

《软件工程》教案

(本 科)

主讲：穆海伦

杭州电子科技大学 计算机软件教研室

E-mail: helen_se@163.com

2015 年 9 月

目录

第一章	软件工程概述	5
§1.1	软件的概念、特点及分类	5
§1.2	软件危机	6
§1.3	软件工程	8
§1.4	小结	13
第二章	系统分析	14
§2.1	系统分析（项目计划）	14
§2.2	问题定义	14
§2.3	可行性研究	15
§2.4	小结	20
§2.5	补充实例	20
第三章	需求分析	23
§3.1	需求分析概述	23
§3.2	结构化分析方法	25
§3.3	验证软件需求	29
§3.4	小结	29
§3.5	补充知识	29
§3.6	补充实例	32
第四章	软件设计概述	36
§4.1	软件设计概述	36
§4.2	软件设计的策略	36
§4.3	概要设计	41
§4.4	结构化系统设计	43
§4.5	小结	53
§4.6	补充实例	53

第五章	详细设计	54
§5.1	详细设计概述	54
§5.2	结构化程序设计	55
§5.3	详细设计的描述工具	56
§5.4	其它的软件设计方法（面向数据结构的设计方法）	61
§5.5	程序复杂度的定量度量	62
§5.6	小结	64
§5.7	补充实例	64
第六章	编码	68
§6.1	编码的目的	68
§6.2	编码的风格	68
§6.3	程序设计语言	70
§6.4	小结	71
§6.5	补充实例	71
第七章	测试	72
§7.1	基本概念	72
§7.2	测试用例的设计	74
§7.3	单元测试	83
§7.4	综合测试	83
§7.5	高级测试	85
§7.6	纠错（调试）	86
§7.7	软件可靠性	87
§7.8	测试文档：《测试分析报告》	87
§7.9	小结	87
§7.10	补充实例	87
第八章	维护	88
§8.1	软件维护	88

§8.2	可维护性	89
§8.3	小结	90
§8.4	补充实例	90
第九章	面向对象分析与设计	91
§9.1	引论	91
§9.2	基本概念	93
§9.3	面向对象建模	95
§9.4	面向对象的分析	98
§9.5	面向对象的设计	99
§9.6	面向对象的实现	102
§9.7	UML 统一建模语言	102
§9.8	小结	112
§9.9	补充实例	112
第十章	软件质量保证	114
§10.1	什么是 <i>SQA</i>	114
§10.2	<i>SQA</i> 活动	114
§10.3	<i>SQA</i> 活动的影响因素	114
§10.4	<i>SQA</i> 策略	115
第十一章	软件项目计划与管理	116

第一章 软件工程概述

引论：

- ◇ 随着计算机的普及与深化，软件数量急剧膨胀，同时软件成本也在逐年上升，质量得不到可靠的保证。软件开发的生产率也远远跟不上普及计算机应用的要求。由此产生了“软件危机”。软件工程正是在此情况下产生的一门新兴学科。
- ◇ 学习软件工程，锻炼思维能力及解决问题的能力。
- ◇ 学习软件工程，努力成为软件界的“白领”。

§ 1.1 软件的概念、特点及分类

一. 软件的定义：

软件是计算机系统中与硬件相互依存的另一部分，它包括程序、数据及相关文档的完整集合。其中，程序是按事先设计的功能和性能要求执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发、维护和使用有关的图文材料。

二. 软件的特点：

1. 软件是一种逻辑实体，而不是具体的物理实体。
2. 软件的生产与硬件不同。（无明显的制造过程，存在软件产品的保护问题。）
3. 在软件的运行和使用期间，没有硬件那样的机械磨损、老化等问题。
4. 软件的开发和运行常常受到计算机系统的限制，对计算机系统有着不同程度的依赖性。
5. 软件的开发至今尚未完全摆脱手工艺的开发方式。
6. 软件是复杂的。（软件复杂性来源于它所反映的实际问题的复杂性。）
7. 软件成本相当昂贵。（定制产品、手工开发，成本高）
8. 相当多的软件工作涉及到社会问题。

三. 软件分类：

1. 按软件功能划分：

- 1) 系统软件：使计算机系统各个部件、相关软件和数据协调、高效的工作的软件。（如：操作系统，数据库管理系统，设备驱动程序等）
 - 2) 支撑软件：协助用户开发软件的工具性软件。（如：文本编辑程序，集成开发工具，图形软件包等）
 - 3) 应用软件：在特定领域内开发为特定目的服务的一类软件。
2. 按软件规模划分：

微型	1 人	1—4 周	0.5K
小型	1 人	1—6 月	1—2K
中型	2—5 人	1—2 年	5—50K
大型	5—20 人	2—3 年	50—100K
甚大型	100—1000 人	4—5 年	1M
极大型	2000—5000 人	5—10 年	1M—10M
微型	1 人	1—4 周	0.5K

3. 按软件的工作方式划分：
- 1) 实时处理软件：在事件或数据产生时，立即予以处理，并及时反馈信号。
 - 2) 分时软件：允许每个联机用户同时使用计算机。
 - 3) 交互时软件：能实现人通信的软件。
 - 4) 批处理软件：把一组输入作业或一批数据以成批处理的方式一次运行，按顺序逐个处理完的软件。
4. 按软件服务对象的范围划分：
- 1) 项目软件
 - 2) 产品软件

§ 1.2 软件危机

一. 软件危机：

指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

1. 软件危机包含的问题：

- 1) 如何开发软件，以满足对软件日益增长的需求。（提高生产率）
- 2) 如何维护数量不断膨胀的已有软件
2. 软件危机的表现形式：
 - 1) 对软件开发的成本和进度的估计常常不准确。
导致：成本提高，工程延期，影响信誉。
权益之计：损害软件质量，又会引起用户不满。
 - 2) 用户对“以完成”的软件系统不满意的现象经常发生。
原因：对用户需求不确切，缺少沟通，仓促上阵，闭门造车。
导致：不符合用户要求。
 - 3) 软件产品质量往往靠不住。
原因：软件可靠性和质量保证未认真执行。
导致：软件质量问题。
 - 4) 软件常常是不可维护的。
原因：程序结构固定、死板、变更困难、错误、难以改正，无法增加新的功能和适应新的环境。
 - 5) 软件通常没有适当的文档资料。
项目负责人：用以控制整体状态，把握工程进度；
开发者：用以相互交流；
维护人员：维护的依据。
 - 6) 软件成本在计算机系统中成本所占比例率上升。
 - ◆ 微电子技术的进步和自动化程度的不断提高，导致硬件成本下降；
 - ◆ 软件需要手工劳动，且大规模和数量不断的扩大，导致软件成本上升。
 - 7) 软件开发生产率提高的速度，远远跟不上计算机普及、深入的趋势。

“供不应求”，无法充分利用硬件。

二. 软件危机产生的原因：

1. 与软件自身的特点有关：

逻辑实体、手工开发、复杂度高、成本昂贵。

2. 与开发、维护方法不正确有关：

忽视用户需求，轻视软件维护。

三. 解决软件危机的途径：

1. 技术措施：方法和工具

2. 组织管理措施：从管理角度进行审查、控制。

软件工程正是从技术和管理两方面研究如何更好地开发和维护计算机软件的一门新兴学科。

§ 1.3 软件工程

一. 软件工程：

是采用工程的概念、原理、技术和方法来指导软件开发和维护的工程学科。

1. 软件工程的基本原理：（七条）

是确保软件产品质量和开发效率的原理的最小的完备的集合。

1) 用分阶段的生命周期计划严格管理。

2) 坚持进行阶段评审。

进行相应的质量保证、尽早发现错误。

3) 实行严格的产品控制。

实行基准配置（给过阶段评审后的软件配置成分，包括文档、程序等）管理，涉及对基准配置的参数，必须按严格规程审批。

4) 采用现代的程序设计技术。

如：结构化分析与设计、面向对象的分析与设计。

5) 结果应能清楚地审查。

规定开发组织的责任和产品质量标准，提高软件开发过程的可见性。

6) 开发小组的人员应该少而精。

开发小组人员的素质和数量是影响产品质量和开发效率的重要因素。

7) 承认不断改进软件工程实践的必要性。

积极采纳新技术，不断总结经验。

2. 软件工程的三要素：方法、工具和过程。

1) 方法：“如何做”，常采用某种特殊的语言或图形的表达方法及一套质量保证标准。

2) 工具：为方法提供的软件支撑环境。（计算机辅助软件工程 CASE）

3) 过程：将方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。

3. 软件工程项目的基本目标：

1) 付出较低的开发成本。

2) 达到要求的软件功能。

3) 取得较好的软件性能。

4) 开发的软件易于移植。

5) 需要较低的维护费用。

6) 能按时完成开发工作，及时交付使用。

4. 软件工程的原则：

1) 抽象

2) 信息隐藏

3) 模块化

4) 局部化

5) 一致性

6) 完全性

7) 可验证性

二. 软件工程的传统途径：

1. 软件工程的传统途径：生命周期方法学

从时间角度对软件开发和维护的复杂问题进行分解，划分为若干个阶段，每个阶段有相对独立的任务，是在阶段结束时进行技术审查和管理复审，最后产生相应的文档资料。

2. 软件生命周期的划分：

1) 三个时期:

- **软件定义:** 确定工程总目标: 可行性、采用的策略, 需求完成的功能, 需要的资源和成本, 工程进度表。
包括: 问题定义, 可行性研究, 需求分析。
- **软件开发:** 具体设计和实现。
包括: 概要设计、详细设计(系统设计), 编码和单元测试、综合测试(系统实现)
- **软件维护:** 使软件持久地满足用户需要。
改正错误, 适应新环境, 满足新需求。

2) 八个阶段:

- **问题定义:** “要解决的问题是什么?”
提出关于问题性质、工程目标和规模的全面报告。
- **可行性研究:** “对上一个阶段所确定的问题有行的通解决办法吗?”
研究问题的范围, 进行成本/效率分析, 探索问题是否值得解和如何解。
- **需求分析:** “为了解决问题, 目标系统必须做到什么?”
确定目标系统所应具备的功能, 建立系统逻辑模型(数据流图、数据字典、简要算法)
- **概要设计:** 概括地谈, 应该如何解决问题
提出几种设计方案: 低成本, 中等成本, 高成本(“十全十美”), 确定解决系统的方案和目标系统需要那些程序, 设计软件的结构, 确定程序模块及模块间关系(层次图或结构图)。
- **详细设计:** 应该怎样具体地实现系统
把解决具体化, 设计出程序的详细规格说明(HIPO图或PDL语言)
- **编码和单元测试:** 编写程序模块的实现代码, 并对其进行测试。

- 综合测试：通过各种类型的测试使软件达到预定要求。
 - ◆ 集成测试：根据设计的软件结构，将单元模块按某种策略装配起来进行联合测试。
 - ◆ 验收测试：由用户根据需求规格说明书对目标系统进行整体验收。
- 软件维护：通过各种必要的维护活动使系统持久满足用户需要。
 - ◆ 改正性维护（21%）
 - ◆ 适应性维护（25%）
 - ◆ 完善性维护（50%）
 - ◆ 预防性维护（4%）

3) 目的和实质：

控制开发工作的复杂性，通过有限的确定步骤，把用户需求从抽象的逻辑概念转化为具体的物理实现。

3. 软件生存期模型：瀑布模型，演化模型，螺旋模型，喷泉模型，智能模型。

1) 瀑布模型：系统的生命周期方法学用瀑布模型来进行模拟。

- 各阶段间具有顺序性和依赖性
 - ◆ 前阶段结束→后阶段开始。
 - ◆ 前阶段输出文档→后阶段输入文档。
- 推迟实现的观点：设置系统分析与设计、推迟物理实现。
- 质量保证的观点：
 - ◆ 每个阶段必须完成规定的文档
 - ◆ 每个阶段结束前要对文档评审，以便尽早发现问题，改正错误。

2) 演化模型：（原型模型）

能够克服瀑布模型的缺点、适当的减少由于软件需求不明确而给开发工作带来的风险。

3) 螺旋模型：

将瀑布模型与演化模型结合起来，并且加入两种模型都忽略了的风险分析，以弥补两者的不足。

螺旋模型沿着螺旋线旋转，在笛卡儿坐标的四个象限上分别表达四个方面的活动：

- ◆ 制定计划：确定软件目标，选定实施方案，弄清项目开发的限制条件。
- ◆ 风险分析：分析所选方案，考虑如何识别和取消风险。
- ◆ 实施工程：实施软件开发。
- ◆ 客户评估：评价开发工作，提出修正意见。

4) 喷泉模型：

- ◆ “喷泉”一词体现了迭代和无间隙特性。系统某个部分常常重复工作多次，相关功能在每次迭代中随之加入演进的系统，无间隙是指在开发活动，即分析、设计和编码之间不存在明显的边界。
- ◆ 支持软件复用，支持面向对象的开发方法。

5) 智能模型：基于知识的软件开发模型

智能模型综合了其他模型，并把专家系统结合在一起。该模型应用于基于规则的系统，采用规约和推理机制，帮助软件人员完成开发工作，并使维护在系统规格说明一级完成。

三. 技术审查和管理复审：

1. 技术审查：

保证软件质量，控制错误的积累和放大，以降低软件成本。

■ 技术审查的标准和方法：从前导和后续，两个阶段进行考虑。

- ◆ 前导：提出解法。
- ◆ 后续：实现解法。

■ 步骤：

- ◆ 准备
- ◆ 简要介绍情况
- ◆ 阅读被审查文档

- ◆ 开审查会
- ◆ 返工
- ◆ 复查

2. 管理复审：

对工程项目的成本、经费、投资回收前景，项目进度等经济因素，从管理角度进行审查。

§ 1.4 小结

第二章 系统分析

§ 2.1 系统分析（项目计划）

一. 两个阶段：

- ✧ 问题定义
- ✧ 可行性研究

二. 目标：

1. 识别用户要求
2. 评价系统的可行性
3. 进行经济分析和技术分析
4. 把功能分配给硬件、软件、人、数据库和其它系统元素
5. 建立成本和进度限制
6. 生成系统规格说明，形成所有后续工程的基础

§ 2.2 问题定义

一. 目的：

弄清用户需要计算机解决的问题根本所在，以及项目所需的经费和资源的文档。

二. 主要任务：

是在向用户调查的基础上，编写一个叫做《系统目标与范围说明书》的文档。这个说明经用户同意后，就作为下一步—可行性分析的依据。

三. 文档：《系统目标与范围说明书》

1. 项目名称
2. 问题说明：当前工作中存在的问题
3. 项目目标：用户对新系统的目标

4. 项目范围：指出解决这一项目所需的投资范围
5. 初步想法：对系统功能提出一些初步设想
6. 可行性研究计划：对可行性研究的时间、费用进行估算

§ 2.3 可行性研究

一. 可行性研究

1. 目的：

用最少的代价，在尽可能短的时间内弄清所定义的项目是不是可能实现和值得进行。（不是解决问题，而是确定问题是否可能解决和值得去解）

2. 实质：

是进行一次大大简化了的系统分析和设计的过程，即在较高层次上以较抽象的方式进行的系统分析和设计的过程。

3. 研究问题解法的可行性：

- 技术可行性：使用现有技术能实现这个系统吗？
- 经济可行性：这个系统的经济效益能超过它的开发成本吗？
- 操作可行性：系统的操作方式在这个用户组织内行得通吗？

4. 根本任务：对以后的行动方针提出建议

5. 步骤：

1) 复查系统规模和目标

改正含糊或不正确的叙述，清晰的描述目标系统的一切限制和约束，确保正在解决的问题，确实是要求解决的问题。

2) 研究目前正在使用的系统

了解现有系统的功能，阅读文档资料和使用手册，确定目标系统必须完成的基本功能，并解决现有系统中存在的问题。

3) 导出新系统的高层逻辑模型

设计过程：现有物理系统→现有系统逻辑模型→目标系统逻辑模型→新物理系统

4) 重新定义问题

重新复查问题定义，工程规模和目标

5) 导出和评价供选择的解法

技术可行性，经济可行性，操作可行性。

6) 推荐行动方针

是否值得开发，选择最好的解法，说明理由。

7) 草拟开发计划

开发计划：工程进度表，开发人员，各种资源，使用时间，系统
生命周期各阶段成本。

8) 书写文档并提交审查

二. 成本/效益分析：

通过估计开发成本，运行费用和经济效益，从而达到从经济角度分析开发一个特定的新系统是否划算，帮助使用部门负责人正确的做出是否投资这项工程开发的决定。

1. 成本估计：

■ 软件开发成本主要表现为人力消耗：

人力消耗×平均工资=开发费用

■ 成本估计技术：

◆ 代码行技术：源代码行数×每行代码平均成本=开发成本

◆ 任务分解技术：按开发阶段划分任务

（每个相对独立的开发任务的）成本累加和=开发成本

◆ 自动估计成本技术：软件工具。

2. 运行费用：

■ 系统操作费用（操作员人数，工作时间，消耗的物资等）

■ 维护费用。

3. 经济效益：

■ 因使用新系统增加的收入

■ 可以节省的运行费用

4. 度量效益的方法：

1) 货币的时间价值:

设年利率为 i ，现已存入 P 元，则 n 年后所得： $F=P*(1+i)^n$ ，即为 P 元钱在 n 年后的价值。反之，若 n 年后能收入 F 元，则其在现在的价值为： $P=F / (1+i)^n$ 。

2) 投资回收期:

是使累计的经济效益等于最初的投资所需要的时间，是衡量一个开发工程价值的经济指标。投资回收期越短，就能越快获得利润，所以工程就越值得投资。

3) 纯收入:

是在整个生存期之内系统的累计经济效益（折合成现在值）与投资之差。

4) 投资回收率:

设 P 为现在的投资的投资额， F_i 为第 i 年底的效益 ($i=1, 2, \dots, n$)， n 为系统的使用寿命， j 为投资回收率。

则 $(\dots((P(1+j)-F_1)(1+j)-F_2)(1+j)-\dots)-F_n=0$

即 $P=F_1 / (1+j) + F_2 / (1+j)^2 + \dots + F_n / (1+j)^n$ 。

三. 技术分析:

评价系统概念的技术价值，同时收集有关性能，可靠性，可维护性及生产率方面的信息。

1. 目的:

对系统的技术可行性进行评估，指明为完成系统的功能和性能需要什么技术？需要哪些新材料、方法、算法或者过程？有什么开发风险？这些技术问题对成本的影响如何？

2. 方法:

- 模型化方法（数学模型、物理模型）
- 优化技术
- 概率和统计
- 排队论
- 控制论等。

四. 系统结构的模型化：系统流程图

1. 系统流程图：

是用来描述系统物理模型的一种传统工具，基本思想是用图形符号、黑盒子形式描绘系统里面的每个部件（程序、文件、数据库、表格、人工过程等），它所表达的是信息在系统各部件之间的流动情况，而不是对信息进行加工处理的控制过程。

2. 描述符号：（书：P25）

1) 基本符号：（如表 2.1）


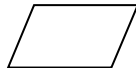


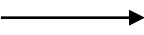



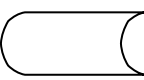
符号	名称	说明
	处理	能改变数据值或数据位置的加工或部件，例如：程序、处理机、人工加工等
	输入/输出	表示输入或输出（或既输入又输出），是一个广义的不指明具体设备的符号
	连接	指出转到图的另一部分或从图的另一部分转来，通常在同一页上
	换页连接	指出转到另一页图上或由另一页图转来
	数据流	用来连接其他符号，指明数据流动方向

表 2.1

2) 系统符号：（如表 2.2）

符号	名称	说明
	穿孔卡片	表示穿孔卡片输入或输出，也可表示一个穿孔卡片文件
	文档	通常表示打印输出，也可表示用打印终端输入数据
	磁带	磁带输入/输出，或表示一个磁带文件
	联机存储	表示任何种类的联机存储，包括磁盘、磁鼓、软盘和海量存储器件等




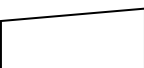



	磁盘	磁盘输入/输出, 也可表示存储在磁盘上的文件或数据库
	磁鼓	磁鼓输入/输出, 也可表示存储在磁鼓上的文件或数据库
	显示	CRT 终端或类似的显示部件, 可用于输入或输出, 也可既输入又输出
	人工输入	人工输入数据的脱机处理, 例如: 填写表格等
	人工操作	人工完成的处理, 例如: 会计在工资支票上签名
	辅助操作	使用设备进行的脱机操作
	通信链路	通过远程通信线路或链路传送数据

表 2.2

3. 实例: (书: P25-26)

五. 文档:

1. 《可行性分析报告》:

1) 系统概述:

- 当前现有系统分析: 系统描述及存在问题
- 目标系统分析: 系统功能和性能描述。(物理模型: 系统流程图)
- 当前系统与目标系统比较: 目标系统的优越性。

2) 可行性分析:

- 技术可行性
- 经济可行性
- 操作可行性。

3) 结论意见:

- 可着手组织开发
- 须待若干条件(如资源、人力、设备等)具备后才能开发
- 需对开发目标进行修改

- 不能进行或不必要进行（如技术不成熟、经济上不合算等）
- 其它…

2. 《项目开发计划》:

1) 系统概述:

包括项目目标, 主要功能, 系统特点, 以及关于开发工作的安排。

2) 系统资源:

包括开发和运行该软件系统所需要的各种资源。如: 硬件、软件、人员、组织、机构等。

3) 费用预算: 分阶段的人员费用, 机时费用及其它费用。

4) 进度安排: 各阶段起止时间, 完成文档及验证方式。

5) 要交付的产品清单

§ 2.4 小结

§ 2.5 补充实例

一. 库存清单系统:

1. 系统说明:

某装配厂有一座存放零件的仓库, 仓库中现有的各种零件的数量以及每种零件的库存量临界值等记录在库存清单主文件中。当仓库中零件数量有变化时, 应该及时修改库存清单主文件, 如果那种零件的库存量少于它的库存量临界值, 则应该报告给采购部门以便订货, 规定每天向采购部门送一次订货报告。

该装配厂使用一台小型计算机处理更新库存清单主文件和产生定货报告的任务。零件库存量的每一次变化称为一个事务, 由放在仓库中的 CRT 终端输入到计算机中; 系统中的库存清单程序对事务进行处理, 更新存储在磁盘上的库存清单主文件, 并且把必要的订货信息写在磁带上。最后, 每天由报告生成程序读一次磁带, 并且

打印出定货报告。

2. 系统流程图：（如图 2.5.1 所示）

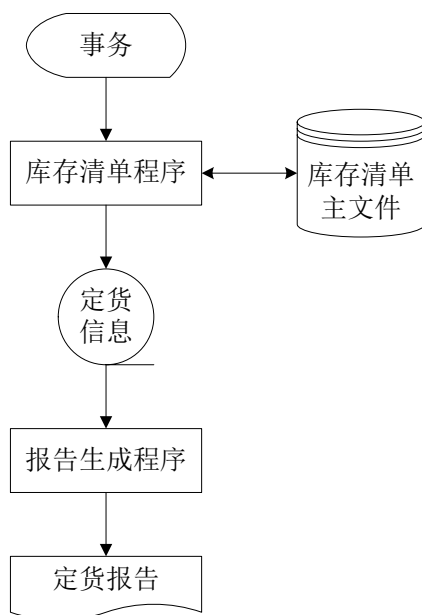


图2.5.1
库存清单系统的系统流程图

二. 教材购销系统：

1. 系统说明：

在教材的销售过程中，首先学生拿着购书申请到会计处审查并开具购书发票，然后到出纳处交款，并开具领书单，学生拿着领书单到书库领书；在开具购书发票的过程中，若教材存量不够，则需要进缺书统计，然后书库根据缺书情况去采购缺书，并通知学生补购教材。

2. 系统流程图：（如图 2.5.2 所示）

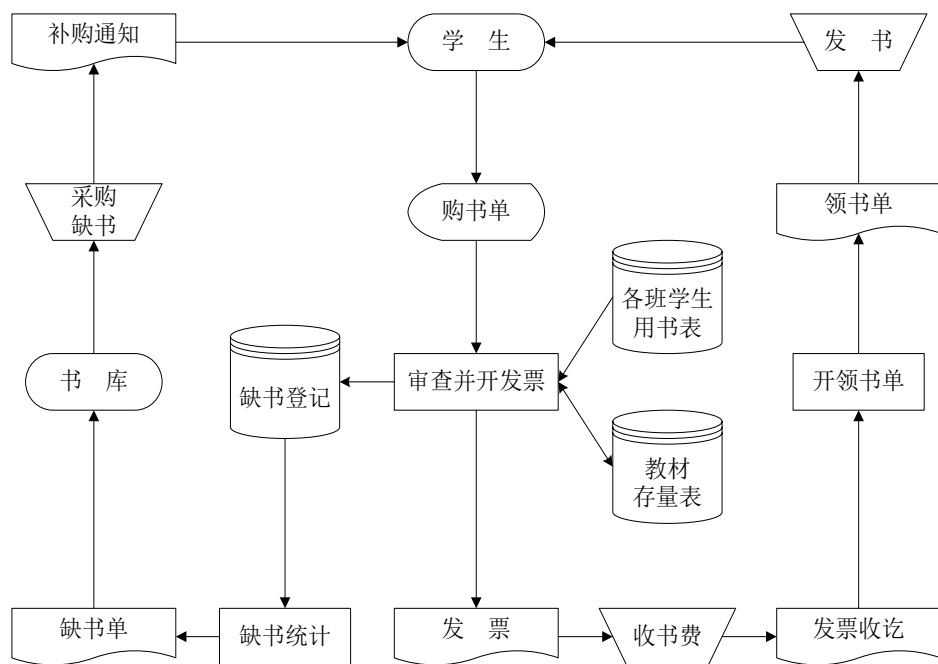


图2.5.2
教材购销系统的系统流程图

第三章 需求分析

§ 3.1 需求分析概述

一. 需求分析的任务：

1. 基本任务：回答“系统必须做什么”？确定目标系统功能和性能。
2. 具体任务：
 - 1) 确定对系统的综合要求：功能要求；性能要求；运行要求；将来可能提出的要求。
 - 2) 分析系统的数据要求：E-R 图（概念模型）。
 - 3) 导出系统的逻辑模型：数据流图，数据字典，加工处理说明书等。
 - 4) 修正系统开发计划。
 - 5) 开发原型系统：使用户对目标系统有一个更直接、更具体的概念，从而能更准确提出用户需求。（关键的困难在于成本）

二. 需求分析的过程：

1. 问题识别：确定软件的需求。

- 1) 功能
- 2) 性能
- 3) 环境
- 4) 可靠性
- 5) 安全保密
- 6) 界面
- 7) 资源
- 8) 成本进度
- 9) 目标

2. 分析与综合：

从数据流和数据结构出发，逐步细化软件功能，找出各元素之间的联系，接口特性和设计上的限制，给出目标系统的详细逻辑模型。

3. 编制需求分析文档：《需求规格说明书》

- 1) 任务概述：系统目标，运行环境，条件与限制
- 2) 数据描述：
 - 概念模型：E-R 图
 - 逻辑模型：数据流图
 - 数据定义：数据字典，加工说明
 - 数据库描述：名称和类型
- 3) 功能描述：软件功能要求
- 4) 性能描述：软件性能要求（处理速度、响应时间、安全限制等）。
- 5) 运行描述：用户界面、硬件接口、软件接口、故障处理等。
- 6) 质量保证：阐明软件在交付使用前需要进行的功能测试和性能测试，并且规定源程序和文档遵守的各种标准。

4. 技术审查和管理复审。

三. 需求分析的原则：

1. 必须能够表达和理解问题的数据域和功能域
 - 1) 数据域：数据流，数据内容和数据结构。
 - 2) 功能域：加工变换。
2. 必须按自顶向下，逐层分解的方式对问题进行分解和不断细化。
3. 要给出系统的逻辑视图和物理视图。
 - 1) 逻辑视图：给出软件要达到的功能和要处理的数据之间的关系。
 - 2) 物理视图：给出处理功能和数据结构的实际表示形式。

四. 需求分析的方法：

1. 需求分析方法：

是由对软件的数据域和功能域的系统分析过程及其表示方法组成。

包括：面向数据流，面向数据结构。
2. 不同的需求分析方法具有的共性：
 - 1) 支持数据域分析的机制：

所有方法都直接或间接地涉及到数据流，数据内容或数据结构等数据域的属性。

2) 功能表示的方法：

一般用数据变换或加工来表示。

3) 接口的定义：

是数据表示和功能表示的直接产物。（功能间的接口—数据流）

4) 问题分解的机制以及对抽象的支持：

在不同抽象层次上表示数据域和功能域，以逐层细化的手段建立分层结构。

5) 逻辑视图和物理视图：

6) 系统抽象模型：

是对现实世界中存在的有关实体和活动的抽象和精化。

§ 3.2 结构化分析方法

一. 结构化分析方法：

是面向数据流进行需求分析的方法，是用抽象模型的概念，按软件内部数据传递、变换的关系，自顶向下逐层分解，直到找到满足功能要求的所有可实现的软件为止。





二. 数据流图：

1. 数据流图（DFD）：

是软件系统逻辑模型的一种图形表示，是从数据传递和加工的角度，以图形的方式刻画数据流从输入到输出的移动变换过程的工具。

2. 组成符号：（书：P24）

1) 基本符号：（如表 3.1）

符号		说明
		数据的源点/终点
		变换数据的处理

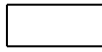


		数据存储
		数据流

表 3.1

2) 附加符号：(如表 3.2)

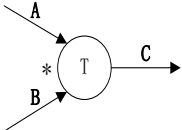
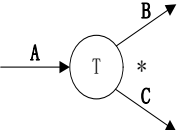
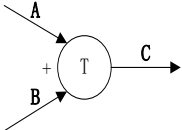
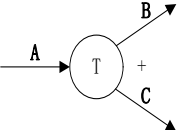
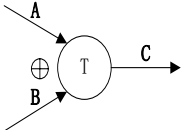
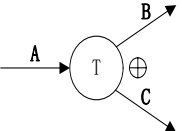
符号	说明
	数据 A 和数据 B 同时输入才能变换成数据 C
	数据 A 变换成 B 和 C
	数据 A 或 B，或 A 和 B 同时输入变换成 C
	数据 A 变换成 B 或 C，或 B 和 C
	只有数据 A 或只有数据 B（但不能 A、B 同时）输入时变换成 C
	数据 A 变换成 B 或 C，但不能变换成 B 和 C

表 3.2

3. 性质：

- 1) 数据流图中的箭头仅能表示在系统中流动的数据，而不是物质流
- 2) 数据流图与程序流程图不同，它不能表示程序的控制结构。(如：选择或循环)
- 3) 数据流图表现的范围具有很大的灵活性，可以画分层 DFD
4. 分层 DFD：由顶向下，逐层分解，逐步细化。

1) 优点：

- 便于实现：逐层细化，有利于控制问题的复杂度。
- 便于使用：使用户中的不同业务人员只选择与自身有关的图形，不必阅读全图。

2) 画分层 DFD 的指导原则：

- 第一层 DFD 应当是基本系统模型
- 注意父图和子图的平衡，维护信息的连续性
- 区分局部文件和局部外部项
- 掌握分解的速度，上快下慢
- 遵守加工编号原则

5. 举例：（书：P25—27）

三. 数据字典：

1. 数据字典：

是关于数据的信息的集合，是对 DFD 中的所有元素定义的集合。

2. 组成符号：（如表 3.3）

符号	含义	说明
=	被定义为	
+	与	例：x=a+b，表示 x 由 a 和 b 组成
[...，...]或[... ...]	或	例：x=[a, b]，x=[a b]，表示 x 由 a 或由 b 组成
{...}	重复	例：x={a}，表示 x 由 0 个或多个 a 组成
m{...}n	重复	例：x=3{a}8，表示 x 中至少出现 3 次 a，至多出现 8 次 a
(...)	可选	例：x=(a)，表示 a 可在 x 中出现，也可以不出现
“...”	基本数据元素	例：x=“a”，表示 x 为取值为 a 的数据元素
...	连接符	例：x=1..9，表示 x 可取 1 到 9 中的任一值

表 3.3

3. 内容：名称，别名，编号，分类，描述，定义，位置等

1) 数据流的描述：

数据流名：

说明：简要介绍作用，即它产生的原因和结果

来源：来自何方

去向：去向何处
组成：数据结构
备注：

2) 数据元素（数据项）的描述：

数据元素名：
类型：数值，文字，…
长度：
取值范围：
相关的数据元素及数据结构：
备注：

3) 数据存储（数据文件）的描述：

数据文件名：
简述：存放的是什么数据
组成：数据结构
存储方式：排列顺序，关键码等
备注：

4) 数据源（终）点描述：

名称：外部实体名
简要描述：什么外部实体
有关数据流：

四. 加工说明：

1. 加工说明：

是对 DFD 中的加工所做的描述，包括：输入数据、加工逻辑、输出数据等。

2. 内容：

- 加工名称
- 加工编号
- 输入数据流
- 输出数据流
- 加工逻辑
- 执行次数

3. 加工逻辑：

阐明把输入数据转换为输出数据的策略，是加工说明的主体，在需求分析阶段，仅需要指出要加工“做什么”。而不是“怎样去做”，描述方法：结构话语言，判定表，判定树。

1) 结构化语言（PDL）：

又称过程设计语言，伪码；它是一种介于自然语言与程序设计语言之间的语言，即具有结构化程序的清晰易读的优点，又具有自然语言的灵活性，不受程序设计语言那样严格的语法约束。

2) 判定表：

采用表格化的形式，适于表达含有复杂判断的加工逻辑。

实例：（书：P86）

3) 判定树：

是判定表的图形表示，其适用场合与判定表相同。

实例：（书：P87）

§ 3.3 验证软件需求

- ✧ 一致性：所有需求必须一致，不能互相矛盾。
- ✧ 完整性：需求必须完整，包含用户需要的所有功能和性能。
- ✧ 现实性：指定需求用现有的软、硬件技术基本上可以实现。
- ✧ 有效性：必须证明需求是正确有效的，确实能解决用户面对的问题。

§ 3.4 小结

§ 3.5 补充知识

✧ 概念模型

1. 数据模型的表示

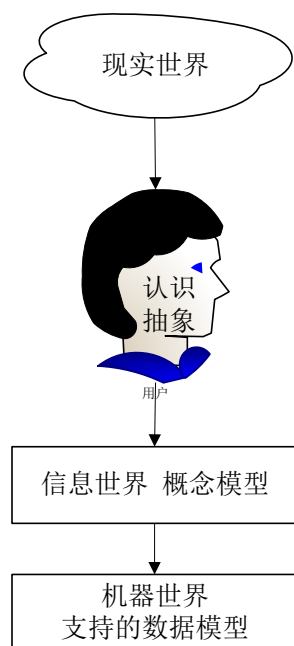


图3-5 数据模型的表示

2. 概念模型的表示方法：

实体—联系方法 (Entity-Relationship): E-R 图

3. E-R 图：

1) 主要概念：

- 实体：客观存在并相互区分的事物
- 属性：实体所具有的某一特性
- 联系：现实世界的事物之间的联系在信息世界的反映
 - ◆ 一对一联系：(1:1)
 - ◆ 一对多联系：(1:n)
 - ◆ 多对多联系：(m:n)

2) 符号表示：

- 用长方形表示实体型，在框内写上实体名。
- 用椭圆形表示实体的属性，并用无向边把实体与其属性连接起来。
- 用菱形表示实体间的联系，菱形框内写上联系名。用无向边把菱形分别与有关实体相连接，在无向边旁标上联系的类型。若实体之间的联系也具有属性，则把属性和菱形也

用无向边连接上。

3) 实体联系类型符号表示：

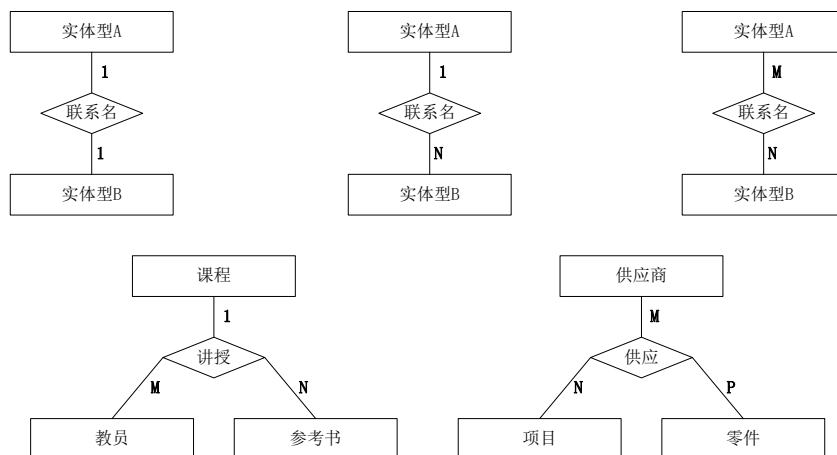


图3-6 实体联系的表示

4) 特点：

- 两个实体型间允许多种联系
- 多个实体型间可以有一个联系
- 一个实体型可以和自身联系
- E-R 图与具体的 DBMS 无关，是概念模型中最常用的一种

5) 举例：用 E-R 图表示某个工厂的物资管理的概念模型

■ 涉及的实体：

- ◆ 仓库：仓库号，仓库面积，电话号码
- ◆ 零件：零件号，名称，规格，单价，描述
- ◆ 供应商：供应商号，名称，地址，电话号码，帐号
- ◆ 项目：项目号，预算，开工日期
- ◆ 职工：职工号，姓名，年龄，职称

■ 实体间的联系：

- ◆ 一个仓库可以存放多种零件，一种零件可以存放在多个仓库中
- ◆ 一个仓库有多个职工当仓库保管员，一个职工只能在一个仓库工作
- ◆ 职工之间具有领导和被领导关系，即仓库主任领导若干保管员

■ E-R 图表示:

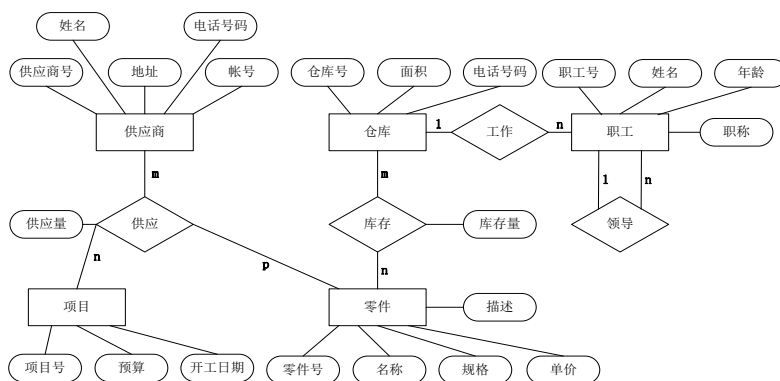


图3-7 某工厂物资管理E-R图

§ 3.6 补充实例

一. 库存清单系统:

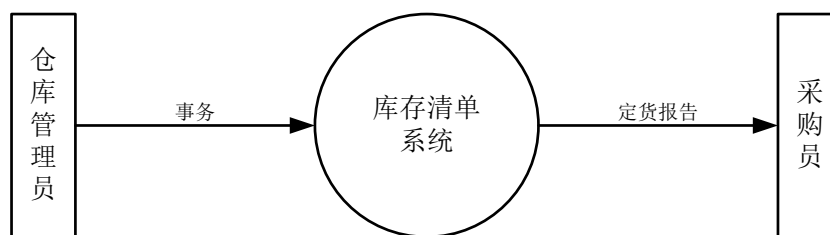
1. 系统说明:

某装配厂有一座存放零件的仓库，仓库中现有的各种零件的数量以及每种零件的库存量临界值等记录在库存清单主文件中。当仓库中零件数量有变化时，应该及时修改库存清单主文件，如果那种零件的库存量少于它的库存量临界值，则应该报告给采购部门以便订货，规定每天向采购部门送一次订货报告。

该装配厂使用一台小型计算机处理更新库存清单主文件和产生定货报告的任务。零件库存量的每一次变化称为一个事务，由放在仓库中的 CRT 终端输入到计算机中；系统中的库存清单程序对事务进行处理，更新存储在磁盘上的库存清单主文件，并且把必要的订货信息写在磁带上。最后，每天由报告生成程序读一次磁带，并且打印出定货报告。

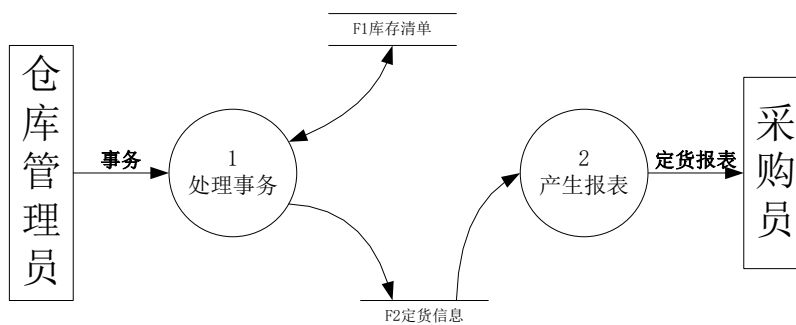
2. 数据流图:

1) 顶层数据流图



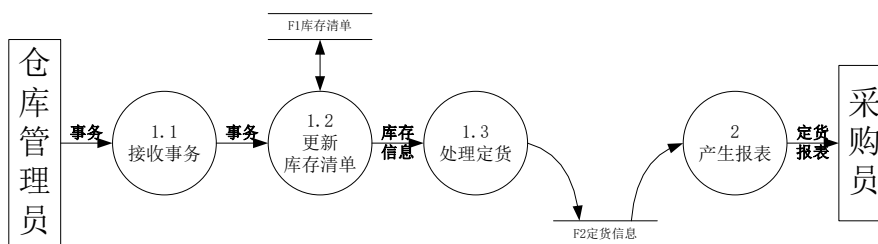
顶层数据流图-库存清单系统

2) 第一层数据流图



第一层数据流图-库存清单系统子系统划分

3) 第二层数据流图



第二层数据流图-分解后的库存清单系统

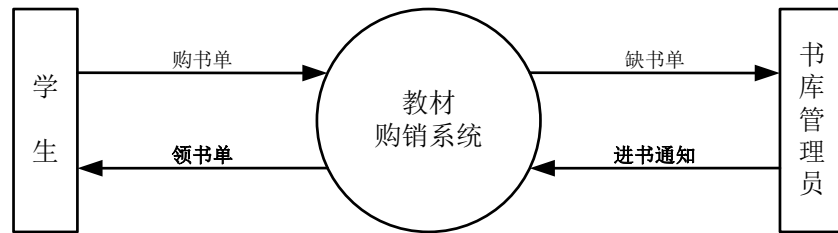
二. 教材购销系统:

1. 系统说明:

在教材的销售过程中，首先学生拿着购书申请到会计处审查并开具购书发票，然后到出纳处交款，并开具领书单，学生拿着领书单到书库领书；在开具购书发票的过程中，若教材存量不够，则需要进缺书统计，然后书库根据缺书情况去采购缺书，并通知学生补购教材。

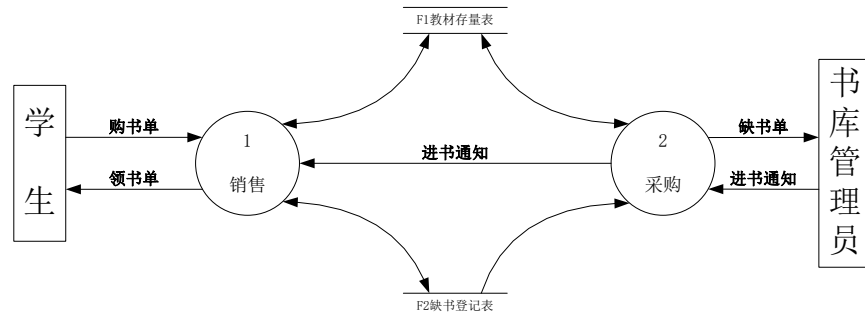
2. 数据流图:

1) 顶层数据流图:



顶层数据流图-教材购销系统

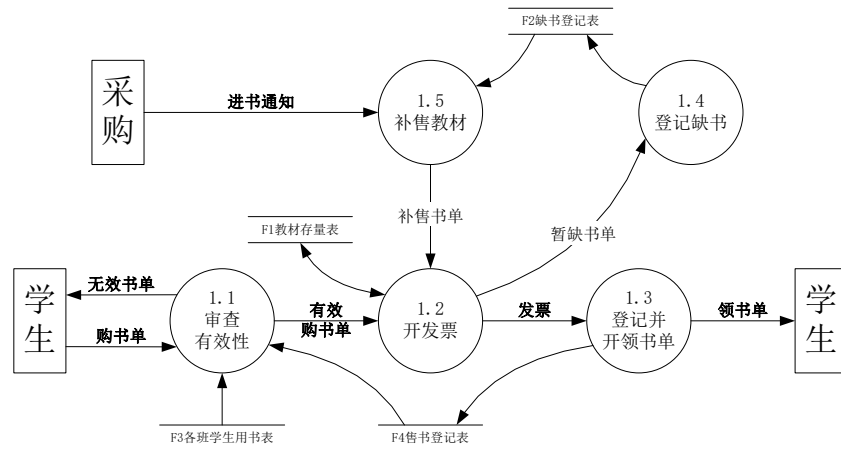
2) 第一层数据流图:



第一层数据流图-教材购销系统子系统划分

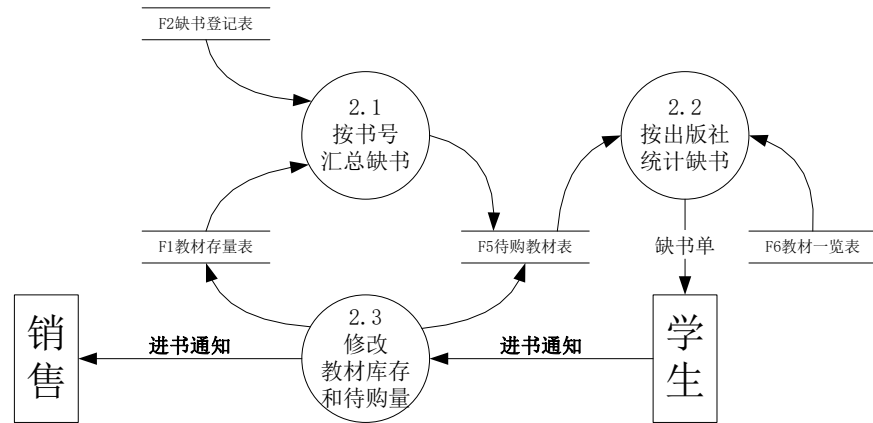
3) 第二层数据流图:

■ 销售子系统 (1)



第二层数据流图-销售子系统

■ 采购子系统 (2)



第二层数据流图-采购子系统

第四章 软件设计概述

§ 4.1 软件设计概述

一. 软件设计的任务：

把需求阶段所产生的软件需求说明转换为用适当手段表示的软件设计文档。“做什么”——>“怎么做”。

二. 软件设计划分两个阶段：

- 概要设计：确定软件的结构，即软件组成，以及各组成成分（子系统或模块）之间的相互转换。
- 详细设计：确定模块内部算法和数据结构，产生描述各模块程序的详细设计文档。

三. 软件设计的方法：面向数据流，面向数据结构。

§ 4.2 软件设计的策略

一. 模块化设计：

1. 模块、模块化：

- 模块：是数据说明，可执行语句等程序对象的集合。例：过程，函数，子程序，宏等。
- 模块化：是把程序划分成若干个模块，每个模块完成一个子功能，把这些模块集中起来组成一个整体，可以完成指定的功能，满足问题的要求。

2. 分解：将一个复杂的问题，划分为几个较小问题。

- 1) “将一个复杂的问题分解为许多小问题，可以减少解决问题的工作量。使原来的问题也就容易解决了。” — 这是模块化设计的依据。

论证：假设 $C(P)$ 是度量对一个问题 P 理解复杂性的函数。 $Z(P)$ 是度量为解决问题 P 所需工作量（用时间计算）的函数，

则给定问题 P_1 , P_2 , 如果 $C(P_1) > C(P_2)$, 那么有 $Z(P_1) > Z(P_2)$, 即一个问题越复杂, 解决它所需要的工作量就越大, 需要花费更多的时间。

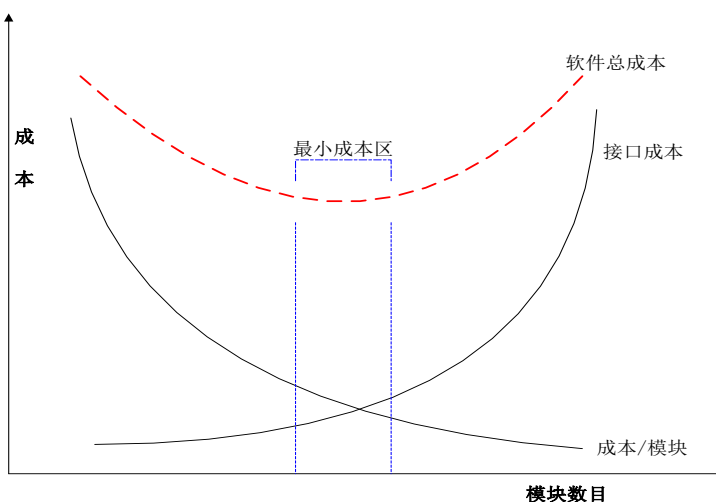
根据人们解决一般问题的实践的经验, 有下面一条客观规律存在:

$$C(P_1 + P_2) > C(P_1) + C(P_2)$$

则可得: $Z(P_1 + P_2) > Z(P_1) + Z(P_2)$

- 2) “无限分解软件, 最后为了开发软件而需要的工作量小的可以忽略” — 不成立。

论证: 随着模块数目增加, 每个模块的规模减少, 成本降低。但相应的设计模块间的接口成本将增加, 使得软件总成本呈抛物线形状, 存在最小成本区。(如图所示)



模块分解论证

3. 信息隐蔽: 指每个模块的实现细节对于其他模块来说是隐蔽的, 即模块中所包含的信息 (数据与过程)。应不允许其他不需要这些信息的模块使用 (即隐蔽起来)。只有为了完成软件的总体功能而必须在模块间交换的信息。才允许在模块间进行传递。

目的: 是软件的修改或错误局限在一个或几个模块内部, 不会涉及软件其他部分。

4. 模块独立性: 模块具有独立功能, 且和其他模块之间没有过多的相互作用。即每个模块完成一个相对独立的特定子功能, 且和其他模

块之间的关系很简单。是软件划分模块时要遵守的准则，也是判断模块构造是否合理的标准。

1) 度量模块独立性的准则：内聚、耦合。

- **内聚**：是模块功能强度（即一个模块内部各个元素彼此结合的紧密程度）的度量。模块内部各元素之间联系越紧密，内聚性越强。
- **耦合**：是模块之间相对独立性（即互相连接的紧密程度）的度量。模块间连接越紧密，联系越多，耦合性越强。
- 模块的独立性越高，其块内联系越紧密（内聚性强），块间联系越弱（耦合性越弱）

2) 内聚：

弱 → 强						
偶然内聚	逻辑内聚	时间内聚	过程内聚	通信内聚	顺序内聚	功能内聚
低 内 聚			中 内 聚		高 内 聚	

- **偶然内聚**：模块内部各组成成分在功能上是互不相关的。
例：几个模块都需要执行“读 A”，“写 B”等相同的一组操作，为避免重复书写，可把这些操作记成一个模块，供有关模块调用。
- **逻辑内聚**：通常由若干个逻辑功能相似的成分组成。
例：一个用于计算机全班学生平均分和最高分的模块，无论计算那种分数，都要经过读入全班学生分数。进行计算、输出计算结果等步骤，除了中间计算外均相同。（两种逻辑相似的功能放入同一模块省去程序中的重复。但却引入用作判断的开关量，增加了块间耦合）。
- **时间内聚**：模块所包含的成分是由相同的执行时间联结在一起的。
例：初始化模块中可能包含“为变量赋初值”，“打开某个文件”等为正式处理做准备的功能。
- **过程内聚**：一个模块内部的处理是相关的，是必须按某一特定次序执行。

例：打开文件，读写文件，关闭文件。

- 通信内聚：模块内部各个成分都使用同一个输入数据。或者产生同一个输出数据。借共用数据联系在一起。

例：

- 顺序内聚：模块中各组成成分是顺序执行的，一个处理框的输出是下一处理框的输入。

例：读入分数，计算平均分，输出结果。

- 功能内聚：模块中的所有成分结合在一起，用于完成一个单一的功能。

例：对一个数开平方；求一组数的最大值；从键盘读入一行字符等。

3) 耦合：

弱 → 强						
非直接耦合	数据耦合	特征耦合	控制耦合	外部耦合	公共耦合	内容耦合
弱 耦 合			中耦合	较 强 耦 合		强耦合

- 非直接耦合：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调用和控制来实现的。

- 数据耦合：模块间通过简单变量所组成的参数表（不是控制参数，数据结构或外部变量）交换数据。

- 特征耦合：模块间通过数据结构所组成的参数表交换数据。

例：房租水电=房租+用水量+用电量（传递参数）。

- 控制耦合：一个模块通过传递开关，标志，名称等控制信息，明显地控制选择另一个模块功能。

例：计算平均分，最高分。

- 外部耦合：一组模块都访问同一个全局简单变量。（不是数据结构，而且不是通过参数表传递的该全局变量的信息）。

- 公共耦合：一组模块都访问同一个公共数据环境（全局数据结构、共享的通信区内存的公共覆盖区等）。

- 内容耦合：两个模块之间发生以下情形。（汇编语言中较多，高级语言中已基本度绝）。

- ◆ 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。
- ◆ 两个模块有一部分代码重叠。
- ◆ 一个模块有多个入口。

4) 为什么说“模块独立性是模块划分时要遵守的准则”，即在模块划分时，为什么要强调模块独立性。

论证：假设把一个问题 P 分解为两个部分 P1 和 P2，如果这两部分不互相独立，用 I1 表示 P1 对 P2 的相互作用因子，I2 表示 P2 对 P1 的相互作用因子，则解决整个问题的实际工作量为：

$$Z(P1 + I1 \times P1) + Z(P2 + I2 \times P2)。$$

当系统的两部分之间联系很松散，即模块独立性很强时，I1，I2 都非常小（ $\rightarrow 0$ ），

则有：

$$Z(P1 + I1 \times P1) + Z(P2 + I2 \times P2) = Z(P1) + Z(P2)$$

根据模块分解的论证，有：

$$Z(P) > Z(P1) + Z(P2)$$

则，可得到：

$$Z(P) > Z(P1 + I1 \times P1) + Z(P2 + I2 \times P2)$$

否则，如果系统的两部分之间联系紧密，即模块独立性很弱时，I1，I2 都很大，

则： $Z(P) > Z(P1 + I1 \times P1) + Z(P2 + I2 \times P2)$ 未必成立。

二. 自顶向下、逐步细化：

1. 自定向下设计：首先要对所设计的系统有一个全面的理解，然后从顶层开始连续地逐层向下分解，直到系统的所有模块都小到便于掌握为止。
2. 逐步细化设计：“细化”的实质，就是分解；而“逐步”则强调每一步分解较其前一步增加“少量”的细节，使得相邻两步之间只有微小的变化，从而容易理解和验证有效性。

§ 4.3 概要设计

一. 概要设计阶段需要完成的工作：

1. 制定规范：软件开发组在设计时应共同遵守的标准。
 - 1) 规定设计文档的编制标准（文档体系、用纸、样式、记述详细程序、图形画法等）。
 - 2) 规定编码的信息形式（代码体系），与硬件、操作系统的接口规约，命名规则等。
2. 软件结构的总体设计：决定软件的总体结构。
 - 1) 采用某种设计方法，将一个复杂的系统按功能划分成模块的层次结构。
 - 2) 确定每一个模块的功能，建立与已确定的软件需求的对应关系。
 - 3) 确定模块间的调用关系。
 - 4) 确定模块间的接口，即模块间传递的信息，设计接口的信息结构。
 - 5) 评估模块划分的质量及导出模块结构的规则。
3. 数据结构的设计：决定文件系统的结构或数据库的模式，子模式以及数据完整性，安全性设计。
 - 1) 确定输入、输出文件的详细的数据结构。
 - 2) 模式设计：确定物理数据库结构。
 - 3) 子模式设计：确定用户使用的数据视图。
 - 4) 数据的完整性，安全性设计。
 - 5) 数据优化：改进模式与子模式，以优化数据的存取。
4. 编写文档：《概要设计说明书》
 - 1) 引言：编写目的，背景，参数资料等。
 - 2) 系统概述：目标，运行环境，需求概述。
 - 3) 结构设计：
 - 软件的总体结构。
 - 模块的外部设计：（包括关于各模块功能，性能及接口的简要描述）。
 - 4) 数据结构设计：文件系统结构，数据库模式，子模式，完整性，

安全性，访问方法，存储要求等。

5) 初步测试计划：对测试的策略，方法和步骤提出要求。

5. 技术审查和管理复审。

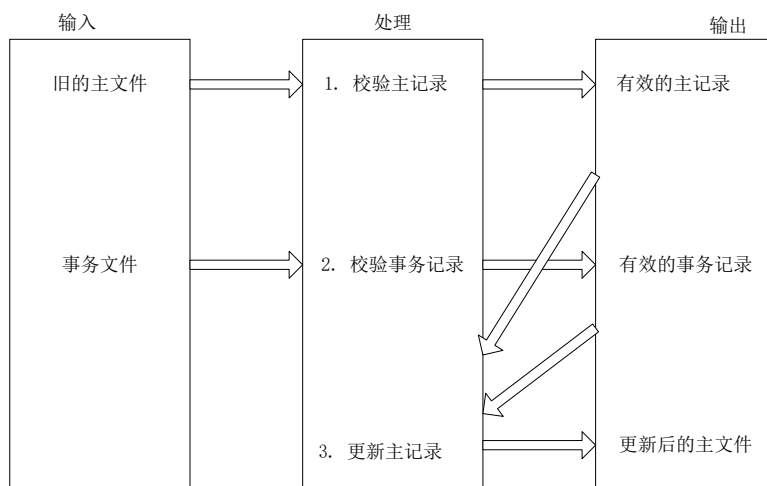
二. 系统结构描述：(HIPO 图)

1. HIPO 图：即 H+IPO。由一张 HC 图加一组 IPO 图组成。

2. HC 图：(层次图)。用于表示软件的层次结构。

3. IPO 图：用来描述 HC 图中的每一个模块，由输入、处理和输出三个框组成。需要时可增加一个数据文件(库)框。

图形表示：系统的 IPO 图，改进的 IPO 图(如图所示)。



HIPO图(典型)

系统: _____	作者: _____
模块: _____	日期: _____
编号: _____	
被调用:	调用:
输入:	输出:
处理:	
局部数据元素:	注释:

HIPO图(改进)

§ 4.4 结构化系统设计

一. 概述

1. 结构化系统设计 (SD): 是面向数据流的系统设计方法, 其要解决的任务是在需求分析的基础上, 将 DFD 图“映射”为软件系统的结构。
2. 结构化系统设计的步骤: (实施要点: 如图所示)

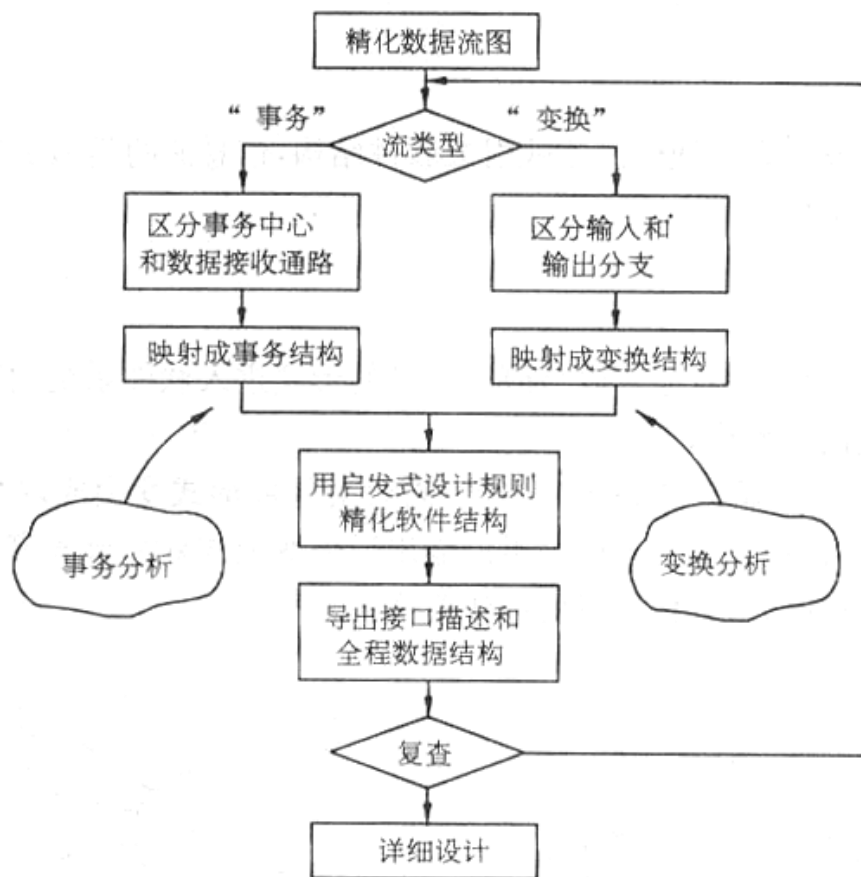


图 4.10 面向数据流方法的设计过程

- 1) 研究、分析、审查 DFD 图, 必要时可再次进行修改和细化。
- 2) 根据 DFD 图来决定软件系统的结构特征。
- 3) 由 DFD 图来决定软件系统的结构图 (SC 图)。
- 4) 按照设计改进原则, 优化和改进初始的 SC 图, 获得最终 SC 图。

二. 软件结构

1. 变换型结构: 信息由传入路径进入系统, 经变换中心加工处理后, 沿传出路径离开系统。在所有过程中信息经历了外部形式 → 内部形式 → 外部形式的输出。由传入路径, 传出路径和变换中心三部

分组成，流经这三个部分的数据，流分别称为传入流，传出流和变换流。（如图所示：书 P74）

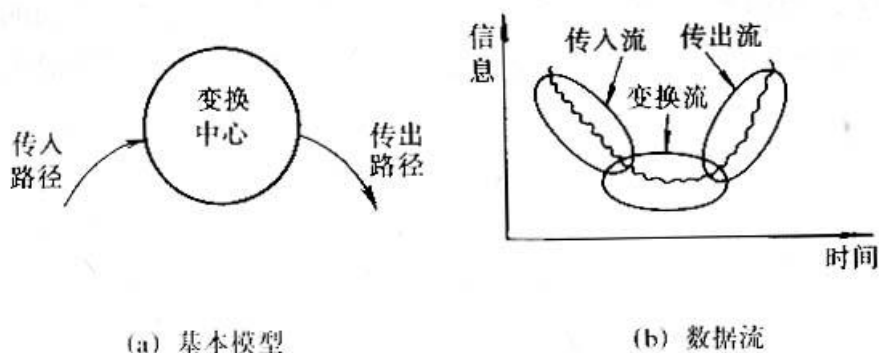


图 5.1 变换型结构的系统

2. 事务型结构：具有在多种事务中选择执行某类事务的能力，由至少一条接受路径，一个事务中心与若干条动作路径组成（如图所示：书 P75）

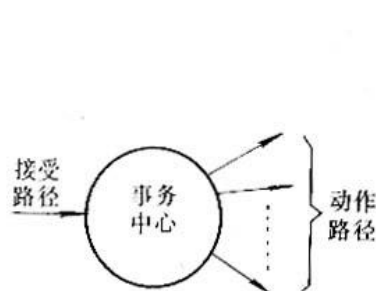


图 5.2 事务型结构的系统基本模型

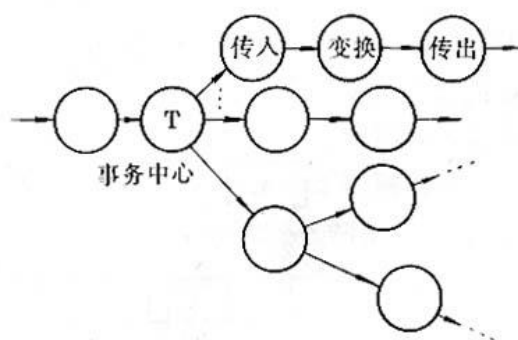
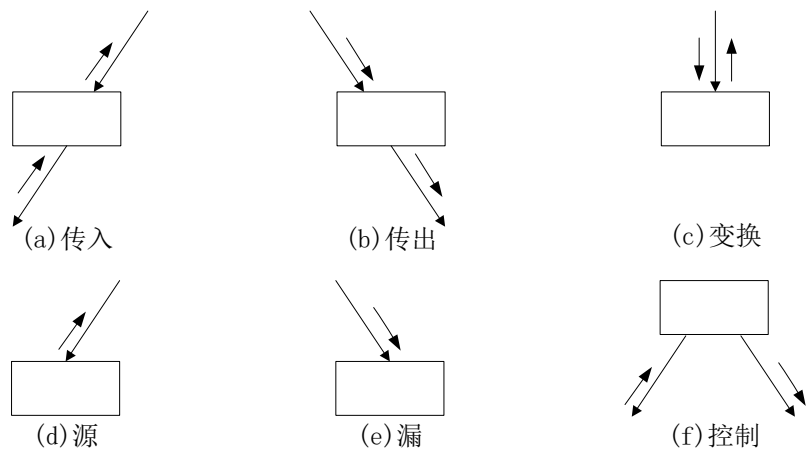


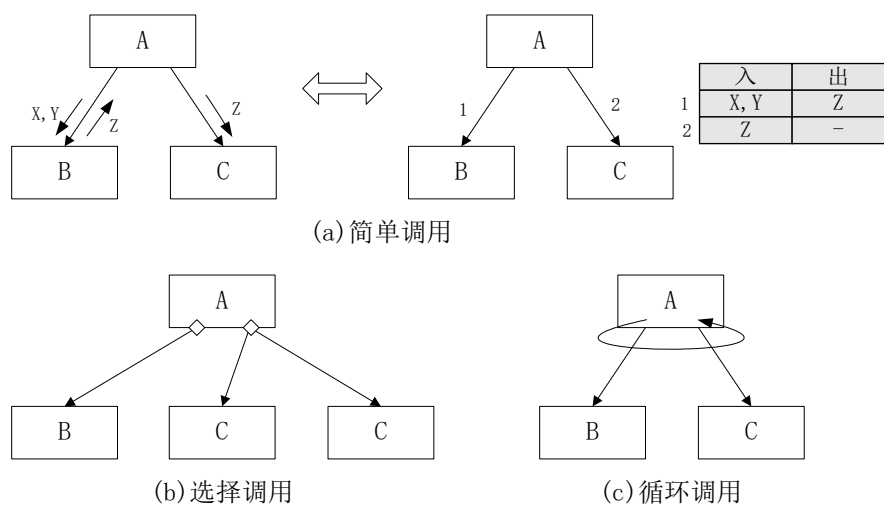
图 5.3 同时存在两类结构的系统

三. 结构图：

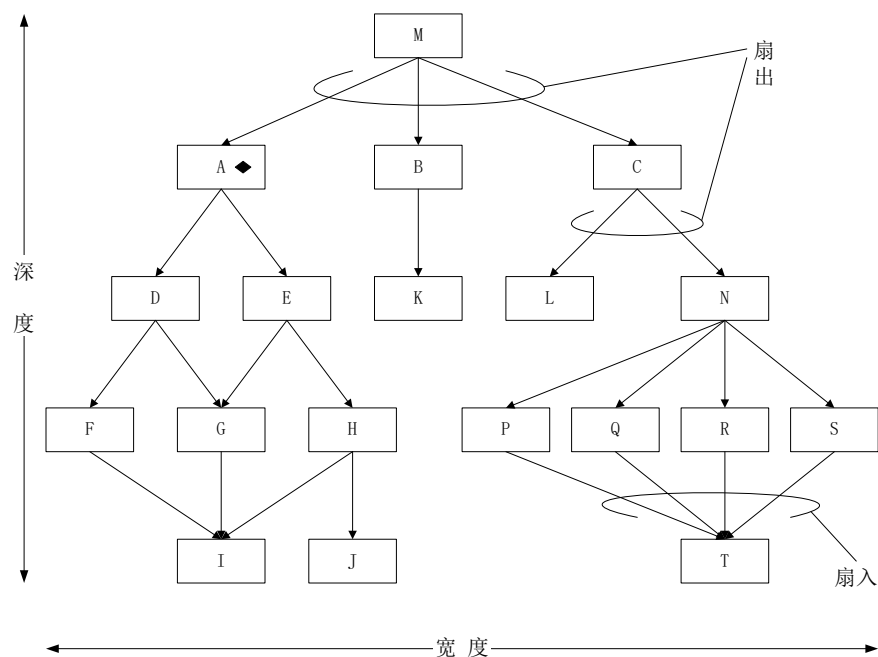
1. 结构图（SC）：是 SD 方法在概要设计中使用的主要表达工具，用来显示软件的组成模块及其调用关系。
 2. 符号：（如图所示：书 P76、77）
 - 矩形框表示模块
 - 带箭头的连线表示模块间的调用关系
 - 在调用线的两端，用空心箭头表示可传入，传出模块的数据流。
- 1) 模块的表示：6 种模块（如图所示）



2) 模块调用的表示：简单调用、选择调用、循环调用（如图所示）



3. SC 图的形态特征：（如图所示）



SC图的形态特征

- 1) SC 图的深度：指在多层次的 SC 图中，其模块结构的层次数。结构图的深度在一定意义上反映了程序结构的规模和复杂程度。
 - 2) SC 图的宽度：SC 图中同一层模块的最大模块数。
 - 3) 模块的扇入：调用（或控制）一个给定模块的模块数目。
 - 4) 模块的扇出：一个模块直接调用（或控制）的其他模块数目。
 - 5) 模块的控制范围：包括模块本身及其所有从属模块，而不论这些从属模块是由该模块直接调用，还是间接调用。
 - 6) 模块的作用范围：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。
4. 改进 SC 图的指导原则：
- 1) 改进软件结构提高模块独立性：降低耦合，提高内聚。
 - 2) 模块的规模比较适中：10—100 条语句。（坚持模块独立性，是划分模块的最高准则）。
 - 3) 高扇入，低扇出：一个模块的扇入数越高，则共享这一模块的上级模块越多，消除重复的效果越明显；扇出越高，则暴露出初始 SC 图中分解太快的缺点。
 - 4) 一个判定的作用范围应限制在判定所在模块的控制范围之内。

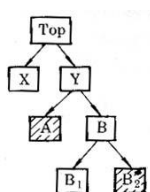


图 5.29 判定位置违反了作用范围/控制范围原则

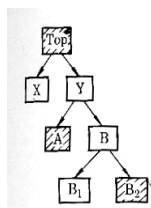


图 5.30 符合作用范围/控制范围原则，但判定位置太高

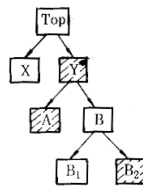


图 5.31 符合作用范围/控制范围原则，判定位置适中

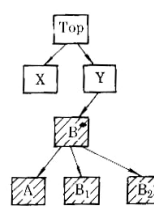


图 5.32 符合作用范围/控制范围原则的理想判定位置

- 5) 降低模块接口的复杂程度：接口复杂是软件发生错误的一个主要原因。
- 6) 设计单入口单出口的模块：避免出现内容耦合。
- 7) 模块功能应该可以预测：输入相同数据，产生相同输出。（若模块有内部“存储器”，其功能不可预见）。

四. 结构化系统设计举例：

1. 变换分析

■ 系统名称

汽车数字仪表板系统

■ 系统功能概述

- 1) 通过模—数转换实现传感器和微处理机接口；
- 2) 在发光二级管面板上显示数据；
- 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。

■ 设计步骤

- 1) 复查基本系统模型。
- 2) 复查并精化数据流图。（如图所示）

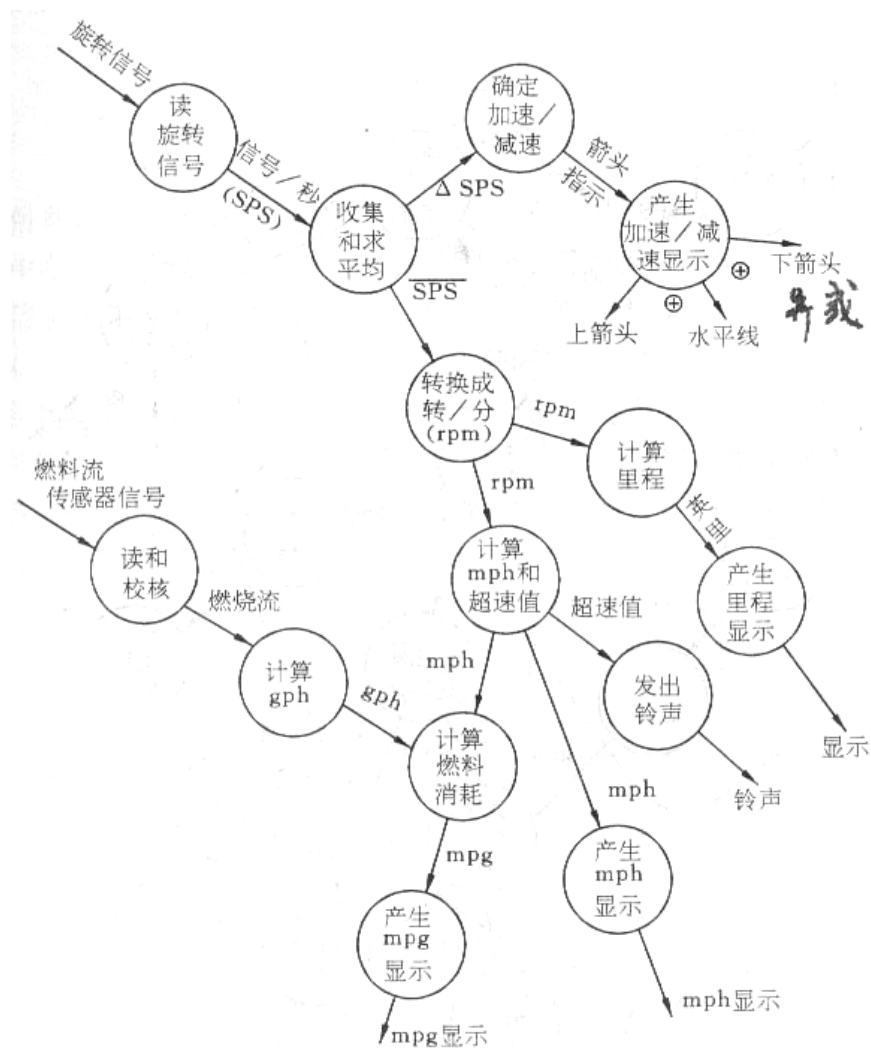


图 4.11 数字仪表板系统的数据流图

- 3) 确定数据流图具有变换特性还是事务特性。
- 4) 确定输入流和输出流的边界，从而孤立出变换中心。

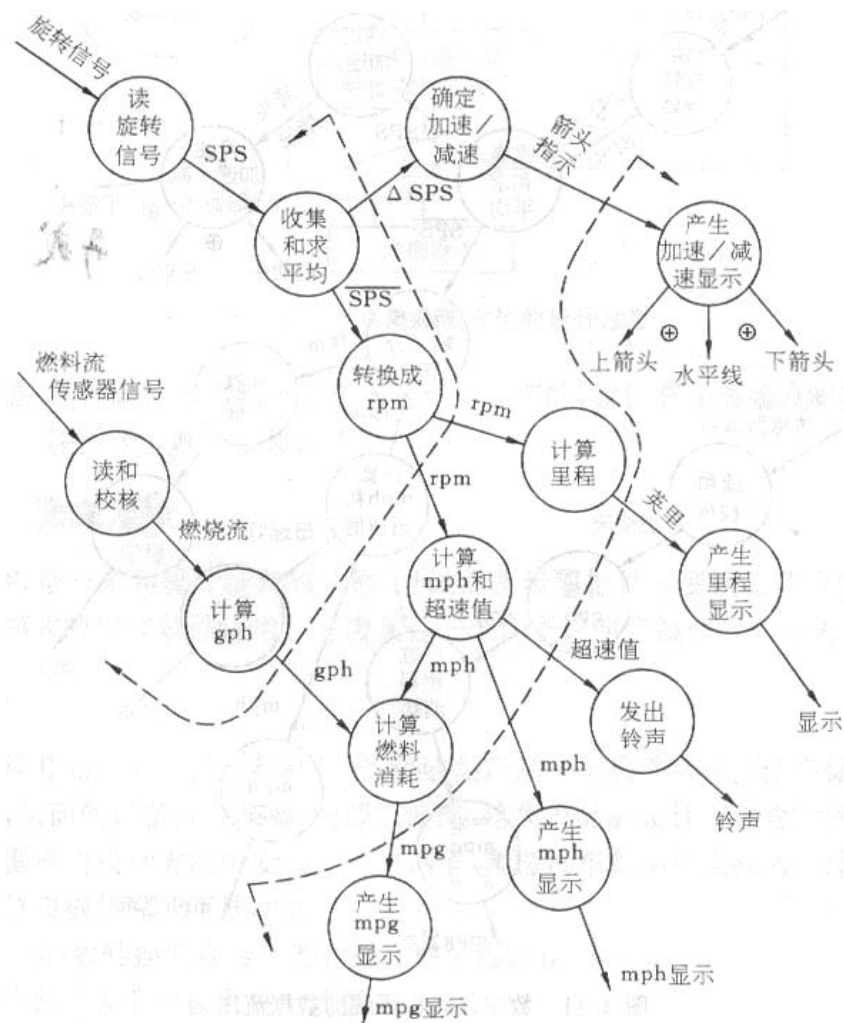


图 4.12 具有边界的数据流图

5) 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。变换流的分解是将数据流图映射成一个特殊的软件结构，这个结构控制输入、变换、输出等信息处理过程（如图所示），位于软件结构最顶层的控制模块 C_m 协调下述从属的控制功能：

- 输入信息处理控制模块 C_a ，协调对所有输入数据的接收；
- 变换中心控制模块 C_t ，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 C_e ，协调输出信息的产生过程。

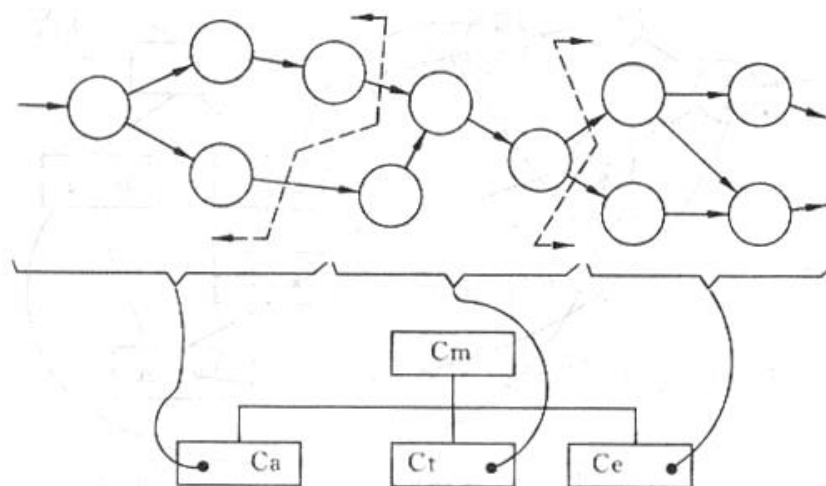


图 4.13 第一级分解的方法

第一级分解得出的结构（如图所示）

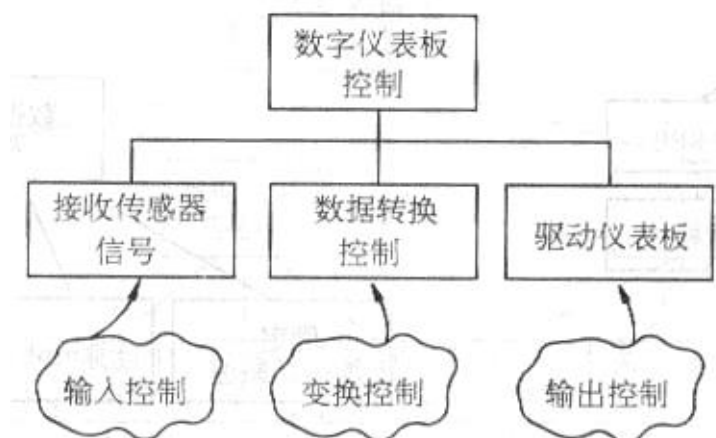
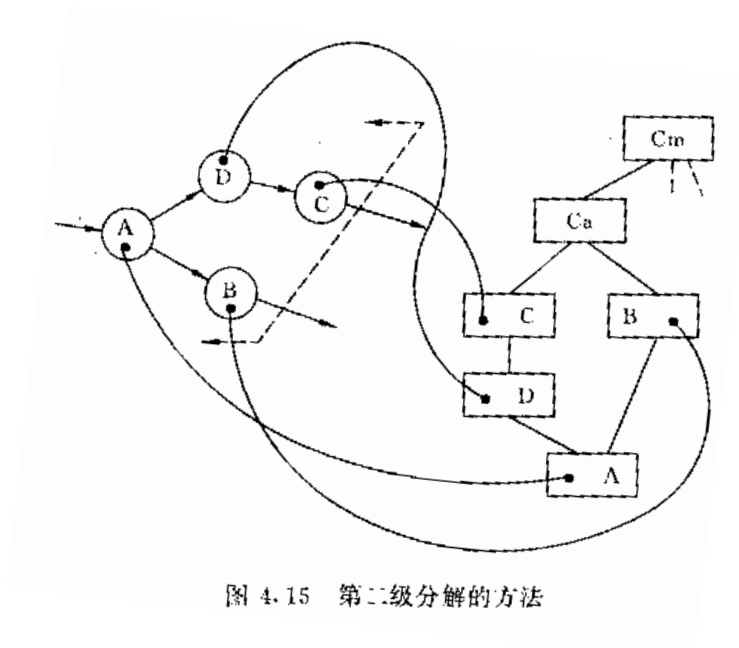


图 4.14 数字仪表板系统的第一级分解

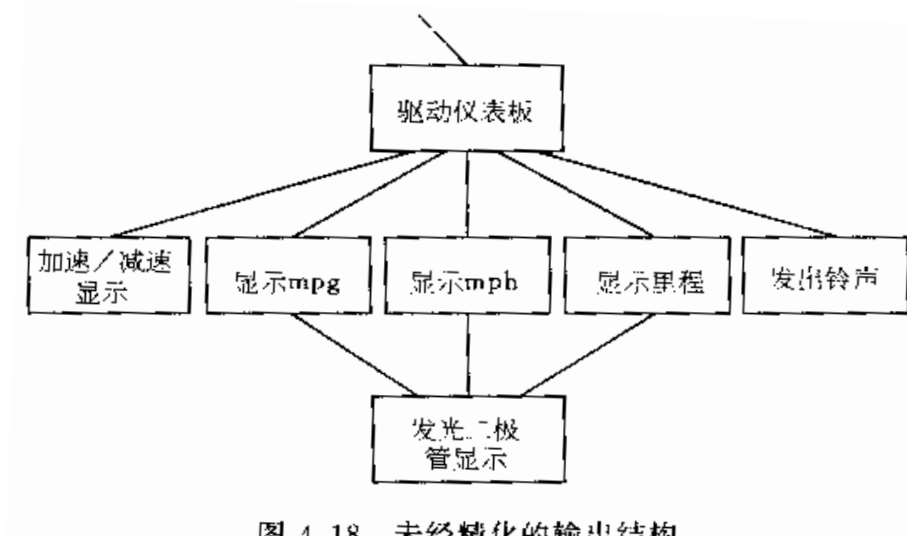
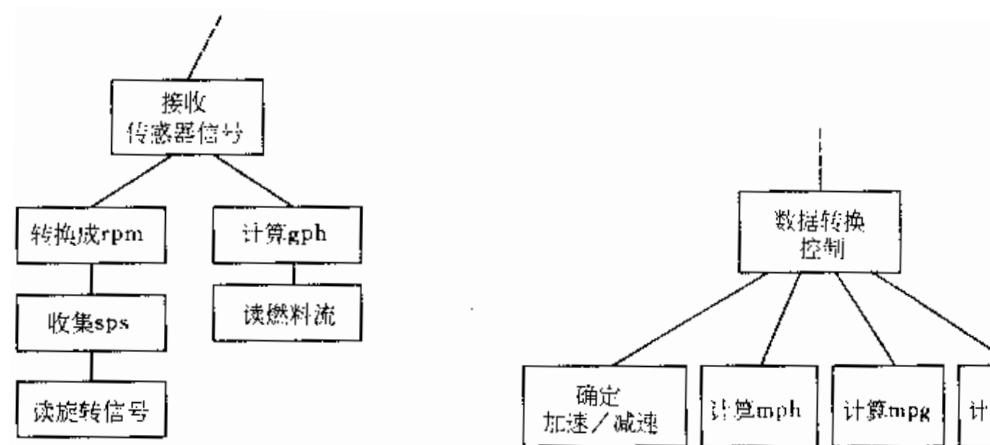
6) 完成“第二级分解”。

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是：

从变换中心的边界开始沿着输入路径向外移动，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模块 Ct 控制的一个模块（如图所示）。



第一级分解得出的结构（如图所示）



7) 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行一步精化。

修改如下：(如图所示)

- 输入结构中的模块“转换成 rpm”和“收集 sps”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。

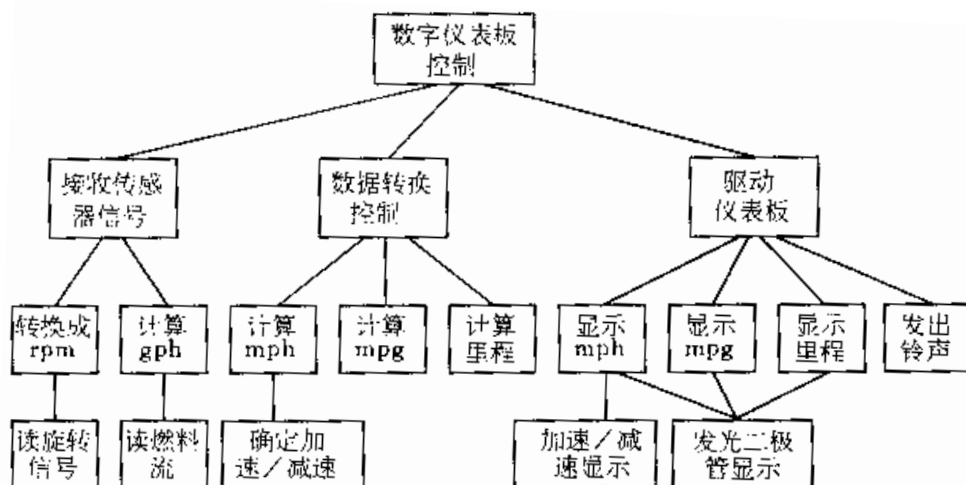


图 4.19 精化后的数字仪表盘系统的软件结构

2. 事务分析

事务分析的设计步骤和变换分析大部分相同或类似，主要差别仅在于由数据流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很相象，即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（如图所示）。

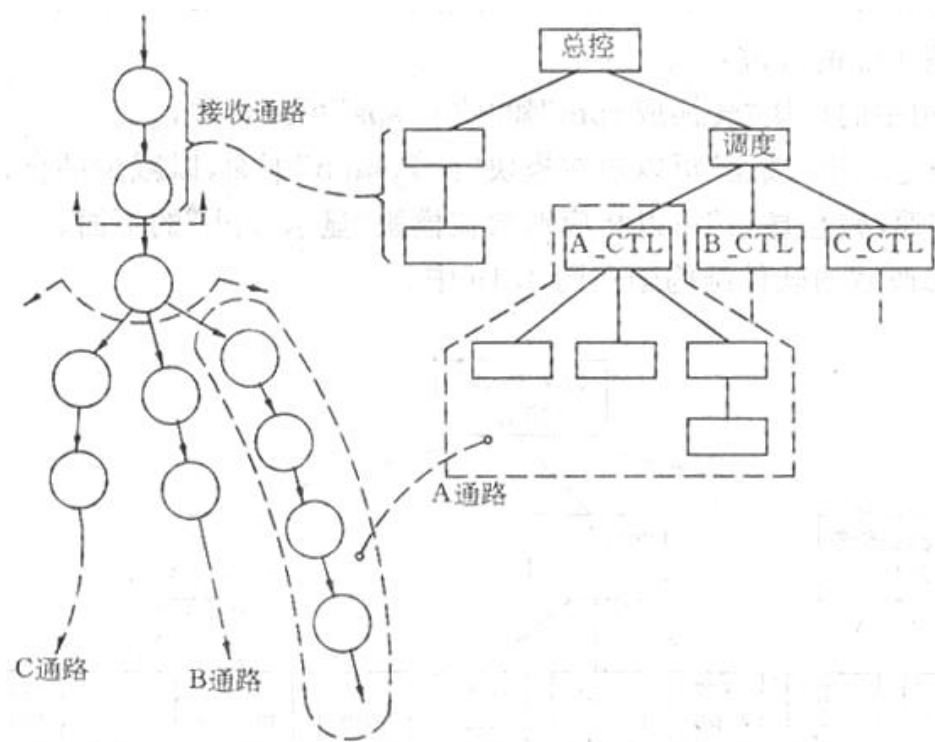


图 4.20 事务分析的映射方法

§ 4.5 小结

§ 4.6 补充实例

第五章 详细设计

§ 5.1 详细设计概述

一. 目的:

是为软件层次图（HC）或结构图（SC）中的各个模块确定采用的算法和块内数据结构，用某种选定的表达工具给出清晰的描述。

二. 详细设计需要完成的工作:

1. 确定软件各个组成部分内部的算法；确定各部分内部的数据结构；确定模块接口细节（包括外部接口，用户界面，模块间接口，以及关于模块输入数据、输出数据及局部数据的全部细节）。
2. 选定某种过程的表达形式来描述各种算法
3. 编写文档：《详细设计说明书》
 - 1) 系统概述
 - 2) 软件结构：给出软件系统的层次图或结构图。
 - 3) 程序描述：逐个模块给出以下说明
 - 功能描述
 - 性能描述
 - 输入项目
 - 输出项目
 - 算法及数据结构：模块所选用的算法和所用到的数据结构。
 - 程序逻辑：详细描述模块的内部实现算法（流程图；N—S 图；PAD 图；PDL 语言等）。
 - 接口细节
 - 测试要点：详细描述模块的主要测试要求，提供一组测试用例。
4. 技术审查和管理复审

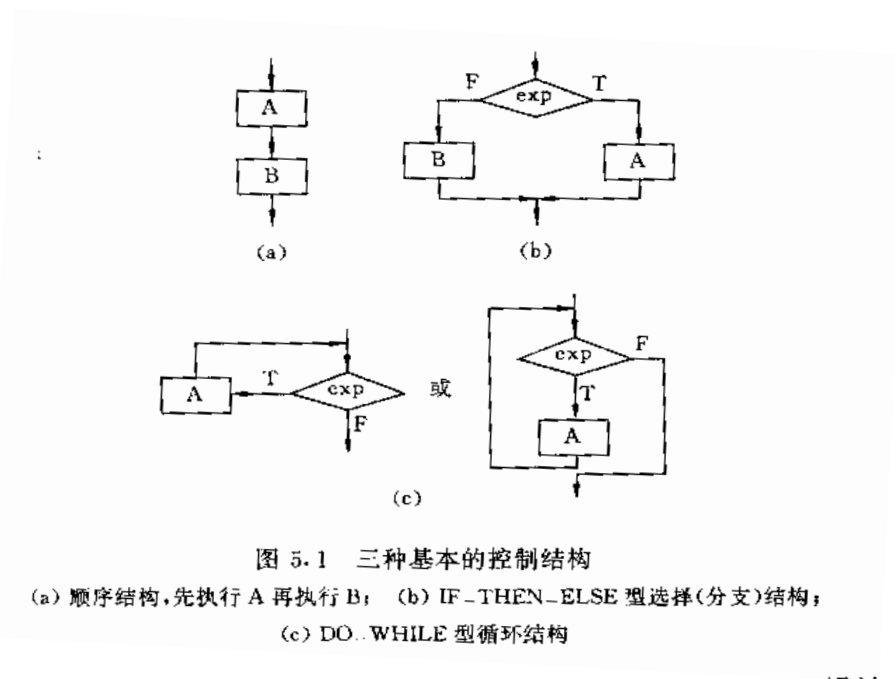
§ 5.2 结构化程序设计

一. 结构化程序设计：

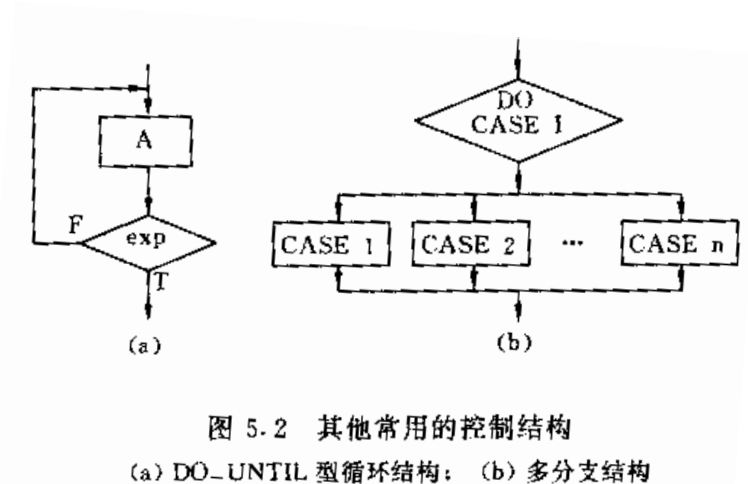
是一种设计程序的技术，它采用自定向下，逐步求精的设计方法和单入口，单出口的控制结构。

二. 结构化的控制结构：

1. 限制使用 GOTO 语句：效率与清晰的权衡。
2. 基本的控制结构：任何程序的逻辑均可用“顺序”、“选择”、“循环”三种控制结构或它们的组合来实现。（如图所示）



3. 补充控制结构：（如图所示）



- DO—UNTIL 循环结构。
- DO—CASE 多分支选择结构。
- 受限制的 GOTO 语句：从循环中快速退出。

三. 逐步求精：

将一个模块的功能逐步分解为一系列具体的处理步骤。

四. 结构化程序设计的优点：

1. 自顶向下、逐步求精的方法是人类解决复杂问题的普通规律，可以显著提高软件开发工程的成功率和生产率。
2. 用先全局后局部，先整体后细节，先抽象后具体的逐步求精过程开发出的程序，有着清晰的层次结构，容易阅读和理解。
3. 不使用 GOTO 语句仅使用单入口、单出口的控制结构，使得程序的静态结构和动态执行情况比较一致。
4. 控制结构有确定的逻辑模式，编写程序代码只限于使用很少几种直截了当的方式，因此源程序清晰流畅，易读易懂易于测试。
5. 程序清晰和模块化使得在修改和重新设计一个软件时可重用的代码量很大。
6. 程序的结构清晰，有利于程序的正确性的证明。

§ 5.3 详细设计的描述工具

一. 程序流程图：（程序框图）

1. 符号：

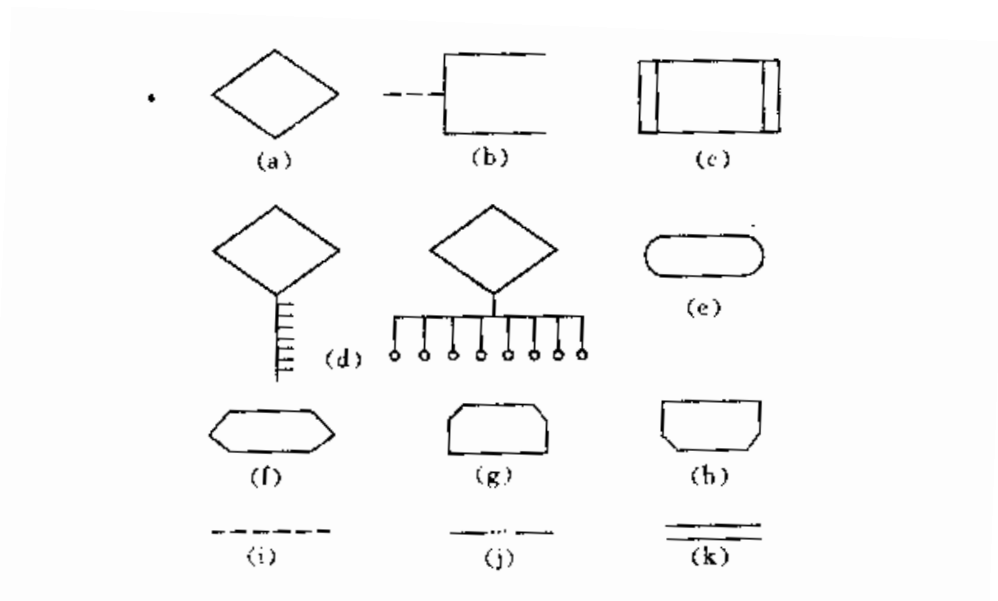


图 5.3 程序流程图中还使用系统流程图中没有的符号

(a)选择(分支);(b)注释;(c)预先定义的处理;(d)多分支;(e)开始或停止;
(f)准备;(g)循环上界限;(h)循环下界限;(i)虚线;(j)省略符;(k)并行方式

2. 控制结构:

3. 扩展: 外部—>; 内部—>;

4. 主要优点: 对控制流的描述很直观, 便于初学者掌握。

5. 缺点:

- 1) 本质上不是逐步求精的好工具, 使程序员过早地考虑程序控制流程, 而忽略了全局结构。
- 2) 流程图中用箭头代表控制流, 程序员可以不受约束, 不顾结构化程序设计的精神, 随意转移控制。
- 3) 不易表示数据结构。

二. N_S 图: (盒图)

1. 控制结构: (书: P84)

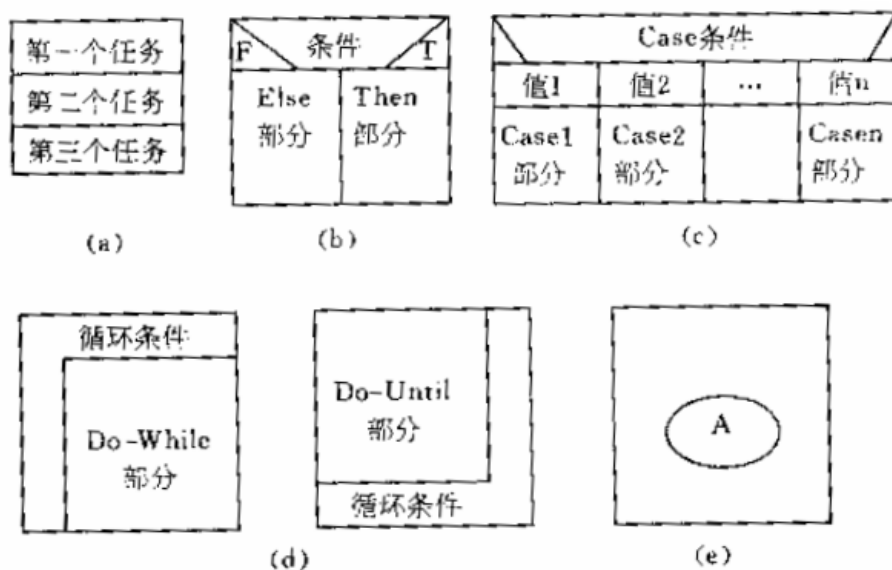


图 5.4 盒图的基本符号

(a) 顺序；(b) IF-THEN-ELSE 型分支；(c) CASE 型多分支；(d) 循环；(e) 调用子程序 A

2. 特点：

- 1) 功能域（即一个特定控制结构的作用域）明确，可以从 N_S 图上一眼看出来。
- 2) 不可能随意转移控制。
- 3) 很容易确定局部和全部数据的作用域。
- 4) 很容易表现嵌套关系，也可以表示模块的层次结构。

3. 扩展：子程序调用。

三. PAD 图：（问题分析图）

1. 控制结构：（书：P85）

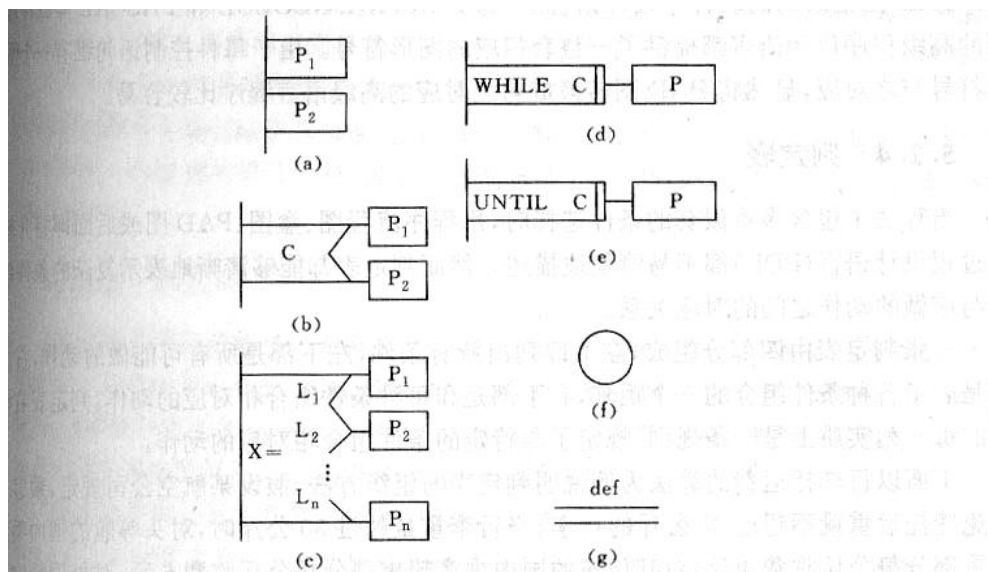


图 5.5 PAD 图的基本符号

(a) 顺序(先执行 P_1 后执行 P_2)；(b) 选择(IF C THEN P_1 ELSE P_2)；(c) CASE 型多分支；

(d) WHILE 型循环(WHILE C DO P)；(e) UNTIL 型循环(REPEAT P UNTIL C)；(f) 语句标号；(g) 定义

2. 扩展：

■ FOR 重复型

■ DEF 格式

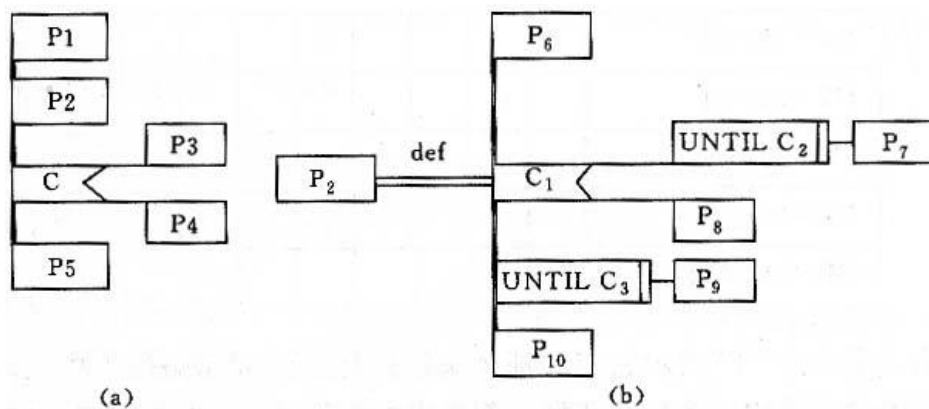


图 5.6 使用 PAD 图提供的定义功能来逐步求精的例子

(a) 初始的 PAD 图；(b) 使用 def 符号细化处理框 P_2

3. 优点：

PAD 图支持自定向下，逐步求精方法的使用，设计出来的程序为结构化程序，表达的软件结构呈树状结构，即克服了传统的流程图不能清晰表现程序结构的缺点，又不像 N—S 图那样把全部程序约束在一个方框内，同时 PAD 图也可以用于描绘数据结构。

四. PDL 语言：（过程设计语言，伪码）

五. 举例：

在一个数组中求最大数，分别用程序流程图、N-S 图、PAD 图来描述这一详细设计过程：

1. 程序流程图：

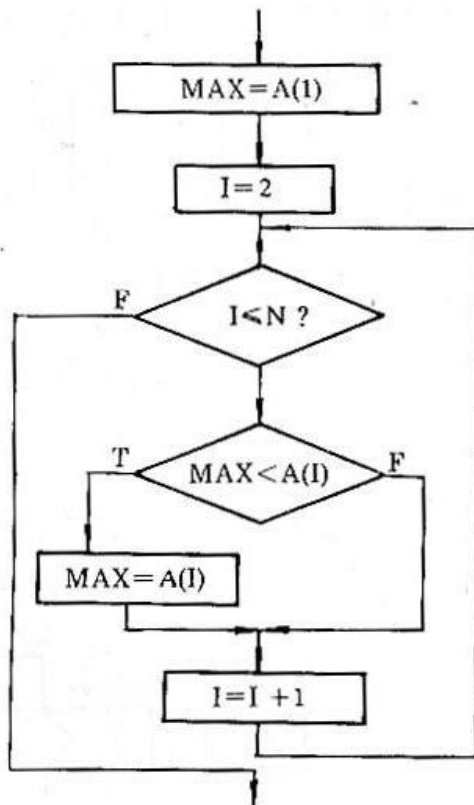


图6.9 例4.1 的流程图

2. N-S 图：

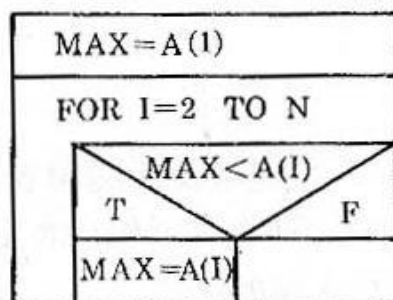


图 6.8 例 4.1 的 N-S 图表述

3. PAD 图：

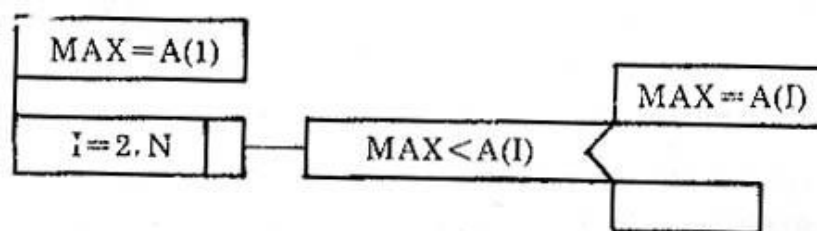


图 6.7 例 4.1 的 PAD 图表述

§ 5.4 其它的软件设计方法（面向数据结构的设计方法）

一. Jackson 设计方法：（英国人 M. A. Jackson）

1. Jackson 图：符号表示（书：P89）
2. Jackson 设计方法的步骤：
 - 1) 分析并确定输入数据和输出数据的逻辑结构，并用 Jackson 图描绘这些数据结构。
 - 2) 找出输入数据结构和输出数据结构中对应关系的数据单元。
 - 3) 以描绘数据结构的 Jackson 图导出描绘程序结构的 Jackson 图。
 - 4) 列出所有操作和条件（包括分支条件和循环结束条件），并且把它们分配到程序结构图的适当位置。
 - 5) 用伪码表示程序。
3. 举例：（书：P91）

二. Warnier 设计方法：（逻辑构造程序方法 LCP）

1. Warnier 图：符号表示（书：P46）
2. Warnier 设计方法的步骤：
 - 1) 分析和确定输入数据和输出数据的逻辑结构，并用 Warnier 图描绘这些数据结构。
 - 2) 主要依据输入数据结构导出程序结构，并用 Warnier 图描绘程序的处理层次。
 - 3) 画出程序流程图并自上而下依次给每个处理框编码。
 - 4) 分类写出伪码指令。
 - 5) 把前一步中分类写出的指令按序号排序，从而得出处理过程的伪码。

3. 举例：（书：P96）

§ 5.5 程序复杂度的定量度量

一. 程序图：

1. 程序图：

是一种简化了的流程图，流程图中各个框（包括加工框和判定框等）被简化为一个用圆圈表示的节点，箭头变换为连接不同点的有向弧。

2. 基本控制结构的表示：（如下图所示）

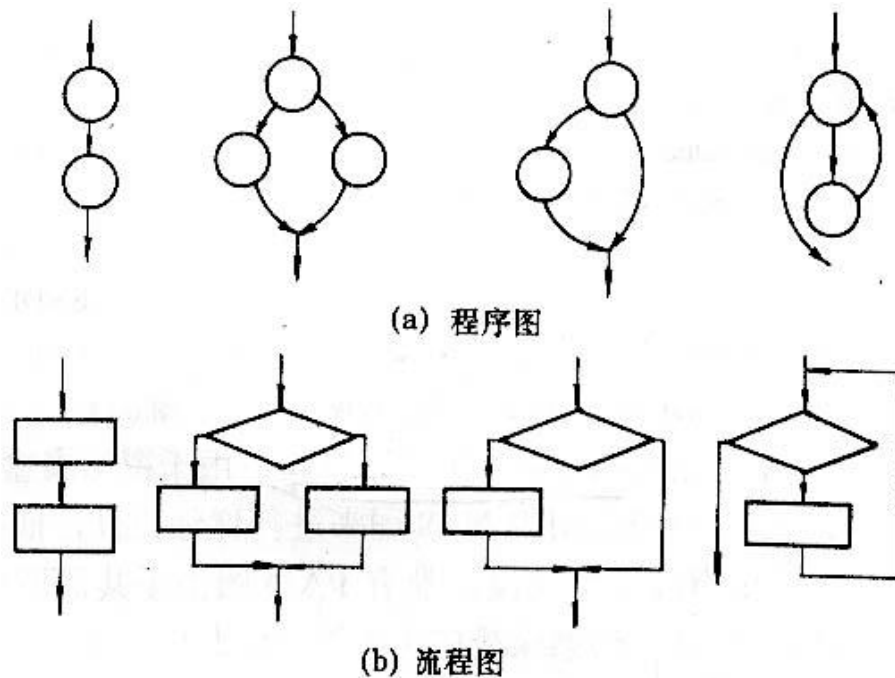


图 6.15 基本控制结构的程序图

3. 举例：

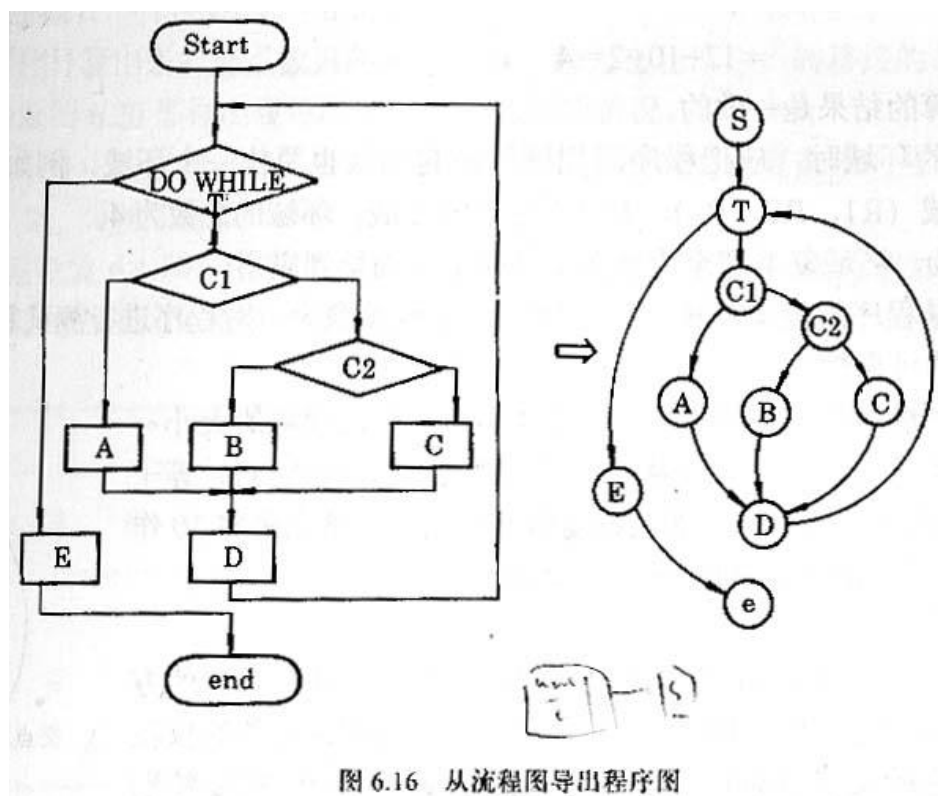


图 6.16 从流程图导出程序图

二. 环域复杂度: (T. McCabe)

1. 决定因素:

一个程序的环域复杂度取决于它的程序图所包含的判定结点的个数。

2. 计算方法:

设 G 为被度量的程序图, V 为环域数, 则有以下计算方法:

$$V(G) = \text{判定结点数} + 1$$

环域数

$$N_e - N_v + 2P$$

其中: N_e 为有向图中的边数; N_v 为有向图中的结点数; P 为有向图中不连通部分的数目, 由于在正常程序图中所有节点都是连通的, 所有 P 恒等于 1。

3. 应用:

- 用来度量程序的测试难度: 一般的说, 环域数愈大, 对程序进行测试和排错也愈难, 最终将影响程序的可靠性。
- 用来限制模块的最大行数: 实践表明, 模块规模以 $V(G) \leq 10$

为宜。

三. 交点复杂度: (M. R. Woodward)

1. 描述: 用程序图中交叉点的个数来度量程序复杂度。
2. 注意: 所有转移线必须画在节点的同一侧, 才能得出正确的交点数目。

四. 程序工作量: (Halstead)

描述:

设 $N1$ 为程序中操作符的总数 (在程序中出现的总次数)。
 $N2$ 为程序中操作数的总数 (在程序中出现的总次数)。
 $n1$ 为程序中操作符的总数 (在程序中出现的种类数)。
 $n2$ 为程序中操作数的总数 (在程序中用例的种类数)。

则有: 程序长度 $N = N1 + N2$
 程序信息量 $V = (N1 + N2) \log_2 (n1 + n2)$
 语言抽象系数 $L = (2 \times n2) / (n1 \times N2)$
 程序工作量 $E = V / L$

$$= n1 \times N2 \times (N1 + N2) \times \log_2 (n1 + n2) / (2 \times n2)$$

§ 5.6 小结

§ 5.7 补充实例

画出与下面的程序流程图相对应的 N-S 图、PAD 图和程序图:

1. 程序流程图:

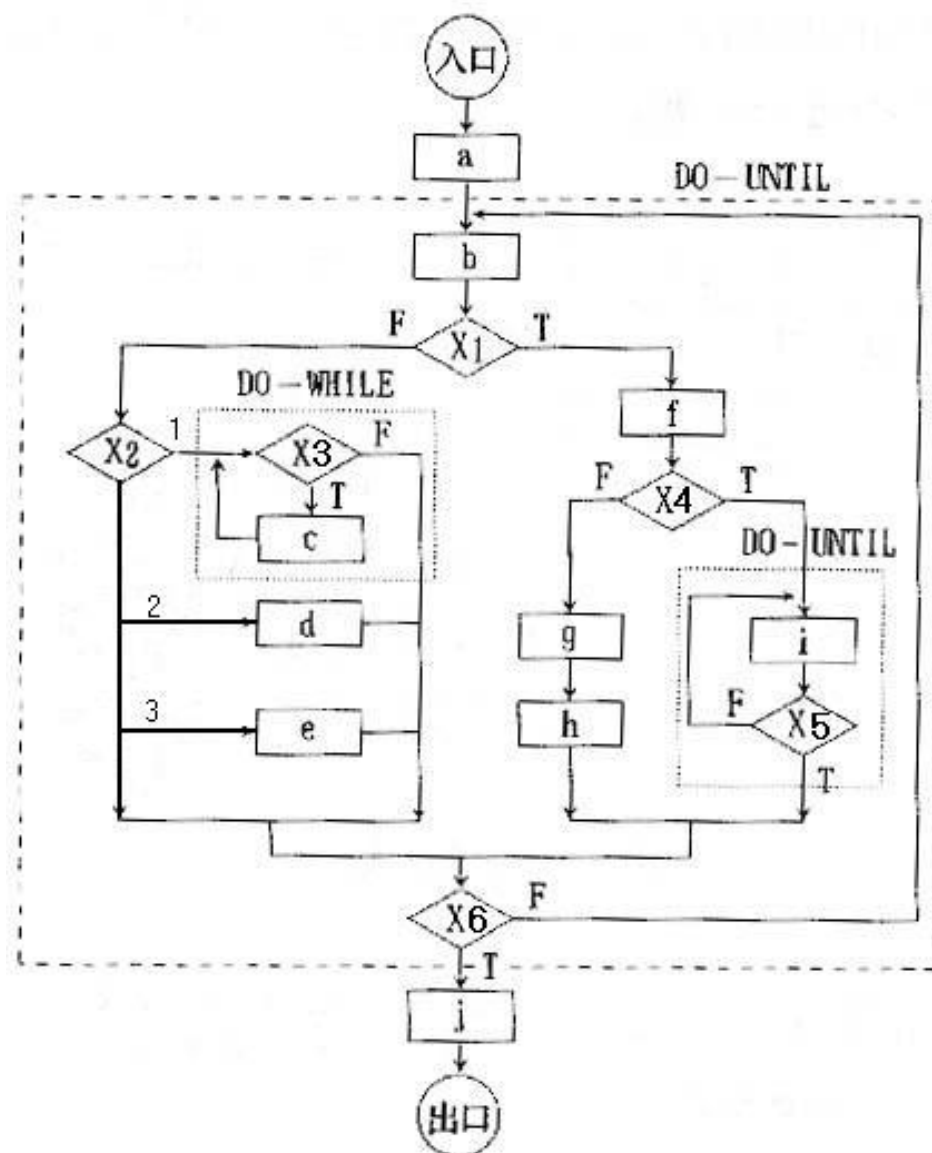


图 4.48 不包含多分支结构的流程图实例

2. N-S 图:

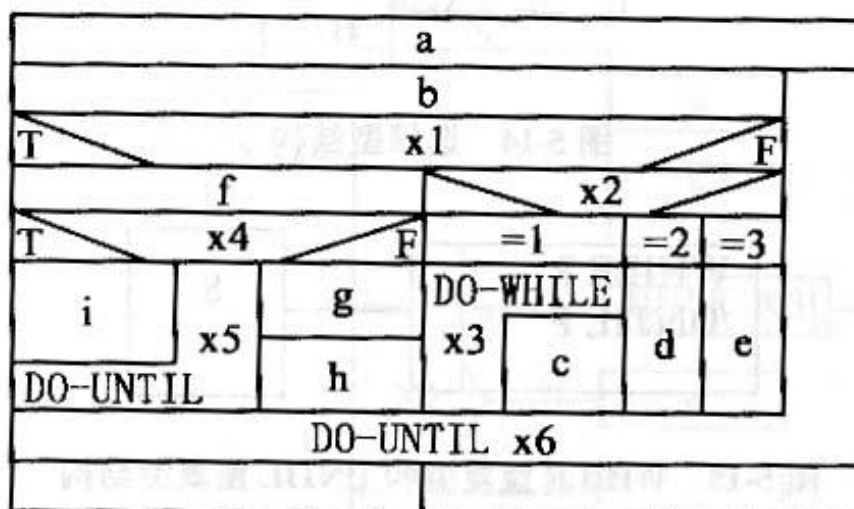


图 5-12 N-S 图举例

3. PAD 图:

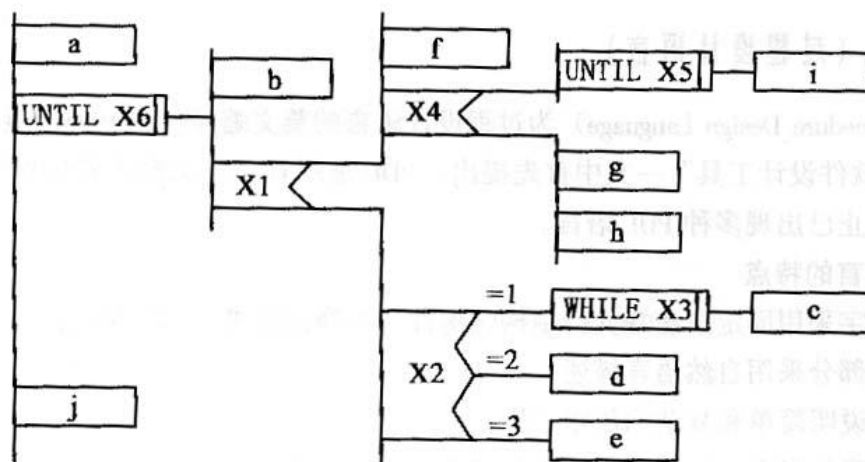
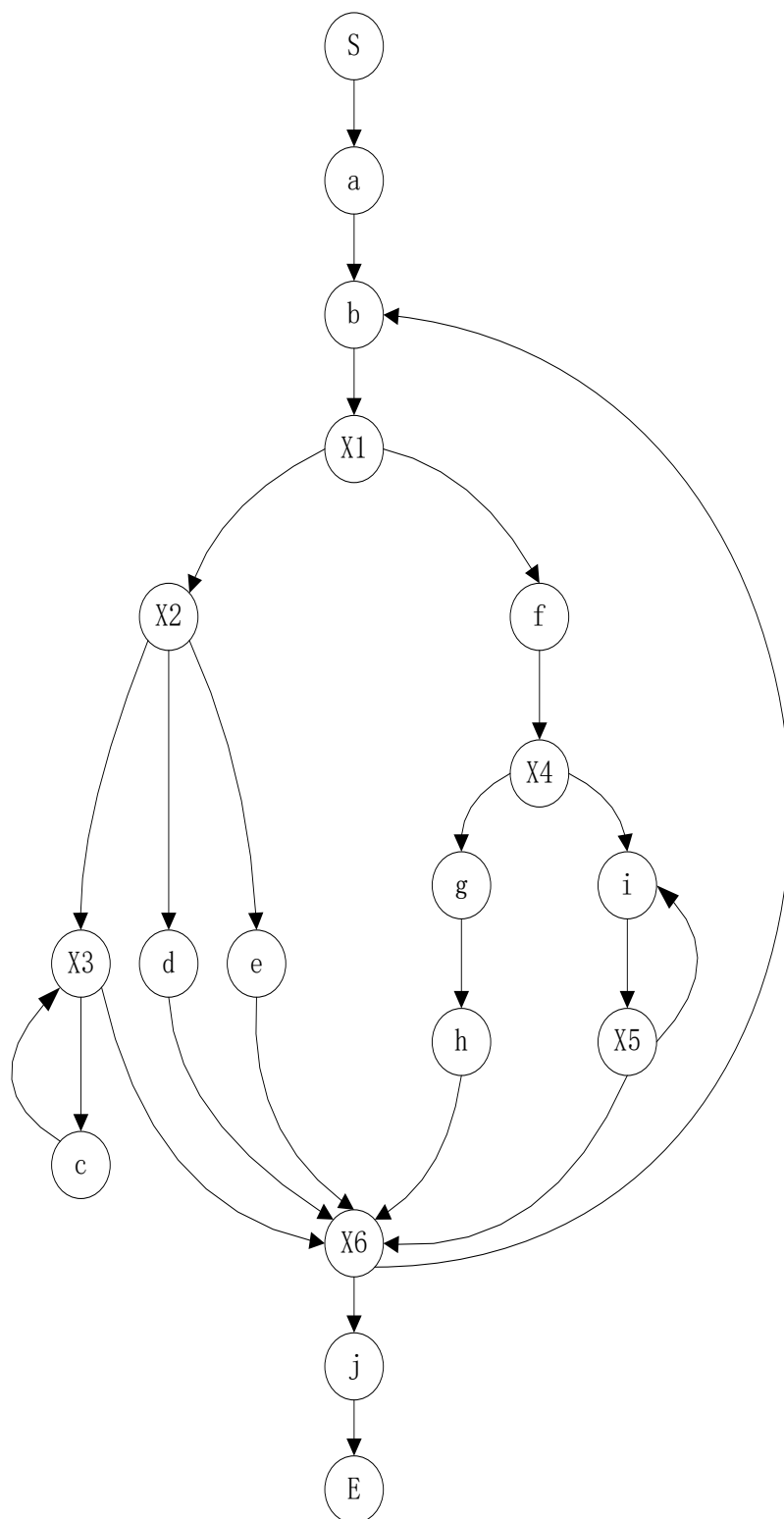


图 5-17 PAD 图举例

4. 程序图:



第六章 编码

§ 6.1 编码的目的

- ◇ 是使用选定的程序设计语言，把模块的过程描述翻译为用该语言书写的源程序（或源代码）。模块的过程描述——>源程序。
- ◇ 编码是设计的自然结果，程序的质量主要取决于设计的质量。

§ 6.2 编码的风格

一. 编码风格：

即程序设计风格，就是指作家、画家和程序员在创作中喜欢和习惯使用的表达自己作品题材的方式。

二. 编码风格的要求：

1. 实现源程序的文档化：

- 符号名（即标识符）的命名：名称应该能构反映其所代表的实际东西，具有一定的实际意义，使其能见名知意，有助于对程序功能的理解。
- 程序进行适当的注解：正确的注解能够帮助读者理解程序，可为后续阶段的测试和维护，提供正确的指导。

注释的位置及情况：

- 1) 每一程序单元开始处。（序号）
- 2) 重要的程序段。（嵌入源代码内部）
- 3) 难懂的程序段。（说明原因或画等效流程图）
- 4) 修改程序。（保持注释，代码一致性）

- 程序的视觉组织：使用标准的，统一的格式书写源程序清单，有助于改进可读性。

- 1) 用分层缩进的写法显示嵌套结构的层次。
- 2) 在注释段周围加上边框。

- 3) 在注释段与程序段以及不同程序段之间插入空行。
 - 4) 每行只写一条语句。
 - 5) 书写表达式时, 适当使用空格或圆括号等做隔离符。
2. 数据说明: 常量, 变量等的声明。
- 数据说明的次序应当规范化, 使数据属性容易查找, 有利于测试, 排错和维护。
 - ◆ 常量说明—>简单变量类型说明—>数组说明—>公用数据模块说明—>所有文件说明。
 - ◆ 整数量说明—>实型量说明—>字符型说明—>逻辑量说明
 - 当每个变量名用同一个语句说明时, 应将变量按字母顺序排列。
 - 如果设计了一个复杂数据结构, 应使用注释说明在程序实现时这个数据结构的特点。
3. 语句结构: 语句构造应力求简单、直接、不能为了片面的追求效率而使语句复杂化。
- 在一行内只写一条语句, 并采取适当的缩进方式, 使程序的逻辑和功能变得更加明确。
 - 程序编写应当首先考虑要清晰性, 不要刻意的追求技巧和效率, 而丧失清晰。
 - 首先要保证程序正确, 然后才要求提高速度。
 - 尽量使用公共过程或子程序去代替重复的功能代码段。
 - 使用括号来清晰地表达算术表达式和逻辑表达式的运算顺序。
 - 使用标准的控制结构, 有规律地使用 GOTO 语句。
 - 尽量减少使用“否定”条件的条件语句。(效率低且不易读)
 - 避免使用过于复杂的条件测试。
 - 避免过多的循环嵌套和条件嵌套。
 - 要模块化, 确保每个模块的独立性。
4. 输入和输出: 输入和输出的方式和格式应尽可能方便用户的使用, 尽量做到对用户友善。
- 对所有输入数据都进行检验。

- 检查输入项，重复组合的合法性。
- 保持输入格式简单。
- 使用数据结束标志，不必要求用户指定数据的数目。
- 明确提示交互式输入的请求，详细说明可用的选择和边界数目。
- 当程序设计语言对格式有严格要求时，应保持输入格式一致。
- 设计良好的输出报表。
- 给所有输出数据加标志，给出必要的说明。

§ 6.3 程序设计语言

一. 程序设计语言的发展分类：

1. 面向机器的语言：（机器语言，汇编语言）依赖于结构，其指令系统随机器而异，生产效率低，容易出错，难以维护。
2. 高级语言：使用的概念和符号与人们通常使用的比较接近，一条语句往往对应若干条机器指令，其特性不依赖于现实这种语言的计算机。

1) 从应用特点分类：

- 基础语言：如：BASIC, FORTRAN, COBOL, ALGOL 等，历史悠久，应用广泛。
- 结构化语言：具有为某种特殊应用而设计的独特的很强的过程能力和数据结构能力。如：PL/1, PASCAL, C, Ada 等。
- 专用语言：具有为某种特殊应用而设计的独特语法形式，应用范围较宽。

如：APL（数据和向量运算），BLISS（开发编译程序和操作系统），FORTH（开发微处理机软件），LISP 和 PROLOG（适合于人工智能领域）。

2) 从语言的内在特点分类：

- 系统实现语言：提供控制语句和变量类型检测等功能，同时允许程序员直接使用机器操作。如：C。
- 静态高级语言：提供某些控制语句和变量说明的机制，但程

序员不能直接控制由编译程序生成的机器操作，静态分配存储。如：COBOL、FORTH。

■ 块结构高级语言：提供有限的动态存储分配。如：ALGOL、PASCAL。

■ 动态高级语言：动态地完成所有存储管理，即执行个别语句可能分配或释放存储。如：某些专用语言。

3. 甚高级语言（4GL）：以数据或知识为基础，以对集合的处理代替对单个记录或元素的处理，能支持对大型数据库进行高效处理的机制。如：SQL。

二. 程序设计语言的选择：

考虑因素：

1. 应用领域。
2. 算法和计算的复杂性。
3. 软件执行环境。
4. 数据结构的复杂性。
5. 效率的考虑。
6. 用户的要求。

§ 6.4 小结

§ 6.5 补充实例

第七章 测试

§ 7.1 基本概念

一. 测试：

1. 测试：

是为了发现错误而执行程序的过程，即根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例，并利用这些测试用例去运行程序，以发现程序错误的过程。

2. 测试的目的：

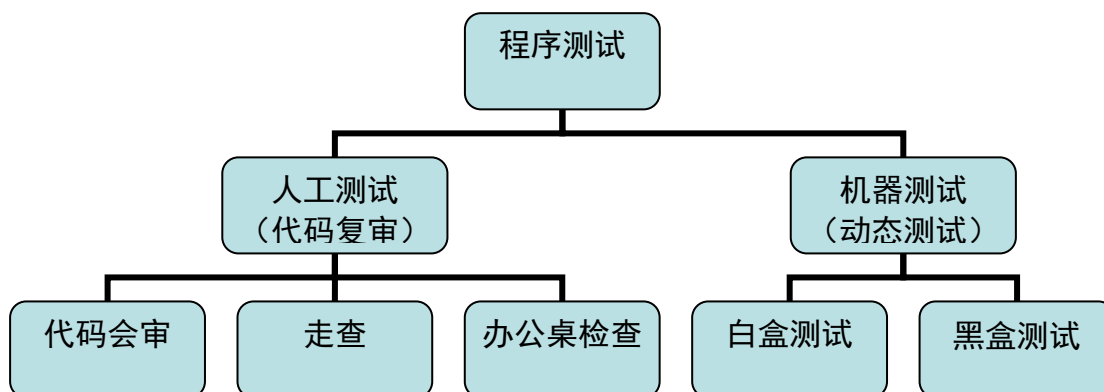
是发现程序中的错误（测试能证明错误的存在，而不能证明错误不存在）。

二. 纠错：

（调试），是为了确定错误的性质，并且加以纠正。

三. 测试方法分类与测试技术：

1. 测试方法分类：



2. 机器测试与人工测试：

- 机器测试：是在设定的测试数据上执行被测试程序的过程。
- 人工测试：指检查程序的静态结构，找出编译不能发现的错误。

3. 白盒测试与黑盒测试：

- 白盒测试：（结构测试）测试者对被测试程序的内部结构是清楚

的，从程序的逻辑结构入手按照一定的原则来设计测试用例，设定测试数据。

- 黑盒测试：（功能测试）测试者把被测试程序看成一个黑盒，完全用不着关心程序的内部结构，设计测试用例时，仅以程序的外部功能为根据，一方面检验程序能否完成一切应做的事情，另一方面要考察它能否拒绝一切不应该做的事情。

4. 代码会审、走查、办公桌检查：

- 代码会审：小组会，作者阅读讲解程序，其他人捕捉程序结构、功能与编码风格上可能存在的问题。
- 走查：提出“测试用例”，与会者扮演“计算机”角色，人工执行。
- 办公桌检查：一个人参加的代码会审。

四. 软件测试的步骤：

1. 测试步骤：单元测试，综合测试，确认测试、系统测试。（如图所示）

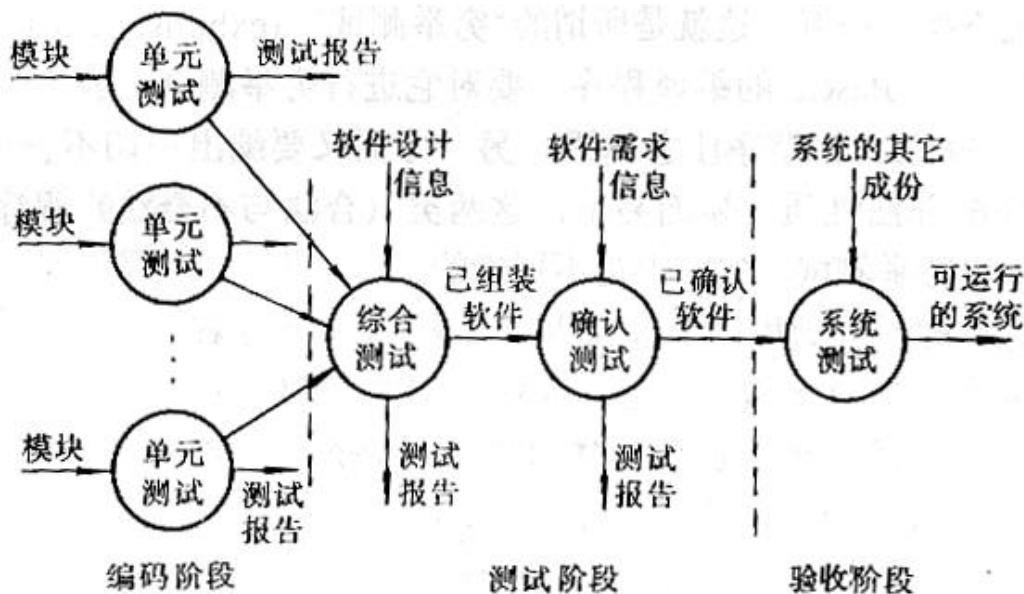


图 9.4 软件测试和步骤

2. 说明：

- 单元测试应该以模块为单位，并包括代码复审，动态测试等。
- 综合测试与确定测试应分别以软件的设计信息与需求信息为依据，确保在测试中分别达到上述信息所规定的要求。

- 确定测试用例时，单元测试可综合运用白盒与黑盒两类技术，其他测试主要采用黑盒测试技术。
- 各级测试均须事先制订测试计划，事后写出测试报告。
- 测试宜由独立的测试小组进行，避免由开发小组测试自己的程序。

§ 7.2 测试用例的设计

一. 测试用例：

是以发现错误为目的而精心设计的一组测试数据，测试用例 = {输入数据 + 期望结构}

二. 黑盒测试方法：以程序功能作为测试依据

1. 等价分类法：

把被测程序的输入域划分为若干各等价类，每个测试用例都代表一类与它等价的其他例子。

■ 步骤：

- 1) 划分等价类并给出定义（如果用这个例子来发现程序错误，则与其等价的其他例子一般也不会发现程序错误）。
- 2) 选择测试用例（原则：有效等价类的测试用例尽量公用，以期进一步减少测试次数；无效等价类必须每类一例，以防漏掉本来可能发现的错误）。

■ 举例：

某城市的电话号码由 3 部分组成，这 3 部分的名称和内容为：

地区码：空白或 3 位数字；

前 缀：非 ‘0’ 或 ‘1’ 开头的 3 位数字；

后 缀：4 位数字。

假定被测程序能接受一切符合上述规定的电话号码，拒绝所有不符合规定的号码，试用等价分类法设计它的测试用例。

第一步：划分等价类。

下表列出了划分的结果，包括 4 个有效等价类，11 个无

效等价类。在每一等价类之后均加有编号，以便识别

输入条件	有效等价类	无效等价类
地区码	空白①；3 位数字②	有非数字字符⑤；少于 3 位数字⑥；多于 3 位数字⑦；
前 缀	从 200 到 999 之间的 3 位数字③	有非数字字符⑧；起始位为 ‘0’ ⑨；起始位为 ‘1’ ⑩；少于 3 位数字⑪；多于 3 位数字⑫；
后 缀	4 位数字④	有非数字字符⑬；少于 4 位数字⑭；多于 4 位数字⑮；

第二步：确定测试用例。

上表中有 4 个有效等价类，可以公用以下两个测试用例：

测试数据	测试范围	期望结果
() 276—2345	等价类①、③、④	有效
(635) 805—9321	等价类②、③、④	有效

对 11 个无效等价类，应选择 11 个测试用例。例如前 3

个无效等价类可能使用下列 3 个测试用例：

测试数据	测试范围	期望结果
(20A) 423—4567	等价类⑤	无效
(33) 534—5678	等价类⑥	无效
(7777) 345—6789	等价类⑦	无效

2. 边界值分析法：

在等价分类法中，将代表一个类的测试数据选在等价类的边界上。（如：X≤400）。

3. 错误推测法：

猜测被测试程序中那些地方容易出错，并据此设计测试用例。

■ 说明：等价类和边界值法有线索可导，而猜错误则依赖测试人员的直觉和经验，仅作为辅助手段，即应首先用其他方法设计测试用例，再用猜错误来补充。

■ 举例：

当对一个排序程序进行测试时，可先用边界值分析法设计以下的测试用例：

- 1) 输入表为空表；
- 2) 输入表中仅有一个数据；
- 3) 输入表为满表；

然后再用猜错法补充以下的用例：

- 4) 输入表已经排好了序；
- 5) 输入表的排序恰好与所要求的顺序相反（例如：程序功能为由小到大排序，输入表为由大到小排序）；
- 6) 输入表中的所有数据全都相同；

等等。

4. 因果图法：

是借助图形（因果图）来设计测试用例的一种系统方法，特别适合于被测试程序有多种较入条件，程序输出又依赖于输入条件的多种组合的情况。

■ 举例：

某电力公司有 A、B、C、D 共 4 类收费标准，并规定，居民用电每月 100 度以下按 A 类收费，100 度及以上按 B 类收费。动力用电以每月 1 万度为分界。非高峰用电不足 1 万度按 B 类收费，达到 1 万度按 C 类收费。高峰用电万度以下为 C 类，达到或超过万度为 D 类。试用因果图法为该公司的电费计算程序设计一组测试用例。

以下列出产生设计用例的 4 点步骤：

- 1) 列出程序的输入条件（因）和输出动作（果），如图所示：

输入条件	输出动作
1. 居民用电	A. A 类计费
2. 动力用电	B. B 类计费
3. <100 度 / 月	C. C 类计费
4. <10,000 度 / 月	D. D 类计费
5. 高峰用电	

图 9.5 输入输出表

- 2) 用因果图表明输入和输出之间的逻辑关系，如图所示：

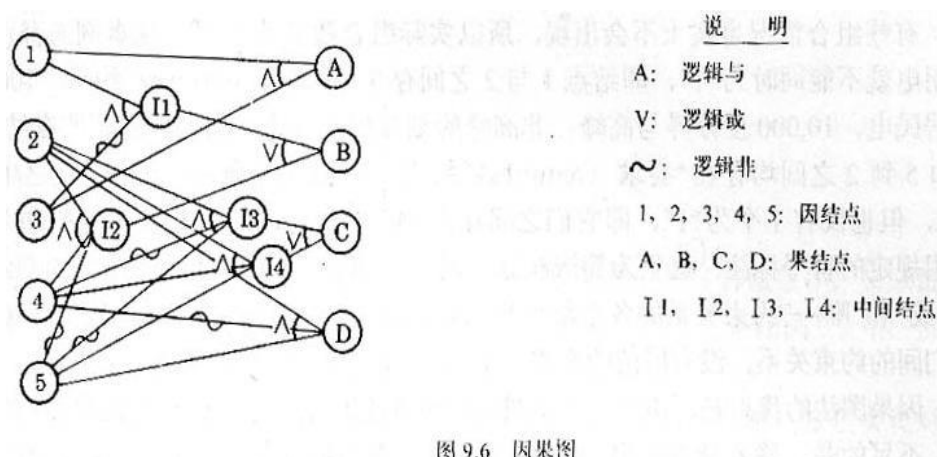


图 9.6 因果图

3) 把因果图转换为判定表。具体做法为:

- 选择一个输出动作，使处于“1”状态；
- 在因果图上从后向前回溯，找出使此动作为“1”的各种输入条件组合；
- 将每个输入条件组合填入判定表中的一列，同时填入在此组合情况下各个输出动作的状态；
- 选择下一个输出动作，重复以上 3 步，直至最后一个输出动作做完为止。

本例的判定表如图所示：表中因结点就是输入条件，果结点就是输出动作。

规 则		1	2	3	4	5	6
因 结 点	1	1	1	0	0	0	0
	2	0	0	1	1	1	1
	3	1	0				
	4			1	0	1	0
	5			0	0	1	1
中 间 结 点	I1		1				
	I2			1			
	I3				1		
	I4					1	
果 结 点	A	1	0	0	0	0	0
	B	0	1	1	0	0	0
	C	0	0	0	1	1	0
	D	0	0	0	0	0	1

图 9.7 从因果图导出的判定表

4) 为判定表中的每一列（或规则）设计一个测试用例，如图所示：

输入数据	预期结果
居民电，90 度/月	A
居民电，110 度/月	B
动力电，非高峰，8000 度/月	B
动力电，非高峰，1.2 万度/月	C
动力电，高峰，0.9 万度/月	C
动力电，高峰，1.1 万度/月	D

三. 白盒测试方法：以程序的内部逻辑作为依据

1. 逻辑覆盖法：

是对一系列测试过程的总称，这组测试过程逐渐进行越来越完整的通路测试。

■ 测试过程分类：

- 1) 语句覆盖：使被测试程序的每条语句至少执行一次。
- 2) 判定覆盖：使被测试程序的每一分支都至少执行一次。
- 3) 条件覆盖：要求判定中的每个条件都按“真”“假”两种结果至少执行一次。
- 4) 判定/条件覆盖：要求判定中的每个条件都取到各种可能的值，而且每个判定表达式也都要取到各种可能的结果。
- 5) 条件组合覆盖：要求判定中每个条件的各种可能组合都至少出现一次。

■ 举例：

下图显示了某程序的逻辑结构。试为它设计足够的测试用例，分别实现对程序的：判定覆盖、条件覆盖、条件组合覆盖
程序结构：（如图所示）

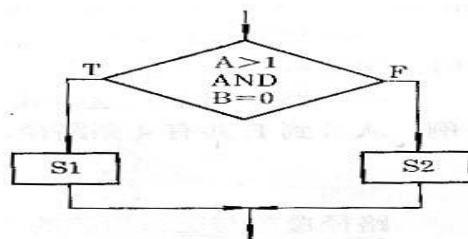


图 9.9 被测程序流程图

测试用例：（如图所示）

表 9.4 不同覆盖标准所需的测试用例

覆盖种类	需满足的条件	测试数据	期望结果
判定覆盖	① $A > 1, B = 0$; ② $\begin{cases} A > 1, B \neq 0 \text{ 或} \\ A \leq 1, B = 0 \text{ 或} \\ A \leq 1, B \neq 0 \end{cases}$	① $A = 2, B = 0$; ② $\begin{cases} A = 2, B = 1 \text{ 或} \\ A = 1, B = 0 \text{ 或} \\ A = 1, B = 1 \end{cases}$	执行 S1 执行 S2
条件覆盖	以下 4 种情况各出现一次: $A > 1; B = 0$; $A \leq 1; B \neq 0$	① $A = 2, B = 0$; ② $A = 1, B = 1$	执行 S1 执行 S2
条件组合覆盖	① $A > 1, B = 0$; ② $A > 1, B \neq 0$; ③ $A \leq 1, B = 0$; ④ $A \leq 1, B \neq 0$	① $A = 2, B = 0$; ② $A = 2, B = 1$; ③ $A = 1, B = 0$; ④ $A = 1, B = 1$	执行 S1 执行 S2 执行 S2 执行 S2

2. 路径测试法：

是借助程序流程图设计测试用例的一种白盒测试方法。

■ 测试过程分类：

- 1) 结点覆盖：程序的测试路径至少经过程序图中的每个结点一次。
- 2) 边覆盖：程序的测试路径至少经过程序图中每条边一次。
- 3) 路径覆盖：要求程序图中每条路径都至少经过一次。

■ 举例：

下图为某程序的简单程序图，试写出路径测试法的覆盖标准。

程序结构：（如图所示）

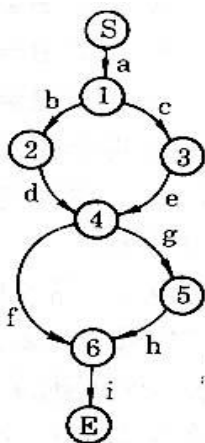


图 9.10 被测程序的程序图

覆盖标准：

- 1) 结点覆盖：abdghi、aceghi；
- 2) 边覆盖：abdfi、aceghi；
- 3) 路径覆盖：abdfi、abdghi、aceghi、acefi。

2. 路径覆盖与逻辑覆盖的区别：

后者着眼于每个单独的判定结点，而前者考察的是整个路径，把路径覆盖与条件组合覆盖结合起来，便可实现查错能力最强的白盒测试。

四. 测试设计策略和设计举例：

1. 测试设计策略：

■ 在综合测试及其后的测试阶段，采用黑盒测试方法，策略包括：

- 1) 用等价分类法和（或）边值分析法提出基本的测试用例。
- 2) 用猜错法补充新的测试用例。
- 3) 如果程序的功能说明中含有输入条件的组合，宜在一开始就用因果图法，然后再按 1)、2) 两步进行。

■ 单元测试的策略是把白盒法与黑盒法结合运用。

2. 设计举例：（书：P148）

某三角形程序的功能为：读入代表三角形边长的 3 个整数，判断它们能否组成三角形。如果能够，则输出三角形是等边、等腰或任意三角形的识别信息。试为此程序设计一组测试用例。（本例将先用黑盒法设计测试用例，然后用白盒法进行检验与补充）

程序结构：（如图所示）

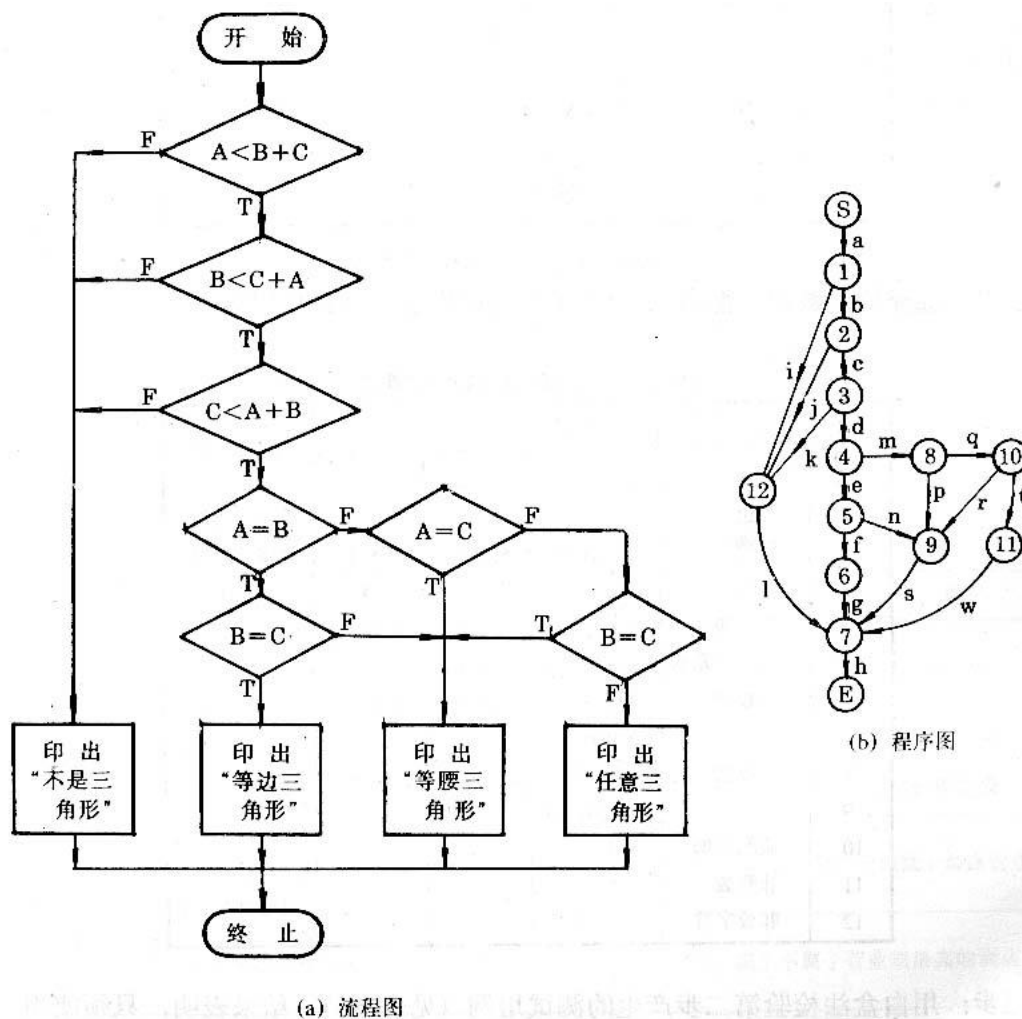


图 9.11 三角形程序的流程图与程序图

第一步：运用等价分类法划分与定义等价类，然后用边值法和猜错法补充。（如图所示）

等价分类法		
有效等价类		无效等价类
输入 3 个正整数:		含有零数据 ⑨
3 数相等 ①		含有负整数 ⑩
3 数中有 2 数相等 { A、B 相等 ② B、C 相等 ③ C、A 相等 ④		少于 3 个整数 ⑪
3 数均不相等 ⑤		含有非整数 ⑫
2 数之和不大于第 3 数 { 最大数为 A ⑥ 最大数为 B ⑦ 最大数为 C ⑧		含有非数字 ⑬
边值法:	2 数之和等于第 3 数 ⑭	
猜错法:	输入 3 个零 ⑮	
	输入 3 个负数 ⑯	

图 9.12 用黑盒法分析程序的输入

第二步：选择测试数据，得出 22 个基本的测试用例。（如图所示）

表9.5 三角形程序的测试用例

序号	测试内容	测试数据			期望结果
		a	b	c	
1	等边	5, 5, 5			等边三角形
2	等腰	4, 4, 5	5, 4, 4	4, 5, 4	等腰三角形
3	任意	3, 4, 5			任意三角形
4	非三角形	9, 4, 4	4, 9, 4	4, 4, 9	} 不是三角形
5	退化三角形	8, 4, 4	4, 8, 4	4, 4, 8	
6	零数据	0, 4, 5	4, 0, 5	4, 5, 0	
7		0, 0, 0			
8	负数据	-3, 4, 5	3, -4, 5	3, 4, -5	
9		-3, -4, -5			} 运行出错 (类型不符)
10	遗漏数据	3, 4, —			
11	非整数	3, 3, 4, 5			
12	非数字符	A, 4, 5			

第三步：用白盒法检验第二步产生的测试用例（如图所示）。结果表明，只须使用 22 个例子中的前 8 个，就能满足程序图的完全覆盖。可见对于本例来讲，用黑盒法设计的测试用例已经足够用，不必再进行补充。

表 9.6 被测试数据覆盖的边和结点

序号	测试数据	覆盖结点	覆盖的边
1	5, 5, 5	1, 2, 3, 4, 5, 6, 7	abcdefgh
2a	4, 4, 5	1, 2, 3, 4, 5, 9, 7	abcdensh
2b	5, 4, 4	1, 2, 3, 4, 8, 10, 9, 7	abcdmqrsh
2c	4, 5, 4	1, 2, 3, 4, 8, 9, 7	abcdmpsh
3	3, 4, 5	1, 2, 3, 4, 8, 10, 11, 7	abcdmqtw
4a	9, 4, 4	1, 12, 7	ailh
4b	4, 9, 4	1, 2, 12, 7	abjlh
4c	4, 4, 9	1, 2, 3, 12, 7	abcklh

§ 7.3 单元测试

一. 测试的目的:

考察模块的接口和内部结构, 看他们是否符合模块功能说明的需求。

二. 测试的重点:

1. 模块的接口
2. 局部数据结构
3. 重要的执行路径
4. 出错处理路径
5. 影响以上多项的边界条件

三. 单元测试的步骤: (在编码阶段进行)

1. 编译(汇编)模块: 发现语法错误。
2. 静态分析: 用专用的软件测试工具, 发现程序结构、功能、编码标准与风格方面的问题。
3. 代码复审: 人工检查。
4. 动态测试: 白盒测试, 黑盒测试。
5. 单元测试报告: 内容包括静态分析, 白盒测试, 黑盒测试三个方面的项目与结果。

四. 驱动模块和桩模块:

1. 驱动模块和桩模块:

在单元测试时, 为测模块编写一些测试模块, 作为它的上级或下级模块的替身, 代替上级模块的称为驱动模块, 代替下级模块的称为桩模块, 替身模块应该是真实模块的简化, 只须模拟与被测试模块直接有关的一部分功能。

2. 举例: (略)

§ 7.4 综合测试

一. 综合测试: (集成测试)

是将模块组装成程序的过程中所进行的测试，其主要目标是发现与接口有关的问题。

二. 综合测试发现的错误：

1. 不正确的接口。
2. 因存取全局（公用）数据引起的块间干扰。
3. 不一致的文件与数据结构。
4. 不适合的模块调用顺序。
5. 出错处理上的错误。

三. 综合测试的测试技术和实施策略：

综合测试通常采用黑盒测试技术、其实施策略分为非渐增式和渐增式两种

1. 非渐增式测试：一次就把通过了单元测试的所有模块组装起来，进行全程序的测试，出了问题很难进行错误定位。
2. 自顶向下测试：（渐增式），测试时从顶层模块开始，沿被测程序的结构图逐步下移，每次只增加一个新的模块。

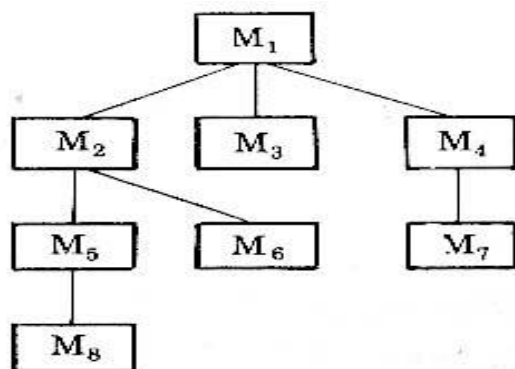


图 9.18

■ 先深度后广度方法：（例：M1-M2-M5-M8-M6-M3-M4-M7）

■ 先广度后深度方法：（例：M1-M2-M3-M4-M5-M6-M7-M8）

■ 特点：

- 1) 能较早的显示出程序的轮廓
- 2) 由顶向下的组装顺序，保证任何模块加进程前，其上级模块已先它装入，所以模块的驱动可以利用真实模块，只须编写桩模块供测试之用。

- 3) 上层模块得到更多的测试机会，使被测程序获得更为彻底的检验。
3. 自底向上测试：(渐增式)，模块组装顺序采取由下向上的路线。
 - 测试步骤：
 - 1) 从程序的较低层中找一个叶模块，由下向上地逐步增加新模块，组成程序的一个子程序或具有某一功能的模块“群”。
 - 2) 从另一子系统或群中选择另一个模块，仿照 1) 组成又一个子系统。
 - 3) 重复第 2) 步，得出所有子系统，然后组装成程序。
 - 特点：
 - 1) 不能在测试的早期显示出程序的轮廓。
 - 2) 测试软件只需要驱动模块，不需要桩模块。
4. 混合测试：是自顶向下与自底向下测试方法的结合。

§ 7.5 高级测试

一. 确认测试：

1. 目的：确定所开发的软件是否符合软件需求规格说明书的要求。
2. 内容：
 - 功能测试：根据 SRS 中的功能要求，找出尚未实现的功能要求。
 - 性能测试：测试程序执行时的响应时间，处理速度，占用内存和外存的容量。以及通道传输能力等是否达到规定的目标。
 - 强度测试：检查程序对强负荷的承受能力。
 - 对文档配置的复审：查明最后通过的程序是否配齐了应有的文档，且文档内容是否与程序完全一致。

二. 系统测试：

把新开发的软件安装到系统中，检查它能否与系统的其余部分协调运行。

§ 7.6 纠错（调试）

一. 纠错的方法：

1. 跟踪法：

- 反向跟踪：从发现错误的地方开始，逐步向后面溯，直至找出错误根源。
- 正向跟踪：查错时沿着程序的控制流，向前跟踪每条语句的执行情况，找到最先出现错误或异常的地方，进行分析、纠错。

2. 演绎法：

根据错误症状，列举出所有可能原因，通过仔细的分析，将其中根据不是或不会发生的病因排除，最后证明剩下的原因确实是错误根源。

3. 归纳法：

从错误征兆出发，分析他们相互间的关系，找出规律，从而总结出错误原因。

4. 试凑法：

对结构比较简单的程序，根据对错误征兆的分析，设定一个可疑区，采取一些简单的纠错手段，获取可疑区的有关信息，从而进行确定错误位置。

二. 纠错的辅助手段：

1. 打印语句：插入打印语句，动态输出变量结果。
2. 调式语句或调试程序：单步调试，断点设置等。
3. 转储命令和程序：输出存储器指定区域的全部内容。

三. 纠错的原则：

1. 重在思考
2. 避免紧张
3. 修改要彻底
4. 防止纠错引入新的错误

§ 7.7 软件可靠性

§ 7.8 测试文档：《测试分析报告》

一. 引言：

二. 测试项目说明：

1. 测试项目名称及测试内容

2. 测试用例：

- 输入：输入的数据和输入的命令

- 输出：预期的输出数据

- 步骤及操作：

- 允许偏差：给出实测结果与预期结果之间允许偏差的范围

三. 测试执行情况：

1. 测试项目

2. 测试结果：

- 实测结果数据

- 与预期结果的偏差

- 该项测试表明的事实

- 该项测试发现的问题

四. 评价：

1. 软件能力

2. 缺陷和限制

3. 建议

4. 测试结论：是否通过

§ 7.9 小结

§ 7.10 补充实例

第八章 维护

§ 8.1 软件维护

一. 软件维护的概念与目的:

1. 软件维护: 是在软件已经交付使用之后, 为了改正错误或满足新的需要而修改软件的过程。
2. 维护的目的: 是满足用户对已开发产品的性能与软件环境不断提高的需求, 进而达到延长软件的寿命。

二. 软件维护的分类:

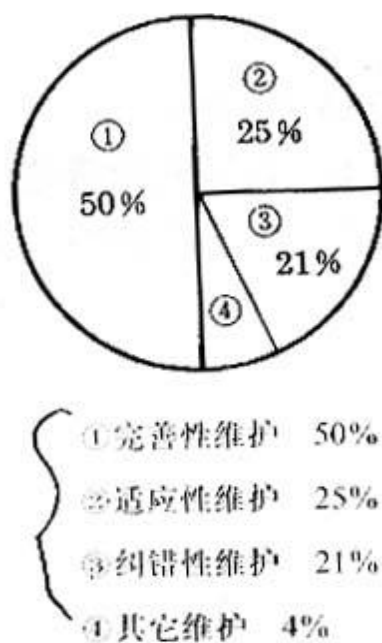


图 10.1 各类维护的比重

1. 完善性维护: 在软件使用过程中, 为了满足用户对软件的功能与性能提出新的需求而进行的维护。(50%)
2. 适应性维护: 使软件适应运行环境(包括软, 硬件环境及数据环境)的变化而进行的维护。(25%)
3. 纠错性维护: 为纠正在开发期间未能发现的错误而进行的维护。(21%)
4. 其他维护: (如: 预防性维护)为改善软件的可维护性, 可靠性等,

以减少今后对其进行维护所需的工作量的工作而进行的维护。(4%)

§ 8.2 可维护性

一. 可维护性:

是指纠正软件系统出现的错误或缺陷, 以及为满足新的要求进行修改, 扩充或压缩的容易程度, 即衡量维护容易程度的一种属性。

二. 影响可维护性的软件属性:

1. 可理解性: 指人们通过阅读源代码和相关文档。了解程序功能及如何运行的容易程度。
2. 可修改性: 指程序容易修改的程序, 可修改性好的程序, 在修改时出错的概率也小一些。
3. 可测试性: 指程序容易被测试的程序。

三. 可维护性的定量度量:

用维护过程中各种活动所耗费的时间来定量度量。

1. 问题识别时间: 确定维护的目标, 弄清要解决的问题。(哪类维护)。
2. 管理延迟时间: 管理部门审批的时间。
3. 收集维护工具时间: 收集工具, 为维护收好准备。
4. 问题分析时间。
5. 修改规格说明书的时间。
6. 改正(或修改)的时间。
7. 局部测试时间。
8. 整体测试时间。
9. 维护复审时间。
10. 分发与恢复时间。

四. 提高可维护性的途径:

1. 提供完整和一致的文档。
2. 采用现代化的开发方法。

§ 8.3 小结

§ 8.4 补充实例

第九章 面向对象分析与设计

§ 9.1 引论

一. 传统的生命周期方法学：

1. 存在的问题：

- 生产率提高的幅度远不能满足需要。
- 软件重用程度很低。
- 软件仍然很难维护。
- 软件往往不能正确满足用户需要。

2. 出现问题的原因：

- 僵化的瀑布模型。
 - 1) 某些类型的系统需求是模糊的。
 - 2) 项目参与者之间存在通信鸿沟。(文档是被动的，静态的通信工具，通过它难于理解一个动态系统。)
 - 3) 预先定义的需求可能是过时的。
- 结构化技术的缺点。
 - 1) 结构分析与技术的本质是功能分析，而需求变化大多是针对功能的
 - 2) 结构分析与设计清楚地定义了目标系统边界，难于扩展。
 - 3) 处理分解带有任意性，软件可重用性差。

二. 面向对象方法学：

面向对象方法是一种运用对象、类、继承、封装、聚合、消息传递、多态性等概念来构造系统的软件开发方法，其基本思想是从现实世界中客观存在的事物出发来构造软件系统，并在系统构造中尽可能运用人类的自然思维方式，即，使问题域和求解域在结构上尽可能一致：

OO = Objects + Classes + Inheritance + Communication With message

特点：

1. 从问题域中客观存在的事物出发来构造软件系统，用对象作为对这些事物的抽象表示，并以此作为系统的基本构成单位。
2. 事物的静态特征（即可以用一些数据来表达的特征）用对象的属性表示，事物的动态特征（即事物的行为）用对象的服务（或操作）表示。
3. 对象的属性与服务结合为一个独立的实体，对外屏蔽其内部细节，称作封装。
4. 把具有相同属性和相同服务的对象归为一类，类是这些对象的抽象描述，每个对象是它的类的一个实例。
5. 通过在不同程度上运用抽象的原则，可以得到较一般的类和较特殊的类。特殊类继承一般类的属性与服务，面向对象方法支持对这种继承关系的描述与实现，从而简化系统的构造过程及其文档。
6. 复杂的对象可以用简单的对象作为其构成部分，称作聚合。
7. 对象之间通过消息进行通信，以实现对象之间的动态联系。
8. 通过关联表达对象之间的静态关系。

主要优点：

1. 与人类习惯的思维方法一致。
2. 稳定性好。(以对象为中心，而不是依赖于系统功能)
3. 可重用性好。(对象类的继承)
4. 可维护性好：
 - 稳定性好
 - 容易修改
 - 容易理解
 - 易于测试和调试

三. 面向对象的软件工程方法：

面向对象的软件工程方法是面向对象方法在软件工程领域的全面运用。它包括面向对象的分析（OOA）、面向对象的设计（OOD）、面向对象的编程（OOP）、面向对象的测试（OOT）和面向对象的软件维护等主要内容。

§ 9.2 基本概念

一. 面向对象的基本概念：

面向对象=对象+类+继承+通信（Coad 和 Yourdon 给出的简单定义）。

1. 面向对象技术的基本观点：

- 1) 客观世界由对象组成，任何客观事物都是对象，复杂对象可以由简单对象组成。
- 2) 具有相同数据和操作的对象可归纳成类，对象是类的一个实例。
- 3) 类可以派生出子类，子类除了继承父类的全部特性外还可以有自己的特性。
- 4) 对象之间的联系通过消息传递来维系。

由于类的封装性，它具有某些对外界不可见的属性，这些数据只能通过消息请求调用可见的方法来访问。

2. 面向对象方法的基本出发点就是尽可能地按照人类认识世界的方法和思维方式来分析和解决问题，使人们分析、设计一个系统的方法尽可能接近认识一个系统的方法。

二. 面向对象的核心元素：

1. 对象：

客观世界里的任何实体都可以被称之为对象，对象可以是具体的有形的物，如：人、汽车等；也可以是无形的事物或概念，如：抽象的规则、计划或事件。

2. 封装：

是面向对象方法的一个重要原则。是指把属性和操作封进一个对象里，它的内部信息对外界隐藏，不允许外界直接存取对象的属性，只能通过对象提供的有限的接口对对象的属性数据进行操作。对于外界来说，只能知晓对象的外部行为而无法了解对象行为的内部实现细节，这样可以保证对象内部属性数据的安全性。

封装有两层含义：

- i. 结合性，即把对象的全部属性和方法结合起来，形成一个独立的不可分割的单位；
- ii. 信息隐藏性，即尽可能隐蔽对象的内部细节，对外形成一个边界，只保留有限的对外接口使之与外部发生联系。

3. 消息

消息就是向对象发出的请求，一个消息包含消息名、接收对象的标志、服务标志、输入标志、输入信息、回答信息等。

4. 类

类是对象的抽象，类好比是一个对象模板。

5. 继承

“泛化”

6. 多态性

同一个操作作用于不同的对象，可以有不同的解释，产生不同的执行结果，即在程序运行的时候，根据对外部触发进行反应的对象的不同，可以动态的选择某个操作的实现方法，这就是多态性。

1) 多态性分为如下两种：

- 编译时的多态性。编译时的多态性是通过重载来实现的。系统在编译时，根据传递的参数、返回的类型等信息决定实现何种操作。
- 运行时的多态性。运行时的多态性就是指直到系统运行时，才根据实际情况决定实现何种操作。

2) 多态性大致有以下三种表现方式：

- 通过接口实现多态性。
- 通过继承实现多态性。
- 通过抽象类实现多态性。

7. 结构与连接

1) 为了使系统能够有效地映射问题域，系统开发者需认识并描述对象之间的以下几种关系：

- 对象的分类关系。
- 对象之间的组成关系。
- 对象属性之间的静态联系。
- 对象行为之间的动态联系。

2) 面向对象方法分别用以下几种结构和连接来反映对象之间的几种关系：

- 一般/特殊结构。
- 整体/部分结构。
- 实例连接。
- 消息连接。

§ 9.3 面向对象建模

一. 三种模型：

1. 对象模型：描述系统的数据结构
2. 动态模型：描述系统的控制结构
3. 功能模型：描述系统功能

二. 对象模型：

表示静态的，结构化的系统的“数据”性质，是对模拟客观世界实体的对象以及对象彼此间的关系的时，描述了系统的静态结构。

图形符号：

1. 类与对象：（类—&—对象、类）

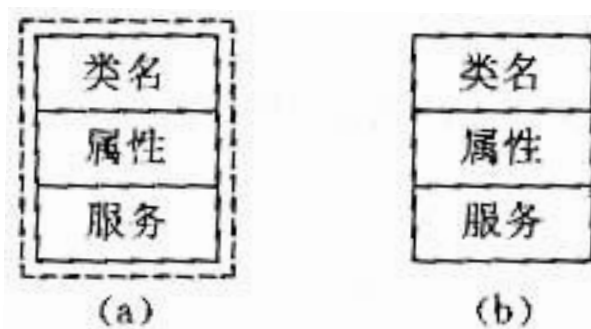


图 9.4 表示符号

2. 结构：

- 归纳关系：“一般—特殊”，继承关系。

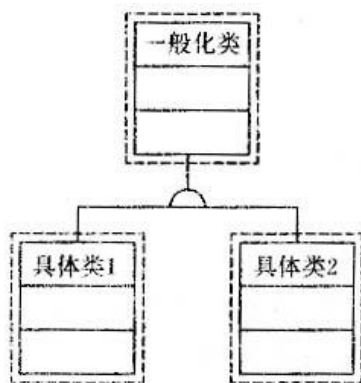


图 9.5 表示归纳关系的图形符号

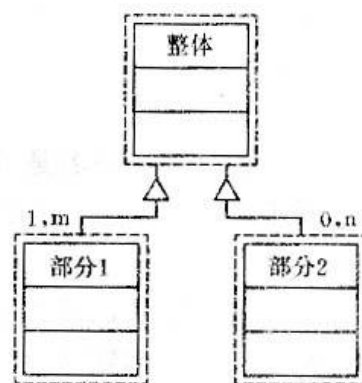


图 9.6 表示组合关系的图形符号

- 组合关系：“整体—部分”，构成关系。

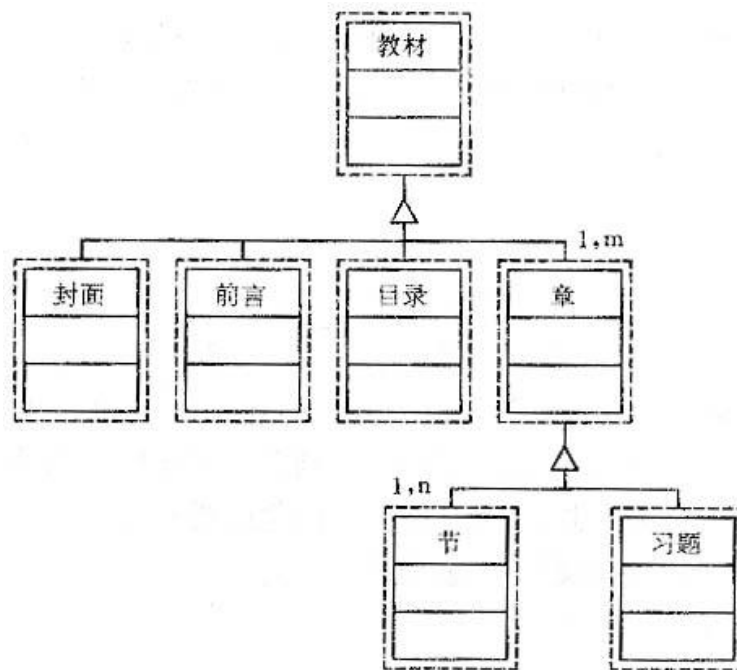


图 9.7 描绘教材结构的聚集树

- 关联关系：反映对象之间相互依赖，相互作用的关系。

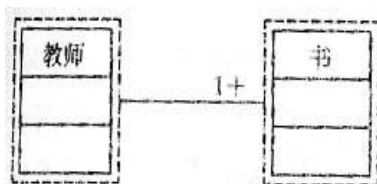


图 9.8 教师与属于他的书之间的关联关系

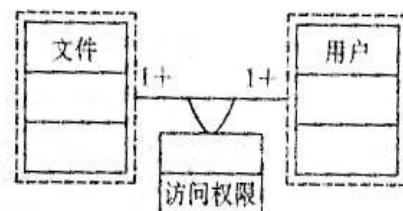


图 9.9 链属性的表示方法

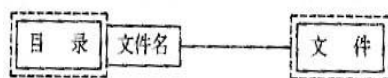


图 9.10 一个受限的关联

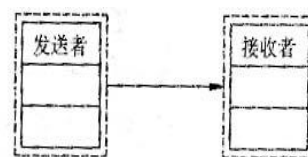


图 9.11 消息连接的表示符号

三. 动态模型：

表示瞬时的，行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。

1. 术语：

- 事件：是某个特定时刻发生的事情，是引起对象状态转换的控制信息。

- 状态：是对象在其生命周期中的某个特定阶段所处的某种情形。
- 行为：是指对象达到某种状态时所做的一系列处理操作。

2. 表示方法：状态图。

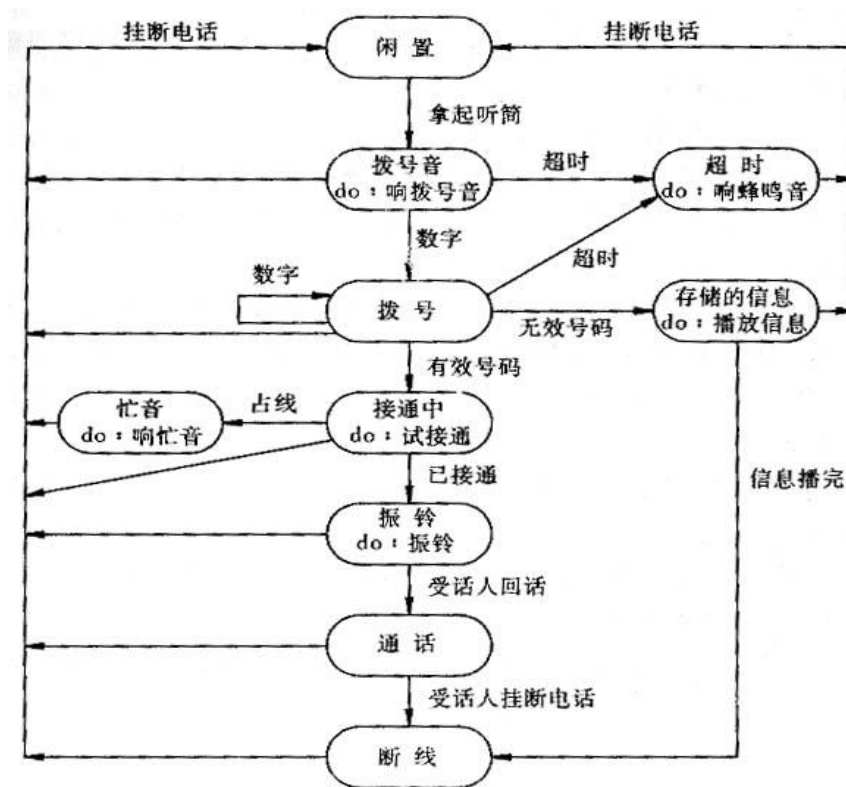
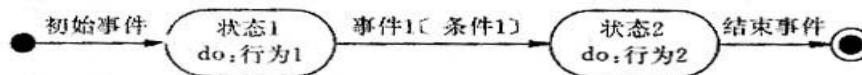


图 9.15 电话的状态图

四. 功能模型：

表示变化的系统的“功能”性质，指明了系统应该“作什么”。

表示方法：数据流图。

五. 三种模型之间的关系：

功能模型指明了系统应该“作什么”；动态模型规定了什么时候（即在何种状态下接受了什么事件的触发）做；对象模型则定义了做事情的实体。

1. 动态模型描述了类实例的生命周期或运行周期。
2. “行为—处理—服务”相对应。
3. “数据存储、源点、终点—对象”相对性。

4. “数据流—属性或对象”相对应。
5. 功能模型中的处理可能产生动态模型中的事件。
6. 对象模型描述了功能模型中的动作对象，数据存储及数据流的结构

§ 9.4 面向对象的分析

面向对象分析（Object-Oriented Analysis），简称 OOA,是指利用面向对象的概念和方法为软件需求建造模型，以使用户需求逐步精确化、一致化、完全化的分析过程。

1. 面向对象分析模型：

面向对象分析中建造的模型主要有对象模型、动态模型和功能模型。面向对象分析的关键是识别出问题领域内的对象，在分析它们之间的相互关系之后建立起问题领域的简洁、精确和可理解模型。

2. 面向对象分析的基本过程：

- 1) 问题论域分析：业务范围、业务规则、业务处理过程、系统责任范围、边界。
- 2) 发现和定义对象分类及内部特征：属性和服务。
- 3) 识别对象的外部联系：一般特殊、整体与部分，实例连接，消息连接。
- 4) 建立系统的静态结构模型：绘制对象类图和对象图，系统与子系统结构图等，编制文档。
- 5) 定义用例，建立系统的动态行为模型：分析系统行为，建立动态模型，并用图形和文字表达。
- 6) 建立详细说明。
- 7) 原型开发。

3. 面向对象分析的基本原则：

1) 抽象原则：

面向对象分析方法中的类就是抽象得到的：系统中的对象使对现实世界中的事物的抽象；类是对对象的抽象；一般类是对特殊类的进一步抽象；属性是事物静态特征的抽象；服务是事物动态特征的抽象。

2) 分类原则：

分类就是把具有相同属性和服务的对象划分为一类，用类作为这些对象的抽象描述。分类原则实际上是抽象原则运用于对象

描述时的一种表现形式，通过不同程度的抽象可以形成一般/特殊结构。

3) 聚合原则：

聚合是把一个复杂的事物看出若干简单事物的组合体，从而简化对复杂事物的描述。在面向对象分析中运用聚合原则将一个较复杂的事物划分为几个组成部分，分别用整体和部分进行描述，这样形成的整体/部分结构不仅能清楚地表达事物的组成关系，还可以简化分析过程。

4) 关联原则：

关联是人类思考问题时常用的方法，通过一个事物可以联想到另外的事物，产生联想的原因是事物之间存在着某些联系。在面向对象的分析过程中运用关联原则可以在系统模型中明确地表示对象之间的静态联系。

5) 消息通信原则：

这一原则要求对象之间只能通过消息进行通信，而不允许在对象外直接地存取对象内部的属性。

§ 9.5 面向对象的设计

分析是提取和整理用户需求，并建立问题域精确模型的过程；设计则是把分析阶段得到的需求转变成符合成本和质量要求的、抽象的系统实现方案的过程。从面向对象分析到面向对象的设计是一个逐步扩充模型的过程，也可以说面向对象设计是用面向对象观点建立求解域模型的过程。

1. 面向对象设计的模型

在设计期间主要扩充 4 个组成部分：

1) 人机交互部分

人机交互部分包括有效的人机交互所必需的实际显示和输入。

2) 问题域

问题域部分放置面向对象分析的结果并管理面向对象分析的某些类和对象、结构、属性和方法。

3) 任务管理

任务管理部分包括任务定义、通信和协调、硬件分配及外部系统。

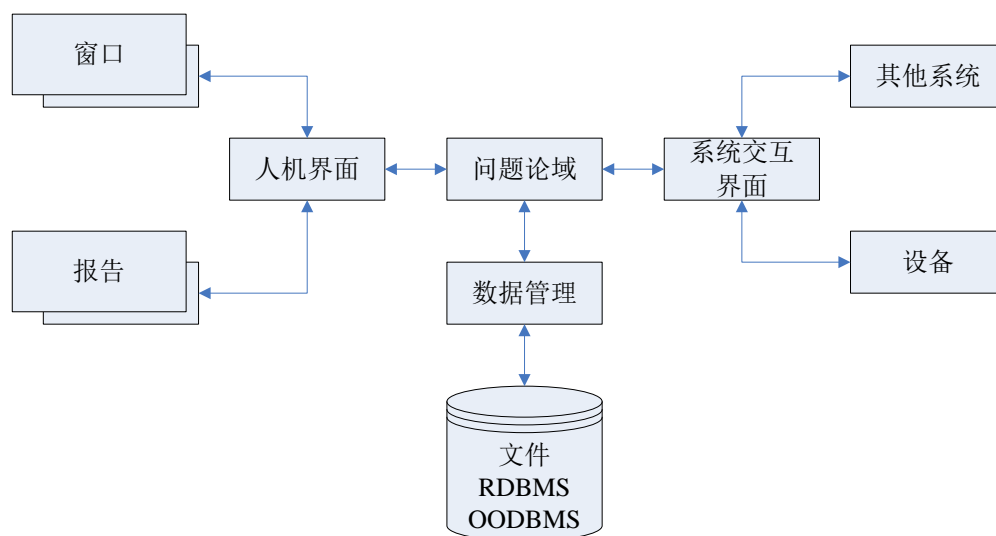
4) 数据管理。

数据库管理部分包括对永久性数据的访问和管理。

2. 面向对象的高层设计

高层设计的主要任务就是设计前面提到的 4 个部分：问题域部分、人机交互部分、任务管理部分和数据管理部分。

高层设计的结构模型如图所示：



1) 问题域子系统的设计

问题域子系统应该包括域应用问题直接有关的全部类和对象。识别和定义这些类和对象的工作在面向对象分析中已经开始，在面向对象分析阶段得到的有关应用的概念模型描述了解决的问题。在面向对象设计阶段，将继续面向对象分析阶段的工作，对分析阶段的结果进行修改和增补，对分析时构造的模型中的某些类与对象、结构、属性、操作进行组合与分解。

问题域子系统的设计工作主要有以下几个方面：

- 复用已有的设计
- 把与问题论域相关的类关联，建立类的层次结构
- 创建一般化类
- 改进系统性能
- 加入较低层的构件

2) 人机交互子系统的设计

在设计阶段必须根据需求把交互细节加入到用户界面设计中，包括人机交互所必需的实际显示和输入。

用户界面部分设计主要由以下几个方面组成：

- 用户分类。
- 描述人及其任务的脚本。
- 设计命令层。
- 设计与用户的详细交互。

用户界面详细设计要尽量做到如下几点：

- ◆ 采用一致的术语、一致的步骤和一致的活动；
- ◆ 减少用户敲键和鼠标点击的次数，减少完成某件事所需要的下拉菜单的距离；
- ◆ 当用户等待系统完成一个活动时，要提供一些反馈信息；
- ◆ 在操作出现错误时，要全部或部分恢复到原来的状态；
- ◆ 采取符合人类习惯图形界面。

- 继续进行原型设计。
- 设计人机交互类。

3) 任务管理子系统的设计

任务是进程的别称，是执行一系列活动的一段程序。当系统中有许多并发行为时，需要依照各个行为的协调和通信关系，划分各种任务，以简化并发行为的设计和编码。

定义任务的工作主要包括如以下几个方面：

- 为任务命名，并简要说明这个问题。
- 定义各个任务如何协调工作，指出它是事件的驱动还是时钟驱动。
- 定义各个任务之间如何通信，任务将从哪里取值，任务执行得到的结果将送往何方。

4) 数据管理子系统的设计

数据管理部分提供了在数据管理系统中存储和检索的对象的基本结构，包括对永久性数据访问和管理。

3. 面向对象的类设计

在面向对象的分析过程中，对问题论域所需的基本类进行了模型化和抽象，但在系统的最终实现应用时只有这些类是不够的，还要根据需要追加一些类。追加辅助类的工作在类设计的过程中完成。

§ 9.6 面向对象的实现

§ 9.7 UML 统一建模语言

一. UML:

UML (Unified Modeling Language), 即统一建模语言, 是一个通用的可视化建模语言, 用于对软件进行描述、可视化处理、构造和建立软件系统制品的文档。UML 是一种绘制软件蓝图的标准语言, 它记录了对必须构造的系统的决定和理解, 可用于对系统的理解、设计、浏览、配置、维护和信息控制。UML 适用于各种软件开发方法、软件生命周期的各个阶段、各种应用领域以及各种开发工具, 是一种总结了以往建模技术的经验并吸收当今优秀成果的标准建模方法。

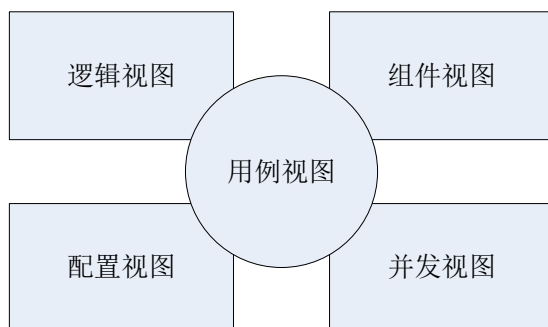
二. UML 的构成:

总体来说, UML 由以下几个部分构成:

1. 视图:

视图是表达系统的某一方面特征的 UML 建模元素的子集, 视图并不是图, 它是由一个或者多个图组成的对系统某个角度的抽象。

UML 中的视图大致分为 5 种: (如图所示)



1) 用例视图:

用例视图强调从系统的外部参与者 (主要是用户) 角度看到的或需要的系统功能。描述系统应该具备的功能。用例模型的用途是列出系统中的用例和参与者, 并显示哪个参与者参与哪个用例的执行。用例视图是其他视图的核心, 它的内容直接驱动其他视图的开发。系统要提供的功能都是在用例视图中描述的, 用例视图的修改会对所有其他的视图产生影响。此外, 通过测试用例视图, 还可以检验和最终校验系统。

2) 逻辑视图:

逻辑视图从系统的静态结构和动态行为的角度显示如何实现系统的功能。描述用例视图中提出的系统功能的实现。与用例视图相比，逻辑视图主要关注系统内部，它既描述系统的静态结构（类、对象以及它们之间的关系），也描述系统内部的动态协作关系。系统的静态结构在类图和对象图中进行描述，而动态模型则在状态图、顺序图、协作图以及活动图中进行描述。逻辑视图的使用者主要是设计人员和开发人员。

3) 并发视图:

并发视图。显示系统的并发性，解决在并发系统中存在的通信和同步问题。并发视图主要考虑资源的有效利用、代码的并发执行以及系统环境中异步事件的处理。除了将系统划分为并发执行的控制以外，并发视图还需要处理线程之间的通信和同步。并发视图的使用者是开发人员和系统集成人员。并发视图由状态图、协作图，以及活动图组成。

4) 组件视图:

组件视图显示代码组件的组织结构。组件视图描述系统的实现模块以及它们之间的依赖关系。组件视图主要由组件图构成，它的使用者主要是开发人员。

5) 配置视图:

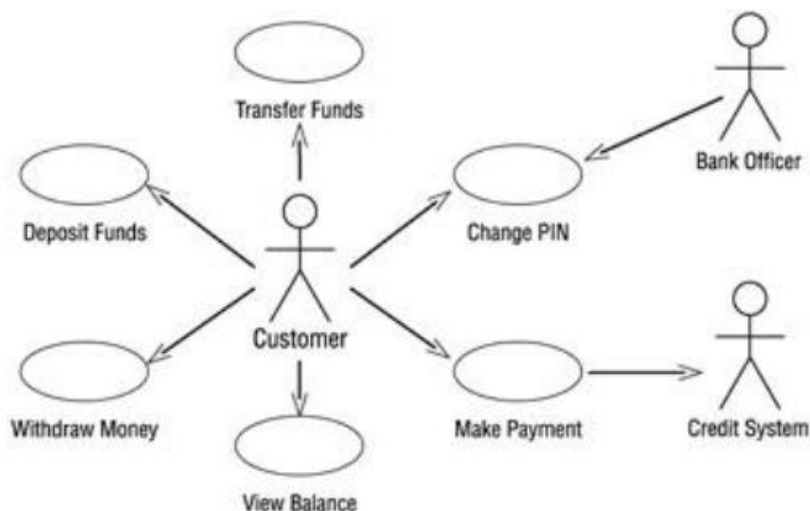
配置视图显示系统的具体部署（部署是指将系统配置到由计算机和设备组成的物理结构上）。配置视图显示系统的物理部署，它描述位于节点上的运行实例的部署情况。配置视图主要由配置图表示，它的使用者是开发人员、系统集成人员和测试人员。

2. 图:

视图由图组成，UML 通常提供 9 种基本的图

1) 用例图(Use Case Diagram):

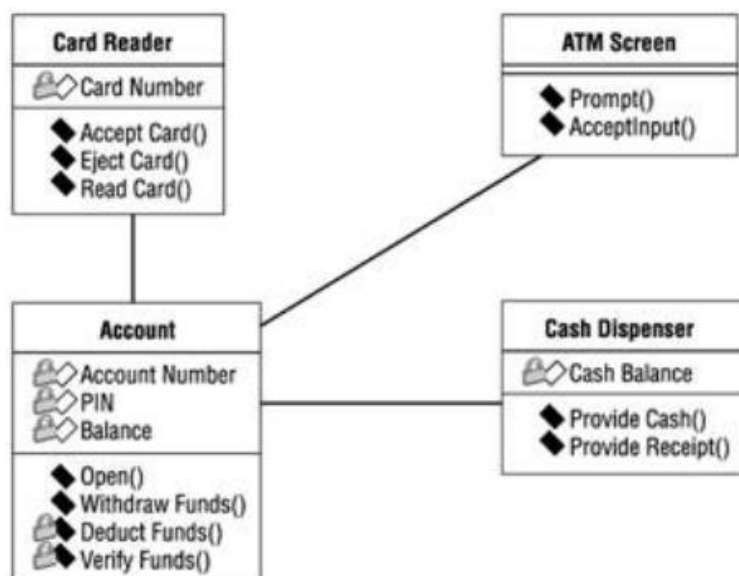
显示多个外部参与者以及他们与系统提供的用例之间的连接。用例是系统中的一个可以描述参与者与系统之间交互作用的功能单元。用例仅仅描述系统参与者从外部观察到的系统功能，



并不描述这些功能在系统内部的具体实现。用例图的用途是列出系统中的用例和参与者，并显示哪个参与者参与了哪个用例的执行。

2) 类图(Class Diagram):

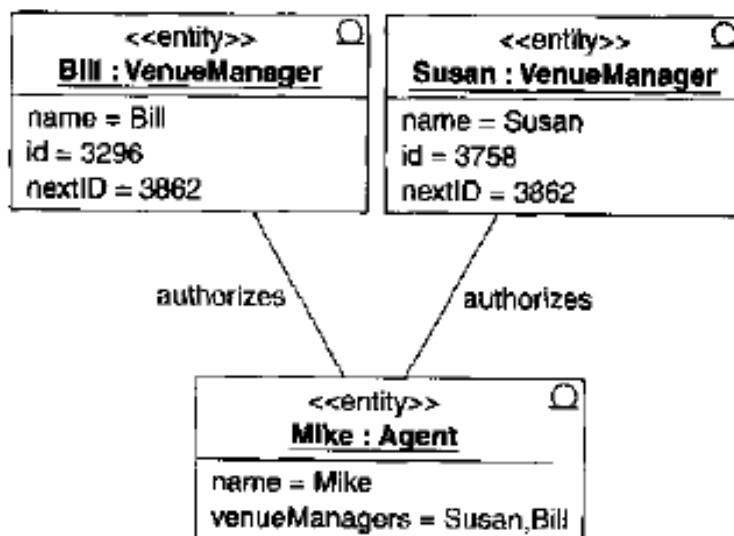
类是对应用领域或应用解决方案中概念的描述。类图以类为中心组织，类图中的其他元素或属于某个类，或与类相关联。



3) 对象图(Object Diagram):

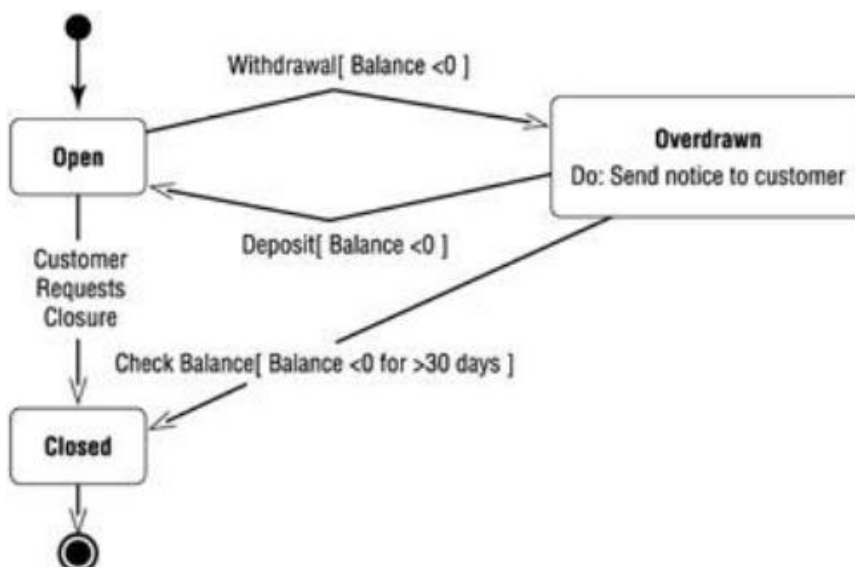
对象图是类图的变体，它使用与类图相似的符号描述，不同之处在于对象图显示的是类的对象实例而非实际的类。可以说，

对象图是类图的一个例子，用于显示系统执行时的一个可能的快照，即在某一时间点上系统可能呈现的样子。



4) 状态图(State Diagram):

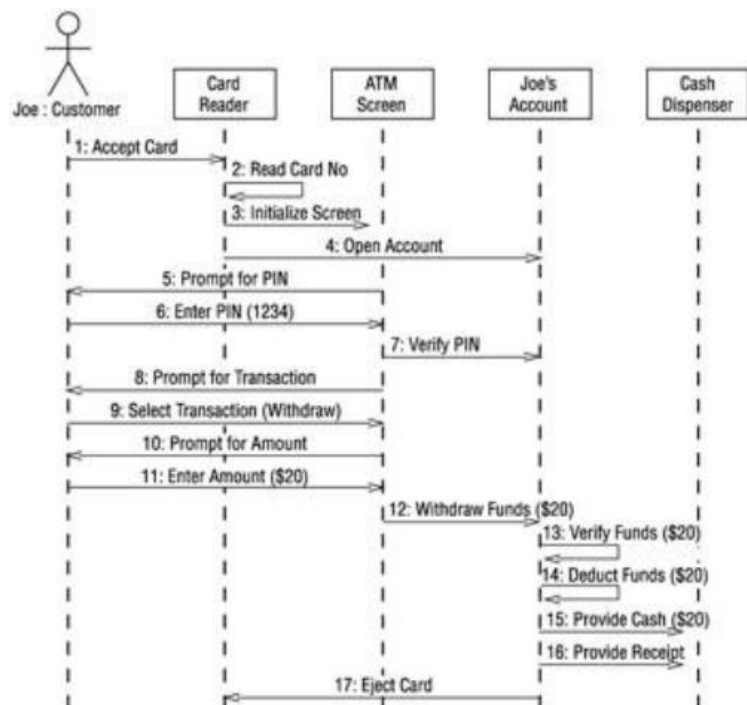
是对类描述的补充，它用于显示类的对象可能具备的所有状态，以及引起状态改变的事件。状态图由对象的各个状态和连接这些状态的转换组成。每个状态对一个对象在其生命周期中满足某种条件的一个时间段建模。事件的发生会触发状态间的转换，导致对象从一种状态转化到另一新的状态。



5) 顺序图(Sequence Diagram):

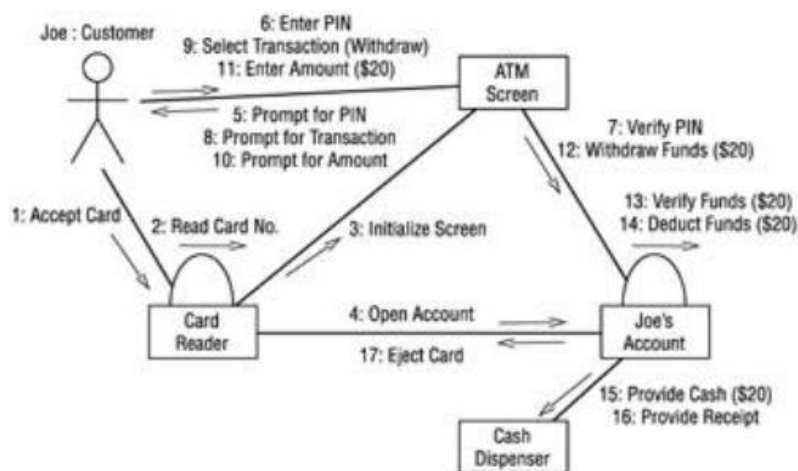
显示多个对象之间的动态协作，重点是显示对象之间发送的

消息的时间顺序。顺序图也显示对象之间的交互，就是在系统执行时，某个指定时间点将发生的事情。顺序图的一个用途是用来表示用例中的行为顺序，当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。



6) 协作图(Collaboration Diagram):

对在一次交互中有意义的对象和对象间的链建模。除了显示消息的交换（称之为交互）以外，协作图也显示对象以及他们之间的关系。

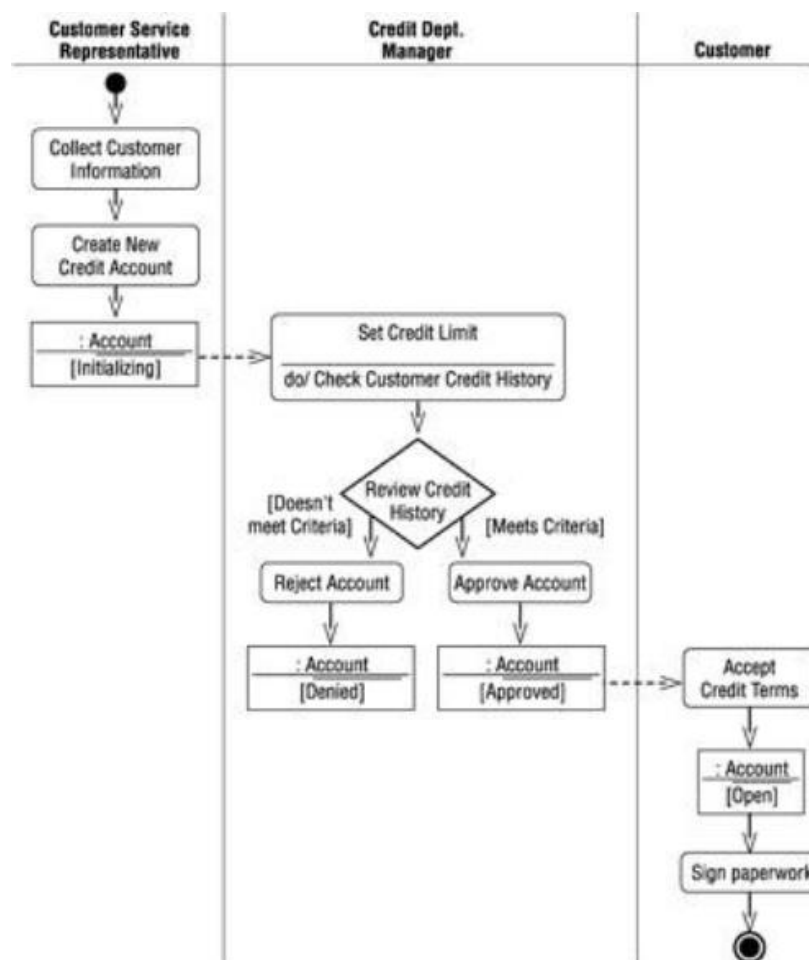


顺序图和协作图都可以表示对象间的交互关系，但它们的侧重点不同。顺序图用消息的几何排列关系来表达消息的时间顺

序，各角色之间的关系是隐含的；协作图用各个角色的几何排列来表示角色之间的关系，并用消息来说明这些关系。

7) 活动图(Activity Diagram):

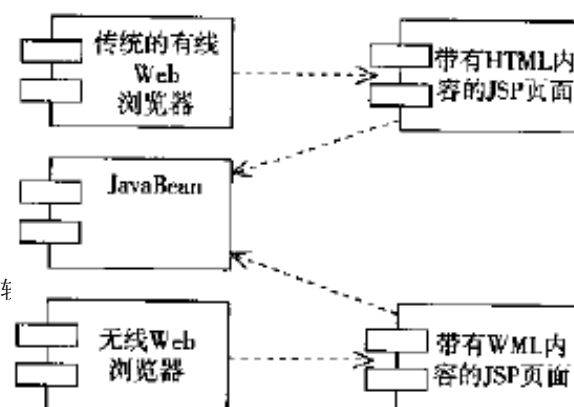
是状态图的一个变体，用来描述执行算法的工作流程中涉及的活动。



状态图表示一个对象在一段时间内的状态变化，而活动图则描述多个对象的状态变化序列。

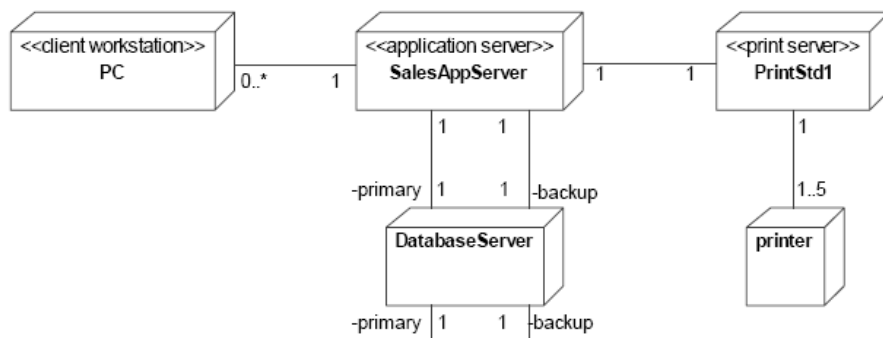
8) 组件图(Component Diagram):

用代码组件来显示代码物理结构，组件可以是源代码组件、二进制组件或一个可执行的组件。一个组件包含它所实现的一个或多个逻辑类的相关信息。

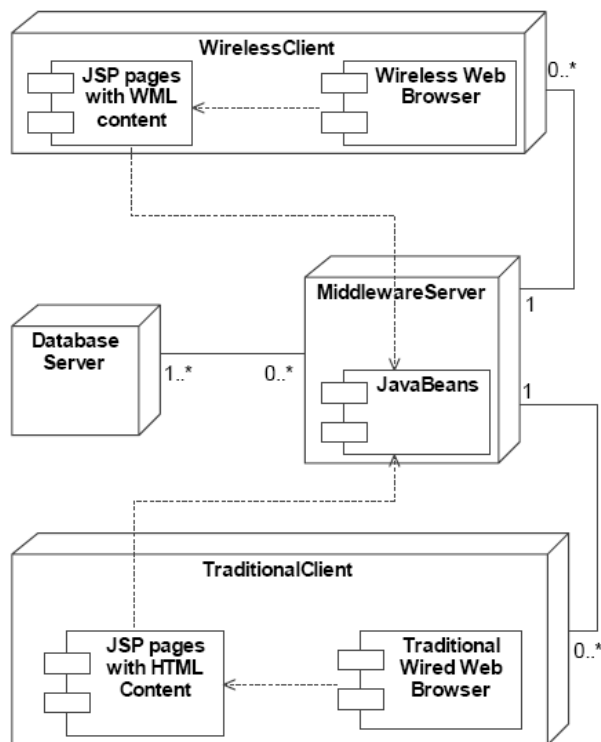


9) 配置图(Deployment Diagram):

用于显示系统中的硬件和软件的物理结构。配置图不仅可以显示实际的计算机和设备（节点），还可以显示它们之间的连接和连接的类型。在配置图中显示哪些节点内已经分配了可执行的组件和对象，以显示这些软件单元分别在哪个节点上运行。



组件图和配置图的组合：（如下图所示）



3. 模型元素:

UML 中的模型元素包括事物和事物之间的联系。

1) 事物:

事物描述了一般的面向对象的概念，是 UML 模型中面向对象的基本的建筑块，它们在模型中属于静态部分，代表物理上或概念上的元素。如：类、对象、接口、消息和组件等。

UML 中的事物可分为以下四类：

■ 结构事物

结构事物共有 7 种：

- ◆ 类
- ◆ 接口
- ◆ 协作
- ◆ 用例
- ◆ 活动类
- ◆ 组件
- ◆ 节点

■ 动作事物

是 UML 模型中的动态部分，它们是模型的动词，代表时间和空间上的动作。

结构事物共有 2 种：

- ◆ 交互
- ◆ 状态机

■ 分组事物

是 UML 模型中组织的部分。

分组事物只有一种，称为包。包是一种将有组织的元素分组的机制，结构事物、动作事物甚至其他的分组事物都可以放在一个包中。

■ 注释事物

是 UML 模型中的解释部分。

2) UML 中的关系

UML 中的关系有以下几种：

- 关联关系
- 依赖关系
- 泛化关系
- 实现关系
- 聚合关系

4. 通用机制：

1) 修饰：

在使用 UML 建模时，可以将图形修饰（如：字体、颜色等）附加到 UML 图中的模型元素上，这种修饰为图中的模型元素增加了语义。

2) 注释:

注释是以自由的文本形式出现的，它的信息类型是不被 UML 解释的字符串。注释可以附加到任何模型中去，可以放置在模型的任意位置上，并且可以包含任意类型的信息。

3) 通用划分。

UML 对其模型元素规定了两种类型的通用划分：

1. 型—实例
2. 接口—实现

4) 扩展机制。

UML 中包含 3 种主要的扩展组件：

■ 构造型

构造型是由建模者设计的新的模型元素，新的模型元素的设计要以 UML 已定义的模型元素为基础。

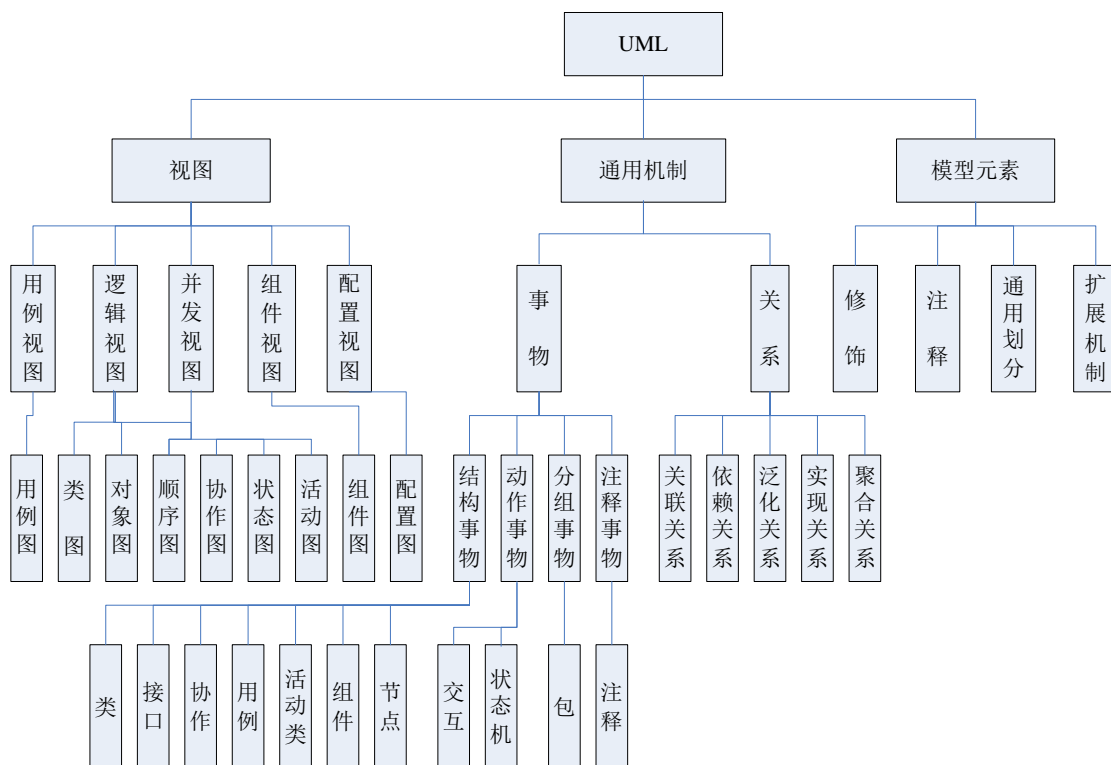
■ 标记值

标记值是附加到任何模型元素上的命名的信息块。

■ 约束

约束是用某种形式化语言或自然语言表达的语义关系的文字说明。

5. UML 的组成结构：（如下图所示）



三. 基于 UML 的软件开发：

a) 基于 UML 的面向对象分析、设计过程

运用 UML 进行面向对象的系统分析设计，通常都要经过如下 3 个步骤：

i. 识别系统的用例和参与者。

首先要对项目进行需求调研，分析项目的业务流程图和数据流程图，以及项目中涉及的各级操作人员，识别出系统中的所有用例和参与者；接着分析系统中各参与者和用例间的联系，使用 UML 建模工具画出系统的用例图；最后，勾画系统的概念层模型，借助 UML 建模工具描述概念层的类图和活动图。

ii. 进行系统分析并抽象出类。

系统分析的任务是找出系统的所有需求并加以描述，同时建立特定领域模型，建立领域模型有助于开发人员考察用例。从实际需求中抽象出类，并描述各个类之间的关系。

iii. 设计系统，并设计系统中类及其行为。

设计阶段由结构设计和详细设计组成。结构设计是高层设计，其任务是定义包（子系统）、包间的依赖关系和主要通信机制。包有利于描述系统的逻辑组成部分以及各部分之间的依赖关系。详细设计主要用来细化包的内容，清晰描述所有的类，同时使用 UML 的动态模型描述在特定环境下这些类的实例的行为。

b) UML 建模的简单流程

利用 UML 建造系统时，在系统开发的不同阶段有不同的模型，并且这些模型的目的是不同的：

- 1) 在分析阶段，模型的目的是捕获系统的需求，建立“现实世界”的类和协作的模型；
- 2) 在设计阶段，模型的目的是在考虑实现环境的情况下，将分析模型扩展为可行的技术方案；
- 3) 在实现阶段，模型是那些书写并编译的实际源代码；
- 4) 在部署阶段，模型描述了系统是如何在物理结构中部署的。

§ 9.8 小结

§ 9.9 补充实例

第十章 软件质量保证

§ 10.1 什么是 SQA

软件质量保证是通过对软件产品和活动有计划的进行评审和审计来验证软件是否符合标准的系统工程活动。

- 确保 SQA 活动要自始至终有计划的进行
- 审查软件产品和活动是否遵守适用的标准、规程和要求并得到客观验证。
- SQA 的活动和结果要保证全员参与，沟通顺畅。
- 逐级解决不符合问题

§ 10.2 SQA 活动

- 技术方法的应用
- 正式技术评审的实施
- 软件测试
- 标准的执行
- 修改的控制
- 度量
- 质量记录和记录保存

§ 10.3 SQA 活动的影响因素

- **知识结构：**专业的技术，例如质量管理与控制知识、统计学知识等。
- **经验**
- **依据：**如果没有这些标准，就无法准确地判断开发活动中的问题，容易引发不必要的争论，因此组织应当建立文档化的开发标准和规程。
- **全员参与：**全员参与至关重要，高层管理者必须重视软件质量保证活动。
- **把握重点：**一定要抓住问题的重点与本质，尽可能避免陷入对细

节的争论之中。

§ 10.4 SQA 策略

SQA 策略主要分三个阶段：

- 以检测为重：产品制成之后进行检测，只能判断产品质量，不能提高产品质量。
- 以过程管理为重：把质量的保证工作重点放在过程管理上，对制造过程中的每一道工序都要进行质量控制。
- 以新产品开发为重：在新产品的开发设计阶段，采取强有力的措施来消灭由于设计原因而产生的质量隐患。

第十一章 软件项目计划与管理