
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
中小学信息科学知识
信息系统分析与设计
BOOK
内部资料 非卖品

第一章　信息系统和系统科学

第一节　信息系统的产生和发展

一、社会信息化的发展和基本特征

在人类发展的历史长河中，科学技术的发展就是人类对客观世界认识不断深化的过程。唯物主义认为：客观世界的基础是物质。而物质又有三重属性，即物质属性、能量属性和信息属性。科学技术的发展反映了人类对这三重属性由浅入深的认识变化，技术革命的出现标志着这种认识取得突破性进展。以机械比为特征的

第一次技术革命（1780—1910）反映了人类对客观世界物质属性认识的升华，以电气化为特征的

第二次技术革命（1911—1945）反映出人类对能量属性认识的升华，而以自动化为特征的

第三次技术革命（1946—1976）和以信息化为特征的

第四次技术革命（1977—）则反映了人类对信息属性认识的升华。

一般认为，社会信息化有如下一些基本特征：

1.信息膨胀

据联合国教科文组织统计，80 年代末期，在自然科学领域内，基础科学有 538 个主要学科，技术科学则有 412 个专业分支。全世界每天发表 16000 篇论文，出版 1800 种图书，登记 800 多件专利。1983 年 3 月，美国化学学会记下了

第 600 万种化学物质。进入 90 年代以来，全世界每年约生产 720 亿条各种媒介的信息，这就是“信息爆炸”。信息膨胀使世界充满了大量各种各样的信息。这些信息在未经加工整序之前，人们将难以利用，即难以找到适合自己需要的信息。

2.信息技术高度发展

信息技术是指为获取、处理、存贮、检索、传递各类信息而采用的技术和设备的总称。可以认为，人脑的功能在于处理信息，而信息技术则是辅助人脑处理信息的技术，是能够扩展人的信息器官功能的技术。信息技术包括计算机、通讯、高密度存贮、声像、复印、印刷等广泛的技术领域及其在信息工作中的应用，其中计算机技术是其核心。

3.信息、信息技术和信息化的重要性日愈为人类所认识

信息是生命生存的要素，是影响社会生产力的重要因素，是现代社会经济发展的必要和重要条件，而信息消费则是信息时代社会生活重要部分。信息和原材料、能源一起，构成了社会生产的三大支柱，因而是一种重要的资源。对于社会生产的发展而言，信息技术是催化剂和倍增器，它对经济发展、社会变革、国家安全乃至整个国家的发展，都具有关键性的作用。因此，对信息资源的开发和利用，对信息技术的研究和应用，已经成为衡量一个国家综合国力的重要指标，也成为国际竞争中必须抢占的制高点。

90 年代正在兴起的“全球信息网络革命”将推动全球信息基础设施（GII，通称全球信息高速公路）建设的热潮，促进

第五次产业革命和信息时代的到来，它将极大地改变人类的生产、工作、学习和生活，改变世界的经济、文化和社会，也改变着人类自身。

4.信息行业迅速发展

信息资源的开发、信息技术的推广和应用，必然会促使信息技术的产业化、市场化，形成一个新的行业，即信息行业，并促进这个行业的大发展。在发达国家这种趋势尤为明显。有的专家认为，社会信息化的标志是：信息行业的产值超过国民生产总值的一半；信息行业的劳动力超过全国劳动力的一半；使用、操作计算机的劳动力占全国劳动力的一半以上。

解决信息膨胀和信息污染的有效途径，就是加强管理，根本的问题是使信息资源有序化，便于查找，便于使用。信息系统建设是信息资源有序化的重要组成部分和基础。

二、信息系统的产生和发展

1.信息系统的产生

20 世纪 50—60 年代期间计算机在信息处理领域得到广泛的应用，它的极高的处理速度、极大的存储能力和极广阔的应用领域向人们展示了其强大的生命力。一时间以电子计算机为基本处理工具的信息处理技术和系统风靡整个西方世界。各公司纷纷出巨资购买计算机，并抽出大量人力、财力建立信息处理系统，以取代日常的人工信息系统，并解决手工情况下人所想做而又没有能力做的数据处理、信息分析，甚至管理决策工作，以期为企业带来巨大的经济效益。总的来看，信息系统经历了以下几个发展时期：

（1）电子数据处理系统

电子数据处理系统即 EDPS（ElectronicDataProcessingSystem），它是最早的信息系统。EDPS 的功能就是利用计算机完成各类数据的存储、加工处理、输出和数据交换。EDPS 主要利用了计算机高速处理，巨大存储量及快速、准确的通讯等特点，解决了传统的手工数据处理中工作量大、不精确等问题。其具体的应用系统有数据统计系统、数据更新系统、数据查询系统和数据分析系统等等。分散孤立的 EDPS 可通过通讯网络相互连接起来，彼此协调工作，形成功能更为强大的信息处理系统。

EDPS 的出现标志着信息系统的产生，它将计算机应用推向了一个高潮，计算机的应用从单纯的数值运算扩大到数据处理的广阔领域。

（2）管理信息系统

所谓管理信息，即用作管理目的，并需要使用一定的方法整理、保存以方便使用的数据。同一般的信息不同，管理信息具有一定的数据规模、相对稳定的数据结构和较固定的处理模式法，并需要保存相当长的时期。人事档案数据、调查统计数据、库房管理数据、企业管理数据及金融财产数据等都属于管理信息。

管理信息系统（MISManagementInformationSystem）是在 EDPS 的基础之上发展起来的，它避免了 EDPS 在管理领域的一些弊病，在处理的方法、手段、技术方面都有了长足的进步。

与 EDPS 相比，MIS 有以下特点：

①强调定量化的管理模型和系统的优化处理；

②强调系统对处理对象和处理过程的预测和控制作用；

③强调系统对数据的深层次开发；

④强调系统化的开发方法。

（3）决策支持系统

自从 20 世纪 60 年代 Gallanher 提出 MIS 的设想到 70 年代初，MIS 经历

了一个迅速发展的时期，但随着时间的推移 MIS 逐渐暴露出来了很多问题。其主要问题是：早期的 MIS 缺乏对社会组织机构和不同阶层管理人员决策行为的深入研究，忽视了人在管理决策过程中的不可替代的作用。因而在实际工作中特别是在辅助高层的管理决策工作中，MIS 常常软弱无力，达不到预期的效果。70 年代初哈佛大学的 Bedder 归纳了早期 MIS 失败的经验和教训，提出了 MIS 的 7 个致命伤，在学术界掀起了一场“ MIS 为何会失败的 ”讨论。在这场讨论中一批学者提出了决策支持系统的概念，把信息系统的研究又推到了一个更新的阶段。经过 20 年的不断丰富和发展，形成了今天的 DSS（DecisionSupportSystem）。较之早期的 MIS，DSS 的概念主要是强调了以下几个方面：

①DSS 是面向决策者的；

②以解决具有不确定性、没有固定处理模式的管理决策问题为主；

③强调决策过程中人的主要作用，信息系统只是对决策者，即人在决策过程中的工作起支持作用。

2.信息系统的驱动方式

信息系统运行的驱动方式是指整体概念上推动信息系统运行的关键因素。也就是说，如果具备了这些关键因素，则一个信息系统的开发和运行原则上是不会有什么问题的。信息系统运行驱动方式一般有如下几种。

（1）数据驱动

数据驱动（DataDriven）多是针对确定型的系统而言。即在这样一类信息系统中解决问题的方式和程序都是确定的，因此收集、加工、整理这些方法和程序所需的数据就成为激活一个系统并使之能够成功的运行的关键。例如：在一个市场预测系统中，市场演变的历史和规律都是已知的，用于预测市场的预测公式也已确定，于是只要能收集到适当的数据，系统就能运行并对市场作出预测。传统的 EDPS、MIS 等都是数据驱动。

（2）模型驱动

模型驱动（ModelDriven）多是针对不确定型的系统而言。这里所说的模型是一个广义的概念，它包括数学的运筹学模型，以及经验公式、程序化方法等等一系列求解问题的技术、方式和模式，因此在不确定型的系统中求解问题的方法和过程都是不确定的。为此，系统首要的任务就是要确定系统的模型，一旦确定了模型，该问题就具备了最终求解的可能性。故我们称这类系统是模型驱动的，模型是推动该系统运行的关键因素。

例如：平时我们在处理一个不确定的决策问题时，首先必须识别问题和明确要求，然后再根据我们已掌握的知识和对问题的了解来动态地选择和构造求解该问题的模型，然后再从多个可选择的方案中确定评价选择的准则、方案等等。只有在这些问题都确定之后，才有可能利用系统来求解实际问题。DSS 以及其发展分支等都是模型驱动的。

（3）目标驱动

目标驱动（GoalDriven）多是针对信息系统的开发方式而言。即在开发过程中所有的方法、工具、步骤的选择都是围绕特定的目标而进行的。如果目标发生了变化，则整个开发系统（包括开发策略、计划、方法工具等等）将发生根本的变化。也就是说，在这类的系统中目标是决定系统运行方向的根本要素。

（4）设备驱动

设备驱动（FacilityDriven）多是针对一些技术系统而言。即在这样一类系统中，系统的结构、系统的先进性以及用户使用系统的方便程度都是由技术设备的发展水平所决定的。也就是说，设备是决定系统各方面特征的关键因素。办公自动化系统是典型的设备驱动的系统。在一个办公自动化系统中，自动化的办公设备、计算机设备、信息通讯设备是系统的主要支柱的技术依托。选用设备的智能化程度、先进性、可靠性直接决定着整个系统的质量。

（5）概念驱动

概念驱动（ConceptDriven）多是针对一些知识型、智能型的模似人脑系统而言。一般认为，在系统中概念对于问题的求解是相当重要的。因为人在解决任何实际问题时，当事人的经验知识、文化背景、价值观念以及它们在人脑中长期积累形成的概念对他解决问题的方式是至关重要的。

（6）用户驱动

用户驱动（UserDriven）多是针对某些不确定的系统而言。系统的构成、系统的运行可以是因人不同而不同的。这样的系统一般称之为用户驱动的系统。例如：决策支持系统就是一个由用户驱动的系统。在决策支持系统中，不同的决策者在使用系统时，输入不同的资料（如：信息、偏好、价值准则等等）将会产生不同的决策方案。

三、信息系统的开发方法

1.信息系统开发中的问题

随着计算机软硬件技术的飞速发展，信息系统的规模和复杂性急剧增加，信息系统的开发过程也日益复杂。规模的扩大导致开发人员增多，开发人员之间需要更密切的配合。然而，传统的开发过程没有完善的组织方法。小规模信息系统开发方法已不适应形势的发展和要求，因而导致了系统开发中一系列难于控制的问题，这些问题主要表现在以下几个方面：

①对信息系统开发的成本和进度估计常常不准确，有时实际成本比预计成本甚至高出一个数量级，实际进度也比预期推迟几个月甚至几年。

②用户对已完成的系统不满意，系统不能满足用户的实际要求。

③系统质量不高，存在着大量的错误，常常导致系统失败。

④现有系统难以维护，甚至无法维护。这就使得要纠正程序中的错误，或者通过修改程序以使之适应新的运行环境或扩充其功能十分困难。

⑤系统开发生产率低下，远远跟不上计算机应用的发展和社会对信息系统需求的发展速度。

以上问题的存在，使人们认识到，要根本解决系统开发中存在的问题，就需要解决众多开发人员之间的合作问题。为此，必须建立新的信息系统开发方法和管理方法。经过研究和系统开发实践，人们逐渐总结出一些规律性的方法。

2.信息系统的生存周期

人们在完成大型复杂工程时，常常将整个任务划分为若干阶段，每个阶段完成一定的任务，各阶段之间相互衔接，一起完成整个工程。信息系统开发也分为若干阶段，或者说，信息系统从研制到运行再到报废，有一个生存周期。这个生存周期可以划分为分析、设计、实现（编程或编码）、测试、运行等几个基本阶段。其中每个阶段都有明确的任务和完成任务的方法。

一般来说，信息系统的开发可分为以下几个阶段：

（1）系统分析阶段

该阶段的任务是理解和表达用户需求，并分析系统应有的结构和功能。

（2）系统设计阶段

该阶段的任务是根据分析阶段的要求，设计系统的结构和功能的实现方法。

（3）系统实现阶段

该阶段的任务是将设计阶段的思想转换为具体的系统。

（4）系统测试阶段

该阶段的任务是发现和排除以上各阶段工作中的错误，产生真正可运行的系统。

（5）系统运行和维护阶段

该阶段的任务是在实际环境中运行系统，发现并改进其中的问题。

目前，以系统生存周期为指导思想的开发方法已成为信息系统开发的主要方法。

3.信息系统开发中的文档

在信息系统开发的每一阶段，都需要编制一定的文档。这些文档连同计算机程序及数据一起，构成为信息系统的软件。文档是计算机软件中不可缺少的组成部分。它的作用是：

①作为开发人员在一定阶段内的工作成果和结束标志；

②向管理人员提供信息系统开发过程中的进展和情况，把开发过程中的一些“不可见的”事物转换成“可见的”文字资料，以便管理人员在各个阶段检查开发计划的实施进展，使之能够判断原定目标是否已经达到，还将继续耗用资源的种类和数量；

③记录开发过程中的技术信息，便于协调以后的开发、使用和修改；

④提供对信息系统的有关运行、维护和培训的信息，便于管理人员、开发人员、操作人员和用户之间相互了解彼此的工作；

⑤向潜在用户报导信息系统的功能和性能，使他们能判定该系统能否满足于自己的需要。

系统开发中的文档有两类：一类是开发过程中填写的各种图表，可称之为工作表格；另一类则是应编制的技术资料或技术管理资料，可称之为文档。文档的编制必须适应系统开发的整个生存周期的需要。

第二节　系统和系统科学

一、系统的基本概念

系统概念来源于古代人类的社会实践经验。英文的“系统”一词（system）即源于古希腊语，有“共同”和“给以位置”的含义。美国韦氏（Webster）大词典中，“system”是“有组织的或被组织化的整体”，并有“结合着的整体所形成的各种概念和原理的综合”，“由有规则的相互作用、相互依存的形式组成的诸要素的集合”等说明。目前系统有各种各样的定义，本书采用钱学森所下的定义，即：“把极其复杂的研究对象称为‘系统’，即由相互作用和相互依赖的若干组成部分结合成的具有特定功能的有机整体，而且这个‘系统’本身又是它所从属的一个更大系统的组成部分”。

系统是物质存在的形式，也是人类社会和人类自身存在的形式。宇宙空

间从基本粒子到庞大的星系，从无机界到有机界，从个体的人到整个人类社会，没有一件事物不自成系统，没有一件事物不从属于一定的系统。

系统的定义是逐步演化的。现代科学技术的发展导致系统的思想方法定量化，形成了具有数学理论基础，能定量描述和处理系统内部联系的一套科学方法，并得到计算机的有力支持，系统思想方法已从一种哲学思维发展成为一门新兴学科。

一门科学形成的重要标志之一是它已具有比较完整的体系结构，这个结构一般应包括四个层次，即哲学、基础科学、技术科学和工程技术。系统科学也不例外，建立在辩证唯物主义基础上的系统观是它的哲学部分，系统学则属于它的基础理论部分，运筹学、控制论、信息论等属于它的技术科学部分，而系统工程则是它的工程技术部分。

二、系统的分类

从不同的角度出发，可以将系统划分为不同的类型，本书仅介绍两种分类。

1.自然系统、人工系统和概念系统

从物质运动发展的角度出发，可把系统分为自然系统、人工系统和概念系统（或称抽象系统）。自然系统指在自然界存在且基本未受到人力影响的系统，如天体系统、太阳系、物质结构系统、河流系统、人体系统等。人工系统指完全由人创造出来的系统或受到人力改造的自然系统，如人类社会、城市、农村、人造机械、水库等。概念系统则指抽象的符号、思维、理论等组成的系统，如数系、软件系统等。当然，在某种情况下，有些系统难以简单区分。

2.开环系统和闭环系统

从本质上看，任何系统总是处在不断的发展变化之中，这种客观物质的运动以三种形态表现出来，我们又将其称为三种流，即物质流、能量流和信息流。一个人工系统，总是要输入这三种流，在系统中经过处理，再输出所需要的三种或某些流。在构造系统的输入和输出关系上，可以有开环（开放）和闭环（封闭）两种基本的模式，从而形成两类人工系统，即开环系统和闭环系统。如图 1—2—1 所示。

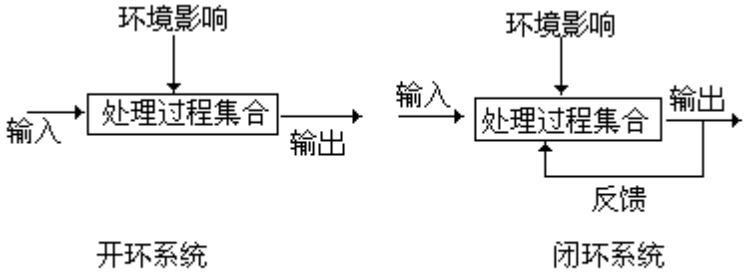


图 1-2-1 开环系统和闭环系统

在开环系统中，系统的输入经过处理后，形成系统的输出。由于系统处在一定的外部环境中，必然要受到外部环境的影响，这种影响体现在系统内处理过程的改变，这种改变也必然会影响到输出。

在闭环系统中，输出的一部分作为输入返回系统。由于改变了系统的输入，因而也就改变了系统的输出。部分输出以输入方式返回系统，被称为反馈。在输入不变的情况下，反馈可以改变输出，即对系统输出有控制作用。因此，我们常采用闭环系统，利用反馈来调整输出，从而产生期望的最佳输

出。

三、系统的特征

一般认为，系统有如下特征：

1.整体性

任何系统都是由各种元素组合形成的。随着社会与生产技术的发展，系统越来越复杂，系统中的元素会越来越多。

客观世界的各种事物、现象和过程，都是有机的整体，也都是自成系统而又互成系统的。机械论认为整体就是部分的简单相加，因而整体的简单分解就得到部分。系统论用整体的系统观点来取代简单分解和简单相加的机械论观点，系统论认为，系统不是其各组成部分的简单相加，而是由各部分按一定规则有机结合形成的整体，是各部分由量变发展到质变而形成的、远比简单相加更复杂的整体，整体的功能大于其各部分功能之和。系统论的创建者贝塔朗菲关于组成系统的著名定律就是：整体大于各孤立部分的总和。

物质结构、动物和人体的构造、社会组成等等，无一不证实了整体性观点的正确。

2.相关性

系统中的各个元素都有其特定的任务，因而这些元素应按一定的规律有序地组合起来，各组成部分具有相互联系和相互制约的关系。例如，一个生产车间，不同的车床要完成不同的任务，需要把各种类型的车床有规则地安放、组织起来，才能完成生产任务，杂乱无序地摆放着的一堆车床，是很难甚至无法进行生产的。

不论是自然界还是人类社会，不论是宏观世界还是微观世界，不论是无机物还是有机物，事物总是存在于某种联系之中，这就是普遍联系的观点。系统论中的相关性正是反映了这种普遍联系的观点。这种观点认为：系统和其所存在的环境之间是相互作用和相互联系的，系统内部的各组成元素之间也是相互作用和相互联系的。因此，一个元素的变化可能引起连锁反应，引起整个系统的变化，甚至破坏系统本身。

事物的联系是多种多样的。传统科学主要研究的是因果联系，因果联系一般又分为肯定因果联系和统计因果联系两种。事实上，除了因果联系之外，事物之间还存在着系统联系、结构联系、功能联系、起源联系等等。联系的多样性，决定了系统的多样性和联系之间界线的相对性。联系的广泛性导致系统的广泛性。这些又导致未知联系向已知联系的转化，形成未知系统向已知系统的过渡。

人们在宏观世界和微观世界取得的巨大进展，使得各种事物、现象和过程的本质联系逐渐显现出来，对整体中各部分的相互联系的研究已经从次要地位提到首要地位。从哲学高度建立相关性原理，为研究系统结构奠定了基础。

3.目的性

这里的目的性指的是广义的目的性，也即事物发展的规律性。任何系统，不论是自然系统、人工系统还是概念系统，都有其发生发展的规律，都是适应于一个或几个问题的解决而出现和建立的。

4.有序性

有序性又称为结构性。有序性是指，系统的任何联系都是按等级和层次进行的，秩序井然、有条不紊。系统的有序性是系统有机联系的反映。

系统的发展是定向变化的，即从层次较低的有序状态向层次较高的有序状态发展，或反过来，由较高有序状态向较低有序状态发展。

5.动态性

任何系统总是存在并活动于一个特定的环境中，与环境不断地发生物质、能量和信息的交换，从而不断地变化和发展。也可以说，具有时间性的程序是系统的特征之一。

动态性系统在不断的发展变化中，联系随时间而变化。动态就是指状态和时间的相关性。

现代科学研究的对象大都是结构复杂和高度活动的系统，动态性就是适应这种客观需要而产生的。人类不仅要研究各种系统发展变化的方向和趋势、活动的速度和方式，还要探索其发展变化的动力、原因和规律，从而主动驾驭系统，使之造福于人类。

四、系统工程

1.系统工程的基本概念

系统工程就是运用系统方法对各类系统进行设计、抉择、控制和管理，以达到最佳效益的一门管理科学。一般认为，系统工程属于系统科学的工程技术部分，是系统科学最具体的应用。从实际应用看，系统工程关心的往往是一种特殊类型的系统的结构。这种系统具有以下 7 个特征：

①为人造系统；

②具有整体性；

③规模大；

④结构复杂；

⑤半自动化；

⑥输入随机；

⑦有竞争性。

系统观点是系统工程最基本的特征。与一般工程不同，系统工程不是以某一专门的技术领域为研究对象，而是跨越各个专业领域，研究各类系统所具有的共性，因而是适用于各个领域的方法性科学。

系统工程不仅涉及传统的工业系统，而且涉及社会经济、环境生态、文化、教育等非工业系统。它不仅要利用现代自然科学和工程技术的新成就，也要利用社会科学如经济学、管理学、心理学等。

2.系统观点和系统工程的方法

分解综合、创造思维、可验证性和反馈，是系统科学的基本观点，也是系统工程的基本方法。

（1）分解综合

所谓分解是指对一个系统由大到小分成相对独立、层次不同的多个子系统，以便于进行研究；而综合是指设计新系统时，选择具有较好性能而又适用的子系统，以形成所需要的新系统。分解和综合常常是相互联系、不断重复的过程。

（2）创造思维

系统工程强调创造性地工作。创造思维的原理有两方面：一是把陌生的事物看作熟悉的东西，用已知的知识去辨别和解决它。这是人们比较熟悉和经常使用的方法。用已知方法去解决未知事物，既可能是旧理论、方法的重复，也可能有新的创造。另一种是把熟悉的事物看作陌生的东西，用新的方

法和新的原理加以研究，从而创造出新的理论或新的方法。这实际是对原有事物的重新认识。科学技术的发展过程中常常有这种情况，当某一领域或课题所使用的理论或技术显得陈旧，已经没有可能促进这一领域或课题进一步发展时，新的理论和技术却使它重又焕发青春。

（3）可验证性

实践观点是唯物辩证法的基本观点。在解决有关系统问题时，如果不能通过现有方法得出结论，总是先提出假设，通过试验进行验证，或对可能出现的故障进行分析判断，或为执行者提供数据进行核实，或为用户提供验收条件，甚至修正已有的理论等等。

（4）反馈

反馈的概念在系统分类中已经说明，即系统输出被送回输入，用以调整系统功能和输出。从哲学的观点看，反馈是在原因和结果之间架起的桥梁，使因果相互作用。同时，反馈也使系统更能适应环境的变化和影响，使系统和环境处于协调的动态统一之中，构成良性的新陈代谢。

现代信息系统是高度复杂的系统，系统管理者的主要任务就是控制系统，使其正常运转，因而必须重视反馈的作用，经常收集各种反馈，以改进系统工作。

3.系统工程的流程

系统工程在进行系统建设时，都要根据系统科学的观点，将设计、施工、验收及系统运行，恰当地划分为若干个工程阶段，明确每个阶段的任务和各阶段之间的联系与衔接。

图 1—2—2 给出一个一般性的系统工程基本流程图。

系统工程的基本方法是整个流程的基础，即把所研究的对象看作是系统，运用分析、综合和评价三个环节的多次重复得出最佳方案。如图 1—2—3 所示。

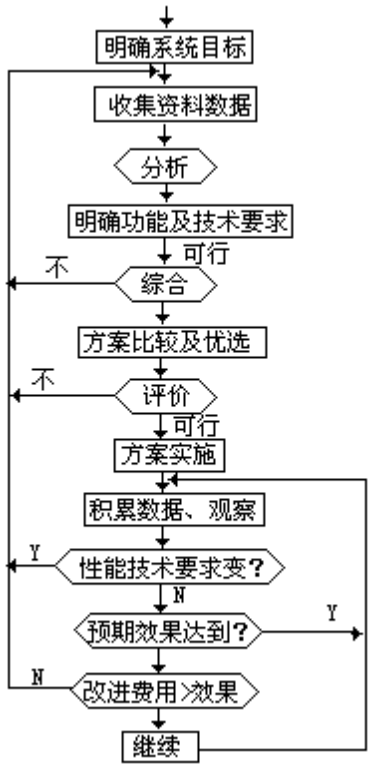


图 1-2-2 一般性系统工程的流程

第二章　系统分析

第一节　系统分析的任务

一、可行性研究与计划

1.可行性研究与计划的任务

可行性研究与计划工作是软件系统开发中第一个阶段的工作。可行性研究与计划工作（FeasibilityStudy&Plan）的任务有以下三个方面的内容：

①了解用户的要求及现实环境，从技术、经济和社会因素等方面研究并论证本软件项目的可行性；

②编写可行性研究报告；

③制订初步项目开发计划。

可行性研究的目的就是用最小的代价在最短的时间内确定问题是否能够得到解决。需要注意的是：可行性研究的目的不是解决问题，而是研究问题是否能够解决，确定问题是否值得去解决。因为并非所有问题都有简单明显的解决办法。如果某个问题在预定的系统规模内得不到解决，找不出可行的解决方法，则再去花费时间、人力和财力所做的任何工作都是浪费。

若所开发的系统较为复杂，客户本身及系统分析人员对目标系统所要解决的问题尚不清楚，应在可行性研究之前，即系统开发之前，先进行问题定义工作（ProblemDefinition）。一般认为，问题定义工作不属于系统开发工作的内容，而是客户事先邀请有经验的系统人员所做的先期性工作。一般规模系统的问题定义工作可在一二天内完成。在问题定义的基础上，再开始可行性研究工作。如有必要，在系统分析阶段应重新定义问题。否则，如果在对问题本身错误的理解上做可行性研究，自然得不到正确的结果。

问题定义工作是由客户同系统分析人员一起，对所要解决的问题进行分析，搞清要解决的到底是什么问题。为此，就必须对问题的性质、问题的目标、解决问题的必要性及解决问题的规模作明确的了解。该项工作结束后，应生成一份内容明确，客户和系统分析人员都满意的书面报告，即项目任务书。

可行性研究工作的长短取决于目标系统的规模。一般来说，可行性研究工作的成本是预期项目总成本的 5%～10%。

2.可行性研究的过程

可行性研究可按以下 5 个步骤进行。

（1）软件开发单位的系统分析人员对客户要求及现实环境进行调查。

系统分析人员同客户一起调查研究当前环境和有关具体问题，如手工工作情况、工作流程、所完成的任务及目前存在的问题，新系统要解决的主要问题，为可行性研究做准备。

（2）在调查研究的基础上编写有关客户提出的问题的书面材料。

（3）依据书面材料及其有关资料对待开发的系统从经济、技术和社会因素等方面进行可行性研究，写出可行性报告。

可行性包括以下三个方面：

①技术可行性

技术可行性通过考察所要解决的问题及其对功能、性能的要求，并对照现行的技术发展，以确定现有技术能否实现本系统。

②经济可行性

经济可行性是整个可行性分析的重点。它主要是考察系统的开发成本，主要是人力投入，此外还有设备投入等等方面。并对照系统运行所产生的效益，看看系统的经济效益能否超过开发成本。当然，对于有些类型的系统，应结合考虑系统开发所带来的社会效益。

③社会可行性

包括法律可行性和操作使用可行性等方面的内容。法律方面主要涉及专利、版权和合同责任等方面的问题。操作使用方面则主要指系统使用单位在行政管理、工作制度和人员素质等因素上能否满足系统操作方式的要求。

（4）评审和审批，决定项目是取消还是继续。

在对现实环境和有关的因素做出可行性研究之后，可根据调查分析的结果进一步导出新系统的高层逻辑模型。所谓高层逻辑模型，就是概括地描述新系统所完成的功能和物理工作过程。这样，就可以此为依据进行评审，以决定项目是否继续。

（5）若项目可行，则制订初步的项目开发计划，并根据需要签署合同。

若经过审批，决定项目继续，则应制定初步的开发方案，推荐行动方针，并草拟开发计划。完成整个可行性研究。

3.可行性研究的文档

信息系统开发的一个显著特点就是文档化。在信息系统开发过程中，文档记录总结了一个阶段工作的具体内容，并作为下一个阶段的工作基础，是不同阶段相互衔接的工具。文档既是系统开发者的工作需要，今后能以此为依据进行检查、修改；同时也是系统使用者的要求，文档向使用者提供了判断开发任务是否完成以及完成质量的依据。

可行性研究与计划工作完成后，应交付以下可以验证的文档：

a.可行性研究报告；

报告中的成本/效益分析应提供几种可供选择的解答。

b.初步的项目开发计划。

项目开发计划中对各阶段的工作的完成应有明确的、可检查的标志。

以下介绍这两个文档应有的内容。

（1）可行性研究报告

编写可行性研究报告主要有以下三个目的：

①说明开发此软件的可行性，即经济、技术和社会条件。

②评述为了合理地达到开发目标而可能选择的各种方案。

③说明并论证所选定的方案。

一个完整详尽的可行性研究报告包括以下内容：

①引言；

②可行性研究的前提；

③对现有系统的分析；

④所建议的系统；

⑤可选择的其它方案；

⑥投资及效益分析；

⑦社会因素方面的可行性；
⑧结论。
（2）项目开发计划
项目开发计划的内容有：
①引言；
②项目概述；
③验收标准；
④完成的最迟期限；
⑤批准者和批准日期；
⑥实施计划；
⑦支持条件；
⑧专题计划要点。

二、需求分析

1.需求分析的任务

需求分析（RequirementsAnalysis）所要完成的任务是：确定所开发的信息系统的运行环境、功能和性能要求，编写用户手册概要和确认测试准则，为概要设计提供需求说明书。通俗地说，通过需求分析应能正确和准确地回答“系统必须做什么”。具体包括以下几个方面：

（1）确定系统的综合要求

需求分析首先从宏观角度出发，采用功能分解的方法，对系统作整体的分析。这样做有助于正确理解系统需求。这是当前流行的结构化分析方法的主要特征，也是人们认识事物的规律。对系统作综合分析，要求从总的方面确定以下内容：

①系统的功能要求；

②系统的性能要求（包括响应时间、存贮容量、后援存贮、安全性等）；

③系统的运行要求（包括环境、如操作系统、数据库管理系统、通信接口等）；

④系统将来可能提出的要求（即当前不可能列入软件开发范围，但将来可能会提出的要求）。

（2）分析系统的数据要求

通过系统中的数据分析系统的需求是一种较有效的分析方法。一般系统有两方面的特征，即属性特征和行为特征。系统的属性可通过其数据结构、数据类型和数据流动体现。由于属性特征相对稳定，变化较小，所以系统的数据分析有助于系统分析的稳定性和有效性。数据分析主要包括确定系统的输入数据、输出数据、数据结构、数据类型、数据流动和数据加工及输入、输出时所要求的设备等。

（3）导出系统的逻辑模型

系统的逻辑模型是以某种形式说明系统属性和系统行为，它是现实系统的抽象，是对具体系统本质的描述。只有充分认识了系统本质，才可能真正了解系统需求，开发出符合需求的软件。

（4）修正系统开发计划

可行性分析已经根据其分析结构推荐了初步的项目开发计划。由于在需求分析中，对目标系统的认识进一步加深，系统中的功能要求、数据要求和性能要求也逐步明确，因而，要根据系统开发工作的主要内容、其中的难点、

要解决的关键问题及时间进度安排等对原来的初步项目开发计划作调整和修正。

（5）开发模型系统

若所开发的系统在早期作需求分析工作困难较大，则可先作简要的需求分析。通过开发模型系统，利用原型方法（本章稍后将介绍）来帮助完成需求分析及此后的一些工作。

2.需求分析的步骤

（1）调查被开发软件的环境

在这项工作中，分析人员应投入到相应的问题领域中，对其中的事物作体察入微的全面考虑，并研究和熟悉其中的所有细节。排除各种模糊的认识，准确掌握有关的基本概念。只有通过理解问题空间并建立它的模型，才能简单明了地用文字详细说明具体的问题需求。

（2）进一步明确用户需求

在可行性分析阶段，虽然已经做过问题定义工作，但它只是初步的和概括性的。在需求分析阶段，可通过结构化的或其它类型的分析方法，运用归纳、推理和比较的过程，对所调查的结果进行分析，经过抽象和概括，进一步明确定义出用户的需求。

（3）确定人机界面

所谓人机界面，即手工处理和计算机处理相衔接的部分。在系统开发初期，分清计算机不能承担的工作和人在系统中的作用，两者有效的协调方式是至关重要的。

例如，在图书馆自动化系统的图书流通管理中，需要由工作人员以某种方式，手工输入有关的借书信息。当计算机办理完借书手续后，很多图书馆要对图书的防盗磁条进行消磁，这也需要由人来完成。又如，当图书归还，计算机办理完还书手续后，需要由工作人员将书放到书架的正确位置上。这些由人来工作完成的工作都是系统的必要处理，是实现系统功能不可缺少的环节。

（4）修改初步的项目开发计划

对可行性分析中所制定的初步的项目开发计划，根据新的分析结果作必要的修改、补充和完善。

（5）制订确认测试计划

确认测试即项目委托方（甲方）最后的验收测试。验收的依据是确认测试计划，而该测试计划的依据是系统目标，即包括系统功能和性能等详细内容的用户需求。由于系统目标是在需求分析阶段制定的，所以，以此为依据的确认测试计划应在同一阶段，由相同的人员制定。这样，能最有效地保证两者的一致性。

（6）编写用户手册概要

在需求分析阶段就开始编写用户手册至少有以下两个好处：

①提早开始编写用户手册，使系统分析和设计人员在系统开发的早期就从用户的角度观察和分析系统，有利于提高系统对用户的友好程度，并有利于提高系统运行的方便性和实用性。

②在需求分析阶段开始编写用户手册，并使其随系统各阶段的开发不断完善，而不是在编码测试完成后组织人员另行编写，有利于保证用户手册的正确性、完备性和同真实系统的一致性，并可保证其按时完成。

（7）评审

在以上各步实施的过程之中，形成有关的文档。最后，由专家、分析人员、其它开发人员、用户组成的评审组对需求分析所得结果的正确性、合理性和有效性进行检查。若通过评审，则可以开始下一阶段工作；否则，需重新进行需求分析工作。

3.需求分析的文档

需求分析结束时，应交付以下文件：

■软件需求说明书

■修改后的项目开发计划

■用户手册概要

■确认测试计划

■数据要求说明书

其中，软件需求说明书是最重要的文件，其它每个文件都对应该阶段的一项工作。当然，在实际的信息系统开发中，可根据具体情况，灵活规定应交付的文件。

以下简要介绍其中两个主要文件的内容。

（1）软件需求说明书

需求说明书应包括引言、任务概述、需求规定、运行环境规定四部分。

①引言

②任务概述

③需求规定

■对功能的规定

■对性能的规定

■输入输出要求

■数据管理能力要求

■故障处理要求

■其它专门要求

④运行环境规定

■设备

■支持软件

■接口

■控制

（2）数据要求说明书

数据要求说明书包括引言、数据的逻辑描述和数据采集三个部分。

①引言

②数据的逻辑描述

包括对以下数据的描述：

■静态数据

所谓静态数据，指在运行过程中主要作为参考的数据，它们在很长一段时间内不会变化，一般不随系统运行而改变。

■动态输入数据

所谓动态数据，包括所有在运行中要发生改变的数据以及在运行中要输入输出的数据。

■动态输出数据

■内部生成数据
■数据约定
针对进一步扩充或使用方面的考虑，提出对各种数据的容量、文件、记录、数据元等个数的最大值，特别是属于临界性的限制。
③数据的采集
要说明以下内容
■输入数据的来源
■数据输入所用的媒体和硬设备
■接受者
■输出数据的形式和设备
■数据值的范围
■量纲
■更新和处理的频度
■输入的承担者
■预处理
■影响

第二节　系统分析的原则和方法

一、结构化分析方法

结构化分析（SAStructuredAnalysis）方法是信息系统开发中需求分析的重要方法，结构化分析方法起源于美国，该方法常同概要设计阶段中的结构化设计方法（SDStructured－Design）和编码实现阶段中的结构化程序设计方法（SPStruc－turedProgramming）衔接使用，成为被最为广泛使用的分析方法。结构化分析方法的基本思想，可以归纳为分析的层次化、功能的模块化和相互关联三个方面。

实际上，结构化分析起源于结构化程序设计。产生于 60 年代的结构化程序设计，在其最早的程序设计工作中，包括了结构化分析和结构化设计工作的大部分内容。但随着强调阶段性开发和开发规范化的软件工程的出现，其中的分析和设计工作被分离出来，形成了结构化的分析和设计工作。

1.结构化分析的特点

结构化分析方法有以下两个基本特点：

（1）自顶向下逐层分解

所谓自顶向下逐层细分，也就是对于一个复杂的事物，先抓住问题的大的方面，形成较高层次的抽象。然后再由粗到细，由表及里地逐步涉及问题的具体细节。即把大问题分解成几个小问题，对于每个小问题，再单独分析。这样逐层分解，从而对整个问题有清楚的了解。比如，对于一个图书馆系统，不可能一下子把它的全部具体工作以及它们相互之间的联系都弄清楚。可先忽略各种细节，从分析整个图书馆大的功能开始。如首先认识到图书馆有教育职能和情报职能，这是

第一层划分，即高层抽象，然后再沿这两个方向逐层分解。或先分为采购、编目、流通、期刊管理、参考咨询等几个部分，明确每一个部分的总体功能及它们之间的相互关系。然后对其中的每一个，如编目等再进行细分，得到

第二层功能划分。以此类推，直到确定所有细节。

（2）抽象

自顶向下逐层细分，实际上就是一个由模糊到清晰，由概括到具体的过程。同时也是一个不断运用抽象的过程。所谓抽象，就是在分析过程中，要透过具体的事物看到问题中的本质属性，能将所分析的问题实例变为一般的概念。抽象是一种手段。只有通过抽象，才能正确认识问题，把握住事物的内部规律，从而达到分析的目的。因为在分析中人们所接触的都是具体的事物，而人们要得到的，却是对该类事物一般问题的通用求解方法。抽象是信息系统分析中的重要原则。它不仅是结构化方法的特征，也是其它软件分析设计方法，如面向对象方法的重要基础。

2.结构化分析方法的类型

结构化分析有以下几种类型的方法：

■功能分解法

■数据流方法

■信息造型方法

其中，数据流方法是最为常用的结构化分析方法，所以它经常也被直接称为结构化分析方法。

3.结构化分析方法的步骤

结构化分析方法有以下 4 个基本步骤：

（1）理解当前系统，得出其具体模型。

（2）通过对当前系统具体模型的分析，抽象出当前系统的逻辑模型。

（3）分析目标系统和当前系统的逻辑差别，建立目标系统的逻辑模型。

（4）修改、充实和完善目标系统的逻辑模型。

二、系统分析人员

1.系统分析人员的任务

系统分析人员就是在信息系统开发初期从事系统分析工作的开发人员。按照 Nichlas 的定义：系统分析人员的任务就是明确需求和资源限制因素并且将它们变成具体的实施方案。系统分析工作终始是由客户和系统分析人员协作完成的。从软件工程的角度来看，系统分析人员的任务，就是负责完成可行性研究和需求分析两个阶段的工作，在用户和设计人员之间进行沟通。从事系统分析工作的是一组人员。由于系统分析工作是系统开发中的

第一项工作，指导着以后各个环节的开发工作，所以系统分析的质量十分重要。分析阶段的重要性决定了系统分析人员的重要性。

2.对系统分析人员的要求

系统分析人员必须具有多种才能，以便有效地工作。这些技能可以分成两类：处理人际关系方面的能力和解决有关技术问题的能力。具体地说，系统分析人员应具备以下基本素质：

①有一定的理论水平，全面、系统掌握计算机系统开发的基本理论和有关标准。

②具有较全面的计算机专业知识和信息系统开发的经验。系统分析人员应该是曾参加过信息系统开发各个阶段工作的高级开发人员。

③有较强的在新的问题领域提取知识的能力。能理解问题，并能准确地把握问题的细节。

④善于掌握非技术因素。具有较强的合作精神，能与各种类型的人友好

相处。有基本的调查艺术并善于掌握被调查者的心理。能够使用用户语言，具有较强的表达能力。

因此，系统分析工作对系统分析人员在理论水平、工作经验、个人素质和个人修养方面都有较高的要求。

第三节　数据流方法

一、数据流图

1.基本思想

数据流图（DFDDataFlowDiagram）是结构化分析中的重要方法和工具，是表达系统内数据的流动并通过数据流描述系统功能的一种方法。数据流图还可被认为是一个系统模型，在信息系统开发中，一般将它作为需求说明书的组成部分。

数据流图的作用：

（1）数据流图是理解和表达用户需求的工具，是系统分析的手段。由于数据流图简明易懂，理解它不需要任何计算机专业知识，便于通过它同客户交流。

（2）数据流图概括地描述了系统的内部逻辑过程，是系统分析结果的表达工具。因而是系统设计的重要参考资料，是系统设计的起点。

（3）数据流图作为一个存档的文字材料，是进一步修改和充实开发计划的依据。

数据流图从数据传递和加工的角度，利用图形符号通过逐层细分描述系统内各个部件的功能和数据在它们之间传递的情况，来说明系统所完成的功能。

2.基本成分及符号规定

（1）数据流

说明系统内部数据的流动，用箭头表示。箭头指向为数据流动方向。箭头旁写数据名。

（2）加工

又称数据处理、数据变换，表示对数据进行的操作。描述符号如图 2—3—la 所示。符号内写加工名。

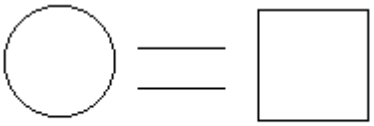


图2-3-1a 数据加工　图2-2-1b 数据存贮　图2-3-1c 外部对象

（3）数据存贮

又称文件，表示系统内需存贮保留的数据。数据存贮是系统内处于静止状态的数据，而数据流是系统内处于运动状态的数据。数据存贮的描述符号如图 2—3—lb 所示。符号内写文件名。

（4）外部对象

外部对象是向系统输入数据和接收系统输出的外部事物。也就是数据流的源点和终点，分别称为数据源和数据池。描述符号如图 2—3—lc 所示。

二、数据字典

1.数据字典和数据流图的关系

数据流图说明了系统内数据的处理，但它未对其中数据的明确含义、结构和组成作具体的说明。因此，仅有数据流图还不能完整地表达系统的全部逻辑属性。这些问题数据流图都无法回答。而它们又都是下一步系统设计要使用的重要内容。数据字典就是用于具体描述数据流图内数据的这些逻辑性质的。

数据字典是关于数据流图内所包含数据元素（数据存贮、数据流、数据项）的定义及说明的集合。其作用就是为系统人员在系统分析、系统设计和系统维护中提供有关数据的描述信息。

数据流图、数据字典和加工说明结合在一起，共同构成了系统的逻辑模型。它们是结构化分析中数据流方法的三个不可缺少的部分。

2.数据字典的条目类型

数据字典由以下三类条目组成：

①数据流；

②文件（数据存贮）；

③数据项（数据元素）。

3.条目组成和使用符号

（1）条目组成

条目给出有关数据的重要内容，各种条目的信息组成如下。

①数据流条目

数据流名字

简述

数据流组成

数据流量

峰值

用途

数据流来源

数据流去向

②文件条目

文件名字

简述

文件组成

存取频率

存取峰值

组织方式

用途

③数据项条目

数据项名字

简述

数据项组成

值类型

取值范围

以上为一般的内容组成。根据不同的系统要求，可增删一些具体内容。

（2）数据的定义方法

为了更加清晰地定义数据，尤其是复杂数据，说明其构成成分，在编制数据字典时引入了一些数据定义符号。这些符号既能明确描述复杂数据的结构和数据项间的关系，又直观、易懂，能使客户和设计人员准确理解其含义。

任何复杂数据基本是由数据元素反复使用顺序、选择和重复三种方式而形成的。为此，描述数据结构的具体符号可有以下几种。

①=

表示等价，即定义为。

②＋

表示顺序连接，即两个分量相连。

③［数据 1｜数据 2.....］

表示选择，即从所列出的多个数据中选择一个使用。

④ {}

表示重复，即花括号内的数据可重复多次。重复次数的上下限可写在花括号的上下角标处，若重复次数固定，则其上下限相同。

⑤ ()

表示可选，即圆括号内的数据可有可无。

三、加工说明

1.加工说明的作用

加工说明是对数据流图中最基本的加工（即取底层的数据流图中不需再分解的加工，称为基层加工）所做说明。非基层的加工的含义，可以通过对照综合有关的基层加工说明得到解释，故无须再作说明。

严格地说，加工说明是数据字典的一部分，是数据字典中的一种条目类型。因此，它通常又被称为加工小说明。但由于同数据字典内其它条目相比，加工说明有自己鲜明的特点，是结构化设计的关键部分，故将其单独列出。

2.加工说明的格式

加工说明包括以下内容：

①加工名；

②编号；

③激发条件；

④加工逻辑；

⑤执行频率；

⑥优先级；

⑦输入；

⑧输出。

其中，加工逻辑是以文字性的描述对加工所作的解释。需要注意的是，它是对加工“做什么”的描述，它通过对加工过程和步骤的概括说明，来明确规定加工要完成的任务及其程度。因为有时若不涉及具体过程就会对要完成的任务及完成的程度作出不同的理解。加工说明不是用来解释加工应该‘怎么做”的。

对加工逻辑的描述没有固定的方法要求，可以使用多种工具，如结构化语言、判定表、判定树、层次方框图、IPO 图、WARNIER 图等。

3.结构化语言

结构化语言，是一种介于计算机程序设计语言和人们日常所用的自然语言之间的语言形式，它虽不如程序设计语言精确，但简单明了，易于掌握使

用，便于用户理解，又避免了自然语言的不严格、存在二义性等缺点。故适合作为需求分析的工具。之所以称其为结构化语言，是因为它限定只使用三种基本的控制结构，即顺序、选择和循环。结构化语言由外层语法和内层语法两部分组成。

（1）外层语法

外层语法用来规定加工处理的基本结构，说明了所控制各部分的逻辑关系。它也只有顺序、分支、循环三种成分，三种基本结构可互相嵌套，形成任何复杂的处理结构。

①顺序结构

可由一个或多个符合内层语法的简单祈使句、符合外层语法的基本结构顺序排列组成。

②选择结构

其基本形式是：

IF 条件　　THEN 顺序结构 1
　　ELSE 顺序结构 2

或：

IF 条件　　THEN 顺序结构

其中的顺序结构表示相应的处理动作。

③循环结构

其基本形式是：

REPEAT 顺序结构 UNTIL 条件

或：

FOREACH 条件
DO 顺序结构

（2）内层语法

内层语法用来规定内部的语句使用。同外层语法相比，内层语法比较灵活，它由系统人员根据加工的具体特点和用户能接受的程度来决定。一般来说，它有以下特点：

①语态

只祈使句一种语态，即用动词＋名词的结构。用以明确表示此加工“做什么”。

②词汇

a.名词应是数据词典中所定义过的，以力求准确，避免含糊性。

b.动词表示加工中的动作，要避免空洞的语词，如处理、控制、掌握等。

c.不用形容词、副词等修饰语，但可用状语短句。

d.可以用些常用的运算符、关系符等帮助说明条件。

例 2—3—1.“审查读者借书权”加工说明

名称.审查读者借书权
编号：2.2.2
激发条件：收到读者借书证和拟借书
加工逻辑：扫瞄接收借书证码，
　　用借书证码调出读者文档相应记录，
　　IF 无此记录
　　THEN 向加工 2.2.4 发“无借书权信息”

（2.2.4 为显示无借书权信息处理），

ELSE 审查记录，

IF 有未交罚款或有过期书或借书

已满

THEN 向加工 2.2.4 发“ 无借书权信息 ”

ELSE 向加工 2.2.3 发“ 有借书权信息 ”

（2.2.3 为登记处理）。

执行频率：500～1000 人次/天

优先级：一般

输入：有效借书证号

输出：借书权信息

4.判定表和判定树

在加工说明中，除常使用的结构化语言外，对于一些较复杂的多条件的多组合判断处理，还经常用到判定表和判定树。对于描述这种依据多条件组合的判断结果来决定在多目标动作中所采取的策略的处理，采用结构化语言的叙述的方法是不合适的，容易造成处理结构复杂，使没有计算机专业知识的客户难于理解。而采用表格说明的形式，却往往可将复杂的过程描述得逻辑清晰，不需专门知识即使人一目了然。判定表和判定树（又称决策表和决策树）是描述具有复杂逻辑关系的多条件判断、多目标动作的广为使用的形式化工具。

（1）判定表

在判定表中，条件和操作（即目标动作或行动）之间的逻辑关系被明确易懂地表达出来。一个判定表由“ 条件定义 ”、“ 行动定义 ”“ 条件取值 ”“ 行动决策 ”四个部分组成。其结构如图 2—3—3 所示。

条件定义	条件取值
行动定义	行动决策

图 2－3－3 判定表的结构

其中“ 条件定义 ”部分自上而下列出了判断中所用的各种条件，条件的上下位置可以交换，无严格的次序要求。“ 行动定义 ”部分也由上而下列出了可采取的所有动作，排列的顺序也没有严格要求。“ 条件取值 ”和“ 行动决策 ”部分依次列出具体的条件取值数据和所选定的操作动作，其排列位置应按处理情况的“ 条件—行动 ”严格对应。

下面通过例子来说明判定表的绘制过程。

例 2-3-2.一般图书馆在读者借书时，就告知读者还书日期。图书馆自动化系统应有根据借书日期和借书期限计算还书日期的功能。设某馆借书期限为 1 个月，则一般的还书月份可为借书月份加 1 个月，还书日仍为借书日。

但以上计算方法显然存在许多问题。首先，若借书月为 12 月份，则还书月份应为 1 月份而不是 13 月份；还书日期还应该是一个存在的日期，如不能是 2 月 30 号或 9 月 31 号。同时，还书日期还应是图书馆的工作时间，若图书馆 10 月 1 日放假闭馆,则不应让 9 月 1 日借书的读者于 10 月 1 日来还书。

考虑各种可能的还书日期情况，并假设图书馆每周工作 7 天，星期六和星期日不休息，则可列出如表 2-3-1 所示的分析。

表 2-3-1 还书日期分析

借书日期	还书日期
1.28～2.1	3.1
3.31～4.2	5.3
5.31	7.1
8.31～9.2	10.3
10.31	12.1
12.1～12.2	1.3
其它	（月＋1） MOD （12）

注：假设图书馆每周工作 7 天

根据以上分析，可直接得出还书日期判定表。详见表 2—3-2。

表 2-3-2 确定还书日期的判定表

借书日期	1.29～2.1	3.31～4.2	5.31	8.31～9.2	10.31	12.1～12.2	其它
还书日期	3.1	5.3	7.1	10.3	12.1	1.3	（月＋1）MOD12

（2）判定树

判定树的作用和判定表相同，它们之间的区别仅在于判定树以树形结构表示条件和动作，可以认为是判定表的一种变型。两者本质上是一样的。由于判定树采用的是图形表示形式，因而更具示意性、直观性和引导性，更易于问题的表达和理解。具体应用时，可根据自己的习惯选择使用。

判定树的基本结构是一棵从左向右生长的树。其树根在左边，表示加工或所要解决的问题。从左到右的每一列（最后一列除外），对应一类判断条件，自上而下，按对应关系写出其所有取值，位置在右边的判断列，其所有条件取值要根据在其左边的判断列的取值个数重复多次。最后一列是对应的目标动作。如图 2—3—4 所示。

图 2—3—4 判定树

结构化语言、判定表、判定树这三种加工说明方法，各自有其优点和缺点：

①判定表、结构化语言形式化程度高，便于机器理解，容易由计算机自动生成程序。

②判定树以图形方式表达，形象直观，易于人们理解。

③判定表的可验证性强。并能简化判定和决策，提高了决策效率。

第四节　原型法

一、原型法的基本概念

信息系统中的软件开发强调文档化和规范化。规范化要求开发人员按照统一的过程开发软件。对软件 开发过程的规定和描述就是软件开发模型。

目前使用最为广泛的软件开发模型是瀑布模型。瀑布模型是 Bohem 于 1976 年提出的，它也是最早的软件开发模型。瀑布模型将软件开发分为三个时期，它们是：软件计划时期、软件开发时期和软件运行时期。每个时期又分为若干个阶段，每个阶段在上一阶段结束后开始。其开发过程可用图 2—4—1 表示。

瀑布模型是最常规的软件开发模型。它就像瀑布的流水一样，一个阶段一旦结束，就不能对其工作结果进行修正，各个阶段间不存在反馈关系。这种连续、无反馈的特点，保证了软件开发进度，但同时也对每一个阶段提出了严格的，在某些条件下甚至是苛刻的工作质量要求。

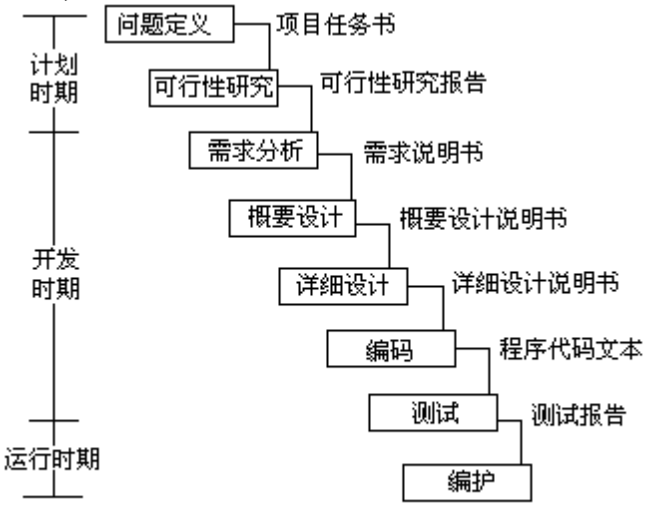


图 2 — 4 — 1 软件开发的瀑布模型

常规的软件生命周期强调开发的阶段性，软件开发过程必须严格遵循分析、设计、实现、测试、维护的次序进行，而且各阶段间没有反馈关系。开发的规范化无疑提高了开发效率，保证了软件的可靠性和可维护性。然而，对于有些类型的实际问题，常规方法也存在着以下一些问题：

①在软件开发的早期，有时很难明确定义出确切的软件需求，提供详细的需求规格说明书。无论是系统人员，还是客户，有的只是对目标系统十分笼统初步的认识。软件系统的很多具体细节往往是随着软件系统的建立而逐步明确的。这样，在需求分析阶段，分析人员常常得花大量时间去捕捉一些非常模糊的想法，并花大量时间以这种模糊的认识为基础去编写包括很多细节内容的需求规格说明书，因而需求规格说明书的一致性、准确性、正确性、有效性很难保证。

②常规的软件开发各阶段相互传递信息的唯一工具是文档。虽然文档内有很多形象的描述方法，如各种图表等，但它们毕竟是实际系统的抽象。即使在软件开发早期作出了明确的需求分析，其后每一个阶段的开发人员都不得不再花大量时间，在一定程度上，通过阅读文档重温前一阶段系统人员的工作。同时，由于这些阶段的系统人员一般不和客户作直接交流，因而，可

能出现的情况是，需求分析中已经得到正确说明的问题，经过这些阶段中不同的系统人员的各种理解和加工后，在继续传递的过程中发生畸变。

以上问题存在的一个很重要的原因，就是在系统人员和客户面前，不存在一个实实在在的事物，这个实体可以充分表达系统人员对问题空间有关概念的理解程度和对目标系统的初步考虑，客户也可通过这个实体，阐明其对目标系统的要求和系统人员当前的一些理解错误。基于这些问题，信息系统开发需要更为实用的方法指导开发过程。原型法即是适应这种需要产生的一种信息系统开发方法。

原型（Prototype）是所开发信息系统的一个可执行模型或引导性版本，它可为客户和软件开发各阶段的系统人员提供系统的原始蓝本，以帮助得到目标系统明确而严格的需求，促进客户和系统人员、各阶段系统人员之间的相互有效的信息交流，并以此为基础进行需求分析和系统设计。从本质上看，原型是未来目标系统的一个模型。

原型不是系统开发的最终产品，而是在某个开发阶段中为达到一定的目的所生成的系统简化模拟版本，它可能只是目标系统的一个式样，或只包含目标系统的某些功能，或仅仅是具有目标系统的一个结构。

二、原型的作用

原型的建立目的可分为以下几种情况：

1.用于验证软件需求的原型

此类原型的目的，就是供系统分析人员在需求分析阶段确定软件需求。这类原型具有两种表现形式：

①垂直原型。原型只包括系统的一部分功能，涉及到一些具体细节。

②水平原型。原型只是整个目标系统的某种简化。

2.用于验证设计方案的原型

此类原型的目的，是供系统设计人员在概要设计和详细设计后，用来验证设计中的某些关键部分的技术性能和可行性。

3.用于产生目标系统的原型

这种原型的建立是一个从简单到全面不断迭代的过程。它体现出系统人员对目标系统认识的逐步完善过程。每当系统人员对系统有了新的认识和想法后，就通过原型实现。这样，随着系统人员工作的深入，原型不断扩充和完善。直到客户满意为止。

三、建立原型的过程

建立不同目的的原型，可使用不同的实现方法和实现过程。下边介绍两种典型原型实现过程。原型实现的过程也可看作是原型系统的生命周期。

1.抛弃式原型建立过程

抛弃式原型（Throw—it—Away）建立方法主要是用来建立验证软件需求和设计方案的原型。由于所需建立的原型是为了满足一时的目的，验证后即弃之不用，因而建立原型时只将注意力集中于要验证的主要方面，而忽视其它一些次要方面。

抛弃式原型的建立步骤是：

（1）确定需求：根据所要完成的任务，确定要建立何种原型，用原型来解决何种问题，以此来决定在构造原型时忽略哪些细节，及重点要验证及解决的问题。

（2）快速设计：在基本的需求分析的基础上，做必要的设计工作。

（3）构造原型：用某种语言或原型开发工具实现原型，得到可执行的原型。

（4）评价原型：通过运行原型，对所要解决的问题进行验证，并可以提出新的要求。通过调整当前原型实现。

（5）实现系统：将运行原型和评价原型所得结果，运用到系统开发中，从而实现系统。得到所要开发的软件产品。

图 2—4—2 所示的流程图说明了以上过程。

2.增量渐进式原型系统的建立过程

增量渐进式原型方法又称软件开发的螺旋模型（SpiralModel），因在其指导下的开发呈现为一个螺旋式上升的过程而得名。增量式原型是由 Boehm 首先提出。其建立目的主要是为了演进出目标系统，因而该类原型的建立过程，正如其名称所表明的那样，是一个循序渐进的过程。原型刚开始很简单，只体现人们最初的想法。以后不断以较小的而又容易

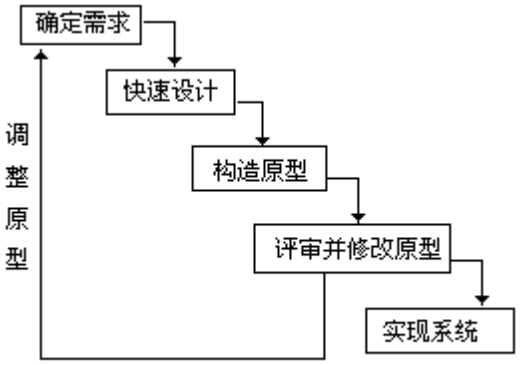


图 2 — 4 — 2 原型构造过程

实现的增量扩展，在已有的原型上加入新的功能。因而，该方法实际上是多次反复原型，并附加相应于不同层次上多次的风险分析。增量渐进式原型的建立过程有以下几个步骤：

（1）通过调查分析，确定客户对目标系统的基本需求。

（2）在对客户基本需求进行分析的基础上，通过简单的设计，用某种方式，快速构造一个系统原型。

（3）将原型交给客户运行，通过运行原型并和客户充分交流，得到对原型进一步的改进意见。

（4）在已有原型的基础上，根据对原型的改进意见，作稍为详细的系统分析和设计，并构造下一代原型系统。

（5）重复以上“运行原型系统—与客户交流—进一步的分析设计—再构造原型”的过程，直到目标系统开发结束。

图 2-4-3 说明了增量渐进式的原型建立过程。

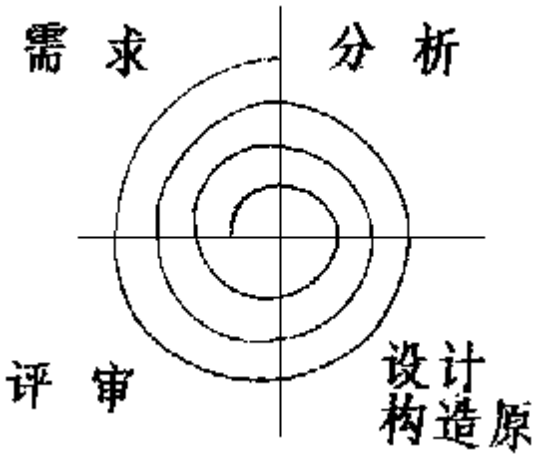


图 2-4-3 增量渐进式的原型建立过程

第三章　系统设计

信息系统开发过程中，系统分析工作完成之后，就应开始系统设计工作。这个时期的主要工作是概要设计和详细设计。这也是信息系统开发过程中的一个非常重要的时期。

第一节　系统设计的内容

一、概要设计

1.概要设计的任务

概要设计又称总体设计或结构设计。概要设计的任务，就是根据需求分析阶段所产生的软件需求规格说明书，建立目标系统的总体结构，即模块划分。具体地说，包括以下几个方面：

①确定各个模块的功能；

②定义各功能模块的接口和调用关系，规定设计限制；

③设计全局数据库和数据结构；

④制定组装测试计划。

概要设计的首要工作，就是将目标系统划分为模块。所谓模块，就是实现某种功能独立、逻辑完整的程序段落。一般来说，子程序、函数、过程、中断、宏、类、程序包等都可称为模块。模块可大可小，可复杂可简单。早期的程序设计不划分模块，在程序规模小、功能简单、使用寿命短的情况下，这样做尚还可以。但随着程序规模增大，程序结构日趋复杂，不划分模块就无法将结构设计同程序设计分开，程序就必然表现为杂乱无章，难以设计、理解和维护。因而，划分模块是软件方法的进步，是软件设计开发中的一项重要工作。

概要设计的依据是需求规格说明书。通过定义模块功能，概要设计完成需求分析工作中数据流图和加工说明所反映出的系统功能。各模块通过调用，一起协调完成系统的整体功能。

2.概要设计的实施步骤

概要设计的实施可分为以下几个步骤：

（1）建立目标系统的总体结构

概要设计首先要确定系统的具体实施方案，然后对目标系统进行功能分解，设计系统的软件总体结构。对不同规模的系统，可有不同的处理层次。对于大型系统，可按主要的软件需求划分成子系统，然后为每个子系统定义功能模块及各功能模块间的关系，并描述各子系统的接口界面。例如图书馆自动化系统，就可先划分为采购、流通、查询等若干子系统，然后在每个子系统内再划分模块；对于一般系统，可按软件需求直接定义目标系统的功能模块及各功能模块间的关系。

（2）给出每个功能模块的功能描述，数据接口描述，外部文件及全局数据定义。

（3）设计数据库及数据结构

数据处理系统或信息系统都处理大量数据，因而对其中的数据库和数据结构要专门进行认真设计。系统分析人员应在需求分析对数据分析的基础上，进一步设计系统数据库和数据结构。数据库的设计分以下四个方面：

■模式设计

■子模式设计

■完整性和安全性设计

■优化

（4）制定组装测试计划

组装测试又叫集成测试，它包括子系统测试和系统测试。一般来说，测试分为三个阶段，即单元测试、组装测试、确认测试。组装测试就是各个模块之间的联调，用以检验模块间的接口性能和模块间协调工作情况，组装测试的实施在单元测试之后，但由于它涉及模块间的关系，故其测试计划的制定同模块划分工作一同完成。同时，在软件开发的早期制定测试计划，有利于软件人员注意提高软件的可测试性。

（5）评审

在以上各步骤结束时，需按软件工程的要求产生相应文档。最后由技术专家和使用部门的管理人员对概要设计的结果从不同角度进行严格的审查，以确认概要设计的结果。广义地说，评审也是一种测试。

3.概要设计的文档

概要设计结束后，应完成所指定的文档。应交付的文件有：

■概要设计说明书；

■数据库/数据结构设计说明书；

■组装测试计划。

以下说明各文档应有的内容。

（1）概要设计说明书

概要设计说明书又叫系统设计说明书。其编制目的在于说明对程序系统的设计考虑：包括基本处理流程、组织结构、模块划分、功能分配、接口设计、运行设计、数据结构设计和出错处理等，为详细设计提供基础。

概要设计说明书包括以下 6 个部分内容：

①引言

②总体设计

总体设计部分，要根据需求规定说明书，进一步明确需求定义和系统的运行环境；尽可能用图表的形式说明系统的基本设计概念、处理流程、系统

结构组成及相互控制关系；通过功能需求与程序的关系表示，说明需求说明书所规定的各项功能的实现同各个程序模块的分配关系。最后阐明系统中人工处理过程和尚未解决的问题。人工处理过程即涉及人机界面的人工处理以及它们和计算机处理过程的关系；尚未解决的问题指在概要设计中还没解决但在系统完成前必须解决的问题。

③接口设计

共有以下三种接口：

用户接口：说明向用户提供的命令和其语法结构，以及软件系统的回答。

外部接口：本系统软件和硬件的接口和各支持软件的接口。

内部接口：系统内各个模块之间的接口和各种控制接口，即总体设计中所定义的模块关系。

④运行设计

运行设计包括以下三方面的问题：

运行模块组合：指当系统处于外界的不同运行控制条件下，系统应采取的不同模块的组合。除了说明各种组合外，还要说明每种运行需要调用的各个模块和支持软件。

运行控制：指出每种外界运行控制的方式、方法和操作步骤。

运行时间：指每种运行模块组合将占用各种资源的时间。

⑤系统数据结构设计

系统数据结构设计包括逻辑结构设计、物理结构设计和数据结构和程序的关系三部分。

逻辑结构设计主要包括：每个数据结构的名称、标识符以及每个结构所包含的数据项（数据元素）、记录、文件等的标识、定义、长度及它们之间的层次或表格间的相互关系。

物理结构设计主要包括：每个数据结构和数据项的存贮要求、访问方法、存取单位、存取的物理关系（索引、设备、存贮区域）、设计考虑和保密条件。

数据结构和访问它们程序的对应关系可用二维表来说明。

⑥系统出错处理设计

系统出错处理设计应包括：出错信息、补救措施、系统维护设计。

出错信息可以用一览表的形式说明每种可能的出错或故障出现时，系统输出信息的形式、含义及处理方法。

补救措施指出错后可能采取的应变措施。一般说，这种措施可以有后备技术、降效技术，恢复或再启动技术。

系统维护设计指在软件系统中安排专为维护用的模块和在运行程序中设置检测点。

（2）数据库/数据结构设计说明书

数据库/数据结构设计说明书的作用在于：对设计中的数据库的所有标识、逻辑结构和物理结构做出具体的设计规定。该说明书共包括引言、外部设计、结构设计、运用设计四部分。

①引言

②外部设计

外部设计包括以下内容：

标识符和状态：定义并说明可唯一标识数据库的标识符和其使用状态，

即是长期使用、临时使用、实验用或测试用等。

使用它的程序，具体内容有：

约定：为了使用本数据库而需要了解的约定，如建立标号、标识的约定，标识不同版本的约定，标识库内文件、记录、数据项的约定等。

专门指导：向研制、测试、维护人员提供的指导，例如被送入数据库的数据格式和标准、输入操作、建立和维护操作等。

③结构设计

结构设计可包括概念结构设计、逻辑结构设计和物理结构设计三部分。

概念结构设计：说明本数据库将反映的现实世界的实体、属性和它们之间关系等的原始数据形式，包括数据项、记录等内容的标识符、定义、类型、度量单位和值域等，建立各种用户视图（E—R 图）。

逻辑结构设计：由现实世界的实体和属性转换成计算机世界的数据结构（数据模型），包括所确定的关键字和属性、重新确定的记录结构和文卷结构、所建立的各个文卷之间的关系等，形成数据库管理员视图。

物理结构设计：建立程序员视图，如：数据在内存的安排（包括索引区、缓冲区的设计）；所使用的外存设备和外存空间的组织（包括索引区、数据块的组织与划分）；访问数据的方式方法。

④运用设计

数据字典设计：把需求说明书中的数据字典转换成数据库所需要的，如标识符的命名、有关信息等。

安全保密设计：说明数据库用户的不同类型和所具有的不同操作权限。

（3）组装测试计划

包括测试策略、方案、测试用例、预期结果、进度计划等。

二、详细设计

1.详细设计的任务

详细设计是系统设计的

第二个阶段。详细设计阶段的主要任务，是在概要设计说明书的基础上，对概要设计中产生的功能模块进行过程描述，设计功能模块的内部细节，包括算法和详细数据结构，为编写源代码提供必要的说明，并建立“ 模块开发卷宗 ”及“ 详细设计说明书 ”。

概要设计解决了信息系统总体结构设计的问题，包括整个信息系统的结构、模块划分、模块功能和模块间的联系等。详细设计则要解决如何实现各个模块的内部功能，即模块设计。具体地说，模块设计就是要为每个模块设计详细的算法、内部数据结构和程序逻辑结构。在概要设计阶段，有时也要进入模块内部，但其目的不是为每一个模块设计算法和数据结构，而是考察该模块的内聚类型，看它是否能被继续分解为更多的模块。

详细设计不是编码，它只是对实现细节作精确的描述。但是，从某种意义上说，详细设计也是系统的实现，它与编码阶段用具体的语言实现的不同之处在于，它是逻辑上的实现。详细设计工作完成之后，产生“ 详细设计说明书 ”及“ 模块开发卷宗 ”，这样，编码阶段可将详细设计中对功能实现的描述，直接翻译、转化为用某种程序设计语言书写的程序。

由于详细设计的结果直接影响代码的生成，所以详细设计的结果基本决定了最终程序代码的质量。因此，详细设计阶段不仅要考虑功能的逻辑实现，逻辑的正确性和性能是否达到要求，也要关注处理过程应简单易懂，易于理

解和维护。尽量引导程序编写人员以良好的风格书写高质量的代码。

因此，详细设计所完成的工作是：

①详细地规定了各程序模块之间的接口，包括参数的形式和传送方式、上下层的调用关系等。

②确定了模块内的算法及数据结构。

2.详细设计的实施步骤

（1）将概要设计产生的构成软件系统的各个功能模块逐步细化，形成若干个程序模块（可编程模块）。

虽然一些模块对概要设计而言已无需进一步划分了，但详细设计是为编码作准备，故就实现而言，从程序结构的清晰和设计算法考虑，常常还可根据情况细化出若干个程序模块。

（2）采用某种详细设计表示方法对各个程序模块进行过程描述。

详细设计的表示方法应便于编码实现时向某种程序设计语言的直接对应，经常使用的方法可分为图形描述方法、语言描述方法和表格描述方法三类。

（3）确定各程序模块之间的详细接口信息。

（4）建立“模块开发卷宗”

“模块开发卷宗”是详细设计产生的重要文档，本节稍后将对其内容作粗略描述。

（5）拟定模块测试方案

模块测试又叫单元测试，它是每一个模块实现后所进行的最基本测试，是对每一个模块功能的确认。模块测试是最先实施的测试。

（6）评审

对以上各步骤的结果进一步审查，确认其正确性和有效性。

3.详细设计的文档

详细设计结束后，应完成以下文档：

■详细设计说明书

■模块开发卷宗

下面详细说明两文件的内容。

（1）详细设计说明书

详细设计说明书又叫程序设计说明书。说明书的目的是用来说明一个软件系统的各个层次中的每个程序（每个模块或子程序）的设计考虑。如果系统比较简单，层次很少，可以不单独编写，把有关的内容并入概要设计说明书中。详细设计说明书的内容包括引言、程序系统的组织结构以及各个程序的设计说明几个部分。

①引言

引言论述详细设计说明书的编写目的、背景，需特别说明的定义和有关的参考资料等内容。

②程序系统的组织结构

用一系列图表列出本程序系统内的每个程序（包括每个模块和子程序）的名称、标识符和它们之间的层次结构关系。

③程序（标识符）设计说明

程序（标识符）设计说明对本程序系统内的各个程序（包括每个模块和子程序）的有关设计内容作逐一说明，对每个程序的说明包括以下 13 项内

容。

程序描述：简要说明程序的主要内容，如：本程序的目的、意义，本程序的特点（是否常驻内存，是否有子程序，是否可重入，有无覆盖要求，是顺序还是并发处理等。）

功能：采用某种设计方法对各个子程序模块进行过程描述。

性能：是对程序系统运行效率的说明，包括空间要求、算法精度、灵活性、时间特性等。

输入项：列举每个输入项的名称、标识、数据的类型和格式、数据值的有效范围、输入的方式、数量和频度、输入媒体、输入数据来源和安全保密等。

输出项：内容同输入项类似，但加上输出的图形和符号说明。

算法：详细描述具体的实现算法，包括计算公式和计算步骤等。

接口：说明本程序和上、下层模块的关系，参数赋值和调用方式，和它直接相关的数据结构（数据库、数据文卷等）。

流程逻辑：用图表的形式说明程序的处理流程。

存贮分配：说明程序对存贮器的使用和分配情况。

注释设计：说明需在程序中安排的注释。如：加在模块首部的注释，加在各分支点处注释，对各变量的功能、范围、缺省条件注释等，对使用逻辑等所加的注释。

限制条件：说明本程序系统在运行时所受到的限制。

测试计划：对本程序进行单元测试的计划，包括：测试的技术要求、输入数据、预期结果、进度安排、人员职责、设备条件驱动程序及桩模块等的规定。

（2）模块开发卷宗

模块开发卷宗是在模块开发过程中逐步编写出来的，每完成一个模块或一组密切相关的模块的复审时编写一份。应把所有的模块开发卷宗汇集在一起，目的是记录和汇总低层次开发的进度和结果，以便于对整个模块开发工作的管理和复审，并为将来的维护提供非常有用的技术信息。模块开发卷宗是组织和保存开发过程中不断产生的文档的有效方法。

模块开发卷宗包括以下内容：

①标题封面；

②模块开发情况表：用二维表格说明开发进度；

③功能说明；

④设计说明；

⑤源代码清单；

⑥测试说明；

⑦复审的结论。

第二节　系统设计的原则

一、结构化设计和结构程序设计

1.结构化设计

结构化设计是概要设计中被广泛使用的一种方法。它最早由美国 IBM 公司的 W.Stevens、G.Myers 和 L.Constantine 三人所提出的。结构化设计的思

想可应用于任何软件系统的设计，而且可成为衔接需求分析阶段的主要方法——结构化分析方法和详细设计阶段的结构化程序设计方法之间的工具，三者可以配合使用，形成一套系统的软件开发方法。

结构化设计的基本思想是将系统设计成由相对独立、功能单一的模块群组成的结构。它的基本内容有以下三个方面：

■研究模块分解的影响。

■提出评价模块结构质量的两个具体标准——耦合度和内聚度。

■从数据流图导出模块结构的规则。

2.结构化程序设计

结构化程序设计是 60 年代产生的一种程序设计理论和方法,它是目前使用最为广泛且为实践所证明行之有效的程序设计方法。

结构化程序设计方法是针对传统的非结构化程序设计而言的。在计算机技术的发展过程中，由于早期的计算机内存容量小，运行速度慢，为使程序得以装入内存并有可接收的运行效率，程序设计人员不得不精打细算，力争节约每一个存贮单元，抓住每一个可以提高运行速度的机会，结果形成强调技巧，重视效率的程序设计习惯。同时，由于软件开发没有一定的规范，程序设计又带有强烈的个人色彩，一些程序设计人员将设计看作是表现个人技巧及个性，显示聪明才智的机会，程序设计风格因人而异，而所设计出的程序往往是结构混乱、晦涩难懂。不仅其他人难以理解，就是设计者本人，经过一段时间之后，都无法读懂。而随着计算机硬件技术的发展，内存和速度已不再是制约程序设计的因素。同时，由于软件需求不断增加，要开发的软件数量和规模越来越大，程序设计已不再是一种个体行为，而成为群体合作的项目；程序设计也不再是一次性的编码，而需要反复地维护。大规模高效率的软件生产要求有新的方法指导程序设计，结构化程序因此而产生了。

结构化程序设计的概念最早是由 E.Dijkstra 提出的。由于 GOTO 语句是实现随意控制的极好工具，极易造成 BS（ABOWLOFSPAGHETTI）型的结构混乱的程序，1965 年 Di－jkstra 主张从一切语言中取消 GOTO 语句。他认为，程序的质量同程序中 GOTO 语句中数量成反比。但由于当时 GOTO 语句是 FORTRAN 语言实现程序控制的重要语句，而 FORTRAN 是当时最为盛行的程序设计语言。因而 Dijkstra 的主张并未引起人们的注意。1964 年 Boehm 和 Jacopini 证明，只用“顺序”、“选择”和“循环”就可实现所有单入口、单出口的程序。图 3—2—1 为这三种结构的控制流程。由于循环控制可用顺序和选择结构来实现，因而本质上只有两种基本结构。1966 年 Boehm 和 Jacopini 以“流图、图灵机和仅含两个格式规则的语言”为题，在极有影响的杂志 Communi－cationofACM 上发表了他们的研究成果，并在文中断言，用任何高级语言编写的程序都可以分解为上述三种基本控制结构及它们的组合。他们的证明使人们开始重视 Dijkstra 的观点,并为结构化程序设计奠定了理论基础。

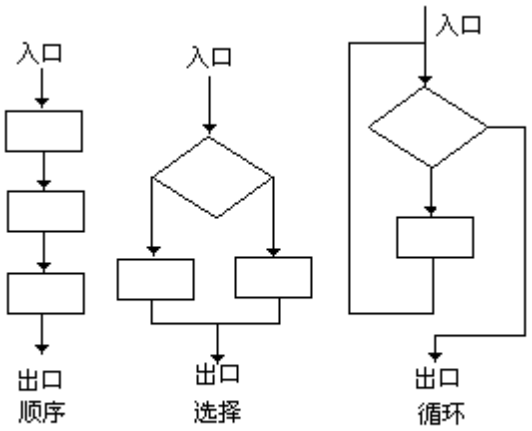


图 3-2-1　结构化程序设计的基本结构

60 年代中，Dijkstra 进一步提出程序设计的层次化和抽象的观点，并多次建议从一切高级语言中取消 GOTO 语句，他的这种对 GOTO 语句根本性的否定态度引起激烈的争论。经过争论，人们意识到，不是简单地取消一个语句的问题，而是要确立一种新的程序设计方法和思想，以显著地提高软件生产率和降低维护代价。经过长期的努力，结构化程序设计的思想和方法逐步建立和完善起来。

随着软件开发工程化观念的提出和建立，结构化程序设计（SP）又向软件设计阶段扩展，形成结构化设计（SD），结构化设计又向软件分析阶段扩展，形成了结构化分析（SA），从而形成了系统的结构化方法。

结构化程序设计的基本原则是：

①采用自顶向下，逐步求精的设计方法；

②用顺序、选择和循环三种基本控制结构实现单入口和单出口的程序。

一个不含 GOTO 语句,并仅由以上三种控制结构形成的具有单入口和单出口的结构化程序有以下两方面的优点：

■程序的动态结构和静态结构一致，易于理解和维护。

■有利于程序正确性的证明。只有三种结构的程序可用严格的方法证明其正确性。

二、系统设计的原则

在长期的软件开发实践中，为了提高软件的开发质量，人们总结出了一些软件开发的基本原则，结构化设计最为直接地体现了这些原则的应用。下边具体说明这些原则的基本内容。

1.模块化

人们在解决问题，尤其是大规模的复杂问题时，常常使用“分解”的方法，将问题划分若干个较小的问题，通过对各个较小问题的求解，达到对复杂问题的解决。模块化就是体现人类在问题求解时的这一方法和思想。为了使一个复杂的大型程序系统能被人的智力所管理，模块化是复杂软件系统必须具备的属性。因为，如果不把一个大型的、复杂的系统分解成若干模块，将很难对其开发和管理。

所谓模块化（Modularity），就是依据一定的原则，将欲开发的软件系统分解为若干部分，即模块。如果对

第一次划分出的模块直接求解复杂度仍较高，则可继续分解，直到划分为易于直接解的规模为止。模块的概念我们在本章

第一节中曾提到。所谓模块，就是实现某种功能独立、逻辑完整的程序段落，模块也是数据说明，是可执行语句等程序对象的集合。模块被单独命名，并能通过名字被访问。模块化降低问题的复杂程度可从下边说明中得到论证。

设函数 C（x）表示问题的复杂程度，函数 E（x）表示求解问题所需的工作量，若有两个问题 P1 和 P2，如果

C（P1）＞C（P2）则显然有

E（P1）＞E（P2）

同时，我们还有另外一条规律，若一个问题 P 可被分解为两个子问题 P1 和 P2，即：

P=P1＋P2

则

C（P）＞C（P1）＋C（P2）

因而

E（P）＞E（P1）＋E（P2）

以上规律充分说明，模块化降低了问题的复杂程度，减少了求解问题的工作量。

但是，我们并不可由此得出结论，在软件开发时，模块划分得越多越好，问题分解得越细越好，当模块被划分成最基本的操作，问题就自然而然得到解决了。事实上，模块化要掌握适当的程度。因为，模块划分降低了问题的复杂程度，但也增加了模块间相互协调工作，即配合完成任务的接口复杂度。若将接口因素考虑进去，则上述规律可做如下修正：

仍设 P=P1＋P2

设 I（i，j）为模块 Pi 对模块 Pj 的接口复杂度因子；

I（i，j）×Pi 为模块 Pi 对模块 Pj 的接口工作量；

则有

C（P）→C（P1＋I（1，2）×P1）＋C（P2＋I（2，1）×P2）

E（P）→E（P1＋I（1，2）×P1）＋E（P2＋I（2，1）×P2）

显然，不同的模块划分有不同的接口因子。只有当模块划分合理，数量规模适当，接口因子较小时，才有下式成立：

C（P）＞C（P1＋I（1，2）×P1）＋C（P2＋I（2，1）×P2）

E（P）＞E（P1＋I（1，2）×P1）＋E（P2＋I（2，1）×P2）

因此，在模块划分时，存在着一个最佳模块数，最佳的模块划分应符合模块独立性原则。

图 3-2-2 表示了最佳模块划分的范围。

结构化设计就是根据这一规律，提出把系统设计成由若干模块所组成的结构。每个模块都相对独立、功能单一。这样，各个模块可以独立地被理解、编写、测试、排错和修改。整个系统结构清晰，易于实现、理解和维护。结构化设计能提高软件系统的质量和可靠性，也有助于整个工程的开发和管理。

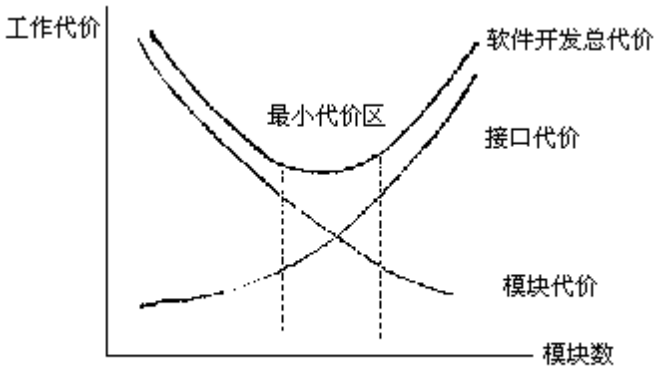


图3-2-2 最佳的模块划分范围

2.抽象

抽象（Abstraction）是人类认识问题和解决问题的基本工具和方法。在解决复杂的具体问题时，人们往往先忽略其细节和非本质的方面，而集中注意力去分析问题的本质和主要方面，搞清所要解决的问题的本质所在；同时人们在总结认识和实验规律时，也往往突出各类问题的共性，找出各种客观事物、状态和过程间的联系和相似性，加以概括和提取，即抽象。抽象是具有有限思维能力的人类个体同复杂外部世界相互了解的有力工具。

抽象在软件开发过程中也具有重要的地位。复杂软件系统的构造就是一个运用抽象的过程。通过对所要解决问题的抽象，进行需求分析；然后借助较低层次上的抽象，采用更加过程化、形式化的方法，进行系统设计；最后，在最低的抽象层次上，用可以直接实现的方法，叙述问题的解法。

因此，在本质上，抽象的过程是一个逐步求精的过程。Wirth 曾对抽象作过如下解释：抽象是我们对付复杂问题最重要的办法，所以，对一个复杂的问题，不应马上用计算机指令、数字与逻辑字来表示，而应该用较为自然的抽象语句来表示，从而得出抽象程序。抽象程序对抽象的数据进行某些特定的运算并用某些合适的记号（可能是自然语言）来表示。对抽象程序作进一步的分解，并进入下一层的抽象，这样的精细化过程一直进行下去，直到程序能被计算机接受为止。

3.信息隐藏和信息局部化

信息隐藏（InformationHiding）和信息局部化（Inforam－tionLocalization）是软件设计中另外两项重要原则。所谓信息隐藏，是指在设计和确定模块时，应使一个模块内包含的信息（过程和数据）对于不需要这些信息的模块来说是不可访问的。

信息隐藏使得模块间尽可能彼此独立，有利于过程和数据的保护，避免了错误的传递，提高了系统的可靠性。信息隐藏尤其为软件系统的维护提供了良好的基础。

信息局部化是指将一些关系密切的成份，设计时放得彼此靠近。局部化有利于模块的单独开发和调试，因而简化了整个系统的设计和实现。同时，局部化也是信息隐藏的手段。

4.一致性、完整性和确定性

一致性、完整性和确定性是针对软件大规模长时间的生产，对生产过程的规范、统一提出的要求。

所谓一致性（Uniformity），是指软件系统各部分中符号的表达使用、

对象及过程的描述和调用形式、操作的控制结构都一致，以免造成混乱。

完整性（Completeness）是说对一对象、过程的表达描述及处理应该完备，没有遗漏重要的内容或成份。

确定性（Comfirmability），又称可验证性，是指系统中的对象、过程定义明确，无二义性，容易测试。

第三节　系统设计中的图形方法

一、图形工具——模块结构图

模块结构图（ModuleStructureChart）是结构化设计的主要工具，它被广泛地使用在概要设计之中。模块结构图是由美国的 Yourdon 于 1974 年首先提出，并用来描述软件系统的组成结构及相互关系。它即反映了整个系统的结构，即模块划分，也反映了模块间的联系。

1.符号规定

（1）模块

用一个方框来表示软件系统中的一个模块，框中写模块名。名字要恰当地反映模块的功能，而功能在某种程度上反映了块内各成份间的联系。如图 3-3-1 所示。

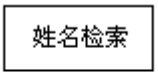


图 3-3-1 模块表示符号

（2）调用

用一个带箭头的线段表示模块间的调用关系。它联结调用和被调用模块，箭头指向被调模块，箭头发出模块为调用模块。如图 3—3—2 所示。根据调用关系，模块可相对地分为上层模块和下层模块。具有直接调用关系的模块之间相互称为直接上层模块和直接下层模块。如图 3—3—2 所示的模块 A 和模块 B，及模块 B 和模块 C。

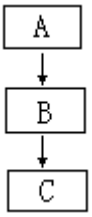


图 3-3-2 模块间的调用

调用是模块间唯一的联系方式。通过调用，各个模块有机地组织在一起，协调完成系统功能。一般只允许上层模块调用下层模块，而不允许下层模块调上层模块。

（3）数据

用小箭头表示模块间在调及过程中相互传递的数据信息。数据信息传递画在调用箭头旁边，小箭头指出传送方向。如图 3—3—3a 和 3—3—3b 所示。

模块间传递的数据信息还可进一步分为两类：作数据用的信息和作控制用的信息。若需要进一步区分，可在小箭头的尾部使用不同的标记表示，具体可分为以下三种箭头：

尾部无标记，表示不区分两类信息。

尾部有小空心圆圈标记，表示作数据用的信息。

尾部有小实心圆圈标记，表示作控制用的信息。

（4）调用编号和参数表

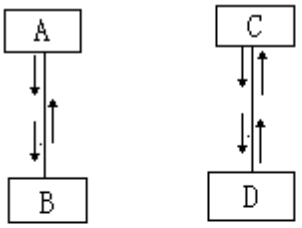


图3-3-3a

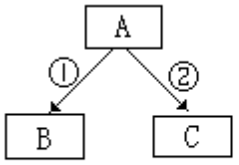
A 模块调用 B 模块。调用时，A 向 B 传送数据 X 和 Y，调用结束时，从 B 返回数据 Z 。

图3-3-3b

C 模块调用 D 模块。调用时，C 向 D 传送数据 X 和 Y ，调用结束返回时，D 向 C 传送数据 Y 和 Z 。需要注意的是：Y 在 D 中已被加工，返回时已不是原来的 Y 。

当模块间输入输出数据较多，用数据小箭头表示无法将数据名称写清楚时，可采用此种方法。模块调用较多时通过参数表，数据传递也表示得更加清晰。

用参数表表示时，给每个调用箭头一个顺序编号，然后按编号列出输入输出参数表。如图 3—3—4a 所示。



	输入	输出
1	X, Y	Z
2	Z	S, T, V

图3-3-4a 参数表表示数据传递

输入输出表和完整的结构图功能是相同的，如图 3-3-4a 和 3-3-4b 所示。采用哪种形式要根据具体情况。

（5）辅助符号

为表示模块间复杂的调用关系，模块结构图使用了两种辅助符号表示不同的调用，它们是：

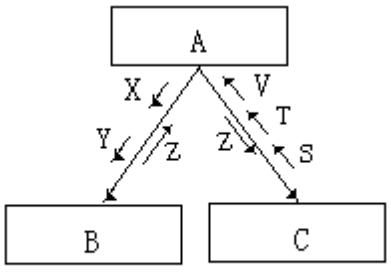
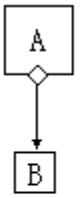


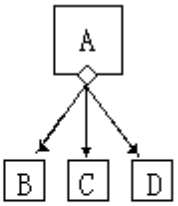
图3-3-4b 完整的模块结构图表示数据传递

选择调用（或称条件调用）：在调用箭头的发出端用一个小菱形框表示。选择调用为上层模块根据条件调用它的多个下层模块中的某一个。如图 3-3-5a 和图 3-3-5b 所示。



若条件成立
则调用B

图 3-3-5a



若条件1成立，则调用B
若条件2成立，则调用C
若条件3成立，则调用D

图 3-3-5b

循环调用：在调用箭头的发出端用一带箭头的圆弧表示。循环调用为上层模块反复调用它的一个或若干个模块。如图 3—3—6 所示。

2.对模块结构图的说明

①模块结构图和程序流程图外型相似，但两者意义完全不同。一个程序系统有两方面的性质，一是过程性，一是层次性。过程性说明程序执行的先后次序，流程图是描述过程性的，其箭头表示的是执行顺序，而不说明程序系统的层次关系；层次性则说明模块之间的层次结构，模块结构图就是说明层次性的，其箭头是表示调用关系（层次关系）而不表示执行的先后次序。

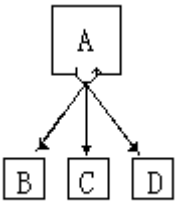


图 3-3-6

②模块结构图的最后形态是多种多样的，有树形的，也有清真寺形的（上下部分窄，中间部分宽）。不同的形态对应不同的结构划分。

③模块结构图并不严格地表明调用次序。虽然多数人习惯按调用次序由左向右画，但模块结构图无此规定。

④模块结构图也不指明什么时候调用下层模块。通常模块中除了调用语句之外，还有其它语句，但模块内究竟还有其它什么内容，执行顺序如何，模块结构图没有说明。

二、其它图形工具

在系统的概要设计和详细设计中，还有一些其它的图形工具，如 HIPO 图、PAD 图，盒图等，被广泛使用。下边逐一作介绍。

1.HIPO 图

HIPO 图是一个同模块结构图等价的结构化设计图形工具，它也被广泛地使用在概要设计阶段。

HIPO 图（HierachyInputProcessOutput），即层次化的输入—处理—输出图。它是美国 IBM 公司于 70 年代中期采用的。HIPO 图实际上是层次图和 IPO 图的结合。有人把 H 部分叫目录表。两者结合后在功能上相当于模块结构图。

为说明 HIPO 图，我们先简要介绍 IPO 图和层次图。

（1）IPO 图

IPO 图是输入—输出—处理图的简称。它也是美国 IBM 公司发展并完善起来的一种图形工具。它具有简单、易用、描述清晰的特点，用来表示一个

加工比较直观，对设计很有帮助。

一个完整的 IPO 图由三个大方框组成。左边的方框内写有关的输入数据，称输入框；中间的方框列出对输入数据的处理，称处理框；右边的方框写处理所产生的输出数据，称输出框。处理框中从上至下的顺序表明系统操作的次序。输入数据同处理的关系，处理同输出数据的关系，用联接有关部分的箭头来表示。如图 3-3-7 所示。

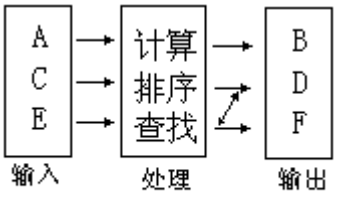


图3-3-7 IPO图的例子

（2）层次图

层次图也叫 H 图，它是一个表示信息系统结构的有效工具。同模块结构图类似，但比较简单。层次图一个方框表示一个模块，方框内写模块名称。用方框间的连线表示模块间的层次关系。层次图非常自然地表达了自顶向下的分析思想。图 3—3—8 为一个层次图的实例。

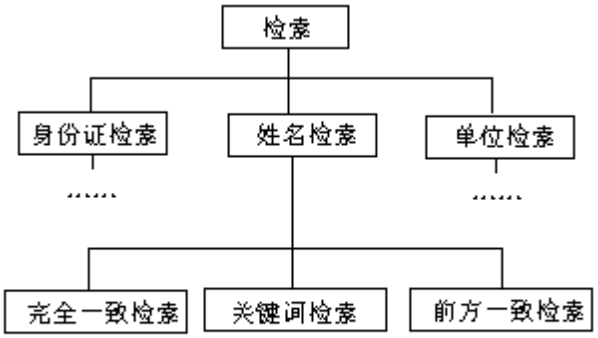


图3-3-8 层次图的例子

层次图除以上部分外，为清晰和方便，还可以使用编号和表格，用表格说明编号的具体名称或内容。

需特别注意的是，虽然层次图和模块结构图外型相似，但两者所表示的内容完全不同。层次图说明模块之间的层次关系，但这种层次关系是包含关系而非调用关系，层次图也无法表达调用过程中的数据交换。

以上简要介绍了 IPO 图和层次图。这两个图形工具不仅可以作为概要设计的工具，也可以作为需求分析的工具，关键在于它们所表达的数据、处理和功能的详略层次。

（3）HIPO 图

HIPO 图是在 IPO 图和层次图基础上发展起来的，它是两图的有机结合。HIPO 图首先用一个层次图描述软件系统的结构，对于层次图中的每一个模块，都附加一个 IPO 图，用以说明具体的输入输出数据和处理过程。即在 HIPO 图中，每一个层次图都对应一套 IPO 图。为使对应关系明确，除最顶层图外，对层次图中每个模块都给一个编号，同该模块对应的 IPO 图也给一个相同的编号，编号规则同数据流图。如图 3-3-9a 和 3-3-9b 所示。

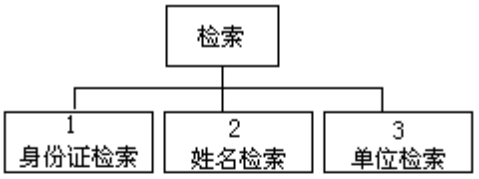


图 3-3-9a HIPO 图中的层次图部分

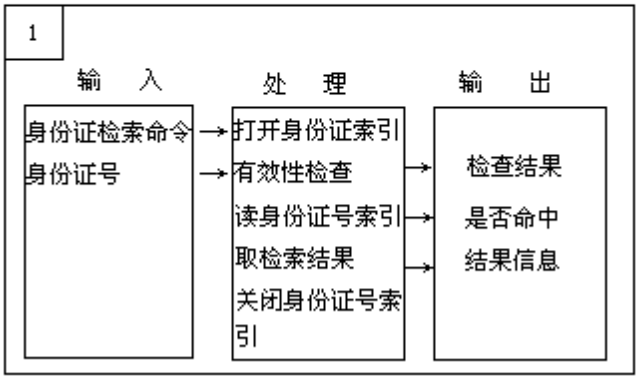


图 3-3-9b HIPO 图中的 IPO 图部分

2.程序流程图、PAD 和盒图

与上边所介绍的概要设计时使用的图形工具不同，程序流程图、PAD 图和盒图是详细设计时所使用的工具。这些工具均能精确描述程序的处理过程，无歧义地表达处理功能和控制流程，乃至数据的作用范围，并能在编码阶段直接将其翻译成为用某种具体的程序设计语言书写的程序代码。

在详细设计阶段，用于设计分析和结果描述的方法有三类，即图形描述方法、语言描述方法和表格描述方法。这里只介绍图形描述方法。

（1）程序流程图

程序流程图（FlowChart）又叫框图，是一种传统的过程描述方法。其特点是直观、灵活、方便。它所使用的基本符号有：

方框：表示一个处理，处理内容写于框内。

菱形框：表示一个判断，判断条件写于框内。

椭圆框：表示开始或结束。

箭头：表示程序流程。

图 3-3-10 为一个流程图的例子。

由于程序流程图有十分灵活的特点，其箭头使用有很强的随意性，使设计人员的思想不受任何约束，因而使用不当会导致产生非结构化程序和十分混乱结果。这是它的一个致命缺点。同时，程序流程图不能表示数据结构；它诱导设计人员过早地考虑程序实现的细节，而非系统的总体结构。因而，它不是结构化设计工具，不能体现自顶向下的设计思想。所以长期以来，很多人一直建议停止使用它。

（2）PAD 图

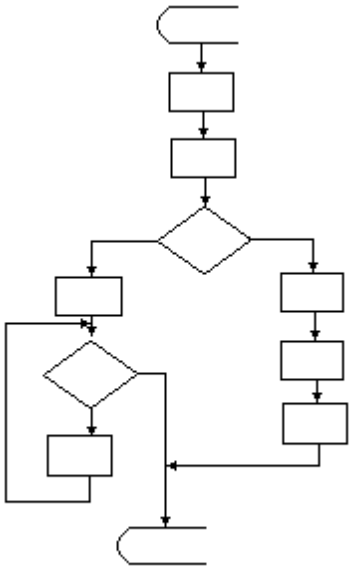


图3-3-10 程序流程图

PAD 图又称问题分析图（ProblemAnalysisDiagram）。它是 70 年代日本的日立公司发明使用的，是一种具有很强结构化特征的分析工具。目前已被广泛地使用在详细设计中。PAD 图是由其基本符号沿两个方向展开，图 3—3—11 显示了它所使用的基本符号。

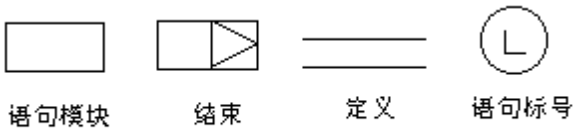


图3-3-11 PAD图的基本符号

PAD 图具有很强的结构化描述特征。它的基本图形符号只能构成三种控制流程，即顺序、选择和循环结构。如图 3—3—12 所示。

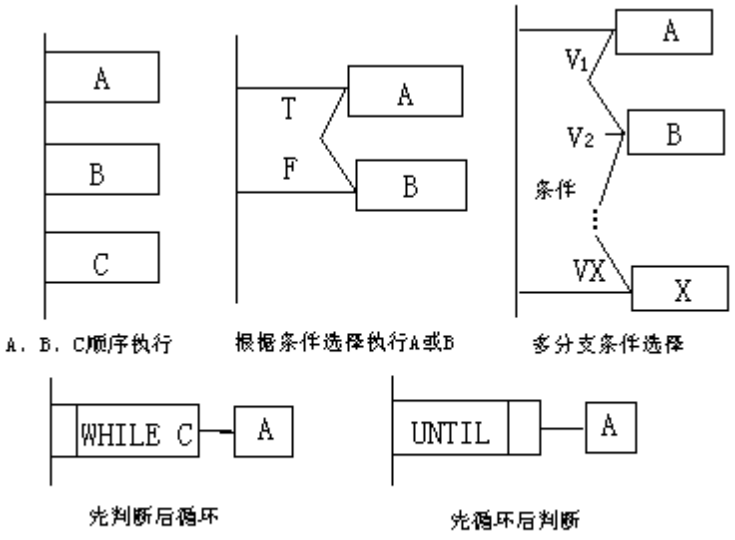


图3-3-12 PAD图的控制结构

PAD 图具有以下特点：

①具有强烈的结构化特征，支持自顶向下、逐步求精的设计方法，设计出的结构必然是结构化的；

②逻辑清晰，易懂、易用；

③即要设计程序结构，又可表示数据结构；

④容易将图直接转换为高级语言程序。

（3）盒图

盒图（BoxDiagram）又称 N—S 图，它是为满足结构化程序设计的需要，克服传统设计工具的缺点，特别是为取消程序流程图的随意转向功能，于 70 年代由 Nassi 和 Shneider－man 提出使用的，故称 N-S 图。盒图的符号规定和使用如图 3-3-13 所示。

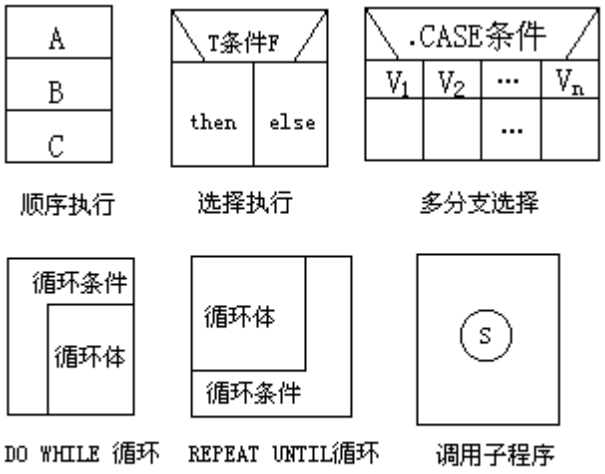


图3-3-13 盒图的符号使用

盒图具有以下特点：

①过程的作用域明确；

②不能随意转移控制；

③容易区分全局变量和局部变量；

④容易表示嵌套关系和层次关系；

⑤强烈的结构化特征。

第四节　内聚度和耦合度

一、联系

当个程序段或语句（指令）引用了其它程序段或语句（指令）中所定义或使用的数据名（即存贮区、地址等）或代码时，他们之间就发生了联系。一个程序被划分为若干模块时，联系既可存在于模块之间，也可存在于一个模块内的程序段或语句之间，即模块内部。联系反映了系统中程序段或语句之间的关系，不同类型的联系构成不同质量的系统。因此，联系是系统设计必须考虑的重要问题。

系统被分成若干模块后，模块同模块的联系称为块间联系；一个模块内部各成份的联系称为块内联系。显然，模块之间的联系多，则模块的相对独立性就差，系统结构就混乱；相反，模块间的联系少，各个模块相对独立性就强，系统结构就比较理想。同时，一个模块内部各成份联系越紧密，该模块越易理解和维护。

二、评判模块结构的标准

1.模块独立性

模块化是软件设计和开发的基本原则和方法，是概要设计最主要的工

作。模块的划分应遵循一定的要求，以保证模块划分合理，并进一步保证以此为依据开发出的软件系统可靠性强，易于理解和维护。根据软件设计的模块化、抽象、信息隐蔽和局部化等原则，可直接得出模块化独立性的概念。所谓模块独立性，即：不同模块相互之间联系尽可能少，应尽可能减少公共的变量和数据结构；一个模块应尽可能在逻辑上独立，有完整单一的功能。

模块独立性（Moduleindependence）是软件设计的重要原则。具有良好独立性的模块划分，模块功能完整独立，数据接口简单，程序易于实现，易于理解和维护。独立性限制了错误的作用范围，使错误易于排除，因而可使软件开发速度快，质量高。

为了进一步测量和分析模块独立性，软件工程学引入了两个概念，从两个方面来定性地度量模块独立性的程度，这两个概念是模块的内聚度和模块的耦合度。

2.块间联系的度量—耦合度

耦合度是从模块外部考察模块的独立性程度。它用来衡量多个模块间的相互联系。一般来说，耦合度应从以下三方面来考虑，即：

耦合内容的数量，即模块间发生联系的数据和代码的多少，同这些数据和代码发生联系的模块的多少，多的耦合强，少的耦合弱；

模块的调用方式，即模块间代码的共享方式。可分为用 CALL 语句调用方式和用 GOTO 语句直接访问方式；

模块间的耦合类型。耦合类型有以下几种方式：

①独立耦合

②数据耦合

③控制耦合

④公共耦合

⑤内容耦合

下面重点对各种类型的耦合作进一步的说明。

（1）独立耦合

指两个模块彼此完全独立，没有直接联系。它们之间的唯一联系仅仅在于它们同属于一个软件系统或同有一个上层模块。这是耦合程度最低的一种。当然，系统中只可能有一部分模块属此种联系，因为一个程序系统中不可能所有的模块都完全没有联系。

（2）数据耦合

指两个模块彼此交换数据。如一个模块的输出数据是另一个模块的输入数据，或一个模块带参数调用另一个模块，下层模块又返回参数。应该说，在一个软件系统中，此种耦合是不可避免的，且有其积极意义。因为任何功能的实现都离不开数据的产生、表示和传递。数据耦合的联系程度也较低。

（3）控制耦合

若在调用过程中，两个模块间传递的不是数据参数而是控制参数，则模块间的关系即为控制耦合。控制耦合属于中等程度的耦合，较之数据耦合模块间的联系更为紧密。但控制耦合不是一种必须存在的耦合。

当被调用模块接收到控制信息作为输入参数时，说明该模块内部存在多个并列的逻辑路径，即有多个功能。控制变量用以从多个功能中选择所要执行的部分，因而控制耦合是完全可以避免的。排除控制耦合可按如下步骤进行：

①找出模块调用时所用的一个或多个控制变量；

②在被调模块中根据控制变量找出所有的流程；

③将每一个流程分解为一个独立的模块；

④将原被调模块中的流程选择部分移到上层模块，变为调用判断。

通过以上变换，可以将控制耦合变为数据耦合。由于控制耦合增加了设计和理解的复杂程度，因此在模块设计时要尽量避免使用。当然，如果模块内每一个控制流程规模相对较小，彼此共性较多，使用控制耦合还是合算的。

（4）公共耦合

公共耦合又称公共环境耦合或数据区耦合。若多个模块对同一个数据区进行存取操作，它们之间的关系称为公共耦合。公共数据区可以是全程变量、共享的数据区、内存的公共复盖区、外存上的文件、物理设备等。当两个模块共享的数据很多，通过参数传递可能不方便时，可以使用公共耦合。公共耦合共享数据区的模块越多，数据区的规模越大，则耦合程度越强。公共耦合最弱的一种形式是：两个模块共享一个数据变量，一个模块只向里写数据，另一个模块只从里读数据。

当公共耦合程度很强时，会造成关系错综复杂，难以控制，错误传递机会增加，系统可靠性降低，可理解、维护性差。

（5）内容耦合

内容耦合是耦合程序最高的一种形式。若一个模块直接访问另一模块的内部代码或数据，即出现内容耦合。内容耦合的存在严重破坏了模块的独立性和系统的结构化，代码互相纠缠，运行错综复杂，程序的静态结构和动态结构很不一致，其恶劣结果往往不可预测。

内容耦合往往表现为以下几种形式：

①一个模块访问另一模块的内部代码或数据；

②一个模块不通过正常入口而转到另一个模块的内部(如使用 GOTO 语句或 JMP 指令直接进入另一模块内部）；

③两个模块有一部分代码重迭（可能出现在汇编程序中，在一些非结构化的高级语言，如 COBOL 中也可能出现）；

④一个模块有多个入口（这意味着一个模块有多种功能）。

一般讲，在模块划分时，应当尽量使用数据耦合。少用控制耦合（尽量转成数据耦合），限制公共耦合的范围，完全不用内容耦合。

3.块内联系的度量——内聚度

内聚度（Cohesion）是模块内部各成份（语句或语句段）之间的联系。显然，模块内部各成份联系越紧，即其内聚度越大，模块独立性就越强，系统越易理解和维护。具有良好内聚度的模块应能较好地满足信息局部化的原则，功能完整单一。同时，模块的高内聚度必然导致模块的低耦合度。理想的情况是：一个模块只使用局部数据变量，完成一个功能。

按由弱到强的顺序，模块的内聚度可分为以下 7 类，现分述于下。

（1）偶然内聚

块内的各个任务（通过语句或指令来实现的）没有什么有意义的联系，它们之所以能构成一个模块完全是偶然的原因。如下图 3-4-1 所示。

在模块 T 有三条语句。至少从表面上看不出这三条语句之间有什么联系，只是由于 P，Q，R，S 四个模块中都有这三条语句，为了节省空间才把它们作为一个模块放在一起。这完全是偶然性的。偶然内聚的模块有很多缺点：

由于模块内

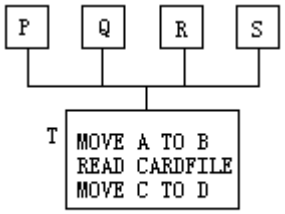


图3-4-1　偶然内聚

没有实质性的联系，很可能在某种情况下一个调用模块需要对它修改而别的模块不需要。这时就很难处理。同时，这种模块的含义也不易理解，甚至难以为它取一个合适的名字，偶然内聚的模块也难于测试。所以，在空间允许的情况下，不应使用这种模块。

（2）逻辑内聚

一个模块完成的任务在逻辑上属于相同或相似的一类（例如，用一个模块产生各种类型的输出），则该种模块内的联系称为逻辑内聚。如图 3-4-2a 和 3-2-2b 所示。

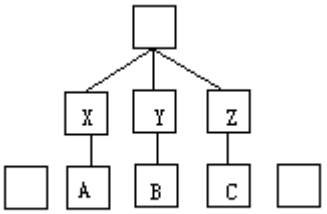


图3-4-2a

在图 3-4-2a 中，模块 A，B，C 的功能相似但不相同，如果把它们合并成一个模块 ABC，如图 3-4-2b 所示，则这个模块就为逻辑内聚，因为它们是由于逻辑上相似而发生联系的。逻辑内聚是一种较弱的联系。实际执行时，当 X，Y，Z 调用合成的模块 ABC 时，由于原 A，B，C 并不完全相同，所以还要判别是执行不同功能的哪一部分。

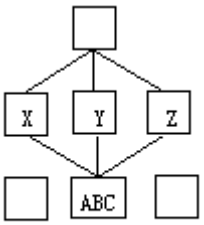


图3-4-2b 逻辑内聚

逻辑内聚存在的问题是：

①修改困难，调用模块中有一个要对其改动，还要考虑到其它调用模块；

②模块内需要增加开关，以判别是谁调用，因而增加了块间联系；

③实际上每次调用只执行模块中的一部分，而其它部分也一同被装入内存，因而效率不高。

（3）时间内聚

时间内聚是指一个模块中包含的任务需要在同一时间内执行（如初始化，结束等所需操作）。如图 3—4—3 所示的模块。与偶然内聚和逻辑内聚相比，这种内聚类型要稍强些，因为至少在时间上，这些任务可以一起完成。但时间内聚和偶然内聚、逻辑内聚一样，都属低内聚度类型。

（4）过程内聚

如果一个模块内的各个处理元素是相关的，而且必须按固定的次序执行，这种内聚就叫做过程内聚。过程内聚的各模块内往往体现为有次序的流程。如图 3-4-4 所示的处理模块。

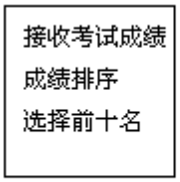


图 3-4-4 过程内聚的例子

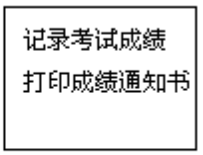


图 3-4-5 通信内聚的例子

（5）通信内聚

若一个模块中的各处理元素需引用共同的数据（同一数据项、数据区或文件），则称其元素间的联系为通信内聚。通信内聚的各部分间是借助共同使用的数据联系在一起的，故有较好的可理解性。如图 3—4—5 所示。通信内聚和过程内聚都属中内聚度型模块。

（6）顺序内聚

若一个模块内的各处理元素关系密切，必须按规定的处理次序顺序执行，这样的模块为顺序内聚类型。顺序内聚的模块内，后执行的语句或语句段往往依赖先执行的语句或语句段，以先执行的部分为条件。由于模块内各处理元素间存在着这种逻辑联系，所以顺序内聚模块的可理解性很强，属高内聚度类型模块。如图 3-4-6 所示的例子。

图 3-4-6 顺序内聚模块的例子

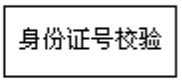


图 3-3-7 功能内聚模块的例子

（7）功能内聚

功能内聚是内聚度最高的一种模块类型。如果模块仅完成一个单一的功能，且该模块的所有部分是实现这一功能所必须的，没有多余的语句，则该模块为功能内聚。功能内聚模块的结构紧凑、界面清晰，易于理解和维护，因而可靠性强；又由于其功能单一，故复用率高。所以它是模块划分时应注意追求的一种模块类型。如图 3—4—7 是模块划分时得到的功能内聚模块。

在模块设计时应力争做到高内聚，并且能够辨别出低内聚的模块，加以修改使之提高内聚度并降低模块间的耦合度。具体设计时，应注意：

①设计功能独立单一的模块；

②控制使用全局数据；

③模块间尽量传递数据型信息。

第五节　系统设计的方法

一、面向数据流的设计方法

数据流图（DFD）是结构化分析的主要方法。作为需求分析的结果，数据流图描述了系统的逻辑结构，而功能模块图是概要设计中重要的设计和表示方法。由于任何可以用计算机处理的流程都可用数据流图来分析和描述，所

以确立一种方法、原则和步骤，将数据流图映射为功能模块图就显得十分重要。

面向数据流的设计方法就是以数据流图为基础，通过一系列系统的步骤，将数据流图转换为功能模块图，从而导出软件的结构的方法。面向数据流的设计方法是需求分析阶段结构化分析方法的延续，是结构化设计的主要方法。

1.数据流的两种类型——事务流和变换流

（1）变换流

任何以数据流图表示的软件系统，从总体上看，都包括三个功能部分，即接收数据、加工处理和输出数据。加工处理部分利用外部的输入数据，完成本身的逻辑功能，并产生新的数据作为输出。抽象地看，加工处理部分可以被看作是一个将输入数据变换为输出数据的变换机构，我们把有以上过程的数据流称为变换流。变换流的一般形式可用图 3-5-1 来表示。

在图 3-5-1 所示的变换流中，引入了几个新的概念。它们是：

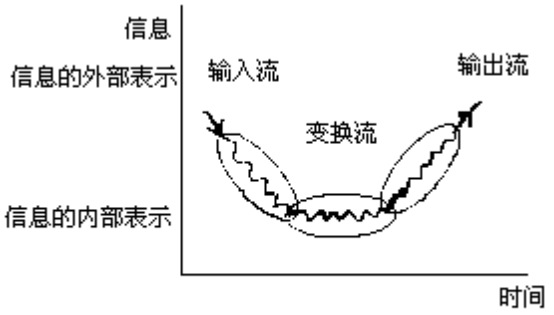


图3-5-1 变换流

1）输入流

输入流由一个或多个数据加工组成。其作用是将最初接收到的系统外部输入的数据，由其外部形式变成内部形式，即将系统得到的物理输入变为系统可用的形式。一般来说，输入流的处理工作是对数据格式进行转换，即对数据进行分类、排序、编辑、整理、有较性检验等。

2）变换流

此处的变换流是指将输入流转换为输出流的数据变换过程和机制。变换流接收的数据是系统可处理的，处理后以系统的内部形式送给输出流。

3）输出流

输出流将变换流发来的内部形式的数据经过加工处理变为外部系统可接收的形式并输出。

请看图 3-5-2 所表示的身份证号查询的数据流过程。

其中，虚线内部分为变换流，虚线外的两部分为输入流和输出流。

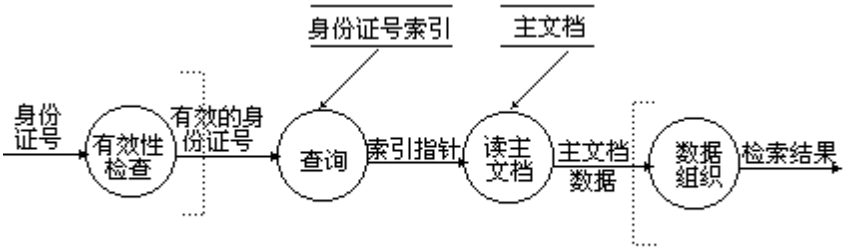


图3-5-2 身份证号查询

（2）事务流

一般来说，所有数据流均可看作是变换流。但是，有一类数据流本身有较明显的特点，可以将它区分出来作单独处理。若在一个数据流中，存在一个加工只接收一个输入数据，然后根据这个输入数据从若干个处理序列中选择一个路径执行，则具有这种类型的数据流叫做事务流（TransationFlow）。如图 3-5-3 所示。

在这里，称输入数据为事务，称根据事务作出判断，并选择多个处理路径中的一条来执行的加工为事务中心，事务中心的作用是：

1）接收输入数据（事务）；

2）根据事务作出判断，并选择处理路径；

3）沿处理路径执行。

2.由变换型数据流图导出模块结构图（变换分析）

对于变换型数据流图，可以根据一定的规则将它直接映射为功能模块图。规则具体步骤为：

（1）确定变换流、输入流和输出流部分

一般说，只要对系统流程比较熟悉，找出变换流和输入流、输出流的边界不是很难的。几个数据流汇集的地方，常常是加工的开始。如果一时找不出，可以用下述方法先区分出输入流部分和输出流部分，这样，变换流也就自然明确了。

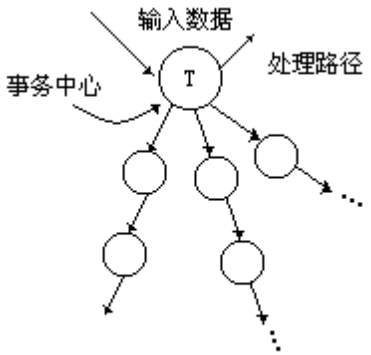


图3-5-3 事务流

从最外层的流入（物理的）出发，逐步向里，直到一个加工的流入数据流不能看作输入，则它以前的数据流就是输入流。

同样，从最外层的流出（物理的）逐步向里，直到一个加工的流出数据流不能看作输出，则它以后的数据流就是输出流。输入流和输出流之间，就是变换流。

也有这样的情况，即输入流和输出流是连在一起的，物理输入的结束就是物理输出的开始，则这样的系统就没有变换流。

（2）设计模块结构的顶层和

第一层

变换流部分即系统结构的顶层所在，同时它也对应一个

第一层模块。而输入流的每一输入数据、变换流给输出流的每一输出数据都分别对应一个

第一层模块。

顶层模块表示整个系统要完成的功能，常常称其为总控模块。对于没有变换流的结构，可以没有变换模块，也可以把输出中重要的加工提升为变换模块，这要根据具体情况而定。在映射过程中要注意，模块之间的数据交换

应当和数据流图中的数据流一致。

（3）设计中下各层

一般说，输入流中的每个加工可以对应成两个模块，即接受输入数据模块和将输入数据变换成其调用模块所需数据模块，然后再如此逐层细分。每个输出部分也可以按输入部分作相似的处理。这两部分的转换可以自上而下递归对应，直到物理输入的数据源和物理输出的数据池为止。

对于变换流中的每一个加工，可依次对一个模块，流入加工的数据映射为模块的输入参数，流出加工的数据流映射为输出参数。具体请看图 3-5-4 所示的过程。

这样得出的模块结构图是和数据流图严格对应的初始结构，一般不是最优的。需要对初始模块结构图进一步修改，才能得到较理想的结果。

3.由事务型数据流图导出模块结构图（事务分析）

对于事务型数据流图，通过事务分析，可以导出它所对应的标准形式的模块结构图。事务分析也是采用“ 自顶向下，逐步求精 ”的分析方法。具体步骤为：

①根据事务功能设计一个顶层总控模块；

②将事务中心的输入数据流对应为一个

第一层的接收模块及该模块的下层模块，具体可用变换分析方法；

③将事务中心对应为一个

第一层的调度模块；

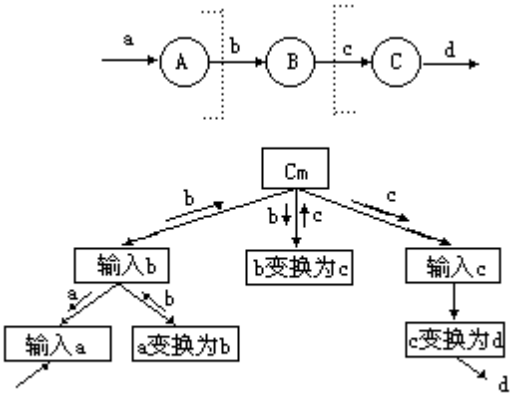


图3-5-4 变换分析

④对每一种类型的事务处理，在调度模块下设计一个事务处理模块；然后为每个事务处理模块设计下面的操作模块及操作模块的细节模块，每一处理的对应设计可用变换分析方法。具体见图 3-5-5 所示对应关系。

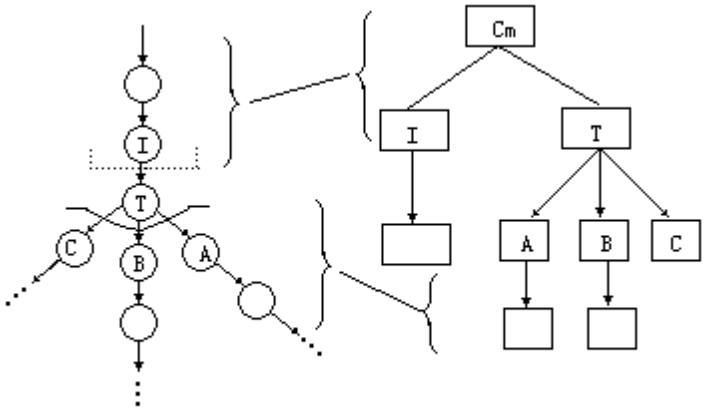


图3-3-5

第四章　系统实现与测试

第一节　系统实现

一、系统实现任务和步骤

1.系统实现的任务

系统分析和系统设计工作完成之后，信息系统的功能、模块结构、数据组成和数据结构已基本确定，接下来的工作就是系统实现。系统实现阶段的任务是将详细设计的结果转化为用具体的程序设计语言所书写的程序，即编写程序代码。同时，系统实现还要对所编写的源程序进行程序单元测试，并验证各程序模块单元间的接口。

因此，系统实现阶段有以下三方面的工作：

①以模块为程序单元编写代码；

②测试每一个程序单元；

③测试各个程序单元间的功能与数据是否协调一致。

如果经过单元测试发现所编写的程序存在错误，则需要修改源程序，然后再测试，直到没有错误为止。

2.系统实现的步骤

系统实现有以下 6 个步骤，它们是：

①对每个程序模块用所选定的程序设计语言进行编码；

②按照测试方案产生测试数据；

③按照测试方案中规定的方法进行程序单元测试；

④书写“ 模块开发卷宗 ”中相应于该阶段的内容；

⑤编写操作手册和用户手册；

⑥评审。

二、系统实现应注意的问题

由于信息系统开发中的困难主要集中在系统分析和系统设计阶段的工作，因而分析和设计工作是整个开发工作的基础。编码是将详细设计中得出的算法转换成用程序设计语言编写出来的程序，相对而言，它较前几个阶段的工作要简单。但是，编码工作也是相当重要的。由于系统的质量最终是通过程序来体现的，如果编码工作未能实现系统分析和系统设计的要求，则整

个系统开发工作就无法完成。同时，编码也有自身的规律和技巧。以下是编码时应注意的一些问题。

1.编码质量要求

大规模复杂信息系统的编码工作要求众多程序员之间的密切合作，因此，编码不仅要考虑程序的正确性，更要考虑程序的可理解性、可测试性和可维护性，因而，编码应注意以下几方面的要求：

①可靠性

可靠性的含义是：在正常情况下，系统能够正确工作；而在意外情况下，系统能做出适当处理，不会造成严重损坏。所谓意外情况，一般指硬件故障、输入输出错误等。

②可读性

可读性即程序的可理解性。逻辑结构清晰、变量命名合理、书写编排规范的程序可读性强，反之，可读性差。

③可测试性

可测试性是指所编程序易于用现有方法验证及发现其中的错误。

④可维护性

可维护性是指易于对所编的程序进行修改和扩充。

要使编码质量达到这些要求，需要注意解决几个问题，其中最主要的是程序的结构性和编写程序的风格。

2.程序的风格

程序的风格即所书写的程序特征。编程人员往往具有各自不同的风格。具有良好风格的程序易于理解、测试和维护。以下是一些与程序风格有关的设计规则。

- 不要为了“效率”而丧失清晰性
- 重复使用的表达式，要用调用公共函数去代替
- 使用括号以避免二义性
- 选用不会混淆的变量名
- 使用有意义的变量名和语句标号
- 用缩排格式限定语句群的边界
- 和谐的格式有助于读者理解程序
- 缩排书写要显示程序的逻辑结构
- 让程序按自顶向下方式阅读
- 避免不必要的转移
- 采用三种基本控制结构
- 首先使用易理解的伪编码语言编写，然后再翻译成你所使用的语言
- 把与判定相联系的动作尽可能近地紧跟着判定
- 对递归定义的数据结构使用递归过程
- 测试输入的合法性和合理性
- 结束输入要用文件结束标记，而不要用计数
- 采用统一的输入格式
- 把输入和输出局限在子程序中
- 确信在使用之前，所有的变量都已被赋初值
- 在出现故障时不要停机
- 注意因错误引起的中断

·避免从循环引出多个出口
·将数据编制成文件
·选用能使程序更简单的数据表示法
·确信注释与代码一致

第二节　系统测试

一、系统测试的基本概念

1.测试的目的

测试是寻找程序中错误的过程。对于系统测试，很多人往往存在着误解，认为测试的目的是证明程序的正确性。事实上，企图通过测试断定程序的正确性几乎是不可能的。对测试目的之理解直接影响测试的过程。对于想通过测试证明程序正确的开发人员来说，常会不自觉地使用那些能证明程序正确的测试数据，结果隐蔽了程序中的错误。正确的测试目的是尽可能多地发现系统存在的问题，进而加以改进。

2.测试工具

由于测试工作量大，且存在很多简单、重复性的工作，所以应尽可能利用计算机来完成其中机械性的工作。近年来已有一些测试工具问世。这些测试工具可在测试中协助人的工作。

①静态分析工具

静态分析是在不执行被测程序的情况下对程序进行的测试。静态分析工具和作用是扫描被测试程序的正文，检查可能导致错误的异常情况，如是否使用了未赋值的变量、是否有从未使用过的变量、实在参数和形式参数类型或个数是否不符、是否有程序从未执行等。

②文件比较程序

文件比较程序通过分析测试结果来测试程序。例如可将预期输出结果编成一个文件，测试后的实际输出结果也编成一个文件，然后通过文件比较程序比较两个结果文件，检索有无差异。由于很多测试输出的数据量都很大，有的还很复杂，所以使用文件比较程序可大大减少测试人员的工作负担。

③覆盖监视工具

覆盖监视工具又叫做动态分析程序。动态分析是通过运行被测程序进行测试的方法。监视工具被安插在程序的适当位置，以便对被测程序进行监视，它们还可以产生带有统计数字的报告。监视工具还可用在编码过程中，开发人员可以在程序中加入监控语句，并通过输出的监控结果来调试程序。

④驱动工具等

采用自底向上的测试方法时，由于被测模块的上层模块尚未经测试，所以需要编写模拟调用本模块的上层模块，驱动工具可协助测试人员完成有关的工作。

3.测试用例

测试用例是测试时所选用的数据。一般认为，测试是通过输入一定的测试数据运行被测程序来完成的。因而，测试的关键是选择好测试用例。但是，需要明确的是，要穷举所有的测试数据来测试程序常常是不可能的，因而需要选择一些被认为是好的测试用例。好的测试用例可以理解为：在一定的时间和经费的限制下，能尽可能多地发现错误的测试用例。事实上，穷举所有

测试用例是完全不必要的，反复地使用那些能证明程序正确执行的测试用例也是绝对不必要的。

测试用例应该包括两部分：输入数据和预期输出结果。输入数据即前面说的测试数据，预期输出结果即根据输入数据推断出来的可能输出，把它也作为测试用例的一部分是为了在执行测试之后，和实际的输出结果相比较。

二、测试的过程

系统测试可分为三个过程，即：单元测试、组装测试和确认测试。

1.单元测试

单元测试是对基本的程序模块所做的测试，其具体内容包括：

①模块间的接口；

②模块内局部数据结构；

③程序的执行路径，包括计算和运行控制；

④程序的错误处理能力。

2.组装测试

（1）组装测试的内容

组装测试是根据概要设计中各功能模块的说明及制定的组装测试计划，将经过单元测试的模块逐步组装并进行测试。

组装测试的内容有：

①测试模块间的连接；

②测试系统或子系统的输入/输出处理，使其达到设计要求；

③测试系统或子系统正确处理能力和经受错误的能力。

具体地说，组装测试主要作以下检查：通过模块间接口的数据是否丢失；一个模块的运行会不会破坏别的模块的功能；一些模块组合之后所形成的总功能是否保持系统设计的要求；全程数据在程序运行中能否保持正常；几个模块运行后的误差积累是否超过规定范围；单元测试中尚未查出的错误等。

（2）组装测试的方式

根据组装测试中各个模块联结方式，可将组装测试分为非渐增式测试和渐增式测试两种方式。

①非渐增式

非渐增式先对系统的单个模块进行个别测试，然后再将所有模块组合在一起测试。

②渐增式

渐增式也是先对系统的单个模块进行个别测试，但在模块组合时，不是一次组合所有的模块，而是先组合测试少数几个模块，然后逐渐加多，直到所有模块都组合在一起为止。

渐增式的组合方向有自顶向下联结和自底向上联结两种。自顶向下联结从模块结构图的顶端开始，逐层向下联结。自底向上与此刚好相反。

3.确认测试

确认测试又叫验收测试，其任务是根据软件需求说明书中定义的全部功能和性能要求，以及确认测试计划测试整个系统是否达到了要求。并提交最终的用户手册和操作手册。

确认测试包括以下内容：

①在模拟的环境中进行强度测试，即在事先规定的一个时期内运行软件的所有功能，以证明该软件无严重错误。

②执行测试计划中提出的所有确认测试。

③使用用户手册和操作手册，以进一步证实其实用性和有效性，并改正其中的错误。

④分析测试结果，找出产生错误的原因。

⑤书写确认测试分析报告。

⑥确认测试结束后，书写整个项目的开发总结报告。

三、测试方法

目前最基本的测试方法有两种，它们是白盒法和黑盒法。

白盒法又叫逻辑覆盖法。之所以称为白盒法，是指测试时需深入到程序内部。对测试人员而言，整个程序就像一个敞开的盒子，因而使用这种方法的基础是对程序内部的逻辑结构有清楚的了解，即测试时需要有被测程序的源代码。白盒法要求测试用例要尽可能复盖程序模块内部的所有逻辑路径。黑盒法和白盒法的测试依据正相反，黑盒法不需要测试人员了解被测程序内部的逻辑结构，程序内部的源代码就像被一个黑盒法隐藏起为，所以称为黑盒法。在黑盒法中，测试人员不是根据程序内部的逻辑结构而是根据程序的功能来设计测试用例。

白盒法和黑盒法各自有其优点和不足，也各有其应用的条件和限制，它们分别适宜于不同情况下的测试。

1.白盒法

（1）白盒法的基本概念

前边说过，白盒法又称逻辑覆盖法。所谓逻辑覆盖是指使用一定的测试数据所能执行的语句范围，这些被执行的语句所组成的路径，称为逻辑路径。如果整个程序是顺序结构，从头到尾只有一条路径，则执行程序必然会覆盖所有路径，测试起来就很简单。而在具有选择结构或循环结构的程序中，由于有了两个或两个以上的分支，程序的一次执行肯定不会覆盖所有的分支，从而产生了多种逻辑覆盖类型。

例 4.2.1 设有如下一程序段。

```
if（A＞1andB＝＝0）
        X＝X/A；
if（A==2orX＞1）
        X=X＋1；
```

该程序段的流程图如图 4-2-1 所示，它有两个复合判定，两条可执行语句。因而逻辑路径则有：

ace，两条语句均被执行；

abd，两条语句均不执行；

acd，只执行 X=X/A；

abe，只执行 X=X＋1。

这样，一共有 4 条逻辑路径。

下面根据对这个程序段的分析来说明各种逻辑覆盖类型。

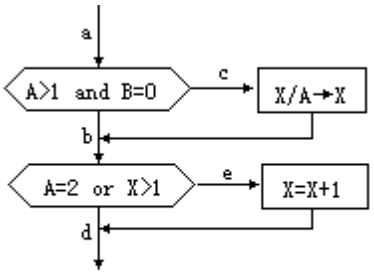


图4-2-1

（2）逻辑覆盖类型

1）语句覆盖

语句覆盖是指选用的测试用例将执行程序中的每个语句。

对于所举的例子，语句覆盖应执行 c 和 e 两个语句。要做到这一点，当然可以有多个测试用例，有的执行 c，有的执行 e，但理想的测试用例应当是最少的，即最好用一个执行 ace 这一逻辑路径的测试用例。选用如下测试用例就能满足语句覆盖的要求。

A=2

B=0

X=3

可以看出，语句覆盖的测试效果是相当弱的。因为就这个程序段讲，一共有 4 条逻辑路径，要对它进行比较全面的测试，则每条路径至少要执行一遍，而语句覆盖只执行了 1 条。

2）判定覆盖

判定覆盖是指执行测试用例时，要使程序中每个判定都能够获得一次真值和伪值。即使每个分支路径都被执行一次。

根据这一要求，比较理想的情况是使用如下测试用例：

A=3

B=0

X=1

则程序运行时

第一个判定为真，能够执行 X/A，但由于执行后 X=1/3，

第二个判定为伪，所覆盖的是 acd 这一逻辑路径。

再用如下测试用例：

A=2

B=1

X=3

则程序运行时

第一个判定为伪，而

第二个判定为真，所覆盖的是 abe 逻辑路径。

由于使用这两个测试用例后，每个判定都取过真和伪，已经满足判定覆盖标准，因而不必再选其它测试用例。

然而这样执行的结果只经过 acd 和 abe 两条逻辑路径，还有其它两条没有被执行。所以判定覆盖尽管比语句覆盖的测试效果要强些，但距完整、全面的测试仍有一定的差距。

3）条件覆盖

条件覆盖是指执行测试用例时，要使得程序判定中的每个条件都能获得一次真值和伪值。

判定由条件组成，只包含一个条件的判定称为简单条件判定。对于简单条件判定，条件覆盖和判定覆盖是一致的；包含两个或两个以上条件的判定称为复合条件判定。对于复合条件判定，由于一个判定由多个条件组成，所以两种覆盖的程序不同。一般讲，条件覆盖的测试效果比判定覆盖强。

在上例中，一共有四个条件，即：

A＞1

B=0

A=2

X＞1

为使每一个条件在测试中都要取一次真和一次伪。可选用如下两个测试用例：

测试用例 1	测试用例 2
A=2	A=1
B=0	B=1
X=4	X=1

4）判定/条件覆盖

判定/条件覆盖是覆盖范围更全面的一种覆盖类型。在判定/条件覆盖中，通过执行足够的测试用例，使每个判定都能取各种可能的值，同时使每个条件也都能取各种可能的值。

对于上面的程序段，条件覆盖所选的两个测试用例就可以满足这种覆盖标准。

5）条件组合覆盖

条件组合覆盖是通过执行足够的测试用例，使得每个判定中条件的各种可能组合至少出现一次。由于考虑到判定中各种条件的可能组合，满足这种覆盖标准的测试用例一定能满足前述的其它覆盖类型。

对于上例中的程序段，四个条件共有八种组合，即：

A＞1，B=0

A＞1，B≠0

A＜1，B=0

A＜1，B≠0

A=2，X＜1

A=2，X＜1

A≠2，X＞1

A≠2，X＜1

用四个测试用例就可以实现上述八种组合，这些测试用例是：

A=2，B=0，X=4

A=2，B=1，X=1

A=1，B=0，X=2

A=1，B=1，X=1

条件组合覆盖的覆盖效果应当是最强的，但它也不一定能覆盖程序所有的逻辑路径。如本例中，acd 就没有执行。

2.黑盒法

和白盒法不同，黑盒法的测试用例很大程度上是根据经验总结出来的。在选择测试用例时，一般有等价分类法、边缘值分析法和错误推断法等几种方法。

（1）等价分类法

黑盒法是根据程序的功能来选择测试用例的。彻底的黑盒法应当用所有可能的输入数据来进行测试，但这常常是不可能的。因此测试时只能挑选适当的、有代表性的测试用例，即组织一个数量有限、但能尽可能多地代表其它各种未被选为测试用例的测试数据集合。这就是等价分类法的基本思想。等价分类法把输入数据的所有可能值分成若干个“等价类”，对测试而言，每个等价类中的数据作用结果是相同的。因而，测试时从中选择一个数据即可。等价分类法的关键是划分等价类。

划分等价类和程序功能有关。一般说来，程序功能所要求的输入决定了等价类，因此应当从程序的功能找出输入条件，再把所有可能的输入条件划分成若干等价类，对于每一个等价类则要举出合理的和不合理的两种例子。

划分的方法是带有试探性的，目前还没有很明确的规则，只有些可供参考的做法。

1）如果输入条件说明了输入数据的取值的范围，则可以把规定的取值范围划分为一个合理的等价类，而把超出此范围的划分为两个不合理的等价类。例如，如果输入值在 1～100 之间则合理等价类为“1≤X≤100”；而两个不合理的等价类为“Y＜1”和“Y＞100”。

2）如果输入条件说明了输入数据的个数，则可以把规定个数作为一个合理的等价类，而把在此之外的个数划分为两个不合理的等价类。例如，如果每个学生可以借 5 本书，则一个合理的等价类是 1～5；两个不合理的等价类分别是未借书和借书超过 5 本。

3）如果输入条件说明了一个“必须成立”的情况，则此情况可以作为一个合理的等价类，而与此相反的情况则是一个不合理等价类。例如，如果某数据要求

第一字节规定为“1F”，则“1F”为合理等价类，而以外字符均属于不合理等价类。

4）如果输入条件说明了输入数据的一组可能的值，而且程序是用不同方式来处理每一种值的，则可以划分合理的和不合理的等价类各一个。如读者的借书量是按工人、学生、研究生、教师和馆员来确定的，则合理的输入等价类是工人、学生、研究生、教师和馆员的集合，而不合理的等价类是这五种以外的其他人。

5）如果认为程序将按不同的方式来处理等价类中的各种例子，则应将这个等价类再细分成几个更小的等价类。

（2）边缘值分析法

边缘值分析法是等价分类法的一种特殊情况。它的“特殊”之处在于：

1）它对同一等价类的所有例子不是等同看待，而是把目标放在反映该等价类的边缘情况的例子上，因而它不是从同一等价类中随意挑选测试用例，而是着重从边界情况来分析可能挑选的测试用例。

2）等价分类法仅仅从输入条件进行分类和挑选测试用例，而边缘值分析法除此之外还要根据输出的情况来设计测试用例。

应用边缘值分析法需要一定的经验和技巧，还要有一定的创造性。这里

只能列举一些例子供参考。

1）如果输入条件说明了取值范围，则可以在取值的边界附近来选取合理的和不合理的测试用例，并使不合理到合理正好越过边界。

如输入范围为；-1.0～1.0，则取-1.001，-1.0，1.0，1.001 四个测试用例，

第 1 和

第 4 是不合理的，

第 2 和

第 3 是合理的，由 1 到 2 和由 3 到 4 正好两次跨越输入范围的边界。

2）如果输入条件指出输入数据的个数，可以取其范围中的最小个数、最大个数，比最小个数少 1、比最大个数多 1 作为边缘值。

如某个数据文件可以 1～65535 个记录，则可取 0 个，1 个，65535 个，65536 个作为边缘值。

3）把 1）用于输出条件。如某个程序的功能是计算商品销售折扣量，最低折扣量为 0 元,最高可到 1050 元,则可设计一些有关商品价格的测试用例，使输出分别为 0 和 1050，还可以考虑是否能设计输出为负值或大于 1050 的例子。

4）把 2）用于输出条件。如某情报检索系统对于命中文献，最多只能打印 100 篇，就可以设计打印出 0 篇，1 篇，100 篇，101 篇作为边缘值。

5）如果程序的输入或输出是有序集合，如顺序文件、线性表等，则应特别注意集合的

第一个和最后一个元素。

3.错误推测法

错误推测法是指测试人员凭借经验来推测程序中可能存在的错误，从而有针对性地编写检查这些错误的测试用例。

错误推测法没有确定的步骤，很大程度上凭借经验，如输入数据为 0 或输出数据为 0 易发生错误；又如，输入表格为空或只有一行也容易发生错误，因而可以选择这些情况作为测试用例。

4.综合策略

由于程序的情况多种多样，因而具体测试时，需要综合运用各种测试方法，即把选择测试用例的各种方法结合起来，形成一些综合性的策略。具体应用要根据情况灵活处理，比如，不了解程序内部的具体逻辑处理流程是无法使用白盒法的。

根据各种方法的特点，综合策略有如下几种：

①任何情况下，都需要用边缘值分析法；

②必要时再用等价分类法补充测试用例；

③再用错误推测法；

④如果能使用白盒法，则可检查上述例子的逻辑覆盖程度，如发现不够，可再增加测试用例以使逻辑覆盖尽可能全面。

总之，系统测试有多种方法，实际测试时应根据具体情况，选择适当的方法。