第5章 总体设计

总体设计的基本目标就是概要地回答系统应该如何实现。设计(Design)在任何工程产品或系统中是开发阶段的第一步。设计可以定义为应用各种技术和原理,对一个设备、一个过程或一个系统,做出足够详细的决策,使之有可能在物理上得以实现的过程。

系统的总体设计是在前面系统分析的基础上,为后期将要构造的系统实体建立一个模型(Model)或表达式(Representation)。与其他学科的工程设计方法一样,软件工程设计随着新理论、新方法的不断出现而继续发展。与其他技术领域比较,软件设计在它的发展中仍处于早期阶段。研究与分析软件设计问题才 30 年左右的时间。由此可见软件设计方法还缺少更为经典的工程设计学科所具有的深度、适应性(Flexibility)和定量性质。但是,已经有一些软件设计技术、设计质量准则及设计符号表示法。

5.1 软件设计的重要性

软件设计处于软件工程过程的技术核心地位。软件开发中不管应用什么样的开发模式(Development Paradigm),都要进行软件设计。当软件需求分析和定义完成后,就进入设计阶段。它是在对系统的信息、功能、行为和各种要求理解的基础上构想未来的系统。这种构想是否正确与完美,需要后面的编码阶段来构造,测试阶段来验证。软件设计、软件构造与验证这三项活动是必不可少的。每一项都是按一定形式变换信息,最终使之成为被确认的计算机软件。在软件工程过程中,这些技术阶段的信息流如图 5-1 所示。

由图 5-1 可以看出,在软件需求提供的信息(Information)、功能(Function)和行为(Behavior)模型上,设计阶段可以使用任何一种设计方法。设计阶段包括把分析阶段所建立的信息域模型变换为数据结构,这种数据结构是软件实现所需要的;也包括定义程序结构构件(Structural Components)之间相互关系的体系结构(Architecture)设计。另外还包括变换结构构件为软件的过程描述的过程(Procedure)设计。生成源代码并通过测试之后,进行软件的组装(Integrate)和确认(Validate)。

在设计中所做的决策将最终影响软件实现的成功与否,也影响软件维护的难易程度。所以,在软件设计过程中的这些决策是开发阶段非常关键的一步。

软件设计的重要性还反映在质量(Quality)上。在开发过程中,设计是对软件最本质的部分进行构造,构造的水平决定软件质量。同时,设计也提供了可以进行质量评价的软件表达式。只有通过设计,才能把用户的需求精确地转换为完美的软件系统。软件设计是整个

软件工程和软件维护的基础,如图 5-2 所示。

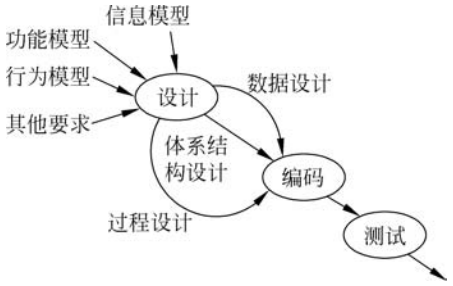


图 5-1 软件设计与软件工程

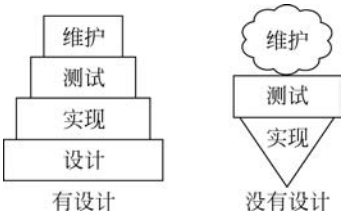


图 5-2 设计的重要性

对于一个软件系统,如果不进行设计而构造一个系统,可以肯定这个系统是不稳定的。这个系统即使发生很小的变动,都可能出现故障,而且很难测试,直到软件工程过程的最后,系统的质量仍无法评价。

5.2 设计过程

软件设计是一个把需求转换为软件表达式的过程。这个表达式过程一般情况下分为两步走。首先,这种表达式只是描绘一个软件的概貌。然后,将表达式细化为一个非常接近于源代码的设计表达式。从软件工程的角度讲,软件设计分为总体设计和详细设计。总体设计主要是把需求转换为数据结构和软件体系结构;而详细设计主要集中在体系结构表达式的细化上,从而产生详细的数据结构和软件的算法表达式。

在早期的设计工作中,软件设计着重在开发模块化程序模块所需要的准则,以及按照自顶向下(Top-Down)的方式逐步细化软件体系结构上。接着在设计定义的过程方面逐渐发展成为一种称为结构化编程(Structured Programming)的原则。之后,提出了把数据流和数据结构翻译成设计定义的方法。近年来,多采用 OO(Object-Oriented,面向对象)的设计方法。总结过去软件设计的发展,可以归纳为是一个持续发展的过程。

在比较小的软件设计中,可以把总体设计和详细设计作为一个过程阶段去完成。但是有一定规模的系统中,总体设计和详细设计是两个明确的阶段,所以它们中的许多设计活动是不同的。除了数据、体系结构和过程设计之外,在现代的许多应用中还包括界面设计活动。界面设计主要是建立人机之间界面的布局和交互的机制。

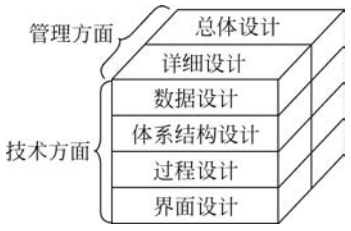


图 5-3 设计技术和管理方面之间的关系

总体设计和详细设计除了必须有先进的设计技术外,还要有同步的管理技术支持。用如图 5-3 所示的形式表明设计技术和管理方面之间的关系。

从图 5-3 可以看出,由技术支持的总体设计和详细设计都伴随着管理技术。

前面已经提到,软件设计的重要性之一就是软件的质量。在整个设计过程中,设计每一步的质量都要进行正式的技术评审(Formal Technical Reviews)。要按照设计准

则对设计表达式的质量进行评价。这里给出软件设计原则如下。

(1) 设计应当模块化(Modularity)。也就是说,软件应被逻辑地划分为能完成特定功能和子功能的构件。

(2) 设计应形成具有独立功能特征的模块(如子程序或过程)。

(3) 设计应使模块之间与外部环境之间接口的复杂性尽量地降低。

(4) 设计应该有一个分层的组织结构,这样人们可对软件各个构件进行理性的控制。

(5) 设计应有性质不同的、可区分的数据和过程表达式。

(6) 设计应利用软件需求分析中得到的信息和可重复的方法。

人们都希望设计一个良好的系统。然而,任何一个良好的系统设计都不是能轻易得到的。它是需要通过基本设计原理、系统化的方法和评审的各项技术的应用共同促成的。

5.3 软件总体设计

需求分析阶段所形成的数据流图是软件总体设计的基础。要为可供选择的每一个方案准备一份系统流程图,列出系统组成的物理元素,进行效益分析,制定实现方案的进度。从合理的方案中选择一个最佳的方案向用户推荐。当用户接受方案后,就要为这个最佳的方案设计软件结构。一般情况下,这个软件结构要通过反复修改使之合理。同时还要进行必要的数据库设计,在分布式系统中还要进行网络设计。另外,还要制订测试计划和确定测试要求。在详细设计前一定要进行软件总体设计。

软件总体设计阶段的任务是概要地回答系统应该如何实现,因此要把握其与详细设计的区别。要完成如下任务。

1. 软件系统结构设计

按照结构化理论,实现一个系统目标需要程序和数据,所以必须设计出组成这个系统的所有程序结构和数据库(文件)。具体方法如下。

(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。

(2) 确定每个模块的功能。

(3) 确定模块之间的调用关系。

(4) 确定模块之间的接口,即模块之间传递的信息。

(5) 评价模块结构的质量。

基于结构化理论的软件结构设计是以模块为基础的。在需求分析阶段,通过某种分析方法已经把系统分解成层次结构。在设计阶段,以需求分析的结果为依据,从实现的角度将需求分析的结果映射为模块,并组成模块的层次结构。

软件总体设计的关键是软件结构的设计,它直接影响到详细设计与编码的工作。软件结构的设计中要决定软件系统的质量及一些整体特性,因此软件结构的设计应由经验丰富的软件人员担任,采用一定的设计方法,选取合理的设计方案。

2. 数据结构及数据库设计

在结构化理论下的软件系统中,数据结构与数据库设计是非常重要的。数据库技术是

一项专门的技术,不是本书讨论的范围。但是作为软件开发人员要知道,在大型数据处理系统的功能分析与设计中,是要同时进行数据分析与数据设计的。

1) 数据结构的设计

根据需求分析阶段对系统数据的组成、操作约束和数据之间的关系的描述,确定数据结构特性。总体设计阶段利用逐步细化的方法对数据结构进行深入的设计,但是也不是像详细设计那样规定具体的实现细节。在总体设计阶段比较适宜使用抽象的数据类型,这些抽象的数据类型到详细设计阶段再用具体的数据结构描述其实现。如"栈"是数据结构的概念模型,在详细设计中可用线性表和链表来实现"栈"。设计有效的数据结构,将大大简化软件模块处理过程的设计。

2) 数据库的设计

一般的软件系统都有数据的存储,存储要借助数据库技术。数据库的设计是指数据存储文件的设计。设计包括以下三个方面。

(1) 概念设计。在数据分析的基础上,从用户角度采用自底向上的方法进行视图设计。一般用 E-R 模型来表示数据模型,这是一个概念模型。E-R 模型既是设计数据库的基础,也是设计数据结构的基础。IDEF1x 技术也支持概念模型,用 IDEF1x 方法建立系统的信息模型,使模型具有一致性、可扩展性和可变性等特性。同样,该模型可作为数据库设计的主要依据。

(2) 逻辑设计。E-R 模型或 IDEF1x 模型是独立于数据库管理系统(DBMS)的,要结合具体的 DBMS 特征来建立数据库的逻辑结构。对于关系 DBMS 来说,将概念结构转换为数据模式、子模式并进行规范,要给出数据结构的定义,即定义所含的数据项、类型、长度及它们之间的层次或相互关系的表格等。

(3) 物理设计。对于不同的 DBMS,物理环境不同,提供的存储结构与存取方法也各不相同。物理设计就是设计数据模式的一些物理细节,如数据项存储要求、存取方式和索引的建立等。

3. 网络系统设计

如果采用的是网络环境,则要进行网络系统设计。要分析网络负荷与容量,遵照网络系统设计原则,确定网络系统的需求。要进行网络结构设计,选择好网络操作系统,确定网络系统配置,制定网络拓扑结构。

4. 软件总体设计文档

总体设计说明书是总体设计阶段结束时提交的技术文档。按《计算机软件文档编制规范》(GB/T 8567—2006)规定,软件设计文档可分为"总体设计说明书""详细设计说明书"和"数据库设计说明书"。这些文档的内容与格式请参考有关资料。

5. 评审

在该阶段,对设计部分是否完整地实现了需求中规定的功能、性能等要求,设计方案的可行性、关键的处理及内外部接口定义正确性、有效性以及各部分之间的一致性等,都一一进行评审。

5.4 设计基本原理

软件设计要回答下列问题。

(1) 使用什么样的准则才能把软件划分成为各个单独的构件?

(2) 怎样把功能或数据结构的细节从软件概念表达式中分离出来?

(3) 定义软件设计的技术质量有统一的准则吗?

软件设计中最重要的一个问题就是软件质量问题,用什么标准对软件设计的技术质量进行衡量呢? 现在介绍软件发展中应用并经过时间考验的软件设计的一些基本原理。

5.4.1 抽象

抽象是认识复杂现象过程中使用的思维工具,即抽出事物本质的共同特性而暂不考虑它的细节,不考虑其他因素。当考虑用模块化的方法解决问题时,可以提出不同层次的抽象(Levels of Abstraction)。在抽象的最高层,可以使用问题环境的语言,以概括的方式描述问题的解。在抽象的较低层,则采用更过程化的方法,在描述问题的解时,面向问题的术语与面向实现的术语结合使用。最终,在抽象的最底层,可以用直接实现的方式来说明。软件工程实施中的每一步都可以看作是对软件抽象层次的一次细化。

随着对抽象不同层次的展开,过程抽象(Procedural Abstraction)和数据抽象(Data Abstraction)就建立了。所谓过程抽象,是指一个命名的指令序列,它具有一个特定的和受限的功能。例如有一个进入某场合的词"入口",对这个词进行分析,会发现其隐含了走到门口、伸出手、握住门把、旋转门把和推门、走进门这一系列的过程序列。数据抽象则是一个命名的说明数据对象的数据集合,例如一个部门员工的"工资单"。这个数据对象实际上是许多不同信息——单位、姓名、工资总额、扣除房租、水电费、煤气费、电话费、电视费、实得金额等的集合。在说明这个数据抽象名时,指的是所有数据。控制抽象(Control Abstraction)是软件设计中的第三种抽象形式。像过程抽象和数据抽象一样,控制抽象隐含了程序控制机制,而不必说明它的内部细节。控制抽象的例子如操作系统中用于进程协调活动的同步信号标。

许多编程语言(如 Ada、MODULA、CLU)都给出了建立抽象数据类型的机制(Mechanisms)。例如,Ada 的包(Package)就是一种支持数据抽象和过程抽象的编程语言机制。这种最初的抽象数据类型可以用作模板(Template)或类属(Generic)数据结构,由此导出的其他数据结构可以是它们的实例。

5.4.2 细化

逐步细化是一种自顶向下的设计策略。程序的体系结构开发是由过程细节层次不断地细化而成的。分层的开发则是以逐步的方式由分解一个宏功能直到获得编程语言语句。

在细化的每一步,已给定的程序的一条或几条指令被分解为更多细节的指令。当所有指令按计算机或编程语言写成时,这样不断地分解或规格说明的细化也将终止。随着任务的细化,数据也要细化、分解或结构化。程序的细化和数据的说明一并进行,这是很自然的事。

每一步细化都隐含着一定的设计决策。重要的是程序员应当通晓一些最基本的准则和存在的可选方案。

细化实际上是一个详细描述(Elaboration)的过程。在高层抽象定义时,从功能说明或信息描述开始,就是说明功能或信息的概念,而不给出功能内部的工作细节或信息的内部结构。细化则是设计者在原始说明的基础上进行详细说明,随着不断的细化(详细说明)给出更多的细节。

5.4.3 模块化

在计算机软件中,几乎所有的软件体系结构都要体现模块化。也就是说,所有的软件结构设计技术都是以模块化为基础的。模块由单独命名和可编址的构件集成,以满足问题的需求。

模块化的概念在程序设计技术中就出现了。何为模块?模块在程序中是数据说明、可执行语句等程序对象的集合,或者是单独命名和编址的元素,如高级语言中的过程、函数和子程序等。在软件的体系结构中,模块是可组合、分解和更换的单元。模块具有以下几种基本属性。

(1) 接口:指模块的输入与输出。

(2) 功能:指模块实现什么有利的作用。

(3) 逻辑:描述内部如何实现要求的功能及所需的数据。

(4) 状态:指该模块的运行环境,即模块的调用与被调用关系。

功能、状态与接口反映模块的外部特性,逻辑反映它的内部特性。

模块化是指解决一个复杂问题时自顶向下逐层把软件系统划分成若干模块的过程。每个模块完成一个特定的子功能,所有的模块按某种方法组装起来,成为一个整体,完成整个系统所要求的功能。

在面向对象设计中,模块和模块化的概念将进一步扩充。模块化是软件解决复杂问题所具备的手段。模块化是软件的一个重要属性,它使得一个程序易于被人们所理解、设计、测试和维护。如果一个软件就是一个模块,是很难让人理解的。因为这么多的控制路径,这么广的涉及范围,这么大量的变量,人们对这样复杂的软件进行了解、处理和管理,几乎是不可能的。

为了说明这一点,可看下面的论据。

设问题 x,表示它的复杂性函数为 $C(x)$,解决它所需的工作量函数为 $E(x)$。对于问题 P_1 和 P_2,如果

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

即 P_1 比 P_2 复杂,那么

$$E(P_1) > E(P_2)$$

即问题越复杂,所需要的工作量越大。

根据解决一般问题的经验,规律为

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

即一个问题同另一个问题组合而成的复杂度大于分别考虑每个问题的复杂度之和。这样,可以推出

$$E(P_1+P_2)>E(P_1)+E(P_2)$$

所得结果对于模块化和软件具有重要的意义。那么，从上面所得的不等式是否可以得出下面的结论：如果把软件无限细分，那么最后开发软件所需要的工作量就小得可以忽略了。但是，事实上影响软件开发工作量的因素还有许多，例如模块接口费用等，所以上述结论不成立。因为随着模块数目的增加，模块之间接口的复杂程度和为接口所需的工作量也在随之增加。上述不等式只能说明：当模块总数增加时，单独开发各个子模块的工作量之和会有所减少。根据这两个因素之间的关系，可以画出模块数量与软件成本的关系，如图 5-4 所示。从曲线看，存在着一个工作量最小或开发成本最小的模块数目 M 区。

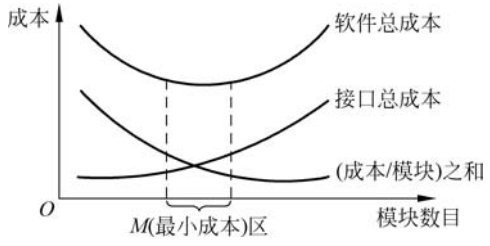


图 5-4 模块数量与软件成本

虽然现在还没有办法算出 M 的准确数值，但在考虑模块时，软件总成本曲线确实提供了非常有用的指导。这就是在模块化的过程中，还必须减小接口的复杂性。

应当指出，一个系统按模块化的概念来设计非常重要，即使它的实现必须是整体结构。有这样的情况（如实时软件/微处理器软件）：由于子程序（如子例程、过程）的引入而使极低的速度和过大的内存开销变得不可接受。在这种情况下，也应当把软件的模块化设计作为最基本的准则。代码可以逐行编写，纵然程序的源代码初看起来不是模块，但模块化的准则应当保持，这样的程序将会有模块系统的所有好处。

从图 5-4 可以看出，随着模块数目的增加，模块开发成本之和减少了，但是模块接口成本之和却增加了，所以模块数目必须适中。图 5-4 中 M 区是一个使软件开发总工作量最小的曲线区。

事实上，图 5-4 中所谓的软件总成本也只考虑了模块接口成本和子模块开发成本。模块的划分、设计还需要遵循其他设计原则。下面介绍模块设计的基本方法和优化原则。

5.4.4 软件体系结构

软件总体设计的主要任务就是软件结构的设计。软件体系结构（Software Architecture）包含了计算机程序的两个重要特性。

（1）过程构件（模块）的层次结构。

（2）数据结构。

软件体系结构通过过程的划分来导出，而这个过程与需求分析时定义的真实世界问题各部分的软件解有关。软件结构和数据结构的演化（Evolution）从问题定义开始。当问题的各部分分别由一个或多个软件元素求解时，问题解也就有了。图 5-5 给出了结构化演化的表示，它表示了软件需求分析到设计的转换。

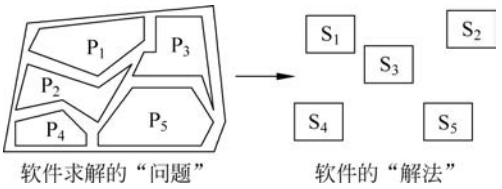


图 5-5 结构化演化

从图 5-6 所示的不同结构可以看出,一个问题可以有多种可供选择的结构,而选择某种结构又由软件设计方法来确定。

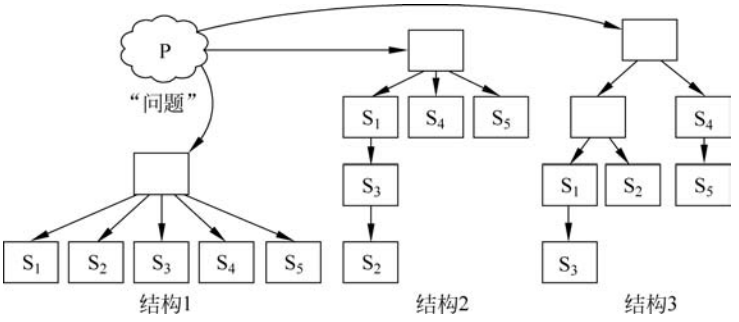


图 5-6 不同结构

由此可以看出,由于各种设计方法的原理不同,因此同一个软件需求也会导出不同的软件结构。那么,当一个问题的软件结构导出后,如何评价其不同结构的优劣?没有一个形式化的准则,因而很难回答。问题是目前还不能对它们做出准确的定量评价。但是,在稍后部分进行讨论体系结构设计的结构特征时,可以通过分析的方法来确定它们的整体质量。

5.4.5 程序结构

程序结构(Program Structure)给出了程序构件(模块)的组织(通常称为分层),这种组织包含了控制的层次。它们不给出软件的过程方面,如过程的序列、决策的出现或次序,以及操作的重复等。

程序结构可以用许多不同的符号来表示。最常用的是如图 5-7 所示的树状结构。其他符号(如具有相同作用的 Warnier 图和 Jackson 图)也可以使用。为了方便讨论结构,定义了一些简单的度量和术语,如图 5-7 所示。

从图 5-7 可见,深度(Depth)表示控制的层次,宽度(Width)表示同一层次上控制的最大数,扇出(Fan-out)是对一个模块直接控制其他模块数目的度量,扇入(Fan-in)则是对一个给定模块被多少个模块直接控制模块的度量。

模块之间的控制关系可用下面的方法表示:控制另一模块的模块称为上级模块(Superordinate Module);反过来,被另一模块控制的模块称为从属模块(Subordinate Module)。例如图 5-7 中,模块 m 是模块 n、o、p、q 的上级模块,模块 f、g 是模块 d 的从属模块,同时也是模块 M 的从属模块。宽度方向上的关系(如模块 d 与 e)虽然也可以表示,但实际没有需要用明确的专门名词来定义。

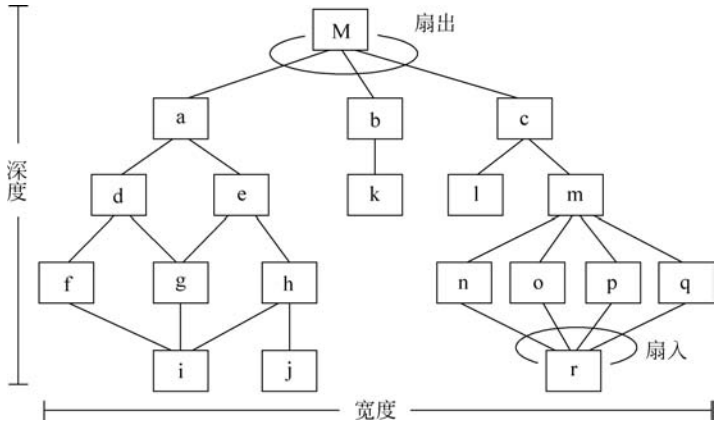


图 5-7 树状结构

根据图 5-7 所示的结构，可以给出软件体系结构特征：可见性(Visibility)和连接性(Connectivity)。所谓可见性，表明该程序构件集合可以引用或使用一个给定构件作为数据，即使是间接完成时；而连接性表明该构件集合直接引用或使用一个给定构件作为数据。两种特征是难以分辨的。

5.4.6 数据结构

在软件体系结构的表达式中，数据结构与程序结构同样重要。数据结构决定信息的组织、存取方法、结合的程度，以及可选的处理方法。这里只给出一些概念，因为它有助于更好地理解这些传统组织信息的方法，以及怎样从基层支持信息层次。这些概念都是重要的。对此不深入讨论，仅给出数据结构的定义，它是一种数据各元素之间逻辑关系的表达式。

一个数据结构的组织和复杂性只受设计者创造性的限制，但是典型的数据结构可以组成更复杂的结构模块。这些典型的数据结构如图 5-8 所示。

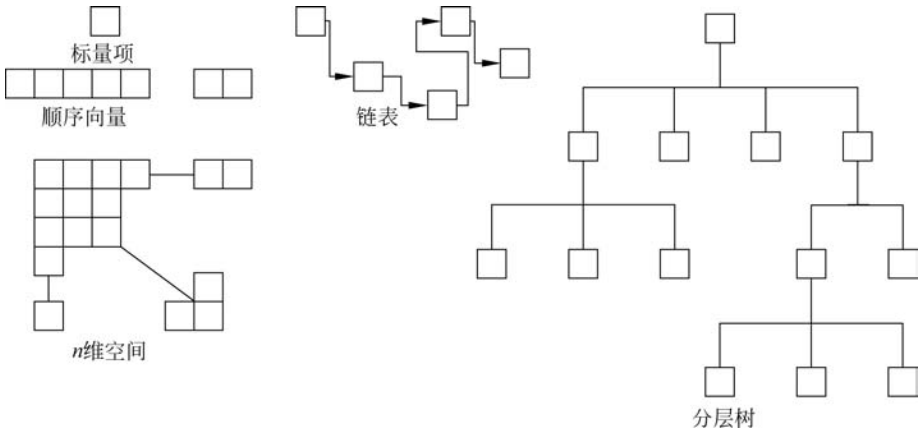


图 5-8 典型的数据结构

标量项(Scalar Item)表示一个用标识符标识的单元信息元素。就是说，在存储器中用一个确定的单一地址就可得到。一个标量项是所有数据结构中最简单的一种。标量项的规

模和格式在一种编程语言中所确定的边界内可以有变化。例如,标量项可以是一个长度为1位的逻辑实体,或者是一个长度为8～64位的整型数或浮点数,甚至是一个长度为几百或几千字节的字符串。

当组织标量项作为一列或连接的组时,就形成一个顺序向量(Sequential Vector)。向量是数据结构中最常用的。可看下面这个C语言程序段的例子:

```
int aa[100];
…
procedure ps (int aa; int n; int sum)
{ int i;
{
    sum = 0;
    for (i = 1;i <= n ;i++)
      sum = sum + aa[i];
};
}
…
```

例子中定义aa为100个标量整数项的顺序向量。在过程ps中标引aa每个元素的存储,这样数据结构的每个元素都可按定义的顺序引用。

如果把顺序向量扩展为多维时,就构成了一个n维的空间(n-Dimensional Space)。在大多数编程语言中,把n维空间称为数组(Array)。标量、向量和空间可以用不同的格式组织。链表(Linked List)就是一种非邻接的标量的组织。向量或空间用一定的方式能使它们作为一个表来处理。每个节点具有适当的组织、一个或多个指针,这个指针表明表中下一个节点在节点存储器中的地址。在链表的结构中,为了适应新表入口的需要,可以在表中的任意点上增加重新定义的指针。

利用上述的基本数据结构可以构造出其他的数据结构。例如,使用包含标量项、向量和可能的n维空间的多链表可以实现层次型的数据结构(Hierarchical Data Structure)。层次型的结构通常在要求信息分类和组合性中应用。这里分类指的是包含了一组由一些类属组成的分类。组合性包含从不同的类组合信息的能力。例如,在微处理器中找出所有价格低于1000美元、主频1GHz和销售商的条目种类。

数据结构可以给出不同层次的抽象。例如,栈(Stack)是数据结构的一个概念模型,它可以由向量或链表来实现。栈内部工作情况取决于设计细节的层次,可以说明,也可以不说明。

5.4.7 软件过程

在讨论软件结构时不考虑处理和决策以及顺序定义的控制层次,而软件过程(Software Procedure)(见图5-9)则侧重于每一个单独模块的处理细节研究。过程必须提供精确的事件的顺序、确切的抉择点、重复的操作,以及数据的组织与结构处理规格说明。

当然,结构与过程是相互关联的。对每个模块所规定的处理都必须包括说明该模块的所有从属模块。软件过程的表示也是分层的,图5-10所示为过程的分层。

过程设计中要用到模块。规定和设计模块应当包含模块内过程和数据的信息,对于其

他不需要这些信息的模块是不可访问的。有效的模块化可以通过定义一组独立的模块来达到。

独立的模块彼此之间仅仅交换那些为了完成系统功能所必需的信息，因为绝大多数数据和过程是软件其他部分不可访问的。这样规定和设计的模块会带来极大的好处。

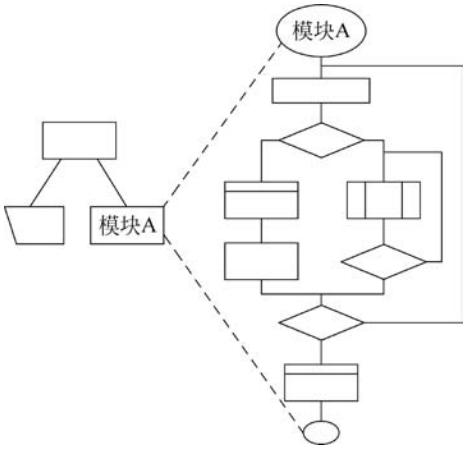


图 5-9 一个模块内的过程

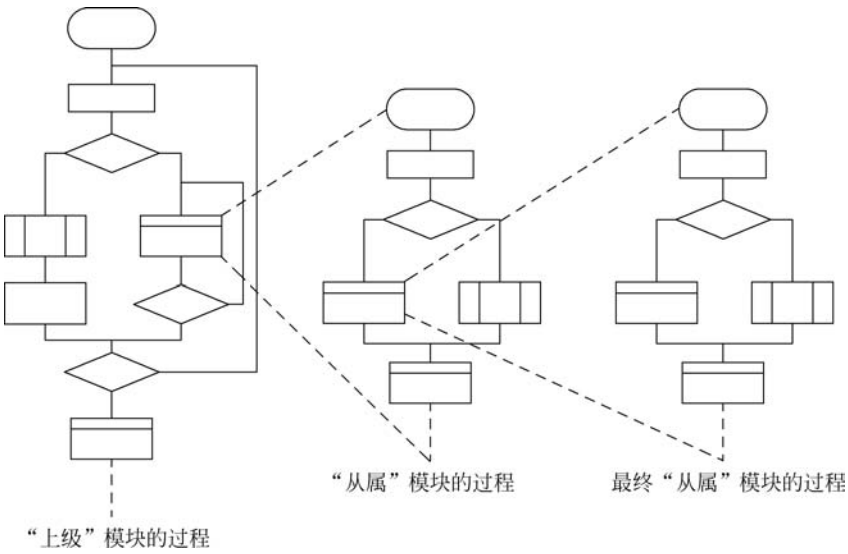


图 5-10 过程的分层

5.5 体系结构设计

软件体系结构设计(Architectural Design)的主要目标是设计一个模块化的程序结构。体系结构设计融合了程序结构和数据结构，接口定义能使数据流经程序。要给出各个模块之间的控制关系。

5.5.1 软件结构图

软件结构图是软件系统的模块层次结构,反映了整个系统的功能实现,即将来程序的控制层次体系。对于一个“问题”,可用不同的软件结构来解决,不同的设计方法、不同的划分和组织,可得出不同的软件结构。

软件结构往往用树状或网状结构的图形来表示。软件工程中,一般采用 20 世纪 70 年代中期美国 Yourdon 等提出的称为结构图(Structure Chart,SC)的工具来表示软件结构。结构图的主要内容如下。

1. 模块

模块用方框表示,并用名字标识该模块,名字应体现该模块的功能。

2. 模块的控制关系

两个模块间用单向箭头或直线连接起来表示它们的控制关系,如图 5-11 所示。按照惯例,总是图中位于上方的模块调用下方的模块,所以不用箭头也不会产生二义性。

调用模块和被调用模块的关系称为上属与下属的关系,或者称为“统率”与“从属”的关系。在图 5-7 中,模块 M 统率模块 a、b、c; 模块 d 从属于模块 a,也从属于模块 M。

3. 模块间的信息传递

模块间还经常用带注释的短箭头表示模块调用过程中来回传递的信息,如图 5-11 所示。

4. 两个附加符号

表示模块有选择调用或循环调用,如图 5-12 所示。

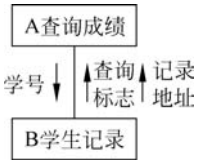


图 5-11 模块间的控制关系及信息传递

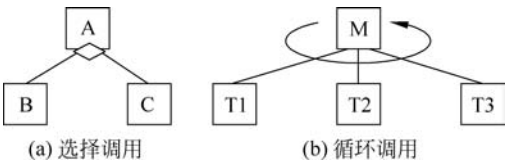


图 5-12 选择调用和循环调用的表示

图 5-12(a)所示为选择调用,A 模块中下方有一个菱形符号,表示 A 中有判断处理功能,它有条件地调用 B 或 C; 图 5-12(b)所示为循环调用,其中 M 模块下方有一个弧形箭头,表示 M 循环调用 T1、T2 和 T3 模块。

5. 结构图的形态特征

为了讨论结构图的特征,将如图 5-7 所示的树状结构图重画,如图 5-13 所示。

结构图的形态特征包括以下几个。

(1) 深度: 指结构图控制的层次,即模块的层数。图 5-13 中,结构图的深度为 5。

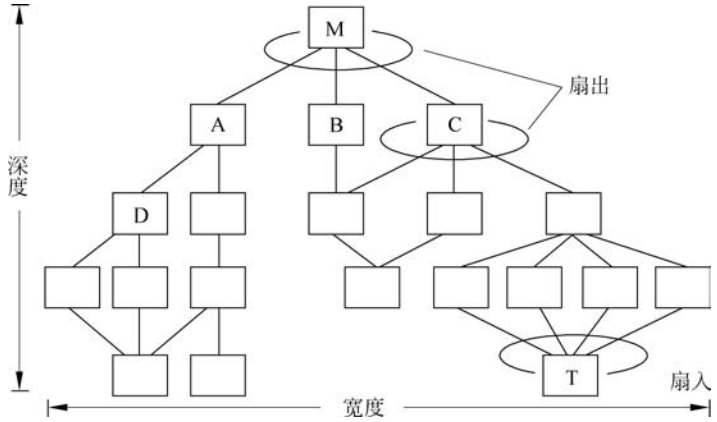


图 5-13 结构图示例

(2) 宽度：指一层中最大的模块个数。图 5-13 中，宽度为 8。

(3) 扇出：指一个模块直接下属模块的个数。图 5-13 中，模块 M 的扇出为 3。

(4) 扇入：指一个模块直接上属模块的个数。图 5-13 中，模块 T 的扇入为 4。

6. 画结构图应注意的事项

(1) 同一名字的模块在结构图中仅出现一次。

(2) 调用关系只能从上到下。

(3) 不严格表示模块的调用次序，习惯上从左到右。有时为了减少连线的交叉，适当地调整同一层模块的左右位置，以保持结构图的清晰性。

5.5.2 模块的大小

前面在讨论模块设计的原理时，已经知道一个系统应当由若干模块构成，其目的是降低系统的复杂度，但是并没有一个明确的准则说明模块应当多大才合适。有些教科书上说明一个模块最好只包含 50～60 条语句(即可打印在一页打印纸上)，这是考虑到开发人员能方便地对设计的模块进行阅读和研究。如果模块的规模增加为 100 条语句，甚至达到几百条语句，那么阅读和研究就比较困难了。目前，国内外关于模块大小的规定也不一样，最多允许一个模块含有 500 条语句。但是过小的模块也不一定好，因为调用子程序入口和出口需要做附加操作。如果接口复杂，这种附加操作可能比子程序本身的操作还要多，那么这样就不合适。

模块设计的准则不应该是语句的多少，而应该是模块是否是一个独立的功能。而且多个上级模块需要调用它，若不设计为单独模块，就要重复多交，这样不仅使程序增大，测试和维护也不方便。在这种情况下，防止影响运行速度的方法可以用类似于 C 语言的 include 语句或汇编语言中的宏功能来解决。

5.5.3 扇出和扇入与深度和宽度

由结构图的形态特征可以知道，一个系统的大小和系统的复杂程度在一定程度上可以用深度和宽度表示。因此可以推理，系统越大越复杂，其深度和宽度显然也越大。而深度与

程序的语句效率和模块大小的划分有关。设计者在结构设计过程中主要关心的是模块的高聚合和低耦合,以及模块的规模。所以,实际上,深度只是对结构设计好坏的一种测度,例如一个程序有 100 条语句,如果将其划分为 20 个模块并用 20 层来调用,则肯定分解过多。

在讨论宽度时,注意到与其相关的最大因素是模块的扇出。从图 5-7 中了解到,如果扇出过大,则使它们上级模块需要过多地控制这些从属模块而增加复杂性,也增加了设计人员在设计过程中的难度。根据历史的经验,扇出一般最好控制为 3～4 个,不要超过 7 个。从讨论深度的过程可以知道,扇出过小,会增加结构的深度。

扇出实际上是对问题解的分解。分解过程中需要考虑的问题就是前面是否已经有一个模块与当前所需要的模块功能相同或相似。若完全相同,则可以共享;若功能类似,则应区分哪些部分相同,这样可以把相同的部分分离出来成为单独的模块,如图 5-14～图 5-16 所示。

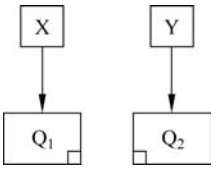


图 5-14 分解模块示意 1

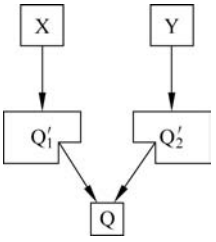


图 5-15 分解模块示意 2

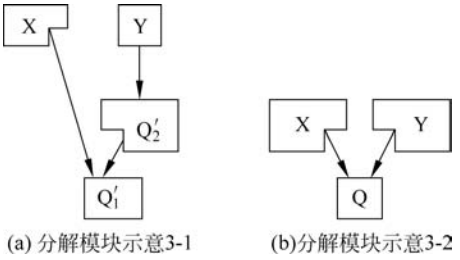


图 5-16 分解模块示意 3

如图 5-14 所示,Q_2 中与已有的 Q_1 相似;图 5-15 把 Q 中相同的部分分离出来;图 5-16(a)中如果 Q 很小,可并入 X,Q_2';图 5-16(b)中如果 Q 也很小,也可以并入 Y,X。

大量的系统研究表明,高层模块应有较高的扇出,低层模块特别是底层模块应有较高的扇入。扇入越大,表示该模块被更多的上级模块共享。多个扇入入口相同,可以避免程序的重复,因此希望扇入高一点。但扇入模块过多又可能是把许多不相关的功能硬凑在一起,形成通用模块,这样的模块必然是低聚合的。

5.5.4 模块的耦合

耦合(Coupling)表示软件结构内不同模块彼此之间相互依赖(连接)的紧密程度,是衡量软件模块结构质量好坏的度量,是对模块独立性的直接衡量指标。软件设计应追求尽可能松散耦合,避免强耦合。模块的耦合越松散,模块间的联系就越小,模块的独立性也就越强。这样,对模块进行测试、维护就越容易,错误传播的可能性就越小。

耦合强弱取决于模块间接口的复杂程度、进入或访问一个模块的点,以及通过接口的数据。如果两个模块中每个模块都能独立地工作,而不需要另一个模块的存在,那么它们彼此之间完全独立,没有任何联系,也无所谓耦合。但是,在一个软件系统内不可能所有模块之间都没有任何连接。一般可以将模块的耦合分成四类:数据耦合、控制耦合、公共环境耦合和内容耦合。

1. 数据耦合

如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合称为数据耦合。

数据耦合是低耦合。系统中必须存在这种耦合，因为只有当某些模块的输出数据作为另一些模块的输入数据时，系统才能完成有价值的功能。

一般说来，一个系统内可以只包含数据耦合。图 5-17 说明了模块 B 与模块 C 的调用关系是数据耦合。

2. 控制耦合

如果传递的信息中有控制信息，则这种耦合称为控制耦合，如图 5-18 所示。控制耦合是中等程度的耦合，它增加了系统的复杂程度。

控制耦合往往是多余的，在把模块适当分解之后通常可以用数据耦合代替它。例如，图 5-18 中模块 B 的内部处理逻辑判断决定是执行 F_1、F_2 还是执行 F_n，这要取决于模块 A 传来的信息“标志”Flag。控制耦合是中等程度的耦合。

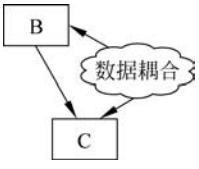


图 5-17 数据耦合

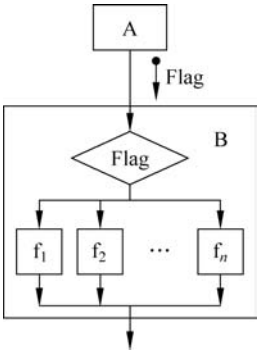


图 5-18 控制耦合

3. 公共环境耦合

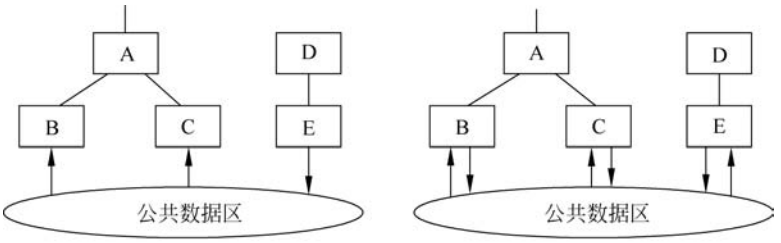
当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合(即公用耦合)。公共环境可以是全局变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等。

公共环境耦合的复杂程度随耦合的模块个数而变化，当耦合的模块个数增加时，复杂程度显著增加。如果只有两个模块有公共环境，那么这种耦合有下述两种可能，如图 5-19 所示。

(1) 一个模块往公共环境送数据，另一个模块从公共环境取数据。这是数据耦合的一种形式，是比较松散的耦合。

(2) 两个模块都既向公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。

如果两个模块共享的数据很多，都通过参数传递可能很不方便，这时可以利用公共环境耦合。

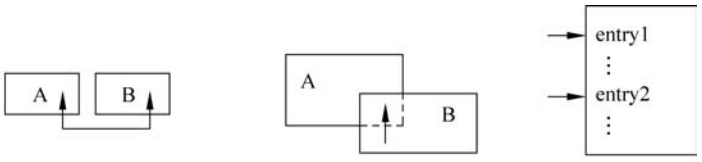


(a) 松散的公共环境耦合　　(b) 紧密的公共环境耦合

图 5-19　公共环境耦合

4. 内容耦合

最高程度的耦合是内容耦合。如图 5-20 所示,两个模块间就发生了内容耦合。



(a) 进入另一个模块内部　　(b) 代码重复　　(c) 多入口

图 5-20　内容耦合

(1) 一个模块访问另一个模块的内部数据;

(2) 一个模块不通过正常入口而转到另一个模块的内部;

(3) 两个模块有一部分程序代码重叠(只可能出现在汇编程序中);

(4) 一个模块有多个入口(这表明一个模块有几种功能)。

应该坚决避免使用内容耦合。事实上许多高级程序设计语言已经设计成不允许在程序中出现任何形式的内容耦合。

总之,耦合是影响模块结构和软件复杂程度的一个重要因素,应该遵循如下设计原则:尽量使用数据耦合,少用控制耦合,限制用公共环境耦合,完全不用内容耦合。

5.5.5　模块的内聚

内聚标志一个模块内各个元素彼此结合的紧密程度,它是信息隐蔽和局部化概念的自然扩展。简单地说,理想内聚的模块只做一件事情。

设计时应该力求做到高内聚,通常中等程度的内聚也是可以采用的,而且效果和高内聚相差不多,但是不要使用低内聚。

内聚和耦合是密切相关的,模块内的高内聚往往意味着模块间的耦合。内聚和耦合都是进行模块化设计的有力工具,但是实践表明内聚更重要,应该把更多注意力集中到提高模块的内聚程度上。

根据模块内部的构成情况,模块的内聚可以划分成高、中、低三大类。常见的内聚可分为功能内聚、信息内聚、通信内聚、过程内聚、时间内聚、逻辑内聚、偶然内聚七类。它们的内聚程度依次从高到低,一般认为功能内聚和信息内聚是高内聚,通信内聚、过程内聚是中内

聚，时间内聚、逻辑内聚和偶然内聚是低内聚。

1. 功能内聚

如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

2. 信息内聚

信息内聚模块能完成多种功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点，例如图 5-21 所示的信息内聚有四个功能，即这个模块将根据不同的要求，确定该执行哪一功能。但这个模块都基于同一数据结构，即符号表。

3. 通信内聚

如果一个模块中所有处理元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚(Communicational Cohesion)。例如图 5-22 所示的模块 A 的处理单元是由同一数据文件的数据产生不同的表格。通信内聚有时也称数据内聚。

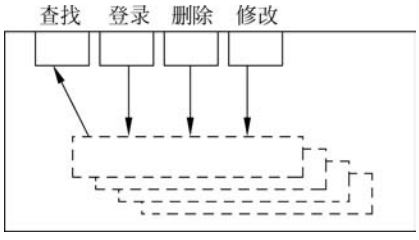


图 5-21 信息内聚

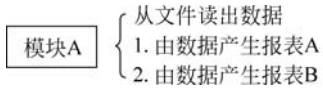


图 5-22 通信内聚

4. 过程内聚

如果一个模块内部的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。使用程序流程图作为工具设计软件时，常常通过研究流程图确定模块的划分，这样得到的往往是过程内聚的模块。

5. 时间内聚

如果一个模块包含的任务必须在同一段时间内执行，就称为时间内聚。如图 5-23 所示，紧急故障处理模块中的关闭文件、报警、保护现场等任务都必须无中断地同时处理，这就是时间内聚。

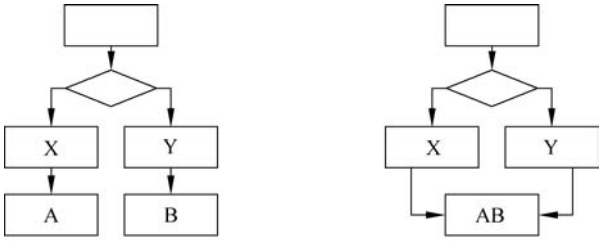
图 5-23 时间内聚

在逻辑内聚的模块中，不同功能混在一起，合用部分程序代码，即使局部功能的修改有时也会影响全局。因此，这类模块的修改也比较困难。

时间关系在一定程度上反映了程序的某些实质，所以时间内聚比逻辑内聚好一些。

6. 逻辑内聚

如果一个模块完成的任务在逻辑上属于相同或相似的一类,则称为逻辑内聚(Logical Cohesion)。对逻辑内聚模块的调用,常常需要有一个功能开关,由上层调用模块向它发出一个控制信号,在多个关联性功能中选择执行某一个功能。这种内聚较差,增加了模块之间的联系,不易修改。将图 5-24(a)中模块 A、B 合并成图 5-24(b)中的模块 AB,那么 AB 就是一个逻辑内聚模块。在此结构中必须增加一个开关量传递,使这些模块之间联系程度增加;此外增加修改难度,例如如果模块 X 需要修改 AB 中某共用段,而其他模块 Y 和 Z 却不希望修改。



(a) 合并前　　(b) 合并后

图 5-24　逻辑内聚

7. 偶然内聚

如果一个模块完成一组任务,这些任务彼此间即使有关系,关系也是很松散的,就称为偶然内聚。

常犯这种错误的一种情况:有时在写完程序后,发现一组语句在多处出现,于是为了节省空间而将这些语句作为一个模块设计,这就出现了偶然内聚。例如,在图 5-25 中,模块 A、B、C 中出现公共代码段 W,于是将 W 独立成一个模块,而 W 中这些语句并没有任何联系。如果在测试中发现模块 A 不需要做 X=Y+Z,而应做 X=Y * Z,那么此时对 W 的维护就很困难了。

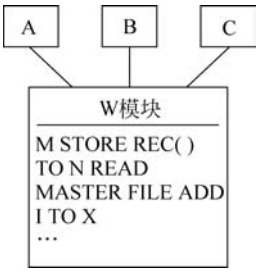


图 5-25　偶然内聚

模块功能划分的粗细是相对的,所以模块的内聚程度也是相对概念。实际上,很难精确确定内聚的级别,重要的是在软件设计中应力求做到高内聚,尽量少用中内聚,不用低内聚。一般地,在系统较高层次上的模块功能较复杂,内聚要低一些;而较低层次上的模块内聚程度较高,达到功能内聚的可能性比较大。

5.5.6　结构设计的一般准则

前面的模块概念实际上已给出了模块设计的一些基本原则。在此基础上,这里介绍几条模块设计与优化准则。这些准则是以后软件结构、求精和复查的重要依据和方法。

1. 模块独立性准则

划分模块时,尽量做到高内聚、低耦合,保持模块相对独立性,并以此原则优化初始的软

件结构。

(1) 如果若干模块之间耦合强度过高,每个模块内功能不复杂,则可将它们合并,以减少信息的传递和公共区的引用。

(2) 若有多个相关模块,应对它们的功能进行分析,消去重复功能。

评价软件的初始结构,通过模块的分解和合并,减少模块间的联系(耦合),增加模块内的联系(内聚)。例如,多个模块共有一个子功能可以独立成一个模块。这些模块调用,有时可以通过分解或合并,以减少控制信息的传递及对全程数据的引用,并且降低接口的复杂程度。图 5-26 所示为模块的分解,图 5-27 所示为模块的合并。

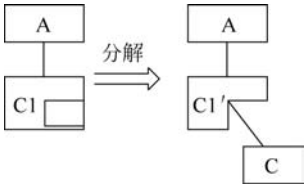


图 5-26 模块的分解

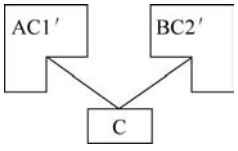


图 5-27 模块的合并

2. 软件结构的形态特征准则

软件结构的深度、宽度、扇入及扇出应适当。

深度是软件结构设计完成后观察到的情况,能粗略地反映系统的规模和复杂程度,宽度也能反映系统的复杂情况。宽度与模块的扇出有关,一个模块的扇出太多,说明本模块过分复杂,缺少中间层。

单一功能模块的扇入数大比较好,说明本模块为上层几个模块共享的公用模块,重用率高。但是不能把彼此无关的功能凑在一起形成一个通用的超级模块,虽然它扇入高,但却低内聚。因此非单一功能的模块扇入高时应重新分解,以消除控制耦合的情况。软件结构从形态上看,应是顶层扇出数较高一些,中间层扇出数较低一些,底层扇入数较高一些。

3. 模块的大小准则

在考虑模块独立性的同时,为了增加可理解性,模块的大小最好为 50～150 条语句,可以用 1～2 页打印纸打印,便于人们阅读与研究。

但是,在进行模块设计时,首先应按模块的独立性来选取模块的规模。例如,如果某个模块功能是一个独立的少于 50 行的程序段,则不要嫌小而去与其他内容拼凑成 50 行的模块;如果一个具有独立功能的程序段占用 1 页半,也不要嫌大而将它划分成两个模块。

应该注意的是,这种用代码行数来衡量模块大小的方法只适合于传统的程序,现代程序的概念已经有了较大的变化,特别是第四代语言(4GL)已不能再用代码的长度来说明一个模块的规模大小和复杂程度了。所以,模块的规模大小主要还是要根据其功能来判断。

4. 模块的接口准则

模块的接口要简单、清晰,含义明确,便于理解,易于实现、测试与维护。

模块接口的复杂性是软件发生错误的一个重要原因。因此,设计模块接口时,应尽量使

传递的信息简单并与模块的功能一致。下面用一个简单例子说明接口复杂性问题。

下面是两个求一元二次方程根的程序模块的例子。

程序模块 1：QUAD-ROOT(TBL,X)

这里使用数组 TBL 带入方程的系数：TBL(1)＝A,TBL(2)＝B,TBL(3)＝C,数组 X 返回方程的根。

程序模块 2：QUAD-ROOT(A,B,C,ROOT1,ROOT2)

对模块 1 而言,接口 TBL 和 X 的意义不明确,而模块 2 的接口简单明了,又与模块功能一致。所以模块 2 比模块 1 的接口复杂程度要低。

5.5.7 模块的作用域与控制域

一个模块的作用范围应在其控制范畴之内,且条件判定所在的模块应与受影响的模块在层次上尽量靠近。

在软件结构中,由于存在着不同事务处理的需要,某一层上的模块会存在着判断处理,这样可能影响其他层的模块处理。为了保证含有判定功能模块软件设计的质量,引入了模块的作用范围(或称影响范围)与控制范围的概念。

图 5-28～图 5-30 给出了三种模块结构图,图中阴影框表示判断影响的模块。它们的作用域都没有超出控制域。但是仅从作用域与判断点位置来看,图 5-28 的判断点在层次结构中位置太高,不太理想。判断点的作用范围超过了控制范围,这种结构最差。这种结构增加了数据的传递量和模块间的耦合,会影响不受它控制的其他模块,这样的结构不易理解与维护。

图 5-29 中的判断模块较适中。判断模块的作用范围在控制范围内,但是判断所在模块与受判定影响的模块位置太远,也存在着额外的数据传递,增加了接口的复杂性和耦合强度。这种结构虽符合设计原则,但不理想。

图 5-30 中的作用域是其直接下层模块,消除了额外的数据传递,是最理想的结构图。

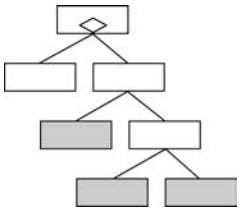


图 5-28 模块示意图 1

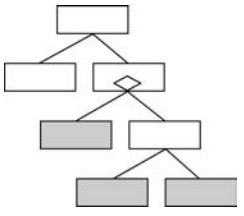


图 5-29 模块示意图 2

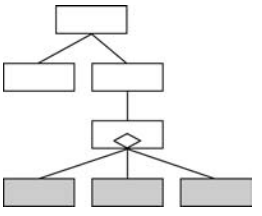


图 5-30 模块示意图 3

如果在设计过程中,发现模块作用范围不在其控制范围之内,可用以下方法加以改进。

(1) 上移判断点,使该判断的层次升高,以扩大它的控制范围。

(2) 下移受判断影响的模块。将受判断影响的模块下移到判断所在模块的控制范围内。

前面所讨论的原则与功能设计是有关系的。模块的功能应该能够预测,也要防止模块功能的过分局限。当一个模块输入的数据相同时就产生同样的输出,那么这个功能模块就是可以预测的。但是要注意,带有内存储的模块的功能可能是不可预测的,因为它的输出可能取决于内存储器的状态。由于内存储器对于上级模块而言是不可见的,因此这样的模块

难以测试与维护。

当一个模块仅完成一项功能，则表现为高内聚。如果一个模块限制局部数据结构的大小，过度限制其在控制流中可以做出的选择或外部接口模式，那么，这种模块的功能就过于局限。

5.6 结构化设计

结构化设计是以结构化分析产生的数据流图为基础，将数据流图按一定的步骤映射成软件结构。L. Constantine 和 E. Yourdon 等人提出结构图是进行软件设计的有力工具。它与结构化分析衔接，构成了完整的结构化分析与设计技术，是目前使用最广泛的软件设计方法之一。

在需求分析阶段，信息流是考虑的关键问题。用数据流图来描述信息在系统中的流动情况。因为任何系统都可以用数据流图表示，所以结构化设计方法理论上可以设计任何软件结构。通常所说的结构化方法也就是基于数据流的设计方法。

5.6.1 数据流的类型

结构化设计的目的是要把数据流图映射成软件结构，而数据流图的类型又确定映射方法，因此必须研究数据流图的类型。在各种软件系统中，不论数据流图如何庞大与复杂，根据数据流的特性，一般都可分为变换型数据流图和事务型数据流图两类。下面分别介绍。

1. 变换型数据流图

根据信息系统的模型，信息一般是以外部形式进入系统，通过系统处理后离开系统的。从其过程可以得出，变换流的数据流图是一个线性结构。变换型的数据流由逻辑输入、变换中心（或称处理）和逻辑输出三部分组成，如图 5-31 所示（虚线为标出的流界）。

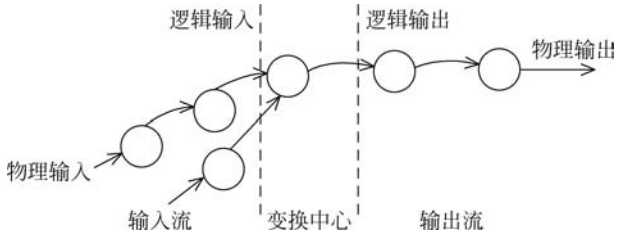


图 5-31 变换型数据流图

变换型数据处理的工作过程一般分为取得数据、变换数据和给出数据。这三步体现了变换型数据流图的基本思想。变换是系统的主加工，是系统的变换中心。变换输入端的数据流为系统的逻辑输入，输出端为逻辑输出。而直接从外部设备输入的数据称为物理输入，反之称为物理输出。外部的输入数据一般要经过输入正确性和合理性检查、编辑及格式转换等预处理，这部分工作都由逻辑输入部分完成，它将外部形式的数据变成内部形式，送给变换中心。同理，逻辑输出部分把变换中心产生的数据的内部形式转换为外部形式然后物

理输出。当数据流图具有这些特征时,这种信息流就称为变换流。

2. 事务型数据流图

基本系统模型意味着变换流,因此原则上可以认为所有的信息流都可以归结为这一类。然而,若某个加工将它的输入流分离成许多发散的数据流,形成许多平行的加工路径,并根据输入的值选择其中一个路径来执行,这种特征的数据流图则称为事务型数据流图。这个加工称为事务处理中心,图 5-32 所示为事务型数据流图。

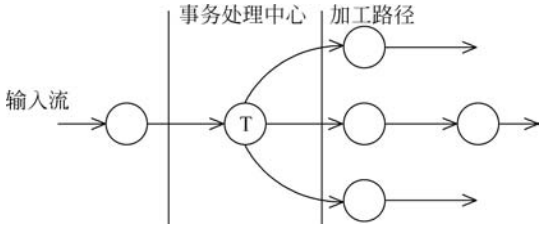


图 5-32 事务型数据流图

图 5-32 中的处理 T 称为事务中心,它完成下述任务。

(1) 接收输入数据。

(2) 分析每个事务,确定其类型。

(3) 根据事务选择一条活动通路。

这两种类型并不是说一个数据流图就是属于其中某种数据流图。一个大型软件系统的数据流图,可能既具有变换型的特征,又具有事务型的特征。例如,事务型数据流图中的某个加工路径可能是变换型。

5.6.2 过程步骤

对于需求分析阶段的结果进行分析的目的是从数据流图到程序结构图的转换方便。在转换前,先介绍有关结构化设计方法转换的步骤。

(1) 分析数据流图。把数据流图转换为软件结构图前,设计人员要参照规范说明书,仔细地研究、分析数据流图并参照数据字典,认真理解其中的有关元素,检查有无遗漏或不合理之处,进行必要的修改。

(2) 确定数据流图类型。通常将系统的数据流图视为变换流。但是,当系统有明显的事务流时,就要按照事务流来处理。要分析系统数据流中的主要数据流,以此来确定其类型。如果是变换型,则确定变换中心和逻辑输入、逻辑输出的界线,映射为变换结构的顶层和第一层;如果是事务型,则确定事务中心和加工路径,映射为事务结构的顶层和第一层。另外,当一个类型系统的数据流中有另外类型的数据流时,可以将其分离出去,作为子系统来处理。

(3) 找出变换中心。输入流是一条路径,经过这条路径数据从外部形式转换为内部形式。输出流则相反,从内部形式转换为外部形式。但是输入/输出流的边界并不明确,因此不同的设计人员可能会选择不同的边界点,那么不同的边界点就得到不同的结构。尽管如此,对于数据流图中一个处理点的变动对软件的结构也不会产生很大的影响。

如果是事务流，则这一步是确定事务中心和每条处理路径的特征。事务中心的位置在数据流图中是容易看出的。

(4) 第一层分解。如果是变换流，则要把数据流图映射为一种输入、变换、输出的特殊结构。在它的顶层是一个主控制器，下面是输入控制器、变换控制器、输出控制器。主控制器协调下属控制功能；输入控制器协调输入数据的接收；变换控制器规范所有内部形式的数据操作；输出控制器协调输出数据的生成。

如果是事务流，则要把数据流图映射为事务处理的程序结构。这种结构包含一个接收分支和一个发送分支。发送分支的结构包括一个模块，它管理所有下属的模块。

(5) 第二层分解。这一步的任务是把数据流图中的各个变换映射成相应的模块。从变换中心的边界起，沿输入路径向前移动，将处理映射成一个一个模块；然后，从变换中心的边界起，再沿输出路径向前移动，将处理映射成一个一个模块。把变换映射成下一层的结构。在映射中，可以将一个处理映射成几个模块，也可将几个处理映射成一个模块。

如果是事务流图，就表示把各个变换映射成程序结构的模块，而将事务处理结构的分支进行分解和细化。

(6) 根据优化准则对软件结构求精。

(7) 描述模块功能、接口及全局数据结构。

(8) 复查，如果有错，转步骤(2)修改完善，否则进入详细设计。

5.6.3 变换分析设计

变换流的设计是将数据流图转换为程序结构图。当数据流图具有较明显的变换特征时，按照下列步骤设计。

1. 确定数据流图中的变换中心、逻辑输入和逻辑输出

如果设计人员经验丰富，则容易确定系统的变换中心，即主加工。如除了几个数据流的汇合外，往往是系统的主加工。若暂且不能确定，则要从物理输入端开始，一步一步沿着数据流方向向系统中心寻找，直到有这样的数据流，它不能再被看作是系统的输入，则它的前一个数据流就是系统的逻辑输入。位于逻辑输入与逻辑输出之间的就是变换中心。同理，从物理输出端开始，逆数据流方向向中间移动，可以确定系统的逻辑输出。介于逻辑输入和逻辑输出之间的加工就是变换中心，用虚线划分出边界，数据流图的三部分就确定了。

2. 设计软件结构的顶层和第一层

变换中心确定以后，就相当于确定了主模块的位置，这就是软件结构的顶层，图 5-33 所示为变换分析设计实例。其功能主要是完成所有模块的控制，它的名称是系统名称，以体现完成整个系统的功能。主模块确定之后，设计软件结构的第一层。第一层至少要有输入、输出和变换三种功能的模块，它们可能是多个的。即为每个逻辑输入设计一个输入模块，其功能为顶层模块提供信息，如图 5-33 中的 f3。为每个逻辑输出设计一个输出模块，其功能为顶层模块提供相应的数据，如图 5-33 中的 f7、f8。同时，也为变换中心设计一个变换模块，其功能是将逻辑输入进行变换加工，然后逻辑输出，如图 5-33 中，将 f3 变换成 f7 和 f8。这些模块之间的数据传送应该与数据流图相对应，这样就得到了软件结构的顶层模块。这里

的主模块是总的控制模块,主模块中的控制逻辑决定着对其他模块的调用。

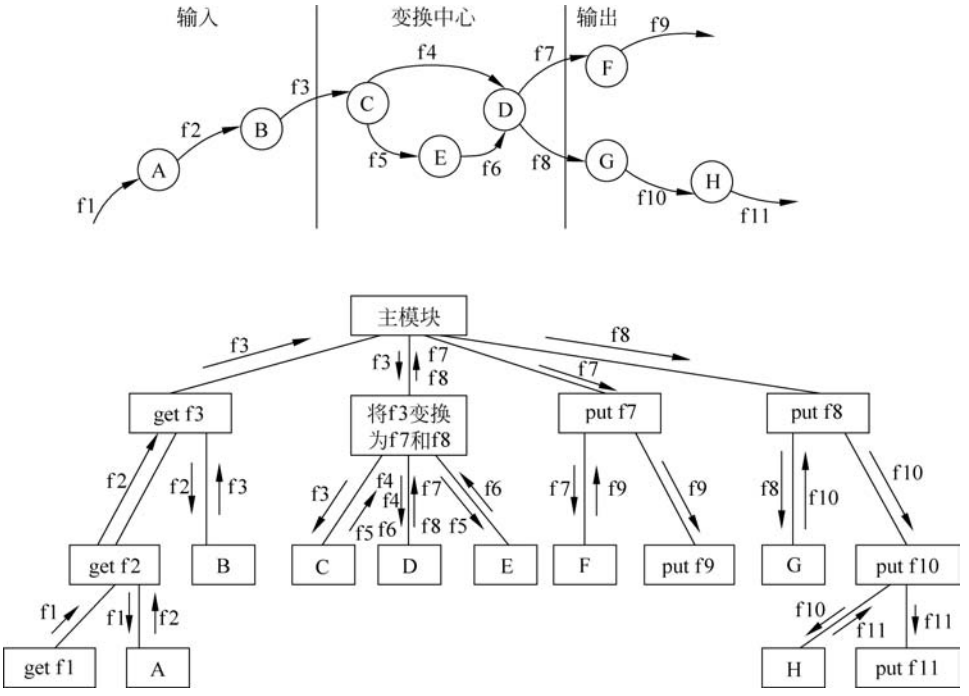


图 5-33 变换分析设计实例

3. 设计中、下层模块

对第一层的输入、变换及输出模块自顶向下,逐层分解,为各类模块设计出其下属模块。

1) 输入模块的下属模块的设计

一般情况下,输入/输出下属模块的输入模块的功能是向调用它的模块提供数据,所以必须要有数据来源。这样输入模块应由接收输入数据和将数据转换为调用模块所需的信息两部分组成。

因此,每个输入模块可以设计成两个下属模块:一个接收,另一个转换。用类似的方法一直分解下去,直到物理输入端,如图 5-33 中模块 get f3 和 get f2 的分解。模块 get f1 为物理输入模块。

2) 输出模块的下属模块的设计

输出模块的功能是将它的调用模块产生的结果送出,它由将数据转换为下属模块所需的形式和发送数据两部分组成。

这样每个输出模块可以设计成两个下属模块:一个转换,另一个发送,一直到物理输出端。如图 5-33 中,模块 put f7、put f8 和 put f10 的分解。模块 put f9 和 put f11 为物理输出模块。

3) 变换模块的下属模块的设计

设计完输入/输出后,就要为变换模块设计其下属模块。变换模块的下属模块的时间要根据数据流图中变换中心的组成情况,研究数据流图的变换情况,按照模块独立性的原则来

组织其结构，一般对数据流图中每个基本加工建立一个功能模块，如图 5-33 中模块 C、D 和 E。

4. 设计的优化

以上步骤设计出的软件结构仅仅是初始结构，还必须根据设计准则对初始结构进行求精和改进，以下为提供的求精办法。

(1) 输入部分的求精。在上述初步结构中，对每个物理输入模块输入，以体现系统的外部接口。结构图中的其他输入模块并非真正输入，当它与转换数据的模块都很简单时，可将它们合并成一个模块；当转换模块较复杂时，可以作为单独的接口模块处理。

(2) 输出部分的求精。与输入部分相似，为每个物理输出设置专门模块，同时注意把相同或类似的物理输出模块合并在一起，以降低耦合度。

(3) 变换部分的求精。根据设计准则，对模块进行合并或调整。

总之，软件结构的求精带有很大的经验性。往往形成数据流图中的加工与 SC 中的模块之间是一对一的映射关系，然后再修改。但对于一个实际问题，可能把数据流图中的两个甚至多个加工组成一个模块，也可能把数据流图中的一个加工扩展为两个或更多个模块，根据具体情况要灵活掌握设计方法，以求设计出由高内聚和低耦合的模块所组成的、具有良好特性的软件结构。

5.6.4 事务分析设计

事务流的设计是从事务数据流图到程序结构的变换。对于具有事务型特征的数据流图，则采用事务分析的设计方法。设计的方法也是自顶向下，逐步精化。先设计主模块，再为每一种类型的事务设计事务处理模块，然后为每个事务处理模块设计其下属的事务处理细节。事务处理模块可以被调用它的模块公用。与变换处理不同的是，其事务中心容易确定。下面结合图 5-34 所示的事务分析设计实例说明该方法的设计过程。

(1) 确定数据流图中的事务中心和加工路径。

当数据流图中的某个加工具有明显的将一个输入数据流分解成多个发散的输出数据流时，该加工就是事务中心。从事务中心辐射出去的数据流为各个加工路径。

(2) 设计软件结构的顶层和第一层。

事务处理中心和事务处理路径确定后，就可以确定它们的软件结构，其结构一般为一个接收分支和一个发送分支。从事务处理中心的边界开始向前移动，一个个地将变换点转换为模块。发送分支也有一个模块，它管理所有的下属处理模块。每一个事务处理的路径设计为相应的结构。最后将其转换设计一个顶层模块，它是一个主模块，有两个功能：一是接收数据；二是根据事务类型调度相应的处理模块。事务型软件结构应包括接收分支和发送分支两个部分。

接收分支：负责接收数据，它的设计与变换型数据流图的输入部分设计方法相同。

发送分支：通常包含一个调度模块，它控制管理所有下层的事务处理模块。当事务类型不多时，调度模块可与主模块合并。

(3) 进行事务结构中、下层模块的设计、优化等工作。

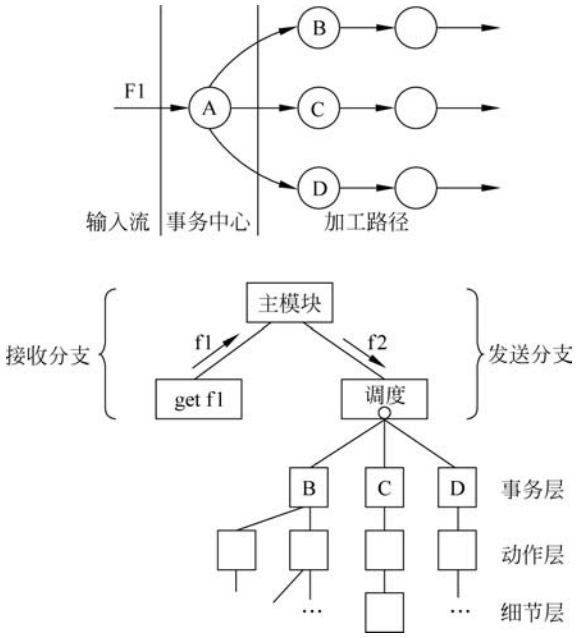


图 5-34 事务分析设计实例

5.6.5 混合流设计

1. 混合数据流图的映射

一般中型以上系统的数据流图中,都会既有变换流,又有事务流。这就是所谓的混合数据流图。其软件结构设计一般采用以变换流为主,事务流为辅的方法。具体步骤如下。

(1) 确定数据流图整体上的类型。事务型通常用于对高层数据流图的变换,其优点是把一个大而复杂的系统分解成若干较小的简单的子系统。变换型通常用于对较低层数据流图的转换。变换型具有顺序处理的特点,而事务型具有平行分别处理的特点,所以两种类型的数据流图导出的软件结构有所不同。只要从数据流图整体的、主要的功能处理分析其特点,就可区分出该数据流图整体的类型。

(2) 标出局部的数据流图范围,确定其类型。

(3) 按整体和局部的数据流图特征,设计出软件结构。

2. 分层数据流图的映射

前面在系统分析时曾经讲过分层数据流图的方法。因此,一个复杂问题的数据流图结果往往是分层的,那么对于分层的数据流图映射成的软件结构图也应该是分层的。这样既便于设计,也便于修改。由于数据流图的顶层图反映的是系统与外部环境的界面,因此系统的物理输入与物理输出都在交换中心的顶层。相应地,软件结构图的物理输入与输出部分应放在主图中,便于同数据流图的顶层图对照检查。分层数据流图的映射方法如下。

(1) 主图是变换型,子图是事务型,如图 5-35 所示。

(2) 主图是事务型,子图是变换型,如图 5-36 所示。

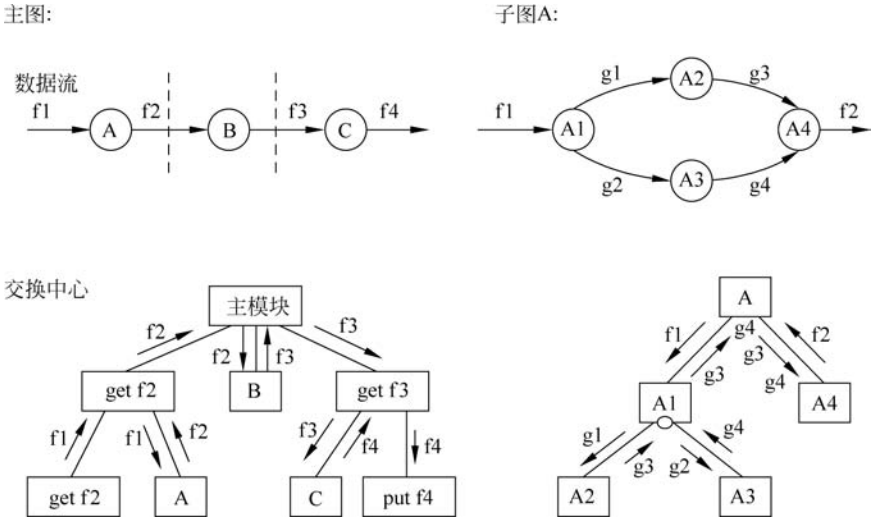


图 5-35 主图是变换型,子图是事务型

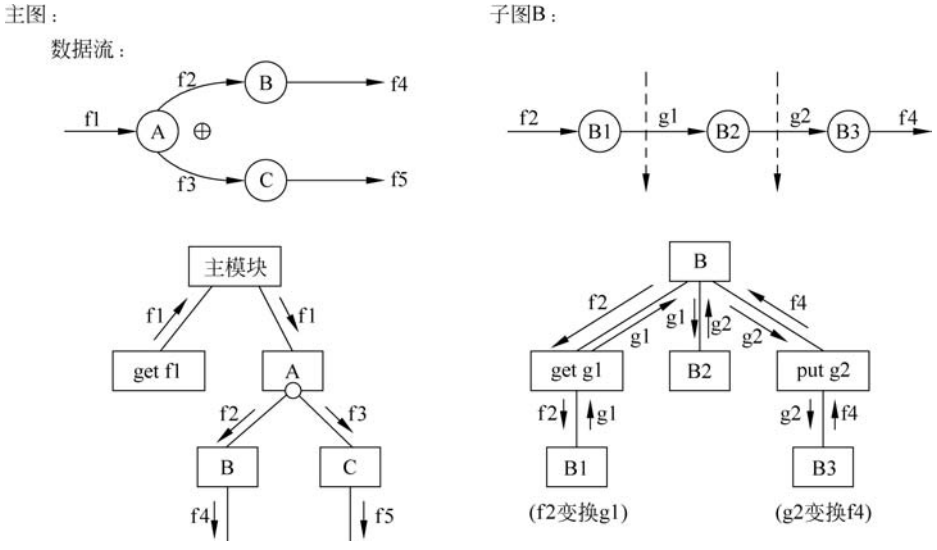


图 5-36 主图是事务型,子图是变换型

5.6.6 结构化设计方法应用示例

将销售管理系统的数据流图转换为软件结构图。分析该系统的 0 层图,它有 4 个主要功能,即订货处理、进货处理、缺货处理和销售统计。其中,订货处理包括订单处理和供货处理两部分。这 4 个处理可平行工作,因此从整体上分析可按事务型数据流图来设计,根据功能键来选择 4 个处理中的一个。

设计出的软件结构如图 5-37 所示。

销售管理系统软件结构子图 1 如图 5-38 所示。

销售管理系统软件结构子图 2 如图 5-39 所示。销售管理系统软件结构子图 3 如图 5-40 所示。

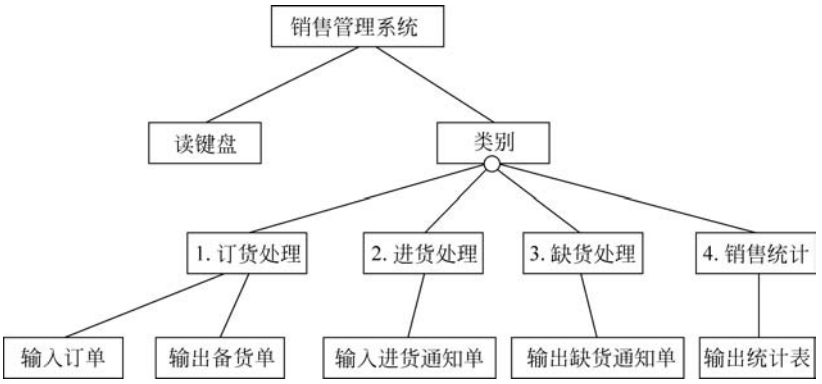


图 5-37 销售管理系统软件结构

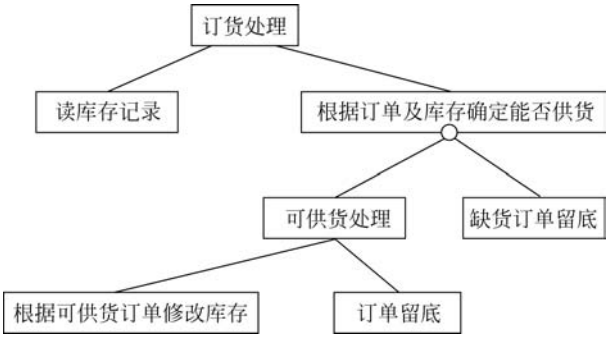


图 5-38 子图 1

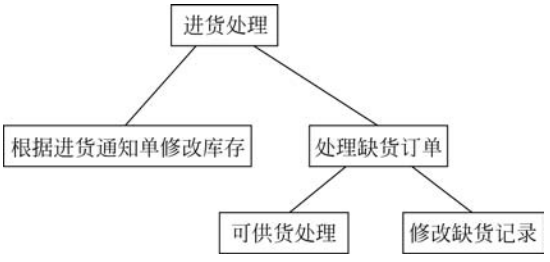


图 5-39 子图 2

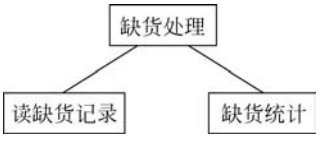


图 5-40 子图 3

销售管理系统软件结构子图 4 如图 5-41 所示。

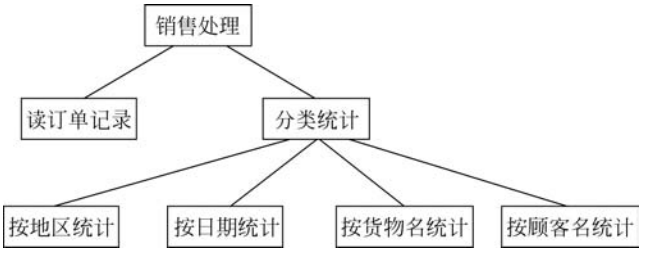


图 5-41 子图 4

5.6.7 设计的后期处理

由设计的工作流程可知，经过变换分析或事务分析设计，形成软件结构。对于这个结构，除了设计文档作为系统设计的补充外，还要对结构进行优化和改进。因此，要做以下工作。

(1) 为每个模块提供一份接口说明。包括通过参数表传递的数据、外部的输入/输出和访问全局数据区的信息等，并指出它的下属模块与上属模块。

为清晰易读，对以上说明可用设计阶段常采用的图形工具——IPO 图来表示。

(2) 为每个模块写一份处理说明。从设计的角度描述模块的主要处理任务、条件抉择等，以需求分析阶段产生的加工逻辑的描述为参考。这里的说明应该是清晰、无二义性的。

(3) 给出设计约束或限制。如数据类型和格式的限制、内存容量的限制、时间的限制、数据的边界值、个别模块的特殊要求等。

(4) 给出数据结构说明。软件结构确定之后，必须定义全局的和局部的数据结构，因为它对每个模块的过程细节有着深远的影响。数据结构的描述可用伪码(如 PDL 语言、类 Pascal 语言)或 Warnier 图、Jackson 图等形式表达。

(5) 进行设计评审。软件设计阶段，不可避免地会引入人为的错误，如果不及时纠正，就会传播到开发的后续阶段中去，并在后续阶段引入更多的错误。因此一旦设计文档完成以后，就可进行评审，有效的评审可以显著地降低后续开发阶段和维护阶段的费用。在评审中应着重评审软件需求是否得到满足，即软件结构的质量、接口说明、数据结构说明、实现和测试的可行性以及可维护性等。

(6) 进行设计优化。设计优化应贯穿整个设计的过程。设计的开始就可以给出几种可选方案，进行比较与修改，找出最好的一种。设计中的每一步都要考虑软件结构的简明、合理及高效等性能，以及尽量简单的数据结构。

5.7 应用案例——成绩管理系统总体设计

引言和任务概述略。

5.7.1 总体设计

1. 需求规定

1) 对功能的规定

通过成绩管理系统实现对每个学生基本情况的添加、修改、删除、查询等操作，同时实现对学生学籍异动情况进行更新。

(1) 超级管理员完成用户类别、基础数据管理和系统基础设置等。

(2) 普通管理员为教务处工作人员和教务人员，主要完成教师基本信息、学生基础信息、课程信息、课程表信息、专业信息等管理。

(3) 学生基本信息主要包含学号、姓名、性别、籍贯、民族、政治面貌、入学时间、所学专

业、所在班级等。

(4) 教师基本信息主要包含教师工号、姓名、籍贯、性别、民族、政治面貌、所在院、职称等。

(5) 专业基本信息主要包括专业编号、专业名称、所属院、系等。

(6) 课程基本信息主要包含课程编号、课程名称、学分、课程学时、课程性质等。

(7) 班级基本信息主要包括班级编号、班级名称、专业编号、班级所在院系。

(8) 课程表信息主要包括课程编号、课程名称、课程类型、任课班级、任课学年、授课教师等。

(9) 登录管理：要求使用者提供合法的用户名、密码和根据相关权限进行登录操作。

(10) 成绩录入：由教师或管理员录入成绩，录入信息包括前面的学生信息、课程信息等。

(11) 成绩基本信息主要包括课程成绩评价体系设置、课程成绩等，可以实现对学生成绩的添加、修改、删除、查询、统计等操作。若学生学籍或成绩有异动，可以实现对学生成绩进行迁移或更新。学生成绩表主要包括如下信息：学号、姓名、课程编号、课程名称、学分、成绩、平均成绩等。

(12) 统计汇总功能：超级管理员、管理员可以对成绩进行分类汇总，比较各班级、各院系成绩，为各种教学评优或制定教学管理计划提供数据依据。

2) 对性能的规定

(1) 精度：说明对软件的输入、输出数据精度的要求。

(2) 时间特性要求：查询服务部分，用户查询返回结果不超过 5s；数据管理部分，提交某一数据录入到结果返回不超过 5s。

(3) 故障处理要求，磁盘碎片过多、数据库存储空间不够，引起数据库访问变慢等问题需要对磁盘进行扩展和维护；执行程序非正常退出，响应缺失，修改源代码前应备份。

3) 其他专门要求

在程序开发过程中，应遵循结构化的程序设计原则，设立运行日志，加强系统的可维护性；注重系统的界面友好性、各程序模块界面的统一。

2. 运行环境

(1) 硬件环境：台式机或笔记本电脑，所需内存至少为 256MB；移动终端(手机等)。

(2) 软件环境：Windows 2000 Professional/XP 或更高版本，SQL Server 2008 或更高版本；移动端操作系统 iOS、Android 等。

3. 基本设计概念和处理流程

该系统为 B/S 三层结构，采用面向过程软件工程方法，它的运行环境分客户端、应用服务器端和数据库服务器端三部分。总体设计用例图见需求分析说明书。

4. 结构

1) 系统体系结构

成绩管理系统采用 B/S 结构，用户界面通过 Web 浏览器来实现，主要的逻辑在 Web 服

务器和应用服务器端实现，数据存储在数据库服务器中，形成常见的 Web 应用三层结构：表示层、业务逻辑层和数据层，如图 5-42 所示。

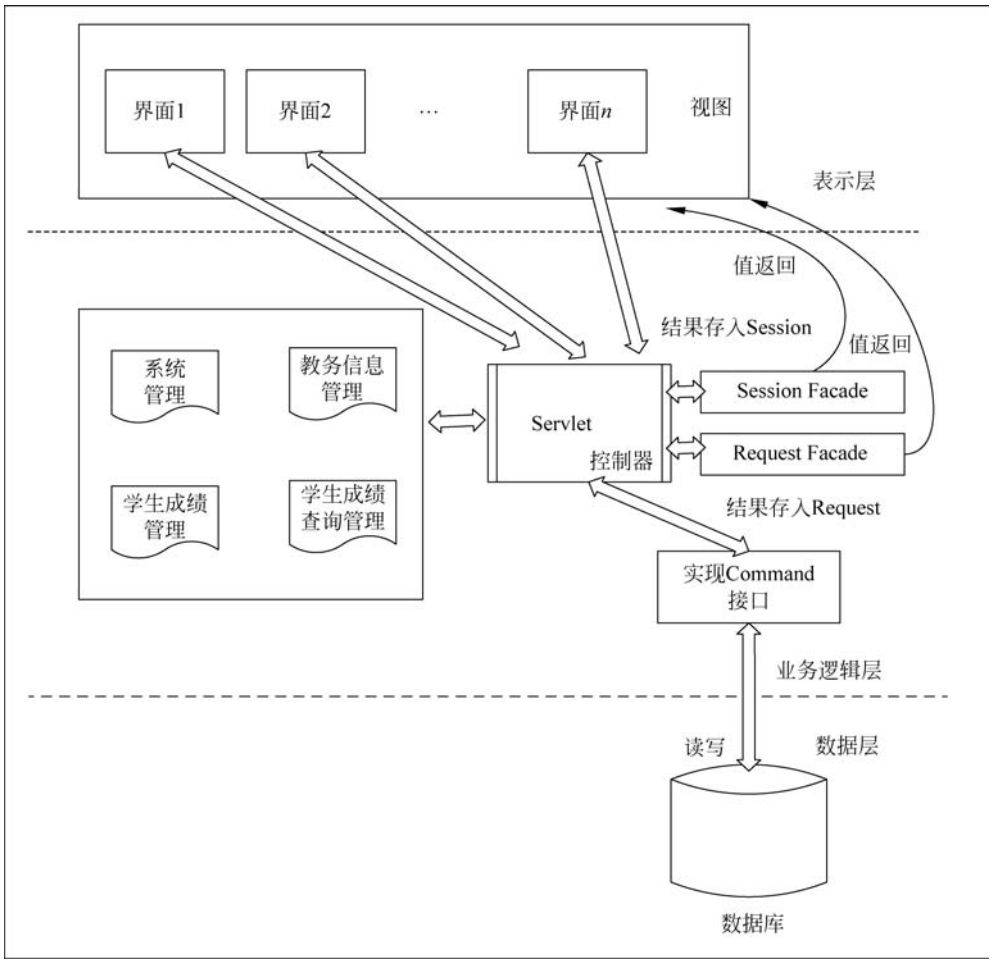


图 5-42 系统体系结构框架图

表示层用来与用户进行交互。提交用户请求给业务层处理和向用户显示从业务层返回用户请求数据的结果。用户直接通过该层来访问系统，实现与系统的交互，从而完成需要实现的工作。

业务逻辑层处理来自表示层传送的请求。这层实现系统的所有核心业务逻辑，例如数据的有效性校验、数据的安全性校验以及业务的流程控制和处理。该层还会根据请求的内容，将执行的结果提交给数据层做统一的处理，并且将用户请求处理的结果返回表示层显示。成绩管理系统的功能模块层主要包括系统管理、教务信息管理、学生成绩管理与学生成绩查询管理等。

数据层数据层主要处理和数据资源相关的逻辑，例如存储从业务层传送来的结果数据或从数据库中读取数据传送给业务层处理。这些组件和服务在功能上和中间层相互独立。系统数据主要由基础信息、学生信息、教务人员、管理员以及成绩组成。

变换分析设计是数据流图到程序结构图的转换，根据总体数据流图与采用变换分析方

法得到的系统结构图,如图 5-43 所示。

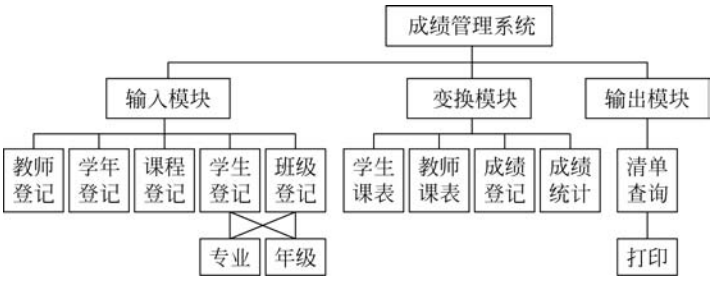


图 5-43 系统结构图

2) 软件结构图

根据 4.6.3 节中的一个数据流图(图 4-16),通过变换分析可以得到系统软件结构图,如图 5-44 所示。

3) 软件结构优化

根据软件结构设计优化准则对系统进行优化得到功能结构图,如图 5-45 所示。

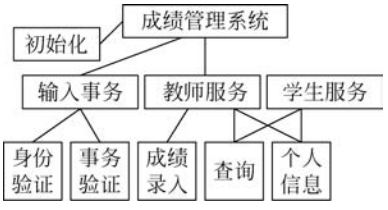


图 5-44 软件结构图

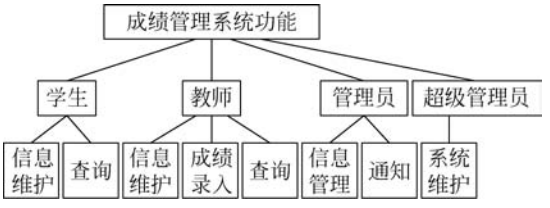


图 5-45 功能结构图

5. 功能需求与程序的关系

功能需求与程序如表 5-1 所示。

表 5-1 功能需求与程序

	程序 1	程序 2	程序 3	程序 4
录入功能	√			
查询功能		√		
修改功能			√	
删除功能				√

6. 人工处理过程

各基础数据需人工录入或导入数据库。

5.7.2 接口设计

1. 用户接口

采用可视化的操作方式作为人机接口,如使用窗口、菜单等,借助鼠标、键盘进行各种操

作。身份验证：对登录的用户进行验证，验证通过才能进入系统。查询学生的基本信息：对学生的基本信息进行查询。查询教师的基本信息：对教师的基本信息进行查询。查询学生的成绩：对学生的成绩进行查询。查询课程的基本信息：对学生课程的基本信息进行查询。课程成绩的构成：根据学生成绩的构成，如平时成绩、期末成绩、实训成绩进行查询。修改功能：对学生的基础信息和成绩信息进行修改。帮助功能：为用户提供使用帮助。

2. 外部接口

硬件接口：P4 1.8GHz 以上，内存 256MB 以上，能运行 IE 6.0 及以上版本。

软件接口：Windows 2000 Professional/XP 或更新版本，SQL Server 2008 或更新版本。系统能够从外部导入 Word、Excel 文档或导出 Word、Excel 文档。

3. 内部接口

查询模块：有相应消息驱动，完成对信息进行查看的功能。

增加模块：具有此权限的人员完成对信息进行增加的功能。

删除模块：具有此权限的人员完成对信息进行删除的功能。

打印模块：完成打印功能。

退出模块：实现退出功能。

5.7.3 运行设计

(1) 运行模块组合：具有软件的运行组合为程序多窗口的运行环境，各个模块在软件运行过程中能较好地交换信息，处理数据，如图 5-46 所示。

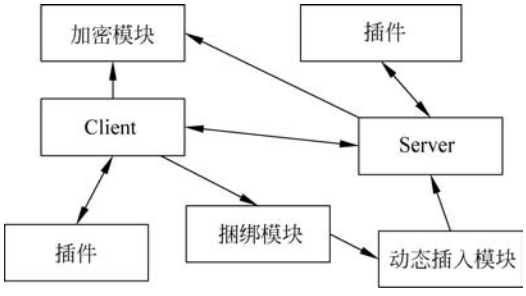


图 5-46 运行模块组合

(2) 运行控制：软件运行时有较友好的界面，基本能够实现用户的数据处理要求。

(3) 运行时间：一般页面的响应时间小于 5s，统计页面响应时间小于 10s。

5.7.4 系统论结构设计

1. 逻辑结构设计要点

概念设计用来反映现实世界中的实体、属性和它们之间的关系等的原始数据形式，建立数据库的每一幅用户视图。成绩管理系统分为八个实体，分别包括各属性。

学生 E-R 图如图 5-47 所示。学生-课程 E-R 图如图 5-48 所示。

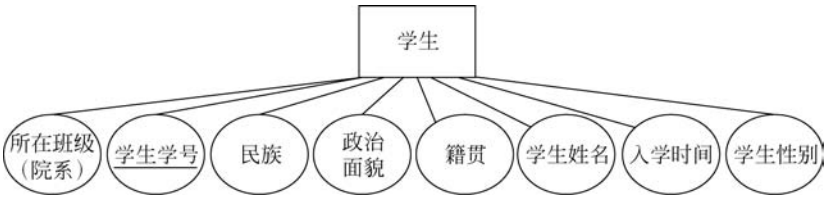


图 5-47 学生 E-R 图

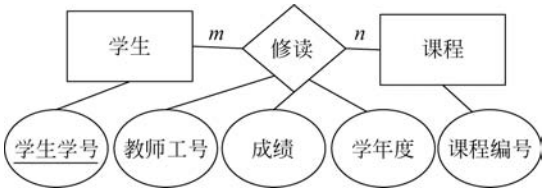


图 5-48 学生-课程 E-R 图

其他 E-R 图略。

2. 物理结构设计要点

数据库的逻辑设计是将各局部 E-R 图进行分解、合并后重新组织起来形成的数据库的全局逻辑结构,包括所确定的关键字和属性、重新确定的记录结构、所建立的各个数据之间的相互关系。本系统的数据库表如表 5-2 所示。

表 5-2 学生信息表

名 称	字段名称	类型	长度	允许空
学生学号	Sid	varchar	50	no
学生姓名	Sname	varchar	50	yes
学生性别	Ssex	char	2	yes
所在班级	Cid	varchar	50	yes
籍贯	Splace	varchar	50	yes
民族	Snation	varchar	50	yes
政治面貌	Spolitical	varchar	50	yes
入学时间	Stime	varchar	50	yes

其他用户、教师、专业、班级、课程、学生-课程、教师-课程信息表略。

5.7.5 故障检测与处理机制

(1) 系统采用镜像备份数据库,以便在系统出现故障时,能够及时恢复。

(2) 系统发生故障可以使用多种检测机制,如自动向上层汇报、由上层定时检测、将故障写入错误文件等。

(3) 对软件及运行环境进行日常维护。

(4) 对软件开发中出现的问题进行修改和补充。

小结

本章主要介绍了总体设计的基本原理和方法。总体设计阶段是用比较抽象的方式确定系统的预定任务，应该确定系统的物理配置方案，然后确定组成系统的结构。因此，要重视软件设计的重要性及设计与软件质量问题。从数据流图出发，设想系统的功能和几种物理方案，对于初步方案要进行仔细的比较与分析。特别是要与用户进行沟通，选择最佳的方案，然后才能进行软件设计，确定软件的组成以及模块间的关系。设计中，层次图和结构图是描述软件结构的常用工具。

在进行软件结构设计时，要合理地控制程序结构，给出程序构件的组织，同时定义最佳的数据结构，确定信息的组织、存取方法、结合的程度。要遵循模块的独立性，在结构化设计中，对于数据流的类型要进行仔细的分析与区别。特别是在混合数据流图中，只有确定了数据流图的类型，才能运用好变换分析设计和事务分析设计的方法。当基本设计完成后，要进行模块的处理说明、模块的接口说明、约束与限制说明、数据结构说明。然后按照模块独立性准则、控制与作用范围之间的准则、结构特征准则和模块的接口准则对结构进行优化。

综合练习 5

一、填空题

1. 软件结构的设计是以_____为基础的，以需求分析的结果为依据，从实现的角度经进一步划分为_____，并组成模块的_____。
2. 软件设计是一个把_____转换为_____的过程，包括_____和_____。
3. 变换型 DFD 由_____、_____和_____三部分组成。

二、选择题

1. 软件设计一般分为总体设计和详细设计，它们之间的关系是(　　)。
   A. 全局和局部　　B. 抽象和具体　　C. 总体和层次　　D. 功能和结构
2. 将几个逻辑上相似的成分放在一个模块中，该模块的内聚度是(　　)的。
   A. 逻辑性　　B. 瞬时性　　C. 功能性　　D. 通信性
3. 在对数据流的分析中，主要是找到变换中心，这是从(　　)导出结构图的关键。
   A. 数据结构　　B. 实体关系　　C. 数据流图　　D. E-R 图

三、简答题

1. 结构化设计方法的基本思想是什么？它是怎样与结构化分析衔接的？
2. 简述软件总体设计阶段的基本任务。
3. 举例说明各种类型的模块耦合。
4. 简述模块、模块化及模块化设计的概念。
5. 什么是模块的独立性？设计中为什么模块要独立？对于独立性怎样度量？

6. 试论“一个模块,一个功能”的优点。

7. 简述变换流的设计步骤。

8. 简述事务流的设计步骤。

9. 试论述软件设计与软件质量的关系。

10. 什么是模块的影响范围?什么是模块的控制范围?它们之间应该建立什么样的关系?

11. 什么是软件结构?简述软件结构设计的优化准则。