第1章

实例分析与设计

本章在介绍程序设计和软件开发基本知识的基础上，重点介绍基于软件工程的软件开发方法和过程，并以软件生命周期的瀑布模型作为开发模型，对某培训机构“学籍管理系统”开发进行需求分析和设计，为其他章节介绍软件开发的后续阶段和方法奠定基础。

1.1 学籍管理实例需求分析

1.1.1 软件开发基础

大家都知道，计算机是由硬件和软件组成的。计算机硬件由中央处理器(CPU)、内存、总线、I/O 接口电路和外部设备组成。计算机软件是指程序及其有关数据和文档的完整集合，它分为内核、操作系统、应用软件不同层次。计算机软件是计算机的灵魂，是计算机应用系统的核心，没有计算机软件，硬件就是废铁一堆。

1. 计算机程序与程序设计

计算机软件主要是计算机程序。程序并非计算机的专用品，人们做任何事情都需要一定的程序，只是人们没有意识到或者对此熟视无睹罢了。例如，每天的作息时间、日常工作流程等，是人们生活和工作的程序。

计算机硬件由一系列的电子部件组成，包括 CPU、内存、总线、I/O 接口和外设，这些电子部件在 CPU 的指挥下进行协调工作。CPU 是计算机的大脑，由它解释一条条指令，协调各个电子部件共同完成指令的执行。计算机没有人类那样的思考能力，即便未来具有思考能力，也是人类赋予的。因此，人类必须给计算机下达指令，告诉它如何操作。

要想计算机能够正确完成人们想要完成的工作任务，就必须告诉计算机需要执行哪些指令，而这些指令序列就构成了计算机程序。为了完成工作任务而编写一串适当指令的过程就是程序设计。

2. 程序设计语言的发展

计算机程序是一系列的指令序列，由 CPU 解释执行，从而完成人们想要完成的任务。这些 CPU 指令是用来表示电路状态的，即高电平和低电平，分别用 0 或 1 来表示。由 1 和 0 的编码构成 CPU 指令。不同的 CPU 能够识别的指令数量、格式是不同的。CPU 能够识别的所有指令称为 CPU 的指令系统。

CPU 指令系统的指令是有限的，它是由 CPU 性能决定的，如 8086 CPU 比 80286 CPU 的指令系统小。而由 CPU 指令系统构成计算机所谓的语言，称为机器语言。

程序设计就是编写一串适当的指令代码，以操纵计算机。用 0 和 1 编码组成的指令——机器语言来进行程序设计，记忆困难，容易出错，且不易修改和维护，程序设计效率低。为了提高程序设计的效率，提高程序的可读性，人们发明了助记符，把由 1 和 0 编码构成机器指令用助记符来标记，从而产生了汇编语言。

使用汇编语言进行程序设计，大大地加快了程序设计的速度，也便于程序的阅读和修改。但是，汇编语言程序必须进行编译，把汇编语言程序翻译成机器语言之后，才能由计算机执行。

用汇编语言编写的程序称为源程序。源程序并不能被计算机解释和执行，必须经过

一个转换工具转换成机器语言之后，才能由计算机执行。这个转换后的程序称为目标程序，而这个转换工具称为编译程序。有了适应不同计算机硬件和操作系统平台的编译工具，同样的源程序可以编译成不同的目标程序，实现跨平台运行。由于程序越来越复杂，程序分成多个程序模块，分别进行编写和编译，最后通过称为连接工具的程序，把各个模块的目标程序连接成一个整体的目标程序，称为可执行程序(EXE)，由计算机执行。

随着编译技术的进步与发展，程序设计语言越来越高级，越来越接近人类的自然语言，如 BASIC、C、VB、VC 等，统称为高级语言。同时，编写程序的工具也快速发展，人们不再只用“记事本”这样的工具来编写程序了，集程序编写、编译和连接的编程的环境大大改善，可视化设计成为时尚。

3. 程序设计方法

(1) 结构化程序设计

早期的计算机运行速度慢，内存有限，程序设计考虑最多的是如何编写短小、高效的程序，以节省资源和提高运行速度，为此，人们绞尽脑汁设计精妙的算法，不会重视程序本身的结构。随着程序功能的增强，程序变得越来越复杂，难于阅读，纠错排错越来越困难，维护起来越来越艰难。一方面是编写程序的效率低下，另一方面是程序的错误不易排除，质量、进度难以保证。因此，自 20 世纪 60 年代开始，人们越来越重视程序可读性问题，于是出现了结构化程序设计的概念和思想。

所谓结构化程序设计，就是采用自顶向下、逐步求精，使用顺序、选择和循环三种控制结构来构造程序的设计方法。

自顶向下、逐步求精符合人们解决复杂问题的一般思路。这种思路就是把一个大的问题分解成多个子问题，子问题再根据其难度分成更小的子问题，直到相对容易解决的、独立的子问题为止。

采用结构化程序设计方法，应该注意：

- 设计过程严格遵守自顶向下的次序。从顶点出发，逐渐向下展开。也就是说，从抽象的问题出发，逐渐分解、延伸到比较具体的问题。
- 同层次的子任务之间保持相对独立。一个子任务的任何改变只会影响该子任务的所属下层，而不会影响其他子任务。

结构化程序设计方法，强调程序模块化，模块具有完整的功能，并相对独立。模块采用单入口、单出口的结构，模块之间通过接口与其他模块交换信息，减少模块之间相互干扰。

结构化程序设计方法是面向过程的设计方法，以算法为核心，把数据和过程作为独立的部分，割裂数据与过程之间本来存在的联系。

(2) 面向对象程序设计

面向对象方法，是 1979 年以后发展起来的，其基本原则就是尽可能地模拟人类习惯的思维方式，把问题空间和求解空间一致起来。通过对象、类、继承和消息通信等概念来描述整个客观世界对象及其联系。

面向对象方法的主要要素有：

- 对象。面向对象方法认为客观世界由对象组成，任何事物都是对象。复杂的对象由简单的对象组成。例如，一张桌子是一个对象，一把方凳也是一个对象。像一张圆桌子的腿也可以是一个对象，此时，圆桌子的腿就是圆桌子的子对象。对象具有自己的属性，这些属性通过数据来描述。例如，一张圆桌子有颜色、大小、高度等规格参数。一个对象还具有施加与该对象的操作（方法）。如变更一张桌子的颜色等。
- 类。把具有相同属性和操作的一组相似对象划分为对象类。同类对象具有相同的属性和方法，但是，每个对象的属性值不同，执行方法的结果也不一样。例如，圆桌子就是各种不同规格一张张圆桌子的类。而一张具体的不同颜色、不同大小的圆桌子就是这个对象类的一个对象，称为类的实例。
- 继承。继承是父类与子类的关系。继承把若干对象类组成一个层次结构系统。父对象类比其子类更为抽象，子类具有其父类的属性和方法（称为继承）；子类比其父类更为具体，子类可以有其父类更多的属性和方法。例如，桌子是圆桌子类的父类，圆桌子是桌子类的子类，方桌子也是桌子类的子类。
- 消息通信。对象之间只有通过消息通信进行相互作用。一个对象要进行某种操作，必须接收消息后才能执行对象的操作。

面向对象方法具有许多优点，是当前广泛使用的软件开发方法之一。它包括面向对象程序设计方法和面向对象程序分析两个步骤，程序设计和程序分析时所用的概念和表示方法一致，可以在不同的阶段交替、回溯。

（3）构件式程序设计

构件式程序设计方法的思路就是把程序看成不同的程序组件，这些组件通过一定体系进行装配，从而完成程序的设计。就像当前个人计算机那样，通过组装标准化的配件组装个人的计算机。

构件式程序设计是建立在面向对象程序设计的基础上，把程序设计变成程序构件设计和装配，大大地提高程序设计的效率，也把程序设计变得更加简单。

4. 软件、软件设计

20 世纪 40～60 年代，电子计算机的价格昂贵，内存容量小，运行速度慢。计算机程序规模较小，程序设计主要集中在如何节省存储单元和提高计算速度上，没有任何的文档资料。那时，软件就等同于程序，软件设计也就是程序设计。

随着计算机技术的快速发展，内存增大，运行速度大大提高，人们期望计算机能为人们做更多的事情。于是，计算机的应用领域快速渗透，软件开发生产率提高的速度远远跟不上计算机应用迅速普及和深入的速度。因此，“软件危机”出现了。

“软件危机”的出现，人们不得不考虑提高软件设计效率和程序质量问题。人们采用工程学原理和方法，把软件看成产品来组织和管理生产，以提高软件生产效率，确保软件产品的质量。

软件设计开始采用工程化的开发方法、标准和规范，软件也不止是计算机程序，还包括软件开发过程中必要的文档和数据，即软件是指计算机程序及其有关数据和文档的完

整集合。软件设计也不再只是程序设计了,还包括软件设计的不同阶段的任务和活动。

5. 软件工程

软件工程是采用工程学原理、技术、方法开发和维护软件的方法,目的就是实现软件产品的优质高产。软件工程方法把软件开发工作划分为多个阶段,并规定每个阶段具有不同的任务、活动、步骤和产品。

软件从问题定义开始,经过开发、使用、维护,直到被淘汰的过程构成软件产品的生命周期。把软件产品的生命周期划分为若干相互区别而又联系的阶段,每个阶段赋予不同的任务,这样,简化了各个阶段的工作,容易制订软件开发计划,便于控制与管理,确保产品的质量。

软件生命周期的划分可以根据不同需要,进行不同的划分,从而形成不同的生命周期模型。软件的生命周期模型又称为软件开发模型,总体上分为两种:瀑布模型和快速原型模型。

(1) 瀑布模型

瀑布模型把软件生命周期划分为计划时期、开发时期、运行时期三个时期,每个时期细分成若干不同的阶段,如图 1-1 所示。

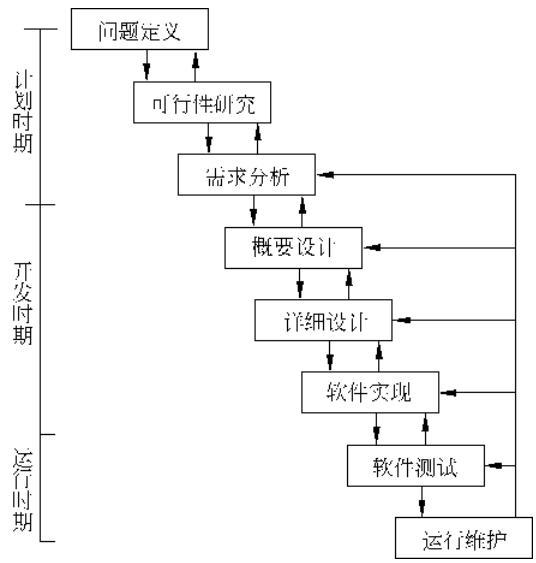

图 1-1　瀑布模型

瀑布模型软件开发分阶段、按顺序进行,只有前一个阶段的任务完成并审核通过之后才能开展下一阶段的任务,前一个阶段的工作成果是下一个阶段任务工作的基础。每个阶段顺序进行,有时会有返工。

瀑布模型软件开发各阶段的主要任务描述如下:

① 计划时期包括问题定义、可行性研究和需求分析阶段。

- 问题定义。确定软件开发的范围、目标、规模和基本要求。
- 可行性研究。从技术、经济等方面分析软件开发是否值得开发,避免人力、物力和

时间的浪费。

- 需求分析。根据问题定义、可行性研究和审核的结果，进行系统功能的确定。

② 开发时期包括概要设计、详细设计、软件实现和软件测试阶段。

- 概要设计。确定系统的设计方案，明确系统的体系结构和软件结构。
- 详细设计。详细描述应该如何实现系统，包括模块实现的算法及数据结构。
- 软件实现。进行程序编码和模块测试。
- 软件测试。在模块测试的基础上，对系统进行综合测试，是在软件交付用户之前的测试。

③ 运行时期包括运行维护和软件寿命的延长阶段。

在软件运行期间，对软件适应环境和延长寿命的维护。

(2) 快速原型模型

在进行软件的需求分析时，软件开发人员需要反复与用户进行信息交流，才能全面和准确地了解用户的需要。很不幸，大多数情况下，用户此时很难准确地描述出用户的真实需求。于是，用户常常提出改变原先的要求，以便系统更加满足他们的实际需要，或者，对系统的成功提出质疑。快速原型模型的提出，是为减少系统由于需求难以定义而造成系统开发成功的不确定性。

快速原型模型就是快速开发一个可以运行的原型系统，请用户使用该原型系统，让用户在使用原型系统时准确地认识到他们实际的真实需求，从而形成系统的需求规格说明文档以获取用户的确切的需求。

快速原型模型开发的原型系统，通常包括用户需求的主要功能。原型系统可以成为最终系统的组成部分，也可以仅仅是系统需求分析的工具。快速原型模型需要用户参与开发过程，并通过用户对原型系统的运行和评价，开发人员可以充分地与用户进行交流，全面、准确地把握用户的真实需求。

根据建立原型的目的不同，原型分为：

- 渐增式原型。快速建立的系统原型是最终系统的一部分，在用户需求出现变更或增加时，在原型的基础上进行增量开发，经过多次的反复增量开发从而产生最终用户需要的系统。
- 验证需求的原型。快速建立的系统原型只是为了获得用户确切的需求，之后，原型被弃用，因此，原型可以在与目标系统不同的环境中构建。
- 验证设计方案的原型。当采用新颖的设计思想进行软件设计时，为了保证软件开发的成功，构造快速原型以验证设计方案的可行性。此类原型可以作为目标系统的组成部分，也可以被弃用。当原型最终被弃用，原型构造的环境可以与构造目标系统的环境不同。

6. 需求分析方法与工具

(1) 结构化分析与工具

结构化分析是面向数据流的需求分析方法，是 20 世纪 70 年代由 Yourdon、Constaintine 及 DeMarco 等人提出和发展的，并得到广泛的应用。

结构化分析方法的基本思想是“分解”和“抽象”。分解是指对于一个复杂的系统，为了将复杂性降低到可以掌握的程度，把大问题分解成若干小问题，然后分别解决。抽象指在进行分层分解时，只考虑当前层次问题本质的属性，忽略细节的过程。然后，逐层添加细节，直至涉及到最详细的内容。

图 1-2 所示是自顶向下逐层分解的示意图。顶层抽象地描述了整个系统，底层具体地画出了系统的每一个细节，而中间层是从抽象到具体的逐层过渡。

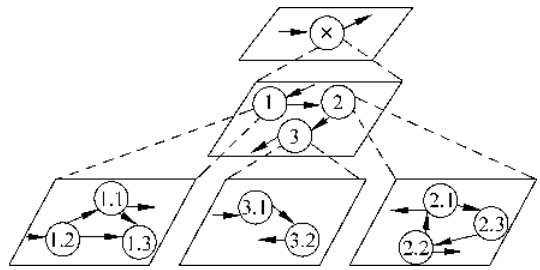

图 1-2　自顶向下逐层分解示意图

结构化分析方法描述工具有分层的数据流图、数据词典、描述加工逻辑的结构化语言、判定表或判定树。

数据流图是描述系统中数据流程的图形工具，它标识了一个系统的逻辑输入和逻辑输出，以及把逻辑输入转换成逻辑输出所需的加工处理。数据流图的基本图形符号如图 1-3 所示。

其中，箭头表示数据流，圆或椭圆表示加工处理，单杠或双杠表示数据存储，矩形表示数据的源点或终点，即外部实体。

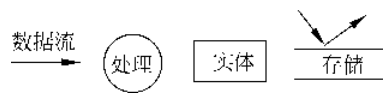

图 1-3　数据流图的基本图形符号

数据流是表示数据在系统内流动的路径，由一组固定的数据项组成。除了与数据存储之间的数据流不用命名外，其余数据流都应该在箭头上方或下方用名词或名词短语命名。数据流可以从处理流向处理，从处理流向存储或从存储流向处理，从源点流向处理或从处理流向终点。

处理，即数据处理，它对数据流进行某些数据操作或变换。每个处理要有命名，通常是动词短语，简明地描述完成什么处理。在分层的数据流图中，处理还应有编号。

数据存储是指保存的数据，它可以是数据库文件或任何形式的数据组织。流向数据存储的数据流可理解为写入或查询，从数据存储流出的数据可理解为读取数据或得到查询结果。

数据源点和终点是软件系统外部环境中的实体（包括人员、组织或其他软件系统），统称为外部实体。一般只出现在数据流图的顶层图中。

特别要注意的是，数据流图不是流程图，也不是控制流。数据流图是从数据的角度来描述一个系统，而流程图则是从对数据进行处理的工作人员的角度来描述系统。数据流图中的箭头是数据流，而流程图中的箭头则是控制流，控制流表达的是程序执行的次序。

绘制数据流图的一般原则是:“先全局后局部,先整体后细节,先抽象后具体”。通常数据流图分为顶层、中间层、底层。顶层图只有一张,说明了系统的边界,即系统的输入和输出数据流。底层由一些称为基本处理的不能再分解的处理组成。在顶层和底层之间的是中间层,描述了某个处理的分解,而它本身需要进一步分解。绘制数据流图的步骤如下:

① 确定系统范围,画出顶层的数据流图。

② 逐层分解顶层数据流图,获得若干中间层数据流图。

③ 画出底层的数据流图。

数据流图只是表达了系统的“分解”,为了完整地描述这个系统,还需借助“数据词典”对图中的每个数据和处理给出解释。对数据流图中包含的所有元素的定义的集合构成了数据词典。它有四类条目:数据流、数据项、存储及基本处理。

数据流条目定义数据流图中数据流,通常描述组成数据流的数据项。例如,名为“学生”的数据流,其数据流条目描述为:

学生=学号+姓名+性别+年龄+政治面貌+籍贯

又如,

报名单=姓名+单位名+年龄+性别+课程名

数据项条目定义某个数据单项,描述数据类型、取值范围等。存储条目定义数据存储格式、数据组织方式等内容。

处理描述处理数据的功能,可以用结构化语言、判定表、判定树等多种形式描述,或者它们的结合。

结构化语言描述形式如下:

- 处理编号:在数据流图中的编号。
- 处理名:在数据流图中的处理名字。
- 处理逻辑:本处理的处理方法说明。
- 有关信息:执行条件等。

判定表与判定树用于描述一些结构化语言不易表达清楚的处理逻辑。在数据流中的处理中,如果判断的条件较多,各条件又相互组合,相应采取的措施即策略较多,在这种情况下用判定表描述。

采用判定表来描述这类处理逻辑比较合适,它表达清晰、简洁。判定表是一种图形工具,呈表格形。判定表的编制,首先要明确加工处理的功能与目标,然后要识别影响策略的各项因素即条件,列出这些因素可能出现的状态,并制定出判定的规则。

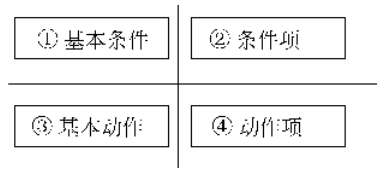

图 1-4 判定表结构

判定表组成如图 1-4 所示。描述某学生奖励政策的判定表如图 1-5 所示。

判定树又称决策树,是一种描述加工的图形工具,适合描述问题处理中具有多个判断,而且每个决策与若干条件有关。使用判定树进行描述时,应该从问题的文字描述中分清哪些是判定条件,哪些是判定的决策,根据描述材料中的连接词找出判定条件的从属关系、并列关系、选择关系,根据它们构造判定树。

条件	已修课程各科成绩比率	优秀≥70% 优秀≥50% 中以下≤15% 中以下≤20%	Y Y Y Y N N N N — — — — Y Y Y Y Y Y N N Y Y N N — — Y Y — — Y Y	状态
	团结纪律评分	优良 一般	Y N Y N Y N Y N N Y N Y N Y N Y	
奖励方案	一等奖 二等奖 三等奖 鼓励奖		* * * * * * * *	判定规则

图 1-5　学生的奖学金评定判定表

某工厂奖励超产工人的处理功能判定树描述，如图 1-6 所示。判定树比起文字叙述，使人一目了然，清晰地表达了在什么情况下采取什么策略，不易产生逻辑上的混乱。因而，判定树是描述基本处理逻辑功能的有效工具。

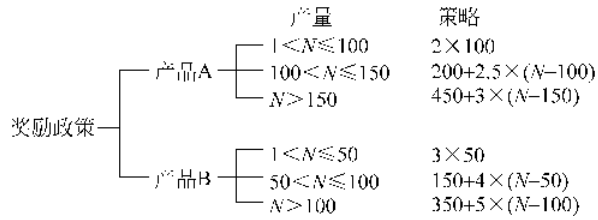

图 1-6　超产奖励判定树

(2) 面向对象需求分析与工具

面向对象方法通过对象、类、继承和消息通信等概念来描述整个客观世界对象以及其联系，非常适合系统的需求分析与设计。UML(Unified Modeling Language，统一建模语言)是面向对象系统分析与设计的主要工具之一。

UML 是一种编制系统蓝图的标准化语言，可以实现大型复杂系统各种成分描述的可视化、说明并构造系统模型，以及建立各种所需的文档，它是一种定义良好、易于表达、功能强大且普遍适用的建模语言。

UML 支持从需求分析开始的软件开发的全过程。UML 通过三类图形建立系统模型：用例(Use Case)图、静态结构图(对象类图、对象图、组件图、配置图)和动态行为图(顺序图、协同图、状态图、活动图)，这些图可以从不同的抽象角度实现系统的可视化。

用例图中包括系统、角色和用例三种元素，是描述系统功能的工具。它是从外部用户的角度观察系统应具有的功能，帮助需求分析人员和用户理解系统的行为。用例示意图如图 1-7 所示。

类图是用类和它们之间的关系描述系统的一种图示，是从静态角度表示系统的。类的示意图如图 1-8 所示。

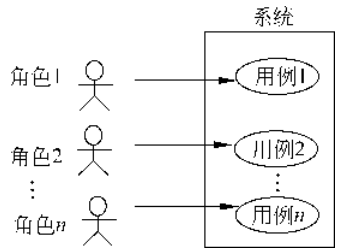

图 1-7 用例示意图

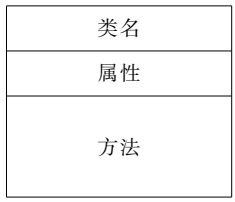

图 1-8 类的示意图

状态图主要是用来描述对象、子系统、系统的生命周期。通过状态图可以了解到一个对象所能到达的所有状态以及对象受到的事件对对象状态的影响等。状态示意图如图 1-9 所示。

图 1-9 状态示意图

UML 具有以下特点：

- 面向对象。UML 支持面向对象技术的主要概念，提供了一批基本的模型元素的表示图形和方法，能简洁明了地表达面向对象的各种概念。
- 可视化，表示能力强。通过 UML 的模型图能清晰地表示系统的逻辑模型和实现模型。可用于各种复杂系统的建模。
- 独立于过程。UML 是系统建模语言，独立于开发过程。
- 独立于程序设计语言。用 UML 建立的软件系统模型可以用 Java、VC++、Smalltalk 等任何一种面向对象的程序设计来实现。
- 易于掌握使用。UML 图形结构清晰，建模简洁明了，容易掌握使用。

使用 UML 进行系统分析和设计，可以加速开发进程，提高代码质量，支持动态的业务需求。UML 适用于各种规模的系统开发。能促进软件复用，方便地集成已有的系统，并能有效处理开发中的各种风险。

UML 的面向对象分析设计过程如下：

① 识别系统的用例和角色。依据项目的业务流程图和数据流程图以及项目中涉及的各级操作人员，通过分析，识别出系统中的所有用例和角色；接着分析系统中各角色和用例间的联系，再使用 UML 建模工具画出系统的用例图，同时，勾画系统的概念层模型，借助 UML 建模工具描述概念层类图和活动图。

② 进行系统分析，并抽取类。系统分析的任务是找出系统的所有需求并加以描述，同时建立特定领域模型。建立域模型有助于开发人员考察用例，从中抽取出类，并描述类之间的关系。

③ 系统设计，并设计类及其行为。

1.1.2 学籍管理实例分析

开发学籍管理系统实例，首先对应用实例进行需求分析。