第 1 章　软件设计基础

软件是计算机程序、规程以及运行程序所需的相关文档和数据。例如，开发一个学生信息管理系统，就要对学生的基本信息、班级信息、课程信息、成绩信息等进行管理，通过该系统，可以做到信息的规范管理、科学的统计和快速的查询，从而减少管理方面的工作量，提高信息管理工作的效率。开发这样一个系统，就像盖一座大楼，是一项工程，这就是软件工程技术研究的内容，几万个学生的各种信息要存储在计算机中，这些信息不是杂乱无章的，而是按照一定的原则来组织和存储的，那么就需要数据库技术和软件设计。计算机软件作为一种逻辑系统，它和计算机硬件有着显著的差别，它主要是对软件进行定义、开发和维护。

本章主要介绍软件的基本概念、软件开发的过程、软件的详细设计，而软件工程是计算机专业的一门专业课程，要提高软件的开发能力还需要进一步学习相关课程，这里仅仅简单介绍软件设计的一些基本概念。

1.1　软件概述

软件的规模大小、复杂程度决定了软件开发的难度。对一个软件而言，它的程序复杂性将随着程序规模的增加而呈指数级上升趋势。因此，必须采用科学的软件开发方法，采用抽象、分解等科学方法降低复杂度，以工程的方法管理和控制软件开发的各个阶段，以保证大型软件系统的开发具有正确性、易维护性、可读性和可重用性。

1.1.1　软件定义

软件是程序、数据及相关文档的集合。其中，程序是软件开发人员根据用户需求开发的、用程序设计语言描述的、适合计算机执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发、维护及使用密切相关的图文资料的总体。

1.1.2　软件发展

软件的发展大致可划分为以下四个阶段。

1. 程序设计阶段（1950 ~ 1965 年）

这个阶段是计算机软件发展的早期阶段，软件开发采用低级语言，开发效率比较低，应用领域基本局限于科学和工程的数值计算，人们不重视软件文档的编制，只注重代码的编写。计算机编程很简单，没有系统化的方法，在硬件的生产已经开始趋于标准化的时候，软件的生产仍是个体化，软件产品处在初级阶段，大多数软件是由使用者自己开发的。

2. 程序系统阶段（1965 ~ 1974 年）

这个阶段是计算机软件发展的第二阶段，相继诞生了大量的高级程序设计语言，程序

开发的效率明显提高，实时系统和第一代数据库管理系统也相继出现，并产生了成熟的操作系统和数据库管理系统。到后期，由于软件规模不断扩大，复杂程度大幅度提高，许多程序的个人化特性使得它们根本不能维护，在软件维护上所花费的精力和消耗资源的速度是惊人的，这就产生了“软件危机”，同时出现了有针对性地进行软件开发方法的理论研究和实践。

3. 软件工程阶段——结构化方法（1974～1989 年）

这个阶段是计算机软件发展的第三阶段，在这一阶段，软件规模越来越大，结构越来越复杂，软件开发管理既困难又复杂，软件开发费用不断增加。软件的开发技术落后，生产方式落后，仍采用手工方式，开发工具也落后，生产效率低，这就是软件危机产生的原因。

在这一阶段，以软件的产品化、系列化、工程化、标准化为特征的软件产业发展起来，消除了软件生产的个体化特征，有了可以遵循的软件工程化的设计原则、方法和标准，并且结构化方法也得到发展。

4. 软件工程阶段——面向对象方法（1989～至今）

这个阶段是计算机软件发展的第四阶段，在这一阶段，已经不再着重于单台计算机和计算机程序，而是面向计算机和软件的综合影响。由复杂的操作系统控制的强大的桌面机、广域网络和局域网络，配以先进的软件应用已成为标准。计算机体系结构迅速地从集中的主机环境转变为分布的客户机/服务器环境。世界范围的信息网提供了一个基本结构，信息高速公路和网际空间连通已成为令人关注的热点问题。事实上，Internet 可以看做能够被单个用户访问的软件。

计算机发展正朝着社会信息化和软件产业化方向发展，由于软件编程方法及软件设计思想不断更新，导致软件工程进入了面向对象方法的时代，出现了占据主导地位的面向对象技术，它将在许多领域中迅速取代传统的软件开发方法。同时，软件开发技术继续发展，并逐步转向智能化、自动化、集成化、并行化和工程化。另外，计算机网络技术、分布式技术对软件工程的发展也起到了促进作用，使得当前采用面向对象技术开发的软件系统越来越多。

1.1.3 软件特点

1. 工具性特征

计算机软件包括程序和文档两个部分。计算机程序包括源程序和目标程序，源程序是用计算机高级语言（如 BASIC、Algol、C、C++等）编写的程序，表现为数字、文字和符号的组合，构成符号化指令序列或符号化语句序列；目标程序是用机器语言编写的，体现为电脉冲序列的一串二进制数(0 和 1)指令编码，直接用于驱动计算机硬件工作，保证计算机系统发挥各项功能，获得一定结果，因而又具有工具性特征。软件在调入计算机运行之前，首先表现为作品性，人们无法通过“阅读”或“欣赏”计算机程序与文档而制造任何有形产品和实现任何操作。但当软件调入计算机运行时，则更主要地表现为工具性，即通过控制计算机硬件动作过程，获得某种结果。

2. 软件开发工作量大、成本高，但复制容易且成本低

软件开发必须经过功能限定、逻辑设计和编码三个步骤，要求软件开发人员必须具有丰富而超前的专业和相关知识，极强的逻辑和形象思维能力，了解计算机硬件的最新发展状况与发展前景，熟练掌握和使用编程语言。开发具有实用商业价值的计算机软件，通常需要按照专业化分工、流水线作业的方式由一批人共同完成。可见，开发计算机软件必须具备相应的物质和技术条件，有充足的开发资金和良好的开发环境。复制是对计算机软件的客观再现，不改变软件内容和本身的价值，复制后的软件以一定的客观物质形式体现，具有可感知性。计算机软件的可复制性决定了其可以广泛传播和有效利用，创造经济和社会效益。

3. 软件具有无形性，可以多次使用

计算机软件是智力劳动产生的精神产品，如计算机程序、说明程序的文档等都是智力劳动的直接产物，不具有任何形状，人们只有借助于一定的物质载体和工具才能感知其存在。计算机软件在同一时间可以为若干人分别使用，软件只要不受操作失误、计算机病毒等影响，就可以无限制反复使用。但是，计算机软件又具有工具性，主要通过“使用”而发挥其功用，因而应该具有使用寿命，使用寿命在流通领域表现为商业寿命。在科学技术飞速发展、新软件层出不穷的今天，计算机软件的商业寿命正在日益缩短。

与硬件相比，软件的特点如下。

① 表现形式不同：软件是逻辑部件，具有很高抽象性，缺乏可见性；硬件是物理部件，看得见、摸得着。

② 生产方式不同：软件的开发，是人的智力的高度发挥，不是传统意义上的硬件制造，软件的成本主要在开发和研制。开发和研制后，通过复制可大量生产。

③ 要求不同：硬件产品允许有误差，而软件产品不允许有误差。

④ 维护不同：由于磨损和老化，硬件会用旧用坏，解决的办法是更换一个相同的备件。而在理论上，软件不会用旧用坏，但软件是有生命周期的，存在退化现象，软件维护要比硬件复杂得多。

1.1.4 软件危机

20 世纪 60 年代初，美国的专业软件公司只有十几家，1968 年发展到 1300 多家。这些公司研制、生产和销售各种应用软件。尽管软件业发展迅速，但随着计算机应用范围的扩大，人们对软件的需求越来越大，软件的规模也越来越大，结构越来越复杂。例如，IBM 公司 60 年代研制的 IBM 360 操作系统，参与开发的单位美国有 11 个、欧洲有 6 个，参与开发的软件工作人员有 700 余人，其他辅助人员 1000 多人，从 1963 到 1966 年历时 4 年，花费 5 亿美元开发的系统仍包含了大量的错误，以后通过不断地被修改、补充，但每个版本还是有上千种错误。因此，在 20 世纪 60 年代硬件迅速发展的同时，软件的发展遇到越来越大的困难，人们称这一现象为“软件危机”。软件危机主要表现在：对软件开发成本和进度的估计常常很不准确，经费预算经常突破，完成时间一再拖延；开发的软件不能满足用户要求，用户对软件不满意的现象经常发生；开发的软件可维护性差、可靠性差。产生

软件危机的原因主要：软件规模越来越大，结构越来越复杂；软件开发管理困难而复杂；软件开发费用不断增加；软件开发技术落后；生产方式落后，即采用手工方式；开发工具落后，生产效率低。核心原因是软件系统的复杂度远大于硬件，计算机硬件产品的制造已经标准化、工程化、产业化，但软件生产离此目标相距甚远。

从软件危机的种种表现和软件作为逻辑产品的特殊性可以总结出软件危机产生的原因。

1. 用户需求不明确

在软件开发过程中，用户需求不明确问题主要体现在以下四个方面。

① 在软件开发之前，用户也不清楚软件的具体需求。

② 用户对软件需求的描述不精确，可能有遗漏、有二义性，甚至有错误。

③ 在软件开发过程中，用户又提出修改软件功能、界面、支撑环境等方面的要求。

④ 软件开发人员对用户需求的理解与用户本来的愿望有差异。

2. 缺乏正确的理论指导

第二个原因是缺乏有力的方法学和工具方面的支持。由于软件不同于大多数其他工业产品，其开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员的智力投入。由于过分地依靠程序设计人员在软件开发过程中的技巧和创造性，加剧了软件产品的个性化，这也是发生软件危机的一个重要原因。

3. 软件规模越来越大

随着软件应用范围的增大，软件规模愈来愈大。大型软件项目需要组织一定的人力共同完成，而多数管理人员缺乏开发大型软件系统的经验，多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确，有时还会产生误解。软件项目开发人员不能有效地、独立自主地处理大型软件的全部关系和各个分支，因此容易产生疏漏和错误。

4. 软件复杂度越来越高

软件开发不仅在规模上快速发展扩大，其复杂性也在急剧增加。软件开发产品的特殊性和人类智力的局限性，导致人们无力处理“复杂问题”。所谓“复杂问题”是相对的，一旦人们采用先进的组织形式、开发方法和工具提高了软件开发效率和能力，新的、更大的、更复杂的问题又会出现。

针对上述情况，1968 年北大西洋公约组织在德国召集了 50 名一流的编程人员、计算机科学家和工业界巨头，制定摆脱软件危机的办法，首次提出了软件工程（Software Engineering）这一概念，倡导以工程的原理、原则和方法进行软件开发，以期解决“软件危机”问题。软件工程是当时的一门新兴的、指导计算机软件开发和维护的工程学科。被定义为“运用系统的、规范的和可定量的方法来开发、运行和维护软件。”

软件工程的主要研究对象是大型软件。软件工程研究的内容主要包括：软件质量保证和质量评价；软件研制和维护的方法、工具、文档；用户界面的设计以及软件管理等。软件工程的最终目的是摆脱手工生产软件的状况，逐步实现软件研制和维护的规范化。

1.1.5 软件生命周期

1. 软件工程过程

软件工程过程是把用户的需求转化为软件产品的一系列活动。为获得软件产品，在软件工具的支持下，由软件开发人员完成的一系列工程活动。每个软件开发机构都可以规定自己的软件工程过程，针对不同类型的软件产品，软件开发机构也可能使用多种软件工程过程。但不管哪一种情况，软件工程过程一般包括四种基本的过程活动。这些活动包括计划（Plan）、开发（Do）、确认（Check）、维护（Action）。实际上，软件工程过程是一个软件开发机构针对某一类软件产品为自己规定的工作步骤。

2. 软件生命周期

一个软件从提出开发、实现、使用、维护直到停止使用的过程称为软件生命周期。软件生命周期一般包括：可行性分析和项目开发计划、需求分析、概要设计、详细设计、编码实现、测试、交付使用及维护等具体环节。大体可分为三个时期：计划阶段（问题定义和可行性分析、需求分析），开发阶段（软件设计、编码、测试）和运行阶段（软件维护）。各阶段的任务及产生的相应文档如表 1-1 所示。

表 1-1 软件生命周期各阶段的任务

时　期	阶　段	任　务	文　档
软件计划	问题定义	理解用户要求，划清工作范围	计划任务书
	可行性分析	可行性方案及代价	
	需求分析	软件系统的目标及应完成的工作	需求规格说明书
软件开发	概要设计	系统的逻辑设计	软件概要设计说明书
	详细设计	系统模块设计	软件详细设计说明书
	软件编码	编写程序代码	程序、数据、详细注释
	软件测试	单元测试、综合测试	测试后的软件、测试大纲、测试方案与结果
软件运行	软件维护	运行和维护	维护后的软件

（1）软件计划

问题定义阶段即进行调研和分析，弄清用户想干什么，不想干什么，以确定工作范围。通过调查抽象出“用户想要解决的问题是什么”。

在上述工作的基础上进行可行性分析，本阶段的具体工作如下：分析所需研制的软件系统是否具备必要的资源，技术上、经济上的可能性和社会因素的影响，回答“用户要解决的问题能否解决”，即确定项目的可行性。

需求分析要解决“做什么的问题”。经过问题定义，可行性分析后，需求分析阶段要考虑所有的细节问题，以确定最终的目标系统做哪些工作，形成目标系统完整的准确要求。

该阶段最后提交说明系统目标及对系统要求的规格说明书。

（2）软件开发

软件开发包括概要设计、详细设计、软件编码和软件测试四个阶段。

概要设计又称为总体设计、逻辑设计。该阶段要回答“怎样实现目标系统”的问题。首先应考虑实现目标系统的可能方案，并选择一个最佳方案。确定方案后应完成系统的总体设计，即确定系统的模块结构，给出模块的相互调用关系，并产生概要设计说明书。

详细设计阶段回答“应该怎样具体实现目标系统”的问题。在概要设计的基础上，要给出模块的功能说明和实现细节，包括模块的数据结构和所需的算法，最后产生详细设计说明书。

详细设计完成后进入软件编码阶段，程序员根据系统的要求和开发环境，选用合适的高级程序设计语言或部分选用汇编程序设计语言编写程序代码。

软件测试分为单元测试和综合测试两个阶段。单元测试指对每一个编制好的模块进行测试，发现和排除程序中的错误。综合测试指通过各种类型的测试检查软件是否达到预期的要求。

（3）软件维护

软件维护阶段是长期的过程，是在软件投入使用以后的时期，因为发生经过测试的软件可能还有问题；或者用户的要求还会发生变化；或者软件运行的环境可能发生变化，在上述情况发生时，都要进行软件的维护。因此，交付使用的软件仍然需要继续排错、修改和扩充，这就是软件维护。

3. 软件生命周期模型

软件生命周期模型（Life-Cycle Mode）也称软件过程模型，是软件系统开发项目总貌的一种描述，着眼于对项目管理的控制和逐步逼近的策略。

（1）瀑布模型

瀑布模型（Waterfall Mode）是 1976 年由 B. W. Boehm 提出的传统的软件生命周期模型，如图 1-1 所示。

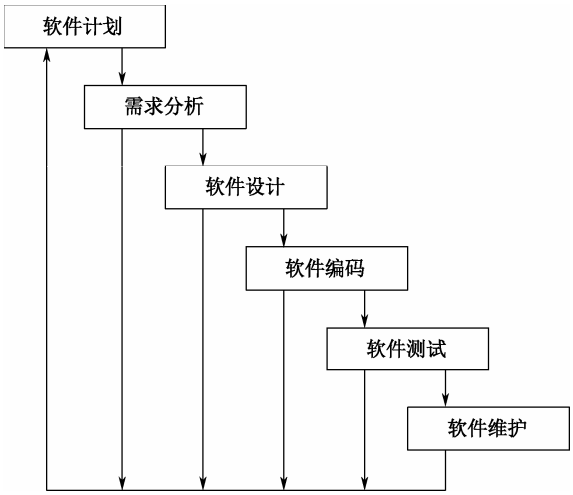

图 1-1　瀑布模型

由图可见，该模型将软件开发过程划分为六个阶段，这六个阶段是顺序进行的，前一阶段的工作完成后，下一阶段的工作才能开始；前一阶段产生的文档是下一阶段工作的依

据。该模型适合在软件需求比较明确，开发技术比较成熟的场合下使用，它是软件工程中应用最广泛的模型。

（2）快速原型法模型

快速原型法模型（Rapid Prototyping）是针对瀑布模型中的缺点提出的一种改进模型，如图 1-2 所示。

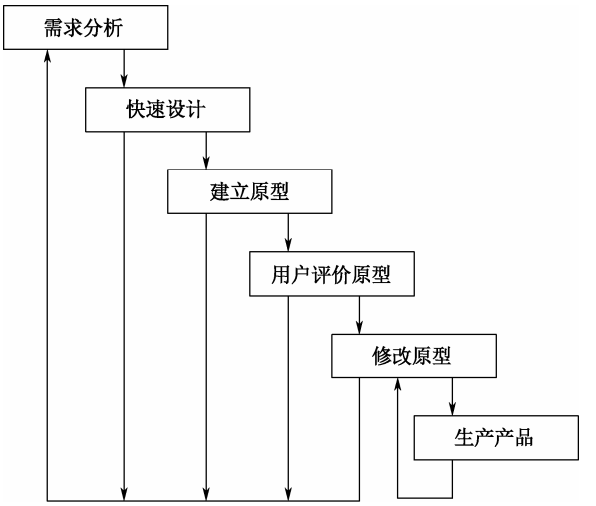

图 1-2　快速原型法模型

快速原型法也从了解需求开始，开发人员和用户一起来定义所有目标，确定哪些需求已经清楚，哪些还需要进一步定义；然后是快速设计，快速设计主要集中在用户能看得见的一些软件表示方面（如输入方法、输出形式等）。快速设计可产生一个原型（试验性产品）。用户有了原型，即可对其评价然后修改要求。重复上述各步骤，直到该原型能满足用户的需求为止。

软件生命周期模型还包括许多其他模型，如螺旋模型（The Spiral Model）、四代技术（Fourth-Generation Techniques，4GT）、面向对象生存期模型（Object-Oriented Life-Cycle Model）等。

近年来，面向对象（Object-Oriented，OO）方法日益受到人们的重视。面向对象方法遵循人类习惯的思维方式，开发出的软件（产品）的稳定性、可复用性和可维护性等都比传统的开发方法好。

目前，经过多年理论完善和实践，传统的瀑布模型已形成了一个较完整的体系，仍是软件开发中使用的最基本的理论基础和技术手段。

1.1.6　软件开发过程

软件开发过程是指软件工程人员为了获得软件产品，在软件工具支持下实施的一系列软件工程活动。科学、有效的软件开发过程应该定义一组适合所承担的项目特点的任务集合。

软件开发过程一般分为如下五个阶段。

1. 问题的定义及规划

此阶段是软件开发与需求者共同讨论的，主要确定软件的开发目标及其可行性。

2. 需求分析

在确定软件开发可行性的情况下，对软件需要实现的各个功能进行详细需求分析。需求分析是一个很重要的阶段，将这一阶段做好，会为整个软件项目的开发打下良好的基础。“唯一不变的是变化本身”，同样，软件需求也是在软件开发过程中不断变化和深入的，因此，必须定制需求变更计划来应付这种变化，以保护整个项目的正常进行。

3. 软件设计

此阶段中要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计、数据库设计等。软件设计一般分为总体设计和详细设计。好的软件设计将为软件程序的代码编写打下良好的基础。

4. 程序编码

此阶段是将软件设计的结果转化为计算机可运行的程序代码。在程序编码中必须制定统一、符合标准的编写规范，以保证程序的可读性、易维护性，提高程序的运行效率。

5. 软件测试

在软件设计完成之后要进行严密的测试，发现软件在整个软件设计过程中存在的问题并加以纠正。整个测试阶段分为单元测试、组装测试、系统测试三个阶段进行。

测试方法主要有白盒测试和黑盒测试。白盒测试时将程序看做一个透明的盒子，即测试人员完全了解程序的内部结构和处理过程。所以测试时按照程序内部的逻辑测试程序，检验程序中的每条通路是否都能按预定的要求正确工作。黑盒测试时完全不考虑程序内部的结构和处理过程，只按照规格说明书的规定来检查程序是否符合要求，黑盒测试只测试功能。

1.1.7 软件开发工具

软件开发工具是为支持软件人员开发和维护活动而使用的软件。它可以帮助开发人员完成一些烦琐的程序编制和调试问题，使软件开发人员将更多的精力和时间投到最重要的软件需求和设计上，提高软件开发的速度和质量。

软件开发工具主要是语言开发工具如，C++、Java、Delphi 开发工具等。软件开发工具的功能如下。

1）认识与描述客观系统。

2）存储及管理开发过程中的信息。

3）代码的编写与生成。

4）文档的编制或生成。

5）软件项目的管理。

软件开发工具的特性如下。

1）表达能力或描述能力。

2）保持信息一致性的能力。
3）使用的方便程度。
4）工具的可靠性。

1.2 软件的详细设计

1.2.1 基本概念

软件设计是一个把软件需求转化为软件表示的过程，即把分析结果加工为在程序细节上接近于源程序的软件表示（软件描述）。软件设计的目标是对将要实现的软件系统体系结构、系统的数据、系统模块间的接口，以及所采用的算法给出详尽的描述。

软件设计阶段通常分为两步：一是系统的总体设计或概要设计，它的任务是确定软件系统结构；二是系统的详细设计，即进行各模块内部的具体设计。软件设计方法有多种，如面向数据流分析（Data Flow Analysis, DFA）的设计，也称为结构化设计（Structured Design, SD），还有面向数据结构的设计，如 Jackson 系统开发（Jackson System Development，JSD）方法和逻辑构造程序（Logically Constructed Program）方法。本节重点采用结构化设计方法来介绍概要设计和详细设计。

结构化设计方法是计算学科的一种典型的系统开发方法，也是被广泛使用的一种传统的软件开发方法。它的基本思想如下：采用自顶向下的模块化设计方法，按照模块化原则和软件设计策略，将需求分析得到的数据流图，映射成由相对独立、单一功能的模块组成的软件结构。

用结构化方法开发软件的过程如下：从系统需求分析开始，运用结构化分析方法建立环境模型（即用户要解决的问题是什么，以及要达到的目标、功能和环境）；需求分析完成后采用结构化设计方法进行系统设计，确定系统的功能模型；进入软件开发的实现阶段，运用结构化程序设计方法确定用户的实现模型，完成系统的编码和调试工作。

结构化方法是由结构化程序设计语言发展而来的。早期的计算机程序设计都是手工式的设计方法，20 世纪 60 年代软件危机的出现促使人们开始对程序设计方法进行研究，经过多年的研究与实践，逐步形成了结构化程序设计的方法。结构化程序设计就是选用最佳的程序设计语言实现编码，其目的是要使程序具有一个合理的结果，以保证程序的正确性。

1. 概要设计

概要设计也称总体设计，任务是确定软件结构。采用结构化设计方法来设计结构，其目标是根据系统的需求，分析资料，确定软件应由哪些系统或模块组成，它们采用什么方式连接，接口如何，才能构成一个好的软件结构，如何用恰当的方法把设计结果表达出来。同时，要考虑数据库的逻辑设计。

概要设计是根据系统分解与抽象的原则，自顶向下、逐步求精地完成系统的开发过程，用模块结构图来表示程序模块之间的关系。模块化是把程序划分成独立命名的、独立访问的模块。每个模块完成一个子功能，将这些模块组合起来就构成了一个整体，可以完成指

定的功能。由于模块之间是相对独立的，所以每个模块可以独立地被理解、编程、调试，模块的相对独立性能有效防止了错误在模块之间蔓延，提高了系统的可靠性，从而使大型信息系统的开发工作得以简化，缩短了软件开发周期。

2. 详细设计

详细设计的主要任务是对概要设计所得的对象类的表达做进一步的细化分析、设计和验证，为软件结构图中的每一个模块确定实现算法和局部数据结构，并用某种工具描述出来。严格地讲，详细设计应当把每个模块用到的每个函数，每个函数的每个参数的定义都精细地提供出来。一个好的详细设计，可以使程序编码的复杂性大大降低。

实际上，从需求分析到概要设计再到详细设计，一个软件项目才完成了一半，换言之，一个大型软件系统在完成了一半的时候，其实还没有开始代码的编写工作。

详细设计中可以采用逐步求精的设计方法，应采用自顶向下的策略，即在模块化分解过程中，问题一步一步细化。由于人的理解能力、记忆力有限，一个复杂问题不可能触及问题的所有方面和全部细节，为将复杂性降低到人们可以掌握的程度，常将其拆成若干个小问题再分别解决。或者说，把要解决的复杂问题分解成若干个子问题，然后分别独立解决这些子问题，再将各个子问题的解以某种方式连接起来，这就是原始问题的正确解答。如果子问题仍较复杂，则又可以把这些子问题看做新的要解决的问题，而对它们继续进行分解。这样不断地分解，最终使得子问题简单到可用若干行程序设计语言来描述，那么整个问题就解决了。这样研制出来的程序具有结构清晰的特点，有较强的可读性和可维护性。

Wirth 曾对逐步细化的方法作过如下说明：“我们对付复杂问题的最重要的办法是抽象，因此，对一个复杂的问题不应该立刻用计算机指令、数学和逻辑符号来表示，而应该用较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符号（可能是自然语言）来表示。对抽象程序做进一步的分解，并进入下一个抽象层次。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。这样的程序可能是用某种高级语言或机器指令书写的。”

详细设计之后就是按照一定的原则编制正确易懂的程序，在这一阶段，选用合适的程序设计语言、良好的编码风格，对保证程序的可读性、可靠性、可测试性和可维护性将产生重要的作用。

软件详细设计的描述工具可分为图形、表格和语言三类。下面介绍常用的详细设计描述工具。

1.2.2 传统流程图

传统流程图用于描述程序的控制流程，其基本描述符号如图 1-3 所示。

传统流程图的优点是直观，便于初学者掌握；缺点是控制流不带任何约束，可随意转移控制，使得过程的结构不清晰，不便于逐步求精。

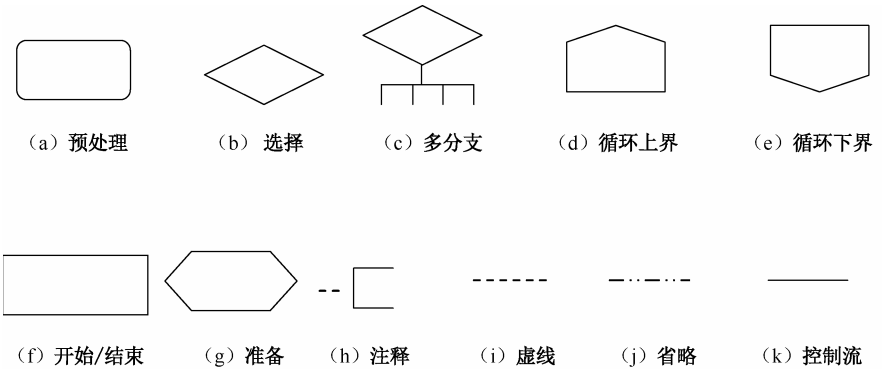

图 1-3　传统流程图的基本描述符号

1.2.3　N-S 图

N-S 图又称为盒图，它是由 Nassi 和 Shneiderman 提出的，其基本描述符号如图 1-4 所示。

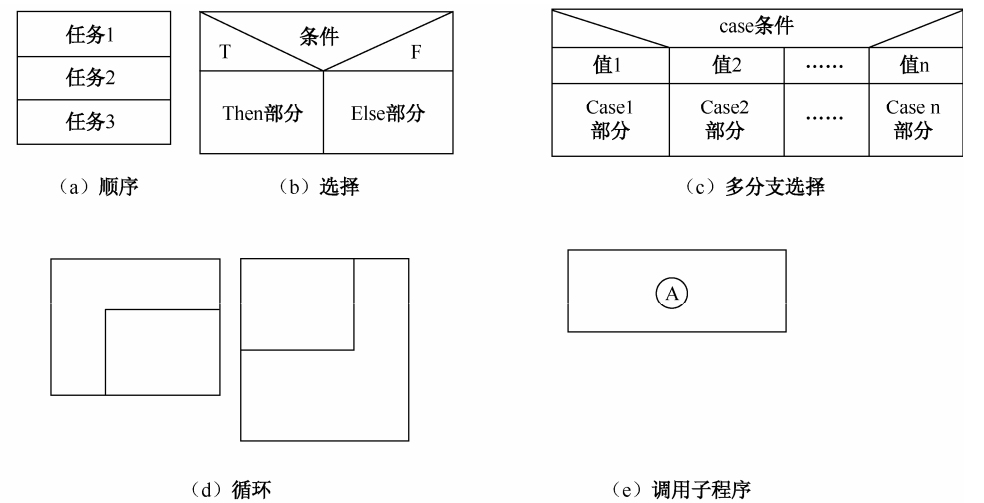

图 1-4　N-S 图基本描述符号

N-S 图易于表示结构化程序的层次结构，确定局部和全局数据的作用域。由于没有转向箭头，因此不允许随意转移控制，使得程序结构较为清晰。

1.2.4　过程描述语言

过程设计语言（PDL）又称伪码或结构化语言，现在已经有多种不同的过程设计语言。PDL 具有严格的关键字外部语法，用于定义控制结构和数据结构。同时，PDL 所表示的实际操作和内部语法通常又是灵活自由的，以便适应各种工程项目的需要。因此，一般来说，PDL 是一种“混合”语言，它用一种自然语言中的词汇和一种结构化程序设计语言中的控制结构来描述算法。

一般来说，PDL 应该具有以下几个特点。

1）关键字的固定语法，提供了结构化控制结构、数据说明和模块化的特点。为了使结构清晰和可读性好，通常在所有可能嵌套使用的控制结构的头和尾都有关键字。例如，if…endif 等。

2）用自然语言的自由语法来描述处理部分。

3）具有数据说明的手段。它应该既包括简单的数据结构(如纯量和数组)，又包括复杂的数据结构（如链表或层次的数据结构）。

4）具有模块定义和调用的机制，提供各种接口描述模式。

PDL 作为一种设计工具，具有以下优点。

1）可以作为注释直接插在源程序中。这样做的好处是，能促使维护人员在修改源程序代码的同时也修改 PDL 注释，有助于保持文档和程序的一致性，提高了文档的质量。

2）可以使用普通的正文编辑程序或文字处理系统，很方便地完成 PDL 的书写和编辑工作。

3）利用已有的自动处理程序，可以自动由 PDL 生成程序代码。

PDL 的缺点是：不如图形工具形象直观；描述复杂的条件组合与动作之间的对应关系时，不如判定表清晰简单。

1.2.5 数据字典

数据字典（Data Dictionary，DD）是开发者与用户相互沟通的有效途径之一。它能形象地向用户描述开发者的意图，使用户明白数据库可能具有的项目，可有效减小开发者和用户之间的交流障碍，也有利于用户向开发者提出自己的需求，避免因理解分歧造成的代价巨大的接口问题。

数据字典是各类数据描述的集合，它是进行详细的数据收集和数据分析后所获得的主要成果，是结构化分析方法的核心。数据字典对数据流图中的各个元素做完整的定义与说明，是数据流图的补充工具。数据流图与数据字典共同构成系统的逻辑模型。

数据字典的内容包括以下信息：图形元素的名称、别名或编号、分类、描述、定义、位置等。数据字典中所有的定义都是严密的、精确的，不可有二义性。

下面是数据字典中经常出现的一些符号的说明，设 X、a 和 b 为数据元素。

X=a+b，表示 X 由 a 和 b 组成。

X=[a,b]，X=[a/b]，表示 X 由 a 或 b 组成。

X=（a），表示 a 可在 X 中出现，也可能不出现。

X={a}，表示 X 由 0 个或多个 a 组成。

X=a .. b，表示 X 可取 a～b 的任一值。

X=m{a}n，表示 X 由 m～n 个 a 组成，即至少有 m 个 a，至多有 n 个 a。

X="a"，表示 X 为取值 a 的基本数据之类，即 a 无需进一步定义。

例 1-1　数据字典示例。数据文件“ 存折 ”的格式如图 1-5 所示。

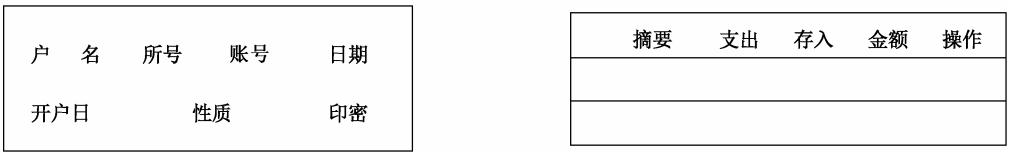

图 1-5　存折的格式

此文件在数据字典中的定义格式如下。

```
存折=户名+所号+账号+开户日+性质+（印密）+1 {存取行}50
户名=2{字母}24　（注：户名中字母至少出现2次，至多出现24次）
所号=“001”··“999”　（注：所号规定为三位数）
账号=“0000000001”··“9999999999”　（注：账号规定为十位数字）
开户日=年+月+日
性质=“1”··“6”　（注：“1”表示普通用户，“5”表示工资用户等）
印密=“0”　（注：印密在存折上不显示）
存取行=日期+（摘要）+支出+存入+余额+操作+复核
日期=年+月+日
年=“0001”··“9999”
月=“01”··“12”
日=“01”··“31”
摘要=1{字母}4　（注：表明此次操作是存还是取）
支出=金额
金额=“0000000.01”··“9999999.99”
操作=“00001”··“99999”　（注：操作指银行职员代码，用五位整数表示）
……
```

1.2.6　程序设计风格

程序设计风格指一个人编制程序时所表现出来的特点、习惯、逻辑思路等。 在程序设计中要使程序结构合理、清晰，形成良好的编程习惯，对程序的要求不仅是可以在机器上执行，给出正确的结果，还要便于程序的调试和维护，增加程序的可读性。

为了提高程序的可阅读性，要建立良好的编程风格，它包括良好的代码设计，函数模块，接口功能及可扩展性等，更重要的是程序设计过程中代码的风格，包括缩进、注释、变量及函数的命名等。程序设计风格应该遵循以下几个原则。

1. 源程序文档化原则

1）标识符应按意取名（见名知意）。

2）程序应加注释。注释是程序员与读者之间通信的重要工具，用自然语言或伪码描述。它说明了程序的功能，特别在维护阶段，对理解程序提供了明确指导。注释分为序言性注释和功能性注释。

序言性注释应置于每个模块的起始部分，它要说明每个模块的用途、功能；说明模块的接口，如调用形式、参数描述及从属模块的清单；重要数据的名称、用途、限制、约束及其他信息的描述。

功能性注释嵌入在源程序内部，说明程序段或语句的功能及数据的状态。注释时，不

要每一行程序都加注释，要使用空行、缩格或括号，以便区分注释和程序。

2. 数据说明原则

1）数据说明顺序应规范，使数据的属性更易于查找，从而有利于测试、纠错与维护。

2）一个语句说明多个变量时，各变量名按字典序排列。

3）对于复杂的数据结构，要加注释，说明在程序实现时的特点。

3. 语句构造原则

简单直接，不能为了追求效率而使代码复杂化。为了便于阅读和理解，不要一行多个语句，不同层次的语句采用缩进形式，使程序的逻辑结构和功能特征更加清晰。要避免复杂的判定条件、多重的循环嵌套。表达式中使用括号以提高运算次序的清晰度等。

4. 输入输出原则

1）输入操作步骤和输入格式尽量简单。

2）应检查输入数据的合法性、有效性，提示必要的输入状态信息及错误信息。

3）输入一批数据时，使用一个特殊数据或文件结束标志，而不要用计数来控制。

4）交互式输入时，提供可用的选择和边界值。

5）当程序设计语言有严格的格式要求时，应保持输入格式的一致性。

6）输出数据规范化、图形化。

5. 追求效率原则

该原则指处理机时间和存储空间的使用，对效率的追求明确以下几点。

1）效率是一个性能要求，目标在需求分析时给出。

2）追求效率建立在不损害程序可读性或可靠性的基础上，要先使程序正确、清晰，再提高程序效率。

3）提高程序效率的根本途径在于选择良好的设计方法，良好的数据结构、算法，而不是靠编程时对程序语句做调整。