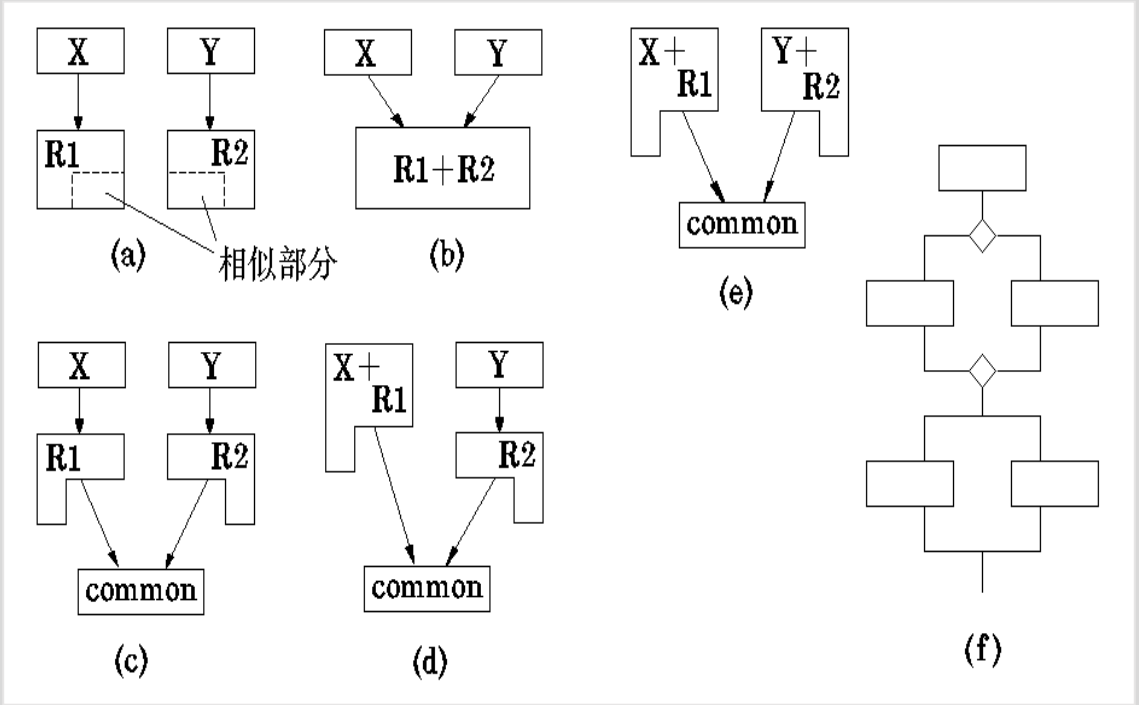
- 模块功能的完善化
- 消除重复功能，改善软件结构
- 模块的作用范围应在控制范围之内
- 尽可能减少高扇出结构
- 避免或减少使用病态联接
- 模块的大小要适中
- 设计功能可预测的模块，避免过分受限制的模块
- 软件包应满足设计约束和可移植性

模块功能的完善化

- 一个完整的功能模块，不仅应能完成指定的功能，而且还应当能够告诉使用者完成任务的状态，以及不能完成的原因。
  - 规定的功能部分；
  - 出错处理部分。当模块不能完成规定的功能时，必须返回出错信息和标志，向它的调用者报告出现这种例外情况的原因
  - 如果需要返回一系列数据给它的调用者，当完成数据加工时应给它的调用者返回一个该模块执行是否正确结束的“标志”。

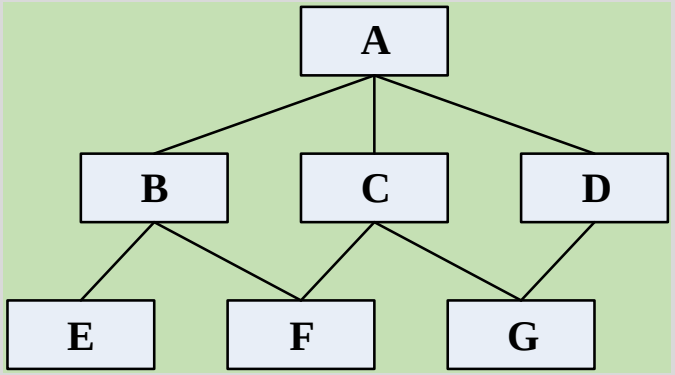
消除重复功能，改善软件结构

- 在得到初始的功能结构图之后，如果发现有几个模块有相似之处，可加以改进
  - 完全相似：在结构上完全相似，可能只是在数据类型上不一致。
  - 局部相似：需要考虑功能之间的耦合及内聚性



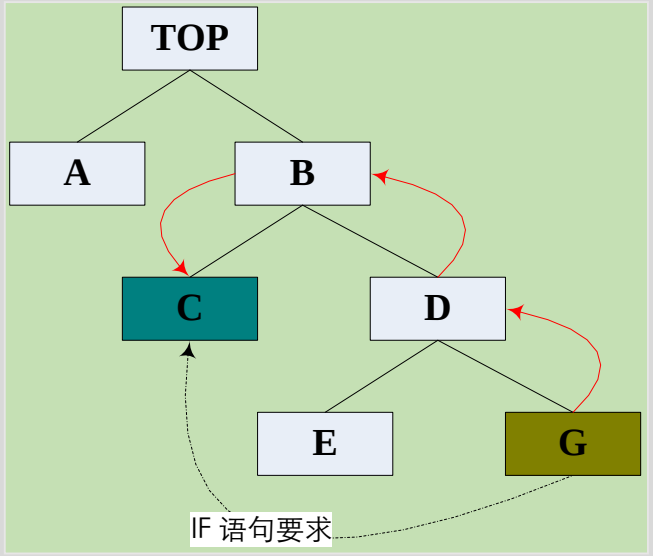
模块的作用范围和控制范围

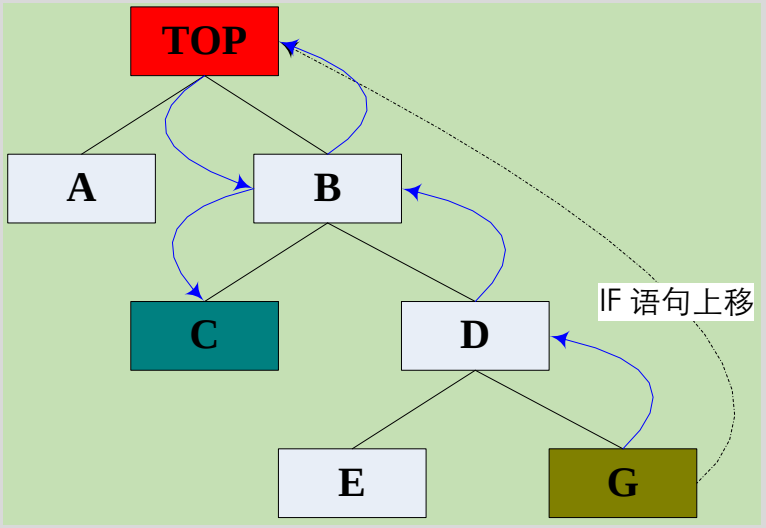
- 模块的控制范围包括它本身及其所有的从属模块；
- 模块的作用范围是指模块内一个判定的作用范围，凡是受这个判定影响的所有模块都属于这个判定的作用范围。
- 如果一个判定的作用范围包含在这个判定所在模块的控制范围之内，则这种结构是简单的，否则，它的结构是复杂的。

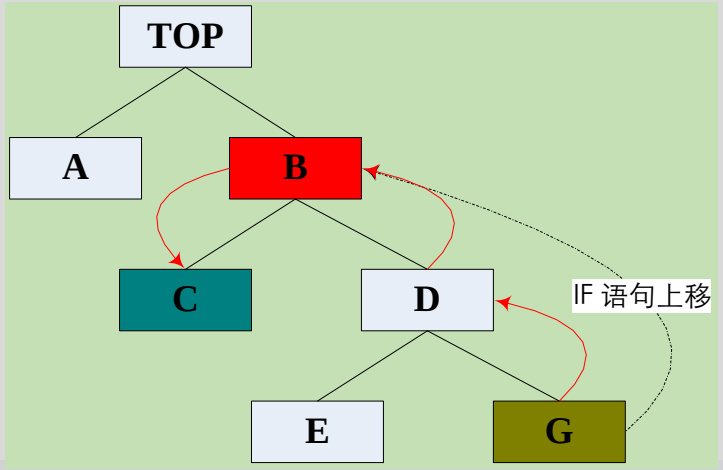


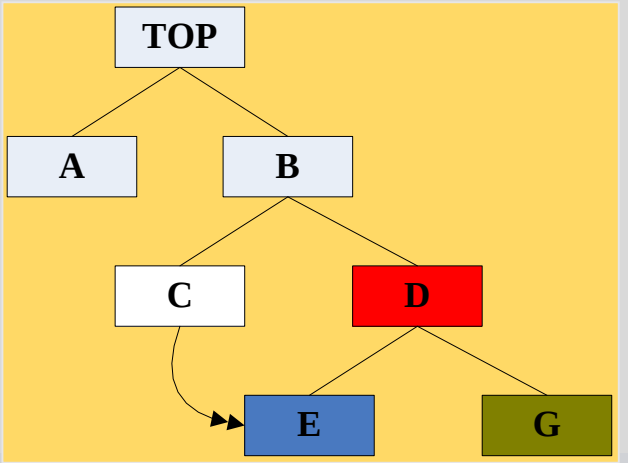
模块A的控制范围为：
ABCDEFG。
模块C的控制范围为：
CFG。

模块的作用范围应在控制范围之内









模块的作用范围与控制范围

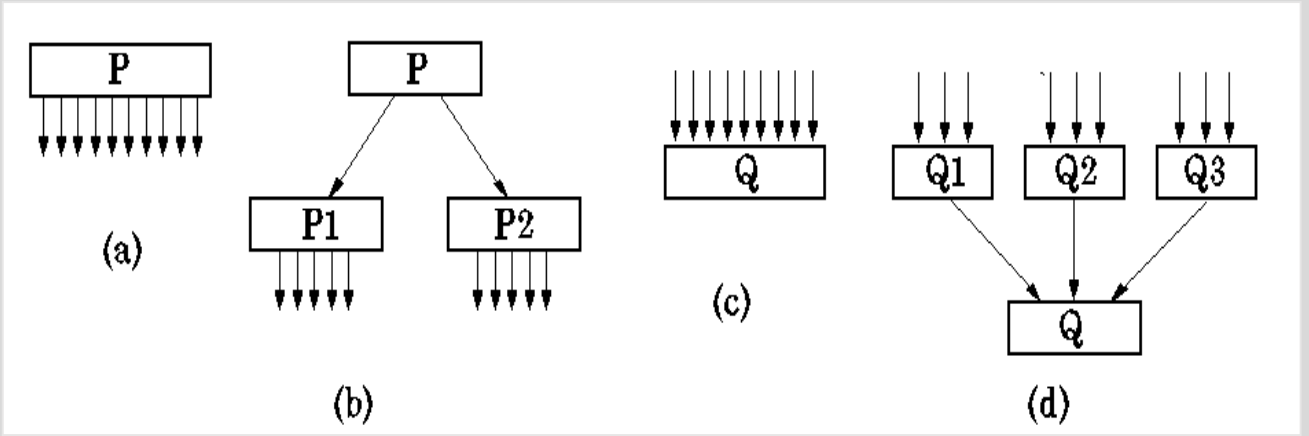
- 建议，所有受一个判定影响的模块应该都从属于该判定所在的模块，最好局限于做出判定的那个模块及其直接下属模块。
- 在设计过程中，当遇到作用范围不在控制范围之内，可应用如下办法把作用范围移到控制范围之内：
  - 将判定所在模块合并到父模块中，使判定处于较高的层次；
  - 将受到判定影响的模块下移到控制范围内；

尽可能减少高扇出结构

- 经验证明，一个设计得很好的软件模块结构，通常上层扇出比较高，中层扇出较少，底层扇入到有高扇入的公用模块中。
  - 模块的扇出过大，将使得系统的模块结构图的宽度变大，宽度越大结构图越复杂。比较适当的模块扇出数目为2~5，最多不要超过9。
  - 模块的扇出过小也不好，这样将使得系统的功能结构图的深度大大增加，不但增加了模块接口的复杂度，而且增加了调用和返回的时间开销，降低系统的工作效率。

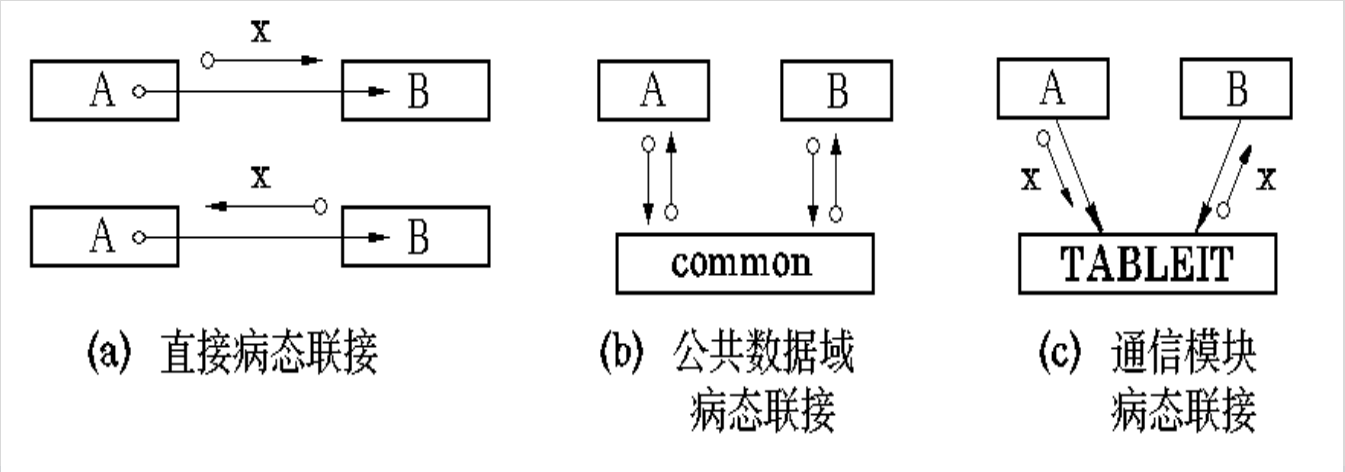
扇入扇出结构的调整

- 模块的扇出指模块调用子模块的个数。如果模块的扇出过大，就表明模块过分复杂，需要协调和控制过多的下属模块。
- 一个模块的扇入数目很大，说明共享该模块的上级模块数目增多。
  - 但如果一个模块的扇入太大，比如超过7或8，而且它又不是公用模块，说明该模块可能具有多个功能。为此应当对其进一步分析并将其功能分解。



避免或减少使用病态联接

- 直接病态联接：即模块A直接从模块B内部取出某些数据，或者把某些数据直接送到模块B内部
- 公共数据域病态联接：模块A和模块B通过公共数据域，直接传送或接受数据，而不是通过它们的上级模块。
- 通信模块联接：即模块A和模块B通过通信模块TABLEIT传送数据。



模块的大小要适中

- 模块的大小是指模块内部结构的多少。
- 限制模块的大小是减少复杂性的手段之一，因而要求把模块的大小限制在一定的范围之内。
  - 通常规定其语句行数在50～100左右，最多不超过500行。
  - 体积过大的模块往往是由于分解不充分，且具有多个功能，因此需要对功能进一步分解，生成一些下级模块或同层模块。
  - 反之，模块体积较小时也可以考虑是否可能与调用它的上级模块合并。

应满足设计约束和可移植性

- **运用变换分析方法建立系统的结构图时应当注意以下几点：**
  - 在选择模块设计的次序时，必须对一个模块的全部直接下属模块都设计完成之后能转向另一个模块的下层模块的设计。
  - 设计下层模块时，应考虑模块的低耦合和高内聚问题，提高初始结构图的质量。
  - 注意黑盒技术的使用。
  - 如果出现了以下情况，就停止模块的功能分解
  - 当模块不能再细分为明显的子任务时；
  - 当分解成用户提供的模块或程序库的子程序时；
  - 当模块的界面是输入 / 输出设备传送的信息时；
  - 当模块不宜再分解得过小时。

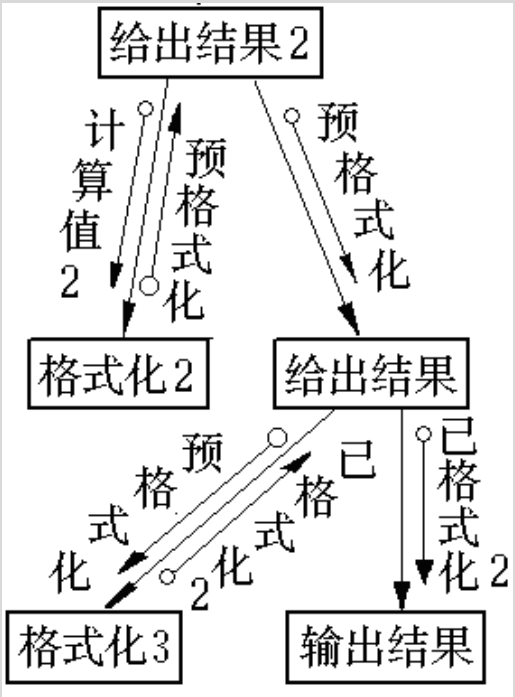
设计后处理

- 在经过变换映射和事务映射之后，还需要为所获得的系统功能结构图进行说明，形成《概要设计说明书》，包括以下内容：
  - 必须为每一个模块写一份处理说明；
  - 为每一个模块提供一份接口说明；
  - 确定全局数据结构和局部数据结构；

处理说明

- 处理说明是一个关于模块内部处理的清晰且无歧义的正确描述，包含了模块的主要处理任务、条件抉择和输入 / 输出。

“给出结果2”模块调用 “格式化2”模块，将内部编码形式的计算结果2转换成以ASCII码表示的文本形式的预格式化数据，再调用 “给出结果” 模块，进一步转换成按预定的图表安排的形式输出。



接口说明

- 接口说明要给出一张表格，列出所有进入模块和从模块输出的数据。应包括
  - 通过参数表传递的信息
  - 对外界的输入 / 输出信息
  - 访问全局数据区的信息
  - 指出其下属的模块和上级模块。

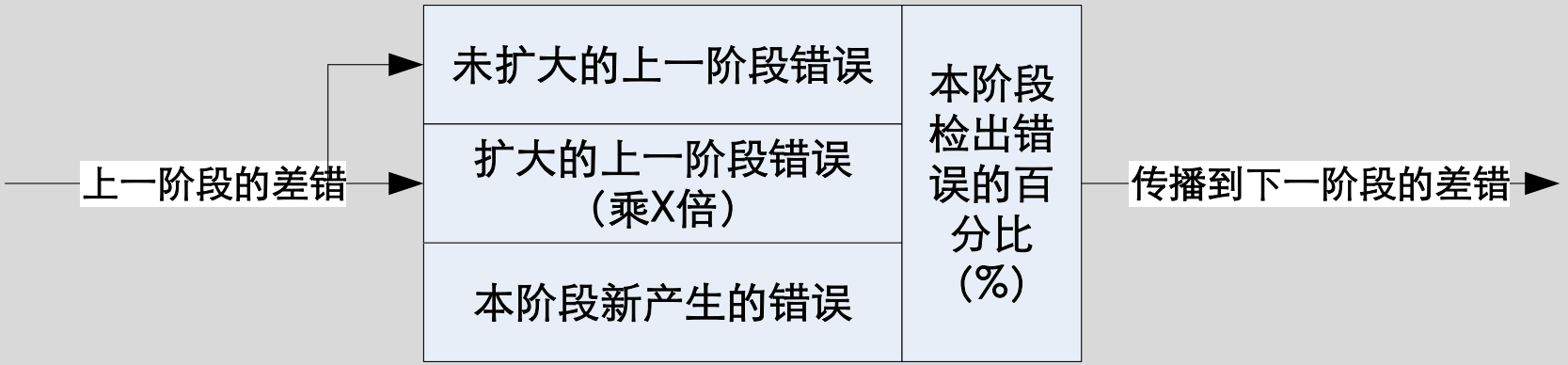
```
PROCEDURE format-2;                          //过程format-2（格式化2）
    INTERFACE ACCEPTS;                       //入口
    TYPE calc-value-2 IS BINARY CODE;        //类型 calc-value-2是二进制码
    INTERFACE RETURNS;                       //出口
    TYPE preformatted-data IS NUMERIC        //类型preformatted-data是数值型
    * no external I／O or global data Used   //无外部I／O或全局数据
    * called by put-result-2                 //所调用：模块put-result-2
    * calls no subordinate modules           //调用：无下属模块
```

数据结构说明

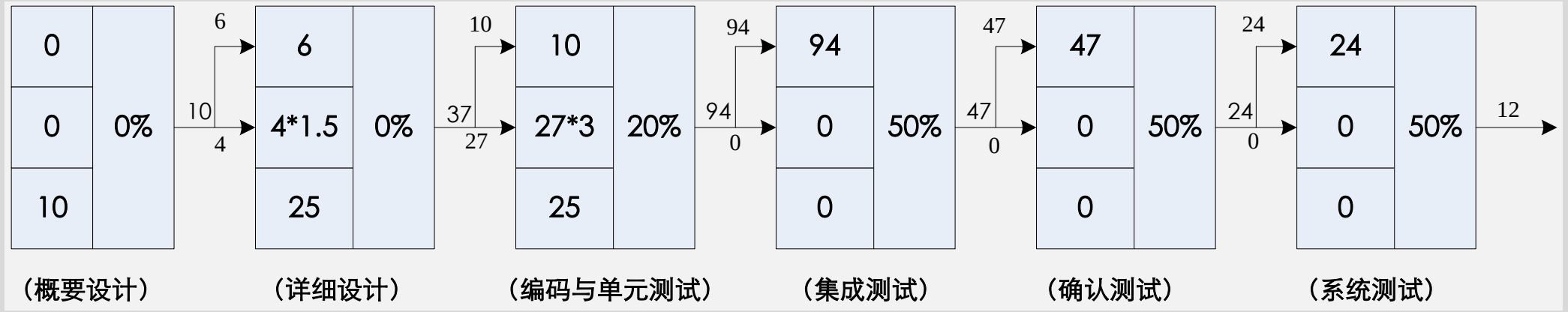
- 数据结构的描述可以用伪码（如PDL语言、类PASCAL语言）或Warnier图等形式表达。

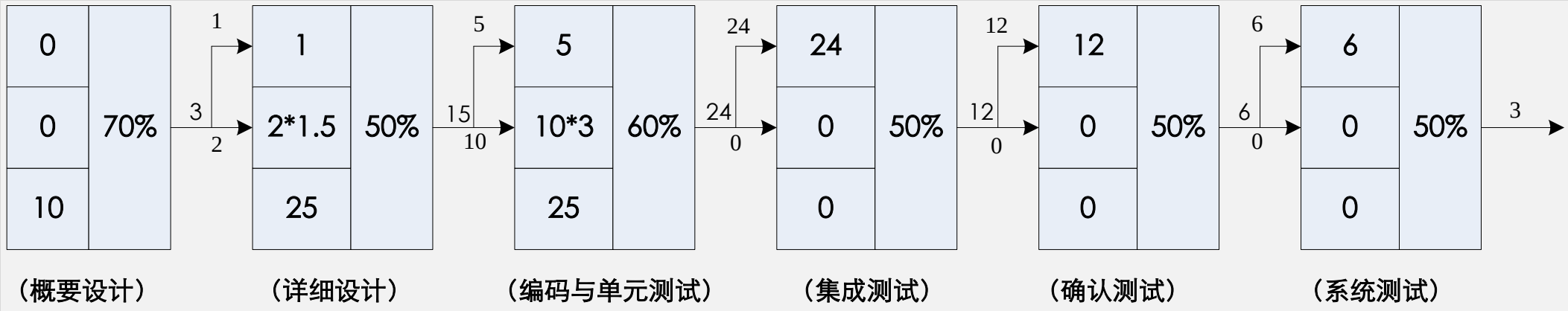
概要设计评审

- 在评审中应着重评审软件需求是否得到满足，软件结构的质量、接口说明、数据结构说明、实现和测试的可行性和可维护性等。
- 评审阶段还需要考虑和解决设计中存在的错误，以防错误在后期的软件过程中产生放大，即错误的扩大效应。



设计评审的作用





设计的优化

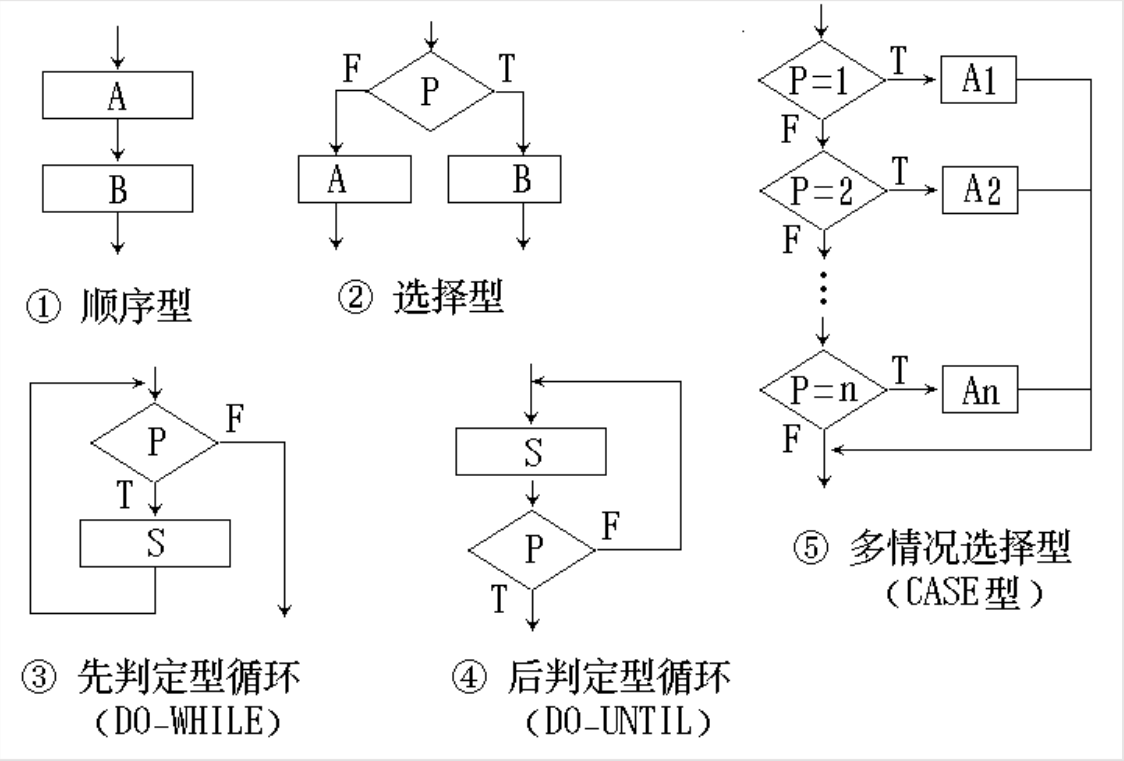
- 简明的结构往往是精巧的和高效的。
  - 因此，优化要力争使模块的个数最少；
  - 同时，还应当寻求尽量简单的满足信息需求的数据结构。
- 对于有时间运行要求的应用问题，在详细设计阶段和编码阶段必须进行优化
  - 在不考虑时间运行要求的条件下构造并改进软件的结构。
  - 在细节设计的过程中，挑出那些有可能占用过多时间的模块，并为这些模块精心设计出时间效率更高的过程（算法）。
  - 用高级程序设计语言编写代码程序。
  - 检测软件，分离出占用大量处理机资源的模块。
  - 如果有必要，用依赖机器的语言(机器指令、汇编语言)重新设计或重新编码，以提高软件的效率。

详细设计

- 从软件开发的工程化观点来看，在编制程序以前，需要对所采用算法的逻辑关系进行分析，设计出全部必要的过程细节，并给予清晰的表达，使之成为编码的依据，这就是详细设计的任务。
- 表达详细设计规格说明的工具叫做详细设计工具，它可以分为三类：
  - 图形工具
  - 表格工具
  - 语言工具

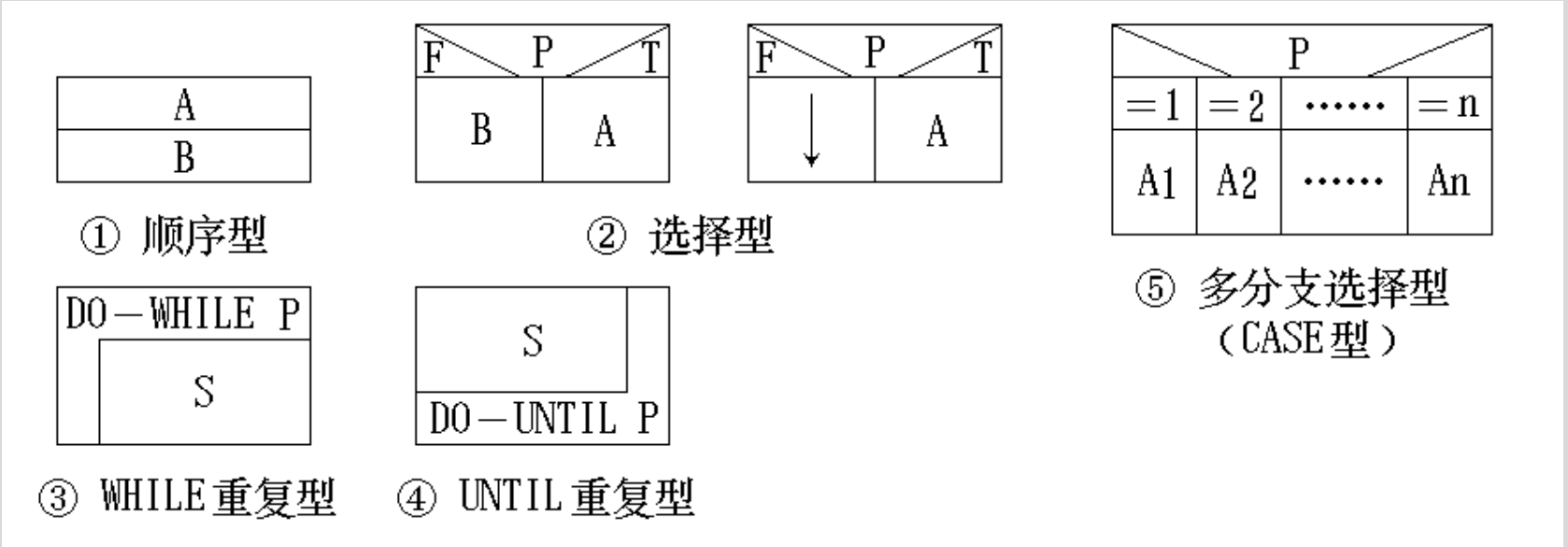
程序流程图

- 程序流程图独立于任何一种程序设计语言，比较直观、清晰，易于学习掌握。
- 为使用流程图描述结构化程序，必须限制流程图只能使用下面给出的五种基本控制结构。



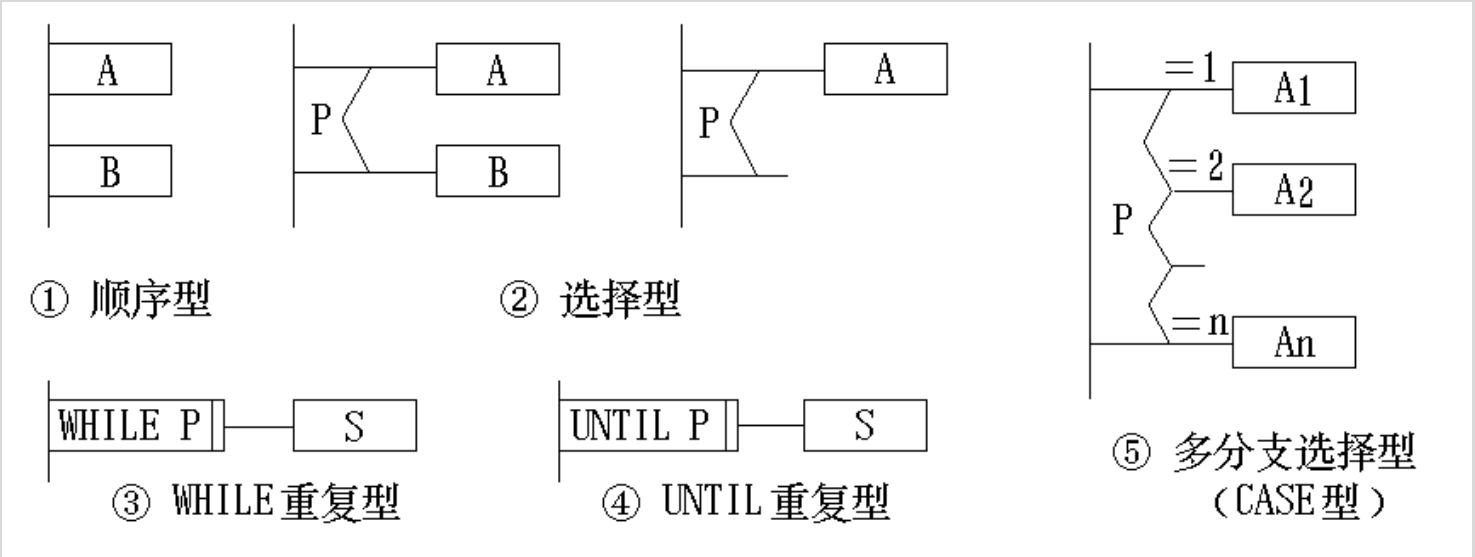
N-S图

- Nassi和Shneiderman 提出了一种符合结构化程序设计原则的图形描述工具，叫做盒图，也叫做N-S图。



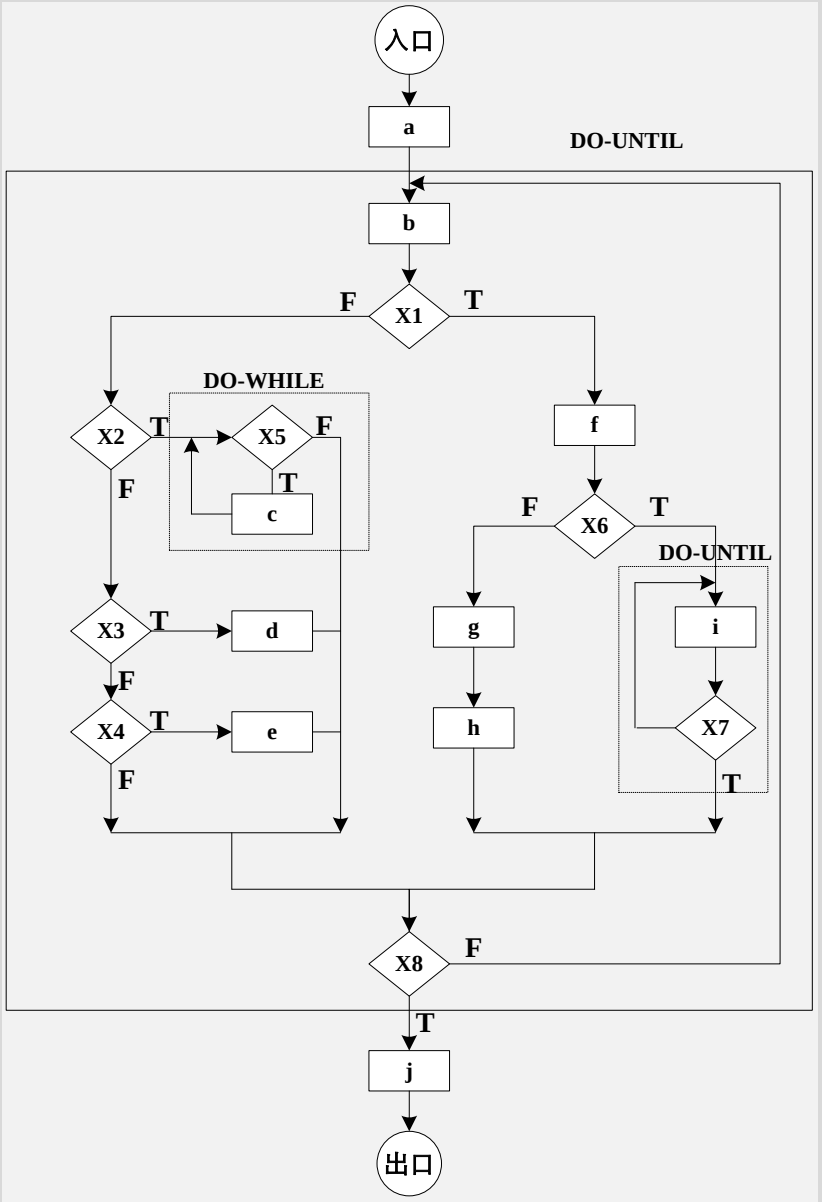
PAD图

- PAD是Problem Analysis Diagram的缩写，是日立公司提出，用结构化程序设计思想表现程序逻辑结构的图形工具。
- PAD也设置了五种基本控制结构的图式，并允许递归使用。现在已为ISO认可。



判定表

- 当算法中包含多重嵌套的条件选择时，用程序流程图、N-S图或PAD都不易清楚地描述。然而，判定表却能清晰地表达复杂的条件组合与应做动作之间的对应关系。
- 判定表要求不能存在多分支判断，必须是两分支的判断



判定表

判定表右半部的每一列实质上是一条规则，规定了与特定条件取值组合相对应的动作。

“T”表示该条件取值为真，
“F”表示该条件取值为假。

空白表示这个条件无论取何值对动作的选择不产生影响。

“Y”表示要做这个动作，空白表示不做这个动作。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X1	T	T	T	T	T	F	F	F	F	F	F	F	F	F
X2	–	–	–	–	–	T	T	T	F	F	F	F	F	F
X3	–	–	–	–	–	–	–	–	F	F	T	T	F	F
X4	–	–	–	–	–	–	–	–	F	F	–	–	T	T
X5	–	–	–	–	–	T	F	F	–	–	–	–	–	–
X6	T	T	T	F	F	–	–	–	–	–	–	–	–	–
X7	T	T	F	–	–	–	–	–	–	–	–	–	–	–
X8	T	F	–	T	F	–	T	F	F	T	T	F	T	F
a	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
b	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
c	–	–	–	–	–	Y	–	–	–	–	–	–	–	–
d	–	–	–	–	–	–	–	–	–	–	Y	Y	–	–
e	–	–	–	–	–	–	–	–	–	–	–	–	Y	Y
f	Y	Y	Y	Y	Y	–	–	–	–	–	–	–	–	–
g	–	–	–	Y	Y	–	–	–	–	–	–	–	–	–
h	–	–	–	Y	Y	–	–	–	–	–	–	–	–	–
i	Y	Y	Y	–	–	–	–	–	–	–	–	–	–	–
j	Y	–	–	Y	–	–	Y	–	–	Y	Y	–	Y	–

PDL

- PDL是一种伪码，用于描述功能模块的算法设计和加工细节的语言。
- 伪码的语法规则分为“外语法”和“内语法”。
  - 外语法应当符合一般程序设计语言常用语句的语法规则；
  - 内语法可以用英语中一些简单的句子、短语和通用的数学符号，来描述程序应执行的功能。

PDL 的特点

- 有固定的关键字外语法，提供全部结构化控制结构、数据说明和模块特征。为了区别关键字，规定关键字一律大写，其它单词一律小写。
- 内语法使用自然语言来描述处理特性。内语法只要写清楚就可以，不必考虑语法错，以利于人们可把主要精力放在描述算法的逻辑上。
- 有数据说明机制，包括简单的（如标量和数组）与复杂的（如链表和层次结构）的数据结构。
- 有子程序定义与调用机制，用以表达各种方式的接口说明。

PDL 举例

- 一个具有查找拼写错误单词功能的算法

```
PROCEDURE  spell_check  IS
    BEGIN
       split document into single  words
       look up words in dictionary
       display words which are not in dictionary
       create a new dictionary
    END spell_check
```

PDL 细化

```
PROCEDURE spell_check
BEGIN
    //split document into single words
    LOOP get next word
            add word to word list in sort-order
            EXIT WHEN all words processed
    END LOOP
    //look up words in dictionary
    LOOP get word from word list
            IF word not in dictionary THEN
            //display words not in dictionary
            display word, prompt on user terminal
          IF user response says word OK THEN
                    add word to good word list
            ELSE
             add word to bad word list
          ENDIF
            ENDIF
    EXIT WHEN all words processed
    END LOOP
    //create a new words dictionary
    dictionary := merge dictionary and good word list
END spell_check
```

界面设计

- 软件界面不仅是软件系统功能体现最直接的表现方式，它也是验证用户需求与功能实现是否匹配的一种有效方式。
- 软件开发人员期望利用自身的专业知识，将用户不完整甚至片面的需求用界面的方式将系统功能体现出来，以达到补充、修改和统一对用户需求的理解和认识。
- 界面设计主要包括三个方面：
  - 设计软件构件间的接口；
  - 设计模块和其他非人的信息生产者和消费者（比如其他外部实体）的接口；
  - 设计人（如用户）和计算机间的界面。

软件界面的特点

- 软件界面具有以下特点：
  - 不仅要能充分体现系统的功能
  - 还要兼顾用户使用的习惯（键盘还是鼠标）和知识水准（一般用户还是具有高水平计算机能力的人员）
  - 甚至还要美观大方和灵巧实用
  - 不仅如此软件的界面还需要紧跟潮流
- **Theo Mandel**提出了三条“黄金规则”
  - 置用户于控制之下；
  - 减少用户的记忆负担；
  - 保持界面一致。

置用户于控制之下

- 是系统的功能控制用户的操作，还是由用户定义并控制系统的功能?
- Mandel定义了一组设计原则允许用户操作控制：
  - 以不强迫用户进入不必要的或不希望的动作方式来定义交互模式；
  - 提供灵活的与界面交互的方式；
  - 允许用户交互可以被中断和撤销；
  - 当技能级别增长时可以使交互流水化并允许定制交互；
  - 使用户隔离内部技术细节。；
  - 设计应允许用户与出现在屏幕上的对象直接交互。

建少用户的记忆负担

- 当用户必须记住的东西越多时，与系统交互时出错的可能性也越大。为此，一个很好的用户界面设计不应加重用户的记忆负担。
- Mandel定义了一组设计原则，使得界面能够减少用户的记忆负担：
  - 减少对短期记忆的要求；
  - 建立有意义的缺省；
  - 定义直觉性的捷径；
  - 界面的视觉布局应该基于真实世界的背景；
  - 以不断进展的方式提示信息。

保持界面一致

- 用户应以一致的方式展示和获取信息：
  - 所有可视信息的组织均按照贯穿所有屏幕显示所保持的设计标准，
  - 输入机制被约束到有限的集合，在整个应用中被一致地使用，
  - 从任务到任务的导航机制被一致地定义和实现。
- Mandel定义了一组保持界面一致性的设计原则：
  - 允许用户将当前任务放入有意义的环境中
  - 在应用系列内保持一致性

界面的设计模型

- 设计用户界面时要考虑四种模型：
  - 软件工程师创建设计模型；
  - 系统分析人员建立的用户模型；
  - 终端用户在脑海里对界面产生的映像称为用户的模型或系统感觉；
  - 系统的实现者创建系统界面模型。
- 为了建立有效的用户界面，“开始界面设计之前，必须对用户加以了解：
  - 新手
  - 对系统有了解的中级用户
  - 对系统有了解的经验用户

界面的设计过程

- 用户界面的设计过程是迭代的，其过程包含四个不同的框架任务：
  - 用户、任务和环境分析及建模；
    - 界面将物理地位于何处？
    - 用户是否将坐着、站着或完成其他和该界面无关的任务？
    - 显示设备是否适应空间、光线或噪音约束？
    - 是否存在特殊的环境因素而需要的特殊考虑？
  - 界面设计；
  - 界面构造；
  - 界面确认；

界面设计的常见问题

- 在进行用户界面设计时，通常会遇到以下四种问题：
  - 系统响应时间
  - 帮助信息
  - 错误信息处理
  - 命令标记

系统联机帮助

- **在进行系统交互时，是否总能得到各种系统功能的帮助?**
  - 提供部分功能的帮助
  - 提供全部功能的帮助。
- **用户怎样请求帮助?**
  - 帮助菜单
  - 特殊功能提示
  - HELP命令。
- **怎样表示帮助?**
  - 在另一个窗口中
  - 指出参考某个文档（非理想方式）
  - 在屏幕特定位置的简单提示。
- **用户怎样回到正常的交互方式?**
  - 屏幕上显示返回按钮
  - 功能键或控制序列。
- **怎样构造帮助信息?**
  - 平面式（所有信息均通过一个关键词来访问）
  - 分层式（用户可以进一步查询得到更详细的信息）
  - 超文本式。

错误信息处理

- ***SEVERE SYSTEM FAILURE –14A***
  - 这是什么意思？晕菜
- 出错消息和警告应具备以下特征：
  - 以用户可以理解的术语描述；
  - 应提供如何从错误中恢复的建设性意见；
  - 应指出错误可能导致哪些不良后果（比如破坏数据），以使用户检查是否出现了这些情况或帮助用户进行改正；
  - 应伴随着视觉或听觉上的提示，即显示消息时应伴随着警告或者消息用闪烁方式，或明显的颜色进行提示；
  - 不能带有判定色彩，即不能指责用户的操作不当，因为错误信息都是系统的问题而非用户造成的。