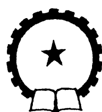
高等教育规划教材

软件工程基础

宋　雨　编著



机 械 工 业 出 版 社

本书理论联系实际，按照软件工程的基本原理，从实用的角度介绍了软件需求分析方法、软件设计方法、软件的编程实现要求、软件的测试及维护方法以及软件项目管理方法，而这些也是软件工程学科的基本内容。书中第7章还给出了若干经典案例供读者学习。为促进学习效果，第8章给出了140个精选软件工程设计题目及其功能要求供读者作为软件工程课题选用，第8章还给出了课程设计的评分标准供教师参考。每章都有结合实际的案例以及供读者练习的相应习题。

本书是软件工程的入门教材，内容通俗、易懂，注重趣味性、故事性和情节性。本书适合作为高等学校计算机相关专业的教学用书，也可作为对软件工程学科感兴趣的工程技术人员的参考用书。

本书配套授课电子课件，需要的教师可登录 www.cmpedu.com 免费注册，审核通过后下载，或联系编辑索取（QQ：2850823885，电话：010-88379739）。

图书在版编目（CIP）数据

软件工程基础 / 宋雨编著. —北京：机械工业出版社，2016.3

高等教育规划教材

ISBN 978-7-111-52511-0

Ⅰ. ①软…　Ⅱ. ①宋…　Ⅲ. ①软件工程－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2016）第 021357 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）

策划编辑：　　　　责任编辑：

责任校对：张艳霞　　责任印制：

印刷（　　　装订）

2016 年 3 月第 1 版 • 第 1 次印刷

184mm×260mm • 20.25 印张 • 499 千字

0001－3000 册

标准书号：ISBN 978-7-111-52511-0

定价：　　　元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
服务咨询热线：（010）88361066	机 工 官 网：www.cmpbook.com
读者购书热线：（010）68326294	机 工 官 博：weibo.com/cmp1952
（010）88379203	教育服务网：www.cmpedu.com
封面无防伪标均为盗版	金　书　网：www.golden-book.com

出版说明

当前，我国正处在加快转变经济发展方式、推动产业转型升级的关键时期。为经济转型升级提供高层次人才，是高等院校最重要的历史使命和战略任务之一。高等教育要培养基础性、学术型人才，但更重要的是加大力度培养多规格、多样化的应用型、复合型人才。

为顺应高等教育迅猛发展的趋势，配合高等院校的教学改革，满足高质量高校教材的迫切需求，机械工业出版社邀请了全国多所高等院校的专家、一线教师及教务部门，通过充分的调研和讨论，针对相关课程的特点，总结教学中的实践经验，组织出版了这套“高等教育规划教材”。

本套教材具有以下特点：

1）符合高等院校各专业人才的培养目标及课程体系的设置，注重培养学生的应用能力，加大案例篇幅或实训内容，强调知识、能力与素质的综合训练。

2）针对多数学生的学习特点，采用通俗易懂的方法讲解知识，逻辑性强、层次分明、叙述准确而精炼、图文并茂，使学生可以快速掌握，学以致用。

3）凝结一线骨干教师的课程改革和教学研究成果，融合先进的教学理念，在教学内容和方法上做出创新。

4）为了体现建设“立体化”精品教材的宗旨，本套教材为主干课程配备了电子教案、学习与上机指导、习题解答、源代码或源程序、教学大纲、课程设计和毕业设计指导等资源。

5）注重教材的实用性、通用性，适合各类高等院校、高等职业学校及相关院校的教学，也可作为各类培训班教材和自学用书。

欢迎教育界的专家和老师能提出宝贵的意见和建议。衷心感谢广大教育工作者和读者的支持与帮助！

机械工业出版社

前　言

人类已进入大数据时代，这是一个信息爆炸的时代，这都应该归因于计算机处理能力的提高，而计算机中起主导作用的是软件。稍加注意就会发现，软件无处不在，软件工程无处不在。毫不夸张地说，软件在技术进步方面起到了决定性的作用，国家财政收入中与软件产业的相关度越来越高，软件产业收入所占的比例也越来越高，软件及软件产业发展的速度超出人们的想象，2014 年我国的软件产业总收入达到了 3.7 万亿元。与此同时，与软件产业紧密相关的软件工程也得到了很大的重视，软件工程作为一门学科虽然只有 40 多年的时间，但从它诞生之日起就显示出极强的生命力。

为适应软件工程学科的快速发展，2011 年国务院学位委员会及教育部将软件工程设立为国家一级学科，这既为软件工程学科发展指明了方向，也说明软件工程的地位和作用十分重要。软件工程涉及的内容广泛、丰富，无论广度还是深度，其他学科都难以覆盖。

本书作者长期从事软件工程教学和科研，在参考了大量同类文献之后，按学生的接受程度、学科的基本内容和要求编写了本书。本书是软件工程的入门教材，适合初学者作为软件工程课程的教材使用，也适用于对软件工程学科感兴趣的读者阅读。软件工程是快速发展的学科，因而，虽然本书是基础教程但在书中也尽量对发展中的新技术有所反映。

本书的特点如下。

1）力求通俗、易懂，尽量使枯燥的内容变得有趣，因而在叙述软件工程技术和方法上增加了趣味性、故事性和情节性。

2）尽量将难点分散，避开高深的知识，并使全书在叙述上有节奏感，增强初学者学习的信心。

3）理论结合实际，每一章都有案例，使理论不抽象、技术能“落地”。

4）增强实用性。书中既有案例又列出了大量的实际题目，增强读者学习的牢固性，使其在其他学科的学习和实践中仍然有参考意义。

全书共 8 章，第 1 章是软件工程概述，介绍了软件的定义、起源和分类以及软件工程的产生和定义，讨论了软件生命周期的概念，叙述了常见的软件开发模型。第 2 章阐述了软件需求分析方法，包括结构化分析方法、原型化分析方法、面向对象建模及 UML 方法，以及软件需求规约说明书（SRS）的构造和要求。第 3 章是软件系统的设计，介绍了软件设计的基本原理，讲述了结构化设计方法和面向对象的设计方法，简要介绍了一些新方法。第 4 章是软件的编程实现，包括编程语言的选择、分类和编程要求，介绍了常用的面向对象的编程语言。第 5 章是软件的测试及维护，叙述了软件测试的基本原理、测试用例设计及面向对象的测试方法，讲述了软件维护的基本原理和软件再工程的内容。第 6 章是软件项目管理，包括软件范围的确定、软件资源的考虑、软件成本估算模型和技术，以及如何使用甘特图和 PERT

图来安排软件工程项目进度。第 7 章是从公开资料中整理而成的典型软件项目案例，具有很高的指导价值。第 8 章是软件工程课程设计的内容、基本要求、考核标准以及交付文档的要求和格式，这一章给出了 140 个具有实际意义的软件工程课程设计题目，可作为课程设计的目标、功能和性能要求。

本书配有习题解答、电子教案等教辅材料。使用本书的课程建议授课学时为 40～48 学时，课程设计可安排 2 周。

本书由华北电力大学宋雨编著。在编写过程中得到了多位同仁的支持和帮助，在这里一并表示感谢。由于时间仓促，书中难免存在不妥之处，请读者批评指正，并提出宝贵意见。

编　者

目　　录

前言
第 1 章　软件工程概述 …… 1
1.1　软件分类和演化 …… 1
1.1.1　软件的起源和分类 …… 1
1.1.2　软件工程的产生和发展 …… 4
1.2　软件的生命周期 …… 8
1.2.1　计划阶段 …… 9
1.2.2　分析和定义阶段 …… 10
1.2.3　设计阶段 …… 11
1.2.4　实现阶段 …… 12
1.2.5　测试阶段 …… 13
1.2.6　运行和维护阶段 …… 14
1.3　软件开发模型 …… 15
1.3.1　瀑布模型 …… 15
1.3.2　演化模型 …… 16
1.3.3　螺旋模型 …… 17
1.3.4　喷泉模型 …… 18
1.3.5　其他模型 …… 18
1.4　实用案例 …… 23
1.4.1　出卷系统的开发模型选择 …… 23
1.4.2　住宅安全系统 SafeHome 的开发模型选择 …… 25
1.5　小结 …… 26
1.6　习题 …… 26
第 2 章　软件需求分析 …… 29
2.1　结构化需求分析方法 …… 29
2.1.1　数据流图及其画法 …… 30
2.1.2　数据词典及其描述 …… 38
2.1.3　功能说明 …… 41
2.2　原型化分析方法 …… 44
2.2.1　开发模型 …… 45
2.2.2　快速原型技术介绍 …… 48
2.2.3　用户界面开发 …… 51
2.3　面向对象建模及 UML 方法 …… 51
2.3.1　面向对象基本概念 …… 51
2.3.2　面向对象建模 …… 54
2.3.3　统一建模语言 UML …… 63
2.4　需求规约说明书（SRS） …… 67
2.4.1　SRS 的内容 …… 67
2.4.2　SRS 的作用 …… 68
2.4.3　SRS 的特征 …… 69
2.4.4　SRS 的构造原则 …… 70
2.4.5　SRS 的评审 …… 71
2.5　案例：图书馆系统的软件需求分析 …… 72
2.5.1　确定系统参与者 …… 72
2.5.2　开发系统场景 …… 73
2.5.3　绘制系统用例图 …… 73
2.5.4　描述用例 …… 74
2.6　小结 …… 75
2.7　习题 …… 76
第 3 章　软件系统的设计 …… 80
3.1　软件设计的基本原理 …… 80
3.1.1　软件设计的概念和原则 …… 83
3.1.2　软件概要设计 …… 90
3.1.3　软件详细设计 …… 97
3.2　结构化设计方法 …… 107
3.2.1　软件结构图的组成 …… 108
3.2.2　软件结构图的画法 …… 109
3.3　面向对象的设计方法 …… 115
3.3.1　面向对象设计过程 …… 115
3.3.2　面向对象设计方法 …… 118
3.4　其他设计方法介绍 …… 130
3.4.1　面向方面程序设计 …… 130
3.4.2　面向 Agent 的设计方法 …… 131
3.4.3　泛型程序设计 …… 131
3.4.4　面向构件的技术 …… 132
3.4.5　敏捷方法 …… 132
3.4.6　Rational 统一过程 …… 132

3.4.7 功能驱动开发模式 FDD …… 132
3.4.8 极端编程 …… 133
3.5 实用案例 …… 133
3.5.1 SafeHome 软件的结构设计 …… 133
3.5.2 基于 UML 的网络管理平台的分析与设计 …… 136
3.6 小结 …… 140
3.7 习题 …… 141
第 4 章 软件的编程实现 …… 147
4.1 编程语言的选择和分类 …… 147
4.1.1 程序设计语言的分类 …… 147
4.1.2 机器语言 …… 148
4.1.3 汇编语言 …… 150
4.1.4 高级语言 …… 150
4.1.5 非过程语言 …… 151
4.2 编程要求 …… 151
4.2.1 程序语句结构的构成原则 …… 152
4.2.2 程序可读性和易理解性的要求 …… 153
4.2.3 数据说明的要求 …… 155
4.2.4 输入和输出应遵守的原则 …… 155
4.3 面向对象的编程语言介绍 …… 156
4.3.1 Smalltalk 语言 …… 156
4.3.2 Eiffel 语言 …… 156
4.3.3 C++语言 …… 156
4.3.4 Delphi 语言 …… 157
4.3.5 Java 语言 …… 158
4.3.6 C#语言 …… 158
4.4 案例：网上招聘系统软件编程规范 …… 158
4.4.1 编码格式规范 …… 158
4.4.2 编码命名规范 …… 160
4.4.3 程序中的声明规范 …… 161
4.4.4 编程语句规范 …… 162
4.4.5 代码注释规范 …… 162
4.4.6 存放代码的目录规范 …… 164
4.5 小结 …… 166
4.6 习题 …… 166
第 5 章 软件的测试及维护 …… 168
5.1 软件测试的基本原理 …… 168
5.1.1 软件测试目标和原则 …… 168
5.1.2 软件的可测试性 …… 169
5.1.3 软件测试的步骤和策略 …… 169
5.2 测试用例设计 …… 179
5.2.1 白盒测试 …… 179
5.2.2 黑盒测试 …… 183
5.2.3 人工测试 …… 185
5.2.4 调试 …… 186
5.3 面向对象的测试 …… 188
5.3.1 测试策略 …… 188
5.3.2 类测试方法 …… 189
5.4 软件维护的基本原理 …… 193
5.4.1 软件维护的分类 …… 193
5.4.2 影响软件可维护性的特性 …… 195
5.4.3 提高软件可维护性的方法 …… 198
5.5 软件再工程 …… 201
5.5.1 业务过程再工程模型 …… 201
5.5.2 软件再工程过程模型 …… 202
5.5.3 软件的逆向工程 …… 203
5.5.4 软件重构 …… 205
5.5.5 软件的正向工程 …… 205
5.5.6 软件再工程的成本-效益分析 …… 206
5.6 案例：微软公司的软件测试 …… 207
5.6.1 软件测试人员的组成及任务 …… 207
5.6.2 软件测试应考虑的问题 …… 209
5.6.3 对软件中 bug 的处理 …… 209
5.6.4 采用的软件测试方法及测试工具 …… 210
5.6.5 主要的软件测试文档 …… 212
5.7 小结 …… 213
5.8 习题 …… 215
第 6 章 软件项目管理 …… 218
6.1 软件范围的确定 …… 218
6.1.1 获取确定软件范围所需的信息 …… 218
6.1.2 软件范围的具体内容及描述要求 …… 219
6.2 软件资源的考虑 …… 219
6.2.1 人力资源 …… 220
6.2.2 可复用软件资源 …… 220
6.2.3 环境资源 …… 221
6.3 软件成本的估算 …… 221

6.3.1 软件估算模型……222
6.3.2 软件估算方法……232
6.3.3 面向对象软件项目的估算……236
6.3.4 软件自行开发或购买的决策……237
6.4 开发进度的安排……238
6.4.1 使用甘特图安排软件工程项目进度……238
6.4.2 使用 PERT 图安排软件工程项目进度……240
6.4.3 两种图相结合安排软件工程项目进度……242
6.5 案例：会计信息系统软件成本估算……242
6.5.1 根据已知条件选择估算方法或模型……243
6.5.2 根据选定的方法或模型进行计算……243
6.5.3 使用其他方法或模型进行交叉检验……244
6.6 小结……244
6.7 习题……245
第 7 章 软件工程项目案例……251
7.1 美国航空航天局的太空机器人系统……251
7.1.1 以用户为中心的迭代设计过程……251
7.1.2 面对的技术挑战……252
7.1.3 太空机器人系统的测试……252
7.2 乐视网 TV 大数据平台……253
7.2.1 问题的提出……253
7.2.2 平台构建过程及采用的工具……253
7.3 有道云笔记云端架构……257
7.3.1 数据结构的设计……257
7.3.2 服务框架的设计……258
7.3.3 架构的重构和升级……258
7.4 网易广告投放系统……259
7.4.1 系统设计目标……260
7.4.2 迭代开发过程……260
7.4.3 实现细节考虑……261
7.5 富士通的基于 ZooKeeper 和 Storm 的车载流式计算框架……262
7.5.1 项目背景介绍……262
7.5.2 软件设计策略……262
7.5.3 案例成功要点……263
7.6 支付宝无线统一测试平台……264
7.6.1 研发模式及实践经验……264
7.6.2 测试平台的模块结构介绍……265
7.6.3 核心模块的功能描述……266
7.7 习题……268
第 8 章 软件工程课程设计……269
8.1 基本要求及考核标准……269
8.1.1 课程设计的内容……269
8.1.2 课程设计的考核……303
8.2 交付文档要求及格式……307
8.2.1 《软件计划》编写格式及要点……308
8.2.2 《需求规约说明书(SRS)》编写格式及要点……309
8.2.3 《软件设计说明书》编写格式及要点……309
8.2.4 《软件测试计划》编写格式及要点……310
8.2.5 《软件测试分析报告》编写格式及要点……310
8.2.6 《开发进度报告》编写格式及要点……311
8.2.7 《用户手册》编写格式及要点……311
8.2.8 《操作手册》编写格式及要点……312
8.2.9 《软件开发总结报告》编写格式及要点……312
参考文献……313

第 1 章　软件工程概述

现在是电子信息时代，大街上的年轻人头上戴着耳机听着什么匆匆走过；咖啡厅、地铁、医院、餐厅、学校、排队等候办事的人群中、甚至公交车上，不少人都在低着头用手机上网、看微信或玩游戏；当我们坐在家里玩电脑或上网查资料的时候，当我们到派出所办理户口或身份证的时候，当我们到医院检查身体或看病的时候，当我们到银行办理业务的时候，当我们去车站或机场购票的时候，等等这一切的一切，是依靠什么顺畅运行的呢？答案是软件！软件是什么，它是怎么发展起来的，没有它行不行，软件工程又是什么呢？本章内容将会回答你这些问题。

1.1　软件的分类和演化

软件工程是围绕软件的开发、维护、管理、质量保证等一系列活动而展开的，本节将介绍什么是软件，它是如何分类，又是如何发展的。

1.1.1　软件的起源和分类

1．软件起源

软件一词起源于程序，是 20 世纪 60 年代出现的概念，也是一个不断发展的概念，软件的形成和发展与计算学科的发展紧密相关，现在计算学科已经发展为计算机科学与技术以及软件工程两大学科。不同时期对软件的解释也有所不同，这是由于软件的功能、模块、开发方式以及使用方式在不断发生变化，因而对它的解释也在发展变化。

2．软件定义

软件是相对于硬件而言的，以解释性的定义为主。

美国软件工程专家罗杰·普雷斯曼（Roger S. Pressman）（如图 1-1 所示）将软件解释为当机器执行时能提供所要求功能和性能的程序，能使程序有效处理信息的数据结构以及描述操作和使用程序的文档，简单地说就是软件由程序、数据和文档组成。

电子与电气工程师协会（IEEE）将软件定义为计算机程序、方法、规则、相关文档及在计算机中运行时所需的数据。

英国软件工程专家兰·萨默维尔（Ian Sommerville）（如图 1-2 所示）认为软件是一个系统，通常由若干程序、用于建立这些程序的配置文件、描述系统结构的系统文档、解释如何使用系统的用户文档以及供用户下载最新产品信息的 Web 站点组成。

一些学者认为软件由程序及其文档组成，其中，程序是计算机任务的处理对象和处理规则的描述，文档是为了理解程序所需的阐述性资料。

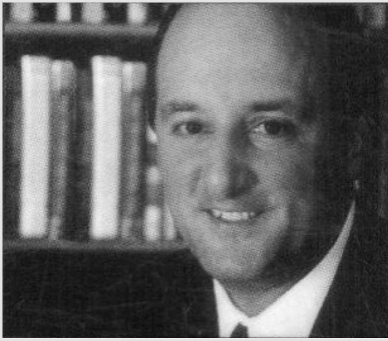


图 1-1　罗杰 • 普雷斯曼

图 1-2　兰 • 萨默维尔

不难理解，软件其实是知识的载体，它包括的内容和范围很广，一般认为执行指令的计算机程序以及与之相关的文档、数据、影视资料、方法、规则、网页及其链接等都可算作软件。

3．软件分类

从软件的起源和定义可以看出，软件具有 3 层含义：一是个体含义，指计算机系统中的程序、文档和数据；二是整体含义，指在特定计算机系统中所有上述个体含义下的软件的总称；三是学科含义，指在研究、开发、维护以及使用前述含义下的软件所涉及的理论、方法、技术所构成的学科。

软件的种类很多，随着复杂程度的增加，软件的分类也在发展变化，界限越来越不明显，按软件的作用，可分为以下 4 类。

（1）系统软件

是指由生产厂家配置，服务于其他系统元素的软件。如操作系统、汇编程序、编译程序、数据库管理系统、计算机通信及网络软件等。如果没有这些软件，计算机、网络、设备等很难发挥功能，甚至无法工作。

（2）应用软件

是指在系统软件的基础上，为解决特定领域问题而开发的软件。这类软件很多，如用于电信、金融、电力、公安、交通管理、招生、考试、录取等领域的专用软件，企事业单位生产、工作、管理、服务的各种事务类软件，监视、分析和控制正在发生的现实世界事件的各种实时软件，各类科学和工程软件，用于工业、民用或军事上的各种功能的与设备融为一体的嵌入式软件，个人计算机软件，手机上的各种实用软件，基于 Web 的软件，儿童玩具中的软件，人工智能软件，等等。

（3）工具软件

用于辅助和支持开发及维护的应用软件，以提高软件开发质量和生产率的软件都可称为工具软件。如软件进度安排软件、软件成本估算工具、软件需求分析工具、软件设计工具、软件编码工具、测试工具、系统维护工具、管理工具等，这些软件工具不仅能完成指定的工作，还能进行完备性、一致性、正确性、安全性、追溯性等方面的检查，完成一些人工所不

能完成的事情。

（4）可重用软件

是指通过修改、剪裁等手段应用于新软件的软件，可重用软件并不仅指代码，也可以是软件设计、规范、数据、测试用例，甚至概念等，可以把它们统称为可重用软件构件，目前，常见的各种图形用户界面一般是使用可重用软件构件创建的，这些构件涉及图形窗口、下拉菜单和各种交互机制。建造用户界面所需要的数据结构和处理细节包含在一个由界面构件所组成的可重用构件库中，开发人员通过可重用构件来开发新的软件。

若按功能对软件进行分类，则可划分为系统软件、支撑软件和应用软件三大类；若按规模进行划分，可将软件分为 7 类，如表 1-1 所示；若按软件工作方式进行划分，则可分为实时处理软件、分时软件、交互式软件和批处理软件 4 类；若按软件服务对象的范围进行划分，则可分为项目软件和产品软件 2 类；若按软件的应用领域进行划分也可将软件分为 7 类，如表 1-2 所示；还可按其他方式对软件进行划分，如按软件的使用频度进行划分、按软件失效的影响进行划分等。

表 1-1　按规模对软件的分类

类别	参加人员数	研制期限	产品规模(源程序行数)
微型	1	1～4 周	0.5 千行
小型	1	1～6 月	1～2 千行
中型	2～5	1～2 年	5～50 千行
大型	5～20	2～3 年	50～100 千行
甚大型	100～1000	4～5 年	1 兆行（=1000 千行）
极大型	2000～5000	5～10 年	1 兆～10 兆行
超大型	具有互联网规模	长期、持续演化	10 兆～10 亿行或更大

表 1-2　按应用领域对软件的分类

类别	用　　途	特　　点	例　　子
系统软件	一整套服务于其他程序的软件	和计算机硬件大量交互；多用户大量使用；需要调度、资源共享和复杂进程管理的同步操作；复杂的数据结构以及多种外部接口	编译器、编辑器、文件管理软件、操作系统构件、驱动程序、网络软件、远程通信处理器等
应用软件	解决特定业务需要的独立应用软件	处理商务或技术数据，以协助业务操作、管理或技术决策	传统数据处理应用程序、超市交易处理软件、实时制造过程控制软件等
工程和科学软件	具有明显“数值计算”算法的软件	不仅仅局限于传统的数值算法，有些工程和科学软件甚至具有系统软件的特征	从天文学到火山学、从自动应力分析到航天飞机轨道动力学、从分子生物学到自动制造业，几乎涵盖了所有的应用领域
嵌入式软件	存在于某个产品或系统中，可实现和控制面向最终使用者和系统本身的特性和功能的软件	可以执行有限但难于实现的功能，提供重要的功能和控制能力	导弹、火箭等武器控制系统，汽车中的燃油控制、仪表板显示、刹车系统等汽车电子功能，各种电子产品、医疗设备，微波炉、洗衣机、电冰箱的按键控制等
产品线软件	为多个不同用户的使用提供特定功能的软件	关注有限的特定专业市场及大众消费品市场	库存产品控制软件、文字处理软件、电子制表软件、电脑绘图软件、多媒体工具软件、娱乐软件、数据库管理软件、个人及公司财务应用软件
Web 应用软件	是一类以网络为中心的软件	涵盖了宽泛的应用领域	可以是一组超文本链接文件，也可以是复杂的计算环境
人工智能软件	应用于专门场合或领域	利用非数值算法解决计算和直接分析无法解决的复杂问题	机器人、专家系统、模式识别、人工神经网络、定理证明和博弈等

软件产品包括两类，通用软件产品和定制软件产品。它们的区别在于，通用软件产品的描述由开发者自己完成，而定制软件产品的描述通常由客户给出，开发者必须按客户要求进行开发。因而，各类数据库软件、字处理软件、绘图软件、工程管理软件等属于通用软件产品，而特定的业务处理系统、电子设备的控制系统、空中交通管制系统等属于定制软件产品。

1.1.2 软件工程的产生和发展

1．软件工程的产生

20 世纪 40 年代中期，美国宾夕法尼亚大学约翰・莫克莱（John Mauchly）和普雷斯帕・埃克特（J.Presper Eckert）（见图 1-3）研制出世界上第一台电子数字计算机 ENIAC。1947 年，数学家冯・诺依曼（J.Von Neumann，如图 1-4 所示）针对 ENIAC 提出了 EDVAC（Electronic Discrete Variable Automatic Computer）方案，在该方案中首次提出了“存储程序”的概念，图 1-5 所示是 1951 年实现的 EDVAC。1949 年，由英国皇家科学院院士莫里斯・威尔克斯制造的世界上第一台实现冯・诺依曼“存储程序”思想的计算机 EDSAC（Electronic Delay Storage Automatic Calculator）问世。从冯・诺依曼提出存储程序概念至今的时间中，软件发生了很大变化，其发展可用表 1-3 概括。

图 1-3　莫克莱和埃克特

图 1-4　冯・诺依曼

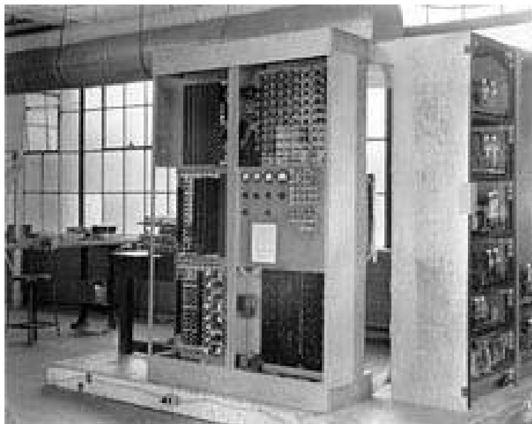


图 1-5　EDVAC（1951）

表 1-3　软件的演化历程及特征

阶段划分	阶段名称	时间	对软件的解释	软件开发方法	决定软件质量的因素	硬件特征	软件特征及技术
第一阶段	程序设计阶段	20 世纪 50 年代	程序	个人	个人程序设计技术	价格昂贵，存储容量小，工作可靠性差	软件不受重视
第二阶段	程序系统阶段	20 世纪 60 年代	程序、说明文档	“软件作坊”式的小组	小组技术水平	降价，速度、存储容量及工作可靠性明显提高	多用户、实时、数据库、软件产品；软件技术的发展不能满足需要，出现了软件危机
第三阶段	软件工程阶段	20 世纪 70 年代至 80 年代中期	程序、文档、数据	开发小组及大、中、小型软件开发机构	管理水平	向超高速、大容量、微型化发展	分布式系统、嵌入式软件出现；软件危机并未摆脱
第四阶段	现代软件工程阶段	20 世纪 80 年代末至现在	程序、文档、数据、Web 页、方法、规则	中间件技术、网络技术、构件技术、管理技术的应用使跨平台、跨区域开发成为可能，超大型软件工厂出现	管理水平	网络化发展迅速，大容量新型高速存储器问世，运算速度达每秒几万亿次至万万亿次的巨型计算机不断出现	强大的桌面系统、面向对象技术、专家系统、人工神经网络、并行计算、客户/服务器环境、面向 Agent 软件工程、网构软件技术、面向自适应和自组织系统的软件工程、面向组织的软件工程、大数据、云计算、“互联网+”等；软件危机仍然存在

从表 1-3 可以看出，在 20 世纪 60 年代由于软件的发展不能满足需求出现了软件危机，所谓软件危机是指在计算机软件开发和维护过程中所遇到的一系列问题，这些问题几乎存在于所有的软件中。

软件危机主要表现为以下 6 个方面。

（1）软件开发成本和进度估计不准确

据统计，软件项目约有 40%延期或超预算，20%不得不取消，真正成功的软件项目案例不足 40%。实际成本比估计成本高出一个数量级，实际进度比预期进度拖延几个月，甚至几年的现象经常发生。例如，微软公司的 Word 3.0 原定于 1986 年 7 月推出，但是直到 1987 年 2 月才问世，但问世后很快就发现有很多错误，据说有 700 多处错误，有的错误甚至会破坏数据，摧毁程序，微软不得不花费 100 万美元在两个月内为用户免费升级。

（2）软件需求分析不充分，开发出的系统与实际需求有差距

软件开发人员和用户之间的信息交流往往很不充分，在没有确切了解问题的情况下，或对用户需求还很模糊的情况下就开始编写程序，导致开发出的软件产品与实际需求有差距，甚至不符合用户的实际需要。在进行软件需求分析时，不但要考虑功能需求，还要考虑质量需求和约束，既要有产品需求，还要有市场需求和组件需求，如果缺少必要的需求，或得到错误的需求，或不断变更的需求，都会最终导致系统失败。例如，哈维兰彗星 1 号客机，由于没有考虑到方形窗口的承压能力而经常坠毁；塔科马海峡大桥在需求和设计模型中没有考虑风力因素而被大风吹垮；Ariane 5 型火箭不恰当地复用了需求，导致在第一次试航中坠毁。据克里斯托夫·埃伯特（Christof Ebert）（如图 1-6 所示）所著《需求工程：实践者之路》一书中给出的数据，87%的项目失败是由于需求工程做得不够而引起的。

（3）软件投入使用后经常出故障

一个著名的例子是美国 IBM 公司的 OS/360，这是第一个功能较强的多道程序操作系统，参加这项开发工作的有 IBM 公司美国国内的 11 个单位，欧洲的 6 个单位，共计 1000 多人，耗费了 5000 人年的工作量，但结果不尽人意，它的每个版本都是在前一版本的基础上找出 1000 个程序错误而修正的结果，项目总负责人费雷德里克·布鲁克斯（Frederick

P.Brooks　Jr.，见图 1-7）在他所著的《人月神话》（The Mythical Man-Month）一书中生动地描述了开发中遇到的困难和问题，提出了很多令人深思的观点。

图 1-6　克里斯托夫 • 埃伯特

图 1-7　费雷德里克 • 布鲁克斯

（4）软件难以维护

软件的维护不像硬件那样通过进行零部件的修复、更换或系统整体升级就能做到，修改了发现的错误后可能还会导致新的错误，更换后的软件也不一定比旧软件好。例如，美国第四大药品批发商福克斯迈尔医药公司在 1994 年决定用 C/S 系统和 SAP 软件代替老化的 Unisys 大型系统，该公司每天要填写成千上万份药房订单，每个药房又订购上千种药品，总量达 50 万种，与系统升级和改造相关的费用就花费了 3800 万美元。但在系统投入实际使用中却发现 SAP 软件每天只能处理几千条数据，在短短几个星期内，由于发出错误订单这一项，就使公司遭受了 1550 万美元的损失，最后迈基森公司为其支付了 8000 万美元现金，才使其子公司没被破产，总公司的股票也从 26 元跌到 3 元，使公司蒙受了巨大损失。

很多程序中的错误很难改正，让一个软件系统适应新的硬件环境，或在原有程序中增加一些新功能都非常困难，有时为了纠正软件的 1 个错误的工作量甚至可能要超出原来软件开发的总工作量。

（5）缺乏文档资料

在软件开发过程中应该有一套与之同步生成的文档资料，它们是软件开发过程的“里程碑”，若缺乏文档资料或事后再补都会给开发及维护带来困难。在计算机诞生的早期，软件规模比较小，应用领域狭窄，文档的作用并不十分突出，但是随着计算机应用领域的扩大及软件技术的发展，特别是软件工程的出现，软件文档的地位和作用也越来越重要，软件文档在软件产品开发过程中起到了重要的桥梁作用。例如，20 世纪 70 年代初，汽车中只有几个控制器，需求规格说明书加在一起也不过上百页，而现代汽车中有 50 多个控制器，需求规格说明书达到了 10 万页。目前，软件文档已经成为项目管理的依据、软件人员进行技术交流的语言和依据、也是进行项目质量审查和评价的重要依据，它们为用户和技术人员提供培训和维护的资料、为维护软件提供技术支持，它们还是未来项目的一种资源，良好的系统文档有助于把程序移植和转移到各种新的系统环境中。

（6）软件在计算机总成本中所占比例逐年上升，维护费用增长迅速

在计算机系统诞生的早期，硬件的价格决定了系统的整体价格，但是随着硬件技术的快速发展及价格的不断下降，硬件与软件在整个计算机系统中的比重快速发生着变化，到 20 世纪 60 年代它们的成本比例就发生了颠倒，软件成本成为主体，例如，某发电厂花几百万元购买的数据采集与监控系统（简称 SCADA 系统），其中的 90%都是软件费用。现在硬件已经成为软件的"包装"，软件的成本就决定了整个系统的价格。据美国军方给出的数据，在 20 世纪 60 年代的 F-4 战斗机中，由软件来完成的功能约为整体功能的 8%，到 90 年代，在 B-2 轰炸机中，由软件来完成的功能就占到整体功能的 65%，而在 21 世纪，在 F-22 战斗机中，由软件来完成的功能占到整体功能的 80%，由此可看出软件已居主导。目前，软件费用已经成为计算机系统的主要费用，其中维护费用又占去了大部分，有些软件机构 70%的工作量都花在维护已有的软件上。

随着计算技术的发展，软件危机的表现形式也在发生变化，旧的危机克服了，还会出现新的危机。为了摆脱日益严重的软件危机，1968 年在德国小镇加米施（Garmisch）召开的北大西洋公约组织（简称 NATO）学术会议上，与会学者们首次提出了软件工程的概念，图 1-8 所示是当时的会场，这次会议是软件开发走向工程化的标志，从此软件技术的发展及软件工程的研究有了长足的进步，并起着越来越大的作用。目前，软件工程已经成为一门大的学科，在它下面又出现了很多新的学科，需要研究的未知领域和课题还很多，但软件危机并未完全解决。

图 1-8　1968 年在 NATO 学术会议上首次提出了软件工程的概念

2．软件工程的定义

软件工程既是工程，又是一门学科，软件工程学科的内容很丰富，定义也是多种多样，最早的定义是弗里斯·鲍尔（Frith Bauer）在 NATO 学术会议上给出的，他指出软件工程是建立和使用一套合理的工程原则，以便经济地获得可靠的、可以在实际机器上高效运行的软件。

IEEE 给出的软件工程定义是：软件工程首先是将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件；其次是与上述有关方法的研究。

美国国家工程院院士、著名的软件工程专家巴利·玻姆（Barry W.Boehm）（如图 1-9 所

示）给出的定义：软件工程是现代科学技术知识在设计和构造计算机程序中的实际应用，其中包括管理在开发、运行和维护这些程序的过程中所必需的相关文档资料。

美国软件工程专家 Roger S. Pressman 把软件工程视作一种层次化技术，软件工程的任何活动都必须建立在高质量的基础之上，支持软件工程的根基就在于高质量，他的层次技术思想可以用图 1-10 来描述。

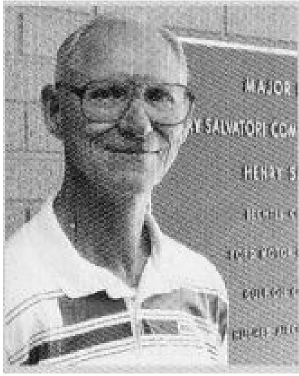


图 1-9　巴利·玻姆

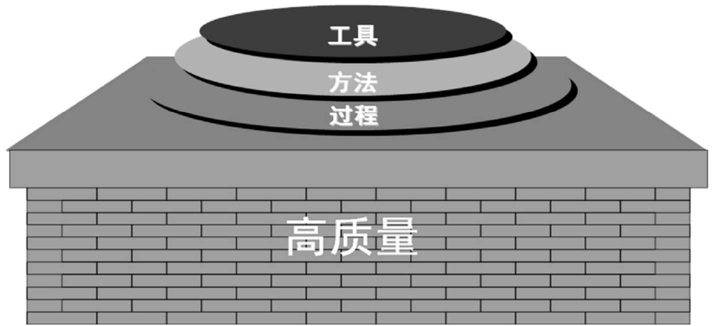


图 1-10　软件工程层次图

从图 1-10 可以看出，软件工程层次结构的基础是过程层，过程层定义了一个框架，构建该框架是有效实施软件工程技术必不可少的。软件过程构成了软件项目管理控制的基础，在软件过程中要建立工作环境以便于应用技术方法、提交工作产品（如模型、文档、数据、报告、表格等）、建立里程碑、保证质量及正确管理变更。

软件工程方法为构建软件提供技术上的解决方法，主要包括沟通、需求分析、设计建模、编程、测试和技术支持。软件工程技术方法依赖于一组基本原则，这些原则涵盖了软件工程所有技术领域，包括建模和其他描述性技术等。

软件工程工具为过程和方法提供自动化或半自动化的支持。这些工具可以集成起来，使得一个工具产生的信息可被另外一个工具使用，这样就建立了软件开发的支撑系统，称为计算机辅助软件工程。

北京大学王立福教授等给出的软件工程定义：软件工程是一类求解软件的工程。它应用计算机科学、数学及管理科学等原理，借鉴传统工程的原则、方法，创建软件以达到提高质量、降低成本的目的。其中，计算机科学、数学用于构造模型与算法，工程科学用于制定规范、设计泛型、评估成本及确定权衡，管理科学用于计划、资源、质量、成本等管理。软件工程是一门指导计算机软件开发和维护的工程学科。

从以上软件工程的定义可以看出，软件工程包含的内容很丰富，它涉及软件开发、管理、维护、质量保证等各个方面，它既有一般工程的特点，也有其特殊性，它不但和软件相关，也和其他学科相关，软件工程是一门多学科交叉的学科。

1.2　软件的生命周期

软件和任何有生命或无生命的事务一样，有它的生命周期（或称为生存周期），一个人

的生命周期是从孕育、出生，经过乳幼期、少年期、青年期、壮年期、老年期直到死亡，如图 1-11 所示，各个阶段的投入、目标、活动显然都是不一样的，在人的生命周期中对社会的贡献主要在成年期。软件与人的生命周期在某些方面很类似，一个软件的生命周期是指从制定软件计划开始，经过软件需求分析、软件设计、编码、软件测试、运行和维护直到软件被弃的全过程，如图 1-12 所示，软件的价值体现在运行和维护阶段，甚至价值曲线的形状也与人对社会的贡献曲线类似，呈“U”字形。

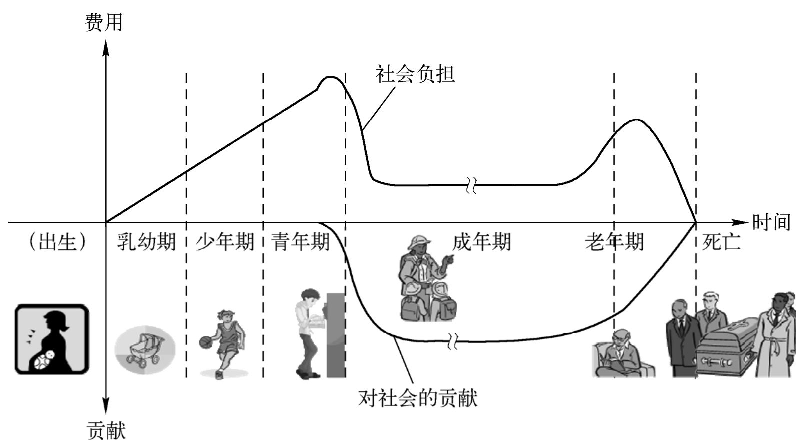


图 1-11　人的生命周期

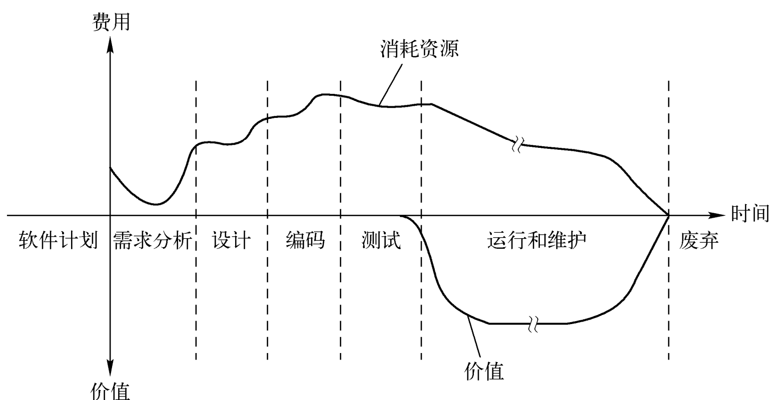


图 1-12　软件的生命周期

软件的生命周期各阶段是从宏观上对软件的划分，当然也可以粗略地划分为计划、开发和维护 3 个阶段。与人的生命周期不同的是，软件的生命周期与开发模型有关，不同的开发模型可能对应着不同的生命周期。生命周期不同，则软件开发阶段的划分、评审次数及基线标准都有所不同。

1.2.1　计划阶段

软件计划阶段相当于婴儿在母腹中的孕育时期，这个阶段的主要工作有五项，一是确定待开发软件系统的工作范围，主要是给出软件的功能、性能、约束、接口和可靠性等方面的

要求；二是预测开发该软件所需要的人力资源、可重用软件资源，以及软件和硬件工具；三是使用估算技术和估算模型对所开发软件的成本进行估算，也包括开发进度、开发工作量等的估算；四是对软件风险进行管理，包括风险的识别、预测、缓解和监测等，风险是对未来不确定性的预测，风险既可能是由人员引起的，也可能是技术、成本方面的原因，还可能是时间方面的原因，不同风险的影响程度也是不一样的，有些风险影响程度较小，甚至可以忽略，而有些风险可能很严重，甚至造成灾难性的后果，因而，风险也是分等级的，只有对风险进行了有效的评估，才有可能有效的管理和控制风险；五是对软件开发进度进行安排，它类似于铁路列车时刻表的编制，如果不按照规定的时间运行，造成的后果是不言而喻的，软件开发进度安排也是如此，关键时间点不能变，否则软件的交付就会延期。软件计划阶段工作结束后，要交付软件项目开发计划，该计划是一个里程碑性质的文件，上述五项是其中的核心内容，书写时应按有关标准和格式进行，经过审查通过后，才能开始进入到正式的软件开发。软件项目计划审查通过后相当于婴儿的诞生，把一个婴儿培养成对社会有用的人，还需要做很多工作。

1.2.2 分析和定义阶段

这一阶段即软件生命周期中的需求分析阶段，需求分析是软件生命周期的重要阶段，目标是深入描述软件的功能和性能，确定软件设计的约束、软件同其他系统元素的接口细节，定义软件的其他有效性需求。这一阶段的工作对软件产品的成功起到很大的作用，据统计，软件项目失败的原因 87%是由于需求分析做得不够而引起的，由于需求分析工作的地位和作用十分重要，因而诞生了一门学科——需求工程，需求工程虽然是软件工程的一部分，但也是一门独立的学科。需求工程是为了系统性地、规范地进行需求获取、需求编写、需求分析、需求协商、需求审查以及需求管理，使顾客对软件的期望以及软件达到的目标在一个产品中实现。软件需求规格说明（Software Reqaiements Specification，SRS）是分析阶段的最终产物，是软件工程过程中的里程碑式的文档，因而是需求分析阶段最重要的文档。俗话说“磨刀不误砍柴工”，需求工程就是在“磨刀”，如果需求工作不充分，就像拿钝斧子砍树，并不能加快软件的交付，反而会更慢。

软件需求分析方法主要有结构化方法、原型化方法和面向对象方法。结构化分析是面向数据流进行需求分析的方法，它从宏观到具体采用自顶向下的方式描述现实问题，经过一系列的分解和抽象，得到的底层数据流图就是描述具体问题的解决方案，用结构化分析方法产生的 SRS 中主要包括数据流图、数据字典、E-R 图以及状态图等。

原型化开发过程有抛弃式、演化式和增量式，主要是针对软件需求很模糊的情况下，通过不断地构造原型，以迭代的方式逐步搞清楚用户的需求。主要原型开发技术有三种：使用动态高级语言、数据库编程和组件复用，原型开发技术对用户界面的设计和实现是一种有效的方法。原型化开发软件是一个不断演化的过程，初始原型通常质量不高，因此，用这种方法开发软件时要废弃掉很多中间原型，而且一定要以质量为第一目标，SRS 中的前述内容仍然需要，也可采用迭代方式在软件迭代的过程中同步产生 SRS，并且要描述原型内容。

面向对象分析是利用面向对象的概念和方法构建软件需求模型，如用统一建模语言（Unified Modeling Language，UML）的用例图定义功能及交互活动的关键步骤、用活动图来说明场景、用泳道图来显示如何给不同的参与者或类分配处理流。面向对象的方法关注对象

的内在性质，以及对象的关系与行为。在开始构建系统之前，必须定义出表示待解决问题的类（对象）、类之间的相互关联和交互的方式、对象的内部结构（即属性和操作）以及允许对象在一起工作的通信机制（消息）。所有这些事情均是在面向对象分析中完成的。在面向对象的方法中，UML 是目前流行的建模工具，可以用它定义类的层次、关系、关联、聚合和依赖等，UML 适用于系统开发的全过程，从需求规格描述直到系统建成后的测试和维护阶段。此外，还可以用 CRC（Class-Responsibility-Collaborafor）索引卡的方式定义类之间的联系。上述建模过程及所建模型都应在 SRS 中体现出来。

1.2.3 设计阶段

软件设计阶段是软件开发中的关键阶段，它比编码工作重要得多。在设计过程中需要软件开发者付出创造性的劳动。软件生命周期中的上一阶段——需求分析阶段，主要解决的是需要让所开发的软件“做什么”的问题，并且已在 SRS 中详尽和充分地进行了描述。进入设计阶段后，应该解决“怎么做”的问题。简单地说，软件设计就是以 SRS 为基础，把确定的软件需求转换成相应的体系结构，对系统结构中的每个成分的具体工作进行描述，编写出设计说明书，提交有关部门评审。

软件设计可以分为两个阶段，概要设计和详细设计，概要设计主要是针对软件体系结构的设计，详细设计则把重点放在对软件过程的描述上，它为下一阶段实现系统奠定基础。

软件体系结构是构造系统的基本框架，就像楼房的架构一样，不同的楼房按用途其构造会完全不同，软件的构造也是如此，不同的软件应用适合于不同的体系结构，因而有不同的体系结构模型，常见的体系结构模型有以数据为中心的黑板模型、以变换数据的构件为主而构成的管道过滤器模型、以程序构件为主要构成成分的返回和调用体系结构模型、以信息隐蔽原理构造的面向对象体系结构模型、按每个层次各自完成不同操作形成的层次体系结构模型，等等，理想化的、所有软件系统皆适用的体系结构模型是没有的。

软件体系结构中各个元素之间都不是孤立的，因而要研究各个成分之间以及成分内部的联系，评价软件结构质量的度量标准是成分之间的耦合性和内聚性，这是一个问题的两个方面。以返回和调用体系结构为例，它的成分一般是程序模块，模块之间的耦合性从强到弱包括内容耦合、公共耦合、外部耦合、控制耦合、标记耦合、数据耦合和非直接耦合 7 种，内容耦合最差、非直接耦合最好，内聚性则是衡量一个成分内部能力的一种度量，还以返回和调用体系结构为例，模块的强度从弱到强包括偶发强度、逻辑强度、时间强度、过程强度、通信强度、顺序强度和功能强度 7 种，偶发强度的模块最差，功能强度的模块最好，软件设计追求的目标是强内聚、弱耦合。

软件设计方法也有多种，如面向数据流的设计方法、面向数据结构的设计方法、面向对象的设计方法、面向方面的设计方法、面向 Agent 的设计方法、面向构件的设计方法、敏捷方法等等，有些方法是成熟的，有些还在发展中。

面向数据流的设计方法也称为结构化设计方法，它是根据问题域的数据流（数据对象）定义一组不同的映射，把问题域的数据流（数据对象）转换为问题解的程序结构。结构化设计方法的要点是“自顶而下，逐步求精”。简单地说就是先构造高层的结构，然后再逐层分解和细化，从宏观到具体设计软件，避免一开始就陷入复杂的细节中，设计中模块可作为插件或积木使用，这样就降低了程序的复杂性，提高了可靠性。

面向数据结构的设计方法则是根据问题的数据结构定义一组映射，以问题的数据结构为基础转换为问题解的程序结构。在面向数据结构的设计方法中，Jackson（1975 年，M.A.Jackson 提出）方法和 LCP（Logically Constracting Program）方法是最具代表性的方法，Jackson 方法简单、易学易用，对于规模不大的数据处理系统非常适用，在了解了系统需处理的数据结构后，如果能找到输入和输出数据结构之间的对应性，其程序结构很容易导出。LCP 方法本质上和 Jackson 方法是一样的，但它是更严格的一种软件设计方法，是建立在数据结构和程序过程结构关系上的方法。在软件生命周期中，数据结构往往会发生变化，一旦数据结构变化了，以数据结构为基础建立的整个程序结构也就需要改变。因而，对于复杂的、比较大的软件系统、数据结构易变化的系统都不适合用面向数据结构的设计方法。

面向对象的设计方法已经成为主流软件设计方法，它以对象为中心，软件中的任何元素都是对象，复杂的软件对象是由比较简单的对象组合而成。在面向对象的设计方法中，把所有对象都划分成各种对象类（简称为类，class），每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性，是对象的状态信息。每当建立该对象类的一个新实例时，就按类中对数据的定义为这个新对象生成一组未用的数据，以便描述该对象独特的属性值。按照子类与父类的关系，把若干个对象类组成一个层次结构的系统。在这种层次结构中，下层的派生类具有和上层的基类相同的特性（包括数据和方法），这种现象称为继承。如果在派生类中对某些特性又做了重新描述，则低层的特性将屏蔽高层的同名特性。对象彼此之间仅能通过传递消息互相联系。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，它是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，它的属性和操作对外界都是隐蔽的，所以不能从外界直接对它的私有数据进行操作，这种灵活的消息传递方式，便于体现并行和分布结构。

以上方法都有一定的局限性，因而人们还在研究新的方法，代表性的方法如面向方面的设计方法、面向 Agent 的设计方法、泛型程序设计、面向构件的设计方法、敏捷方法、Rational 统一过程、FDD 方法、XP 方法等，这些方法目前还不成熟。

软件设计结束后，要交付里程碑式的文档，大中型以上规模的软件系统既要交出概要设计说明书又要交出详细设计说明书，甚大型软件系统还要交出数据库设计说明书，系统越大、越复杂要求交付的文档越多，小型软件系统只需交付一个文档，即交出软件设计说明书即可。软件设计说明书是重要文档，完成后要组织评审。

1.2.4 实现阶段

这一阶段是根据软件详细设计说明书，编程实现系统，并进行测试，所以也称为编码阶段，简单地说，就是将软件设计转换成计算机程序代码，并编写测试用例，对模块进行测试。

不同的程序设计语言适用于不同的应用领域，选择合适的语言可提高编程效率和提高程序的质量，但十全十美的语言是不存在的，一般应选择尽可能熟悉的、能较好满足应用领域要求的语言。对于成熟的软件企业由于积累了丰富的管理经验，过程管理到位，软件的构件库、类库以及各种资源丰富，软件的实现就相对容易。

计算机程序设计语言有成百上千种，分类方法也不太一样。从时间上看计算机语言的发展大致经历了四代，第一代是机器语言时代，第二代是汇编语言时代，第三代是高级语言时代，目前主要使用的是高级语言，高级语言是过程性语言，第四代语言是非过程性的，第四

代语言的发展将逐步改变程序设计的方式。在实现软件时要重视编码风格，因为它直接影响到软件的可维护性和易理解性，编码风格就像作家的写作风格，好的作品不但给读者以享受，而且会影响到读者的思想和行为，程序虽然最终要转换成机器去执行的指令，但更重要的是给人看的，因此要重视编码风格。好的编码风格对软件企业的发展、对软件过程管理水平的提高以及对软件企业文化的形成都有很大的影响。

程序设计也是艺术，编程人员不但要遵守规范，而且要考虑它的效率、它的价值、以后是否能重用以及重用的方式，这样才能使软件企业的生产效率不断提高，编码质量不断提高，使企业不断地向高等级迈进。良好的编码风格要求使用合适的语句结构，源程序要文档化，数据说明要规范并易于理解，输入/输出要简单、格式尽可能一致。

1.2.5　测试阶段

简单地说软件测试阶段就是检验开发的软件是否完成了所要求的功能，整体性能、结构如何，是否满足用户的要求以及是否能与其他系统元素协同工作等。严格地说，软件测试从编码阶段就应该开始。

软件测试是保证软件质量的重要活动，也是软件开发过程中占有最大百分比的技术工作。软件测试的目的是为了发现错误，找出软件产品的缺陷，因而，没有发现错误的测试是不成功的，为了使测试工作有效且系统化，将测试阶段又划分成单元（或组件）测试、集成测试、确认测试和系统测试几个子阶段，目的是为了有针对性地发现错误。软件测试的步骤可以用图 1-13 形象地描述，单元测试是检查每个程序模块是否正确地实现了规定的功能。在面向对象的测试中，单元测试就是对组件的测试，组件可以是对象类或函数，或者是这些实体的一个相关集合；集成测试是将经过单元测试的模块组装起来进行测试；确认测试是检查软件是否满足 SRS 中确定的各种需求以及软件配置是否完整、正确；系统测试则是把软件纳入实际运行环境中，与其他系统元素一起进行测试。

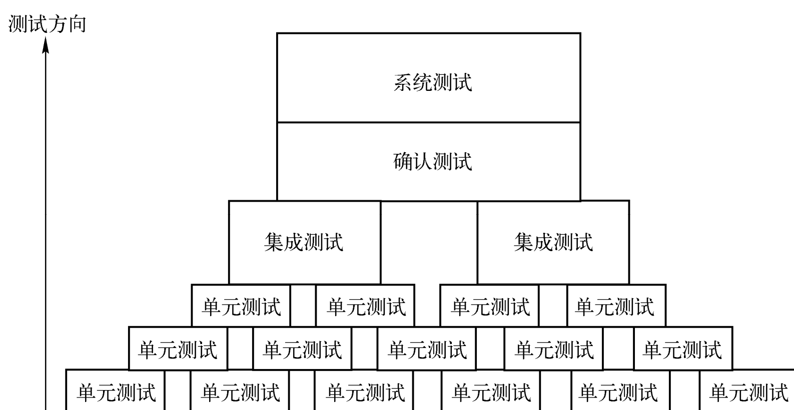


图 1-13　软件测试活动示意图

由于软件各个开发阶段的任务不同，因而所采用的测试方法和策略也不同。如集成测试可分为增殖测试、非增殖测试和混合增殖测试，增殖方式又可分为自顶向下以及自底向上增殖方式。确认测试包括功能测试、软件配置审查、验收测试以及 α 测试和 β 测试 4 种。

软件测试的种类大致可分为人工测试和基于计算机的测试，基于计算机的测试又可分为

白盒测试和黑盒测试。白盒测试是结构测试，它以程序的逻辑为基础设计测试用例，基本思想是选择测试数据使其满足一定的逻辑覆盖，以此来尽可能地发现这一类中的错误。逻辑覆盖可分为语句覆盖、分支覆盖、条件覆盖、分支/条件覆盖、多重条件覆盖和路径覆盖 6 种。黑盒测试是功能测试，它不考虑程序的内部结构，根据程序的规格说明来设计测试用例，主要方法有等价类划分法、边界值分析法、错误推测法和因果图法，它的假设前提是，如果测试中发现了错误，那么满足某个条件的测试用例发现的就是这一类中的错误。

人工测试主要有桌面检查、代码会审和走查，代码会审和走查一般以会议的方式进行，3～5 人开会，每次会议 1.5 小时左右。

测试工作结束后要进行软件调试。调试与测试不同，软件测试的目的是要尽可能多地发现软件中的错误，而调试是要进一步诊断和改正程序中潜在的错误，调试的方法主要有强力法排错、回溯法排错、归纳法排错和演绎法排错。

软件测试阶段主要交付的文档有软件测试计划和软件测试报告，软件测试计划起到一个框架结构的作用，它规划了测试的步骤和安排。一个测试计划的基本内容包括基本情况分析、测试需求说明、测试策略和记录、测试资源配置、问题跟踪报告、测试计划的评审等。软件测试报告是产品部门与技术部门沟通的主要技术手段，测试报告直接影响软件缺陷的修改速度。因此，软件测试报告必须是客观的。一个好的测试报告，应该使阅读报告的人获益。软件测试报告的主要内容包括测试内容、测试意义及目标、测试环境与测试标准、测试用例设计及测试结果、软件功能的结论、可靠性和有效性评价、软件缺陷及改进建议、测试消耗等。软件测试报告评审通过后，软件才能投入使用或投放市场。

1.2.6　运行和维护阶段

这一阶段相当于人的生命周期中的成年期，软件的价值也是在这个阶段才能体现出来，人在这一时期不但要给社会做贡献，同时也要维护自身的健康，得病了需要治疗，身体弱了要加强锻炼，要注意保健、提升自身素质等。软件也是如此，软件系统在投入运行后，还会逐步暴露出错误，另外，用户可能也会不断提出一些新的要求，或者软件要适应新的环境。因此，软件系统需要不断地排错、修改、扩充功能，这些工作都是维护。

软件维护是软件生命周期中消耗时间最长、最费精力、费用最高的一个阶段。如何提高可维护性、减少维护的工作量和费用是软件工程的一个重要任务。系统规模越大、越复杂则相应的维护工作也越多。按照不同的目标，维护活动可以分为 4 类：以纠正软件错误为目的的纠错性维护；为适应运行环境变化而进行的适应性维护；以增强软件功能为目标的完善性维护，以及为了改善未来软件的可靠性和可维护性所进行的预防性维护。这 4 种维护分别占总维护工作量的 21%、25%、50%和 4%，如图 1-14 所示。

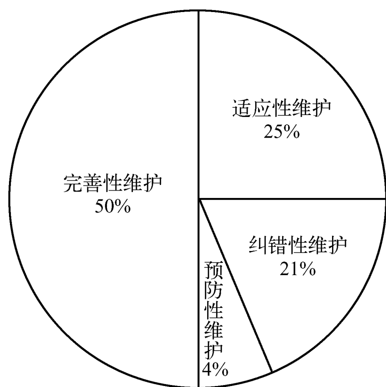


图 1-14　软件维护工作量的分布

软件可维护性是软件开发各个阶段的关键目标之一，为使软件具有较高的可维护性，在软件开发过程中，要使软件尽可能地具有较高的可理解性、可测试性、可修改性、可靠性、可移植性、可使用性和高的运行效率。

为提高软件的可维护性，在软件开发过程中要提供完整和一致的文档，进行严格的测试和阶段审查，建立明确的软件质量目标和优先级，使用现代化的开发技术和工具，并选用可维护性好的程序设计语言。

随着软件技术的飞速发展，软件维护技术也随之发展，对于新软件体系结构应采用新的软件维护方法。软件再工程是一种新的预防性维护方法，它通过逆向工程和软件重构等技术，来有效地提高现有软件的可理解性、可维护性和复用性。对现有应用系统实施再工程之前，应进行成本效益分析，有价值的系统才适宜进行软件再工程。对于多个欲将再工程的应用系统应按照成本效益分析的优先级进行排序，先对再工程整体效益高的应用系统实施软件再工程。

1.3 软件开发模型

软件开发模型也称为软件生存期模型，是软件开发过程的一个宏观框架，该框架反映了软件生命周期的主要活动以及它们之间的联系。

进行一场战役决策者们要制作沙盘，以便排兵布阵，讨论战役的进展，掌控全局。类似地，软件开发模型其实就是从宏观上描述软件的开发进程，讨论如何安排软件生命周期中的各项工作和任务，如何组织软件生命周期中的各种活动，各个阶段如何衔接。

1.3.1 瀑布模型

瀑布模型是提出最早的。目前使用最广泛的软件开发模型，图 1-15 是瀑布模型的基本形式。从图中可以看出，整个模型如同瀑布流水，开发活动互相衔接，逐级下落，每个阶段都必须在上一阶段的工作结束后才能开始。这种模型描述的是很规范的软件开发方式，每个阶段都有明确的工作目标和任务。

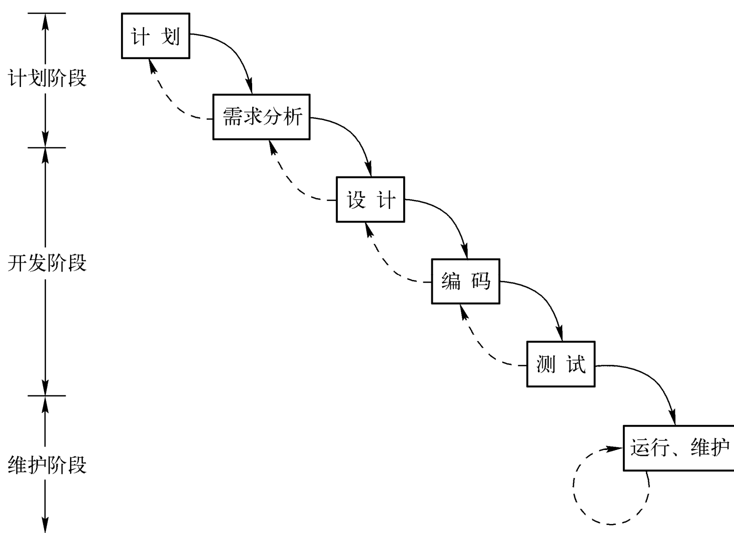


图 1-15 瀑布模型的基本形式

瀑布模型的形式多种多样，如图 1-16 和图 1-17 所示，都是基本形式的变种。在图 1-16 中，将软件维护活动进行了展开，也和开发活动一样有序进行，由此就构成了软件生命周期的循环形式。

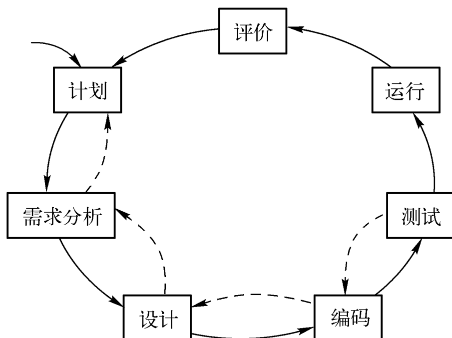


图 1-16　循环形式的瀑布模型

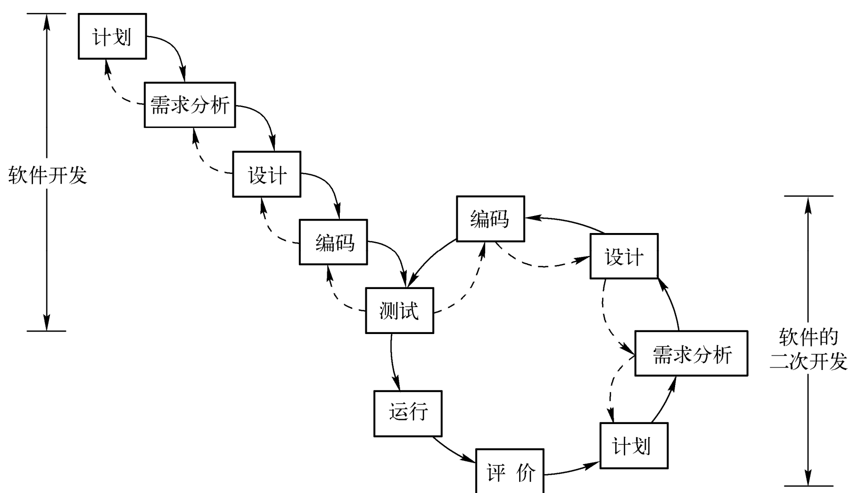


图 1-17　b 型软件生存周期

图 1-17 其实是图 1-15 和图 1-16 合成后得到的，整体形状像英文字母“b”，由于软件在投入运行后要不断地维护，为把开发活动和维护活动区别开来，所以提出了 b 型软件生存期模型，并且把维护看作是软件的二次开发。

瀑布模型是其他模型的基础，是规范的开发模型，它支持结构化开发，为软件开发和维护提供了较为有效的管理模式，它对控制软件开发复杂度、制定开发计划、进行成本预算、组织阶段评审和文档控制等各项软件工程活动都较为有效，对保证软件质量具有较好的作用。但它的突出缺点是缺乏灵活性，无法应付软件需求不明确、不准确的问题，特别是，由于各阶段工作次序固定，使前期工作中造成的差错越到后期影响越大，带来的损失也越大，而要想纠正它们所花费的代价也越高，而这又是不可避免的。

1.3.2　演化模型

演化模型也叫原型开发模型，主要是针对事先不能完整定义需求的软件开发而提出的，

它是瀑布模型的变种。开发人员根据用户的需求，先开发一个原型，让用户试用，用户提出改进、精化及增强系统能力的需求。开发人员根据用户的反馈意见，实施开发的迭代过程。这一过程反复进行，逐渐演化成最终的系统。每一迭代过程均由需求、设计、编码、测试、集成等阶段组成，图 1-18 所示是一种演化模型的表示。如果在一次迭代中，有的需求不能满足用户的要求，可在下一次迭代中进行修正。

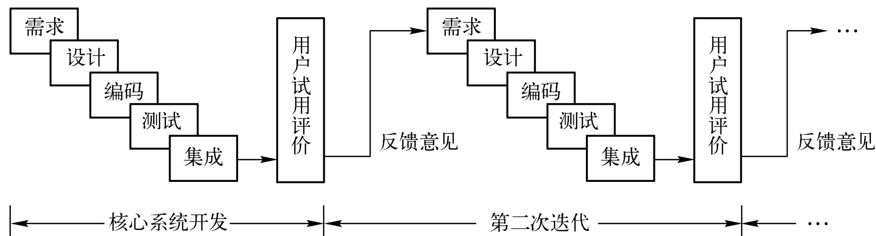


图 1-18　一种演化模型

演化模型也有多种形式，如丢弃型、样品型、渐增式演化型等，它的特点是突出一个“快”字，用户可以很快看到未来系统的“样品”，但它存在的问题也比较严重。一方面，为了尽快构造出原型，开发人员常常不得不使用不适当的开发环境、编程语言以及效率不高的算法，而这些有可能集成到系统中，成为实际系统的一部分。另一方面，构造原型时很难考虑到软件的整体质量和系统以后的可维护性问题，由此，可能造成开发出的软件质量不高。

1.3.3　螺旋模型

螺旋模型整体是螺旋形状，它也是反复迭代的过程，如图 1-19 所示，螺旋模型其实是把瀑布模型和演化模型相结合所建立的一种软件开发模型，它的显著特点是加入了二者所忽略的风险分析。软件风险是任何软件项目中都普遍存在的现象，也是多方面的。风险分析的目的是在造成危害之前，及时对风险进行识别，并采取对策，进而消除或减少风险造成的损害。

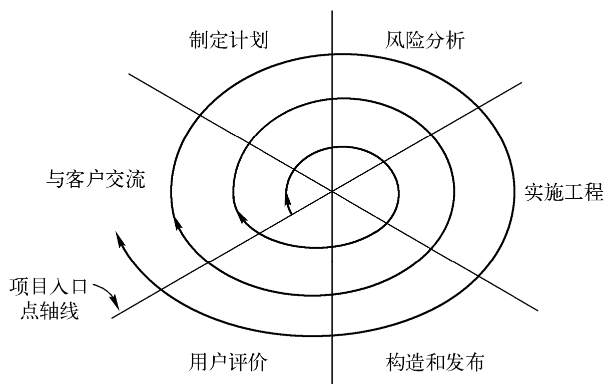


图 1-19　一个典型的螺旋模型

一般按软件生命周期的工作任务将螺旋模型划分为若干框架活动，也称为任务区域，每一个区域均包含若干适应待开发项目的工作任务，称为任务集合。对于较小的项目，工作任务较少，形式化程度较低；对于较大的、关键的项目，每个任务区域都有较多的工作任务且形式化

程度较高，典型的螺旋模型有 3 到 6 个任务区域，图 1-19 是 6 个任务区域的螺旋模型。

从中心开始，顺时针按螺旋线向外移动，就可一步一步地建立起完整的软件版本。在第 1 圈可能产生软件产品的归约，第 2 圈可能产生一个原型，第 3 圈用于软件产品的增强，第 4 圈可能是软件产品的维护。每次经过“制定计划”区域是为了对软件项目计划进行调整，根据用户的评价来调整费用和进度，根据每次的风险分析结果，都要做出继续还是停止的决策，如果项目风险太大就只能停止。

螺旋模型是当前大型软件系统开发的最现实方法，但它要求有风险评价的专门技术，这些专门技术决定了评价是否成功，若主要风险不能发现，则会造成重大损失。

1.3.4 喷泉模型

喷泉模型如图 1-20 所示，是描述面向对象软件开发过程的模型，喷泉的特点是连续无间隙。用面向对象技术开发软件时，软件生命周期中的各个阶段之间并无明确的边界，工作是连续的，就像喷泉一样，在分析阶段根据分析员的理解建立了相关概念，如建立用例图、类图、类的层次结构图、建立实例联系等，并确定了类的属性和操作。由于分析和设计是连续的过程，进入设计阶段后，可能还会派生出一些对象类，并要建立对象间的联系，在实现阶段，为适应问题描述及解法可能还会设计一些对象类，在测试阶段可能会根据测试需要又要派生出新的类，这个过程是迭代的。系统的某个部分常常会重复工作多次，相关功能在每次迭代中被加入演进的系统，因此，用“喷泉”一词来描述是很贴切的。

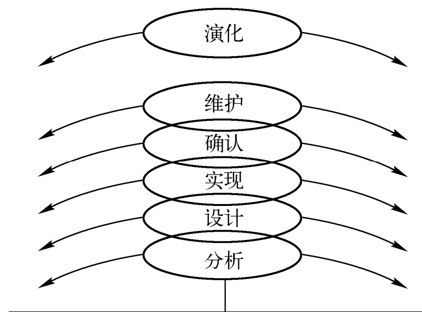


图 1-20 喷泉模型

1.3.5 其他模型

软件生存期模型只是对软件生命周期各个阶段工作的一种图示化描述，它描述了软件开发都有哪些工作，各项工作如何衔接，同时也是为了指导软件开发过程。除了以上几种典型模型外，还有一些比较有影响的模型。

1．智能模型

智能模型是以专家系统或知识库为核心，所有的软件工程工作都与这个核心有关，因而该模型是基于知识的软件开发模型，如图 1-21 所示，知识库中存放模型、知识、规则，软件开发人员采用规约和推理机制，辅助进行相关开发工作。从图中可以看出，软件的维护不在程序一级上进行，而是在功能归约也就是需求分析一级进行，这就把问题的复杂性大大降低了，从而可把精力更加集中于具体描述的表达上。具体描述可以使用形式功能归约，也可

以使用知识处理语言描述等。

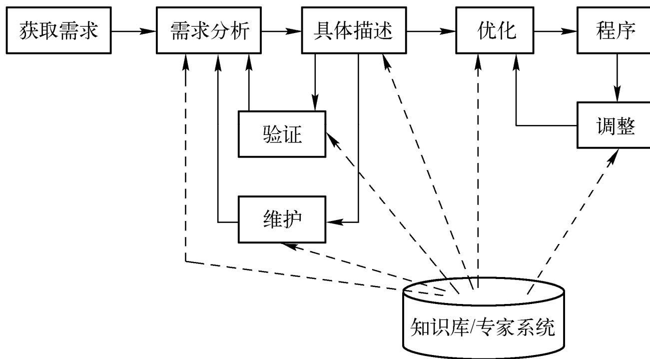


图 1-21　智能模型

2．增量模型

增量模型也是从瀑布模型演化而来，如图 1-22 所示，它融合了瀑布模型和原型开发模型的优点，也可以看作是演化模型的一种。宏观上看开发过程是迭代进行的，每次迭代都像是一个瀑布模型，它的每个增量都是可交付的软件。通常，每个增量的建造是基于那些已经交付的增量而进行的，任何增量均可以按原型开发模型来实现。例如，使用增量模型开发一个字处理软件，在第 1 个增量中发布基本的文件管理、编辑和文档生成功能，在第 2 个增量中发布更加完善的编辑和文档生成能力，第 3 个增量实现拼写和语法检查功能，第 4 个增量完成高级页面布局功能，等等。在不断地演化中，产品的功能、性能不断地提高。

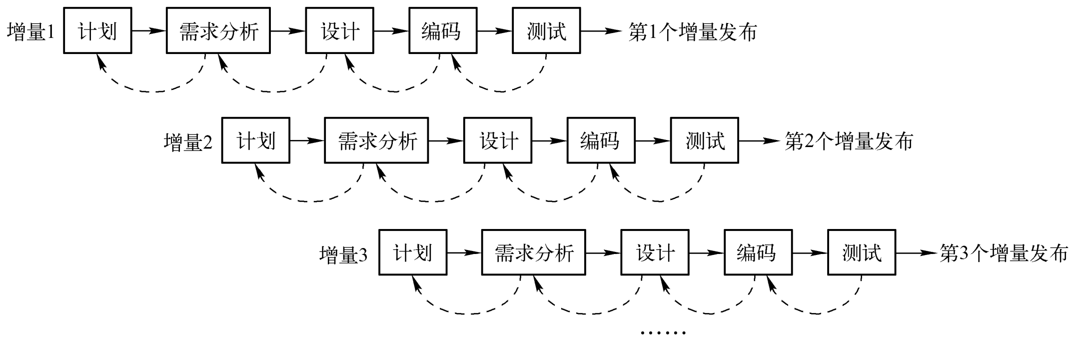


图 1-22　增量模型

3．并发过程模型

并发过程模型有时也称并发工程，它定义了一系列事件，这些事件将触发软件工程的主要技术活动、动作或者任务的状态转换。例如，设计的早期阶段（建模活动期间发生的主要软件工程动作），发现了需求模型中的不一致性，于是产生了分析模型修正事件，该事件将触发需求分析动作从“完成”状态到“等待改变”状态。图 1-23 是并发过程模型中一个活动的图形表示，该活动是分析，其他活动也用类似的方式表示。

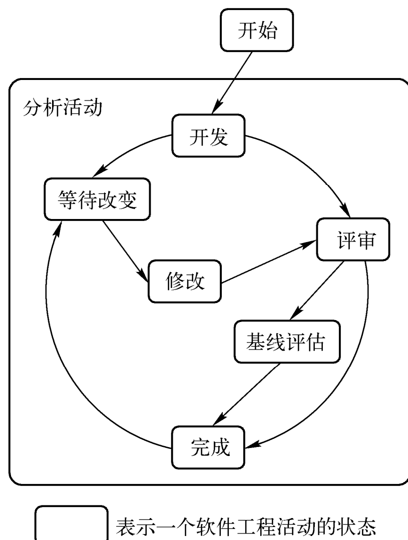


图 1-23　并发过程模型的一个元素

并发过程模型可用于所有类型的软件开发，它能够提供精确的项目当前状态图。它不是把软件工程活动、动作和任务局限在一个事件的序列，而是定义了一个过程网络。网络上每个活动、行为和任务与其他活动、行为和任务同时存在。过程网络中某一点产生的事件可以触发状态的转换。

4．基于构件的开发模型

基于构件的开发（Component-Based Development，CBD）模型是在面向对象技术的基础上发展起来的，如图 1-24 所示，它融合了螺旋模型的许多特征，利用预先包装好的软件构件（有时称为类）来构造应用系统。统一软件开发过程（Unified Software Development Process）是近年来产业界提出的一系列基于构件开发模型的代表。使用建模语言 UML，统一过程定义了将被用于建造系统的构件和将用于连接构件的接口。使用迭代和增量开发的组合，统一过程通过应用基于场景的方法（从用户的视角）来定义系统的功能，然后将功能和体系结构框架耦合，体系结构框架标识了软件将呈现的形式。

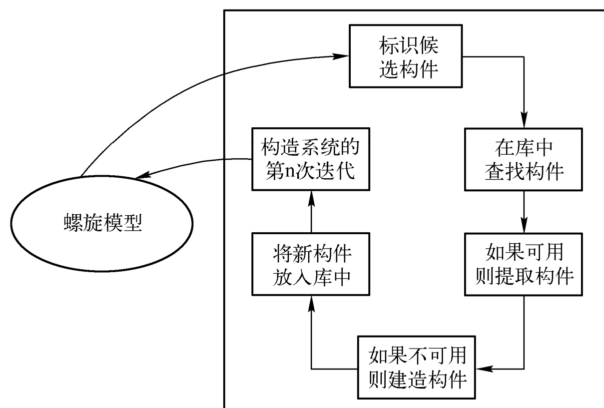


图 1-24　基于构件的开发模型

基于构件的软件工程在特定的应用领域内标识、构造、分类和传播一系列软件构件。这些构件经过合格性检验、适应性修改，并集成到新系统中。对于每个应用领域，应该在建立了标准数据结构、接口协议和程序体系结构的环境中设计可复用构件。

5．面向复用的软件开发模型

面向复用的软件开发模型也称为面向复用的软件工程，它与基于构件的开发模型的思想是一致的。

在大多数的软件项目中，都存在一定程度的软件复用。例如，当人们注意到某项目中的设计或代码是与当前项目中所需要的部分很相像的时候，一般不会再重做一次，因而复用就自然地发生了。人们搜寻这些可复用的东西，而后根据需要修改它们，再将其纳入到自己的系统中来。但是，这样随意性的复用并没有考虑到所采用的开发过程。

在 21 世纪的今天，注重复用现存软件的开发过程得到了广泛采用。面向复用的方法依赖于存在大量可复用的软件组件以及能组合这些组件的集成框架。有时，这些组件本身就是一个系统（例如 COTS，即商业现货系统），它能提供专门的功能，例如字处理或制表软件。

用于面向复用过程的软件组件有三种类型：一种组件类型是通过标准服务开发的 Web 服务，可用于远程调用；另一种组件类型是对象的集合，如.NET 或者 J2EE 等集成在一起作为一个包和组件框架来使用；还有一种组件类型是独立的软件系统，通过配置在特定的环境下使用。

面向复用开发的一般过程模型如图 1-25 所示，尽管初始需求描述阶段和有效性验证阶段与其他过程差不多，但是面向复用过程的中间阶段是不一样的，下面将这几个中间阶段的工作简单描述如下。

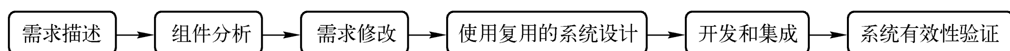


图 1-25　面向复用的软件开发模型

（1）组件分析

在组件分析阶段先给出需求描述，然后搜寻能满足需求的组件。很多情况下没有正好合适的组件供选择，因而得到的组件往往只能提供所需要的部分功能。

（2）需求修改

在这个阶段，先根据得到的组件信息分析需求，然后修改需求以反映可得到的组件。当需求修改无法做到的时候，就需要重新进入组件分析活动以搜索其他可能的替代方案。

（3）使用复用的系统设计

在这个阶段，设计系统的框架或者重复使用一个已存在的框架。设计者分析那些将被重复使用的组件，并组织框架使之适应这些组件。当某些可复用的组件不能得到时，必须重新设计一些新的组件。

（4）开发和集成

当组件不能买到时，就需要自己开发，然后集成这些自己开发的组件和现货组件，使它们成为一个整体。在这个模型中，系统集成虽然是一项独立的活动，但它已经成为软件开发过程的一个部分。

从上述描述可以看出，面向复用模型的明显优势是它减少了开发软件的工作量，使用该模型不但可以降低软件开发成本，也可以降低开发中的风险，同时也可使软件快速地交付。该模型的缺点是需求妥协不可避免，而这可能会导致一个不符合用户真正需要的系统。此外，对系统进化的控制也将失效，因为可复用的组件新版本可能是不受机构控制的。

6．形式化方法模型

形式化方法模型是一种严格的软件工程方法，是一种强调正确性的数学验证和软件可控性认证的软件过程模型，其目标和结果是使软件的出错率非常低，这是其他方法所难以达到的。图 1-26 为形式化方法模型的一种，称为净室过程模型。由于形式化方法模型使用很费时且昂贵，因而应用较少。

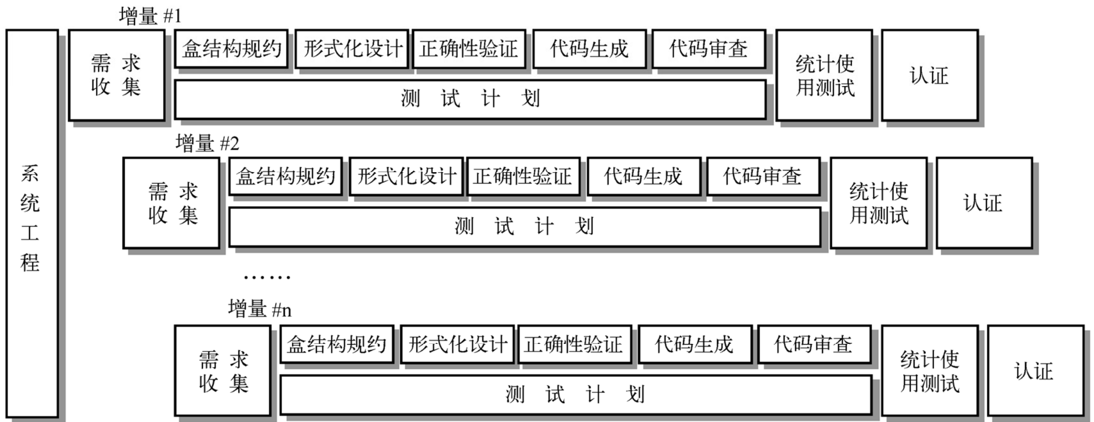


图 1-26　一种形式化方法模型——净室过程模型

7．第四代技术模型

第四代技术（4th Generation of Technology，4GT）包含了一种组件工具，它们都具有共同点，能使开发人员在较高的级别上规约软件的某些特征，并把这些特征自动生成源代码。

目前，支持 4GT 模型的软件开发环境包括以下各部分和全部工具：数据库查询的非过程性语言、报表生成、数据处理、屏幕交互和定义、代码生成、高层图形、电子表格、以及使用 HTML 和用于 Web 站点的工具等。这些工具都很适用，但都局限于一些专门的应用领域。现在，还没有一种 4GT 环境能够同时方便地使用上面所介绍的各类应用软件。软件工程的 4GT 模型如图 1-27 所示，目前，围绕该模型的应用还有很多争论。支持者认为，它可以极大地减少开发时间，提高软件开发效率；反对者认为，目前的 4GT 工具并不比编程语言容易，同时使用这样的工具生成的源代码效率不高，特别是用 4GT 开发大型软件系统可维护性很差。

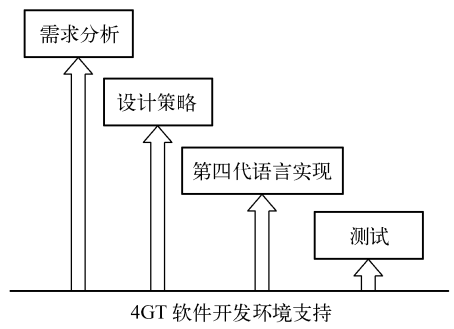


图 1-27　4GT 模型

4GT 已经成为一种重要的软件工程方法，当它与 CBD 方法结合起来时，可能会成为软件开发的主流方法。

8．混合模型

每种软件生存周期模型都不是十全十美的，要让它们适应各种项目的开发和各种情况的需要也是很难的。为此，可开发混合模型（Hybrid Model），如图 1-28 即为一个混合模型。开发混合模型的目的是为了发挥各自模型的优势，对于具体开发组织也可使用 2～3 种不同的模型组成一个较实用的混合模型，以便获得最大的效益。

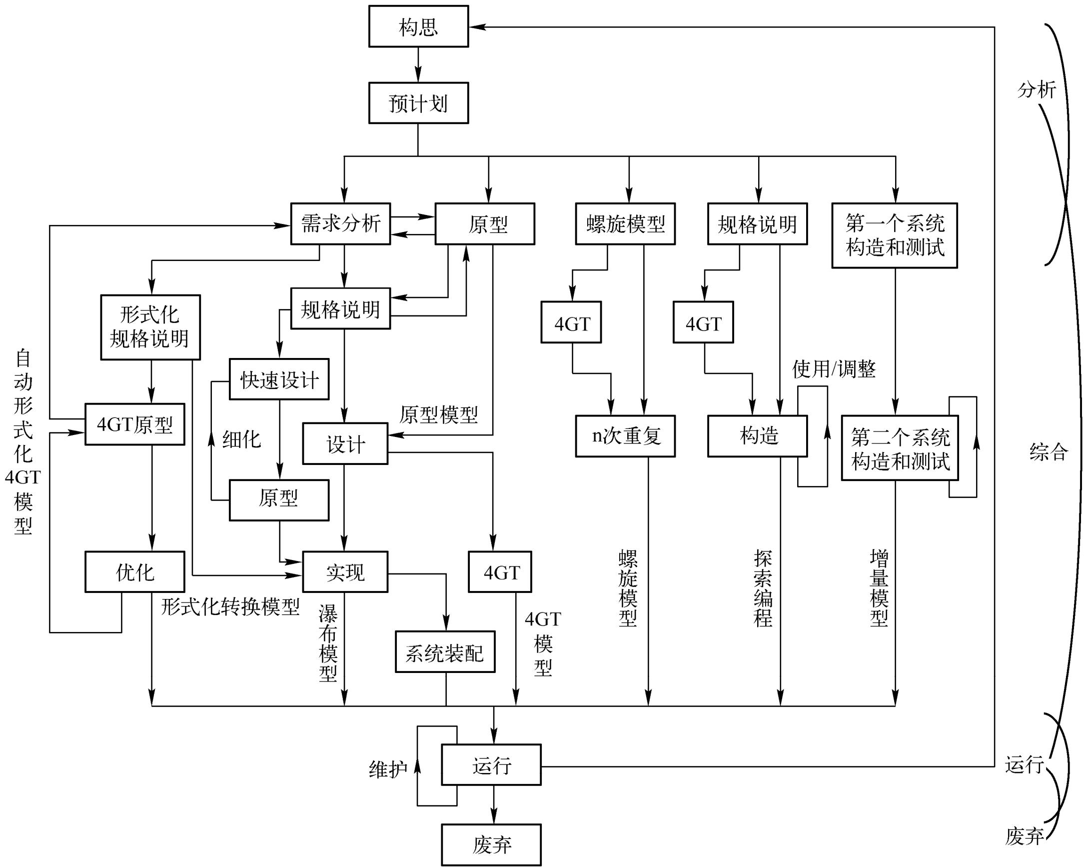


图 1-28　一个混合软件生存期模型的例子

1.4　实用案例

现实中的软件系统千千万万，软件开发方式也千差万别，同一个问题不同的开发组织可能会选择不同的开发模型去解决，开发出的软件系统也不会完全一样，但是基本目标都是一样的，那就是应该满足用户的基本功能要求，否则，再好的系统也是没有意义的。以下两个案例一个是读者都熟悉的业务活动的出卷系统，另一个是伴随互联网广泛应用后出现的家庭安全住宅系统。

1.4.1　出卷系统的开发模型选择

某学校要开发一个出卷系统，以整体提高考试出卷的客观性、规范性和科学性，提高试

卷质量和管理水平，该系统可完全自动、也可以自动和手动结合、还可以完全手动出卷。

该系统其实是将以前人工工作计算机化，表面上看起来比较简单，但当认真考虑后却发现有很多约束，例如，一张卷子不能有重复的题目，连续几年（可根据需要设定年限）不能有重复试卷和试题，根据考试时间确定题量，确定各种类型考题的比例及分值，根据各类型总分及题量确定各小题的分值等等。为了建立实用的出卷系统，需要做很多非开发方面的工作，例如，一门课的题目数至少要满足某个最低限，如大于 1000 道题，一门课的试卷数不少于 20 份等等。如果试卷太少、题量太小，则开发出的系统不具有实用性。

系统的功能结构如图 1−29 所示。主模块应提供课程选择、联机帮助等功能，用户可根据需要选择调用“试卷管理”子系统、“题库管理”子系统、“在线考试”子系统或“系统维护”子系统。

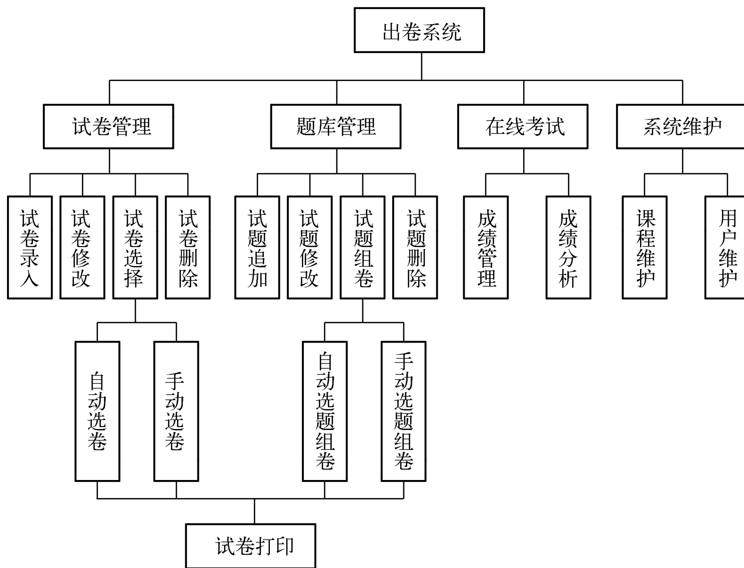


图 1−29　出卷系统的功能结构要求

“试卷管理”子系统主要包括试卷录入、试卷修改、试卷选择及试卷删除 4 项功能。

“题库管理”子系统主要包括试题追加、试题修改、试题组卷及试题删除 4 项功能。

“在线考试”子系统可包括“成绩管理”及“成绩分析”两个下级模块。“成绩管理”模块可为学籍管理系统提供规范的成绩，或直接将成绩写入学籍库中，也可打印成绩单，对成绩排名，转换成学分绩，对不及格的学生给出报警等等；“成绩分析”模块可自动生成考试成绩分析表，可根据需要给出定量的数据，如最高分、最低分、平均分、优秀人数及比例、良好人数及比例、中等人数及比例、及格人数及比例、不及格人数及比例，可给出饼图、柱图等，也可根据评价规则给出定性评语，如试卷的总体评价、学生掌握基本理论情况分析、学生灵活运用知识、分析、解决问题能力情况分析、存在问题分析、今后改进意见等。

“系统维护”子系统可包括课程维护及用户维护等，可根据需求分析的结果来确定其功能，在软件设计阶段确定软件结构。

系统可以统一建库，也可以按课程建库，系统应建立课程名称库、每门课程的试题库以及试卷库等。为便于组卷，可将试题按难度分为难、中、易 3 类；试题类型一般应有是非题、选择题、分析题、综合题、出题时间、使用时间，每个题目都应该有唯一的编号、覆盖的知识点、答案和难度，在试题录入时系统应能提供图形和文字编辑功能。

这是用户非常熟悉的业务活动，需求、目标、功能都很明确， 建议采用规范的瀑布模型进行开发。

1.4.2 住宅安全系统 SafeHome 的开发模型选择

该案例从美国软件工程专家 Roger S.Pressman 所著《软件工程：实践者的研究方法》一书中整理而来。一家专门开发家用和商用消费产品的公司有一个拳头产品——通用的无线盒，只有火柴盒大小，可以把它放在各种传感器上，放在任何电子产品中，比如数码相机里，可以通过无线连接获得它的输出。利用这一拳头产品，他们计划开发一个住宅安全系统 SafeHome，该产品采用新型无线接口，给家庭和小型商务使用者提供一个由计算机控制的系统——家庭安全、监视、应用和设备控制。例如，你可以在回家的路上关闭家里的空调，或者诸如此类的家电。该公司的工程部已经做了相关的技术可行性研究，该产品可行且制造成本不高，由于所需的大多数硬件可以在市场上购买到成品，因而 SafeHome 系统的主要工作是软件开发。在美国，70%的家庭拥有计算机。如果该产品开发成功且价格合理，那么将会有广阔的市场前景。到目前为止，只有该公司拥有这一无线控制盒技术，而且将在这个方面保持两年的领先地位。以下是在该公司软件工程部会议室开会讨论开发模型的选择过程。

1．第一次会议简况

会议主要成员：项目经理 Lee Warrer，软件工程经理 Doug Miller，软件团队成员 Jamie Lazar、Vinod Raman 和 Ed Robbins。

Lee：正如我们现在所看到的，我已经花了很多时间讨论 SafeHome 产品的产品线。毫无疑问，我们做了很多工作定义这个东西，我想请各位谈谈你们打算如何做这个产品的软件部分。

Doug：看起来，我们过去在软件开发方面相当混乱。

Ed：Doug，我不明白，我们总是能成功开发出产品来。

Doug：你说的是事实，不过我们的开发工作并不是一帆风顺，并且我们这次要做的项目看起来比以前做的任何项目都要大而且更复杂。

Jamie：没有你说的那么严重，但是我同意你的看法。我们过去混乱的项目开发方法这次行不通了，特别是这次我们的时间很紧。

Doug：我希望我们的开发方法更专业一些，我们现在需要一个过程。

Jamie：我的工作是编程，不是文书。

Doug：在你反对我之前，请先尝试一下。我想说的是，似乎瀑布模型并不适合我们，……它假设我们此刻明确了所有的需求，而事实上并不是这样。

Vinod：同意你的观点，瀑布模型太 IT 化了……，也许适合于开发一套库存管理系统或者什么，但是不适合我们的 SafeHome 产品。

Doug：对。

Ed：从原型开发模型的特点来看，正适合我们现在的处境。

Vinod：有个问题，我担心用原型开发方法不够规范。

Doug：别怕，我们还有许多其他选择。我希望在座的各位选出最适合我们小组和我们这个项目的开发模型。

2．第二次会议简况

会议主要成员：项目经理 Lee Warrer，软件工程经理 Doug Miller，软件团队成员 Jamie Lazar 和 Vinod Raman。

Doug 首先介绍了一些可选的演化模型，比较了它们各自的特点。

Jamie：我现在有了一些想法，增量模型挺有意义的，我也很喜欢螺旋模型，听起来很实用。

Vinod：我赞成用增量模型。我们先交付一个增量产品，听取用户的反馈意见，再重新计划，然后交付另一个增量。这样做也符合产品的特性。我们能够迅速投入市场，然后在每个版本或者说在每个增量中添加功能。

Lee：等等，Doug，你的意思是说我们在螺旋的每一轮都重新生成计划？这样不好，我们需要一个计划，一个进度，然后严格遵守这个计划。

Doug：你的思想太陈旧了，Lee，就像他们说的，我们要现实。我认为，随着我们认识的深入和情况的变化来调整计划更好。这是一种更符合实际的方式。如果制定了不符合实际的计划，这个计划还有什么意义？

Lee：我同意这种看法，可是高管人员不喜欢这种方式，他们喜欢确定的计划。

第三次会议他们达成了共识，决定采用增量模型开发 SafeHome 产品软件。

1.5 小结

计算机软件是计算机系统的主要组成部分，它包括程序、数据、文档、方法、规则等，软件主要有系统软件、应用软件、工具软件和可重用软件 4 类。软件工程是集成计算机软件开发的过程、方法和工具的学科，是针对软件危机而发展起来的一门学科。软件的生命周期是一个重要的概念，是指从软件计划开始直到软件被弃为止的所有阶段，

软件开发模型也称为软件生存期模型，它是反映软件开发过程、开发活动和开发任务的结构框架，实际进展中允许进行改进或适当的变化，它是指导我们进行软件开发的一个宏观框架，不能被它完全束缚。

软件生存期模型目前虽有十余种，但常用的只有几种，使用最多的仍然是瀑布模型，它是其他模型的基础。

1.6 习题

1．如何理解软件概念，简述软件有哪些分类方法？

2．简述以下概念。

1）系统软件。

2）应用软件。

3）工具软件。

4）可重用软件。

3．解释下列软件的特点。

1）嵌入式软件。

2）产品线软件。

3）人工智能软件。

4．软件的发展经历了哪几个阶段？简述各阶段名称及特点。

5．什么是软件危机，主要有哪些表现？

6．美国 IBM 公司的 OS/360 项目总负责人费雷德里克·布鲁克斯（Frederick P.Brooks Jr.）写过一本计算机界人人皆知的名著，该名著的名称是什么？

7．简述软件工程的起源。

8．计算机系统的成本是由硬件决定还是由软件决定？

9．下列哪个概念是 1968 年在德国小镇加米施（Garmisch）召开的北大西洋公约组织（简称 NATO）学术会议上与会学者们首次提出的。

软件工程、互联网、云计算、物联网、大数据

10．简述 Roger S. Pressman 的软件工程层次化定义。

11．为什么说软件工程是一门多学科交叉的学科？

12．简述北京大学王立福教授等给出的软件工程定义。

13．什么是软件的生命周期？

14．给出瀑布模型的基本形式，简述它的特点。

15．给出喷泉模型的图示表示，它是描述什么开发过程的模型，喷泉模型的特点是什么？

16．螺旋模型的特点是什么？典型的螺旋模型有几个任务区域，试给出一种螺旋模型的图示表示。

17．说出面向复用的软件开发模型的主要步骤，简述该模型的主要特点。

18．演化模型又称作什么模型？说出几种演化模型的形式，说出演化模型的主要优点和缺点。

19．常见的软件生存期模型主要有哪些？给出名称。

20．按照软件生命周期顺序，对下列任务进行排序。

1）交付软件项目开发计划。

2）制定软件测试计划。

3）获取需求。

4）编写需求。

5）估算软件开发成本、进度、工作量。

6）预测开发软件所需要的资源。

7）管理软件风险。

8）安排软件开发进度。

9）评审软件项目计划。

10）分析需求。

11）确定待开发软件系统的工作范围。

12）协商需求。

13）管理需求。

14）审查需求。

15）编写软件需求规格说明（SRS）。

16）评审 SRS。

17）编写设计规格说明（即软件设计说明书）。

18）评审设计规格说明。

19）设计软件体系结构。

20）设计数据结构。

21）描述软件过程。

22）编写程序代码。

23）进行确认测试。

24）进行软件调试。

25）进行单元测试。

26）进行系统测试。

27）进行集成测试。

28）撰写软件测试报告。

29）评审软件测试计划。

30）评审软件测试报告。

21．常见的软件维护有哪几种类型，各占维护工作量的百分比是多少？

22．在案例 2 中，为什么 SafeHome 产品软件不采用瀑布模型？

第 2 章　软件需求分析

一对年轻夫妇贷款买了一套新房，拿到钥匙后要装修，怎么装修，要什么风格，各个房间怎么分配，用做什么？夫妻意见可能相左，即使双方意见达到了统一，与装修人员之间也总有理不清的矛盾，甚至他们提不出具体的装修要求，装修完以后才提出不满意或具体想法。软件需求分析出现的状况与此类似，理解、归纳并正确描述用户需求是开发人员面临的一项困难工作，正确完善的需求分析将为软件开发奠定坚实的基础。

软件需求包括三个不同层次：业务需求、用户需求和功能需求，不同层次的需求从不同角度与不同程度反映着细节问题。业务需求反映组织机构或客户对系统、产品高层次的目标要求；用户需求描述用户使用产品必须完成的任务；功能需求则定义软件必须实现的功能，使得用户能通过所实现的软件完成他们的任务，从而满足业务需求。

由于软件需求分析在软件工程中的地位和作用越来越重要，因而形成了软件需求工程。作为一门学科，软件需求工程主要研究软件开发如何满足用户的需要以及如何对软件需求管理等问题。

汉语中的需求是指“由需要而产生的要求”，软件需求就是用户需要软件“干什么”的要求，软件需求分析就是详细地精化已建立的软件范围，创建所需数据、信息和控制流以及操作行为的模型，分析可选择的解决方案并将它们分配到各软件元素中去。

软件需求分析主要包括 7 个阶段的工作：起始、导出、精化、协商、规格说明、确认和管理。在需求起始阶段，软件工程师问一组与上下文无关的问题，目的是为了建立对问题、人员以及解决方案的初步理解，并在客户和开发人员之间进行有效的初步沟通与协作；在导出阶段要导出问题的范围、对问题的理解以及问题的变动；在精化阶段集中于开发精确的技术模型，用这些模型来说明软件的功能、特征和约束；不同的客户或用户可能会提出相互冲突的需求，协商阶段就是通过协商来调解这些冲突，应该将发生冲突的需求按优先级排序、进行讨论，采用迭代方式删除、组合或修改需求，使各方达到一定的满意度；规格说明是软件需求阶段完成的最终工作产品，它是后续软件工程活动的基础，一个规格说明可以是一份写好的文档、一套图形化的模型、一个形式化的数学模型、一组使用场景、一个原型或上述各项的任意组合；需求确认是检查规格说明以保证需求无歧义性、无不一致性，并检测出是否有遗漏和错误；需求管理从需求的识别开始，每个需求被分配一个唯一的标识符，一旦需求被标识，就要建立跟踪表，每个跟踪表将标识的需求与系统或其环境的一个或多个方面相关联。可以使用的跟踪表有很多，主要有特征跟踪表、来源跟踪表、依赖跟踪表。子系统追踪表和接口跟踪表。

2.1　结构化需求分析方法

结构化分析方法（Structured Analysis，SA）是最早的软件开发方法，20 世纪 70 年代末

由爱德华·纳什·尤顿（Edward Yourdon）（如图 2-1 所示）等人提出并得到发展，SA 方法是面向数据流进行需求分析的方法，适合于开发数据管理类型的应用软件的需求分析，是使用非常广泛的一种方法。该方法与结构化设计方法（Structured Design，SD）衔接起来，形成结构化分析设计技术（Structured Analysis and Design Technique，SADT），是有效的软件开发方法之一，它易学、易用，结构化分析方法的一些重要的概念也渗透到其他开发方法中。

图 2-1　结构化开发方法的创始人爱德华·纳什·尤顿

结构化方法的基本手段是“抽象”和“分解”，即用抽象模型的概念，按照软件内部数据传送、变换的关系，由顶向下逐层分解，直到找到满足功能需要的所有可实现的软件为止。

结构化分析方法使用的工具有数据流图、数据词典、结构化英语、判定表和判定树。因而，用 SA 方法进行需求分析所得到的 SRS 中应包括一套分层数据流图，完整的数据词典以及用结构化英语或判定表或判定树对处理逻辑（即变换）的说明。

2.1.1　数据流图及其画法

数据流图（Data Flow Diagrams，DFD），英文也称为 Bubble Chart，意为泡泡图，是以图示的方式来描述数据在解决问题过程中的变化过程。如果数据流图画得正确，软件的设计就变得容易了。DFD 不但是结构化分析方法的工具，用其他方法进行需求分析，DFD 也是有用的工具。

1．数据流图的组成元素

先看一个实际例子，若要为某培训中心开发一个计算机信息管理系统，代替目前的人工业务工作。为此，先要调查清楚实际的人工处理情况，经过详细调查分析得到如下描述。

培训中心为在职人员开设有若干课程，有关人员可通过电话、短信、E-mail 或信件方式报名选修某门课程，培训中心根据规定对修课学员收取一定费用，学员也可查询课程设置及课程简介等事宜。培训中心将学员发来的电子邮件、电话、信件等收集分类后，按不同情况处理。如果是报名的，则将报名数据送给负责报名事务的职员，他们查阅课程文件，检查是否能满足要求，若能满足则在学生文件、课程文件及账目文件上登记，开出报名单交财务部

门，财务人员收款后开出发票（或收据）经复审后通知学员。如果是查询的，则交由有关人员查询课程文件后给出答复。如果是想注销原来已选修的课程，则由有关职员修改课程、学生及账目文件，填写注销通知书，经复审后交与学员。对要求不合理的函电、不合理的事务以及报名后未付款的情况，则拒绝处理。

经过以上分析，可将培训中心管理系统的数据流图描述成如图 2-2 所示的形式。该图表明系统分解成收集、分类、注销、报名、付款、查询、开发票及复审 8 部分，这些部分之间通过数据对象相联系。从图 2-2 可以看出，数据流图有 4 种基本成分，如图 2-3 所示。

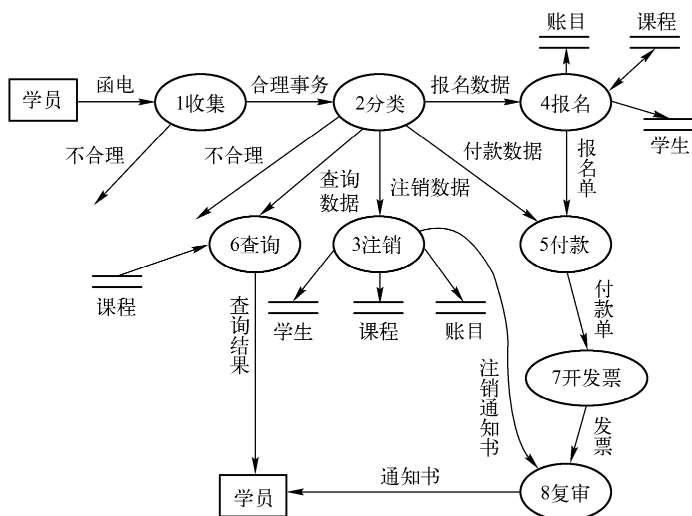


图 2-2　培训中心管理系统的 DFD

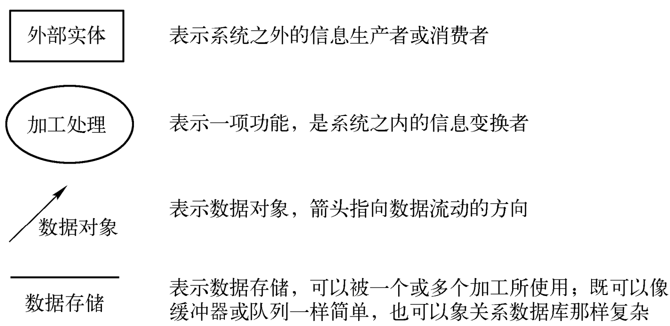


图 2-3　基本 DFD 符号

（1）数据对象

用带箭头的线表示，它反映的是数据的路径和流向，可以用名词或名词短语来命名。它的成分是固定的，如图 2-2 中，“发票”由“姓名”、“单位名”、“日期”和“金额”组成。除了流入和流出数据存储的数据对象外，每个数据对象都必须有一个合适的名字，名字要尽量具有实际意义，使人容易理解。

（2）加工处理

用椭圆表示，它反映的是对数据对象的变换，如图 2-2 中“查询”“分类”等。每一个加工处理都要有个合适的名字来反映对数据的变换过程，为了便于管理，还要对它们进行编号。

（3）数据存储

用双线来表示，可以是数据文件或记录，如图 2-2 中的“账目”“课程”等。使用时要注意箭头的方向，若读出数据，则从双线指向椭圆；若写入数据或修改数据，则从椭圆指向双线。

（4）外部实体

用矩形来表示，它是系统之外的人或事物，用以帮助理解系统，如图 2-2 中，“学员”是数据起源的地方（称为数据源），也是数据最终的目的地（称为数据池）。

2．数据流图的画法

就像画家作画一样，不同的画家应该有不同的画风和画法。由于现实中的系统千差万别，因而数据流图的画法也不应千篇一律，这里介绍的只是画数据流图的一般步骤，随着开发经验的增加，读者不但会画出合理、正确的数据流图，而且会形成自己的风格和画法。一般的做法是从当前的人工处理情况出发，由外向内、由顶向下、从粗到细、逐步求精，和画家作画的过程类似。

最初应该反映当前的实际情况，尽管当前可能有很多不合理的情况，但是在分析时要如实地反映出来，就像作画，先求形似，再求神似。用户使用的文件、数据、表格、发票、清单等，在 DFD 上就是数据对象或数据存储，用户做的工作就是加工处理，他们的名字就是平常使用的名字。

刚开始分析时，系统应该包含什么功能还不清楚，这时可使系统的范围稍大一些，把可能有关的内容都包括进去，这时应向用户了解清楚“系统从外界接受什么信息”以及“系统向外界送出什么信息”得到这两类问题的回答就等于确定了系统的边界，如“培训中心管理系统”，从外界接受“函电”，向外界送出“查询结果”和“通知书”，则 DFD 的范围就确定了，如图 2-4 所示。

图 2-4　培训中心管理系统的边界

确定了系统的边界之后，需要逐步将系统的输入和输出数据对象用若干加工处理连接起来，一般可从输入端逐步画到输出端，也可以反过来从输出端追溯到输入端，凡是在数据对象的组成或值发生变化的地方就画上一个加工处理，它的作用就是实现这一变化。

对每一个数据对象应该了解它的组成是什么，其数据项来自何处，这些组成项如何组合成这一数据对象，为实现这一组合还需要什么加工处理和数据，等等，以方便编写数据字典。

对于大型的系统，一般用分层的方式来画 DFD，以便控制复杂性，实现从抽象到具体的逐步过渡，一套分层的 DFD 由顶层、底层和中间层组成，顶层图只有一张，它给出系统

的边界，即系统的输入和输出数据对象。底层图由不能再分解的基本加工处理组成，中间层即是上层（父图）某个加工处理的分解，它的组成部分又要被进一步分解。一张图有几个加工处理就可以最多分解成几张图（即子图）。系统越大，中间层越多。

为了便于管理，需要为 DFD 和其中的加工处理编号，子图号与父图中相应加工处理的编号相同。子图中加工处理的编号由子图号、小数点和局部号 3 部分组成。

顶层图只有一张，不用编号，第一层一般也是一张，可编为 0，图中的加工处理即为 0.1，0.2，0.3，……，通常可把 0 删去。这样加工处理的编号就成为 1，2，3，……，如图 2-2 所示，可以看出，它位于第一层。

由此，我们只要数一下子图编号的小数点数，就能知道这张图位于哪一层，还知道它的父图是哪一张，如某图的编号为 5.2.3.2，则该图位于第 5 层，其父图为图 5.2.3，该图在父图中与编号为 5.2.3.2 的加工处理相对应。

为简单起见，在一张图的内部，每个加工处理可只用它们在该图中的局部号表示，但是在数据词典中要给出完整的编号，在图 2-5 中，图中加工处理的编号只给出了局部号，这是允许的。

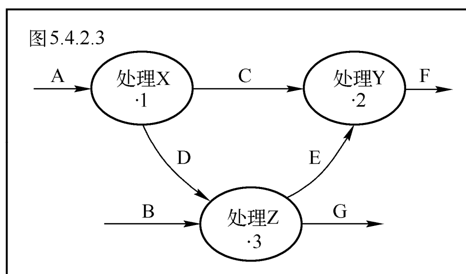


图 2-5　DFD 编号举例

应当注意，数据对象的连续性必须保持，即每个子图的输入和输出数据对象要与其父图中相应加工处理的输入和输出相同，这一概念，有时也叫分层 DFD 的平衡，如图 2-6 所示为图 2-5（图 5.4.2.3）的一张子图，它的输入和输出与父图中编号为 5.4.2.3.3（即加工处理“处理 Z”）的输入和输出是完全一致的，因而这两张图是平衡的。

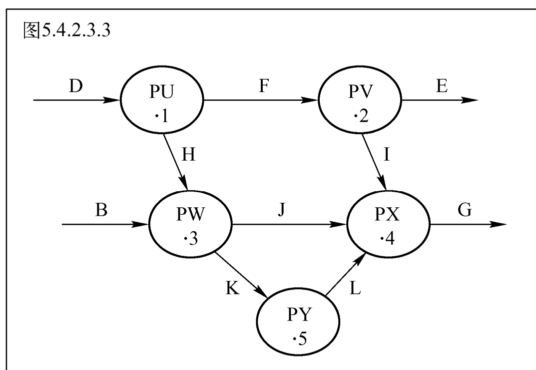
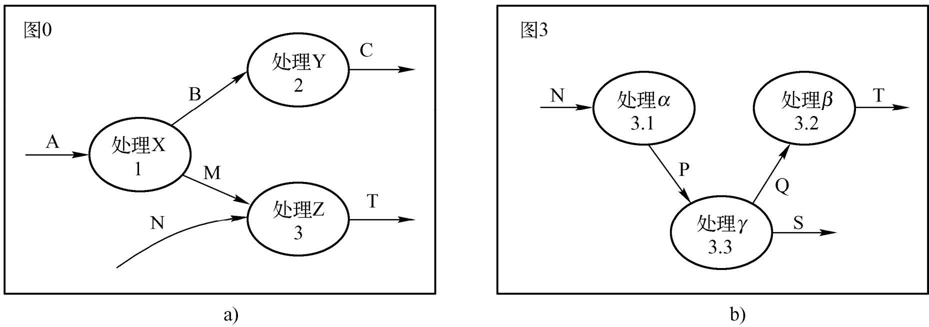


图 2-6　分层 DFD 的平衡举例

图 2-7 的父图和子图是不平衡的，因为子图中没有输入数据对象与父图中加工处理“处理 Z”的输入 M 相对应，子图的输出数据对象 S 在父图中也没有出现。

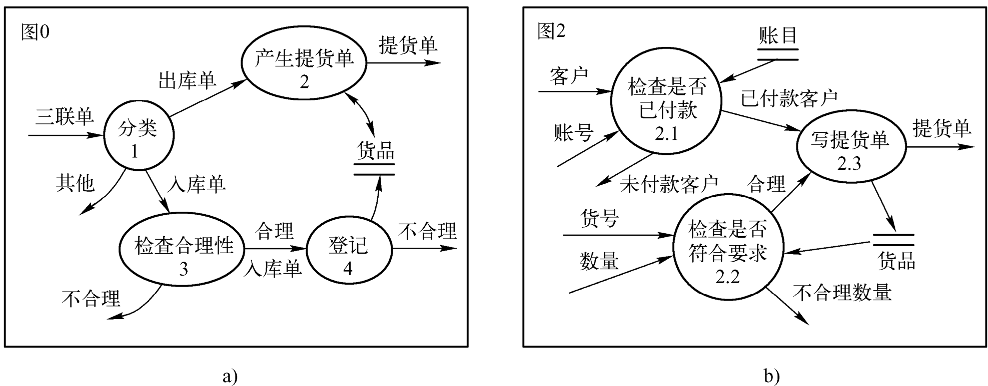


a) b)

图 2-7 父图和子图不平衡

a) 父图 b) 子图

若对加工处理和数据对象同时进行分解，则检查平衡需借助数据词典方能实现。如图 2-8，从图中看父图中“产生提货单”的输入和输出数据对象与相应子图的输入和输出不相同，但从数据词典中查出父图中“出库单”的数据对象是由“客户”、“账号”、“货号”和“数量”4 部分构成的，因此这两张图是平衡的。



a) b)

图 2-8 平衡的 DFD

a) 父图 b) 子图

在考虑平衡时，可忽略枝节性数据对象，如图 2-8b 中，“未付款客户”“不合理数量”属于枝节性数据对象，尽管它们在父图中没有出现，仍然可以认为图 2-8a 和图 2-8b 是平衡的。

使用分层 DFD 是为了控制复杂性，实现从抽象到具体的过渡，这需协调层数与每张图的数目，经验表明，人们在理解或处理 7 个以下的问题时效率较高，当超过 7 个时效率明显降低，因此，根据这一经验原则，可以让每张图中的加工处理不超过 7 个，以便能更好地理解系统，有效的控制 DFD 的复杂性，但也不是绝对的，以下 3 条原则可作为分解 DFD 的参考。

1）分解要自然，概念上要合理、清晰，这样才能便于理解。

2）只要容易理解，一张图就可以多包含几个加工处理，这样分层 DFD 的层数就会减少，相当于降低了整体复杂性。

3）由于越往上层抽象性越高，越往下层越具体，因此，可以让上层 DFD 分解得快些，下层 DFD 分解的慢些。

3．数据流图的改进

数据流图虽然是现实数据处理系统的真实再现，但在画的时候，应该加入理解和分析，应该有提炼和概括，最初是真实再现，完成后还需要改进和提高。就像画家写生一样，也有创作。画数据流图，就像写生作画，数据流图完成后，不应该立即进入数据词典的编写等其他工作，而应该像画家后期的精细刻画加工一样，要再检查 DFD 有没有错误，必要时可能需要重画或重新对 DFD 分解。概括地说就是全面检查、整体调整、有效改善、精益求精。

（1）检查 DFD 的正确性

在分析一个大型系统时，不可能一开始就对问题理解得很正确，需要有一个反复的过程，当把 DFD 画出来之后还需要修改和完善。为了保证 DFD 的正确性，可以从以下 3 个方面进行检查。

1）检查数据是否守恒。所谓数据守恒是指输入到某个加工处理的数据流与从该加工输出的数据流大小一样，就像水路中的水流和电路中的电流一样，某个节点的输入和输出应该一致。数据不守恒的情况有两种：一种是输入到某个加工处理的数据没有从这个加工处理输出；另一种是从某个加工处理输出的数据没有输入到该加工处理。前者也许是错误，也许不是错误。若不是错误，则可把多余的数据去掉，后者则一定有错误，说明在分析时漏掉了数据。参考图 2-9，查阅数据词典可知，运动员名册包含班级、姓名及比赛项目，数据对象“比赛项目运动员”包含班级、比赛项目、姓名及运动员编号，而“运动员编号”这一数据项并没有进入到“确定比赛名单”这个加工处理中，因而该加工处理的数据不守恒。

2）检查数据存储的使用是否正确。通过检查数据存储的使用是否正确也是查找 DFD 中错误的有效方式，若某个加工处理读数据，则从双线指向该加工处理的椭圆；若加工处理写数据，则从椭圆指向双线；若加工处理修改数据，则也是从椭圆指向双线。这样就很容易发现图 2-10 的错误。

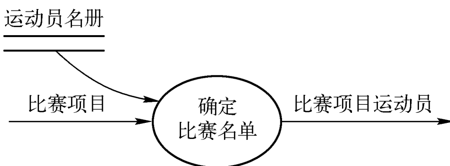


图 2-9　数据不守恒的例子

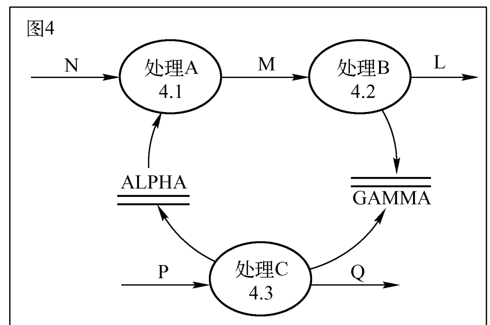


图 2-10　对数据存储 GAMMA 不正确的使用

3）检查父图和子图是否平衡。父图和子图不平衡是一种常见的错误，在数据流图的分解过程中不小心就会出现，为了避免此类错误，不但每分解一层数据流图时要检查，而且每当对 DFD 进行了修改后，都要检查父图和子图是否平衡。

（2）改进 DFD 的效能

就像画家作画，肖像画可以传神，但它首先以像所描绘的人物为前提，经过画家的思考加工，融入了“神”的元素，如果不像所描绘的人物，再高的技法也没有用。同样，改进 DFD 就像画肖像画，要以现实为基础，不能背离现实，在此基础上，再采用增加、删除、合并等手段，使数据流图逐渐完美。为使 DFD 容易理解，方便后面的设计、编程、测试、维护等工作，可以从以下 3 方面对 DFD 进行改进。

1）简化加工处理间的数据对象。数据流图中加工处理间的数据对象数目越多，彼此之间的依赖性就越强，理解起来也就越困难。反之，加工处理间的数据对象数目越少，则它们就越独立，就越容易理解。理想的分解是将一个问题划分成大小均匀的几部分，各个部分可单独的理解，这样一个复杂问题就被几个简单问题代替了。图 2-11 中，图 2-11a 中的加工处理 2 有 9 条数据对象与之联系，加工处理 2 的独立性低，对于这种图，应该重新分解。

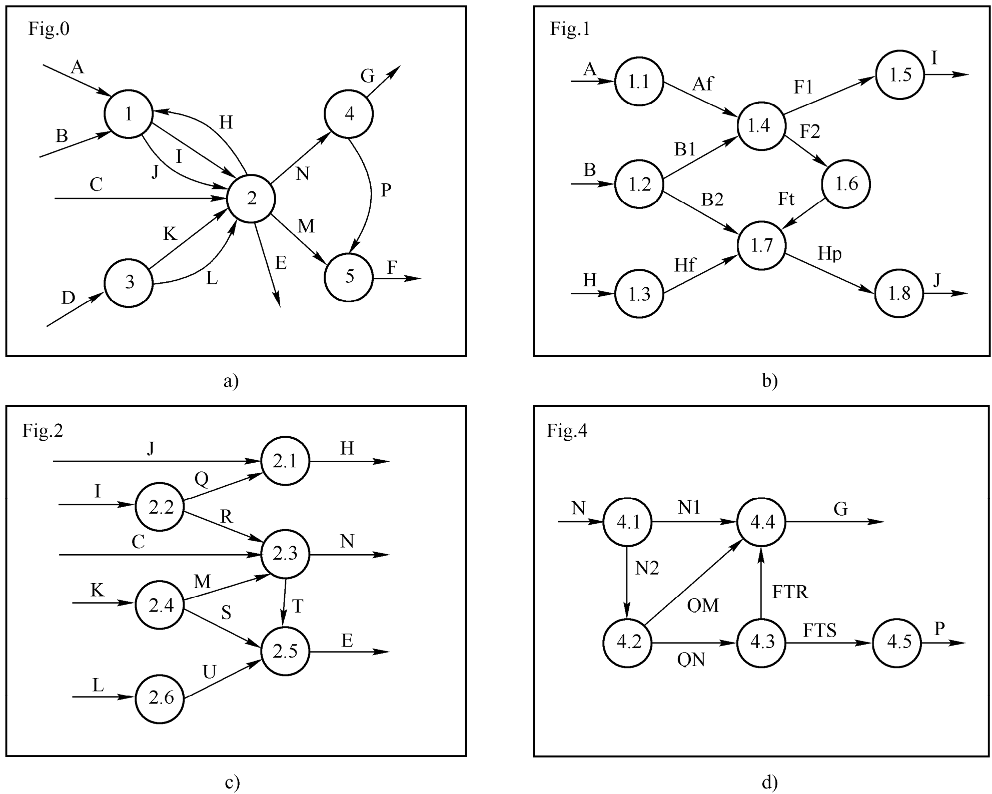


图 2-11　分解不均匀的 DFD

a) 父图　b) 子图　c) 子图　d) 子图

2）分解要均匀。理想的分解是将一个问题划分成大小均匀的几个部分，当然这点是不容易做到的，但是应当避免特别不均匀的分解，如果在一张图中某些加工处理已是基本的操作，而有些却要进一步分解成 3、4 层，这样的图就不容易理解，因为某些部分描述的是细节，而有些部分描述的却是较高层的抽象，遇到这种情况也应该重新分解。

3）命名要合适。数据流图中各成分的命名与 DFD 的“易理解性”直接有关，如果名字贴

切、到位，对后面的设计、实现、测试、调试、质量保证、管理及维护等都会有帮助，所以应该注意适当地命名。

请看下面几个加工处理的名字。

计算总工作量

写发票

存储和打印提货单

处理订货单

处理输入

做杂事

前两个名字的意义很明确，所以容易理解。第三个可将它分解成两个加工处理。后面几个名字就很不好，不易理解。因为“处理”是个很不具体的动词，它没有说明这个处理框究竟做什么。“处理输入”则具有双重的缺点，不仅动词空洞，它的宾语也不具体。“做杂事”就更差了，这相当于没有给加工处理命名。

理想的加工处理名应该由一个具体的动词加一个具体的宾语组成，在底层尤其应该用这样的方式来命名。

同样，对数据对象、数据存储也应适当地命名，尽量避免产生错觉，以减少设计、编写等阶段的错误。

如果难以为 DFD 中的某个成分取合适的名字，这往往是 DFD 中的加工及数据对象分解不当而造成的，此时可以考虑重新分解。

（3）重新分解 DFD

在许多情况下，需要对数据流图作重新分解，例如在画第 N 层时意识到在第 N−1 层或第 N−2 层所犯的错误，此时就需要对第 N−1 层、第 N−2 层作重新分解。整体上考察数据流图时如果发现某一层分解很不均匀，也应对这层和下层数据流图重新分解。

以下是重新分解的一种机械的做法。

1）把需要重新分解的某张 DFD 图的所有子图连接起来。

2）将连接好的 DFD 图重新划分成几部分，划分要尽量均匀，使各部分之间的联系尽量少。也就是说，判断一个加工处理应该属于哪部分，要根据它与各个部分之间的联系来确定。

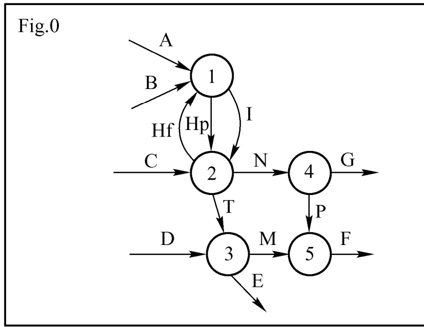
3）重新建立父图。将上一步中的每个部分用一个椭圆（即加工处理）来代替，并起个合适的名字，名字最好由一个具体的及物动词加一个具体的宾语构成，使其容易理解。数据对象及数据存储的名字一般应保持不变。

4）重新建立各子图，把第 2）步所确定的图中的每一部分画成一张 DFD 图。

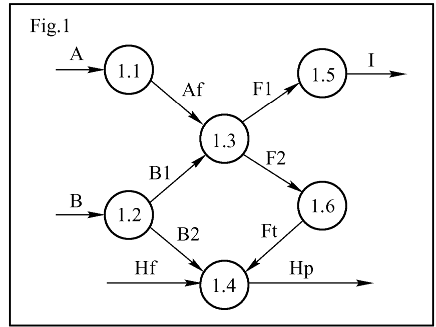
5）为所有的加工处理编号并命名，编号时按前面所讲的规则进行。

上述步骤完成后，要再整理一下，看看各个子图号与父图中的加工处理是不是对应，父图和子图是不是平衡，有没有漏掉数据对象和数据存储，等等。

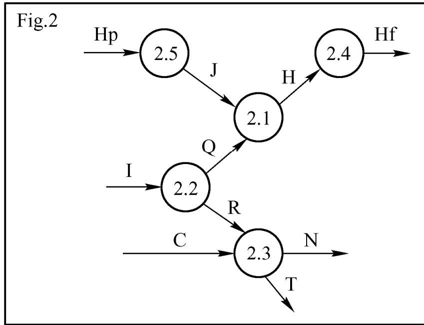
图 2-12 是按以上步骤对图 2-11 重新分解后得到的结果，它比原来简单，也更容易理解。但要注意，重新分解也不是万能的，一定要考虑问题本身的特性，概念上、逻辑上要合理，不能把一个本来是整体的成分强行分开，也不要把彼此没有关系的加工或数据合在一起，以免引起新的误解。



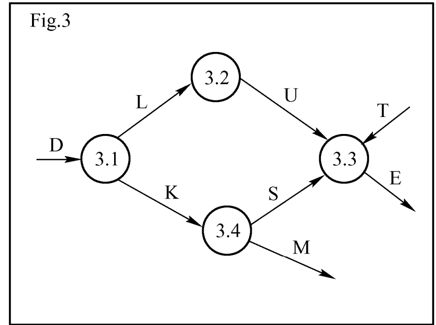
a)



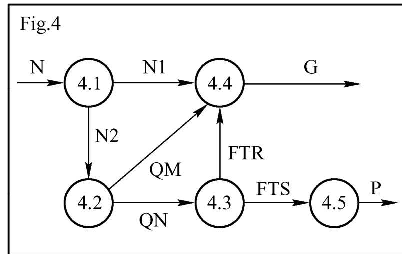
b)



c)



d)



e)

图 2-12　对图 2-11 重新分解后得到的 DFD

a)父图　b)子图　c)子图　d)子图　e)子图

2.1.2　数据词典及其描述

大家在读书看报时，当遇到不认识的字或词时会去查阅字典，通过字典，不但认识了这个词，知道了它的读音，而且还知道了它的用法、与其他词的搭配，等等。现实中的字典或词典是所有字或词的定义和解释，同样，数据词典是 DFD 中所有成分的定义，它是一个工具，同日常使用的词典的作用类似，异曲同工，如果不知道 DFD 中某个成分的含义，查找数据词典就能知道。

同日常所用的词典类似，对数据词典的基本要求是 DFD 中的所有名字在数据词典中都要出现，不能只定义一部分，定义顺序是：基本数据项（数据元素）→数据结构→数据对象→数据存储。定义时所用的词汇要准确，否则容易造成歧义，和日常使用的词典有所不同的是，数据词典中的一个条目只能有一个名字，一个名字只能有一个条目，不允许重复，

字典序也是很重要的，否则无法使用。

1．基本数据项的描述

基本数据项是构成数据结构、数据存储以及数据对象的基础，一般用两方面的信息来描述，一是给出它的名称，包括该名称的形式；二是给出说明，说明一般是名称的取值范围。

【例 2-1】 对数据项“日期”的描述。

名称：日期=（年，月，日）

说明：年=（2015~2025）

月=（1~12）

日=（1~31）

2．数据结构的描述

此处的数据结构其实是具体的数据构成，也用名称和说明来定义。说明部分显然比基本数据项的内容要多，它应该描述该数据结构还包含有哪些数据项或数据结构，取值方面有什么要求，等等。

【例 2-2】 对数据结构“付款通知单”的描述。

名称：付款通知单

说明：日期

付款人姓名

付款人住址

{支票|现金}　　　　　　　　　　（付款方式）

付款人银行名

付款人银行账号

{发票}$^{(1\sim n)}$

号码

“{支票|现金}”表示付款方式可以在“支票”或“现金”2 种方式中选择，“{发票}$^{(1\sim n)}$”表示可以有 1 至 n 张发票。

3．数据对象的描述

数据对象在 DFD 中用箭头来表示，数据对象其实是数据结构的“运动”，对它的描述应包括该数据对象所含有的数据结构、它从何处流出、流向何处、数据对象的“大小”以及数据对象产生的原因和结果。

【例 2-3】 对“未发货事项”数据对象的描述。

名称：未发货事项　　　　索引号：

说明：所含数据结构　　　信息量：规定每周不得出现五次

定货单

顾客目录

{书目}^{n}

数据对象来处：6　　　说明：检查存货单

数据对象去处：13　　　说明：建立回单及申请书

简要说明：未发货原因是顾客要求购买的数量品种不能满足要求

“{书目}^{n}”表示“订货单”中可以包含 n 个“书目”，数据对象“未发货事项”来自于图 0 中编号为 6 的加工处理，该加工处理的名称是“检查存货单”。数据对象“未发货事项”流向图 0 中编号为 13 的加工处理，该加工处理的名称是“建立回单及申请书”。此例用了卡片的形式来描述数据对象，DFD 中所有的成分都可用卡片的形式来描述，为了便于查找，可给出“索引号”，也可以定义一种形式化的符号，表 2-1 为 Pressman 给出的数据词典中所用的描述符号。

表 2-1　Pressman 给出的数据词典的描述符号

数 据 结 构	记　　号	意　　义
	=	由…构成
顺序	+	和
选择	[|]	或
重复	$\{\}^n$	n 次重复
	( )	可选择的数据
	**	限定的注释

4．数据存储的描述

数据存储在 DFD 中用双线表示，可以是数据记录、数据文件或数据库，对它的描述也应给出名称和说明，在说明部分可给出数据存储的组成、组织方式，数据项来自何处，送往何处等。

【例 2-4】 对“定期存款”的描述。

名称：定期存款

内容：账号

户名

地址

款额

存期

组织：以账号递增次序排列

存期=[1│3│5│8]

【例 2-5】 对“职工”的描述。

数据库名：职工

简要说明：包括专职职工的所有信息

组成：姓名

性别

年龄

婚姻状况

工号

开始工作日期

工资

部门

组织：按工号递增排列

2.1.3 功能说明

在 SRS 中的功能说明部分对 DFD 中的加工处理进行说明，说明的内容主要包括该处理如何读入数据对象、如何输出数据对象、输入和输出数据对象的逻辑关系，除此之外也可对加工处理的激发条件、优先级、执行频率、出错处理等细节进行说明。

SA 方法常用的说明方式有结构化英语、判定树和判定表 3 种。

1．结构化英语

日常生活中使用的是自然语言，自然语言的优点是容易理解，其缺点是具有歧义性和模糊性，形式语言的优缺点正好与自然语言相反，它的优点是严格精确，缺点是不容易理解，如自然语言“John gives a book to Mary”，但凡学过一点英语的人都能明白它的意思，若用形式语言来描述，则一阶谓词的形式为：

（∃x）（Give（John， x，Mary）∧Book（x））

用二元谓词表示则为：

ISA（G1，GIVING-EVENTS）

GIVER（G1，John）

RECIP（G1，Mary）

OBJECT（G1，Book1）

ISA（Book1，Book）

还可以增加一些信息，例如：

HUMAN（John）

HUMAN（Mary）

上述形式语言的描述虽然很严密，但难于理解。能否既要保持自然语言通俗易懂的优点，又让其像形式语言那么严密呢？于是，SA 方法的研制者们提出了结构化英语。结构化英语是介于自然语言和形式语言之间的一种语言，称为半形式语言，它是自然语言受约束和限制后得到的一个子集。虽不如形式语言精确，但具有自然语言简单易懂的优点，又避免了自然语言的某些缺点。

结构化英语具有如下的约束和原则。

（1）语句之间的联系，只使用如下结构：

IF______THEN______ELSE（SO）_____（CASE）

WHILE______DO______

REPEAT______UNTIL_____

（2）语态只能是简单祈使句（陈述句、疑问句、惊叹句均不行）。

（3）名词必须在词典中定义过。

（4）动词应明确具体（如 do，make，get，have，process，take 等尽量不用）。

（5）形容词、副词（如 very，often 等）尽量不用。

（6）注意 OR/AND 的使用。

（7）分清 GT（大于）、GE（大于或等于）、LE（小于或等于）和 LT（小于）的含义。

【例 2-6】 用结构化英语描述加工处理“GENERATE INVOICE”。

GENERATE INVOICE

  DO COMPUTE-INVOICE-TOTAL

```
    DO COMPUTE-DISCOUNT
    DO COMPUTE-SHIPPING-HANDLING
    Subtract discount from invoice-total to get invoice-net
    Add shipping-handling-fee to invoice-net to get total-payable
    Write invoice
COMPUTE-INVOICE-TOTAL
    REPEAT EXTEND-ITEM-LINE UNTIL all item-lines have been extended
    Add all item-line-totals to get invoice-total
EXTEND-ITEM-LINE
    Multiply quantity by unit-cost to get item-line-total
COMPUTE-DISCOUNT
    IF invoice-total is GE $1000
        THEN discount is 5% of invoice-total
        ELSE IF invoice-total is GE $250 but LT $1000
            THEN discount is 2 1/2% of invoice-total
            ELSE IF invoice-total is GE $100 but LT $250
                THEN discount is 1% of invoice-total
                ELSE (invoice-total is LT $100) SO discount is nil
COMPUTE-SHIPPING-HANDLING
    IF order specified air shipment
        THEN DO COMPUTE-AIR-FREIGHT
        ELSE (order specified surface shipment or method is open)
            SO DO COMPUTE-SURFACE-FREIGHT
    Multiply rate by current-unit-value to get shipping-handling-fee
COMPUTE-AIR-FREIGHT
    IF weight is LE 2
        THEN rate is 6 units
        ELSE IF weight is GT 2 but LE 20
            THEN multiply each pound of weight by 3 units to get rate
            ELSE (weight is GT 20)
                SO subtract 20 from weight to get excess
                Multiply excess by 2 units per pound and add 60 to get rate
COMPUTE-SURFACE-FREIGHT
    IF destination is local AND service-code is express
        THEN multiply each pound of weight by 2 units to get rate
        ……
And so on
```

简单解释一下，加工处理“GENERATE INVOICE”中包含 6 项操作，以 DO 开头用大写字母描述的操作需要进一步展开，以小写字母描述的操作是不再进一步分解的基本操作。

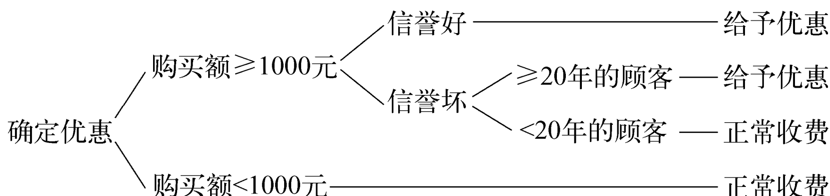
如果将英语用汉语代替，将上述结构化英语的 7 条约束和原则用于汉语，就可得出对 DFD 中加工处理的结构化汉语的描述。

2．判定树

判定树是树的一种，每个节点有两个分支，一个是条件成立（即“是”）；一个是条件不成立（即“否”）。每个分支都可以展开，直到确定了具体操作为止，这种表示形式，容易理解，适合于描述具有组合条件的加工处理。在画树的时候要进行分析，条件次序不一样，在“是”或“否”上展开不同，都会导致不同的判定树，虽然操作是一样的，但分析考虑周到后比提起笔就画而得出的判定树相对简洁。

【例 2−7】 图书销售系统中对加工处理“确定优惠”的说明。

若一次购买大于等于 1000 元并且信誉好，或虽然信誉不好但其是 20 年以上的老顾客则给予优惠。用判定树来描述，则非常简单。



3．判定表

判定表和判定树本质上是一样的，都是适合于具有组合条件的加工处理的说明，但当组合条件复杂时，用判定树来描述，将使树节点过多，分叉过多，不容易看清楚。若用判定表来表示，可能就清楚一些，因此，判定表更适合于描述具有复杂的组合条件的加工处理。

判定表由 4 部分组成，如图 2−13 所示，左上部分列出所有的条件，右上部分列出所有可能的条件组合，左下部分列出所有的动作（操作），右下部分说明在对应的条件下某个操作是否执行。如果条件有 n 个，则条件组合为 2^n，表 2−2 为【例 2−7】的判定表描述。其中 Y 表示条件满足，N 表示条件不满足，X 表示选中判定结论。表 2−3 为简化表，其中“—”表示条件任意，对于复杂的组合条件，简化表的优点更加明显。

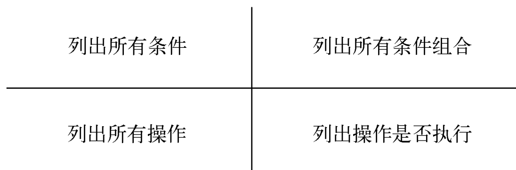


图 2−13　判定表的结构

表 2−2 【例 2−7】的判定表描述

	1	2	3	4	5	6	7	8
购买额≥1000 元	Y	Y	Y	Y	N	N	N	N
信誉好	Y	Y	N	N	Y	Y	N	N
≥20 年的顾客	Y	N	Y	N	Y	N	Y	N
给予优惠	X	X	X					
正常收费				X	X	X	X	X

表 2-3　表 2-2 的简化形式

	1	2	3	4
购买额≥1000 元	Y	Y	Y	N
信誉好	Y	N	N	—
≥20 年的顾客	—	Y	N	—
给予优惠	X	X		
正常收费			X	X

2.2　原型化分析方法

开发一款新型汽车，厂家往往先造一个模型，广泛地征求意见，并不断改进，直到成为真正的样品，然后才批量生产。软件工程原型化方法的思想与此类似，主要是针对事先不能完整定义需求的软件开发而提出的，开发人员根据用户的需求，先开发一个原型，让用户试用，用户提出改进、精化及增强系统能力的需求。开发人员根据用户的反馈意见，实施开发的迭代过程。这一过程反复进行，逐渐演化成最终的系统。每一迭代过程均由需求、设计、编码、测试、集成等阶段组成，如果在一次迭代中，有的需求不能满足用户的要求，可在下一次迭代中进行修正。需要注意的是，软件是信息产品，与物理产品的原型制造思想虽然一致，但演化过程不一样。

软件开发中的一个重要风险来自于需求错误和需求遗漏，软件工程经济学家 Boehm 的实验表明，原型能减少需求描述中出现问题的数量，总开发成本在有原型系统的情况下要比没有原型系统的低。原型系统是需求工程过程中的一个组成部分，一个软件原型支持需求工程过程中的两项活动。

1．需求的导出

系统原型虽然不是最终系统，但允许用户在上面进行实验，以便了解未来系统是如何支持他们的工作的。在用户的使用实验过程中，用户可能产生有关需求的许多新的想法，同时发现系统的优点和不足，进而提出新的系统需求。

2．需求的有效性验证

原型系统可以暴露出错误和遗漏的东西。一个经过描述的功能可能是很有用且已经是定义了的，但是，当这个功能模块与其他模块一起工作时，用户可能会发现他们的初始想法是错误的或是不完善的，必须修改系统描述以反映对需求的新的理解。

除了允许用户改进需求描述以外，开发系统原型还有如下的其他好处。

（1）软件开发人员和用户之间的理解偏差可在功能展示时显露出来。

（2）软件开发小组可能会在原型设计中发现需求的不完善和不一致。

（3）可以迅速展示一个应用系统对管理的可行性和作用。

（4）原型可以用作书写产量-质量系统描述的基础。

原型一旦开发出来，还可以用于用户培训和系统测试。用原型化方法在软件开发初期成本会增加，主要原因是由于用户复用了效率较差的原型代码，而这些代码导致了整个系统性能的降低。在开发后期成本会降低，因为在迭代过程中开发效率会越来越高。

2.2.1 开发模型

原型开发过程可用如图 2-14 所示的简要模型来表示，这是英国软件工程专家 Ian Sommerville 给出的模型。在开发过程的一开始就要明确原型开发的目的，或是对用户界面的原型设计，或是为系统功能需求进行的有效性验证，还可能是为了说明应用系统管理上的可用性。

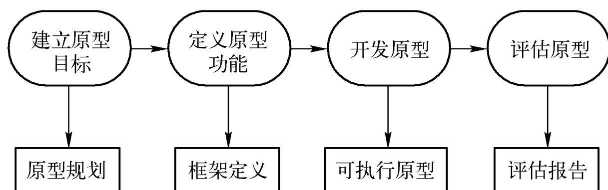


图 2-14　原型开发过程的宏观模型

开发过程的下一步就是确定哪些东西要加到原型系统中，更为重要的是，确定哪些应该从原型中除去。为了降低原型开发的费用和加快开发进度，需要抛开一些功能模块，也可以放松一些性能要求，如响应时间和内存耗费，同时可以对错误处理和管理忽略或是做简单处理。除非设计原型的目标是建立用户界面，否则可靠性标准和程序质量也不予考虑。

最后阶段的工作是原型的评估。对用户培训的有关规定要在这一阶段给出，同时要基于原型的设计目标导出评估计划。用户需要一定的时间来适应新系统并逐渐使使用方式变得规范化。一旦使用方式规范了，他们就能发现需求上的错误和纰漏，从而提出改进的需求。

原型开发过程模型的具体形式也是多样的，如抛弃式（也称为丢弃型）、演化式（也称为样品型）、增量式（也称为渐增式）等。

1．抛弃式原型开发

基于抛弃式原型开发的软件过程模型如图 2-15 所示。这种方法在降低总的生存期成本的情况下，增强了需求分析过程。在这种开发方法中，原型的根本作用是弄清楚需求并为管理人员评估过程风险提供额外的信息。经过评估，原型被抛弃，不再作为系统开发的基础。

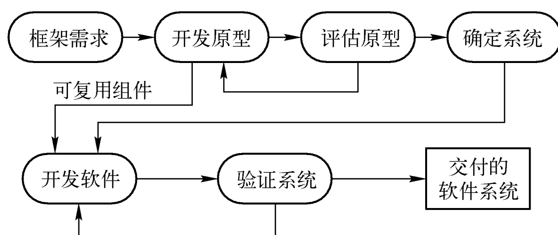


图 2-15　抛弃式原型开发模型

这种方法在硬件系统开发中用得较多，在开发一个昂贵的硬件系统之前，原型用作设计验证。一个电子系统的原型往往利用现成的电路组件来做。正式投产时，再制作专门用途的集成电路来实现该系统。

抛弃式软件原型用于提供系统需求，这种原型通常不作为设计有效性验证，原型与最终

系统差距很大。原型必须尽快拿出来，以使用户能尽早反馈对系统描述的意见。抛弃式原型中的功能经过原型设计而得到深刻理解，但质量标准和性能指标在原型中被忽略，原型开发和最终系统开发使用的语言也往往不一样。

在图 2-15 中，假设原型是从粗略的系统描述开始的，接着进行交付试验，然后再修改，直到用户对其功能满意为止。在这一阶段，阶段性的过程模型被采用，从原型中提炼需求，而最后系统却要重新建立。原型中的组件也许会用于最终系统中，这样能够降低一些开发成本。原型除了能导出系统描述之外，有时原型实现本身就是系统描述。

抛弃式原型开发的主要问题如下。

1）为了尽快拿出原型，可能对系统做了很多简化，因而不可避免地会漏掉一些重要特征，有些对安全要求很高的系统很难用原型来表现其中的某些重要部分。

2）在用户和开发人员之间没有一个能写进合同的、对于原型实现的合法规定。

3）非功能需求，如可靠性、鲁棒性和安全性，在原型实现中得不到充分体现。

开发一个可能执行的抛弃式原型通常遇到的问题是：原型的使用方式和最终系统的使用方式可能不一样。

抛弃式原型在需求工程过程中并不一定是很有用的可执行的软件原型，纸上的用户界面模型在帮助提炼界面设计和设计使用情景时也十分有效，也很经济，在几天之间就可以完成。

2．演化式原型开发

演化式原型开发过程模型如图 2-16 所示，先给出一个系统的最初实现，就像其他工程产品的一个样品，让用户使用和评论，之后进行不断细化和完善，经过多次反复后形成最后的应用系统。

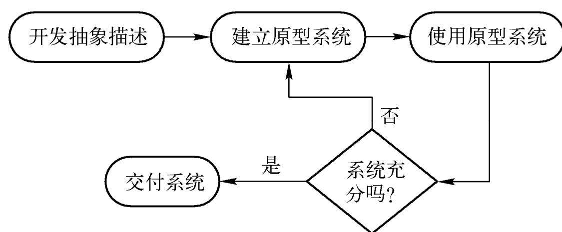


图 2-16　演化式原型开发过程模型

演化式原型开发主要有两个方面的优势，一是可加快系统交付的进度，在某些情况下，快速交付软件可能比提供完备的功能或保证长期的可维护性更重要；二是用户能够参与，用户在软件开发过程中的介入可使系统开发人员更好地理解软件需求，并使用户逐渐喜欢系统，在工作中依赖它。

用演化式原型开发方法对于大规模、开发周期长的系统是重要和有效的方法，在使用这种方法时要注意 3 方面的问题。

（1）管理问题

大型软件系统的开发需有专门管理机构处理软件过程模型，软件过程模型定期产生可交付的文档来评估项目的进展状况，由于原型开发太快，会产生大量的系统文档，从而增加成本。此外，快速原型开发可能需要一些不熟悉的技术。

（2）维护问题

由于要不断地对原型进行修改，因而可能会引起系统结构的崩溃，如果某个开发人员不是一开始就参与到项目中，他就很可能难以理解系统。而且，如果快速原型开发中使用了某种专门技术，这种技术可能会过时，不再被使用，这使得以后再寻找具有相关知识的人来维护系统变得十分困难。

（3）契约问题

用户和开发者之间正规的模型契约是基于系统描述的。没有这样的描述很难拟定该系统开发的合同。如果一份合同只约定开发时间并按照该时间计算开发费用，付给开发人员，则用户是不满意的。因为这可能会引起系统功能性的降低，使开发费用过高，开发人员也不愿意接受一个固定价格的合同，因为他们无法控制最终用户不断改变的需求。

从这些困难可以看出，使用演化式原型开发技术要有一个现实的态度，允许从一个小型或中等规模的系统做起，以缩短系统的交付日期。要降低开发成本，就要尽量提高可用性。如果用户参与开发，则原型就会贴近真实需求，软件系统的生命周期会相对缩短。随着维护问题的增加，系统可能不得不被替换或彻底重写。对于大型系统的开发，可能有很多开发机构和开发人员，演化式开发中的管理问题将非常突出，变得难以驾驭。系统中的部分原型可能是抛弃式原型。

3．增量式原型开发

基于增量式的原型开发过程模型如图 2-17 所示，它避免了演化式原型开发中的经常性变更问题。一旦建立了一个总的系统体系结构，就可在早期阶段作为系统的总框架。一旦该框架得到认可，除非在以后开发过程中发现错误，否则就不对框架和组成部分作任何改动，但用户对各交付组件的反馈意见会影响后续交付的组件的设计。

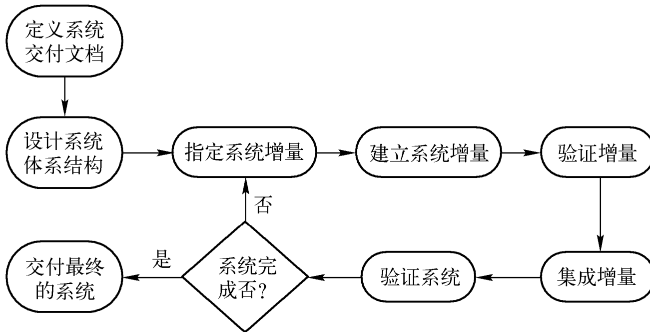


图 2-17　增量式原型开发过程模型

增量式原型开发是建立在软件总体设计基础之上的，而演化式原型开发的设计则是不断发展的。在增量式原型开发中，首先要进行总体设计，然后完成模块设计，并顺序增加。在增量开发中，必须给出每一部分的计划和文档，这就保证了能够及时得到用户的反馈，从而限制了错误的出现。增量式原型开发方法主要影响实现阶段，与演化式原型开发比较，它能修改的范围较小，因而具有易于控制和管理的优点。

原型化方法的主要特点是突出一个“快”字，用户可以很快看到未来系统的“样品”，但它存在的问题也比较严重。一方面，为了让用户快速看到系统“样品”，开发人员常常使

用不适当的开发环境、编程语言以及效率不高的算法，而这些有可能集成到系统中，成为实际系统的一部分；另一方面，由于时间紧迫，构造原型时来不及考虑软件的整体质量和系统以后的可维护性问题，由此，可能造成开发出的软件质量不高。

2.2.2 快速原型技术介绍

快速原型技术强调了交付的速度，而不是系统的性能、可维护性和可靠性。目前，有 3 种比较实用的快速原型技术。

1．使用动态高级语言开发应用系统

动态高级语言包含运行时的数据管理功能，使用的硬件成本也相对较低。由于减少了许多存储分配和管理中的问题，因而极大地简化了程序的开发，这些语言中的功能一般都是用像 Ada 或 C 语言中的基本结构来构造的，如基于表结构的 Lisp 语言，基于逻辑的 Prolog 语言以及基于对象的 Smalltalk 语言等。

到目前为止，非常高级的动态语言并没有广泛应用于大型系统中，因为大型系统需要大型运行支撑环境。大型运行支撑环境对存储的需求高，且降低了用这些语言编写的程序的执行速度。

对于很多商业应用系统的开发，动态高级语言可以代替诸如 C、COBOL、Ada 等一类的语言。Java 是一种主流的开发语言，它源于 C++语言，还具有许多 Smalltalk 语言的特征，如平台独立性和自动存储管理等。Java 具有许多高级语言的特征，还具有传统的第 3 代语言所拥有的性能优化的能力。目前，已有很多的可复用 Java 组件，显然，它是一种非常适合演化式原型设计的语言。

表 2-4 为常用于原型开发的动态语言。每当选择一种原型开发语言时，需要问以下几个问题。

（1）问题的应用领域是什么？

如表 2-4 所示，不同的语言适合不同的领域。如果是有关自然语言处理的应用，那么用 Lisp 或 Prolog 比用 Java 和 Smalltalk 更合适。

表 2-4　原型开发中常用的高级语言

语言	类型	应用领域
Smalltalk	面向对象	交互式系统
Java	面向对象	交互式系统
Prolog	逻辑	符号处理
Lisp	基于列表	符号处理

（2）需要与什么样的用户交互？

不同的语言对用户交互的支持能力不一样，许多语言如 Smalltalk 和 Java 与 Web 浏览器集成得相当好，而其他语言如 Prolog，较适合文本界面。

（3）语言提供的支撑环境如何？

成熟的支撑环境可以包含许多的支持工具，而且有很多可复用的组件，这些将大大简化原型开发过程。

实际当中可以混合使用动态高级语言来构建系统原型，先将系统的不同部分用不同的语言

编写，然后建立不同部分之间的通信框架，如某电话网络系统原型开发中使用了 4 种语言，Prolog 用于数据库原型开发，Awk 用于记账，CSP 用于协议描述，PAISLey 用于性能仿真。

由于大系统的不同部分之间差异很大，因此，没有一种语言能够适合系统构造过程中的所有部分。用多种语言来共同实现一个大系统的优点在于对应用的逻辑部分可以选择一种最适合的语言，从而加快原型开发的速度，但不足之处在于建立一个能够在多种语言之间的通信框架比较困难。由于不同语言中的实体差异很大，所以需要大量的代码用于将一种语言中的实体转换到另一种语言中去。

2．使用第 4 代语言开发数据库管理系统

由于绝大多数商业应用软件系统都涉及数据库中的数据的操作，因而现在所有的商用数据库管理系统都支持数据库程序设计。数据库程序设计使用的是专门语言，这类语言内嵌有数据库知识和数据库的操作方法，其支撑环境提供界面定义的工具、数字计算工具以及报告生成工具，第 4 代语言（4GL）指的就是数据库程序设计语言和支撑环境。

4GL 环境提供的工具如图 2-18 所示，包括以下几种。

（1）数据库查询语言。主要是 SQL，其中的命令可以直接输入，也可以由用户填写的表格自动生成。

（2）界面生成器。可以创建用于数据输入和显示的表格。

（3）分析和操作的记录。

（4）报告生成器。用于定义和创建来自数据库的信息报告。

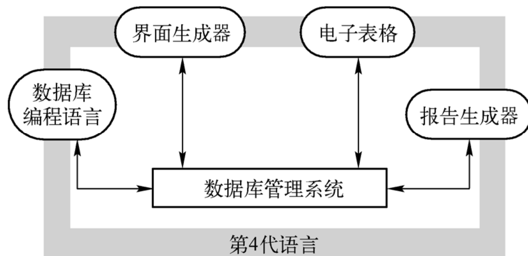


图 2-18　4GL 组件

绝大多数 4GL 建立了基于 WWW 浏览器的数据库界面，这使得只要在有互联网的地方就可以对数据库进行访问，这就大大降低了培训成本、软件开发成本，还允许外界用户访问数据库，但是 Web 浏览器固有的局限性以及互联网协议的因素，使得这一方法对那些需要极快交互响应速度的系统来说不太合适。

基于 4GL 的开发既可用于演化式原型系统的开发，也可以结合到基于方法的分析中去。基于方法的分析用系统模型来产生原型系统。用 CASE 工具生成的应用结构及相关文档使得用这种方法生成的演化式原型比手工开发的系统更容易维护。CASE 工具可以生成 SQL 或其他传统程序设计语言写成的代码。

尽管 4GL 很适合原型的开发，但是把它用于产品软件的开发却有很大的不足，原因是用 4GL 写成的代码，其执行速度比使用传统程序设计语言写成的程序的执行速度慢，而且对内存的需要量也大于传统程序设计语言。有人做过试验，把用 4GL 写的程序改用 C++重写，结果使内存的使用量降低了一半，而运行速度却比 4GL 提高了 10 倍。

用 4GL 书写的程序是非结构化的，难以维护。另外，4GL 还未标准化，因此交付后的维护成本可能会很高。而当用 4GL 开发的系统不得不重建时，就会出现一些特别的问题。4GL 之间缺乏标准和一致性使得用户必须重写系统，因为它们先前用的语言可能已经过时，不再使用。

3．使用组件复用技术通过集成来构造应用软件系统

使用这种技术开发需要在系统描述中说明哪些可复用组件是可利用的，这就需要对需求给出一个折中方案，一般地，有效组件的功能并不能与用户需求完全吻合，因而需要对用户需求调整。大多数情况下，用户需求是可以调整的，因此这一方法可用于原型的开发。

可复用组件的组成如图 2-19 所示，如果系统中许多部分都可以复用而且不需要重新设计和实现，则系统的开发时间将会缩短，若有许多可复用组件以及组合这些组件的机制，那么原型的构造就会快很多。这种组合机制必须包括控制设施以及组件之间通信的机制。

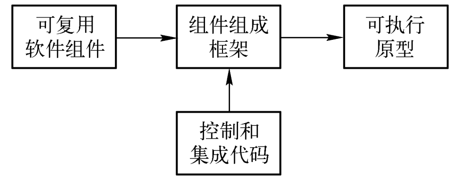


图 2-19　可复用组件的组成

基于复用的原型开发在如下的两个层次上实现。

（1）应用层

整个应用系统与原型结合在一起，功能模块可以共享。例如，如果原型需要一个文本处理功能，则可以通过在其中集成一个标准的文字处理系统来达到这种功能。

（2）组件层

单个组件集成进标准的框架形成系统构造。标准框架可以是设计用于演化式开发的脚本语言，如 Visual Basic，也可以是通用的基于 CORBA、DCOM 或 Java Beans 的组件集成框架。

应用层的复用可以获得应用中所有的功能，如果该应用又提供脚本或改编功能（如 Excel 的宏），则也可以用于开发原型系统的功能。

对一个应用完整地复用并不总是可行的或合理的，基于复用的开发还依赖于最小粒度（finer-grain）的可复用组件，这些组件可能是函数或是带有诸如排序、搜索、显示等特别操作的对象。原型系统定义一个总的控制结构，然后将各功能组件集成到该结构中来，如果没有满足需要的功能组件可供使用，就要开发一个组件，该组件还可以用于以后的系统构造。

可视化开发系统，如 Visual Basic，在应用开发中支持这种复用方法，程序员交互地建立系统，通过屏幕、域、按钮和菜单来定义用户界面。先是对这些界面元素进行命名，然后将处理脚本与这些界面元素相连，这些脚本可能去调用可复用的组件，或者调用一段专门设计的代码或两者都有。

脚本语言是一些非模式的高级语言，用于通过组件集成来构造系统。脚本语言通常包括控制结构和图形工具集，使用脚本语言将大大缩短系统的开发时间。UNIX 的 Shell 语言是早期的典型脚本语言，以后功能更强大的脚本语言不断出现。

基于复用的可视化编程方法特别适合于个体或小规模团体进行小型、简单系统的快速开发软件。对于大型系统，需要人员多，项目的组织困难，很难有一个明确的系统体系结构，系统中不同部分之间的关联相当复杂，这些问题使系统的变更非常困难。对于大型软件系统的开发，可将交互限定在一个特别的对象集上，但是这样做对于建立非标准的用户界面比较麻烦。

2.2.3 用户界面开发

用户界面的开发是原型化方法的重要内容，在软件系统的开发中，对用户界面的描述、设计和实现占有相当大的比重。有人做过统计，平均 60%的程序代码用于用户界面，用户界面是人与系统交互的媒介，开发好会提高使用效率，以用户为中心进行设计的观点强调用户在系统开发过程中要从始至终地参与、并强调原型开发的重要。目前，图形化用户界面已成为交互式系统的标准界面形式。

原型开发是用户界面设计过程的基本部分，由于用户界面的动态特性，文本的描述和图示不足以清楚表达用户界面需求，采用演化式原型方法并让用户参与是唯一可行的开发软件系统图形化界面的方法。

界面生成器是一个图形化屏幕设计系统。界面组件如菜单、域、图标以及按钮可以从菜单中选择并放到界面合适位置完成，这类系统是数据库程序设计系统的重要组成部分。Visual Basic 就是基于这类系统实现的标准开发技术。由于界面生成是基于描述来创建出良好结构的程序，而演化式开发的迭代过程不会损坏这种软件结构，因而无须任何返工。

由于 WWW 浏览器支持页面定义语言（HTML），因而成为一种可理解的用户界面描述语言。按钮、域、表单以及二维表格都可包含在 Web 页面中。同时，多媒体对象可以使人们获得声音、影像甚至虚拟现实的显示。处理脚本可以关联到用户界面的对象上，这样无论是在 Web 的客户端还是在服务器端，都可以通过对该对象的选取来执行相应的处理功能。

由于 Web 浏览器的有效使用和 HTML 强大的表示和处理能力，使越来越多的用户界面设计成基于 Web 的界面。基于 Web 的用户界面可以用标准的网站设计软件来设计，网站中用户界面编辑器其实就是界面生成器。Web 页面中实体的定义、摆放以及相关的动作连接都是这种网页编辑器内嵌的页面定义语言（HTML）能力的体现（如链接到另一个网页），制作基于 Web 的界面也可以使用 Java 或 CGI 脚本。

2.3 面向对象建模及 UML 方法

面向对象技术是 20 世纪 80 年代问世的，它的模块性、封装性、继承性、多态性和动态绑定能满足软件工程要求的局部化、易维护、可重用、易扩充以及当今多媒体和分布式计算的诸多要求。80 年代末以来，出现了几十种面向对象方法，其中 Coad/Yourdon、Booch、OMT、Jacobson、Wirfs-Brock 的方法得到了广泛的认可。统一建模语言 UML 结合了 Booch、OMT 和 Jacobson 方法的优点，统一了符号体系，并从其他方法和工程实践中吸收了许多经过实际检验的概念和技术，UML 方法已于 1997 年被对象管理组织（Object Management Group，OMG）正式确定为面向对象方法的国际标准，目前，UML 方法已经得到广泛使用。

2.3.1 面向对象基本概念

1．对象及对象模型

对象（Object）是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。一个对象由一组属性和对这组属性进行操作的一组服务构成。属性和服务是构成对象的两个主要因素，属性是用来描述对象静态特征的一个数据项，服务是用来描述对象动态特征（行为）的

一个操作序列。一个对象可以有多项属性和多项服务。一个对象的属性和服务被结合成一个整体，对象能够保存属性（信息）并提供一系列操作服务来读取或者改变这个属性。

2．类和实例

在面向对象的方法中，类（Class）是具有相同属性和服务的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和服务两个主要部分。类代表的是一种抽象，是代表对象本质的、主要的、可观察的行为。类给出了属于该类的全部对象的抽象定义，而对象则是符合这种定义的一个实体。因此类能创造新的对象，并且每个对象属于一个类，通常把属于某个类的一个对象称为该类的实例。

事物（对象）既具有共同性，也具有特殊性。运用抽象的原则舍弃对象的特殊性，抽取其共同性，则得到一个具有广泛共性的类。如果在这个类的范围内，来考虑具有某些特殊性的一组事物的话，这些事物也可以被抽象为另一个类。这个新类不仅具有前一个类的共性，同时它还有自己特殊的属性。通常，把前一个类称为“基类”（或“父类”），把后来生成的那个类称为“派生类”（或“子类”）。

实例的行为和属性由类定义，每个实例有一个唯一的标识。不同的实例可以由同一个类产生，每个实例由定义在类上的服务操作。不同的实例可以被不同的操作序列操纵，因而会有不同的内部状态，如果这些实例以完全相同的方式操纵，它们的状态也会相同。

3．封装

封装（Encapsulation）是面向对象方法的一个重要原则。它主要包括两层意思，第一是指把对象的全部属性和全部服务操作结合在一起，形成一个不可分割的整体；第二是指对象只保留有限的对外接口使之与外界发生联系，外界不能直接地访问和存取对象的属性，只能通过允许的接口操作，这样就尽可能地隐蔽了对象的内部细节，对外形成了一个保护的边界。

4．继承

继承（Inheritance）是类的特性。子类中不必重新定义已在它的父类中定义过的属性和行为，而它却可以自动地、隐含地拥有父类的所有属性和行为。例如在类“人”中，描述那些对“男人”和“女人”都是相同的属性和行为，将他们的共同特征都收集到类“人”中，并让其他的类继承这个类。如“男人”和“女人”继承“人”。这样只需要在“男人”和“女人”的类定义中，描述那些对于类“人”来说是新的特征。从而，使“男人”和“女人”这两个类获得类“人”中定义的所有特征。继承的实现则是通过面向对象（Object Oriented，OO）系统的继承机制来保证的。

继承关系是可以传递的。通过继承，能够简化对程序的修改。比如想修改“人”中的某种特征，只需在类“人”的定义中做修改就足够了。

通过提取和共享公共的特征，能够把公共性的类放到继承层次较高层。同样想加一个新类，可能会找到一个类，它已经提供了一个新类所需要的一些操作和信息结构，这时，可以让新类继承这个类，并且只加上那些只有新类才有的属性和操作。在继承层次中位于一个类下面的那些类叫作这个类的子类，位于一个类上面的类叫作它的父类。因为继承具有传递性。所以，一个类实际上继承了层次结构中在其上面的所有类的全部描述。这样，属于某个类的对象除具有该类所描述的特征外，还具有层次结构中该类上面所有类的描述的全部特征。

继承关系是一个类关系，即是类之间的关系。在类的层次结构中，一个类可以有多个子

类，也可以有多个父类。因此，一个类可以直接继承多个类。这种继承方式称为多重继承（Multiple Inheritance）。如果限制一个类最多只能有一个父类，则一个类至多只能继承一个类，这种继承方式称为单继承或简单继承（Single Inheritance）。在简单继承情况下，类的层次结构为树结构，而多重继承是网状结构。

5．消息

对象通过它对外提供的服务接口在系统中发挥自己的作用。当系统中的其他对象（或其他系统成分）请求这个对象执行某个服务时，它就响应这个请求，完成指定的服务所应完成的职责。在 OO 方法中把向对象发出的服务请求称作消息（Message）。对象之间是通过消息来进行通信的，这也是 OO 方法中的一个原则，它与封装的原则有密切的关系。封装使对象成为一些各司其职、互不干扰的独立单位。消息通信则为它们提供了唯一合法的动态联系途径，使它们的行为能够相互配合，构成一个有机的运动的系统。

6．结构与连接

在任何一个较为复杂的问题域中，事物之间并不是相互孤立、各不相关的，而是具有一定的关系，而因此构成一个有机的整体，从而使对象之间的交互与合作构成更高级的（系统的）行为。为了使系统能够有效地映射问题域，系统开发者需认识并描述对象之间的以下几种关系。

1）对象的分类关系。

2）对象之间的组成关系。

3）对象属性之间的静态联系。

4）对象行为之间的动态联系。

OO 方法运用一般-特殊结构、整体-部分结构、实例连接和消息连接描述对象之间的以上的 4 种关系。

一般-特殊结构又称为分类结构（Classification Structure），是由一组具有一般-特殊关系（继承关系）的类所组成的结构。它是一个以类为节点，以继承关系为边的连通有向图。

整体-部分结构又称作组装结构（Composition Structure），它描述对象之间的组成关系，即一个（或一些）对象是另一个对象的组成或部分。客观世界中存在着许多这样的现象。一个整体-部分结构由一组彼此之间存在着这种组成关系的对象构成。

整体-部分结构有两种实现方式。第一种方式是用部分对象的类作为一种广义的数据类型来定义整体对象的一个属性，构成一个嵌套对象。第二种方式是独立地定义和创建整体对象和部分对象，并在整体对象中设置一个属性，它的值是部分对象的对象标识，或者是一个指向部分对象的指针。在第二种方式下，一个部分对象可以属于多个整体对象，并具有不同的生存期。第二种方式便于表示比较松散的整体-部分关系。

实例连接（Instance Connection）反映对象之间的静态联系。例如教师和学生之间的任课关系、单位的公用汽车和驾驶员之间的使用关系等，这种双边关系在实现中可以通过对象（实例）的属性表达出来，所以这种关系称作实例连接。

消息连接（Message Connection）描述对象之间的动态联系，即若一个对象在执行自己的服务时，需要通过消息请求另一个对象为它完成某个服务，则说第一个对象与第二个对象之间存在着消息连接。消息连接是有向的，从消息发出者指向消息接收者。

一般-特殊结构、整体-部分结构、实例连接和消息连接均是面向对象分析（Object-

Oriented Analysis，OOA）与面向对象设计（Object-Oriented Design，OOD）阶段必须考虑的重要概念。只有在分析、设计阶段认清问题域中的这些结构与连接关系，编程时才能准确而有效地反映问题域。

7．多态性

对象的多态性（Polymorphism）是指在一般类中定义的属性或服务被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为，这使得同一个属性或服务名在一般类及其各个特殊类中具有不同的语义。

2.3.2 面向对象建模

医生在给病人看病时，如果只从表面观察一下就下结论可能会不确切，医生不但要对病人用听诊器听，还要对其进行透视、做 CT、化验等一系列检查，可能就会准确地诊断出病人是否有病、得的是什么病，从而给出治疗方案。面向对象建模，就像对人进行全面体检，为了全面了解需求，面向对象方法要进行一系列的建模，需求模型实际上是一组模型，就像从不同方面去诊断人体，需求模型是系统的第一个技术表示。

需求建模的重要性不言而喻，需求模型必须实现 3 个主要目标：1）描述客户需要什么；2）为软件设计奠定基础；3）定义在软件完成后可以被确认的一组需求。分析模型在系统级描述和软件设计之间建立了桥梁。这里的系统级描述给出了在软件、硬件、数据、人员和其他系统元素共同作用下的整个系统或商业功能，而软件设计给出了软件的应用程序结构、用户接口以及构件级的结构。这个关系如图 2-20 所示。

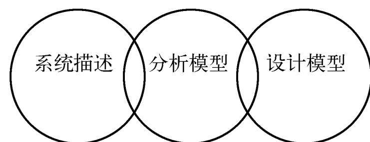


图 2-20　分析模型在系统描述和设计模型之间起到桥梁作用

需求分析产生软件工程特征的规格说明，指明软件和其他系统元素的接口、规定软件必须满足的约束。需求分析让软件工程师细化在前期需求工程的起始、导出、谈判任务中建立的基础需求。

在需求建模期间可以建立以下一种或多种模型类型。

1）场景模型：出自各种系统“参与者”观点的需求。

2）数据模型：描述问题信息域的模型。

3）面向类的模型：表示面向对象类（属性和操作）的模型，其方式为通过类的协作获得系统需求。

4）面向流程的建模：表示系统的功能元素并且描述当功能元素在系统中运行时怎样进行数据变换的模型。

5）行为模型：描述如何将软件行为看作是外部“事件”后续的模型。

这些模型为软件设计者提供信息，这些信息可以转化为结构、接口和构件级的设计。最终，在软件开发完成后，需求模型以及需求规格说明就为开发人员和客户提供了评估软件质量的手段。

1．基于场景建模

为了更好地了解最终用户希望如何与系统交互，首先应使用 UML 从开发用例、活动图及泳道图形式的场景开始。

（1）创建初始用例

“用例”只是帮助定义系统之外存在什么“参与者”以及系统应完成什么功能。本质上用例捕获了信息的产生者、使用者和系统本身之间发生的交互。用例从某个特定参与者的角度出发，采用简明的语言描述一个特定的使用场景。如果想让用例在需求建模阶段提供价值，那么必须回答并重点关注 4 个内容：1）编写什么？2）写多少？3）编写说明应该多详细？4）如何组织说明？

两个首要的需求工程工作是起始和导出，它们提供了开始编写用例所需要的信息。开始开发用例时，应列出特定参与者执行的功能或活动。这些可以从所需系统功能的列表通过与用户及所有相关人员的交流，或通过评估活动图来获得。

（2）细化初始用例

为全面理解用例描述功能，对交互操作给出另外的描述是非常有必要的。因此，要通过如下提问对初始用例主场景中的每个步骤进行评估。

1）在这一状态点，参与者能进行一些其他动作吗？

2）在这一状态点，参与者有没有可能遇到一些错误的条件？如果有可能，这些错误会是什么？

3）在这一状态点，参与者有没有可能遇到一些其他的行为（如被一些参与者控制之外的事件调用）？如果有，这些行为是什么？

这些问题的答案导致创建一组次场景，次场景属于原始用例的一部分但是表现了可供选择的行为。

（3）编写正规的用例

前面表述的非正规用例对于需求建模常常是够用的。但是，当一个用例包括关键活动或描述一套具有大量异常处理的复杂步骤时，就会希望采用更为正规的方法。

在很多情况下，不需要创建图形化表示的用户场景。然而，当场景比较复杂时图形化的表示更有助于理解。图 2-21 为 SafeHome 产品描述的一个初步的用例图。

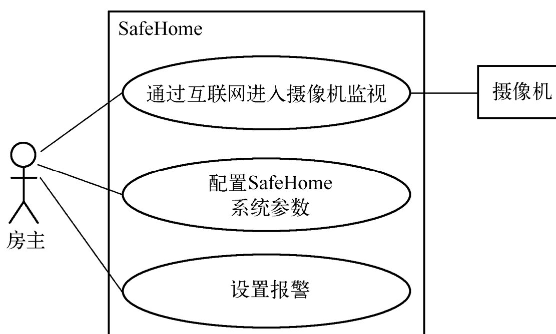


图 2-21　SafeHome 系统的初步用例图

每一种建模注释方法都有其局限性，用例方法也不例外。和其他描述形式一样，如果描述不清晰，用例可能会误导或有歧义。一个用例关注功能和行为需求，一般不适用于非功能需求。对于必须特别详细和精准的需求建模情景（例如安全关键系统），用例方法就不够用了。

然而，软件工程师会遇到的所有场景中的绝大多数情景都适用基于场景建模。如果开发得当，用例作为一个建模工具能提供很大的好处。

2．补充用例的 UML 模型

用例图比较简单，但它提供的信息有限，就像医生从外观看一个人，他是否真的有病，还需借助于其他手段才能知道更多的情况。为了补充用例，可以从大量的 UML 图形模型中进行选择。

（1）开发活动图

UML 活动图在特定场景内通过提供迭代流的图形化表示来补充用例。类似于流程图，活动图使用两端为半圆形的矩形表示一个特定的系统功能，箭头表示通过系统的控制流，菱形表示判定分支，实心水平线表示并行发生的活动，通过互联网进入摄像机监视并显示摄像机视图功能的活动图如图 2-22 所示，应注意到活动图增加了额外的细节，而这些细节是用例不能直接描述的（是隐含的）。例如，用户尽可以尝试有限次数地输入用户身份证号（ID）和密码，这可以通过“提示重新输入”的判定菱形来体现。

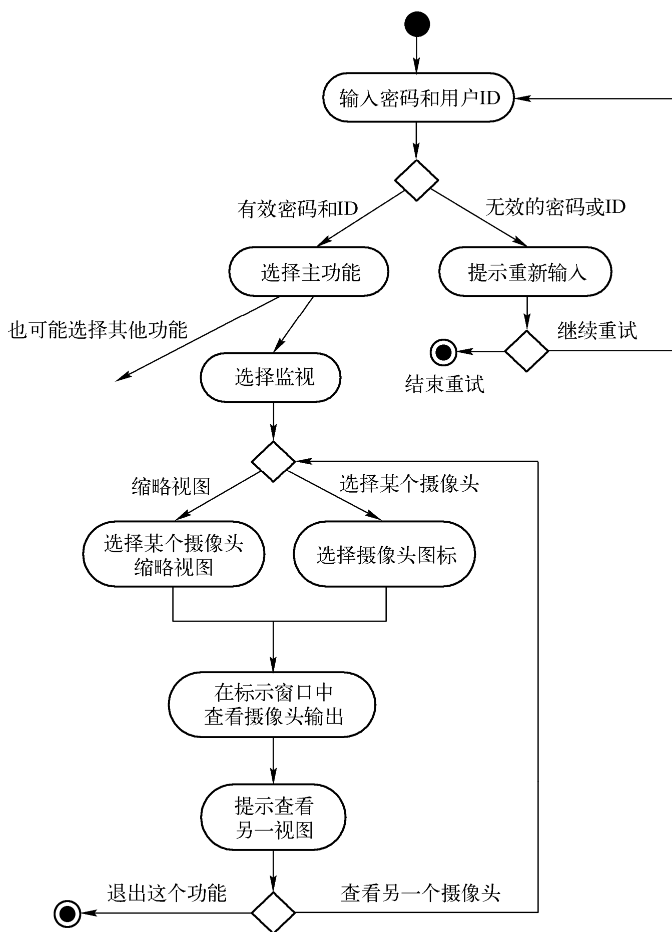


图 2-22　通过互联网进入摄像机监视并显示摄像机视图功能的活动图

（2）泳道图

UML 泳道图表现了活动流和一些判定，并指明由哪个参与者实施。UML 泳道图是活动图的一种有用的变形，可让建模人员表示用例所描述的活动流，同时指示哪个参与者（如果在某个特定用例中涉及了多个参与者）或分析类是由活动矩形所描述的活动来负责。职责由纵向分割图的并行条表示，就像游泳池中的泳道。

考虑图 2-22 所表示的活动图情景，应该有 3 种分析类（房主、摄像机和接口）有直接或间接的责任。参看图 2-23，重新排列活动图，与某个特殊分析类相关的活动按类落入相应的泳道中。例如，“接口”类表示房主可见的用户接口。活动图标记出对接口负责的 2 个提示——“提示重新输入”和“提示查看另一视图”。这些提示以及与此相关的判定都落入了“接口”泳道。但是，从该泳道发出的箭头返回到“房主”泳道，这是因为房主的活动在房主泳道中发生。

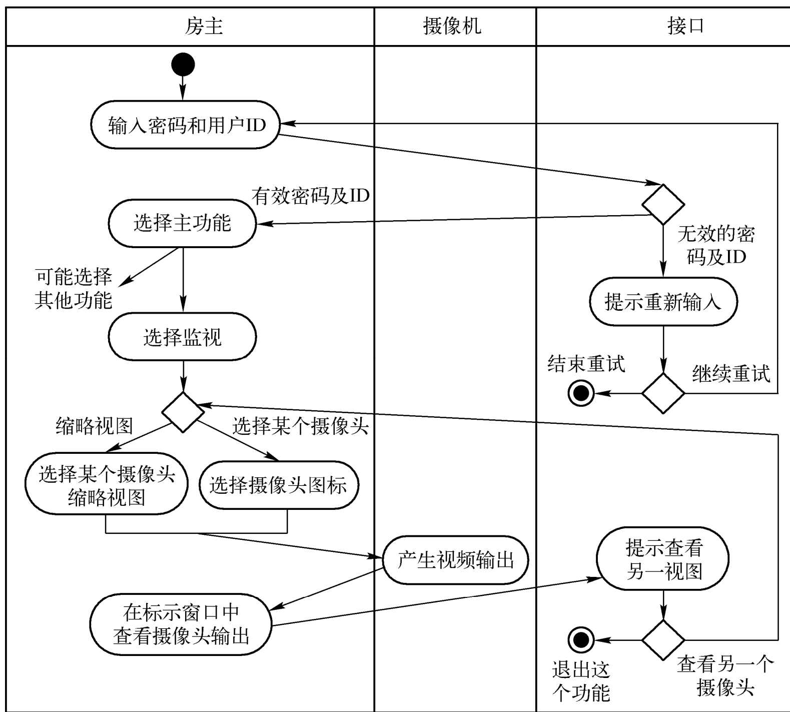


图 2-23　通过互联网进入摄像机监视并显示摄像机视图功能的泳道图

伴随着活动图和泳道图，面向过程的用例表示各种参与者行使一些特定功能或其他处理步骤，以便满足系统需求。但是需求的过程视图仅表示系统的单一维度。

3．数据建模

如果软件需求包括建立、扩展需求，或者具有数据库的接口，或者必须构建和操作复杂的数据结构，那么，可以选择建立一个数据模型作为全部需求建模的一部分。这时需要定义在系统内处理的所有数据对象，数据对象之间的关联以及其他与此相关的信息。可以使用实体关系图（E-R 图）来描述这些问题并提供在一个应用项目中输入、存储、转换和产生的所有数据对象。

（1）数据对象

数据对象是必须由软件可理解的复合信息表示。复合信息是指具有若干不同的特征或属

性的事物。因此，单个值的“宽度”不是有效的数据对象，但是“维度”可以被定义为一个对象，因为它包括宽度、高度和深度。

数据对象可能是外部实体（例如产生或使用信息的任何东西）、事物（例如报告或显示）、偶发事件（例如电话呼叫）或事件（例如警报）、角色（例如销售人员）、组织单位（例如财务部）、地点（例如仓库）或结构（例如文件）。例如，一个人或一部车可以被认为是数据对象，在某种意义上它们可以用一组属性来定义。数据对象描述包括了数据对象及其所有属性。

数据对象只封装数据——在数据对象中没有操作数据的引用，这种区别将数据对象与面向对象方法中的“类”或“对象”区分开来。因此，数据对象可以表示成一张表。

（2）数据属性

属性命名某数据对象，描述其特征，并在某些情况下引用另一个对象。数据属性定义了数据对象的性质，可以具有 3 种不同的特性之一。它们可以用来：1）为数据对象的实例命名；2）描述这个实例；3）建立对另一个表中的另一个实例的引用。另外，必须把一个或多个属性定义为标识符——也就是说，当要找到数据对象的一个实例时，标识符属性成为一个索引关键字。在某些情况下，标识符的值是唯一的，但不是必需的。

（3）关系

关系指明数据对象相互“链接”的方式，数据对象可以以多种不同的方式与另一个数据对象连接。考虑一下两个数据对象：“人”和“汽车”。这些对象可以使用图 2-24a 所示的简单标记表示。在“人”和“汽车”之间可以建立联系，因为这两个对象之间是相关的。但这个关系是什么呢？为确定答案，必须理解在将要构建的软件环境中“人”和“汽车”的角色。可以用一组“对象-关系对”来定义相互的关系，例如：

- 拥有汽车的人。
- 汽车驾车投保人。

关系“拥有”和“驾车投保”定义了“人”和“汽车”之间的相互连接。图 2-24b 以图形方式说明了这些对象-关系对，图 2-24b 标注的箭头提供了关联方向的重要信息，这一方向信息通常可以减少歧义或误解。

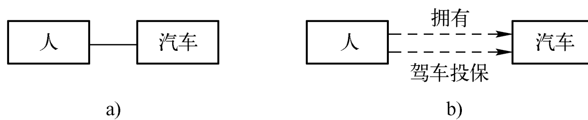


图 2-24　数据对象之间的关联关系

a）数据对象之间的基本连接　b）数据对象之间的关系

4．类建模

类建模是基本的建模内容，基于类的建模表示了系统操作的对象、应用于对象间能有效控制的操作（也称为方法或服务）、这些对象间的关系以及已定义类之间的协作。

（1）识别分析类

当环顾房间时就可以发现一组容易识别、分类和定义的物理对象。但当环顾软件应用的问题空间时，了解类和对象就没有那么容易了。

通过检查需求模型开发的使用场景，对系统开发的用例进行“语法解析”，可以开始进行类的识别。方法是，把每个名词或名词词组用下划线标出来，将它们确定为类，并将这些

名词输入到一个简单的表中，同时标注出同义词，相同的名词或名词词组只标一次。如果要求某个类（名词）实现一个解决方案，那么这个类就是解决方案空间的一部分；否则，如果只要求某个类描述一个解决方案，那么这个类就是问题空间的一部分。

分离出所有的名词后，要确定它是什么类，分析类表现为如下方式之一。

1）外部实体（例如其他系统、设备、人员），产生或使用基于计算机系统的信息。

2）事物（例如报告、显示、字母、信号），问题信息域的一部分。

3）偶发事件或事件（例如，所有权转移或完成机器人的一组移动动作），在系统操作环境内发生。

4）角色（例如经理、工程师、销售人员），由和系统交互的人员扮演。

5）组织单元（例如，部门、组、团队），和某个应用系统相关。

6）场地（例如传感器、四轮交通工具、计算机），定义了对象的类或与对象相关的类。

这种分类只是已提出的大量分类方法之一，为了说明在建模的早期如何定义分析类，考虑对 SafeHome 安全功能的“处理叙述”进行语法分析，对第一次出现的名词加下划线，第一次出现的动词采用斜体。

<u>SafeHome 安全功能</u>***能使***<u>房主</u>在***安装***时***配置***<u>安全系统</u>，***监视***所有***链接***到安全系统的<u>传感器</u>，通过<u>因特网</u>、<u>个人计算机</u>或<u>控制面板</u>和房主***交互***。

在<u>安装过程</u>中，用 SafeHome 个人计算机***设计***和***配置***<u>系统</u>。为每个传感器分配一个<u>编号</u>和<u>类型</u>，用<u>主密码</u>***启动报警***和***关闭报警***系统，而且当<u>传感器事件</u>发生时会***拨打输入***的<u>电话号码</u>。

当***识别***出一个传感器事件时，软件***激活***装在系统上<u>可发声的警报</u>，在房主系统配置活动中***指定***的<u>延迟时间</u>后，软件拨打<u>监测服务</u>的电话号码并***提供***<u>位置信息</u>，***报告***检测到的事件性质。电话号码将每隔 20 秒***重拨***一次，直至***达到***<u>电话接通</u>。

房主通过控制面板、个人计算机或浏览器这些<u>接口</u>来***接收***安全信息。接口在控制面板、计算机或浏览器窗口中***显示***<u>提示信息</u>和<u>系统状态信息</u>。房主采用如下形式进行交互活动……

抽取上述描述中的名词，可以得到如表 2-5 所示的一些潜在类。这个表应不断完善，直到将处理叙述中所有的名词都考虑到为止。注意，表 2-5 中的每一输入项只是潜在的对象，还不是最终确定的类，在进行最终决定之前还必须对它们每一项进行分析。

表 2-5　对 SafeHome 安全功能进行语法解析后得出的潜在类

潜在类	一般分类
房主	角色或外部实体
传感器	外部实体
控制面板	外部实体
安装过程	事件
系统（别名安全系统）	事物
编号、类型	不是对象，是传感器的属性
主密码	事物
电话号码	事物
传感器事件	事件
可发声的警报	外部实体
监测服务	组织单元或外部实体

如何确定某个潜在类是否应该真的成为一个分析类，可考虑是否具有如下特征。

1）是否是保留信息。在分析期间只有当潜在类的信息必须记下来，系统才能工作，这样的潜在类就是有用的。

2）是否是所需服务。潜在类必须具有一组可确认的操作，这组操作能用某种方式改变类的属性值。

3）是否具有多个属性。在需求分析过程中，焦点应在于主要的信息。当然，只有一个属性的类可能在设计中有用，但是在分析活动阶段，最好把它作为另一个类的某个属性。

4）是否具有公共属性。可以为潜在类定义一组属性，这些属性适用于类的所有实例。

5）是否是公共操作。可以为潜在类定义一组操作，这些操作适用于类的所有实例。

6）是否是必要的需求。在问题空间中出现的外部实体，以及生产或消费对任何系统解决方案在运行时所必须的信息，几乎都被定义为需求模型中的类。

考虑包含在需求模型中的合法类，潜在类应全部或几乎全部满足这些特征。判定潜在类是否包含在分析模型中多少有点主观，而且后面的评估可能会舍弃或恢复某个类。然而，基于类建模的首要步骤就是定义类，因此即使是主观的也必须进行决策。根据上述选择特征进行筛选后，可列出 SafeHome 潜在类如表 2-6 所示。

表 2-6　对表 2-5 的潜在类进行筛选的结果

潜　在　类	适用的特征编号
房主	拒绝：6 适用但是 1、2 不符合
传感器	接受：所有都适用
控制面板	接受：所有都适用
安装过程	拒绝
系统（别名安全系统）	接受：所有都适用
编号、类型	拒绝：3 不符合，这是传感器的属性
主密码	拒绝：3 不符合
电话号码	拒绝：3 不符合
传感器事件	接受：所有都适用
可发声的警报	接受：2、3、4、5、6 适用
监测服务	拒绝：6 适用但是 1、2 不符合

应该注意到：1）表 2-6 并不全面，必须添加其他类以使模型更完整；2）某些被拒绝的潜在类将成为被接受类的属性（例如，“编号”和“类型”是“传感器”的属性，“主密码”和“电话号码”可能成为“系统”的属性）；3）对问题的不同陈述可能导致做出“接受或拒绝”不同的决定。（例如，如果每个房主都有个人密码或通过声音确认，那么“房主”类有可能接受并满足特征 1）和 2）。

（2）描述属性

属性是在问题的环境下完整定义类的数据对象集合，属性描述了已经选择包含在需求模型中的类。为了给分析类开发一个有意义的属性集合，应该研究用例并选择那些合理属于类的事物。此外，每个类都应回答如下问题：什么数据项能够在当前问题环境内

完整地定义这个类？

为了说明这个问题，考虑为 SafeHome 定义 System 类。房主可以配置安全功能以反映传感器信息、报警响应信息、激活或者关闭信息、识别信息等。可以用如下方式表现这些组合数据项。

识别信息=系统编号+确认电话号码+系统状态

报警应答信息=延迟时间+电话号码

激活或者关闭信息=主密码+允许重试次数+临时密码

等式右边的每一个数据项可以进一步地精化到基础级，本例中可以为 System 类组成一个合理的属性列表，如图 2-25 的中部所示。

图 2-25　System 类的类图

传感器是整个 SafeHome 系统的一部分，已经定义 Sensor 为类，多个 Sensor 对象将和 System 类关联。通常，如果有超过一个项和某个类相关联，就应避免把这个项定义为属性。

（3）定义操作

操作定义了某个对象的行为。尽管存在很多不同类型的操作，但通常可以粗略地划分成 4 种类型：1）对数据的操作，例如添加数据、删除数据、重新格式化数据、选择数据；2）执行计算的操作；3）请求某个对象的状态的操作；4）监视某个对象发生某个控制事件的操作。这些功能通过在属性及相关属性上的操作来实现。因此，操作必须理解为类的属性和相关属性的性质。

在第一次迭代要导出一组分析类的操作时，可以再次研究处理叙述（或用例）并合理地选择属于该类的操作。为了实现这个目标，可以再次研究语法解析并分离动词。这些动词中的一部分将是合法的操作并能够很容易地连接到某个特定类。

另外，对于语法分析，可以通过考虑对象间所发生的通信来获得对其他的操作更为深入的了解，对象通过传递信息与另一个对象互相通信。

（4）类-职责-协作者建模

类-职责-协作者（Class-Responsibility-Collaborator，CRC）建模是一种简单的建模方法，可以识别和组织与系统或产品需求相关的类。CRC 建模实际上是表示类的标准索引卡片的集合。每张卡片分 3 部分，顶部写类名，卡片主体左侧部分列出类的职责，右侧部分列出类的协作者。使用 CRC 卡片的一个目的是早期舍弃、频繁舍弃，并且低成本舍弃。事实上抽出一叠卡片要比改编大量源代码要容易得多。

事实上，CRC 模型可以使用纸质的或电子的索引卡，建立索引卡的目的是开发有组织表示的类。CRC 模型中的职责是和类相关的属性和操作。简单地说，职责就是“类所知道或能做的任何事”，协作者是提供完成某个职责所需要信息的类。通常，协作意味着信息请求或某个动作请求。

例如，SafeHome 安全住宅系统中 FloorPlan 类的一个简单 CRC 索引卡可以用图 2-26 所示的形式来描述。CRC 卡上所列出的职责只是初步的，可以添加或修改。在职责栏右边的 Wall 和 Camera 是需要协作的类。

Class:FloorPlan	
说明	
职责：	协作者：
定义住宅平面图的名字和类型	
管理住宅平面图的布局	
缩放显示住宅平面图	
合并墙，门和窗户	Wall（墙）
显示摄像头的位置	Camera（摄像头）

图 2-26　CRC 模型索引卡例子

CRC 建模将类分为 3 种。

1）实体类。也称做模型或业务类，是从问题说明中直接提取出来的，例如 FloorPlan 和 Sensor 就是属于实体类，这些类一般代表保存在数据库中和贯穿应用程序的事物。

2）边界类。用于创建用户可见的和在使用软件时交互的接口，如交互屏幕或打印的报表。实体类包含对用户来说很重要的信息，但是并不显示这些信息。边界类的职责是管理实体对象对用户的表示方式，例如，一个称作 Camera Window 的边界类负责显示 SafeHome 系统监视摄像机的输出。

3）控制类。用于自始至终管理“工作单元”。也就是说，控制类可以管理 4 个方面的工作：实体类的创建及更新；当边界类从实体对象获取信息后的实例化；对象集合间的复杂通信；对象间或用户和应用系统间交换信息的确认。通常，直到设计开始时才开始考虑控制类。

在给类分配职责时可以采用如下指导原则。

1）系统的智能应分布在所有类中以求最佳地满足问题的需求。每个应用系统都包含一定程度的智能，也就是系统内所含有它知道的以及所能完成的事情。智能在类中可以有多种分布方式，建模时可以把“不灵巧类”（几乎没有职责的类）作为一些“灵巧”类（有很多职责的类）的从属。尽管该方法使得系统中的控制流简单易懂，但同时带来两个方面的缺点：一是把所有的智能集中在少数类，使得变更更为困难；二是这样做可能会需要更多的类，因此需要更多的开发工作。

如果将系统的智能更平均地分布在应用系统的所有类中，每个对象只了解和执行其中的一些事情，并提高系统的内聚性，这就会提高软件的可维护性并减少变更的副作用影响。

为了确定是否恰当地分布了系统智能，应该评估每个 CRC 模型索引卡上标记的职责，以确定某个类是否具有超长的职责列表，如果有这种情况就表明智能太集中。此外，每个类的职责应表现在同一抽象层上。

2）每个职责的说明应尽可能具有普遍性。这条指导原则意味着应在类的层级结构的上层保持职责（属性和操作）的通用性，因为它们更有一般性，它们将适用于所有的子类。

3）信息和与之相关的行为应放在同一个类中。这实现了面向对象原则中的封装，数据和操作数据的处理应包装在一个内聚单元中。

4）某个事物的信息应局限于一个类中而不要分布在多个类中。应由一个单独的类负责

保存和操作某特定类型的信息，通常这个职责不应由多个类分担。如果信息是分布的，软件将变得更加难以维护，测试也会面临更多挑战。

5）职责应由相关类共享。很多情况下，各种相关对象必须在同一时间展示同样的行为。例如，对于一个视频游戏，必须显示如下类：Player、PlayerBody、PlayerArms、PlayerLegs 和 PlayerHead。每个类都有各自的属性（例如，position、orientation、color 和 speed）并且所有这些属性都必须在用户操纵游戏杆时更新和显示。因此，每个对象必须共享职责 update()和 display()。Player 知道在什么时候发生了某些变化并且需要 update()操作。它和其他对象协作获得新的位置或方向，但是每个对象控制各自的显示。

协作方面的内容讲解如下。

类通过一种或两种方法实现其职责：一是类可以使用其自身的操作控制各自的属性，从而实现特定的职责；二是一个类可以和其他类协作。

协作是从客户职责实现的角度表现从客户到服务器的请求。协作是客户和服务器之间契约的具体实现。如果为了实现某个职责需要发送任何消息给另一个对象，就说明这个对象和其他对象有协作。单独的协作是单向流，即表示从客户到服务器的请求。从客户的角度看，每个协作都和服务器的某个特定职责相关。

要识别协作可以通过确认类本身是否能够实现自身的每个职责。如果不能实现每个职责，那么需要和其他类交互，因此就要有协作。

当开发出一个完整的 CRC 模型时，可以使用如下方法评审模型。

1）所有参加 CRC 模型评审的人员拿到一部分 CRC 模型索引卡。对有协作的卡片要拆分，也就是说每个评审员不得有两张存在协作关系的卡片。

2）分类管理所有的用例场景以及相关的用例图。

3）评审组长细致地阅读用例，当评审组长看到一个已命名的对象时，给拥有相应类索引卡的人员一个令牌。例如，SafeHome 的一个用例包含如下描述：

房主观察 SafeHome 控制面板以确定系统是否已经准备接收输入。如果系统没有准备好，房主必须手工关闭窗户和门以便指示器呈现就绪状态（若指示器未就绪说明某个传感器是开启的，也就是说某个门或窗户是打开的）。

当评审组长看到用例说明中的“控制面板”，就把令牌传给拥有 ControlPanel 索引卡的人员。“暗示着某个传感器是开启的”语句需要索引卡包含确认该暗示的职责（由 determine-sensor-status()实现该职责）。靠近索引卡职责的是协作者 Sensor，然后令牌传给 Sensor 对象。

4）当令牌传递时，Sensor 卡的拥有者需要描述卡上记录的职责。评审组确定（一个或多个）职责是否满足用例需求。

5）如果记录在索引卡上的职责和协作不能满足用例，就需要修改卡片。修改可能包括定义新类和相关的 CRC 索引卡，或者在已有的卡上说明新的或修改的职责、协作。

该过程持续进行直到用例编写结束。当评审完所有的用例，将继续进行需求建模。

2.3.3 统一建模语言 UML

Rational 软件公司的三位学者，Grady Booch、Jim Rumbaugh 和 lvar Jacobson 经过三年多的努力正式提出面向对象系统的通用统一模型语言 UML 1.0 版，这是 OO 行业中一件具有里程碑性质的进展。UML 语言是在已有的三大 OO 方法学的基础上，抽象出表示它们的模型语

言，并吸取了其他 OO 开发方法和近三十年软件工程的成果，它对 OO 技术的发展有着深远的影响。1997 年 11 月，OMG 组织采纳 UML 作为面向对象建模的标准语言。UML 还在不断发展变化，在对象管理组织网站 www.omg.org 上可查看到 UML 的版本内容，熟悉 UML 对于软件工程人员是至关重要的。

1．UML 的元模型理论

模型规定了对象的属性操作以及聚集、结合和通信。利用表示法系统表达的层次称为模型层。

一个系统往往是由多个模型的聚集、相互结合和通信组成。需要一种手段构成各个模型，为此把属性、操作、结合、通信进一步抽象为结构元素、行为元素来表达模型，并提供表达系统的机制（包），这一层称为元（Meta）模型层。为了准确地表达模型的语义，提供分类、说明（标记值）和约束。在这样的元模型描述下生成的模型实例，语义可以得到准确地刻画。

元模型的基础结构是元-元模型层，它定义了用于描述元模型的语言。为了准确定义元模型的元素和各种机制，元-元模型指出，每个元类中有元属性和元操作，最基本的机制是类-实例化机制，即元模型是元-元模型的实例。为了和 OMG 组的元对象设施（Meta Object Facility，MOF）提供的元-元模型一致，UML 的元模型体系结构直接从 MOF 的元模型生成。UML 是元模型层的描述语言，它的实例兼及模型层，并可以直接对应 OO 语言中的类、类型、消息、继承、聚集、概括、接口。

2．大型逻辑包装

一个软件系统，由不同模型的系统组成。每个模型由模型元素按照某种组合机制构成。UML 从表示角度上用无语义但有结构关系的包把相关元素封装在一起。包有包容（子包）和继承关系。系统和模型均按包的形式提供，即一个包可以封装一个模型，若干子包聚集为一个系统包。用户可以在系统提供的包的基础上定义自己的系统和各模型。UML 提供的包是基础包、行为元素包和模型管理包。基础包描述一个软件系统提供的最基本支持；行为元素包为模型元素定义各种动作、通讯，管理其使用情况、状态描述等；模型管理包定义了模型元素如何组织成模型、包和子系统。

3．图形化的 UML 表示法

UML 为对象的结构模型和行为模型定义了语义。UML 中的图可分为两大类：结构图和行为图。结构图描绘系统组成元素之间的静态结构，包括它们的类、接口、属性和关系。行为图描绘系统元素的动态行为，包括它们的方法、交互作用、协作性和状态历史。UML 表示法是 UML 语义的可视化表示，是描述模型的工具。

（1）结构图

结构图的类型包括类图、构件图、对象图、部署图、组合结构图、包图和用例图 7 种。类图是使用 UML 建模时最常用的图，它展示了系统中的静态事物、它们的结构以及它们之间的相互关系，如图 2-25 就是一个简单例子，System 类有 8 个属性和 6 个操作；构件图可以展示一组构件的组织和彼此间的依赖关系，它用于说明系统如何实现，以及软件系统内构件如何协同工作；对象图可以展示系统中的一组对象，它是类图在某一时刻的快照；部署图可以展示物理系统运行时的架构，同时可以描述系统中的硬件和硬件上驻留的软件；组合结构图可以展示模型元素的内部结构；包图用于包之间的依赖关系；用例图描述系统外部参与

者如何使用系统提供的服务。

（2）行为图

行为图包括活动图、状态图、顺序图、通信图、时间图和交互概述图 6 种。活动图显示系统内的活动流，通常用于描述不同的业务过程；状态图显示一个对象的状态和状态之间的转换，状态图中包括状态、转换、事件和活动。状态图是一个动态视图，对事件驱动的行为建模尤其重要。

在描述系统中，反映对象之间通过消息进行通信的图称为交互图。交互图包含 4 种类型：顺序图、通信图、时间图和交互概述图。顺序图也称为时序图，它描述了系统中对象间通过消息进行的交互，强调了消息在时间轴上的先后顺序；通信图也称为协作图、合作图，它描述了系统中对象间通过消息进行的交互，强调了对象在交互行为中承担的角色；时间图也称为定时图，是一种特殊的顺序图，它描绘与交互元素的状态转换或条件变化有关的详细时间信息；交互概述图是一种高层视图，用于从总体上显示交互序列之间的控制流。

（3）关系元素

在 UML 中，共定义了 24 种关系，这 24 种关系在建模时可以用关联关系、实现关系、泛化关系、依赖关系和扩展关系 5 种来表示。关联关系提供了通信的路径，它是所有关系中最通用同时也是语义最弱的关系。在关联关系中，有两种比较特殊的关系，它们是聚合关系和组合关系；实现关系是用来规定接口和实现接口的类或组件之间的关系。接口是操作的核心，这些操作用于规定类或组件提供的服务；泛化关系描绘了从特殊事物到一般事物之间的关系，即子类到父类之间的关系；依赖关系表示两个或多个模型元素之间语义上的关系，客户元素以某种形式依赖于提供者元素，依赖关系可以细分为使用依赖、抽象依赖、授权依赖和绑定依赖四大类，实际上，关联、实现和泛化都是依赖关系；扩展关系是一种 UML 提供的底层的扩展机制，应用并不广泛。扩展表示把一个构造型附加到一个元类上，使得元类的定义中包含这个构造型。

4．元模型的形式化语言 OCL

用户借助 UML 提供的表示法定义自己系统的元模型，以图形化的表示方法来表示模型元素时，其语义解释不够准确。为此 UML 提供了形式化语言即对象约束语言（Object Constraint Language，OCL）以一阶谓词逻辑模型描述各种约束。

实际上 UML 继承了软件工程中形式化规格说明语言研究的成果。因为只有形式规格说明描述的软件体系结构在其各开发阶段中才能保证语义的一致性。但是形式语言约束只能准确刻画静态语义，对于动态语义除了加上 OCL 描述外，还必须以自然语言加以说明，完全形式化是极为复杂的。UML 在给出自身的语义说明时运用了这个办法，对于每个包都给出以下 3 个层次的说明。

1）抽象的语法（Abstract Syntax）：由一个 UML 类图给出各元类之间的关系。

2）良构的规则（Well-formedness Rules）：用形式语言 OCL 表述无边界效应的约束。

3）语义（Semantics）：用自然语言描述引入的新概念和动态语义。

总的来说，UML 元模型是由图形表示法、自然语言和形式语言组成的描述。这种合成强调表述性和易读性间的平衡。

5．基于 UML 的软件开发

软件开发通常按照如下步骤进行：首先是了解系统（客户）需求，然后是系统分析、设

计，之后是代码编写，最后是测试，投入运行后还要进行代码的维护和扩展。UML 允许从一个基本的图开始，之后添加需要的额外特性。

（1）系统（用户）需求

它是问题提出的过程，软件大体要实现什么功能，有什么参与者（Actor）。而用例（Use Case）图所表达的正是参与者与用例、用例和系统响应之间的关系。在了解系统需求的过程中，就可以给用例图添加上相应的 Actor 与 Use Case。在了解系统需求的后期，则可以把不同 Actor 与 Use Case 间的关系用相应的连线连接，并对连线定义，使用例图更加清晰。之后可对 Use Case 的系统响应进行分析，即把系统执行顺序画出来，这已经进入系统分析阶段。

（2）系统分析

在这一阶段，需要把软件的功能细化，同时考虑数据间的关系（层次），然后建立与之对应的对象，接着分析对象间的关系（即软件结构）并定义对象的属性和方法。

其过程是：首先对 Use Case 进行分析，可按时间顺序把一个个过程标注出来，并分析哪些过程属于哪一个对象，形成顺序图。同时 Use Case 中还可以使用协作图和状态图。协作图是按对象之间的关系来连接，适用于时间顺序不严格的过程变化。状态图的分析重点在于数据的变化，常用于观察过程的参数和数据的变化，它在类图分析中也使用。其次是把上一步分析的过程和数据整理成对象的方法和属性，并在类图中画出来。对于类中的操作，还可用状态图来分析运行过程。完成类的绘制后，要把相近的类划归在一起，形成一个包，接着把这些类用相应的关系（继承、接口等）连接起来。类与类之间的关系在不同的面向对象的语言中有着各自的特殊之处。但都可用 UML 的关系（关联关系、实现关系、泛化关系、依赖关系和扩展关系）表达出来。

（3）设计

这一阶段，在基于操作系统和特定语言的基础上分析整个系统，如果设计的系统模块处于不同物理层次，比如网络系统中有多个 Server（Database Server、Web Server 等）和多个 Client，而软件模块分布在不同的 Server 里运行，这时需要使用部署图，用来显示系统中软件和硬件的物理架构，部署图既可以显示运行时系统的结构，同时还可表明构成应用程序的硬件和软件元素的配置和部署方式。

需要说明的是基于 UML 的软件分析与设计是一个连续的过程，设计阶段是分析阶段的结果的扩展，通过加入新的类来定义软件系统的技术方案细节，设计阶段用和分析阶段类似的方式使用 UML。

（4）代码编写

这一过程是把抽象的软件模型转变为具体的代码。使用基于 UML 的 CASE 工具建立了相应的模型，或者说建立起一个多视角的相关数据库系统之后，则可使用软件直接产生基于此模型的对应代码框架。至此一个具有清晰结构的软件系统就已经有了雏形，往后的阶段只需要把某一种具体代码往框架里填充。

（5）测试

通常，测试可分为单元测试、集成测试、系统测试和验收测试几个步骤，UML 模型可作为测试阶段的依据，不同测试小组可使用不同的 UML 图作为他们测试工作的依据。单元测试使用类图和类规格说明，集成测试使用构件图和协作图，系统测试使用用例图来验证系

统的行为，验收测试用与系统测试类似的方法来验证系统是否满足在分析阶段确定的所有需求，验收测试是由用户进行的。

（6）代码的维护和扩展

面向对象的程序设计方法，主要目的就是为了让程序更容易维护、扩展。为了改动和添加代码，一般情况下就得通读源代码和注释，然后才明白程序的结构和完成的功能，进而修改。但对于 UML 设计出来的软件，只要看其模型中不同的视图就可以把软件结构清晰的表达出来了。同时在建立模型时所写的文档，也会加入源代码中。另外，有的 CASE 工具还提供 COM 接口，通过编程对建立的模型进行分析，输出相应的报表、文档。因而，用 UML 建立模型更利于软件的维护和扩展。

2.4 需求规约说明书（SRS）

软件需求规约说明书（Software Requirements Specification，SRS）也叫软件需求规约，是分析阶段的产物，写出 SRS 并经复审通过，就意味着分析阶段结束。因而 SRS 是开发人员与用户必须遵守的“合同”。它既是软件人员进行软件设计的依据，也是用户考虑验收方案的基础。SRS 是软件开发过程中一个重要的里程碑，它是所有相关人员对所开发软件的共同理解、共同意志的体现。

2.4.1 SRS 的内容

SRS 中的内容以及详细程度取决于所要开发的系统的类型以及所使用的开发过程，复杂的、要求极高的系统需要有详尽的需求，因为系统安全性和信息安全性都需要详细的分析。如果系统是由某个外部机构承担的时候，对 SRS 的描述应该精确和详细。如果需求中有较大弹性，而且系统是由本机构内部开发的话，文档可以写得不太详细，一些二义性问题可以在开发阶段得以解决后补充进去。

一般可以从以下 5 个方面去描述 SRS。

1．概述

给出软件需求的简单描述，包括课题目标、用户、约束、功能性能规定等。

2．软件需求描述

1）功能和行为建模。给出用例图、功能和特征列表，给出候选类清单，建立类的层次关系，绘制基于 UML 的状态图、需求的活动图、顺序图等，给出类定义模板（类的整体说明，属性说明，方法和消息说明）。

2）数据建模。确定数据对象和数据属性，给出详细的数据流图及数据词典描述，使用 E-R 图描绘数据对象之间的关系。

3．界面

规定软件同系统其他元素（硬件、软件、人机接口、数据通信协议等）的功能联系，给出初步用户手册。硬件界面包括计算机特性、内外存容量、I/O 设备能力等。软件界面包括操作系统特性、公用程序和支持软件以及它们相互之间的连接特性。

4．质量评审

规定软件功能的正式确认需求和测试限制，给出软件的初步测试计划。

5．补充说明

给出一些便于读者阅读本规格说明书的注释，例如本项目的一些背景材料，以增进对本规格说明书内容的理解。

表 2-7 是一个基于 IEEE 标准的 SRS 的结构，IEEE 标准是一个通用的标准，可以调整以适应特殊场合。

表 2-7　软件需求规约说明书的结构

章节题目	描述要求及内容
前言	定义文档的读者对象，说明版本的修正历史，包括新版本为什么要创建，每个版本间的变更内容的概要
绪论	描述为什么需要该系统，简要描述系统的功能，解释系统是如何与其他系统协同工作的。要描述该系统在机构总体业务目标和战略目标中的位置和作用
术语	定义文档中的技术术语和词汇。假设文档读者是不具有专业知识和经验的人
用户需求定义	描述系统应该提供的服务以及非功能系统需求，该描述可以使用自然语言、图表或者其他各种客户能理解的标记系统。产品和过程必须遵循的标准也要在此定义
系统体系结构	对待建系统给出体系结构框架，该体系结构要给出功能在各个模块中的分布。能被复用的结构中组件要用醒目方式标示出来
系统需求描述	对功能和非功能需求进行详细描述。如有必要，对非功能需求要再进一步描述，例如，定义与其他系统间的接口
系统原型	给出一个或多个系统模型，以表达系统组件、系统以及系统环境之间的关系。这些模型可以是对象模型、数据流模型和语义数据模型
系统演化	描述该系统建成后的基本设想，并预测由于硬件演化和改变用户需求时系统将如何变动。这部分对系统设计人员来说是有用的，因为这有助于他们避免一些不合理的设计决策，这些决策可能会限制未来系统的变更
附录	提供与所开发的应用软件有关的详细的、专门的信息，例如，硬件和数据库的描述，硬件需求定义了系统最小和最优配置，数据库需求定义了系统所用的数据的逻辑结构和数据之间的关系
索引	包含文档的多重索引，除了标准的字母顺序索引外，还应有图表索引、功能索引等

当然，SRS 中的内容是和被开发软件的类型以及开发中使用的方法紧密相关的，如果使用演化模型来开发软件产品，则表 2-7 中的许多有关细节可能会省去。重点也将会被放在用户需求的定义和高标准的非功能需求上面。在这种情况下，因为 SRS 提供的信息有限，所以设计者和编程人员将根据他们的判断和知识来决定如何设计系统以满足用户的需求。

然而，当软件系统是大型系统工程项目的一部分时，大系统本身包含交互式硬件和软件系统，一般就必须在细粒度层次上定义需求，这意味着 SRS 内容会非常多，可能还会包含初步用户手册、初步测试计划等，对于长文档，尤其需要一个详细的目录和文档索引，以便读者能快速找到所需的信息。

2.4.2　SRS 的作用

在外部承包商开发软件系统时必须有 SRS，然而敏捷开发模式的使用表明由于需求的快速变化，致使 SRS 在写完时已经过时，也就浪费了大量的精力。于是像极限编程这类的方法相应产生，这种方法是增量式收集用户需求，并把它们作为用户故事情景写在卡片上，然后用户对要实现的需求给出优先级排序，最为紧要的需求将在下一个增量中优先考虑。

这种方法很适合需求不稳定的业务系统，但是有一份定义系统的业务和可靠性需求的短的支持文档仍然是有用的，当专注于系统下一个版本的功能性需求时，很容易忘记应用到整个系统上的需求。

SRS 的作用可以用图 2-27 来表示，SRS 对不同的人员起的作用是不一样的，在编写 SRS

时必须在以下几方面采取折中：与客户关于需求的沟通；为开发者和测试者在细节层次上定义需求；附带可能对系统所做的演化的有关信息。对可预见变更方面的信息能帮助系统设计者避免作出一些苛刻的设计决策，也能帮助系统维护工程师避免为增加新需求而去调整系统。

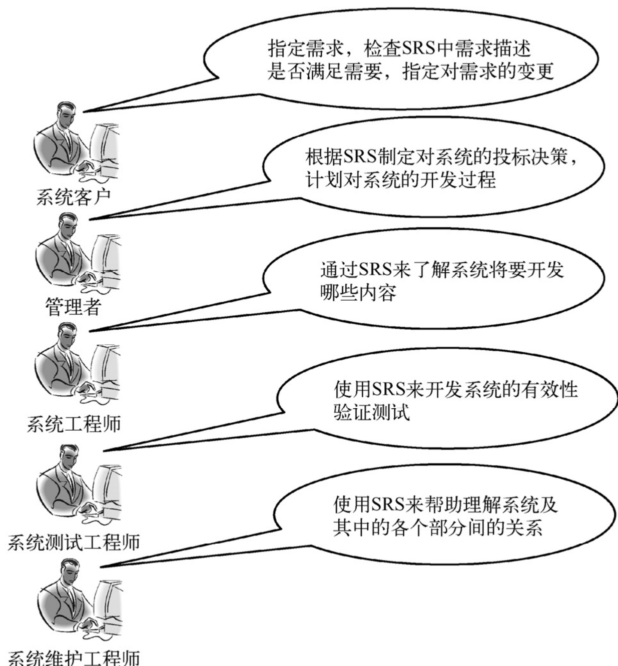


图 2-27　SRS 的用户及作用

2.4.3　SRS 的特征

由以上讨论可以看出，所谓 SRS 其实就是对所开发的软件系统的功能、性能、用户界面以及运行环境等作出的详细说明。那么，怎样才能把 SRS 写好呢？一般来说，一份好的 SRS 应具有如下 7 个特征。

1．唯一性

唯一性是指在 SRS 中每一种需求仅有一种解释。

1）最终软件产品的每一个特征仅用一个条款来叙述。

2）在特殊场合某个术语可以是多义的，但必须在词典中对它的含义做出规定。

由于自然语言有歧义性，所以当用自然语言写 SRS 时要特别细心检查，应尽量避免其歧义性。用形式语言（Formal Language）或半形式语言来写 SRS 是避免歧义性的一种方法。有了形式语言，还可以进一步实现需求自动化，其语法、语义方面的错误可以通过形式语言处理器来自动地检出。

当然，形式语言也有一个缺点，就是需要一段时间的学习，不像自然语言那样人人都会用。

2．完整性

完整性是指：

1）包含全部重要的需求。如功能、性能、设计约束、特征和外部接口等。

2）规定各种情况下对每种输入数据的软件响应，注意，规定不正确的输入值的响应同

规定正确的输入值的响应是同样重要的。

3）符合需求规范标准，如果某一部分不符合标准，则要说明标准中不适用于本需求规范书的原因。

4）全部标记和规范书中涉及的所有图表均应齐全，定义所有的术语和测量单位，还要指出各部分的含义。

3．可检验性

可检验性是指 SRS 中描述的每一个需求都应该是可检验的。也就是说，当人或机器通过有限步处理，能检查出软件产品是否满足要求。

有时在编写 SRS 时会遇到用户认为可检验的要求，而编写该 SRS 的人却认为是不可检验的要求。对此，有 3 种处理方法：

1）修改这些要求，使之能被检验。

2）在编写 SRS 时暂时还不能确定检验方法，但是在以后的开发阶段是可以确定检验的方法，这样的要求应暂时保留下来，到能够确定的时候就补充进去。

3）去掉那些确实不可检验的要求。

4．一致性

一致性是指在 SRS 中不出现相互冲突的需求。在 SRS 中出现的冲突有如下 3 种。

1）两个不同的术语描述同一对象。

2）规定客观对象的特征有冲突。

3）两个规定的动作之间有逻辑上或时间上的冲突。

5．可修改性

可修改性是指当改动 SRS 的结构或文体时是十分容易的，且与以前的版本相容。

6．可跟踪性

如果 SRS 中每条需求的由来都很清楚，在开发软件的过程中，或在后续编写其他文档过程中，在 SRS 中都可以找到依据，这就是可跟踪性。

7．可利用性

在软件运行和维护阶段，甚至万一替换该软件时 SRS 必须是有用的，这就是运行和维护阶段的可利用性。

软件的维护可能与原来开发该软件无关的人员频繁地进行。局部的改变可以根据代码的注解来进行，然而，对更加广的范围，如设计乃至需求的改变，其 SRS 是必须的。此时，除了要求 SRS 具有可修改性以外，SRS 也应提供来龙去脉。假如原因或功能的来由不清楚，欲对其进行有效的维护往往是办不到的。

2.4.4 SRS 的构造原则

如果希望 SRS 完美无缺可能也不现实，但是应该抓住用户对系统需求的本质。不管完成 SRS 的方式有什么不同，规约可被看作是一个表示过程。需求被以最终导向成功的软件实现的方式来表示。Balzer 和 Goldman 给出了如下的一系列规约原则。

1）把功能和实现分开。

2）开发一个系统的行为模型，该模型应描述出系统的数据和功能对来自外部的“敏感数据”是如何响应的。

3）通过刻画其他系统构件和软件交互的方式，建立软件操作的语境。

4）定义系统运行的环境，指明一组高度缠绕在一起的动作者如何对环境中由其他动作者产生的改变对象的“敏感数据”作出反应。

5）创建认知模型而不是设计或实现模型，该认知模型按用户感觉系统的方式来描述系统。

6）认识到规约是不完整的和可扩充的。规约总是一个模型，通常是现实中（或想象中）某个相当复杂情形的一个抽象，因此，它将是不完整的，并将存在于多个细节层次。

7）在建立 SRS 的内容和结构时，应使它能够适应未来的变化。

2.4.5 SRS 的评审

评审也称为复审，是保证软件质量的重要手段，软件生命周期每个阶段工作结束后都应进行评审。软件需求规约是软件工程后续各个阶段的工作基础，其质量直接影响到软件的质量，因而 SRS 的评审是十分重要的，它是保证规约质量的重要措施。

1．SRS 的评审内容

对软件需求规约的复审可以从粗到细地进行，首先从客观性上保证规约的完整性、一致性、精确性，复审时可考虑下列问题。

1）叙述的软件目标和系统的目标是否保持一致？

2）所有系统元素的重要接口是否进行了描述？

3）是否合适地定义了问题域的信息流和结构？

4）图是否清楚？每个图可以没有文字补充而单独存在吗?

5）主要的功能是否保留在范围中，并且均已合适地描述？

6）软件的行为和它所必须处理的信息及必须完成的功能是否一致？

7）设计约束是否现实？

8）开发的技术风险是否考虑过？

9）是否考虑过其他可选的软件需求？

10）是否详细地说明了检验标准？它们对成功描述的系统是合适的吗？

11）是否存在不一致性、信息遗漏或冗余？

12）是否和客户进行了全面接触？

13）用户是否复审过初步的用户手册或原型？

14）对计划阶段的估算产生什么样的影响？

之后着重对规约细节的评审，重点关注措辞，发现隐含的问题，明确模糊的术语。

2．SRS 的质量度量

Davis 等人于 1993 年提出了用以下特征来评价分析模型和 SRS 的质量：确定性（无歧义性）、完整性、正确性、可理解性、可验证性、内部与外部的一致性、可完成性、简洁性、可跟踪性、可修改性、精确性和可复用性。他们建议每个特征用一个或多个特征来表示。例如，假设在 SRS 中有 n_r 个需求，则

$$n_r = n_f + n_{nf}$$

其中，n_f 为功能需求数，n_{nf} 为非功能（如性能）需求数。

为确定需求的确定性（无歧义性），Davis 等人提出了一种基于评审者对每个需求的解释

的一致性的度量：

$$Q_1 = n_{ui} / n_r$$

其中，n_{ui} 是所有评审者都有相同解释的需求数，Q_1 的值越接近 1，SRS 的歧义性越低。

功能需求的完整性可以通过下列表达式来确定：

$$Q_2 = n_u / [n_i \times n_s]$$

其中，n_u 是单一功能的需求数，n_i 是由 SRS 定义的或隐含的输入的个数，n_s 是在 SRS 中所确定的状态数，Q_2 是 SRS 中为一个系统所确定的必要功能的百分比度量。但是，它没有考虑非功能性需求，为了把非功能性需求结合到整体度量中以求完整性，必须考虑需求已经被确认的程度：

$$Q_3 = n_c / [n_c + n_{nv}]$$

其中，n_c 是已经确认为正确的需求数，n_{nv} 是尚未确认的需求数。

2.5 案例：图书馆系统的软件需求分析

图书馆系统是常见的应用软件系统，图书馆有大有小，藏书量、业务范围、读者人群、功能、服务及质量有很大的差别，大型图书馆系统是一个非常复杂的系统，涉及到硬件、软件、计算机网络、通信、管理、人员、考古、科研、对外交流、出版等很多学科和技术，小型图书馆则功能比较单一，本节给出小型图书馆系统 LMIS 的软件需求分析过程。

小型图书馆要实现图书查询、图书借出、图书归还和图书管理，系统设图书管理员和普通读者两种用户，普通读者先要进行注册才能使用系统。

图书管理员负责添加、更新和删除图书信息，并登记和查阅图书的借出及归还情况，普通读者可以根据作者名或主题来检索图书信息，并且可以预定暂时借不到的图书，一旦预定的图书被归还或已购买到将立即通知预订者。

该系统应该在 Web 环境下运行，要求界面友好、系统响应速度快，并具有良好的可扩展性。

2.5.1 确定系统参与者

系统参与者是与系统交互的外部实体，它既可以是人员也可以是外部系统或硬件设备，可以通过提出以下问题来确定系统的参与者。

1）谁使用该系统的主要功能？

2）谁需要该系统的支持以完成日常工作任务？

3）谁从该系统获取信息？

4）谁负责维护和管理该系统以保证该系统能正常运行？

5）该系统需要和哪些外部系统交互？

通过以上问题的答案可以确定“图书管理员”和“普通读者”是 LMIS 的两个主动参与者，“图书管理员”负责维护系统的信息并使用系统的主要功能，“普通读者”从系统中获取所需的信息；另外，系统需要使用外部的“邮件系统”通知预订者，因此，“邮件系统”也是一个参与者，如图 2-28 所示。

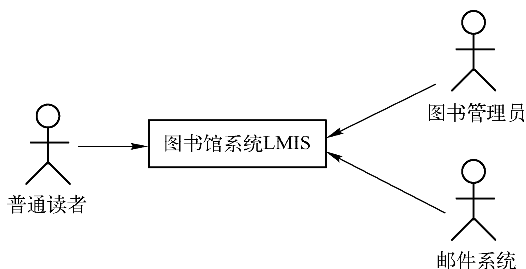


图 2-28　图书馆系统 LMIS 的参与者

2.5.2　开发系统场景

场景是从单个参与者的角度观察系统特性的具体化和非正式的叙述性描述，对于软件开发人员来说，确定系统参与者和场景的关键在于理解业务领域，这需要理解用户的工作过程和系统的范围，为此，可以提出以下问题。

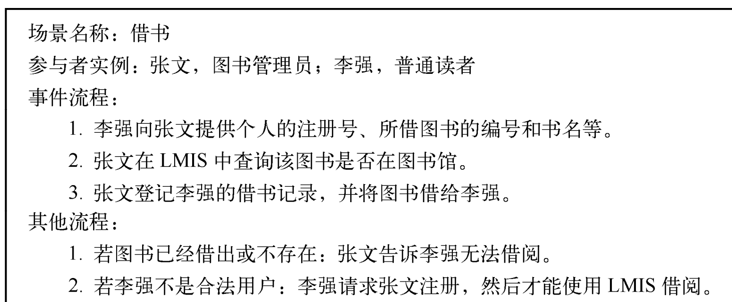
1）系统参与者希望该系统执行什么任务？

2）系统参与者访问什么信息？该系统的数据由谁生成？

3）系统参与者需要通知系统的哪些外部变化？

4）系统需要通知参与者什么事情？

通过对以上问题的回答来确定系统的场景，如图 2-29 描述的就是 LMIS 的一个借书场景。


场景名称：借书

参与者实例：张文，图书管理员；李强，普通读者

事件流程：

1. 李强向张文提供个人的注册号、所借图书的编号和书名等。
2. 张文在 LMIS 中查询该图书是否在图书馆。
3. 张文登记李强的借书记录，并将图书借给李强。

其他流程：

1. 若图书已经借出或不存在：张文告诉李强无法借阅。
2. 若李强不是合法用户：李强请求张文注册，然后才能使用 LMIS 借阅。

图 2-29　LMIS 的借书场景描述

2.5.3　绘制系统用例图

用例用于描述一个完整的系统事件流程，其重点在于参与者与系统之间的交互而不是内在的系统活动，并对系统参与者产生有价值的可观测结果。实际上，从识别参与者开始，发现用例的过程就已经开始了，对于已经识别的系统参与者，可以通过提出如下问题来确定可能的用例。

1）系统参与者要从系统中获得什么功能？参与者需要做什么？

2）系统参与者需要读取、产生、删除、修改或存储系统的某些信息吗？

3）系统中发生事件需要通知参与者吗？参与者需要通知系统某件事情吗？

4）系统需要的输入、输出信息是什么？这些信息从哪里来？到哪里去？

5）系统采用什么方法来满足某些特殊要求？

通过以上问题的回答在 LMIS 中可以识别出以下用例：

（1）与“图书管理员”有关的用例

1）读者管理：在系统中维护普通读者的注册信息。

2）图书管理：在系统中增加、修改和删除图书的基本信息。

3）书目管理：在系统中增加、修改和删除书目信息。

4）借书登记：在系统中登记普通读者的借书记录。

5）还书登记：在系统中登记普通读者的还书记录。

（2）与“普通读者”有关的用例

1）图书预定：在 LMIS 系统中预定图书。

2）预定取消：在 LMIS 系统中取消已有的图书预定。

（3）作为系统的合法注册用户与“图书管理员”和“普通读者”共同有关的用例

1）登录：使用 LMIS 的人员需要进行登录，以验证其身份是否合法，是否具有相应权限。

2）查询浏览：用户可以检索图书信息、读者注册信息及读者借还书信息等。

在确定出每一个系统参与者的用例之后，需要将参与者和确定的每一个用例联系起来，由此就可绘制出系统的用例图，图 2-30 就是通过以上分析后绘制的 LMIS 用例图。

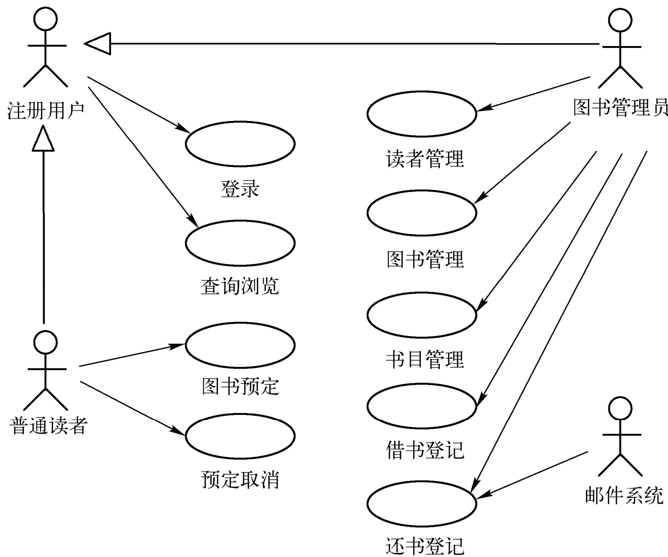


图 2-30 LMIS 用例图

2.5.4 描述用例

从图 2-30 可以看出，用例图提供的信息是有限的，要想更多地了解用例，还需要使用文字来描述那些不能反映在图形上的信息。描述用例就是关于系统的参与者与系统如何交互的规格说明，描述上应该清晰明确，没有二义性。在描述用例时，应该注重外部，尽量不涉及内部细节。

描述用例可从以下 5 个方面进行。

（1）用例目标

简要描述用例的最终任务和结果。

（2）用例中的事件流

1）说明用例是怎样启动的，即哪些系统参与者在什么情况下启动执行该用例。

2）说明系统参与者和用例之间的信息处理过程，如哪些信息是通知对方的，怎样修改和检索信息，系统使用和修改了哪些实体等。

3）说明用例在不同的条件下，可以选择执行的多种方案。

4）说明用例在什么情况下，才能被视作完成，完成时结果应传给系统参与者。

（3）用例的特殊需求

说明此用例有什么特殊要求。

（4）用例的前提条件

说明此用例开始执行的前提，如系统参与者登录成功等。

（5）用例的后置条件

说明此用例执行结束后，结果应传给什么系统参与者。

图 2-31 为 LMIS 中“登记借书”的用例描述。

用例：登记借书

1. 用例目标

本用例允许图书管理员登记普通读者的借书记录。

2. 用例中的事件流

2.1 基本流程

当普通读者希望借书，图书管理员准备登记有关借书记录时，开始执行本用例。

1）系统请求图书管理员输入读者的注册号和所借图书的书目。

2）图书管理员输入有关信息后，系统产生一个唯一的借书记录号。

3）系统显示新生成的借书记录。

4）图书管理员确认后，系统增加一个新的借书记录。

2.2 可选流程

（1）读者没有注册

在主流程中，如果系统中没有读者的注册信息，系统将显示错误信息，用例结束。

（2）所借图书的书目不存在

在主流程中，如果所借图书已被借出或者系统中没有该图书的书目，则系统将显示错误信息，用例结束。

3. 用例的特殊需求

无。

4. 用例的前提条件

此用例开始执行前，图书管理员必须在系统登录成功。

5. 用例的后置条件

如果此用例执行成功，则该读者的借书记录将被更新，否则，系统状态不变。

图 2-31　LMIS 中“登记借书”的用例描述

2.6 小结

需求分析是软件生命周期的重要阶段，目标是深入描述软件的功能和性能，确定软件设计的约束、软件同其他系统元素的接口细节，定义软件的其他有效性需求。软件需求的任务包括

起始、导出、精化、协商、规格说明、确认和管理。软件需求规格说明（SRS）是分析阶段的最终产物，是软件工程过程中的里程碑式的文档，因而是需求分析阶段最重要的文档。

软件需求分析方法主要有结构化方法、原型化方法和面向对象方法。结构化分析是面向数据流进行需求分析的方法，是一种建模技术；原型化开发过程有抛弃式、演化式和增量式，主要原型开发技术有 3 种：使用动态高级语言、数据库编程和组件复用，原型开发技术对用户界面的设计和实现是一种有效的方法；面向对象分析是利用面向对象的概念和方法构建软件需求模型，它关注对象的内在性质，以及对象的关系与行为。在面向对象的方法中，UML 适用于系统开发的全过程，从需求规格描述直到系统建成后的测试和维护阶段。

需求建模的目标是创建各种表现形式，用其描述什么是客户需求，建立生成软件设计的基础。基于场景的模型从用户的角度描述软件需求。用例是主要的建模元素，它叙述或以模板驱动方式描述了参与者和软件之间某个交流活动。在需求获取过程中得到的用例定义了特定功能或交互活动的关键步骤。还可以使用活动图说明场景，即一种类似于流程图的图形表示形式，描述在特定场景中的处理流。泳道图显示了如何给不同的参与者或类分配处理流。数据建模常用于描述软件构建或操作的信息空间。为了识别分析类，基于类的建模方法从基于场景和面向流的建模元素中导出信息。可以使用语法分析从文本叙述中提取候选类、属性和操作，然后根据选择特征对候选类作出接受或拒绝的决定。CRC 建模是一种简单的建模方法，可以用 CRC 索引卡组织类并定义类之间的联系，CRC 模型中的职责是和类相关的属性和操作，协作者则是提供完成某个职责所需要信息的类。

2.7 习题

1．结构化分析方法是什么年代提出的，主要创建者是谁？

2．结构化分析方法使用的工具主要有哪些？

3．总结一下数据流图的画法步骤。

4．某系统有一张数据流图的编号为 2.1.2.2.5，其父图中与该图相对应的加工处理的编号是什么？该图位于分层 DFD 的第几层，写出其父图的编号。

5．DFD 的基本组成元素有哪些，如何表示？

6．什么是数据词典，编写数据词典应注意哪些问题？

7．分层数据流图的平衡指的是什么？

8．一套分层 DFD 中每张图的加工处理不宜超过几个，依据是什么？

9．DFD 中数据守恒指的是什么？

10．如何检查数据流图的正确性，如何改进数据流图？

11．指出图 2-32 所示的数据流图中的错误。

图中相关的数据流及包含的数据项如下。

A: a1, a2, b1, m;

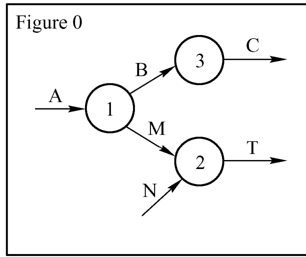
B: a1, b1, b2;

M: m, a2;

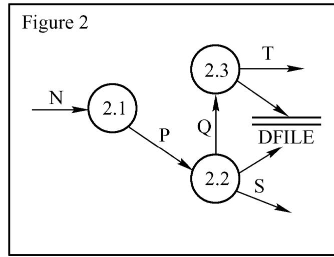
N: t, ns;

C: a1, b1, b2;

T: m, a2, ns。



a)

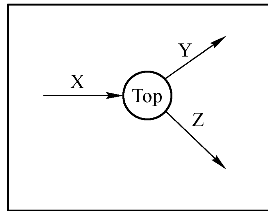


b)

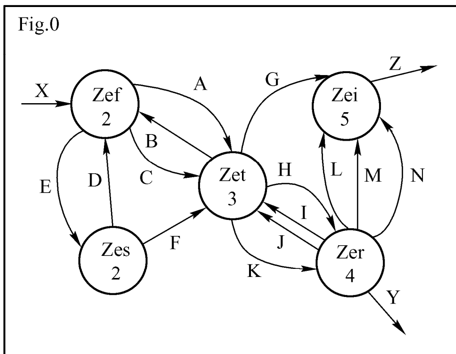
图 2-32　题 11 的数据流图

a）父图　b）子图

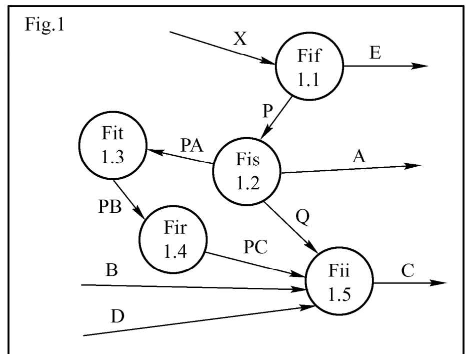
12．图 2-33 是某系统的分层数据流图，试将其重新分解，使各部分之间的联系最少。



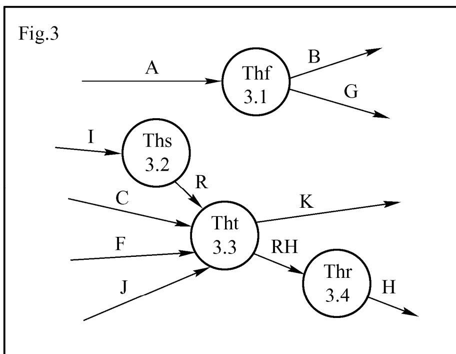
a)



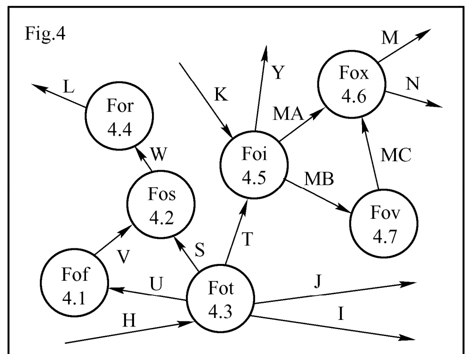
b)



c)



d)



e)

图 2-33　题 12 的数据流图

a) 父图　b) 子图　c) 子图　d) 子图　e) 子图

13．试针对图 2-2 编写数据词典。

14．写出结构化英语的约束和原则。

15．分别用结构化英语（或汉语）、判定树和判定表描述下列问题。

某商场顾客购物时收费有 4 种情况：普通顾客一次购物累计少于 100 元，按 A 类标准收费（不打折），一次购物累计等于或多于 100 元，按 B 类标准收费（打 9 折）；会员顾客一次购物累计少于 1000 元，按 C 类标准收费（打 8 折），一次购物累计等于或多于 1000 元，按 D 类标准收费（打 7 折）。

16．开发系统原型有什么好处？

17．Ian Sommerville 给出的原型开发模型将软件开发过程分为几个阶段，给出各阶段的名称。

18．抛弃式原型开发有什么特点，分析抛弃式原型开发的主要问题。

19．演化式原型开发有什么优势，在使用这种方法时要注意哪些问题？

20．增量式的原型开发过程有什么特点？

21．有哪些比较实用的快速原型技术？

22．目前，原型开发中常用的高级语言主要有哪些，分别适合什么应用领域？

23．基于复用的可视化编程方法适合于小型、简单的软件系统的开发，还是大型、复杂的软件系统的开发，为什么？

24．如何通过组件复用和集成来构造应用软件系统？

25．为什么用原型化方法开发用户界面，为什么越来越多的用户界面设计成基于 Web 的界面？

26．为什么用户界面的开发是原型化方法开发软件的重要内容？

27．面向对象方法有哪些基本概念？

28．什么是类，什么是对象，它们的关系是什么？

29．什么是面向对象方法中的封装？

30．什么是面向对象方法中的继承，什么是简单继承，什么是多重继承？

31．什么是面向对象方法中的消息？

32．面向对象方法中对象之间的结构与连接关系主要有哪几种？

33．什么是对象的多态性？

34．简述需求建模的重要性，需求模型必须实现的主要目标有哪些？

35．如果想让用例在需求建模阶段提供价值，在创建初始用例时必须回答并重点关注哪些内容？

36．如何细化初始用例？

37．图 2-21、图 2-22 和图 2-23 是什么关系？

38．数据对象与面向对象技术中说的对象是同一个概念吗？区别是什么？

39．简述 2.3.2 节中介绍的类建模中识别分析类的分类方法，这种方法将分析类分为哪些方式？

40．什么是 CRC 建模，CRC 卡片是如何组成的？

41．CRC 建模将类分为几种，分别称作什么类？

42．在 CRC 建模过程中，在给类分配职责时应该采用哪些指导原则？

43．当开发出完整的 CRC 模型时，用什么方法评审 CRC 模型？写出评审过程。

44．简要说明 UML 的主要模型图有几种，每种图的作用是什么？

45．简述基于 UML 的软件开发过程，与传统方法相比有什么特点？

46．软件需求分析中主要应完成哪些工作？

47．什么是 SRS，为什么说 SRS 是软件开发过程中一个重要的里程碑，一份好的 SRS 应具有哪些特征？

48．简要叙述 SRS 的构造原则。

49．用图示的方式说明 SRS 与哪些人员相关以及 SRS 的作用。

50．软件需求分析阶段需要建立一系列跟踪表，写出几种主要的跟踪表名称。

51．简述 Davis 等人所提出的度量 SRS 质量的方法。

第 3 章　软件系统的设计

一群朝气蓬勃的年轻创业者用合理价位买了一块地，他们决定在这里建设企业，现在面临的是怎么干的问题，哪里盖工作间和生产厂房，哪里盖库房，哪里盖办公室，盖多大、盖几层，布局、朝向如何，内部外部使用什么材料，如何施工，等等，一大堆具体问题困扰着他们，建筑设计方案可能不止一个，需要制定最合理的一个方案。软件系统的设计与此类似，软件设计是软件工程的技术核心，是软件需求的转化，是创造性的活动，它涉及软件体系结构、数据结构、程序结构、结构内元素之间的关系、用户界面布局、过程描述，等等，好的软件设计就像好的建筑设计一样，既赏心悦目，又实用。

3.1　软件设计的基本原理

软件设计是软件工程活动的技术核心，它是软件工程后续活动的基础，如果直接将需求转换成软件的实现，虽然当时会加快软件的交付，但它存在构造不稳定的隐患，而且难以测试，图 3-1 可以形象地说明进行软件设计和不进行软件设计的后果，进行了软件设计后软件过程是稳固的，如果不进行设计，即使后续工作做得再好，软件也会随时“倒塌”。

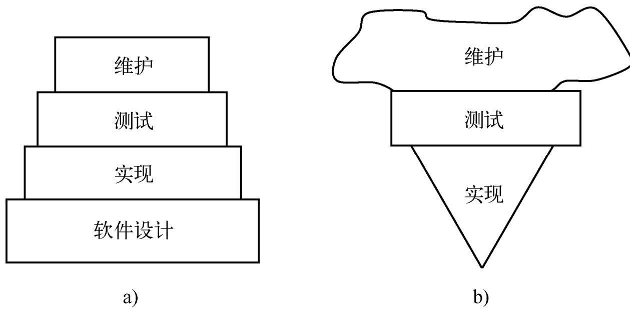


图 3-1　软件设计的重要性

a) 软件设计是软件开发过程稳定的基础　b) 不进行软件设计随时会使软件开发过程“倒塌”

软件设计是需求分析的转换，在需求分析阶段所建立的各种模型可以作为设计的依据，将它们输入到设计中，产生的输出就是设计，甚至有些模型可以直接转换成设计，图 3-2 是软件需求分析模型与软件设计模型的对应关系，数据设计及类设计将类模型转化为设计类的实现以及软件实现所要求的数据结构。CRC 模型中定义的对象和关系、类属性和其他表示法刻画的详细数据内容为数据设计活动提供了基础，在与软件体系结构

设计的连接中可能会进行部分的类设计，更详细的类设计在设计每个软件构件时进行。软件体系结构设计，一是要定义软件的主要构造元素之间的关系；二是要定义实现软件需求的体系结构风格和设计模式；三是定义影响体系结构实现方式的约束。体系结构设计表示是基于计算机系统的框架，可以从需求模型导出。接口设计描述了软件和环境之间、软件和使用人员之间是如何通信的，接口意味着信息流和特定的行为模型。因此，使用场景和行为模型为接口设计提供了所需的大量信息。构件级设计则将软件体系结构的构造元素变换为对软件构件的过程性描述，基于类的模型、流模型和行为模型获得的信息是构件设计的基础。

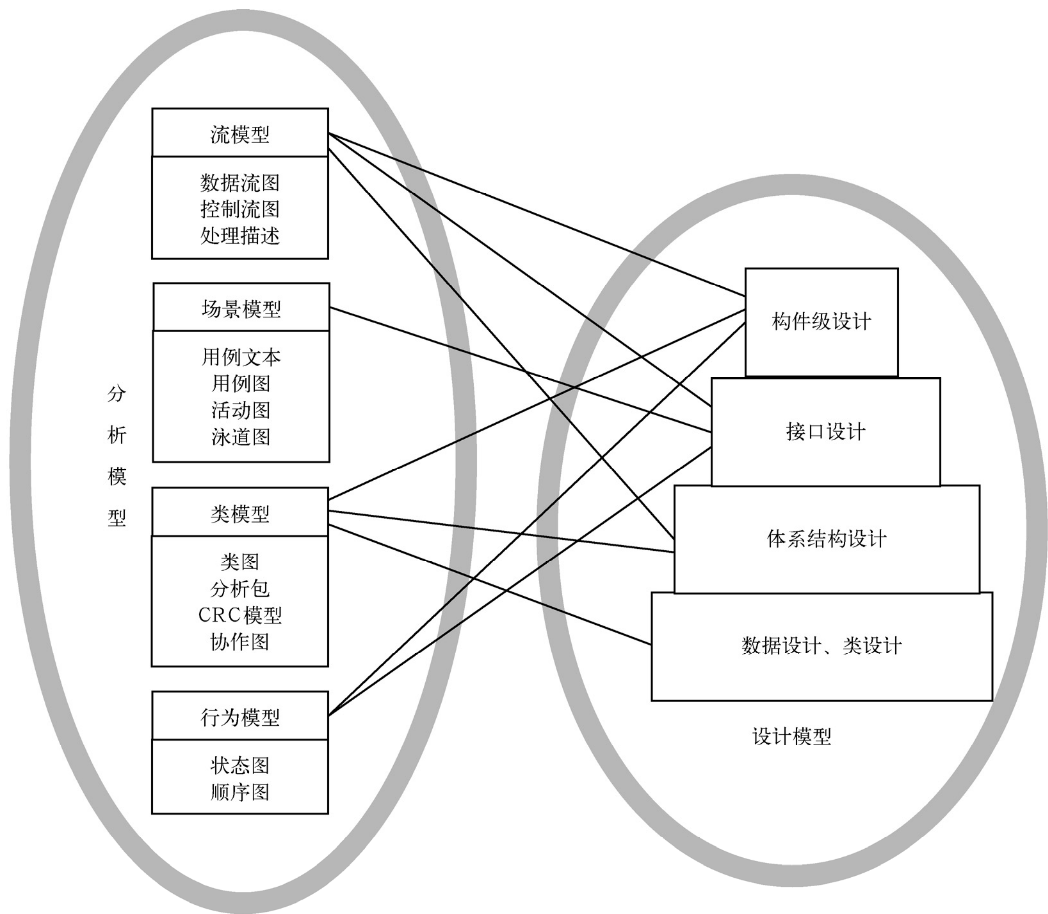


图 3-2　软件需求分析模型与软件设计模型的对应关系

在软件需求分析阶段，主要解决的是需要让所开发的软件“做什么”的问题，并且已在 SRS 中详尽和充分地进行了描述。进入设计阶段，开始着手对软件需求的实施，即着手解决“怎么做”的问题。从宏观上看，可把软件设计分为概要设计和详细设计两个阶段。软件设计的流程可用图 3-3 表示。

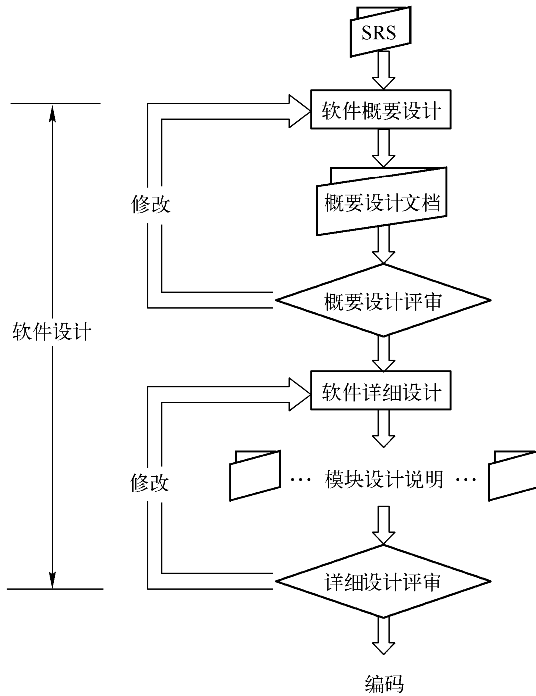


图 3-3　软件设计流程

1．概要设计

在概要设计阶段主要应完成的工作包括如下。

1）决定软件系统的总体结构，包括整个软件系统应分为哪些部分，各部分之间有什么联系以及如何将确定的需求分配到各组成部分去实现。

2）与数据有关的设计，包括文件系统的结构设计、数据库的结构设计、模式设计、数据的完整性及安全性设计。

3）对需求阶段编写的初步用户手册进行审定，在概要设计的基础上确定用户的使用方式和要求，完成系统的用户手册。

4）完成概要设计以后，应进一步细化和完善软件的初步测试计划，对测试策略、方法和步骤等提出明确要求，定义详细的测试用例。在此基础上，经过进一步完善和补充，可作为将来系统化测试工作的重要依据。

5）上述工作结束后，要组织对概要设计工作的质量进行评审，特别要评审软件的整体结构、各子系统的结构、各部分之间的联系、软件的结构如何保证需求的实现、用户的接口如何等。

2．详细设计

这一阶段主要是构件级设计，通常是采用现有的可复用软件构件，实现软件详细设计，主要有如下的 3 项工作。

1）确定实现软件各组成部分功能的算法以及各部分的内部数据组织，定义接口特征和分配给每个软件构件的通信机制。

2）选择合适的表达方式来描述各个算法的处理逻辑。

3）进行详细设计的评审，重点评估数据结构、接口和算法是否能够工作，考察算法的正确性和效率。

3．软件设计的目标和准则

软件设计所追求的目标是使设计出的软件系统易维护、易理解、具有高可靠性和高效率。

如何衡量一个软件设计是否达到了上述目标，其准则是：首先应使设计出的软件结构具有明显的层次性，以便于软件元素之间的控制；其次应使设计出的软件结构呈模块化，这些模块要具有完全独立的功能，内聚性要高，耦合性要低；三是设计出的软件与环境的界面应当清晰；最后，软件设计说明书应当清晰、简洁、完整和无歧义性。软件设计说明书也是软件工程中里程碑式的文档，是软件设计的最终成果，它既是编码人员书写源程序的依据，又是将来对系统进行测试及维护的指南。

3.1.1　软件设计的概念和原则

1．软件体系结构

随着软件系统的规模越来越大、越来越复杂，整个系统的结构和规格说明显得越来越重要。大型系统总是被分解成若干子系统，这些子系统提供一些相关的服务，概要设计过程中要识别出这些子系统并建立起子系统控制和通信的框架，这个过程称为体系结构设计。对大型复杂的软件系统来说，对总体系统结构的设计和规格说明比算法和数据结构的选择更为重要，因而研究软件体系结构对提高软件生产率和软件的可维护性都很有意义。

软件体系结构不是可运行的软件，它是一种表达，它使我们能够对设计在满足既定需求方面的有效性进行分析、在设计变更相对容易的阶段考虑体系结构可能的选择方案、降低与软件构造相关的风险。

在体系结构设计中，软件构件可能会像程序模块或者面向对象的类那样简单，也可能扩充到包含数据库和能够完成客户机与服务器网络配置的“中间件”。构件的属性是理解构件之间如何相互作用的必要特征。在体系结构层次上，不会详细说明内部属性（如算法的细节）。构件之间的关系可以像从一个模块对另一个模块进行过程调用那样简单，也可以像数据库访问协议那样复杂。

软件体系结构问题包括软件系统总体组织和全局控制、通信协议、同步、数据存取，为系统各部分分配特定功能，各部分的组织、规模、性能，在各设计方案间如何选择等。现在，软件体系结构的研究已独立于软件工程的研究，成为计算机科学的一个研究方向和独立学科分支。软件体系结构研究的主要内容涉及软件体系结构描述、软件体系结构风格、软件体系结构评价和软件体系结构的形式化方法等。解决好软件的重用、质量和维护问题，是研究软件体系结构的根本目的。

体系结构的设计过程主要关心的是为系统建立一个基本架构，它包括要识别出系统的主要组件以及这些组件之间的通信。体系结构设计过程的输出是一个体系结构的设计文档，文档包括一系列的图形化的系统模型描述和一些相关的描述文本。该文档描述了系统如何由子系统构成以及每个子系统如何由模块构成，系统的不同图解模型是从不同角度来分析体系结

构的产物。开发的体系结构模型可能包括以下内容。

1）静态结构模型。给出子系统或组件，将其作为一个个独立的单元来开发。

2）动态过程模型。给出系统在运行时的过程组成，它可能不同于静态模型。

3）接口模型。定义每个子系统从它们的公共接口能得到的服务。

4）关系模型。给出子系统间的关系，如数据流关系。

体系结构的设计要基于一些特别的体系结构模型或风格，只有对这些模型相当熟悉，使用起来才能得心应手。图 3-4 至图 3-7 是一些常见的体系结构风格，图 3-4 是以数据为中心的体系结构，这种结构的中心是数据存储，其他构件会经常访问该数据存储，并对存储中的数据进行更新、增加、删除或者修改。图 3-5 是管道-过滤器结构的例子，每个构件称为过滤器，这些构件通过管道连接，管道将数据从一个构件传送到下一个构件。每个过滤器独立于其上游和下游的构件而工作，过滤器的设计要针对某种形式的数据输入，并且产生某种特定形式的数据输出。调用和返回体系结构也是常用的风格，它有几种子风格，如主程序/子程序体系结构风格、远程过程调用体系结构风格等，图 3-6 是主程序/子程序体系结构风格的例子，这种结构中，主程序调用一组程序构件，这些程序构件又去调用其他构件。图 3-7 是层次体系结构风格，这种结构由一系列不同层次构成，每个层次各自完成操作，这些操作逐渐接近机器的指令集。在外层，构件完成建立用户界面的操作，在内层，构件完成建立操作系统接口的操作。中间层提供各种实用工具服务和应用软件功能。面向对象的体系结构也是常用的风格，该结构中的构件封装了数据和必须用于控制该数据的操作，构件间通过信息传递进行通信与合作。

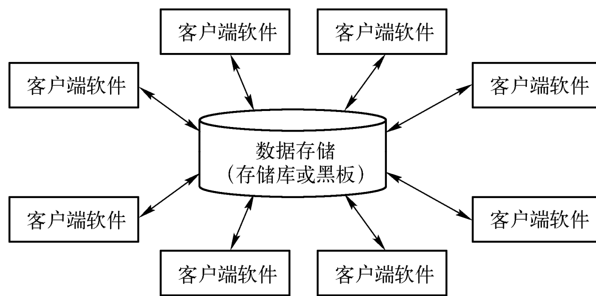


图 3-4　以数据为中心的体系结构

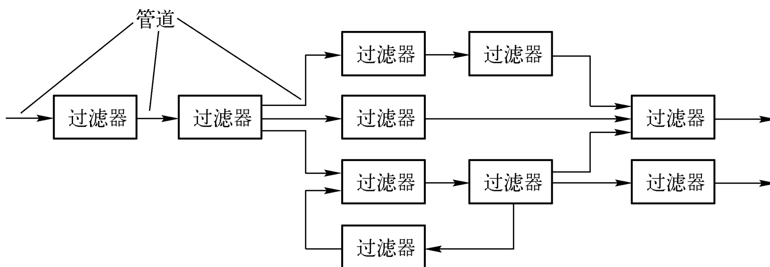


图 3-5　管道-过滤器体系结构

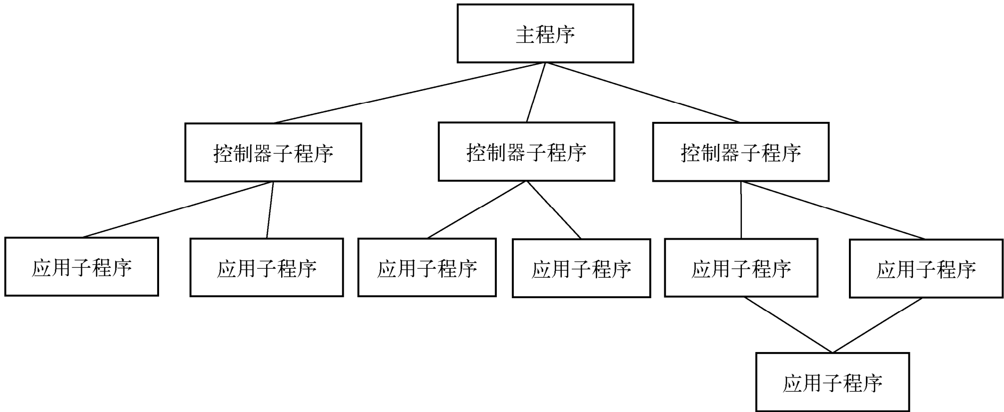


图 3-6 主程序/子程序体系结构

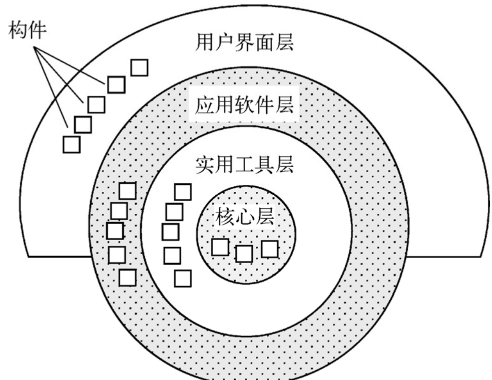


图 3-7 层次体系结构

常见的体系结构风格还有客户-服务器风格、翻译风格、过程控制风格等，就像建筑上有公寓风格、办公楼风格、商业楼风格、工厂风格、博物馆风格、哥特式风格等一样，每种风格还有若干子风格，软件体系结构每种风格也有若干子风格，一旦需求工程确定了待建系统的特征和约束，就可以选择最适合这些特征和约束的体系结构风格或风格的组合。在很多情况下，可能会有多种风格适合当前的系统，这时要对选择的体系结构风格进行评估，以确保它与需求以及它的构件之间的正确性、清晰性、完整性和一致性。

不但要了解模型本身，还要了解它们的应用领域及各自的优缺点。如结构化模型中的容器模型、客户机/服务器模型、抽象机模型，控制模型中的集中式模型和事件驱动模型，模块分解模型中的面向对象模型和数据流模型等虽都是通用模型，但都有不同的特点，除此以外，对于特别的应用可能还需要特别的体系结构模型。这些体系结构的模型为领域相关的体系结构，有两种领域相关的体系结构模型：类模型和参考模型。类模型是从许多实际系统中抽象出来的一般模型，它们封装了这些系统的主要特征。参考模型是更抽象且是描述一大类系统的模型，它是对设计者有关某类系统的一般结构的指导。

2．软件结构中的若干概念

软件结构是软件元素（即模块）之间关系的表示。由于软件元素间的关系是各种各样

的，如调用关系、包含关系、从属关系和嵌套关系等，因而软件的结构也是多种多样的。图 3-8 为一个调用和返回的软件结构，其中的方框表示模块。

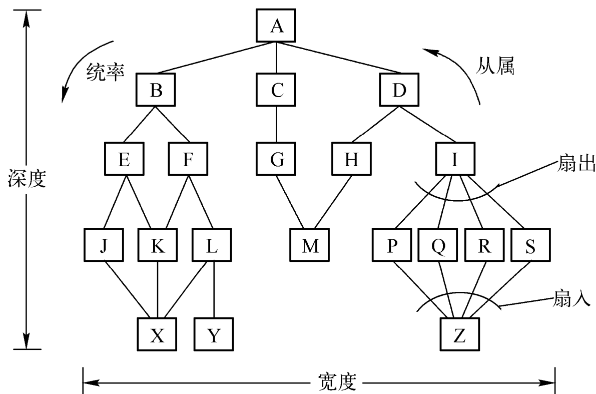


图 3-8　软件结构图示例

（1）深度（Depth）

表示软件结构中的控制层数，如图 3-8 中的结构，Depth=5。深度往往能粗略地反映一个系统的大小和复杂程度，深度和程序长度之间也应有粗略的对应关系，当然这个对应关系与模块的大小有关。

（2）宽度（Width）

表示软件结构的总跨度，如图 3-8 中 Width=8。宽度也能粗略地反映一个系统的大小和复杂程度，一般说来，宽度越大，则系统越复杂。对宽度影响最大的是模块的扇出。

（3）扇入数（Fan-in）

指有多少模块直接控制一个给定的模块，图 3-8 中 Z 的扇入为 4，H 的扇入为 1。一个模块的扇入越大，说明共享该模块的上级模块数越多，这虽有一定的益处，但一定要考虑到模块的独立性原理。

（4）扇出数（Fan-out）

指由一个模块所直接控制的其他模块数，图 3-8 中 D 的扇出为 2，I 的扇出为 4。扇出越大表明模块越复杂，因为它要协调和控制过多的下级模块，扇出过小（例如总是 1）也不好，一般认为，一个设计得好的系统平均扇出数是 3 或 4。

（5）统率（Superordinate）和从属（Subordinate）

若一个模块控制另一个模块则说前者统率后者，或者说后者从属于前者，如图 3-8 中，模块 C 统率 G，也统率 M，A 统率所有模块，H 从属于 D，最终从属于 A。

3．软件的模块化

模块化的概念在软件中已使用了几十年，所谓模块就是单独命名的可编址的元素，若组合成层次结构形式就是一个可执行的软件，也就是满足一个软件项目需求的可行解。

Myers 曾说："模块化是软件的唯一属性，它使得一个程序易于进行智能管理"，单个模块组成的大型程序（即不分模块）是不易掌握的，控制路径多，引用变量多，全局的复杂性使它几乎无法理解。

模块化的目的是为了降低软件的复杂性，使软件设计、调试、维护等操作简单、容易。

关于模块化可以降低软件复杂性这一说法，有人用下面的分析来加以论证。

设 $C(x)$表示问题 x 的复杂度函数，$E(x)$是解决问题 x 所需的工作量，对于两个问题 p_1 和 p_2，若

$$C(p_1) > C(p_2) \tag{3-1}$$

则

$$E(p_1) > E(p_2) \tag{3-2}$$

显然，解决复杂的问题比解决简单的问题需花费更多的工作量。若 $p = p_1+p_2$，即问题 p 可分解为两个问题 p_1 和 p_2，则根据经验，p_1 和 p_2 组合后的复杂性比单独解决每个问题时的复杂性要大，即

$$C(p_1 + p_2) > C(p_1) + C(p_2) \tag{3-3}$$

由（3-1）和（3-2）可得出

$$E(p_1 + p_2) > E(p_1) + E(p_2) \tag{3-4}$$

由此可得出结论：如果把软件无限细分，则开发它所需的工作量将趋于零，但事实并非如此。当把模块划分得越小，则花在单个模块上的工作量确实越来越小，但同时，花在模块与模块之间联系的工作量——即接口工作量却越来越大，如图 3-9 所示，实际总工作量是两条曲线的叠加，即图中虚线所示。从图上可以看出，存在一个使软件开发工作量最小的区域 M，当模块数落在该区域时，才能使开发工作量和成本都最小。

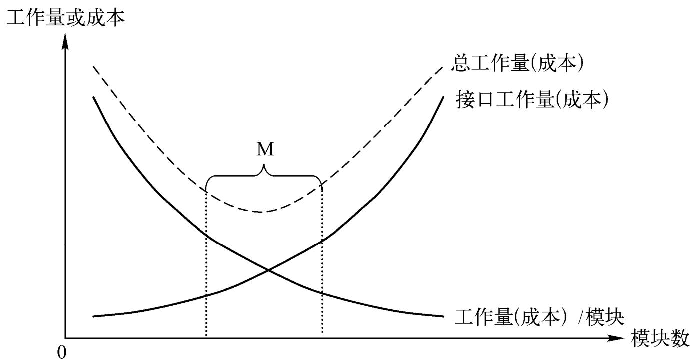


图 3-9　软件模块化和工作量（成本）的关系

前面论证的错误在于没有考虑图 3-9 中的接口工作量曲线，实际上关系式（3-3）的右端应为 $C(p_1 + I_1 p_1) + C(p_2 + I_2 p_2)$ ，此处的 I_1 和 I_2 分别为 p_1 和 $p2$ 与外界联系的因子。当 I_1 和 I_2 较大时，大于号就不成立了，表明模块与外界联系多，模块的独立性差；当 I_1 和 I_2 较小时，大于号才成立，表明模块与外部联系少，模块的独立性强。

由以上分析可知，软件模块化的过程中必须致力于降低模块与外部的联系，提高模块的独立性，才能有效地降低软件复杂性，使软件设计、调试、维护等过程变得容易和简单。虽然获得 M 的值是困难的，但我们可以定性地分析，由美国 IBM 公司的 Myers 等人给出了两种定性准则来度量模块的独立性，即耦合性和内聚性。

（1）耦合性（Coupling）

耦合性或称为耦合度，是模块之间关联性的尺度，它提供了一个对一个模块与其他模

块、全局数据和外部环境的连接的指标。Myers 等人从耦合的机制上将耦合分为非直接耦合、数据耦合、标记耦合、控制耦合、外部耦合、公共耦合和内容耦合 7 种类型，并对其进行了比较和分析，图 3-10 是 7 种类型的耦合关系图。

图 3-10　7 种类型的耦合关系图

1）内容耦合。指一个模块直接引用另一个模块的内容，被引用模块的任何变化，或者用不同的编译程序文本对它再编译都会造成程序出错。这种耦合一般出现在汇编语言中。

2）公共耦合。指一组模块都引用同一全局数据结构。例如 FORTRAN 程序中，访问 COMMON 区中数据的那些模块、访问绝对存储单元的那些模块都是公共耦合。

3）外部耦合。指一组模块都引用同一全局数据项，外部耦合与公共耦合引起的问题类似，但外部耦合中不存在依赖于一个结构内各项的物理安排的问题。

4）控制耦合。指一个模块通过某种信息控制另一个模块，这种耦合的实质是要在单一接口上选择多功能模块中的某项功能。因此，对被控制模块接口的任何修改，都将影响控制模块，此外，控制模块必须知道被控制模块的一些逻辑关系，这些都有损于模块的独立性。

5）标记耦合。若模块之间通过变元表传递数据结构，则模块之间的联系就是标记耦合。例如 A 模块把一个记录传送给 B 模块，则 A 和 B 都需了解记录的数据结构。在设计中，应尽量避免标记耦合，因为它增加了模块之间不必要的联系。例如，若 B 只需记录中的某些项，则不必传送整个记录，如果采用信息隐蔽的办法，把数据结构上所有操作都孤立成一个模块，则可以消除这种耦合。

6）数据耦合。若模块之间通过变元表传递数据，则模块之间的联系就是标记耦合。例如，上例中 B 的功能是打印雇员的一个信封，则不用把全部记录传送给 B，而是只传送雇员名字、地址、邮政编码等数据项，这些都是作为参数来传送，B 模块不依赖于私人记录，A 和 B 更加独立。

7）非直接耦合。指模块之间没有直接关系，它们之间的联系完全是通过上级模块的控制和调用来实现的，因而模块的独立性最强。

原则上讲，我们总是希望模块之间的耦合均为非直接耦合方式，但有时很难做到。一方面是由于问题本身固有的复杂性，另一方面如果片面追求非直接耦合方式，可能时、空两方面所花的代价太大。此外，有时其他耦合类型可能更适合问题的特性。例如，某个程序有 4 种类型的错误信息，把它们分别放在相应的模块中处理将不增加耦合性，从耦合性的度量上看这是一种好的方案。另一种方案是把 4 种错误信息集中放入同一模块中，通过调用模块和传送错误类型到模块接口上来进行处理，这就形成了控制耦合，但这样做可以消去重复信息，使所有错误信息格式标准化，从而可把设计人员的精力更好地集中在效率和弄清错误信息方面，所以，耦合类型的选择，应根据实际情况，全面权衡，综合考虑。

（2）内聚性(Cohesion)

也称为聚合性或聚合度，它是度量一个模块能完成一项功能的能力。一个好的内聚模块应当恰好做一件事，人们总是希望模块的内聚性越高越好，即模块强度越强越好，根据模块内部的构成情况，可以把内聚分为高内聚、中内聚和低内聚 3 类。其中高内聚包含功能内聚和顺序内聚；中内聚包含通信内聚和过程内聚；低内聚包含时间内聚、逻辑内聚和偶发内聚。图 3-11 是这 7 种类型内聚的关系图。从图中可以看出，功能内聚的模块独立性最强，偶发内聚模块独立性最弱。

图 3-11　7 种类型的内聚关系图

1）偶发强度。模块内的元素间没有实质性的联系，因而它的强度是最弱的。例如，为了节省空间，将几个模块中共同的语句抽出来放在一起组成的模块就属于此类。这种模块不仅难以修改，而且无法定义其功能，因而要尽量避免这种强度的模块。

2）逻辑强度。将几个逻辑上相关的功能组合起来形成的模块，实际上这些“相关”功能并无实质上的联系。这类模块的特点也是不易修改，另外，当调用时需要进行参数传递，这就增加了模块与模块之间的耦合，将用不到的部分也同样调入内存，因而降低了效率。例如，将各种错误信息处理集中起来定义为一个模块，但这些错误信息处理彼此间并无关系，这就形成了具有逻辑强度的模块。

3）时间强度。也称为古典强度，是顺序完成一类“相关”功能的模块，这种强度的模块其实是将若干在同一时间段内要做的工作集中在一起，这种模块的强度虽然比逻辑强度的模块略好些，但由于模块内各组成元素间并无实质性联系，因而修改和维护比较困难。例如，初始化模块就属于这种模块，由于初始化模块要为所有变量置初值，这就使该模块与被置初值的模块存在一种隐含关系，不易修改。

4）过程强度。顺序完成某一类顺序相关功能的模块，这种模块内的各元素是相关的，且必须以特定次序执行。例如先输入 x，再决定 y，等等。和时间强度的模块相比较，过程强度强调严格的执行次序，而时间强度强调在同一个时间段内执行，对执行次序并无要求。显然，过程强度的模块比时间强度的模块的内聚性要高。

5）通信强度。模块中所有处理元素都集中在数据结构的一个区域，通信强度的模块也具有过程强度，但它比过程强度的内聚性高。例如，一个文件的删除修改模块，删除、修改功能都是对文件这一公用数据发生关系。

6）顺序强度。模块内各个处理元素都紧密相关于同一功能，并且必须按顺序执行，其特征是模块内前一元素的输出是另一元素的输入。这类模块可以看成是多个功能强度的组合，以达到信息隐蔽，即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。这种模块的内聚性比以上各种强度的模块都高。

顺序强度与过程强度的区别在于：前者强调的是数据的顺序，而后者强调的是加工处理的先后。

7）功能强度。一个模块仅完成某一特定的具体功能，因而它的内聚性最高，具有“黑盒”的作用，如“开平方”子程序模块。

常常期望一个模块具有最强的模块强度，最低的耦合度，即设计成具有完成某种具体功能的模块，模块间传送尽可能少的数据型参数，但这并不是说其他类型的模块完全不能设计，例如初始化系统时，古典强度的模块还是需要的。

模块的内聚性和耦合性是一个问题的两个方面，一个模块当它的内聚性高的时候，与其他模块的联系——耦合性就必然小。耦合性越小，模块的相对独立性越大。内聚性越大，模块各成分之间联系越紧密，其功能越强。因此在划分软件模块时，应尽量做到“耦合性尽量小，内聚性尽量大”。

3.1.2　软件概要设计

在软件概要设计阶段应该宏观、整体、全面地考虑问题，本节讨论在软件设计中应当考虑哪些内容、采取什么策略、设计中应遵循什么准则以及模块设计中的具体方法。

1．充分理解 SRS，确定设计策略

（1）确定软件设计方法

目前，使用广泛且成熟的软件设计方法主要有面向数据流的方法和面向对象的方法。面向数据流的设计方法适用于数据处理和实时控制，如 SD 方法。面向对象的设计方法符合人类的常规思维方式，主要特征是信息隐蔽、数据抽象以及信息继承，适用于各种软件的开发，使编程更加容易，维护更加方便。20 世纪 70、80 年代时人们还曾广泛使用过面向数据结构的设计方法，这种方法适用于事务处理，如 Jackson 方法、LCP 方法等。以上方法也可混合使用，要依具体情况和设计人员对某种方法的熟悉程度来定。每种方法都有不足和局限，软件设计方法仍在不断发展中，有人把软件设计划分为结构化设计、面向对象设计和后面向对象方法 3 个发展阶段，在后面向对象时代出现了许多新的设计思想，如面向方面的方法、面向 Agent 的方法、泛型程序设计、面向构件的方法及敏捷方法等。面向 Agent 的设计方法是较活跃的研究内容，Agent 除具有对象特征外，还具有智能性、主动性、自治性、社会性和移动性，软件设计方法未来将如何发展，是否有极限，还在探讨中。

（2）考虑冗余和防卫设计

这两方面都是软件可靠性需求。冗余设计是指对同一问题由不同的程序员采用不同的程序设计风格和不同的算法设计软件，虽然提高了软件的可靠性，但大大增加了软件开发成本，降低了运行效率。

防卫设计指的是在软件设计中插入自动检错、报错和纠错功能，这种防卫性功能可以是周期性地或空间时间内主动地对整个软件系统进行校验和考核，搜索和发现异常情况，也可在软件运行时进行相应的检查和考核。检查的项目通常有：输入数据类型、属性和范围、用户输入数据的性质和顺序、栈的溢出、循环变量、选择变量、表达式中的零分母、输出数据格式等。实现这种设计常用的手段有：对数据值报表的交叉求和、数据量的跟踪、方程解的验算、概率值域判定等。

防卫性设计也将增大软件开发工作量并降低运行效率。因而，对于防卫性设计采用到何等程度将取决于对软件可靠性的要求。

对于航天类的软件系统，冗余和防卫设计是必须考虑的重点内容。

（3）确定对操作系统的引用方式

开发应用软件时要考虑的对操作系统的引用方式主要有两种，一种是把应用软件的每个模块都纳入到操作系统控制之下，这种方式比较简单，开发工作量相对较少，但运行的时空效率较低。一般用于软件功能要求还没有完全确定、是否需要扩充、如何扩充和扩充范围都不十分清楚的情况下。

另一种方式是在操作系统的支持下进行开发，引用操作系统中的部分功能模块生成一个独立的可运行的软件系统。这种方式设计工作量较大，一般说来，对它所进行的修改和扩充都要重新置于操作系统之下进行开发、重新生成，开发代价大，成本高，但它的运行效率高，维护方便。

以上策略性问题都是在设计工作一开始就要确定的，不确定或者确定不当都会给设计过程带来困难。

2．模块化准则

模块化准则是指应用信息隐蔽原理进行软件的初步设计，以提高模块的独立性，信息隐蔽原理是软件工程学中的一条重要原理，也是面向对象方法的重要特征。信息隐蔽原理由Parnas 提出，他认为在设计模块时，包含在模块内的过程和数据对于无须这些信息的其他模块是不可存取的。根据信息隐蔽原理，在概要设计中应该列出将来可能发生变化的因素，并在划分模块时将一些可能发生变化的因素隐含在某个模块内部，使其他模块与此因素无关。对于这样构造的软件系统，当在测试期间及往后的软件维护期间要修改时，只改一个模块就够了，其他模块不受影响。因为大多数数据和过程都是隐蔽的，是其他模块所不知道的，所以在修改期间由于疏忽所引起的错误就很少会传播到软件内的其他位置。

3．模块设计中的具体方法

以下讨论的问题及解决方法有些是概要设计方面的，有些则是详细设计方面的，还有些可能是共同的。

（1）功能强度模块的组成

一个具有功能强度的模块，不仅要能完成指定的具体功能任务，还应告诉它的调用者完成任务的状态和不能完成的原因，因而一个完整的功能模块应包括执行某项指定任务的部分、出错处理部分和结束标志，这 3 部分应当看作是一个功能强度模块的有机组成部分，不应当分离到其他模块中去，否则将会增加模块的耦合性。

（2）消除重复功能

在设计时经常会出现几个模块具有类似的功能，这不仅浪费编写和测试时间，还可能因编写不一致给修改带来麻烦。

相似有完全相似和局部相似两种。对于完全相似的情况可完全合并，只需在数据类型的描述或变量的定义上予以改进即可，而对于局部相似则要根据情况具体分析，图 3-12 中的 Q_1 和 Q_2 两个模块具有类似的功能 Q，如果将它们改造成图 3-13 的结构是不合适的，因为其实际结构如图 3-14 所示，其中 Q_m 中既含有 Q_1 和 Q_2 可共用的部分 Q，又含有非共同部分 Q_{1s} 和 Q_{2s}，Q_m 必须从模块 X 及 Y 接受一个开关量以识别是 X 调用还是 Y 调用，因而 Q_m 的内聚性是逻辑强度的，而 X 和 Y 的耦合性也较高，Q_m 的流程如图 3-15 所示。

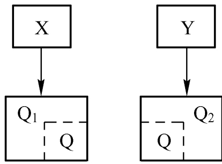


图 3-12　具有相似功能的例子

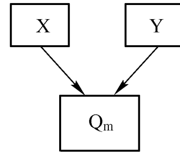


图 3-13　对图 3-12 的不合适的合并

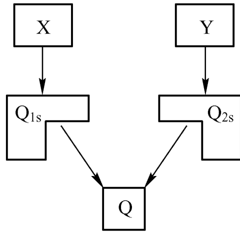


图 3-14　与图 3-13 等价的实际结构

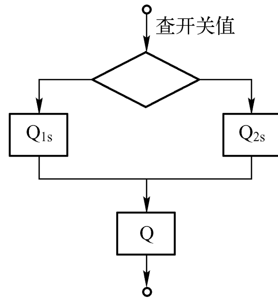


图 3-15　图 3-13 中模块 Q_m 的流程

正确的做法是：先分析 Q_1 和 Q_2，找出这两个模块中相同的功能，把它分离出来构成一个独立的下属模块 Q，然后根据 Q_{1s} 和 Q_{2s} 的大小可考虑同其父模块合并或单独构成一个模块，图 3-16a 至图 3-16d 是可能的几种方案。

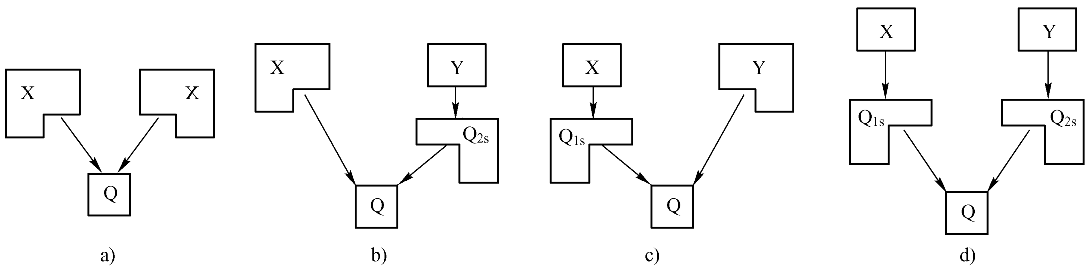


图 3-16　处理图 3-12 结构的几种可能的方案

a) Q_{1s} 和 Q_{2s} 均较小可与其父模块合并　b) Q_{1s} 较小 Q_{2s} 较大　c) Q_{1s} 较大 Q_{2s} 较小　d) Q_1s 和 Q_2s 均较大，均可单独构成模块

（3）将模块的影响范围限制在模块的控制范围之内

在软件结构的某一层上的判定可能影响其他层的处理或数据，一个模块的影响范围定义为该模块中的一个判定所影响的所有其他模块。一个模块的控制范围则是从属于或最终从属于该模块的所有模块，图 3-17a 中模块 B 的控制范围是 B、B1 和 B2，模块 B2 的控制范围是 B2，若模块 B2 中的判定影响到模块 A 和 B1 的操作，则 B2 的影响范围是 B2、B、Y、A、B1 ，这样的结构是不好的，因为 B2 的判定要传送给 B，再传送给 Y，才能传送给 A，增加了模块间的耦合性。

如果一个判断的影响范围包含在这个判断模块的控制范围之内，则称这种模块是简单的，这个概念称为“影响范围/控制范围原则”，图 3-17a 不符合这一原则，因此这个结构不好。

图 3-17b 至图 3-17d 为改进后的结构，3-17b 中影响范围在控制范围之内，但判断所处

层次太高，这样也要经过多次传送，增加了信息传送量，虽符合影响范围/控制范围原则，但不是最好的结构；3-17c 中影响范围在控制范围之内，只有一个判断分支含有一个不必要的穿越，比 3-17b 的结构好；3-17d 的结构最好，是最理想的设计，判定的影响范围恰好在判定所在模块的下一层，但整体结构发生了改变。

若在软件设计中，发现影响范围不在控制范围之内，可以选用下面的手段对结构图作改进。

1）将作判断的模块合并到它的父模块中，使判断处于足够高的位置。

2）将受判定影响的模块下移到控制范围之内，如图 3-17d 所示。

3）把判断上移到层次中足够高的位置，如图 3-17b、图 3-17c 所示。

这些手段在实现时并不容易，可能会受到其他因素的影响，这需综合考虑。

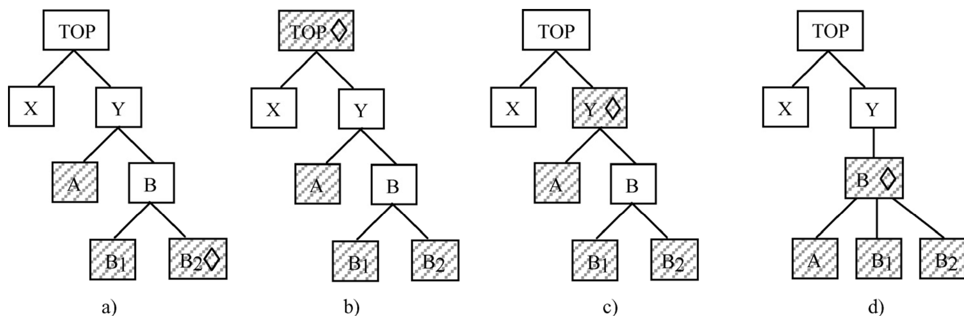


图 3-17　同一问题的不同结构

1—小菱形◇表示判定　2—阴影表示判定的影响范围

a) 不好的结构　b) 较好的结构　c) 更好的结构　d) 最好的结构

（4）关于模块的扇入数和扇出数

模块的扇入和扇出数均不宜过高。如图 3-18a 所示，一个模块被 5 个以上的模块调用，模块的扇入数偏高，如果该模块是公用的服务性模块则是正常的，否则就是该模块含有过多功能，这时可将其分解成几个同层的模块，如图 3-18b 所示，图中下层的那个模块执行所有调用模块都需要的功能。

如图 3-19a，一个模块需调用 5 个以上的模块，模块的扇出数偏高，如果该模块的功能是“分类”则是正常的，否则也是该模块含有过多功能，这时可将其分解成图 3-19b 的结构。

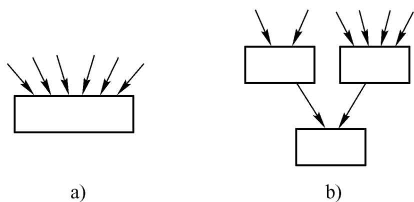


图 3-18　降低扇入数的例子

a) 模块的扇入太高　b) 分解后降低了扇入

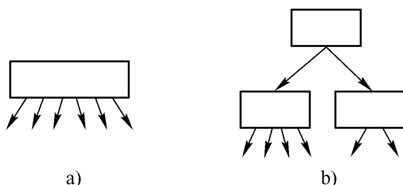


图 3-19　降低扇出数的例子

a) 模块的扇出太高　b) 重新分解后降低了扇出

设计中应尽量避免如图 3-20 所示的扁平结构，遇到这种情况可将其进行适当改进，一

个好的设计结构应该是一个寺庙的形状（即 mosque 结构），顶是尖的——一个主模块，中间较宽——中间层被分解成较多个模块，底部较窄——底层的模块可被上面的多个模块共用，如图 3-21 所示。

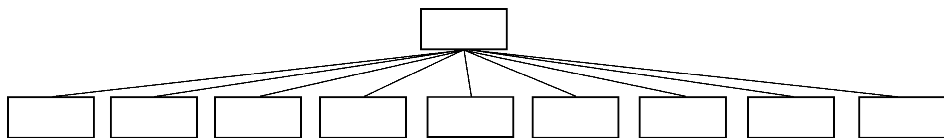


图 3-20　不好的软件结构（扁平形状）

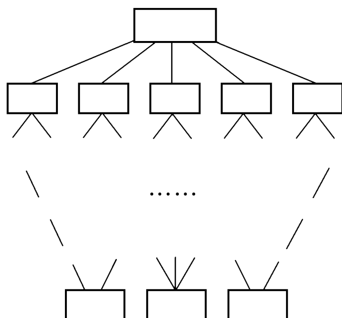


图 3-21　好的软件结构（寺庙形状）

（5）尽量设计单入口、单出口模块，避免“病态连接”

这个要求是为防止内容耦合，当一个模块在顶端进入，在底端退出时，比较容易理解和维护，“病态连接”是指转移到或引用到模块的中间。

（6）关于模块的大小

限制模块的规模是控制复杂性的有效手段之一，在设计过程中应掌握每个模块的篇幅，一般说来，模块大小以一页为宜，即 30～50 行比较合适，因为这样的篇幅比较容易阅读和理解，便于测试和维护。

在设计过程中，要估计一下每个模块的篇幅，特大的模块不易理解，可检查一下它是否包含了好几个功能，可从中分离出一些功能来构成同层或下层的其他模块。对一些特小的模块可考虑同它的父模块合并，但如果该模块的内聚性是功能强度的，或它与其他模块的耦合性低，或它的父模块很复杂，或它有多个父模块，在这些情况下都不宜合并。

4．用户界面设计

（1）界面设计的黄金规则

用户界面就像一个人的相貌和外观，如果“颜面”很差，别人就不愿意与之交往，因而用户界面设计是软件设计的重要内容，它直接影响到使用者对软件的感受，在设计用户界面时应遵循以下 3 条黄金规则。

- 软件应置于用户的控制之下。
- 减少用户的记忆负担。
- 保持界面一致。

1）软件应置于用户的控制之下。这条规则要求以不强迫用户进入不必要的或不希望的

动作的方式来定义交互模式、对不同用户能提供不同的灵活的交互、允许用户交互被中断和撤销、随着用户使用技能增长可以使交互流线化并允许定制交互、用户与内部的技术细节应隔离开来、所设计的界面应允许用户与出现在屏幕上的对象直接交互。

2）减少用户的记忆负担。用户必须记住的东西越多，和系统交互时出错的可能性也就越大。一个经过精心设计的用户界面不会加重用户的记忆负担。因而，这条规则要求减少用户的短期记忆、建立有直观意义的缺省、定义直观的快捷方式、界面的视觉布局应该基于真实世界的象征、以不断进展的方式逐渐揭示信息。

3）保持界面一致。这条规则有三个方面的要求，一是允许用户将当前任务放入有意义的环境中；二是一组应用系统都应该实现相同的设计规则，以保持所有交互的一致性；三是如果过去的交互模型已经建立起了用户期望，除非有不得已的理由，否则不要改变它。

（2）界面设计步骤

界面设计是一个迭代过程，每个用户界面设计步骤都要进行很多次，每次细化和精化的信息都来源于前面的步骤。可以使用类似于第 1 章介绍的螺旋模型来表示，如图 3-22 所示，用户界面设计过程包含以下 4 个不同的框架活动。

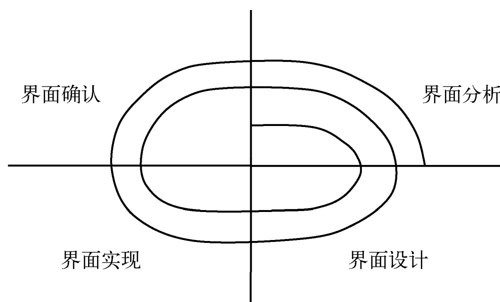


图 3-22　用户界面设计迭代过程

1）通过界面分析定义界面对象和操作。

2）定义导致用户界面状态发生变化的事件，并对行为建模，设计界面。

3）描述每个界面状态，实现界面，就像最终用户实际看到的那样。

4）判定用户能否从界面提供的信息来解释系统状态，评估界面是否满足用户的需求。

以上 4 步活动是迭代进行的，不要试图第一轮就刻画所有的细节，后续的迭代过程将精化界面的任务细节、设计信息和运行特征。

（3）WebApp 界面设计

网络已经成为人们日常生活中不可缺少的部分，Web 应用软件（简写为 WebApp）已经发展成为成熟的工具，这些工具不仅可以为最终用户提供独立的功能，而且已经同公司数据库和业务应用集成在了一起。WebApp 用户界面是它的“第一印象”，如果一个网站站点非常好用，但它的用户界面非常糟糕、缺少美感、设计风格不合适，同样会失败。一个良好的 WebApp 界面是易于理解的、宽容的，能给用户可控制感，为了达到这些特性，在设计 WebApp 界面时应遵循如下准则。

1）能在 WebApp 的当前使用中预测出用户的下个步骤，自动提供导航而无需用户查找

这一功能。

2）导航控制、菜单、图标和美学风格的使用应该在整个 WebApp 中保持一致。

3）界面应该辅助用户在整个 WebApp 中移动，但是应该坚持使用已经为应用系统建立起来的导航习惯，以这样的方式来辅助用户。

4）WebApp 的设计和界面应该优化用户的工作效率，而不是优化设计与构造 WebApp 的 Web 工程师的效率，也不是优化运行 WebApp 的客户/服务器环境的效率。

5）界面应该足够灵活，既能够使其中一些用户直接完成任务，也能够使另一些用户以一种比较随意的方式浏览 WebApp。

6）WebApp 界面及界面表示的内容应该关注在用户正在完成的任务上。

7）如果一个用户任务定义了选项或标准化输入的顺序，第一个选项物理上应该与下一个选项在一起。

8）WebApp 不应该让用户等待内部操作的完成，而应该利用多任务处理方式，从而使用户继续他的处理工作，看起来就像前面的操作完成一样。

9）界面应该简单、直观，将内容和功能分类组织，让用户学习 WebApp 的时间减到最少，并且一旦已经学习过了，当再次访问此 WebApp 时，将所需要的再学习时间减到最少。

10）隐喻应该采用用户熟悉的图片和概念，但是并不要求是现实生活的精确再现。

11）用户填写的表单、专用数据清单等工作产品必须自动保存，使得在有错误发生时数据不会丢失。

12）界面展示的所有信息对于老人和年轻人都应该是易读的。

13）应该跟踪和保存用户状态，使得当用户退出系统稍后返回时又能回到退出的地方。

WebApp 界面设计的工作流程由以下 11 项组成。

1）对需求模型中的信息进行评审，并根据需要进行优化。

2）开发 WebApp 界面布局的草图。

3）将用户目标映射到特定的界面行为。

4）定义与每个行为相关的一组用户任务。

5）为每个界面行为设计情节故事板的屏幕图像。

6）利用从美学设计中的输入来优化界面布局和情节故事板。

7）明确实现界面功能的界面对象。

8）开发用户与界面交互的过程表示。

9）开发界面的行为表示。

10）描述每种状态的界面布局。

11）优化和评审界面设计。

5．概要设计文档

概要设计主要交付的文档有概要设计说明书、数据库/数据结构设计说明书和软件测试计划。概要设计说明书是最重要的交付文档，主要包括软件系统的基本处理流程、组织结构、功能分配、模块划分、接口设计、运行设计、数据结构设计和出错处理设计等，概要设计说明书应包含的具体内容可参看表 3-1，具体开发软件系统时可参考有关标准。

表 3-1　概要设计说明书

1. 引言 　1.1 编写目的 　1.2 背景 　1.3 定义 　1.4 参考资料 2. 总体设计 　2.1 需求规定 　2.2 运行环境 　2.3 基本设计概念和处理流程 　2.4 结构 　2.5 功能需求与程序（模块）的关系 　2.6 人工处理过程 　2.7 尚未解决的问题 3. 接口设计 　3.1 用户接口 　3.2 外部接口 　3.3 内部接口 4. 运行设计 　4.1 运行模块组合 　4.2 运行控制 　4.3 运行时间 5.系统数据结构设计 　5.1 逻辑结构设计要点 　5.2 物理结构设计要点 　5.3 数据结构与程序（模块）的关系 6. 系统出错处理设计 　6.1 出错信息 　6.2 补救措施 　6.3 系统维护设计

设计结束后要组织评审，主要应集中于软件的结构设计，分析软件设计是否覆盖了已确定的软件需求，软件的每一成分是否可追溯到某一项需求。评审可由结构设计负责人、设计文档的作者、课题负责人、项目经理、对系统开发进行技术监督的软件工程师、技术专家和其他方面的代表，评审是保证软件质量的重要手段，地位和作用十分重要。

3.1.3　软件详细设计

软件的详细设计就是对软件过程的描述，由此可直接而简单地导出实现系统的代码，目前通用的详细设计工具主要有图示工具、语言工具和表格工具，不管哪种工具，都应表示出处理的顺序、精确的判定位置、重复的操作以及数据的组织和结构等，但很多工具在数据的组织和结构方面有欠缺。

1．图示工具

使用图示工具将软件的过程细节表示成图的一部分。在图中，逻辑构造采用特殊的形式来表示，以下讨论 4 种图示工具：结构化流程图、N-S 图、PAD 图和 HIPO 图。

（1）结构化流程图

流程图是使用非常广泛的方法，它的构成很简单，方框表示一个处理步，菱形表示逻辑判断，箭头表示控制流，图 3-23 为结构化流程图的基本构造，有了这些基本构造通过嵌套就可以构造出复杂的结构。如图 3-24 所示的整体是顺序结构，由两项子任务构成，第二项任务是一个 If-then-else 结构，该结构的 then 部分是一个 Repeat-until 形式的重复结构，而 else 部分又是一个 If-then-else 结构。

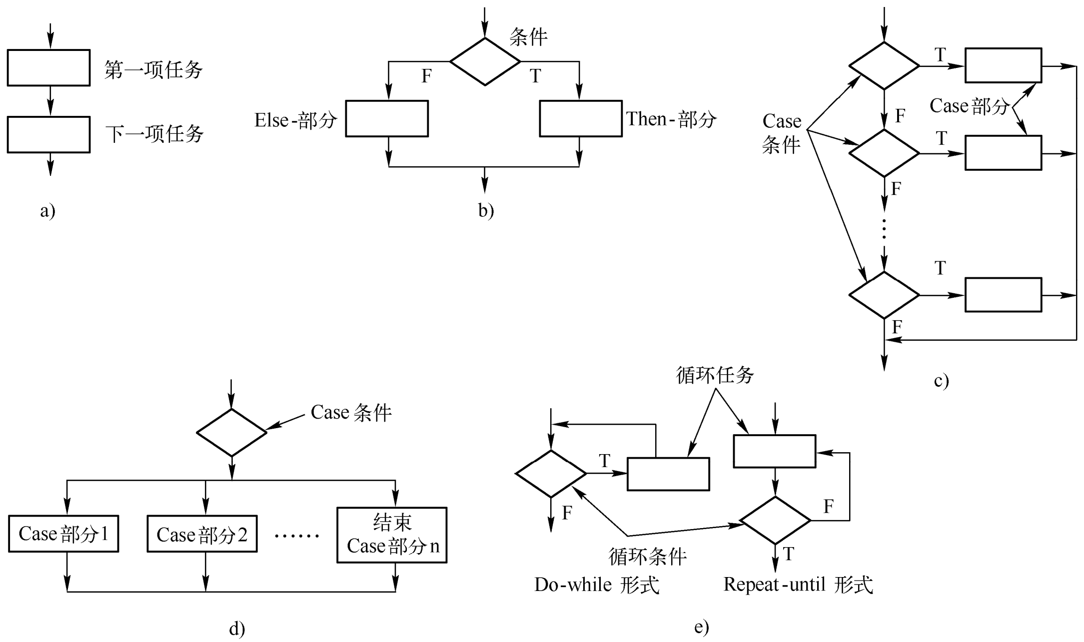


图 3-23　结构化流程图的基本构成

a) 顺序结构　b) If-then-else 结构　c) 选择结构　d) 另一种形式的选择结构　e) 重复结构

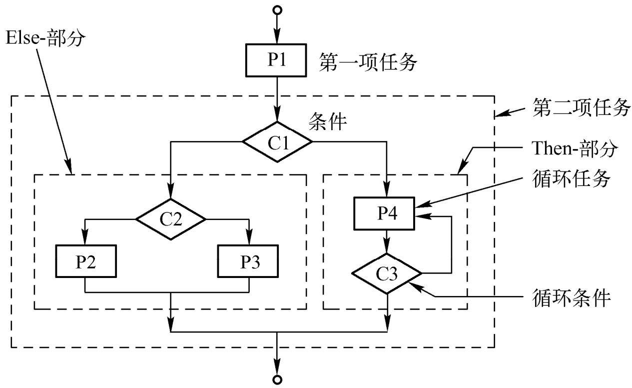


图 3-24　结构化流程图的嵌套

（2）N-S 图

也叫盒图（Box Diagram），或框图，它的基本元素是 box，由 Nassi 和 Shneiderman 开发并经 Chapin 拓展，故称为 N-S 图，或 Chapin 图。N-S 图其实是结构化流程图的变形，但它去掉了影响结构的箭头。两框或多框首尾相接表示顺序结构，一个条件框后接一个 then 部分框和一个 else 部分框并列的形式表示 If-then-else 结构。重复结构则是在框中用方框表示要重复处理的部分（Do-while 部分或 Repeat-until 部分），方框外的弯形框表示重复条件。选择形式则是将流程图中的相应表示放倒后再旋转后变形得到的，也可以看作图 3-23d 的变形。图 3-25 为 N-S 图的基本构成。由于没有从盒子内部到外部的转移方法的表示，因此保证了结构化的构造。

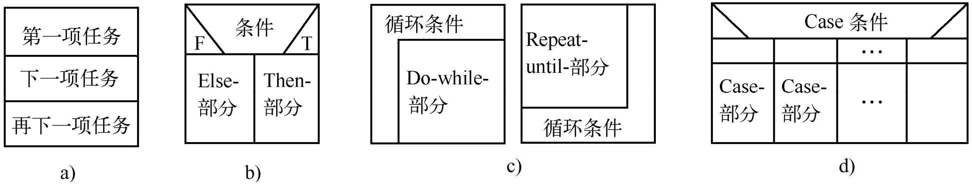


图 3-25　N-S 图的基本构成

a) 顺序结构　b) If-then-else 结构　c) 重复结构　d) 选择结构

和流程图一样，用图 3-25 的基本成分通过嵌套可以构造出复杂的结构，图 3-24 的例子用 N-S 图可表示成如图 3-26 的形式。

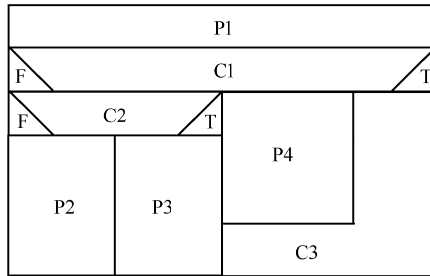


图 3-26　图 3-24 的 N-S 图表示

（3）PAD 图

问题分析图（Problem Analysis Diagram，PAD）是日本日立公司提出的一种图形工具，它综合了多种技术特征，基于 PASCAL 的控制结构，以二维树的形式描述程序的逻辑，其主要优点是程序结构清晰，能够直接导出程序代码，并对其一致性进行检查，PAD 的基本符号如表 3-2 所示，PAD 图的基本控制结构如图 3-27 所示，图中 3-27a 为顺序结构，先执行 A，再执行 B。3-27b 中的 P 是判断条件，P 取真时执行 A 框，取假时执行 B 框。3-27c 表示多分支选择。3-27d 是 Do-while 型循环。3-27e 是 Repeat-until 型循环，S 是循环体。用这些基本结构构造的图 3-24 所示的结构如图 3-28 所示。

表 3-2　PAD 的基本符号

序号	符　号	名　称	注　释
1		输入框	框内写出输入变量名
2		输出框	框内写出输出变量名
3		处理框	框内写出处理或语句名
4		重复框	先判断，再循环，框内写出重复条件
5		重复框	先执行，后判断，再循环，框内写出重复条件
6		选择框	可一路、二路、三路或多路选择，框内写出选择条件
7	或	子程序框	框内写出子程序名
8		定义框	框内写出定义名

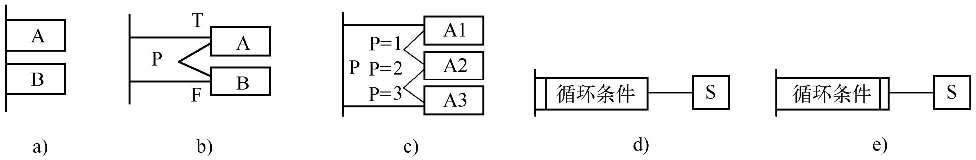


图 3-27　PAD 图的基本控制结构

a)　顺序结构　b)　If-then-else 结构　c)　选择结构　d)　Do-while 型循环　e)　Repeat-until 型循环

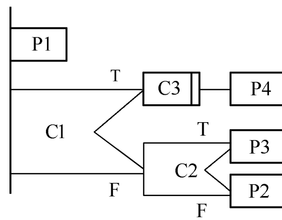


图 3-28　图 3-24 的 PAD 表示

（4）HIPO 图

层次加输入-处理-输出（Hiberarchy Plus Input-Process-Output，HIPO）图是根据 IBM 公司研制的软件设计与文件编制技术发展而来。在概要设计、详细设计、设计评审、测试和维护的不同阶段都可以使用 HIPO 图对设计进行描述。HIPO 图的最重要特征是它能够表示输入/输出数据与软件过程之间的关系。完整的 HIPO 图由如下的 3 部分组成。

1）H 图：以层次方框形式表达程序主功能模块与次功能模块之间的关系。

2）高层 IPO 图：描述 H 图中主功能模块和次功能模块的输入、处理及输出。

3）低层 IPO 图：给出 H 图中最低层次的具体设计。

图 3-29 为一个订单系统的 H 图，图 3-30 为“2.0 每月发票处理”的高层 IPO 图。

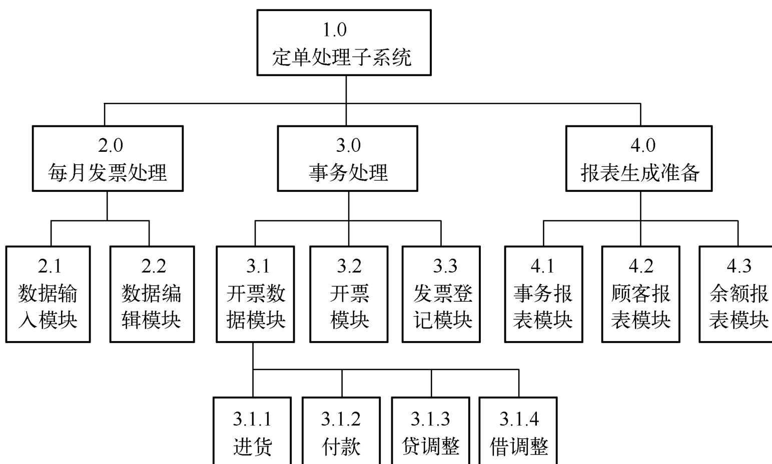


图 3-29　订单处理的 H 图

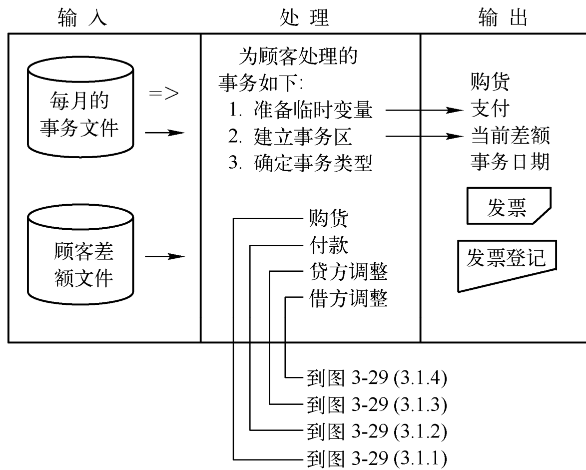


图 3-30 每月发票处理的 IPO 图

在中央处理方框中给出过程，并把它与输入、输出和数据库信息相连接。这样就能使设计者明确地把信息流与过程流联系起来。通常，处理过程用一个处理步骤来说明，但也可用流程图、N-S 图或程序设计语言来表示。

为了降低规模和维护上的复杂性，大多数项目的 HIPO 图只深入到某一层。但在通常情况下，对于编码来说仍然层次太高，一般可用伪码或其他工具做中间步骤的描述，再来编写代码。

2．表格工具

利用表格来表示软件过程细节、表格描述动作及相应的各种条件或输入处理和输出信息。判定表既是需求分析中描述处理的工具，又可用来作为软件详细设计的工具。

3．语言工具

用伪码（Pseudo Code）来表示软件过程细节，伪码表示很接近程序设计语言（PDL），结构化英语就是理想的伪码表示，是较好的语言工具。

4．工具的比较

以上介绍了一些工具，还有一些工具，有的不常使用，有的还在使用，并且工具也在不断推出，面对一个详细设计工具，如何判断它是否优劣呢？以下介绍一种方法可以对工具做出定量的评价。

给定分析条目论域 $U=\{u_1, u_2, ..., u_n\}$，看作是评语组成的集合，任一评语 u_i 的取值为：“很好”、“好”、“较好”、“一般”、“较差”、“差”和“很差”（或“不适用”）7 档或“高”、“中等”、“低”及“不适用”4 档。令“很好”=1，“好”=0.8，“较好”=0.6，“一般”=0.5，“较差”=0.4，“差”=0.2，“很差”=0；“高”=1，“中等”=0.7，“低”=0.4，“不使用”=0。取值理由如下：自然语言中“很好”就是很满意，因而“很好”取 1 是合理的，一般地说，“较好”算合格，因而取为 0.6，“好”介于它们之间，取 0.8，若“很差”（即不能适用）的语言值为 0，则“差”应为 0.2，“较差”为 0.4。“一般”是既不好也不坏故取 0.5；由于“高”、“中等”、“低”是用来描述使用频率的，因而“低”取 0 值是不合理的（若有语言值“不使用”则取 0 值），若“低”取 0.4，则由于“中等”介于“高”和“低”之间，故取 0.7。

设 $U=\{u_1, u_2, ..., u_n\}$ 为分析条目域，$u_1, u_2, ..., u_n$ 为分析条目，n 为分析条目个数，设 $\underset{\sim}{A}$ 为理想设计工具，则 $\underset{\sim}{A} \in F$ (U)，$\underset{\sim}{A}=1/u_1+1/u_2+\cdots 1/u_n$，对于任一设计工具

$\underset{\sim}{B} = x_1/u_1 + x_2/u_2 + \cdots x_n/u_n$, $x_i \in [0, 1]$，i=1, 2, …, n，定义：

$$S(\underset{\sim}{A},\underset{\sim}{B}) = 1 - e(\underset{\sim}{A},\underset{\sim}{B}) = 1 - \frac{1}{\sqrt{n}}(\sum_{i=1}^{n}(\mu_{\underset{\sim}{A}}(u_i) - \mu_{\underset{\sim}{B}}(u_i))^2)^{1/2} \quad (3-5)$$

为设计工具 $\underset{\sim}{B}$ 的满意度。其中 $e(\underset{\sim}{A},\underset{\sim}{B})$ 为模糊集 $\underset{\sim}{A}$ 与 $\underset{\sim}{B}$ 的 Euclid 距离。

取评语论域 U={u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}}，其中：u_1=易学易用性，u_2=逻辑表达，u_3=逻辑验证，u_4=易编码性，u_5=适合自动化处理方面，u_6=可修改性，u_7=结构化实施，u_8=数据表示，u_9=初期测试的简易性，u_{10}=使用频率。假设提出一种新的详细设计工具 N，对其评价如下：易学易用性好，逻辑表达能力较好，逻辑验证能力较好，易编码性很好，适合自动化处理方面一般，可修改性较差，结构化实施较差，数据表示较差，初期测试的简易性较好，使用频率暂不评论，则设计手段 N 的满意度是：

$$\begin{aligned}S(\underset{\sim}{A},\underset{\sim}{N})&=1-e(\underset{\sim}{A},\underset{\sim}{N})\\&=1-\frac{1}{\sqrt{9}}(\sum_{i=1}^{9}(\mu_{\underset{\sim}{A}}(u_i)-\mu_{\underset{\sim}{N}}(u_i))^2)^{1/2}\\&=1-\frac{1}{3}((1-0.8)^2+(1-0.6)^2+(1-0.6)^2+(1-1)^2+(1-0.5)^2\\&\quad+(1-0.4)^2+(1-0.4)^2+(1-0.4)^2+(1-0.6)^2)^{1/2}\\&=0.55\end{aligned}$$

综合评价接近较好（0.6），建议推广使用。

软件详细设计工具有 30 多种，由于某些描述并不能直接转换为程序设计语言，因此，通过归纳可选出有代表性的 6 种。表 3-3 为这 6 种设计工具的评语。设 F 表示论域 U 上的模糊集（设计工具集合），$\underset{\sim}{B_i} \in F(\mathrm{U})$，$i$=1, . . . , 6；$\underset{\sim}{B_1}$：流程图，$\underset{\sim}{B_2}$：判定表，$\underset{\sim}{B_3}$：N-S 图，$\underset{\sim}{B_4}$：程序设计语言，$\underset{\sim}{B_5}$：HIPO 图，$\underset{\sim}{B_6}$：PAD 图，若把每个 u_i（评语）看做一个模糊集，则对每个 $\underset{\sim}{B_i}$ 的单项评语可化为属于 u_i 的隶属度，根据前面的分析可得出详细设计工具 $\underset{\sim}{B_i}$ 在评语模糊集 u_i 上的投影，如表 3-4 所示。

表 3-3　6 种详细设计工具评语

序号	评语内容（分析条目）	设计工具					
		流程图	判定表	NS 图	程序设计语言	HIPO 图	PAD 图
1	易学易用性	好	较好	差	很好	好	好
2	逻辑表达	较好	很好	好	好	差	好
3	逻辑验证	差	很好	较好	较好	差	差
4	易编码性	较好	好	好	较好	较好	好
5	适合自动化处理方面	差	很好	差	很好	较好	差
6	可修改性	差	好	差	好	较好	较好
7	结构化实施	差	不适用	很好	好	较好	好
8	数据表示	差	差	差	较好	好	很好
9	初期测试的简易性	差	好	较好	差	较好	好
10	使用频率	高	低	低	中等	低	低

表 3-4　6 种详细设计工具 $\underset{\sim}{B_i}$ 在评语模糊集 u_i 上的投影

序号	u_i	F 集合					
		$\underset{\sim}{B_1}$	$\underset{\sim}{B_2}$	$\underset{\sim}{B_3}$	$\underset{\sim}{B_4}$	$\underset{\sim}{B_5}$	$\underset{\sim}{B_6}$
1	u_1	0.8	0.6	0.2	1	0.8	0.8
2	u_2	0.6	1	0.8	0.8	0.2	0.8
3	u_3	0.2	1	0.6	0.6	0.2	0.2
4	u_4	0.6	0.8	0.8	0.6	0.6	0.8
5	u_5	0.2	1	0.2	1	0.6	0.2
6	u_6	0.2	0.8	0.2	0.8	0.6	0.6
7	u_7	0.2	0	1	0.8	0.6	0.8
8	u_8	0.2	0.2	0.2	0.6	0.8	1
9	u_9	0.2	0.8	0.6	0.2	0.6	0.8
10	u_{10}	1	0.4	0.4	0.7	0.4	0.4

设 $\underset{\sim}{A} \in F(U)$，$\underset{\sim}{A} = 1/u_1 + 1/u_2 + \ldots\ldots + 1/u_n$ 表示理想设计工具，根据式（3-5）和表 3-4 计算设计工具 $\underset{\sim}{B_i}$ 的满意度

$$S(\underset{\sim}{A},\underset{\sim}{B_1})= 0.35$$

$$S(\underset{\sim}{A},\underset{\sim}{B_2})= 0.52$$

$$S(\underset{\sim}{A},\underset{\sim}{B_3})= 0.42$$

$$S(\underset{\sim}{A},\underset{\sim}{B_4})= 0.64$$

$$S(\underset{\sim}{A},\underset{\sim}{B_5})= 0.50$$

$$S(\underset{\sim}{A},\underset{\sim}{B_6})= 0.55$$

由此可见，$\underset{\sim}{B_4}$ 的满意度最高，按满意度由大到小的排列顺序是：程序设计语言、PAD 图、判定表、HIPO 图、N-S 图、流程图。

通过以上计算可知，$\underset{\sim}{B_l}$（流程图）虽是使用频率最高的详细设计工具，但却是最差的一种，$\underset{\sim}{B_l}$ 中只有 u_{10} 取值最大（u_{10}=1），表示使用频率高，使用频率高并不能说明这种设计工具好，若去掉这一评语，使

$$U=\{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9\}$$

则 $\underset{\sim}{B_l}$ 的满意度降得更低，按满意度由大到小的排列顺序仍然不变。通过计算可知，程序设计语言及 PAD 图是较好的详细设计工具，应提倡多使用。据调查，使用 PAD 图可使软件生产率提高 2 倍以上。

随着软件工程技术的发展，仍将有新的详细设计工具研制出。对于新问世的详细设计工具，如何判断其是否优良，首先应对其逐项给出评语，然后将评语转化成[0, 1]区间的数（即该工具对该项的隶属度），然后计算该工具的满意度。若“满意度＞0.5”才算是比较好的，

若“满意度＜0.5”则不够理想。若“满意度= 0.5”则应对比较内容逐项考察，择取用户最关心的主要内容计算满意度，若仍然“满意度＜0.5”则该工具不理想。

5．程序复杂性的度量

经过详细设计之后每个模块的内容都非常具体了，由此可衡量其复杂程度，但一般都是定性的，定量度量并不多，以下介绍两种较成熟的方法。

（1）Halstead 方法

严格地说该方法在代码生成之后才能使用，因为它的度量原始依据是运算符个数 n1、运算数个数 $n2$、运算符总数 $N1$ 和运算数总数 $N2$，而这些量只有在代码生成之后才能导出。

Halstead 给出的长度 N 的计算公式为：

$$N=n_1log_2\,n_1+n_2log_2n_2 \qquad (3\text{-}6)$$

根据上式可预测出程序中包含的错误个数，计算公式为：

$$E=Nlog_2(n_1+n_2)/3000 \qquad (3\text{-}7)$$

有人曾对从 300 至 12000 条语句范围的程序检验上述公式，发现预测的错误数与实际错误数相比误差在 8%之内。

虽然 Halstead 的定量度量学在实际中应用极为有限，但它有望成为关于软件可靠性、软件开发工作量及软件维护工作量的定量工具，也有望成为软件复杂性和模块性的一种形式度量。

（2）McCabe 的环形复杂性度量

该度量方法由美国人托马斯・麦克凯（Thomas J. McCabe）（如图 3-31 所示）提出，它是根据算法流程来衡量程序的复杂度。首先将算法表示成程序图，然后统计程序图中节点个数和弧数，再根据节点个数和弧数计算出程序的复杂度，也可根据封闭区域数或判定数来给出复杂度的值，此处介绍它的基本方法。

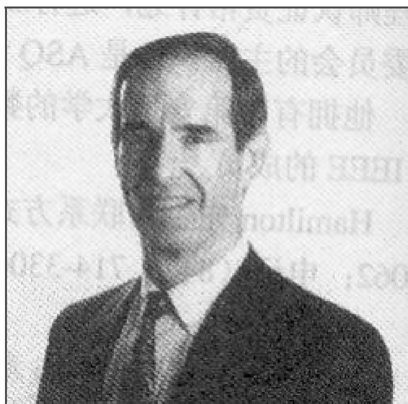
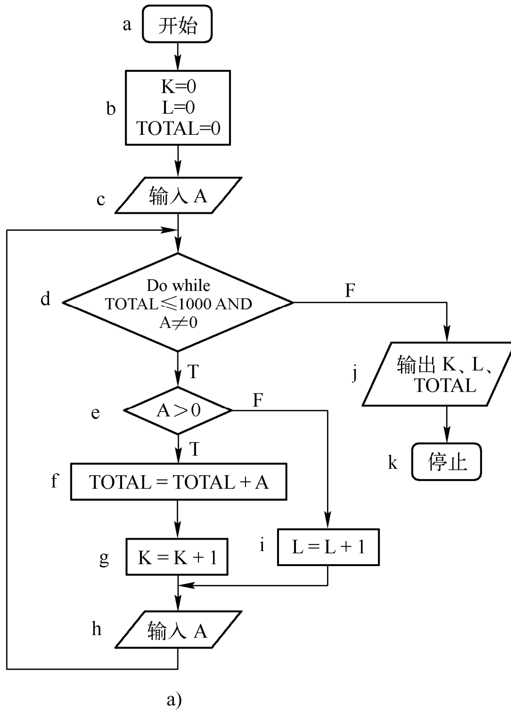


图 3-31　托马斯・麦克凯

1）程序图。程序图是程序流程图的变形，只要把流程图中所有的处理符号都用节点表示，原来连接不同处理符号的箭头变成连接不同节点的有向弧，就可得到程序图，图 3-32a 是一个程序流程图，3-32b 是相应的程序图表示。程序图只是描述程序内部的控制流程，不表现出对数据的具体操作以及分支或循环的具体条件。



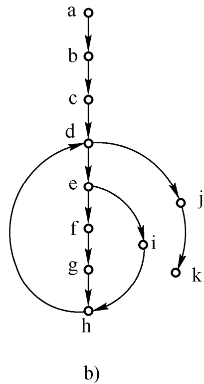


图 3-32　程序流程图转换成程序图例子

a) 程序流程图　b) 对应的程序图

通常称程序图中开始点后面的那个节点为入口点，称停止点前面的那个节点为出口点，图 3-32b 中入口点为 b，出口点为 j。

2）环形复杂度计算方法。根据图论知识，一个强连通的有向图中环的个数由式（3-8）给出，其中 $V(G)$是有向图 G 中的环数，m 是 G 中的弧数，n 是节点数，p 是分离部分数。

$$V(G) = m - n + p \tag{3-8}$$

对于一个正常的程序来说，应该能从程序图内的入口点到达图中任何一个节点，因为一个不能达到的节点表示永远不能被执行的程序代码，这显然是错误的，因此程序图总是连通的，即 p=1。

所谓强连通图是指从图中任一个节点出发都可以到达所有其他节点。程序图虽是连通的，但通常不是强连通的，因为从图中位置较低的节点不能到达较高的节点。但是，如果从出口点到入口点增加一条虚弧，则程序图就成为强连通的了。如图 3-33 是强连通图，它是图 3-32b 从出口点到入口点增加一条虚弧后得到的。

所谓环型复杂度是指强连通的程序图中环的个数 $V(G)$，图 3-33 中节点数 n=11，弧数 m=13，因而环形复杂度为

$$V(G)=13－11+1=3$$

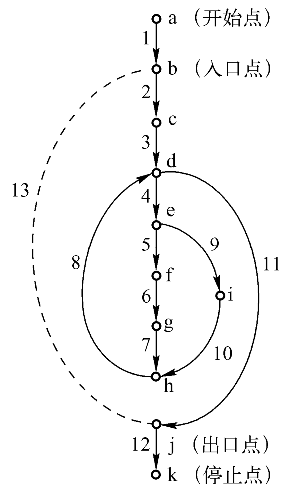


图 3-33　由图 3-32b 得到的强连通图

计算环形复杂度还有其他方法，对于平面图，环形复杂度等于强连通的程序图在平面上围成的区域个数。图 3-33 中有(b, c, d, j, b)、(d, e, f, g, h, d)和(e, i, h, g, f, e)三个区域，因而 $V(G)=3$。

3）环形复杂度的用途。程序的环形复杂度反映了程序控制流的复杂程度，当程序内分支数或循环个数增加时，环形复杂度也随之增加，因此它是对测试难度的一种定量度量，也能对软件最终的可靠性给出某种预测。

McCabe 研究大量程序后发现，环形复杂度越高程序越复杂，越容易出问题，因而通过限制 $V(G)$就可控制模块的规模和复杂程度。实践表明，模块规模以 $V(G)\leqslant 10$ 为宜。

（3）其他度量方法

对于程序的复杂程度还有其他一些定量度量方法，如 Henry 和 Kafura 提出的信息流方法是根据程序的长度 *length*、输入数据个数 *fanin* 和输出数据个数 *fanout* 来计算程序的复杂程度 P，即

$$P=lenth \times (fanin \times fanout)^2 \qquad (3\text{-}9)$$

其相对结果和环形复杂度一致。

6．详细设计文档

详细设计文档有详细设计说明书和模块开发卷宗，详细设计说明书又称程序设计说明书，是对软件系统的每个模块的详细描述，包括实现算法及逻辑流程等，编写内容可参见表 3-5，也可参考有关标准编写。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，应该把所有的模块开发卷宗汇集在一起。编写的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。模块开发卷宗的编写内容见表 3-6 和表 3-7。

表 3-5　详细设计说明书

1. 引言
   1.1 编写目的
   1.2 背景
   1.3 定义
   1.4 参考资料
2. 程序(模块)系统的组织结构
3. 程序(模块)1(标识符)设计说明
   3.1 程序(模块)描述
   3.2 功能
   3.3 性能
   3.4 输入项
   3.5 输出项
   3.6 算法
   3.7 流程逻辑
   3.8 接口
   3.9 存储分配
   3.10 注释设计
   3.11 限制条件
   3.12 测试计划
   3.13 尚未解决的问题
4. 程序(模块)2(标识符)设计说明
   …

表 3-6　模块开发卷宗

1. 标题 2. 模块开发情况表(表 3-7) 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

表 3-7　模块开发情况表

模块标识符					
模块的描述性名称					
代码设计	计划开始日期				
	实际开始日期				
	计划完成日期				
	实际完成日期				
模块测试	计划开始日期				
	实际开始日期				
	计划完成日期				
	实际完成日期				
组装测试	计划开始日期				
	实际开始日期				
	计划完成日期				
	实际完成日期				
代码复查日期及签字					
源代码行数	预计				
	实际				
目标模块大小	预计				
	实际				
模块标识符					
项目负责人批准日期及签字					

软件的详细设计完成以后，要组织评审。评审可从正确性和可维护性两个方面出发，对它的逻辑、数据结构和界面等进行检查。详细设计评审可采用以下 3 种形式之一。

1）设计者和设计组的另一个成员一起进行静态检查。

2）由一个检查小组对软件过程描述进行较正式的检查。

3）由检查小组以会议的方式进行正式的设计检查，对软件设计质量给出评价。

软件开发的实践表明，正式的详细设计评审在发现某些类型的设计错误方面和测试一样有效。因为在设计过程中发现错误更容易，以后扩大错误的机会也会减少，错误的个数也就会减少。

3.2　结构化设计方法

结构化设计方法（Structured Design, SD）是使用广泛的一种方法，由理查德·史蒂文斯（W.Richard Stevens）（1951-1999，图 3-34）、G.Myers、Edward Yourdon（第 2 章图 2-1）及拉里·康斯坦丁（Larry L.Constantine）（图 3-35）等人提出，是面向数据流的设计方法，适

用于任何软件系统的设计，它可以同分析阶段的 SA 方法及编码阶段的 SP 方法前后衔接起来使用。

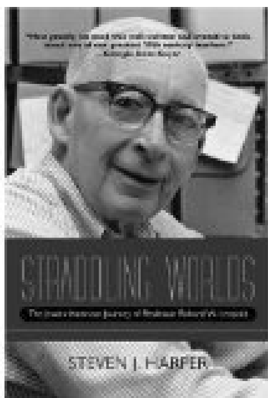


图 3-34　理查德·史蒂文斯（1951-1999）

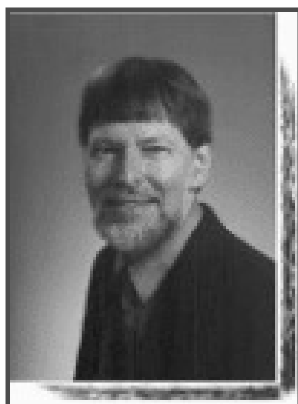


图 3-35　拉里·康斯坦丁

SD 方法的基本思想是将系统设计成由相对独立、单一功能的模块组成的结构，用 SD 方法设计的系统，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、编写、测试、报错和修改，这就使复杂的研制工作得以简化。此外，模块的相对独立性也能有效地防止错误在模块之间扩散蔓延，因而提高了系统的可靠性。

3.2.1　软件结构图的组成

软件结构图中所用的基本符号包括方框、箭头或直线、小箭头、菱形及弧形箭头。矩形方框表示模块，框中写上反映该模块功能的名字，用从一个模块指向另一个模块的箭头或直线来表示调用关系，调用旁边的小箭头表示数据传送关系，用菱形表示有条件的调用，用弧形箭头表示循环调用关系。调用顺序可依据数据传送关系确定，但一般是由左至右进行，以同一名字命名的模块在一张图中只允许出现一次。

图 3-36 表示将输入数据作计算后，将结果打印成一份报告的程序的软件结构图，该图反映了模块调用关系，数据传送关系及重复调用关系。

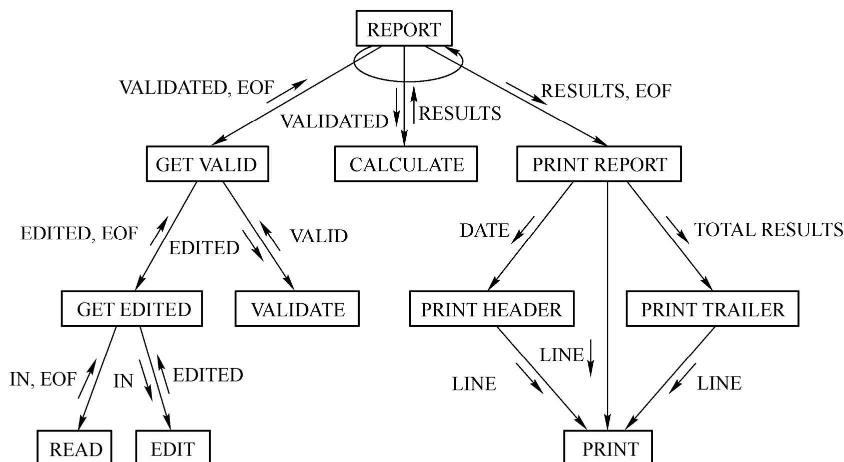


图 3-36　打印报告的软件结构图

由图 3-36 所示，顶层的报告主模块 REPORT 首先得到控制，REPORT 按从左到右的顺序调用 3 个下属模块 GET VALID、CALCULATE 和 PRINT REPORT，GET VALID 按顺序调用下属模块 GET EDITED 和 VALIDATE，GET EDITED 按顺序调用下属模块 READ 和 EDIT。

因而，最先执行的是 READ 模块，输入数据从该模块读入，将读入的数据传给父模块 GET EDITED，GET EDITED 再传给子模块 EDIT 经编辑后，将编辑后的数据 EDITED 再传给父模块 GET EDITED，GET EDITED 再将其传给它的父模块 GET VALID，GET VALID 模块再传给子模块 VALIDATE，经合理性检查后向上传送给主模块 GET VALID，GET VALID 模块将经检验是合格的数据 VALIDATED 及文件结束标志 EOF 传给系统主模块 REPORT，主模块 REPORT 再将数据传送给计算模块 CALCULATE，计算模块完成计算后将结果 RESULTS 上传送给主模块，最后主模块将结果交给打印模块 PRINT REPORT，打印模块分别要调用打印报告头 PRINT HEADER、打印一行 PRINT 和打印报告尾 PRINT TRAILER 诸模块来完成其功能，半圆弧形箭头表示以上过程将反复执行。

3.2.2 软件结构图的画法

每个软件的功能和结构可能并不一样，但有内在的规律可循，SD 方法将系统按数据流的变换规律将数据流图分为两类，一类是变换型，一类是事务型。

1．基本概念

数据处理系统的数据流图可归纳为两种典型的结构，一类是中心变换型结构，一类是事务处理型结构。中心变换型结构是一种线性状的结构，它可以明显地分成输入（又称传入 Afferent）、变换中心（Transform Center）和输出（又称传出 Efferent）3 部分。图 3-37 为具有变换流的系统模型，从该模型可以看出，信息沿输入通道进入系统，同时由外部形式变换成内部形式，进入系统的信息经过变换中心，经加工处理后再沿输出通道变换成外部形式离开系统。当数据流图具有这些特征时，这种信息流就叫做变换流。

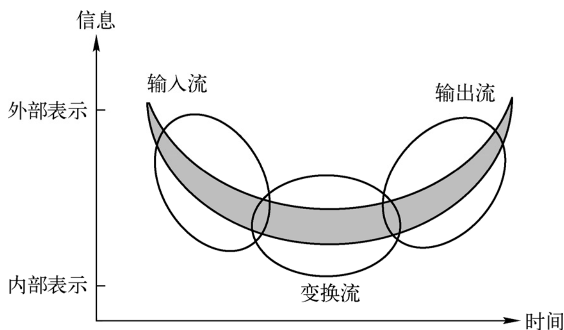


图 3-37　变换流模型

原则上，所有信息流都可以归结为变换流，但是，当数据流图具有图 3-38 类似的形状时，这种数据流是“以事务为中心的”，也就是说，数据沿输入通道到达某一个变换 T，这个变换 T 相当于一个中心，它将输入的信息分离成一串平行的数据流，然后根据输入数据的类型选择后面的若干个动作中的一个来执行。这类数据流称为事务流。

图 3-38 中的变换 T 称为事务中心，它完成下述任务。

1）接收输入数据（称为事务）。

2）分析每个事务以确定它的类型。

3）根据事务类型选取一条活动通路。

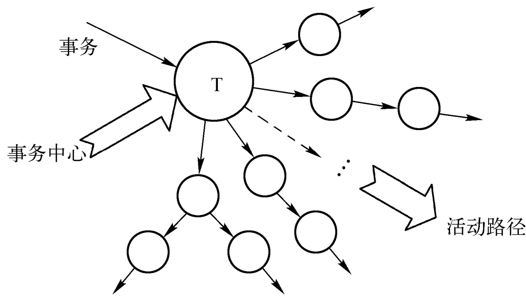


图 3-38　事务流模型

2．设计步骤

面向数据流的设计方法一般有以下 7 个步骤。

1）复审 SRS 中的 DFD，如果不够详细，则应进一步求精。

2）确定 DFD 类型。

3）把 DFD 转换成软件结构图，建立软件结构基本框架，有时也称为第一级分解。

4）进一步分解结构中的模块，有时也称为第二级分解。

5）求精并改进得到的软件结构，以便获得一个最合理的软件结构。

6）描述接口和全局数据结构。

7）复审。

上述步骤可用图 3-39 表示。

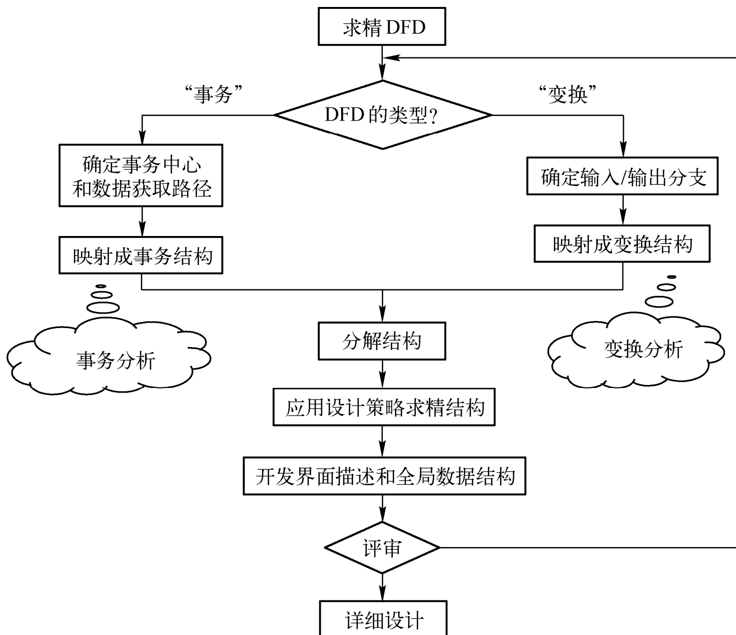


图 3-39　SD 方法设计步骤

3．变换分析方法

变换型结构的数据流图可分成 3 部分：输入、主变换和输出。主变换是系统的中心工作，主变换的输入数据对象（数据流）称为系统的逻辑输入，主变换的输出数据对象（数据流）称为系统的逻辑输出，相对地，整个系统输入端的数据对象（数据流）称为物理输入，系统输出端的数据对象（数据流）称为物理输出。从输入设备获得的物理输入一般要经过编辑、格式转换、有效性检验等一系列辅助性变换后变成纯粹的逻辑输入才传送给主变换，同样，主变换产生的纯粹的逻辑输出要经过格式转换、组成物理单元、缓冲处理等辅助性变换后成为物理输出最后才从系统送出。

使用变换分析技术可以从中心变换型结构的数据流图导出标准形式的程序结构，其过程如下（如图 3-40 所示）。

（1）确定系统的主变换、逻辑输入和逻辑输出

这一步可暂不考虑数据流图中的一些支流，如出错处理等。若设计人员参与了需求分析，对该系统的 SRS 又很熟悉，则决定哪些变换是系统的主变换是比较容易的，例如，几股数据对象的汇合处往往是系统的主变换。

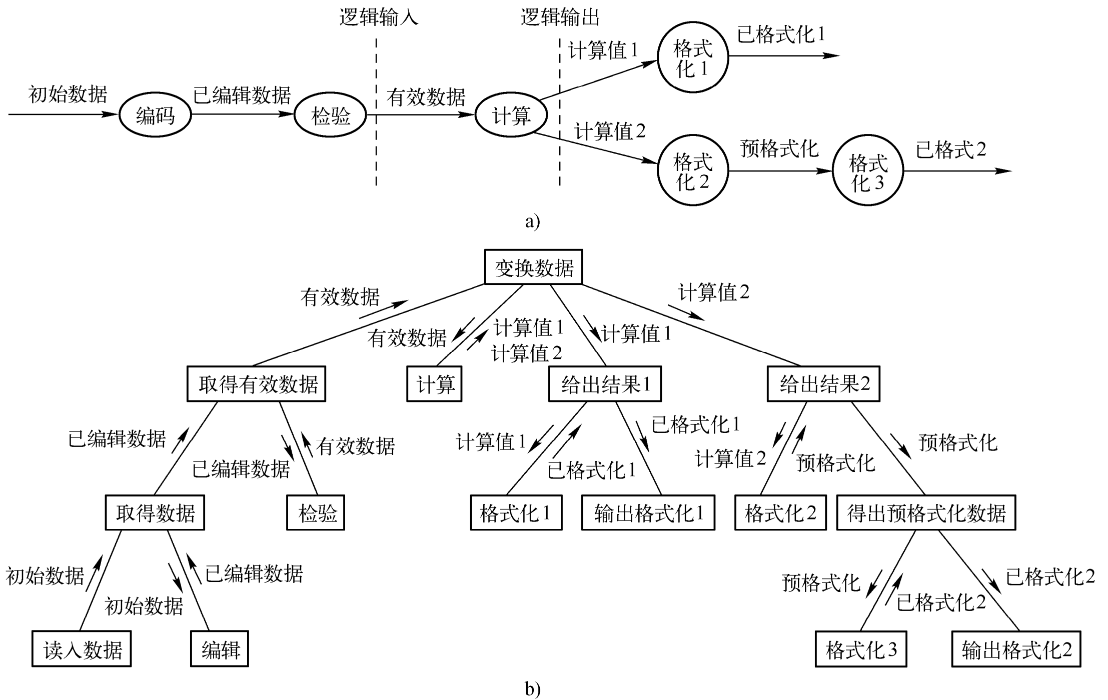


图 3-40　变换分析方法例子

a) 数据流图　b) 导出的软件结构图

如果一时不能确定主变换在哪里，则可以先确定逻辑输入和逻辑输出。方法是从物理输入端开始，一步步向系统的中间移动，直到达到这样一个数据对象：它已不能再被看作为系统的输入，则其前一个数据对象就是系统的逻辑输入。同样，从物理输出端开始，一步步向系统的中间移动，也可以找出离物理输出端最远的，但仍可被看作是系统的输出的那个数据对象就是逻辑输出。

对系统的每一股输入和输出，都可用上面的方法找出相应的逻辑输入和逻辑输出，而位于逻辑输入和逻辑输出之间的变换就是系统的主变换。

由于每个人的看法不同，不同的人找出的主变换可能也不同，但一般不会相差很远。

（2）设计模块结构的顶层和第一层

由顶向下设计的关键是找出“顶”在何处，决定了系统的主变换其实就是决定了程序结构的“顶”的位置。因而，可以先设计一个主模块，并将它画在与主变换相应的位置上，主模块的功能是完成整个程序要做的工作。程序结构的“顶”设计好之后，就可以按输入、变换、输出等分支来处理，即设计结构的第一层。先为每一个逻辑输入设计一个输入模块，它的功能是向主模块提供数据；再为每一个逻辑输出设计一个输出模块，它的功能是将主模块提供的数据输出；最后为主变换设计一个变换模块，它的功能是将逻辑输入变换成逻辑输出。

第一层模块同主模块之间传送的数据应该同数据流图相对应。

这样就得到了结构图的第一层，这里主模块控制并协调输入模块、变换模块和输出模块的工作。一般说来，它要根据一些逻辑（条件或循环）来控制对这些模块的调用。

（3）设计中、下层模块

这一步是由顶向下、逐步细化地为每一个模块设计它的下属。

输入模块的功能是向它的父模块提供数据，所以它本身必定要有一个数据来源，因此输入模块可由两部分组成，一部分是接收数据，另一部分将这些数据变换成其父模块所需要的数据。所以应该为每一个输入模块设计两个下属模块，其中一个是输入模块，另一个是变换模块。

同理，输出模块的功能是将其父模块提供的数据输出，所以它也应该由两部分组成，一部分是将父模块提供的数据变换成输出的形式，一部分是输出。所以也应该为每一个输出模块设计两个下层模块，其中一个是变换模块，另一个是输出模块。

上述设计过程可以一直进行下去，直至达到系统的物理输入端和物理输出端。

为变换模块设计下层模块，没有一定的规律可遵循，此时需研究数据流图中相应变换的组成情况。

需要注意的是调用模块与被调用模块间传送的数据应同数据流图相对应。每设计出一个新的模块应给它起一个适当的名字，以反映出这个模块的功能。

运用上述变换分析技术，可以较容易地获得与数据流图相对应的软件结构图，即与问题结构相对应的程序结构，这种软件结构符合变换型程序的标准形式，所以质量是比较高的。

4．事务分析方法

依据事务处理类型数据流图导出初始模块结构图也应先找出事务处理中心，而后由顶向下，逐步求精地进行。

虽然，在任何情况下都可以使用变换分析方法设计软件结构，但是在数据流图具有明显的事务特点时，也就是有一个明显的“发射中心”（事务中心）时，还是以采用事务分析方法为宜。

事务分析的设计步骤和变换分析的设计步骤大部分相同或相似，主要差别仅在于由数据流图到软件结构的映射方法不同。

由事务流映射成的软件结构包括一个接收分支和一个发送分支。映射出接收分支结构的方法和变换分析映射出输入结构的方法相似，即从事务中心的边界开始，把沿着接收通路的

处理映射成模块。发送分支的结构包含一个调度模块，它控制下层的所有活动模块；然后把数据流图中的每个活动流通路映射成与它的流特征相对应的结构。图 3-41 说明了上述映射过程。

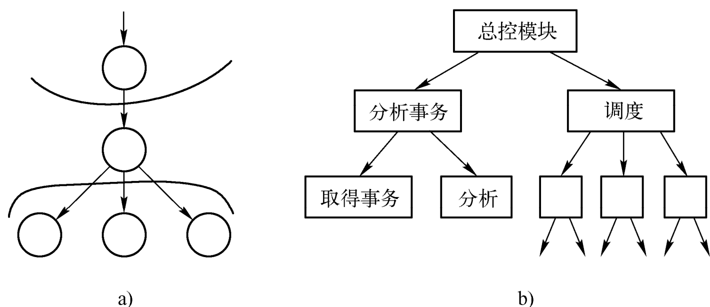


图 3-41　由事务流导出的软件结构

a)　数据流图　b) 软件结构图

具体步骤如下。

1）设计主模块。

2）设计输入、输出模块。

3）为每一种类型的事务处理设计一个事务处理模块。

4）为各个事务处理模块设计下层的操作模块。

5）为操作模块设计细节模块。

图 3-42 所示的例子说明了这一过程，导出的软件结构具有明显的 4 个层次，它们是调度层、事务处理层、操作层和细节层。

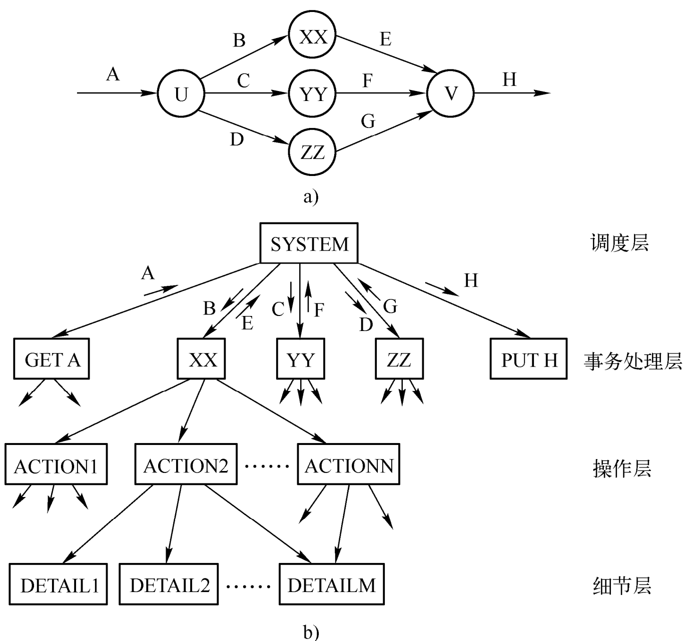


图 3-42　由事务型数据流图导出的软件结构例子

a) 数据流图　b) 软件结构图

5．混合型分析方法

实际系统的数据流图都是两种类型的混合，并不具有上述典型的形式，这时候整体上可以把数据流图看成是变换流的结构，用变换分析方法映射软件结构，在局部根据流的特征具体运用“变换分析”或“事务分析”就可得出软件结构的某个方案。如图 3-43 所示的数据流图，整体将其看作变换流，D 为逻辑输入，K 为逻辑输出，在输入部分从 B 到 D 正好是事物流结构，导出的软件结构如图 3-44 所示。

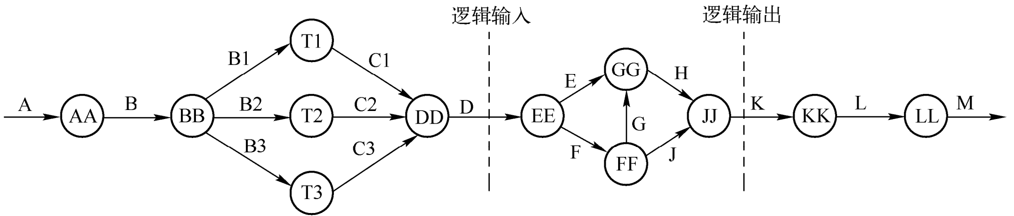


图 3-43　混合型数据流图

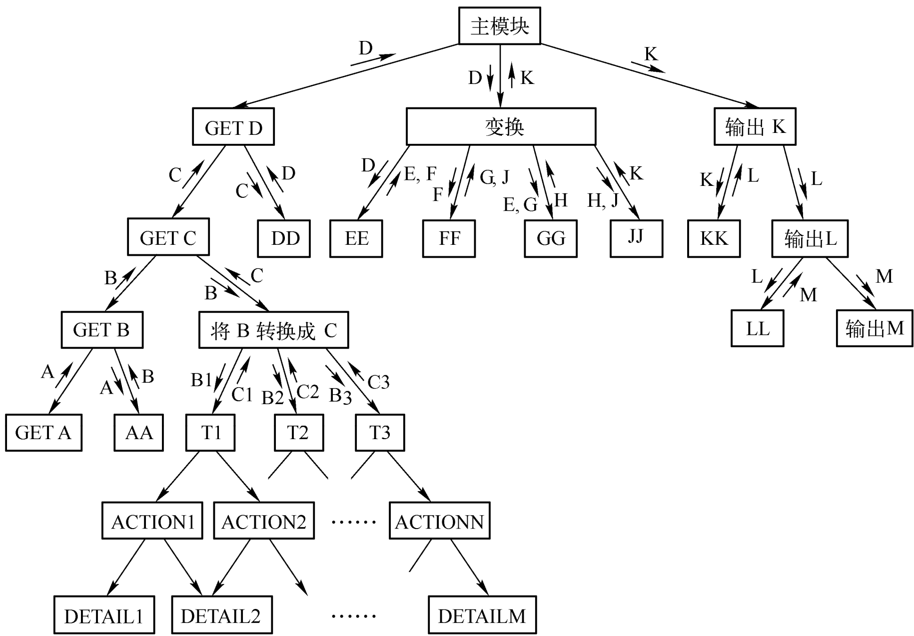


图 3-44　由图 3-43 导出的软件结构图

变换分析和事务分析方法应灵活运用，机械地遵循上述方法和步骤有时会得到一些不必要的控制模块，这时可将其合并。反之，若控制模块功能过分复杂，则可将其分解成多个控制模块，或增加中间层次。应该在设计阶段对程序结构不断地精化和评估，结构上的简单往往反映出程序的优雅和高效。设计优化应在满足模块化要求的前提下尽量减少模块数量，在满足信息需求的前提下尽量减少复杂的数据结构。例如图 3-45 所示的 DFD 也可以转换成图 3-46 所示的结构，读者可以根据前面的方法和步骤，说明转换理由。对于对性能要求很高的系统来说，可能还需要在设计的后期甚至编码阶段进行优化。实践表明，占系统 10%～20%

的程序往往占用处理时间的 50%～80%，因此，对性能要求很高的系统中最消耗时间的模块的算法要进行时间优化，以提高效率。总之，由于各个系统的具体特点不同，软件结构图的设计方法也应多样，任何满足 SRS 要求的结构图都可以作为软件结构图。

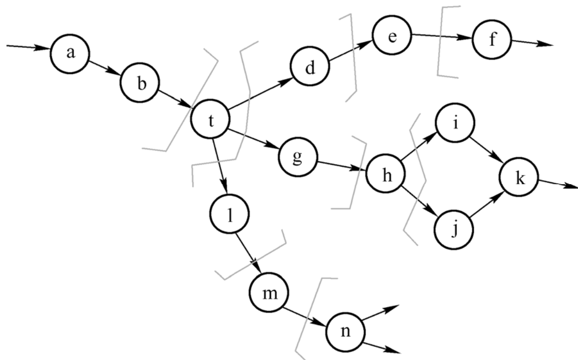


图 3-45　某系统的数据流图

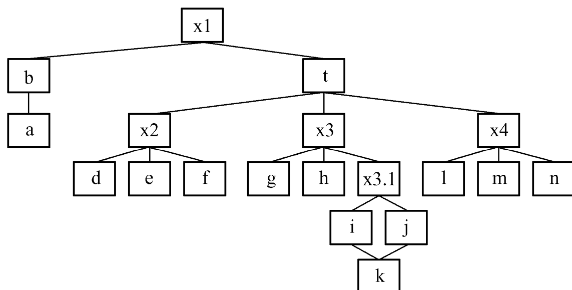


图 3-46　与图 3-45 的 DFD 对应的一种软件结构图

3.3　面向对象的设计方法

由于面向对象分析（OOA）和面向对象设计（OOD）之间没有明显的边界，所以很难区分出两个阶段，从 OOA 到 OOD 是一个逐渐扩充模型的过程，分析处理以问题为中心，可以不考虑任何与特定计算机系统有关的问题，而 OOD 则把人们带进了面向计算机系统的“实地”开发活动中。

3.3.1　面向对象设计过程

OOD 可分为两个阶段：高层设计和低层设计。高层设计是对系统的体系结构的设计，低层设计集中于类的详细设计。

1．高层设计

高层设计阶段开发软件的体系结构，构造软件的总体模型。在这个阶段，标识在计算机环境中解决问题所需要的概念，并增加一批需要的类。这些类包括那些可使应用软件与系统的外部世界交互的类。此阶段的输出是适合应用软件要求的类、类间的关系、应用的子系统视图规格说明。

2．类设计的目标

（1）设计出单一概念的模型

在分析与高层设计阶段，常常需要使用多个类来表示一个“概念”。一般地，人们在使用面向对象方法开发软件时，常常把一个概念进行分解，用一组类来表示这个概念。当然，也可以只用一个独立的类来表示一个概念。

（2）设计出可复用的“插接相容性”构件

人们希望所开发的构件可以在未来的应用中使用。因此，需要一些附加特性。例如，在相关的类的集合中界面的标准化、在一个集合内部的类的“插接相容性”等。

（3）设计出可靠的构件

应用软件的可靠性与它的构件有关，因而每个构件必须经过充分的测试。但是由于成本关系，测试往往不够完备。然而，如果要建立可复用的类，则通过测试确保构件的可靠性是绝对必要的。

（4）设计出可集成的构件

人们希望把类的实例用到其他类的开发和应用中，这要求类的界面应当尽可能小，一个类所需要的数据和操作都包含在类定义中。因此，类的设计应当尽量减少命名冲突。面向对象语言的消息语法可通过鉴别带有实例名的操作名来减少可能的命名冲突。

类结构提供的封装特性使得把概念集成到应用的工作变得很容易。封装特性保证了把一个概念的所有细节都组合在一个界面下，而信息隐蔽则保证了实现级的名字将不会同其他类的名字相互干扰。

3．通过复用设计类

面向对象技术的一个重要优点是利用既存类来设计类，许多类的设计都是基于既存类的复用。

（1）选择

设计类最简单的方法是从既存构件中简单地选择合乎要求的构件，大多数 OO 语言环境都带有原始构件库（如整数、实数和字符），它是基础层。任何基本构件库（如“基本数据结构”构件）都应建立在这些原始构件库上。它们都是一般的和可复用的类。原始构件库还包括一组提供其他应用论域服务的一般类，如窗口系统和图形图元。表 3-8 显示了建立在这些层上面的特定域的库。最低层的论域库包括了应用论域的基础概念并支持广泛的应用开发。特定项目和特定组的库包括一些论域库，它包含为相应层所定义的信息。

表 3-8　一个面向对象构件库的层次

构件层次	构件来源
特定组的构件	一个小组为组内所有成员使用而开发
特定项目的构件	一个小组为某一个项目而开发
特定问题论域的构件	购自某一个特定论域的软件销售商
一般构件	购自专门提供构件的销售商
特定语言原操作	购自一个编译器的销售商

（2）分解

最初识别的“类”常常是几个概念的组合。在设计时，可能会发现所标识的操作落在分

散的几个概念中，或者会发现，数据属性被分开放到模型中拆散概念形成的几个组内。这样必须把一个类分成几个类，希望新标识的类容易实现，或者它们已经存在。

（3）配置

在设计类时，可能会要求由既存类的实例提供类的某些特性。通过把相应类的实例声明为新类的属性来配置新类。例如，一种仿真服务器可能要求使用一个计时器来跟踪服务时间。设计者不必开发在这个行为中所需的数据和操作，而是应当找到计时器类，并在服务器类的定义中声明它。

（4）演变

要开发的新类可能与一个既存类非常类似，但不完全相同。此时，可以将一个既存类演变成一个新类，利用继承机制来表示一般化-特殊化的关系。特殊化处理有以下 3 种可能的方式。

1）由既存类建立子类。现要建立一个新类“起重车”。它的许多属性和服务都在既存类“汽车”中。关系如图 3-47 所示。新类是既存类的特殊情形，这时直接让“起重车”类作为“汽车”类的子类即可。

2）建立继承层次由既存类建立新类。现要增加一个新类“拖拉机”。它的属性与服务有的与“汽车”类相同，有的与“汽车”类不同。关系如图 3-48 所示。这时，调整继承结构。建立一个新的一般的“车辆”类，把“拖拉机”与“汽车”类的共性放到“车辆”类中，“拖拉机”与“汽车”类都成为“车辆”类的子类。“车辆”是抽象类，相关操作到子类“汽车”类去查找。

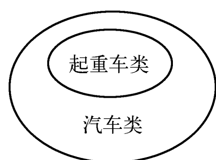
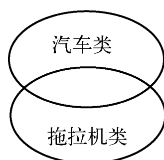


图 3-47　建立子类



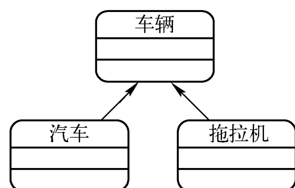


图 3-48　调整继承结构

3）建立既存类的父类。另一种情形是想在既存类的基础上加入新类，使得新类成为既存类的一般类。例如，已经存在“三角形”类、“四边形”类，想加入一个“多边形”类，并使之成为“三角形”类和“四边形”类的一般类。继承结构如图 3-49 所示。从这个“多边形”类又可派生出新的类，如“六边形”类。

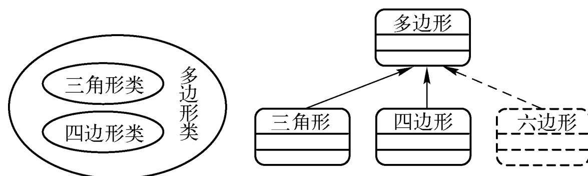


图 3-49　建立一般类

后两种方法涉及既存类的修改。在这两种情况下，既存类中定义的操作或数据被移到新类中。如果遵循信息隐蔽和数据抽象的原理，这种移动应不影响已有的使用这些类的应用。类的界面应保持一致，虽然某些操作是通过继承而不是通过类的定义延伸到这个类的。

4．类设计方法

类的设计是由数据模型化、功能定义和抽象数据类型定义混合而成。类是某些概念的一个数据模型，类的属性就是模型中的数据域，类的操作就是数据模型允许的操作。

类的标识有主动和被动之分，在被动类和主动类的设计之间不存在明显的差别，许多类是主动和被动的混合。在设计主动类时，需要优先确定数据类型，稍后再确定操作。在设计被动类时，把类提供的服务“翻译”成操作，在标识了服务之后再设计为支持服务所需要的数据。

类中对象的组成包括了私有数据结构、共享界面操作和私有操作。而消息则通过界面，执行控制和过程性命令。

类的设计描述包括以下两个部分内容。

（1）协议描述

协议描述定义了每个类可以接收的消息，建立一个类的界面。协议描述由一组消息及对每个消息的相应注释组成。

（2）实现描述

实现描述说明了每个操作的实现细节，这些操作应包含在类的消息中。实现描述必须包含充足的信息，以提供对协议描述中所描述的所有消息的适当处理。接受一个类所提供服务的用户必须熟悉执行服务的协议，既定义“什么”被描述。而服务的提供者（对象类本身）必须关心服务如何提供给用户，即实现细节的封装问题。

3.3.2　面向对象设计方法

1．Coad 与 Yourdon 方法

Coad 与 Yourdon 方法严格区分了面向对象分析 OOA 和面向对象设计 OOD。其中 OOA 的主要考虑在于与一个特定应用有关的对象以及对象与对象之间在结构和相互作用上的关系。通过 OOA 建立的系统模型以对象概念为中心，称为概念模型，由下述 5 层组成。

1）类和对象层：识别类和对象，形成整个分析模型的基础。

2）属性层：定义类和对象所要保存的信息及对象之间的实例连接。

3）服务层：定义类和对象所能提供的服务及对象之间的消息连接。

4）结构层：包括组装结构和分类结构。组装结构即整体与部分的结构，该结构用来表示聚合，即由不同的类的成员聚合而形成新的类；分类结构分为泛化与特化，该结构捕获了定义出的类的层次或网络结构。

5）主题层：主题由一组类及对象组成，用于将类和对象做进一步的组合。

Coad 与 Yourdon 在设计阶段继续采用分析阶段中的 5 个层次。不同的是，在设计阶段中，这 5 个层次是用在建立系统的 4 个组成成分上。这 4 个组成成分是：问题论域、用户界面、任务管理和数据管理。

问题论域部分包括与所面对的应用问题直接有关的所有类和对象。识别与核定这些类和对象的工作在 OOA 中已经开始，这里只是对它们做进一步细化，例如，加进有关如何利用现有的程序库的细节，以便于系统的实现。在其他的 3 个部分中，识别和定义新的类和对象。这些类和对象形成问题论域部分与用户、与外部系统和末用设备、与磁盘文件和数据管理系统的界面。这 3 部分的作用主要是保证系统基本功能的相对独立，以加强软件的可复用性。假如外部的系统更新了，相应的通信协议也应有所变化。在这种情况下，只需修改任何

管理部分中的某些类和对象，而不必对其他几个部分做任何修改。

（1）问题论域部分的设计

对在 OOA 中得到的结果进行改进和增补，主要是根据需求的变化，对 OOA 产生模型中的某些类与对象、结构、属性、操作进行组合与分解。要考虑对时间和空间的折中、内存管理、开发人员的变更以及类的调整等。另外根据 OOD 的附加原则，增加必要的类、属性和联系，主要工作内容如下。

1）复用设计。

2）把问题论域的未用类关联起来。

3）为建立公共操作集合建立一般类。

4）调整继承级别。

（2）用户界面部分的设计

根据需求把交互的细节加入到用户界面的设计中，包括有效的人机交互所必需的实际显示和输入。用户界面部分设计主要包括以下内容。

1）用户分类。

2）描述人及其任务的场景。

3）设计命令层。

4）设计详细的交互。

5）继续做原型。

6）设计 HIC（人机交互）类。

7）根据图形用户界面进行设计。

（3）任务管理部分的设计

当系统中有许多并发行为时，需要依照各个行为的协调和通信关系划分各种任务，以简化并发行为的设计和编码。任务管理主要包括任务的选择和调整，它的工作包括如下内容。

1）识别事件驱动任务。

2）识别时钟驱动任务。

3）识别有限任务和关键任务。

4）识别协调者。

5）评审各个任务。

6）定义各个任务。

（4）数据管理部分的设计

数据管理部分提供了在数据管理系统中存储和检索对象的基本结构，包括对永久性数据的访问和管理。数据管理方法主要有文件管理、关系数据库管理和面向对象数据库管理 3 种。文件管理提供基本的文件处理能力。关系数据库管理系统（Relational Database Management System，RDBMS）建立在关系理论的基础上，它使用若干表格来管理数据。面向对象数据库管理系统（Object-Oriented Database Management System，OODBMS）以两种方法实现：一是 RDBMS 的扩充，二是面向对象程序设计语言（Object-Oriented Programming Language，OOPL）的扩充。数据存储管理部分的设计包括数据存放方法的设计和相应操作的设计。数据存放设计可采用文件存放数据，采用关系数据库存放数据和采用面向对象数据库存放数据。设计相应的操作则是为每个需要存储的对象及其类增加用于存储管理的属性和

操作，在类及对象的定义中加以描述。

2．层次化 OOD 方法

层次化面向对象方法（Hierarchical Object Oriented Design, HOOD）是欧洲航天局（ESA）提出的专门针对实时嵌入式系统开发的 OOD 方法，该方法在 OOD 基础上运用面向问题域中实体的层次化分解思想，吸收了结构化程序设计中自顶向下、逐步求精的精髓，以更自然、更接近人类思维方式的解空间来映射问题域。HOOD 支持软件生命周期中的所有活动，已有越来越多的大型软件工程项目采用 HOOD 方法。

（1）对象

在 HOOD 中，按照对象是否影响系统的动态行为，把对象划分成被动对象和主动对象。被动对象需要消息的触发才能执行，自身不含控制流程。主动对象是作为系统的线程或进程投入运行。主动对象中，至少有一个服务不需要接收消息就能主动执行。外界对象请求它的服务时，控制流并不转移到对象体内。请求对象只是告知主动对象，发出了对主动对象的服务请求。至于请求的服务是否执行、何时执行、以什么方式执行，则完全由主动对象根据内部状态来决定。

（2）层次化

用 HOOD 设计系统时，首先要针对整个系统确认一个对象，该对象叫做根对象。接着，根据整个问题域中出现的不同功能实体，把根对象分解成若干子对象。这种划分在所有的对象内部递归地进行，直至划分出来的子对象足够简单，可以直接用代码实现为止。这样，设计得到的结果是一棵 HOOD 设计树（HOOD Design Tree，HDT），树的根节点代表要实现的系统，叶子节点称作终端对象。

（3）控制结构

HOOD 中实现系统的控制结构的方法是在系统所有的对象之外定义一个主函数，主函数调用它所需要的服务。任何时候，系统中都只有一个活动的控制流。这样的控制结构，或多或少地破坏了系统面向对象的特性，因为主函数并不对应问题域中的对象。HOOD 根据问题中的客观对象的实体特性，引入了被动对象、主动对象、操作控制结构和对象控制结构等概念，使得 HOOD 可以更加灵活、有效地控制系统的动态行为。

（4）设计表示

HOOD 使用两种形式化的工具表示系统设计，一种是 HOOD 图，另一种是对象描述框架（Object Description Skeleton，ODS）。

1）HOOD 图。HOOD 图用一组特定的符号表示系统的设计过程和结果。这些符号可以表示系统中的对象、对象接口（即提供的数据类型、服务）、对象之间的关系（使用、包含）等。

HOOD 图把整个系统的设计过程和层次过程结构用图形直观、清晰地表示出来，便于系统的设计和维护。HOOD 图的缺点是它的不完整性，因此其作用只限于辅助设计、维护人员理解问题以及进行系统体系结构设计。

2）ODS 表示。ODS 是一组结构化的文字域，它提供了所有刻画对象属性的域，这些域又可分成外部可见部分和内部实现部分。将 HOOD 图和 ODS 两种表示结合起来使用，能直观、全面地映射问题域，同时它们提供了对设计的一致性和完整性的检验，可以在较早的阶段检查出存在的问题。

（5）设计过程

HOOD 提供了一个标准化的基本设计步骤：问题定义、一个非正式的解决策略的说明、策略的形式化、解决方案的形式化，按照这个标准化的过程，可以非常规范地实现系统的设计，设计过程如图 3-50 所示。

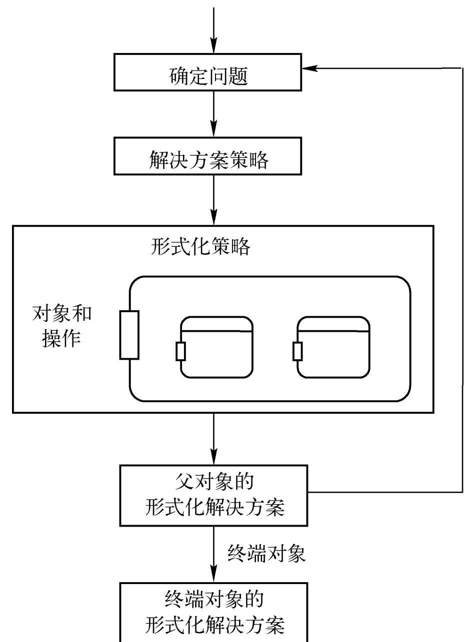


图 3-50　HOOD 的基本设计步骤

对于项目规模庞大、使用寿命长、实时嵌入式、过程控制类的系统设计，HOOD 比一般的 OOD 方法更具优势，不过由于 HOOD 是和 Ada 编码直接相联系的，因而限制了 HOOD 的推广和使用。

3．UML 方法

运用面向对象概念来构造系统模型时，要建立起从概念模型直至可执行体之间明显的对应关系，同时着眼于那些有重大影响的问题以及创建一种对人和机器都适用的建模语言。因此，建模语言是面向对象建模中的一个非常关键的因素。UML 以前，没有一个主导的建模语言。很多建模语言共享一套被广泛接受的概念，只是用不同的语言来表达，有细微的差别，这样一来，面向对象建模领域的厂商不得不支持多个相似的、有细微差别的建模语言。而 UML 的设计目标就是建立一套语义和符号，能够用于各个领域，解决各种程度的结构复杂性问题。UML 出现之后，提供了工具之间、过程之间以及领域之间集成的新机会，降低了培训和重组的费用，更为重要的是，使得开发者能够把精力集中于商业价值上，并为此提供一个范型。

（1）UML 模型

UML 用模型来描述系统的结构和特征。UML 从不同的角度为系统建模，从而形成不同的视图，每个视图表示一个系统描述中的某个特定的抽象。同时，每个视图又由一组图构成，图中含有一个系统在某个方面的具体信息。UML 包括 10 种图和 5 种视图。这 10 种图为用例图、类图、对象图、包图、构件图、活动图、状态图、顺序图、合作图和配置图；5 种视图为用例视图、逻辑视图、并发视图、构件视图和部署视图。

1）用例图。在 UML 中，用例图用来描述用例视图，用例视图由角色、用例、关联和系统边界组成。系统边界用矩形框表示，框内为系统的功能，框外是与本系统相关的其他系统。图 3-51 是某公司销售业务系统的用例视图。

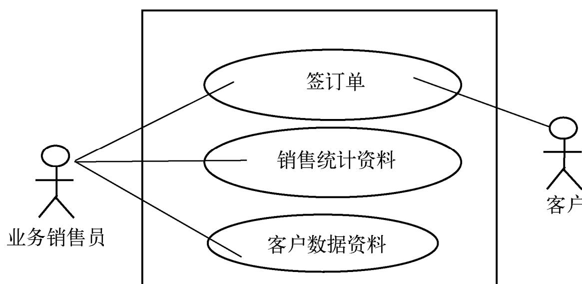


图 3-51　销售业务系统的用例视图

在创建用例图并获取用例模型时，关键问题在于执行者和用例的获取。其中获取执行者时要考虑：谁是系统的主要使用者以及系统需要与其他哪些系统交互等问题；获取用例时要考虑：执行者要求系统提供哪些功能，执行者需要读、产生、删除、修改或存储系统中的信息有哪些类型以及怎样把这些事件表示成用例中的功能等问题。

用例图着重于从系统外部执行者的角度来描述系统需要提供哪些功能，并且指明了这些功能的执行者是谁；用例图在 UML 方法中占有十分重要的地位，它是客户和开发者共同协商而确定的系统基本功能，人们甚至称 UML 是一种用例图驱动的开发方法。但是用例图只是在宏观上给出模型的整体轮廓，用例的实现细节必须以文本的方式进行描述。也就是说，从图形化的模型只能看出系统应当具有哪些功能，每个功能的含义和具体实现步骤必须通过其他的用例图和实现步骤的文本描述来确定。

2）类图和对象图。

● 类图

在面向对象技术中，现实生活中的对象经过抽象，映射为程序中的对象。这样，对象是现实世界中个体或事物的抽象表示，是其属性和相关操作的封装。所谓类是描述对象的“基本原型”，它定义一类对象所能拥有的数据和能完成的操作。而对象是类的实例。类描述同类对象的属性和行为。在 UML 中，类和对象模型分别由类图（Class Diagram）和对象图（Object Diagram）表示。类可表示为一个划分成 3 格的长方形（参见第 2 章图 2–25 示例，下面两个格子可省略），如图 3–52 所示。

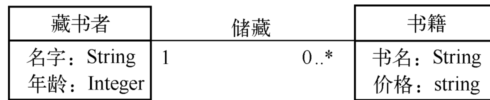


图 3–52　类图

● 对象图

对象图（Object Diagram）表示某个时刻对象和对象之间的关系，代表了系统某时刻的状态，它包含带有值的对象。由于类图中已经包含了对象，所以那些只有对象而没有类的类图就是一个“对象图”，因此，一个对象图可看成一个类图的特殊用例，而实例和类可在对象图中显示。在 UML 中，对象可表示为一个划分成 3 格的长方形（下面两个格子同样可省略），如图 3–53 所示。

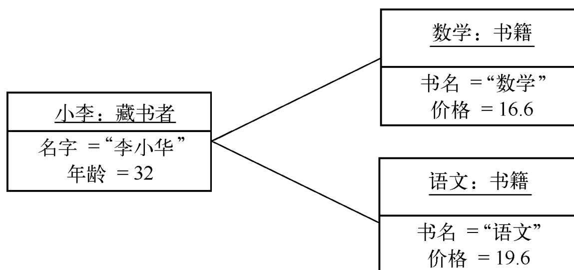


图 3–53　对象图

● 关联

关联（Association）主要用来连接模型元素及链接实例，表示两个类之间存在的某种语

义上的联系，即与该关联连接的类的对象之间的链接。例如，一名教师在一所大学工作，一所大学有许多院系，就可认为教师和大学、大学和院系之间存在某种语义上的联系。在分析或者设计类图模型时，就应该对应于在教师类和大学类、大学类和院系类之间建立关联关系。根据链接的类对象间的具体情况，关联又可分为普通关联、递归关联、多重关联以及或关联，具体内容如下。

a）普通关联：最常见的关联可在两个类之间用一条直线连接，直线上写上关联名。关联可以有方向，表示该关联的使用方向。可以用线旁的小实心三角表示方向，也可以在关联上加上箭头表示方向，在 UML 中也称为航向（Navigability）。只在一个方向上存在航向表示的关联，称为单向关联，在两个方向上都有航向表示的关联，称作双向关联，图 3-52 表示藏书者类和书籍类之间存在双向关联。

在关联的两端可写上一个被称为“重数”（Multiplicity）的数值范围，表示该类有多少个对象可与对方的一个对象连接。重数的符号表示有以下形式。

0 . .1　　表示 0 或 1。

0 . .*　　表示 0 或多，可以简化表示为 *。

1　　　　表示 1 个对象，重数的默认值为 1。

1 . .*　　表示 1 或多。

2 . .4　　表示 2 至 4。

图 3-52 表示一个藏书者可以储藏 0 或多本书籍，而一本书籍只能属于一个藏书者。

b）递归关联：UML 中允许一个类与自身关联，这种关联称为递归关联。例如在一个大学中，一个校长管理多名教师，而校长和教师都是大学的教工，属于教工类。这样就形成了“教工类”到“教工类”的递归关联，图 3-54 表示了这种关联。关联的两端标的是角色名，表示类在这个关联中所扮演的角色。

c）多重关联：多重关联是指两个以上的类之间互相关联。例如，教师指导学生完成论文，可用图 3-55 表示，图中省略了重数和角色名。

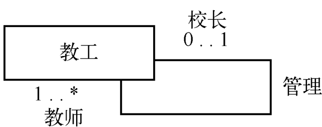


图 3-54　递归关联示例

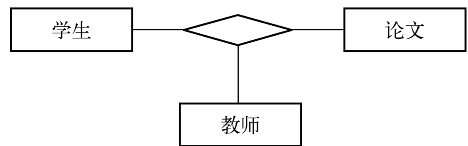


图 3-55　三重关联示例

d）或关联：教师可以购买多台计算机，大学也可以购买多台计算机，但教师和大学不能购买同一台计算机，即一台计算机只能归属于教师或大学中的一个。此时要在两个关联之间加上虚线，上面标以{or}来描述，称为或关联。如图 3-56 所示。

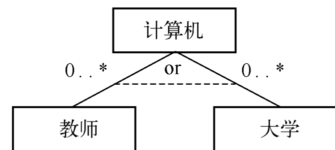


图 3-56　或关联示例

● 泛化

泛化（Generalization）用于描述类与类之间一般与特殊的关系。具有共同特性的元素可抽象成一般类，并通过增加其内涵，进一步抽象成特殊类。例如，生物可分为植物和动物，物体的状态可分为固体、液体和气体。

泛化也可以理解为是两个同类的可泛化元素之间的直接关系。其中一个元素被称为父，

另一个为子。对类而言，父称为超类（Superclass）或父类，子称为子类。父所说明的直接实例带有所有子的共同特点，子所说明的实例是上述实例的子集，不仅有父的特征，还有独有的特征。泛化是一种反对称关系。按照一个方向转变成为父，另一个方向引向子。在父类方向上经过一个或几个泛化关系的元素称为祖先；在子类方向上经过一个或几个泛化关系的元素称为子。不允许出现泛化环，一个类不能既是自己的祖先又是自己的后代。

在最简单的情况下，类（或者其他可泛化元素）有单一的父。在复杂情况下，子有多个父。泛化也称为继承，子继承了父的所有结构、行为和约束，称为多重继承（或者多重泛化）。

a）单一泛化：具有泛化关系的两个类之间，特殊类继承了一般类的所有信息，称为子类，被继承类称为父类，且一个子类最多只能有一个父类。子类可以继承父类的属性、操作和所有的关联关系。在 UML 中，泛化常表示为一端带空心三角形的连线，空心三角形紧挨着父类。如图 3-57 所示，父类是交通工具，车、船和飞机是它的子类；类的继承关系可以是多层的，例如车是交通工具的子类，同时又是卡车、轿车和客车的父类。

没有具体对象的类称为抽象类，可用于描述它的子类的公共属性和操作。图 3-57 中的交通工具就是一个抽象类，一般用一个附加标签值{abstract}来表示。

b）多重泛化：多重泛化即多重继承，指的是子类的子类可以同时继承多个上一级子类，也就是说，子类的子类可以有多个父类。图 3-58 中，“水陆两栖”类就是通过多重继承得到的，子类“陆地动物”和“水生动物”能被“水陆两栖”类同时继承。允许多重继承的父类“动物”被“水陆两用”类继承了两次。

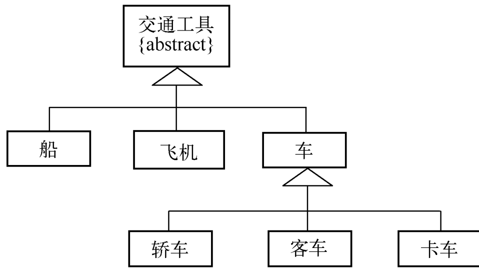


图 3-57　单一泛化示例

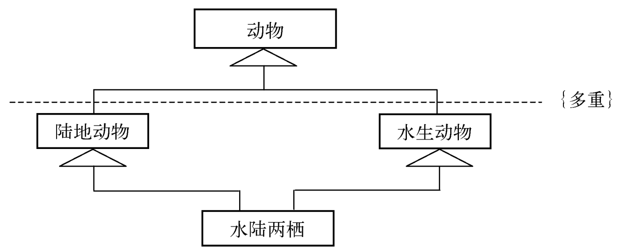


图 3-58　多重继承示例

● 依赖

依赖关系（Dependency）描述的是两个模型元素（类、用例等）之间的语义上的连接关系。假设有两个元素 A、B，如果修改元素 A 的定义可能会引起对另一个元素 B 的定义的修改，则称元素 B 依赖于元素 A。例如，若某个类中使用另一个类的对象作为操作中的参数；一个类存取另一个类中的全局对象；或一个类调用另一个类的类操作等，都表示这两个类之间有依赖关系。依赖关系的图形表示为带箭头的虚线，箭头指向独立的类，箭头旁边可以带一个标签，具体说明依赖的种类。图 3-59 所示的是一个友元依赖（Friend Dependency）关系。依赖关系能使其他类中的操作可以存取该类中的私有或保护属性。

● 聚集

聚集（Aggregation）是一种特殊形式的关联。聚集表示类之间的关系是整体与部分的关系。一辆轿车包含 4 个车轮、一个方向盘、一个发动机和一个底盘，这是聚集的一个例子。在需求分析中，“包含”“组成”“分为……部分”等经常设计成聚集关系。除了一般的聚集

外，还有两种特殊聚集：共享聚集和组合聚集。在 UML 中，共享聚集表示为空心菱形，组合聚合表示为实心菱形。

a）共享聚集：共享聚集（Shared Aggregation）的特征是，它的“部分”对象可以是多个任意“整体”对象的一部分。例如，课题组包含许多个人，但是每个人又可以是另一个课题组的成员，即部分可以参加多个整体，图 3-60 表示课题组类和个人类间的共享聚集。

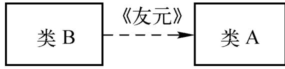


图 3-59　友元依赖关系

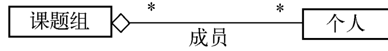


图 3-60　共享聚集

b）组合聚集：在组合（Composition）聚集中，整体拥有各部分，部分与整体共存。如整体不存在了，部分也会随之消失。例如，一个目录之下有 3 个文件，一旦目录消亡，则各部分同时消失。“整体”的重数必须是 0 或 1，而“部分”的重数可以是任意的，如图 3-61 所示。

● 包图

包被作为访问及配置控制机制，以便允许开发人员在互不妨碍的情况下组织大的模型并实现它们。自然，它们将成为开发人员希望的样子。更为特殊的是，要想能够起作用，包必须遵循一定的语意规则。因为它们是作为配置控制单元，所以它们应该包含那些可能发展到一起的元素。包也必须把一并编译的元素分组。如果对一个元素的改变会导致其他元素的重新编译，那这些元素也应该放到相同的包内。

每个模型元素必须包含在一个且仅一个包或别的模型元素里。否则的话，模型的维护、修改和配置控制就成为不可能。拥有模型元素的包控制它的定义。它可以在别的包里被引用和使用，但是对这个包的改变会要求访问授权并对拥有该包的包进行更新。

从另一个角度看，包是类的集合。包图所显示的是类的包以及这些包之间的依赖关系。因此，如果两个包中的任意两个类之间存在依赖关系，则这两个包之间存在依赖关系。包的依赖是不传递的。在大的软件工程项目中，包图是一种重要工具。但是使用包图时要注意，由于依赖会产生耦合，应该尽量将依赖性减少到最低程度。此外，包的概念对测试也是特别有用的。

图 3-62 是一个订单处理子系统的包结构。这个子系统包含了几个包。而包之间的依赖关系通过点画线来表示，这意味着包之间存在着依赖关系，比如订单包依赖于顾客包，订单获取应用包依赖于订单包的有无，等等。更为复杂的包图还有内部包和外部包之分，包之间的关系有依赖和泛化等关系。

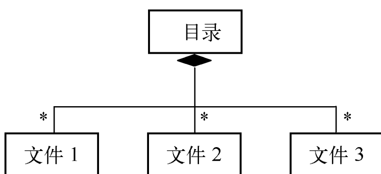


图 3-61　组合聚集

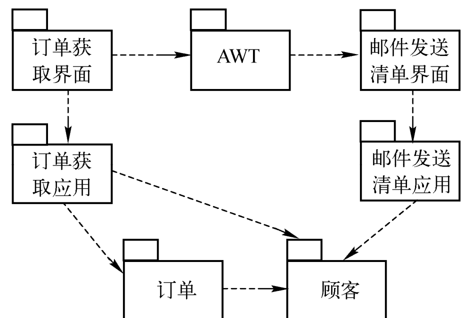


图 3-62　包图示例

3）交互图。

交互图包括顺序图和合作图，用来表示一个用例中对象之间的相互作用的关系。

● 顺序图（Sequence Diagram）

顺序图也叫时序图，顺序图是用来显示对象之间按照事件发生的先后顺序安排的相互作用的图。它主要表明了参与相互作用的对象和对象之间交换消息的顺序。对象之间的相互作用是按照时间次序安排的对象之间的通信的集合。顺序图包括了关于对象的若干事件发生的时间顺序，但是并不包括对象之间的联系。顺序图或者以描述的形式存在（描述所有可能的场景），或者以实例的形式存在（描述实际的场景）。

简言之，顺序图描述了对象之间动态的交互关系，着重体现对象间消息传递的时间顺序。因此，顺序图由一组对象构成。

顺序图具有两个方向，垂直方向和水平方向。垂直方向代表时间；水平方向代表参与相互作用的对象。每个对象都带有一条垂直线，称作对象的生命线，它代表时间轴，时间沿垂直线向下延伸。消息用从一条垂直的对象生命线指向另一个对象的生命线的水平箭头表示。图中还可以根据需要增加其他的说明和注释。

例如，图 3-63 描述了一个打印机工作的顺序图，其中每一列表示参与交互的一个对象，竖轴从顶到底表示时间，消息用箭头表示，箭头上的标识表示消息名并可能包含有参数，操作由垂直方框表示。打印系统首先由用户触发打印功能，计算机对象处理该打印请求，由打印驱动程序根据当前打印机的任务情况调用打印机，如果打印机空闲，则马上打印文件，如果打印机忙，则把文件放入打印队列等待进一步的处理。

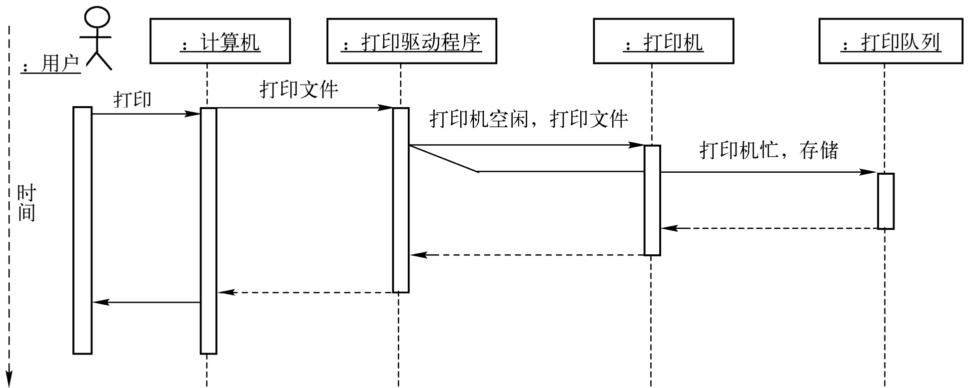


图 3-63　顺序图示例

● 合作图

合作图也叫协作图，与顺序图的作用相同。合作图也是用来描述系统中对象之间的某种动态协作关系。在合作图中，与顺序图类似，对象同样是用一个对象图符来表示，箭头表示消息发送的方向。但是合作图又与顺序图不同，合作图明确地表示了角色之间的关系，侧重于描述各个对象之间存在的消息收发关系（交互关系）。另一方面，合作图也不把时间表示为单独的维，因此消息执行的顺序则由消息的编号来表明。例如，图 3-64 描述了一个打印机工作的合作图。

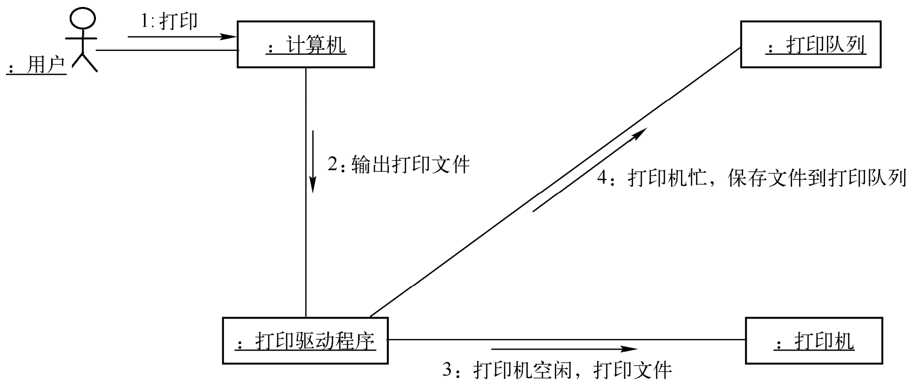


图 3-64　合作图示例

顺序图和合作图都是用来描述系统中对象之间的某种交互关系的，二者通过使用不同的方法表达的是类似的信息。但是它们又有明显的不同之处，合作图的布局方法能更清楚地表示出对象之间静态的连接关系；而顺序图突出执行的时序，能更方便地看出事情发生的次序。

如果要描述在一个用例中的几个对象协同工作的行为，交互图是一种有力的工具。交互图擅长显示对象之间的合作关系，尽管它并不对这些对象的行为进行精确的定义。但是如果想要描述跨越多个用例的单个对象的行为，应当使用状态图；而如果想要描述跨越多个用例或多个线程的多个对象的复杂行为，则需考虑使用活动图。

4）行为图。

行为图包括状态图和活动图两种。

● 状态图

状态图是用对象的多个状态及这些状态之间的转换来描述相应对象的行为。状态图是对类的补充描述，它展示了此类对象所具有的全部可能的状态以及当有某些事件发生时该类对象状态的转移情况。简单说来，状态图表示了一个类的生命历史，表明了引起从一个状态到另一个状态的转变的事件和由一个状态的改变而引发的动作。

状态图有起始状态、中间状态和终止状态。一个状态图可以有一个初始状态，而终止状态可以有多个。图 3-65 是电梯的状态图。图中电梯从底楼开始移动，除底楼外，它能上下移动。如果电梯在某一层上处于空闲状态，当上楼或者下楼事件发生时，电梯就会向上或者向下移动；而当超时事件发生时，电梯就会返回底楼。

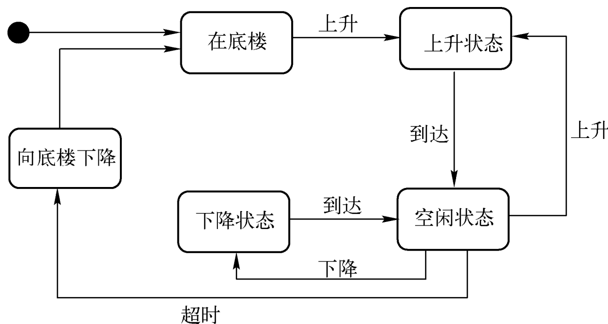


图 3-65　电梯状态图示例

在状态图中，●表示初始状态。⊙表示终止状态。

● 活动图

活动图以活动的形式表述系统，描述系统中各种活动的执行顺序，通常用于描述一个操作的执行状态，主要包括该操作中要进行的各项活动的执行流程。同时，它也常被用来描述多个用例的处理流程，或者用例之间的交互流程。

活动图由一些活动组成，图中同时包括了对这些活动的说明。当一个活动执行完毕之后，控制将沿着控制转移箭头转向下一个活动。活动图中还可以方便地描述控制转移的条件以及并行执行等要求。

活动图与流程图都能用来表示控制流和数据流，从这一点上说，活动图是结构化开发中流程图和数据流程图的面向对象的等同体。

活动图的图符表示如下。

●实心圆表示活动图的起点。

⊙带边框的实心圆表示终点。

▭圆角矩形表示执行的过程或活动。

◇菱形表示判定点。

⟶箭头表示活动之间的转换，各种活动之间的流动次序。

[条件] 箭头上的文字表示继续转换所必须满足的条件，总是使用格式“[条件]”来描述。

▬粗线条表示可能会并行进行的过程的开始和结束。

例如，图 3-66 表示某人找饮料的活动图。首先，他去找饮料，如果找到茶叶，加水到茶壶中，同时可能有并行进行的活动——把茶叶放入杯中；然后把茶壶放到炉上，点燃火炉；当水烧开之后，将水倒入杯中，此处的粗线条表示“把茶叶放入杯中”并且“水烧开”。这两项活动均结束之后才能进行“将水倒入杯中”这项活动；最后喝饮料，活动结束。如果在开始时，没有找到茶叶，则进行判定，如果找到雪碧，就取一听雪碧，喝饮料，活动结束。而在判定时，如果连雪碧都没有找到，则找饮料的活动就直接结束。

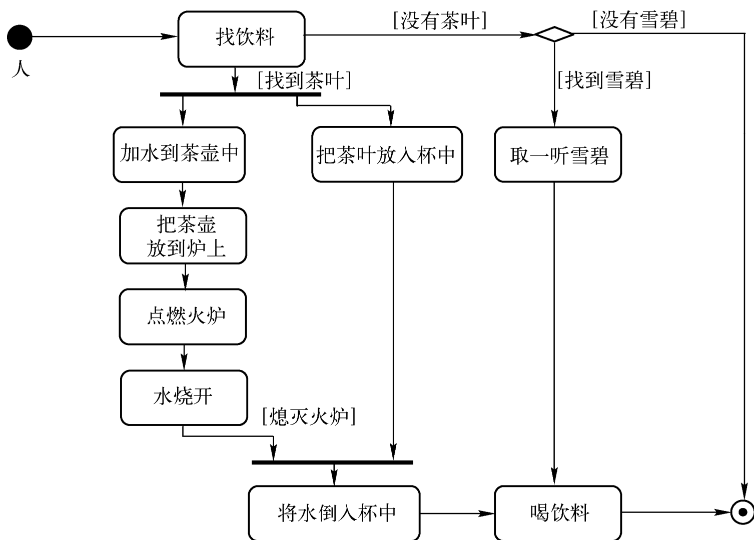


图 3-66　找饮料活动图

在分析用例、理解牵涉多个用例的工作流以及处理多线程应用时一般使用活动图来表示其流程；而在显示对象间合作和显示对象在其生命周期内的运转情况时，一般不使用活动图。

5）实现图。

实现图包括构件图和配置图。

● 构件图

构件图表明了软件构件之间的依赖关系，包括源代码构件、二进制目标码构件、可执行代码构件和文档构件。软件模块可以用一个构件来表示。有些构件存在于编译时，有些存在于连接时，有些存在于执行时，有些在多种场合存在。一个编译时构件只在编译时有意义，运行时构件是可执行的程序。

当修改某个构件时，利用构件图便于人们分析和发现可能对哪些构件产生影响，以便对它们做相应的修改或更新。例如图 3-67 表示的是用面向对象语言编写程序并把相应的图形和结果显示在窗口中的构件图。其中，客户程序依赖于图形库、窗口处理器和主类；而其他构件之间也存在着依赖关系。

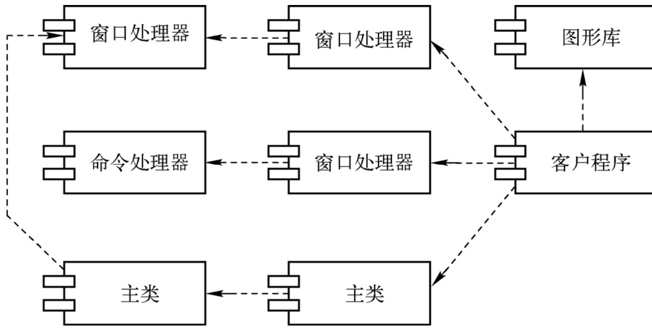


图 3-67　构件图示例

● 配置图

配置图描述系统中硬件和软件的物理配置情况和系统体系结构。配置图含有用通信链相连的节点实例。节点实例包括运行时的实例，如构件实例和对象。构件实例和对象还可以包含对象。配置图有实例形式和描述符形式。实例形式是配置图的常见形式，表明作为系统结构的一部分的具体节点上的具体构件实例的位置。描述符形式说明哪种构件可以存在于哪种节点上，哪些节点可以被连接，类似于类图。

在配置图中，用结点（立方体）表示实际的物理对象，如学生个人计算机和学校服务器等，根据它们之间的连接关系，将相应的结点连接起来，并说明其连接方式。在结点里面，说明分配给该结点上运行的可执行构件或对象，从而说明哪些软件单元被分配在哪些结点上运行。例如图 3-68 表示的是学生的个人计算机与校园网相连时的配置图。其中，立方体分别表示学生的个人计算机、学校的服务器和数据库服务器。可以看出，学生的个人计算机与学校的 02 号服务器相连，又通过 02 号服务器与学校的 VAX 数据库服务器相连。学生个人计算机通过 TCP/IP 相连，而 02 号服务器与 VAX 数据库服务器通过 DecNet 协议相连。

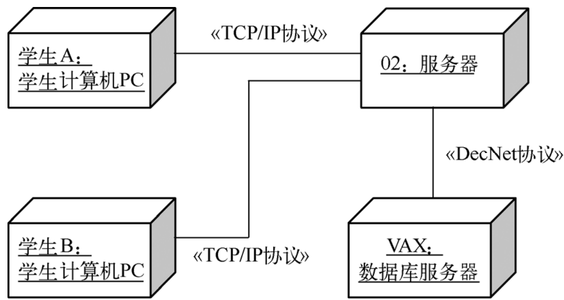


图 3-68　配置图示例

3.4　其他设计方法介绍

面向对象方法使得软件具有良好的体系结构，便于软件构件化、软件复用和良好的扩展性和维护性，抽象程度高，因而具有较高的生产效率。面向对象方法也有其不足，如许多软件系统不完全都能按系统的功能来划分构件，仍然有很多重要的需求和设计决策，比如安全、日志等，它们具有一种“贯穿特性”（Crosscutting Concerns），无论是采用面向对象语言还是过程型语言，都难以用清晰的、模块化的代码实现。最后的结果经常是：实现这些设计决策的代码分布贯穿于整个系统的基本功能代码中，形成了常见的“代码散布”（Code Scattering）和“代码交织”（Code Tangling）现象。代码交织现象是现有软件系统中许多不必要的复杂性的核心。它增加了功能构件之间的依赖性，分散了构件原来假定要做的事情，造成了许多程序设计出错的机会，使得一些功能构件难以复用，源代码难以开发、理解和发展。

为此又出现了一些新技术，以便更好地解决软件开发中的问题。

3.4.1　面向方面程序设计

面向方面程序设计（Aspect-Oriented Programming, AOP）方法最早是由施乐（Xerox）公司在美国加州硅谷 PaloAlto 研究中心（PARC）的首席科学家、加拿大大不列颠哥伦比亚大学教授 Gregor Kicgales 等人在 1997 年的欧洲面向对象编程大会（ECOOP 97）上提出的。

所谓的 Aspect，就是 AOP 提供的一种程序设计单元，它可以将传统程序设计方法学中难于清晰地封装并模块化实现的设计决策，封装实现为独立的模块。Aspect 是 AOP 的核心，它超越了子程序和继承，是 AOP 将贯穿特性局部化和模块化的实现机制。通过将贯穿特性集中到 Aspect 中，AOP 就取得一种单一的结构化行为，该行为在传统程序中分布于整个代码之中——这样就使 Aspect 代码和系统目标都易于理解。在 AOP 中，Aspect 是 AOP 中的一阶实体，AOP 中的 Aspect 就像 OOD 中的类。现有对 Aspect 的认识有错误校验策略、设计模式、同步策略、资源共享、分布关系和性能优化等。

Aspect 的实现与传统开发方法中模块的实现不同。Aspect 之间是一种松耦合的关系，各 Aspect 的开发彼此独立。主代码的开发者甚至可能没有意识到 Aspect 的存在，只是在最后系统组装时刻，才将各 Aspect 代码和主代码编排融合在一起。因此，主代码和 Aspect 之间可以是一种不同于传统“显式调用”关系的“隐式调用”。在软件复杂性日益增加的今天，隐式调用有巨大的优点，因为某一应用的领域专家，不太可能对分布、认证、访问控制、同步、加密、冗余等问题的复杂的实现机制很熟悉，因此就不能保证它们在程序中进行

正确的调用。在当前强调程序演化的情况下，这一点尤其重要，因为开发人员很难正确预见到未来对程序的新需求。

AOP 是一种关注点分离技术，通过运用 Aspect 这种程序设计单元，允许开发者使用结构化的设计和代码，反映其对系统的认识方式。要使设计和代码更加模块化、更具结构化，就要使关注点局部化而不是分散于整个系统中。同时，需使关注点和系统其他部分保持良好定义的接口，从而真正达到“分离关注点，分而治之”的目的。

3.4.2 面向 Agent 的设计方法

Agent 作为人工智能研究重要而先进的分支，引起了科学、工程、技术界的高度重视。斯坦福大学的 Barbara Hayes-Roth 在 IJCAI 1995 的特约报告中提及：智能的计算机主体既是人工智能最初的目标，也是人工智能最终的目标。Agent 的概念作为一个自包含、并行执行的软件过程能够封装一些状态并通过传递消息与其他 Agent 进行通信，其被看作是面向对象设计方法的一个自然发展。Agent 具有以下主要特征。

（1）代理性（Action On Behalf Others）

Agent 具有代表他人的能力，即它们都代表用户工作。这是 Agent 的第一特征。

（2）自制性（Autonomy ）

一个 Agent 是一个独立的计算实体，具有不同程度的自制能力。它能在非事先规划、动态的环境中解决实际问题，在没有用户参与的情况下，独立发现和索取符合用户需要的资源、服务，等等。

（3）主动性（Proactivity）

Agent 能够遵循承诺采取主动，表现面向目标的行为。例如，互联网上的 Agent 可以漫游全网，为用户收集信息，并将信息提交给用户。

（4）反应性（Reactivity）

Agent 能感知环境，并对环境做出适当的反应。

（5）社会性（Social Ability）

Agent 具有一定的社会性，即它们可能同用户、其他 Agent 进行交流。

（6）智能性（Intelligence）

Agent 具有一定程度的智能，包括推理到自学习等一系列的智能行为。

（7）移动性（Mobility）

Agent 具有移动的能力，为完成任务，可以从一个节点移动到另一个节点。比如访问远程资源、转移到环境适合的节点进行工作等。

面向 Agent 的设计（Agent-Oriented Design，AOD）方法与面向对象设计方法的最基本区别在于 Agent 的社会性。面向 Agent 程序设计的主要思想是：根据 Agent 理论所提出的代表 Agent 特性的、精神的和有意识的概念设计了 Agent。根据概念直接设计 Agent 其实是人们想通过意愿来抽象一个复杂系统。由于 Agent 的上述特性，基于 Agent 的系统应是一个集灵活性、智能性、可扩展性、稳定性、组织性等诸多优点于一身的高级系统。

3.4.3 泛型程序设计

泛型程序设计（Generic Programming，GP）是一种范型（Paradigm)，它致力于将各种

类型按照一小组功能性的需求加以抽象，然后以这些需求为条件实现算法。由于算法在其操作的数据类型上定义了一个严格的窄接口，同一个算法便可以应用于各种类型之上。GP 为应用程序开发人员提出了十分美妙的承诺。它使“从‘一种一个’的软件系统向自动制作软件的各不相同的变体发展”这种思路变得十分真实可信。简单地说，GP 以“确定软件开发中自动化的好处”为中心进行软件开发。

3.4.4 面向构件的技术

面向构件技术是指通过组装一系列可复用的软件构件来构造软件系统的软件技术，通过运用构件技术，开发人员可以有效地进行软件复用，减少重复开发，缩短软件的开发时间，降低软件的开发成本。

由于构件隐藏了具体的实现，只用接口提供服务。这样，在不同层次上，构件均可以将底层的多个逻辑组合成高层次上的粒度更大的新构件，甚至直接封装到一个系统，使模块的重用从代码级、对象级、架构级到系统级都可能实现，从而使软件像硬件一样，可进行装配定制以实现要求的功能和系统。因而，面向构件的技术实现了更高层次的抽象。

面向构件技术还包括了另一个重要思想，这就是程序在动态运行时构件的自动装载。

3.4.5 敏捷方法

敏捷方法（Agile Methodologies，AM）也称作轻量级开发方法，对许多人来说，这类方法的吸引之处在于对繁文缛节的“官僚过程”的反叛。它们在无过程和过于烦琐的过程中达到了一种平衡，使得能以不多的步骤过程获取较满意的结果。敏捷型方法强调“适应性”而非“预见性”，敏捷型方法变化的目的就是成为适应变化的过程，甚至能允许改变自身来适应变化。敏捷型方法是“面向人”（People-Oriented）的而非“面向过程”（Process-Oriented）的，敏捷型方法认为没有任何过程能代替开发组的技能，过程起的作用是对开发组的工作提供支持。

3.4.6 Rational 统一过程

“统一过程”是一个软件开发过程，一个通用过程框架，可以应付种类广泛的软件系统、不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。“统一过程”是基于组件的。然而，真正使“统一过程”与众不同的方面有 3 个：它是用例驱动的、以基本架构为中心、迭代式和增量性的。

3.4.7 功能驱动开发模式 FDD

功能驱动开发模式（Feature-Driven Development，FDD）是由 Peter Coad、Jeff de Luca、Eric Lefebvre 共同开发的一套针对中小型软件开发项目的开发模式，它强调的是简化、实用、易于被开发团队接受，适用于需求经常变动的项目。简单地说，FDD 是一个以 Architecture 为中心的、采用短迭代期、日期驱动的开发过程。它首先对整个项目建立起一个整体的模型，然后通过两周一次“设计功能——实现功能”迭代完成项目开发。此处的“功能”是指“用户眼中最小的有用的功能”，它是可理解的、可度量的，并且可以在有限的时

间内（两周）实现。在开发过程中，开发计划的制定、报告的生成、开发进度的跟踪均是以上述“功能”为单位进行的。在 FDD 中，只有良好定义的并且简单的过程才能被很好地执行。另外，由于在 FDD 中采用了短周期的迭代，最小化的功能划分法，因而可以对项目的开发进程进行精确及时地监控。

在 FDD 中，将开发过程划分为如下 5 个阶段。

1）制定整体的模型。

2）根据优先级列出功能的详细列表。

3）依据功能制定计划。

4）依据功能进行设计。

5）实现功能。

在 FDD 中主要存在 3 类人员：开发人员、类的所有者和功能团队。

3.4.8 极端编程

Willy Farrel 和 Mary-Rose Fisher 讨论了开发者引入软件项目的 4 点价值：沟通、简单、反馈和勇气。

极端编程（Extreme Programming，XP）是一套应用这些有价值的东西来创造一个环境的惯例，在这个环境中，开发人员可以快速而正确地开发商业应用。XP 给出了 12 个基本惯例，也称为规则，它们是规划策略（The Planning Game）、成对编程（Pair Programming）、测试（Testing）、重新划分（Refractoring）、简单的设计（Simple Design）、集合体代码所有权（Collective Code Ownership）、持续的集成（Continuous Integration）、现场客户（On-site Customer）、小发行版本（Small Releases）、一周工作 40 小时（40-hour Week）、编码标准（Coding Standards）、系统比喻（System Metaphor）。

XP 的优点在于可以让软件开发人员发挥他们的专长，消除了大多数不必要的重量型过程。

3.5 实用案例

以下是应用本章的知识，结合实际进行软件设计的两个案例。

3.5.1 SafeHome 软件的结构设计

SafeHome 产品软件是 Pressman 教授贯穿全书的一个例子，在本书第 2 章曾涉及相关内容，图 3-69 是 SafeHome 系统的顶层数据流图；图 3-70 是 SafeHome 系统的第 1 层数据流图，其中的每个加工处理都要进一步展开；图 3-71 是其中的“监控传感器”的展开；图 3-72 是细化后的第 3 层，因为该图不是其中某个加工变换的展开，而是整个图的展开，所以其数据流图的图号为 Fig.5.0；图 3-73 是由图 3-72 映射成的相应的软件结构；图 3-74 是第二级分解；图 3-75 是按照本章介绍的方法映射成的“监控传感器”功能的完整结构；该结构中由于有些构件相对比较简单，所以经过精化、合并后得到如图 3-76 所示的最终结构。

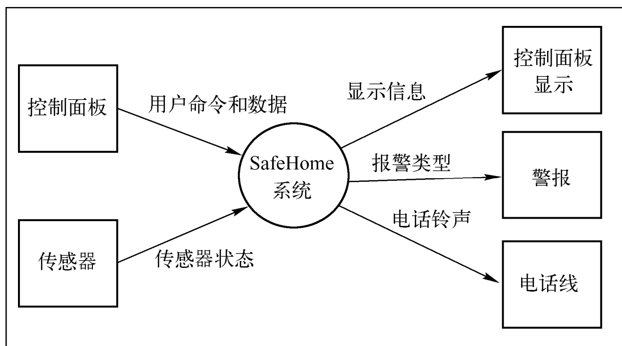


图 3-69　SafeHome 安全住宅系统的顶层数据流图

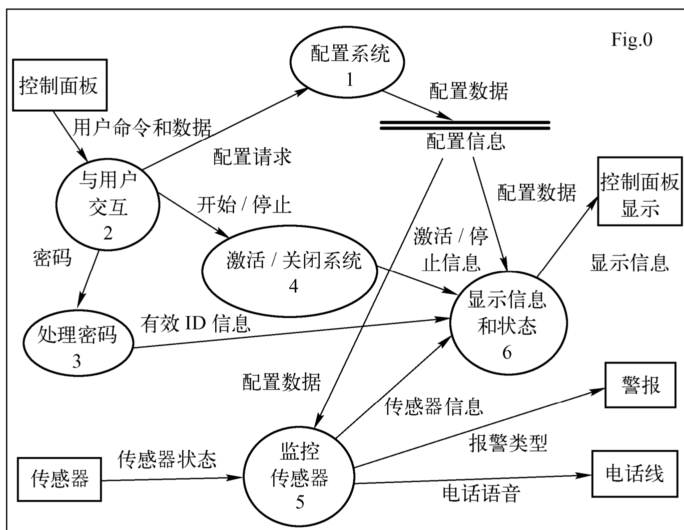


图 3-70　SafeHome 系统的第 1 层数据流图

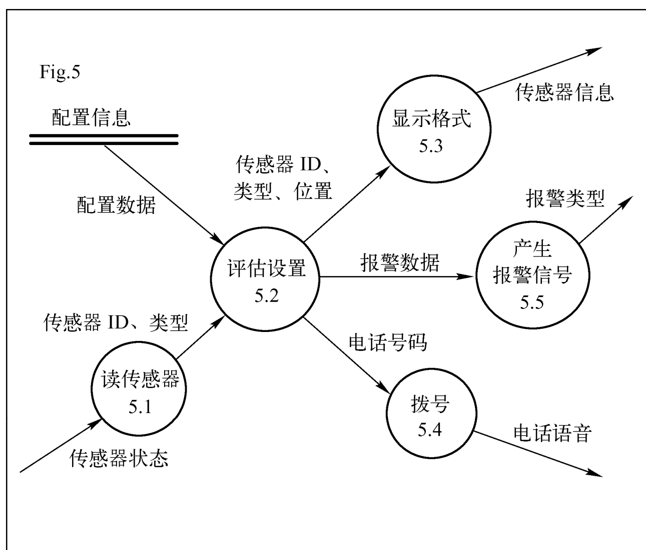


图 3-71　SafeHome 系统第 2 层的一张 DFD——图 3-70 中“监控传感器”的展开

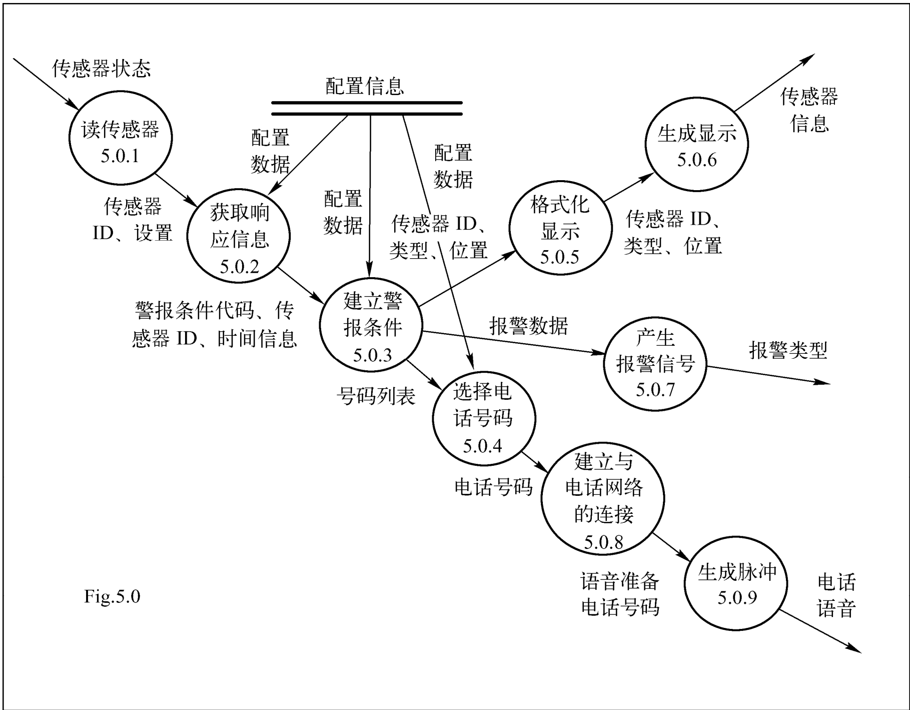


图 3-72　SafeHome 系统第 3 层的一张 DFD——图 3-71 的细化

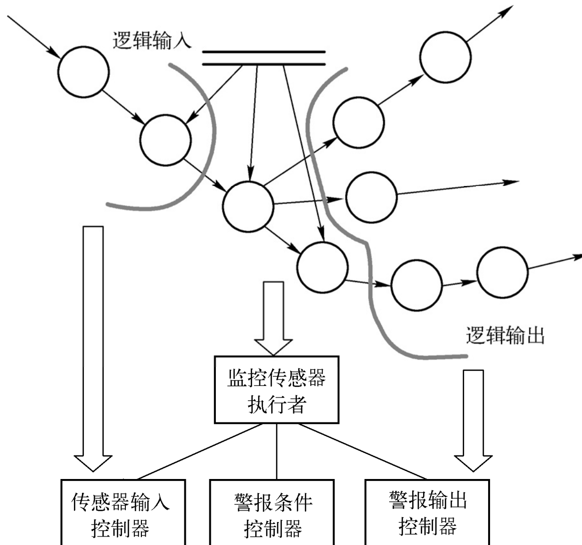


图 3-73　按变换分析方法由图 3-72 映射成软件结构的顶层和第 1 层——第一级分解

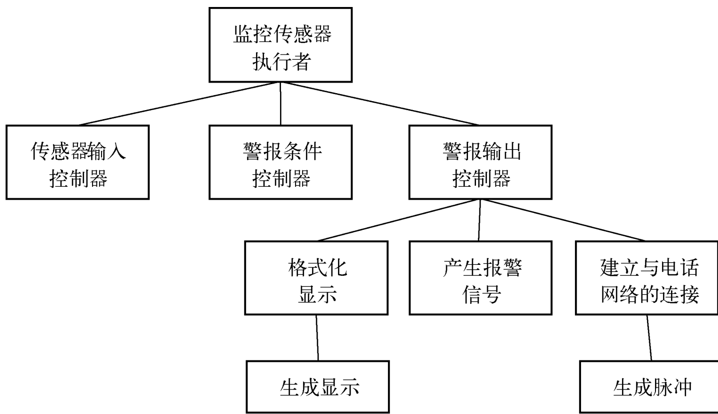


图 3-74　图 3-72 的第二级分解

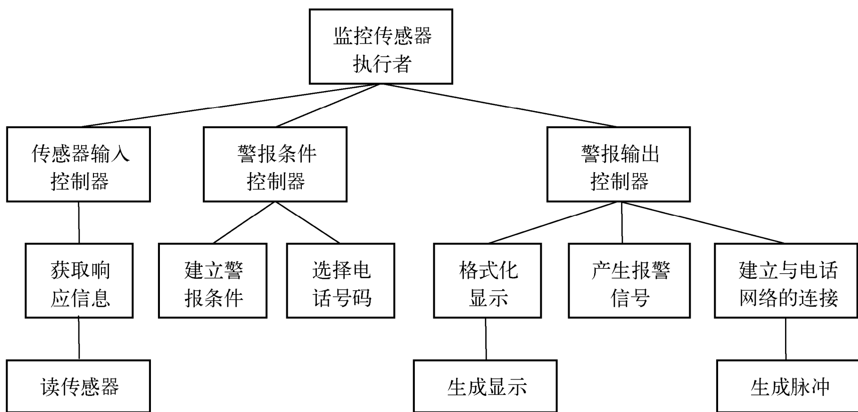


图 3-75　图 3-72 完全映射后的结构

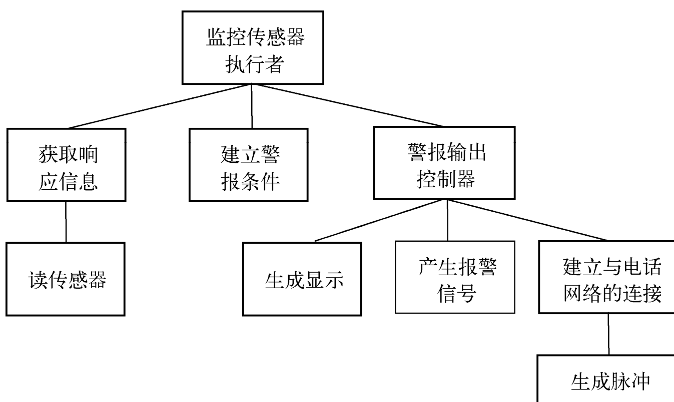


图 3-76　对图 3-75 精化后的最终结构

3.5.2　基于 UML 的网络管理平台的分析与设计

网络管理是监视和控制一个复杂的计算机网络，以确保其尽可能长时间地正常运行，或

当网络出现故障时尽可能地发现并排除故障，使之最大限度地发挥其应有效益的过程。本节利用 UML 对网络管理平台进行建模，在本节的例子中，主要用到以下几种模型图：用例图、顺序图、类图、构件图和配置图。UML 以面向对象的图的方式来描述任何类型的系统，支持从系统需求、系统分析到系统设计的整个建模过程，包括建立系统的静态模型，以及描述系统的动态模型，因此非常适合网络管理平台的复杂建模过程。

1．理解需求

为网络管理平台系统的用户提供一份文本需求说明。这份说明应包括如下内容：图形用户接口（Graphical User Interface，GUI）；网络拓扑图；数据库管理系统（DBMS）；查询设备的标准方法；可制定的菜单系统；事件日志。

2．特定领域分析

首先定义用例图，确定系统的功能需求。通过分析可知，网络管理平台系统的角色有网络管理员，用户图形接口，网络管理应用（配置管理、拓扑管理、网路信息管理），核心系统（通信模块、信息管理）。

分析阶段的另一项工作是特定领域分析，以列出系统中的特定领域类。可以通过阅读规格说明以及寻找系统处理的“概念”来进行特定领域分析，也可以通过用户和领域专家的讨论，以识别出要处理的所有关键类及它们的相互关系。网络管理平台的用例图如图 3-77 所示。

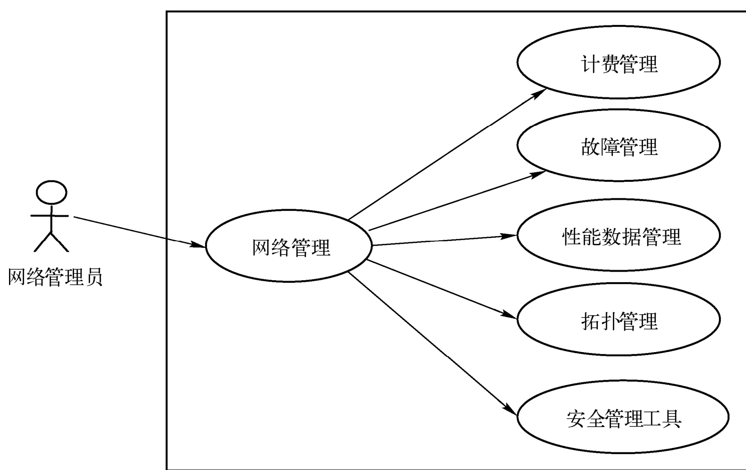


图 3-77　网络管理平台用例图

3．网络管理系统的设计

设计阶段的任务是通过综合考虑所有的技术限制，以扩展和细化分析阶段的模型。设计的目的是指明一种易转化成代码的工作方案，是对分析工作的细化，即进一步细化分析阶段所提取的类（包括其操作和属性）并且增加新类以处理诸如数据库、用户接口、设备信息等技术领域的问题。

设计阶段可以分为两个部分：第一部分是结构设计，属于高层设计，其任务是定义包（子系统）、包和包之间的依赖性和主要通信机制，要求结构尽可能简单和清晰，各部分之间的依赖尽可能地少，并尽可能减少双向依赖关系；第二部分是详细设计，细化包的内容，使编程人员得到所有类的一个足够清晰的描述，同时使用 UML 中的动态模型，描述特定情况

下这些类的实例之间的行为。

（1）结构设计

一个设计良好的系统结构是系统可扩充和可变更的基础。包实际上是一些类的集合。类图中包有助于用户从技术逻辑中分离出应用逻辑（领域类），从而减少它们之间的依赖性。这就是软件结构设计强调的模块间高聚合、低耦合的原则。

在网络管理中，存在以下包（或子系统）。

1）管理员接口包，包括管理员和系统交互的图形界面类。管理员接口包允许管理员访问网络信息数据、加入新数据和修改数据。在网络配置管理中，管理员接口包跟其余对象包合作，调用其余包对象的操作，实施管理数据的收集、修改和删除。

2）网络信息采集配置包，包括网络管理站对网络中的设备信息进行采集并作分析。通过对网络中的设备信息进行参数设置的方法，然后生成对网络进行有效配置的类。网络信息采集配置包通过管理站与别的网络设备之间交换信息来实现网络性能的分析；配置管理向管理员提供了对网络资源的写访问，通过管理协议改变网络设备配置。

3）网络设备代理认证包，网络设备代理能够限制只有被授权的管理站才可以访问它的网络管理信息库（Management Information Bose，MIB）。从管理站发往设备代理的每个报文都包括一个共同体名，这个字符串起着口令作用。只要报文的发送方获得口令，该消息就被认为是可信的。

4）网络设备数据库管理包，网络管理实体可以通过读取 MIB 中的对象值来监视网络资源，它反映了该节点被管资源的状态，并且网络管理实体能通过改变这些值来控制资源。各个包的关系如图 3-78 所示。

（2）详细设计

详细设计的目的是通过创建新的类图、状态图和动态图，描述新的技术类，并扩展和细化分析阶段的网络管理对象类。这些图在分析阶段也曾用过，不过在详细设计阶段，它们是从技术层次上对系统进行更详尽的描述。如分析阶段的用例描述被用来验证它们是否在设计阶段都得到处理，而顺序图用来展示系统中每个用例在技术上如何实现。

1）类图。

● 管理员接口包

此包的任务是使网络管理员对平台的访问更为方便，并且实现网络拓扑图管理。在网络图中对故障使用颜色显示，失效管理工具可以帮助确定故障原因。配置管理工具能够以图示的方法显示出网络的物理和逻辑配置情况。性能管理也可以用不同颜色或不同图形显示出设备当前的性能状况。管理员接口包内部关系如图 3-79 所示。

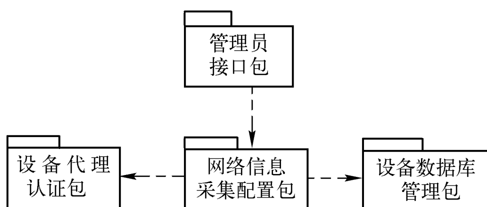


图 3-78　系统包关系

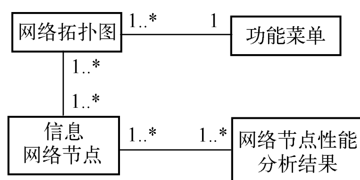


图 3-79　管理员接口包内部关系

● 网络信息采集配置包

此包的任务是使网络管理站产生管理协议数据单元（Protocol Data Unit，PDU），PDU是网络中的设备代理，网络管理员根据需要发出命令，对设备代理进行MIB库的访问和修改，以达到对设备进行控制的目的以及让发出信息收集包对网络中的设备信息进行收集，以达到性能管理的目的。网络信息采集配置包内部关系如图3-80所示。

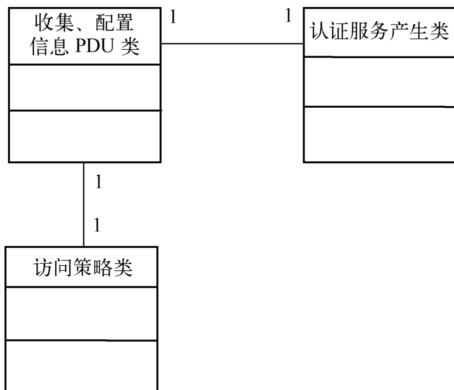


图3-80　网络信息采集配置包内部关系

● 网络设备代理认证包

此包的任务是对网络管理站发来的报文进行来源的认证，即确认报文发出的位置。在此处，简单网管协议SNMP采用共同体名的方法来确认报文是否可信。

● 网络设备数据库管理包

此包的任务是使每个设备代理中对该设备的资源都由一个对象所代表。管理信息库MIB就是这样由一些对象组成的结构化的集合。管理系统中的每个节点都有一个MIB，它反映了该节点中被管资源的状态。网络管理实体可以通过读取MIB中的对象来监视网络资源，也可以通过更改这些值来控制资源。

2）顺序图。顺序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互，网络设备信息收集的顺序图如图3-81所示。

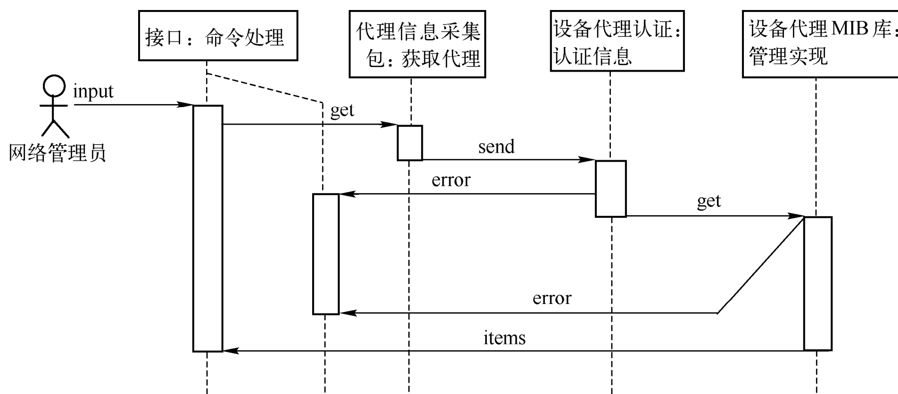


图3-81　网络设备信息收集顺序图

3）构件图。构件图描述了代码部件的物理结构及各部件之间的依赖关系。某一个部件可能是一个资源代码部件、一个二进制部件或一个可执行部件。它包含逻辑类或实现类的有关信息。构件图有助于分析和理解部件之间的相互影响程度。网络管理平台的构件图如图3-82所示。

4）部署图。部署图定义了系统中软硬件的物理体系结构。它可以显示实际的计算机和设备以及它们之间的连接关系，也可显示连接的类型及部件之间的依赖性。在节点内部放置可执行部件和对象，以显示节点与可执行软件单元的对应关系。

在本系统中，网络管理平台采用分层的体系结构。网络管理员可以配置多个独立的客户系统来监视和轮询网络中的不同部分。分层的网络管理结构可以使用客户机/服务器数据库

技术。客户机没有单独的数据库，但可以通过网络访问中央服务器的数据库。鉴于中央服务器系统在层次结构中的重要性，需要对其进行冗余备份，这在任何情况下都是必要的。

层次体系结构的主要特点：不依赖于单一的系统；网络管理任务是分布的；在网络各处进行网络监控；集中进行信息储存。对应的部署图如图 3-83 所示，图中的 NMS 是网络管理系统 Network Management System 的缩写。

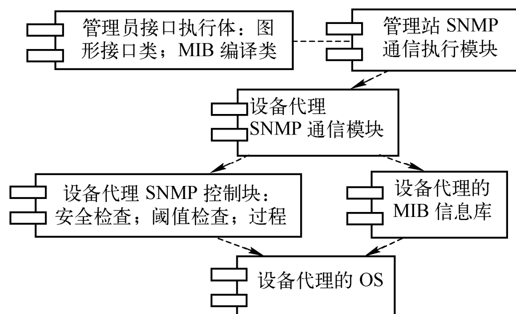


图 3-82　系统构件图

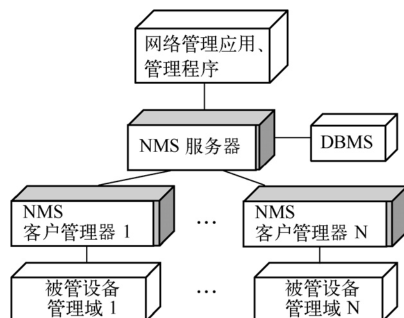


图 3-83　系统部署图

3.6　小结

软件设计是软件开发中的关键阶段，在设计过程中需要软件开发者付出创造性的劳动，它比编码工作要重要得多。从宏观上看软件设计主要有两个阶段，概要设计和详细设计。概要设计是对软件体系结构的设计，详细设计是对软件过程的描述，由此转换成实现系统的代码。大中型以上规模的软件系统既要交付概要设计说明书又要交付详细设计说明书，小型软件系统可合并成一个软件设计说明书。软件设计说明书是重要文档，完成后要组织评审。

软件体系结构是指系统的一个或者多个结构，它包括软件构件、构件的外部可见属性以及它们之间的相互关系。软件体系结构不是可运行的软件，软件体系结构是构造系统的基本框架。大型系统要使用若干体系结构模型，不同的抽象层次上使用不同的模型。常见的体系结构有以数据为中心的体系结构、管道-过滤器体系结构、主程序/子程序体系结构、层次体系结构、客户-服务器体系结构、解释器体系结构、过程控制体系结构等。

划分软件模块时的原则是使各个模块“相对独立，功能单一”。一个好的模块必须具有高度独立性和相对较强的功能。划分模块的好坏，通常用内聚度和耦合度两个特性从不同侧面而加以度量。耦合度是指模块之间相互依赖性大小的度量，耦合度包括内容耦合、公共耦合、外部耦合、控制耦合、标记耦合、数据耦合和非直接耦合 7 种，耦合度越小，模块的相对独立性越大。内聚度是指模块内各成分之间相互依赖性大小的度量，内聚度包括偶发强度、逻辑强度、时间强度、过程强度、通信强度、顺序强度和功能强度 7 种，内聚度越大，模块各成分之间联系越紧密，其功能越强。因此在划分软件模块时，应尽量做到高内聚低耦合。

用户界面是软件产品的重要元素，3 条重要的黄金规则可用于指导用户界面设计：软件应置于用户的控制之下、减少用户的记忆负担以及保持界面一致。

面向数据流的设计（Data-Flow-Oriented Design，DFOD）是根据问题域的数据流（数据对象）定义一组不同的映射，把问题域的数据流（数据对象）转换为问题解的程序结构，这种方法也叫 SD 方法。SD 方法的要点是“自顶而下，逐步求精”。自顶而下的出发点是从问题的总体目标开始，抽象低层的细节，先构造高层的结构，然后再逐层分解和细化，这使

设计者可从宏观到具体进行设计。避免一开始就陷入复杂的细节中，使复杂的设计过程变得简单明了，过程的结果也容易做到正确可靠。独立功能、单出单入口的模块结构减少了模块的相互联系，使模块可作为插件或积木使用，降低了程序的复杂性，提高了可靠性。

SD 方法按数据流的变换规律将数据流图分为变换型和事务型两类，不同类型数据流图映射出的软件结构有所不同，对于混合型的 DFD 应以变换分析方法为主，事物分析方法为辅。

面向对象方法以对象为中心，软件中的任何元素都是对象，复杂的软件对象是由比较简单的对象组合而成。在 OO 中，把所有对象都划分成各种对象类（简称为类，class），每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性，是对象的状态信息。每当建立该对象类的一个新实例时，就按类中对数据的定义为这个新对象生成一组未用的数据，以便描述该对象独特的属性值。按照子类与父类的关系，把若干个对象类组成一个层次结构的系统。在这种层次结构中，下层的派生类具有和上层的基类相同的特性（包括数据和方法），这种现象称为继承。如果在派生类中对某些特性又做了重新描述，则低层的特性将屏蔽高层的同名特性。对象彼此之间仅能通过传递消息互相联系。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，它是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，因为它具有“封装性”，所以不能从外界直接对它的私有数据进行操作，这种灵活的消息传递方式，便于体现并行和分布结构。

面向对象技术也有它的局限性，一是它对软件职责的划分是“垂直”的，在一个标准的对象继承体系中，每一继承类主要承担软件系统中一个特定部分的功能，对象的行为是在编译期间被决定的。二是接口问题，在传统的 OO 环境下，对象开发者没有任何办法确保使用者按照自己的要求来使用接口。

针对面向对象技术的缺点，人们又提出了许多其他方法，有人称为“后面向对象方法”，如面向方面的设计方法、面向 Agent 的设计方法、泛型程序设计、面向构件的设计方法、敏捷方法、Rational 统一过程、FDD 方法、XP 方法等。

随着软件系统规模和复杂程度的提高，人们对软件设计方法提出了越来越高的要求，因而软件设计方法和技术将不断向前发展，更加实用、先进、满足人们不断增长的需求的新方法还将继续出现。

3.7 习题

1．软件设计阶段应主要完成哪些工作？简要给出软件设计的流程。

2．简述软件设计的目标和准则。

3．软件体系结构是不是可运行的软件，它是什么？

4．软件体系结构主要研究哪些内容？

5．给出下列软件体系结构的图示表示。

以数据为中心的体系结构

管道-过滤器体系结构

主程序/子程序体系结构

层次体系结构

6．用图示方式说明软件设计的重要性。

7．图 3-84 左端是软件需求分析模型，右端是软件设计模型，将需求分析的结果作为软件设计的输入，产生的输出就是设计，试用直线或箭头建立它们的对应关系。

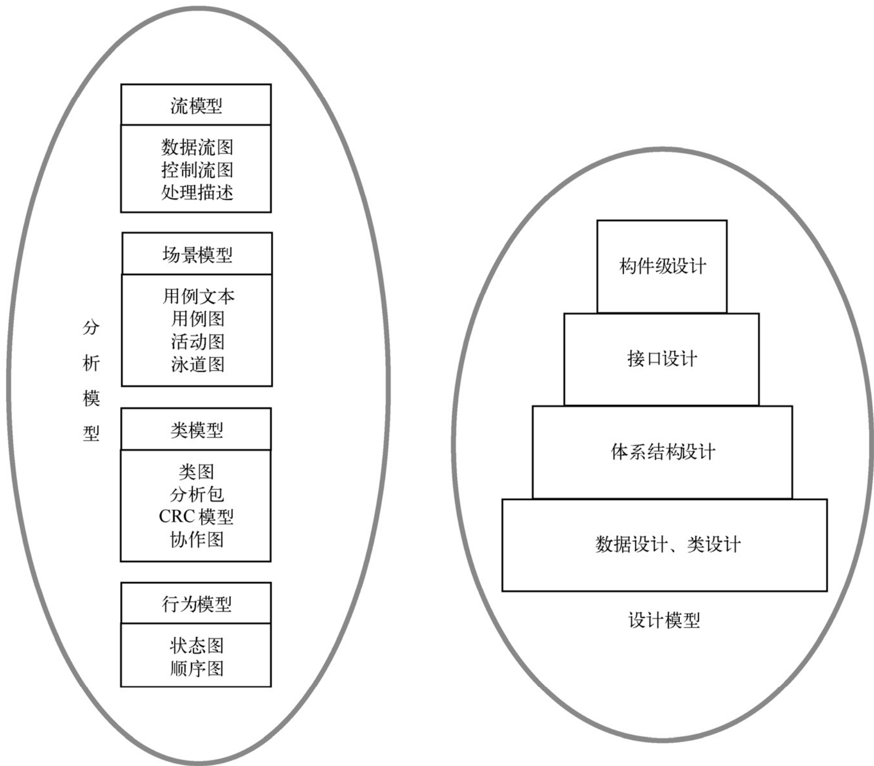


图 3-84　习题 7 的软件需求分析模型与软件设计模型

8．软件模块化是为了降低软件复杂性，以减少设计、编程、测试及维护工作量和成本。

设 $C(X)$ 为问题 X 的复杂度，$E(X)$ 为解决 X 所花费的工作量，

若有 X_1 和 X_2 且 $C(X_1)>C(X_2)$，则 $E(X_1)>E(X_2)$，

因为 $C(X_1+X_2)>C(X_1)+C(X_2)$，所以 $E(X_1+X_2)>E(X_1)+E(X_2)$。

可以得出结论，若将软件无限模块化就可将以后的工作量及成本降低为 0，这种说法显然不对，试给出有说服力的说明。

9．指出图 3-85 所示软件结构的宽度、深度，模块 E 的扇入、扇出数，哪些模块统领了 E，哪些模块从属于 E？

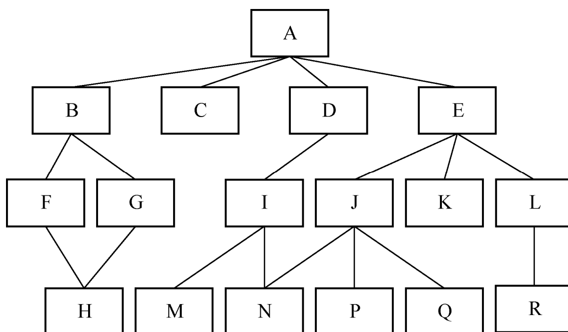


图 3-85　习题 9 的软件结构化图

10．Myers 给出了两种衡量模块独立性的度量，给出它们的名称。每一种又是如何划分的，给出从弱到强或从强到弱的排列，以及追求的目标是什么？

11．按从强到弱的顺序对下列耦合度进行排列。

公共耦合、内容耦合、非直接耦合、外部耦合、数据耦合、控制耦合、标记耦合

12．按从强到弱的顺序对下列聚合度进行排列。

过程强度、偶发强度、时间强度、逻辑强度、顺序强度、功能强度、通信强度

13．简述用户界面设计时应遵循的 3 条黄金规则。

14．简述用户界面设计过程的 4 个框架活动。

15．设计 WebApp 界面时应遵循哪些准则？

16．简述 WebApp 界面设计的工作流程。

17．比较面向数据流的设计方法、面向对象的设计方法以及面向 Agent 的设计方法。

18．在软件结构的设计过程中，若发现一个判定的作用范围不在该判定模块的控制范围之内应如何改进？

19．概要设计和详细设计有什么不同？

20．什么是信息隐蔽原理？

21．什么是软件的冗余设计、防卫设计？

22．给出典型的“变换型”数据流图的结构，给出典型的“事务型”数据流图的结构，它们有什么不同？

23．简要介绍 McCabe 的环形复杂度的用途。

24．McCabe 提出通过限制环形复杂度 $V(G)$数可有效控制模块的规模和复杂程度。请问他提出限制的 $V(G)$数多大？

25．分别计算图 3-86 所示程序图 3-86a 和 3-86b 的环行复杂度。

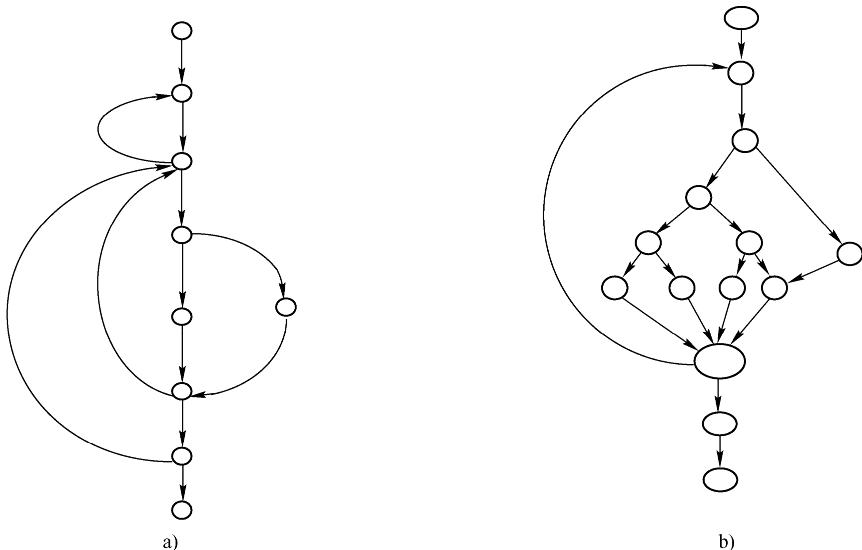


图 3-86　习题 25 的程序图

26．已知 n_1=462，n_2=141，试用 Halstead 的软件科学估算程序中的错误数。

27．把图 3-87 的结构化流程图分别转换成 PAD 图和 N-S 图。

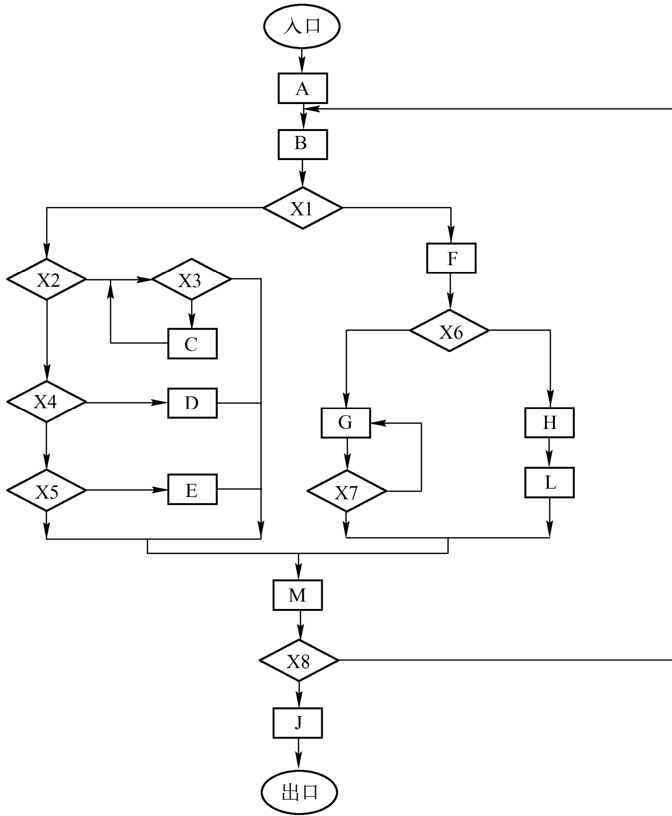


图 3-87　习题 27 的结构化流程图

28．先将图 3-88 描绘的 N-S 图转换为结构化流程图，然后计算它的环行复杂度（要求：计算之前先画出程序图）。

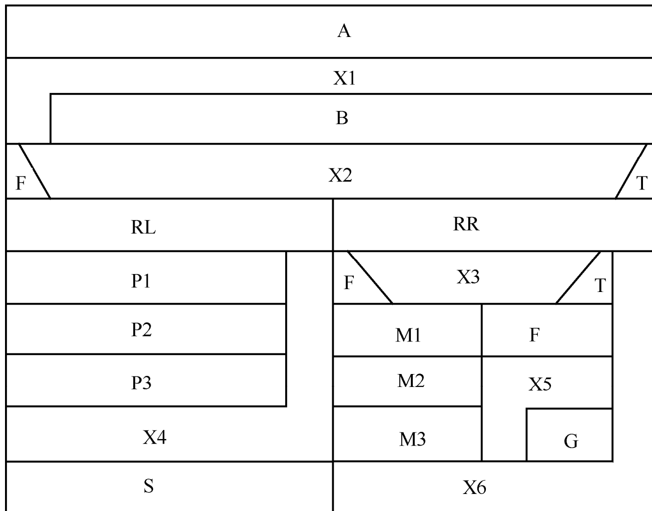


图 3-88　习题 28 的 N-S 图

29．试用 SD 方法将图 3-89 的数据流图映射成软件结构图。

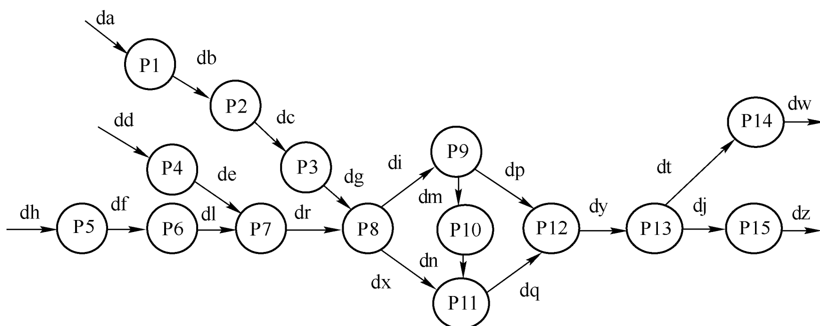


图 3-89　习题 29 的数据流图

30．简述 OOD 的任务。

31．解释图 3-46 所示的软件结构图的转换原理。

32．解释 OOD 基本概念：对象、类、封装、继承、消息、结构与连接、多态性。

33．简述 Coad 与 Yourdon 的 OOD 方法。

34．简述层次化 OOD 方法。

35．求一元二次方程 $ax^2+bx+c=0$ 的两个实根 x_1 和 x_2，并打印结果。用结构化英语描述解决该问题的算法。

36．分别用结构化英语、判定表和判定树描述下列问题。

所有住户 50 平方米以内，每平方米售价 5800 元；超过 50 平方米且在本人住房标准面积以内每平方米售价 7600 元。其中住房标准为：教授 140 平方米，副教授 120 平方米，讲师 90 平方米，标准面积以外每平方米售价 9000 元。

37．某公路收费站对载客车过路费的收费标准如下。

20 座及以下客车：7 元/车；21 座及以上至 50 座客车：15 元/车；51 座及以上客车：25 元/车；特殊车辆不收费。

试分别用结构化英语（或汉语）、判定表和判定树对上述收费问题进行描述。

38．敏捷方法是面向人的还是面向过程的？

39．说出用 UML 建模时常用的图的名称。

40．在软件详细设计过程中，你认为下列哪个数据比较合理？

30～50 行/模块

50～100 行/模块

100～300 行/模块

300～500 行/模块

500～2000 行/模块

41．已知一种详细设计工具 T，易学易用（取“好”）；逻辑表达能力较好，逻辑验证能力较好，很容易转换成程序设计编码（取“很好”）；不适合自动化处理（取“差”）；可修改性较好，结构化实施好，数据表示能力一般，初期测试的简易性好，使用频率较低（取“较差”），试用 3.1 节中介绍的方法计算工具 T 的满意度。

42．现开发一个酒店订房系统，试根据常识，画出酒店订房的 UML 用例图。

43．用 UML 画出酒店订房系统中顾客通过前台订房的顺序图。

44．根据下列对网店货物“订单处理”的叙述，绘出“订单处理”的 UML 活动图。

1）用户下订单。

2）生成送货单，同时等待用户付款。

3）用户付款超时或用户取消订单，则订单取消；否则，收款。

4）供应商接到送货单及货款后生成有效送货单。

5）供应商送货。

6）修改订货单状态。

7）若货物送完则结束，否则继续送货。

45．参看案例 1 中的图 3-73，解释如何将数据流图转换成该软件结构图。