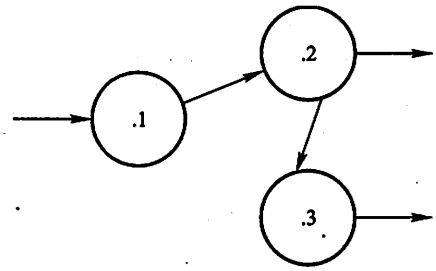
综上可知，分层 DFD 图产生了系统的全部数据和加工，通过对这些数据和加工的定义，常常对分析员提出一些新问题（例如前述的教材单价应从哪里取得），促使他们进行新的调查和思考，并可能导致对 DFD 的修改。画 DFD，定义加工和数据，再画，再定义，如此循环，直至产生一个为用户和分析员一致同意的文档——SRS。

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。

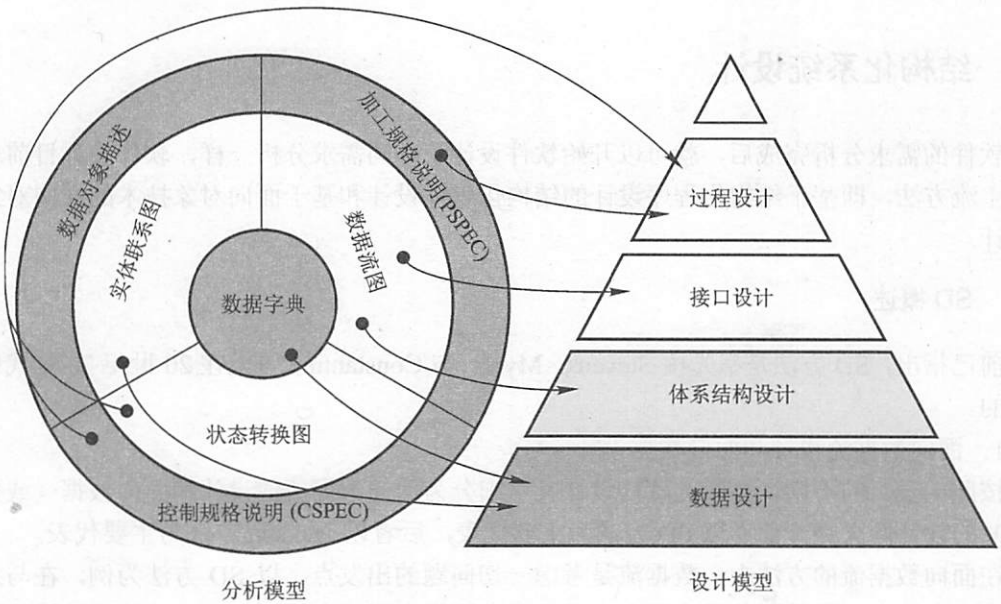


图 3.19　将分析模型转换为软件的设计模型

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。

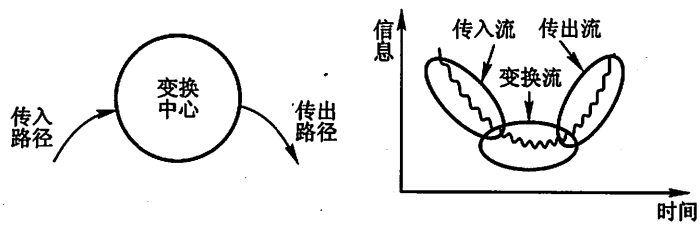


图 3.20 变换型结构的系统

（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。

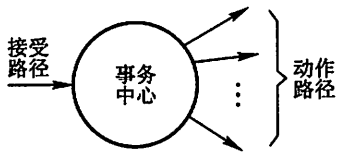


图 3.21　事务型结构的基本模型

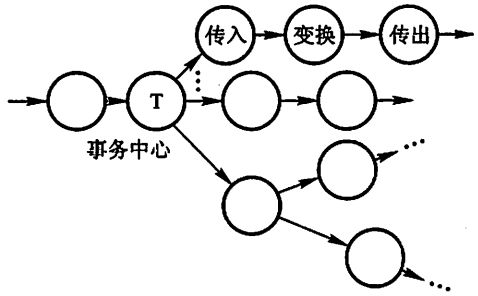


图 3.22　同时存在两类结构的系统

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。

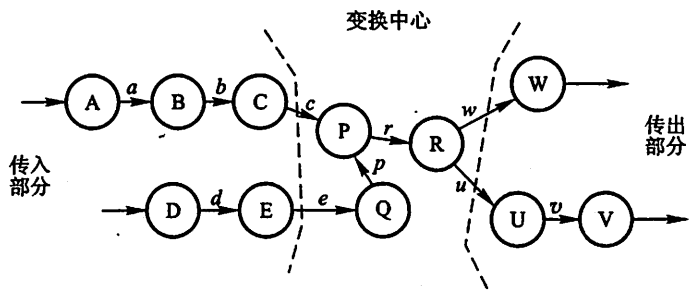


图 3.23　在 DFD 图上划分传入、传出和变换中心部分

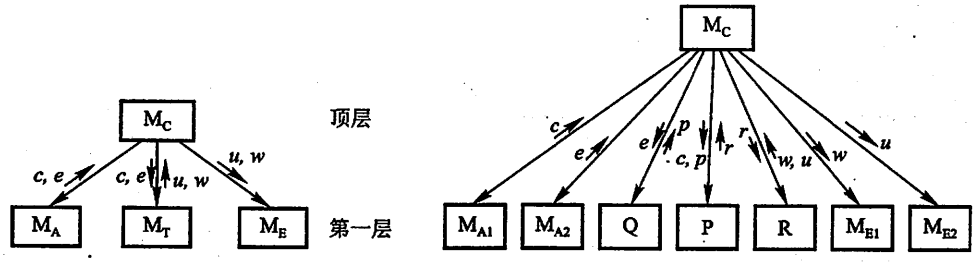
[解]　第一步：区分传入、传出和变换中心 3 个部分，在 DFD 图上标明它们的分界线。

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。

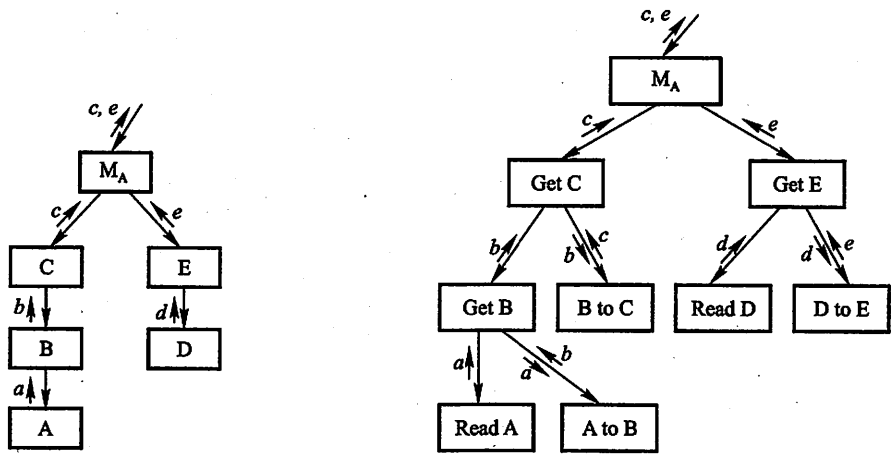
在图 3.25（a）中，传入模块 M_A 直接调用模块 C 与 E，以取得它所需的数据流 c 与 e。

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

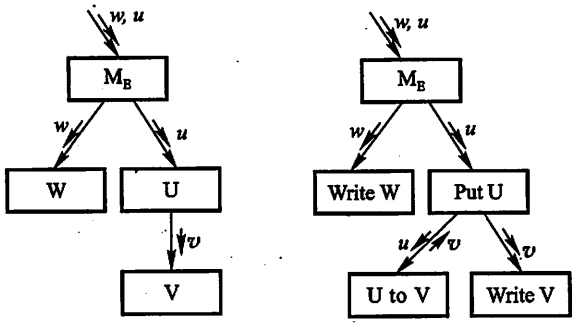
图 3.25　传入分支的分解

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

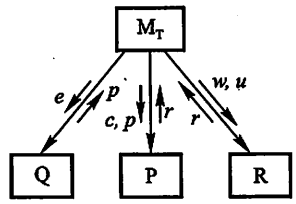


图 3.27　变换中心分支的分解

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。

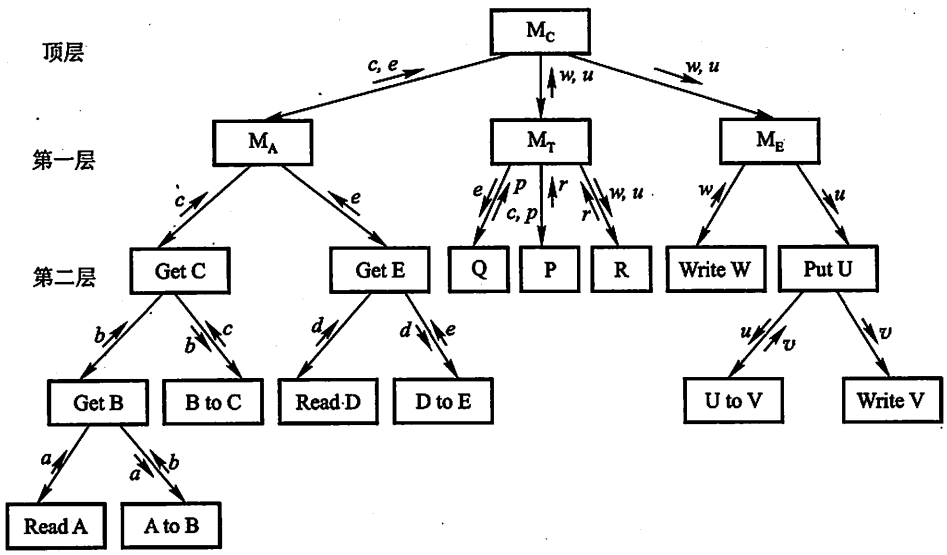


图 3.28　从图 3.23 导出的初始 SC 图

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。

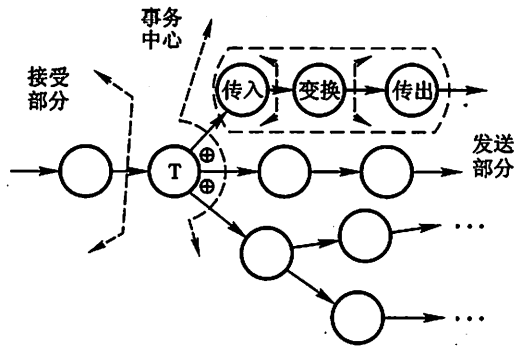


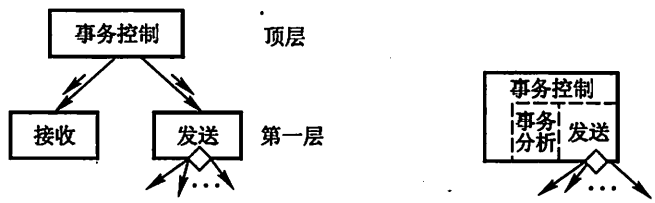
图 3.29 事务型 DFD 图的划分

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。

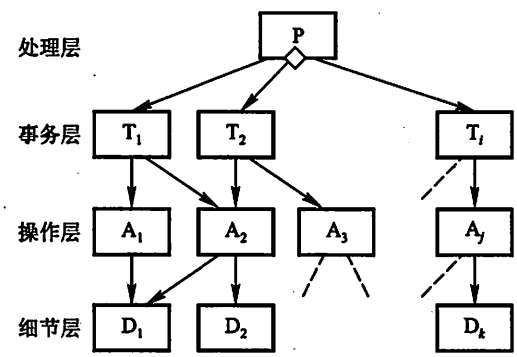


图 3.31 动作分支的典型结构

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。

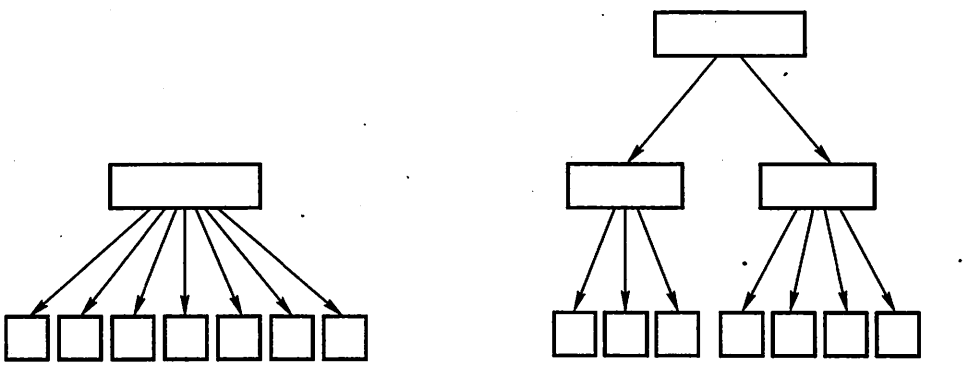
在改进 SC 图时，有些模块在图上的位置可能要上升、下降或左右移动，从而变更模块调用关系。模块位置应否变更，应视对计算机处理是否方便而定，不必拘泥于它与 DFD 图上对应的加工是否位置一致。

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。

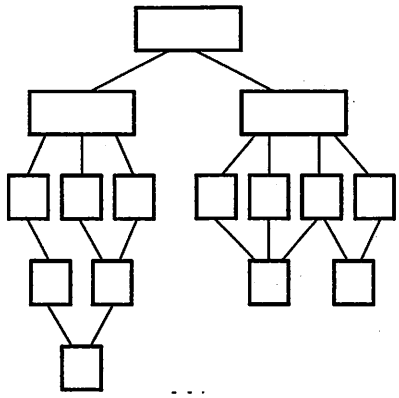


图 3.34 瓮形结构

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。

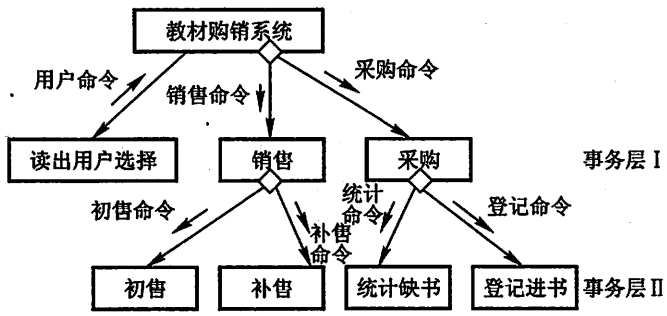


图 3.35 教材购销系统的上层框架

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。

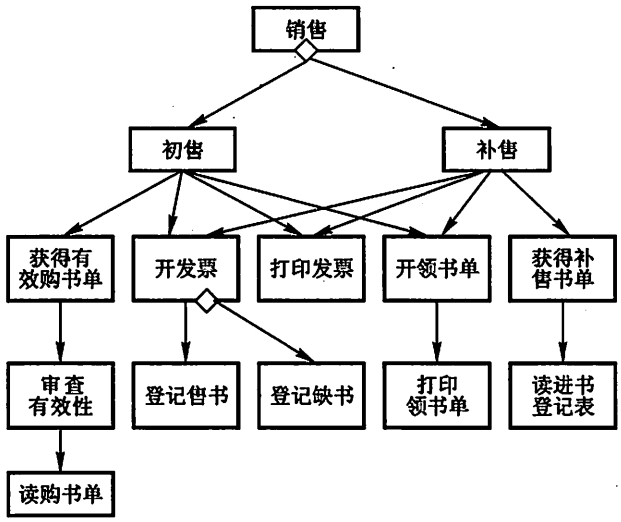


图 3.36　销售子系统初始 SC 图

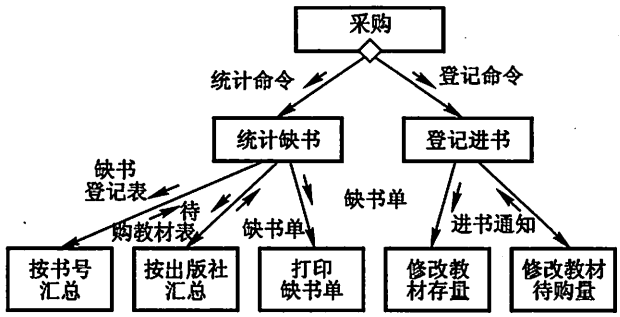


图 3.37　采购子系统初始 SC 图

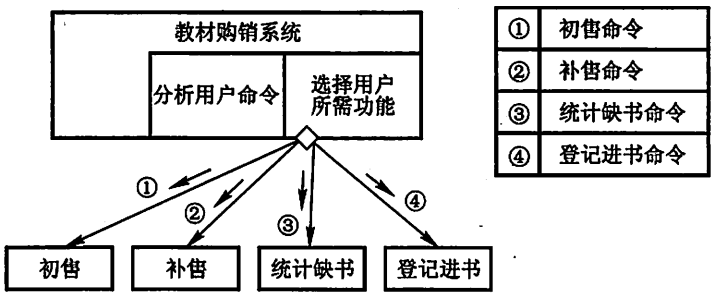


图 3.38　最终 SC 图的上层框架

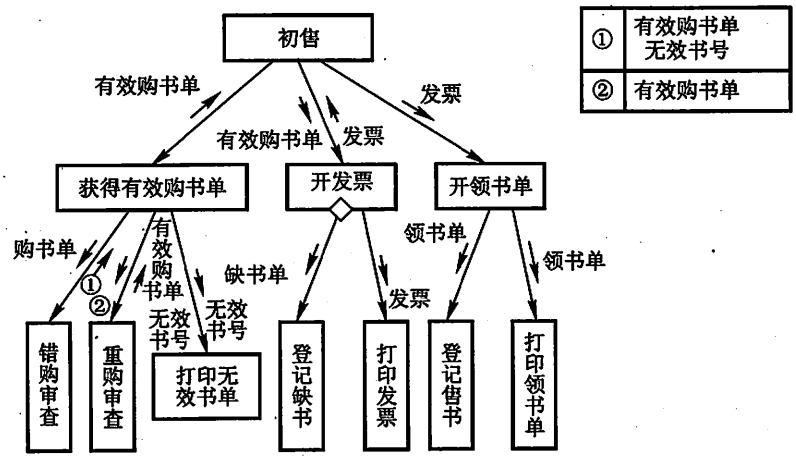


图 3.39 初售动作分支的最终 SC 图

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。