软件工程导论复习

1 软件工程学概述

2 可行性研究

3 需求分析

4 形式化说明技术

5 总体设计

6 详细设计

7 实现

8 维护

9 面向对象相关内容

10 软件项目管理

第1章　软件工程学概述

1、什么是软件危机？P1

2、软件危机产生原因有哪些？P3

3、什么是软件工程？P5

4、软件工程方法学包含哪些要素？P9

5、在软件过程中有哪些模型？他们各自特点是哪些？P15—P22

第2章　可行性研究

1、可行性研究目标是什么？P25

2、应从哪些方面研究可行性？P25

3、怎样画系统流程图？P27—P30

4、怎样画数据流图？P30—P37

5、了解数据字典及成本效益分析。

2.4 数据流图（DFD）

数据流图（Data Flow Diagram，DFD）是描述系统中数据流程图形工具，它标识了一个系统逻辑输入和逻辑输出，以及把逻辑输入转换为逻辑输出所需加工处理。

一、数据流图图符

四种基本图形符号：

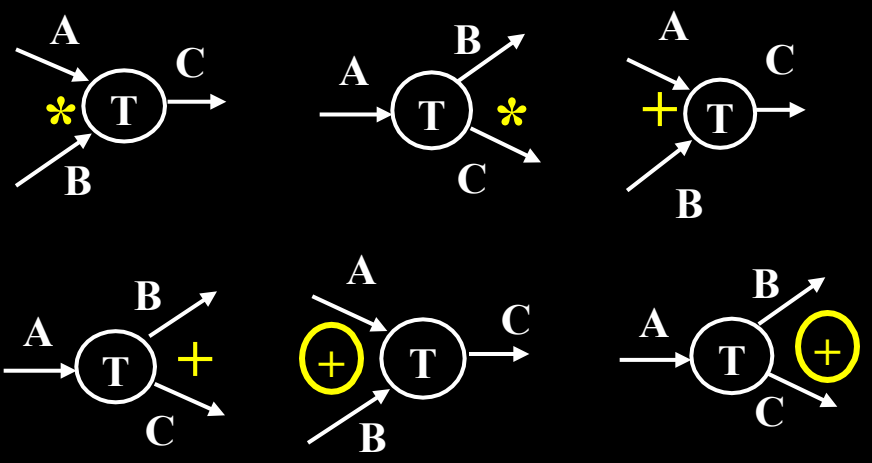
数据流 数据流名 箭头

处　理 加工名 圆或椭圆

数据存放 文件名 单或双杠

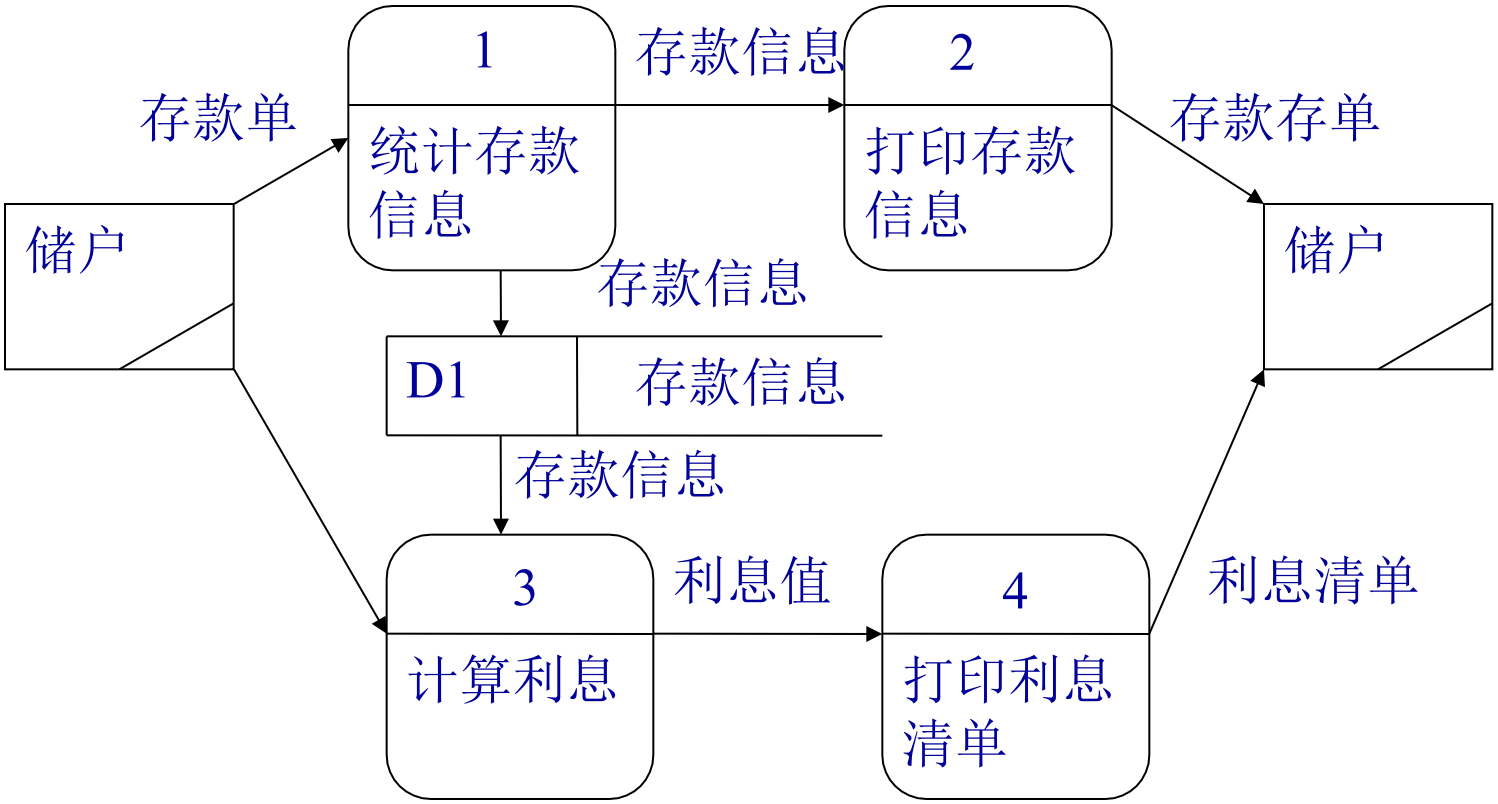
数据源点或终点 实体名 矩形框

还有一些辅助图例：

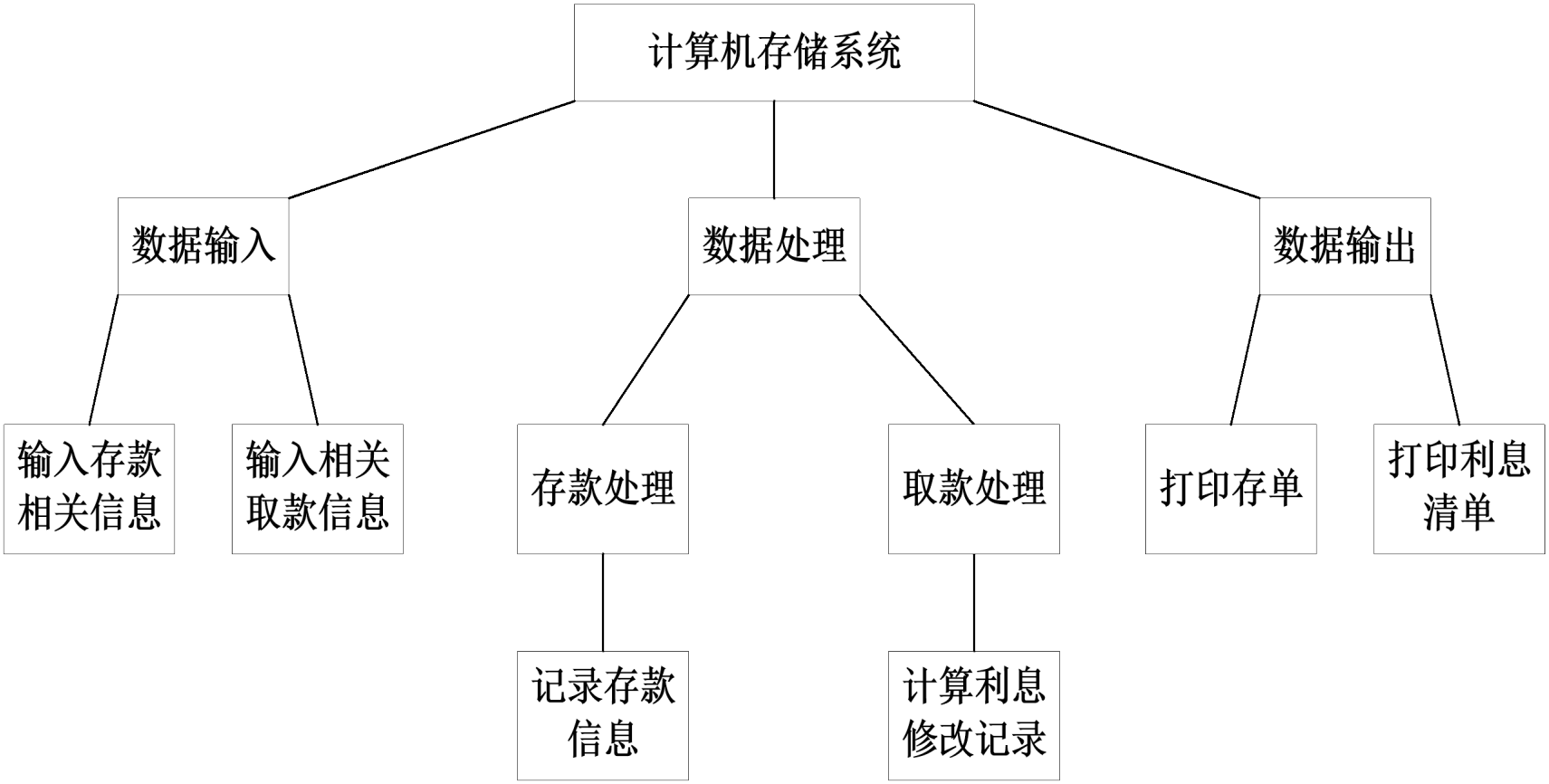


* 与　　+ 或　　⊕ 互斥

举例：


1
统计存款信息
2
打印存款信息
存款单
存款信息
存款存单
储户
储户
存款信息
D1
存款信息
存款信息
3
计算利息
利息值
4
打印利息清单
利息清单

细化后计算机储蓄系统软件结构



第3章　需求分析

1、需求分析基本任务是什么？P46-P48

2、分析建模

2.1什么是模型？ 模型：就是为了了解事物而对事物做出一个抽象，是对事物一个无歧义书面描述。通常，模型由一组图形符号和组织这些符号规则组成。

2.2数据模型（E-R）、 功效模型（2.4节数据流图）、 行为模型（状态转换图是行为模型基础）

3、要从哪些方面验证软件需求？P60

第4章　形式化说明技术

软件工程使用方法

- 非形式化：用自然语言描述需求规格说明
- 半形式化：用数据流图或实体—联络图建立模型
- 形式化：描述系统性质基于数学技术

1、有穷状态机P67—P72

2、Petri网技术P72—P75

第5章　总体设计

1、在设计过程中，总体设计普通有哪两个主要阶段组成？P81

2、什么模块化？模块独立性包含哪些内容？度量准则是什么？P85—P89

3、启发规则有哪些？P90—P92

4、描绘软件结构图形工具P92-P94

5、面向数据流设计方法P95—P102

5.2.1 模块化

- 模块是程序对象有名字集合。比如，过程、函数、子程序、宏等，是组成软件系统结构基本元素。
- 模块化就是将系统划分为若干个模块，每个模块完成一个子功效。模块化目标是将系统“分而治之”，模块化能够降低问题复杂性，使软件结构清楚，易阅读、易了解，易于测试和调试，因而也有利于提升软件可靠性。

5.2.5 模块独立

“模块”，又称“构件”，普通指用一个名字可调用一段程序。它普通含有以下三个基本属性：

(1)功效 即指该模块实现什么功效，做什么事情。必须注意：模块功效，应是该模块本身功效加上它所调用全部子模块功效。

(2)逻辑 即描述模块内部怎么做。

(3)状态 即该模块使用时环境和条件。

所谓模块独立性，是指软件系统中每个模块只包括软件要求详细子功效，而和软件系统中其它模块接口是简单。即功效专一，模块之间无过多相互作用模块。

这种类型模块能够并行开发，模块独立性越强，开发越轻易。独立性强模块，还能降低错误影响，使模块轻易组合、修改及测试。

模块独立性度量标准是两个定性准则：

耦合性　用于描述模块之间联络紧密程度。

内聚性　用于描述模块内部联络紧密程度。

模块独立性比较强模块应该是含有高内聚性和低耦合度。

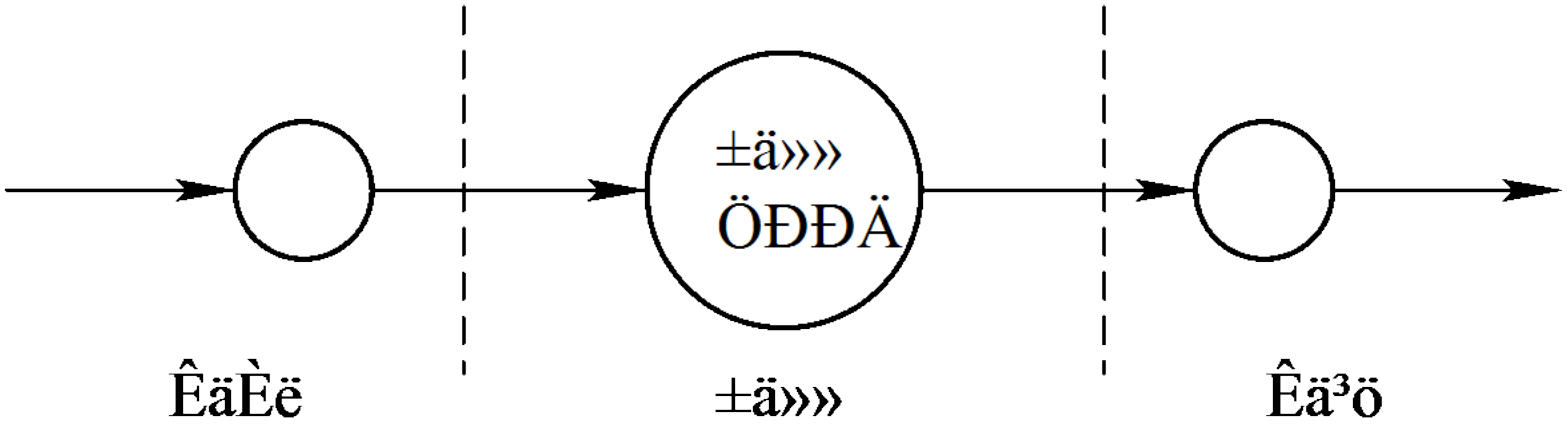


图5.8 变换型数据流图基本模型

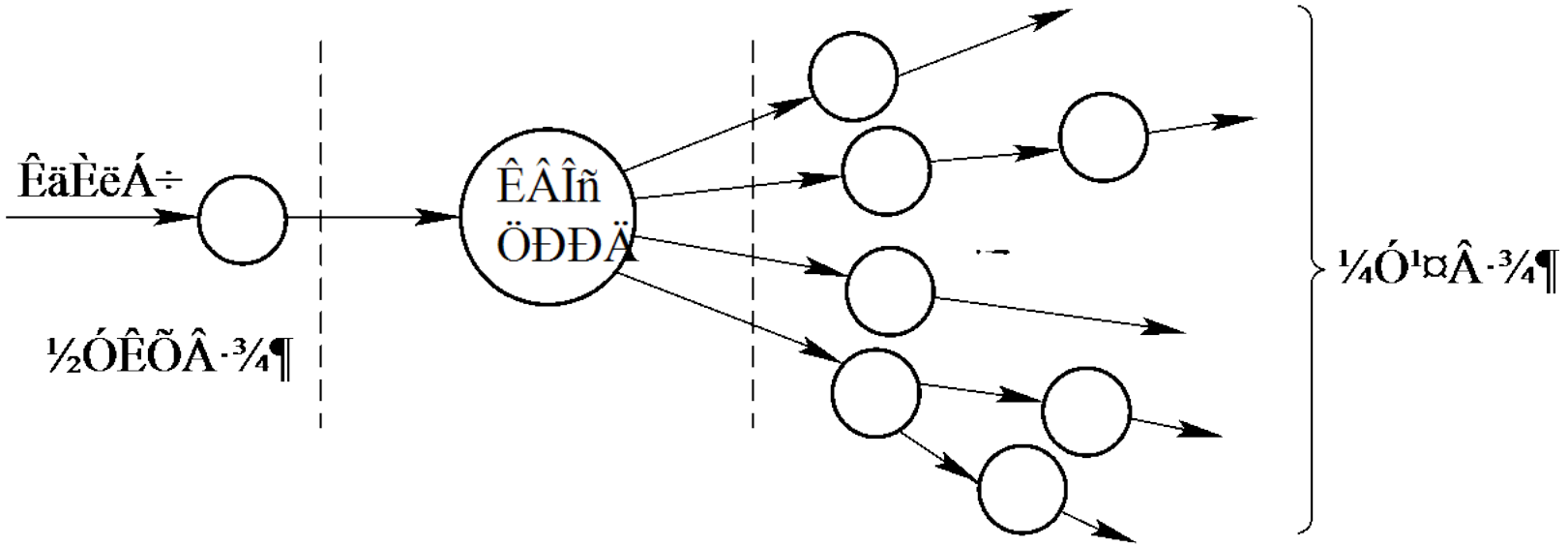


图　事务型数据流图基本模型

- 变换分析设计：把含有变换流特点数据流图按预先确定模式映射成软件结构。不含有显著事务特点。

- 即使在任何情况下都能够使用变换分析方法设计软件结构，不过在数据流含有显著事务特点时，也就是有一个显著“发射中心”(事务中心)时，还是以采取事务分析方法为宜。

- 二者主要差异仅在于由数据流图到软件结构映射方法不一样。

第6章　详细设计

1、结构程序设计中有哪几个基本控制结构？P108

2、了解人机界面设计。

3、掌握过程设计工具（程序流程图、盒图、PAD图，判定树）P114—P119

4、面向数据结构设计方法（Jackson图）

5、程序复杂度定量度量（McCabe）

6.1 结构程序设计

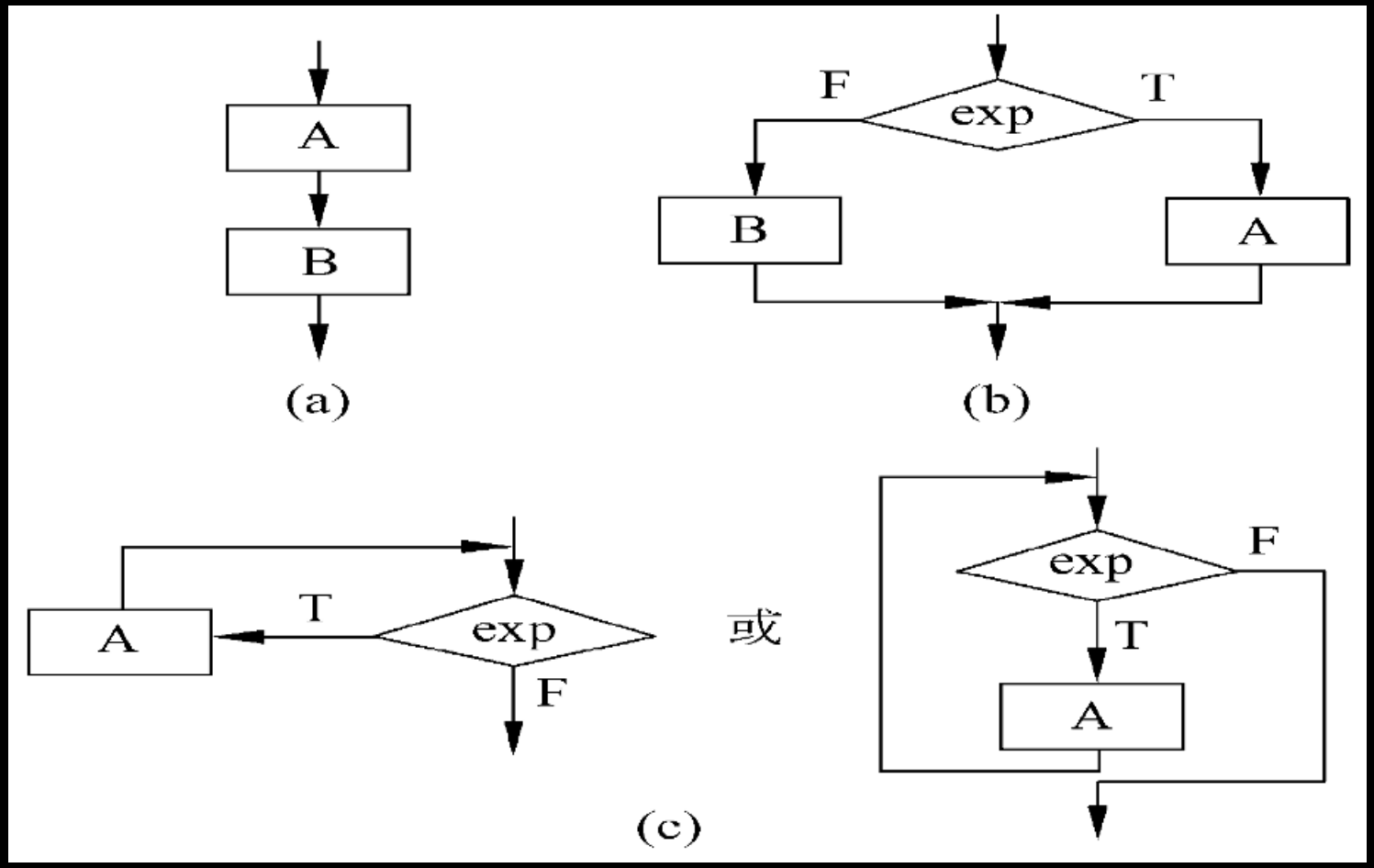


图6.1 3种基本控制结构

其它惯用控制结构

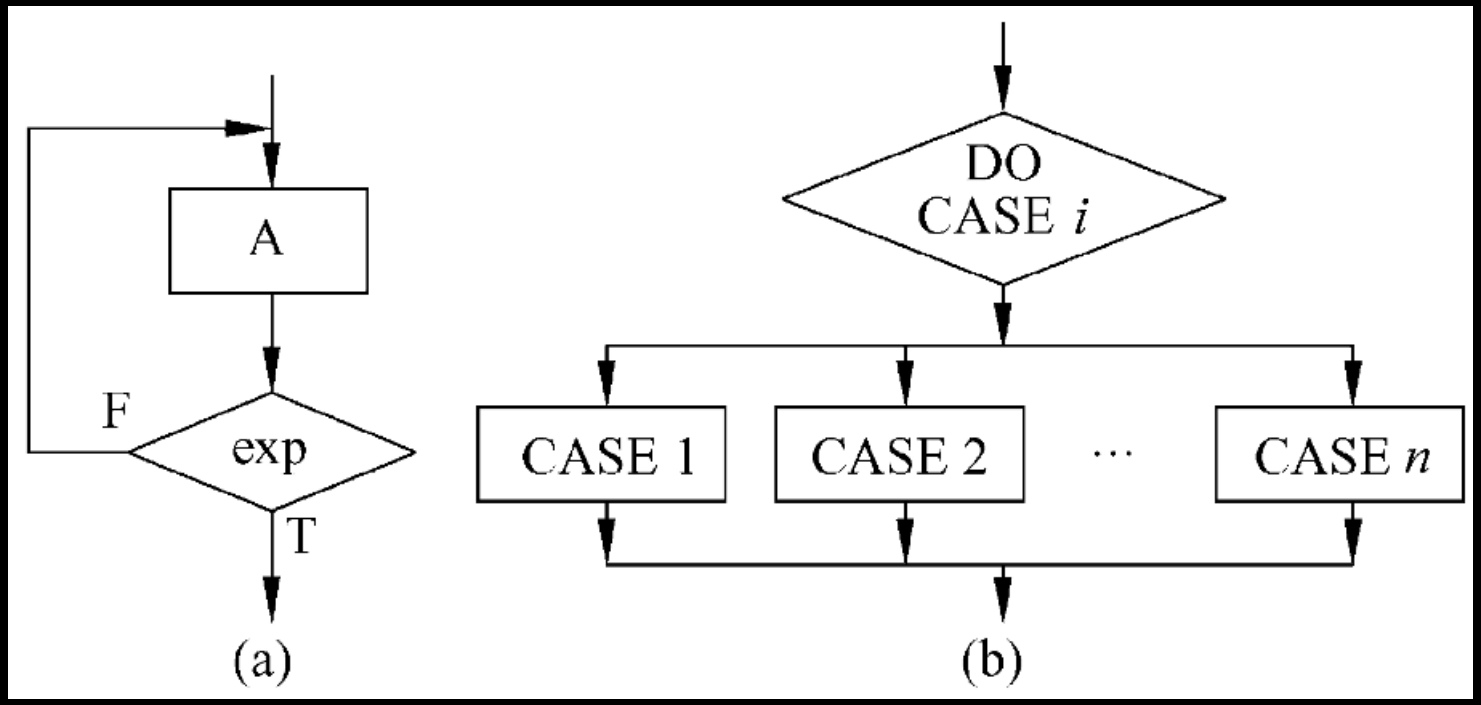


图6.2　其它惯用控制结构

程序流程图是最早出现且使用较为广泛算法表示工具之一，能够有效地描述问题求解过程中程序逻辑结构。程序流程图中经常使用基本符号如图6.3所表示。

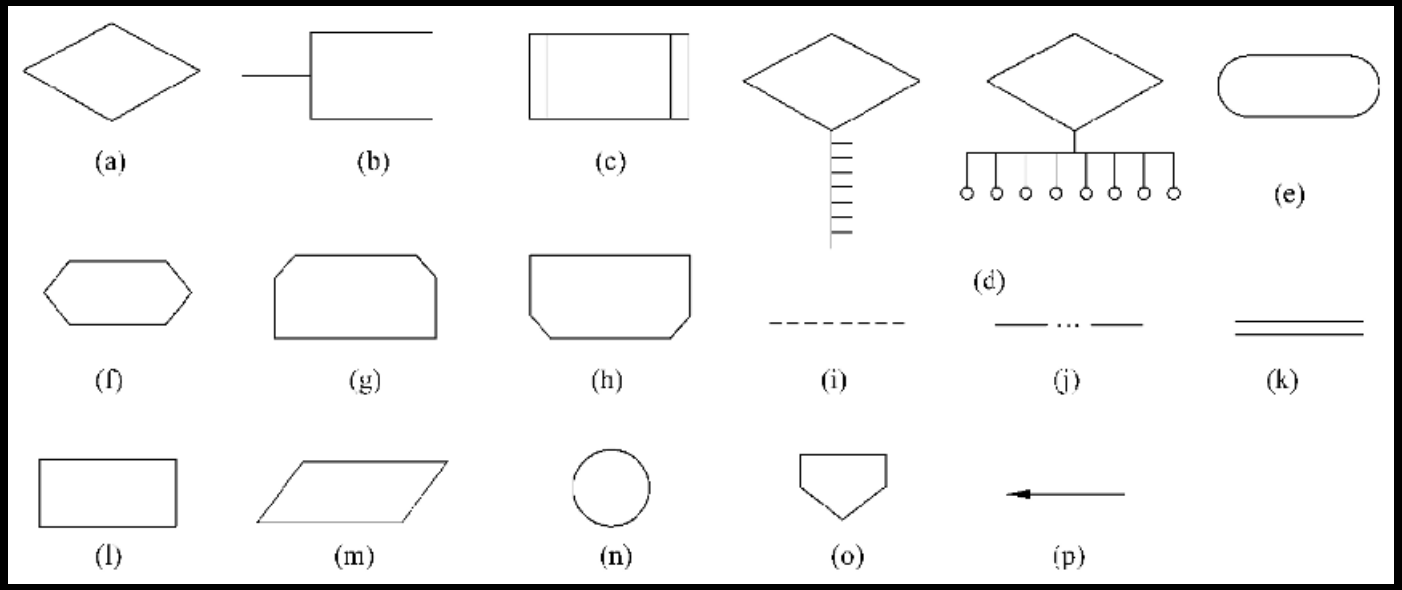


图6.3 程序流程图中使用符号

◆N—S图又称为盒图，它是为了确保结构化程序设计而由Nassi和Shneiderman共同提出一个图形工具。

◆在N—S图中，全部程序结构均使用矩形框表示，它能够清楚地表示结构中嵌套及模块层次关系。

◆N—S图中，基本控制结构表示符号如图6.4所表示。

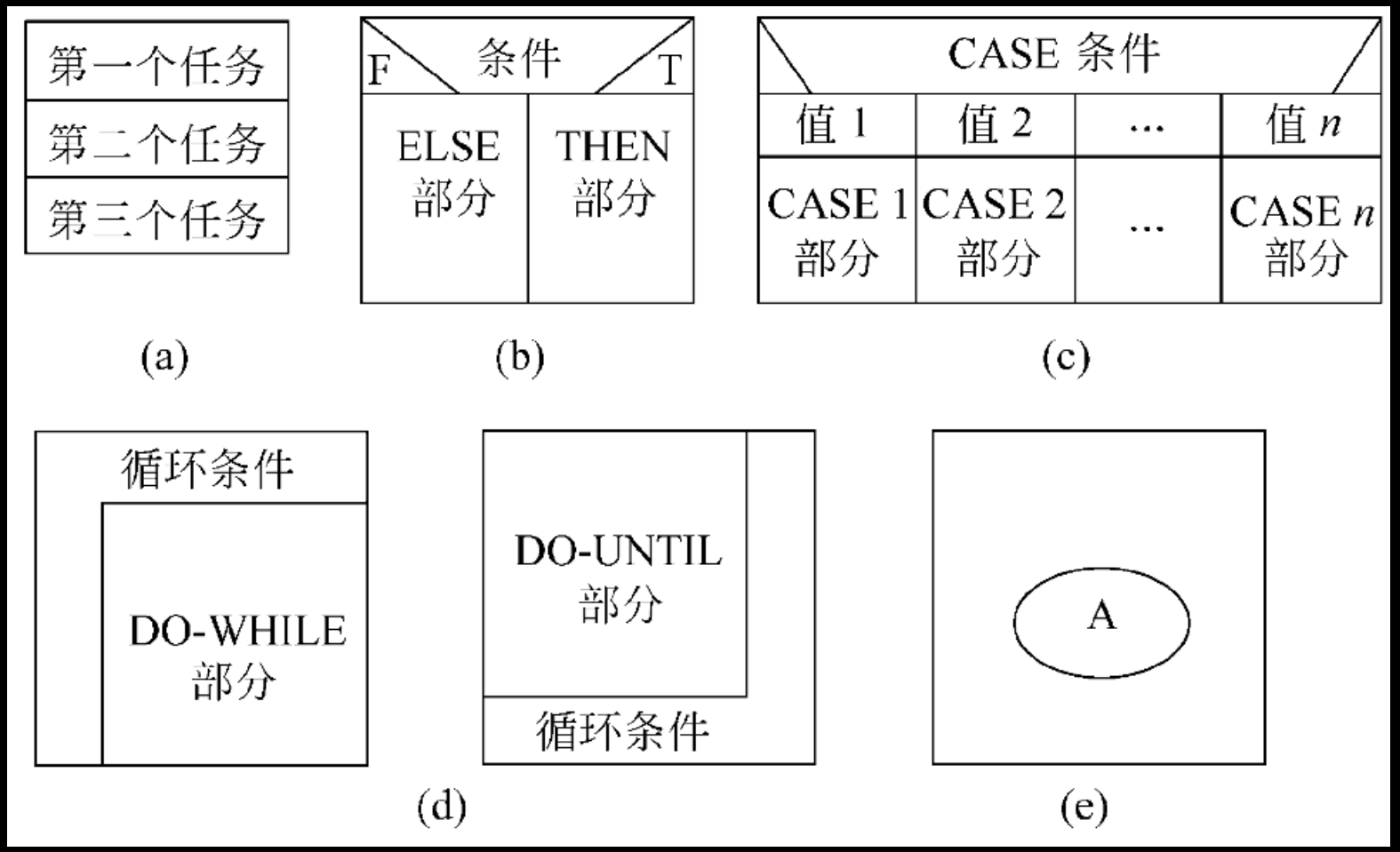

第一个任务
第二个任务
第三个任务
F
条件
T
ELSE
部分
THEN
部分
CASE 条件
值 1
值 2
…
值 n
CASE 1
部分
CASE 2
部分
…
CASE n
部分
(a)
(b)
(c)
循环条件
DO-WHILE
部分
DO-UNTIL
部分
循环条件
A
(d)
(e)

图6.4 盒图基本符号

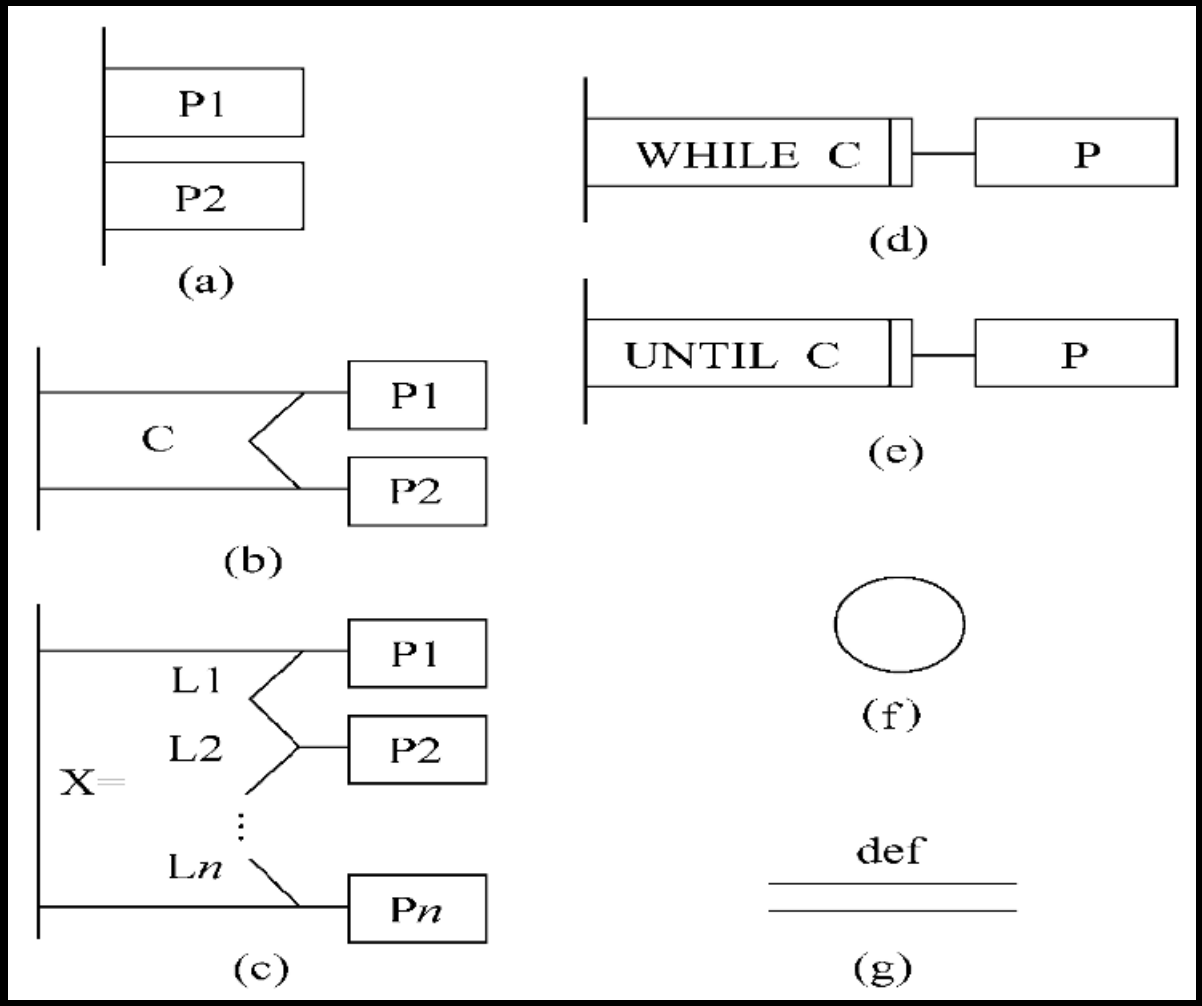


图6.5 PAD中基本符号

(a)次序结构；(b) 分支结构；(c) 多分支CASE结构；(d) 当型循环；(e)；直到型循环；(f) 语句标号；(g) 定义

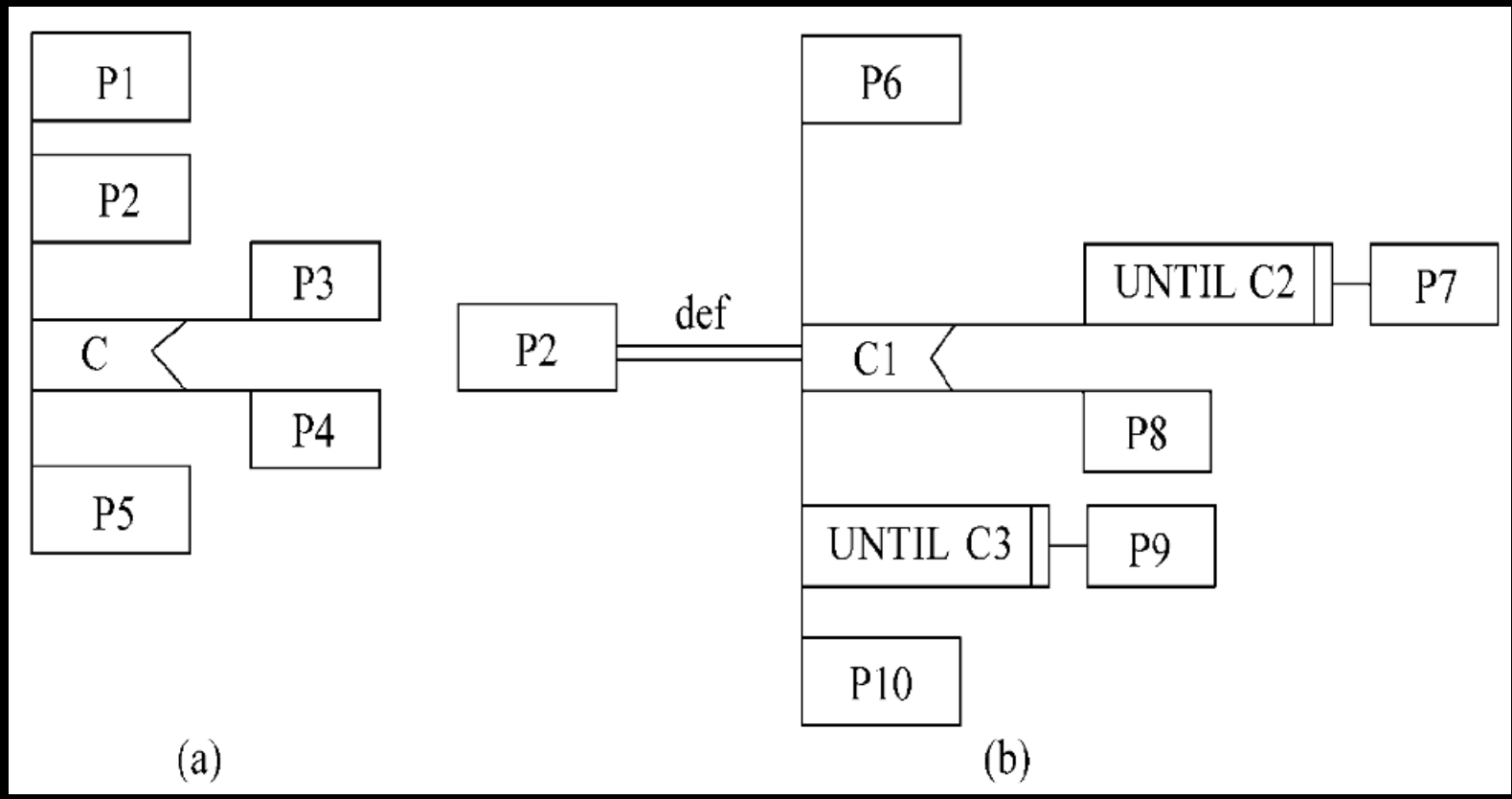


图6.6 使用PAD图提供定义功效来逐步求精例子

6.4 面向数据结构设计方法

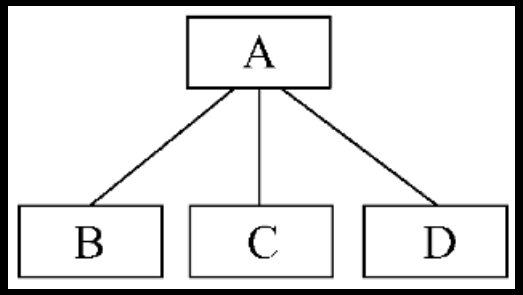
Jackson方法和Warnier方法是最著名两个面向数据结构设计方法

6.4.1 Jackson图

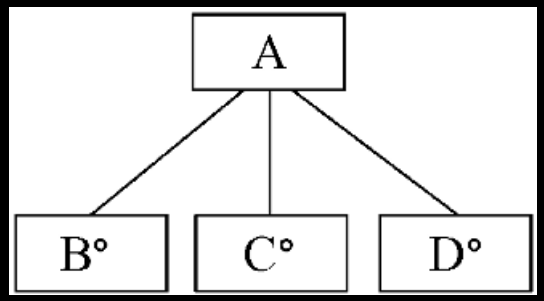
◆Jackson方法是由英国M.A.Jackson在1975年首先提出，他同时还提出了与这种方法配套使用、用于描述系统数据结构和程序结构图形工具，被称为Jackson图。

◆Jackson方法从目标系统输入、输出数据结构入手，导出程序框架结构，再补充其它细节，就可得到完整程序结构图。

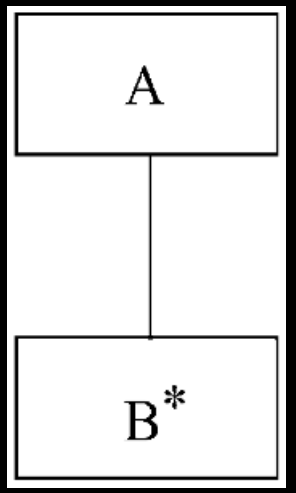
◆这一方法对输入、输出数据结构明确中、小型系统尤其有效，如商业应用中文件、表格处理。该方法也可与其它方法结合，用于模块详细设计。



（a）



（b）



（c）

图6.8 三种基本结构在Jackson图中表示符号
（a）次序结构；（b）选择结构；（c）；循环结构

6.4.2 改进 Jackson图

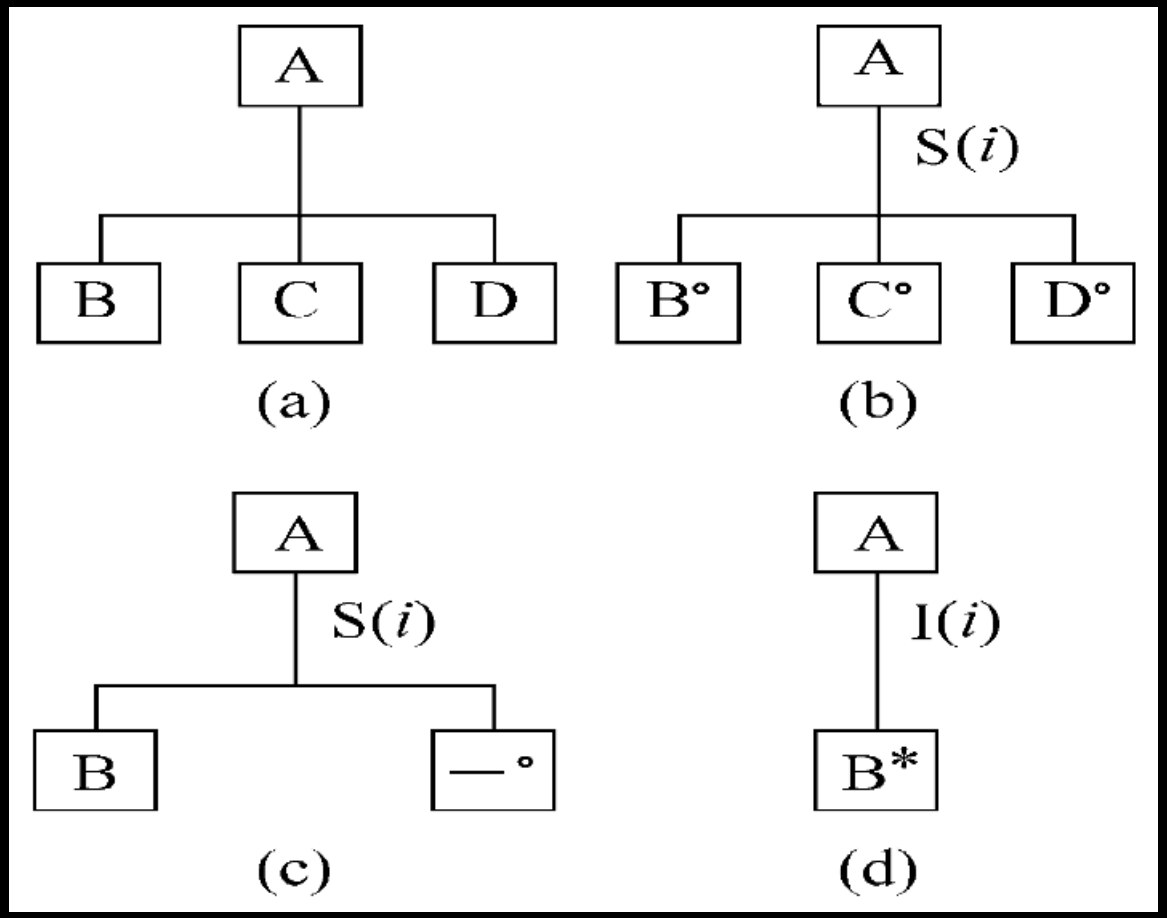


图6.9 改进Jackson图

定量度量程序复杂程度方法很有价值：把程序复杂程度乘以适当常数即可估算出软件中错误数量以及软件开发需要用工作量，定量度量结果能够用来比较两个不一样设计或两个不一样算法优劣；程序定量复杂程度能够作为模块规模准确程度。

6.5.1 McCabe方法

- McCabe方法依据程序控制流复杂程度定量度量程序复杂程度，这么度量出结果称为程序环形复杂度。
- 为了突出表示程序控制流，人们通常使用流图(也称为程序图)。所谓流图实质上是“退化了”程序流程图，它仅仅描绘程序控制流程，完全不表现对数据详细操作以及分支或循环详细条件。

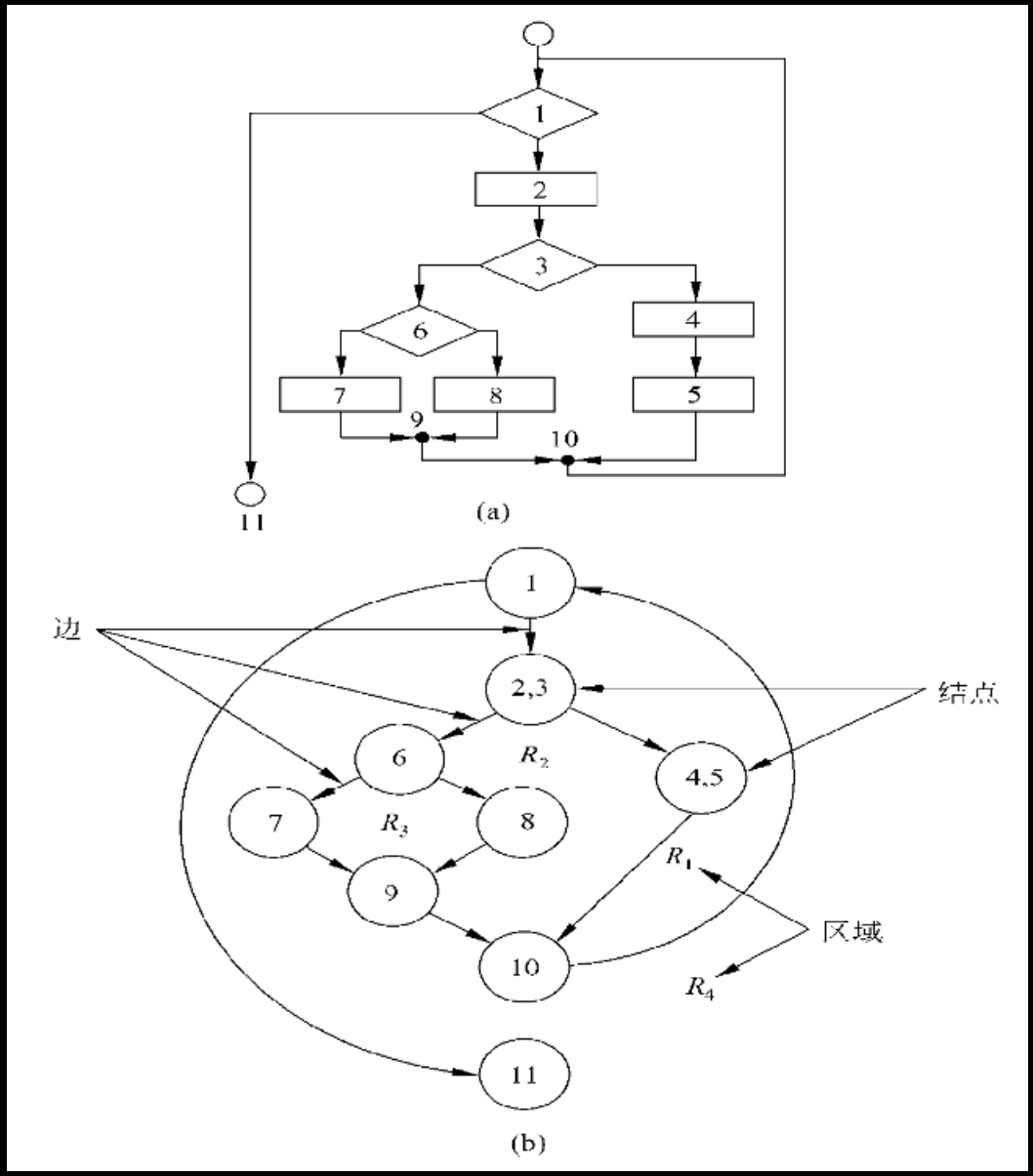


图6.16 把程序流程图影射成流图

环形复杂度定量度量程序逻辑复杂度。有了描绘程序控制流流图之后，能够用下述3种方法中任何一个来计算环形复杂度。

(1) 流图中区域数等于环形复杂度。

(2) 流图G环形复杂度V(G)=E-N+2,其中，E是流图中边条数，N是结点数。

(3) 流图G环形复杂度V(G)=P+1，其中，P是流图中判定结点数目。

第7章　　实现

1、编码

2、测试技术P139—P169

3、调试路径有哪些？P168

4、软件可靠性和可用性基本概念P169

编码和测试

- 编码：把软件设计结果翻译成用某种程序语言书写程序。
- 测试：软件测试目标是在软件投入生产性运行前，尽可能多发觉软件中错误。

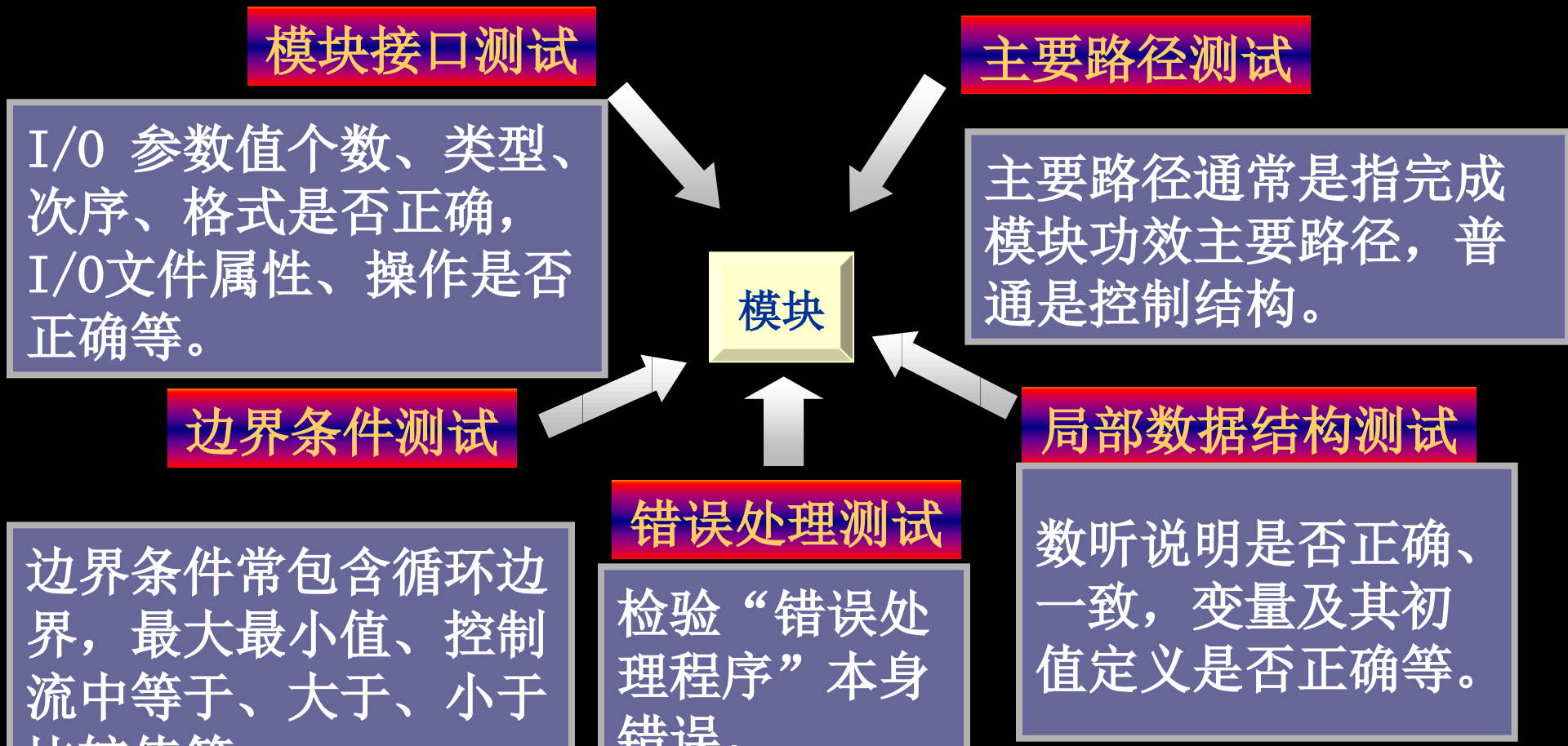
7.2 软件测试基础

- 软件测试：为发觉程序中错误而执行程序过程。
- 软件测试准则（尽早和不停测试、彻底测试不可能、软件测试是有风险行为、并非全部软件错误都能恢复、反向思维逻辑、由小到大测试范围、防止检验自己代码、追溯至用户需求）
- 测试方法（黑盒测试和白盒测试）
- 测试步骤（模块测试、子系统测试、系统测试、验收测试、平行运行）

7.3 单元测试

也称模块测试（module testing ）

- **测试内容**



7.6 白盒测试技术

- 白盒法又称为逻辑覆盖法，其测试用例选择，是按照不一样覆盖标准确定。

白盒法惯用覆盖标准

① 语句覆盖： 选择足够测试用例，使得程序中每个语句最少都能被执行一次。

② 判定覆盖： 执行足够测试用例，使得程序中每个判定最少都取得一次“真”值和“假”值。

③ 条件覆盖：执行足够测试用例，使得判定中每个条件取得各种可能结果。

④ 判定/条件覆盖： 执行足够测试用例，使得判定中每个条件取到各种可能值，并使每个判定取到各种可能结果。

⑤ 条件组合覆盖： 执行足够例子，使得每个判定中条件各种可能组合都最少出现一次。