第3章　结构化分析与设计

经过40年的发展，软件工程已从第一代（传统软件工程）经历第二代（OO软件工程）发展到第三代（基于构件的软件工程）。作为软件工程的一个重要组成部分——应用软件的开发过程，也先后出现了瀑布模型、快速原型模型、增量模型、螺旋模型、转换模型、净室模型和构件集成模型等开发模型。从第3章起，将按照第一、二、三代软件工程的发展顺序，在前两章的基础上，依次展示3代软件工程的常用开发技术。

本章重点介绍基于瀑布模型的结构化分析与设计。为了方便读者学习，本章将相关的技术（例如模块设计）集中在一章中，并精简了一些过时的技术。

3.1　概述

3.1.1　结构化分析与设计的由来

结构化分析与设计最初是由结构化程序设计扩展而来的。

早在20世纪70年代中期，Stevens、Myers 与 Constantine 等人就在结构化程序设计的基础上，率先倡导了一种称为结构化设计（structured design，SD）的软件设计技术。20世纪70年代后期，Yourdon 等人又倡导了与SD配套的结构化分析（structured analysis，SA）技术，合称为结构化分析与设计方法。它是第一代软件工程时期最有代表性的应用系统开发方法，不仅适用面广、流行时间长，而且与模块设计共同构成为第一代软件工程时期最为常用的技术，至今仍在某些特定类型的软件开发中有所应用。

1．瀑布模型的首次实践

瀑布模型系由传统的生存周期过程演变而来。作为一种系统开发方法，结构化分析与设计是瀑布模型的首次实践。由第2.2.1节图2.2可见，该模型一般可划分为以下阶段：

需求定义与分析→总体设计→详细设计→编码→测试→使用维护

其中，测试阶段又可细分为单元测试、综合（或集成）测试、确认测试及系统测试等子阶段。鉴于下文还将详细介绍（见8.7节），这里就不说了。

2．SA与SD的流程

根据瀑布模型的上述流程，需求分析与软件设计是进入编码阶段前必须完成的任务。具体地说，系统开发从需求分析开始，首先建立系统的需求模型；接着通过 SD 方法提供的映

射规则，把分析模型转化为初始设计模型；然后再优化为系统的最终设计模型。系统的整个开发流程，可以简明地表示为：

结构化分析（工具：DFD、PSPEC）——→分析模型（分层 DFD 图）+ SRS

结构化设计（工具：SC 图）—映射→初始设计模型（初始 SC 图）

初始设计模型（初始 SC 图）—优化→最终设计模型（最终 SC 图）

简言之，SA 与 SD 的流程其实也是为待开发系统建立分析模型和设计模型的过程。

3．基本任务与指导思想

（1）结构化分析

SA 有两项基本任务，即建立系统分析模型（analysis model）和编写软件需求规格说明书（software requirements specification，SRS），二者都是分析阶段必须完成的文档。

① 建立分析模型。SA 模型包含描述软件需求的“一组”模型，通常有功能模型、数据模型和行为模型 3 种模型，分别表示待开发系统的功能需求、数据需求与行为需求。由于它们一般都用图形符号来表示，直观易懂，因而是形成 SRS 文档、完成软件设计的基础。

② 编写需求规格说明书。SRS 是分析阶段编写的、以文字为主的文档。当一个项目决定开发后，开发人员就应与用户共同确定软件的目标和范围（见第 2.6.3 节“项目实施计划”）。在分析阶段，上述问题定义（或称为问题陈述）被进一步细化为 SRS。在国际标准 IEEE 830—1998 标准和中国国家标准 GB856D—88 中，都建议了 SRS 文档的主要内容，其中包括引言、信息描述、功能描述、行为描述、质量保证、接口描述以及其他需求等。这些标准还同时强调：

- SRS 应该具有准确性。任何微小的错漏都可能铸成大错，在纠正时需付出巨大的代价。
- SRS 应该防止二义性。不要采用用户不容易理解的专门术语，以免导致误解。
- SRS 应该直观易改。尽可能采用图形和符号，例如将分析模型作为附录放在 SRS 之后，使不熟悉计算机的用户也能一目了然。

③ 主要指导思想。抽象与分解，是结构化分析的主要指导思想。现实世界中的系统不论表现形式上怎样杂乱无章，总可以通过“分析与归纳”从中找出一些规律，再通过“抽象”构建系统的模型。由于软件工程是一种层次化的技术，所以抽象通常也可分层次进行。当需要获得系统的细节时，就应该移向低层次的抽象。抽象的层次愈低，呈现的细节也会愈多。

（2）结构化设计

① 软件设计 = 总体设计 + 详细设计。由上述第一点可知，瀑布模型的软件设计包含了总体设计和详细设计两个阶段。SD 阶段把分析模型中的 DFD 图转换为最终 SC 图（即图 2.2 中的“软件结构图”），这仅仅完成了软件设计的第一步。在随后的详细设计中，还需用适当的工具对各个模块采用的算法和数据结构进行足够细致的描述（即图2.2中的“模块说明”），这也是模块设计阶段的基本任务。

结构化设计与模块设计相结合，即共同形成传统软件开发的常用技术。

② SC 图需分两步完成。从上述 SD 的流程可知，结构化设计产生的 SC 图一般需分为两步完成：即首先通过“映射”获得初始 SC 图；然后通过“优化”获得最终 SC 图。

综合上述①、②两步，可以表示为：

软件设计 { 总体设计 { 初始 SC 图 / 最终 SC 图 } / 详细设计：完成模块说明 }

③ 软件设计的指导思想。在软件开发的所有阶段中，软件设计是最富有活力、最需要发挥创造力的阶段。

分解和细化，历来是重要的软件设计策略。细化是与抽象相反而又互补的一对概念。1971 年，N. Wirth 就发表了“用逐步细化（stepwise refinement）的方法开发程序”一文，指出程序设计是一个“渐进”的过程：“对于一个给定的程序，每一步都把其中的一条或数条指令分解为较多的更详细的指令。”Yourdon 称赞该文“开创了自顶向下设计的先河”，“虽然把一个大系统分为小片，然后又分为更小的小片在今天已众所周知，但在当时（20 世纪 70 年代初）确实是一种革命的思想”。有人甚至誉之为“结构化程序设计的心脏”。

细化的实质就是分解。在传统的软件开发中，“逐步细化”不仅相继应用于结构化程序设计、结构化设计和模块设计中，而且也扩展应用于结构化分析中（例如分层 DFD 图就是逐步细化的应用）。由此可见，它早已超出了设计策略的范畴，已成为问题求解的一种通用技术了。

3.1.2 SA 模型的组成与描述

以下将结合一个引例，进一步对 SA 和 SD 两种模型及其描述工具进行说明。

[例 3.1] 引例：教材销售系统。

从用户调查中得知，在计划经济时期，某高校向学生销售教材的手续是：先由系办公室的张秘书开一购书证明，学生凭证明找教材科的王会计开购书发票，向李出纳员交付书款，然后到书库找赵保管员领书。现欲将上述手工操作改用计算机处理，开发一个“教材销售系统”。试按照上述步骤进行需求分析。

[解] 如果把用户目前使用的系统称为“当前系统”，用计算机实现的系统称为“目标系统”，则本例的需求分析大体上可以按下述 4 步进行：

第一步：通过对现实环境的调查研究，获取当前系统的具体模型，如图 3.1 所示。

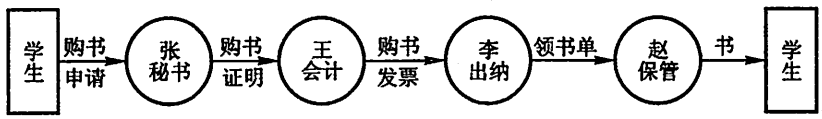


图 3.1 学生购买教材的当前系统模型

第二步：分析需求，建立系统分析模型，包括当前系统模型和目标系统模型。

① 去掉上述模型中的非本质因素，提炼出当前系统的逻辑模型。

在图 3.1 中，张、王、李、赵等具体的人是可能变动的，但需要他们处理的工作，例如审查购书有效性、开发票、开领书单等则是不变的，后者才是本质的内容。经过这样的分析，就可抽象出学生购买教材这一系统的逻辑模型，如图 3.2 所示。

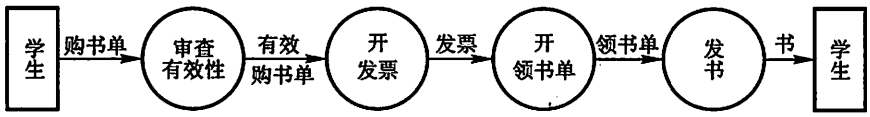


图 3.2　学生购买教材的逻辑模型

② 分析当前系统与目标系统的差别，建立目标系统的逻辑模型。

目标系统是一个基于计算机的系统。一般说来，它的功能应该比当前的现行系统更强，不必也不应该完全模拟现行的系统。例如在销售教材的计算机系统中，“有效性审查”及“开发票”就可合并进行，省去“开有效购书单”这一手续，如图 3.3 所示。

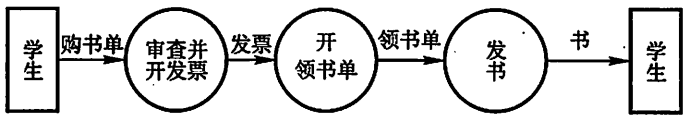


图 3.3　目标系统的逻辑模型

第三步：整理综合需求，编写系统需求规格说明书。（此处从略）

第四步：验证需求，完善和补充对目标系统的描述。

① 通过目标系统的人机界面，和用户一起确认目标系统功能，主要是区分哪些功能交给计算机去做，哪些功能由人工完成。例如在图 3.3 所示的系统逻辑模型中，按照书费收款和发书这两项工作仍需由人工完成。

② 复审需求规格说明书，补充迄今尚未考虑过的细节，例如确定系统的响应时间、增加出错处理等。在本例中，假如购书单中出现了学生不该购买或已经卖完的教材，就可通过“无效书单”将相关的信息通知学生。

经过以上的修正和补充，即可得到改进后的目标系统逻辑模型，如图 3.4 所示。至此，销售系统的分析模型即告完成。它主要用图形符号来表达，在 SA 中称为 DFD 图。

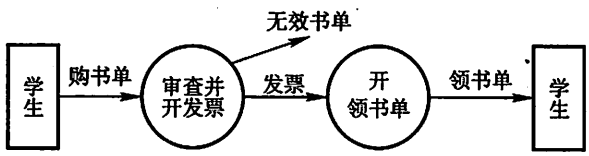


图 3.4　改进了的目标系统模型

以下将结合引例，对 SA 模型的组成及其常用描述工具举例说明。

1．SA 模型的组成

图 3.5 显示了 SA 模型的组成。由图可见，数据字典（data dictionary，DD）处于模型的核心，它是系统涉及的各种数据对象的总和。从 DD 出发可构建 3 种图：

① 实体联系图（entity-relation diagram，E-R 图）用于描述数据对象间的关系，它代表软件的数据模型，在实体联系图中出现的每个数据对象的属性，均可用数据对象说明来描述。

② 数据流图（data flow diagram，DFD）主要作用是指明系统中的数据是如何流动和变换的，以及描述使数据流进行变换的功能。在 DFD 图中出现的每个功能，则可在加工规格说明（process specification, PSPEC）中进行描述，它们一起构成软件的功能模型。

③ 状态变换图（status transform diagram，STD）用于指明系统在外部事件的作用下将如何动作，表明系统的各种状态以及状态间的变换（transfer，或称变迁），从而构成行为模型的基础。关于软件控制方面的附加信息，还可用控制规格说明（control specification, CSPEC）来描述。

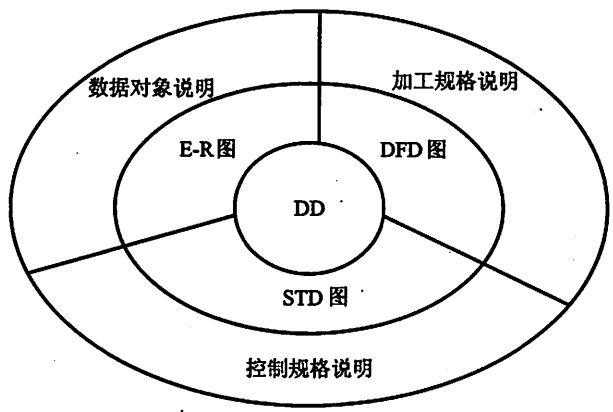


图 3.5 SA 模型的组成

需要指出，早期的 SA 模型仅包括 DD、DFD 和 PSPEC 等 3 个组成部分，主要用于描述软件的数据模型（用 DD 表示）与功能模型（用 DFD 和 PSPEC 表示）。随着社会信息化的迅速发展，许多应用系统包含了较复杂的数据信息。于是在数据建模（data modeling）时，有人将原用于关系数据库设计的 E-R 图移用于 SA，以便描述具有复杂数据对象的信息模型。另一方面，随着计算机实时系统（real-time system）应用的不断扩大，人们在分析建模中发现，有些数据加工（data processing）不是由数据来触发，而是由实时发生的事件来触发/控制的，无法用传统的 DFD 图来表示。因而在 20 世纪 80 年代中期，以 Ward 和 Hatley 等为代表的学者又在功能模型之外扩充了行为模型，推荐用控制流图（control flow diagram，CFD）、CSPEC 和 STD 等工具进行描述。今天，SA 模型已可同时覆盖信息模型、功能模型和行为模型等 3 种模型，其适用的软件范围也更加扩大了。

2．SA 模型的描述工具

综上可见，SA 模型的组成及其常用描述工具可以归结为：

① DFD、DD 和 PSPEC。它们是早期 SA 模型的基本组成部分。

② CFD、CSPEC 和 STD。它们是早期 SA 模型的扩展成分，可适应实时软件的建模需要。

③ E-R 图。适用于描述具有复杂数据结构的软件数据模型。

现分别例示如下。

（1）数据流图（DFD）

任何软件系统（或计算机系统）从根本上来说，都是对数据进行加工（processing）或变换（transform）的工具。图 3.6 是一个高度抽象了的软件系统功能模型。

① 组成符号。数据流图只使用 4 种基本图形符号：圆框代表加工；箭头代表数据的流向，数据名称总是标在箭头的边上；方框表示数据的源点和终点；双杠（或单杠）表示数据文件或数据库。

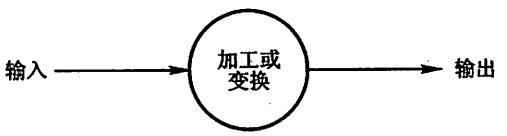


图 3.6　软件功能模型

文件与加工之间用带箭头的直线连接，单向表示只读或只写，双向表示又读又写。注意，每一图形符号都必须标上名字，加工框还应该加上编号，以帮助识别。

[例 3.2]　把图 3.4 转换成 DFD 图。

[解]　图 3.4 现有 2 个加工（审查并开发票、开领书单），4 个数据流（购书单、发票、领书单、无效书单），数据的源点和终点都是学生。但图中没有数据文件。实际上在审查购书单和开出发票之前，至少要查阅 2 个文件：一个文件是各班学生用书表，用以核对学生是否需用这些教材；另一个文件是教材存量表，以确定是否有该学生要买的教材。把这 2 个文件加到图 3.4 中，并给加工添上编号，就得到基于计算机的教材销售系统的 DFD，如图 3.7 所示。

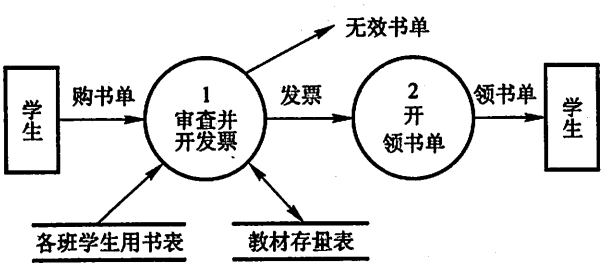


图 3.7　教材销售系统的数据流图

加工 1（即“审查并开发票”）要从教材存量表中读出数据，以判断有没有可卖的教材；售出教材后又要在原存量中减去售出的数量，把新存量写回教材存量表，所以在加工 1 与教材存量表之间使用了带双箭头的连线。各班学生用书表只读不写，应用单向箭头连线连接，

图中箭头的方向表示从文件读出。

② DFD 的性质。与程序流图（flow diagram）不同，DFD 不能表示程序的控制结构，例如选择结构或循环结构等。前一种图形用于表示程序的模块设计（procedural design），后一种图形则用作软件分析阶段的工具。由于分析阶段只需考虑软件“干什么”，不必过问“怎么干”，当然不应包括控制流、控制结构或激发条件之类的信息。

DFD 所表现的范围，可大到整个系统，小到一个模块。在需求分析中，常常用一组 DFD 图由粗到精地表示同一软件在不同抽象级别上的功能模型，并称之为分层数据流图（leveled DFD，参阅下文第 3.2.1 节）。

（2）数据字典（DD）

一个软件系统含有许多数据。数据字典的作用，就是对软件中的每个数据规定一个定义条目，以下结合图 3.7 的教材销售系统举例说明。

① 数据流。

[例 3.3] 以图 3.7 中的“发票”为例，编写一个字典条目。

[解] “发票”是一个数据流，其条目内容与书写格式如表 3.1 所示。

表 3.1 数据流“发票”的字典条目

数据流名：发票
别　名：购书发票
组　成：学号＋姓名＋｛书号＋单价＋数量＋总价｝＋书费合计
备　注：

在组成栏中，“学号”、“姓名”、“书号”等都是数据项的名称。就本例而言，“学号”、“姓名”与“书费合计”在数据流中仅出现一次，其余各数据项则每购买一种书就要出现一次。条目中用｛｝表示重复。在条目内容中列入“别名”，是因为对同一数据可能存在不同的称呼，并不是说允许同一数据在系统中使用不同的名字。

② 数据文件。

[例 3.4] 为教材销售系统中的各班学生用书表编写一个字典条目。

[解] 如表 3.2 所示，在该条目中，每个记录记载一个班在一学年中需用的教材。“组织”是数据文件条目所特有的内容，用于说明文件中的记录将按照什么规则组合成文件。

表 3.2 数据文件“各班学生用书表”的字典条目

文件名：各班学生用书表
组　成：｛系编号＋专业和班编号＋年级＋｛书号｝｝
组　织：按系、专业和班编号从小到大排列
备　注：

③ 数据项。

[例 3.5]　数据项字典条目示例。

[解]　表 3.3 及表 3.4 分别列出了两个数据项字典条目："数量"及"书费合计"。

表 3.3　数据项"数量"的字典条目

数据项名：数量
别　　名：购书量
取　　值：正整数
备　注：

表 3.4　数据项"书费合计"的字典条目

数据项名：书费合计
别　　名：
取　　值：00.00～99.99
备　注：

（3）加工规格说明

加工规格说明通常用结构化语言（structured language）、判定表（decision table）或判定树（decision tree）作为描述工具。每个加工规格说明可以像字典中的条目一样记在卡片上。以下将分别对加工规格说明卡片常用的 3 种描述手段作简单介绍。

① 结构化语言。

[例 3.6]　在图 3.7 的教材销售系统中，使用结构化语言来描述加工 1（其功能是"审查并开发票"）的加工逻辑。

[解]　图 3.8 显示了用结构化语言描述的加工规格说明。

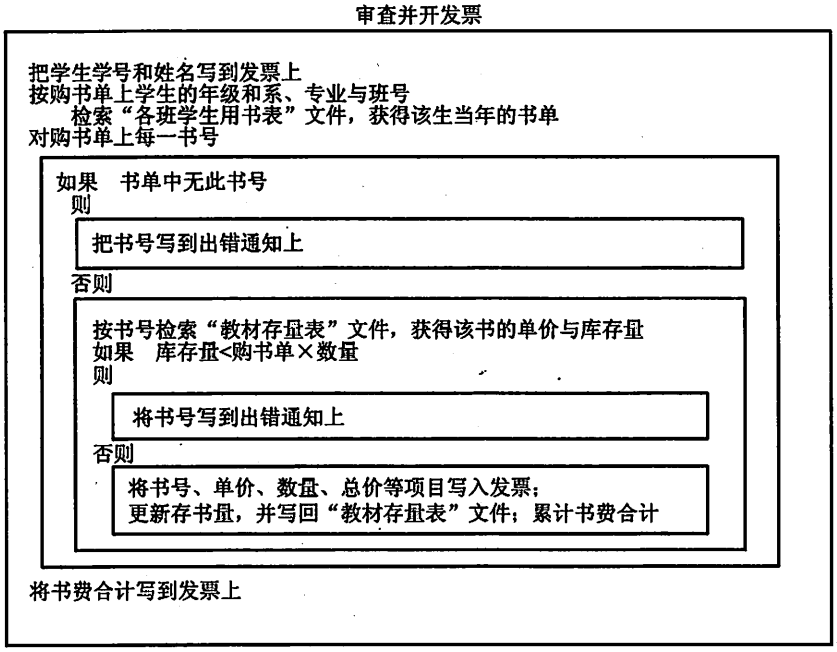


图 3.8　"审查并开发票"加工逻辑

② 判定表或判定树。

[例3.7] 某公司为推销人员制定了奖励办法，把奖金与推销金额及预收货款的数额挂钩。凡每周推销金额不超过 10 000 元的，按预收货款是否超过 50%，分别奖励推销额的 6%或 4%。若推销金额超过 10 000 元，则按预收货款是否超过 50%，分别奖励推销额的 8%或 5%。对于月薪低于 1 000 元的推销员，分别另发鼓励奖 300、200 和 500、300 元。

试分别采用判定表和判定树为 DFD 图中用来“计算奖金”的加工写出 PSPEC。

[解] 图 3.9 与图 3.10 分别显示了用判定表和判定树描述的 PSPEC，二者的含义相同。

推销奖金策略				
	规　则			
条　件	1	2	3	4
推销金额	>10 000	≤10 000	>10 000	≤10 000
预收货款	>50%	>50%	≤50%	≤50%
动　作				
置奖金率为	8%	6%	5%	4%
置奖金额=奖金率×推销金额				
如果推销员月薪低于1 000元				
另加奖金额	500	300	300	200

图 3.9　兼有结构化语言的判定表

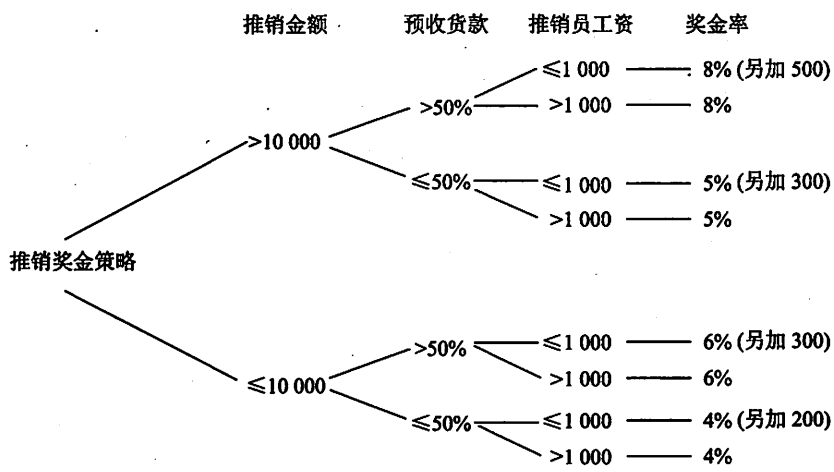


图 3.10　判定树示例

3.1.3　SD 模型的组成与描述

1．SD 模型的组成

SD 模型是由 SA 模型映射而来的。例如在结构化设计中，SA 模型的数据字典可转换为待开发系统的数据设计；数据流图可转换为体系结构设计（SC 图）与接口设计；加工规格说明可转换为模块内部的详细过程设计；等等。图 3.11 显示了设计模型的组成。

2．SD 模型的描述工具

在图 3.11 中，由下向上包含了数据设计、体系结构设计、接口设计与过程设计。顾名思义，体系结构设计是用来确定软件结构的，其描述工具为结构图（structure chart），简称 SC 图。过程设计主要指模块内部的详细设计，其描述工具将在“3.4 模块设计”中一并介绍。

（1）SC 图的组成符号

在 SC 图中，用矩形框来表示模块，带箭头的连线表示模块间的调用，并在调用线的两旁标出传入和传出模块的数据流。图 3.12 显示了 SC 图允许使用的 6 种模块。其中，传入、传出和变换模块用来组成变换结构中的各个相应部分；源模块是不调用其他模块的传入模块，只用于传入部分的始端；漏模块是不调用其他模块的传出模块，仅用于传出部分的末端；控制模块是只调用其他模块，不受其他模块调用的模块，例如变换型结构的顶层模块，事务型结构的事务中心等，均属于这一类。

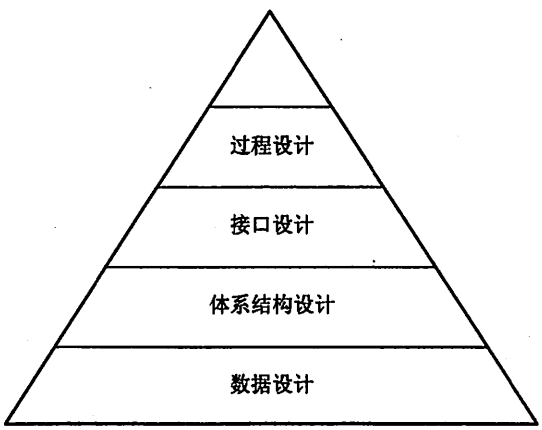


图 3.11　SD 模型的组成

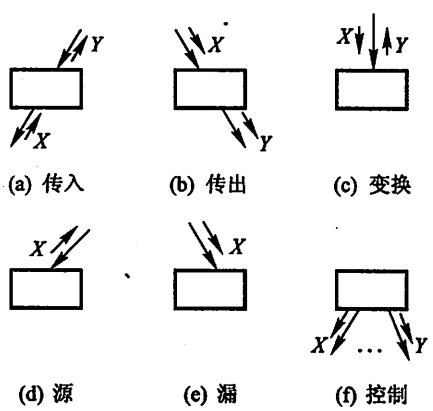
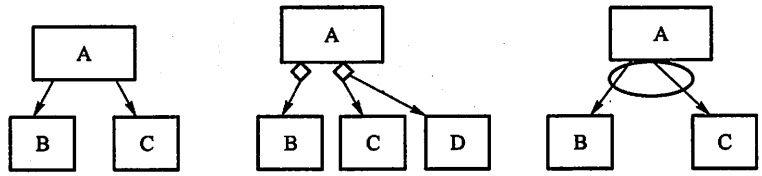


图 3.12　SC 图使用的模块符号

（2）SC 图的模块调用

图 3.13 显示了 SC 图中模块调用关系的表示方法，从左至右分别为简单调用、选择调用和循环调用。

为了画面简洁，在图 3.13 中调用线的两旁均未标出数据流。在实际的 SC 图中不允许这种省略。



（a）简单调用　　（b）选择调用　　（c）循环调用

图 3.13　SC 图中模块调用关系的表示

3.2　结构化系统分析

按照 T. DeMarco 的定义，“结构化分析就是使用 DFD、DD、结构化英语、判定表和判定树等工具，来建立一种新的、称为结构化说明书的目标文档”。这里的结构化说明书就是 SRS。随着计算机应用的发展，有些学者还提出了用于结构化分析的补充工具，使之适用于实时系统的结构化分析模型，例如 CFD、CSPEC、STD 等。

结构化分析的基本步骤是：自顶向下对系统进行功能分解，画出分层 DFD 图；由后向前定义系统的数据和加工，编制 DD 和 PSPEC；最终写出 SRS。以下仍结合实例进行说明。

3.2.1　画分层数据流图

大型复杂的软件系统，其 DFD 可能含有数百乃至数千个加工，不可能一次将它们画完整。正确的做法是：从系统的基本功能模型（把整个系统看成一个加工）开始，逐层地对系统进行分解。每分解一次，系统的加工数量就增加一些，加工的功能也更具体一些。继续重复这种分解，直到所有的加工都足够简单为止。通常把这种不需再分解的加工称为“基本加工”，把上述逐步分解称为“自顶向下、逐步细化”（top-down stepwise refinement），最终为待开发的系统画出一组分层的数据流图，以代替一张含有系统全部加工的包罗万象的总数据流图。

[例 3.8]　将图 3.7 的教材销售系统扩展为教材购销系统。

例 3.2 的教材销售系统只支持销售，不支持采购。为弥补这一缺陷，拟扩展该系统的功能，使之能根据教材存量表，以“缺书单”的形式通知书库保管员，并在采购后用“进书通知”更新教材存量。试用 SA 方法，为扩展后的教材购销系统画出分层 DFD 图。

[解]　画系统分层数据流图的第一步，是画出顶层图。通常把整个系统当作一个大的加工，标明系统的输入、输出以及数据的源点与终点（统称为“外部项”）。图 3.14 显示了教材购销系统的顶层图。它表明，系统从学生那里接受购书单，经处理后把领书单返回给学生，使学生可凭单到书库领书。对脱销的教材，系统用缺书单的形式通知书库保管员；新书进库后，也由保管员将进书通知返回给系统。

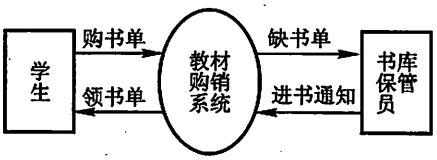


图 3.14　教材购销系统的顶层 DFD

接下来画第二层 DFD 图，把系统分解为销售和采购两大加工，如图 3.15 所示。显然，外部项学生应与销售子系统联系，外部项书库保管员应与采购子系统联系。两个子系统之间也存在两项数据联系：其一是缺书登记表，由销售子系统把脱销的教材传送给采购子系统；其二是进书通知，直接由采购子系统将教材入库信息通知销售子系统。

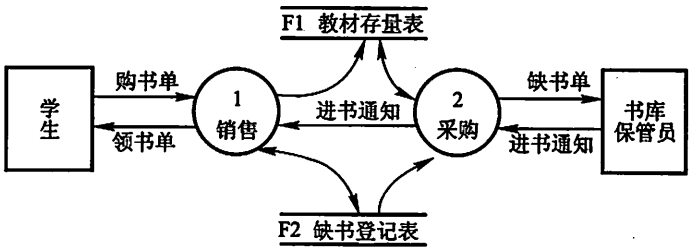


图 3.15　教材购销系统的第二层 DFD 图

继续分解，可获得第三层数据流图。其中图 3.16 由销售子系统扩展而成，图 3.17 由采购子系统扩展而成。

在图 3.16 中，销售子系统被分解为 6 个子加工，编号从 1.1 至 1.6。审查有效性时，首先要校核购书单的内容是否与学生用书表（F3）相符，还要通过售书登记表（F4）检查学生是否已买过教材。若发现购书单与学生用书不符或已买过该教材，应发出无效书单，只将通过审查的教材留在有效购书单中。“开发票”加工按照购书单查对教材存量表（F1），把可供应的教材写入发票，数量不足的教材写入缺书单。前者在 F4 中登记后开出领书单发给学生，后者则登记到缺书登记表（F2），待接到进书通知后再补售给学生。补售的手续及数据流程和第一次购书相同。这里要注意的是，在上一层 DFD（图 3.15）中，采购是系统内部的一个加工框，但在图 3.16 中，“采购”却是处于销售之外的一个外部项。

采购子系统在图 3.17 中被分解为 3 个子加工。由销售子系统建立起来的缺书登记表(F2)，首先按书号汇总后登入待购教材表（F5），然后再按出版社分别统计制成缺书单，送给书库保管员作为采购教材的依据。在汇总缺书时要再次核查教材存量表（F1），按出版社统计时还要参阅教材一览表（F6），从而确定所缺教材是哪个出版社出版的。新书入库后，要及时修改教材存量表和待购教材表中的有关教材数量，同时把进书信息通知销售子系统，使销售人员能通知缺书的学生补买。

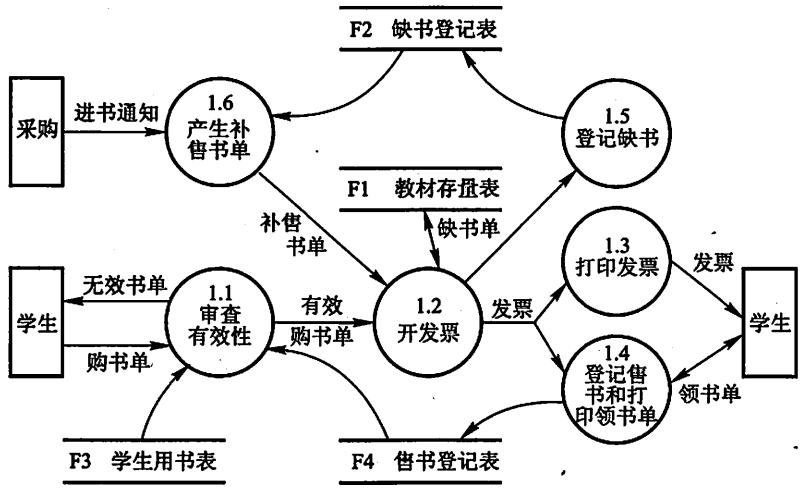


图 3.16 第三层 DFD 图——销售子系统

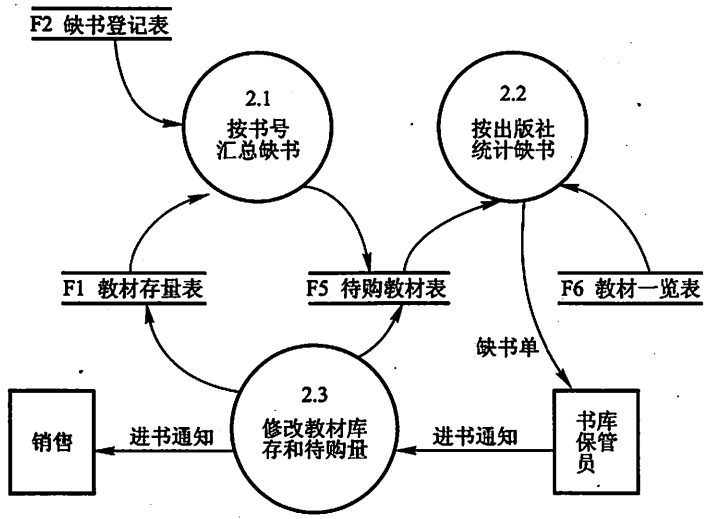


图 3.17 第三层 DFD 图——采购子系统

以上共画了 3 层、4 个 DFD 图（图 3.14～图 3.17），组成了教材购销系统的分层 DFD 图。愈到下层加工愈细。第三层的加工框已是足够简单的“基本加工”，不必再分解了。

由本例可见，分层 DFD 具有下列优点：

① 便于实现。采用逐步细化的扩展方法，可避免一次引入过多细节，有利于控制问题的复杂度。

② 便于使用。用一组图代替一张总图，使用户中的不同业务人员可各自选择与本身有

关的图形，不必阅读全图。例如在本例中，销售人员可只阅读图 3.16，书库保管员可只阅读图 3.17，各取所需。

3.2.2　确定数据定义与加工策略

分层 DFD 图为整个系统勾画了一个概貌。下一步就应考虑系统的细节，例如定义系统的数据、确定加工的策略等问题了。

最低一层 DFD 图已包含了系统的全部数据和加工，从哪里开始分析呢？W. Davis 认为，一般应从数据的终点开始。因为终点的数据代表系统的输出，其要求是明确的。由这里开始，沿着 DFD 图一步步向数据源点回溯，较易看清楚数据流中每一个数据项的来龙去脉。

以图 3.16 为例，“领书单”是系统的主要输出数据流，至少应包括学号、姓名、书号和数量 4 个数据项。而它的源数据流，图中 1.4 框的输入数据流的组成是：

发票 ＝ 学号＋姓名＋｛书号＋单价＋数量＋总价｝＋书费合计

可见领书单中的内容都能在发票中找到。1.4 框的策略之一，就是从发票中选择有用的数据项写入领书单。其次，该框还要登记售书，防止学生重复购买，所以售书登记表（F4）组成应与领书单组成相同。再往前回溯是 1.2 框，输入该框的“有效购书单”的组成内容是：

有效购书单 ＝ 学号＋姓名＋｛书号＋数量｝

与发票的组成比较，它缺少单价、总价和书费合计等 3 个数据项。显然，1.2 框在开出发票前，必须先计算每种书的总价和所有售书的合计书费。但单价从哪里得来呢？由图 3.16 可知，1.2 框可访问的文件仅有一个，即教材存量表（F1）。因此该文件不仅要存储各种教材的现有数量，还应该包括它们的单价。即

教材存量表 ＝｛书号＋单价＋数量｝

现在再考察 1.2 框的加工策略。接到有效购书单以后，它首先要访问教材存量表（F1），查清有哪些书的数量不能满足购书单的要求，将缺书单送到 1.5 框，并由该框把信息存入缺书登记表（F2）。为便于以后补售，缺书单、补售书单和缺书登记表的组成，都可以与有效购书单相同，即：

缺书单 ＝ 学号＋姓名＋｛书号＋数量｝

补售书单 ＝ 学号＋姓名＋｛书号＋数量｝

缺书登记表 ＝｛学号＋姓名＋｛书号＋数量｝｝

对当前可以供应的教材，一方面要计算每种书的总价和书费累计，另一方面要更改存书数量，把剩余书数量写回教材存量表。有效购书单的上述数据流程及处理方法，也适用于补售书单，不再重复。

继续回溯，就可循有效购书单找到 1.1 框，或者循补售书单找到 1.6 框，分别得出这两个框的加工策略和各个有关数据的定义。分析的方法与以上相似，就不详细说明了。

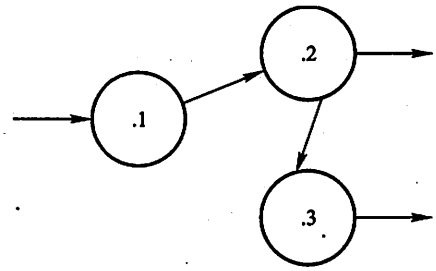
上述分析结束后，分析员应为 DFD 的每个数据逐一写出定义，每个基本加工逐个进行加工规格说明，并汇编成数据字典。

综上可知，分层 DFD 图产生了系统的全部数据和加工，通过对这些数据和加工的定义，常常对分析员提出一些新问题（例如前述的教材单价应从哪里取得），促使他们进行新的调查和思考，并可能导致对 DFD 的修改。画 DFD，定义加工和数据，再画，再定义，如此循环，直至产生一个为用户和分析员一致同意的文档——SRS。

3.2.3 需求分析的复审

需求分析的文档完成后，应由用户和系统分析员共同进行复审，并吸收设计人员参加。

复审的重点，是 DFD、DD 和加工规格说明等文档的完整性、易改性和易读性，尽量多地发现文档中存在的矛盾、冗余与遗漏。例如，要注意 DFD 图的加工编号：通常顶层加工不编号，第二层加工编为 1，2，…，n 号，第三层编为 1.1，1.2，1.3，…，n.1，n.2，n.3，依此类推。与此相应，各层 DFD 的图号是：顶层 DFD 图无图号，第二层编为图 0；第三层编为图 1、图 2，依此类推直至图 n。当层次较多时，编号允许用简化方法表示，如图 3.18 所示。



DFD图号：3.6.5
加工编号：.1相当于 3.6.5.1
.2相当于 3.6.5.2
.3相当于 3.6.5.3

图 3.18 加工编号的简化

3.3 结构化系统设计

软件的需求分析完成后，就可以开始软件设计了。同需求分析一样，软件设计目前也有两种主流方法，即基于结构化程序设计的结构化软件设计和基于面向对象技术的面向对象软件设计。

3.3.1 SD 概述

前已指出，SD 方法是率先由 Stevens、Myers 与 Constantine 等人在 20 世纪 70 年代中期倡导的。

1．面向数据流设计和面向数据设计

按照出发点的不同，传统的软件设计又可细分为面向数据流的设计和面向数据（或数据结构）的设计两大类。前者以 SD 方法为主要代表，后者以 Jackson 方法为主要代表。

在面向数据流的方法中，数据流是考虑一切问题的出发点。以 SD 方法为例，在与之配套的 SA 方法中，通常用数据流图来表示软件的逻辑模型；在设计阶段，又按照数据流图的不同类型（变换型或事务型）将它们转换为相应的软件结构。Jackson 方法则不同，它以数据结构作为分析与设计的基础。众所周知，算法和数据结构是传统程序设计中不可分割的两个侧面。根据 Hoare 的研究（详见“Notes on Data Structures”，1972），算法的结构在很大程度

上依赖于它要处理的数据结构。例如当问题的数据结构具有选择性质时，就需用选择结构来处理；如果数据结构具有重复性质，就需用循环结构来处理；分层次的数据结构总是导致分层次的程序结构；等等。因此，如果事先知道了问题的数据结构，即可由此导出它的程序结构。这是面向数据结构设计方法的根据与基本思想。

基于数据流还是基于数据结构，标志了两类设计方法的不同出发点；不仅如此，它们的最终目标也不相同。SD 方法把注意力集中在模块的合理划分上，其目标是得出软件的体系结构图；Jackson 方法则要求最终得出程序的过程性描述，并不明确提出软件应该先分成模块等概念。因此，后一类方法虽然也可作为独立系统设计方法应用于小规模数据处理系统的开发，但在一般情况下更适合于在模块设计阶段使用。这两类方法也存在着许多共同点。例如，它们都遵守结构化程序设计、逐步细化等设计策略；都要从分析模型导出设计模型；并服从“程序结构必须适应问题结构”的原则。

鉴于 Jackson 方法已基本过时，以下不再展开介绍，本章仅讨论 SD 方法。

2．从分析模型导出设计模型

设计是把用户的需求准确地转换为软件产品或系统的唯一方法。无论是传统的设计或面向对象的设计，都要从分析阶段得到的分析模型导出软件的设计模型。Pressman 用简明的图形说明了这种导出关系，如图 3.19 所示。

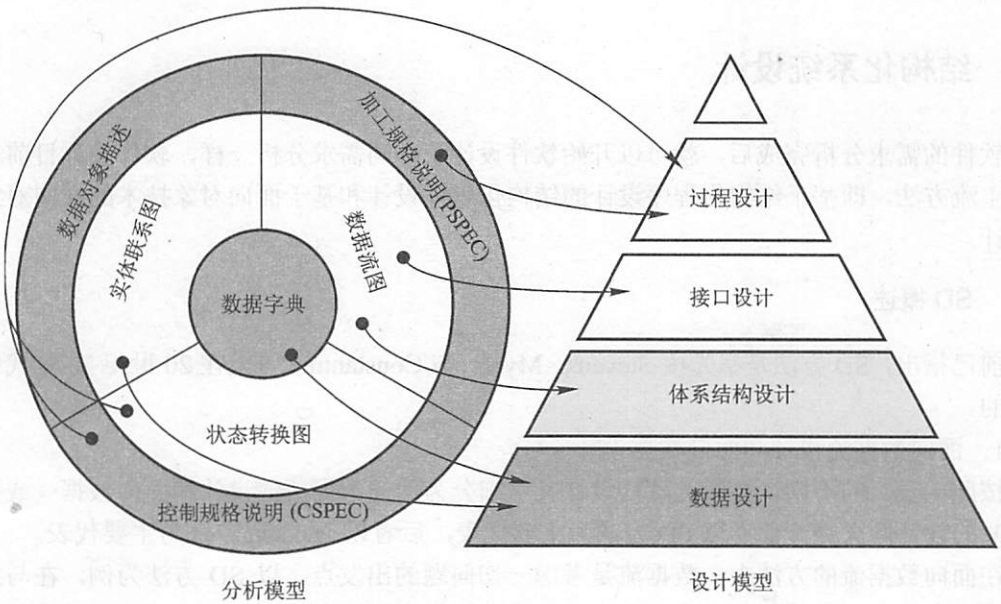


图 3.19　将分析模型转换为软件的设计模型

由图 3.19 可见，传统软件设计所产生的数据设计、体系结构设计、接口设计和过程设计，

均可从分析模型所包含的各种图形和说明中找出所需的信息。例如，体系结构设计与接口设计可以由数据流图导出，过程设计可根据加工规格说明、控制规格说明和状态转换图来定义，等等。其中不少系统设计方法都提供了将分析描述直接转换为设计描述的映射（mapping）规则，使软件设计变得更加容易。

3.3.2 SD 的步骤：从 DFD 图到 SC 图

结构化软件的设计，通常从 DFD 图到 SC 图的映射开始。

1．数据流图的类型

从 SA 获得的 DFD 中，所有系统可归结为变换型结构和事务型结构两种类型。

（1）变换型结构

由传入路径、变换中心和传出路径 3 部分组成。图 3.20 显示了该型结构的基本模型和数据流。

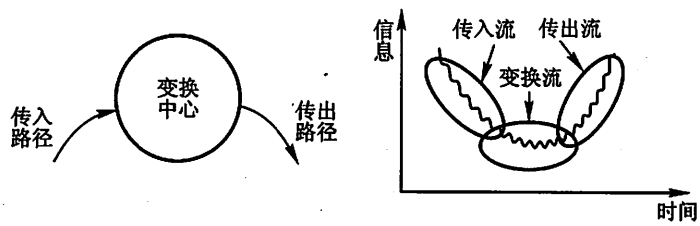


图 3.20 变换型结构的系统

（2）事务型结构

事务一词常见于商业数据处理系统，一笔账目，一次交易，都可以看作一次事务。在更广泛的意义上，一次动作、事件或状态变化也可成为一次事务。事务型结构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，其基本模型如图 3.21 所示。当外部信息沿着接受路径进入系统后，经过事务中心获得某一个特定值，就能据此启动某一条动作路径的操作。

在一个大型系统的 DFD 中，变换型和事务型结构往往同时存在。例如在图 3.22 所示的 DFD 中，系统的总体结构是事务型的，但是在它的某（几）条动作路径中，很可能出现变换型结构。在另一些情况下，在整体为变换型结构的系统中，其中的某些部分也可能具有事务型结构的特征。

2．SD 方法的步骤

为了有效地实现从 DFD 图到 SC 图的映射，结构化设计规定了下列 4 个步骤：

① 复审 DFD 图，必要时可再次进行修改或细化。

② 鉴别 DFD 图所表示的软件系统的结构特征，确定它所代表的软件结构是属于变换型还是事务型。

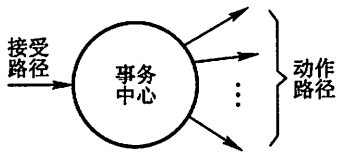


图 3.21　事务型结构的基本模型

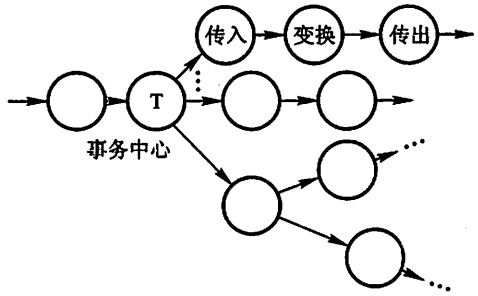


图 3.22　同时存在两类结构的系统

③ 按照 SD 方法规定的一组规则，把 DFD 图映射为初始 SC 图。

$$\text{变换型 DFD 图} \xrightarrow{\text{变换映射}} \text{初始 SC 图}$$

$$\text{事务型 DFD 图} \xrightarrow{\text{事务映射}} \text{初始 SC 图}$$

④ 按照优化设计的指导原则改进初始 SC 图，获得最终 SC 图。

3.3.3　变换映射

SD 方法提供了一组映射规则，大大方便了初始 SC 图的设计。其主要步骤包括：

① 划分 DFD 图的边界。

② 建立初始 SC 图的框架。

③ 分解 SC 图的各个分支。

[例 3.9]　用变换映射规则从图 3.23 导出初始 SC 图，假设该图已经过细化与修改。

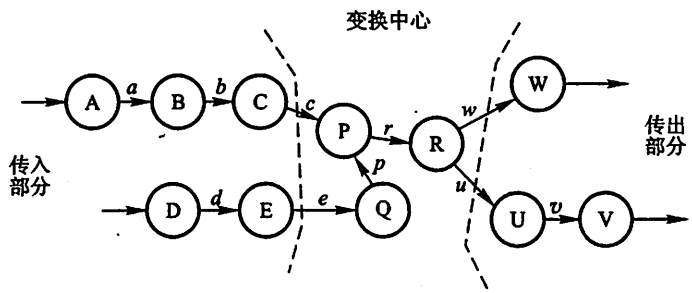


图 3.23　在 DFD 图上划分传入、传出和变换中心部分

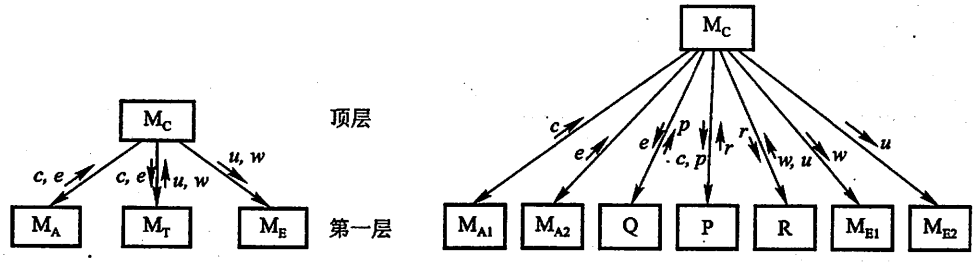
[解]　第一步：区分传入、传出和变换中心 3 个部分，在 DFD 图上标明它们的分界线。

第二步：完成第一级分解，建立初始 SC 图的框架，如图 3.24 所示。

第三步：完成第二级分解，细化 SC 图的各个分支。

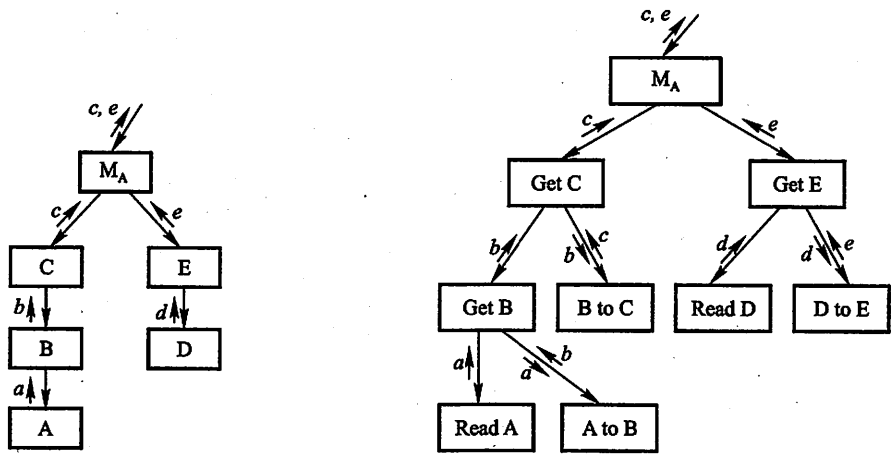
在图 3.25（a）中，传入模块 M_A 直接调用模块 C 与 E，以取得它所需的数据流 c 与 e。

继续下推，模块 C、E 将分别调用下属模块 B、D，以取得 *b* 与 *d*；模块 B 又通过下属模块 A，取得数据 *a*。



（a）画法之一　　（b）画法之二

图 3.24　第一级分解后的 SC 图



（a）方法之一　　（b）方法之二

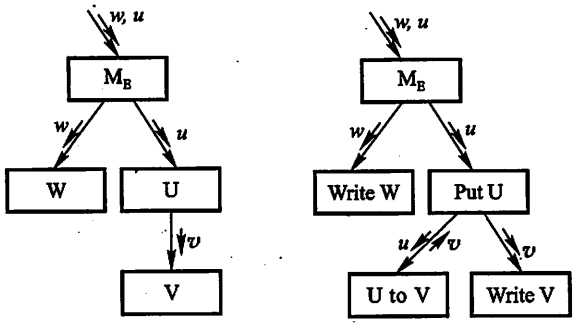
图 3.25　传入分支的分解

实际上数据流在传入的过程中，也可能经历数据的变换。以图 3.23 中的两个传入流为例，其中一路从 *a* 变换为 *b*，再变换为 *c*；另一路则从 *d* 变换为 *e*。为了显式地表示出这种变换，可以在图 3.25（a）中增添 3 个变换模块，分别是“A to B”、“B to C”、“D to E”，并在模块名称前加上 Read、Get 等字样，如图 3.25（b）所示。这一改变的实质是，除了处于物理输入端的源模块以外，让每一传入模块都调用两个下属模块，包括一个传入模块和一个变换模块。图 3.25（b）所示的结构，显然较图 3.25（a）更加清楚、明了。

仿照与传入分支相似的分解方法，可得到传出分支的两种模块分解图，如图 3.26 所示。

与传入、传出分支相比，变换中心分支的情况繁简迥异，其分解也较复杂。但建立初始的 SC 图时，仍可以采取“一对一映射”的简单转换方法。图 3.27 显示了本例变换中心分支

第二级分解的结果。



（a）方法之一　　（b）方法之二

图 3.26　传出分支的分解

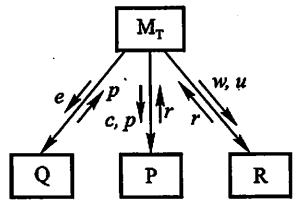


图 3.27　变换中心分支的分解

将图 3.25（b）、图 3.26（b）与图 3.27 合并在一起，就可以得到本例的初始 SC 图，如图 3.28 所示。

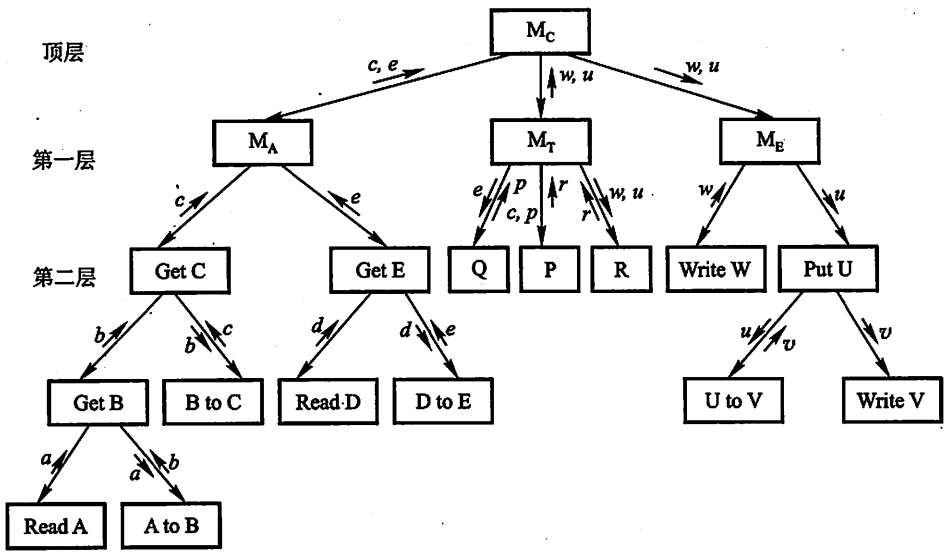


图 3.28　从图 3.23 导出的初始 SC 图

3.3.4　事务映射

与变换映射相似，事务映射也可以分为 3 个步骤：

① 在 DFD 图上确定事务中心、接受部分（包括接受路径）和发送部分（包括全部动作路径）。

② 画出 SC 图框架，把 DFD 图的 3 个部分分别映射为事务控制模块、接受模块和动作发送模块。

③ 分解和细化接受分支和发送分支，完成初始的 SC 图。

[例 3.10] 用事务映射方法，从图 3.29 所示的 DFD 图导出初始的 SC 图。

[解] 事务中心通常位于 DFD 图中多条动作路径的起点，向事务中心提供启动信息的路径，则是系统的接受路径。动作路径通常不止一条，可具有变换型或另一个事务型的结构。

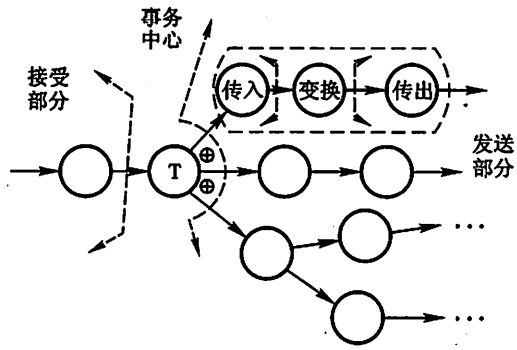


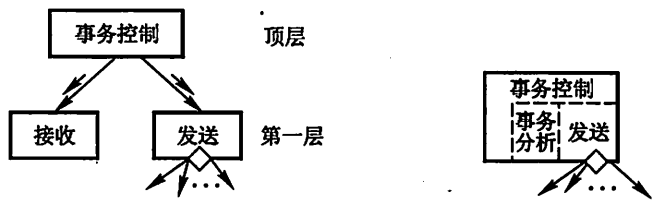
图 3.29 事务型 DFD 图的划分

第一步：划分 DFD 图的边界，并做出标记，如图 3.29 所示。

第二步：画出相应 SC 图的初始框架。图 3.30（a）显示了由二层组成的典型结构。

如果第一层的模块比较简单，也可以并入顶层，如图 3.30（b）所示。

第三步：重点是对动作（即发送）分支进行分解。其接受分支因一般具有变换特性，可以按变换映射对它进行分解。



（a）典型二层结构　　（b）简单结构

图 3.30 事务型 SC 图的上层结构

图 3.31 显示了动作分支的典型结构，含有 P、T、A、D 共 4 层。P 为处理层，相当于图 3.30 中的发送模块。T 为事务层，每一动作路径可映射为一个事务模块。在事务层以下可以再分解出操作层（actions 层）及细节层（details 层）。由于同一系统中的事务往往含有部分相同的操作，各操作又可能具有部分相同的细节，这两层的模块常能为它们的上层模块所共享，被多个上级模块调用。

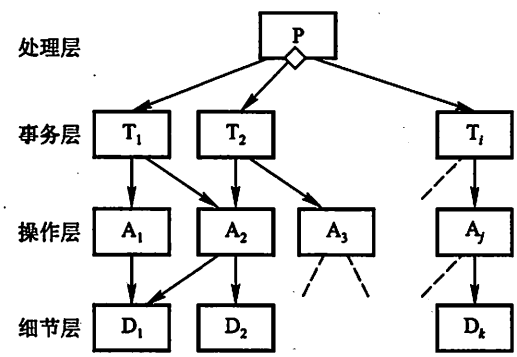


图 3.31 动作分支的典型结构

3.3.5　优化初始 SC 图的指导规则

把初始 SC 图变成设计文档中的最终 SC 图，需要进一步细化和改进。本节将介绍 SD 方法中常用于优化软件初始 SC 图的两条指导规则。

1．对模块划分的原则

一般来说，模块的总行数应控制在 10~100 行的范围内，最好为 30~60 行，能容纳在一张打印纸内。过长的模块往往是分解不充分的表现，会增加阅读理解的难度；但小模块太多也会使块间联系变得复杂，增大系统在模块调用时传递信息所花费的开销。

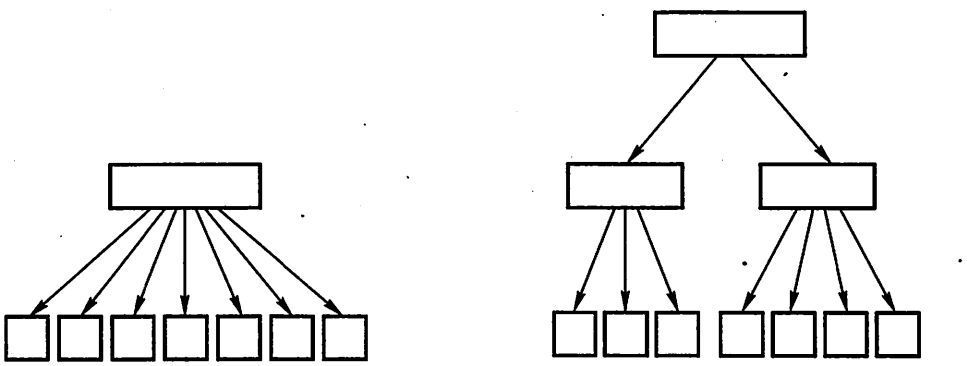
在改进 SC 图时，有些模块在图上的位置可能要上升、下降或左右移动，从而变更模块调用关系。模块位置应否变更，应视对计算机处理是否方便而定，不必拘泥于它与 DFD 图上对应的加工是否位置一致。

2．高扇入/低扇出的原则

扇入（fan-in）/扇出（fan-out）是从电子学借用过来的词，在 SC 图中可用于显示模块的调用关系，如图 3.32 所示。扇入高则上级模块多，能够增加模块的利用率；扇出低则表示下级模块少，可以减少模块调用和控制的复杂度。通常扇出数以 3～4 为宜，最好不超过 5～7。如扇出过大，软件结构将呈煎饼形（pancaking），如图 3.33（a）所示，此时可用增加中间层的方法使扇出减小，如图 3.33（b）所示。

（a）M 的扇入　　（b）M 的扇出

图 3.32　模块的扇入和扇出



（a）煎饼形结构　　（b）塔形结构

图 3.33　增加中间层可减少扇出

设计良好的软件通常具有瓮形（oval-shaped）结构，两头小，中间大，如图 3.34 所示。这类软件在下部收拢，表明它在低层模块中使用了较多高扇入的共享模块。煎饼形一般是不可取的，因为它常常是高扇出的结果。

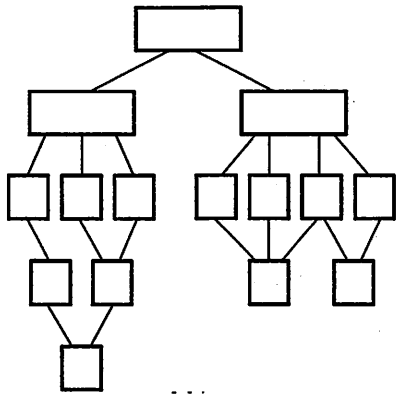


图 3.34 瓮形结构

3.3.6 教材购销系统的总体结构

[例 3.11] 在例 3.8 中，已获得教材购销系统第三层的两张 DFD 图，即图 3.16 的销售子系统 DFD 图和图 3.17 的采购子系统 DFD 图。试用 SD 方法从上述两张 DFD 图导出教材购销系统的总体结构，包括初始的 SC 图和改进后的最终 SC 图。

[解] 为节省篇幅，本例仅列出初始的 SC 图和最终 SC 图的结果，步骤从略。

1. 初始 SC 图

包括上层框架、销售子系统和采购子系统，详见图 3.35～3.37。

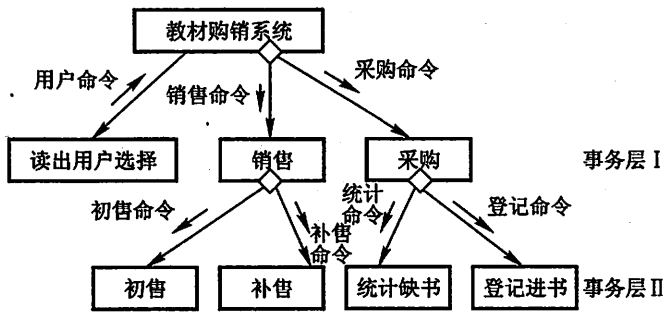


图 3.35 教材购销系统的上层框架

2. 最终 SC 图

改进后的上层框架如图 3.38 所示，包括初售、补售、统计缺书和登记进书 4 个子系统。

作为示例，这里仅显示初售子系统的最终 SC 图（如图 3.39 所示），以示一斑。

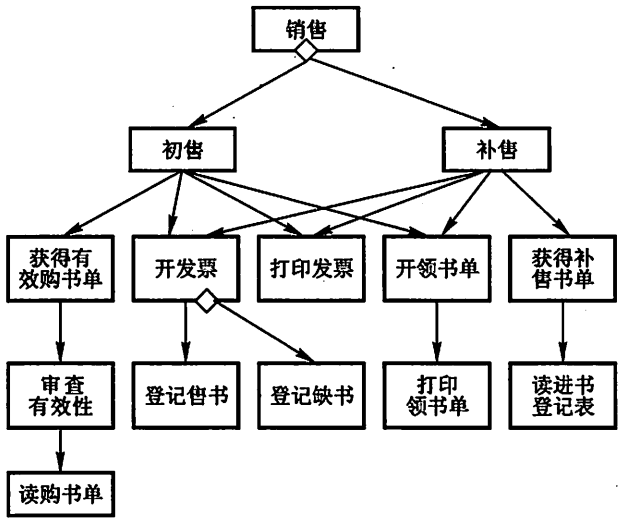


图 3.36　销售子系统初始 SC 图

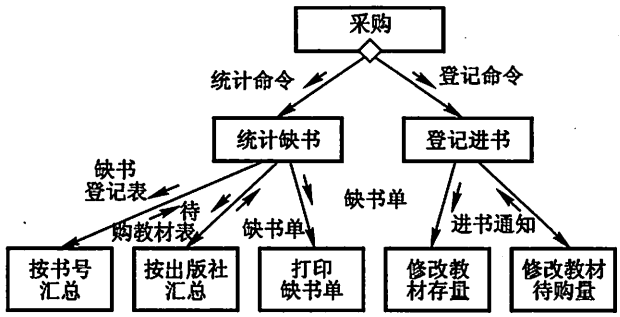


图 3.37　采购子系统初始 SC 图

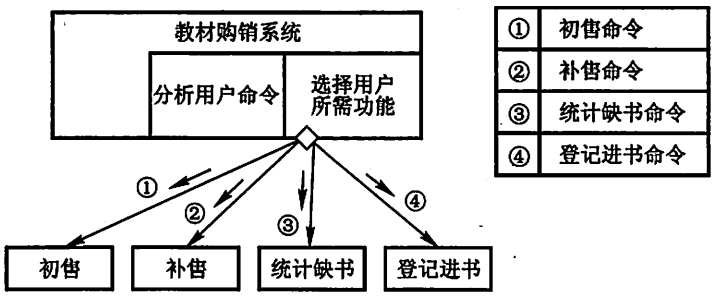


图 3.38　最终 SC 图的上层框架

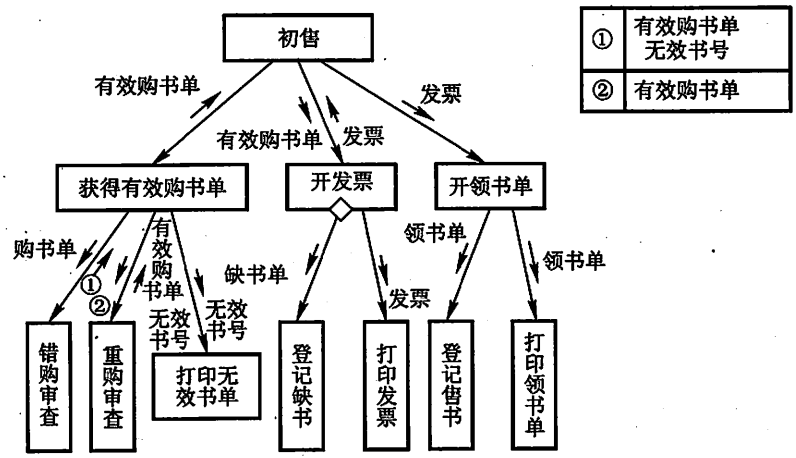


图 3.39 初售动作分支的最终 SC 图

3.4 模块设计

把 DFD 图转换为最终 SC 图，仅仅完成了软件设计的第一步。传统的软件工程将软件设计分成两步走：总体（或结构）设计——用最终 SC 图表示；模块设计——用逐步细化的方法来实现。模块设计用于对系统中的每个模块给出足够详细的逻辑描述，故亦称详细设计。这些描述可用规范化的表达工具来表示，但它们还不是程序，一般不能够在计算机上运行。

本节除说明模块设计的目的、任务与表达工具外，还要介绍如何用结构化程序设计的基本原理来指导模块内部的逻辑设计。

3.4.1 目的与任务

详细设计的目的，是为 SC 图中的每个模块确定采用的算法和块内数据结构，用选定的表达工具给出清晰的描述。表达工具可由开发单位或设计人员自由选择，但它必须具有描述过程细节的能力，而且能在编码阶段直接将它翻译为用程序设计语言书写的源程序。

这一阶段的主要任务，是编写软件的模块设计说明书。为此，设计人员应：

① 为每个模块确定采用的算法。选择某种适当的工具表达算法的过程，写出模块的详细过程性描述。

② 确定每一模块使用的数据结构。

③ 确定模块接口的细节，包括对系统外部的接口和用户界面，对系统内部其他模块的接口，以及关于模块输入数据、输出数据及局部数据的全部细节。

3.4.2　模块设计的原则与方法

结构程序设计的原理和逐步细化的实现方法，是完成模块设计的基础。现分述如下。

1．清晰第一的设计风格

早在 20 世纪 60 年代，E. W. Dijkstra 就公开提出了从高级语言中取消 GOTO 语句的主张。他认为，一个程序包含的 GOTO 语句愈多，其可读性就愈差。时而跳转过来，时而跳转过去，可能把程序搞成一团乱麻。Dijkstra 的主张得到许多人的支持，但也有人反对这种“一刀切”的做法，因而激起了一场激烈的辩论。这场辩论促使许多人改变了单纯强调程序效率的旧观念，认识到在大多数情况下，应该遵守“清晰第一，效率第二”的设计风格。

2．结构化的控制结构

1966 年，Bohem 和 Jacopini 就在一篇文章中证明：任何程序的逻辑均可用顺序、选择和循环（DO-WHILE 型）3 种控制结构或它们的组合来实现，从而在理论上为结构程序设计奠定了基础。1968 年，Dijkstra 建议仅用这 3 种控制结构来构成程序。1972 年，Mills 进一步提出每个控制结构只应有一个入口和一个出口的原则。综上可见，如果所有的模块在详细设计中都只使用单入口、单出口的 3 种基本控制结构，如图 3.40 所示，则不论一个程序包含多少个模块，整个程序将仍能保持一条清晰的线索。这就是常说的控制结构的结构化，它是详细设计阶段确保模块逻辑清晰的关键技术。

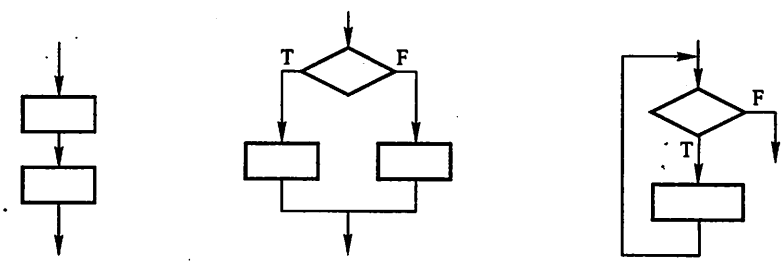


图 3.40　3 种基本控制结构的流程图

这里还有几点要补充说明：

① 为了方便使用或者提高程序效率，大多数软件开发项目还允许在详细设计中补充使用 DO-UNTIL 和 DO-CASE 两种控制结构。

② 在许多情况下，当程序执行到满足某种条件时，需要立即从循环中转移出来。如果死抠单出口的原则，就会不必要地使循环重复下去，延长程序的执行时间。为了解决这类问题，在 PDL 语言（详见第 3.4.3 节）中允许用 UNDO 语句提前退出循环，如图 3.41 所示。

3．逐步细化的实现方法

把给定的模块功能转换成它的详细逻辑描述，通常都采用逐步细化的策略。实践表明，逐步细化尤其适合于详细设计。由此产生的程序逻辑一般错误较少；可靠性也比较高。

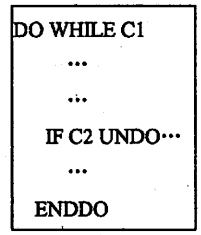


图 3.41 有两个出口 DO-WHILE 的结构

[例 3.12] 在一组数中找出其中的最大数。

[解] 首先，把问题的解答描述为：

第一步：

1 输入一组数。

2 找出其中的最大数。

3 输出最大数。

以上 3 条中，第 1、3 两条都比较简单，所以下一步主要细化第 2 条。

第二步：

2.1 任取一数，假设它就是最大数。

2.2 将该数与其余各数逐一比较。

2.3 若发现有任何数大于该一假设的最大数，即取而代之。

以上 3 条是第一步中第 2 条分解的结果。

把第一步中的第 1 条具体化，同时对 2.1～2.3 继续进行相应的细化，可得到：

1′ 输入一个数组。

2.1′ 令最大数＝数组中的第一元素。

2.2′ 从第二元素至最末一个元素依次做。

2.3′ 如果新元素>最大数，

则 最大数＝新元素。

3′ 输出最大数。

通过这一实例，可以把逐步细化的设计步骤归结为：

① 由粗到细地对程序进行逐步的细化。每一步可选择其中的一条至数条，将它（们）分解成更多或更详细的程序步骤。

② 在细化程序的过程时，同时对数据的描述进行细化。换句话说，过程和数据结构的细化要并行地进行，在适当的时候交叉穿插。

③ 每一步细化均使用相同的结构化语言，最后一步一般直接用伪代码来描述，以便编码时直接翻译为源程序。

逐步细化设计受到许多人的赞同，并不是偶然的。它的主要优点是：

① 每一步只优先处理当前最需要细化的部分（如上例第一步中的找出最大数），其余部分则推迟到适当的时机再考虑。先后有序，主次分明，可避免全面开花，顾此失彼。

② 易于验证程序正确性，比形式化的程序正确性证明更易被非专业人员接受，因而也

更加实用。传统的程序正确性验证采用公理化的证明规则，方法十分繁琐。一个不长的程序，常常需要一长串的验证。用逐步细化方法设计的程序，由于相邻步之间变化甚小，不难验证它们的内容是否等效。所以这一方法的实质，是要求在每一步细化中确保实现前一步的要求，不要等程序写完后再来验证。

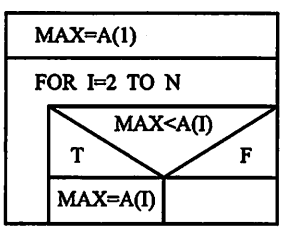
用“结构化”保证程序的清晰、易读，用“逐步细化”实现程序的正确、可靠，由此很自然地得出如下结论：模块的逻辑设计必须用结构程序设计的原理来指导。

3.4.3　常用的表达工具

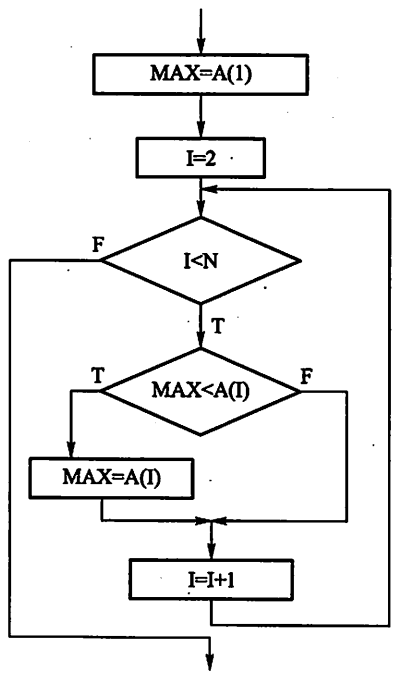
本节介绍详细设计中经常使用的几种设计表达工具。

1．流程图和 N-S 图

流程图（flow diagram）是最古老的设计表达工具之一，至今仍常常使用。但随着结构化程序设计的普及，流程图在描述程序逻辑时的随意性与灵活性，恰恰变成了它的缺点。1973 年，Nassi 和 Shneiderman 发表了题为“结构化程序的流程图技术”的文章，提出用方框图来代替传统流程图的思路和方法。根据这两位创始人的姓氏，人们把它简称为 N-S 图。图 3.42 显示了采用 N-S 图和流程图来表达同一段程序的情况。



（a）用 N-S 图描述



（b）用流程图描述

图 3.42　相同的程序用 N-S 图和流程图来表达的比较

2．伪代码和 PDL 语言

伪代码（pseudo code）属于文字形式的表达工具。它形式上与代码相似，但并非真正的代码，也不能在计算机上执行。1975 年，Caine 与 Gordon 在“PDL：一种软件设计工具”一文中，报导了他们所设计的一种伪代码，命名为 PDL（program design language）。

图 3.43、图 3.44 分别显示了用 PDL 描述选择结构和循环结构的方法。其中的循环结构又可区分为 DO-WHILE、DO-UNTIL 和 DO-FOR 等 3 类情况，如图 3.45 所示。

```
IF <条件>
   一或数条语句
ELSEIF<条件>
   一或数条语句
      ⋮

ELSEIF<条件>
   一或数条语句
ELSE
   一或数条语句
ENDIF
```

图 3.43　PDL 描述的 IF 结构

```
DO　判断重复的条件
一或数条语句
ENDDO
```

图 3.44　PDL 描述的 DO 结构

```
DO WHILE THERE ARE INPUT RECORDS
DO UNTIL "END" STATEMENT HAS BEEN PROCESSED
DO FOR EACH ITEM IN THE LIST EXCEPT THE LAST ONE
```

图 3.45　几种不同的 DO 结构

PDL 语言还设有 DO-CASE 语句，用以描述多分支的选择结构。

为了方便对比，我们把图 3.42（b）所示的流程图用 PDL 来描述，如图 3.46 所示，或进一步简化为图 3.47 的样式。

还需指出，上述 4 种工具除 N-S 图以外，其余 3 种工具也可在总体设计阶段用于表达软件的结构。实际上，除了 SC 图主要用作总体设计的表达工具外，其他设计工具的描述大都可粗可细，其范围也可大可小，并不限于仅在某个设计阶段使用。

```
Enter a vector
Set Maximum to the value of
    the first element in the vector
 DO for each element
    from the second one to the last
IF value of the element is greater than
   the Maximum value
    Set Maximum to value of the element
   ENDIF
ENDDO
Print the Maximum value
```

图 3.46　用 PDL 描述的图 3.41 的程序逻辑

```
Input array A
 MAX = A(1)
 DO for I = 2 to N
  IF MAX< A(I)
    Set MAX = A(I)
  ENDIF
 ENDDO
 Print MAX
```

图 3.47　图 3.45 的简化程序逻辑

小　结

本章讨论传统软件工程的系统开发技术，重点放在基于瀑布模型的结构化分析与设计和模块设计上，但不涉及同为传统软件工程的快速原型开发等内容。全章以实例（从“教材销售”到“教材购销”）为主线，依次展示了结构化分析、结构化设计和模块设计的常用技术。

1. 用 SA 方法建立分析模型

抽象与分解，是结构化分析的指导思想。通过由顶向下逐步细化得出的一组分层 DFD 图，是在不同的抽象级别上对系统所作的描述。逐层展开可以使问题的复杂性变得容易控制，使分析员不致突然面临一大堆细节。

2. 将分析模型映射为设计模型

建模与映射都是软件工程最常使用的技术。需求分析需建立分析模型，软件设计需建立软件模型，二者之间可通过映射实现转换。

本章例 3.1 展示了为“教材销售系统”建立分析模型的步骤。从该例可见，在面向数据流的软件系统开发中，DFD 图和加工规格说明是建立分析模型最常用的两种描述工具。通过画分层 DFD 图，可以获得待开发系统的一组分析模型；通过对加工规格说明的细化，可以获得软件的需求规格说明书（SRS）。在设计阶段，设计模型通常用 SC 图作为描述工具。SD 方法的主要任务，就是提供一组映射规则，把主要由 DFD 图和 SRS 组成的分析模型转换成由初始 SC 图描述的系统设计模型；然后通过对结构设计的优化，形成由最终 SC 图表示的系统设计模型。以下是从分层 DFD 图到最终 SC 图的一般过程：

结构化分析（工具：DFD、PSPEC）——→分析模型（分层 DFD 图）+ SRS

结构化设计（工具：SC 图）$\xrightarrow{\text{映射}}$初始设计模型（初始 SC 图）

初始设计模型（初始 SC 图）$\xrightarrow{\text{优化}}$最终设计模型（最终 SC 图）

3. 模块设计方法

模块设计是传统软件工程的重要组成部分，其性质属于详细设计，大致上相当于第二代软件工程的面向对象设计（OOD）。瀑布模型将软件设计分成总体设计和详细设计两大步：把 DFD 图转换为最终 SC 图，以及为 SC 图中的每个模块确定采用的算法和数据结构。从例 3.12 可以看出：

① 模块设计要用结构化程序设计的基本原理来指导，具体地说，就是要用“结构化”的思想保证程序的清晰易读，用“逐步细化”实现程序的正确、可靠。

② 算法和数据结构应该在逐步细化过程中并行地细化。

③ 模块设计的任务，是从 SC 图导出模块的过程性描述。这些描述可以采用图形的或者文字的表达工具。但是不论采用哪一种工具，都必须具有描述过程细节的能力，且能在编码阶段直接将它翻译为用程序设计语言书写的源程序。

正如本章开头所指出，传统软件工程技术目前仅在某些特定类型的软件开发中使用，但它们所蕴涵的思想与方法，仍经常在第二、三代软件工程的软件开发中反映出来。本章将这些技术集中介绍，目的是方便读者学习，并进而了解 3 代软件工程之间既不断发展、又一脉相承的关系。

习　题

1. 需求分析的任务是什么？怎样理解分析阶段的任务是决定“做什么”，而不是“怎样做”？
2. 需求分析要经过哪些步骤？
3. 什么是结构化分析？它的“结构化”体现在哪里？
4. 需求规格说明书由哪些部分组成？各部分的主要内容是什么？
5. DFD 和 CFD 有什么区别？
6. 为什么 DFD 要分层？画分层 DFD 要遵循哪些原则？
7. 选择一个系统（例如工资管理系统、飞机订票系统、图书馆管理系统等），用 SA 方法对它进行分析，并给出分析模型。
8. 简释 SC 图的作用。
9. 简释事务型结构。
10. 简释变换型结构。
11. 为什么事务型软件的结构常常具有中间大、两头小的形状？
12. 某事务系统具有下列功能：

（1）读入用户命令，并检查其有效性。

（2）按照命令的编号（1～4 号）进行分类处理。

（3）1 号命令计算产品工时，能根据用户给出的各种产品数量，计算出各工种的需要工时和缺额工时。

（4）2 号命令计算材料消耗，根据产品的材料定额和用户给出的生产数量，计算各种材料的需求量。

（5）3 号命令编制材料订货计划。

（6）4 号命令计算产品成本。

试用结构化分析和设计方法画出该系统的 DFD 图并据此导出系统的 SC 图。

13．简述模块详细说明书的主要内容。

14．简单比较本章讲解的几种模块设计表达工具的优缺点。

15．选一种排序（从小到大）算法，分别用流程图、N-S 图和 PDL 语言描述其详细过程。

16．试从指导原则、出发点、最终目标与适用范围等方面，比较面向数据流和面向数据结构两类设计方法的异同。