高等学校计算机教育规划教材

软 件 工 程

钟　珞　主编

钟　珞　袁景凌　魏志华　汤练兵　吕　品　编著

胡金柱　主审

清 华 大 学 出 版 社

北　京

内 容 简 介

本书面向普通高等院校本科教学的需要及软件工程技术的发展。主要内容包括：软件工程概述、可行性研究和需求定义、需求分析、系统设计、详细设计、程序编码、软件测试、软件维护、软件项目计划与管理、软件过程能力成熟度模型 CMM、软件的可靠性及软件工具及环境。本书主要特色在于理论、方法与应用相结合，不仅对软件的分析、设计、开发到维护过程进行全面地讲述，而且配有丰富的实例。除了对传统的软件工程方法进行讲述外，还增添了面向对象的软件工程方法、CMM 成熟度模型以及软件工具与环境等较为成熟的内容。

本书概念清楚，内容丰富，每章配有小结和习题，便于教学和学习。本书可供高校本科生学习与后续技术开发使用，也可供广大计算机爱好者阅读。

图书在版编目(CIP)数据

软件工程 / 钟珞主编. —北京：清华大学出版社，2005.10
（高等学校计算机教育规划教材）
ISBN 7-302-11849-3

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

中国版本图书馆 CIP 数据核字（2005）第 109323 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
　　http://www.tup.com.cn　　邮　　编：100084
　　社 总 机：010-62770175　　客户服务：010-62776969
责任编辑：张瑞庆
封面设计：常雪影
版式设计：肖　米
印 刷 者：
装 订 者：肖　米
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：20.75　字数：497 千字
版　　次：2005 年 10 月第 1 版　2005 年 10 月第 1 次印刷
书　　号：ISBN 7-302-11849-3 / TP・7699
印　　数：1~5000
定　　价：25.00 元

编　委　会

序 言 PREFACE

随着信息社会的到来，我国的高等学校计算机教育迎来了大发展时期。在计算机教育不断普及和高等教育逐步走向大众化的同时，高校在校生的人数也随之增加，就业压力随之加大。灵活应用所学的计算机知识解决各自领域的实际问题已经成为当代大学生必须具备的能力。为此，许多高等学校面向不同专业的学生开设了相关的计算机课程。

时代的进步与社会的发展对高等学校计算机教育的质量提出了更高、更新的要求。抓好计算机专业课程以及计算机公共基础课程的教学，是提高计算机教育质量的关键。现在，很多高等学校除计算机系（学院）外，其他系（学院）也纷纷开设了计算机相关课程，在校大学生也必须学习计算机基础课程。为了适应社会的需求，满足计算机教育的发展需要，培养基础宽厚、能力卓越的计算机专业人才和掌握计算机基础知识、基本技能的相关专业的复合型人才迫在眉睫。为此，在进行了大量调查研究的基础上，通过借鉴国内外最新的计算机科学与技术学科和计算机基础课程体系的研究成果，规划了这套适合计算机专业及相关专业人才培养需要的、适用于高等学校学生学习的《高等学校计算机教育规划教材》。

“教育以人为本”，计算机教育也是如此，“以人为本”的指导思想则是将“人”视为教学的主体，强调的是“教育”和“引导”，而不是“灌输”。本着这一初衷，《高等学校计算机教育规划教材》注重体系的完整性、内容的科学性和编写理念的先进性，努力反映计算机科学技术的新技术、新成果、新应用、新趋势；针对不同学生的特点，因材施教、循序渐进、突出重点、分散难点；在写作方法上注重叙述的逻辑性、系统性、适用性、可读性，力求通俗易懂、深入浅出、易于理解、便于学习。

本系列教材突出计算机科学与技术学科的特点，强调理论与实践紧密结合，注重能力和综合素质的培养，并结合实例讲解原理和方法，引导学生学会理论方法的实际运用。

本系列教材在规划时注重教材的立体配套，教学资源丰富。除主教材外，还配有电子课件、习题集与习题解答、实验上机指导等辅助教学资源。有些课程将开设教学网站，提供网上信息交互、文件下载，以方便师生的教与学。

《高等学校计算机教育规划教材》覆盖计算机公共基础课程、计算机应用技术课程和计算机专业课程。既有在多年教学经验和教学改革基础上新编

著的教材，也有部分已经出版教材的更新和修订版本。这套教材由国内三十余所知名高校从事计算机教学和科研工作的一线教师、专家教授编写，并由相关领域的知名专家学者审读全部书稿，多数教材已经经受了教学实践的检验，适用于本科教学，部分教材可用于研究生学习。

我们相信通过高水平、高质量的编写和出版，这套教材不仅能够得到大家的认可和支持，也一定能打造成一套既有时代特色，又特别易教易学的高质量的系列教材，为我国计算机教材建设及计算机教学水平的提高，为计算机教育事业的发展和高素质人才的培养作出我们的贡献。

《高等学校计算机教育规划教材》编委会

2005 年 7 月

前 言 FOREWORD

21 世纪是信息社会高速发展的世纪，软件作为信息技术的核心起着至关重要的作用。面对计算机日益广泛的应用需求，研究如何更快、更好、更经济地开发出相应的软件，是软件开发技术及软件工程师所面临的问题。

计算机技术的飞速发展也促进了软件开发技术的深刻变化。为摆脱软件危机，软件工程学——从 60 年代末期开始迅速发展起来，现在已成为计算机科学技术的一个重要分支。20 世纪 90 年代以来，软件工程不仅从方法论的角度为管理人员和开发人员提供可见的结构和有序的思考，而且大量的成功软件总结出的设计经验，使软件开发人员可以充分利用设计模式、框架和部件等。

本书特点在于理论、方法与应用相结合，不仅对软件的分析、设计、开发到维护过程进行全面地讲述，而且配有丰富的实例，每章还提供典型习题。除了对传统的软件工程方法进行讲述外，还与时俱进地增添了如面向对象的软件工程方法、CMM 成熟度模型及软件工具与环境等较为成熟的内容。软件工程学在计算机专业及信息类专业的课程中占有非常重要的地位，对学生将来从事计算机研究及软件开发工作具有很大的指导作用。

本书的作者长期从事软件工程学的教学和科研工作，积累了丰富的教学经验和实践开发经验，对软件工程的发展及技术有比较全面的认识。

本书由钟珞、袁景凌、魏志华、汤练兵、刘天印、吕品、胡霞、杨舒、胡燕、牛冀平合作编著，是集体智慧的结晶。其中，本书的第 1 章和第 6 章由汤练兵编写，第 2 章和第 3 章由刘天印编写，第 4 章由吕品、钟珞编写，第 5 章由钟珞、魏志华编写，第 7 章由钟珞、袁景凌编写，第 8 章由胡霞编写，第 9 章由杨舒编写，第 10 章由胡燕、钟珞编写，第 11 章由魏志华编写，第 12 章由牛冀平编写。全书由钟珞、袁景凌统稿。

本书的出版得到了武汉理工大学、武汉大学、华中科技大学、国防科技大学、湖北大学、武汉化工学院、黄石理工学院、长江大学等高校教师和清华大学出版社的大力支持，作者在此一并致以衷心的感谢。

目前，国内外有关软件工程技术与设计方面的资料很多，新理论、新技术层出不穷。因时间和水平有限，书中有不少不周到和不准确之处，恳请专家学者提出宝贵意见，以便进一步完善。

作 者

2005 年 6 月

目 录 CONTENTS

第 1 章　软件工程概述 …… 1
1.1　软件工程的产生 …… 1
1.1.1　计算机软件及其特点 …… 1
1.1.2　软件危机 …… 3
1.1.3　软件工程的定义 …… 4
1.2　软件工程的研究对象和基本原理 …… 5
1.2.1　软件工程的研究对象 …… 5
1.2.2　软件工程的基本原理 …… 6
1.3　软件的生存期及常用的开发模型 …… 7
1.3.1　软件的生存期 …… 7
1.3.2　常用的软件开发模型 …… 9
本章小结 …… 12
习题 1 …… 14
第 2 章　可行性研究和需求定义 …… 15
2.1　问题定义 …… 15
2.1.1　问题定义的基本任务 …… 15
2.1.2　问题定义报告 …… 16
2.2　可行性研究 …… 16
2.2.1　可行性研究的内容及过程 …… 17
2.2.2　可行性研究报告 …… 18
2.3　需求定义 …… 19
2.3.1　需求获取的内容 …… 19
2.3.2　需求获取的方法 …… 20
2.3.3　需求规格说明的内容 …… 21
2.3.4　需求规格说明的评审 …… 22
2.3.5　需求规格说明书 …… 23
2.4　典型应用分析 …… 25
本章小结 …… 29
习题 2 …… 29
第 3 章　需求分析 …… 30

3.1 需求分析的目标与原则 30
3.1.1 需求分析的目标 31
3.1.2 需求分析的原则 31
3.2 需求分析的过程及方法 32
3.2.1 需求分析的过程 32
3.2.2 需求分析方法 33
3.3 需求分析的工具 50
3.3.1 SADT 50
3.3.2 PSL/PSA 51
3.4 传统的软件建模 52
3.4.1 软件建模 52
3.4.2 数据模型的建立 53
3.4.3 功能模型、行为模型的建立及数据字典 54
3.5 用例建模 56
3.5.1 用例图 57
3.5.2 参与者及用例的描述 60
3.5.3 用例建模过程 62
3.6 面向对象建模 63
3.6.1 面向对象基础 63
3.6.2 面向对象分析模型 69
3.6.3 对象模型的建立 70
3.6.4 行为模型的建立 72
3.6.5 功能模型的建立 76
3.7 统一建模语言 UML 78
3.7.1 UML 的基本实体 79
3.7.2 UML 的目标及范畴 79
3.7.3 UML 图的使用实例 80
3.8 典型应用分析 84
3.8.1 结构化分析示例 84
3.8.2 面向对象分析示例 88
3.8.3 面向问题域的分析示例 91
本章小结 94
习题 3 95
第 4 章 系统设计 96
4.1 系统设计的任务和过程 96
4.1.1 系统设计的任务 96
4.1.2 系统设计的过程 96
4.2 系统设计的基本原则 97
4.2.1 软件设计 97

4.2.2 模块设计 100
4.2.3 结构设计 101
4.3 面向数据流图的设计方法 101
4.3.1 典型的系统结构图 101
4.3.2 变换分析 103
4.3.3 事务分析 105
4.3.4 软件模块结构的改进 106
4.4 面向对象的设计方法 106
4.4.1 面向对象的基本概念和特征 106
4.4.2 面向对象的技术要点 106
4.4.3 面向对象分析模型 107
4.5 面向对象软件设计模型 107
4.5.1 设计模式描述 107
4.5.2 设计模式的分类 107
4.6 模型-视图-控制器框架 108
4.6.1 MVC 模式 108
4.6.2 MVC 中的模型类、视图类和控制类 109
4.6.3 MVC 的实现 110
4.7 系统设计说明书 111
4.8 典型应用分析 112
4.8.1 类设计的目标 112
4.8.2 类设计的方针 112
4.8.3 通过复用设计类 113
4.8.4 计数器类设计的实例 113
本章小结 114
习题 4 115
第 5 章 详细设计 116
5.1 详细设计的任务及过程 116
5.1.1 详细设计的任务 116
5.1.2 详细设计的过程 116
5.1.3 详细设计的原则 117
5.1.4 详细设计工具 117
5.2 结构化设计方法 122
5.2.1 基于数据流的结构化设计方法 122
5.2.2 面向数据结构的结构化设计方法 123
5.3 Jackson 程序设计方法 123
5.3.1 Jackson 方法的基本思想 123
5.3.2 Jackson 方法的设计技术及实例 124
5.4 Warnier 程序设计方法 131

5.4.1 Warnier 方法的基本思想 131
5.4.2 Warnier 方法的设计技术及实例 131
5.5 基于组件的设计方法 135
5.5.1 基于组件的基本思想 135
5.5.2 基于组件的设计技术及实例 136
5.5.3 应用 142
5.6 详细设计说明书 142
5.6.1 引言 142
5.6.2 总体设计概述 143
5.6.3 程序描述 143
本章小结 143
习题 5 144
第 6 章 程序编码 145
6.1 程序设计语言 145
6.1.1 程序设计语言的发展及分类 145
6.1.2 程序设计语言的选择 147
6.2 程序设计风格 148
6.2.1 源程序文档化 148
6.2.2 数据说明 149
6.2.3 表达式和语句 149
6.2.4 输入输出 150
6.3 程序设计方法 150
6.3.1 结构化程序设计方法 150
6.3.2 面向对象的程序设计方法 152
6.4 程序的复杂性及度量 154
6.4.1 程序的复杂性 154
6.4.2 McCabe 度量法 154
6.4.3 Halstead 方法 155
本章小结 156
习题 6 157
第 7 章 软件测试 158
7.1 软件测试的基本方法 158
7.1.1 静态测试和动态测试 158
7.1.2 白盒测试和黑盒测试 159
7.1.3 ALAC 测试 160
7.2 软件测试过程 160
7.2.1 单元测试 160
7.2.2 集成测试 160
7.2.3 确认测试 161

7.2.4 系统测试 …… 162
7.3 软件测试 …… 163
7.3.1 软件测试角色 …… 163
7.3.2 软件测试环境 …… 164
7.3.3 软件测试的需求规格说明 …… 164
7.3.4 软件测试设计说明 …… 170
7.3.5 测试评价 …… 171
7.4 面向对象软件测试 …… 174
7.4.1 面向对象测试模型 …… 174
7.4.2 面向对象分析的测试 …… 174
7.4.3 面向对象设计的测试 …… 175
7.4.4 面向对象编程的测试 …… 175
7.4.5 面向对象的单元测试 …… 176
7.4.6 面向对象的集成测试 …… 176
7.4.7 面向对象的系统测试 …… 176
7.5 典型应用分析 …… 177
本章小结 …… 183
习题 7 …… 184
第 8 章 软件维护 …… 185
8.1 软件维护的基本概念 …… 185
8.1.1 软件维护的定义 …… 186
8.1.2 软件维护的分类 …… 186
8.2 软件维护的特点及过程 …… 187
8.2.1 影响软件维护的因素 …… 187
8.2.2 软件维护的标准化 …… 188
8.2.3 软件维护的特点 …… 189
8.2.4 软件维护过程 …… 190
8.3 软件的可维护性 …… 192
8.3.1 软件可维护性的定义 …… 192
8.3.2 软件可维护性的度量及评估 …… 192
8.3.3 提高软件可维护性的方法 …… 195
本章小结 …… 196
习题 8 …… 196
第 9 章 软件项目计划与管理 …… 197
9.1 成本估计 …… 199
9.1.1 项目成本估计的基本要素与模式 …… 200
9.1.2 软件开发成本估算的常用方法 …… 203
9.1.3 软件成本估算的经验模型 …… 208
9.2 效益分析 …… 213

9.2.1 几种效益度量方法 …… 214
9.2.2 效益分析方法 …… 215
9.3 项目组织与计划 …… 217
9.3.1 项目计划的制定 …… 218
9.3.2 项目组人员配备规则 …… 220
9.3.3 人员组织与管理 …… 222
9.4 进度计划 …… 225
9.4.1 制定开发进度计划 …… 225
9.4.2 甘特图与时间管理 …… 226
9.4.3 工程网络与关键路径 …… 227
9.4.4 项目进度跟踪与控制 …… 229
9.5 风险管理 …… 230
9.5.1 风险识别与分类 …… 231
9.5.2 风险评估与分析 …… 233
9.5.3 风险策划与管理 …… 235
9.5.4 风险规避与监控 …… 237
9.6 软件质量 …… 238
9.6.1 软件质量特性与度量 …… 240
9.6.2 软件质量体系与控制 …… 245
9.6.3 软件质量保证与评审 …… 247
9.6.4 软件配置项及其管理 …… 250
9.7 软件工程标准 …… 253
9.7.1 软件工程标准化及其意义 …… 253
9.7.2 软件工程标准的类型与层次 …… 254
9.7.3 软件质量标准与认证 …… 258
9.7.4 软件文档标准化 …… 260
本章小结 …… 263
习题 9 …… 264
第 10 章 软件过程能力成熟度模型 CMM …… 265
10.1 软件过程与软件过程成熟度 …… 265
10.1.1 软件过程 …… 265
10.1.2 软件过程成熟度 …… 266
10.1.3 软件过程改进框架 …… 266
10.2 CMM 简介 …… 267
10.2.1 CMM 的发展过程 …… 268
10.2.2 软件过程成熟度的基本概念 …… 268
10.2.3 全面质量管理和 CMM …… 269
10.2.4 基于模型改进的优点与风险 …… 273
10.3 软件过程成熟度框架 …… 275

10.3.1 成熟度的 5 个级别 275
10.3.2 软件过程的可视性 277
10.3.3 跳越成熟度级别 278
10.4 能力成熟度模型的结构 279
10.4.1 成熟度级别的内部结构 279
10.4.2 关键过程域 280
10.4.3 关键实践 280
10.4.4 共同特性 281
10.5 CMM 的应用 281
10.5.1 基于 CMM 的估价方法 281
10.5.2 软件过程评估及软件能力评价 283
10.5.3 软件过程改进 285
10.5.4 使用 CMM 287
10.5.5 CMM 实施工具 288
本章小结 290
习题 10 291
第 11 章 软件的可靠性 292
11.1 软件可靠性基本概念 292
11.1.1 软件可靠性定义 292
11.1.2 软件可靠性的主要指标 293
11.1.3 软件生存期与软件寿命 293
11.2 软件可靠性评估 294
11.2.1 软件可靠性模型 294
11.2.2 估算软件中错误的方法 296
11.3 软件可靠性技术 298
11.3.1 算法模型化 298
11.3.2 软件容错技术 299
本章小结 301
习题 11 302
第 12 章 软件工具及环境 303
12.1 软件工具 304
12.1.1 软件工具的作用与功能 304
12.1.2 软件工具的分类 305
12.2 软件开发环境 305
12.2.1 软件开发环境的分类 306
12.2.2 软件开发环境的特点 306
12.2.3 软件开发环境的组成与结构 307
12.3 计算机辅助软件工程（CASE） 307
12.3.1 CASE 的概念及现状 308

12.3.2 CASE 技术的功能及组成 …… 308
12.3.3 CASE 工具分类及特点 …… 309
12.3.4 CASE 与软件工程的关系 …… 310
本章小结 …… 311
习题 12 …… 311
参考文献 …… 312

第1章 软件工程概述[1, 2, 27, 29]

1.1 软件工程的产生

1.1.1 计算机软件及其特点

世界上第一个编写软件的人是阿达（Augusta Ada Lovelace），在19世纪60年代尝试为巴贝奇（Charles Babbage）的机械式计算机编写软件。尽管限于当时的制造条件，巴贝奇最终也没有造成理想中的计算机，但阿达和巴贝奇对后来计算机技术的诞生和发展产生了深远的影响，他们的名字被永远载入了计算机发展的史册。

在20世纪中叶，软件伴随着第一台电子计算机的问世诞生了。以编写软件为职业的人也开始出现，多是经过训练的数学家和电子工程师。20世纪60年代，美国大学里开始出现计算机专业，教学生如何编写软件。软件产业从零开始起步，在短短的五十多年的时间里迅速发展成为推动人类社会发展的龙头产业，并造就了一批百万、亿万富翁。随着信息产业的发展，软件对人类社会越来越重要。

软件对于人类而言是一个全新的东西，其发展历史不过五六十年。人们对软件的认识经历了一个由浅到深的过程。

随着计算机硬件性能的极大提高和计算机体系结构的不断变化，计算机软件系统更加成熟和更为复杂，从而促使计算机软件的角色发生了巨大的变化，其发展历史大致可以分为如图1-1所示的4个阶段。

第一阶段是20世纪50年代初期至20世纪60年代初期的十余年，是计算机系统开发的初期阶段。当时的软件几乎都是为每个具体应用而专门编写的，编写者和使用者往往是同一个或同一组人。这些个体化的软件设计环境，使软件设计成为在人们头脑中进行的一个隐含过程，最后除了程序清单外，没有其他文档资料保存下来。

注：章题目右上角数字为参考文献对应条目。

实际上，初期开发的计算机系统采用批处理技术，提高了计算机的使用效率，但不利于程序设计、调试和修改。在这个阶段，人们认为计算机的主要用途是快速计算，软件编程简单，不存在什么系统化的方法，开发没有任何管理，程序的质量完全依赖于程序员个人的技巧。

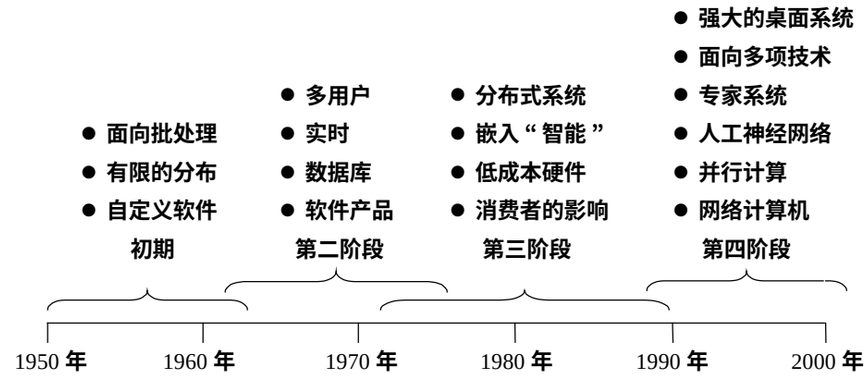


图 1-1　计算机软件发展的 4 个阶段

第二阶段跨越了从 60 年代中期到 70 年代末期的十余年，多用户系统引入了人机交互的新概念，实时系统能够从多个源收集、分析和转换数据，从而使得进程的控制和输出的产生以毫秒而不是分钟来进行，在线存储的发展产生了第一代数据库管理系统。

在这个时期，出现了软件产品和“软件作坊”的概念，设计人员开发软件不再像早期阶段那样只为自己的研究工作需要，而是为了用户更好地使用计算机，但“软件作坊”仍然沿用早期形成的个体式的软件开发方法。随着计算机应用的日益普及，软件需求量急剧膨胀。在程序运行时发现的错误必须设法更正；用户有了新需求时，必须相应地修改或添加程序；硬件或操作系统更新时，又可能要修改程序以适应新的环境。这样，软件的维护工作以惊人的比例耗费资源，更严重的是，程序设计的个体化和作坊化特性使软件最终成为不可维护的，从而出现了早期的软件危机。人们随之也就开始寻求采用软件工程的方法来解决软件危机问题。

第三阶段是 20 世纪 70 年代中期至 20 世纪 80 年代末期，分布式系统极大地提高了计算机系统的复杂性，网络的发展对软件开发提出了更高的要求，特别是微处理器的出现和广泛应用，孕育了一系列的智能产品。硬件的发展速度已经超过了人们对软件的需求速度，因此使得硬件价格下降，软件的价格急剧上升，导致了软件危机的加剧，致使更多的科学家着手研究软件工程学的科学理论、方法和时限等一系列问题。软件开发技术的度量问题受到重视，最著名的有软件工作量估计 COCOMO 模型、软件过程改进模型 CMM 等。

第四阶段是从 20 世纪 80 年代末期开始的。这个阶段是强大的桌面系统和计算机网络迅速发展的时期，计算机体系结构由中央主机控制方式变为客户机/服务器方式，专家系统和人工智能软件终于走出实验室进入了实际应用，虚拟现实和多媒体系统改变了与最终用户的通信方式，出现了并行计算和网络计算的研究，面向对象技术在许多领域迅速取代了传统软件开发方法。

在软件的发展过程中，软件的需求成为软件发展的动力，软件的开发从自给自足模式发展为在市场中流通以满足广大用户的需要。软件工作的考虑范围也发生了很大变化，人们不再只顾及程序的编写，而是涉及软件的整个生命周期。

软件从个性化的程序变为工程化的产品，人们对软件的看法发生了根本性的变化，现在，软件的正确的定义应该是：软件（software）是计算机系统中与硬件（hardware）相互依存的另一部分，包括程序（program）、相关数据（data）及其说明文档（document）。

其中程序是按照事先设计的功能和性能要求执行的指令序列；数据是程序能正常操纵信息的数据结构；文档是与程序开发维护和使用有关的各种图文资料。

软件同传统的工业产品相比，有以下独特的特性。

（1）软件是一种逻辑产品，与物质产品有很大的区别。软件产品是看不见摸不着的，因而具有无形性，是脑力劳动的结晶，是以程序和文档的形式出现的，保存在计算机存储器和光盘介质上，通过计算机的执行才能体现其功能和作用。

（2）软件产品的生产主要是研制，软件产品的成本主要体现在软件的开发和研制上。软件一旦研制开发成功后，通过复制就产生了大量软件产品。

（3）软件在使用过程中，没有磨损、老化的问题。软件在使用过程中不会因为磨损而老化，但会为了适应硬件、环境以及需求的变化而进行修改，而这些修改又不可避免地引入错误，导致软件失效率升高，从而使得软件退化。当修改的成本变得难以接受时，软件就被抛弃。

（4）软件对硬件和环境有着不同程度的依赖性。这导致了软件移植的问题。

（5）软件的开发主要是进行脑力劳动，至今尚未完全摆脱手工作坊式的开发方式，生产效率低，且大部分产品是“定做”的。

（6）软件是复杂的，而且以后会更加复杂。软件是人类有史以来生产的复杂度最高的工业产品。软件涉及人类社会的各行各业、方方面面，软件开发常常涉及其他领域的专门知识，这对软件工程师提出了很高的要求。

（7）软件的成本相当昂贵。软件开发需要投入大量、高强度的脑力劳动，成本非常高，风险也大。现在软件的开销已大大超过了硬件的开销。

（8）软件工作牵涉很多社会因素。许多软件的开发和运行涉及机构、体制和管理方式等问题，还会涉及人们的观念和心理等因素。这些人的因素，常常成为软件开发的困难所在，直接影响到项目的成败。

1.1.2　软件危机

在 19 世纪 60 年代，很多的软件最后都得到了一个悲惨的结局。很多的软件项目开发时间大大超出了规划的时间。一些项目导致了财产的流失，甚至某些软件导致了人员伤亡。同时软件开发人员也发现软件开发的难度越来越大。

IBM/360 被认为是一个典型的案例。IBM/360 的开发总投资 5 亿美元，达到美国研究原子弹的曼哈顿计划投资 20 亿美元的 1/4。在研制期间，布鲁克斯（Frederick Phillips Brooks, Jr.）主持了这个项目，率领着 2000 名程序员夜以继日地工作，单单 OS/360 操作系统的开发就用了 5000 个人年。在当时，IBM/360 以其通用化、系列化和标准化的特点，

对全世界计算机产业的发展产生了深远的影响，以致被认为是划时代的杰作，至今仍然被使用在 IBM360 系列主机中。但据统计，这个操作系统每次发行的新版本都是从前一版本中找出上千个程序错误而修正的结果。如今经历了数十年，这个极度复杂的软件项目甚至产生了一套不包括在原始设计方案之中的工作系统。后来，布鲁克斯在他的作品《人月神话》（The Mythical Man-Month）中对这个项目的开发研制过程进行分析及总结，承认由于管理方面的原因，项目在某些方面来说是失败的，甚至犯了一个价值数百万美元的错误。

软件的错误可能导致巨大的财产损失，2002 年 12 月欧洲阿里亚娜火箭的爆炸就是一个最为惨痛的教训。而且由于计算机软件被广泛应用于包括医院等与生命息息相关的行业，这也使得软件的错误导致人员伤亡成为了可能。在工业上，某些嵌入式系统导致机器的不正常运转，从而使工作人员陷入险境。

20 世纪 60 年代末至 20 世纪 70 年代初，软件危机一词在计算机界广为流传。事实上，软件危机几乎从计算机诞生的那一天起就出现了，只不过到了 1968 年，北大西洋公约组织的计算机科学家在联邦德国召开的国际学术会议上第一次提出了“软件危机”（software crisis）这个名词。

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。这类问题绝不仅仅是“不能正常运行的软件”才具有的，实际上几乎所有软件都不同程度地存在这类问题。概括来说，软件危机包含两方面问题：其一是如何开发软件，以满足不断增长、日趋复杂的需求；其二是如何维护数量不断膨胀的软件产品。

具体地说，软件危机主要有下列表现。

（1）对软件开发成本和进度的估计常常不准确。开发成本超出预算，实际进度比预定计划一再拖延的现象并不罕见。

（2）用户对“已完成”系统不满意的现象经常发生。

（3）软件产品的质量往往靠不住。“缺陷”一大堆，“补丁”一个接一个。

（4）软件的可维护程度非常之低。

（5）软件通常没有适当的文档资料。

（6）软件的成本不断提高。

（7）软件开发生产率的提高赶不上硬件的发展和人们需求的增长。

之所以出现软件危机，其主要原因一方面是与软件本身的特点有关；另一方面是由软件开发和维护的方法不正确有关。

软件的特点前面已经有一个简单介绍。软件开发和维护的不正确方法主要表现为忽视软件开发前期的需求分析；开发过程没有统一的、规范的方法论的指导，文档资料不齐全，忽视人与人的交流；忽视测试阶段的工作，提交给用户的软件质量差；轻视软件的维护。这些大多数都是软件开发过程管理上的原因。

1.1.3 软件工程的定义

1968 年秋季，NATO（北约）的科技委员会召集了近 50 名一流的编程人员、计算机科学家和工业界巨头，讨论和制定摆脱“软件危机”的对策。在会议上第一次提出了软

件工程（software engineering）这个概念。当时提出这个概念的 Fritz Bauer 的主要思路是想将系统工程的原理应用到软件的开发和维护中。

软件工程是一门研究如何用系统化、规范化、数量化等工程原则和方法去进行软件的开发和维护的学科。可以定义为：软件工程是一类设计软件的工程。软件工程应用计算机科学、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、数学用于构建模型与算法；工程科学用于制定规范、设计规范、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。软件工程学是一门指导计算机软件开发和维护的科学。

软件工程包括两方面内容：软件开发技术和软件项目管理。其中，软件开发技术包括软件开发方法学、软件工具和软件工程环境，软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

统计数据表明，大多数软件开发项目的失败，并不是由于软件开发技术方面的原因，而是由于不适当的管理造成的。遗憾的是，尽管人们对软件项目管理重要性的认识有所提高，但在软件管理方面的进步远比在设计方法学和实现方法学上的进步小，至今还提不出一套管理软件开发的通用指导原则。

在软件的长期发展中，人们针对软件危机的表现和原因，经过不断的实践和总结，越来越清楚地认识到：按照工程化的原则和方法组织软件开发工作，是摆脱软件危机的一个主要出路。今天，尽管“软件危机”并未被彻底解决，但软件工程三十多年的发展仍可以说是硕果累累。

1.2　软件工程的研究对象和基本原理

1.2.1　软件工程的研究对象

软件工程是相当复杂的。涉及的因素很多，不同软件项目使用的开发方法和技术也是不同的，而且有些项目的开发无现成的技术，带有不同程度的试探性。一般来说，软件工程包含 4 个关键元素：方法（methodologies）、语言（languages）、工具（tools）和过程（procedures）。

软件方法提供如何构造软件的技术，包括以下内容：与项目有关的计算和各种估算，系统和软件需求分析，数据结构设计，程序体系结构，算法过程，编码，测试和维护等。软件工程的方法通常引入各种专用的图形符号，以及一套软件质量的准则。概括地说，软件工程方法规定了以下内容：明确的工作步骤与技术；具体的文档格式；明确的评价标准。

软件语言用于支持软件的分析、设计和实现。随着编译程序和软件技术的完善，传统的编程语言表述能力更强、更加灵活，而且支持过程实现更加抽象的描述。与此同时，规格说明语言和设计语言也开始有更大的可执行子集。而且现在还发展了原型开发语言，所谓原型开发语言就是除必须具有可执行的能力外，还必须具有规格说明和设计这两种语言的能力。

软件工具是人类在开发软件的活动中智力和体力的扩展和延伸，为方法和语言提供

自动或半自动化的支持。软件工具最初是零散的，后来根据不同类型软件项目的要求建立了各种软件工具箱，支持软件开发的全过程。更进一步，人们将用于开发软件的软、硬件工具和软件工程数据库（包括分析、设计、编码和测试等重要信息的数据结构）集成在一起，建立集成化的计算机辅助软件工程（computer-aided software engineering）系统，简称 CASE。

软件过程贯穿于软件开发的各个环节。软件过程定义了方法使用的顺序、可交付产品（文档、报告以及格式）的要求、为保证质量和协调变化所需要的管理，以及软件开发过程各个阶段完成的标志。

从内容上说，软件工程包括软件开发理论和结构、软件开发技术以及软件工程管理和规范。其中，软件开发理论和结构包括：程序正确性证明理论、软件可靠性理论、软件成本估算模型、软件开发模型以及模块划分原理；软件开发技术包括：软件开发方法学、软件工具以及软件环境；软件工程管理和规范包括：软件管理（人员、计划、标准、配置）以及软件经济（成本估算、质量评价）。即软件工程可分为理论、结构、方法、工具、环境、管理和规范等。理论和结构是软件开发的基础；方法、工具、环境构成软件开发技术；好的工具促进方法的研制，好的方法能改进工具；工具的集合构成软件开发环境；管理是技术实现与开发质量的保证；规范是开发遵循的技术标准。

在软件工程中，软件的可靠性是软件在所给条件下和规定的时间内，能完成所要求的功能的性质。软件工程的软件可靠性理论及其评价方法，是贯穿整个软件工程各个阶段所必须考虑的问题。

软件工程的目标在于研究一套科学的工程化方法，并与之相适应，发展一套方便的工具与环境，供软件开发者使用。

1.2.2 软件工程的基本原理

自从 1968 年提出“软件工程”这一术语以来，研究软件工程的专家学者们陆续提出了许多关于软件工程的准则或信条。美国著名的软件工程专家 Boehm 综合这些专家的意见，并总结了 TRW 公司多年开发软件的经验，于 1983 年提出了软件工程的 7 条基本原理。Boehm 认为，这 7 条原理是确保软件产品质量和开发效率的原理的最小集合。这 7 条原理是相互独立、缺一不可的最小集合，同时又是相当完备的。

下面简要介绍软件工程的 7 条原理。

1．用分阶段的生命周期计划严格管理

这一条是吸取前人的教训而提出来的。统计表明，50%以上的失败项目是由于计划不周而造成的。在软件开发与维护的漫长生命周期中，需要完成许多性质各异的工作。这条原理意味着，应该把软件生命周期分成若干阶段，并相应制定出切实可行的计划，然后严格按照计划对软件的开发和维护进行管理。Boehm 认为，在整个软件生命周期中应指定并严格执行六类计划：项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划和运行维护计划。

2．坚持进行阶段评审

统计结果显示大部分错误是设计错误，大约占 63%；错误发现得越晚，改正错误付

出的代价就越大，相差大约 2 到 3 个数量级。因此，软件的质量保证工作不能等到编码结束之后再进行，应坚持进行严格的阶段评审，以便尽早发现错误。

3．实行严格的产品控制

开发人员最痛恨的事情之一就是改动需求。但是需求的改动往往是不可避免的。这就要求开发人员要采用科学的产品控制技术来顺应这种要求，也就是要采用变动控制，又叫基准配置管理。当需求变动时，其他各个阶段的文档或代码随之相应变动，以保证软件的一致性。

4．采纳现代程序设计技术

从 20 世纪 60、70 年代的结构化软件开发技术，到最近的面向对象技术，从第一、第二代语言，到第四代语言，人们已经充分认识到：方法比气力更有效。采用先进的技术既可以提高软件开发的效率，又可以减少软件维护的成本。

5．结果应能清楚地审查

软件是一种看不见、摸不着的逻辑产品。软件开发小组的工作进展情况可见性差，难于评价和管理。为更好地进行管理，应根据软件开发的总目标及完成期限，尽量明确地规定开发小组的责任和产品标准，从而使所得到的标准能清楚地审查。

6．开发小组的人员应少而精

开发人员的素质和数量是影响软件质量和开发效率的重要因素，应该少而精。这一条基于两点原因：高素质开发人员的效率比低素质开发人员的效率要高几倍到几十倍，开发工作中犯的错误也要少得多；当开发小组为 N 人时，可能的通信信道为 $N(N-1)/2$，可见随着人数 N 的增大，通信开销将急剧增大。

7．承认不断改进软件工程实践的必要性

遵从上述 6 条基本原理，就能够较好地实现软件的工程化生产。但是，上述 6 条原理只是对现有的经验的总结和归纳，并不能保证赶上技术不断前进发展的步伐。因此，Boehm 提出应把承认不断改进软件工程实践的必要性作为软件工程的第 7 条原理。根据这条原理，不仅要积极采纳新的软件开发技术，还要注意不断总结经验，收集进度和消耗等数据，进行出错类型和问题报告统计。这些数据既可以用来评估新的软件技术的效果，也可以用来指明必须着重注意的问题和应该优先进行研究的工具和技术。

1.3　软件的生存期及常用的开发模型

1.3.1　软件的生存期

从某个待开发软件的目的被提出并着手实现，直到最后停止使用的这个过程，一般称之为软件生存期。软件工程采用的生命周期方法学就是从时间角度对软件开发和维护的复杂问题进行分解，把软件生存的漫长周期依次划分为若干个阶段，每个阶段有相对独立的任务，然后逐步完成每个阶段的任务。

把软件生存周期划分成若干个阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发工程的困难程度；在软件生存周期的每个阶段都采用科学的管理技术和良好的技术方法，而且在每个阶段结束之前都从技术和

管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发工程的全过程以一种有条不紊的方式进行，保证了软件的质量，特别是提高了软件的可维护性。

目前划分软件生存周期阶段的方法有许多种，一般来说，软件生存周期可以划分为以下几个阶段。

1．定义阶段

主要是确定待开发的软件系统要做什么。即软件开发人员必须确定处理的是什么信息，要达到哪些功能和性能，建立什么样的界面，存在什么样的设计限制，以及要求一个什么样的标准来确定系统开发是否成功；还要弄清系统的关键需求；然后确定该软件。大致分为 3 个步骤。

（1）系统分析：在这个阶段，系统分析员通过对实际用户的调查，提出关于软件系统的性质、工程目标和规模的书面报告，同用户协商，达成共识。

（2）制定软件项目计划：软件项目计划包括确定工作域、风险分析、资源规定、成本核算，以及工作任务和进度安排等。

（3）需求分析：对待开发的软件提出的需求进行分析并给出详细的定义。开发人员与用户共同讨论决定哪些需求是可以满足的，并对其加以确切的描述。

2．开发阶段

主要是要确定待开发的软件应怎样做，即软件开发人员必须确定对所开发的软件采用怎样的数据结构和体系结构，怎样的过程细节，怎样把设计语言转换成编程语言，以及怎样进行测试等。大致分为 3 个步骤。

（1）软件设计

主要是把对软件的需求翻译为一系列的表达式（如图形、表格、伪码等）来描述数据结构、体系结构、算法过程，以及界面特征等。一般又分为总体设计和详细设计。其中总体设计主要进行软件体系结构的分析；详细设计主要进行算法过程的实现。

（2）编码

主要依据设计表达式写出正确的容易理解、容易维护的程序模块。程序员应该根据目标系统的性质和实际环境，选取一种适当的程序设计语言，把详细设计的结果翻译成用选定的语言书写的程序，并且仔细测试编写出的每一个模块。

（3）软件测试

主要是通过各种类型的测试及相应的调试，以发现功能、逻辑和实现上的缺陷，使软件达到预定的要求。

3．维护阶段

主要是进行各种修改，使系统能持久地满足用户的需要。维护阶段要进行再定义和再开发，所不同的是在软件已经存在的基础上进行。

通常有 4 类维护活动。改正性维护，即诊断和改正在使用过程中发现的软件错误；适应性维护，即修改软件使之能适应环境的变化；完善性维护，即根据用户的新要求扩充功能和改进性能；预防性维护，即修改软件为将来的维护活动预先作准备。

在软件工程中的每一个步骤完成后，为了确保活动的质量，必须进行评审。为了保证系统信息的完整性和软件使用的方便，还要有相应的详细文档。

1.3.2　常用的软件开发模型

软件开发模型是软件开发的全部过程、活动和任务的结构框架。软件开发模型能清晰、直观地表达软件开发全过程，明确规定了要完成的主要活动和任务，是用来作为软件项目开发的基础。常见的软件工程模型有：瀑布模型、原型模型、演化模型、螺旋模型、喷泉模型、第四代技术过程模型等。

1．瀑布模型（waterfall model）

瀑布模型依据软件生命周期方法学开发软件，各阶段的工作自顶向下从抽象到具体的顺序进行，如图 1-2 所示。

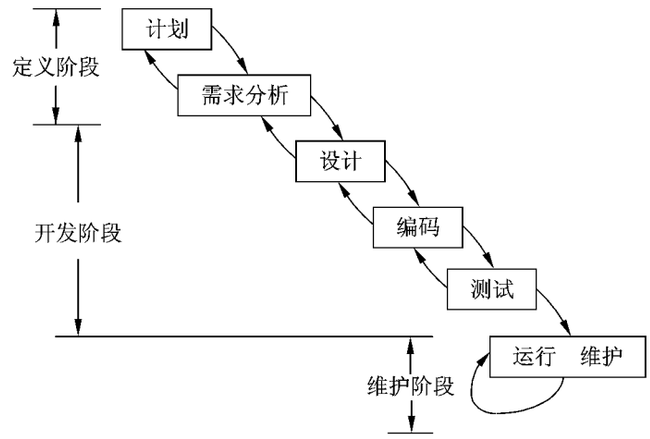


图 1-2　瀑布模型

利用瀑布模型开发软件系统时，每一阶段完成确定的任务后，若其工作得到确认，就将产生的文档及成果交给下一个阶段；否则返回前一阶段，甚至更前面的阶段进行返工。而不同阶段的任务，一般来说是由不同级别的软件开发人员承担的。

这种软件开发方法其特点是阶段间具有顺序性和依赖性，便于分工合作，文档便于修改，并有复审质量保证，但与用户见面晚，纠错慢，工期延期的可能性大。适合在软件需求比较明确、开发技术比较成熟、工程管理比较严格的场合下使用。

2．原型法模型（prototype model）

原型法是针对瀑布模型提出来的一种改进方法。其基本思想是从用户需求出发，快速建立一个原型，使用户通过这个原型初步表达出自己的要求，并通过反复修改、完善，逐步靠近用户的全部需求，最终形成一个完全满足用户要求的新系统。

依据这种模型开发软件时，开发人员和用户在“原型”上达成一致，避免了许多由于不同理解而造成的错误。提高了系统的实用性、正确性以及用户的满意度。由于是对一个有形的“原型产品”进行修改和完善，即使前面的设计有缺陷，也可以通过不断地修改原型产品，最终解决问题。缩短了开发周期，加快了工程进度。原型法本身不需要大量验证性测试以及前两点的原因，降低了系统的开发成本。但当开发者在不熟悉的领

域中不易分清主次，产品原型在一定程度上限制了开发人员的创新；资源规划和管理较为困难，随时更新文档也带来麻烦；还有可能只注意原型是否满意，忽略了原型环境与用户环境的差异。

一般又把原型分为 3 类：抛弃式，目的达到即被抛弃，原型不作为最终产品；演化式，系统的形成和发展是逐步完成的，是高度动态迭代和高度动态的，每次迭代都要对系统重新进行规格说明、重新设计、重新实现和重新评价，所以是对付变化最为有效的方法，这也是与瀑布开发的主要不同点；增量式，系统是一次一段地增量构造，与演化式原型的最大区别在于增量式开发是在软件总体设计基础上进行的。很显然，其对付变化比演化式差。

3．演化模型（evolutionary model）

演化模型主要针对事先不能完整定义需求的软件开发。用户可以给出待开发系统的核心需求，并且当看到核心需求实现后，能够有效地提出反馈，以支持系统的最终设计和实现。软件开发人员根据用户的需求，首先开发核心系统。当该核心系统投入运行后，用户试用之，并提出精化系统、增强系统能力的需求。软件开发人员根据用户的反馈，实施开发的迭代过程。第一迭代过程均由需求、设计、编码、测试、集成等阶段组成，为整个系统增加一个可定义的、可管理的子集。

在开发模式上采取分批循环开发的办法，每循环开发一部分的功能，成为这个产品的原型的新增功能。于是，设计就不断地演化出新的系统。实际上，这个模型可看作是重复执行的多个“瀑布模型”。

演化模型要求开发人员有能力把项目的产品需求分解为不同组，以便分批循环开发。这种分组并不是绝对随意性的，而是要根据功能的重要性及对总体设计的基础结构的影响而作出判断。

演化模型的特点是通过逐步迭代，建立软件系统。其适合场合为需求没有或者难以完整定义的软件，注意与原型模型之间的区别。

4．螺旋模型（spiral model）

螺旋模型将瀑布模型与原型模型结合起来，并且加入风险分析，构成具有特色的模式，弥补了前两种模型的不足，是演化模型的一种具体形式。螺旋模型将工程划分为 4 个主要活动：制定计划、风险分析、实施工程、用户评价。4 个活动螺旋式地重复执行，直到最终得到用户认可的产品，如图 1-3 所示。

在螺旋模型中，软件开发是一系列的增量发布。在每一个迭代中，被开发系统的更加完善的版本逐步产生。螺旋模型被划分为若干框架活动，也称为任务区域。典型地，有以下任务区域。

（1）客户交流：建立开发者和客户之间的有效通信，正确定义需求。

（2）计划：定义资源、进度及其他相关项目信息，即确定软件目标，选定实施方案，弄清项目开发的限制条件。

（3）风险分析：评估技术的及管理的风险，即分析所选方案，考虑如何识别和消除风险。

（4）工程：建立应用的一个或多个表示，即设计软件原型。

（5）设计与制作：构造、测试、安装和提供用户支持，即实施软件开发。

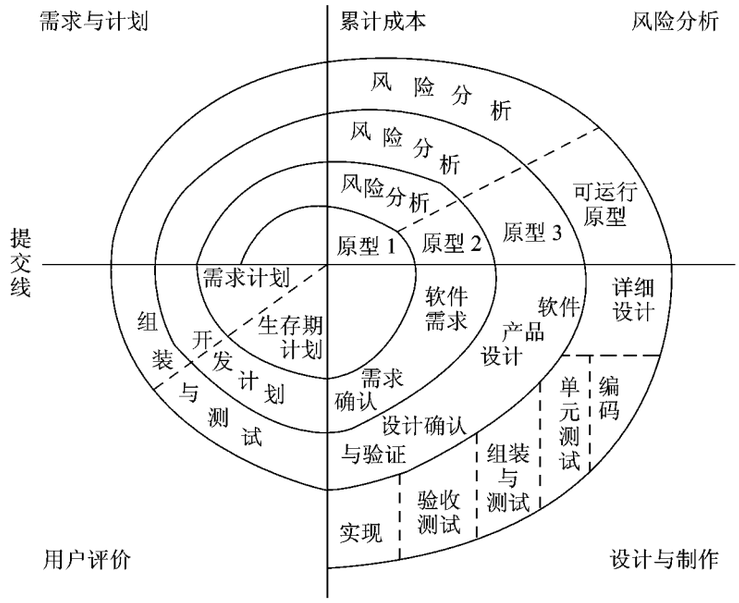


图 1-3　螺旋模型

（6）客户评估：基于对在工程阶段产生的或在安装阶段实现的软件表示的评估，获得客户反馈，即评价开发工作，提出修正建议。

对于大型系统及软件的开发，螺旋模型是一个很好的方法。开发者和客户能够较好地对待和理解每一个演化级别上的风险。但需要相当的风险分析评估的专门技术，且成功与否依赖于这种技术。很明显，一个大的没有被发现的风险，将会导致问题的发生，可能导致演化的方法失去控制。

5．喷泉模型（fountain model**）**

喷泉模型是一种面向对象的生存期模型。与传统的结构化生存期比较，具有更多的增量和迭代性质，生存期的各个阶段可以相互重叠和多次反复，而且在项目的整个生存期中还可以嵌入子生存期。就像水喷上去又落下来，可以落在中间，也可以落在最底部，如图 1-4 所示。

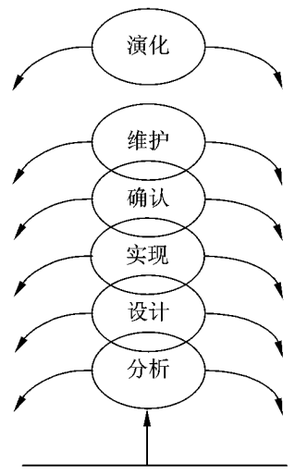


图 1-4　喷泉模型

喷泉模型体现了软件创建所固有的迭代和无间隙的特征。所谓迭代开发，是指基于对一个系统进行连续的扩充和精化，需要经历若干个开发周期，每个周期都需要经历分析、设计、实现和测试阶段。每个开发周期只针对比较小的一部分需求。所谓无间隙是指软件开发活动，即分析、设计和编码之间不存在明显的边界。这一模型表明了软件刻画活动需要多次重复。

6．第四代技术（fourth-generation techniques）

第四代技术（4GT）过程模型拥有一组工具（如数据查询、报表生成、数据处理、屏幕定义、代码生成、高层图形功能及电子表格等），每个工具都能使开发人员在高层次上定义软件的某些特性，并把开发人员定义的这些软件自动地生成为源代码。第四代技术需要四代语言（fourth-generation language，4GL）的支持，4GL 不同于三代语言，其主要特征是用户界面极端友好，即使没有受过训练的非专业程序员，也能用 4GL 编写程序；是一种声明式、交互式和非过程性编程语言。4GL 还具有高效的程序代码、智能默认假设、完备的数据库和应用程序生成器。目前市场上流行的 4GL（如 FoxPro 等）都不同程度地具有上述特征。

和其他模型一样，4GT 也是从需求收集这一步开始的，要将一个 4GT 实现变成最终产品，开发者还必须进行彻底的测试，开发有意义的文档，并且同样要完成其他模型中同样要求的所有集成活动。

在过去十余年中，4GT 的使用发展得很快，且目前已成为适用于多个不同的应用领域的方法。与计算机辅助软件工程（CASE）工具和代码生成器结合起来，4GT 为许多软件问题提供了可靠的解决方案，已经成为软件工程的一个重要方法，但 4GT 目前主要限于事务信息系统的中、小型应用程序的开发。

本 章 小 结

随着计算机硬件性能的极大提高和计算机体系结构的不断变化，计算机软件系统更加成熟和更为复杂，从而促使计算机软件的角色发生了巨大的变化。从最初的编写简单程序到如今大型软件的开发，大致经历了 4 个阶段。软件从个性化的程序变为工程化的产品，人们对软件的看法发生了根本性的变化。

所谓计算机软件是计算机系统中与硬件相互依存的另一部分，包括程序、相关数据及其说明文档。其中程序是按照事先设计的功能和性能要求执行的指令序列；数据是程序能正常操纵信息的数据结构；文档是与程序开发维护和使用有关的各种图文资料。

软件同传统的工业产品相比，有其独特的特性：是一种逻辑产品；其成本主要体现在开发和研制上；在使用过程中，没有磨损、老化的问题，但肯定会进行修改或淘汰；对环境有一定的依赖性；软件生产效率较低；软件产品复杂；成本昂贵；软件工作涉及的社会因素较多。

由于软件自身的特点及软件开发和维护的不正确方法，导致在软件发展过程中不可避免地出现软件危机。所谓软件危机就是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。主要体现在：不能准确估计开发成本及进度；不能满足用户的要求；产品质量较低；可维护性低；开发成本不断提高；软件开发远远落后于硬件的发展和用户的需求。

在软件的长期发展中，人们针对软件危机的表现和原因，经过不断的实践和总结，越来越清楚地认识到：按照工程化的原则和方法组织软件开发工作，是摆脱软件危机的一个主要出路。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。软件工程包括两方面内容：软件开发技术

和软件项目管理。其中，软件开发技术包括软件开发方法学、软件工具和软件工程环境。软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

一般来说，软件工程包含4个关键元素：方法、语言、工具和过程。软件方法提供如何构造软件的技术；软件语言用于支持软件的分析、设计和实现；软件工具是人类在开发软件的活动中智力和体力的扩展和延伸,为方法和语言提供自动或半自动化的支持；软件过程贯穿于软件开发的各个环节，定义了方法使用的顺序、可交付产品的要求、为保证质量和协调变化所需要的管理，以及软件开发过程各个阶段完成的标志。

从内容上说，软件工程包括：软件开发理论和结构、软件开发技术以及软件工程管理和规范。

在软件工程中还必须遵循如下7条基本原则：用分阶段的生命周期计划严格管理；坚持进行阶段评审；实行严格的产品控制；采纳现代程序设计技术；结果应能清楚地审查；开发小组的人员应少而精；承认不断改进软件工程实践的必要性。

软件工程采用的生命周期方法学就是从时间角度对软件开发和维护的复杂问题进行分解，把软件生存的周期依次划分为若干个阶段，每个阶段有相对独立的任务，然后逐步完成每个阶段的任务。一般来说，软件生存周期可以划分为以下几个阶段：定义阶段、开发阶段和维护阶段。其中定义阶段主要是确定待开发的软件系统要做什么，大致分系统分析、制定软件项目计划和需求分析3个步骤；开发阶段主要是要确定待开发的软件应怎样做，大致分为软件设计、编码和软件测试3个步骤；维护阶段主要是进行各种修改，使系统能持久地满足用户的需要。通常有改正性维护、适应性维护、完善性维护及预防性维护。

软件开发模型是软件开发的全部过程、活动和任务的结构框架。软件开发模型能清晰、直观地表达软件开发全过程，明确规定了要完成的主要活动和任务，是用来作为软件项目开发的基础。常见的软件工程模型有：瀑布模型、原型模型、演化模型、螺旋模型、喷泉模型、第四代技术过程模型等。这些模型各有优缺点，适用于不同的实际情况。

瀑布模型将软件生存周期的各项活动规定为依固定顺序连接的若干阶段工作，是一种线性模型。各阶段活动为，提出系统需求、提出软件需求、需求分析、设计、编码、测试和运行。每个开发阶段具有以下特征，从上一阶段接受本阶段工作的对象作为输入，对上述输入实施本阶段的活动，给出本阶段的工作成果作为输出传入下一阶段，对本阶段工作进行评审，若本阶段工作得到确认，则继续下阶段工作，否则返回前一阶段甚至更前阶段。瀑布模型最为突出的缺点是该模型缺乏灵活性。

原型模型从需求收集开始，开发者和客户在一起定义软件的总体目标，标识已知的需求并且规划出需要进一步定义的区域。然后是“快速设计”，集中于软件中那些对客户可见的部分的表示，这将导致原型的创建，并由客户评估并进一步精化待开发软件的需求。逐步调整原型使其满足客户的需求。但可能导致系统设计差、效率低，难于维护。

演化模型主要针对事先不能完整定义需求的软件开发，其开发过程一般是首先开发核心系统，当核心系统投入运行后，软件开发人员根据用户的反馈，实施开发的迭代过程，每一迭代过程均由需求、设计、编码、测试、集成等阶段组成，直到软件开发结束。演化模型在一定程度上减少了软件开发活动的盲目性。

螺旋模型是在瀑布模型和演化模型的基础上，加入两者所忽略的风险分析所建立的

一种软件开发模型。沿螺旋模型顺时针方向，依次表达了 4 个方面的活动，即制定计划、风险分析、实施工程、客户评估。

喷泉模型体现了软件创建所固有的迭代和无间隙特征，喷泉模型主要用于支持面向对象开发过程。

瀑布模型、演化模型、螺旋模型都分为多个阶段，而瀑布模型一次完成软件，演化模型分为多次完成，每次迭代完成软件的一个部分，螺旋模型也分为多次完成，每次完成软件的一个新原型，并考虑风险分析。

习 题 1

1.1 什么是软件？软件有什么特点？

1.2 什么是软件危机？软件危机有哪几种表现？其产生的原因是什么？怎样消除软件危机？

1.3 什么是软件工程？构成软件工程的要素是什么？

1.4 软件工程研究的基本内容是什么？

1.5 软件工程的基本原理有哪些？在过去的软件开发实践中有哪些是符合这些原理的？

1.6 软件生存期分哪几个阶段？这种循序渐进的方式的优点在哪里？

1.7 简述你所知道的软件开发模型，各有什么特点？

第2章 可行性研究和需求定义 3~9

一个软件是否值得开发，需要多长的开发时间和多少投资，存在着哪些风险，如何预防这些风险，都要在开发前预先研究，才能减少失误。这就是计划时期的主要任务——可行性研究，也是本章重点阐述的问题之一。一旦确定开发某个项目，就进入需求定义阶段。

本章主要介绍可行性研究和需求定义两个阶段的内容，最后给出典型应用分析。在进入可行性研究之前需要分析和确定问题的定义。在问题定义阶段初步确定软件目标和规模，如果正确就进行下一步的研究；如果错误，就及时进行改正。

2.1 问 题 定 义

问题定义（problem definition）即软件定义，是计划时期的第一个阶段，为软件的可行性研究和软件开发计划的制定提供功能与性能的依据。其目的是弄清用户需要计算机解决的根本问题，以及项目所需的资源和经费。

2.1.1 问题定义的基本任务

问题定义阶段要回答的关键问题是：“要解决的问题是什么？”因此，其基本任务就是分析要解决的问题，提交问题定义报告。经用户同意后，就可作为下一步工作——可行性研究的依据。

系统分析员在问题定义阶段应通过对系统的实际用户和使用部门负责人的访问调查，写出对问题的理解，搞清楚用户为什么会提出这样的问题，问题的背景和用户的目标是什么。然后据此提出关于问题的性质、工程的目标和规模的书面报告，并在用户和使用部门负责人参加的会议上认真讨论这份书面报告，澄清含糊不清的地方，改正理解不正确的地方。最后形成一份双方都满意的文档，以确保开发人员、用户和使用部门的负责人对问题的性质、工程的目标和规模取得一致的看法，从而进入下一阶段的工作。

问题定义阶段是软件生命周期中最简短的阶段，一般只需要一天甚至更短的时间。

2.1.2 问题定义报告

在对问题作出定义之后，应提出关于问题的性质、工程的目标和规模的书面报告。问题定义报告没有统一的文档格式，但一般来说应该包括以下内容：

（1）工程项目名称；
（2）使用方；
（3）开发方；
（4）对问题的概括定义；
（5）项目的目标；
（6）项目的规模。

有时还可以加入对项目的初步设想和对可行性研究的建议等内容。

问题定义报告的内容应简洁、清楚，一般在一页纸以内，如图 2-1 所示。

问题定义报告

用户单位：×××××学校计财处
负 责 人：×××
分析员单位：××软件开发公司
分 析 员：×××
项目名称：工资管理系统
问题概述：计财处每月的工资管理工作太忙，在工资管理事务上花费精力太大……（其他的主要问题）
项目目标：开发一个效率较高的工资管理系统
项目规模：这个项目的开发成本约是×万元
可行性研究：建议进行一周，费用不超过×××元

××年×月×日　　签字：×××

图 2-1　问题定义报告

在问题定义阶段对要解决的问题做了概要的描述，并分析了项目的目标和大致的规模，但在预定的规模内，问题是否可以解决呢？这将是可行性研究阶段的主要任务。

2.2 可行性研究

开发一个基于计算机的系统会受到时间和资源上的限制。所以，在一个新项目开发之前，应该根据客户提供的时间和资源条件进行可行性研究，这样可以避免人力、物力和财力上的浪费。

2.2.1　可行性研究的内容及过程

1．可行性研究的内容

可行性研究的任务不是研究如何解决问题，而是要用最小的代价在最短的时间内，确定问题定义阶段所定义的问题是否值得解决。在预定的规模内是否有可行的解，即在较高层次上以较抽象的方式进行系统分析，论证系统开发的可行性。

一般情况下，主要应从三方面论证系统开发的可行性。

1）技术可行性

技术可行性是根据客户提出的系统功能、性能以及实现系统的各项约束条件，从技术的角度研究实现系统的可行性。即分析使用现有的技术是否能实现这个系统，能否解决系统的技术难点，系统对技术人员有什么要求，现有的技术人员能否胜任，开发系统的软件、硬件资源是否能如期得到等。

技术可行性研究往往是系统开发过程中难度很大的工作。由于系统分析和定义过程与系统技术可行性评估过程同时进行，因此系统目标、功能和性能的不确定性会给技术可行性论证带来一定的困难。

技术可行性研究应该包括风险分析、资源分析和技术分析。

（1）风险分析的任务是在给定的约束条件下，判断能否设计并实现系统所需的功能和性能。

（2）资源分析的任务是论证是否具备系统开发所需的各类管理人员和专业技术人员、软件、硬件资源和工作环境等。

（3）技术分析的任务是当前的科学技术是否支持系统开发的全过程。

技术可行性评估是系统可行性研究的关键。这一决策的失误将会给开发工作带来灾难性的影响。在技术可行性研究过程中，系统分析员应该采集系统的性能、可靠性、可维护性等各方面的信息，分析实现系统功能及性能所需要的各种设备和技术、方法和过程，分析项目开发在技术方面可能担负的风险，以及技术问题对开发成本的影响等。

2）经济可行性

经济可行性就是通过成本-效益分析，评估系统的经济效益是否超过其开发成本，并将估算的成本与预算的利润进行对比，分析系统开发对其他产品或利润的影响。即分析开发这个系统是否能取得经济效益。要作出投资的估算和系统投入运行后可能获得的收入或可节约的费用的估算。

因此，基于计算机系统的成本-效益分析是可行性研究的重要内容，但由于项目开发成本受项目的特性、规模等多种因素的制约，所以系统分析员很难直接估算基于计算机系统的成本和利润，得到完全精确的成本-效益分析结果十分困难。

系统成本主要包括：

（1）购置硬件软件的费用；

（2）有关设备的工程安装费用；

（3）系统开发费用；

（4）系统的安装、运行和维护费用；

（5）人员培训费用等。

系统效益包括：

（1）经济效益。系统为用户增加的经济收入，可以通过直接的或者统计的方法估算。

（2）社会效益。只能用定性的方法估算，例如产品广告宣传、影响等。

3）操作可行性

操作可行性，又称为社会可行性或运行可行性，是对开发系统在一个给定的工作环境中能否运行或运行好坏程度的量度。操作可行性分析可以确定在当前政治意识形态、法律法规、社会道德、民族意识以及系统运行的组织机构和人员等环境下，系统是否可行。

总之，可行性研究最根本的任务，是对以后的行动方针提出建议。如果问题没有可行解，分析员应该建议停止这项开发工作，以避免时间、资源、人力和金钱的浪费；如果问题值得解决，分析员应该推荐一个较好的解决方案，并为系统制定一个初步的开发计划。

2．可行性研究的过程

典型的可行性研究主要步骤如下：

① 对问题定义阶段的分析结果和报告书进行复查，改正含糊或不确切的叙述，重新确定工程目标与规模，清晰地描述对目标系统的一切限制和约束。

② 研究现在正在使用的系统，找出其基本功能和需要的基本信息，绘制系统流程图和高层数据流图，并指出其局限性。

③ 导出新系统的高层逻辑模型，绘制系统流程图和数据流图，并与现有系统进行比较。

④ 导出若干高层次的物理解法，通过对解法的技术可行性、经济可行性、运行可行性进行比较选择，推荐行动方案。

⑤ 如果分析员认为应该继续这项开发，应该选择一种最好的解法，说明理由，并为推荐的系统草拟一份开发计划；若分析员认为不值得继续进行这项开发，应提出停止开发的建议，最后提交全部的可行性研究文档进行审查和复审。

2.2.2 可行性研究报告

可行性研究实际上是一个较高层次的、较抽象的系统分析和设计过程。可行性研究的结果可作为系统规格说明书的一个附件。可行性研究报告有多种形式，其中一个具有普遍性的可行性研究报告目录模板如图 2-2 所示。

可行性研究应该保证有明显的经济效益和较低的技术风险，一定要没有各种法律问题，以及其他更合理的系统开发方案，否则应该做进一步的研究。

××××××可行性研究报告

1　引言
　1.1　问题
　1.2　实现环境
　1.3　约束条件
2　管理
　2.1　重要的发展
　2.2　注解
　2.3　建议
　2.4　效果
3　方案选择
　3.1　选择系统配置
　3.2　选择方案的标准
4　系统描述
　4.1　缩写词
　4.2　各个子系统的可行性
5　技术风险评价
6　成本-效益分析
7　有关法律问题
8　其他
9　结论意见

图 2-2　可行性研究报告

2.3　需 求 定 义

如果系统分析员已完成可行性分析报告，而且用户方的决策者认为该报告合理，决定进行项目的开发，这时就可以进入系统开发的需求定义阶段，包含需求获取、需求规格说明以及需求规格说明书等内容。

2.3.1　需求获取的内容

针对需求分析的目标，需求获取的内容主要有以下几个方面。

1．功能需求

列举出所开发的软件在功能上必须完成的任务，也就是系统在功能上“做什么”。

2．性能需求

给出所开发软件的技术性能指标，包括存储容量限制、运行时间限制、安全保密性等。

3．环境需求

这是对软件系统运行时所处环境的要求。例如，在硬件方面，采用什么机型、什么外部设备、什么数据通信接口等；在软件方面，采用什么系统软件（指操作系统、网络软件、数据库管理系统等）等；在使用方面，使用部门应具备什么样的条件等。

4．安全保密要求

工作在不同环境的软件对其安全、保密的要求显然是不同的。应对这方面的需求作

出恰当的规定，以便给予特殊的设计，从而保证其具有安全保密方面的性能。

5．用户界面需求

友好的用户界面是用户方便、有效、愉快地使用该软件的关键之一。因此，在需求分析时，必须对用户界面细致地给出规定。

6．资源使用需求

这是指所开发的软件运行时所需的数据、软件、内存空间等各项资源。另外，软件开发时所需的人力、支撑软件、开发设备等则属于软件开发的资源，需要在需求分析时加以确定。

7．软件成本消耗与开发进度需求

在软件项目立项后，要根据合同规定，对软件开发的进度和活动的费用提出要求，作为开发管理的依据。

用户总是关注功能性需求，忽视非功能性需求。其实非功能性需求也是非常重要的。非功能性需求反映了软件的特性，这些特性包括产品的易用程度、执行速度、可靠性、异常处理等。这些特性被称为软件质量属性或质量因数。

2.3.2 需求获取的方法

1．会谈

会谈是收集用户需求的一种重要的方式。会谈是指开发组的成员和用户方的成员将要开发的系统进行面对面的交谈，通常需要进行多次会谈才能完成。会谈之前，开发方应该事先做好充分的准备，包括需要讨论的一些重要问题。会谈之后，开发方应该尽快将会谈的结果整理出来，并且最好将整理出的会谈纪要发送给开发方的相关人员，以便开发方的相关人员能够对本次会谈的结果提出宝贵的意见。

2．问卷调查

另外一种收集用户需求的方式是事先充分准备好问卷，将问卷发送给用户方的相关人员。这种方式在需要调查的人特别多时非常有用。而且，因为采用这种方式时用户可以有充分的时间对问题进行思考（会谈时通常都是边问边答），所以这种方式得到的调查结果可能会比会谈得到的结果更准确。当然，这种方式也有不如会谈的地方，会谈时，如果会谈的双方都很积极主动并且组织得很好的话，会比问卷调查得到更好的效果，这是因为问卷是事先做好的，所提的问题可能很完善，但却不能像会谈那样可以临时修改。

3．收集用户表格和报表

还有一种收集用户需求的方法是收集用户机构中使用的相关表格和报表。从这些表格和报表中，可以了解用户目前所使用的系统（可能存在需要修改或升级的老的计算机系统，也可能完全是手工操作）的一些基本情况，对这些基本情况进行快速分析，即可找出其中存在的一些问题，然后对这些问题采取一些其他的需求收集方法（如会谈和问卷调查），则可以更快地收集到用户的一些真正需求。

4．使用用例

Ivar Jacobson 在 1992 年提出了一种用例（use case，UC）驱动的面向对象软件工程（OOSE）。在 OOSE 中，Ivar Jacobson 把用例的作用提高到项目开发基本要素的高度。这种方法很快为大家所接受，并且成为面向对象方法的重要组成部分。实际上，使用其他方法进行软件开发时也可以将用例的使用作为收集用户需求的重要技术和方法。

与用例密切相关的一个概念是参与者（actor）。参与者是所有与系统有信息交换的系统之外的事物。参与者与用户是不完全相同的，参与者是指用户在与系统交互时所充当的角色。当用户使用系统时，会执行一个行为相关的事务序列，这个序列是在与系统的会话中完成的，这个序列即称为用例。

找出用例的最简单的方法是与典型用户进行交谈，请用户讲出希望系统做哪些事情，能够提供哪些服务。软件开发人员则记录用户想要做的事情，并为这些事情取一些相应的名字，并写上简短的文字说明。当然，用例的记录也可以使用其他的形式。如近年来在面向对象领域普遍被接受的统一建模语言（UML）及其相应的建模工具（如 rational rose 工具）中就使用了图形化的用例来捕获用户的需求，有关内容将在以后的章节中讨论。

5．建立快速原型

快速原型是一种快速建立起来的能够展示目标系统的关键功能的、可实际在计算机上运行的软件，已经在第 1 章软件生存周期模型的相关内容中做了介绍，并指出，作为一种生存周期模型，快速原型存在一些缺点，因此，实际情况中完全使用快速原型来完成整个软件开发的很少。但是，将快速原型作为一种用户需求收集方法，则是非常适合的。因为用户可以很快看到可运行的系统，从而能够比较中肯地指出哪些部分是想要的，哪些部分是不想要的，哪些功能应该加进去，等等。然后，开发人员可以根据用户的反馈再次建立快速原型并再次让用户使用，征得用户的反馈。这样的过程重复若干次后，如果用户满意了，即可认为收集到了用户的真实需求。收集到用户的需求之后，即可使用下一节的需求规格说明，将用户的需求以规范的文档形式记录下来。

2.3.3　需求规格说明的内容

需求规格说明的内容应包括功能与行为的需求描述以及非功能（行为）需求描述。编写需求规格说明时，有许多软件需求规格说明的模板可以使用，许多软件人员选用的是 IEEE 推荐的软件需求规格说明的方法（IEEE830-1998 标准）。以下是在 IEEE 830 标准改写并扩充的软件需求规格说明的模板。事实上，可以根据项目的实际需要修改这个模板。

1　引言
- 1.1　目的
- 1.2　文档约定
- 1.3　预期的读者和阅读建议
- 1.4　产品的范围
- 1.5　参考文献

2　综合描述
- 2.1　产品的前景
- 2.2　产品的功能
- 2.3　用户类和特征
- 2.4　运行环境
- 2.5　设计和实现的限制

2.6 假设和依赖

3 外部接口

3.1 用户界面

3.2 硬件接口

3.3 软件接口

3.4 通信接口

4 系统特性

4.1 说明和优先级

4.2 激励/响应序列

4.3 功能需求

5 其他非功能需求

5.1 性能需求

5.2 安全设施需求

5.3 安全性需求

5.4 软件质量属性

5.5 业务规则

5.6 用户文档

6 其他需求

附录 A：词汇表

附录 B：软件分析模型

附录 C：待确定的问题

2.3.4 需求规格说明的评审

在软件设计之前，必须对需求规格说明进行评审。需求评审是一个手工过程，需要用户和开发者共同参与，检查文档中是否存在不规范和遗漏的地方。如果在评审过程中发现存在错误或者缺陷，应该及时进行更改和补充，重新进行相应部分的需求分析、需求建模等，然后再进行评审。

评审需求规格说明有如下主要指标。

（1）正确性。需求规格说明的功能、行为、性能描述等，必须与用户对目标软件产品的期望相吻合。

（2）无歧义性。对于用户、分析人员、设计人员和测试人员，对需求规格说明书中的任何语法单位只能有惟一的语义解释。

（3）完全性。需求规格说明不能遗漏任何用户需求。

（4）可验证性。对于规格说明中的任意需求，均存在技术和经济可行的手段进行验证和确认。

（5）一致性。需求规格说明的各部分之间不能相互矛盾。这些矛盾可能表现为表达术语使用方面的冲突、功能和行为特征方面的冲突以及时序方面的前后不一致。

（6）可理解性。软件需求规格说明对于用户、设计人员和测试人员等是否容易理解。

（7）可修改性。需求规格说明的格式和组织方式应该容易修改，能够比较容易地接纳后续的增加、删除和修改，并使修改后能够较好地保持其他各项属性。

（8）可追踪性。需求规格说明必须将分析后获得的每一项需求与用户的原始需求联系起来。

具体可以参考以下主要的评审内容。

（1）系统定义的目标是否与用户的要求一致。

（2）系统需求分析提供的文档资料是否齐全。

（3）文档中的所有描述是否完整、清晰、准确地反映了用户要求。

（4）与所有其他系统成分的重要接口是否都已经描述。

（5）所开发项目的数据流与数据结构是否足够，是否确定。

（6）所有图表是否清楚，在没有补充说明时是否易于理解。

（7）主要功能是否已包括在规定的软件范围之内，是否都已充分说明。

（8）设计的约束条件或限制条件是否符合实际。

（9）开发的技术风险是什么。

（10）是否考虑过软件需求的其他方案。

（11）是否考虑过将来可能会提出的软件需求。

（12）是否详细制定了检验标准，能否对系统定义的成败进行确认。

（13）有没有遗漏、重复或不一致的地方。

（14）用户是否审查了初步的用户手册。

（15）软件开发计划中的估算是否受到了影响。

2.3.5　需求规格说明书

根据软件需求规格说明 IEEE 指南（IEEE guide to software requirements specifications，ANSI/IEEE Std. 830 1984）的表述，应该按照功能描述规格，而不是按照结构或过程描述规格。许多文献把需求工程文档分为需求（分析）文档和规格说明文档，而其他文献通常把这两个文档结合在一起。

需求文档所包含的内容：问题域描述 + 待满足的需求列表（待求解的问题）。

规格说明文档包含：将满足需求的解系统的一种行为的定义（解决问题）。

规格说明书的核心是新的解系统的行为描述。因此，建议规格说明书包括如下内容：

文档细节（标题、授权、修订本历史等）

概述（问题域和解系统）

需求（从需求文档中复制而来）

　　功能需求

　　性能需求

　　设计约束

系统行为（通常是最大的部分）

参考书目

词汇表（DD）

索引

另外可选的内容有：

- 假设——最好能解决任何未知问题，这样不需要假设。但如果这是不可能的，那么最好明确地给出假设；
- 未实现的功能——有时不可能满足所有功能，那么通常最好说明未实现的功能。

1．文档细节

对于任何技术文档都是一样的：

- 标题；
- 作者；
- 文档说明约定／标准；
- 版本；
- 修改授权；
- 修改历史；
- 内容；
- ……。

通常按层次构造规格说明书，而且为了方便阅读，应有一个良好的子划分。其后的内容表通常相当长，但如果结构良好，则会为浏览文档带来很大方便。

2．概述

概述也可以称为“引言”（或者“概要”）。其目的是为新读者介绍主题并设置上下文。概述通常将以概述正要描述的系统开始，介绍总体目的，可能包括问题域的简短描述，但不应该复制需求文档中所给出的完整描述。还可能提供规格说明文档其他部分的概要或大纲，但应小心谨慎。因为内容表应当已给出了一个很好的描述，进一步重复可能会导致两个版本的不一致。当文档演化时，越来越可能产生这种不一致问题。

3．参考书目

参考书目是那些与项目密切相关的文档。相关文档可以包括：

- 需求获取记录；
- 需求／分析文档；
- 接口系统（端子）的规格说明书；
- 项目计划；
- 质量保证计划；
- 任何已有系统的用户手册；
- 设计约束下的开发过程；
- 系统必须满足的规定需求；
- 目标硬件的规格说明；
- ……。

4．词汇表（DD）

词汇表应该定义与具体项目相关的所有术语。规格说明将继承需求文档中所采用的

术语，但可以通过增加数据对这些术语进行扩充。

把所有这些定义放在 DD 中以方便用户使用，并有助于防止遗漏和重复。定义机制和命名原则在 DD 部分讨论。

5．索引

索引不是必需的，但有助于浏览。使用现代工具，可以相当容易地实现和维护索引。

2.4　典型应用分析

前面从“理论”上进行了探讨，下面从典型应用实例讨论需求开发规格说明。规格说明的目标是设计能满足需求的解系统的行为，并用文档说明。本案例是一个电梯控制系统。在编制规格说明书时，主要的参考是需求文档，包括初始系统的需求（通常复制到规格说明书中）：

R1　电梯不能移到最顶层之上或最底层之下（这里有一个紧急关闭系统，如果电梯移到最顶层之上或最底层之下（超出 l0cm），紧急关闭系统将停止电机，但这个关闭系统不属于控制系统）。

R2　电梯不能在快速模式下停下来，必须在停止之前至少在 1 秒钟内切换到慢速模式。

R3　电梯在移动时不能改变其极性（这会损坏卷扬设备）。

R4　当电梯门打开时，电梯不能移动。

R5　正确设置停止延迟后，电梯应在正予以服务的楼层的误差不超过±1.5cm 处停下来。

R6　通过按下相关楼层的发送按钮（在电梯内），或相关方向的调用按钮（在电梯外），可以建立一个调用请求（忽略重复调用请求）。

R7　只有当已得到电梯服务时，才能撤销一项调用请求。

R8　相关的电梯必须在指定的楼层停下来，以服务一个发送按钮的调用请求。正在以正确方向行进的任何电梯都可停在指定的楼层，以服务一个调用按钮的调用请求。

R9　电梯在以下情形时在某一楼层停下来：

R9.1　有一个发送到该楼层的调用请求；

R9.2　有一个来自调用按钮的调用请求，而且电梯正以相同方向行进；

R9.3　该楼层是最顶层或最底层。

如果不止一部电梯时：

R10　来自发送按钮的调用请求必须由相关的电梯提供服务。

R11　来自调用按钮的调用请求由可能最早到达的那部电梯提供服务（但这一点不能得到保证，因为所选择的电梯在行进途中可能要求服务新的请求，这是可以理解的）。

R12　每部电梯的使用量应大致相同。

R13　只有当电梯停在某一楼层时，才能改变方向。

R14　只有当电梯未收到当前行进方向的调用请求时，才能改变方向。

R15 对于每部电梯，在某一时间只有一个指示器亮，也就是与选定楼层距离（大约）最近的那部电梯的指示器亮。

R16 技术服务员必须能设置：

R16.1 电梯数；

R16.2 楼层数；

R16.3 每个电梯的停止信号延迟（在 0.10s 到 0.40s 的范围内）。

R17 所控制的最大电梯数是 4，最小是 1。

R18 最大楼层数是 20，最小是 2。

R19 控制系统不能违反任何安全性要求。非安全关键性控制错误（如电梯送到错误楼层）在每周的运作中不能超过一次。

R20 控制系统必须能放在 $1\times0.5\times0.5m^3$ 的容积内。

R21 控制系统必须在 0~400℃的温度范围内运作。

R22 无线电发射频率应遵守 BS50081-2 规定。

首先，合理考虑的是新的解系统（电梯控制系统）及其操作环境之间具有什么接口。问题框图和（稍许扩充了的）上下文图在这里很有帮助，因为从中可以确定控制系统与多种硬件（按钮、传感器、电机等）直接交互，与电梯和用户间接交互。

尽管问题并未直接提出，但需求（R16）表明必须还有一个技术服务员接口。这与其他部分是分离的，因而可单独定义。事实上，技术服务员接口在很大程度上已作为外部设计来处理。然而，技术服务员与系统的其他部分还有一些交互，有必要表明的是（事实上是警告）当技术服务员转动钥匙时，所有的电梯都将停下来。迄今为止，值得注意的是如何在技术服务员接口中寻找一些性能需求表达（R17 和 R18），任何超出所定义限制范围的尝试都将导致相应的错误响应。

对于该系统而言，用户接口的界面事实上是由给定硬件（按钮和指示器）事先确定的。然而系统的行为还有待定义，这些行为对用户是显而易见的。在很大程度上，用户所感受到的是电梯出现的（即受控的）行为及其相关硬件，因此有必要先定义。

硬件接口“界面”归结为控制系统与硬件之间的关联。发现那些关联的细节是外部设计的任务。这种情形下惟一能给出的实际指导原则是应用某些逻辑（或常识）。但一旦设计完成，则用适当的表来描述端口和引线的分配。表 2-1 给出了一个较小的例子。

表 2-1 端口和引线的分配

端口	用途	引线 0	引线 1	引线 2	引线 3	…	引线 29	引线 30	引线 31
1	备用								
2	向上调用按钮	0 层	1 层	2 层	3 层	… （到 20 层）	未使用	未使用	未使用
3	向下调用按钮	未使用	1 层	2 层	3 层	… （到 20 层）	未使用	未使用	未使用

电梯的行为必须用涉及所有其他接口（按钮、电机等）的“交叉接口”功能描述。例如，电机必须启动和停止以响应按下按钮和传感器信号。

经验表明，对于事件驱动系统的这种交叉接口功能，FSM 通常证明是最有效的定义

技术。因此，可以尝试使用 STD 或状态图。

对于初学者，这种模型首先要考虑的问题通常是很棘手的。一个有用的方法是从列举解系统可以发现的事件（从需求文档的问题子域描述中抽取而来）开始。在这里，列举出以下事件：

- 上方楼层传感器为 hi；
- 上方楼层传感器为 lo；
- 下方楼层传感器为 hi；
- 下方楼层传感器为 lo；
- 本楼层传感器为 hi；
- 本楼层传感器为 lo；
- 调用按钮为 hi；
- 调用按钮为 lo；
- 发送按钮为 hi；
- 发送按钮为 lo；
- 关门传感器为 hi；
- 关门传感器为 lo。

这些事件并不都是相关的（控制器需要知道本楼层传感器信号何时变为 hi，但无须知道何时变为 lo），但列举出这些事件仍是必需的。考察所考虑的系统（这里是电梯）可能具有的状态通常是有帮助的。没有必要一开始就给出全部的列表。如果能确定一些状态，就可以开始画状态图。而不要等到知道“答案”后才开始画（因为这样就永远不会开始），也不要期望第一次就完全正确。

这里首先考虑的一些明显的状态是“电梯停止”,“电梯快速移动”和“电梯慢速移动”。那么可以考虑触发相关转移的事件，画一个初期的状态图。然后是开发模型，主要的考虑是如何准确和全面地解决相关需求以及如何容易理解。作为这种考虑的一个例子，尝试用并发 FSM 描述速度和方向，但因为太多 ER 取决于速度和方向，用非并发模型会更清楚。

需求 R8，R11 和 R12（描述电梯对请求的分配）难以在模型中反映出来，但满足这些需求的行为易于用文本描述：

当不止一部电梯时，对调用按钮信号按以下方式处理：

- 把该请求分配给具有最短响应时间的电梯；
- 如果这些电梯具有相同的响应时间，则随机选择其中一部电梯。

每部电梯的响应时间按如下方式计算：

（1.2×楼层高度（m）×要行进的楼层数）+行进途中要停下来的次数×等待时间（s）+8

注：C 为某个常数

R5（停止位置的精确度）是一个有趣的需求。使用有关问题域的给定事实，可以通过一些计算来确定满足该需求的可行性。这里做了详细讨论，其结论是：

- 电梯将以 0.3m/s（±10%）的速度移动。
- 当电梯在所请求楼层的 20cm 的范围内时，临近传感器信号将变为 hi。
- 通过把慢速线路和快速线路设置为 lo，使电梯电机停下来。

- 电梯在实际停止之前将移动 0.15m（±20%）。

因此，可以计算出，在“最坏”情形下，电梯将以 0.33m/s 的速度行进，在电梯停下来之前行进 18cm。这样，在必须发出停止信号之前，只允许电梯移动 2cm，耗时 0.06s。在另一个极端，停止延迟将不得不是 0.3s。这看上去可行。需求文档也表明停止延迟可由技术员设置。现在，知道了必须允许的范围和粒度（0.01s），可以此为依据设计技术员接口。

由于要求停止精确度是在 1.5cm 之内，停止位置上的±3cm（0.15m 的 20%）的容许误差会出现问题。关键问题（需求文档中未说明）是每次停止时间不同还是每部电梯不同（但对于任何给定电梯是恒定的）。最好还是与客户讨论后再定。如果是前一个假设为真，就不能满足该需求（如果不改变硬件）。如果是后一个假设为真，可以满足该需求，但意味着必须单独地为每部电梯设置停止延迟（这样，再以此为依据描述技术员接口）。

有关电梯数和楼层数的性能需求（R17 和 R18）也可在技术员接口中找到。其可能的外部设计方案在完整的规格说明中进行了描述。

尽管已全面地描述了系统行为，为了有助于理解和验证，还可以显式地建模紧急行为的用户视图。用例是一种合适的技术，下面举例说明。

```
电梯行程
    用户调用电梯并行进到所要求的目标楼层
执行者
    用户
前置条件
    无
后置条件
    用户将到达所要求的目标楼层
交互
    1．用户到达电梯门外，按下相关的调用按钮
    2．电梯到达，电梯门打开
    3．用户进入电梯
    4．用户按下所要求的目标楼层的发送按钮
    5．电梯门关闭，电梯行进到目标楼层（可能在行进途中停在其他楼层），电梯门打开
    6．用户走出电梯
```

所能生成的用例的个数是任意的，例如还有其他的可能：

```
电梯已到达起始楼层，电梯门打开。
电梯已到达起始楼层，电梯门关闭。
```

这些可作为单独的用例编写，也可作为扩展或者另一种可选方案。进一步的选择是引入某些选择逻辑，并把许多情形结合起来，类似于功能分解。

```
用户到达电梯入口处
对于电梯：
    正在此处，电梯门打开
        用户进入电梯
    正在此处，电梯门已关或正在关闭
```

```
                用户按下所要行进方向的调用按钮
                电梯门打开
                用户进入电梯
            不在此处：
                用户按下所要行进方向的调用按钮
                当电梯到达时
                    电梯门打开
                    用户进入电梯
        用户按下所要的发送按钮
        电梯行进到所要求的楼层（可能在行进途中停在其他楼层）
        电梯在所要求的楼层停下来
        电梯门打开
        用户离开电梯
```

剩下的需求是可靠性需求（R19）和设计约束（R20 到 R22）。这些需求在规格说明中不能反映出来，而只是简单地作为需求。

本 章 小 结

计划时期是软件生存期的第一个时期，由问题定义、可行性分析两个阶段组成。问题定义的目的，在于澄清用户要解决的问题，虽然时间一般较短，却规定了整个开发工作的方向。其成果为问题定义报告。可行性分析是计划时期的主要阶段，目的在于确定问题有没有解，是不是值得去解。其目标是用较小的代价（约占开发费用的 5%~10%），尽快搞清投资的价值，避免冒太大的风险，确定开发后，应制定项目实施计划表，作为项目管理的依据。其成果为可行性研究报告。

用户作出项目开发的决策后，需求定义进入需求收集和需求功能规格说明两个子阶段。需求获取的内容有：信息需求（或数据需求）、功能需求、性能需求、运行需求及未来需求。需求获取的方法有：会谈、问卷调查、收集用户表格和报表、使用用例及建立快速原型。软件需求规格说明书是需求分析阶段的主要文档，提交并通过复审的需求规格说明书是需求分析结束的里程碑。

习　题　2

2.1　名词解释：可行性研究，需求收集，功能规格说明。

2.2　试说明问题定义阶段的任务。

2.3　可行性研究的目的是什么？简述其重要性。

2.4　可行性研究的内容是什么？如何进行技术可行性分析？

2.5　什么是操作可行性？试举例说明。

2.6　有哪些常用的需求收集的方法和技术？试选择某一系统并根据某方法进行需求收集。

2.7　需求规格说明的内容有哪些？

2.8　评审需求规格说明的指标及内容是什么？

2.9　你认为一个优秀的系统分析人员应该具备哪些知识，进行哪些方面的训练？

第3章 需求分析 3~9

需求分析要求详细、准确地分析清楚系统必须“做什么”，有时又称为软件系统分析，处于软件工程的开始部分，提供了构建软件项目其余部分的根基，关系到软件开发的成败。同时，随着面向对象（OO）、可视化编程（VP）、计算机辅助软件工程（computer aided software engineering，CASE）等软件开发技术的发展和应用，软件设计、编码、测试等环节的技术日益成熟和稳定，需求工程却由于没有可现成套用的方法成为一个困难的课题，因此目前软件工程学科的焦点和重心正呈现出逐渐转移到前期需求阶段的趋势。只有通过软件需求分析，才能把用户对软件功能和性能的总体要求描述为具体的软件需求规格说明，从而奠定软件开发的基础。

本章主要介绍软件需求分析的概念、目标与原则、方法及过程、需求分析的工具、统一建模语言（unified modeling language，UML）及软件需求建模，包括传统的软件建模、用例建模、面向对象建模等，最后给出典型应用分析。

3.1 需求分析的目标与原则

软件需求是指用户对目标系统在功能、行为、性能等方面的期望。需求分析是发现、求精、建模和产生规格说明的过程，软件开发人员需要对应用问题及环境进行理解和分析，为问题涉及的信息、功能及行为建立模型。需求分析实际上是对系统的理解与表达的过程，是一种软件工程的活动。

理解的含义就是开发人员充分理解用户的需求，对问题及环境的理解、分析与综合，逐步建立目标系统的模型。通常软件人员和用户一起了解系统的要求，即系统要做什么。

表达的含义是产生规格说明等有关的文档。规格说明就是把分析的结果完全地、精确地表达出来。系统分析员经过调查分析后建立好模型，在这个基础上，逐步形成规格说明。需求规格说明是一个非常重要的文档。

经过软件的需求分析建立起来的模型称为分析模型或需求模型，注意分

析模型实际上是一组模型，是一种目标系统逻辑表示技术，可以用图形描述工具来建模，选定一些图形符号分别表示信息流、加工处理、系统的行为等，还可以用自然语言给出加工说明。

为了理解需求分析，首先必须了解需求分析在整个软件开发中的目标。

3.1.1　需求分析的目标

Bertrand Meyer 在他的著作 *Object-Oriented Software Construction* 中总结了系统需求分析的如下 8 项目标。

A0：决定是否建立一个系统。

A1：理解最终的软件系统应该解决哪些问题。

A2：引出这些问题和系统的一些相关问题。

A3：提供一个与这些问题和系统特征有关的回答问题的基础。

A4：决定系统应该做什么。

A5：决定系统不应该做什么。

A6：确认系统将能够满足用户的需要，并且定义相应的验收标准。

A7：为系统开发提供一个基础。

需求分析的这些目标可由 3 个子阶段完成：可行性分析主要是完成 A0 目标，即要决定是否建立一个系统；需求收集主要完成目标 A1~A6；目标 A7 则由需求规格说明完成。

3.1.2　需求分析的原则

近年来已提出了大量的分析建模方法，虽然各种方法都有独特的描述方法，但在进行软件的需求分析时，都应该遵循一些基本的原则。

1．必须能够表达和理解问题的数据域和功能域

理解和表示问题的信息域，可用数据模型描述；定义软件将完成的功能，可用功能模型描述；表示软件的行为（服务、操作），可用行为模型描述。

软件定义与开发工作的最终目的是解决数据处理问题，就是将一种形式的数据转换成另一种形式的数据。其转换过程必须经历数据输入、加工和产生结果等步骤。对于计算机程序处理的数据，其数据域应包括数据流、数据内容和数据结构。

数据流即数据通过一个系统时的变化方式，如图 3-1 所示。数据结构即各种数据项的逻辑组织。数据是组织成表格，还是组织成有层次的树型结构？在结构中数据项与其他哪些数据项相关？所有数据是在一个数据结构中，还是在几个数据结构中？一个结构中的数据与其他结构中的数据如何联系？这些问题都由数据结构分析

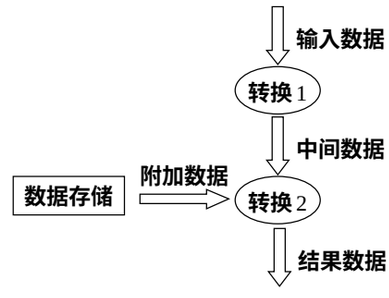


图 3-1　数据流

来解决。

2．必须自顶向下、逐层分解和细化问题

对描述的信息、功能和行为模型必须被划分，使分析模型可以用层次的方法展示细节。划分就是把系统进行分解。一个庞大而又复杂的问题在整体上往往很难被完全理解，为此，人们常常把一个复杂问题分解成若干个子问题，如果问题被分解后，还不足以被理解，又把子问题再进一步分解，直到问题能被完全理解为止。分析过程应该从要素信息移到实现细节。可以采用逐步求精的技术。如图 3-2 所示，问题的分解包括横向分解和纵向分解。

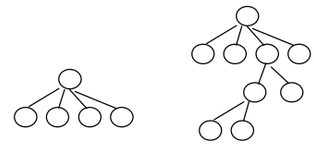


图 3-2　问题的分解

3．必须给出系统的逻辑视图和物理视图

给出系统的逻辑视图（逻辑模型）和物理视图（物理模型），对满足处理需求所提出的逻辑限制条件和系统中其他成分提出的物理限制条件是必不可少的。

软件需求的逻辑视图给出软件要达到的功能和要处理数据之间的关系，而不是实现的细节。例如，一个商店的销售处理系统要从顾客那里获取订单，系统读取订单并不关心订单数据的物理形式和用什么设备读入。类似地，系统检查库存的功能时也只关心库存文件的数据结构，而不关心在计算机中的具体存储方式。软件需求的逻辑描述是软件设计的基础。

软件需求的物理视图给出处理功能和数据结构的实际表示形式，这往往是由设备决定的。如一些软件靠终端键盘输入数据，另一些软件靠模-数转换设备提供数据。分析员必须弄清系统元素对软件的限制，并考虑相应的功能和信息结构的物理表示方式。

3.2　需求分析的过程及方法

3.2.1　需求分析的过程

为了确保分析结果的一致性、全面性、精确性，软件需求分析过程要有用户的参与和积极的协助。就需求分析而言，一个系统的成功与否，开发者与用户双方的合作是非常重要的。需求开发是分析人员对问题及环境的理解、分析与综合，建立目标系统的模型，最后形成软件需求规格说明。需求分析过程如图 3-3 所示。

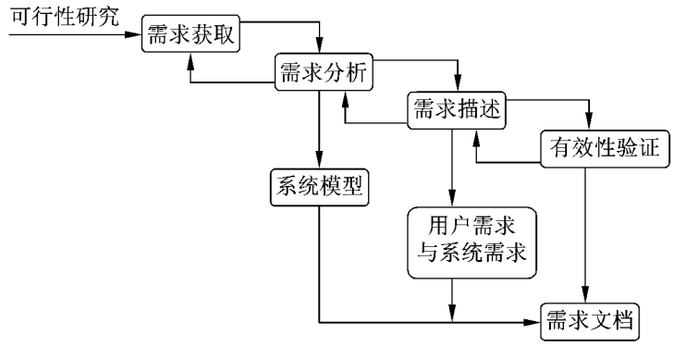


图 3-3　需求分析过程

需求分析工作可以分成以下 4 个过程。

1．问题获取

系统分析人员应该研究计划阶段产生的可行性分析报告（如果存在的话）和软件项目实施计划，主要是从系统的角度来理解软件并评审用于产生计划估算的软件范围是否恰当，确定对目标系统的体系结构和综合要求，即软件的需求，并提出实现这些需求的条件，以及需求应达到的标准，也就是解决将要开发的软件做什么、做到什么程度等问题。具体内容在第 2 章中有详细阐述。

2．需求分析

需求分析包括提炼、分析和审查已收集的需求，以确保所有的风险承担者都明白这些需求的含义，并且找出其中的错误、遗漏或者不足的地方。

在传统的软件工程中，分析员可以从系统的数据流和数据结构出发，逐步细化所有的软件功能，找出系统各元素之间的联系、接口特性和设计上的限制，分析它们是否满足功能要求，是否合理，依据功能需求、性能需求、运行环境需求等，剔除其不合理的部分，增加其需要部分，最终将其综合成系统的解决方案，给出目标系统的详细逻辑模型。分析和综合工作需要反复进行，直到分析员与用户双方都感到有把握正确地制定该软件的规格说明为止。

常用的分析方法有面向数据流的结构化分析方法（SA）、面向数据结构的 Jackson 方法（JSD）、面向对象的分析方法，以及用于建立动态模型的状态迁移图或 Petri 网等。这些方法都采用图文结合的方式，可以直观地描述软件的逻辑模型。

3．需求规格说明

通常，把描述需求的文档叫做软件需求规格说明，分析模型是需求规格说明中的一部分。分析员应以调查分析及分析模型为基础，逐步形成规格说明书。注意，最终确立的分析模型是生成需求规格说明书的基础，也是软件设计和实现的基础。

同时，为了确切表达用户对软件的输入输出要求，还需要制定数据要求说明书及编写初步的用户手册，着重反映被开发软件的用户界面和用户使用的具体要求。

此外，根据在需求分析阶段对系统的进一步分析，从目标系统的精细模型出发，可以更准确地估计所开发项目的成本与进度，从而修改、完善并确定软件开发实施计划。

4．需求评审

需求分析完成后，应该对功能的正确性、完整性和清晰性以及其他需求给予评价。

为保证软件需求定义的质量，评审应指定专人负责，并严格按规程进行。评审结束后应由评审负责人签署评审意见。通常，评审意见中包括一些修改意见，分析人员须按这些修改意见对软件进行修改，待修改完成后还要再评审，直至通过才可进行软件设计。

3.2.2　需求分析方法

需求分析的方法虽然种类繁多，但根据目标系统被分解的方式不同，基本上可以分辨出是属于哪一类广泛应用的方法。20 世纪 70 年代，开发和推出了各种冠以“结构化

分析（structured analysis，SA）”头衔但各具特色的方法。直到 20 世纪 90 年代初，结构化分析方法才面临严峻的挑战，同时，面向对象分析方法（object oriented analysis，OOA）已悄然成型，而且同样也是随之出现了大批派生的方法。如今，对于面向对象分析方法的批评已经开始出现，一类所谓的第三种方法（尚未命名但却是基于问题框架的）正发展成型，称之为面向问题域的分析方法（problem domain oriented analysis，PDOA）。

1．结构化分析

从方法学的角度来看，最伟大的革新莫过于告别了那些基于文本的分析和规格文档而迎来了图形建模表示法的使用。这种基于模型的方法开创了一个至今不衰的先河。这在很大程度上是针对早期那些用于开发冗长、无结构的文档的例行做法所作出的反应。这些冗长、无结构文档的内容相当混杂，其描述的范围包括：问题域、原有系统、种类繁多的需求、不完整的新系统规格说明以及其他一些或多或少与系统有点相关的问题等。

与早期的例行做法相符，结构化分析通常强调对原有系统的建模。这虽然并不完全等同于问题域的研究，但对于信息系统而言，二者间的差别并不大。当时的软件开发的大部分工作耗在了针对那些相当俗套的、文书性的信息系统所做的“计算机化”处理上。事实上，原有的、基于文档的手工系统常常也能够为基于计算机的解决方案提供一个合理的模板。

综合性的结构化分析方法正是通过开发各种“模型”而利用了这一点。结构化分析，使用数据流建模方法，主要工具是数据流图（data flow diagram，DFD），来对问题进行分析。结构化分析一般包括下列工具：

- 数据流图；
- 数据字典；
- 结构化语言；
- 判定树；
- 判定表。

结构化系统分析方法从总体上看是一种强烈依赖数据流图的自顶向下的建模方法，不但是需求分析技术，也是完成规格说明文档的技术手段。在结构化分析中，分析文档与规格说明书二者之间并没有明显的界限。逻辑模型（至少由一组分层的数据流图和相关处理说明组成）虽然也可以称为“规格说明书”，但常常与原有系统的“分析”模型没多大区别。可见，结构化规格说明理应得到足够的重视。

DFD 模型也可以看作是一种行为模型。从理论上讲，这是可能的，并且假如只是涉及那些“外部可见”的功能，DFD 可以用于为某一真正的功能分解建立文档。而实际上，不仅其中掺杂有内部功能，而且功能分解本身也已成了内部结构的重要依据。

所以，结构化分析在相当程度上尚未对需求以及满足需求的方法二者加以区分。需求的说明实际上只是根据满足这些需求的某一特定系统的设计而做出的。新系统中除了最底层的内部设计外都已被从内部设计阶段“强行掳走”并归并到所谓的规格说明中去了。

（1）数据流模型

数据流图（DFD）是以图形的方式表达数据处理系统中信息的变换和传递过程，有 4 种基本符号：

→ 一个命名的向量表示数据流，箭头的始点和终点分别表示数据流的源和目标；

□ 用方框表示数据源（终点）；

○ 用圆形表示对数据的加工（处理）；

[或= 用一端开口的长方形表示数据的存储。

数据流是一组成分已知的信息包。这信息包中可以有一个或多个已知的信息。两个加工间可以有好几个数据流。数据流应有良好的命名，不仅是作为数据的标识，而且有利于深化对系统的认识。同一数据流可以流向不同加工，不同加工可以流出相同的数据流。流入/流出简单存储的数据流不需要命名。数据流不代表控制流。数据流反映了处理的对象。控制流是一种选择或用来影响加工的性质，而不是对控制流进行加工的对象。

数据流模型（也记为 DFD）虽然可以标识数据但对于数据的定义的作用甚微。为此，数据流模型从一开始就提供了刻画数据流本身性质和结构的描述列表。这类列表常被称作数据字典（data dictionary，DD）。尽管数据字典主要侧重于数据的语法，但是也可用于说明数据所代表的含义。因此，数据字典可以说是相当有用的。当然，不足的是数据字典并不总是很好地反映系统数据的总体结构。

20 世纪 70 年代后期，Chen（1976）通过增加数据结构模型（见本书 3.4.2 节）以增强结构化分析的适用性，从而使得上述不足以得到解决，其中数据结构模型通常以实体关系图的形式出现。而对于信息系统而言，数据结构模型使得对于问题域的认识更加深刻。

然而，数据模型（ERD）与处理模型（DFD）之间的关系却并非是一个简单的关系，并且经证明将二者合二为一大有问题。从需求工程的角度来看，数据模型的非过程式特性或许被认为是其得天独厚的一大优势，但同时这些特性也太不易于转换为可执行代码，这一事实经常反映在以下现象，即每当后续的结构化设计阶段开始之时，数据结构模型便随即销声匿迹。

（2）结构化分析的演化

“结构化系统分析与设计方法学”（structured systems analysis and design methodology，SSADM）（见 CCTA, 1995，或 Ashworth 和 Slater, 1993）圆满地解决了这最后的一道难题。该方法引入了实体生命历史（entity life history，ELH）的概念，把各种实体的状态变化与负责获取这些变化的处理联系起来，从而在过程式模型与数据模型之间搭建了一座桥梁。与此同时，还定义了在问题域中所允许出现的合法的事件序列，这对于某些类型的问题是一项重要的举措。

SSADM 同样也解决了长期以来存在的一个问题，即对形成清晰的需求的忽视。在其他一些相关的阐述中，SSADM 引入了问题需求列表（PRL）。这是一种基于文本的、本质上属于无结构的（当前系统的）问题列表和需求（由新系统予以满足）。

而后出现的实时结构化分析（real-time structured analysis，RTSA），提出了便于针对实时系统（可参见 Ward and Mellor，1985）建模的论述。这包括：扩展数据流图的基本符号以适应控制数据（即由数据触发、启动或禁止处理的执行）以及定时的要求。引入实体生命建模法（又名实体状态建模，与 ELH 很类似）以帮助对当前系统的状态变化的建模。

在实时系统内部过程的建模开始之际，这些扩展的效用是毋庸置疑的，并且在对实

时问题域的建模方面，同样被证明是有效的。然而，根据结构化分析的规则，所讨论的系统接下来就是原有系统，或之后新的解系统。因此，可以这么说，这些扩展的作用充其量只是局限在与需求工程对立的内部设计的圈子里而已。

（3）现代结构化分析

虽然结构化分析取得了一定的进展，但到了 20 世纪 80 年代后期，一些主要的业界人士开始认识到结构化分析正陷入困境。其中最突出的问题就是所谓的“分析抑制”现象。曾一度是“增强型”的方法，遭遇上规模日渐增大的亟待解决的问题，结果却是导致许多项目在原有系统的建模上举步维艰。

一大批文档也相继诞生，其中大部分是关于原有系统的详细处理的建模。是否该对原有解系统的处理做深入细致的研究？这一点着实令人为难；若是问题域的处理，回答肯定是当然，若非，则说明原有系统的功能与问题域的处理或许不是一回事。

大量文档有时未能得到利用，这也许就是问题的症结所在。为了生成文档，投入了大量的精力与开销，但随后却被轻易地抛弃了。

除了那些大型、复杂的系统的建模问题之外，还出现了“非存在”系统的建模问题。随着软件开发正进入一个新领域，某些原有系统存在与否已不再被人们所关注，即使存在，也主要体现在新系统的需求或设计方面而已。什么样的系统，比如说，能够比基于计算机的引擎管理系统更优先呢？再有，一个人能利用研究别人使用打字机而了解多少有关字处理应用程序呢？

有趣的是，对于这两类问题居然也有一个解决方案；这就是，根本不必理会原有系统，直接开始新系统的建模。正如 Ed Yourdon 所建议的：

系统分析员应尽可能避免建模用户的当前系统。在本书第二部分所讨论的建模工具应尽快用于开发用户真正想要的新系统的模型。

（Yourdon，1989a，p.323；my italic）

在某种程度上，这就等于是承认做了错事，但停止做错事并不等同于开始做正确无误的事情。因此，有人自然而然地提议，新系统的模型应尽可能地抽象，不带任何实现细节；然而实际的情况却是，模型通常就是新系统的内部设计赖以存在的基础。进一步讲，恰恰是设计从根本上实现了功能分解，所以，一方面这也许是一个开发功能规格说明的绝佳的机制，而另一方面能否由此得到令人满意的结构化设计就不得而知了。

坐享“后见之明”其成，不难发现，一个更基本的缺陷就摆在眼前，即分析的主要（战略）目的——研究问题域——很大程度上一直被淡忘了。

2．面向对象分析

面向对象方法的产生可以追溯到 20 世纪 60 年代后期，那时第一个面向对象的编程语言诞生了。在初期，面向对象方法只是一种为系统的结构进行建模的方式，然而不久，面向对象方法就从编程扩展到内部设计，并被证明是一种非常有用的方法。

面向对象方法扩展到分析阶段只是近年来的事情，而且仍处于不断的发展之中。其基本思想是：如果仅仅是着重于对来自问题域的对象类进行建模，那么面向对象的基本原理、模型以及表示法均可应用于分析。然而，正如 Jackson（1995）所强调的，事情似乎并不像预料中的那样顺利。假如建模技术间存在共性以及某些对象类表面上看起来对于两个领域是公共的，那么分析与内部设计之间的界线可能就成为一条难以觉察的细线

了。因此，在 OOA 中，术语“分析”的意义可能会与本书中所描述的含义不一样。“调查问题域”的一般意义通常存在一些重叠交叉之处，而且似乎更多地涉及解系统高层的内部设计（体系结构设计），而较少涉及问题域的分析。

Jacobson 则对这一点十分清楚，认为面向对象软件工程（OOSE）可能在需求规格说明存在的那一刻起启用，而隐含的假设是将分析以及规格说明作为面向对象“分析”的前身。这样会带来术语上的混淆，但不会造成太大的问题。然而，其他部分却相当难办，同时留下一个疑问，归根到底，面向对象体系结构能够取代什么？究竟什么可以称为“真正的”问题域分析？

有意思的是，在这方面，OOA 有几点特性与 SA 是相同的：

- 主要的模型是一个结构模型（与行为模型相对立）；
- 尽管存在完全相反的观点，而实际上，焦点通常都集中在对解系统的建模（而不是问题域）上；
- 大多数文献都倾向于强调表示法的细节（而不是基本原理）；
- 隐含地假设需求获取行为的发生，但大多数文献很少提及；
- 分析与规格说明（或就此而言，内部设计）之间没有明显差异；
- 所有问题域均服从于类似的处理。

根据需求工程的观点，这些特性均可以被认为是严重的缺陷。但是在许多组织里，OOA 已经取代了 SA，同时 OOA 也已成功地应用在一些重要的开发上。这也许是因为前期工作已由其他方法顺利完成，或者是由于一些其他因素（如原有问题域知识）允许“走捷径”。

正如前面提到的，一般而言，OOA 中有关需求获取的部分很少提及，因此按以前曾提到的方法假定这些曾发生过。

OOA 的大致方法是：

- 标识出问题域中的对象类；
- 定义这些类的属性和方法；
- 定义这些类的行为；
- 对这些类间的关系建模。

后续步骤随后通过添加与解系统的行为及实现相关的类对模型加以扩展。这种对同一模型做渐进式精雕细刻的做法导致了“无缝”开发概念的产生，即在整个开发阶段，维护相同的概念模型和表示法。这一概念目前仍受到青睐，但也有人认为它没有任何有说服力的理由（Jackson，2001）。但无论如何，完全有理由相信，这一概念对于存在于问题域的结构模型，解系统的行为模型和解系统的结构模型之间的“缝隙”是非常合适的。

根据之前所述，出现不同版本的类模型之间的差别经常被掩盖的现象并不足为奇。然而，有些人却认识到这些差别的重要性。如，Fowler 和 Scott（2000）突出强调了类模型的 3 个“方面”：

- 概念化（即问题域类—属于分析）；
- 规格说明（即接口类）；
- 实现（即属于内部设计）。

显然第一点与这里有关。但必须注意的是，即使一个类是来源于现实世界，在问题域对象以及任何与之同名的软件表示体之间仍存在着本质的区别。第二点在规格说明一章中作简要的回顾，而最后一点更适于将其视作面向对象设计，因此也不作进一步讨论。然而，人们普遍缺乏对其区别的认识，这一点读者应心里有数。

就在前不久，方法中新增了一个等待步骤，即使用用例来帮助建立需求。而今到处都在宣扬，在早期阶段，用例应尽早用于帮助获取和记录需求。然而，也应该认识到，用例记录的只是解系统的功能（而不是需求），因此其作用更多地体现在规格说明部分而不是决定需求。该方法创建了一个捷径，因而避开了对问题域及其相关的问题做全面考虑，并直接对解系统的行为进行定义。这可比直接进行解系统的体系结构的设计要强得多，只要对问题域有了清楚的了解，那么这会是一个能够正常工作的捷径。

3．面向问题域的分析

面向问题域的分析（PDOA）是一项很新的技术，人们对于 PDOA 尚没有全面的了解，并且文档资料也不多。据说，PDOA 仅仅在部分内容上是全新的。

与 SA 或 OOA 相比，PDOA 更多地强调描述，而较少强调建模。只要是合适的场合，该描述即可结合使用那些在以前提到过的建模技术，但通常没有这样做，直接靠文本来完成。

描述大致划分为两个部分：一部分关注于问题域；而另一部分关注解系统的待求行为。同时建议有两个单独的文档：第一个文档含有对问题域相关部分的描述以及一个需在该域中求解的问题列表（即需求）；第二个文档（规格说明书）包含的是对解系统的待求行为的描述以解决需求。其中只有第一个文档才是通过分析产生的；第二个文档推迟到后续的规格说明任务。

问题框架作为一个全新的模型被引入（参见 Jackson，1995；Kovitz，1999 及 Jackson，2001），该模型不仅有助于把需求从问题域的内在性质中区分出来，还有助于建立问题域的类型。PDOA 并非对所有的问题域均一视同仁。根据问题域类型，分析者被指导去收集和记录不同的信息。然而，整个方法过程的基本步骤可以定义得相当好：

- 搜集基本的信息并开发问题框架，以建立问题域的类型；
- 在问题框架类型的指导下，进一步搜集详细信息并给出一个问题域相关特性的描述；
- 基于以上两点，收集并用文档说明新系统的需求。

对于第二步至关重要的是指导原则，由 Kovitz（1999）提出，可以针对每种类型的问题域，列出那些有待探查和文档化说明的问题域的各元素。

1）问题框架

问题框架是将问题域建模成一系列相互关联的子域，而一个子域（也常简称做“域”）可以是那些可能算是精选出来的问题域的任一部分。

也可将问题框架视作是开发上下文图，但其侧重点不同：

- 上下文图建模对象是解系统上下文；
- 问题框架建模对象是问题上下文。

基于这一目的，与上下文图相比，问题框架的目标就是大量地捕获更多的有关问题域的信息。上下文图展示的只是问题域的元素，这些元素对于新的解系统是外部可见的，

并且直接与之相连接。这些元素通常又称做端子，因为由它们最终完成新系统的输入与输出数据流。然而，端子并不一定就是惟一适当的子域，而如果这样认为，一些重要子域可能会因此被忽略掉。

上下文图也会省略一些子域间的关系。例如，定义一个问题子域与另一问题子域之间关系的规则根本未出现在一个上下文图中（按照约定，端子之间没有关系需要展示），但实际上却对应必须予以满足的真正需求。

将问题框架应用于软件开发问题只是最近才有的事，这或许会令人感到不可思议，而这却是事实。并且需要强调的是，问题框架的应用尚处于早期阶段，且仍在迅速发展之中。尽管如此，已有迹象表明，问题框架的应用有着重要的优点，并且可以展望，随着技术的成熟，必然会带来更多优点。

（1）问题框架类型

PDOA 与其他分析方法的最重要的区别之一在于对问题域的分类方式，基于该分类标准，分门别类做不同的处理。了解问题的类型有益于正确指导分析（通过指出哪些问题需要询问，哪些方面需要建模等）以及规格说明（也就是最终的内部设计）。

Jackson（1995，2001）提出了一种远比早期所尝试的那些方法要客观得多的分类法，该分类法基于不同问题子域的本质及存在于问题子域间的关系。下面先作简要介绍。

- 工件系统——系统必须完成针对只存在于系统中的这些对象的直接操作。
- 控制系统——系统控制部分问题域的行为。这里可以有两种变种：待求行为框架（待求行为完全由规则预先确定），以及受控行为框架（行为的控制取决于操作员发出的命令）。
- 信息系统——系统将提供有关的问题域的信息，这里也有两种变化：信息是自动提供的（通常是持续的），以及信息只在响应具体的请求时提供。
- 转换系统——系统必须将某种特定格式的输入数据转换成相应的、另一种特定格式的输出数据。
- 连接系统——系统必须维持那些相互没有直接连接的子域间的通信。

上述这些问题类型的每一种随即拥有一个典型问题框架，但在对这些不同的问题框架做详细分析之前，先来看看该方法是如何在一个特定的样例系统，即 Petri 网图表处理工具上运作的。

上下文图（图 3-4）除了展示其中有一个用户之外并没有什么更多的内容。该 Petri 网图表处理工具可以轻易地将其确定为一个工件问题。相应的问题框架（图 3-5）不仅满足了 Petri 网文档的要求，而且也使得那些规则一目了然，这些规则定义用户所执行的绘制 Petri 网操作的结果。下面先对有关表示法做些解释。

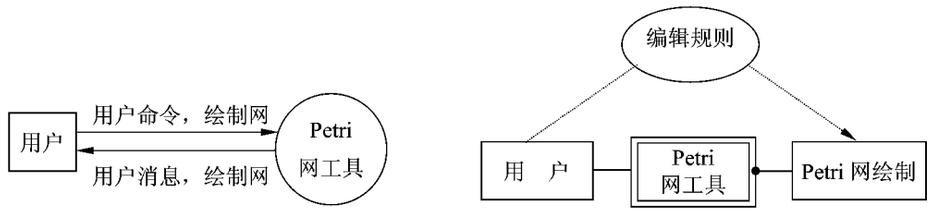


图 3-4　上下文图

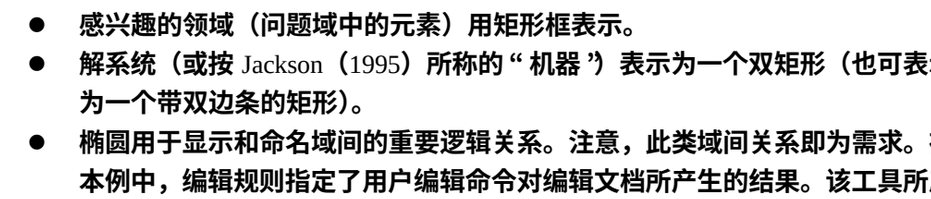


图 3-5　问题框架

- 感兴趣的领域（问题域中的元素）用矩形框表示。
- 解系统（或按 Jackson（1995）所称的“机器”）表示为一个双矩形（也可表示为一个带双边条的矩形）。
- 椭圆用于显示和命名域间的重要逻辑关系。注意，此类域间关系即为需求。在本例中，编辑规则指定了用户编辑命令对编辑文档所产生的结果。该工具所产生的结果其实就是需求。
- 连接域的实线指出这些域之间存在着某种关系。对于软件系统，完全可以使用数据流，但任何一种交互都适用。关键的判别标准在于域是否有共享的迹象；即域之间是否存在任何可以共享的价值或事件。
- 虚线代表一个需求的引用；也就是说在被引用的子域中，存在涉及现象的需求。与只是简单地加以引用相反，箭头指示的是需求在子域范围内对现象加以限制。在上面的例子中，需求将（从用户那里）引用编辑命令，并且规定了这些命令将对文档产生什么样的影响。
- 最后，大的实心点用于指出，一个域包含在另一个域当中。在本例中，Petri 网图包含在（且仅包含在）工具中。

该方法最大的优点是，一旦识别了问题类型，即可根据指示对问题的相关方面做调查和记录。表 3-1 和表 3-2（基于 Kovitz 的内容表，Kovitz，1999）特意指出了有关工件问题的需求文档和规格说明书的内容。正好，对于工件问题，这些列表虽然简短但却非常值得作一番了解。

表 3-1　需求文档

需求文档
工件的合法数据结构
所需操作和它们对工件所应产生的影响

表 3-2　规格说明书

规格说明书
用户接口以及操作过程

需要提醒的是，这些表格因问题类型的不同而变化很大，另外指出什么该忽略与指出什么该包含二者是同等有用的。例如，对于一个工件问题，试图去识别出工件子域的内在行为是毫无意义的，因为按照定义，并不表现任何行为。类似地，对于一个转换问题，试图去标识出问题域事件是毫无意义的，因为按照定义，根本就没有问题域事件；当然，不管怎样，定义输入输出数据集映射始终是至关重要的。

（2）问题域、需求及语态

与其他方法不同，PDOA 清楚地区分问题域的内在特性以及在问题域中要求产生的变更。Jackson（1995）将其刻画为语态（在语法意义下）的区别。关于问题域内在质量的声明是在指示语态下做出的。以下是摘自案例研究的一个例子：

当电梯在传感器额定位置的垂直方向（之上或之下）20cm 范围内，传感器发出一个 hi 信号，否则发出一个 lo 信号。

以上是对问题域的真实反映，完全超出了所要构建的新的解系统能够控制或影响的范围。

另一方面，也确实有一些新系统能够控制的东西，即要求产生的结果。对这些结果的声明是祈使语态下做出的（反映了有客户选定的选项），以下是一些例子：

电梯不应当在快速模式时停止，而应在停止前至少 1 秒钟切换至慢速模式。
只有当电梯停于某一楼层时才能改变其方向。

只要给出问题域的内在特征，新系统完全有能力满足这些需求。

（3）子域交互

正如在上文提到的，连接各子域的线段代表了它们之间的关系或交互作用；而这些都是以共享或引用现象的形式出现。全面描述这些交互作用是非常可能的，并且当涉及为解系统派生一个适合的逻辑行为的时候，能够提供很大的优势。尽管其意义重要，但令人可惜的是，该项议题过于庞大，这里很难对其做详尽研究，有兴趣的读者可参考 Jackson（2000）。

（4）子域类型和问题框架调整

正确识别问题框架类型以适应问题的做法虽然易于理解，但同时也带来处理上的困难。一个有效策略就是利用问题框架施加于子域特性上的限制。例如，工件框架要求工件本身必须是动态、惰性的且包含于机器之中（也就是说，必须是一个无形的软件文档）。如果假设的工件子域并不适应该模式，那么说明并没有工件问题。

可从考虑一个域是否随时间而变化这一点来定义各种子域类型，其中大多数域在多数情况下被描述为动态的，与此相对肯定也有部分是静态的。

动态域可以进一步分类为能够修改自身（自修改）的域和仅由外部机构修改的域两种。后者也称为是惰性的，而在软件系统领域，典型的例子就是文件。

自修改域根据在修改时是否要求有外部激励的参与还可以进一步进行划分。例如，一个抽象数据类型（ADT）可能仅仅是响应一个外部刺激或请求而改变其自身状态，而这种类型的域也被称为是反应性的。

其他一些域可能会自主地改变，而对于这些域，必须考虑如何使得它们的行为可控或可以预知。如果是完全可预知（即使只是推测性的可预知）的，那么该域就被称为是可编程的，同时这当然也是大多数（不是全部）软件应用的基本特性。如果一个域的行为是部分可预知的，那么，就可以被称为是顺从的，而人类用户就是这类域的典型例子。例如，用户可能被要求输入口令，而用户可能会输入，但这并没有明确的保证。另外，一个域也可以是完全不受控制的，而这些域则被称为是“自治的”。这方面的例子可以是某一股票市场指数。

图 3-6 对这些考虑因素进行了总结。

与以上考虑因素相对应，域也可以归类为有形的或无形的两大类。有形域是那些具有物理存在的域，诸如人类用户以及硬件，无形域则包括软件以及其他非物质的对象如规则集成或以前给出的百科全书的例子。

域一旦具有了特征，那么就可以很容易地检查域是否履行了由选取的框架所施加的需求。继而给出那些有关框架的基本描述，其中包括每一个子域的特征。

（5）工件框架

先前曾举过一个特定的例子，即 Petri 网图表处理工具，而图 3-7 所示的是该例的总

体框架。

这表明工件（典型情形是某种形式的文档）包含在机器中，因此只有机器才能够对其产生作用。机器（也就是需要构建并配置了相应软件的计算机）对接收的请求进行响应，从而完成不同的工件操作。请求的来源可能是某一人类用户，当然也可能是另一部机器，对工件的操作请求的影响由操作性质定义。

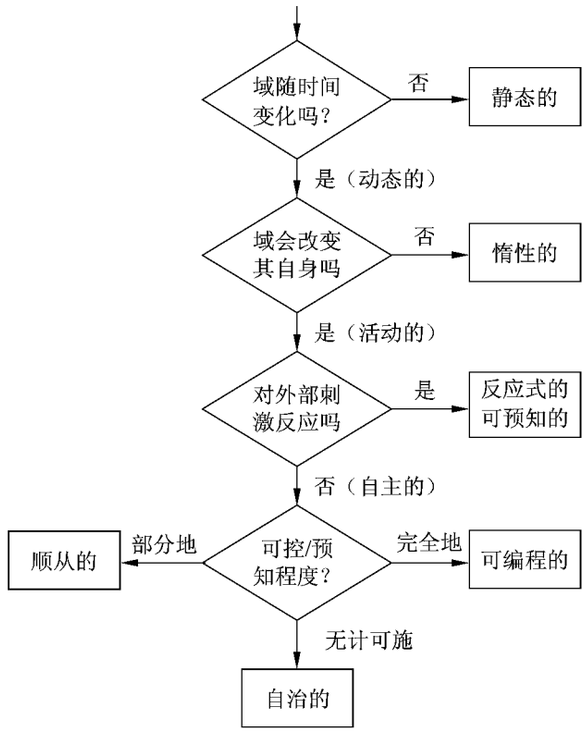


图 3-6　域类型

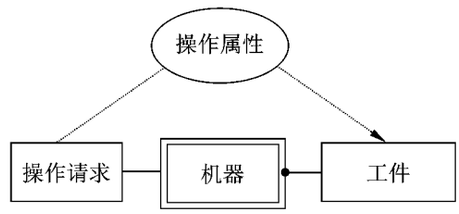


图 3-7　总体框架

由此可见，子域具有以下一些特征：

- 工件是动态的，但也是惰性的（会改变但却必须被动地改变），同时也是无形的。
- 操作请求（通常指用户）也是动态的，但具有本能的主动性，并且是顺从的。机器也可向用户提供若干确定选项，并且虽然一般都能正常地响应请求，但也有可能不响应。
- 机器本身也是动态的、主动的，因而当然也是可以编程的。

- 工件本身组成了一个“现实化的”域；工件在机器内部创建（且只能存在于机器中）。而信息系统也有可能包含某个外部现实的模型，工件是客观存在的。

工件问题绝不少见，包括：

- 绘图工具；
- 计算机辅助软件工程（CASE）工具；
- 许多的办公实用程序，如文字处理器和电子表格；
- 桌面出版以及网站开发工具。

一旦问题与某一特定框架相适配，即可致力于研究 Kotivz 表的内容以获得相关指南，即哪些信息必须获取并记录于需求文档以及规格说明书中。将在下一章再来讨论规格说明书方面的问题，表 3-3 所示的是用于工件问题的需求文档内容。

表 3-3　需求文档

工件问题——需求文档	
内　　容	（某些）相关技术
工件的合法数据结构 操作属性（事件响应），即所需操作和它们应对工件产生的影响	BNF，文件映像，结构图 有限状态机（FSM），文本，决策表，用例

需求文档包含了问题域的描述以及需求本身。在使用问题框架时，需求被表示为图中的椭圆部分。因此，在该例中，由操作属性（事件响应）形成需求，而其余部分则是问题域的描述部分。

如何将这些方方面面的信息正确地用文档说明是另外一个问题。总是可以使用文本式的描述，但一些建模技术也常常带来便利，在上述的表格中就指出了一些相关的建模技术。本书第二部分会详细介绍这些技术，而在本章的稍后部分会有若干例子说明这些建模技术的应用方法。

（6）控制框架

控制框架应用在系统或机器依照某一指定的行为规则集对问题域的某个组成部分的行为实施控制这样的一些场合。按这一主题，控制框架又可以分为两类变种，即待求行为框架和稍微复杂的受控行为框架。

① 待求行为框架

在该变种内部，被操控系统的待求行为完全由一组预先确定的行为规则来定义。其框架图如图 3-8 所示。

图 3-8　框架图

受控域并不一定是单一部件，可以由若干子域组成，但必须表现出某种行为（否则就无需被操控了）并且不能是自治的（因为按照定义，这算是不可控制的）。除此之外，也可能是反应性的、可编程的或顺从的。

② 受控行为框架

在该变种内部，待求行为由用户通过命令方式加以控制而不是完全由预先确定的规则来决定。用户则被假定为一个本能的自治域，该域可以随意发布命令也可以响应系统的提示，是顺从的，但该部分在本框架内不予考虑。框架图如图 3-9 所示。行为规则（由椭圆形表示）限制操作员所发布的命令并且定义被操控系统的最终行为。

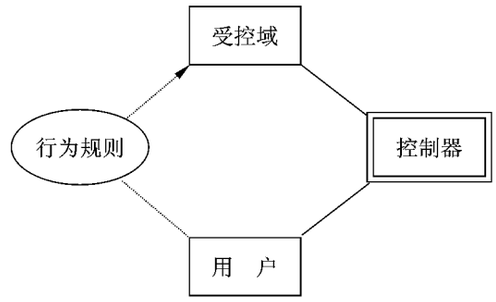


图 3-9 框架图

控制问题也是比较常见的，例如：

- 电梯（升降机）控制系统；
- 用于现代汽车工业的引擎管理系统；
- 用于温室环境控制的系统；
- 安全或火警系统。

上述的前两者（第三个或许也可以算上），都是受控行为系统。最后一个应该是（缺少更多的详情）一个待求行为系统。这样决策并不总是一目了然。

无论哪种情况，都可以从 Kovitz 表中获得相关指示，明确哪些信息必须被获取并且记录到需求文档（参见表 3-4）中。对相关方面的信息建档的方法将在以后的章节中做详细的分析研究。

表 3-4 需求文档

控制问题——需求文档	
内 容	（部分）相关技术
受控域中相关子域的数据模型，如果有的话（注意该内容是可选的）	ERD 和 DD
受控域中每个子域的特征及其内在行为，包括因果定律以及由子域完成/履行的动作/事件	文本，ELH，FSM，决策表
共享现象，解系统通过该现象能够对受控域进行监视	文本（事件列表）
受控域中的动作，解系统可以对其进行初始化	文本（事件列表）
由任何连接域（因太微不足道，毋须单独建档）引入的失真和延迟	文本
行为规则（也就是说，受控域作为一个整体如何表现行为）以及受控行为的合法命令	文本，FSM，决策表

而此时，行为规则恰恰构成了需求——除此之外就是问题域的描述了。

还要注意的是，虽然受控域常常是有形的，但并非需要是这样。很有可能会有某个系统本身又控制着另一个虚拟机器，如计算机网络交换机或电话交换机。

（7）信息系统框架

信息系统是为了能够提供有关问题域中某个组成部分的信息而存在的。信息系统在当前已极为常见，虽然如此，在过去，人们却一直未能透彻地了解它。关于信息系统框

架有两个变种，第一个变种稍为简单，是指系统自动地（常常也是连续地）供应信息的情形。该问题框架如图 3-10 所示。由图中可以看到，根据某个已定义的信息功能，信息系统能够提供有关“现实世界”某个组成部分的信息（报告）。

在第二个变种（图 3-11）中，信息以响应具体请求的方式提供。由图 3-11 可以看出，信息系统根据某个已定义的信息功能，利用关于“现实世界”某个组成部分的信息，对信息请求提供应答。

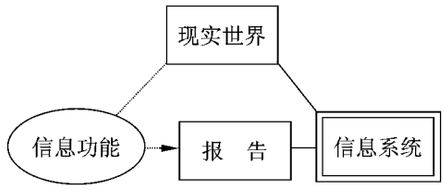


图 3-10　信息系统框架一

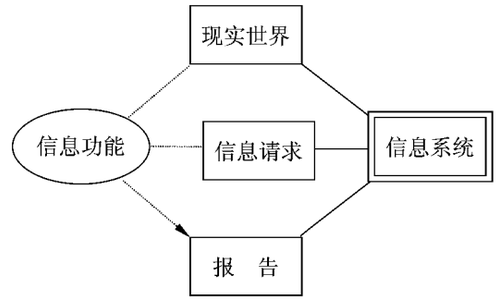


图 3-11　信息系统框架二

按照定义，信息系统主要处理数据，因此，无论怎样，“现实世界”中人们所关心的部分却是由实体所组成的，可以用数据模型来表示（经常采用 ERD 的形式），也可以显示在“现实世界”子域框内，进一步，可以推出这个问题域右边的线段（表示关系）实际上是单向的数据流，而左端的线段则代表逻辑关系。考虑将这些因素加以综合即得到所示的图 3-12。

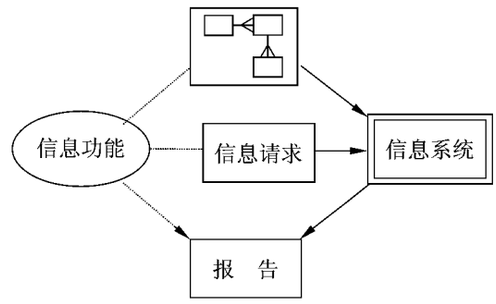


图 3-12　信息系统框架（子域框）

正如刚才所强调的，对于由请求驱动的信息系统而言，有两种类型截然不同的输入：

- 问题域状态的更新；
- 问题域有关信息的请求。

虽然在问题框架图中分别表示为两种不同的线段，但这也意味着没有关于这些数据来源的说明。当然也完全存在着这样的情形，即二者来自同一数据源（如在赛艇比赛成绩案例中，用户会输入一些有关赛艇、比赛等方面的详细资料，而同时也会请求获得比赛成绩的输出）。

问题域的“现实世界”部分有可能是一个静态域。虽然如此，但正如在前面所指出的，所要描述的域常常是动态的，甚至更重要的是原本就是自治的。而且应当注意与控制系统相反，信息系统并不对问题域（仅对其所包含的所有模型）施加控制。

问题域的相关部分有时是无形的，并且存在于与解系统相同的机器之中，从而使机器有可能直接存取现实实体并报告它的状态。

更多情况下，信息系统无法直接存取问题域的相关部分，因此信息系统通常都会包含一个问题域相关部分的模型并使用该模型作为直接的信息源。显然这就必须要有一个

机制，由它来维持问题域与问题域模型间的通信。对信息系统问题框架做进一步的详细阐述（图 3-13）可以有助于强调这一重点。

然而更新机制就不是那么简明易懂了。也许将其显式地建模为一个连接域是最佳的选择。

有关信息系统需求文档内容的一些指导性原则如表 3-5 所示，在该例中，信息功能表示的是需求，而其他部分则是问题域描述。

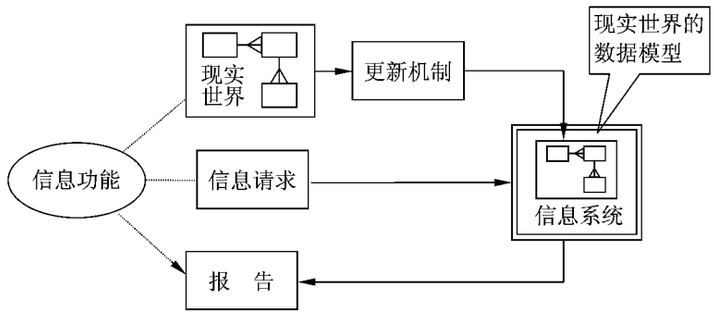


图 3-13　信息系统问题框架

表 3-5　需求文档内容的指导性原则

信息问题——需求文档	
内　　容	**（一些）相关技术**
有关子域（实体）的数据模型、子域属性那些必须予以报告的问题域（报告内容包括其属性和关系）	ERD 和 DD
每个问题子域的特征，包括所有改变问题域状态的问题域事件（因此也包括查询的结果）以及这些事件发生所有可能的序列	文本，事件列表和 ELH
系统如何存取问题域当中的相关子域的状态和事件（或者说，对于某个静态的信息系统，软件开发者如何进行存取）	文本
由任何连接域（太微不足道而无需单独建档）所引入的失真和延时	文本
其初始数据可以摘自原有的文件、相关格式以及存取规程（希望可以有原有的文档供引用）	文件映像，结构图，BNF
信息功能也就是所需的报告和与现实世界的状态之间的关系，以及任何相关的系统支持的查询	绘图，文本，表格

除了以上这些，还有一个较常见的准则，就是在解系统内部包含一个关于问题域的存储数据模型。虽说是常见，但对此多少还存在些争议，所以说，解系统数据模型的开发完全可以推迟至规格说明阶段，甚至内部设计阶段。

有时在信息问题与工件问题之间可能会产生一些混淆，以一个基于机器的电话簿为例（就像通常存储在移动电话里的那种）。所存储的关于人员和他们的电话号码的数据是某一现实世界域的模型（毕竟，该数据代表着真实的人和真实的电话号码），或者只是一个存储于机器内部且可以由用户进行编辑的工件文档（一个现实化的域）。

虽然二者之间的差别看上去似乎并不明显，但却可能会是至关重要的。这或许不会在本例中出现（可是现实与模型之间的任何不匹配都将导出错误的或者是无法获取的号码），但目前通行的做法却是采用现实化的财务域。早期计算机化的财务系统提供了一种存储和操纵数据的方法，但只是相关的纸上系统的模型。纸才是（如发票和支票）真正的、合法装订的文档。但时过境迁，许多组织现在已经认同 Oracle 为计算机化的记录，过去那些通常被认为是信息问题的，现已被工件问题取而代之。

（8）转换问题框架

在图 3-14 中所示的这类框架对转换问题进行建模，即系统将某一特定格式的输入数据转换为与之对应的、另一种特定格式的输出数据。当然，机器并不改变任何输入数据，也就是说，要求输入数据域必须是静态的。输出数据域的改变只能通过机器来实现，是惰性的。

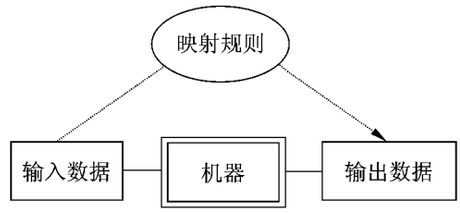


图 3-14　问题框架

由机器来直接存取也意味着输入与输出域二者均是无形的，然而虽然这只是最简单的情况，但如果能提出适当的连接域，同样可以适用于那些有形的输入域。

在表 3-6 中给出了适用于转换系统的需求文档内容指导性原则。其中输入与输出之间的映射代表需求，其余部分是问题域描述。

表 3-6　需求文档内容

转换问题——需求文档	
内　　容	（一些）相关技术
输入与输出数据集	DD（BNF），结构图
数据来源与去向	文本
输入与输出间的映射	文本，映射表，结构图

转换框架也称做 JSP 框架，这是由于 Michael Jackson（1975）发明了一种解决这类问题的被称作 Jackson 结构化程序设计（JSP）的方法（一种处理内部设计的方法）。

JSP 解决方法对输入域和输出域的性质设立了较为苛刻的前提条件。本质上，二者都必须是顺序的，而且可以由正规表达式定义。映射规则必须相对地简单易懂，而设立的条件则可以避免某些问题的发生（如在上面提到的 OCR 问题）。当然，只有 JSP 解决方法才会受到影响，而这也不是当前迫切需要关心的，因为还有需求获取以及文档化的指导原则可以有效地解决同类问题。

（9）连接框架

连接框架应用在必须维持那些相互间没有直接连接的子域间通信的场合。如果两个域之间总是存在一个直接连接（也就是所说的共享现象），事情应该有所简化，然而这常常是绝无可能的。

即使如此，只要容忍连接域所带来的某些失真，那么就可以忽略该连接域，但是假如做不到这一点（而且再小心谨慎也总有犯错的时候），就应该对连接域明确地做一番研究。首先必须定义共享现象的本质属性以及连通域之间可实现的通信。

通常，这样的问题往往出现在某个较大型问题的某些部分，如信息系统需要搜集有关问题域的信息，或者某个控制系统必须与其所控制的部分相连接。因此，经常出现这样的情形，即机器的一端与另一个必须构建的机器相连接。

目前至少存在两种版本的连接问题，而第一种的应用是，一个需求工程师在对连接域进行外部设计时有了一定的选择余地。一个典型例子就是，信息系统依赖于用户的输入以维持它的问题域视图。HMI 也就是必须要生成的连接机器。图 3-15 对该问题的框

架做了说明。缩写 RC 和 CM 分别表示现实与连接域间共享现象以及连接域与机器间的共享现象。（这些共享现象总是存在于相互作用的域之间，但通常没有明确地予以显示）。

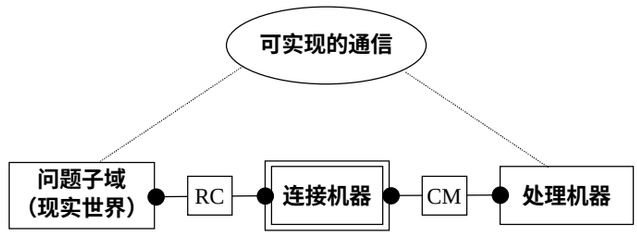


图 3-15　连接域

另一种场景是连接域是给定的（如传感器或传动器）、且机器必须予以容纳的情形。该问题的建模如图 3-16 所示。

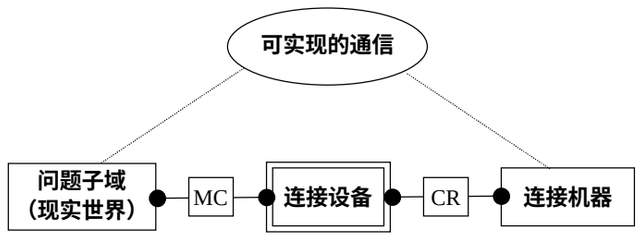


图 3-16　连接域

在该场景中，问题域的相关部分以及连接设备都是有形、自治的域，而机器则是可编程的。然而鉴于连接问题多少带有点变数，因此，必须灵活地对待所列检查清单（表 3-7）。在这里，期望的映射表示的是需求，其余的则是问题域描述。

表 3-7　连接问题

连接问题——需求文档	
内　　容	（一些）相关技术
问题域中相关的状态和事件	事件列表，FSM
问题域数据中的冗余（如果有），以及在有冗余的地方，用于确定最可靠数据的规则	文本，决策表
由任何已有的连接设备所引入的信息映射（包括任何失真以及延时）	并发 FSM，文本，映射表，决策表
期望的映射，在连接的域之间发挥作用	并发 FSM，文本，映射表，决策表

2）待求子域性质小结

为了完全适合相关问题框架，关键的子域必须体现某些性质，表 3-8 对这些性质做了总结。其中一行中存在多个项表明有多个可供选择的选项。

3）多框架问题

现实生活中许多应用域会形成一些复杂的问题，只是将它们作为简单的问题来对待

不仅不切实际而且很可能行不通。解决复杂问题的关键在于将复杂的问题分割或者分解为若干较简单的问题，而每个单独的简单问题更易于解决。这一做法也称为“分而治之”策略。

表 3-8　子域性质

框架	子域	静态的	惰性的	反应的	可编程的	顺从的	自治的
工作	操作请求					¡Ì	
	工作		¡Ì				
控制	受操控域			¡Ì	¡Ì	¡Ì	
信息	现实	¡Ì					¡Ì
	请求					¡Ì	
	输出		¡Ì				
转换	输入	¡Ì					
	输出		¡Ì				
连接	连接设备				¡Ì		
	现实				¡Ì	¡Ì	¡Ì

问题框架法的优点之一就是为问题分解提供了合理性原则。这在其他的分析法中是没有的。该方法从识别适合于简单问题框架的问题元素开始进行。问题的某一部分会适合于一种框架而其他一些部分则会适合于另一种框架。除非正在处理的是两个完全不同的问题（不仅可能而且简明易懂，只需独立地分别加以处理），否则在不同的框架之内总会有一些重叠或共性。这是多框架问题。

4）问题框架的应用

应用问题框架法时，建议采用直截了当的策略：

- 抽象问题域：
  - 标识子域；
  - 标识子域间的交互（根据共享现象）；
  - 刻画每个子域的特征；
  - 生成一个（成 *n* 个扩展的）上下文图。
- 识别出相关的标准框架；
- 调整框架，使之适配于问题（尽所能而为之）；
- 使用关于相关框架的 Kovitz 表来指导进一步的分析与文档编制任务。

这些步骤都是按显然的次序给出，当然也有可能这些步骤间存在很大部分的重叠和迭代。

5）问题框架小结

为了能够刻画问题的特征，问题框架将问题域建模成为一系列相互关联的子域。问题框架同时也是一个上下文图的开发，而上下文图则是为使得模型与特定问题之间的适配关系更趋密切而做的准备。

作为 PDOA 的组成部分，问题框架具有以下几点优势：

- 问题框架提倡在早期将注意力集中在问题域和需求上，这与任何其他原有的系统或解系统相反；

- 问题框架有助于标识所处理的问题域类型（单靠一个上下文图是无法做到的，所有其他系统的上下文图也都大致相同）；
- 问题框架容纳了那些真正组成需求的域间关系（毕竟标识需求是需求工程的一件头等大事）；
- 问题框架为各种类型问题的处理提供了专门的指南。

3.3 需求分析的工具

软件需求分析方法最初是为人工使用而开发出来的。对于一些大的软件项目，使用这些方法手工进行分析，就显得很麻烦而且容易出错。因此，针对这些方法，开发出了一些计算机辅助工具以帮助分析员进行需求分析，可以改善分析的质量和生产率。

需求分析的自动工具按不同的方式可以归为两类。

一类工具是为自动生成和维护系统的规格说明（以前是以手工方法制作的）而设计的。这类工具主要利用图形记号进行分析，产生一些图示，辅助问题分解，维护系统的信息层次，并使用试探法来发现规格说明中的问题。更重要的是，这类工具能够对要更新的信息进行分析，并跟踪新系统与已存在系统之间的连接。例如，Nastec 公司的 CASE2000 系统能够帮助分析员生成数据流图和数据字典，保存在数据库中，以便进行正确性、一致性和完备性的检查。事实上，这种工具与其他多数自动需求分析工具相比，其好处在于将“智能处理”应用到问题的规格说明中。

另一类需求分析工具要用到一种特殊的以自动方式处理的表示法（多数情况是需求规格说明语言），用需求规格说明语言来描述需求。规格说明语言是由关键字指示符号与自然语言（例如英语）叙述组合而成。规格说明语言被一个处理器处理以产生需求规格说明，更重要的，产生有关规格说明的一致性和组织方面的诊断报告。

3.3.1 SADT

SADT（Softtech 公司的商标）是 D.T.ROSS 等人于 1977 年提出来的一种结构化分析与设计的技术，已广泛地应用于系统定义、软件需求分析、系统设计与软件设计。最初，SADT 是作为一种手工方法而被开发的。

SADT 由 3 部分组成：

（1）分解软件（或系统）功能的过程；

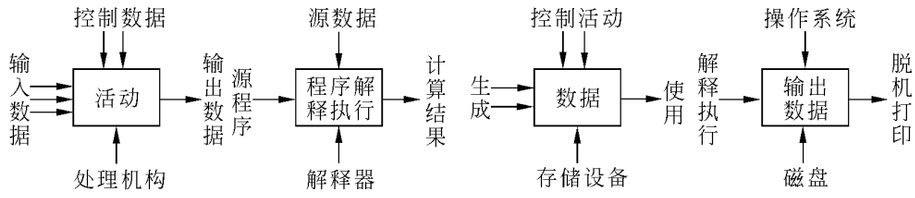
（2）能沟通软件内部的功能和信息联系的图解表示，即 SADT 活动图和数据图；

（3）使用 SADT 进行项目管理的指导书。

使用 SADT，分析员可以建立一个由许多分层定义的活动图和数据图组成的模型，此模型也叫做 SA 图。这种表示法的格式如图 3-17 所示。图 3-17（a）给出的是活动图的组成部件，图 3-17（b）给出的是数据图的基本组成部件。

在活动图中，结点表示活动，弧表示活动之间的数据流，因此，活动图是数据流图的另一种形式。但要注意不可将活动图与数据流图混淆。活动图有 4 种不同的数据流与每一个结点有关。如图 3-17（a）所示，一个结点的输出可以作为另一个结点的输入或

控制，而某些结点的输出则是整个系统对外界环境的输出。这样，每个结点的输出都必须连到其他结点或外部环境。输入与控制也必须来自其他结点的输出或者来自外部环境。



（a）活动图组成部件　　（b）数据图组成部件

图 3-17　活动图和数据图的组成

在数据图中，结点表示数据对象，弧表示对于数据对象的活动。其中，输入是生成数据对象的活动，输出是使用数据对象的活动，而控制是针对结点数据对象的一种控制，如图 3-17（b）所示。因而数据图与活动图是对偶的。在实践中，活动图应用得更广泛。不过，数据图也是很有用的，表现在：

（1）能指明一个给定数据对象所影响的所有活动；

（2）可利用由活动图构造数据图的办法去检查某个 SADT 模型的完全性和一致性。

描述软件工程定义阶段最初几个步骤的 SADT 活动图如图 3-18 所示。

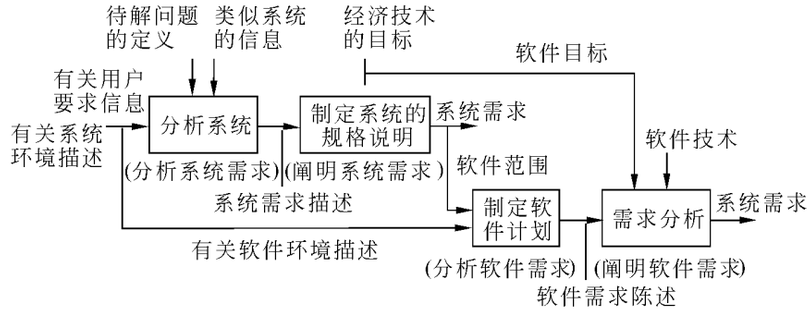


图 3-18　描述需求定义的 SADT 活动图

3.3.2　PSL/PSA

系统分析辅助工具 PSL/PSA 由问题说明语言（problem statement language，PSL）与问题说明分析器（problem statement analyzer，PSA）组成，是美国密执安（Michigan）大学开发的 ISDOS 系统的一部分。

问题说明语言 PSL 是一种用计算机处理的形式语言，可以按一定的语法用它正确地完整描述用户对系统的功能要求和性能要求。问题说明分析器 PSA 则可以对用 PSL 书写的需求规格说明进行分析，产生定义系统的许多报告。

PSL 可以从系统的信息流、系统结构、数据结构、数据的推导、系统的规模和容量、系统的动态特性、系统的性质和项目管理 8 个方面描述系统中的每个对象，以及这个对

象与其他对象之间的关系。

在用面向数据流的分析方法分析数据处理系统时，借助于 PSL/PSA 工具，可以一边对用户的数据处理活动做自顶向下的逐层分解，一边将分析过程中遇到的数据流、文件、加工等对象用 PSL 描述出来，并将这些描述输入到 PSL/PSA 系统。PSA 将对输入数据做一致性、完整性检查，并保存这些描述信息，形成一个可用计算机管理的数据库（或数据字典）。

PSL 有确定的语法，是一种计算机可处理的语言。所以，PSA 可以按系统分析员的要求，对用 PSL 书写的字典进行查阅、分析，打印出许多有用的报告。其中包括当分析员修改系统描述时，PSA 修改规格说明数据库的记录；以各种形式（包括数据流图）提供数据库信息的引用报告；提供开发项目管理信息的小结报告，以及评价数据库特性的分析报告等。

显然，PSL/PSA 系统提供的自动化方法，确实有以下好处：

（1）通过标准化和产生报告，改善了文档编制的质量；

（2）由于数据库可供大家使用，改善了分析员之间的协作关系；

（3）通过相互参照各种图表和报告，可以很容易发现一些漏洞、疏忽和不一致之处；

（4）比较容易追踪修改的影响；

（5）降低了对规格说明进行维护的成本。

3.4 传统的软件建模

3.4.1 软件建模

在很多的科学领域中，为了更好地理解和表达问题，常常采用问题模型化的方法。模型可以是一种物理实体模型，也可以是一种图表或者数学抽象概念的模型。

在软件工程中，在解决问题之前，首先要分析和理解问题空间，即调查分析。当对问题彻底理解后，需要把问题表达清楚。这就是对问题的“理解－表达”过程，在这过程中也常采用模型的表达方法。

所谓模型，就是为了理解事物而对事物做出的一种抽象，是对事物的一种无歧义的书面描述。简单地说，模型就是某一事物的抽象表示方式。通常，软件工程中的模型可以由一组图形符号和组织这些符号的规则组成。

模型是一种思考工具。利用模型可以把知识规范地表示出来，可以降低问题的复杂度，使问题更容易理解，便于进行系统分析与设计，便于开发人员与用户的交流。一个好的抽象化模型能够将问题重要的特性表达出来，排除无关的、非本质的干扰。

模型用于描述软件目标系统所有的数据信息、处理功能、用户界面及运行的外部行为等。一般来说，模型并不涉及软件的具体实现细节。

经过软件的需求分析建立起来的模型可以称之为分析模型或者需求模型。分析模型实际上是一组模型，是一种目标系统逻辑表示技术，可以用图形描述工具来建模，如选定一些图形符号分别表示信息流、加工处理以及系统的行为等，还可以用自然语言给出加工说明。

软件分析模型应包含以下的基本目标：

（1）描述用户对软件系统的需求；

（2）为软件设计奠定一个良好的基础；

（3）定义一组需求，并且可以作为软件产品验收的标准。

需求分析模型如图 3-19 所示。

在技术上，需求分析过程实际上是一个建模过程。分析模型的核心是数据字典，围绕着数据字典有 3 个层次的子模型，即数据模型、功能模型和行为模型。3 个子模型有着密切的联系，3 个子模型的建立不具有严格的时序性，是一个迭代过程。

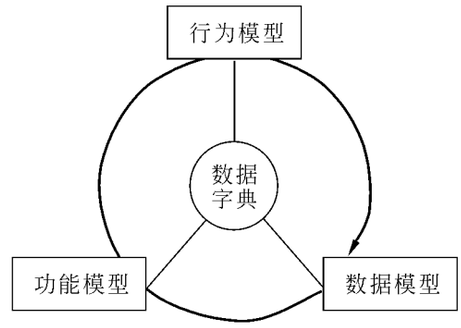


图 3-19　需求分析模型

3.4.2　数据模型的建立

在进行数据建模活动中，应该充分考虑系统处理哪些主要的数据对象；每一个数据对象如何组成，具有哪些属性；每一个数据对象与其他数据对象有哪些关系；数据对象与变换之间有哪些关系，等等。

数据模型用于描述数据对象之间的关系，数据模型应包含 3 种相关的信息，即数据对象、属性和关系。

1．数据对象

数据对象是几乎所有必须被软件理解的复合信息的表示。复合信息是指具有若干不同特性或者属性的事物。所以，仅有单个值的事物不是对象，例如宽度、长度等。

数据对象可以是一个外部实体、事物、行为、事件、角色、单位、地点、结构等。例如，人和汽车可以被认为是数据对象，都可以用一组属性来定义，例如汽车可以用制造厂家、车型、颜色等来描述。

数据对象只封装数据，没有引用对作用于数据对象的操作。这里所说的数据对象与面向对象方法中所描述的“对象”、“类”有着显著的区别。

2．属性

属性定义了数据对象的性质，具有 3 种不同的特性：

（1）为数据对象实例命名；

（2）描述该实例；

（3）引用另一个实例。

一个数据对象往往具有很多属性。应该根据要解决的问题和对问题语境的理解来确定数据对象的属性，选取一组本质的属性，排除与问题无关的非本质的属性。例如，教师的属性有教工号、姓名、性别、职称、职务、专业、研究方向、科研成果、担任课程、住址、电话、家庭成员等。对于设计一个教学管理系统来说，关心的是与教学有关的属性，应排除与教学无关的属性。

在一个系统中，数据对象的描述应包括数据对象以及数据对象所具有的属性，数据

对象描述可以用一个数据表来表示。

3．关系

数据对象彼此之间是有关联的，也称为关系。例如，数据对象“教师”和“课程”的连接关系是“教”，数据对象“学生”和“课程”的连接关系是“学”，等等。这种关联的形态有 3 种。

（1）一对一关联（1∶1）。例如，一所学校只有一位校长，所以，学校与校长的关联是一对一的。

（2）一对多关联（1∶*N*）。例如，一位教师可以“教”多门课程，所以，教师和课程的关联是一对多的。

（3）多对多关联（*M*∶*N*）。例如，一名学生可以学多门课程，一门课程有多名学生学习，所以，学生和课程的关联是多对多的。

4．实体-关系图

数据模型常常用“实体-关系图”ERD（entity-relationship diagram）来描述。实体-关系图重点关注的是数据，表示了存在于系统中的一个“数据网络”，也称为 E-R 模型。

ERD 包含 3 种基本元素，即实体、属性和关系。通常用矩形表示实体即数据对象；用圆角矩形或者椭圆形表示实体的属性；用菱形连接相关实体表示关系。例如，图 3-20 所示就是一个简化的教学管理 ERD。

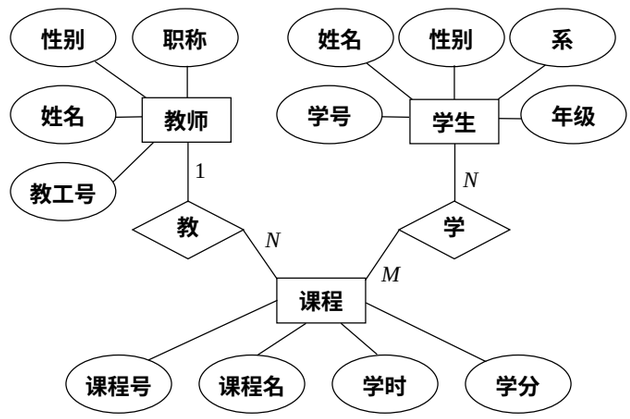


图 3-20　简化的教学管理 E-R 图

数据模型与实体-关系图为分析员提供了一种简明的符号体系，数据建模创建了分析模型的一部分。另外，由于 E-R 模型简单，容易理解，可以作为分析员与用户交流的工具。

3.4.3　功能模型、行为模型的建立及数据字典

1．功能模型

功能模型可以用数据流图描述，所以又称为数据流模型。人们常常用数据模型和数据流模型来描述系统的信息结构。当信息在软件系统移动时，会被一系列变换所修改。

数据流模型描绘信息流和数据从输入移动到输出，以及被应用变换（加工处理）的过程。

数据流图 DFD（data flow diagram）是一种图形化技术，数据流图符号简单、实用。用数据流图可以表达软件系统必须完成的功能。系统分析是把软件系统自顶向下逐层分解、逐步细化的过程，由此所获得的功能模型是一个分层数据流图，也就描述了系统的分解。

图 3-21 所示为一个加工数据流的一般画法。注意要对数据流、加工、文件等命名，还要对加工编号。

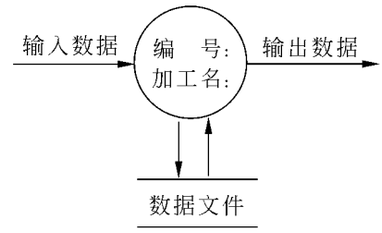


图 3-21　一个加工数据流的画法

数据流图中的有以下基本元素。

（1）数据流。数据流表示含有固定成分的动态数据，可以用箭头符号 → 表示。数据流包括有输入数据和输出数据（流动的数据）。

输入数据可能是由用户输入的一系列数据，也可能是网络连接传输的信息包，或者是从磁盘提取的数据文件等。输出数据是经过加工（变换）后的数据。

（2）加工处理。加工处理又称为变换或者功能模块，表示对数据进行的操作逻辑，可以用圆符号○表示。

加工（变换）是一个广义的概念，可以表示一个复杂的数值计算、逻辑运算、文字处理、作图、数据检索、分类统计等操作。加工可能产生新数据，也可能不产生新数据。通常，每一个加工都应该具有数据流入（进入箭头）和对数据加工后的数据流出（离开箭头）。

（3）文件。文件表示处于静态的、需要存储的数据，可以用符号 ═ 表示，同时，文件名写在两条直线之间。一般地，文件当被用于数据流中某一些加工之间的界面接口时，需要画出。通常，文件出现在中间和底层的数据流图中。

（4）源点和终点。源点和终点表示数据的产生和最终抵达处，通常是系统边界，例如入部门、人员、组织等，可用符号矩形□表示。

通常，数据流图是由许多的加工用箭头互相连接构成，数据流存在从加工→加工、加工→文件、加工→终点、源点→加工、文件→加工等情况，因为数据流总是与加工有关系的，而不会存在文件→文件、文件→终点、源点→文件、源点→终点等情况。

（5）分解的程度。系统自顶向下逐层分解时，可以把一个加工分解成几个加工，当每一个加工都已分解到足够简单时，分解工作就可以结束了。足够简单的不再分解的加工称为基本加工。如果某一层分解不合理、不恰当就要重新分解。

（6）加工说明。加工说明或者说加工处理（process specification）过程，用于描述系统的每一个基本加工处理的逻辑，说明输入数据转换为输出数据的加工规则。

加工逻辑仅说明“做什么”就可以了，而不是实现加工的细节。加工说明的描述方式可以是结构化语言、判定表、判定树、IPO（输入-处理-输出）图等。

所谓结构化语言是自然语言加上结构化的形式，是介于自然语言与程序设计语言之间的半形式化语言，其特点是既有结构化程序清晰易读的优点，又有自然语言的灵活性。

判定表是一种表格化表达形式，主要用于描述一些不容易用语言表达清楚，或者用语言需要很大篇幅才能表达清楚的加工。判定树是判定表的图形形式。

2．行为模型

在传统的数据流模型中，控制和事件流没有被表示出来。在实时系统的分析和设计中，行为建模显得尤其重要。事实上大多数商业系统是数据驱动的，所以非常适合用数据流模型，相反，实时控制系统却很少有数据输入，主要是事件驱动，因此，行为模型是最有效的系统行为描述方式。当然，也有同时存在数据驱动和事件驱动两类模型的系统。

行为模型常用状态转换图（简称状态图）来描述，又称为状态机模型。状态机模型通过描述系统的状态以及引起系统状态转换的事件来表示系统的行为。状态图中的基本元素有事件、状态和行为等。

事件是在某个特定时刻发生的事情，是对引起系统从一个状态转换到另一个状态的外界事件的抽象。简单地说，事件是引起系统状态转换的控制信息。

状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。状态规定了系统对事件的响应方式。系统对事件的响应可能是一个动作，或者一系列动作，也可能是仅仅改变系统本身的状态。

系统从一种状态转换到另一种状态，如图 3-22 所示。

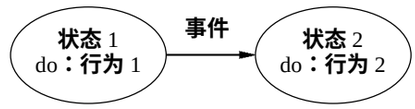


图 3-22　状态转换

在状态图中，用圆形框或者椭圆形框表示状态，在框内标上状态名，在表示状态的框内用关键字 do，标明进入该状态时系统的行为。从一个状态到另一个状态的转换用箭头线表示，箭头表明转换方向，箭头线上标上事件名。必要时可在事件名后面加一个方括号，括号内写上状态转换的条件。也就是说，仅当方括号内所列出的条件为真时，该事件的发生才引起箭头所示的状态转换。

系统的状态机模型可以理解为在任一时刻，系统处于有限可能的状态中的一个状态，当某一个激励（条件）到达时，激发系统从一个状态转换到另一个新状态。

3．数据字典

在结构化分析模型中，数据对象和控制信息是非常重要的，需要一种系统的、有组织的描述方式表示每一个数据对象和控制信息的特征。这可以由数据字典来完成。

简单地说，数据字典（data dictionary）用于描述软件系统中使用或者产生的每一个数据元素，是系统数据信息定义的集合。

数据字典方便人们对不了解的条目进行查阅，人们可以借助数据字典查出每一个名字（包括数据流、加工名、文件名等）的定义和组成，以免产生误解。

值得注意，对于一个大型的软件系统，数据字典的规模和复杂性会迅速地增长。事实上，人工维护数据字典是非常困难的，因此，需要使用 CASE 工具来创建和维护数据字典。

3.5　用 例 建 模

用例建模是一种从用户使用系统的角度来建立系统功能需求模型的方法。用例建模既不是从数据模型开始，也不是从系统数据流着手，而是从组成系统的实际操作入手。

用例图，又称用例模型，是用例建模的主要成果，从系统外部执行者的角度来描述系统需要提供哪些功能以及谁使用这些功能。

3.5.1 用例图

用例图非常简单直观，主要有 4 种基本成分：系统、参与者、用例和关系。

1．系统

系统是指待开发的任何事物，包括软件、硬件或者过程。在建模的过程中，首先要清晰地确定系统的边界，即系统中有什么，系统外有什么（尽管不须创建，但须考虑其接口）。通过确定系统的参与者和用例便可确定系统边界。在 UML 的用例图中，系统用一个矩形方框来描述，中间标明了系统的名称，如图 3-23 所示。

2．参与者

参与者用于表示使用系统的对象，是系统外部的一个实体，以某种方式参与用例的执行过程。参与者可以是人或者其他系统，用其参与用例时所担当的角色来代表。在 UML 中，参与者用一个“火柴棒形人”来表示，如图 3-24 所示。

图 3-23 系统的表示

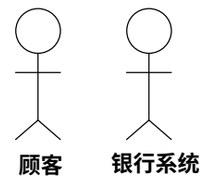


图 3-24 参与者的表示

根据参与者与用例关系的不同，参与者可分为两类：主要参与者（primary actor）和次要参与者（secondary actor）。主要参与者为主动发起人，通过使用用例从系统中获得业务价值。次要参与者参加用例的执行以为其他参与者创造业务价值。一个参与者在一个用例中是主要参与者，在其他用例中可能是次要参与者。用例的主要参与者可能是一个，也可能是多个。

3．用例

用例是参与者为达到某个目的而与系统进行的一系列交互，执行结果将为参与者提供可度量的价值（measurable value）。从参与者的角度来看，用例应该是一个完整的任务，在一个相对较短的时间段内完成。如果用例的各部分被分在不同的时间段，尤其是被不同的参与者执行时，最好将各部分作为单独的用例看待。在 UML 中，用例用一个椭圆形符号表示，其中标明用例的名称，如图 3-25 所示。

用例的命名非常重要。创立一个用例名时，要尽量使用主动语态动词和可以描述系统执行功能的名词。

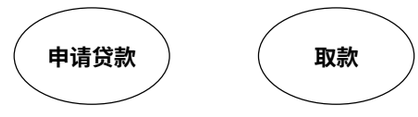


图 3-25 用例的表示

4．关系

在用例图中，关系用来描述参与者和用例间的关系，主要有 4 类关系：通信关系、泛化关系、包含关系和扩展关系。

1）通信关系

通信关系用来描述参与者与用例之间的关系。参与者触发用例，与用例交换信息，用例执行完后向参与者返回结果。不管参与者与用例之间有多少次交互，一个用例与一个参与者之间至多有一个通信关系。在 UML 中，通信关系采用执行者与用例之间的连线来表示。有时，为了明确谁是发起者，采用带箭头的连线，箭头表明发起的方向，如图 3-26 所示。

（a）未标明发起方向　　（b）标明发起方向

图 3-26　通信关系示例

2）泛化关系

在用例图中，参与者与参与者之间以及用例与用例之间存在泛化关系。参与者之间的泛化关系意味着一个参与者可以完成另一个参与者同样的任务，也可补充额外的任务。用例之间的泛化关系意味着一个用例是另一个用例的特殊版本。特殊用例（子用例）可以在一般用例（父用例）的执行序列的任意位置插入额外的动作序列，也可修改某些继承而来的操作和顺序。一般用例的任何包含关系和扩展关系也可被特殊用例继承。在 UML 中，泛化关系用一个带连线的三角形来表示。图 3-27 给出了泛化关系描述的示例。

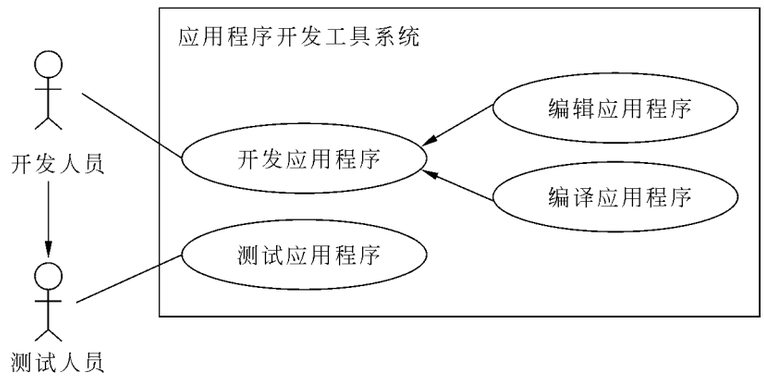


图 3-27　泛化关系描述示例

3）包含关系

包含关系描述了用例间的共同行为。当两个或两个以上用例有共同的执行序列片断

时，可以将这些执行序列片断抽出，形成被包含用例。同时，当一个用例描述的执行序列是另一个用例的执行序列的一部分时，也可使用包含关系。在 UML 中，包含关系用标有“《include》”的虚箭头线来表示，如图 3-28 所示。

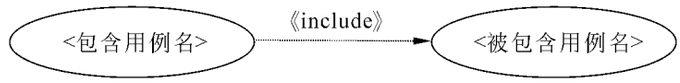


图 3-28 包含关系的图形表示

在描述包含用例（including use case）的动作序列时，一般在动作序列的某位置用“include（被包含用例名）”声明被包含用例。当包含用例执行到该位置时，转去执行“被包含用例”对应的动作序列，执行完“被包含用例”后返回，继续执行该位置以后的动作序列。包含关系有点类似于程序间的调用关系。

4）扩展关系

当对一个已存在的用例增加新的功能时，可使用扩展关系。扩展关系一般用于有条件地扩展已有用例的行为，是一种不改变原始用例的情况下增加用例行为的一种方法。在被扩展用例中，通常包含一些扩展点表明用例中允许扩展的地方。应该注意的是，扩展点并不要求用例一定被扩展，但如果扩展的话，表明可以发生扩展的地方。每个扩展点，在一个用例中都有一个惟一的名字和位置描述，通常在椭圆形的用例描述符号中添加一个扩展点分区来表示。显然，被扩展用例的描述并没有因此而改变。在 UML 中，扩展关系用虚线箭头线表示，其上标明“《extend》”扩展点列表和扩展条件，如图 3-29 所示。

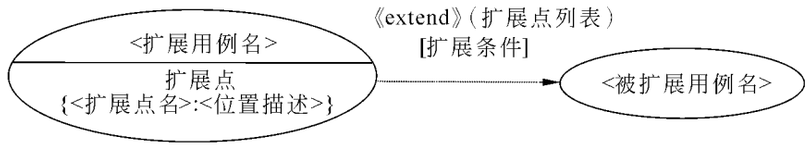


图 3-29 扩展关系表示

在使用扩展关系时应注意以下几个方面问题。

（1）扩展用例是一些用例片断，用来扩展“被扩展用例”的行为；扩展用例一般没有实例，若有也不包含“被扩展用例”中的行为。

（2）扩展用例可以在“被扩展用例”的多个扩展点扩展“被扩展用例”的行为。只要在“被扩展用例”执行到扩展用例引用的扩展点时扩展条件为真，则所有扩展用例的动作序列片断均插入“被扩展用例”中扩展点描述的相应位置。

（3）扩展用例可以继续被其他用例扩展。

（4）若“被扩展用例”的某个扩展点上有多个对应的扩展用例，则这些扩展用例执行的相对顺序是不确定的。

图 3-30 给出了简单 ATM 系统较完整的用例图。

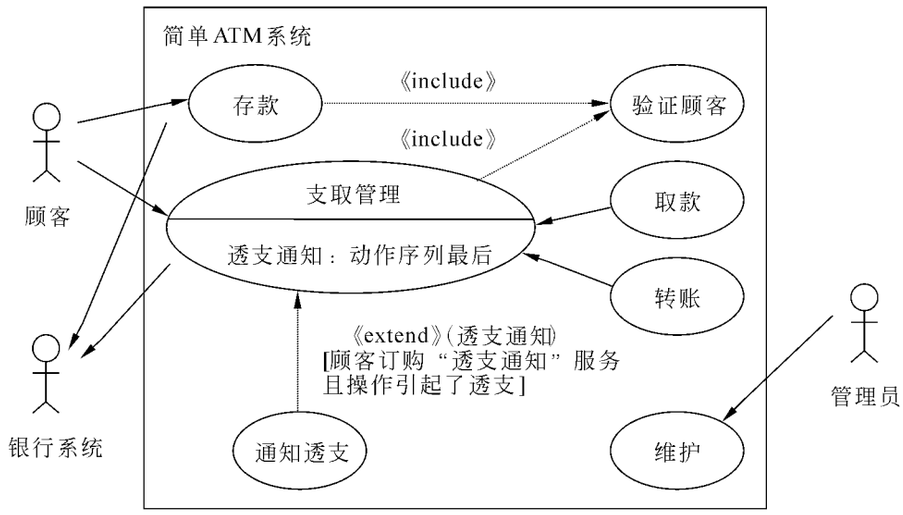


图 3-30　简单 ATM 系统用例图

3.5.2　参与者及用例的描述

在用例图中，参与者和用例均是简单的名字，在用例建模过程中需要进一步描述。

1．参与者描述

参与者描述的内容主要包括参与者的名称、是否为抽象参与，以及对参与者的简要描述。图 3-31 给出了关于参与者描述的一个简单示例。

参与者规格说明	
参与者名称：顾客	是否抽象参与者：否
简要描述： 使用 ATM 系统提取现金、转移资金和存款的所有用户，这些用户持有相应的银行卡且知道银行卡对应账号的密码。	

图 3-31　参与者描述示例

2．用例描述

用例描述有许多种方法，如简单文字、模板、表格、形式化语言和图形等，开发人员可根据项目进展及用户特点灵活选择。下面介绍几种常用方式。

1）简单文字

简单文字一般用于用例建模的早期，其内容主要是对用例提供功能的简单说明，例如，“获取呼叫历史”用例的简单文字说明如下：

用例“获取呼叫历史”使客户可以查阅账上已付费的所有呼叫的细节。呼叫历史可以以文本或声音的形式提供给客户。

2）模板

模板也是一种文字形式的描述，规定了开发人员需要阐述的有关项目。例如，RUP

（rational unified process）风格的用例描述模板主要包括用例名、事件流、特殊需求、前提条件、后置条件和扩展点。其中，前提条件表示用例执行前系统必须满足的状态；后置条件表示用例执行后系统所处的状态；事件流表示的是用例所执行的动作序列；特殊需求一般用来说明用例的非功能性要求。对事件流的描述一般采用编号的步骤序列来表示，采用陈述语句，从参与者的观点来描述。图 3-32 给出了一个用例模板描述示例。

用例名：购物
参与者：顾客（发起者）、出纳员
事件流：
1．顾客带着要购买的商品到达一个销售终端时用例开始。
2．出纳员录入商品。
3．系统显示商品信息和价格。
4．重复 2 和 3 步，直到顾客要购买的商品录入完毕。
5．系统计算并显示该顾客的商品价值总额。
6．顾客支付现金或信用卡刷卡或用支票支付。
7．系统打印收据。
可选路径：
2a．输入的商品标识符无效。
2a.l　系统显示出错信息。
6a．顾客不能足额支付所选的商品。
6a.l　取消本次交易。

图 3-32　用例模板描述示例

对于用例间关系的描述，不同的使用者可以采取不同的方法。一种较常用的做法是：如果用例包含其他用例，则在事件流的步骤序列中用 include 后跟被包含用例名来描述；如果用例是某用例的特殊用例，则在说明该用例的事件流步骤序列时用不同字体来表明该步是继承的、添加的还是覆盖一般用例的事件流步骤序列；如果用例是对某个用例的扩展，则对应扩展关系的不同扩展点描述各自的扩展事件流步骤序列。

开发人员在选用模板来描述用例时，可根据项目的需要增加或删除有关选项，例如增加对用例的非功能性需求、与用例有关的用户界面原型等。

3）表格

用表格描述用例时采用二维表格描述参与者的动作和系统的响应，主要描述用例的动作序列。图 3-33 给出了一个“取消订单”用例的表格描述形式。

客 户 代 表	系　　统	记 账 系 统
1．收到一个取消订单的请求		
2．输入订单的标识号		
	3．显示订单内容	
4．选择取消		
	5．给该订单打上取消标记	
		6．向客户账号增加订单支付的资金

图 3-33　表格描述用例示例

除上述几种描述方式外，还可采用图形来对用例进行描述，如 Petri 网（petri net）、UML 的顺序图、UML 的活动图、ITU 的消息顺序图等。此外，有些形式化语言，如 Z 语言、Occam 语言等也可用来描述用例，用于一些关键系统的用例建模。

3.5.3 用例建模过程

用例建模主要用来建立系统的功能模型，下面是其基本步骤。

1．找出系统的参与者和用例

确定系统的参与者和用例，也即确定系统边界，这是用例建模的第一步。参与者是系统之外与系统交互的所有事物。每个系统之外的实体可以用一个或者多个参与者来代表。下面一些问题有助于开发人员发现参与者。

- 谁使用系统的功能？
- 谁需要系统支持他们的日常工作？
- 谁来维护、管理系统使其正常工作？
- 哪些其他系统使用该系统？
- 系统需要与其他哪些系统交互？
- 系统需要控制哪些硬件？
- 对系统产生结果感兴趣的是哪些人或物？
- 是否有事情在预计的时间自动发生？

在选择参与者时，有两个非常有用的标准：首先，应该能至少确定一个用户来扮演参与者；其次，与系统相关的不同参与者实例所充当的角色间的重叠应该最少。

对于每一个参与者，确定其参与执行的用例。用例是系统的一种行为，通过执行用例向参与者提供可度量的业务价值。列出每个参与者需使用系统完成的事情，其中向参与者提供可度量的业务价值的事情可作为一个候选用例。下列一些问题的回答有助于开发人员发现用例。

- 参与者希望系统提供哪些功能？
- 系统存储信息吗？参与者将要创建、读取、更新或删除什么信息？
- 系统是否需要把自身内部状态的变化通知给参与者？
- 系统必须知道哪些外部事件？参与者如何通知系统这些事件？
- 系统需要进行哪些维护工作？

在确定用例时，“有价值的结果”和“特定参与者”是两个有用的准则。“有价值的结果”是针对主要参与者的，有助于避免确定太小的用例。“特定参与者”准则可使得用例是向真实的用户（实实在在的用户）提供价值，可以确保用例不会变得太大。

2．区分用例的优先次序

区分用例的优选次序，也即确定哪一项任务是最关键的，哪些用例涉及全局认识，哪些用例可以为其他用例所重用等。这样，优先级高的用例需要较早开发。区分用例的优先次序的技巧来源于经验，除考虑技术因素外，还需考虑非技术因素，如经济方面。开发人员需要与用户一起协商。

3．详细描述每个用例

详细描述每个用例的主要目的是为了详细描述用例的事件流，包括用例如何开始，如何与参与者进行交互以及如何结束。

用例开发人员需要与用例的真实用户密切合作。开发人员通过多次面谈，记录用户对用例的理解，并与用户讨论有关用例的各种建议。在用例描述完成之后，需要请这些真实的用户对用例描述进行评审。

用例描述的内容和方式可根据项目和用户特点，灵活选择。

4．构造用户界面原型

构造用户界面原型的目的是便于用户能有效地执行用例，为最关键的参与者确定用户界面的外观和感觉。首先，从用例入手，设法确定为使每个参与者能执行用例，需要用户界面提供哪些信息。然后，开发一个界面原型以说明用户如何使用系统来执行用例。界面原型可以是开发人员给出的界面草图，也可以是利用某种开发环境工具（如可视化编程环境）设计的用户界面。

5．构造用例图

构造用例图的目的是借助用例图中各元素的关系，如包含、泛化、扩展等，给出系统的合理结构，以便用户更容易地理解和处理。在构造用例图时，应该注意以下几个方面：

- 用例的结构关系应尽量反映真实情况；
- 需要将每个单独的用例视为一个单独的制品（artifacts）；
- 应尽量避免从功能上分解用例。

3.6 面向对象建模

传统的结构化方法学适合需求比较确定的应用领域，这一点已成为软件工程界大多数学者和实践者的共识。实际上，系统的需求却往往是变化的，而且用户对系统到底要求些什么也不是很清楚，而这些在面向对象方法中不再成为问题。面向对象技术发展十分迅速，成为20世纪90年代十分流行的软件开发技术。

从狭义上看，面向对象的软件开发包括3个主要阶段：面向对象分析（object-oriented analysis，OOA）、面向对象设计（object-oriented design，OOD）和面向对象程序设计（object-oriented programming，OOP）。其中，OOA是指系统分析员对将要开发的系统进行定义和分析，进而得到各个对象类以及对象类之间的关系的抽象描述；OOD是指系统设计人员将面向对象分析的结果转化为适合于程序设计语言中的具体描述，是进行面向对象程序设计的蓝图；OOP则是程序设计人员利用程序设计语言，根据OOD得到的对象类的描述，建立实际可运行的系统。

3.6.1 面向对象基础

1．基本概念

简单地说，面向对象方法学的基本原则有3条：

（1）一切事物都是对象；

（2）任何系统都是由对象构成的，系统本身也是对象；

（3）系统的发展和进化过程都是由系统的内部对象和外部对象之间（也包括内部对象与内部对象之间）的相互作用完成的。

从面向对象技术的实际应用情况来看，Smalltalk 语言是坚持这 3 条基本原则的典型代表，而 C++语言则在第一条原则上进行了一些修正。

面向对象方法之所以会如此流行，主要是因为面向对象方法非常适合于人们认识和解决问题的习惯。首先，面向对象方法是一种从一般到特殊的演绎方法，如面向对象中的继承，这与人们认识客观世界时常用的分类思想非常吻合。其次，面向对象方法也是一种从特殊到一般的归纳方法，如面向对象中的类，是由一大批相同或相似的对象抽象而得。面向对象方法的主要特征如下：

- 客观世界是由各种对象（object）组成的，任何事物都是对象，复杂的对象可由比较简单的对象以某种方式组合起来。因此，面向对象的软件系统是由对象组成的，软件中的任何元素都是对象，复杂的对象由比较简单的对象组合而成。
- 把所有的对象都划分为各种类（class），每个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，描述对象的状态信息；方法是对象所能执行的操作，也就是类中所能提供的服务。
- 按照子类（也称为派生类）和父类（也称为基类）的关系，把若干个类组成一个层次结构的系统。在这种类层次结构中，通常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），这一特性称为继承（inheritance）。
- 对象与对象之间只能通过传递消息进行通信（communication with messages）。

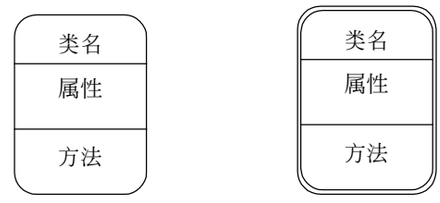
以上 4 个要点概括了面向对象方法的精华，可用如下公式概括：

object-oriented＝objects + classes + inheritances + communication with messages

下面介绍与这 4 个要点有关的一些基本概念。

1）类与对象

类是面向对象的一个基本概念，封装了客观世界中实体的特征和行为，即类的属性（数据抽象）和方法（过程抽象，也称为操作）两个方面。图 3-34（a）所示的表示符号用来描述类，类中的对象的表示符号见图 3-34（b）。其中，对象的表示符号是在类的表示符号的基础之上加了一个细实线的圆角矩形边框。图 3-34（a）表示该类只能用于派生新类而不能定义类的实例（即对象），称为抽象类；图 3-34（b）表示该类既能用于派生新类也能用于定义类的实例（即对象），外部细实线的圆角矩形框表示对象，内部粗实线的圆角矩形框表示类。



（a）类的表示符号　　（b）对象的表示符号

图 3-34　类与对象的表示符号

类是对一组相似对象的一般化描述。同一个类中的对象继承类的属性和方法。对一组相似的类进行抽象可以得到这一组类的超类（superclass），相应地，超类中的每一个类称为子类（subclass）。类、超类和子类的定义隐含地表示了类层次（class hierarchy）的概念。在类层次结构中，超类的属性和方

法可以由子类继承，而子类中又可能加入新的属性和方法。同时，子类中从超类继承而来的属性和方法以及子类新定义的属性和方法都可以由这一子类的子类继承。图 3-35 是类层次结构示意图的例子。

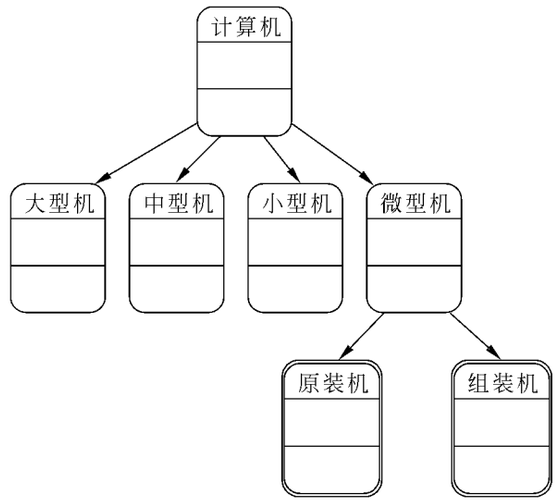


图 3-35　类层次结构示意图

2）属性

属性附属于类与对象，描述了类与对象区别于其他类与对象的特性。不论是物理对象还是人，对象的每一个特性可以看作一个类与一个特定的域之间的一种二元关系，这也就是说属性具有域中定义的某个值。在大多数情况下，一个域可以简单地用一组特定的值来表示。例如，假定类“人”有一个属性为“肤色”。而“肤色”对应的域为｛黄色，白色，黑色｝。这时，属性“肤色”就可以取这 3 种颜色值之一。在一些应用当中，可以为属性设置默认值。例如，可以将“肤色”的默认值设为“黄色”。

3）方法

方法也是附属于类与对象的，方法描述了类与对象的行为。面向对象中的每一个对象都封装了数据和算法两个方面。数据由一组属性表示，算法则用来处理这些数据。这里的算法也就是图中所示的方法，有时也称为操作或服务。类与对象封装的每一个方法代表着类与对象能够进行的一个“动作”。例如，对象“人”的“取肤色”方法可以获取存储在属性“肤色”中的值。

4）消息

消息是对象之间进行交互的手段和方法。向一个对象发送消息，接收消息的对象会执行相应的动作来作出响应，也就是执行某一方法。当方法执行结束的时候，这一响应动作也就完成了。图 3-36 表示了对象之间消息传递的过程。

发送消息的对象（以后简称发送对象）通过其内部的某个方法产生如下形式的一个消息。

消息：[目标，方法，参数]

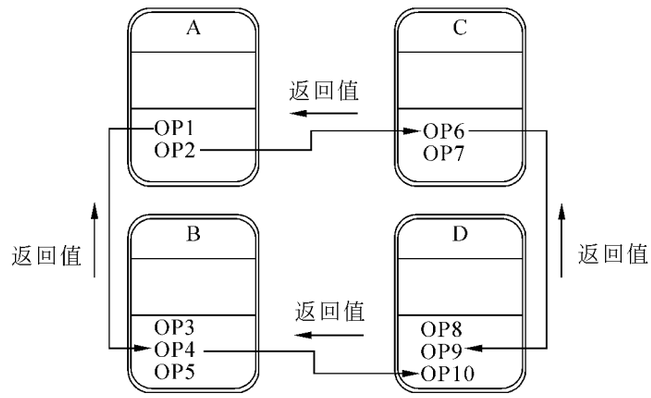


图 3-36　对象之间的消息传递

这里，目标代表消息将要传递到的对象（即接收消息的对象，以后简称接收对象），方法是指将要激活的接收对象的方法，参数是方法需要成功执行所需要的信息。下面通过一个例子来看一下对象之间通过消息传递进行通信的过程。图 3-36 所示的面向对象系统中，有 4 个对象 A，B，C 和 D。对象 A 有若干个属性和 2 个方法；对象 B 有若干个属性和 3 个方法；对象 C 有若干个属性和 2 个方法；对象 D 有若干个属性和 3 个方法。发送对象向一个对象发送消息，实质上是请求一个对象完成某项服务，在这种意义上，发送对象和接收对象之间是类似于客户机和服务器之间的关系。接收对象通常是这样响应消息请求的：首先选择要执行的方法，然后执行该方法，执行完该方法后将控制权返回给发送对象（必要时向发送对象提供返回值）。如图 3-36 中所示有 2 条消息传递的链路：一条是从对象 A 的方法 OP1 内部出发，经过对象 B 的方法 OP4，最后到达对象 D 的方法 OP10；另一条是从对象 A 的方法 OP2 内部出发，经过对象 C 的方法 OP6，最后到达对象 D 的方法 OP9。

传统的结构化方法中，数据和操纵数据的函数（或过程）是分开的，重点是将数据送到函数中，请求执行相应的功能。而在面向对象方法中，数据和操纵数据的方法合在一起构成对象，这时的重点是发送消息到对象以请求对象完成相应的功能。

5）封装，继承与多态

封装、继承与多态是面向对象方法区别于传统的结构化方法的 3 个主要特点。

面向对象的封装是指其类与对象把数据和操纵数据的方法合在一起构成一个整体，外部只能通过消息来同对象打交道。这样做有许多优点：

（1）数据和过程的内部实现细节对外部是隐藏起来的（即实现了信息隐藏）。这样一来，当改变发生时可以减少副作用的传播。

（2）数据结构和操纵数据结构的算法合在一个实体即类中，这样可以使复用变得很方便。

（3）封装起来的对象之间的接口简化了。发送消息的对象不必考虑接收消息的对象的内部数据结构的细节，只需要知道接收对象向外部提供的公用接口就可以了。这样一来，不仅接口问题得到了简化，系统间的耦合也降低了。

继承是面向对象方法区别于传统方法的一个关键特点。子类可以继承其超类的所有

属性和方法。这就意味着原来在超类中已经设计和实现的数据结构和算法在子类中不需要进行任何修改即可立即使用，软件复用可以直接实现。

类层次结构就是一种能够将高层次的修改立即逐层向下传播到整个系统的一种机制。因此继承还有一个很重要的特点：在类层次结构的适当的子类中，可以加入新的属性和方法。而这些新的属性和方法可以沿着类层次结构被下边层次中的子类所继承。

在某些情况下，可能需要同时从多个类中继承属性和方法，这种继承称为多继承。如果任何子类都只能从惟一的超类继承属性和方法，则称为单继承。通常来讲，单继承比多继承结构清晰。但是，在某些情况下，多继承会比单继承更为灵活。有的面向对象的程序设计语言（或其不同的实现）只支持单继承，如 Delphi；有的还支持多继承，如 C++。

多态是能够对现有的面向对象系统进行扩展而无须付出大的努力的一种特征。为了理解什么是多态性，考虑绘制不同类型的图（线条图、饼图和直方图）的例子。首先考虑使用传统的方法，这时的控制逻辑可以使用下面的伪语言表示：

```
CASE OF 作图类型
    IF 作图类型=线条图 THEN 作线条图（数据）
    IF 作图类型=饼图 THEN 作饼图（数据）
    IF 作图类型=直方图 THEN 作直方图（数据）
END CASE
```

尽管这种设计方法很直接，但是，如果要加入一种新的作图类型的话就比较麻烦了。这时，必须增加一种新的作图模块，而且对上述控制逻辑也必须作相应的修改。

下面使用面向对象的方法解决这一问题。从各种特殊的图形（线条图、饼图和直方图等）可得到一个一般化的类“图形”，在图形类中可定义方法“作图”。然后，可以从一般化的图形类派生若干子类（这里为线条图类、饼图类和直方图类）。在各个子类中重载其超类中的“作图”方法。这样，当向这些子类中的任何一个对象发送“作图”消息时，该对象会自动地执行相应的“作图”方法绘制出线条图、饼图或者直方图。如果后来需要增加一种新的作图类型到系统中去，也只需要从一般化的图形类再次派生一子类并重载相应的“作图”方法即可，一般化的图形类和原有的各图形子类都不需要进行任何修改。所以多态就是使得完成不同功能的方法可以使用同样的名字。换句话说就是，同一接口可以具有不同功能。

2．对象模型基本元素的标识

前面介绍的类与对象、属性、方法和消息等基本元素合在一起就构成了面向对象模型。但是，对于一个实际问题如何标识这些基本元素呢？下面给出如何标识这些基本元素的有关建议。

1）标识类与对象

任何对象总是附属于某一类的，因此，标识出了系统中的对象也就标识出了系统中的类。通常，标识对象是从考察系统的问题陈述或者系统的处理描述开始的，可以首先找出问题陈述或处理描述中的名词或名词性短语，是类与对象的候选者。然后，对这些名词或名词性短语进一步分析（可能会删掉一些，可能会合并一些，也可能会增加一些），即可找到系统中的类与对象。下面是一些可能的类与对象的类型。

- 外部实体：产生或消费计算机系统中信息的外部实体。如其他系统、设备、人等。
- 事物：作为信息域一部分的一件事物，如一份报告、一次显示结果等。
- 事件：系统运行过程中发生的事件，如一个电话或一个警报等。
- 角色：人与系统交互时所扮演的角色，如学生或老师等。
- 组织机构：与应用相关的组织机构，如财务部门等。

根据实际情况，可能还会存在其他类型的候选者。标识类与对象时，能够正确地找出这些类与对象的候选者是很重要的。但是，与这一点同样重要的是，要能够经过分析确定这些候选者当中哪些不是真正的类与对象，从而应从候选列表中去掉它们。

2）标识属性

属性描述对象的稳定特性。在标识对象属性的时候，分析员仍然需要研究问题陈述或处理描述，并且选择那些适合于对象的特征。另外，对每一个对象应该回答这样一个问题，“哪些数据项完整定义了问题中的这一对象？”

3）标识方法

方法定义了对象可以执行的动作。方法通常以某种方式改变对象中属性的值，因此，方法必须能够“知道”对象的某些属性，方法也必须能够操纵这些属性对应的数据结构。方法大致可以分为如下 3 类：

- 以某种方式操纵数据的方法（如增加、删除、重新格式化、选择等）；
- 完成某个计算的方法；
- 为了获得某个控制事件序列而监视对象的方法。

为了标识对象的方法，分析员仍然可以研究问题陈述或处理描述（与标识类与对象时的找名词或名词性短语不同，标识方法时通常是找动词）。在初步标识出一些基本的方法之后，进一步的工作是将方法分解成一些更小的子方法。标识方法的另外一种方法是考察对象之间的消息通信。

4）标识消息

面向对象系统中，对象与对象之间的通信是通过相互之间发送和接收消息来实现的。因此，应该考察所有对象在整个生存周期中发送和接收消息的情况。这通常是通过考察系统中发生的事件而获得的。消息总是与方法对应的，因此，标识出了“新”的消息，也就意味着找到了“标识方法”时尚未找到的方法。

3．面向对象软件开发模型

面向对象系统通常都具有随着时间演化的特征，因此，与组件装配（本质上是组件复用）结合的各种演化模型就是面向对象软件开发模型的最好选择。如图 3-37 所示的是面向对象开发中常用的组件装配模型。

图 3-37 的左边部分是一个典型的螺旋模型，该螺旋模型的每一轮（即每一个螺旋，代表一次迭代过程）包括 3 个阶段：风险分析、面向对象软件构造、用户确认。

图 3-37 的右边部分是对每一次迭代过程中的面向对象软件构造阶段的更详细的子阶段划分。每一次迭代的面向对象软件构造起始于标识系统中候选的类与对象，然后到现有的库中查找是否存在这些类，如果存在，则直接从库中获取；如果不存在，则使用面向对象分析、设计、编程和测试这一整套的方法构造这些类，然后把新构造出的类放

到库中（以备复用）。最后，用从库中查找到的类或采用面向对象方法构造出的新类，构造当前这一轮系统。如此循环，完成 n 轮的面向对象软件构造。

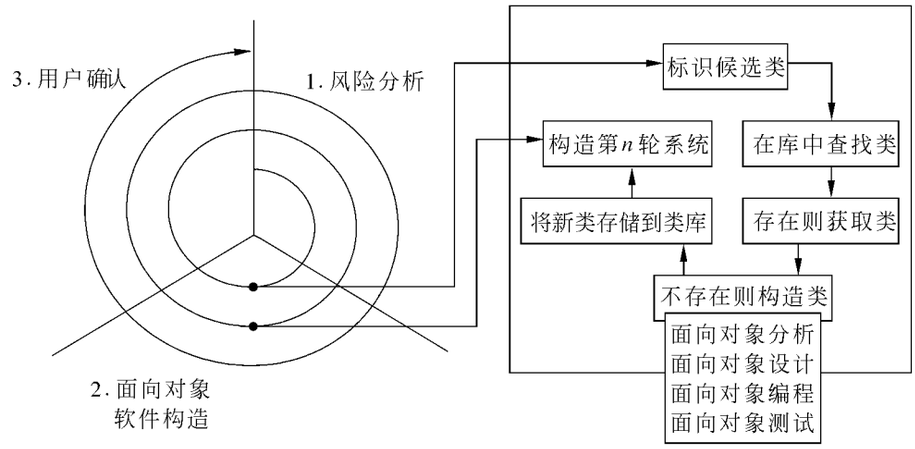


图 3-37　组件装配模型

面向对象的方法要求采用一种演化的方法进行软件开发，因为只通过一次循环定义完系统中所有的类是相当困难的。随着面向对象分析和设计模型的演化，很可能会需要加入一些新的类。组件装配模型能较好地适应这一演化过程。

3.6.2　面向对象分析模型

面向对象分析的目标是要建立一系列的模型来描述能够满足用户需要的计算机软件。面向对象分析模型需要表示出系统的信息（或数据）、功能和行为 3 个方面的基本特征。相应地，在进行面向对象分析时，需要建立面向对象的对象模型、功能模型和行为模型 3 种模型。

1．对象模型

建立对象模型时，首先要确定系统中有哪些对象（或类），每一个对象（或类）需要哪些属性，然后确定对象（或类）之间的关系。对象模型通常使用类似于实体-关系图这样的图形工具进行表示。对象模型有时也称为类模型。对象模型是面向对象分析模型中最重要的一种模型。

2．行为模型

建立行为模型，是要确定系统的动态行为，即对象能够发送或接收的事件（或刺激）以及系统状态发生转移的情况。行为模型通常使用类似于状态转换图等图形工具进行表示。

3．功能模型

建立功能模型的目的，是要确定如何来对数据（即对象中的属性对应的数据结构）进行计算和处理。功能模型通常使用类似于数据流图的图形工具进行表示。一般地说，功能模型是面向对象分析模型中最易被忽视的一种模型。

下面将以软件工程中的一个经典例子（电梯问题）介绍这 3 种模型的建立过程。假定在一栋 *m* 层的建筑物中有 *n* 个电梯，要求设计一个软件以确保电梯的正常运行。电梯运行时要求满足下面的约束条件：

（1）每一个电梯有 *m* 个按钮，每一个按钮对应建筑物的一个楼层。按钮按下时会发亮显示并且电梯会运行到相应的楼层。

（2）除了顶层和底层以外的所有楼层都有 2 个按钮，一个用于请求上楼，一个用于请求下楼。按钮按下时会发亮显示。当电梯运行到某一楼层然后离开时相应楼层按钮的显示会取消。

（3）当没有请求时，电梯会停留在当前楼层，并且电梯的门处于关闭状态。

一般地说，应该首先建立对象模型，然后建立行为模型，最后建立功能模型。但是，这一建立过程不完全是线性的，而应该是迭代的。也就是说，随着新的模型的建立，应该回过头考察已建立的模型，并且根据需要对已建立的模型进行必要的修改。

3.6.3 对象模型的建立

类与对象除了应该有一个适当的名字以外，其两个重要的方面就是属性和方法。确定类与对象的属性是这一阶段要做的工作，而确定类与对象的方法通常要延迟到面向对象设计阶段去做。除了确定类与对象的属性以外，这一阶段要做的另一个主要的工作是确定类之间的关系和对象之间的关系。类之间的关系是类之间的类层次结构关系，即一般类与特殊类之间的继承关系。对象之间的关系则有很多种，其中最为重要的一种是整体与部分之间的关系，其他的对象与对象之间的关系统称为对象与对象之间的关联关系，反映对象间的相互依赖、相互作用，有一对一、一对多和多对多等基本类型。类之间的关系和对象之间的关系通常使用类似于实体-关系图的图形符号进行表示。

在具体介绍电梯问题的对象模型之前，首先介绍对象模型的表示符号。然后通过电梯问题介绍类与对象及其属性的确定，类之间的关系和对象之间的关系的确定，进而建立电梯问题的对象模型。

1．对象模型的表示符号

前面已经介绍了类（实际上是抽象类的表示符号）和对象的表示符号。这里再介绍几种符号，以表示类之间的关系和对象之间的关系。

图 3-38（a）是类与类之间一般和特殊的继承派生关系的表示符号，该符号的半圆弧指向父类，半圆的弦边与各个子类连接。图 3-38（b）是对象与对象之间的整体和部分之间的包含关系的表示符号，该符号的三角形的一个角指向“整体”对象，与该角相对的边同各个“部分”对象连接，在“整体”对象那边的数字（或数字对）表示有多少个“整体”对象与一个“部分”对象有关系，在“部分”对象那边的数字（或数字对）则表示有多少个“部分”对象与一个“整体”对象有关系。图 3-38（c）是对象与对象之间的除整体和部分之间的关系以外的任何关联关系的表示符号，该符号只是一条实线将两个不同的对象连接起来（连线中间标上关联的名称），在线段一边的数字（或数字对）表示线段这一边有多少个对象与线段另一边的一个对象有关系。图 3-38（d）带实心箭头表示消息联系。不管是对象与对象之间的整体和部分之间的关系，还是对象与对象之

间的其他关联关系，其表示符号上边的数字（或数字对）都是表示对象与对象之间的约束关系。因此，正确地识别表示符号是很重要的。

（a）A 类派生 B 类　　（b）A 类包含 B 类　　（c）其他关联关系　　（d）A 给 B 发送消息

图 3-38　对象模型中的关系表示符号

2．对象模型的建立

如果已经有了比较规范的问题描述，那么要确定候选类与对象是比较容易的。通常用下划线标识出问题描述中的名词或名词性短语，这些名词或名词性短语将是类与对象的候选者。

下面以电梯问题为例说明。

（1）一个电梯有 m 个按钮，每一个按钮对应建筑物的一个楼层。当按下时按钮会显示并且电梯会运行到相应的楼层。

（2）除了顶层和底层以外的所有楼层都有 2 个按钮，一个用于请求上楼，一个用于请求下楼。当按下时按钮会发亮显示。当电梯运行到某一楼层然后离开时该楼层中相应按钮将不再发亮显示。

（3）当没有请求时，电梯会停留在当前楼层，并且电梯的门将关闭。

上面的描述中有电梯、按钮、建筑物、楼层、顶层、座层、显示、请求、门共 9 个名词。其中建筑物、楼层、顶层、底层、门不属于电梯系统这一问题，因此，可以忽略。另外，显示、请求是抽象名词，抽象名词通常是类与对象的属性，而不是类或对象，因此，这两个名词也可以排除。剩下的就只有电梯、按钮两个候选者，也就是第一轮中找到的类与对象。相应地，显示是按钮的属性（为了更好理解，将显示这一属性的名称改为显示状态），而请求在这里是一个多余的名词，既不是类或对象，也不是属性。

上面的问题描述中有两类按钮，一类是电梯按钮，另一类是楼层按钮，这里按钮就是超类，电梯按钮和楼层按钮就是按钮的子类。再来看一下电梯的属性，至少可以定义电梯的门是开还是关的状态，这一属性可取名为开门状态。有了这些类与对象及其属性，还有类之间的关系，就可以画出第一轮的对象模型，如图 3-39 所示。

这样得到的第一轮对象模型是否合理呢？在一个实际的电梯系统中，按钮不直接与电梯打交道（或通信），而是通过电梯控制器间接与电梯打交道（由电梯控制器来决定哪一台电梯响应电梯外面的用户请求，电梯里边用户的请求肯定是由所在的电梯响应）。但是，在上面的问题描述中，根本就没有提到电梯控制器。考虑到这种实际情况，应该将电梯控制器看作电梯问题中的类与对象。

考虑电梯控制器的对象模型就是第二轮得到的对象模型（图 3-40）。现在看来，这一对象模型已经比较令人满意了。于是，可以开始下面的阶段，建立电梯控制系统的行为模型。

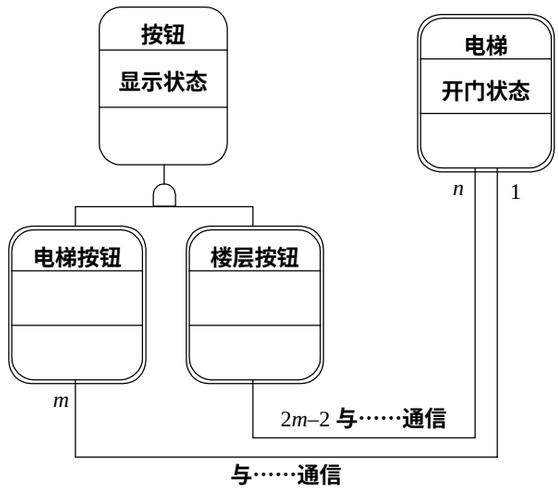


图 3-39　第 1 轮对象模型

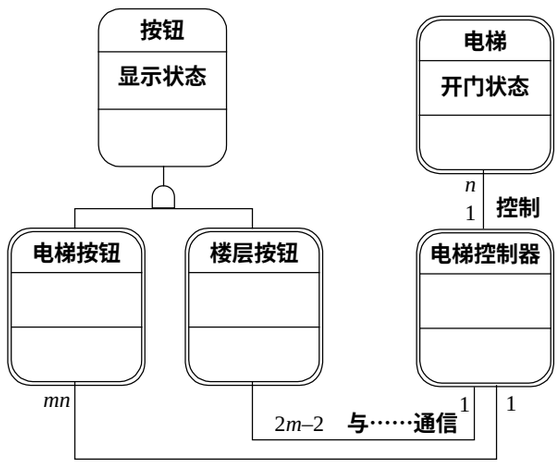


图 3-40　第 2 轮对象模型

3.6.4　行为模型的建立

行为模型指出系统如何对外部事件作出响应，可以使用行为模型来描述系统的动态行为。为了建立行为模型，系统分析员需要采取下列步骤：

① 仔细评价需求收集阶段所编写的各种用例（use cases），以充分理解系统中的各种交互序列；

② 标识出驱动这些交互序列的各种事件，同时要理解这些事件如何与特定的对象发生关系；

③ 为每一个用例建立事件跟踪图；

④ 为每一个对象建立状态转换图；

⑤ 复查行为模型以验证其准确性和一致性，必要时返回到上一阶段修改对象模型。

下面通过电梯问题介绍如何建立系统的行为模型，为了简单起见，仅讨论建筑物中只有一部电梯的情况。

1．通过用例标识事件

建立行为模型之前充分考察一些典型用例是非常必要的，可以使得分析员充分地理解所要解决的问题，从而能够建立正确的行为模型。分析员在需求收集阶段收集典型用例的时候，不仅要收集正常情况的用例，而且要收集异常情况的用例。对电梯问题而言，正常情况的用例是指能够按照用户最希望的方式（这里是指最快的方式）将用户送到想要去的楼层；异常情况是指不能以最快的方式将用户送到指定的楼层，而不是指错误的情况，比如，不能出现用户进入电梯后电梯一直运行而不开门的错误情况等。下面列举两个典型的用例，假定电梯起始时在 1 楼。

1）典型用例之一：正常情况

（1）用户 A 在 4 楼按了向上的楼层按钮请求电梯，希望去 6 楼。

（2）4 楼向上的楼层按钮变亮显示。

（3）电梯自下而上运行到 4 楼。电梯里面还有另一个用户 B，用户 B 已经按了去 10 楼的电梯按钮。

（4）4 楼向上的楼层按钮变暗。

（5）电梯门打开。

（6）用户 A 进入电梯。

（7）用户 A 按去 6 楼的电梯按钮。

（8）6 楼的电梯按钮变亮显示。

（9）一定时间过后电梯门自动关闭。

（10）电梯运行到 6 楼。

（11）6 楼的电梯按钮变暗。

（12）电梯门打开，允许用户 A 走出电梯。

（13）用户 A 走出电梯。

（14）一定时间过后电梯门自动关闭。

（15）电梯继续带着用户 B 运行到 10 楼。

2）典型用例之二：异常情况

（1）用户 A 在 4 楼按了向上的楼层按钮，但是用户 A 希望去 1 楼。

（2）4 楼向上的楼层按钮变亮显示。

（3）电梯自下而上到达 4 楼。电梯里面还有另一个用户 B，用户 B 已经按了去 10 楼的电梯按钮。

（4）4 楼向上的楼层按钮变暗。

（5）电梯门打开。

（6）用户 A 进入电梯。

（7）用户 A 按去 1 楼的电梯按钮。

（8）1 楼的电梯按钮变亮显示。

（9）一定时间过后电梯门自动关闭。

（10）电梯运行到 10 楼。

（11）10 楼的电梯按钮变暗。

（12）电梯门打开，允许用户 B 走出电梯。

（13）用户 B 走出电梯。

（14）一定时间过后电梯门自动关闭。

（15）电梯继续带着用户 A 运行到 1 楼。

标识事件的一种常用方法是找出用例中的动词或动词短语，是候选的事件。直接在上面的典型用例中用下划线标出动词或动词短语。这里只讨论用例一（某些明显不是事件的动词没有标出，如“希望去”等）。标识出了事件之后，应该明确事件是谁发出的（可能是外部实体，也可能是内部对象）和事件作用于哪个对象，还应该指出事件携带的信息以及一些条件或约束等。这里考虑一个典型的事件——“按了向上的楼层按钮”。这一事件的发出者是“用户”，作用于“楼层按钮”对象，该事件的效果是使得按钮变亮显示（如果在该用户按下该按钮前已经变亮，则仍然保持亮的状态）。一旦用例中所有事件都已标识出来，则可以建立这一用例的事件跟踪图。

2．建立用例的事件跟踪图

在事件跟踪图中，用户和对象用垂直线表示，事件用对象之间（或用户与对象之间）的有向线段表示，事件之间的时序关系则自上而下隐含地表示出来。图 3-41 是电梯问题一个事件的事件跟踪图例子。事件跟踪图可以很好地表示系统行为的内部细节，为面向对象设计阶段识别每个对象的方法提供了有价值的信息。

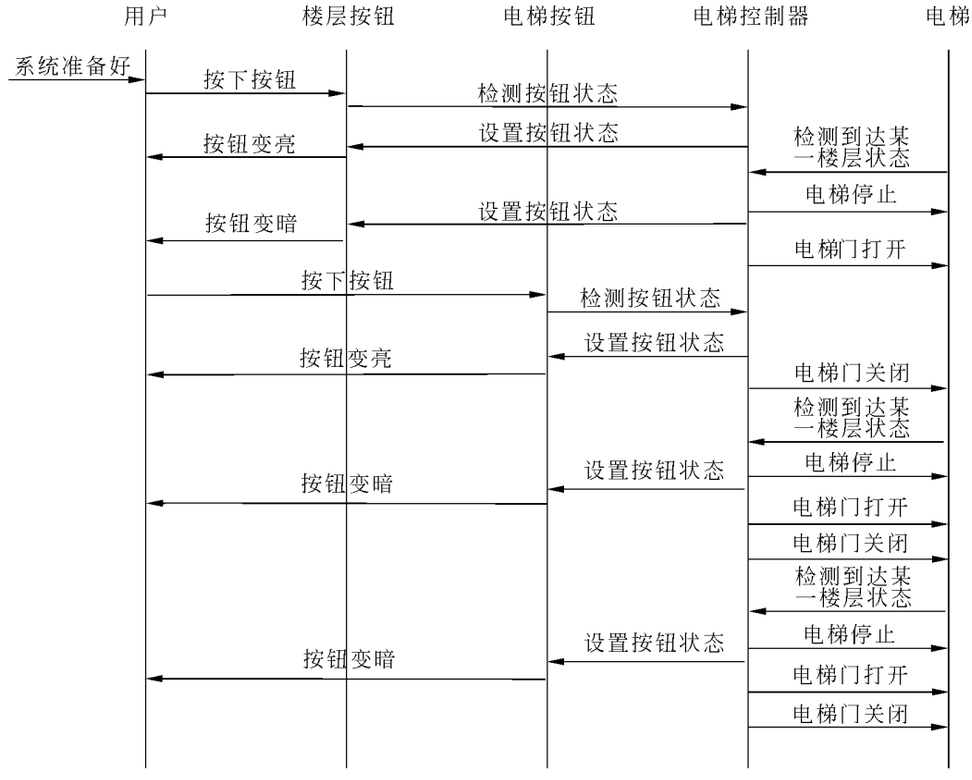


图 3-41　一个典型用例的事件跟踪图

有关该事件跟踪图的几点说明如下：

（1）图 3-41 中所列事件名称与“典型用例之一”中的下划线部分的候选事件有所不同，都采用了简化的形式。

（2）图 3-41 中增加了一些在“典型用例之一”中的下划线部分的候选事件中没有的事件，如“检测／设置按钮状态”和“检测到达某一楼层状态”等。

（3）为清晰起见，图中没有指出当前到达了哪一楼层的信息，也没有指出当前要设置哪一个按钮的“变亮”还是“变暗”信息等。读者对照“典型用例之一”可以了解这些信息。

（4）读者可以考虑一下，该图中为什么“电梯按钮”只有一次变亮而有两次变暗？

3．建立对象的状态转换图

有了用例的事件跟踪图，可以据此画出用例所涉及的对象的状态转换图。当然，要画出完整的状态转换图，只有一个用例的事件跟踪图是不够的。应该尽可能多收集一些用例，并画出每一个用例的事件跟踪图。在此基础上，建立每一个对象的状态转换图。从事件跟踪图导出对象状态转换图的基本方法和步骤是：首先选择事件跟踪图中的一个对象，然后标识事件开始时的对象状态，最后沿着事件跟踪图的垂直方向检查各个事件。如果对象由于事件改变了状态，则引入新的状态，并从当前状态向新的状态做一条有向边；如果对象在输入事件下没有改变状态，则向对象本身做一条有向边。边上用事件名称和将要激活的处理进行标注。

图 3-42 是“电梯控制器”对象的第一轮状态转换图。图中的圆圈表示初始化系统状态。矩形框代表系统的状态，箭头代表状态之间的转换（也即引起状态转换的事件）。每一个箭头上面都有相应的标注。标注的上边部分说明引起状态转换的事件，下边部分说明该事件将引起的动作。要注意的是，状态与加工之间不一定是一对一的关系。

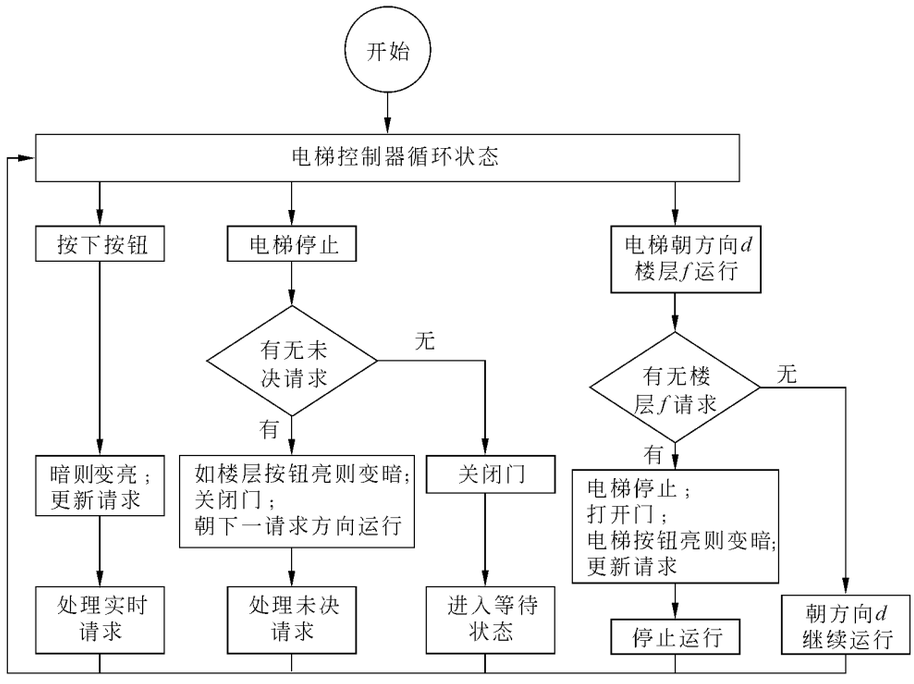


图 3-42　第 1 轮状态转换图

状态转换图通常用来描述实时系统的动态行为特征。但是，由于将要开发的信息系统可能会具有某些实时系统部件，因此，状态转换图在这种情况下就非常有用。那么，如何找出系统的状态呢？一条非常有用的启发式规则是找系统中的某些硬件。上面的例子中，“按钮”、“电梯”和“电梯控制器”都是典型例子。这里，“电梯控制器”具有“电梯控制器循环状态”、“处理实时请求”、“处理未决请求”、“确定是否停止”、“继续运行”、“停止运行”和“进入等待状态”等各种状态。其他对象的状态转换图请读者自己练习。

4．复查行为模型

上面从用例出发，首先做出了各个用例的事件跟踪图，然后做出各个对象的状态转换图。实际上，各个对象的状态转换图就构成了系统的行为模型。在完成各个对象的状态转换图之后，应该检查对象的准确性和一致性。一般来说，只要是事件，必然会有产生事件的对象（或用户），也必然会有事件的接收对象（或用户）。当然，发送对象和接收对象可能是同一对象（或同一类型的对象）。因此，可以从这一角度进行重点复查。最后，当完成了整个行为模型的建立过程，转入下一个模型即功能模型的建立之前，应该回到前面已经建立的对象模型，如果有必要，应该对前面建立的对象模型进行修正。由于当前这一个例子中暂时还没有必要修正对象模型，因此下面就直接开始建立功能模型。

3.6.5 功能模型的建立

面向对象分析建模的第三个阶段是建立功能模型。功能建模类似于结构化方法中的数据流建模。这里的功能模型就类似于前面介绍的数据流图。不像行为模型中要考虑动作或处理的时间顺序，功能模型中不对处理的时间顺序进行表示。电梯问题的第一轮功能模型见图 3-43。图中表示了各种动作（或处理），反映了数据的流动情况，但是，该图没有反映动作（或处理）的顺序。

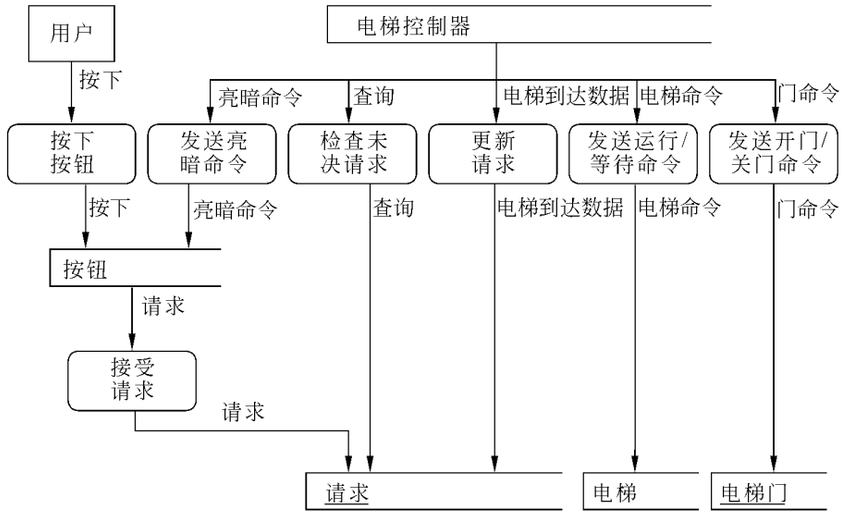


图 3-43　第 1 轮功能模型

数据流图中有源点/终点、处理、数据流和数据存储 4 种基本元素。在以前的结构化

方法中，要区分是源点/终点还是数据存储比较容易。但是，在面向对象的功能模型中，要区分源点/终点还是数据存储则需要考虑一些新的问题，关键是要理解“状态”这一概念。面向对象的类中的属性有时被称为“状态变量”，原因是在大多数面向对象的实现中，系统的状态通常是由一些对象的属性值决定的。在上一节介绍的行为模型与结构化方法中的面向行为的建模中得到的状态转换图在很多方面是相同的。这样一来，状态的概念在面向对象方法学中扮演的作用非常之大也就不足为奇了。状态的概念也能用于建立功能模型时决定某些数据项是归为源点/终点，还是归为数据存储。如果某个数据项在系统运行过程中可能会发生改变，则应该看作是数据存储，如该图中的电梯控制器、请求、电梯和门都被看作数据存储。但是用户只是简单地按下按钮，用户的状态不会改变，因此用户被看作源点。

结构化方法中的数据流图和面向对象方法中的功能模型的一个重要区别在于数据存储。具体地说，结构化方法中的数据存储几乎总是由某种类型的文件来实现。但是，对象中的属性（状态变量）也是一种数据存储。因此，功能模型中包括两类数据存储：一类是对象代表的数据存储，另一类是非对象代表的数据存储（通常由某种类型的文件实现）。为了表示这两类不同的数据存储，通过给数据存储的名称加或不加下划线予以区分。本书中约定，不加下划线表示对象代表的数据存储，加下划线表示非对象代表的数据存储。为简单起见，功能模型图中删除了所有的编号，且有些用动词表示的数据流名是事件，如“按下”表示“按下按钮事件”。

在建立好第一轮功能模型之后，回过头来复查对象模型和行为模型。从图 3-43 可以看出，电梯门应标识为一个类，而不是第一轮和第二轮对象模型中作为“电梯”类的一个属性。原因是面向对象方法学中允许状态被隐藏在对象里边，从而能够避免非授权的访问。如果建立“电梯门”类（门的“打开”、“关闭”状态是作为该类的一个属性保存的），就只能通过向“电梯门”类的对象发送消息才能打开或者关闭电梯门。如果将这一属性设置在“电梯”类中，可能会因不慎而改变了这一属性的状态，这将导致非常严重的后果。因此，应该在原有的对象模型中加入一个新的类：电梯门。同样的道理，也应另外设置一个新的类：请求。这样，就可以得到第二轮的对象模型（见图 3-44）。

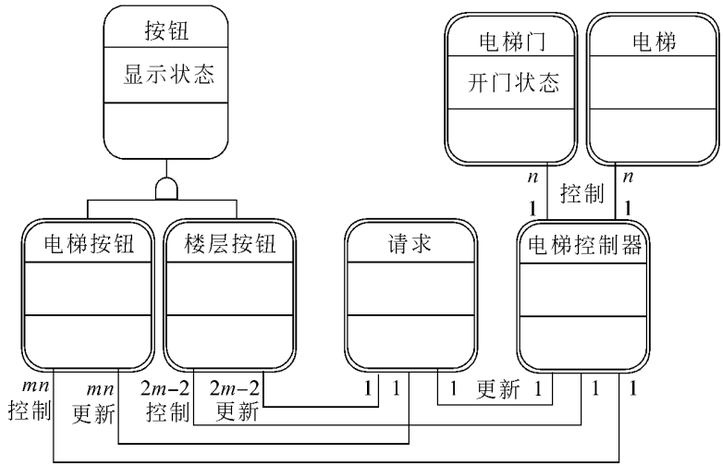


图 3-44　第 2 轮对象模型

在修改了对象模型之后，应该再一次复查行为模型和功能模型，必要时对行为模型和功能模型进行修改。由于将第一轮功能模型中的两个非对象的数据存储（“电梯门”类和“请求”类）修改成了对象，因此可以得到修改后的第三轮的功能模型（见图 3-45）。实际上，第二轮功能模型与第一轮功能模型之间只有很小的差别，即“电梯门”和“请求”两个数据存储的名称在第二轮功能模型中不带有下划线。

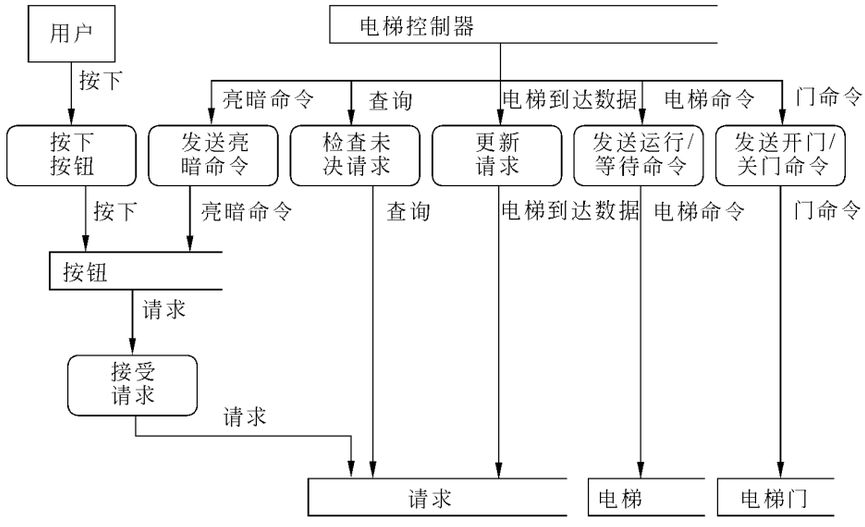


图 3-45　第 3 轮功能模型

最后指出，虽然回过头修改了对象模型，但是行为模型仍然没有进行修改。当然，对于其他实际系统，修改对象模型时可能会牵涉修改行为模型，以及反过来又要修改功能模型等。总之，面向对象分析模型主要由对象模型、行为模型和功能模型 3 种模型组成，而且一般是先建立对象模型，再建立行为模型，最后建立功能模型。但是，随着各个模型相继建立，对系统了解的信息越来越充分，可能需要回过头来对已经建立的模型做必要的修改和补充。只有经过若干轮修改和完善并确认所建立的模型确实满足了用户需要的时候，才可以考虑进行下一阶段的面向对象设计。

3.7　统一建模语言 UML

统一建模语言 UML（unified modeling language）是一种用于描述、构造可视化和文档化软件系统的语言，由 Rational Software 公司及其合作伙伴开发。许多公司正在把 UML 作为一种标准整合到其开发过程和产品当中，包括商务建模、需求管理、分析、设计、编程、测试等。

UML 是 Booch 方法、OOSE（object-oriented software engineering）方法、OMT（object modeling technique）方法和其他一些建模方法的组合和延伸。其开发始于 1994 年末，当时 Rational Software 公司的 Grady Booch 和 Jim Rumbaugh 各自开始了 Booch 方法和 OMT 方法的统一工作。1995 年秋天，Ivar Jacobson 连同其 Objectory 公司加入了 Rational

Software 公司，并将其 OOSE 方法也合并了进来。

本节主要对 UML 1.3 版这一标准文本进行简单介绍，其主要内容取自 1999 年 6 月发布的长达 800 多页的 OMG unified modeling language specification（Version 1.3）。希望更多了解该规范的读者可参阅有关文献。

3.7.1 UML 的基本实体

UML 的基本实体由两大类构成：定义 UML 本身的实体和使用这些实体产生的 UML 项目实体。

1．定义 UML 本身的实体

定义 UML 本身的实体包括 UML 语义描述、UML 表示法和 UML 标准 profile 文件。

2．UML 项目实体

选择哪一种模型和创建哪些图表对于如何解决问题及构建解决方案有极大的影响。集中注意相关细节而忽略不必要细节的抽象方法，是学习和交流的关键。

- 每一个复杂系统最好通过一个模型的几个接近于独立的视图进行描述；
- 每一个模型可以在不同的精确级别上进行表达；
- 最好的模型是与现实世界相关的模型。

根据一个模型、多个视图的观点，UML 定义了下面几种图形表示：

- 用例图（use case diagram）；
- 类图（class diagram）；
- 行为图（behavior diagrams）；
- 状态图（statechart diagram）；
- 活动图（activity diagram）；
- 交互图（interaction diagram）；
- 顺序图（sequence diagram）；
- 协作图（collaboration diagram）；
- 实现图（implementation diagrams）；
- 构件图（component diagram）；
- 配置图（deployment diagram）。

这些图提供了对系统进行分析或开发时的多角度描述，基于这些图就可以分析和构造一个自一致性（self-consistent）系统。这些图与其支持文档一起，是从建模者角度看到的基本的实体。当然，UML 及其支持工具还可能会提供其他一些导出视图。

UML 不支持数据流图。简单地说，是因为数据流图不能很好地融入到一个一致性的面向对象方法学中。活动图和协作图能够表示人们想要从数据流图中得到的大部分内容。此外，活动图在进行工作流建模时也非常有用。

3.7.2 UML 的目标及范畴

1．UML 的目标

UML 的主要设计目标如下：

- 为用户提供即时可用的、表达能力强的可视化建模语言，以开发和交流有意义的模型；
- 提供了扩展核心概念的扩展机制和特殊化机制；
- 支持独立于编程语言和开发过程的规格说明；
- 提供一种理解建模语言的形式化基础；
- 鼓励对象工具领域的发展；
- 支持更高级的开发概念，如组件、协作、模式和框架；
- 整合了最好的工程实践经验。

2．UML 的范畴

UML 是一种描述、构造、可视化和文档化软件系统的语言。

UML 融合了 Booch 方法、OMT 方法和 OOSE 方法的概念，目标是为使用这些方法和其他方法的用户提供一个单一的、公用的、普遍适用的建模语言。

UML 致力于对现有各种方法进行建模。例如，UML 开发者围绕并发、分布式系统进行建模，确保 UML 能够完全表示这些领域。

UML 专注于一种标准的建模语言，而不是一个开发过程。尽管实际使用时 UML 必须应用于某一种具体的建模过程，经验表明，不同的组织机构和不同的问题域要求不同的建模过程。因此，重点首先放到了一个公用的元模型上（metamodel，该元模型统一了各种语义），其次放到了一种公用的表示法上（notation，该表示法提供对这些语义的表示）。UML 开发者提倡这样一种开发过程：用例驱动、以体系结构为中心、迭代式的和渐增式的开发过程。

UML 包含了面向对象业界一致认同的核心建模概念。UML 允许导出形式，即以其扩展机制进行表示。UML 提供了如下一些设施：

- 以一种直接而经济的方式表示当前各种建模问题的语义和表示法；
- 表示未来可能的建模问题的语义，特别是与组件技术、分布式计算、框架和可执行性（executability）相关的语义；
- 通过扩展元模型，使得单个的项目能够以低成本完成可扩展机制；
- 使得未来的建模方法能够在 UML 基础之上形成可扩展机制；
- 在不同工具之间方便地进行模型转换的语义；
- 为共享和存储模型实体刻画到信息中心库（repository）的接口的语义。

3.7.3 UML 图的使用实例

UML 可以表示几种不同的图形，从而可视化地表示系统的不同方面。本节分别从语义、表示法和实例 3 个方面简单介绍以下几种 UML 图：

- 用例图（use case diagram）；
- 类图（class diagram）；
- 状态图（statechart diagram）；
- 活动图（activity diagram）；
- 顺序图（sequence diagram）；
- 协作图（collaboration diagram）；

- **构件图**（component diagram）；
- **配置图**（deployment diagram）。

1．**用例图**

用例图表示角色和用例之间的关系。用例代表的是一个系统或分类器（classifier）**的功能，通过与这一系统或分类器相关的外部交互者进行交互予以呈现。**

一个用例图是由一些角色、一组用例、还可能有一些接口以及这些组成元素之间的关系构成的图，关系是指角色和用例之间的联系，用例通常用矩形框起来以表示系统或分类器的边界，如图 3-46 **所示。**

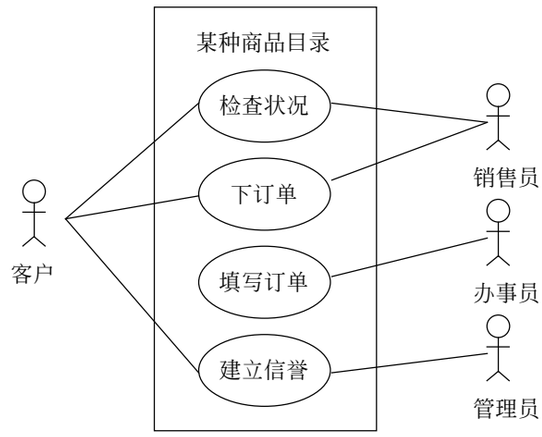


图 3-46　用例图

2．**类图**

类图是一组静态的描述性模型元素相互连接的集合图。模型元素包括类、接口和类与接口之间的关系等，如图 3-47 **所示，＃表示受保护成员，＋表示公有成员，–表示私有成员。**

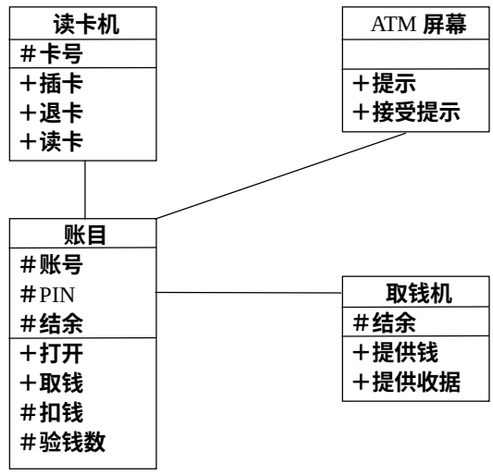


图 3-47　类图

3．状态图

状态图用于描述模型元素在接收到事件后的动态行为。最典型的情况是用于描述类的行为，当然也可用于描述其他实体元素（如用例、角色、子系统、操作或方法等）的行为。

状态图是用于表示状态机（state machine）的图。图形中的状态和各种其他类型的顶点（伪状态）用适当的状态或伪状态符号表示，状态之间的转换则用有向弧连接表示，如图 3-48 所示。

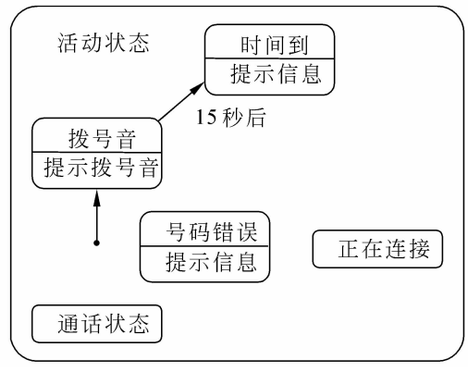


图 3-48　状态图

4．活动图

活动图是状态图的一种特殊情况。在活动图中，绝大部分状态都是动作或子活动状态，并且绝大部分甚至所有的转换都是通过动作或子活动的完成所触发。活动图通常用于描述绝大多数甚至是所有的事件都是由内部动作的完成所引起的情况，如图 3-49 所示，而一般的状态图则用于表示异步事件。

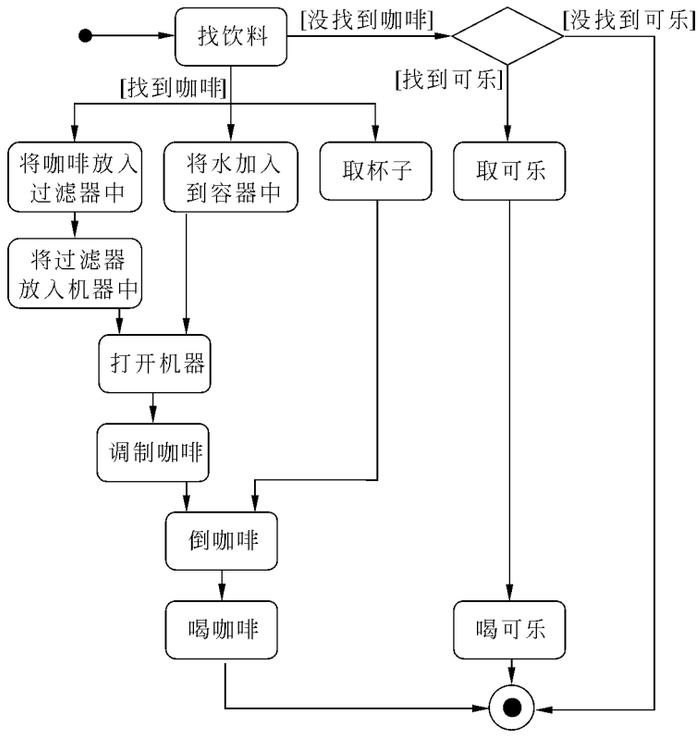


图 3-49　活动图

5．顺序图

顺序图表示交互，是指为得到一个期望的结果而在多个分类器角色（classifer role）之间进行的交互序列。

顺序图有两维，垂直维代表时间，水平维表示对象。通常，垂直维自上至下代表时间向前推进，如图 3-50 所示。

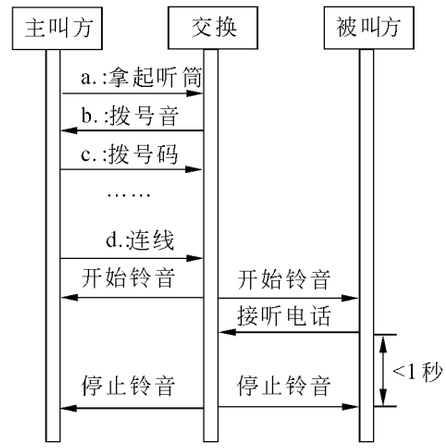


图 3-50 顺序图

6．协作图

协作图表示协作，包含一组由对象扮演的角色，以及在一个特定的上下文中的关系。

协作图描述相互联系的对象之间的关系，或者分类器角色（classifier role）和关联角色（association role）之间的关系。协作图有两种不同的形式，即实例级（instance level）的图示和规格级（specification level）的图示。图 3-51 给出一个规格级的例子。

7．构件图

构件图表示软件构件之间的依赖关系。软件构件包括源代码构件、二进制代码构件和可执行构件。一些构件存在于编译时刻，一些存在于链接时刻，一些存在于运行时刻，还有一些可能存在于不止一个时刻。

构件图是由依赖关系连接各个构件而成的图，也可能与代表复合关系的物理包容体构件进行连接。在构件图中，使用带箭头的破折线表示依赖关系，如图 3-52 所示。

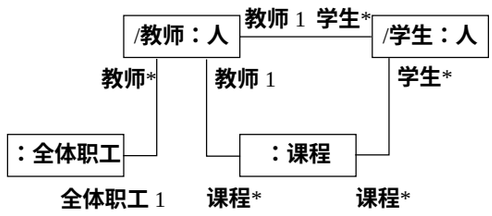


图 3-51 协作图

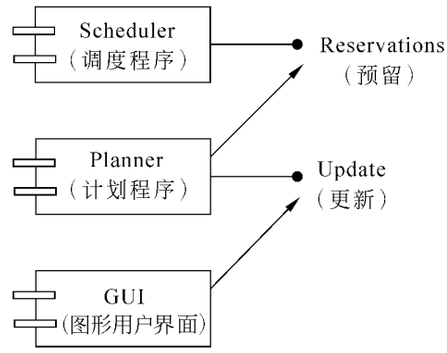


图 3-52 构件图

8．配置图

配置图表示运行时处理元素、软件构件以及基于处理元素和软件构件的进程和对象的配置情况。不处在运行状态的实体的软件构件不在配置图中出现，而在构件图中表示。

配置图是一个由通信关系连接起来的结点图。结点可能包含构件实例，构件可能包含对象。构件与构件之间的依赖关系用带箭头的破折线表示，如图 3-53 所示。

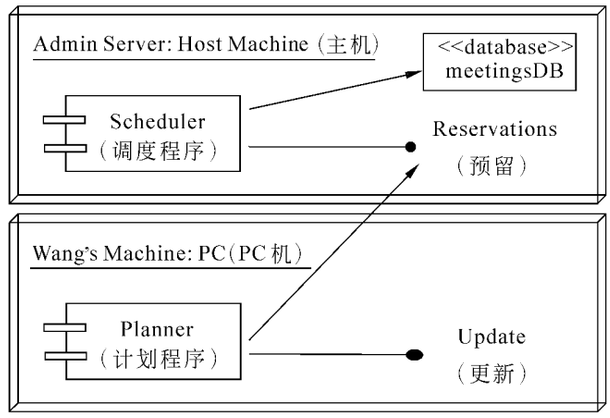


图 3-53　配置图

3.8　典型应用分析

本章以电梯控制系统作为典型应用，从结构化分析、面向对象分析和面向问题域的分析 3 种需求分析方法入手，所有应用分析均假设读者已熟练掌握了各种不同的建模技术。

3.8.1　结构化分析示例

对于某些类型的应用，结构化分析的适用性远胜过其他方法。但也绝不应该因其表现上佳就避而不谈那些结构化分析拙劣的例子。待求系统是一个基于软件的、用于电梯的控制器。假定已经获取了某种形式的需求记录作为信息源。然而应指出的是，虽然认可了将需求获取记录作为信息源，但这些文档并非 SA 所明确要求的一部分。

对于本案例，其运作难以把握，并且也不是一个开展分析的可靠的基础。所以，依选择，或者对问题域建模（就像在“传统的”SA 中），或者是对解系统建模（像在现代 SA 中）。首先尝试使用前者。

标识出问题域中那些感兴趣的不同的问题子域，即：

- 用户；
- 调用按钮；
- 发送按钮；
- 指示器；
- 电机；
- 传感器；
- 电梯；
- 门控器。

而指定任务就是将这些问题子域合并成一个 DFD，要做到这一点，必须首先将这些

问题子域归类为端子、处理、数据存储以及数据流。由于没有哪个问题子域是数据存储或数据流，因此，或许可以按图 3-54 所示的方法，用某种通过数据流互连的方框来表示这些问题子域。即使是允许对其做“艺术的创意”(电机与电梯间的连接肯定是机械式的并且也几乎不会作为数据流的一部分)，图中的各部分却是不连通的，并且很明显地缺少了处理部分。

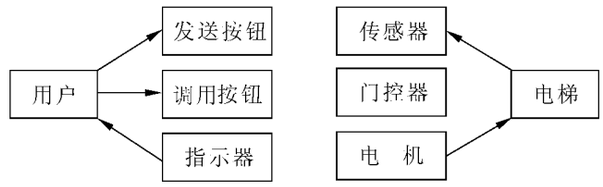


图 3-54　问题子域

要应对上述难题，可以采用现代 SA 中的策略，引入解系统并生成一个如图 3-55 所示的上下文 DFD。看起来更像是有了进步，但有几点应加以注意。用户和电梯并不直接与解系统（电梯控制器）交互，因而也就无法并入上下文图，另一个棘手的问题就是，极少谈及任何端子（毕竟是问题域的主要组成部分）的特性。

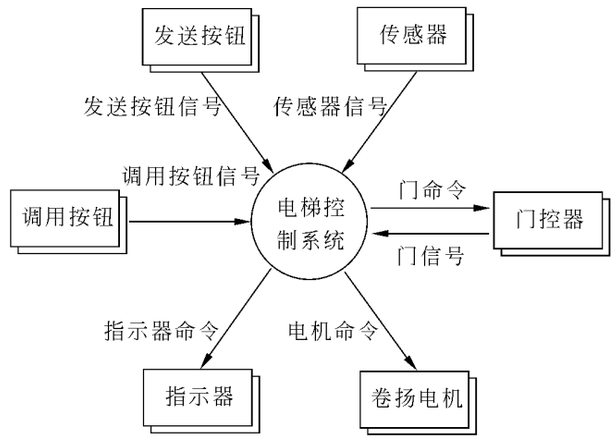


图 3-55　上下文 DFD

有相当多的内容是需要予以说明的，然而 SA 却未明确地指出应在何处以及该如何妥善处理这一问题。

已经标识出来的数据流能够而且也应该在 DD 中加以定义，而且如果一切顺利的话，前述的案例研究已经充分地说明了这一情况。现在该继续为解系统开发一个底层的 DFD。某些处理（功能）就包含在外部数据流中，图 3-56 展示了这些处理如何被合并起来。

然而，在这中间仍有一处明显的不足，即没增加什么实质性的东西。如果希望继续描述解系统的行为，那么就必须引入一些假定的处理以作为容纳此类描述的“地方”。图 3-57 展示了一个可能的“解决方案”。

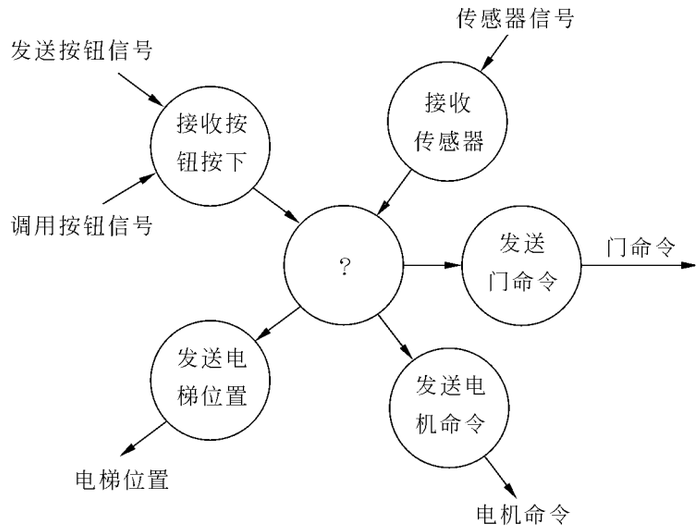


图 3-56　底层的 DFD

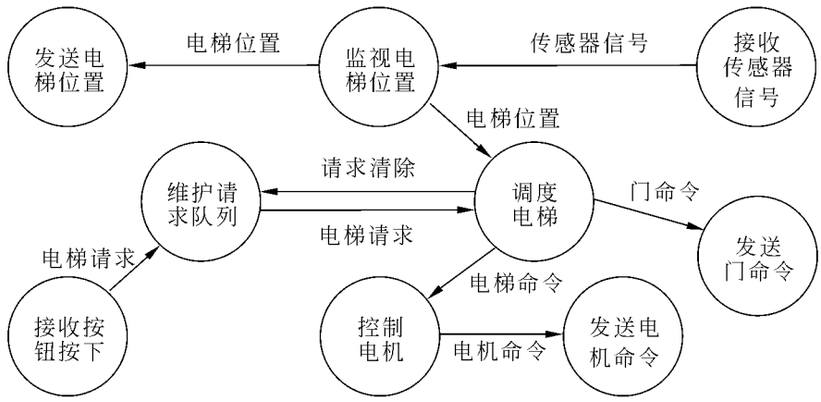


图 3-57　解决方案

迈出了这一步（而且很可能对某些处理做了很细致的分解），现在才有可能根据这些子处理的操作来描述系统的功能。例如，处理“调度电梯”的一个结构化文本描述就完全可能含有以下部分：

检查电梯位置
if **电梯正接近某一楼层，**then
　if **是顶层或底层** or
　　有去往该层的发送请求 or
　　（有调用请求 and **电梯正向着该层的方向移动）**then
　　发出电梯停止命令

几经周折，总算把需求中的一员合并到模型中了。

现代 SA **的另一项任务是数据分析并且导出如图** 3-58 **所示的** ERD**。 照例，这些实**

体可以根据属性加以定义并且导出以下的 DD 的补充部分：

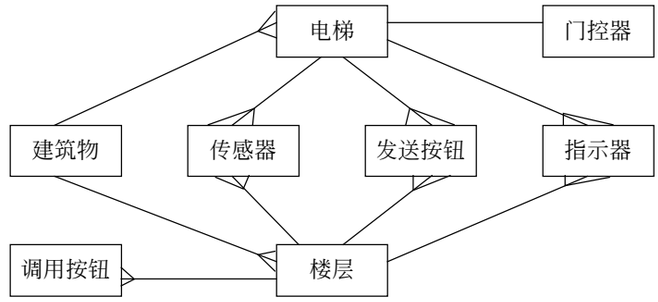


图 3-58　ERD

建筑物　　::=楼层数，电梯数。
楼层　　::=楼层标识。
电梯　　::=电梯标识。
方向　　::=“上”,“下”。
位置　　::=楼层标识。
门控器　　::=电梯标识，门状态。
门位置　　::=“已关闭”,“已开启”,“正开启”,“正关闭”。
传感器　　::=电梯标识，楼层标识，传感器类型。
传感器类型　　::=“上方”,“下方”,“所在”。
调用按钮　　::=楼层标识，方向。
发送按钮　　::=电梯标识，楼层标识。
指示器　　::=电梯标识，楼层标识。
……

这确实提供了有关问题域的一些有用的信息。通过给不同的术语赋予语义的解释，其性能还可得到进一步增强，尽管这一点并非 SA 所追求的目标。

对上述模型的进一步开发将集中于对 DFD 做详尽的描述和精化。然而，这很显然已成了解系统的内部设计了，因此将这一工作搁置一边。现在该回顾一下已取得的在问题分析上的成果。

问题域的描述以及理解似乎很难一见。特定的问题子域（端子）以及子域的某些属性都已被标识出来，但仍存在一些需要开发者了解清楚的地方。一个中心问题就是，无需被提示去询问许多关于问题域的、关键性疑难问题及其编写其答案的文档，便有可能完成所指定的模型。例如，在何处描述不同类型的传感器？何时触发一个传感器，对于电梯位置该指示些什么？等等。

需求也未得到更妥善的处理。需求可以容身的惟一场所是被掩盖在各种不同的假设的内部处理的描述中。此外，解系统行为的任何规格说明的顺利完成只能付出引入这样一个假定的解系统结构的代价了。

对于这类批评的标准回答就是，所有这些已包含在“附随的解说词”中。但这也不

是令人满意的回答，部分是由于未提供关于附随的解说词的指南，同时也是由于太多的输入推迟到这一被遗忘了的装东西的地方。

因此，对于本例的控制系统来说，在描述问题域以及标识需求这些关键性分析方面，这种方法作用似乎不大。而问题域的分析一经完成，该方法即可以提供一种有效的结构化设计方法，当然这是另外的话题。

3.8.2 面向对象分析示例

待求系统为一个用于电梯的软件控制器。同前述的例子一样，假定已经完成了某种形式的需求获取，并且需求获取记录已经存在。继续进行基本分析任务，即描述或建模问题域及其相关联的需求。

按照前述的案例研究，标识问题域的对象类只是一个开头。下列候选的对象类（表3-9 所示）可以标识出来，一些基本属性和操作也同时列出。

表 3-9 对象类

对 象 类	属 性	操 作
建筑物（楼）	楼层数，电梯数	
楼层	楼层标识，调用	设置调用，清除调用
电梯升降梯井		
电梯	电梯标识，方向，位置	启动，停止，获取方向，设置方向
发送按钮	电梯标识，楼层标识，已按下	按下，松开，检查按动
调用按钮	楼层标识，方向，已按下	按下，松开，检查按动
传感器	楼层标识，电梯标识，状态	获取状态
指示器	电梯标识，楼层标识，开/关	设置开，设置关
用户		
门	门位置	关，开，获取位置
电机	电梯标识	启动，停止，设置方向

由于没有明显的属性或操作，用户类和电梯升降梯井类看来不太可能作为候选类。然而用户类可以暂缓取消，理由是用户类使用了一个由其他类（按钮类）提供的服务（按键）。电梯与电机间也存在一些相互交叉之处，所共享的大部分操作似乎真正应该属于电机，而将两种类型的按钮类从一个按钮超类继承得到也是完全可以的。

稍后将再回到这些问题上来，首先密切关注一下到目前为止所有圆满完成的任务。

毋庸置疑，这些类在现实世界都有相应的对应物，但随着解系统的偏差不断地渗入，这一点可能会受到一定的影响。例如，一个楼层明显地是在现实世界中存在的，然而，一经建模，便拥有了一个属性，即是否该有一个未完成的调用。很少会有面向对象的爱好者对此存有疑问。可是控制系统却需要清楚地了解该信息并且显然存储该信息的位置是楼层对象。然而，实际的楼层是不存储任何信息的，即使允许用拟人的做法，仍无法得知是否该有一个未完成的调用。同样地，实际的调用按钮同样无法得知自己是否曾被按下（不同于当前正被按下），按钮没有记忆。解系统的按钮对象负责处理这一点。

总之，随着分析的不断深入，自然会涉及更多的关于问题域类的内容，而对于一个控制系统，首要的问题则是类的行为。面向对象分析提供了状态图作为行为建模的工具，图 3-59 所示的是一个电梯电机的例子。

虽然该图用简练的方式把握住了电机的重要特征，但还有更多需要讨论的（例如关于加速与减速），也明确要求附随的解释说明（对于这点面向对象分析提供很少指导）。由此看来，用这种方式建模问题子域的行为在本质上与面向对象没什么关系。任何一种分析方法都应能够提供这样一个模型。

需要注意的是，图 3-59 确实涉及的是真正的电机。从图上可以看出，即使当电梯是在高速运行中，也有可能不依赖电机速度而对极性加以反转（或改变方向）。虽然这会导致灾难性的后果，但改变极性却是电机所固有的性质。这就难怪有中止这类行为的发生的需求，因此，可以尝试对行为方式建模。这就导致了图 3-60 所示的，一种截然不同的模型的出现。图 3-60 中的模型可能也会出现在控制器的规格说明书中，但那样会让人误以为（甚至可能是危险的）该图也可以作为问题域模型的一部分。

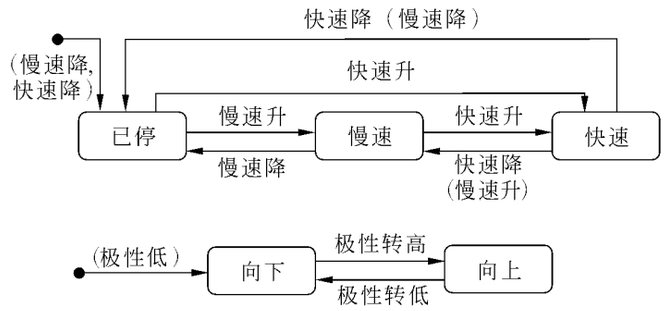


图 3-59　电梯电机状态图

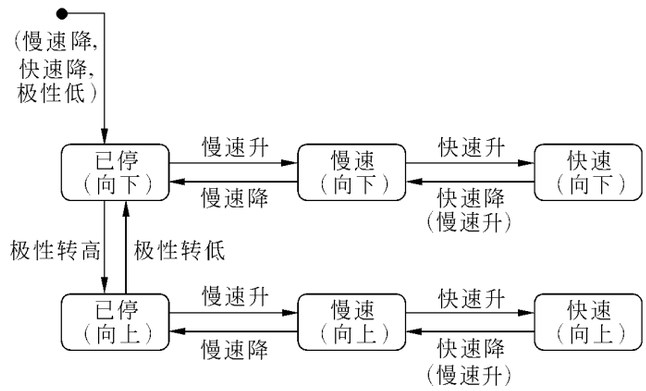


图 3-60　电梯电机状态图

描述完类的内在行为之后，不妨把注意焦点转向类间的相互关系上。图 3-61 展示了该如何描述这些关联。

可以将该类图与本节中的 ERD（图 3-58）作一比较，可以看出，除了按钮的继承关系，正如人们所预料的，二者在逻辑上非常地相似。

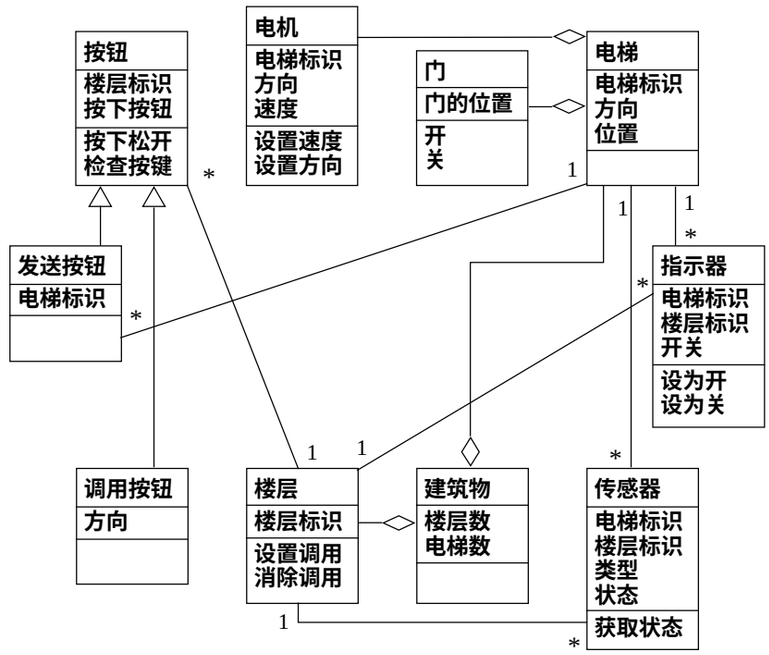


图 3-61　类的关联描述

然而，人们总是期望可以从面向对象获取更多，并且当然就能够表示出协作（也称作“使用”或服务）连接。为避免太多的混乱，将在一个单独的协作图中展示（见图 3-62）

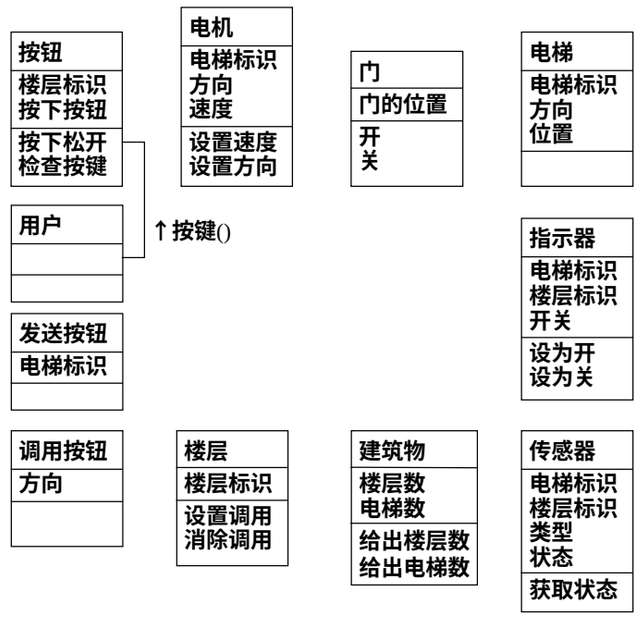


图 3-62　协作图

那些连接，但凑巧的是，大部分问题域对象类并不是直接地相互作用，至少不是通过消息传递的方式作用。展示一些的确存在于问题域中的其他形式的交互作用（例如，电机与电梯之间的机械交互作用以及电梯和传感器之间的机械交互作用），然而，这种做法却是真正地用消息传递机制来建模这些交互作用。

与讨论 SA 一样，尝试通过引入控制器对象来“解决”问题。这不仅起到了保持与其他对象相连接的媒介体的作用，而且系统需求也能够在其行为上得到反映。在问题域中，控制器其实是根本不存在的，这的确是个难题，并且对于控制器是否通过消息传递与外界进行通信仍然是存有争议的。这进一步突出了对解系统设计的偏爱，然而，正式地讲，无需沿着内部设计的道路走下去太远，图 3-63 展示了源自于这条路径的一个可能的输出。虽然在该模型上没有投入大量精力，但作为一个设计，该模型可以留下一些分析员所期望得到的结果。当然应把握的要点是，既然是一个设计，描述的是解系统而不是问题域。

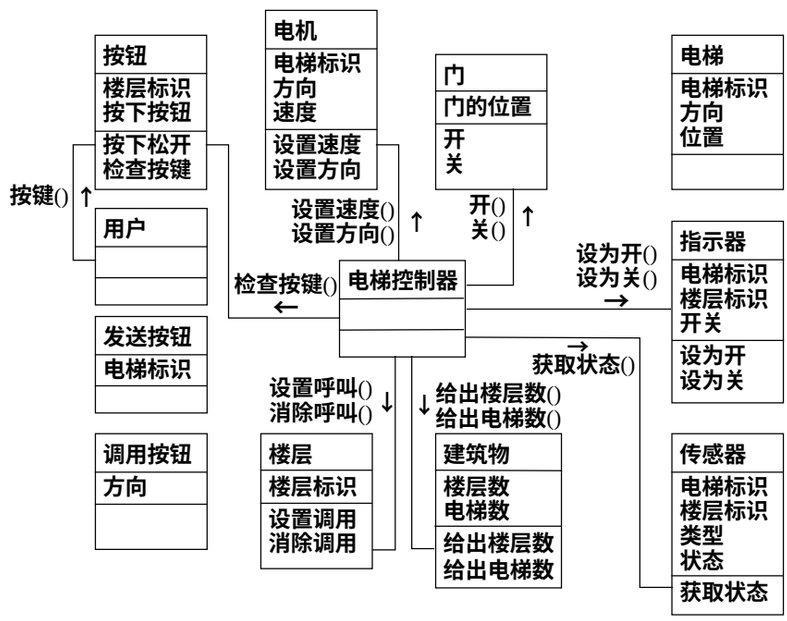


图 3-63　输出模型

或许所有这一切的主要问题就是源于许多需要了解的有关问题域的信息仍未被记录在案。例如，何为电梯电机的操作特性，如何很快地加速或减速？尤其伤脑筋的是，在对象模型中，还很难找到可以记录实际需求的场所。该类信息必须以某种方式与系统构建者进行通信。但这似乎已超出 OOA 的范围了。

3.8.3　面向问题域的分析示例

对于 PDOA 而言，问题域始终是任务开展的出发点。首先假设新的系统由某个电梯制造商委托，该制造商希望获得一个全新的、基于软件的控制系统作为某个基于初步硬

件设计的电梯模型的一部分。

可以通过图 3-64 所示的上下文图获得对系统的初步认识，虽然在以前曾提到，上下文图确实会忽略掉某些与控制器没有直接交互的问题子域（诸如，电梯）。

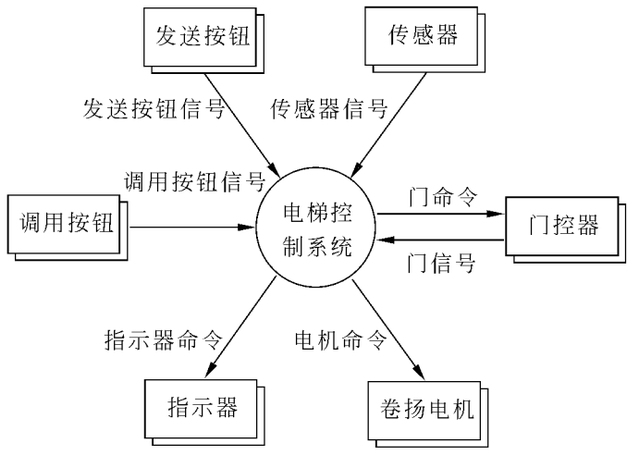


图 3-64　上下文图

从初始的需求获取记录做起，或许能够标识出大部分（即使不是全部）相关的问题子域，这可以通过上下文图来完成，或者直接开发出一个如图 3-56 所示的问题框架模型。

注意该模型如何精确地反映了所获取的事实，如在电梯控制器（机器）与电机和传感器之间存在连接，但与电梯之间却没有直接的连接。当然检查各问题子域是否具有了正确的特性也是可能的。在本案例中，用户是顺从性的，其他子域则是反应性的，或者是可编程的，这对于受控域是非常理想的结果。

顺便要指出的是，如果电梯的硬件没有预先定义并且所要解决的需求工程是有关某个完整的电梯系统，那么问题以及问题框架是截然不同的。在这种情况下，问题就是关于在某一建筑物内垂直地移动人员，而必须使用电梯这一情况实际只是一个设计上的限制。问题域包括潜在的用户（和一些相关情况，如用户的大小、重量，可以接受的垂直加速度，可以接受的气压变化率，移动速度等）、建筑物（楼层数等）以及期望的吞吐率。

一旦将系统确定为一个控制问题，即可致力于研究相关的指导原则（见表 3-10），以明确问题域的哪些方面是相关的，以及哪些建模技术（如果有的话）可能有利于问题的处理。以表 3-10 为指南，继续进行深入的需求获取工作。应当注意的是，并不需要过于严格按照表中所述的步骤进行，该表仅仅作为一种指导，而随意地调整每个条目的次序以及将某些元素组合起来，都是完全可以的。然而，出于示例的目的，下面给出一个有关表 3-10 中每个条目如何得以实现的例子。

在使用结构化方法所得到的模型中，数据模型是非常“普通”的一个，并且使用 ERD 即可很方便地予以记录。除了受控域的元素，还包括实体，如建筑物、楼层以及电梯升降机井等。虽然这些实体从任何意义上说都无法予以控制，但系统却需要“了解”有关这些实体的一些情况（如，有多少个电梯升降机井）。

表 3-10 需求文档

控制问题——需求文档	
内 容	（部分）相关技术
受控域中相关子域的数据模型，如果有的话（注意该内容是可选的）	ERD 和 DD
受控域中每个子域的特征及其内在行为，包括因果定律以及由子域完成/履行的动作/事件	文本，ELH，FSM，决策表
共享现象，解系统通过该现象能够对受控域进行监视	文本（事件列表）
受控域中的动作，解系统可以对其进行初始化	文本（动作列表）
由任何连接域（因太微不足道，毋须单独建档）引入的失真和延迟	文本
行为规则（也就是说，受控域作为一个整体如何表现行为）	文本，FSM，决策表

接下来是集中精力关注问题子域的描述。在本案例中，最棘手的问题域或许就是明确哪些信息应该记录，而对此问题的最佳答案就是，考虑清楚系统开发者需要了解些什么信息。电梯卷扬电机被认为是最复杂的子域之一，描述如下：

卷扬电机

每台卷扬电机都有三条控制线：慢速、快速和极性——并根据决策表（表 3-11）进行操作。在快速模式下，电机以 1.2m/s（+/–10%）的速度移动电梯。在慢速模式下，电机以 0.3m/s（+/–10%）的速度移动电梯。电机机制本身确保了逐渐加速和减速，如表 3-12 所示。

表 3-11 卷扬电机决策表

慢 速	hi				Lo			
快 速	hi		lo		hi		lo	
极 性	hi	lo	hi	lo	hi	lo	hi	lo
快速卷上	¡Ì				¡Ì			
快速卷下		¡Ì				¡Ì		
慢速卷上			¡Ì					
慢速卷下				¡Ì				
停 止							¡Ì	¡Ì

表 3-12 卷扬电机机制

	时 间	距 离
静止到快速	2s	1.2m
快速到慢速	1s	0.75m
慢速到静止	1s	0.15m

（所有数字允许误差为+/–20%）

由此可见，使用决策表可以非常简洁地记录大多数必要的信息，当然对于每种情况选择适当的技术也是必不可少的。

或许在开发对于问题子域的描述的同时，还可以标识并用文档说明相关的共享现象（问题域与解系统之间的）。这类事件包括按键和向卷扬电机与电梯门发送信号。对这些

事件可以采用如表 3-13 和表 3-14 所示的表格来表示。

表 3-13　事件

事　　件
按下调用按钮
按下发送按钮
楼层传感器信号
…

表 3-14　动作

动　　作
设置电机变高
设置电机变高
设置指示器显示变高
…

下一个应注意的问题是在建立问题域与解系统之间连接时所造成的失真与延时，在本案例中，信号的稳定性就是一个值得关注的问题，而对此可以相应地做一个简单的声明：

所有的输入信号（也就是从端子至控制系统）稳定在指定范围内最多耗时 10 毫秒。

最后，必须记录下所有实际的需求，下面给出几个有关电梯系统的例子：

R1　电梯不允许移到项层之上或底层之下（如果电梯移到顶层之上或底层之下（约超出 l0cm），紧急关闭系统将停止电机，但这个关闭系统超出了本控制系统的范围）。

R2　电梯不能在快速模式下停下来，必须在停止之前至少在 1 秒钟内切换到慢速模式。

R3　电梯在移动时不能改变电机极性（这会损坏卷扬设备）。

R4　当电梯门打开时，电梯不能移动。

R5　正确设置停止延迟后，电梯会在正予以服务的楼层的+/–1.5cm 处停下来。

与以前的例子一样，该需求（分析）文档与采用“传统的”方法所得到的有着很大的不同。该需求文档完全着重于描述问题域并对需求作出陈述。同样地，在进入规格说明阶段之前，应对分析进行相应的验证。

本 章 小 结

需求分析是软件产品生存期中的一个重要阶段，其根本任务是确定用户对软件系统的需求。本章介绍了需求分析的目标与原则、方法及过程。对于需求分析的方法，重点介绍了结构化分析方法、面向对象分析方法和面向问题域的分析方法。接着介绍了需求分析的工具：SADT 和 PSL/PSA。其后详细地介绍了各种建模方法，如传统的软件建模、用例建模、面向对象建模方法等，然后对 UML 语言进行了简单介绍，最后做了典型应用分析。

结构化分析方法（SA）以 E-R 图、DFD、DD 等描述手段为工具，用直观的图表和简洁的语言来描述软件系统的模型，获得了广泛的承认与应用。在分析结束后，这些图表又相互补充组成需求规格说明书，成为需求分析阶段的正式文档。抽象与分解，是结构化分析的指导思想。

面向对象分析方法（OOA）通过对对象、属性和操作的表示来对问题建模。

面向问题域分析方法（PDOA）较少强调去建模，更多地强调描述。作为全新的模型，问题框架被引入，并给出一些说明性的摘录以及有关分析过程的说明。

典型应用分析根据上述 3 种需求分析方法，分别举例阐明。

习 题 3

3.1 名词解释：可行性分析，需求收集，功能规格说明，建模方法，用例，UML。

3.2 需求分析的原则主要有哪些？

3.3 什么是结构化分析？“结构化”体现在哪里？

3.4 什么是面向对象分析？其主要思想是什么？

3.5 选择一个系统（例如工资管理系统、飞机订票系统、图书馆管理系统等），分别用 SA 方法和 OOA 方法进行分析，并给出分析模型。

3.6 试述需求分析的过程。

3.7 分析需求分析工具 SADT 和 PSL/PSA 的特点，并指出这些工具的缺陷。

3.8 数据流图有哪些用途？复杂系统为何要使用分层数据流图来描述？

3.9 数据流图和数据字典之间有什么关系？

3.10 试用系统流程图描述到医院看病的过程。

3.11 试考察某一学校的成绩管理系统，用实体-关系图描述系统的主要数据。

3.12 试考察某五星级酒店的电梯控制系统，用面向对象方法建立系统的对象模型、行为模型和功能模型。

3.13 有一小型图书馆管理系统，需要完成如下工作：

A．借书、还书和预约；

B．在系统中增加或删除一本图书；

C．找出最近借走某本图书的读者；

D．在系统中按书名或作者名或专业领域进行图书检索；

E．对过期未还图书的读者进行罚款处理。

系统要求如下：

A．一个读者总共可从本系统借 5 本图书；

B．有过期图书的读者不能从本系统借书；

C．每本图书的借阅期限为 3 个月。

试画出系统的 DFD 图（包括数据字典和加工说明）和用例图。

3.14 下列哪些用户需求不精确？对于不精确的需求，请给出相应的需求分析对策。

A．系统必须采用菜单来驱动。

B．系统要能够进行模糊查询。

C．系统运行时所占用的内存空间不能超过 256KB。

D．系统要有一定的安全保密措施。

E．系统崩溃时不能破坏用户数据。

F．系统响应速度要快。

3.15 实地考察学校的成绩管理系统，并用用例模型来描述。

第4章

系统设计[1,34,35]

4.1 系统设计的任务和过程

4.1.1 系统设计的任务

软件设计是一个把软件需求变换成软件表示的过程。最初这种表示描绘出软件的总的框架，然后进一步细化，在此框架中填入细节，加工成在程序细节上非常接近于源程序的软件表示。

4.1.2 系统设计的过程

将软件需求转化为数据结构和软件的系统结构，具体地讲，有以下几个方面：

1．制定规范

在进入软件开发阶段之前，首先应为软件开发组制定在设计时应该共同遵守的标准，以便协调组内各成员的工作。这些工作包括：

(1) 阅读和理解软件需求说明书，确认用户的要求能否实现。若不能实现，则需明确实现的条件，从而确定设计的目标，以及目标的优先次序。

(2) 根据目标确定最合适的设计方法。

(3) 规定设计文档的编制标准。

(4) 规定编码的信息形式，与硬件、操作系统的接口规约，命名规则等。

2．软件系统结构的总体设计

在需求分析阶段，已经从系统开发的角度出发，把系统按功能逐次分割成层次结构，使每一部分完成简单的功能且各个部分之间保持一定的联系，这就是所谓的功能设计。基于这个功能的层次结构把各个部分组合起来成为系统。总体设计包括：

(1) 将系统按功能划分成模块的层次结构。

（2）确定每个模块的功能。

（3）确定模块间的调用关系。

（4）确定模块间的接口。

（5）评估模块划分的质量及导出模块结构的规则。

3．处理设计

（1）确定系统的功能需求所必需的算法，评估算法的性能。

（2）确定系统的性能需求所必需的算法和模块间的控制方式。

（3）确定外部信号的接收发送形式。

4．数据结构设计

确定输入、输出文件的详细的数据结构，并确定逻辑数据及其操作，确定对逻辑数据结构所必需的那些操作的程序模块。若需要操作系统或调度程序接口所必须的控制表等数据时，确定其详细的数据结构和使用规则。

5．可靠性设计

可靠性设计也叫做质量设计。软件可靠性是指程序和文档的正确性、容错性的高低。由于软件维护往往会产生新的故障，所以要求经常在软件开发期间应当尽可能找出更多的差错，并在软件开发的一开始就要确定软件可靠性和其他质量指标，考虑相应措施，使得软件易于修改和易于维护。

6．编写系统设计阶段的文档

系统设计阶段应完成的文档包括：系统设计说明书，数据库设计说明书，用户手册，制定初步的测试计划。

7．系统设计评审

软件设计的最终目标是节省开发费用，降低资源消耗，缩短资源开发的条件，选择能够赢得较高的生产效率、缩短开发时间的条件，选择能够赢得较高的生产效率、较高的可靠性和可维护性的方案。

4.2　系统设计的基本原则

4.2.1　软件设计

软件设计的概念成为软件设计的基础，成为软件设计人员应用更复杂的设计方法的基础。

1．自顶向下，逐步细化

这是 Wirth 提出的设计策略。将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层循环分解，直到用程序设计语言的语句能够实现为止，从而最后确立整个系统的体系结构。

2．软件结构

软件结构包括两部分：一是程序模块的层次结构；二是数据结构。通常，软件的体系结构通过一个划分过程来完成。该过程从需求分析确立的目标系统的模型出发，对整个问题进行分析分割，使其每一个部分用一个或几个软件成分加以解决，整个问题就得

到解决。

3．程序结构

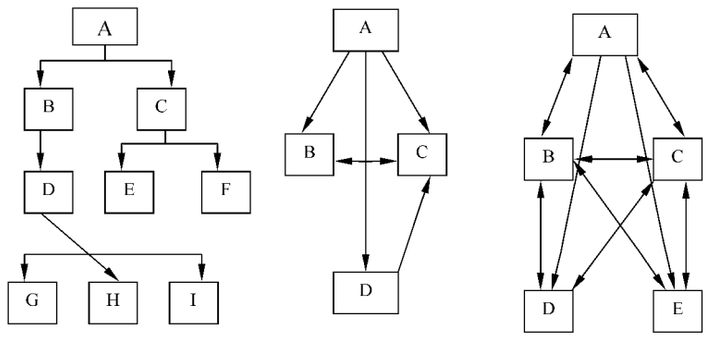
程序结构表明了程序各个部件的组织情况，通常是树状或网状结构，并蕴含了在程序控制上的层次关系。

1）程序的树状结构和网状结构

在树状结构中，位于最上层的根部是顶层模块，是程序的主模块。与其联系的有若干下属模块，各下属模块还可以进一步引进更下一层的下属模块。

图 4-1(a)树状结构的特点是：整个结构只有一个顶层模块，而对于任何一个下属模块来说，只有同一层模块之间不发生联系。

图 4-1（b）和图 4-1（c）给出了网状结构的两个例子。



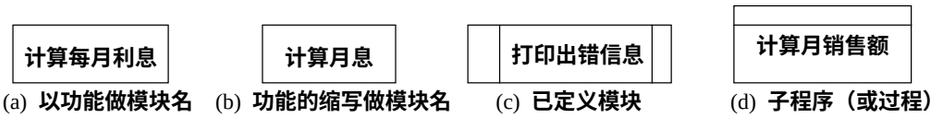
(a) 树状结构　　(b) 网状结构　　(c) 网状结构

图 4-1　程序的树状结构和网状结构

2）结构图 SC

结构图是精确表达程序结构的图形表示方法。结构图清楚地反映出程序中模块之间的层次调用关系和联系：它不仅严格地定义了各个模块的名字、功能和接口，而且还集中的反映了设计思想。下面给出结构图的主要内容。

（1）模块——模块的表示应当能够表明该模块的功能。矩形表示模块，现成的模块用双纵边矩形框表示。如图 4-2 所示。



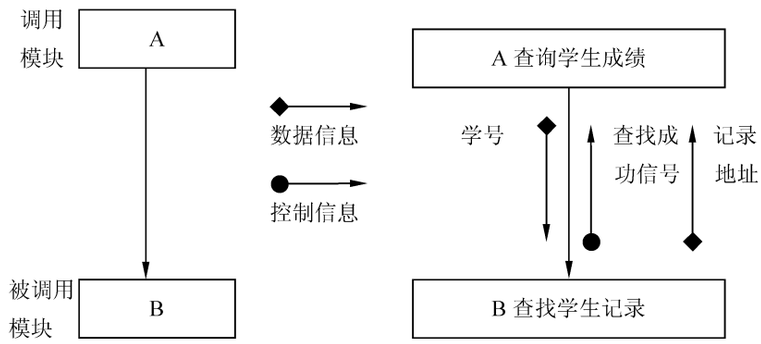
(a) 以功能做模块名　(b) 功能的缩写做模块名　(c) 已定义模块　(d) 子程序（或过程）

图 4-2　模块的表示

（2）模块的调用关系和接口——在结构图中，两个模块间单向箭头联结表示模块的调用关系和接口。箭头从调用模块指向被调用模块。图 4-3（a）表示模块 A 调用了模块 B。

（3）模块间的信息传递——当一个模块调用另一个模块时，调用模块把数据或控制

信息传送给被调用模块，以使被调用模块能够运行。为了表示模块之间传递的数据或控制信息，在联结模块的箭头旁边给出短箭头，并用尾端带有菱形的短箭头表示数据信息，用尾端带有实心圆的短箭头表示控制信息。如图 4-3（b）所示。



(a) 模块调用关系　　(b) 模块间接口的表示

图 4-3　模块间的调用关系和接口表示

（4）两个辅助符号——当模块 A 有条件地调用另一个模块 B 时，在模块 A 的箭头尾部标以一个菱形符号。当一个模块 A 反复地调用模块 C 和模块 D 时，则在调用箭头尾部标以一个弧行符号。如图 4-4 所示。

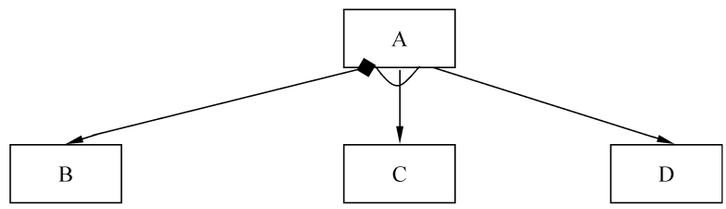


图 4-4　条件调用和循环调用的表示

（5）结构图的形态特征——图 4-5 是一个结构图的示例。在多层次的结构图中，其模块结构的层次数称为结构图的深度，同一层模块的最大模块数称为结构图的宽度。扇出定义为一个给定模块调用的模块个数，扇入定义为调用一个给定模块的模块个数。在图 4-5 中，结构图的深度为 5，宽度为 7，模块 M 的扇出为 3，模块 T 的扇入为 4。

3）数据结构

数据结构设计应确定数据的组织、存取方式、相关程序，以及信息的不同处理方法。基本的数据结构可以组合以构成更复杂的数据结构。必须注意，数据结构和程序结构一样，可以在不同的抽象层次上表示。

4）模块化

整个软件被划分为若干单独命名和可编址的部分，称之为模块。一个大而复杂的软件系统能够划分为易于理解的比较单纯的模块结构。所谓比较单纯，是指模块和其他模块之间的接口应尽可能独立。

5）抽象化

当对软件系统进行模块设计时，可以有不同的抽象层次。在最高的抽象层次上，使用问题所处环境的语言概括地描述问题的解法。而在较低的抽象层次上，则采用过程化的方法。

（1）过程的抽象：在软件工程中，从系统定义到实现，每进展一步都可以看作是对软件解决方法的抽象化过程的依次细化。

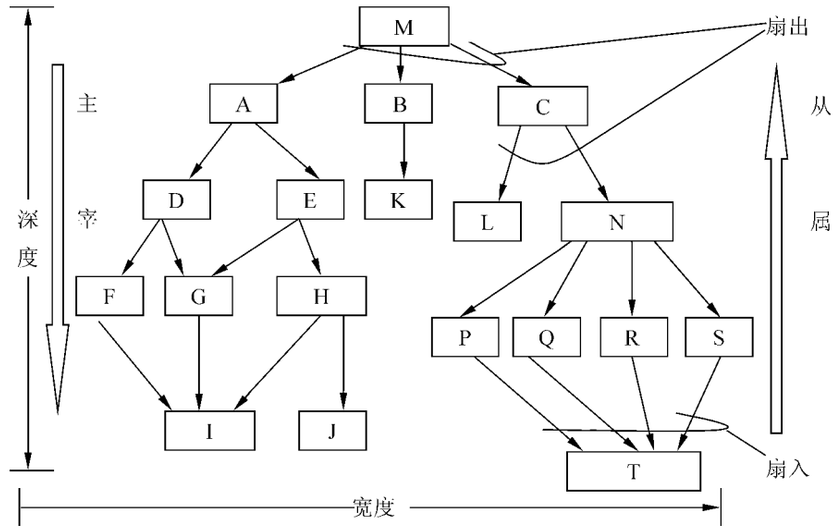


图 4-5　结构图示例

（2）数据抽象：与过程抽象一样，允许设计人员在不同层次上描述数据对象的细节，可以通过定义与该数据对象相关的操作来规定数据对象。

（3）控制抽象：控制抽象可以包含一个程序控制机制而无须规定其内部细节。

6）信息隐蔽

由 Parnas 方法提倡的信息隐蔽是指，每个模块的实现细节对于其他的模块来说是隐蔽的。即模块中所包含的信息不允许其他不需要这些信息的模块使用。

4.2.2　模块设计

模块化设计带来的好处：

（1）降低了系统的复杂性，使系统容易修改；

（2）推动了系统各个部分的并行开发，提高了软件的生产效率。

模块设计包括了模块、模块独立性、耦合性和内聚性等因素。

模块又称“组件”，一般指用一个名字就可调用的一段程序。模块的独立性是指软件系统中每个模块只涉及软件要求的具体的子功能，而与软件系统中其他模块的接口比较简单。

一般用两个准则度量模块独立性，即模块间的耦合性和模块的内聚性。

耦合性是程序结构各个模块之间相互关联的度量，取决于各个模块之间接口的复杂

程度、调用模块的方式以及哪些信息通过接口。一般模块之间的连接方式有 7 种，分别是非直接耦合、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。如图 4-6 所示。

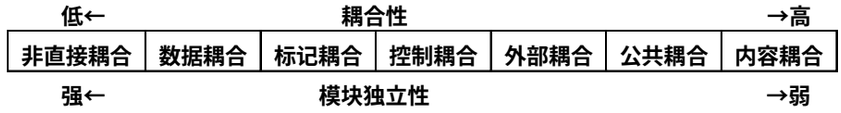


图 4-6　耦合性

一个内聚性高的模块应当只做一件事。内聚也有 7 种，分别是功能内聚、信息内聚、通信内聚、过程内聚、时间内聚、逻辑内聚和巧合内聚。如图 4-7 所示。

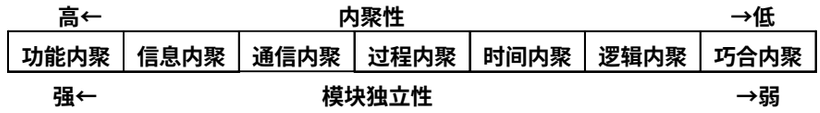


图 4-7　内聚性

4.2.3　结构设计

结构化设计方法就是根据问题域的数据来定义一组不同的“ 映射 ”，利用这些映射把数据流图变换成软件结构。

结构设计方法是基于模块化、自顶向下细化、结构化程序设计等基础上发展起来的。该方法实施的要点是：

（1）首先研究、分析和审查数据流图。

（2）然后根据数据流图决定问题类型是变换型还是数据型。

（3）由数据流图推导出系统的初始结构图。

（4）利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。

（5）修改和补充数据字典。

（6）制定测试计划。

4.3　面向数据流图的设计方法

4.3.1　典型的系统结构图

1．在系统结构图中的模块

在系统结构图中不能再分解的底层模块称为原子模块。如果一个软件系统的全部实际加工都是由底层的原子模块来完成，而其他所有非原子模块仅仅执行控制或协调功能，这样的系统就是完全因子分解的系统。在系统结构图中有 4 种类型的模块：传入模块，

传出模块，变换模块，协调模块。如图 4-8 所示。

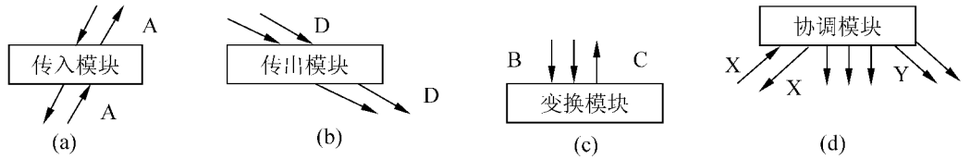


图 4-8　系统结构图的 4 种模块类型

图 4-8（a）是从下属模块取得数据的传入模块；图 4-8（b）是从上级模块获得数据的传出模块；图 4-8（c）是变换模块，是从上级模块取得数据，进行特定的处理，转换成其他形式，再传送给上级模块；图 4-8（d）是协调模块，是对所有下属模块进行协调和管理的模块。

2．变换型系统结构图——典型的系统结构图之一

变换型数据处理问题的工作过程大致分为 3 步，即取得数据，变换数据和给出数据。这 3 步反映了变换型问题数据流图的基本思想。其中，变换数据是数据处理过程的核心工作。

变换型数据流图由取得数据、变换数据、给出数据 3 个部分组成，如图 4-9 所示。变换型系统结构图由输入、中心变换和输出 3 个部分组成，如图 4-10 所示。

图 4-9　变换型数据流图

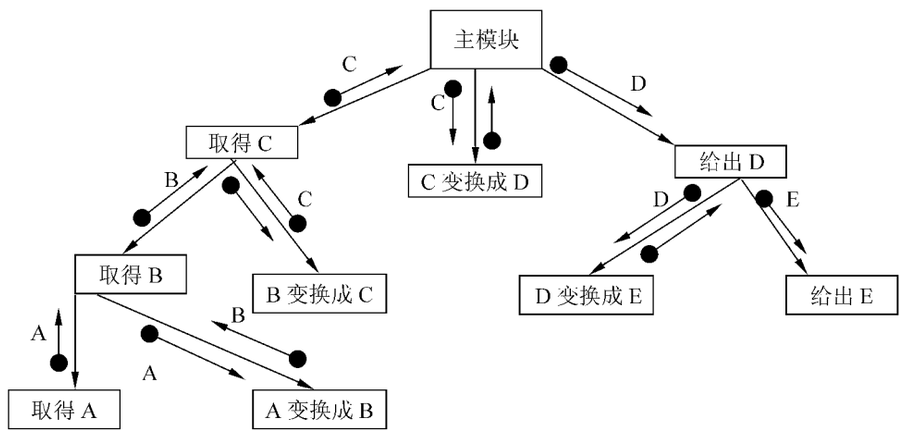


图 4-10　变换型的系统结构图

3．事务型系统结构图——典型的系统结构图之二

一项事务，根据事务处理的特点和性质，选择分派一个适当的处理单元，然后给出结果。在事务型系统结构图中，事务中心模块按接受的事务的类型，选择某一个事务处

理模块执行。各个事务处理模块是并列的，依赖于一定的选择条件，分别完成不同的事务处理的工作。每个事务处理模块可能要调用若干个操作模块，而操作模块有可能调用若干个细节模块。由于不同的事务处理模块可能有共同的操作，所以某些事务处理模块又可以共享一些细节模块。同样，不同的操作块可以有相同的细节，所以，某些操作模块又可以共享一些细节模块。事务型系统的结构图可以有多种不同的形式。如图 4-11 所示。

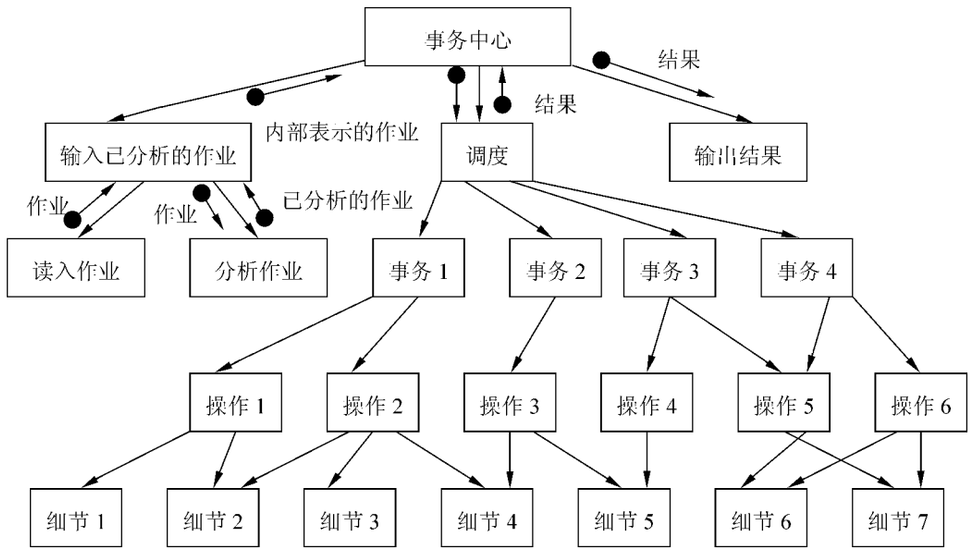


图 4-11 事务型系统结构图

4.3.2 变换分析

运用变换分析方法建立初始的变换型系统结构图，然后对其做进一步的改进，最后得到系统的最终结构图。变换分析方法由以下 4 步组成。

1．重画数据流图

出发点是描述系统中的数据是如何流动的，以需求分析阶段得到的数据流图为基础重画数据流图时，可以从物理输入到物理输出，或者相反，还可以从顶层加工框开始；在图上不要出现控制逻辑；省略每一个加工框的简单例外处理。

2．在数据流图上区分系统的逻辑输入、逻辑输出和中心变换部分

从数据流图的物理输入端开始，一步一步向系统的中心移动，一直到遇到的数据流不再被看作是系统的输入为止，则其前一个数据流就是系统的逻辑输入。也就是说，逻辑输入就是离物理输入最远的，但仍被看作是系统输入的数据流。类似地，从物理输出端开始，一步一步地向系统的中心移动，可以知道离物理输出端最远的，但仍被看作是系统输出的数据流，就是系统的逻辑输出。从物理输入端到逻辑输出，构成输出部分；夹在输入部分和输出部分就是中心变换部分。中心变换部分是系统的中心加工部分。从输入设备获得的物理输入经过编辑、数制转换、格式变换、合法性检查等一系列预处理，最后才变成逻辑输入传送到中心变换部分。如图 4-12 所示。

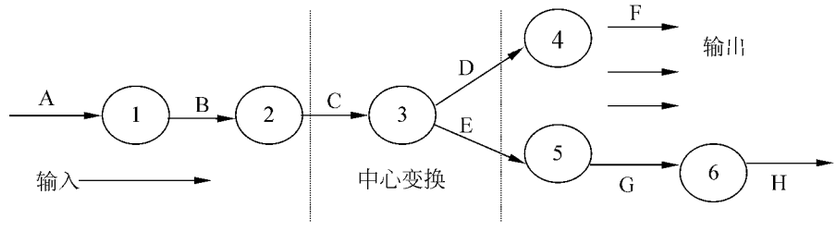


图 4-12　数据流图中的输入、中心变换与输出部分

3．进行一级分解，设计上层模块

自顶向下设计的关键是找出系统树形结构图的根或顶层模块。程序结构的第一层可这样来设计：为每个逻辑输入设计一个输入模块，为主模块提供数据；为每个逻辑输出设计一个输出模块，将主模块提供的数据输出；为中心变换设计一个变换模块，将逻辑输入转换成逻辑输出。如图 4-13 和图 4-14 所示。

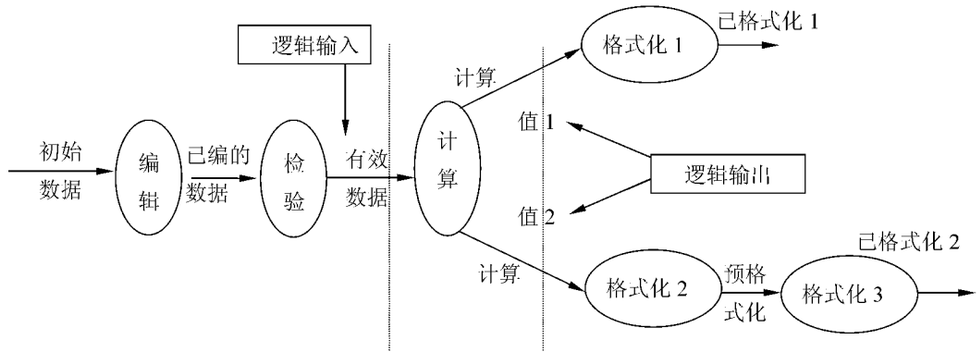


图 4-13　事务型问题数据流图

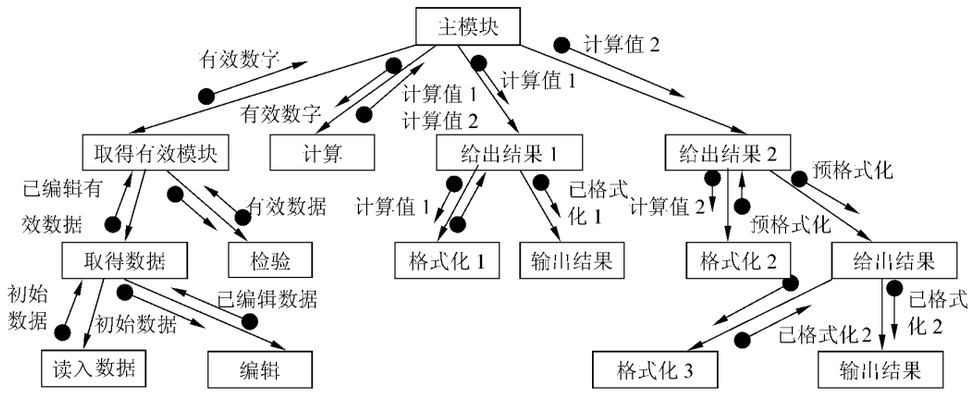
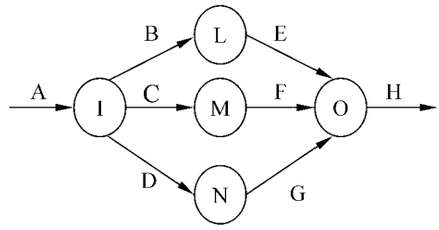


图 4-14　事务型问题的数据流图导出结构图

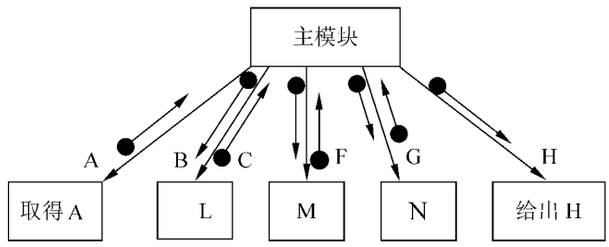
4．进行二级分解，设计输入、输出和中心变换部分的中、下层模块

这一步工作是自顶向下，逐层细化，为每一个输入模块、输出模块设计从属模块。

设计下层模块的次序是任意的。但一般是先设计输入模块的下层模块。输入模块的功能是为调用该模块的上级模块提供数据，所以必须要有一个数据来源。因而输入模块必须有两个下属模块：一个是接收数据；另一个是把这些数据变换成上级模块所需要的数据。但调用输入模块的上级模块接收数据，用以输出，因而也应当有两个下属模块：一个是将上级模块提供的数据变换成输出的形式；另一个是将数据输出。设计中心变换模块的下层模块没有通用的方法，一般应参照数据流图的中心变换部分和功能分解的原则来考虑如何对中心变换模块进行分解。如图 4-15 所示。



(a) 数据流图



(b) 结构图

图 4-15 事务型问题导出的系统结构图

4.3.3 事务分析

事务是某种作业数据流，可以引发一个或多个处理，这些处理能够完成作业要求的功能。事务分析从分析数据流开始，自顶向下，逐层分解，建立系统的结构图。具体步骤是：

① 识别事务源——利用数据流图和数据字典，从问题定义和需求分析的经过中，找出各种需要处理的事务。

② 规定适当的事务型结构——在确定了该数据流图具有事务型特征之后，根据模块划分理论，建立适当的事务型结构。

③ 识别各种事务和事务定义的操作——从问题定义和需求分析中找出的事务及其操作所必需的全部信息，也可以在问题定义和需求分析规格说明中找到。

④ 构造公用模块——在事务分析过程中，如果不同事务的一些中间模块可由具有类似的语法和语义的若干低层模块组成，则可以把这些低层模块构造成公用模块。

⑤ 建立事务处理模块——如果发现在系统中有类似的事务，可以把类似的事务组

成一个事务处理模块。

⑥ 规定事务处理模块的全部下层操作模块——下层操作模块的分解方法类似于变换分析。

⑦ 规定操作模块的全部细节模块——对于大型系统的复杂事务处理，可能有若干层细节模块。另外，尽可能使类似的操作模块共享公用的细节模块。

通常利用以变换分析为主，事务分析为辅的方式进行软件结构设计。在结构设计时，首先利用变换分析方法把软件系统分为输入、中心变换和输出 3 个部分设计上层模块，即主模块和第一层模块。然后根据数据流图各部分的结构特点，适当地利用变换分析或事务分析，可以得到初始系统结构图的某个方案。

4.3.4 软件模块结构的改进

(1) 模块功能的完善化；
(2) 消除重复事物，改善软件结构；
(3) 模块的作用范围应在控制范围之内；
(4) 尽可能减少高扇出结构，随着深度增大扇入；
(5) 模块的大小要适中；
(6) 设计功能可预测的模块，但要避免过分受限制的模块；
(7) 软件包应满足设计约束和可移植性。

4.4 面向对象的设计方法

4.4.1 面向对象的基本概念和特征

面向对象是一种新的非常有效的程序设计，以对象为核心，表达具有相同属性和服务对象的集合之间相似性的机制。面向对象=对象+类+继承+通信，如果一个软件是使用这样 4 个概念设计和实现的，则认为这个软件系统是面向对象的。

面向对象的特性有以下两种。

(1) 模块化、信息隐蔽和数据抽象。对象的一个重要特点是封闭性。对象是数据和过程紧密结合在一起构成的整体，具有很强的独立性，是实现模块的理想机制。数据抽象是信息隐蔽的一种表现形式，一个类就是一种抽象的数据类型。

(2) 继承性。一个类的上层可以有父类，下层可以有子类。这种层次结构的一个重要性质是继承性，一个类直接继承其父类的全部描述。继承性使得相似的对象可以共享代码和数据。

4.4.2 面向对象的技术要点

(1) 方法的惟一性；
(2) 从生存期的一个阶段到下一阶段的高度连续性，即生存期后一阶段的成果只是

前一阶段成果的补充和修改；

（3）面向对象分析（OOA）、面向对象设计（OOD）和面向对象程序设计(OOP)集成到生存期的相应阶段。

把数据和过程作为相互独立的实体，数据用于表达实际问题中的信息，过程用于处理这些数据，基本原理是：对问题领域进行自然的分解，按照人们习惯的思维方式建立问题领域的模型，设计出尽可能直接自然地表现问题求解的软件。

4.4.3　面向对象分析模型

通过 OOA 建立的系统模型是以概念为中心的，因此称为概念模型。这样的模型由一组相关的类组成。OOA 可以采用自顶向下的方法，逐层分解建立系统模型，也可以自底向上地从已经定义的基本类出发，逐步构造新的类。

构造和评审 OOA 概念模型的次序由 5 个层次组成。这 5 个层次不是构成软件系统的层次，而是分析过程中的层次，也可以说是问题的不同侧面。每个层次的工作都为系统的规格说明增加了一个组成部分。当 5 个层次的工作全部完成时，OOA 的任务也就完成了。这 5 个层次是：类与对象、属性、服务、结构和主题。

4.5　面向对象软件设计模型

4.5.1　设计模式描述

面向对象设计针对于实现有关的因素而发展面向对象分析的 5 个活动，包括问题域，人机交互，任务管理和数据管理等 4 个部分的设计，从横向看是 4 个部分，纵向看是 5 个层次活动，即为主体层、类与对象层、结构层、属性层和服务层。活动不必按顺序进行。面向对象设计的问题域部分是把面向对象分析模型直接拿来，针对实现的要求进行必要的增补和调整，其他 3 个部分则是面向对象分析阶段考虑的，全部在面向对象设计阶段建立。

4.5.2　设计模式的分类

由 Rumbaugh 等人提出的对象模型技术，把分析时收集的信息构造在 3 类模型中，即对象模型、功能模型和动态模型。如图 4-16 所示。

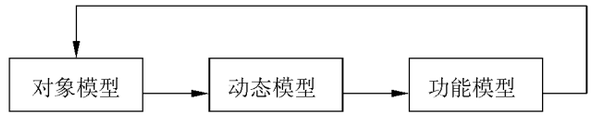


图 4-16　分析阶段 3 个模型之间的关系

对象模型的作用是描述系统的静态结构，包括构成系统的类和对象，类和对象的属性

和操作，以及类和对象之间的关系。事实上，这个模型可以看作扩充的实体-关系模型（E-R）。

动态模型描述时序和改变的状态。动态模型着重于系统的控制逻辑，包括状态图和事件追踪图。

1．状态图

状态图是一个状态和事件的网络，侧重于描述每一类对象的动态行为。在状态图中，状态是对某一时刻中属性特征的概括。而状态迁移则表示这一类对象在何时对系统内外发生的哪些事件做出何种响应。如图 4-17 所示。

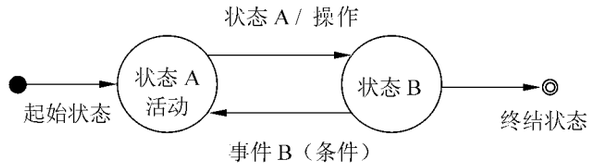


图 4-17　状态图的表示方法

2．事件

事件从一个对象向另一个对象传送信息。有些事件类可能传送的是简单的信号“要发生某件事”。而有些事件类则可能传送的是数据值。

3．事件追踪图

事件追踪图侧重于说明发生于系统执行过程中的一个特定“场景”。场景也叫做脚本，是完成系统某个功能的一个事件序列。

功能模型是模型化三脚架的第三只脚。功能模型由多个数据流图组成，指明从外部输入，通过操作和内部存储，直到外部输出，这整个的数据流情况。功能模型还包括了对象模型内部数据间的限制。数据流图不指出控制或对象的结构信息，这些结构信息包含在动态模型和对象模型中。图 4-18 给出了数据流图的表示方法。

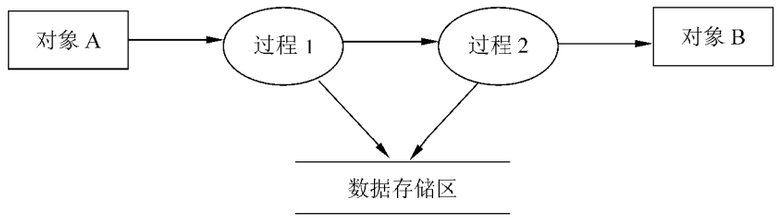


图 4-18　数据流图的表示方法

4.6　模型-视图-控制器框架

4.6.1　MVC 模式

MVC(model-view-controller)由 Trygve Reenskaug 提出，首先被应用在 SmallTalk-80 环境中，是许多交互和界面系统的构成基础。Microsoft 的 MFC 基础类也遵循了 MVC

的思想。MVC 结构是为那些需要为同样的数据提供多个视图的应用程序而设计的，很好的实现了数据层与表示层的分离。MVC 作为一种开发模型，通常用于分布式应用系统的设计和分析中，以及用于确定系统各部分间的组织关系。对于界面设计可变性的需求，把交互系统的组成分解成模型、视图、控制器 3 种部件。

模型部件是软件所处理问题逻辑在独立于外在显示内容和形式情况下的内在抽象，封装了问题的核心数据、逻辑和功能的计算关系，独立于具体的界面表示和 I/O 操作。

视图部件把表示模型数据及逻辑关系和状态的信息及特定形式展示给用户，从模型获得显示信息，对于相同的信息可以有多个不同的显示形式或视图。

控制部件处理用户与软件的交互操作，其职责是控制提供模型中任何变化的传播，确保用户界面与模型间的对应联系；接受用户的输入，将输入反馈给模型，进而实现对模型的计算控制，使模型和视图协调工作。通常一个视图具有一个控制器。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其他依赖于这些数据的视图都应反映这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，实现显示的更新。这实际上是一种模型的变化-传播机制。

4.6.2 MVC 中的模型类、视图类和控制类

MVC 中的模型类、视图类和控制类如图 4-19 所示。

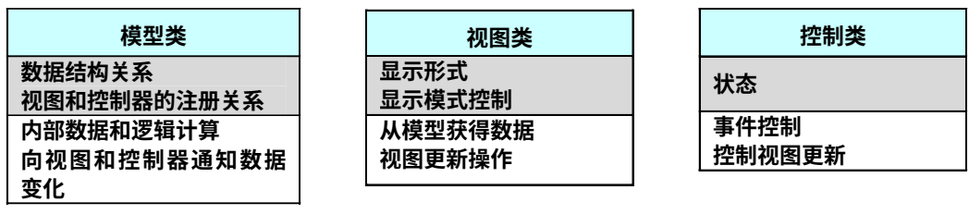


图 4-19 MVC 中的模型类、视图类和控制类

1．模型类

模型包含应用问题的核心数据、逻辑关系和计算功能，封装所需的数据，提供完成问题处理的操作过程。控制器依据 I/O 的需要调用这些操作过程。模型还为视图获取显示数据而提供了访问其数据的操作。这种变化-传播机制体现在各个相互依赖部件之间的注册关系上。模型数据和状态的变化会激发这种变化-传播机制，它是模型、视图和控制器之间联系的纽带。

2．视图类

视图通过显示的形式，把信息转达给用户。不同视图通过不同的显示，来表达模型的数据和状态信息。每个视图有一个更新操作，它可被变化-传播机制所激活。当调用更新操作时，视图获得来自模型的数据值来更新显示。在初始化时，通过与变化-传播机制的注册关系建立起所有视图与模型间的关联。视图与控制器之间保持着一对一的关系，每个视图创建一个相应的控制器。视图提供给控制器处理显示的操作。因此，控制器可

以获得主动激发界面更新的能力。

3．控制类

控制器通过时间触发的方式，接受用户的输入。控制器如何获得事件依赖于界面的运行平台。控制器通过事件处理过程对输入事件进行处理，并为每个输入事件提供相应的操作服务，把事件转化成对模型或相关视图的激发操作。

如果控制器的行为依赖于模型的状态，则控制器应该在变化-传播机制中进行注册，并提供一个更新操作。这样，可以由模型的变化来改变控制器的行为，如禁止某些操作。

4.6.3 MVC 的实现

实现基于 MVC 的应用需要完成以下工作，如图 4-20 所示。

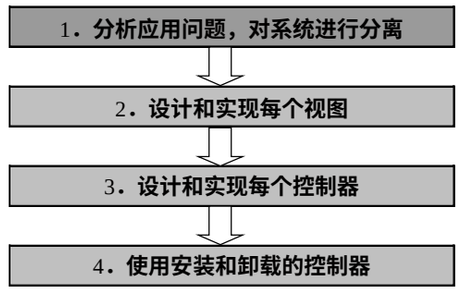


图 4-20　MVC 的实现

1．分析应用问题，对系统进行分离

分析应用问题，分离出系统的内核功能、功能的控制输入、系统的输出行为 3 大部分。设计模型部件使其封装内核数据和计算功能，提供访问显示数据的操作，提供控制内部行为的操作以及其他必要的操作接口。以上形成模型类的数据构成和计算关系。这部分的构成与具体的应用问题紧密相关。

2．设计和实现每个视图

设计每个视图的显示形式，从模型中获取数据，将数据显示在屏幕上。

3．设计和实现每个控制器

对于每个视图，指定对用户操作的响应时间和行为。在模型状态的影响下，控制器使用特定的方法接受和解释这些事件。控制器的初始化建立起与模型和视图的联系，并且启动事件处理机制。事件处理机制的具体实现方法依赖于界面的工作平台。

4．使用可安装和卸载的控制器

控制器的可安装性和可卸载性，带来了更高的自由度，并且帮助形成高度灵活性的应用。控制器与视图的分离，支持了视图与不同控制器结合的灵活性，以实现不同的操作模式，例如对普通用户、专业用户或不使用控制器建立的只读视图。这种分离还为在应用中集成新的 I/O 设备提供了途径。

4.7　系统设计说明书

计算机软件工程和硬件工程可以看作一门更广的科学———“计算机系统工程”内的活动。这些活动所要做的都是想要按一定的次序开发基于计算机的系统。

基于计算机的系统是“某些元素的一个集合，这些元素被组织起来以实现某种方法、过程或借助处理信息进行控制。”这些元素是文档、过程、硬件、软件、人和数据库。

基于计算机的系统本身可以成为一个更庞大的基于计算机系统中的一个元素，并称为那个大系统的宏元素。这样的系统可能十分复杂。例如，考虑一个工厂自动化系统。该系统基本上是一个由若干系统组成的层次结构，如图 4-21 所示。

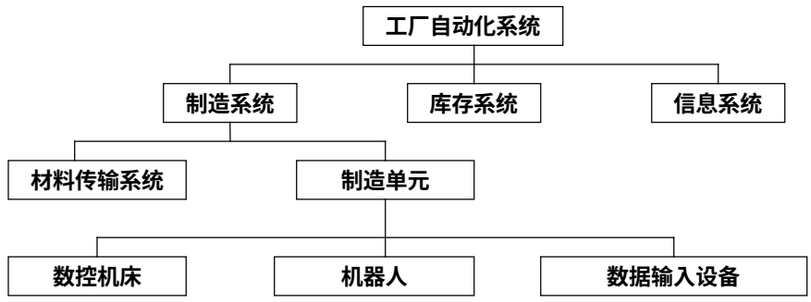


图 4-21　若干系统组成的层次结构

为了开发系统模型，使用一个“结构模板”，如图 4-22 所示。

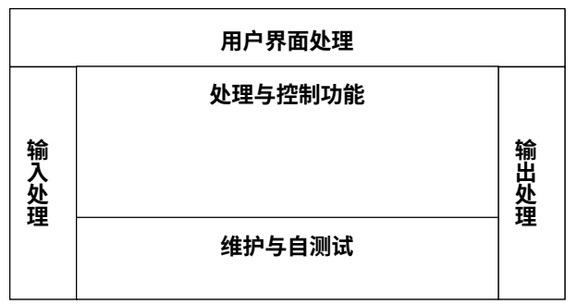


图 4-22　结构模板

结构模块能够帮助分析员建立一个细节的层次结构，其中位于层次结构顶层的是结构上下文图（ACD），ACD 定义了系统使用的所有数据的外部产生者，由系统建立的所有数据的外部使用者，以及通过接口进行通信或实施维护与自测试的所有实体。结构图的规格说明（ADS）给出有关每个子系统的信息及各子系统之间的信息流。ADS 还说明了每个子系统的“系统模块描述”。系统模块描述说明了系统要做什么，处理什么信息，与其他子系统又是如何接口的，等等。ADS 还包含一个“结构词典”，即在 ADS 中出现的每一个信息项的清单以及信息项的说明。

结构图规格说明中的最后一项是结构互连图（AID）。AID 和相应的规格说明描述了

信息是电学地传送，光学地传送，还是机械地传送。

4.8 典型应用分析

应用分析过程包括了对问题论域所需的类的模型化；但最终实现应用不只有这些类，还需要追加一些类。

4.8.1 类设计的目标

1．单一概念的模型

在分析与高层设计阶段，常常要使用多个类来表示一个“概念”。一般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繁荣类来表示一个概念。

2．可复用的“插接相容性”部件

人们希望所开发部件可以在未来的应用中使用。

3．可靠的部件

每个部件必须经过测试，但测试不够完备。然而，如果建立的是可复用的类，则通过测试确保部件的可靠性是绝对必要的。

4．可集成的部件

类的设计应尽量减少命名冲突，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。类结构提供的封装使得把概念集成到应用的工作变得很容易。

4.8.2 类设计的方针

1．信息隐蔽

软件设计通过信息隐蔽可增强抽象，并可保护类的存储表示不被抽象数据类型实例的用户直接存取。对其表示的惟一存取途径只能是界面。

2．消息限制

在类的数据表示中可以包括其他类似的实例。若类 A 的实例发送一个消息给类 B 的一个部件，而这个部件是类 B 的实现部分，则类 A 必须知道该部件的实现，并把这种知识包括到自己的实现中去。当变更类 B 的实现时，类 A 的实现也必须变更。

3．狭窄界面

前一方针限制了数据属性，使其对外部实例隐蔽。这些方针指明类的操作主要用于存取数据属性。然而，不是所有的操作都是公共的。

4．强内聚

类的操作中必须有存取函数简单地返回某些属性的值。

5．弱耦合

耦合程度部分依赖于所使用的分解方法。类 A 之所以依赖于类 B，是因为类 A 要求

类 B 提供服务。这个依赖性可以通过复制类 A 中的类 B 的功能来消除。但代码的复制减少了系统的灵活性并增加了维护的困难。继承结构损害了弱耦合的概念。

6．派生类当做派生类型

每个派生类应该当做基类的特殊化来开发，而基类所具有的公共界面成为派生类的共有界面的一个子集。

7．抽象类

每个继承结构的根类应当是目标概念的一个抽象模型。这个抽象模型生成一个类，不用于产生实例；它定义了一个最小的共有界面，许多派生类可以加到这个界面上以给出概念的一个特定视图。

4.8.3 通过复用设计类

利用既存来设计类，有 4 种方式：选择、分解、配置和演变。

1．选择

设计一个类最简单的方法是从既存的部件中简单地选择合乎需要的软件部件。这就是开发软件库的目的。一个面向对象开发环境应提供一个常用部件库。

2．分解

把一个类分成几个类，希望新标识的类容易实现或者已经存在。

3．配置

通过把相应类的实例声明为新类的属性来配置新类。

4．演变

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，不宜采用“选择”操作，但可以从一个既存类演变成一个新类，可以利用继承机制表示一般化-特殊化的关系。

4.8.4 计数器类设计的实例

计数器类是一个频繁使用的类，许多系统都需要计数器。一组数据的累加统计可以使用一个简单计数器组。

计数器类的设计可以具体说明在类的设计中常用的两种不同的方法。首先，设计一个类，体现所能考虑到的该类的所有特色。类设计的结果可能有一个界面，如图 4-23 所示。

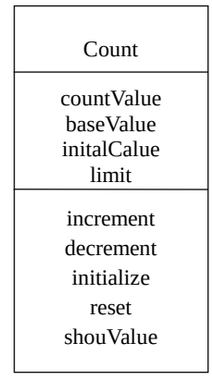


图 4-23 计数器设计的大类方法

这个设计的缺点是：类常常比需要的大而且可能降低系统效率，类的使用者可能不当心使用了不正确的方法，就会导致不正确的结果。这个设计的优点是：这一个部件将具有所有计数器的全部特色。

图 4-24 所示的设计使用了一种分层的渐增式方法来开发一组计数器。这个结构的根是一个最基本的类 Simple Counter。这个类有一个值，可以被初始化为开始值，可以增加，可以返回其当前

值。这个类起着一个标准的基本计数器的作用，它不会因偶然事故被置零。

第二个类 Unidirectional Counter（单向计数器）是在 Simple Counter 上建立起来的，增加了置数的能力，随着每次“ 撞击 ”，计数器加一个步长。由于在这个类中所使用的步长可以是正的也可以是负的，因此，单向计数器可以加或减，而一给定的实例只可以做一种计数。在下一层定义了一个类 Modulo Counter ，这个计数器计数到一个指定的限制为止，然后返回零。

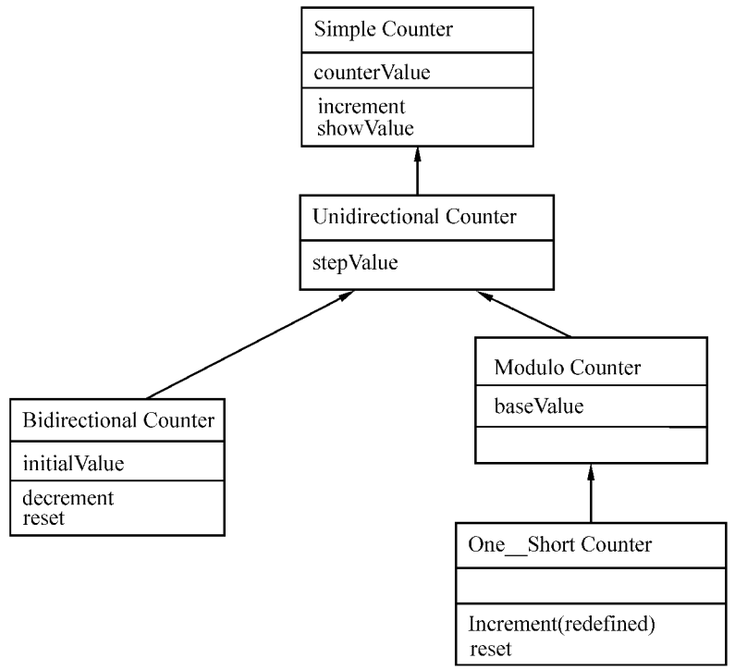


图 4-24　设计计数器的渐增式小类方式

本 章 小 结

本章主要讲了系统设计。进行系统设计，从数据流图出发设想完成系统功能的若干种合理的物理方案，分析员应该仔细分析比较这些方案，并且和用户共同选定一个最佳方案，然后进行软件结构设计，确定软件由哪些模块组成以及这些模块之间的动态调用关系。层次图和结构图是描绘软件结构的常用工具。

抽象是人类认识复杂事物时最有力的思维工具，在进行软件结构设计时惟一有效的方法是，由抽象到具体地分析和构造出软件的层次结构。

在进行详细的过程设计和编写程序之前，首先进行结构设计，其好处在于可以在软件开发的早期站在全局高度对软件结构进行优化。在这个时期进行优化付出的代价不高，却可以使软件质量得到重大改进。

习 题 4

4.1 什么叫面向对象？面向对象方法的特点是什么？

4.2 如何理解模块独立性？用什么指标来衡量模块独立性？

4.3 模块的内聚程度与该模块在分层结构中的位置有关系吗？说明自己的论据。

4.4 利用结构化设计方法建立一个系统（例如一个计算机辅助的图书馆的管理信息系统）的系统结构图，说明哪些部分是变换型结构，哪些部分是事务型结构。

4.5 在类的设计中需要遵循的方针是什么？抽象、信息隐蔽和模块化如何才能做到？

4.6 从供选择的答案中选出正确答案填入下列叙述中的（ ）。

面向对象型的程序设计语言具有数据抽象、信息隐蔽、(A)等特征。作为运算单位的对象应具有下列特性：(B)、(C)、(D)。(E)是面向对象的语言。

【供选择的答案】

A： ① 对象调用 ② 对象变换 ③ 非过程性 ④ 信息继承 ⑤ 并发性

B，C，D：① 对象把数据和处理数据的操作结合为一体
② 在程序运行时对象都处于活动状态
③ 对象在计算机中可向其他对象发送信息
④ 接受消息的对象必须给发送消息者以回答
⑤ 对象的内部状态只根据外部送来的消息才进行操作

E： ① C++，SMALLTALK，objectC ② C，Ada，Modula2
③ PASCAL，C++，APL ④ Ada，objectC，C

第5章 详细设计[1,3]

软件设计阶段定义了问题的结构，是软件的一级蓝图，可以用系统流程图及数据流图或结构化语言表示，或以形式化软件设计语言表示。

软件总体设计确定了软件结构，即确定了模块划分，模块接口，可称作二级蓝图。

软件详细设计，也叫软件过程设计，或称软件算法设计、软件逻辑设计。软件详细设计确定软件模块的实现算法，可称软件设计三级蓝图。可用流程图描述，或用伪码描述，或用形式化软件设计语言描述。

5.1 详细设计的任务及过程

5.1.1 详细设计的任务

详细设计不是在机器上实现，不是具体的进行程序编码设计，而是将总体设计中划分的模块的算法做出描述，这个描述是给下一阶段进行编码提供依据的设计蓝图，简言之，详细设计是根据总体设计提供的文档及相关设计结果，确定每一个模块的算法及详细数据；并选定适当的工具清晰准确地描述出来，其设计步骤为：

① 细化总体设计的结构图（SC）图。

② 为每一个模块选定算法，选择描述工具详细描述。

③ 为每一个模块确定使用数据组织。

④ 确定模块接口细节：包括内部接口（模块间相互接口）、外部接口（用户界面、外部软件接口、硬件接口）、输入输出数据、局部数据。

⑤ 为模块写出测试用例。

⑥ 编写详细设计说明书。

5.1.2 详细设计的过程

在详细设计过程中，需要完成的工作是：

（1）确定软件各个组成部分内的算法以及各部分的内部数据组织。

（2）选定某种工具来描述各种算法。

（3）进行详细设计评审。

5.1.3 详细设计的原则

由于详细设计是给程序员编码提供依据的，因而要求做到：

（1）模块的逻辑描述一要清晰易读，二要正确可靠。

（2）采用结构化设计方法，改善控制结构，降低程序复杂程度，提高程序的可读性、可测试性和可维护性。

根据在高级语言中取消 goto 语句及用顺序、选择、循环 3 种结构可构造任何程序结构，并能实现单入口单出口的程序结构，IBM 公司进一步提出程序结构应该坚持单入口单出口的原则，同时对结构化程序设计的逐步求精、抽象分解作了总体概括，从而形成了下列结构化程序设计的基本方法与原则。

① 程序语言中尽量少用 goto 语句，以确保程序的独立性。

② 用单入口单出口的控制结构，确保程序的静态结构与动态执行情况相一致，保证程序容易理解。

③ 程序的结构一般采用顺序、选择、循环 3 种结构来构成，确保结构简单。

④ 用自顶向下逐步求精的方法做程序设计。

结构化程序设计的缺点是存储容量和运行时间增加 10%～20%，优点是可读性和可维护性好。

（3）选择适当的描述工具来描述模块的算法。

5.1.4 详细设计工具

详细设计工具分为 3 种：图形、表格、语言。这些工具都必须能够明确指出算法的控制流程、处理功能、数据组织等细节，为编码阶段提供依据。这些工具是：

1．程序流程图 PFC（program flow chart）

流程图的优点是：直观、易掌握、历史长、普及广泛。

流程图的缺点是：诱使程序员过早考虑控制流而忽略整体结构；箭头代表控制流，转移太多，容易破坏结构；不容易表示数据结构；不适合理解大型程序，适合具体模块小程序。

流程图的基本结构用判断三角形种类的例子进行说明，如图 5-1 所示。

2．问题分析图 PAD(problem analysis diagram)

问题分析图是日立公司 1973 年提出的，也是由 PFC 演变而来的，主要思想是用二

维树型结构来表示程序的控制流及程序结构。图 5-2 是用 PAD 图描述的 3 种基本控制结构。

例如：计算$(a+b)^n$的展开系数，已知系数算法

P{I，J}=（I–1）！/（(J–1）！(I–1）！）　（I≥1，1≤J≤I）

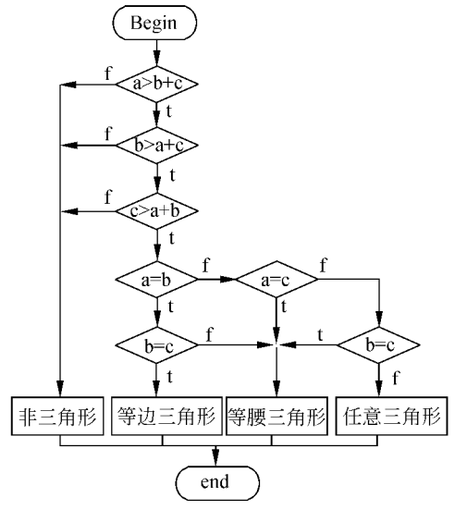


图 5-1 举例说明流程图的基本结构

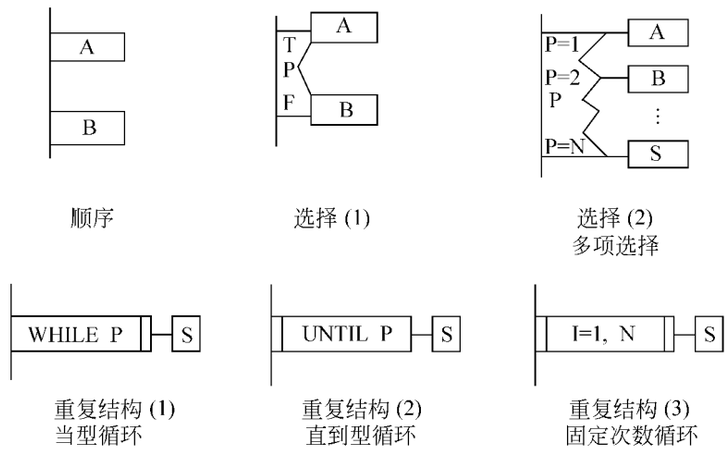


图 5-2 3 种基本控制结构的 PAD 图

则用 PAD 图描述计算系数的模块如图 5-3 所示。

PAD 图作为详细设计的描述工具的优点是：

（1）PAD 是二维树层次结构，自上而下，自左至右遍历所有节点，易读、易记、易懂。

（2）可描述程序，又可描述数据。

（3）支持自顶向下，逐步求精的结构化方法。

（4）竖线标志程序的结构层次数。

（5）PAD 描述的程序算法容易译为高级程序语言。

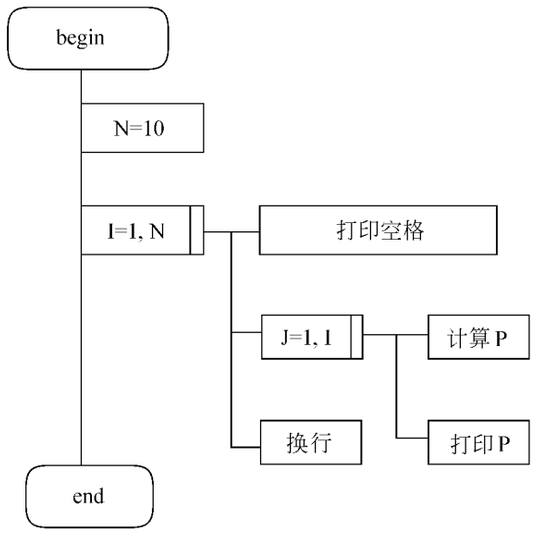


图 5-3　用 PAD 图描述的模块结构

3．**盒式图** N-S(Nassi-Shneiderman)

由于不违背程序结构设计的原则，由 Nassi-Shneiderman **提出了盒式图，**3 **种基本控制结构的** N-S **图如图** 5-4 **所示。**

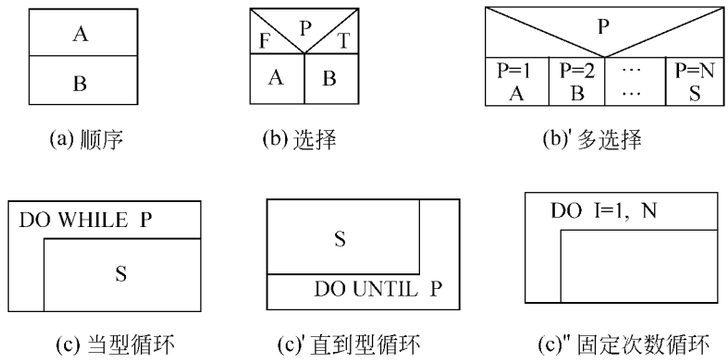


图 5-4　3 种基本控制结构的 N-S 图

对于图 5-3**，可以用盒式图描述成如图** 5-5**。**

N-S **图的优点是：**

（1**）域可以从图上明显看出，功能域的表达明确。**

（2**）容易确定局部数据与全局数据的作用域。**

（3**）很容易表达模块的层次与嵌套关系。**

N-S **图的缺点是：**

（1**）有控制流线，不能任意转移。**

（2**）控制关系隐含，循环关系隐含。**

4．层次输入处理输出图 HIPO（hierarchy plus input process output)

层次输入处理输出图是 IBM 公司发明的，分为 H 图和 IPO 图，其特点是：

（1）表示模块层次和输入输出数据及处理功能。

（2）图主要表示主功能模块与次功能模块的关系。

（3）高层 IPO 图描述主功能模块与次功能模块的输入、处理和输出。

（4）低层 IPO 图描述 H 图中低层次的具体设计。

（5）每层 3～4 个模块为宜。

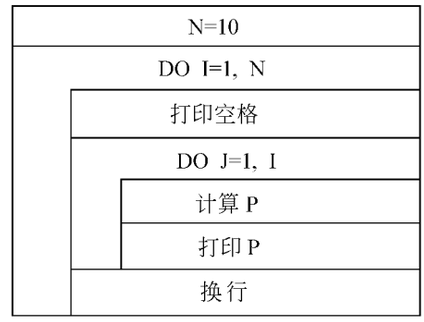


图 5-5 用盒式图描述的模块结构

HIPO 图也存在着不足，主要有：

（1）不能仔细描述算法。

（2）不容易转换成高级语言。

5．判定表(decision table)

当算法中包括多重嵌套组合条件时，PFC、PAD、N-S、HIPO 均不能清楚地描述这些条件，而判定表可以清楚的描述；判定表左上为条件，右下为动作；右上为对应条件的取值，右下是对应动作的取值；每一列构成一条规则，即满足不同条件组合，有不同的动作。

例如：某人事部门对职工分配工作的原则如下：

年龄	文化	性别	分配政策
<18	小学	男（女）	学习
	中学	男（女）	电工
	大学	男（女）	技术员
≥18，<40	小学（中学）	男	钳工
		女	车工
	大学	男（女）	技术员
≥40	小学（中学）	男（女）	材料员
	大学	男（女）	技术员

分析：

政策条件		分配政策
性别 C1	0 男　1 女	学习 A1
年龄 C2	0<18 岁	电工 A2
	1≥18，<40	钳工 A3
	2≥40	车工 A4
文化 C3	0 小学	技术员 A5
	1 中学	材料员 A6
	2 大学	

根据以上分析，得到判定表如表 5-1 所示。

表 5-1　人事分配政策判定表

政策条件	C1	/	/	/	0	0	/	/	/	/	1	1
	C2	0	0	0	1	1	1	2	2	2	1	1
	C3	0	1	2	0	1	2	0	1	2	0	1
分配政策	A1	¡Ì										
	A2		¡Ì									
	A3				¡Ì	¡Ì						
	A4										¡Ì	¡Ì
	A5			¡Ì			¡Ì			¡Ì		
	A6							¡Ì	¡Ì			

6．判定树（decision tree）

判定树是判定表变化得来的，可以省去许多不必要的条件组合，更适合结构化语言的表达。表 5-1 可以表示为判定树，如图 5-6 所示。

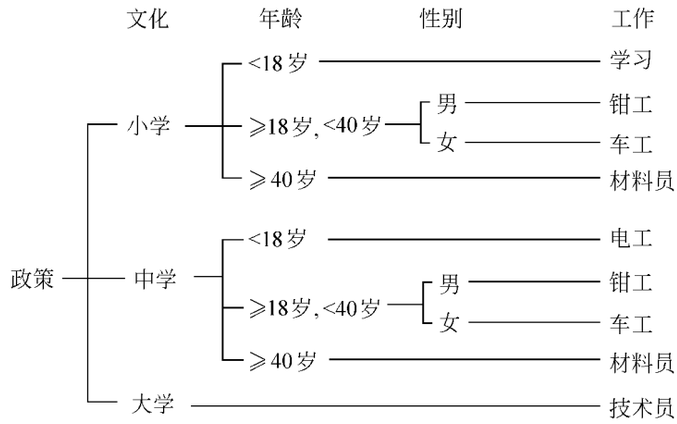


图 5-6　用判定树描述的人事分配政策

7．过程设计语言 PDL(procedure design language)

过程设计语言也称伪码，表示算法灵活自由，便于书写，并可用自动处理程序直接将其译成目标程序代码。下面是 PDL 的基本构成。

1）数据说明

declare <数据名>as<限定词>

数据名为变量或常量名表

限定词为 scalar, array, list, char, structure

2）子程序结构

procedure <子程序名>
interface <参数表>

< 分程序和（或）PDL 语句>
return
end <子程序名>

3）分程序结构

begin <分程序名>
<PDL 语句>
end < 分程序名>

下面是用伪码表示的 3 种基本控制结构。

顺序结构	选择结构	循环结构
A seq	A select cond1	A iter
until(while)cond		
B	B	B
C	A or cond2	A end
D	C	
A end	A or cond3	
	D	
	A end	

与其他工具一样，PDL 也有其固有的不足：那就是不如图形工具直观清晰。

5.2 结构化设计方法

结构化程序设计的概念最早是由 E.W.Dijkstra 提出的。1965 年 Dijkstra 在一次会议上指出：“可以从高级语言中取消 goto 语句”，“程序的质量与程序中所包含的 goto 语句的数量成反比”。1966 年，Böhm 和 Jacopini 证明了，只用 3 种基本数据结构就能实现任何单入口单出口的程序。这一证明为结构程序设计技术奠定了理论基础。结构化的设计方法是基于模块化、自顶向下逐层细化、结构化程序设计等程序设计技术上发展起来的。

5.2.1 基于数据流的结构化设计方法

基于数据流的结构化设计方法是对于数据流问题给出的一个设计程序结构的系统化的途径。其基本思想是：在软件需求分析阶段得到的数据流图的基础上，对数据流图进行细化，然后把细化的数据流图“映射”成结构图。

1．基本概念

（1）事物流：当数据沿输入通路到达一个处理，这个处理根据输入数据的类型在若干个动作中选出一个来执行，这种以“事物为中心”的数据流称为事物流。

（2）变换流：当数据沿输入通路由外部信息经过变换中心变成内部信息进入系统，经系统处理后输出，输出的信息再经过变换中心将内部信息变成外部信息输出，称具有这种特征的数据流为变换流。

2．设计步骤

下面给出基于数据流的结构化设计方法的设计步骤。

① 复查系统的模型。

② 复查并细化数据流图。

③ 确定数据流的特征，辨别事物流还是变换流。

④ 确定输入流和输出流的边界，从而孤立出事物中心或变换中心。

⑤ 做第一级细化，即完成数据流图的细化。

⑥ 做第二级分解，即将细化的数据流图映射成结构图。

其设计过程如图 5-7 所示。

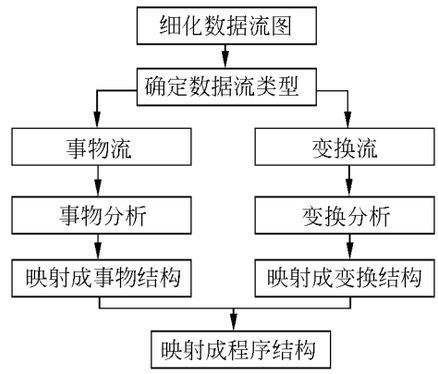


图 5-7　基于数据流的结构化设计方法的设计过程

5.2.2　面向数据结构的结构化设计方法

除了数据流外，另一类问题是数据结构类问题。面向数据结构的结构化程序设计方法，典型的应用是 Jackson 程序设计方法和 Warnier 程序设计方法，这两种方法的应用，将在 5.3 节和 5.4 节详细介绍。

5.3　Jackson 程序设计方法

Jackson 程序设计方法是一种典型的面向数据结构的程序设计方法，也叫 JSD（jackson system development）方法，早期的 Jackson 方法主要用于较小系统的设计，是按输入、输出和内部信息的数据结构进行程序设计，即先用数据结构描述，然后“映射”成程序结构。对于大的系统，其结构较为复杂，当结构发生冲突时，早期的方法变得难以应付，因此，M.J.Jackson 便提出了 JSD 即 Jackson 系统开发方法。这就是现在所说的 Jackson 程序设计方法。

5.3.1　Jackson 方法的基本思想

Jackson 程序设计方法的基本思想是：先找出问题的基本结构，并用 3 种基本数据结构进行描述，然后“映射”成程序结构。

1．Jackson 图

1）顺序结构

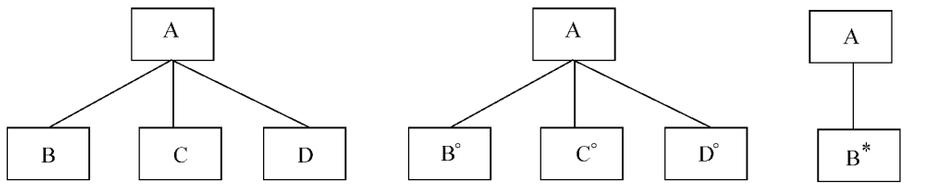
顺序结构是由一个或多个数据元素组成，每个元素按一定顺序出现一次，如图 5-8（a）所示。

图 5-8（a）也可以描述成，动作 A 是由动作 B、动作 C 和动作 D 按一定顺序执行而组成的。

2）选择结构

选择结构包含两个或多个数据元素，每次使用这些元素时，按条件选择其中的一个，如图 5-8（b）所示。

图 5-8（b）也可以描述成，动作 A 按一定条件选择动作 B 或动作 C 或动作 D 来构成。



（a）顺序结构　　（b）选择结构　　（c）重复结构

图 5-8　3 种基本数据结构

3）重复结构

重复结构的数据是根据条件由一个元素出现零次或多次构成，图 5-8（c）是重复结构的 Jackson 图。

同样，重复结构也可以描述成，动作 A 由动作 B 按条件重复执行零次或多次构成。

2．Jackson 程序设计方法的基本步骤

Jackson 程序设计方法基本上是由下述 5 个步骤组成的。

① 分析并确定输入数据和输出数据的 Jackson 结构。并用 Jackson 图描述这些数据。

② 找出输入数据结构和输出数据结构中有对应关系的数据单元。

③ 用相应的规则从描述数据结构的 Jackson 图导出程序结构图。这些规则是：

第一，为每对有对应关系的数据单元，按照其在数据结构中的层次在程序结构图中的相应层次画一个处理框，若这对数据单元在输入数据结构和输出数据结构中所处的层次不同时，则和其对应的处理框在程序结构中所处的层次与之在数据结构图中层次低的那个对应。

第二，根据输入数据结构中剩余的每个数据单元所处的层次，在程序结构图中的相应层次分别为其画上相应的处理框。

第三，根据输出数据结构中剩余的每个数据单元所处的层次，在程序结构图中的相应层次分别为其画上相应的处理框。

④ 列出所有操作与条件 ，并且把这些操作与条件分配到程序结构图的适当位置。

⑤ 用伪码表示程序

需要指出的是，前三步属于系统分析阶段，后两步属于系统设计阶段。

5.3.2　Jackson 方法的设计技术及实例

前面对 Jackson 方法的设计步骤进行了说明，作为基本原则，操作性不强，不利于大家的应用，下面就设计的全过程举例进行说明。

例：某大学有两个校区，为了便于学生上课，计划建立交通车服务系统。有一辆车，两个车站，车站设有呼叫按钮，学生可按下按钮，要求乘车，若车停于此站，等到学生上车，就驶向对方，若学生呼叫按钮（按下按钮）时，车正在行驶，则学生等待；等车到达，搭乘学生（若有）返回；若学生呼叫时，车正停在对方站，得到呼叫信号，车就赶回来。若没有呼叫，则车总是等待。

第一步：实体动作分析。

从所有出现的名词中选出实体，从命题看，可能作为实体的名词有：大学、校园、学生、上课、车站、车、按钮等，其中大学、校园、上课、学生、车站与上述问题无直接关系，已超出要解决问题的模型，从实体中排除，不选作实体。最后只考虑按钮、车两个实体。

动作是在特定的时刻施加在实体上的。可以通过检查命题中出现的动词来选择。经过分析发现，与实体相关的动作只有“到站”、“按下”和“驶离”，因而，只将“到站”、“按下”和“驶离”选作实体按钮和车的动作。

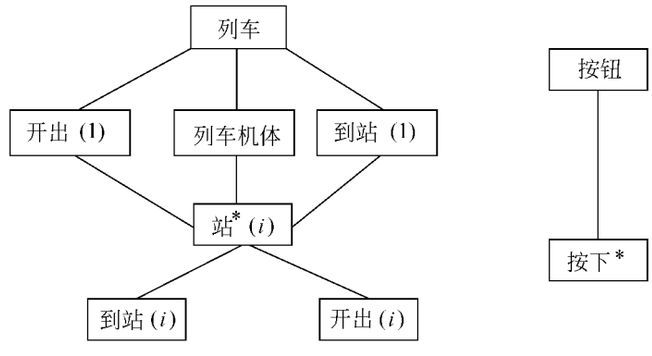
第二步：实体结构分析。

实体的结构通过在一段时间内的动作来描述实体的运动情况。通过分析，车只有到站和驶离两个动作；按钮只有被按下一个动作。

如图 5-9 给出了实体“列车”和“按钮”的实体结构图。

从图 5-9（a）可以看出，列车的活动从站 1 开始和结束。作用于实体的动作只有到站（arrive）和驶离(leave)。开始时，列车停在站 1，其后，列车往返于站 1 和站 2 之间，最后返回站 1。在到达某站后，紧接着的动作必然是从该站驶离。用 arrive（i）和 leave（i）表示这一现象，其中的 i 是站号。本例中，i 的取值范围是 1 或 2。

图 5-9（b）表示作用于实体“按钮”上的动作只有重复“按下”。



(a) 列车的实体结构　　(b) 按钮的实体结构

图 5-9　列车和按钮的结构图

结构图给出了实体执行的动作或施加于实体动作的时序关系，因而可以准确地描述现实世界。

第三步：定义初始模型。

第一、二步选定了实体和动作，并用结构图建立了实体和动作的联系，是对现实世界的一种抽象描述；这一步则是系统的规格说明图（system specification diagram），使其成为现实世界的模型。如图 5-10 是大学交通服务系统的系统规格说明图。其中数据流连接用于描述进程之间的信息流传入/传出的关系，矩形框表示信息流的进程，箭头表示信息流的传递方向，圆圈表示数据流。

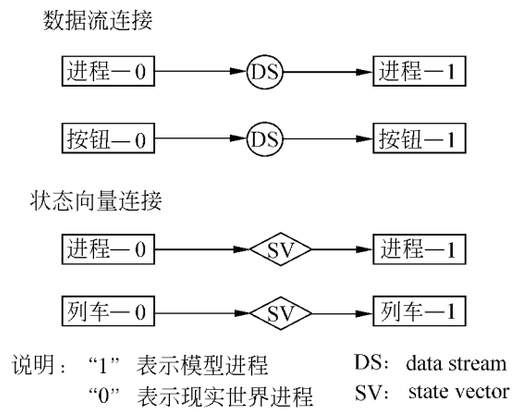


图 5-10　大学交通服务系统的系统规格说明图（SSD）

状态向量连接用于描述一个进程直接检查另一个进程的状态向量。其中，矩形框及箭头与数据流连接一致，菱形框表示相关的状态向量。框中的 SV 即 state vector 的缩写。后缀“0”表示现实世界进程，后缀“1”表示系统模型进程。

只要有可能，模型中的进程与现实世界的实体总是通过数据流联系起来，以确保模型行为和现实世界的行为一致。在上面实例中，呼叫按钮—0 按下时，按钮进程就发送一个脉冲，这个脉冲传送给按钮—1 进程。这就是一个数据流连接。同样，检测列车到站或驶离的传感器不发送脉冲，而是一种状态，功能类似于开关（开/关）并且可以访问，因此需要一个状态向量连接来检测开关的状态。检测的过程就是不断地查询按钮的状态，以确定列车的行为（到站/驶离）。

系统进程不断地检查现实世界的实体。这一点用操作 get sv（获取向量）来实现。该操作可取得现实世界实体的状态向量。需要指出的是，即使在状态向量没有改变时，系统进程也会多次读取状态向量，以确保所有动作不被遗漏。

第四步：功能描述。

Jackson 系统中，功能描述的目的是利用数据流连接和状态向量连接，把已定义的功能进程连接到系统模型进程中。从而扩充系统规格说明图。

在 Jackson 系统开发方法中定义了 3 种功能：

（1）嵌入功能。此功能把操作分配（或写入）到模型进程的结构正文内。

（2）强制功能。此功能检查模型进程的状态向量，并给出输出结果。

（3）交互功能。此功能检查模型进程的状态向量，写入一个施加于模型进程活动的数据流，或引入一个可以写出结果的操作。

功能进程的输出就是系统的输出，可以是报告、（对硬件设备的）命令或其他输出信息。

下面利用大学交通服务系统的实例来说明功能的描述，首先考察列车（shuttle）模型。

在列车上安装一个指示灯（lamp）板，灯亮表示列车到达，指示灯 i 用亮灯命令 lon（i）与熄灯命令 loff（i）控制其接通或切断。当列车到达站 i 时，给灯写出一个接通命令，当列车驶离车站 i 时，给灯写出一个切断命令，因此，当列车在两个车站运行时，将写出一个与指示灯的控制命令完全一致的数据流。

同样，开始时，先发出一个命令（lon（1）），接通指示灯面板上的指示灯，表明列车停在站 1 的显示信息，这是列车的初始状态，列车没有驶出站 1 之前将一直保持这个状态，一旦传感器探测到列车出站，便熄灭相应的指示灯（loff（i））；当列车到站时，便接通相应的指示灯（lon（i））。

还有一种功能是产生动力的命令：启动（start）命令和制动(stop)命令，用以控制列车的运行，当探测器探测到列车到站时，就发出制动（stop）命令；当按钮被按下（有人请求服务），且列车正停在某一车站时，就发出启动（start）命令。

制动命令的发出由列车到站惟一决定，启动（start）命令发出的时间却受按钮和列车二者的制约。因此引入一个动力控制（m control）的功能进程，根据从列车—1 进程和按钮进程接收到的数据进行处理，发出启动（start）命令和制动（stop）命令。

列车—1（shuttle—1）进程和动力控制（m control）进程之间通过数据流（S1D）进行连接。这就表示列车—1 进程不能错过列车到达的信息，必须不断地检测状态向量，以确保所有状态不被遗漏。

图 5-11 是加入了功能的结构图及系统规格说明图。

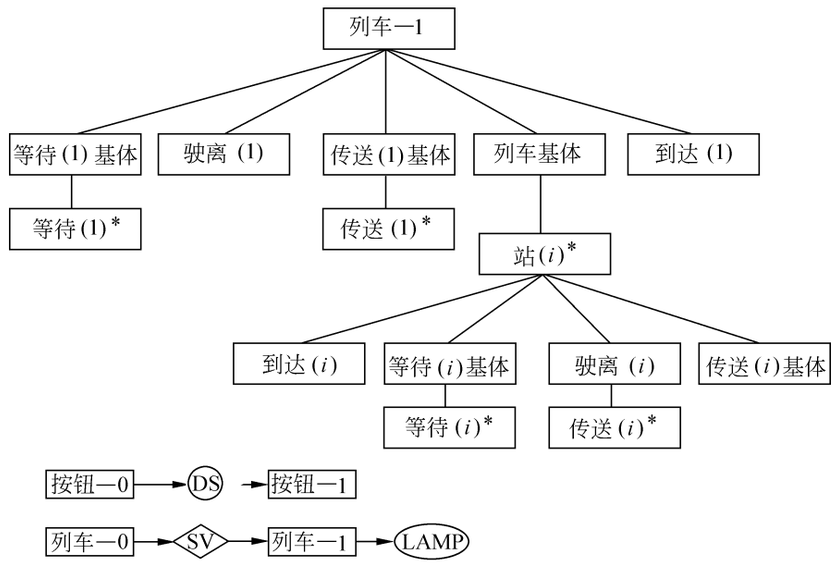


图 5-11 加入了功能的结构图及系统规格说明图

图 5-12 是加入了功能的二级进程系统规格说明图。

对于按钮，原来的“按钮”模型是一个对按钮活动的描述，但现在需要区分是列车开动前请求乘车（按钮被按下），还是列车开动后请求乘车。为了满足这些需求，设计了一个二级进程。其结构图如图 5-13 所示。

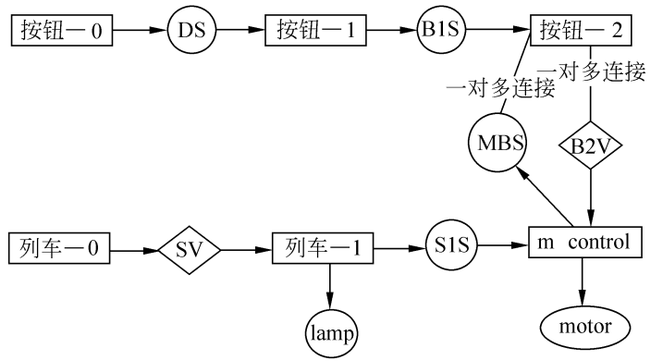


图 5-12　加入了功能的二级进程系统规格说明图

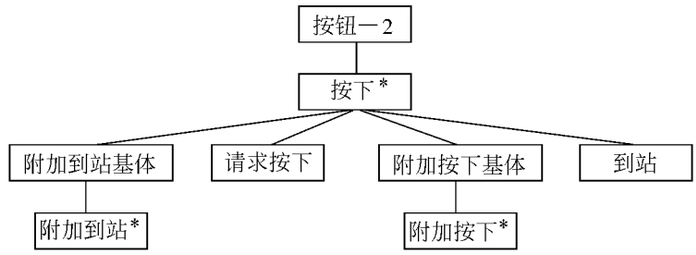


图 5-13　按钮二级进程的结构图

对于图 5-12 和图 5-13，在这里不作详细说明。

第五步：实现。

实现的本质就是：“问题应当被分解成可以用 3 种结构形式表示的部件”。具体方法是：用过程描述语言对模型进程的内部细节进行描述。

例如：按钮—1 的结构正文是：

```
button—1                          按钮 1
      read DS；                   接收一个数据流（DS）
   push—DS iter   while   DS      当 DS 未完时重复按钮循环基体
(push-DS)
      push；                      按下
      read   DS                   接受下一个数据流（DS）
      push—DS   end               按钮循环基体完
button—1   end                    按钮 1 完
```

button—0 的结构与 button—1 的结构几乎完全对应，只不过后者的结构正文中增加

了将现实世界连接到系统的读（read）操作。

进程“列车—1”不能通过数据流连接与其现实世界的对应体建立联系，只能不断地查询开关的状态来判断列车的到站和驶离。

下面是列车—1的结构正文，描述如下：

```
shuttle—1   seq                              列车 1
getsv SV;                                    获取状态向量（SV）
wait—DS   iter    while    wait1             只要仍是 wait1，则一直检测
  getsv SV;                                  获取状态向量（SV）
wait—DS   end                                等待循环基体完
leave(1);                                    列车驶离站 1
transit—DS1   iter   while   transit1        只要仍是 transit1，则一直检测
      getsv SV;                              获取状态向量（SV）
transit—DS1   end                            移动循环基体 1 完
shuttle—DS1   iter                           列车循环基体 1
  station   seq                              车站
    arrive(i);                               列车到站 i
    wait—DS   iter   while   wait i          只要仍是 wait i，则一直检测
      getsv   SV;                            获取状态向量（SV）
    wait—DS   end                            等待循环基体完
      leave(i);                              列车驶离站 i
      transit—DS iter   while transit i      只要仍是 transit i，则一直检测
        getsv   SV;                          获取状态向量（SV）
      transit—DS end                         移动循环基体完
    station   end                            车站基体完
   shuttle—DS1   end                         列车循环基体 1 完
   arrive(1);                                列车到达站 1
 shuttle—1   end                             列车—1 完
```

加入功能1（灯）以后，列车—1的结构正文为：

```
shuttle—1   seq                              列车 1
lon（1）；                                   点亮指示灯 1
getsv SV;                                    获取状态向量（SV）
wait—DS   iter    while    wait1             只要仍是 wait1，则一直检测
  getsv SV;                                  获取状态向量（SV）
wait—DS   end                                等待循环基体完
loff（1）；                                  熄灭指示灯 1
leave(1);                                    列车驶离站 1
transit—DS1   iter   while   transit1        只要仍是 transit1，则一直检测
      getsv SV;                              获取状态向量（SV）
transit—DS1   end                            移动循环基体 1 完
shuttle—DS1   iter                           列车循环基体 1
  station   seq                              车站
    arrive(i);                               列车到站 i
```

```
      lon（i）;                                  点亮指示灯（i）
      wait—DS   iter   while   wait i            只要仍是 wait i，则一直检测
        getsv   SV;                              获取状态向量（SV）
      wait—DS   end                              等待循环基体完
       loff（i）;                                熄灭指示灯（i）
        leave(i);                                列车驶离站 i
        transit—DS iter   while transit i        只要仍是 transit i，则一直检测
            getsv   SV;                          获取状态向量（SV）
        transit—DS end                           移动循环基体完
      station   end                              车站基体完
     shuttle—DS1   end                           列车循环基体 1 完
     arrive(1);                                  列车到达站 1
 shuttle—1   end                                 列车—1 完
```

加入了指示灯和动力控制 m control **的** shuttle—1 **的结构正文作了如下修改：**

```
shuttle—1   seq                                  列车 1
lon（1）;                                        点亮指示灯 1
getsv SV;                                        获取状态向量（SV）
wait—DS   iter    while    wait1                只要仍是 wait1，则一直检测
  getsv SV;                                      获取状态向量（SV）
wait—DS   end                                    等待循环基体完
loff（1）;                                       熄灭指示灯 1
leave(1);                                        列车驶离站 1
transit—DS1   iter   while   transit1            只要仍是 transit1，则一直检测
       getsv SV;                                 获取状态向量（SV）
transit—DS1   end                                移动循环基体 1 完
shuttle—DS1   iter                               列车循环基体 1
  station   seq                                  车站
     arrive(i);                                  列车到站 i
     write arrive to S1D;                        将列车到达信息写入 S1D；
     lon（i）;                                   点亮指示灯（i）
     wait—DS   iter   while   wait i             只要仍是 wait i，则一直检测
       getsv   SV;                               获取状态向量（SV）
     wait—DS   end                               等待循环基体完
      loff（i）;                                 熄灭指示灯（i）
       leave(i);                                 列车驶离站 i
       transit—DS iter   while transit i         只要仍是 transit i，则一直检测
           getsv   SV;                           获取状态向量（SV）
       transit—DS end                            移动循环基体完
     station   end                               车站基体完
    shuttle—DS1   end                            列车循环基体 1 完
    arrive(1);                                   列车到达站 1
     write arrive to S1D;                        将列车到达信息写入 S1D
 shuttle—1   end                                 列车—1 完
```

5.4 Warnier 程序设计方法

Warnier 方法是由法国人 J.D. Warnier 提出的另一种面向数据结构的设计方法，也称逻辑构造程序方法，其原理与 Jackson 方法类似，也是从数据结构出发设计程序，但这种方法的逻辑更严密。

5.4.1 Warnier 方法的基本思想

1．Warnier 图

Warnier 图是表示信息层次结构的一种图形工具，类似于方框图，却比方框图提供的手段更丰富。图 5-14 就是 Warnier 图的一个示例。

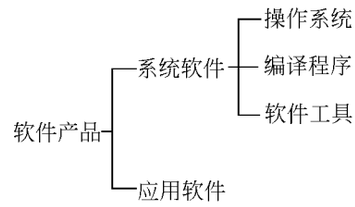


图 5-14 Warnier 图的一个示例

2．Warnier 程序设计方法的基本步骤

Warnier 程序设计方法的最终目的同样是得出对处理过程的详细描述，这种设计方法由以下步骤组成。

① 分析和确定输入数据和输出数据的逻辑结构，并用 Warnier 图描述这些结构。

② 根据输入数据结构导出程序结构，并用 Warnier 图描述处理层次。

③ 画出程序的流程图并给处理框编号。

④ 分类用伪码写出指令。

⑤ 将分类的伪码指令按序号排列，从而得到描述处理全过程的伪码。

5.4.2 Warnier 方法的设计技术及实例

Warnier 作为面向数据结构的程序设计方法，其优点是能够很好地深入到问题的逻辑结构。下面举例说明其设计方法。

例：假设需要设计一个系统来定期产生库存情况的报表。已知的输入文件中，每件产品有一个头记录，后接若干个事务（活动）记录。图 5-15 是输出报表和输入记录。

输出报表

产品号　产品名称　事务号　出库数

入库数

原库存　新库存　出库总数　入库总数

产品号　产品名称

原库存　新库存

（a）输出报表

图 5-15 输出报表和输入记录

输入记录

产品号	产品名称		原库存
产品号	事务号	数量	代码

（b）输入记录

图 5-15（续）

图 5-15 描述了输出报表形式和输入记录格式，在事务记录中的“代码”用于指出产品事务的一次变化，表明是“出库”或“入库”。

第一步，根据输出报表和输入记录的格式，得出输出数据和输入数据的逻辑结构，如图 5-16 所示是用 Warnier 图描述的这些数据结构。

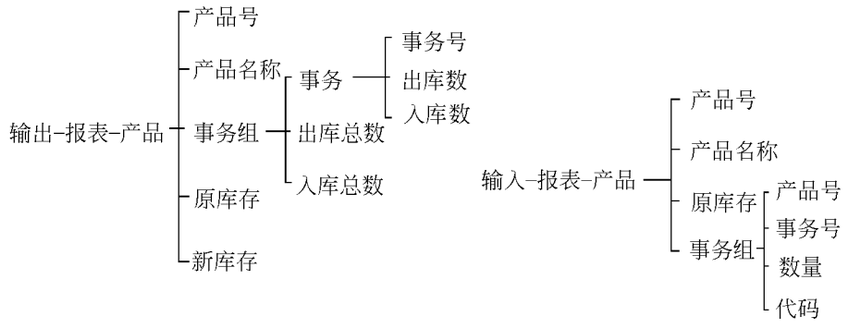


图 5-16　输出数据和输入数据的结构

第二步，利用输入数据的数据结构导出程序的处理层次，如图 5-17 所示。

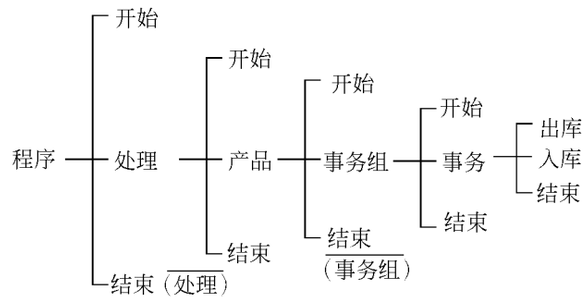


图 5-17　处理层次

第三步，画出与处理层次相对应的程序流程图，如图 5-18 所示。

第四步，如下所示，分类写出伪指令代码。

分类指令表：

010——若没有结束	转至	030
020——		140
040——产品号正确		060

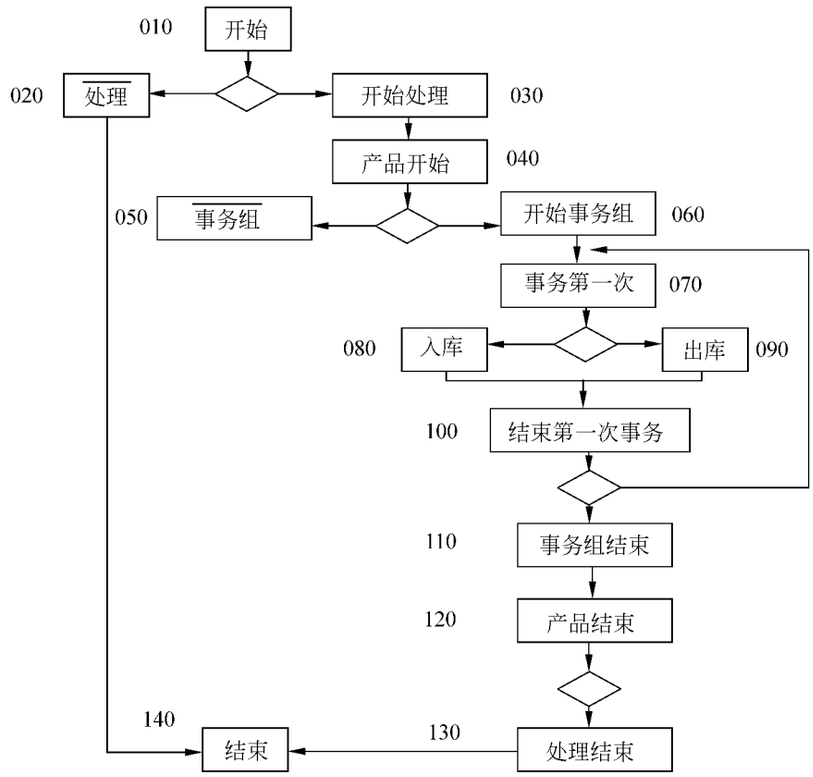


图 5-18 程序流程图

050——**若没有结束**	120
070——**若有代码**	090
080——**若没有结束**	100
090——**若没有结束**	100
100——**产品号正确**	070
120——**若没有结束**	040

计算：

040——**原库存处理**
060——**清出库总数**
——**清入库总数**
080——**入库加总入库数**
090——**出库加总出库数**
110——**从总库存中减去出库数**
——**把入库数加入总库存数**

输出及相关准备：

030——**打印**
040——**产品标题**

——输出
070——编事务号
080——入库数
——编入库数
090——出库数
——编出库数
110——编入库总数
——编出库总数
120——原有总库存
——新总库存
——输出

输入：

010——读
040——读
100——读

第五步，将伪指令代码按序号排列如下：

010——读
020——
030——打印
040——产品标题
——输出
——原库存处理
——产品号正确
——读
050——
060——清出库总数
——清入库总数
070——若有代码
——编事务号
080——入库加总入库数
——编辑
090——出库加总出库数
——编辑
100——打印
——读
110——从总库存中减去出库数
——把入库数加总库存数
——编辑总数
120——编辑
——输出
130——事务结束
140——结束

5.5 基于组件的设计方法

自1968年提出软件工程这一概念以来，软件开发有了一些明显的变化。传统的结构化方法和面向对象技术在解决软件危机的过程中起到了相应时期的历史作用，但是，这些解决方案都没有真正解决软件开发和维护过程中的一系列问题。人们已经认识到，软件开发是困难的，其所面临的各种各样的问题没有单一的解决方案。所以长期以来，人们期待着软件工程师能设计一种组件，从而能像搭积木一样来组装软件，使得软件的设计能像目前的硬件一样，满足不同的需求。

20世纪90年代初期，作为一种基于复用的软件系统开发方法，基于组件的程序设计方法在人们的不断实践中产生了。相比于面向对象技术，组件比对象类更抽象，并且能独立地提供服务。当一个系统需要服务时，可以随时调用能够提供该项服务的组件，而不需要了解组件在哪里执行、逻辑如何描述、用什么语言编写等问题。

5.5.1 基于组件的基本思想

组件技术是应用级别的集成技术，其基本思想是将应用软件分解成一个个独立的单元，将软件的开发过程转变成类似于“搭积木”的搭建过程，通过不同的软件组件单元来构建不同的软件。

面向对象技术使得我们对事物的认识水平和能力有了较大的提高，而组件的思想更多地将重点从建模本身发展到对软件的工业化生产。

基于组件的软件工程（component—base soft engineering，CBSE）强调的是使用可复用的组件来构造软件系统。即从传统的“实现”转向“集成”，这不仅仅是生产方式的改变，更是思维的变革。

1．组件的基本要求

要从真正意义上改变软件的生产模式，对组件要实现以下基本要求。

（1）提供一种手段，使应用软件可以用预先编好的、功能明确的产品部件构建而成，并实现扩展和更新。

（2）利用模块化方法，将复杂的、难以维护的系统分解为相互独立、协同工作的部件，努力使部件可反复使用。

（3）不受时间、空间及硬件的限制，利用统一的接口实现跨平台工作。

2．组件的属性

组件有以下5个基本属性。

（1）组件是可以独立配置的单元。

（2）组件强调与环境和其他组件的分离，因此封装严格，外界没机会或没必要知道组件的内部细节。

（3）组件可以在适当的环境中被复合使用，因此组件需要提供清楚的接口规范，与环境进行交互。

（4）组件不应当是持续的，即不具备个体的基本属性。

（5）组件可以是若干个组件的集合，即用组件可以构成新的、功能更强的组件。

3．组件的支撑技术

组件技术的不断发展，从某种程度上说，依赖于组件的支撑技术的发展。

（1）组件建模。主要用来研究组件本身的本质特征及组件之间的关系。

（2）组件描述语言。以组件模型为基础，解决组件的精确描述、理解、组装问题。

（3）组件分类。描述组件的分类策略、组织模式等，建立组件库，对组件进行管理。

（4）组件复合组装。包括源代码级组装和运行级组装。

（5）标准化。包括模型的标准化和组件库的标准化。

（6）组件架构。研究如何快速、可靠地应用可复用组件系统进行系统构造的方式，着重于软件系统本身的整体结构和组件的互联。

4．组件的使用

（1）理解组件：理解组件包括理解组件的功能与行为、相关的领域知识、约束条件及例外、可预见性的修改及修改方法。

（2）修改组件：对组件库中的组件不需要进行任何改动就直接应用是人们的理想目标，但许多情况下，为了适应不同的需求，必须对组件进行或多或少的修改。

（3）合成组件：如何使用组件技术去实现目标系统，是合成组件的目的。组件合成技术可以分为基于子程序调用和参数传递的功能合成技术、基于数据的合成技术和面向对象的合成技术。

5.5.2 基于组件的设计技术及实例

基于组件的设计技术的研究是从“模式”开始的，起源于上个世纪中期，通常依赖于对象特性，其原理适合于所有软件设计方法。

1．组件的设计内容及要求

（1）以接口为核心，使用开放标准。

（2）形式化组件的语言描述。

（3）组件的封装过程。

（4）设计模式的重用。

（5）开发工具的利用。

2．实现组件技术必须具备的条件

实现组件技术必须具备如下条件。

（1）有标准软件体系结构，保证组件间通讯协议统一，实现同步和异步操作控制，突破空间限制，充分利用网络环境。

（2）有标准接口，保证系统可分解成多个独立单元，用组件组装而成。

（3）组件独立于编程语言。

（4）组件提供版本兼容，以实现应用系统的扩展和更新。

3．基于组件的系统设计

基于组件的软件系统设计类似于面向对象的设计。不同的是，基于组件的系统设计在需求分析进行过程中，除了常规的需求分析外，还应该查找和评价组件，以确定什么

样的组件能满足系统需求，或是什么样的组件组合能满足系统需求。以下给出的是基于组件的系统设计步骤。

① 系统结构分层和逻辑与数据分离；

② 选择和评价已有组件；

③ 使用组件；

④ 实现。

4．实例

这里用组件对象模型 COM（component object model）来说明与组件开发有关的内容，包括 COM 类与 COM 对象、COM 接口、Iunknown 接口定义与实现、IclassFactory 接口与实现、基本的 COM API 函数等。

1）COM 类与 COM 对象

在微软的 32 位操作系统平台上，一个 COM 组件可以是一个 DLL 文件、一个 EXE 文件或一个 Windows NT 服务。一个 COM 组件可以包含一个或多个 COM 类，且每个 COM 类可以实现一个或多个接口。当其他的 COM 组件（或客户程序）调用组件的功能时，首先在 COM 组件中创建 COM 类的实例，也就是 COM 对象，当然也可以通过其他途径获得 COM 对象，然后通过 COM 对象所实现的接口调用组件提供的服务。服务结束后，如果不需要该 COM 对象，则释放该对象所占用的资源包括对象本身。

由于大多数 COM 组件都只包含一个 COM 对象，所以大多数时候对于 COM 类和 COM 对象不作严格区分。图 5-19 是 COM 类与 COM 对象及 COM 接口三者之间的关系。

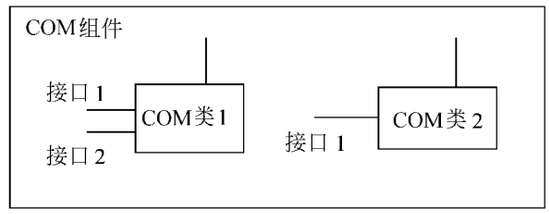


图 5-19 COM 组件、COM 类和 COM 接口之间的关系

COM 标识：客户程序不直接访问 COM 组件，而是通过一个惟一的标识符进行对象的创建与初始化，为此，COM 采用 128 位的全局惟一标识符 GUID（global unique identifier），DUID 是一个随机数，尽管不能保证 GUID 的惟一性，但在概率上发生重复的可能性非常小。

COM 类的标识 CLSID（class identifier）是用来标识 COM 类的 GUID，因此，CLISD 与 GUID 在定义上是一致的。但是，GUID 并不专门用来定义 CLSID，也用来定义其他的标识。下面给出一个 GUID 的例子。

{ 2E76371C –5FB3–4E1C–AC06–E0F9D5BB806E}

用下面的结构来描述 GUID：

```
typedef struct -GUID
{DWORD    Data1 ;
 WORD    Data2;
```

```
 WORD    Data3;
 BYTE    Data4[8]
} GUID;
```

对上面的例子，定义为：

```
extern " C " const    GUID    CLISD- some CoClass =
{0X2E76371C , 0X5FB3 , 0X4E1C ,{0XAC , 0X06 , 0XE0 , 0XF9 , 0XD5 , 0XBB , 0X80 , 0X6E}}；
```

2）COM **接口**

COM **标准包括** COM **规范和** COM **实现两大部分，前者定义了** COM **组件与** COM **组件之间的通信；**COM **实现为** COM **规范提供了一些服务。**COM **接口则是** COM **规范的核心。对于一个能提供算术四则运算的** COM **组件，运用** C++**语言定义的接口如下。**

COM **接口的声明：**

```
Class IMath                接口名称通常以字母 I 开头（下同），I 是 Interface 首字母
{
public:
        virtual     long    Add (long    x ,    long    y)=0;
        virtual     long    Sub (long    x ,    long    y)=0;
        virtual     long    Mul (long    x ,    long    y)=0;
        virtual     long    Div (long    x ,    long    y)=0;
};
```

实现：

```
class Cmath: public Imath
{
public:
    long    Add (long    x ,    long    y)=0;
    long    Sub (long    x ,    long    y)=0;
    long    Mul (long    x ,    long    y)=0;
    long    Div (long    x ,    long    y)=0;

public:
    Cmath( );
    ~Cmath( );

    private :
        long    temp;
};

long    CMath: :Add( long    x,    long    y)
{
      temp    = x + y;
      return    temp;
}

long    CMath: :Sub( long    x,    long    y)
```

```
{
    temp   = x – y;
    return   temp;
}

long   CMath: :Mul( long   x,   long   y)
{
    temp   = x * y;
    return   temp;
}

long   CMath: :Div( long   x,   long   y)
{
    temp   = x / y;
    return   temp;
}

CMath : : CMath( )
{
}

CMath : :~ CMath( )
{
}
```

与 COM 对象一样，为了使客户程序能方便地访问，COM 接口也需要标识，COM 的标识也采用全局惟一的标识符 GUID 进行标识。

3）Iunknown **接口定义与实现**

使用 COM 规范定义的每一个 COM 接口都必须从 Iunknown 接口继承而来。下面仍以四则运算的例子介绍 Iunknown 接口的定义。

```
Class   Iunknown
{
public:
    virture   HRESULT   QueryInterface (REFIID iid , void**ppv)=0
    virture   ULONG   AddRef( ) = 0;
    virture   ULONG   Release( ) = 0;
};
```

实现：

Iunknown **接口的实现引入“引用计数”(**reference counting**)的方法，以有效地控制 COM 对象的生存期。下面介绍引用计数的实现方法。**

COM 采用“引用计数”技术来解决内存管理问题。COM 对象通过引用计数来决定继续生存还是销毁。每一个 COM 对象都记录一个称为“引用计数”的数值，该计数表示有多少个有效指针正在引用该 COM 对象。当客户程序得到一个指向该 COM 对象的

接口指针时，该引用计数值加 1，客户使用完该接口指针并不再使用该 COM 对象时，引用计数值减 1，当引用计数值为 0 时，COM 对象就从内存中销毁。

首先对 Imath 接口重新定义如下。

```
 class    Imath : public    Iunknown
{
public:
virtual    HERSULT    Add(long    x ,    long y * z) = 0
virtual    HERSULT    Sub(long    x ,    long y * z) = 0
virtual    HERSULT    Mul(long    x ,    long y * z) = 0
virtual    HERSULT    Div(long    x ,    long y * z) = 0
};
```

AddRef 方法的实现：

```
CMath : : CMath ()
{
m–Ref = 0
}

CMath : :~CMath ()
{
}

UNLONG CMath : :Addref()
{
    return    ++m–Ref;
}

UNLONG CMath : : Release()
{
        if (– –m–Ref    ! = 0)
           return m–Ref ;
              delete    this;                    销毁 COM 对象
                return    0;
    }
```

根据 COM 规范，一个 COM 对象可以实现多个 COM 接口，客户程序在获得 COM 对象的一个初始接口指针后，可以在运行时刻对该 COM 对象的其他接口指针进行查询，因而存在着接口查询的实现，在此不做说明。

4）IclassFactory 接口与实现

类工厂接口（IclassFactory）也必须依附于对象进行工作，实际上是 COM 对象的生产基地，同时也是一个 COM 对象，支持一个特殊的接口——IclassFactory。IclassFactory 接口的定义如下。

```
Class IclassFactory : public Iunknown
{
```

```
public:
    virtual HRESULT CreateInstance ( Iunknown * pUnknown Outer, REFIID    iid, void **ppv) = 0;
    virtual HRESULT lockServer (bool    block) = 0;
};
```

实现：

```
class CmathClassFactory: public IclassFactory
{
public :
   HRESULT QueryInterface(REFIID iid, void    **ppv);
   ULONG    AddRef();
   ULONG    Release();

public :
    HRESULT createInstance(Iunkown * pUnknownOuter ,REFFIID iid, void    **ppv);
    HRESULT LockServer(Bool ,block) ;

Public :
    CmathClassFactory    () ;
   ~CmathClassFactory () ;

private:
     ULONG m–Ref;
};
```

对于 CmathClassFactory **类的定义与实现，在此不作说明。**

5）基本的 COM API **函数**

只有通过 COM API **函数调用，才能让程序真正运行起来，表** 5-2 **列出了** COM API **的基本函数。**

表 5-2 基本的 COM API 函数

CoInitialize,CoIntializeEx	**初始化** COM **库供进程使用**
CoUninitialize	**释放** COM
DLLGetClassObject	**获取进程内服务器的类工厂对象**
DLLCanUnloadNow	COM **定时调用以确定** DLL **是否卸载**
DLLRegisterserver	**进程内服务器注册**
DLLUnregisterserver	**清除注册表中进程内服务器**
CORegisterClassObject	**进程外服务器注册类工厂对象**
CORevokeClassObject	**清除注册表中进程外服务器类工厂对象**
COGetClassObject	**获取一个指定的** COM **类的类工厂对象**
CoCreateIntance,CoCreateIntanceEx	**创建一个指定的** COM **类的对象**

需要说明的是，本部分的内容在考虑不重复的前提下，基本上是按照交叉顺序进行详细说明的，没有提及的部分请大家查阅参考文献。

5.5.3 应用

组件技术的发展，使得其应用无论从范围和程度都得到了迅速提高。如 Microsoft DCOM 就是以其公共对象模型 COM 为基础提出的分布式应用集成框架。而 Windows DNA 则是一个概念性的框架。

1．关于 COM（component object model）

组件对象模型（COM）是微软公司专有的对象总线技术，是基于对象的程序设计模型。COM 的目的在于使得多个软件组件能够彼此协同工作。COM 定义了二进制互操作标准及结构独立的网络协议，从而支持组件的互操作性、软件进化及分布式计算等。其系统服务体系包括协同工作、位置透明、安全性、网络化及其他服务等。

2．关于 DNA（distributed internet application architecture）

DNA 即分布式网间应用体系结构，是 Windows 平台上基于三层模型的应用程序开发模式。

（1）用户层：用于向用户展示信息并收集信息。常用 VB（Visual BASIC）、Delphi 等工具进行开发。

（2）业务层：用于执行业务和数据规则。

（3）数据层：用于对数据进行存储和管理。一般采用数据库管理系统或文件系统等。

3．基于 COM 组件的 Windows DNA 软件开发模式

如图 5-20 是典型的基于组件的软件开发模式。

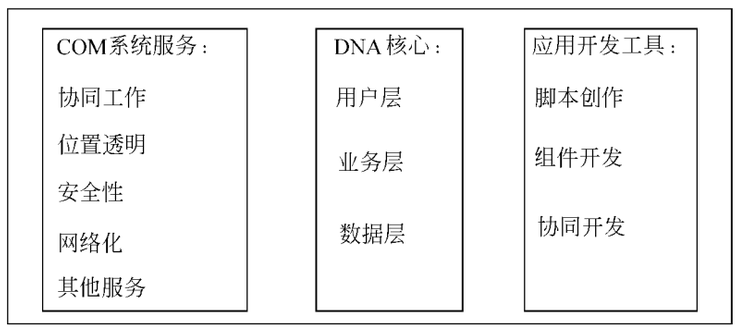


图 5-20　基于 COM 组件的 Windows DNA 软件开发模式

5.6 详细设计说明书

5.6.1 引言

1．编写目的：说明编写这份详细说明书的目的，指出预期的读者。

2．背景：

1）待开发软件的名称；

2）项目的提出者、开发者、用户等。

3．定义：列出本文件中用到的专门术语和词组。

4．参考资料：列出有关的参考资料，如经核准的计划任务书、合同、开发标准等。

5.6.2　总体设计概述

用相应的工具描述软件结构。

5.6.3　程序描述

1．程序 1

1）程序描述：给出对该程序的简要描述，主要说明设计本程序的目的、意义及特点。

2）功能描述：说明该程序应具有的功能，可以采用相应的图形工具如 IPO 图描述。

3）性能描述：说明该程序的全部性能要求。包括精度、灵活性、时间特性等要求。

4）输入输出项：给出每一个输入/输出项的特性，如名称、标识、数据类型、数据格式、数据有效值的范围、输入/输出方式等，还包括数量、频度、输入/输出媒体、输入/输出数据的来源及安全保密条件等。

5）算法：详细说明本程序所采用的算法，并用合适的工具进行描述。

6）流程逻辑：如果需要可以用适当的工具描述程序的流程逻辑。如流程图。

7）接口描述：用适当的工具说明程序所隶属的上一层模块及下一层模块、子程序，说明参数赋值和调用方式，说明与程序直接相关的数据结构（如数据库）。

8）存储分配：根据需要，说明程序的存储分配。

9）注释设计：说明程序中安排的注释。

10）限制条件：说明程序运行中所受到的限制条件。

11）测试计划：说明对程序进行单元测试的计划安排。

12）尚未解决的问题：说明在程序中应该解决而尚未解决的问题，并在以后的工作中解决。

2．程序 2

按上面程序 1 的要求对程序 2 进行同样的描述。

依此类推。

本 章 小 结

详细设计作为软件工程中进行逻辑设计的一个阶段，其设计质量直接决定软件编码实现。本章在介绍软件详细设计过程的同时，还用大量篇幅介绍了详细设计的工具及使用。重点对 Jackson 程序设计方法和 Warnier 程序设计方法进行了用例说明。作为软件工程的新技术，本章对组件进行了简单介绍，以帮助大家加强对 CBSE 的了解。

习 题 5

5.1 详细设计的主要任务是什么？

5.2 程序设计的逻辑结构有哪些？请用不同的工具进行构造。

5.3 指出不同设计工具的使用对象，并比较这些工具的优缺点。

5.4 已知判断三角形种类的程序模块的算法用 PFC 图描述如图 5-21 所示，请改用图 PAD 图、N-S 图进行描述。

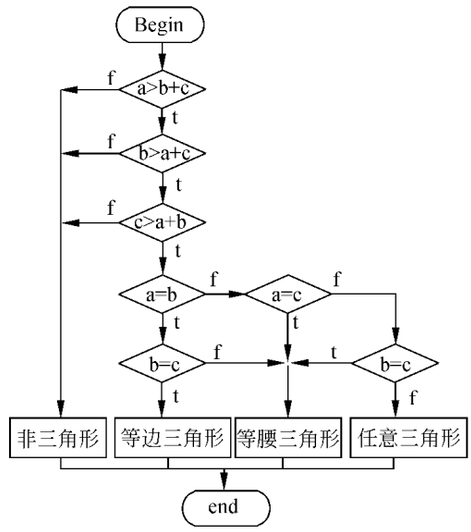


图 5-21 判断三角形种类的 PFC 图

5.5 说说你对组件的理解，找出一个 COM 的开发实例。

5.6 试说明 Jackson 程序设计方法和 Warnier 程序设计方法的设计步骤，并举例简要说明。

5.7 思考题：

（1）从程序员的角度出发，说说详细设计的意义。

（2）试用 PDL 描述一、二、三级蓝图。

（3）结构化程序设计的基本思想。

（4）传统的软件开发方式与基于组件的软件开发方式本质的区别是什么？

（5）基于组件的程序设计与面向对象程序设计有什么不同？

（6）你了解 CBSE 吗？从“实现”走向“集成”意味着什么？

第6章 程序编码 [1, 2, 27, 29]

软件开发的最终目的，是将软件的定义、描述和设计转换成计算机能理解和执行的程序，即进行程序编码。在软件项目的开发中，主要的困难是在需求分析和设计，这些工作做好了，就为编码打下了良好的基础，编码只是将软件设计的结果转换成用某种程序设计语言编写的源程序。程序设计语言和开发工具的性能、编程风格与技巧，在相当大的程度上影响着软件的质量和软件的可维护性。为了保证编码的质量，程序员必须深刻地理解、熟练地掌握并正确地运用程序设计语言的特性；且要求编写出的程序有良好的结构性和良好的程序设计风格。

6.1 程序设计语言

6.1.1 程序设计语言的发展及分类

程序设计语言是人与计算机交换信息的中间媒介和工具，是人工设计的语言。软件工程师用程序设计语言来编写计算机程序，指挥计算机工作。自从数字计算机诞生以来，人们设计并实现了数千种程序设计语言，但只有其中很少一部分得到了广泛的应用。

按发展历史进程的不同，程序设计语言一般分为低级语言和高级语言，大致经历了5代。第一代语言是指低级语言，后面各代均为高级语言。

低级语言面向机器，与具体机器的体系结构紧密联系，不同型号的计算机对应不同的低级语言。低级语言的指令系统随机器而异，难学难用，编写的程序缺乏通用性且冗长，编制、维护困难；但程序运行速度快，效率高，且易于系统接口。低级语言又分为机器语言和汇编语言。机器语言是数字电子计算机问世后最先使用的计算机编程语言，由0、1构成计算机使用的指令代码，直接被计算机接受。汇编语言采用助记符构成指令系统，汇编语言指令与机器语言指令基本上是一对一的关系。但采用汇编语言编写的程序，需经汇编程序翻译成机器语言程序，才能在相应的计算机上执行。

高级语言面向过程或面向对象，主要特征是不依赖于具体的计算机，程序的通用性强，可移植性好；且由于高级语言是自然语言和数学语言的结合，易学习，易使用；但程序的运行速度相对于低级语言程序较慢。一般按实现计算方式又分为解释型和编译型两种高级语言。

第二代语言是指在 20 世纪 50 年代末至 20 世纪 60 年代初先后出现的高级语言。其主要特征是脱离机器、面向算法过程。有变量、赋值、自程序、函数调用概念；有少量的基本数据类型，有限的循环嵌套和一般的函数调用等。第二代语言应用广泛，为人们熟悉和接受，有大量成熟的程序库。这一代的语言主要有 FORTRAN，ALGOL，COBOL 和 BASIC 等。

FORTRAN 语言是世界上第一个被正式推广使用的计算机高级语言，它适合于科学计算，数据处理能力极强。长期的发展过程中，其在向量处理和循环优化方面的技术已相当成熟。ALGOL 语言是一种用于描述计算过程的算法语言，其中 ALGOL60 和 ALGOL68 都支持块结构的概念、动态存储分配、递归及其他属性。COBOL 语言是一种广泛应用于商业数据处理的语言，支持与事务数据处理有关的各种过程技术。BASIC 语言是 20 世纪 60 年代开发的一种初学者的通用符号指令码，其特点是简单易学。

第三代语言也称为结构化编程语言，出现于 20 世纪 60 年代中期。其特点是直接支持结构化构件，具有很强的过程和结构化的能力。这类语言又可分为通用高级语言、面向对象高级语言和专用语言。

通用高级语言的典型代表是 Pascal、C 及 Ada 语言。Pascal 是第一个体现结构化编程思想的现代高级语言。Pascal 语言的模块清晰、控制结构完备、有丰富的数据类型和数据结构，且语言表达力强，移植容易，在科学计算及实时处理以及系统软件开发中有较广泛应用。C 语言最初是用来描述 UNIX 和操作系统软件的语言，它提供完善的数据结构，并有较好的分类，特性是大量地使用指针，具有丰富的运算符和强大的数据处理能力等。C 语言是编写系统软件、编译程序的重要语言之一。Ada 语言是在 Pascal 的基础上开发出来的，适用于嵌入式计算机系统，是第一个充分体现软件工程思想的语言，既是编程语言又可用作设计表达工具。

面向对象高级语言直接支持类的定义、继承、封装和消息传递等概念，使软件工程师能实现面向对象分析和面向对象设计所建立的分析和设计模型。现在使用较为广泛的面向对象高级语言有 Smalltalk、C++、Objective C、Eiffel 及 Java 等。

Smalltalk 语言是 70 年代早期开发的面向对象的程序设计语言，现在使用这种语言开发软件的很少。现在面向对象语言已形成几大类别：一类是纯面向对象的语言，如 Smalltalk 和 Eiffel；另一类是混合型的面向对象语言，如 C++和 Objective C；还有一类是与人工智能语言结合形成的面向对象语言，如 LOOPS、Flavors 和 CLOS；适合网络应用的面向对象语言有 Java 语言。

专用语言语法形式独特，一般翻译过程简便、高效，应用范围较窄，可移植性较小。其中具有代表性的语言有 Lisp、PROLOG、APL 及 FORTH 等。Lisp 是一种函数型编程语言，PROLOG 是一种逻辑型编程语言，这两种语言广泛用于专家系统和专家系统编译程序的开发。APL 是一种具有专门处理数组和向量运算的编程语言。FORTH 是为微处理机软件开发而设计的编程语言。

第四代语言最早出现于70年代末期。其主要特征是用户界面极端友好，是声明式、交互式和非过程式的，有高效的程序代码，软件工程师可以直接使用许多已开发的功能，具备完善的数据库，且具备应用程序生成器。现在使用最广的是数据库查询语言，如FoxPro和Oracle等。随着计算机技术的发展，现在的第四代语言又加入了许多新技术，如事件驱动、分布式数据共享和多媒体技术等。用第四代语言开发的应用可适用于多种数据源，极大地提高了开发效率，降低了开发和维护费用。

第五代语言几乎是与第五代智能计算机同一时期提出的，但就目前而言，其研究工作只能说是刚刚起步，其研究和实现将是一个长期的、艰巨的任务。

6.1.2 程序设计语言的选择

当开始编写程序时，程序员一般都习惯于选择自己熟悉的语言，然而自己熟悉的语言并不一定就是最合适的语言。根据实际需要选择合适的程序设计语言，就会使编码过程中遇到的困难较少，测试工作量较少，且代码维护容易。

一般来说，程序设计语言的选择应该遵循以下几条准则。

（1）软件的应用领域。程序设计语言并不是对所有应用领域都适用的，具有各自的特点和相对最为适合的应用领域。在实时系统中或很特殊的复杂算法、代码优化要求高的应用领域，如过程控制方面或缩写操作系统、编译系统等系统软件，可以优先考虑使用汇编语言、Ada或C等语言；在科学与工程计算领域中，FORTRAN语言是首选，也可以选择使用C、Pascal等语言；在信息管理、数据库操作方面，可以选用COBOL、SQL、FoxPro、Oracle、Access或Delphi等语言；在大量使用逻辑推理和人工智能的专家系统领域，当首选Lisp或PROLOG语言；在网络计算应用中，选择Java语言较为合适；等等。

（2）算法和数据结构的复杂性。科学计算、实时处理和人工智能领域中的问题算法较复杂，而数据处理、数据库应用、系统软件领域内的算法简单，数据结构比较复杂，因此选择语言时可考虑是否有完成复杂算法的能力，或者有构造复杂数据结构的能力的语言。

（3）系统用户的要求。如果软件系统是由用户自己负责维护，通常应该在合适的语言中选用用户较为熟悉的语言。

（4）软件运行环境。软件运行的软件、硬件环境也影响着语言的选择。如：只有在汉化操作系统的支持下，才能选用汉化的程序语言处理汉字信息。

（5）软件开发的方法。有时编程语言的选择依赖于开发的方法，如果要用快速原型模型来开发，要求能快速实现原型，因此宜采用第四代语言；如果是面向对象方法，宜采用面向对象的语言编程。

（6）可得到的软件工具。若有些语言有支持程序开发的软件工具，对于目标系统的实现和验证较为容易。良好的编程环境不但能提高软件生产率，同时能减少错误，提高软件质量。

（7）工程规模。如果软件开发的规模很庞大，已有的语言又不完全适用，那么就可能有必要设计并实现一种能够实现这个系统的程序设计语言。

（8）软件的可移植性要求。如果系统的预期使用的寿命较长，或要在几种不同型号的计算机上运行，就应该选用标准化程度高、程序可移植性好的程序设计语言。

（9）程序员的知识水平。在选择编程语言时，还应该考虑到程序员对语言的熟练程度及实践经验。这就要求程序员必须不断地学习新理论、新方法和新知识，及时更新知识结构和层次，应该客观地而不是主观地选用合适的语言。

实际上，在评价和选用何种语言时，通常要对上述各种因素加以综合考虑，权衡各方面的得失，然后作出合理的决定。

6.2 程序设计风格

软件的质量不但与所选定语言的性能有关，而且与程序员的编程技巧、编程风格及编程的指导思想密切相关。编码风格即为程序设计风格或编程风格，其主要作用是使无论是程序员本人还是其他人，都能比较容易地阅读、理解及修改程序源代码。

写好一个程序，当然需要使程序符合语法规则、修正其中的错误和使其运行得足够快，但是实际应该做的远比这多得多。程序不仅需要给计算机读，也要给程序员读。一个写得好的程序比那些写得差的程序更容易读、更容易修改。

程序设计风格的原则不是随意的规则，是源于由实际经验中得到的常识。代码应该是清楚的和简单的——具有直截了当的逻辑、自然的表达式、通行的语言使用方式、有意义的名字和有帮助作用的注释等，应该避免耍小聪明的花招，不使用非正规的结构。一致性是非常重要的东西，如果大家都坚持同样的风格，程序员就会很容易读懂其他人的代码。风格的细节可以通过一些局部规定，或管理性的公告，或者通过程序来处理。如果没有这类东西，那么最好就是遵循大众广泛采纳的规则。

程序设计风格一般表现在 4 个方面：源程序文档化，数据说明的方法，表达式和语句结构及输入／输出方法。

6.2.1 源程序文档化

编码阶段主要是产生源程序，但为了提高源程序的可维护性，需要对源代码进行文档化。所谓文档化就是在编写源程序中要注意以下几个方面：标识符、注释及源程序的布局等。

1．标识符

所谓标识符就是源程序中用作变量名、常量名、数组名、类型名、函数名、程序名、过程名等用户定义的名字的总称。在满足程序设计语言的语法限制的前提下，含义清晰的标识符有助于对程序的理解。

2．注释

几乎所有的语言都允许用自然语言在程序中进行注释，以便于程序员们通过注释比较容易阅读自己或他人编写的源程序，更重要的是注释对如何理解甚至修改解源程序提供了明确的指导。

注释内容一定要正确，一般分为序言性注释和功能性注释。序言性注释通常在每个

模块的开始，简要描述模块的功能、主要算法、接口特征、重要数据及开发简史，它对于理解程序本身具有引导的作用等。功能性注释插在源程序当中，它着重说明其后的语句或程序段的处理功能。

3．源程序的布局

源程序的布局即源程序的正文编排格式。在源程序中，如说明部分和执行部分之间，完成不同功能的执行模块之间都可以用空行显式地隔开，能显著地改善可读性；水平方向添加适当的空格也可以改善程序的可读性；等等。

6.2.2 数据说明

一个需要用较长时间才能解决的问题，如果寻求到好的数据结构和算法，就可能在分秒之间得到解决。软件系统所涉及的数据结构的组织和复杂的程度是在设计阶段就已经确定了，但如何说明却是在编程时进行的。为了使数据说明便于理解和维护，必须注意下述几点。

（1）数据说明的次序应规范。例如按常量说明、简单变量类型说明、数组说明、公用数据块说明、所有的文件说明的顺序说明；在类型说明中还可进一步要求，例如可按整型量说明、实型量说明、字符量说明、逻辑量说明顺序排列；当用一个语句说明多个变量名时，应当对这些变量按字母的顺序排列。

（2）对于复杂数据结构，应利用注释说明实现这个数据结构的特点。

6.2.3 表达式和语句

在软件的设计阶段确定了软件的逻辑结构，但单个语句的构建则是在编码阶段完成的。程序员应该以尽可能一目了然的形式写好表达式和语句。表达式及语句的构造应力求简单、直接，不要为了片面追求效率而使表达式或语句复杂化。一般应从以下几个方面加以注意。

（1）编写程序时，首先应考虑程序的清晰性，不要刻意追求技巧性；若对效率没有特殊要求，在程序的清晰性和效率之间，同样要首先考虑程序的清晰性。

（2）在使用表达式时，尽量采用其自然形式，如尽量减少使用逻辑运算中的“非”运算；在混合使用互相无关的运算符时，用加括号的方式排除二义性；将复杂的表达式分解成简单的容易理解的形式；避免浮点数的相等的比较等。

（3）程序中经常有一些诸如各种常数、数组的大小、字符位置、变换因子以及程序中出现的其他以文字形式写出的数值，对于这些数值应命名合适的名字，有必要的话加以适当的注释，加强程序的可阅读性、理解性。因为在程序源代码里，一个具有原本形式的数对其本身的重要性或作用没提供任何指示性信息,这也导致程序难以理解和修改。

（4）在编程时，尽量一行只写一条语句，增强程序的可读性；尽量采用简单明了的语句，避免过多的循环嵌套或条件嵌套；同时注意，在条件结构或循环结构的嵌套中，里层结构语句往里缩排，即逻辑上属于同一个层次的互相对齐，逻辑上属于内部层次的推到下一个对齐位置，这样可以使程序的逻辑结构更清晰。

（5）虽然现在仍有很多高级语言允许使用无条件转移语句（goto 语句），但最好避免使用，因为 goto 语句的使用可能使得对程序的执行流程理解起来较为困难。

（6）尽可能使用标准函数；同时，对于重复使用的、具有独立功能的代码段，尽量使用公共过程或子程序的形式；对于递归定义的数据结构使用递归过程。

（7）尽可能按照初始化或数据输入、数据处理、结果输出 3 部分安排层次结构。

6.2.4 输入输出

软件最终是要交付给用户使用的，因此输入和输出的方式和风格应尽可能方便用户的使用。输入和输出的风格与人机交互级别有关。在批处理的输入输出中，要能按数据的逻辑顺序组织输入，以及合理的输出格式等。在交互式的输入输出中，应有简单且带提示的输入方式，由用户指定输出格式，以及输入输出格式的一致性。此外，这两种方式还都应考虑下列原则。

（1）要有完备的出错检查和出错恢复功能。对所有输入数据都进行检验，以确保每个数据的有效性；对多个相关输入项的组合进行检查，拒绝无效的输入值；应允许默认值。

（2）采用简单的步骤和操作，并采用简单的输入格式；允许使用自由格式输入。

（3）使用交互界面时，明确地向用户给出提示信息，并说明允许的数据的选择范围和边界值。

（4）使用数据结束标志或文件结束标志来终止输入。

（5）给所有的输出加以必要的说明，并使所有的报表或报告具有良好的格式。

6.3 程序设计方法

程序设计初期，由于计算机硬件条件的限制，运算速度与存储空间都迫使程序员追求高效率，编写程序成为一种技巧与艺术，而程序的可理解性、可扩充性等因素被放到第二位。随着计算机硬件与通信技术的发展，计算机应用领域越来越广泛，应用规模也越来越大，程序设计不再是一两个程序员可以完成的任务。在这种情况下，编写程序不再片面追求高效率，而是综合考虑程序的可靠性、可扩充性、可重用性和可理解性等因素。正是这种需求刺激了程序设计方法与程序设计语言的发展。

所谓程序设计方法学简单地说，就是讨论程序的性质、程序设计的理论和方法的一门学科。程序设计方法学包含的内容比较丰富，例如，结构程序设计，程序正确性证明，程序变换，程序的形式说明与推导，程序综合，自动程序设计等。

在程序设计方法学中，结构程序设计占有十分重要的地位，可以说，程序设计方法学是在结构程序设计的基础上逐步发展和完善起来的。

6.3.1 结构化程序设计方法

结构化程序设计的思想是在 20 世纪 60 年代末、70 年代初为解决“软件危机”而形

成的。多年来的实践证明，结构化程序设计策略确实使程序执行效率提高，并且由于减少了程序的出错率，而大大减少了维护费用。

所谓结构化程序设计就是按照一定的原则与原理，组织和编写正确且易读的程序的软件技术。结构化程序设计的目标在于使程序具有一个合理结构，以保证和验证程序的正确性，从而开发出正确、合理的程序。其基本思想是：自顶向下、逐步求精；程序结构是按功能划分若干个基本模块，这些模块形成一个树状结构；各模块之间的关系尽可能简单，在功能上相对独立，每一模块内部均是由顺序、选择和循环 3 种基本结构组成；其模块化实现的具体方法是使用子程序。

按照结构化程序设计的要求，设计出的程序设计语言称为结构化程序设计语言。利用结构化程序设计语言，或者说按结构化程序设计的思想和原则编制出的程序称为结构化程序。

1．采用“自顶向下逐步求精、分而治之”的原则进行大型程序的设计

从欲求解的原问题出发，运用科学抽象的方法，把问题分解成若干相对独立的小问题，依次细化，直至各个小问题获得解决为止。所谓“自顶向下”是说，程序设计时，应先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标；所谓“逐步求精”是说，对复杂问题，应设计一些子目标，作过渡，逐步细节化；所谓“分而治之”是说，对已经细化的问题进行分别实现解决的过程。

2．模块化方法

在进行程序设计时把一个大的程序按照功能划分为若干小的程序，每个小的程序完成一个确定的功能，在这些小的程序之间建立必要的联系，互相协作完成整个程序要完成的功能。这些小的程序被称为程序的模块。通常规定模块只有一个入口和出口，使用模块的约束条件是入口参数和出口参数。

将一个大的程序划分为若干不同的相对独立的小程序模块，不仅可以保证设计的逻辑正确性，而且更适合项目的集体开发。各个模块分别由不同的程序员编制，只要明确模块之间的接口关系，模块内部细节的具体实现可以由程序员自己随意设计，而模块之间不受影响。

3．采用 3 种基本结构

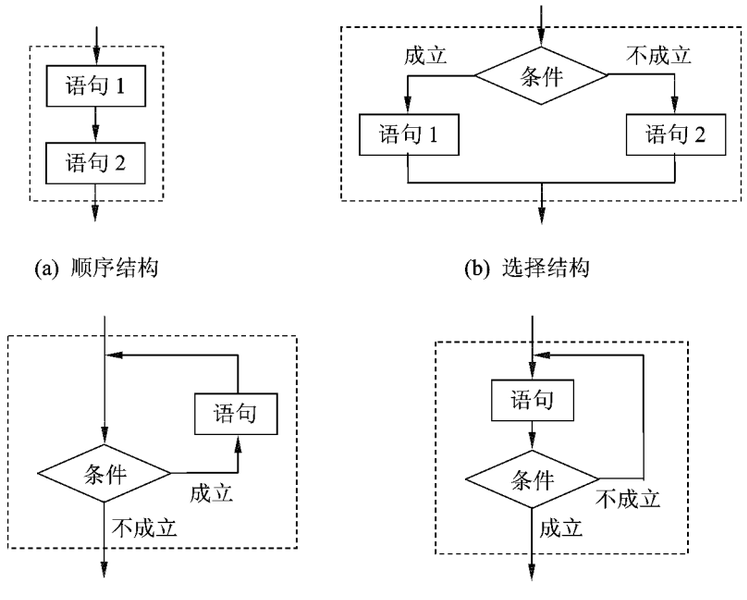
一个程序按结构化程序设计方式构造时，一般地总是一个结构化程序，即由 3 种基本控制结构：顺序结构、选择结构和循环结构构成，典型的 3 种基本结构如图 6-1 所示。这 3 种结构都有以下共同特点：单入口/单出口，结构中每一部分都有机会被执行，结构中不存在无终止的循环。实际应用中，往往还有一些派生出来的具有上述特点的基本结构。

4．有限制地使用 goto 语句

鉴于 goto 语句的存在使程序的静态书写顺序与动态执行顺序十分不一致，导致程序难读难理解，容易存在潜在的错误，难于证明正确性，有人主张程序中禁止使用 goto 语句，但有人则认为 goto 语句是一种有效设施，不应全盘否定而完全禁止使用。结构程序设计并不在于是否使用 goto 语句，因此作为一种折衷，允许在程序中有限制地使用 goto 语句。

在结构化程序设计中，自顶向下是一种分解问题的技术，与控制结构无关；逐步求

精是对结构化程序设计连续分解，形成模块性，最终成为 3 种基本控制结构（顺序、选



(a) 顺序结构　(b) 选择结构

(c) 循环结构（当型循环）　(d) 循环结构（直到型循环）

图 6-1　3 种基本结构

择、循环）的组合。结构化程序设计的结果是使一个结构化程序最终由若干个过程组成，每一过程完成一个确定的工作。这种方法在相当程度上解决了软件的可靠性、可生产性和可维护性等方面的问题，使软件危机得到了缓解。

6.3.2　面向对象的程序设计方法

对于结构化的程序设计语言，其数据结构是解决问题的中心。一个软件系统的结构是围绕一个或几个关键数据结构为核心而组成的。但软件系统的规模越来越大，复杂性不断增长，以致不得不对“关键数据结构”进行重新评价。在这种系统中，许多重要的过程和函数（子程序）的实现严格依赖于关键数据结构，如果这些关键数据结构的一个或几个数据有所改变，则涉及整个软件系统，许多过程和函数必须重写，甚至因为几个关键数据结构改变，导致软件系统的彻底崩溃。在这种情况下，软件开发一直被两大问题所困扰：一是如何超越程序的复杂性障碍，二是计算机系统中如何自然地表示客观世界。

面向对象的思想来自于抽象数据类型。对于面向对象来说，最重要的改进就是作为克服复杂性的手段，把世间万物都描述为对象，即把与事物密切相关的数据与过程定义放在一起，作为一个相互依存、不可分割的整体来处理。而一旦作为一个整体定义了之后，就可以使用对象，而无需了解其内部的实现细节，即从程序员的角度看来，面向对象代码侧重于对象之间的交互，多个对象各司其职，相互协作以完成目标。这样较符合

人类的思维习惯，能够自然地表现现实世界的实体和问题。

1．基本思想

现实世界中的问题，都是由一些基本原始事物组成，且各事物之间存在着一定的联系。面向对象的设计方法，就是按问题领域的基本事物实现自然分割，按人们通常的思维方式建立问题领域的模型，设计出尽可能自然的表示问题求解方法的软件系统，即使设计出的软件尽可能地描述现实世界，构造出模块化的、可重用的、可维护性好的软件，并能控制软件的复杂性和降低开发维护费用。其中，建立模型就是建立问题领域中事物间的相互关系，表示求解问题的方法就是人们思维方式的描述方法。为此，面向对象技术引入“对象”来表现事物、用“消息”建立事物间的联系，再用“类”和“继承”来描述对象，即按通常的思维方式，建立微观的和宏观的问题域模型，尽可能直接自然地表现问题求解的过程。

2．基本概念及特性

1）对象

对象是一个真实或抽象的事物，在面向对象的程序设计方法中，对象是与所描述的事物密切相关的数据（称之为对象的属性），及与事物密切相关的处理过程（将其称为对象的方法或操作或服务）组织在一起形成的整体。任何对象都包含其他对象，这些对象又可包含其他对象，直到系统中最基本的对象被揭示出来。属性实际上是对象的一个特征并在系统的具体环境中有其对应的数值。方法实际上是允许其他对象与之交互的方式。而属性一般只能通过执行对象的方法来改变。

对象总是作为一个整体使用。对外而言，其外部特性（即能接受哪些消息，有哪些方法）是可见的，而对象的内部（即方法的实现和属性）是不可见的。外界不能直接使用对象的方法，也不能直接改变其属性，这种信息隐蔽的形式，减少了程序之间的相互依赖，极大地降低了程序的复杂度，提高了软件的可构造性和易维护性。

可以通过消息传递来使用对象的方法、了解或改变对象的属性。消息就是请求对象执行某种方法或回答某些信息的要求。可见，对象间进行处理及相互之间的联系是通过消息来实现的，消息激活已公布的方法，而方法则代表发送者对接收者实施一个功能。“激活一个方法”指发送者等待接收者处理这个方法并返回，而“发送一个消息”指发送者可以不等待，即允许并行动作的发生。由于对象的执行过程是由消息启动的，当同时传递多个消息时，就能同时启动多个对象，称之为并发性。

2）类

在面向对象系统中，定义若干对象后，可能会有一些对象，这些对象的属性和方法均一致。那么就可以将这些对象的集合定义为一个类。类是一个抽象定义，是具有相同特性的多个对象的一种描述，即类描述了对象的方法和属性结构；而对象则是类的实例，其当前属性值及状态由该对象上执行的方法来定义。用类实现的软件模块独立性较高，为软件的重用提供了基础和保证。

3）继承

继承是使用已存在的定义作为基础建立新定义的技术。面向对象技术中的继承是将多个类的共同特征抽象为一个更普遍的类，这个类即为这些特殊类（或子类）的父类（或超类）。可见类具有层次结构，一个类的上层为其父类、下层为其子类，子类可以从自

己的基类继承所有的数据和操作，并扩充自己的特殊的数据和操作。如果子类只从一个父类继承，则称为单继承；如果子类从一个以上父类继承，则称为多继承。

有了继承性和类的层次结构，不同对象的共同性质只需定义一次，用户就可以充分利用已有的类，符合软件重用的目标。通过类和类的继承，使多个对象共享一个描述，这种方式符合人们的思维习惯，也符合自顶向下和自底向上的设计思想，具有自然性。使得软件能较好的适应复杂大系统不断发展和变化的要求。

4）多态性

多态是指消息的发送者不必知道接收消息的实例所属的类，即一个消息可以与不同的实例结合，而且这些实例可以属于不同类。发送者只提供一个特定事件的请求，接收者知道怎样处理这一事件。多态性使得一个消息如何解释由消息的接收者确定，而不依赖于发送者；消息的发送者只要确定另外的实例可以执行其指定的操作即可。多态使得一系列不同的操作具有相同的名字，减少实现系统所需的代码行数并方便修改。

面向对象方法是一种新方法，具有很多优点，但是还要解决许多问题才能更广泛地使用，其中人们对这种方法的接受程度也直接影响这种方法的普及。

6.4 程序的复杂性及度量

6.4.1 程序的复杂性

程序的复杂性主要是指模块内程序的复杂性，反映了软件的可理解性、模块性、简洁性等属性。软件开发规模相同、复杂性不同的软件，花费的时间和成本会有很大的区别。减少程序复杂性，就可提高软件的可理解性和简洁性，并使软件开发费用减少，开发周期缩短，软件内部潜藏错误减少。

定量度量程序的复杂性，就可以定量估算出软件的开发成本、软件中的故障数量、软件开发需要用的工作量 就可以定量比较一个软件产品的不同设计或不同算法的优劣；就可以作为模块规模的精确限度等。

目前，比较广泛采用的程序复杂性度量方法有：代码行度量法、McCabe 度量法和 Halstead 方法。所谓代码行度量法，就是统计程序的源代码包括代码和注释的行数，并以此行数作为程序复杂性的度量。这种方法计算简单，与所用的高级程序设计语言类型无关。然而，这种方法只是一个简单粗糙的算法，显然没有办法区分相同代码行长的不同复杂程度。McCabe 度量法是一种基于程序控制流的复杂性度量方法，又称环路复杂度。Halstead 方法根据程序中运算符和操作数的总数来度量程序的复杂程度。

6.4.2 McCabe 度量法

T.J.McCabe 于 1976 年提出软件复杂性度量模型。McCabe 认为，程序的复杂性很大程度上取决于程序控制流的复杂性。单一的顺序结构最简单，选择和循环所构成的环路越多，程序也就越复杂。用 McCabe 度量法得到的程序复杂度称为环路复杂度。

McCabe 度量法实质上是对程序拓扑结构复杂性的度量。在这种度量方法中，把程

序看成是有一个入口和一个出口的有向图，这种图被称为程序图。程序中每个语句或一个顺序流程的程序代码段，或流程图中每个处理符号，对应程序图中的一个结点；程序中的流程控制，或程序流程图中原来连接不同处理符号的带箭头的线段，对应程序图中连接不同点的有向弧。程序图仅描绘程序内部的控制流程。假设有向图中的每个节点都可以由入口节点到达，图中环的个数就是环路复杂度。对于强连通图的环路数的计算公式为：

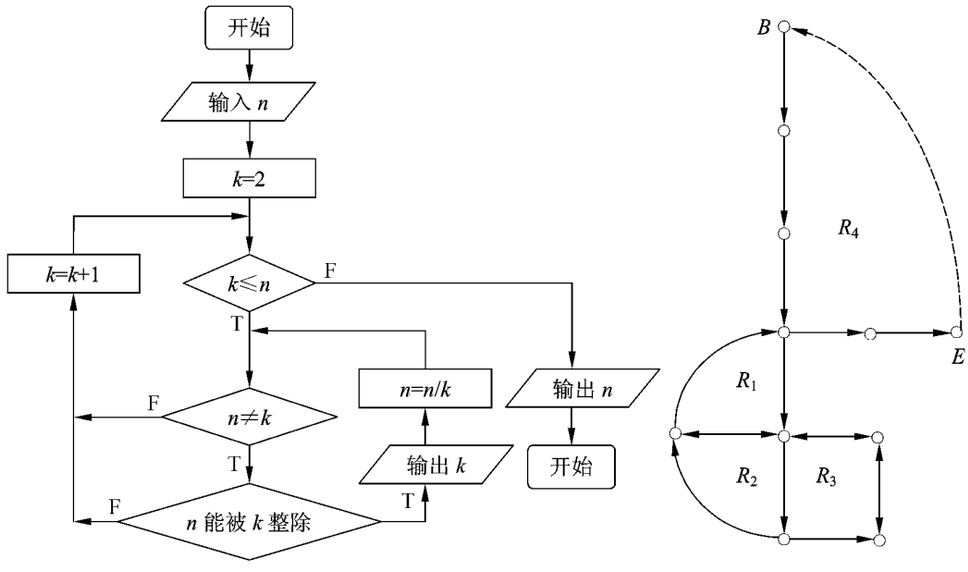
$$V(G)=e-n+p$$

其中，$V(G)$是有向图 G 中的环数，e 是有向图 G 中的弧数，n 是有向图 G 中的结点数，p 是有向图 G 中分离部分的数目（程序图中 p 为 1）。

图 6-2(a)是一个程序流程图，(b)为对应的程序图，如果从出口结点 E 加一条返回到入口结点 B 的边（图中虚线），则此图为强连通图。由上述计算公式可以算出此图的线性无关环路数为：14（边数）–11（结点数）+1=4。

此外，$V(G)$还可以用其他两种方法求得：包括强连通域在内的环路数（如图 6-2(b)中有 4 个线性无关的环路 R_1、R_2、R_3、R_4）；或判定结点数加 1（如图 6-2(b)中为 3+1=4）。

一般，$V(G)$越大，标志程序越复杂，测试工作量越大，潜在的错误个数越多，维护工作越难。据经验，一般一个模块以 $V(G)\leq 10$ 为宜。



(a) 程序流程图　　(b) 程序图

图 6-2　程序图流程图及程序图

6.4.3　Halstead 方法

Halstead 方法也称为文本复杂度量。这种方法用程序中出现的操作符和操作数的总次数来估算程序总长度。操作符是由程序设计语言定义并在程序中出现的语法符号，如

运算符、关键字等；操作数是操作对象，是由程序设计语言定义并引用的，如变量、常量等。假设已估算或计算出几个参数：

n_1：程序中运算符出现的种类；

n_2：程序中运算对象出现的种类；

N_1：程序中运算符的总数；

N_2：程序中运算对象的总数。

根据这几个参数，可以根据下面的公式进行一系列的估算。

（1）程序的预测长度

$$N' = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

（2）程序的实际长度

$$N = N_1 + N_2$$

（3）程序量

$$V = N \log_2 (n_1 + n_2)$$

（4）语言抽象级别

$$L = (2 \times n_2) / (n_1 \times n_2)$$

（5）程序工作量

$$E = V/L$$

（6）程序潜在的错误数

$$B = N' \log_2 (n_1 + n_2) / 3000$$

目前，Halstead 方法是一种较好的软件质量的度量方法，但由于 n_1、n_2、N_1、N_2 相同的程序在控制结构和数据复杂性等方面能存在较大的差别，程序员使用程序设计语言描述算法的水平和熟练程度也有很大的区别，因此 Halstead 的估算方法有一定的局限性。许多人对其算式做了许多改进，出现了许多 Halstead 算式的变形。

本 章 小 结

程序编码就是将软件设计的结果翻译成为用某种程序设计语言描述的源代码。其中涉及方法、工具和过程。程序设计语言是程序员编写程序的工具。程序设计语言已经发展了 4 代，有通用的程序设计语言，也有专用的程序设计语言。在软件发展史上，每次革新性的变化都与新的程序设计方法和程序设计语言紧密相关。为此，要研究和了解程序设计语言的各种特性，对程序编写有哪些影响，根据问题需要应如何选择适当的程序设计语言。需要了解选择程序设计语言的几个原则。

从程序设计风格方面，引入了编程时需要遵循的一些事项，是长期程序设计实践的总结，从不同角度阐明一个好的程序应是什么样子，应该如何编写，不应该如何编写。

结构化程序设计方法的出现，大大改善了程序的质量，使得程序的编写条理化、结构化，可读性、可维护性都有了很大提高。其设计思想是：自顶向下、逐步求精；程序结构按功能划分若干个基本模块，这些模块形成一个树状结构；各模块之间的关系尽可能简单，在功能上相对独立，每一模块内部均是由顺序、选择和循环 3 种基本结构组成；模块化实现的具体方法是使用子程序。

面向对象程序设计将数据及对数据的操作放在一起，作为一个相互依存、不可分割的整体来处理，采用数据抽象和信息隐藏技术。面向对象程序设计将对象及对对象的操作抽象成一种新的数据类型——类，并且考虑不同对象之间的联系和对象类的重用性。

面向对象的程序设计最重要的改进就是把世间万物都描述为对象，而类则描述了同一种对象的特征。从程序员的角度看来，面向对象代码更侧重于对象之间的交互，多个对象各司其职，相互协作以完成目标。面向对象程序设计有希望解决软件工程的两个主要问题——软件复杂性控制和软件生产率的提高。此外它还符合人类的思维习惯，能够自然地表现现实世界的实体和问题，对软件开发过程具有重要意义。面向对象技术的发展，是朝着更加贴近人们世界观的方向发展。

本章最后讨论程序复杂性度量。程序复杂性与程序出错率直接相关，因而与软件可靠性也密切相关。本章主要介绍 3 种程序复杂性度量的方法。代码行度量法；McCabe 环路复杂性度量法，是从程序已有的判断和循环的数目，也就是从结构的角度来计算程序复杂性的；Halstead 软件科学度量法，是从程序的词汇表和词汇量的数目来度量程序的。需要注意的是运算对象(操作数)和运算符(操作符)的选择。

习 题 6

6.1 简述程序设计语言的发展及分类。

6.2 选择一种语言的实用标准是什么？

6.3 什么是编码风格？为什么要强调编码风格？

6.4 程序的编码风格主要体现在哪几个方面？

6.5 简述结构化程序设计的基本思想和面向对象的程序设计思想。

6.6 什么是程序复杂性，为什么要度量程序复杂性？

6.7 用自己所熟悉的语言编写对一组数进行排序的程序模块，然后用 McCabe 方法和 Halstead 方法计算其复杂性。

第7章 软件测试 17~20

软件测试（software testing）是发现软件中错误和缺陷的主要手段。在一般情况下，软件测试过程与整个软件开发过程基本上是平行进行的。当然，测试计划应该在需求分析阶段就已经开始制定了。随后的工作则会伴随着软件开发的过程逐步展开。

通俗地讲，软件测试是发现并指出软件系统缺陷的过程。缺陷在开发和维护的任何阶段都有可能发生，并由此产生一个或多个“漏洞”——错误、误解和冗余，有时甚至会误导开发者。测试包括寻找缺陷，但不包括跟踪漏洞及其修复。换句话说，测试不包括调试和修复。

测试是很重要的，因为测试能充分保证应用软件做想做的事。有些测试的重点延伸到确保一个应用程序仅仅做该做的事。软件的缺陷会造成时间、财产、客户甚至生命的流失，而测试在任何场合都能对防止软件出现错误做出重要贡献。

软件测试是软件工程过程的一个重要阶段，是在软件投入运行前，对软件需求分析、设计和编码各阶段产品的最终检查，是为了保证软件开发产品的正确性、完全性和一致性，从而检测软件错误、修正软件错误的过程。软件开发的目的是开发出实现用户需求的高质量、高性能的软件产品，软件测试以检查软件产品内容和功能特性为核心，是软件质量保证的关键步骤，也是成功实现软件开发目标的重要保障。

7.1 软件测试的基本方法

7.1.1 静态测试和动态测试

依据测试任务要求的类型，可以对软件进行有效性测试（validation test）。有效性测试以实现用户需求为根本点，确认软件的功能、性能和其他特性是否与用户的要求相一致，内容包括：需求规格说明、用户文档、程序文档等的有效性确认。有效性测试有静态分析和动态分析两种基本方式。

静态分析，运用人工分析和程序正确性证明的方法对被测设计文档或程序文档进行特性分析，检查设计和编码的正确性。

动态分析，通过执行程序检查程序的执行状态进行程序测试，动态分析采用测试用例，依据软件设计的功能需求，设定输入条件和推断理论输出，比较测试输出和理论输出检测被测程序的正确性，包括内部程序结构的正确性和程序功能实现的正确性、完备性。

7.1.2　白盒测试和黑盒测试

软件测试可应用多种测试方法来实现测试任务要求，黑盒测试与白盒测试是广泛使用的两种基本的测试方法。

1．黑盒测试

黑盒测试是功能测试、数据驱动测试或基于规格说明的测试。在不考虑程序内部结构和内部特性的情况下，测试者依据该程序功能上的输入输出关系，或是程序的外部特性来设计和选择测试用例，推断程序编码的正确性。黑盒测试是在已知产品应具有的功能的条件下，通过测试来检测每个功能是否都能正常使用。在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。

黑盒测试检测的基本内容有：

（1）功能错误或遗漏；

（2）输入和输出接口的正确性；

（3）数据结构或外部信息访问错误；

（4）性能要求实现情况；

（5）初始化或终止性错误。

黑盒法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。黑盒法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

黑盒测试有两种基本方法，即通过测试和失败测试。在进行通过测试时，实际上是确认软件能做什么，而不会去考虑其能力如何。软件测试员只运用最简单，最直观的测试案例。在设计和执行测试案例时，总是先要进行测试。在进行破坏性试验之前，看一看软件基本功能是否能够实现。这一点很重要，否则在正常使用软件时就会奇怪地发现，为什么会有那么多的软件缺陷出现？在确信了软件正确运行之后，就可以采取各种手段通过搞“垮”软件来找出缺陷。纯粹为了破坏软件而设计和执行的测试案例，被称为失败测试或迫使出错测试。

2．白盒测试

与黑盒测试法相反，白盒测试法的前提是可以把程序看成装在一个透明的白盒子里，也就是完全了解程序的结构和处理过程。这种方法按照程序内部的逻辑测试程序，检验程序中的每条道路是否都能按预定要求正确工作。白盒测试又称为结构测试，是结

构测试、逻辑驱动测试或基于程序的测试。

白盒测试的程序模块检测类别如下：

（1）程序模块独立执行路径检测；

（2）逻辑判定（true or false）各种情况检测；

（3）循环检测（循环边界和循环界内执行情况）；

（4）程序内部数据结构的正确性。

7.1.3 ALAC 测试

ALAC（act like a customer）测试是一种基于客户使用产品的知识开发出来的测试方法，是基于复杂的软件产品有许多错误的原则的测试。最大的受益者是用户，缺陷查找和改正将针对客户最容易遇到的那些错误。

7.2 软件测试过程

软件测试可运用多种不同的测试策略来实现，最常用的方式是自底向上分阶段进行，对不同开发阶段的产品采用不同的测试方法进行检测，从独立程序模块开始，然后进行程序测试、设计测试到确认测试，最终进行系统测试，共分为 4 个阶段过程。

7.2.1 单元测试

单独检测各模块，验证程序模块和详细设计是否一致，消除程序模块内部逻辑上和功能上的错误及缺陷。单元测试集中检验软件设计的最小单元——模块。正式测试之前必须先通过编译程序检查并且改正所有语法错误，然后用详细设计描述作指南，对重要的执行通路进行测试，以便发现模块内部的错误。单元测试可以使用白盒测试法，而且对多个模块的测试可以并行进行。

单元测试检查模块界面的输入输出数据，判断模块是否符合设计要求、模块所涉及的局部数据结构的状况和改变、模块内部重要执行路径（包括出错处理路径）的正确性。

7.2.2 集成测试

将已测试的模块组装进行检测，对照软件设计检测和排除子系统或系统结构上的错误。一般采用黑盒测试法。

集成测试的重点是检测模块接口之间的连接，发现访问公共数据结构可能引起的模块间的干扰，以及全局数据结构的不一致，测试软件系统或子系统输入输出处理、故障处理和容错等方面的能力。

由模块组装成程序时有两种方法。一种方法是先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，这种方法称为非渐增式测试方法；另一种方法是把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把

下一个应该测试的模块结合起来进行测试。这种每次增加一个模块的方法称为渐增式测试，这种方法实际上同时完成了单元测试和集成测试。这两种方法哪种更好一些呢？下面对比这两种方法的主要优缺点：

第一，因为非渐增式测试方法分别测试每个模块，需要编写的测试软件通常比较多，所以需要的工作量比较大；渐增式的测试方法利用已测试过的模块作为部分测试软件，因此开销比较小。

第二，渐增式测试可以较早发现模块间的接口错误；非渐增式测试最后才把模块组装在一起，因此接口错误发现得晚。

第三，非渐增式测试一下子把所有模块组合在一起，如果发现错误则较难诊断定位；反之，使用渐增式测试方法时，如果发生错误则往往和最近加进来的那个模块有关。

第四，渐增式测试方法把已经测试好的模块和新加进来的那个模块一起测试，已测试好的模块可以在新的条件下受到新的检验，因此，这种方法对程序的测试更彻底。

第五，渐增式测试需要较多的机器时间，这是因为测试每个模块时所有已经测试完的模块也要跟着一起运行，当程序规模较大时增加的机器时间是相当明显的。当然，使用渐增式测试方法时需要开发的测试软件较少，因此也能节省一些用于开发测试软件的机器时间，但是总的来说，增加时间是主要的。

第六，使用非渐增式测试方法可以并行测试所有模块，因此能充分利用人力，加快工程进度。

上述前4条是渐增式测试方法的优点，后2条是其缺点。考虑到硬件费用逐年下降人工费用却在上升，软件出错的后果严重，而且发现错误越早纠正错误的代价越低等因素，前四条的重要性增大，因此，总的来说，渐增式测试方法比较好。

当然，在实际测试一个软件系统的时候，并没有必要机械地按照上述某一种方法进行。如果大部分模块可以用简单地测试软件充分测试，则可以先测试好这些模块，再用渐增的（或接近渐增的）方式把测试好的模块逐渐结合到软件系统中去。当把一个已经充分测试过的模块结合进来时，可以只着重测试模块之间的接口；当一个没有充分测试过的模块结合进来时，则需要利用已测试过的模块充分测试这个模块。这种混合方式如果使用得当，可能兼有渐增式或非渐增式两种方法的优点。

7.2.3 确认测试

下面从对所有确认测试适用的8条基本原理开始。

（1）测试可用于显示错误的存在而不是错误的不存在。

（2）测试最困难的问题之一是不知道何时终止。

（3）避免使用未经计划、不能重复使用且用后即扔的测试用例，除非该程序是真正的用后即扔的程序。

（4）测试用例必不可少的一部分是给出预期输出或结果。仔细比较每一测试的实际结果和预期结果。

（5）测试用例必须考虑无效和预期之外、有效和预期内的输入条件。“无效”定义为有效条件之外的条件，并且被测程序的诊断也是如此。

（6）测试用例不能生成理想的输出条件。经验较少的测试人员倾向于只从输入的角度思考。经验丰富的测试人员能确定生成预先设计的输出所要求的输入。

（7）除单元和集成测试以外，程序不应由开发该程序的个人或组织测试。出于成本方面的考虑，往往要求开发人员进行单元和集成测试。

（8）没有发现的错误数与已经发现的错误数成正比例。

IEEE/ANSI 的定义如下：确认是在开发过程之中或结束时评估系统或组成部分的过程，目的是判断该系统是否满足规定的要求。8 条基本原理非常有用，但在实践当中怎样判断一个程序是否确实符合要求呢？有以下两种办法可以解决问题。

（1）开发可以判断产品是否满足在需求规格说明中注明的用户需求的测试。

（2）开发可以判断产品的实际行为是否按照功能设计规格说明中描述的那样与预期行为相匹配的测试。

按规定需求，逐项进行有效性测试。检验软件的功能和性能及其他特性是否与用户的要求相一致，一般采用黑盒测试法。确认测试的基本事项有：

① 功能确认：以用户需求规格说明为依据，检测系统需求规定功能的实现情况。

② 配置确认：检查系统资源和设备的协调情况，确保开发软件的所有文档资料编写齐全，能够支持软件运行后的维护工作。文档资料包括：设计文档、源程序、测试文档和用户文档等。

7.2.4 系统测试

检测软件系统运行时与其他相关要素（硬件、数据库及操作人员等）的协调工作情况是否满足要求，包括强度测试、性能测试、恢复测试和安全测试等内容。

（1）强度测试：强度测试检查程序对异常情况的抵抗能力。强度测试总是迫使系统在异常的资源配置下运行。例如，①当中断的正常频率为每秒一至两个时，运行每秒产生十个中断的测试用例；②定量地增大数据输入率，检查输入子功能的反映能力；③运行需要最大存储空间（或其他资源）的测试用例；④运行可能导致虚存操作系统崩溃或磁盘数据剧烈抖动的测试用例，等等。

（2）性能测试：程序的响应时间、处理速度、精确范围、存储要求以及负荷等性能的满足情况。对于那些实时和嵌入式系统，软件部分即使满足功能要求，也未必能够满足性能要求，虽然从单元测试起，每一测试步骤都包含性能测试，但只有当系统真正集成之后，在真实环境中才能全面、可靠地测试运行性能，系统性能测试是为了完成这一任务。性能测试有时与强度测试相结合，经常需要其他软硬件的配套支持。

（3）恢复测试：系统在软硬件故障后保存数据和控制并进行恢复的能力。恢复测试主要检查系统的容错能力。当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。恢复测试首先要采用各种办法强迫系统失败，然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化（reinitialization）、检查点（checkpointing mechanisms）、数据恢复（data recovery）和重新启动（restart）等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。

（4）安全测试：检查系统对于用户使用权限进行管理、控制和监督以防非法进入、 篡

改、窃取和破坏等行为的能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。例如，①想方设法截取或破译口令；②专门定做软件破坏系统的保护机制；③故意导致系统失败，企图趁恢复之机非法进入；④试图通过浏览非保密数据，推导所需信息，等等。理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值，此时非法侵入者已无利可图。

7.3 软 件 测 试

7.3.1 软件测试角色

每个测试步骤都对应一个测试角色。表 7-1 是软件测试角色的定义，另外还定义测试配置管理的角色。

表 7-1 软件测试角色的定义

软件测试角色	输 入	输 出
测试需求分析	1．软件测试的方法与规范 2．软件需求规格说明 3．软件设计说明（概要设计说明和详细设计说明）	软件测试计划： 1．软件测试的定位 2．软件测试线索 3．软件测试环境的定义 4．软件需求的追踪矩阵
测试过程设计	1．软件测试的方法与规范 2．软件测试计划	软件测试说明： 1．软件测试步骤 2．软件测试基准 3．测试线索的追踪矩阵
测试实现	1．软件测试的方法与规范 2．软件测试说明 3．软件测试工具	软件测试的实现配置： 1．软件测试环境 2．测试步骤的计算机表示（用于回归测试的测试代码/测试数据） 3．测试基准的计算机表示
测试实施	1．软件测试的方法与规范 2．软件测试说明 3．软件测试的实现配置	软件测试记录： 1．测试运行结果的计算机表示 2．测试比较结果的计算机表示 3．测试日志 4．软件问题报告
测试评价	1．软件开发文档 2．软件测试文档 3．软件测试配置 4．软件测试记录	软件测试报告： 1．测试结果的统计信息 2．测试结果的分析/评判
测试配置管理	测试配置管理项： 1．软件测试的描述性表示（测试文档/文件） 2．软件测试的计算机表示（测试代码/数据/结果）	1．软件测试配置管理项的标识管理 2．软件测试配置管理项的存储管理 3．软件测试配置管理项的引用控制 4．软件测试配置管理项的版本控制 5．软件测试配置管理项的更动控制
测试维护	测试配置管理项	1．测试配置管理项的使用报告 2．测试配置管理项的软件问题报告

		3．测试配置管理项的更动控制文件

表 7-1 明确区分各类测试角色，并明确定义其资源（人/物/时间）的安排。明确区分测试需求分析角色和测试过程设计的角色是保障软件测试工作有序开展、有效管理的关键。

7.3.2 软件测试环境

配置测试环境是测试实施的一个重要阶段，测试环境适合与否会严重影响测试结果的真实性和正确性。测试环境包括硬件环境和软件环境，硬件环境指测试必需的服务器、客户端、网络连接设备，以及打印机/扫描仪等辅助硬件设备所构成的环境；软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。在实际测试中，软件环境又可分为主测试环境和辅测试环境。主测试环境是测试软件功能、安全可靠性、性能、易用性等大多数指标的主要环境。一般来说，配置主测试环境可遵循下列原则。

（1）符合软件运行的最低要求。测试环境首先要保证能支撑软件正常运行。

（2）选用比较普及的操作系统和软件平台。例如，一个软件若声称支持“Windows9X/ME/NT Workstation/2000 professional”和“MS Office 97/2000/XP”，一般会采用如“Windows 2000professional+MS Office 2000”的流行环境。

（3）营造相对简单、独立的测试环境。除了操作系统，测试机上只安装软件运行和测试必需的软件，以免不相关的软件影响测试实施。

（4）无毒的环境。利用有效的正版杀毒软件检测软件环境，保证测试环境中没有病毒。

辅测试环境常常用来满足不同的测试需求或特殊测试项目：①兼容性测试：在满足软件运行要求的范围内，可选择一些典型的操作系统和常用应用软件对其安装卸载和主要功能进行验证。②模拟真实环境测试：有些软件，特别是面向大众的商品化软件，在测试时常常需要考察在真实环境中的表现。如测试杀毒软件的扫描速度时，硬盘上布置的不同类型文件的比例要尽量接近真实环境，这样测试出来的数据才有实际意义。③横向对比测试：利用辅测试环境“克隆”出完全一致的测试环境，从而保证各个被测软件平等对比。

7.3.3 软件测试的需求规格说明

下面是软件需求规格说明（IEEE 830 标准）。

a. 引言
- a.1 目的
- a.2 文档约定
- a.3 预期的读者和阅读建议
- a.4 产品的范围
- a.5 参考文献

b. 综合描述
- b.1 产品的前景
- b.2 产品的功能
- b.3 用户类和特征
- b.4 运行环境
- b.5 设计和实现上的限制
- b.6 假设和依赖

c. 外部接口需求
- c.1 用户界面
- c.2 硬件接口
- c.3 软件接口
- c.4 通信接口

d. 系统特性
- d.1 说明和优先级
- d.2 激励/响应序列
- d.3 功能需求

e. 其他非功能需求
- e.1 性能需求
- e.2 安全设施需求
- e.3 安全性需求
- e.4 软件质量属性
- e.5 业务规则
- e.6 用户文档

f. 其他需求

附录 A：词汇表

附录 B：分析模型

附录 C：待确定问题的列表

可以通过参考其他已编写好的项目文档（如项目视图和范围文档或接口规格说明）来将每一部分内容具体化，而不是复制信息或者把所有的内容组成一个单一的文档。不要生搬硬套这个模板，应该把这个摸板转换为自己所需要的文档。

a. 引言

引言提出了对软件需求规格说明的纵览，这有助于读者理解文档如何编写并且如何阅读和解释。

a.1 目的

对产品进行定义，在该文档中详尽说明了这个产品的软件需求，包括修正或发行版本号。如果这个软件需求规格说明只与整个系统的一部分有关系，那么只定义文档中说明的部分或子系统。

a.2 文档约定

描述编写文档时所采用的标准或排版约定，包括正文风格、提示区或重要符号。例如，说明了高层需求的优先级是否可以被其所有细化的需求继承，或者每个需求陈述是否都有其自身的优先级。

a.3 预期的读者和阅读建议

列举了软件需求规格说明所针对的不同读者，如开发人员、项目经理、营销人

员、用户、测试人员或文档的编写人员。描述了文档中剩余部分的内容及其组织结构。提出了最适合于每一类型读者阅读文档的建议。

a.4 产品的范围

提供了对指定的软件及其目的的简短描述，包括利益和目标。把软件与企业目标或业务策略相联系。可以参考项目视图和范围文档而不是将其内容复制到这里。

a.5 参考文献

列举了编写软件需求规格说明时所参考的资料或其他资源。这可能包括用户界面风格指导、合同、标准、系统需求规格说明、使用实例文档，或相关产品的软件需求规格说明。在这里应该给出详细的信息，包括标题名称、作者、版本号、日期、出版单位或资料来源，以方便读者查阅这些文献。

b. 综合描述

这一部分概述了正在定义的产品以及其所运行的环境、使用产品的用户和已知的限制、假设和依赖。

b.1 产品的前景

描述了软件需求规格说明中所定义的产品的背景和起源。说明了该产品是否是产品系列中的下一成员，是否是成熟产品所改进的下一产品、是否是现有应用程序的替代品，或者是否是一个新型的、自含型产品。如果软件需求规格说明定义了大系统的一个组成部分，那么就要说明这部分软件是怎样与整个系统相关的，并且要定义出两者之间的接口。

b.2 产品的功能

概述了产品所具有的主要功能。其详细内容将在 d 中描述，所以在此只需要概略地总结，例如用列表的方法给出。很好地组织产品的功能，使每个读者都易于理解。用图形表示主要的需求分组以及需求之间的联系，例如数据流程图的顶层图或类图，都是有用的。

b.3 用户类和特征

确定可能使用该产品的不同用户类并描述这些用户类相关的特征。有一些需求可能只与特定的用户类相关。将该产品的重要用户类与那些不太重要的用户类区分开。

b.4 运行环境

描述了软件的运行环境，包括硬件平台、操作系统和版本，还有其他的软件组件或与其共存的应用程序。

b.5 设计和实现上的限制

确定影响开发人员自由选择的问题，并说明这些问题为什么成为一种限制。可能的限制包括如下内容：必须使用或者避免的特定技术、工具、编程语言和数据库；所需求的开发规范和标准，例如，如果由客户的公司负责软件维护，就必须定义转包者所使用的设计符号表示和编码标准；企业策略、政府法规或工业标准；硬件限制，例如定时需求或存储器限制；数据转换格式标准。

b.6 假设和依赖

列举出在软件需求规格说明中影响需求陈述的假设因素（与已知因素相对立）。

这可能包括打算要用的商业组件或有关开发或运行环境的问题。可能认为产品将符合一个特殊的用户界面设计约定，但是另一个读者却可能不这样认为。如果这些假设不正确、不一致或被更改，就会使项目受到影响。

此外，确定项目对外部因素存在的依赖。例如，如果打算把其他项目开发的组件集成到系统中，那么就要依赖那个项目按时提供正确的操作组件，如果这些依赖已经记录到其他文档（如项目计划）中了，那么在此就可以参考其他文档。

c. 外部接口需求

外部接口需求可以保证新产品与外部组件正确连接的需求，关联图表示了高层抽象的外部接口。如果产品的不同部分有不同的外部接口，那么应把这些外部接口的详细需求并入到这一部分的实例中。

c.1 用户界面

陈述所需要的用户界面的软件组件。描述每个用户界面的逻辑特征。下面是可能要包括的一些特征。

① 将要采用的图形用户界面（GUI）标准或产品系列的风格。

② 屏幕布局或解决方案的限制。

③ 将出现在每个屏幕的标准按钮、功能或导航链接（例如一个帮助按钮）。

④ 快捷键。

⑤ 错误信息显示标准。

对于用户界面的细节，例如特定对话的布局，应该写入一个独立的用户界面规格说明中，而不能写入软件需求规格说明中。

c.2 硬件接口

描述系统中软件和硬件每一接口的特征。这种描述可能包括支持的硬件类型、软硬件之间交流的数据和控制信息的性质以及使用的通信协议。

c.3 软件接口

描述该产品与其他外部组件（由名字和版本识别）的连接，包括数据库、操作系统、工具、库和集成的商业组件，明确并描述在软件组件之间交换数据或消息的目的，描述所需要的服务以及内部组件通用的性质，确定将在组件之间共享的数据，如果必须用一种特殊的方法来实现数据共享机制，例如在多任务操作系统中的一个全局数据区，那么就必须把该方法定义为一种实现上的限制。

c.4 通信接口

描述与产品所使用的通信功能相关的，包括电子、Web 浏览器、网络通信标准或协议及电子表格等。定义了相关的消息格式。规定通信安全或加密问题、数据传输速率和同步通信机制。

d. 系统特性

文档编写者可能更喜欢通过使用实例、运行模式、用户类、对象类或功能等级来组织这部分内容（IEEE1998），还可以使用这些元素的组合。总而言之，必须选择一种使读者易于理解预期产品的组织方案。仅用简短的语句说明特性的名称，例如“4.1 拼写检查和拼写字典管理”。无论想说明何种特性，阐述每种特性时都将重述从 d.1～d.3 这 3

步系统特性。

d.1 说明和优先级

提出了对该系统特性的简短说明并指出该特性的优先级是高、中，还是低，或者还可以包括对特定优先级部分的评价，例如利益、损失、费用和风险，其相对优先等级可以从 1（低）到 9（高）。

d.2 激励/响应序列

列出输入激励（用户动作、来自外部设备的信号或其他触发器）和定义这一特性行为的系统响应序列。就像在第 8 章讲述的那样，这些序列将与使用实例相关的对话元素相对应。

d.3 功能需求

列出与该特性相关的详细功能。这些是必须提交给用户的软件功能，使用户可以使用所提供的特性招待服务或者使用所指定的使用实例招待任务。描述产品如何响应可预知的出错条件或者非法输入或动作。就像本章开头所描述的那样，必须惟一的标识每个需求。

e. 其他非功能需求

这部分列举出了所有非功能需求，而不是外部接口需求和限制。

e.1 性能需求

阐述了不同的应用领域对产品性能的需求，并解释其原理以帮助开发人员作出合理的设计选择。确定相互合作的用户数或者所支持的操作、响应时间以及与实时系统的时间关系。还可以在这里定义容量需求，例如存储器和磁盘阵列的需求或者存储在数据库中表的最大行数。尽可能详细地确定性能需求。可能需要针对每个功能需求或特性分别陈述其性能需求，而不是都集中在一起陈述。例如，“ 在运行微软 Windows 2000 的 450MHz Pentium II 的计算机上，当系统至少有 50%的空闲资源时，95%的目录数据库查询必须在两秒内完成 ”。

e.2 安全设施需求

详尽陈述与产品使用过程中可能发生的损失、破坏或危害相关的需求。定义必须采取的安全保护或动作，还有那些预防的潜在的危险动作。明确产品必须遵从的安全标准、策略或规则。一个安全设施需求的范例如下：“ 如果油箱的压力超过了规定的最大压力的 95%，那么必须在 1 秒内终止操作 ”。

e.3 安全性需求

详尽陈述与系统安全性、完整性或与私人问题相关的需求，这些问题将会影响到产品的使用和产品所创建或使用的数据的保护。定义用户身份确认或授权需求，明确产品必须满足的安全性或保密性策略。一个软件系统的安全需求的范例如下：“ 每个用户在第一次登录后，必须更改自己的最初登录密码，最初的登录密码不能重用。”

e.4 软件质量属性

详尽陈述与客户或开发人员至关重要的产品质量特性。这些特性必须是确定、定量的并在可能时是可验证的。至少应指明不同属性的相对侧重点，例如易用程度优于易学程度，或者可移植性优于有效性。

e.5 业务规则

列举出有关产品的所有操作规则，例如什么人在特定环境下可以进行何种操作。这些本身不是功能需求，但可以暗示某些功能需求执行这些规则。一个业务规则的范例如下：“只有持有管理员密码的用户才能执行$100.00 或更大额的退款操作。”

e.6 用户文档

列举出将与软件一同发行的用户文档部分，例如，用户手册、在线帮助和教程。明确所有已知的用户文档的交付格式和标准。

f. 其他需求

定义在软件需求规格说明的其他部分未出现的需求，例如国际化需求或法律上的需求。还可以增加有关操作、管理和维护部分来完善产品安装、配置、启动和关闭、修复和容错，以及登录和监控操作等方面的需求，如果不需要增加其他需求，就省略这一部分。

附录 A：词汇表

定义所有必要的术语，以便读者可以正确地解释软件需求说明，包括词头和缩写。可能希望为整个公司创建一张跨越多项项目的词汇表，并且只包括特定于单一项目的软件需求规格说明中的术语。

附录 B：分析模型

这个可选部分包括或涉及到相关的分析模型的位置，例如数据流程图、类图、状态转换图或实体-关系图。

附录 C：待确定问题的列表

编辑一张在软件需求规格说明中待确定问题的列表，其中每一表项都是编上号的，以便于跟踪调查。

需求规格说明模板

文档编号：_____

文档名称：_____

项目名称：_____

项目负责人：_____

编写：_____ 年 月 日

校对：_____ 年 月 日

审核：_____ 年 月 日

批准：_____ 年 月 日

开发单位：

需求规格说明

1．任务概述

2．数据描述

1．数据库描述

2．数据流图

3．数据流条目

4．加工说明

3．功能需求

1．功能划分

2．功能描述

4．运行需求

1．用户接口

2．硬件接口

3．软件接口

5．属性需求

7.3.4 软件测试设计说明

下面是软件测试设计模板。

1．引言

1.1 编写目的：说明编写这份详细设计说明书的目的，指出预期的读者。

1.2 背景：说明待开发软件系统的名称；本项目的任务提出者、开发者、用户和运行该程序系统的计算中心。

1.3 定义：列出本文件中用到专门术语的定义和外文首字母组词的原词组。

1.4 参考资料：列出有关的参考资料，例如：①本项目的经核准的计划任务书或合同、上级机关的批文；②属于本项目的其他已发表的文件；③本文件中各处引用到的文件资料，包括所要用到的软件开发标准。列出这些文件的标题、文件编号、发表日期和出版单位，说明能够取得这些文件的来源。

2．程序系统的结构

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

3．程序 1（标识符）设计说明

对于一个具体的模块，尤其是层次比较低的模块或子程序，其很多条目的内容往往与其所隶属的上一层模块的对应条目的内容相同，在这种情况下，只要简单地说明这一点即可。

3.1 程序描述

给出对该程序的简要描述，主要说明安排设计本程序的目的意义，还要说明本程序的特点（如是常驻内存还是非常驻？是否子程序？是可重入的还是不可重入的？有无覆盖要求？是顺序处理还是并发处理等）。

3.2 功能

说明该程序应具有的功能，可采用 IPO 图（即输入-处理-输出图）的形式。

3.3 性能

说明对该程序的全部性能要求，包括对精度、灵活性和时间特性的要求。

3.4 输入项

给出对每一个输入项的特性，包括名称、标识、数据的类型和格式、数据值的有效范围、输入的方式。 数量和频度、输入媒体、输入数据的来源和安全保密条件等。

3.5 输出项

给出对每一个输出项的特性，包括名称、标识、数据的类型和格式，数据值的有效范围，输出的形式、数量和频度，输出媒体、对输出图形及符号的说明、安全保密条件等。

3.6 算法

详细说明本程序所选用的算法，具体的计算公式和计算步骤。

3.7 流程逻辑

用图表（例如流程图、判定表等）辅以必要的说明来表示本程序的逻辑流程。

3.8 接口

用图的形式说明本程序所隶属的上一层模块及隶属于本程序的下一层模块、子程序，说明参数赋值和调用方式，说明与本程序相直接关联的数据结构（数据库、数据文卷）。

3.9 存储分配

根据需要，说明本程序的存储分配。

3.10 注释设计

说明准备在本程序中安排的注释，如：加在模块首部的注释；加在各分枝点处的注释；对各变量的功能、范围、默认条件等所加的注释；对使用的逻辑所加的注释等。

3.11 限制条件

说明本程序运行中所受到的限制条件。

3.12 测试计划

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

3.13 尚未解决的问题

说明在本程序的设计中尚未解决而设计者认为在软件完成之前应解决的问题。

4．程序 2（标识符）设计说明

用类似 3 的方式，说明第 2 个程序乃至第 n 个程序的设计考虑。

7.3.5 测试评价

1．验收测试

工程项目验收测试，是工程项目在正式运行前的质量保证测试，是软件工程一个独立且必要的质量保证环节。通过系统、专业的验收测试，验证软件系统是否符合设计需求，功能实现的正确性及运行安全可靠性。通过系统、专业的验收测试，可发现软件存在的、潜在的重大问题，最大限度保证软件工程质量，是工程验收的最后阶段，也是信息化工程监理中一个质量保证阶段。通过系统、专业的验收测试，修改软件问题，保证

工程项目正常顺利实施。

中国软件评测中心对验收项目实行项目组负责制，评测中总工参与方案评审、问题单评审与报告评审。评测工作包括文档分析、方案制定、现场测试、问题单提交和测试报告。

验收测试内容包括：功能度、安全可靠性、易用性、可扩充性、兼容性、效率、资源占用率和用户文档 8 个方面。

2．高级确认

高级确认测试是商品化软件的高级产品认证，也是中国软件评测中心最高级别的确认测试。该测试为用户提供整体的产品测试服务方案，测试规程严格监控，基于用户认可范围的应用测试与评估。

测试从功能度、安全性、可靠性、兼容性、可扩充性、性能、资源占用率、易用性、用户文档和用户满意度调查 10 个质量特性给予评分等级，高级确认测试结果分为确认和优秀 2 个等级。

中国软件评测中心所提交的高级确认测试报告和证书上加盖中国软件评测中心、中国实验室国家认可委员会（CNACL）、国家质量技术监督局计量认证标志（CMA）3 枚印章，该报告具有国际互认性。

高级确认测试不同于常规确认测试：首先，测试项目的质量特性不同，区别项为安全性、可靠性、用户调查项。参加高级确认测试的软件产品必须是开发后试运行半年以上，有固定的客户群或试运行客户，同时参加中国软件评测中心高级确认测试的软件必须是比较成熟的商品化软件产品。

高级确认测试过程中，开发商可以根据评测中心提交的问题单进行两次回归，改进并提高软件质量。

3．常规确认

常规确认测试是对商品化软件的普通级别产品认证，通过常规确认测试，对软件产品在功能度、安全可靠性、兼容性、可扩充性、效率、资源占用、易用性和用户文档 8 个质量特性给予测试评价，测试通过后颁发确认测试证书和报告，加盖中国软件评测中心公章，是国内权威的测试类型之一。

下面是测试项说明。

- 功能度：软件产品实现预期功能承诺，包括功能正确性与数据准确性，主要检查软件产品是否满足规格说明及用户文档承诺实现的功能要求。
- 安全可靠性：安全可靠性是软件非常重要的质量特性指标，软件系统的安全管理、数据安全、权限管理及防止对程序及数据的非授权的故意或意外访问的能力有关的软件属性。可靠性指与在规定的一段时间和条件下，软件能维持其性能水平能力有关的一组属性。
- 兼容性：考察系统在软件、硬件及数据格式的兼容性方面的应用表现。
- 可扩充性：测试软件产品是否留有与异种数据的接口，采用模块化开发方式，满足业务模块的扩充需求。
- 效率：软件运行效率、数据处理的响应时间。
- 资源占用：主要是测试软件安装/卸载前后对计算机资源的占用情况，包括软件

对内存的占用率、对 CPU 的占用率以及占用的硬盘空间等指标。

- 易用性：考察评定软件的易学易用性，各个功能是否易于完成，软件界面是否友好等方面进行测试，这在很多类型的管理类软件中是非常重要的。
- 用户文档：考察软件用户文档的安装。这个测试项着重测试的是软件相关文档描述与软件实际功能一致性，文档的易理解程度等。

4．鉴定测试

中国软件评测中心作为国家级软件质量鉴定机构，对申报部委科技项目、科技成果的软件产品或工程项目进行技术鉴定，作为项目申报或专家鉴定的技术依据。从技术与应用的角度对软件应用情况作全面的质量评测，作为申报国家部委项目及科技成果奖的技术鉴定依据(测试方法、测试过程、测试收费标准和常规确认测试相同)。中国软件评测中心根据测试结果出具详细的技术报告，测试结果为“通过”、“不通过”2 种。

5．登记测试

为配合信息产业部软件企业认定和软件产品登记而进行的测试工作。登记测试采用固定的测试规范和测试报告，该报告仅供软件产品登记和软件企业认定工作用。登记测试的测试周期一般为 6 天左右。需要提交的资料：详细的用户操作说明书及安装手册、详细的系统应用环境(包括硬件配置、平台、数据库等)、测试样品盘 1 套。

6．政府委托测试

中国软件评测中心除接受国家质量技术监督局和信息产业部每年委托的全国软件质量监督抽查测试外，还接受各有关部委和政府单位的委托测试，测试方案和测试费用由委托单位和评测中心根据有关文件的精神协商而定，测试流程控制与常规确认测试相同。

7．法律鉴定测试

接受司法部门、检察部门、消协、用户协会委托进行软件法律仲裁测试。

8．媒体试用测试

软件产品的试用测试，在相关媒体上发表 2000 字左右的文章介绍产品性能和特点。

9．单元测试/集成测试

为帮助软件企业在开发过程中严把测试关，中国软件评测中心提供单元测试和集成测试等白盒测试。单元测试主要是在软件开发过程中针对程序模块进行正确性检验。集成测试是在单元测试的基础上将所有模块按照设计要求组装成系统或子系统，对模块组装过程和模块接口进行正确性检验。测试过程严格监控，对测试规范、测试问题、回归测试、测试报告进行严格评审，最大限度的保证测试质量。

10．单项功能确认测试

中国软件评测中心单项功能确认测试，严格按照中国实验室国家认可委员会认可的质量管理体系和单项功能确认测试规程进行测试。根据厂商具体测试需求，对产品的某些功能进行测试，测试流程控制和常规确认测试相同。颁发单项功能测试报告，加盖中国软件评测中心公章。

在每月《中国计算机报》上公布的“中国软件评测中心测试公告”中，刊登被测产品的基本信息。在 CSTC 网站(www.cstc.org.cn)、赛迪网(www.ccidnet.com)、《中国计算机报》和 CSTC 自办发行的刊物上刊登被测产品的基本信息。根据测试模块和测试工作

量收费。

作为专业的评测机构，中国软件评测中心在信息工程控制和检测方面有着独特的优势，把专业评测手段运用到监理工作中，将大大增强项目开发和实施过程的质量控制能力。

7.4 面向对象软件测试

在编程语言中，面向对象编程的特性显然对测试的某些方面有影响。比如，类的继承和接口等特性支持多态，使得代码可以控制对象而不管对象到底是什么。测试者必须确保代码能正常工作而不管与那个对象确切相关的类是什么。支持和加强数据隐藏的特性可能使测试复杂化，因为有时为了支持测试必须向一个类的接口添加操作。与此同时，这些特性都能更好地、重复地使用测试软件。

编程语言的改变对测试有影响，开发过程的变化以及分析和设计重点的改变也会对测试产生影响。许多面向对象的软件的测试活动都可以在传统的过程中找到对应的活动。最重要的不同是面向对象的软件被设计成一系列的对象，这些对象从根本上形成了问题的模型，并由这些对象共同作用于一个解决方案。这种解决办法基于这样一种概念：问题的解决方案可能需要经常改变，而问题本身的结构和组件却不需要经常改变。因此，如果程序是以问题为出发点构建的（而不是直接构建在需求方案上），将更能适应以后的变化。

以往针对面向功能的程序所使用的一般测试原则仍可用于面向对象的程序，即通过生成测试数据、中间结果的检查或验证、程序运行路径的跟踪以及测试事件的指明等工作，使被测试程序从确定的初始状态开始，对测试数据进行处理而产生一个实际结果状态。只有当程序的结果状态与预想的结果状态相一致时，才确认该程序。但是，这种观点在面向对象的环境中只对系统测试有意义，而对单元测试和综合测试不适用。

7.4.1 面向对象测试模型

面向对象程序的结构不再是传统的功能模块结构，作为一个整体，原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。而且，面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同以往的要求和结果,已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析（OOA），面向对象设计（OOD）和面向对象编程（OOP）3 个阶段。针对这种开发模型，结合传统的测试步骤的划分，把面向对象的软件测试分为：面向对象分析的测试，面向对象设计的测试，面向对象编程的测试，面向对象单元测试，面向对象集成测试，面向对象系统测试。

7.4.2 面向对象分析的测试

传统的面向过程分析是一个功能分解的过程，是把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。而面向对象分析（OOA）是把 E-R 图和语义网络模型，即信息造型中的概念，与面向对象程序设计语言中的重要概念结合在一起而形成的分析方法，最后通常是得到问题空间的图表的形式描述。OOA 直接映射问题空间，全面的将问题空间中实现功能的现实抽象化。将问题空间中的实例抽象为对象，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和操作表示实例的特性和行为。对一个系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性。OOA 的结果是为后面阶段类的选定和实现，类层次结构的组织和实现提供平台。因此，对 OOA 的测试，应该从以下几个方面考虑：

（1）对认定的对象的测试；
（2）对认定的结构的测试；
（3）对认定的主题的测试；
（4）对定义的属性和实例关联的测试；
（5）对定义的服务和消息关联的测试。

7.4.3 面向对象设计的测试

通常的结构化的设计方法，用的是面向作业的设计方法，把系统分解以后，提出一组作业，这些作业是以过程实现系统的基础构造，把问题域的分析转化为求解域的设计，分析的结果是设计阶段的输入。而面向对象设计（OOD）采用造型的观点，以 OOA 为基础归纳出类，并建立类结构或进一步构造成类库，实现分析结果对问题空间的抽象。由此可见，OOD 不是在 OOA 上的另一思维方式的大动干戈，而是 OOA 的进一步细化和更高层的抽象。所以，OOD 与 OOA 的界限通常是难以严格区分的。OOD 确定类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便实现功能的重用和扩增，以不断适应用户的要求。因此，对 OOD 的测试，应从如下 3 个方面考虑：

（1）对认定的类的测试；
（2）对构造的类层次结构的测试；
（3）对类库的支持的测试。

7.4.4 面向对象编程的测试

典型的面向对象程序具有继承、封装和多态的特性，这使得传统的测试策略必须有所改变。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降

低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点，继承使得代码的重用率提高，同时也使错误传播的概率提高。多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内同一函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。因此，在面向对象编程（OOP）阶段，忽略类功能实现的细则，将测试的目光集中在类功能的实现和相应的面向对象程序风格，主要体现为以下两个方面：

（1）数据成员是否满足数据封装的要求；

（2）类是否实现了要求的功能。

7.4.5 面向对象的单元测试

传统的单元测试的对象是软件设计的最小单位——模块。单元测试的依据是详细设计描述，单元测试应对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。单元测试多采用白盒测试技术，系统内多个模块可以并行地进行测试。

当考虑面向对象软件时，单元的概念发生了变化。封装驱动了类和对象的定义，这意味着每个类和类的实例(对象)包装了属性(数据)和操纵这些数据的操作。最小的可测试单位是封装的类或对象，而不是个体的模块。类包含一组不同的操作，并且某特殊操作可能作为一组不同类的一部分存在，因此，单元测试的意义发生了较大变化。单元测试不再孤立地测试单个操作，而是将操作作为类的一部分。

7.4.6 面向对象的集成测试

传统的集成测试，有两种方式通过集成完成的功能模块进行测试。自顶向下集成：自顶向下集成是构造程序结构的一种增量式方式，从主控模块开始，按照软件的控制层次结构，以深度优先或广度优先的策略，逐步把各个模块集成在一起。自底向上集成：自底向上测试是从“原子”模块（即软件结构最低层的模块）开始组装测试。

因为面向对象软件没有层次的控制结构，传统的自顶向下和自底向上集成策略就没有意义，此外，一次集成一个操作到类中（传统的增量集成方法）经常是不可能的，这是由于“构成类的成分的直接和间接的交互”。对面象对象软件的集成测试有两种不同策略，第一种称为基于线程的测试，集成对回应系统的一个输入或事件所需的一组类，每个线程被集成并分别测试，应用回归测试以保证没有产生副作用。第二种称为基于使用的测试，通过测试那些几乎不使用服务器类的类（称为独立类）而开始构造系统，在独立类测试完成后，下一层的使用独立类的类，称为依赖类，被测试。这个依赖类层次的测试序列一直持续到构造完整个系统。

7.4.7　面向对象的系统测试

通过单元测试和集成测试，仅能保证软件开发的功能得以实现。但不能确认软件在实际运行时，是否满足用户的需要。为此，对完成开发的软件必须经过规范的系统测试。系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。系统测试时，应该参考 OOA 的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全再现问题空间。系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认。

7.5　典型应用分析

1．测试用例

（1）软件测试用例的定义

软件测试用例可以被定义为如下 6 元组：测试索引，测试环境，测试输入，测试操作，预期结果，评价标准。表 7-2 给出软件测试用例的定义。

表 7-2　软件测试用例的定义

软件测试用例		
元　素	含　义	给出定义的测试角色
测试索引	被标识过的测试需求	测试需求分析
测试环境	进入测试实施步骤所需的资源及其状态	测试设计（描述性定义） 测试实现（计算机表示）
测试输入	运行本测试所需的代码和数据，包括测试模拟程序和测试模拟数据	
测试操作	建立测试运行环境，运行被测试对象，获取测试结果的步骤序列	
预期结果	用于比较测试结果的基准	
评价标准	根据测试结果与预期结果的偏差，判断被测对象质量状态的依据	

（2）软件测试用例的生成和执行

软件测试的核心任务是生成和执行软件测试用例。由表 7-2 知，在软件测试用例的 6 元组定义中：

① 测试索引和测试环境在测试需求分析步骤中定义，是软件测试计划的内容；

② 测试输入、测试操作、预期结果和评价标准的描述性定义在软件设计步骤中定义，是软件测试说明的内容；

③ 测试输入、测试操作、预期结果和评价标准的计算机表示（代码/数据定义）在软件测试实现步骤中给出，是软件测试程序产品。

软件测试用例是软件测试结果的生成器，即每执行一次测试用例都产生一组测试结果。若测试用例被有效地由描述性定义转换为计算机表示，则测试的执行和结果的比较都可以利用软件测试工具自动或半自动地执行，在需要大量回归测试的复杂软件系统中，这种转换和自动执行是降耗增质的关键策略之一。

（3）软件测试用例的配置管理

基于以下原因，对软件测试用例需要进行配置管理：

① 大型复杂软件系统的功能/性能要求将对应于大量的软件测试用例，这些测试用例需要标识规则和规范的存储结构；

② 软件测试用例也存在引用控制；

③ 软件测试用例也存在版本控制；

④ 软件测试用例也存在更动控制。

软件测试用例的配置管理类似于一般软件的配置管理，可以实现安全存储、追踪变化和并行开发，其特色在于：区分测评人员和一般测试人员，前者独具生成和更新测试基准（预期结果的计算机表示）的权限。

（4）软件测试用例的组织

软件测试用例的设计和实现对应于被测对象的需求、设计和环境要求，因此同被测对象一样，软件测试用例可以被组织成层次结构，即：依据某种原则（如被测对象的层次或测试类型）将测试用例划分为测试用例组；测试用例组又可以划分为更高层次的测试用例组。测试用例组反映多个测试用例/测试用例组之间的偏序关系，也标识了具有某种共性的测试用例的集合。测试实施时可以根据具体需要/环境，选择性地执行多个测试用例/测试用例组。

（5）软件测试用例的复用

测试用例的层次性还表现在：低层被测对象的测试用例或其部分内容可以复用在对高层被测对象的测试中。

① 单元测试阶段的功能确认类测试用例组可以复用在部件集成测试阶段中；

② 部件确认测试阶段可以复用单元测试阶段的测试输入；

③ 部件确认测试阶段的测试用例组可以复用在配置项组装测试阶段和配置项确认测试阶段中；

④ 配置项确认测试阶段的测试用例组可以复用在系统综合测试阶段和系统验收测试中。

当然，每个测试阶段的对象和目标都不同，因此测试用例或其部分内容的复用通常有选择的、有限的和需更改的。

2．设计测试方案

设计测试方案是测试阶段的关键技术问题。所谓测试方案包括预定要测试的功能，应输入的测试数据和预期的结果。其中最困难的问题是设计测试用的输入数据（即，测试用例）。

设计方案的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。下面主要介绍的设计技术有：适用于黑盒测试的等价类划分、边值分析以及错误推测法；适用于白盒测试的逻辑覆盖法。

（1）等价类划分

等价类划分是一种典型的黑盒测试方法，用这一方法设计测试用例可以不用考虑程序的内部结构，只以对程序的要求和说明，即需求规格说明书为依据，仔细分析和推敲说明书的各项需求，特别是功能需求，把说明中对输入的要求和输出的要求区别开来并

加以分解。

等价类划分的办法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据当作测试用例。每一类的代表性数据在测试中的作用等价于这一类中的其他值，也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能出现同样的错误。使用这一方法设计测试用例，首先必须在分析需求规格说明的基础上划分等价类，列出等价类表。

在考虑等价类划分时，先从程序的功能说明中找出每个输入条件，然后为每个输入条件划分两个或更多个等价类。等价类可分两种情况：有效等价类和无效等价类。有效等价类是指对程序的规格说明是有意义的、合理的输入数据所构成的集合；无效等价类是指对程序的规格说明是不合理的或无意义的输入数据所构成的集合。

（2）边界值分析

软件测试常用的一个方法是把测试工作按同样的形式划分。对数据进行软件测试，就是检查用户输入的信息、返回结果以及中间计算结果是否正确。

即使是最简单的程序，要处理的数据也可能数量极大。还记得在计算器上简单加法的全部可能性吗？再想一想字处理程序、导航系统和证券交易程序。使这些数据得以测试的技巧（如果称得上的话）是，根据下列主要原则进行等价分配，以合理的方式减少测试案列：边界条件、次边界条件、空值和无效数据。

边界值分析（boundary value analysis，BVA）是一种补充等价划分的测试用例设计技术，不是选择等价类的任意元素，而是选择等价类边界的测试用例。实践证明，在设计测试用例时，对边界附近的处理必须给予足够的重视，为检验边界附近的处理专门设计测试用例，常常可以取得良好的测试效果。BVA 不仅重视输入条件边界，而且也从输出域导出测试用例。

边界值设计测试遵循以下 5 条原则：

① 如果输入条件规定了取值范围，应以该范围的边界内及刚刚超范围边界外的值作为测试用例。如以 a 和 b 为边界，测试用例应当包含 a 和 b 及略大于 a 和略小于 b 的值；

② 若规定了值的个数，分别以最大、最小个数及稍小于最小、稍大于最大个数作为测试用例；

③ 针对每个输出条件使用上述①、②条原则；

④ 如果程序规格说明中提到的输入或输出域是个有序的集合（如顺序文件、表格等），就应注意选取有序集的第一个和最后一个元素作为测试用例；

⑤ 分析规格说明，找出其他的可能边界条件。

（3）错误推测

错误推测法在很大程度上靠直觉和经验进行。错误推测法的基本想法是列举程序中可能有的错误和容易发生错误的特殊情况，并且根据这些特殊情况选择测试方案。一般说来，即使是一个比较小的程序，可能的输入组合数也往往十分巨大，因此必须依靠测试人员的经验和直觉，从各种可能的测试方案中选出一些最可能引起程序出错的方案。对于程序中可能存在哪类错误的推测，是挑选测试方案时的一个重要因素。

（4）逻辑覆盖

有选择地执行程序中某些最有代表性的通路是对穷尽测试的惟一可行的替代办法。所谓逻辑覆盖是对一系列测试过程的总称，这组测试过程逐渐进行越来越完整的通路测试。测试数据执行（或叫覆盖）程序逻辑的程度可以划分成哪些不同的等级呢？从覆盖源程序语句的详尽程度分析，大致有以下一些不同的覆盖标准。

① 语句覆盖

为了暴露程序中的错误，至少每个语句应该执行一次。语句覆盖的含义是，选择足够多的测试数据，使被测试程序中每个语句至少执行一次。

例如，图 7-1 是一个被测模块的流程图，源程序如下：

```
PROCEDURE EXAMPLE(A,B：REAL；VAR X：REAL) ；
BEGIN
IF(A>1)AND(B=0)
        THEN   X:=X/A;
IF(A=2)OR(X>1)
        THEN   X:=X+1；
END;
```

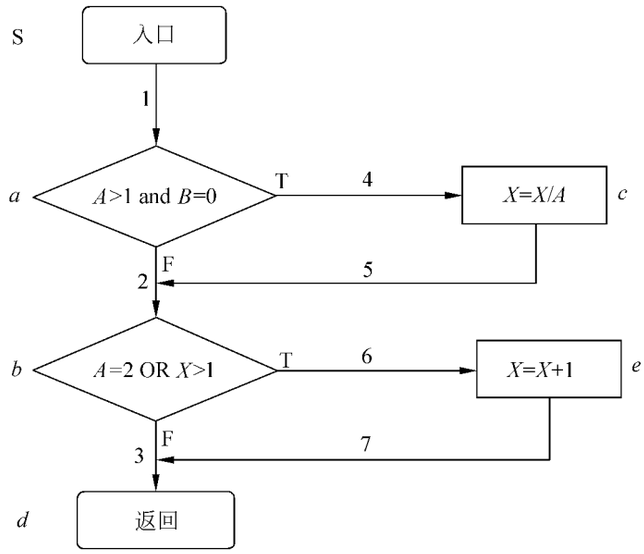


图 7-1　被测试模块的流程图

为了使每个语句都执行一次，程序的执行路径应该是 *sacbed*，为此只需要输入下面的测试数据（实际上 *X* 可以是任意实数）：

A=2，*B*=0，*X*=4

语句覆盖对程序的逻辑覆盖很少，在上面例子中两个判定条件都只测试了条件为真的情况，如果条件为假时处理有错误，显然不能接受。此外，语句覆盖只关心判定表达式的值，而没有分别测试判定表达式中每个条件取不同值时的情况。在上面的例子中，为了执行 *sacbed* 路径，以测试每个语句，只需两个判定表达式(*A*>1) AND (*B*=0)和

(*A*=2)OR(*X*>1)都取真值，因此使用上述一组测试数据就够了。

综上所述，可以看出语句覆盖是很弱的逻辑覆盖，为了更充分地测试程序，可以采用下述的逻辑覆盖标准。

② 判定覆盖

判定覆盖又叫分支覆盖，含义是，不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次，也就是每个判定的每个分支都至少执行一次。

对于图 7-1 所示的例子，用下面两组测试数据就可做到判定覆盖：

Ⅰ. *A*=3, *B*=0, *X*=3 （覆盖 *sacbd*）

Ⅱ. *A*=2, *B*=1, *X*=1 （覆盖 *sabed*）

判定覆盖比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高，例如上面的测试数据只覆盖了程序全部路径的一半。

③ 条件覆盖

条件覆盖的含义是，不仅每个语句至少执行一次，而且使判定表达式的每个条件都取到各种可能的结果。图 7-1 的例子中共有两个判定表达式，每个表达式中有两个条件，为了做到条件覆盖，应该选取测试数据使得在 *a* 点有下述各种结果出现：

A>1, *A*≤1, *B*=0, *B*≠0

在 *b* 点有下述各种结果出现：

A=2, *A*≠2, *X* >1, *X*≤1

只需要使用下面两组测试数据就可以达到上述覆盖标准：

Ⅰ. *A*=2, *B*=0, *X*=4 （满足 *A*>1, *B*=0, *A*=2 和 *X* >1 的条件，执行路径 *sacbed*）

Ⅱ. *A*=1, *B*=1, *X*=1 （满足 *A*≤1, *B*≠0, *A*≠2 和 *X*≤1 的条件，执行路径 *sabd*）

条件覆盖通常比判定覆盖强，因为条件覆盖使判定表达式中每个条件都取到了两个不同的结果，判定覆盖却只关心整个判定表达式的值。但是，也可能有相反的情况：虽然每个条件都取到了两个不同的结果，判定表达式却始终只取一个值。例如，如果使用下面两组测试数据，则只满足条件覆盖标准并不满足判定覆盖标准（第二个判定表达式的值总为真）：

Ⅰ. *A*=2, *B*=0, *X*=1 （满足 *A*>1, *B*=0, *A*=2 和 *X*≤1 的条件，执行路径 *sacbed*）

Ⅱ. *A*=1, *B*=1, *X*=2 （满足 *A*≤1, *B*≠0, *A*≠2 和 *X*>1 的条件，执行路径 *sabed*）

④ 判定/条件覆盖

既然判定覆盖不一定包含条件覆盖，条件覆盖也不一定包含判定覆盖，自然会提出一种能同时满足这两种覆盖标准的逻辑覆盖，这就是判定/条件覆盖。判定/条件覆盖的含义是，选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。

对于图 7-1 的例子而言，下述两组测试数据满足判定/条件覆盖标准：

Ⅰ. *A*=2, *B*=0, *X*=4

Ⅱ. *A*=1, *B*=1, *X*=1

但是，这两组测试数据也就是为了满足条件覆盖标准最初选取的两组数据，因此，有时判定/条件覆盖也并不比条件覆盖更强。

⑤ 条件组合覆盖

条件组合覆盖是更强的逻辑覆盖标准，要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。对于图 7-1 的例子，共有 8 种可能的条件组合，如下所示：

Ⅰ. $A>1, B=0$

Ⅱ. $A>1, B\neq 0$

Ⅲ. $A\leq 1, B=0$

Ⅳ. $A\leq 1, B\neq 0$

Ⅴ. $A=2, X>1$

Ⅵ. $A=2, X\leq 1$

Ⅶ. $A\neq 2, X>1$

Ⅷ. $A\neq 2, X\leq 1$

和其他逻辑覆盖判定标准中的测试数据一样，条件组合Ⅴ～Ⅷ中的 X 值是指在程序流程图第二个判定框（b 点）的 X 值。

下面的 4 组测试数据可以使上面列出的 8 种条件组合每种至少出现一次：

Ⅰ. $A=2, B=0, X=4$　（针对Ⅰ和Ⅴ两种组合，执行路径 *sacbed*）

Ⅱ. $A=2, B=1, X=1$　（针对Ⅱ和Ⅵ两种组合，执行路径 *sabed*）

Ⅲ. $A=1, B=0, X=2$　（针对Ⅲ和Ⅶ两种组合，执行路径 *sabed*）

Ⅳ. $A=1, B=1, X=1$　（针对Ⅳ和Ⅷ两种组合，执行路径 *sabd*）

显然，满足条件组合覆盖标准的测试数据，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此，条件组合覆盖是前述几种覆盖标准中最强的。但是，满足条件组合覆盖标准的测试数据并不一定能使程序中的每条路径都执行，例如上述四组测试数据都没有测试到路径 *sacbd*。

⑥ 点覆盖

图论中点覆盖的概念定义如下：如果连通图 G 的子图 G' 是连通的，而且包含 G 的所有节点，则称 G' 是 G 的点覆盖。在正常情况下程序图是连通的有向图，图中每个节点相当于流程图的一个框（一个或多个语句）。满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过程序图中每节点一次。所以点覆盖标准和语句覆盖标准是相同的。

⑦ 边覆盖

图论中边覆盖的定义是：如果连通图 G 的子图 G' 是连通的，而且包含 G 的所有边，则称 G' 是 G 的边覆盖。

为了满足边覆盖的测试标准，要求选取足够多的测试数据，使得程序执行路径至少经过程序图中每条边一次。例如，以图 7-1 为例。为了使程序经过程序图的边覆盖（1，2，3，4，5，6，7），至少需要两组测试数据（分别执行路径 1－2－3 和 1－4－5－6－7；或者分别执行路径 1－4－5－3 和 1－2－6－7）。通常边覆盖和判定覆盖是一致的。

⑧ 路径覆盖

路径覆盖的含义是，选取足够多的测试数据，使程序的每条可能路径都至少执行一次（如果程序图中有环，则要求每个环至少经过一次）。

在图 7-1 的例子中共有 4 条可能的路径：1－2－3；1－2－6－7；1－4－5－3；1－4－5－6－7。因此对于这个例子而言，为了做到路径覆盖必须设计 4 组测试数据。例如，

下面4组测试数据可以满足路径覆盖的要求：

Ⅰ. A=1, B=1, X=1　　(执行路径 1－2－3)

Ⅱ. A=1, B=1, X=2　　(执行路径 1－2－6－7)

Ⅲ. A=3, B=0, X=1　　(执行路径 1－4－5－3)

Ⅳ. A=2, B=0, X=4　　(执行路径 1－4－5－6－7)

路径覆盖是相当强的逻辑覆盖标准，能保证程序中每条可能的路径都至少执行了一次，因此这样的测试数据更具有代表性，暴露错误的能力也比较强。但是，为了做到路径覆盖只需考虑每个判定表达式的取值，并没有检验表达式中条件的各种可能组合情况。如果把路径覆盖和条件组合覆盖结合起来，可以设计出检错能力更强的测试数据。对于图7-1的例子，只要把路径覆盖的第三组测试数据和前面给出的条件组合覆盖的4组测试数据联合起来，共有5组数据，就可以做到既满足路径覆盖标准又满足条件组合覆盖标准。

3．实用策略

以上简单介绍了设计测试方案的几种基本方法，使用每种方法都能设计出一组有用的测试方案，但是没有一种方法能设计出全部测试方案。此外，不同方法各有所长，用一种方法设计出的测试方案可能最容易发现某些类型的错误，对另外一些类型的错误可能不易发现。

因此，对软件系统进行实际测试时，应该联合使用各种设计测试方案的方法，形成一种综合策略。通常的做法是，用黑盒法设计基本的测试方案，再用白盒法补充一些必要的测试方案。具体地说，可以使用下述策略结合几种方法。

(1) 在任何情况下都应该使用边界值分析的方法。经验表明，用这种方法设计出的测试方案暴露程序错误的能力最强。注意，应该既包括输入数据的边界情况又包括输出数据的边界情况。

(2) 用等价划分法补充测试方案。

(3) 用错误推测法补充测试方案。

(4) 对照程序逻辑，检查程序逻辑，检查已经设计出的测试方案。可以根据对程序可靠性的要求采用不同的逻辑覆盖标准，如果现有测试方案的逻辑覆盖程度没有达到要求的覆盖标准，则应再补充一些测试方案。

应该强调指出，即使使用上述综合策略设计测试方案，仍然不能保证测试将发现一切程序错误；但是，这个策略确实是在测试成本和测试效果之间的一个合理的折衷。通过前面的叙述可以看出，软件测试确实是一件十分艰巨繁重的工作。

本 章 小 结

目前软件测试仍然是保证软件可靠性的主要手段。测试阶段的根本任务是发现并改正软件中的错误。软件测试是软件质量保证的重要手段。有些研究数据显示，国外软件开发机构40%的工作量花在软件测试上，软件测试费用占软件开发总费用的30%至50%。对于一些要求高可靠、高安全的软件，测试费用可能相当于整个软件项目开发所有费用的3至5倍。由此可见，要成功开发出高质量的软件产品，必须重视并加强软件测试工作。

软件测试是软件开发过程中最艰巨最繁重的任务，大型软件的测试应该分阶段地进

行，通常至少分为单元测试、集成测试和系统测试等几个阶段。

设计测试方案是测试阶段的关键技术问题，基本目标是选用最少量的高效测试数据，做到尽可能完善的测试，从而尽可能多地发现软件中的问题。设计测试方案的实用策略是，用黑盒法（边界值分析、等价划分和错误推测法等）设计基本的测试方案，再用白盒法补充一些必要的测试方案。

目前国内的软件测试一般有下列几种形式：一是软件公司内部进行的功能性测试，主要是验证设计的功能是否完成；二是用户进行的测试，大量的用户一起寻找使用中遇到的错误；还有就是第三方测试，就是专业软件测试人员运用一定的测试工具对软件的质量进行检测。在软件业较发达的国家，绝大多数软件产品的认定，需要第三方测试的介入。而在国内，仅有软件公司的自测是很不完善的。

测试计划、测试方案和测试结果是软件配置的重要成分，对软件的可维护性影响很大，因此必须仔细记录和保存。

习 题 7

7.1 请描述软件测试活动的生命周期。

7.2 画出软件测试活动的流程图。

7.3 简述静态测试和动态测试的区别。

7.4 对图 7-2 给出的程序控制图，分别以各种不同的测试方法写出最少的测试用例。

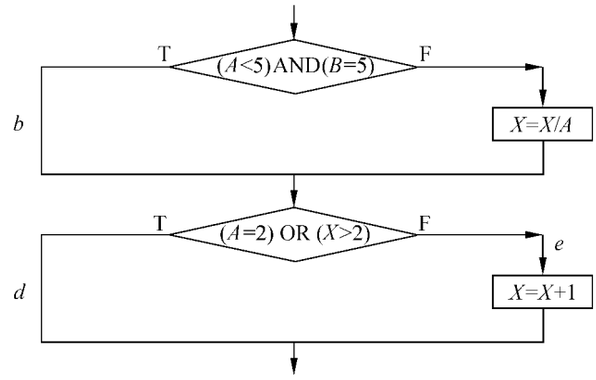


图 7-2 被测程序的程序控制图

7.5 在三角形计算中，要求三角形的三个边长：*A*、*B* 和 *C*。当三边不可能构成三角形时提示错误，可构成三角形时计算三角形周长。若是等腰三角形打印“等腰三角形”，若是等边三角形，则提示“等边三角形”。画出程序流程图、控制流程图、找出基本测试路径，对此设计一个测试用例。

第8章

软件维护 1, 14, 15

软件维护是软件生命期的最后一个阶段，处于系统投入生产运行以后的时期中，不属于软件开发过程。随着软件的大型化和使用寿命的延长，软件的维护费用日益增长。平均来说，大型软件的维护成本高达开发成本的 4 倍左右。因此，提高软件的可维护性，开发有效的软件维护支持工具，降低软件维护费用，满足用户新的需要以延长软件寿命就成为软件生命期中不可缺少的工作。

传统上，软件系统交付之后对其实施更改的工作叫做软件维护。在过去的几十年里，软件系统一直在广泛的工作环境中发挥作用。在日常运营中使用软件系统的各行各业数不胜数，包括制造业、财务公司、信息服务公司、医疗服务机构和建筑业。人们越来越严重地依赖软件系统，软件系统越来越重要，这些系统不仅要完成期望完成的工作，而且还要完成得很好。换句话说，系统的有用性是至关重要的。如果系统失去了有用性，就不能被用户接受，也就是不能被使用。

在当今世界中，正确地使用软件系统，发挥系统的作用，可能是生死攸关的大问题。软件系统有用性中包含的一些要素是功能、灵活性、程序可用性和正确操作。为了满足性能改进、功能增强或处理系统中发现的错误的要求，可能需要进行一些更改。

软件工程师所面临的最大挑战是，软件系统更改的管理和控制。交付软件系统之后使系统能够正常运行所花费的时间和工作，可以清晰地说明这一点。有关软件系统交付之后对系统实施更改的特点和成本调查结果说明，软件系统整个生存周期总成本的大约 40%到 70%要用于软件维护。

8.1 软件维护的基本概念

软件维护是软件生命周期中的最后一个阶段，处于系统投入生产性运行以后的时期中，因此不属于系统开发过程。

软件维护需要的工作量非常大，虽然在不同应用领域维护成本差别很

大，但是，平均说来，大型软件的维护成本高达开发成本的 4 倍左右。目前国外许多软件开发组织把 60%以上的人力用于维护已有的软件，而且随着软件数量增多和使用寿命延长，这个百分比还在持续上升。将来维护工作甚至可能会束缚软件开发组织的手脚，使软件开发组织没有余力开发新的软件。

本书前面各章讲述的软件工程方法学的主要目的就是要提高软件的可维护性，减少软件维护所需要的工作量，降低软件系统的总成本。

8.1.1 软件维护的定义

所谓软件维护就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程，即软件在交付使用后对软件所做的一切改动。

为了理解软件维护，需要清楚什么是“软件”这个词的真正含义。人们常常有错误的概念，认为软件就是程序。这种错误概念会导致对包括“软件”在内的一些术语的错误理解。例如，在考虑软件维护活动时，往往只考虑对程序所实施的活动，而不是软件系统的其他成分。McDermid 的定义明确说明，软件不仅仅包括程序，即源代码和目标代码，还包括程序所涉及的各种文档，例如需求分析、规格说明、设计、系统和用户手册、设置并操作软件系统所使用的规程。

8.1.2 软件维护的分类

为了达到人们所需要的维护目标，可能需要对软件产品进行各种各样的更改。有很多作者都试图对这些更改分类，导致有大量由纠正性、适应性、完善性和预防性更改构成的分类。根据这几种更改，可以把维护分为改正性维护、适应性维护、完善性维护和预防性维护。如图 8-1 所示。

改正性维护：诊断并修改程序在使用过程中出现的各种错误。因为软件测试不可能暴露出一个大型软件系统中所有潜藏的错误，所以必然会有第一项维护活动：在任何大型程序的使用期间，用户必然会发现程序错误，并且把遇到的错误报告给维护人员。

适应性维护：为和变化了的环境适当地匹配而进行的修改软件的活动。计算机科学领域的各个方面都在迅速进步，大约每过 36 个月就有新一代的硬件宣告出现，经常推出新操作系统或旧系统的修改版本，时常增加或修改外部设备和其他系统部件；另一方面，应用软件的使用寿命却很容易超过十年，远远长于最初开发这个软件时的运行环境的寿命。因此，适应性维护，也就是为了和变化了的环境适当地配合而进行的修改软件的活动，是既必要又经常的维护活动。

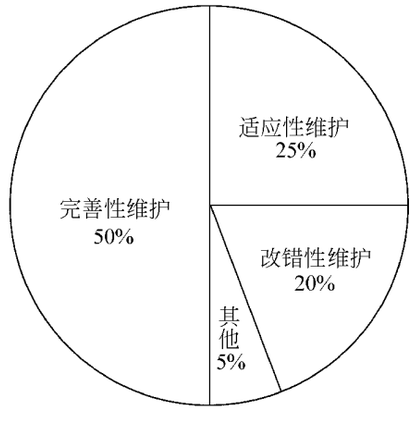


图 8-1 各类维护的工作量的百分比

完善性维护：在软件交给用户使用后，用户

往往会要求修改现有功能或增加新的功能等，这样，就需要进行完善性维护。当一个软件系统顺利地运行时，常常出现第三项维护活动：在使用软件的过程中用户往往提出增加新功能或修改已有功能的建议，还可能提出一般性的改进意见。为了满足这类要求，需要进行完善性维护。这项维护活动通常占软件维护工作的大部分。

预防性维护：为了进一步提高软件的可维护性和可靠性，需要进行预防性维护。为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件时，出现了第四项维护活动。这项维护活动通常称为预防性维护，目前这项维护活动相对来说比较少。

从上述关于软件维护的定义不难看出，软件维护绝不仅限于纠正使用中发现的错误，事实上在全部维护活动中一半以上是完善性维护。国外的统计数字表明，完善性维护占全部维护活动的50%～66%，改正性维护占17%～21%，适应性维护占18%～25%，其他维护活动只占4%左右。值得注意的是，上述4类维护活动不但要应用于软件可执行的代码，同样要应用于软件文档。

8.2　软件维护的特点及过程

8.2.1　影响软件维护的因素

1．维护的费用和代价

软件维护的成本是巨大的。随着技术的迅速发展，使新的系统几个月后就显得过时，大多数维护都是对在一定程度上过时的系统实施的。在过去的几十年中，软件维护的费用稳步上升。1970年用于维护已有软件的费用只占软件总预算的35%～40%，1980年上升为40%～60%，1990年上升为70%～80%。

软件维护工作量可分为生产性活动（分析评价、修改设计、编写代码等）和非生产性活动（理解代码功能、解释数据结构、接口特征与性能约束等）。维护工作量的一种估算模型是

$$M = P + K\mathrm{e}^{c-d}$$

其中，M是维护所用工作量；P是生产性工作量；K是经验常数；c是复杂度，标志设计好坏及文档完整程度；d是维护人员对该软件的熟悉程度。

维护费用仅仅是软件维护的明显代价。还有许多不明显的代价。因为可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机，这是软件维护的一个无形的代价。其他无形的代价还有：

（1）当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满；

（2）由于维护时的改动，在软件中引入了潜伏的故障，从而降低了软件的质量；

（3）当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱。

软件维护的最后一个代价是生产率的大幅度下降，这种情况在维护旧程序时常常遇到。例如，据Gausler在1976年的报道，美国空军的飞行控制软件每条指令的开发成本是75美元，然而维护成本大约是每条指令4000美元，也就是说，生产率下降了50倍

以上。

2．维护的问题

软件维护中出现的大部分问题都是由软件需求分析和设计过程的缺陷而引起的，在软件生存周期的最初两个阶段如果不进行严格而科学的管理和规划，必然会造成其生存周期最后的维护阶段产生种种问题。下面是与软件维护有关的常见问题。

（1）理解别人写的程序通常非常困难，而且困难程度随着软件配置成分的减少而迅速增加。如果仅有程序代码没有说明文档，则会出现严重的问题。

（2）需要维护的软件往往没有合格的文档，或者文档资料显著不足。认识到软件必须有文档仅仅是第一步，容易理解的并且和程序代码完全一致的文档才真正有价值。

（3）当要求对软件进行维护时，不能指望由开发人员仔细说明软件。由于维护阶段持续的时间很长，因此，当需要解释软件时，往往原来写程序的人已经不在附近了。

（4）绝大多数软件在设计时没有考虑将来的修改。除非使用强调模块独立原理的设计方法论，否则修改软件既困难又容易发生差错。

（5）软件维护不是一项吸引人的工作。形成这种观念很大程度上是因为维护工作经常遭受挫折。

注意：上述种种问题在现有的没采用软件工程思想开发出来的软件中，都或多或少地存在着。不应该把一种科学的方法论看作万应灵药，但是，采用软件工程的思想方法，可避免或减少上述问题。

8.2.2 软件维护的标准化

1989 年美国的 Schneidewind 强调了软件维护标准化的必要性，1993 年 IEEE 计算机学会的软件工程标准分委会颁布了 IEEE1219 即《软件维护标准》。

（1）修改请求：一般由用户、程序员或管理人员提出，是软件维护过程的开始。

（2）分类与鉴别：根据软件修改请求（MR），由维护机构来确认其维护的类别（纠错性、适应性还是完善性维护），即对 MR 进行鉴别并分类，并对该 MR 给予一个编号，然后输入数据库。这是整个维护阶段数据收集与审查的开始。

（3）分析：先进行维护的可行性分析，在此基础上进行详细分析。可行性分析主要确定软件更改的影响、可行的解决方法及所需的费用。详细分析则主要是提出完整的更改需求说明、鉴别需要更改的要素（模块）、提出测试方案或策略、制订实施计划。最后由配置控制委员会（CCB）审查并决定是否着手开始工作。

通常维护机构就能对更改请求的解决方案做出决策，仅仅需要通知 CCB 就可以了。但要注意的是维护机构应清楚哪些是可以进行维护的范围，哪些不是。CCB 要确定的是维护项目的优先级别，在此之前维护人员不应开展维护更改工作。

（4）设计：汇总全部信息开始着手更改，如开发过程的工程文档、分析阶段的结果、源代码、资料信息等。本阶段应更改设计的基线、更新测试计划、修订详细分析结果、核实维护需求。

实现：本阶段的工作是制订程序更改计划并进行软件更改。有如下工作：编码、单元测试、集成、风险分析、测试准备审查、更新文档。风险分析在本阶段的结束时进行。

所有工作应该置于软件配置管理系统的控制之下。

（5）系统测试：系统测试主要测试程序之间的接口，以确保加入了修改的软件满足原来的需求，回归测试则是确保不要引入新的错误。测试是十分重要的，但总是做不到那么好，有兴趣的读者可以参考阮教授的《软件测试技术》。测试有两种：手工测试和计算机测试。手工测试如走代码，这是保证测试成功的重要手段。值得注意的是许多维护机构都没有独立的测试组，而将这些工作交给维护编程人员来进行，这样做的风险很大。建立一个并行测试数据库对于软件维护成功具有十分重要的意义。如果没有就请建立一个。

（6）验收试验 ：这是全综合测试，应由客户、用户或第三方进行。此阶段应报告测试结果、进行功能配置审核、建立软件新版本、准备软件文档的最终版本。

（7）交付：此阶段是将新的系统交给用户安装并运行。供应商应进行实物配置审核、通知所有用户、进行文档版本备份、完成安装与训练。

8.2.3　软件维护的特点

图 8-2 描绘了作为维护要求的结果可能发生的事件流。

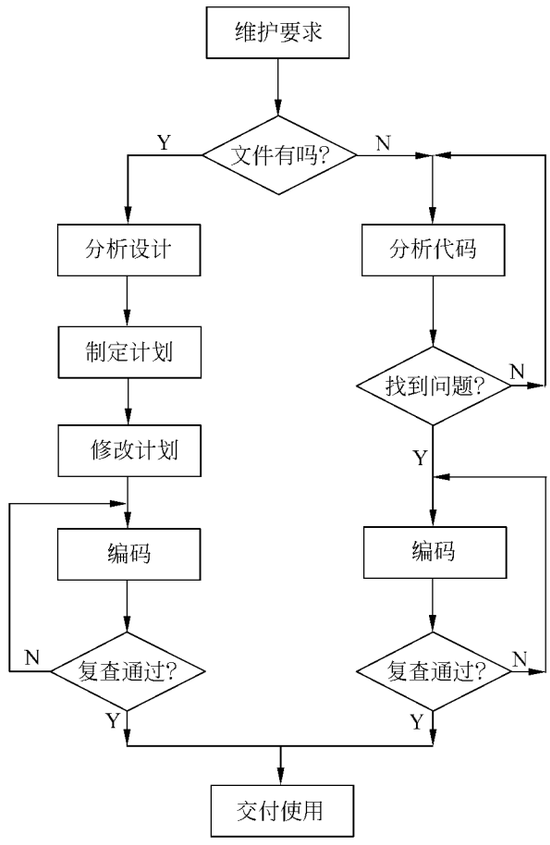


图 8-2　结构化维护与非结构化维护

图 8-2 中左半部分为结构化维护。从阅读需求、设计文档开始，从修改设计入手，对所作的修改进行复查，并重复过去的测试，以确保没有引入新的错误。这种以完备的文档为基础的维护称为结构化维护。

图 8-2 中右半部分为非结构化维护。没有完整的文档，维护只能从代码着手，后果难以估计，也不可能进行回归测试。这种维护称为非结构化维护。

8.2.4 软件维护过程

1．维护组织

并非每个软件开发机构都必须建立正式的维护组织，但至少应设立专门负责维护的非正式组织，起码应有一个了解整个软件配置、配合相应维护人员的组织。即使对于一个小的软件开发团体而言，非正式地委托责任也是绝对必要的。

每个维护要求都通过维护管理员转交给相应的系统管理员去评价。软件开发机构收到用户的维护要求后，交给负责此事的维护管理员，由维护管理员将申请交给适当的系统管理员去评价。系统管理员必须熟悉产品程序的一部分，特别应当对将被修改的部分熟悉。由系统管理员做出评价后再由修改管理员决定如何实行修改。在维护活动开始之前明确维护责任可以大大减少维护过程中可能出现的混乱，因而是十分必要的。

2．维护报告

1）维护要求表

软件维护人员应当向用户提供空白的维护要求表，由要求维护的用户填写，该表应能完整描述产生错误的情况（包括输入数据、输出数据及其他有关信息）。维护要求表由维护管理员和系统管理员负责研究处理。

2）软件修改报告

应该用标准化的格式表达所有软件维护要求。软件维护人员通常给用户提供空白的维护要求表——有时称为软件问题报告表，这个表格由要求一项维护活动的用户填写。如果遇到了一个错误，那么必须完成描述导致出现错误的环境（包括输入数据，全部输出数据，以及其他有关信息）。对于适应性或完善性的维护要求，应该提出一个简短的需求说明书。如前所述，由维护管理员和系统管理员评价用户提交的维护要求表。

维护要求表是一个外部产生的文件，是计划维护活动的基础。软件组织内部应该制定出一个软件修改报告，给出下述信息：

（1）满足维护要求表中提出的要求所需要的工作量；

（2）维护要求的性质；

（3）这项要求的优先次序；

（4）修改有关的事后数据。

3．工作流程

整个维护活动与新软件开发的设计、编码和测试各步是平行的，不过由于时间上的限制，可能省略或简化某些步骤。用户的维护要求和软件开发者的修改报告被批准后的工作流程如图 8-3 所示。

当然，有的申请的处理过程并不完全符合图 8-3 所示的事件流。例如出现紧急软件

问题时，就出现所谓“救火”维护要求。在这种情况下，就需要立即投入人力进行抢救。

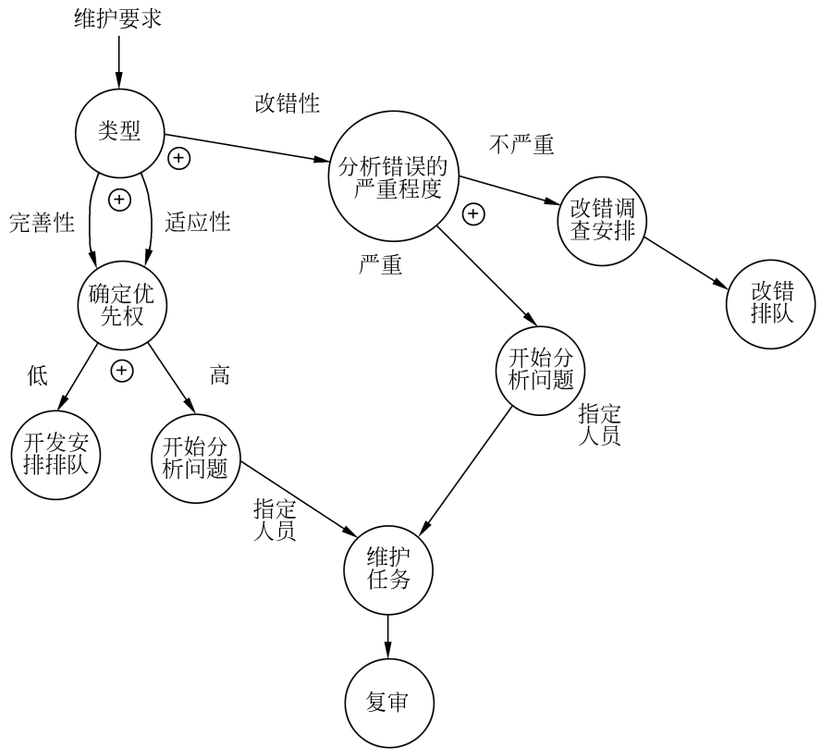


图 8-3　软件维护工作流程图

完成软件维护后，可进行一次情况复审，可为提高软件组织的管理效能提供重要意见。

4．维护记录的保存

对于软件生命周期的所有阶段而言，以前记录保存都是不充分的，而软件维护则是根本没有记录保存下来。由于这个原因，人们往往不能估计维护技术的有效性，不能确定一个没有产品程序的“优良”程度，而且很难确定维护的实际代价是什么。

保存维护记录遇到的第一个问题就是，哪些数据是值得记录的？Swanson 提出了下述内容：

（1）程序标识；　　（2）源语句数；
（3）机器指令条数；　　（4）使用的程序设计语言；
（5）程序安装的日期；　　（6）自从安装以来程序运行的次数；
（7）自从安装以来程序失效的次数；　　（8）程序变动的层次和标识；
（9）因程序变动而增加的源语句数；　　（10）因程序变动而删除的源语句数；
（11）每个改动耗费的人时数；　　（12）程序改动的日期；
（13）软件工程师的名字；　　（14）维护要求表的标识；
（15）维护类型；　　（16）维护开始和完成的日期；
（17）累计用于维护的人时数；　　（18）与完成的维护相联系的纯效益。

应该为每项维护工作都收集上述数据。可以利用这些数据构成一个维护数据库的基础，并且像下一小节介绍的那样对这些数据进行评价。

5．对维护的评价

缺乏有效的数据就无法评价维护活动。如果已经开始保存维护记录了，则可以对维护工作做一些定量度量。至少可以从下述 7 个方面度量维护工作：

（1）每次程序运行平均失效的次数；

（2）用于每一类维护活动的总人时数；

（3）平均每个程序、每种语言、每种维护类型所做的程序变动数；

（4）维护过程中增加或删除一个源语句平均花费的人时数；

（5）维护每种语言平均花费的人时数；

（6）一张维护要求表的平均周转时间；

（7）不同维护类型所占的百分比。

根据对维护工作定量度量的结果，可以做出关于开发技术、语言选择、维护工作量规划、资源分配及其他许多方面的决定，而且可以利用这样的数据去分析评价维护任务。

8.3 软件的可维护性

8.3.1 软件可维护性的定义

软件可维护性即维护人员对该软件进行维护的难易程度，具体包括理解、改正、改动和改进该软件的难易程度。 提高可维护性是指导软件工程方法所有步骤的基本准则，也是软件工程追求的主要目标之一。

仅就程序自身来考虑，影响软件维护难易程度的因素如下：

（1）系统的大小。系统越大，维护的困难也越大。

（2）系统的年龄。一般来讲，老程序比新程序需要更多的维护工作，有些老程序未使用模块化和结构设计技术，因而就更难维护。

（3）结构合理性。程序的结构若不合理，对其维护难度就较大。结构的合理性主要是以以下几点为基础的：模块化、层次组织、系统文档的结构，命令的格式和约定、程序的复杂性等等。

其他影响维护难易程度的因素还有：应用的类型、程序设计的语言、使用的数据库技术、开关与标号的数量、IF 语句的嵌套层次、索引或下标变量的数量等等。

除程序的自身因素外，文档是影响软件可维护性的决定因素。清晰完整的文档甚至比程序代码更重要，只有和代码完全一致的文档才是真正有价值的文档。

8.3.2 软件可维护性的度量及评估

人们一直期望对软件的可维护性做出定量度量，但要做到这一点并不容易。常用的

度量方法是质量检查表、质量测试、质量标准。

质量检查表是用于测试程序中某些质量特性是否存在的一个问题清单。评价者针对检查表上的每一个问题，依据自己的定性判断，回答 Yes 或者 No。

质量测试与质量标准则用于定量分析和评价程序的质量。

由于许多质量特性是相互抵触的，要考虑几种不同的度量标准，相应地去度量不同的质量特性。

1．可理解性

可理解性表明人们通过阅读源代码和相关文档，了解程序功能及其如何运行的容易程度。一个可理解的程序应具备以下一些特性：模块化，风格一致性，不使用令人捉摸不定或含糊不清的代码，使用有意义的数据名和过程名，结构化，完整性等。

2．可靠性

可靠性表明一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。关于可靠性，度量的标准主要有：

① 平均失效间隔时间 *MTTF*

② 平均修复时间 *MTTR*

③ 有效性 $A = MTBD/(MTBD+MDT)$

3．可测试性

可测试性表明论证程序正确性的容易程度。程序越简单，证明其正确性就越容易。而且设计合用的测试用例，取决于对程序的全面理解。一个可测试的程序应当是可理解的、可靠的、简单的。用于可测试性度量的检查项目如下：

① 程序是否模块化？结构是否良好？

② 程序是否可理解？程序是否可靠？

③ 程序是否能显示任意中间结果？

④ 程序是否能以清楚的方式描述输出？

⑤ 程序是否能及时地按照要求显示所有的输入？

⑥ 程序是否有跟踪及显示逻辑控制流程的能力？

⑦ 程序是否能从检查点再启动？

⑧ 程序是否能显示带说明的错误信息？

4．可修改性

可修改性表明程序容易修改的程度。一个可修改的程序应当是可理解的、通用的、灵活的、简单的。通用性是指程序适用于各种功能变化而无需修改。灵活性是指能够容易地对程序进行修改。聚合性高、耦合性低、信息局部化、接口简单的程序结构通常具有良好的可修改性。

5．可移植性

可移植性表明程序转移到一个新的计算环境的可能性的大小。或者表明程序可以容易地、有效地在各种各样的计算环境中运行的难易程度。一个可移植的程序应具有结构良好、灵活、不依赖于某一具体计算机或操作系统的性能。用于可移植性度量的检查项目如下：

① 是否是用高级的独立于机器的语言来编写程序？

② 是否使用广泛使用的标准化的程序设计语言来编写程序？是否仅使用了这种语

言的标准版本和特性?

③ 程序中是否使用了标准的普遍使用的库功能和子程序?

④ 程序中是否极少使用或根本不使用操作系统的功能?

⑤ 程序在执行之前是否初始化内存?

⑥ 程序在执行之前是否测定当前的输入／输出设备?

⑦ 程序是否把与机器相关的语句分离了出来，集中放在了一些单独的程序模块中，并有说明文件?

⑧ 程序是否结构化? 并允许在小一些的计算机上分段（覆盖）运行?

⑨ 程序中是否避免了依赖于字母数字或特殊字符的内部表示?

6．效率

效率表明一个程序能执行预定功能而又不浪费机器资源的程度。这些机器资源包括内存容量、外存容量、通道容量和执行时间。用于效率度量的检查项目如下：

① 程序是否模块化? 结构是否良好?

② 是否消除了无用的标号与表达式，以充分发挥编译器优化作用?

③ 程序的编译器是否有优化功能?

④ 是否把特殊子程序和错误处理子程序都归入了单独的模块中?

⑤ 是否以快速的数学运算代替了较慢的数学运算?

⑥ 是否尽可能地使用了整数运算，而不是实数运算?

⑦ 是否在表达式中避免了混合数据类型的使用，消除了不必要的类型转换?

⑧ 程序是否避免了非标准的函数或子程序的调用?

⑨ 在几条分支结构中，是否最有可能为“真”的分支首先得到测试?

⑩ 在复杂的逻辑条件中，是否最有可能为“真”的表达式首先得到测试?

7．可使用性

从用户观点出发，可使用性定义为程序方便、实用及易于使用的程度。一个可使用的程序应是易于使用的、能允许用户出错和改变，并尽可能不使用户陷入混乱状态的程序。用于可使用性度量的检查项目如下：

① 程序是否具有自描述性?

② 程序是否能始终如一地按照用户的要求运行?

③ 程序是否让用户对数据处理有一个满意的和适当的控制?

④ 程序是否容易学会使用?

⑤ 程序是否使用数据管理系统来自动地处理事务性工作和管理格式化、地址分配及存储器组织?

⑥ 程序是否具有容错性?

⑦ 程序是否灵活?

⑧ 其他间接定量度量可维护性的方法；

⑨ 问题识别的时间；

⑩ 因管理活动拖延的时间；

祐 收集维护工具的时间；

焆 分析、诊断问题的时间；

焘 修改规格说明的时间；

骼 具体的改错或修改的时间；

旸 局部测试的时间；

诺 集成或回归测试的时间；

瞭 维护的评审时间。

这些数据反映了维护全过程中检错——纠错——验证的周期，即从检测出软件存在的问题开始至修正错误并经回归测试验证这段时间。可以粗略地认为，这个周期越短，维护越容易。

8.3.3　提高软件可维护性的方法

1．建立明确的软件质量目标和优先级

一个可维护的程序应是可理解的、可靠的、可测试的、可修改的、可移植的、效率高的、可使用的。要实现这所有的目标，需要付出很大的代价，而且也不一定行得通。某些质量特性是相互促进的，例如可理解性和可测试性、可理解性和可修改性。另一些质量特性是相互抵触的，如效率和可移植性、效率和可修改性等。每一种质量特性的相对重要性应随程序的用途及计算环境的不同而不同。例如，对编译程序来说，可能强调效率；但对管理信息系统来说，则可能强调可使用性和可修改性。应当对程序的质量特性，在提出目标的同时还必须规定目标的优先级。

2．使用提高软件质量的技术和工具

1）模块化

如果需要改变某个模块的功能，则只要改变这个模块，对其他模块影响很小；如果需要增加程序的某些功能，则仅需增加完成这些功能的新的模块或模块层；程序的测试与重复测试比较容易；程序错误易于定位和纠正。

2）结构化程序设计

程序被划分成分层的模块结构；模块调用控制必须从模块的入口点进入，从其出口点退出。模块的控制结构仅限于顺序、选择、重复 3 种，且没有 goto 语句。每个程序变量只用于惟一的程序目的，而且变量的作用范围应是明确的、有限制的。

使用结构化程序设计技术，提高现有系统的可维护性，有以下几种方法。

（1）采用备用件的方法——用一个新的结构良好的模块替换掉整个要修改的模块。

（2）采用自动重建结构和重新格式化的工具（结构更新技术）——把非结构化代码转换成良好结构代码。

（3）改进现有程序的不完善的文档——建立或补充系统说明书、设计文档、模块说明书以及在源程序中插入必要的注释。

3．进行明确的质量保证审查

质量保证审查对于获得和维持软件的质量，是一个很有用的技术。审查可以用来检测在开发和维护阶段内发生的质量变化。一旦检测出问题来，就可以采取措施来纠正，以控制不断增长的软件维护成本，延长软件系统的有效生命期。

① 在检查点进行复审。

② 验收检查。

③ 周期性地维护审查。

4．对软件包进行检查

选择可维护的程序设计语言，程序设计语言的选择，对程序的可维护性影响很大。

改进程序的文档，程序文档是对程序总目标、程序各组成部分之间的关系、程序设计策略、程序实现过程的历史数据等的说明和补充。即使是一个十分简单的程序，要想有效地、高效率地维护这个程序，也需要编制文档来解释其目的及任务。对于程序维护人员来说，要想按程序编制人员的意图重新改造程序，并对今后变化的可能性进行估计，缺了文档是不行的。因此，为了维护程序，人们必须阅读和理解文档。

另外，在软件维护阶段，利用历史文档，可以大大简化维护工作。通过了解原设计思想，可以判断出错之处，指导维护人员选择适当的方法修改代码而不危及系统的完整性。 历史文档有 3 种：系统开发日志；错误记载；系统维护日志。

本 章 小 结

维护是软件生命周期的最后一个阶段，也是持续时间最长代价最大的一个阶段。软件工程学的主要目的就是提高软件的可维护性，降低维护的代价。需要掌握软件维护的内容、特点、文档、维护技术、维护任务的实施、维护的副作用，理解可维护性的含义及提高可维护性的方法和技术。

软件维护工作一般发生在客户使用软件产品的过程中，通常分为如下几类：适应性维护、完善性维护、纠错性维护。整个维护过程需要建立一套体系，即维护管理体系。对于产品的维护不能随便进行，要求建立申请——判断——实施——检查的过程，也就是任何维护工作都要向有关部门申请，相关部门判断决定是否实施及实施优先级，维护完成后还需要检查确认，上述维护工作都要有记录。

习 题 8

8.1 请讨论使维护成本居高不下的因素。如何尽可能降低这些因素的影响？

8.2 什么是软件的可维护性？软件的可维护性与哪些软件质量的特性有关？如何提高软件的可维护性？为什么在软件开发过程中，要特别重视软件的可维护性？

8.3 请挑选一种读者熟悉的已经改进过多个版本的软件包，列出在不同版本中所实现的一些修改。什么因素会影响在较早的版本上实现这些特定功能？

8.4 理解程序为什么很重要？当试图理解程序时，读者要达到什么目标？

8.5 假设读者作为一名程序员，被分配的工作是：(i) 为一个正在运行的管理信息系统（MIS）提供一种消息标题；(ii) 把这个 MIS 集成到另一个办公自动化软件包中。为了实现这些更改，需要了解 MIS 的什么信息？请说明自己的理由。

8.6 软件的可维护性与哪些因素有关？在软件开发过程中应该采取哪些措施才能提高软件产品的可维护性？

8.7 假设你的任务是对一个已有的软件做重大修改，而且只允许从下述文档中选取两份：(a) 程序的规格说明；(b) 程序的详细设计结果（自然语言描述加上某种设计工具表示）；(c) 源程序清单（其中有适当数量的注解）。

你将选取哪两份文档？为什么这样选取？你打算怎样完成任务？

第9章

软件项目计划与管理[1, 27, 36]

软件项目管理是为完成项目既定目标而对软件任务进行组织、计划、实施、管理、评估等活动的一次性过程。在实施此过程中必须具有范围、时间、成本等方面的约束限制。

软件项目管理的提出起源于20世纪70年代中期的美国，当时美国国防部专门研究了软件开发中不能按时交付，预算超支和质量达不到用户要求的原因，结果发现70%的项目是因为管理不善引起的，而非技术问题。由此软件研发者们开始逐渐重视起软件开发过程中的各项管理。到了20世纪90年代中期，软件项目管理虽然得到了较大的改善，但其中的诸多问题依然存在，造成软件研发项目是否能够在预定的费用和进度下如期按质量交付的状况仍然难以预测，这也使得软件项目管理成为现今软件开发中的热门课题。

1996年美国项目管理学会（PMI）颁布了针对所有产品生产过程的项目管理知识体系（PMBOK），定义项目管理为“将各种知识、工具、技能应用于项目工作，以达到或超过项目干系方对项目的要求和期望”。该学会还从8个知识范畴描述了项目管理知识体系的应用。这8个范畴分别是：范围管理、时间管理、成本管理、质量管理、人力资源管理、交流沟通管理、风险管理和采购与分包管理。

软件是一种信息产品，当然也需要对软件的开发生产过程进行有效的项目管理。但软件是逻辑的而不是物理的产品，具有信息业管理的特点：动态、灵活、不确定、不可简单重复。所以，软件项目管理和其他的项目管理相比有其相当的特殊性。首先，软件是纯知识产品，不是制造出来的产品，即软件项目开发的终端产品只是程序代码和技术文档，其开发进度和质量很难估计和度量，生产效率也难以预测和保证。其次，软件系统的复杂性也导致了开发过程中各种风险的难以预见和不易控制。例如Windows 2000有5000多万行代码，3000多位工程师，几百个小团队，同时有数千个程序员在进行开发，项目经理多达上百个，从开发到推出历时长达4年。这样庞大的系统如果没有很好的管理，其软件质量是难以想象的。这好比软件项目是一台交响乐，虽说每个乐师都能独立地演奏出动听的音乐，但一群好的乐师凑在

一起就不见得能奏出雄壮恢宏的交响曲，因为还必须拥有一位好的指挥家。软件开发正是如此，有好的程序员只是前提条件，要开发出好的软件，还要有一个好的项目管理。

正因如此，软件项目管理不但需要，而且这种管理还涉及相当广泛的范围，包括范围管理、成本管理、人员管理、进度管理、风险管理、质量管理、标准化管理等方面的内容。而管理的对象更是揽扩了范围与计划、工作量与成本、人员与配置、风险与质量、标准与文档等领域。软件项目管理的目标是确保软件开发能按时间、按预算、按质量地完成任务，这不仅需要制订完善的项目计划，还必须具有对项目能否按计划执行的跟踪与动态调整；不仅需要遵循质量标准规范，还必须通过严格的评审来把握开发进度和控制风险；不仅需要合理地调配人力、物力、财力等资源，还必须在各环节管理中提交相关文档和阶段成果。

由此可见，软件项目管理是一个多目标、多任务的复杂系统工程，没有一个周密的规划和严格的控制是难以有效实施的。因此，在实施软件项目之前必须首先制定完善的软件项目计划，为软件项目的展开提供工作的范围、经历的里程碑、可能遇到的风险、需要的资源（人、硬／软件）、要完成的任务、所花费的成本（工作量），以及进度的安排等方面的信息，并以此为据对项目实施监控和管理。按常规，这些针对软件项目开发的管理型活动必须在技术工作开始之前就已经进行，并且需要在软件从概念到实现的过程中一直持续地实施，而只有当软件开发工作最后结束时才告终止。因此，软件项目管理所涉及的范围和时限应该说覆盖了整个软件生命周期，是一系列自始至终伴随项目开发进行的必要活动。

正因为软件项目管理是对软件开发与生产过程进行管理的一系列关键活动，所以涵盖项目策划过程、组织实施过程、跟踪监控过程和质量保证过程等方面的管理。主要的工作任务包括以下几个方面：软件项目的估算与计划、参与人员的组织及管理、风险控制与跟踪评审、质量保证及软件配置管理、标准文档编制等。

为有效地实施一个软件项目的管理，可将管理过程分为 5 组：启动过程组、计划过程组、执行过程组、控制过程组、结束过程组。每组又有一个或多个子过程组成。具体应该完成以下主要职能。

（1）启动软件项目：确定项目的目标和范围，界定系统的功能和任务。拟订战略方案、进行资源测算、提出建立项目组方案。

（2）制定项目计划：明确任务要求、规划实施细则，估算并安排资源、人力、进度、成本等要素，编制并提交相关文档评审。

（3）执行项目过程：建立分工明确的责任机构、以及资源配备、人员任用，并协调各种层次的人力、物力、财力。采取必要措施强化执行计划、实现既定目标。

（4）监控项目实施：监督和检测软件过程的实施，提供开发进度的报告。进行工作指导，激励开发人员按期完成所分配的任务。并分析规避各类风险，动态监理确保质量。

（5）阶段结束评审：对照项目计划或检查标准评审计划的完成程度。全面务实地评价当前项目进展状况的好坏。取得项目或阶段的正式认可，并且有序地结束该阶段或整个项目。提交软件产品或成果，同时提供最直接、最有效的参考与备案标准文档。

软件项目管理的对象是软件工程项目。软件项目管理就是对软件工程项目开发过程

的管理。具体地说，就是对整个软件生存期的一切活动进行管理，以达到提高生产率、改善产品质量、按期完成任务的目的。

9.1　成 本 估 计

软件项目管理过程始于项目计划。在做项目计划时，第一项活动就是对成本、人员、进度、资源的估计。这些估计应当在软件项目开始时的一个有限时间段内做出，并且随着项目的进展定期地进行控制和调整。

随着计算机科技的迅猛发展，软件成本已经成为计算机应用系统中开销最大的部分，从而使人们愈来愈重视软件开发成本的估算，这也就成为了制定软件项目计划之前首要解决的问题，倘若估算结果出现一个较大的偏差那将会造成营利及亏损间的巨大差异，而费用的超支对于开发者来说无疑是难以接受的。

通常，在进行软件项目开发之前，软件开发者和软件用户应该一起对项目的目标和范围进行明确的定义和规划。虽然目标说明只给出了软件项目的总体目标，并不考虑这些目标究竟是如何实现，但范围说明却给出了与问题相关的主要数据、功能和行为，通过范围说明可以量化一些重要的约束特性。当明确了项目的目标和范围时，即可以开始充分考虑待选的解决方案。尽管这一步并不讨论细节，但使得管理者和开发者可以选择一条“最好的”途径，并根据软件产品交付的期限、预算的限制、可用的人员、技术的接口以及各种其他因素定义技术和管理的有限约束，以获取项目相关的必要信息。因为没有这些信息，就不可能进行合理的、准确的成本估算，进而给出具有意义明确的且带有项目进度标志的项目管理计划。

由于项目进度计划只是从时间的角度对项目进行了部署，那么成本估算则必须从费用的角度对项目进行规划。因此，成本估算是对完成软件项目所需费用的估计和计划，是软件项目计划中的一个重要组成部分。项目管理者要实行成本控制、风险规避，首先就要进行成本估算。通常理想的情况是，根据已经完成的某项任务所具有的历史标准来估算现行费用。但对于大多数应用项目来说，由于其要求不同、目标各异、变化多端、情况复杂，想要把以前的活动与现实的情况对比起来衡量几乎是不可能的。因此，关于软件项目费用的信息，不管是否根据历史标准数据，还是现时重做都只能将其视为一种估算。

为了可靠地实施项目成本及工作量的估算，可以考虑以下几种选择。

（1）将估算推延到项目的最后阶段进行——虽然在项目完成之后能够得到精确的花费值，但这是不现实的，因为项目要求成本估算必须预先提出。

（2）基于已经完成的类似项目进行估算——“借鉴”可以有据可依，但不幸的是，过去的经验并不总是未来结果的指示器，使得借用过去难以成行。

（3）使用简单的“分解技术”进行估算——采用“分而治之”的策略将项目分解成若干个主要的功能任务及相关的软件工程活动，通过化整为零、逐步求精的方式进行估算。

（4）使用一个或多个经验模型进行估算——经验估算模型可用于补充分解技术，并提供一种潜在的有价值的估算方法。

每一种估算方案的实施都与用于估算的软件成本参量有关，而这些参数又大多取决

于历史数据和相似性类比，如果失去有效的和可靠的历史数据和衡量参照系，那么软件成本估算也就建立在一个很不稳定的基础之上了。纵观许多项目失败的真正原因，主要是没有很好的项目管理，其中也就包括成本失控。

因此，成本估算将要为软件项目提供重要的量化约束条件，这正体现了估算工作的意义和重要。所以软件项目的成本估算应该注意以下 3 个原则：

（1）真实性与预见性原则——能够真实地反映客观的需要，又能对未来的需求有所预见；

（2）透明性与适应性原则——提供的参算因素准确可靠，推出的处理算法恰当适用；

（3）方便性与稳定性原则——估算过程简单明确，处理步骤条理清晰，各项指标相对独立。

由于成本估算本身的预计性，那么这种预计就存在着一定的风险。所以软件项目成本测算的风险因素也是应该在估算时加以考虑的：

（1）对目标系统的功能需要、开发人员、开发环境等情况的了解的正确性如何；

（2）所运用历史数据及模型参数的可靠性如何；

（3）系统分析中逻辑模型的抽象程度、业务处理流程的复杂程度及软件的可度量程度；

（4）软件新技术、替代技术的出现及应用会对成本测算方法的冲击及影响；

（5）用户的参与程度，开发人员的素质以及所采用开发模式对开发成本的影响；

（6）对软件开发队伍人员组成比、复杂因素及其稳定性的认识程度；

（7）软件开发和维护经费，时间要求等方面的变更等非技术性因素所带来的风险等。

总之，在软件项目管理过程中的关键活动之一就是制定项目计划，在制定计划之前就必须对项目各任务所需多少工作量、人力资源（以人月为单位）、项目持续时间（以年份或月份为单位）、软件过程成本（以元为单位）做出近似估算，而有效的项目成本估算和预算是分配资源的参考，也是控制的标准。由于费用的分配和安排是与项目进度计划相适应的，所以项目成本估算在项目管理中具有重要的地位和作用。

在软件项目管理中，项目成本估算的结果一般以书面报告或报表的形式表述出来，并提交相关部门评审。

9.1.1 项目成本估计的基本要素与模式

进行软件项目成本估算，首先应该划分涉及成本估算的各种要素，并确定各要素之间的联系以及相互的影响，以便应用合适的算法及有效的工具实施计算处理。

1．软件项目成本

软件的开发成本是以一次性开发过程所花费的代价来计算的，将随着系统的类型、范围及功能的不同而有较大差异。从软件系统生命周期构成的两大阶段，即开发阶段和维护阶段来看，软件项目的成本可划分为：开发生产成本和运行维护成本两大类。其中开发成本由软件开发费用、硬件配置费用和其他费用组成；而运行维护成本则由培训费用、运行/维护费用、管理费用等费用构成。

1）开发生产成本

（1）分析设计费用：来自于系统调研、系统规划、需求分析、系统分析、系统设计诸方面的费用——属于系统设计开发成本。

（2）系统实施费用：来自编程与调试开销、硬件平台购置与安装、网络施工投入、系统软件或工具软件引进、数据整理录入及消耗等费用——属于系统平台成本。

2）运行维护成本

（1）专业培训费用：人员培训费、技术资料获取等——属于培训成本。

（2）系统运行费用：消耗材料费、固定资产折旧、人工费等——属于运行成本。

（3）维护改进费用：硬件维护费、软件纠错性/适应性维护费、二次开发费用等——属于维护成本。

（4）行政管理费用：办公费、系统服务费、行政管理费等——属于管理成本。

从管理学的角度来看，软件项目的成本作为一个经济学范畴，也应能反映软件产品在其生产过程中所耗费的各项费用，主要为：人工费、管理费用、财务费用等各项开支的总和。而软件成本按产生和存在的不同形式还可分成固定成本、可变成本、半变动成本、直接成本、间接成本（分摊成本）和项目总成本（又称为全寿命周期成本）。

以财务管理科目细分，列入软件项目的成本构成成分主要有如下的条目：

1）直接材料成本

（1）硬件购置费：如计算机及相关设备的购置，不间断电源、空调器等的购置费。

（2）软件购置费：如操作系统软件、数据库系统软件和其他应用软件的购置费。

2）直接劳动力成本

人工费：主要是开发人员、操作人员、管理人员的工资福利费等。

3）项目的实施费用成本

① 培训费：主要是资料编制、打印复印、参考书籍购买费用等。

② 通信费：如购置计算机网络设备、通信线路器材、租用公用通信线路等的费用。

4）其他直接成本

① 基本建设费：如新建、扩建机房、购置终端桌、文件柜等的费用。

② 财务费用。

③ 管理费用：如办公费、通信费、差旅费、会议费、交通费等。

④ 材料费：如打印纸、色带、墨粉、磁盘等耗材的购置费。

5）间接成本

① 水电、燃气费。

② 专有技术购置费：如专利技术引进、系统或工具软件购买等费用。

③ 其他费用：如资料费、固定资产折旧费及咨询费等。

2．软件项目成本基本因素

软件项目的成本体现在软件产品上其实包含了技术和管理两方面的支出。从技术的角度分析，在实际的软件开发过程中，可能影响软件成本模型计算结果的各种成本因素主要分为 6 类。

1）系统规模类的成本因素

包括：程序指令的估算条数、交付的机器指令数、交付的源语言代码行数、新指令与旧指令的百分比、生成与编写指令的百分比、判定指令的数目、非判定指令的数目、

信息存储和检索指令的百分比、交付代码的百分比及其他关于系统规模的因素。

2）数据库类的成本因素

包括：数据库中记录数及数据量、存储过程数目、触发器数目及其他关于数据库的因素。

3）系统复杂性类的成本因素

包括：整个复杂性的级别、接口的复杂性、系统的惟一性、系统开发难度、硬件-软件接口、程序的结构、文件报告和应用程序的数目、生存期人力总数、开发期人力总数、测试和验证期人力总数、生存期总时间、开发期总时间、作业类型及其他关于系统复杂性的因素。

4）软件开发类的成本因素

包括：面向问题分析法、面向功能的软件开发方法、面向数据流的软件开发方法、面向数据结构方法与结构化开发方法、面向过程方法、面向对象方法、可视化开发方法及其他关于程序开发类的因素。

5）编写文档类的成本因素

包括：文档类别、文档数量、文档发布及其他关于文档类的因素。

6）环境与项目属性类的成本因素

包括：硬件配置状况、网络运行环境、专用设备购置、配套软件外购、相关技术资料价格、各种通信交流支出、不可预见成本及其他关于项目环境属性类的因素。

3．软件项目成本估算

从各方面的分析来看，涉及并影响软件成本的要素不仅繁多而且还比较复杂。这也预示着软件估算本身具有的风险,完成这项工作不仅需要丰富的经验和有用的历史信息，而且要有足够的定量数据和作出定量度量的把握，所以在实施估算时需要十分的谨慎和周密。一般情况，估算的精确程度还可能受到以下因素的影响。

（1）项目的复杂性。增加了软件计划的不确定性。复杂性越高，估算的风险也就越高。复杂性是相对度量的，与项目参加人员的经验和能力有关。

（2）项目的规模。随着软件规模的扩大，软件相同元素之间的相互依赖、相互影响也迅速增加，造成估算时进行问题分解变得更加困难。

（3）项目的结构化程度。这里的结构是指功能分解的简便性和处理信息的层次性，结构化程度提高，进行精确估算的能动性就提高，相应风险将会减少。

（4）历史信息的有效性。借鉴过去项目的历史数据进行综合性的软件度量，可以用来类比分析，所以其准确有效是估算的保证。

（5）其他。用户需求的不确定性、以及频繁的变更计划等都会给估算带来非常大的影响，这也是要加以注意的。

在实际工作中影响软件项目估算精度的因素是比较复杂的，管理者即要根据具体情况具体分析，还要实行必要的成本控制。只有详细地划分并有效地把握与成本相关的要素才能够使管理者做到心中有数、实施有据、协调有效、控制有力，使软件项目的成本估算处于合理可信的基础之上。

软件项目成本估算是对完成项目各项任务所需要的各种资源成本的近似估算，目的是估计项目的总成本和误差范围。在实际工作中通常采用下面 3 种项目成本估算模式。

1）自上而下估算（经验估算法、类比估算法）

多在有类似项目已完成或根据总成本要求需要进行分解估算的情况下应用。此方法利用以前类似项目的实际成本或历史数据作为基本依据，通过经验判断得出项目整体成本和各个子任务的成本预算。此方法一般在项目的初期或信息不足时进行，需要具有较高的分析水平和实际经验。

2）自下而上估算（分解估算法、周期估算法）

可借助工作任务分解结构 WBS（work breakdown structure）法将项目任务分解到最小单位工作包，通过对项目工作包进行详细的成本估算，然后通过各个成本汇总将结果累加起来得出项目总成本。换句话说，这种技术通常首先估计出各个独立工作的费用，然后再将其汇总，从下往上、从部分到全局估计出整个项目的总费用。另一种做法是将整个系统按生命周期分解为若干阶段并估算，然后汇总出总工作量和成本。

由于该方法可使项目相关人员都参与项目的预算，考虑的综合因素较多，所以此方法比较准确。但代价是耗费的管理成本也会因涉及的人员较多而相应增加。另外，缺少各项子任务之间相互联系所需的工作量，也缺少许多与软件开发有关的系统级即全局所需的工作量，所以，此法必须考虑系统集成时所需的工作量及其成本。

3）参数模型估算

这是一种统计建模技术，如回归分析和学习曲线。将项目的特征参数作为预测项目费用数学模型的基本参数。此方法需要数据的积累，根据同类项目的管理状况和成本数据，建立模型，在遇到同类项目时可以直接套用。如果模型是依赖于历史信息，且模型参数容易数量化，当模型应用仅与项目范围的大小相关时，则此套用一般是可靠的。

进行项目成本估算处理可使用一种或多种技术，这些技术主要分为两大类：分解和经验建模。分解技术对项目按系统、子系统或者生命周期分解，分别估算出各个子系统或子任务的成本，然后再把这些成本汇总，估算出整个项目的成本。经验技术则根据以往经验导出的公式来预测工作量和时间。另外，也可使用自动工具辅助实现某一特定的经验模型。如项目管理软件及电子表格软件都可以辅助项目费用的估计。

9.1.2　软件开发成本估算的常用方法

软件开发成本估算主要针对软件产品开发过程所花费的工作量及其相应的代价。这不同于其他物理产品的成本，主要体现在项目干系人的劳动消耗所需代价方面，这种代价就是软件产品的开发成本。另一方面，软件产品开发成本的计算方法也不同于其他物理产品成本的计算。软件产品不存在重复制造过程，其开发成本是以一次性开发过程所支出的代价来计算的。因此软件开发成本的估算，应该以项目计划、需求分析、软件设计、编码实现、调试测试这整个软件开发过程所有花费的代价作为依据。

解决软件开发成本和工作量的估算主要是依靠软件规模的度量。通常，规模度量方法有代码行（LOC）方法和功能点（FP）方法以及专家判定（Deiphi）方法。

1．软件代码行（line of code，LOC）方法

用代码行数表示软件的开发规模显得十分自然、直观。这样不仅能度量软件的规模，而且可以度量软件开发的生产率、每行代码的平均成本、千行代码出错率等。

软件开发的生产率：　　$P_l = L/E$

其中，L 为软件总代码行数；E 为软件的总工作量，用人／月（PM）度量；P_l 为开发生产率，每人月完成代码行数（LOC／PM）。

每行代码的平均成本：　　$C_l = S/L$

其中，S 为开发的总成本，用人民币元度量；L 为软件总代码行数；C_l 为每行代码平均成本，人民币元／代码行。

软件代码行方法的优点为：

- 用软件代码行估算软件的开发规模直观、简单易行。

软件代码行方法的缺点为：

- 代码行数估算依赖于开发工具的功能和表达能力；
- 项目开发初期估算代码行数十分困难；
- 用代码行估算方法会对需要设计精巧的项目产生不利影响；
- 代码行估算不适宜非过程式的程序设计（如面向对象程序设计）。

2．软件功能点（function points，FP）方法

功能点度量是一种涉及多种因素（输入、输出、数据文件、查询和外部接口等）的针对软件规模和软件生产率进行间接度量的方法。此方法的关注点在于程序的“功能性”和“实用性”，而不是直接针对软件代码行计数。通过利用软件信息域中的一些计数度量以及软件复杂性估计的经验关系式从而导出功能点 FP，再由一系列分析比对求取软件规模进而估算出开发成本。

该方法通过填写 CT 度量表得到 5 个信息量的“加权值”结果，而后合计得出 CT 信息量，并用 14 个复杂性调节值 $F_i(i=1,2,\cdots,14)$ 修正功能点值。这样在系统分析的初期就能估算出软件开发的规模。

首先，要确定 5 个信息域的特征，并在 CT 的度量表格中相应位置给出计数值。其信息域的计数值可由以下分析定义给出。

（1）用户输入数：计算每个用户的输入。这些输入向软件提供面向应用的数据。输入应该与查询区分开来，分别计算。

（2）用户输出数：计算每个用户的输出。这些输出向用户提供面向应用的信息。这里，输出是指报表、屏幕、出错信息等。但一个报表中的单个数据项不单独计算。

（3）用户查询数：查询是一种联机输入，导致软件以联机输出的方式生成某种即时的响应。故每一个不同的查询都要计数。

（4）文件数：每一个逻辑主文件也应计数。所谓的逻辑主文件，是指逻辑上的一组数据，可以是一个大的数据库的一部分，也可以是一个单独的文件。

（5）外部界面数：对所有被用来将信息传送到另一个系统中的机器可读写的接口（即磁带或磁盘上的数据文件）均应计数。

一旦收集到上述数据，就可以计算出与每一个计数项相关的复杂性数值，并可由此展开相关计算。使用功能点方法还可以自行拟定一些准则以确定一个特定项的加权值是最佳的、可能的还是悲观的，以保证结果更加合理与可信。

使用如下的关系式计算功能点 FP：

$$FP = CT \times [0.65 + 0.01 \times SUM(F_i)] \quad i=1, 2, \cdots, 14$$

其中，CT 为信息量，是由 CT 度量表中所得到的全部加权计数项的合计（见表 9-1）。

F_i（i = 1~14）为影响因子，通过逐一回答 Fi 定值表中所提问题来确定；F_i 取值为 6 级：当 F_i＝0 时，表示否定或不起作用；F_i＝5 时，表示肯定或 F_i 影响最大。由此计算出复杂性校正值 [0.65 ＋ 0.01 × SUM（F_i）]（见表 9-2）。

SUM（F_i）为求和函数 $\sum$；$\sum F_i$ 为复杂度，是 F_i (i=1, 2,⋯, 14) 的累加和。

求解 FP 等式中的常数项和应用于信息域计数的加权因子均可根据经验予以确定。

表 9-1　CT 度量表

测 量 参 数	值	运算符	加权值	等　号	加 权 结 果
用户输入数	（　）	×	4	＝	（　）
用户输出数	（　）	×	5	＝	（　）
用户查询数	（　）	×	4	＝	（　）
文件数	（　）	×	10	＝	（　）
外部界面数	（　）	×	7	＝	（　）
合计：CT				＝	（　）

表 9-2　F_i 定值表

评定每个提问的校正因素尺度（F_i）是 0～5		
	0　1　2　3　4　5	
	没有影响　偶然的　适中的　普通的　重要的　极重要的	
i	所提问题	F_i 取值
1	系统需要可靠的备份和恢复吗？	0～5
2	系统需要数据通信吗？	
3	系统需要有分布式处理的功能吗？	
4	性能是关键（临界状态）的吗？	
5	系统是否将运行在现有的高度实用化的操作环境中？	
6	系统要求联机数据项吗？	
7	联机数据项是否要求建立在多重窗口显示或操作上的输入事务？	
8	是否联机地更新主文件？	
9	输入、输出、文件、查询复杂吗？	
10	内部处理过程复杂吗？	
11	程序代码是否要设计成可复用的？	
12	设计中是否包含变换和安装？	
13	系统是否要设计成多种安装形式以安装在不同的机构中？	
14	应用系统是否要设计成便于修改和易于用户使用？	

一旦计算出功能点 FP，就可以仿照 LOC 的方式度量软件的生产率、质量和其他属性：

生产率＝FP／PM（人月）
成本＝元／FP
质量＝错误数／FP
文档＝文档页数／FP

利用功能点方法也可以定义软件的开发效率、成本等度量：

软件开发的生产率：

$$Pf = FP/E$$

其中，Pf 表示每人月完成的功能点数；E 为人数。

每功能点的平均开发成本：

$$Cf = S/Pf$$

其中，Cf 为每功能点的平均成本；S 为开发总成本。

【例 9-1】 考察一个为计算机辅助设计（CAD）应用而开发的软件包。在这个实例中，使用 *FP* 作为估算变量。因为基于 FP 估算的分解是集中于信息域值，而不是软件功能。所以根据实际情况，项目计划者可预先估算出 CAD 软件的输入、输出、查询、文件及外部接口，给出 CT 度量表，并将计算出的相关估算结果存于表 9-3 中。

表 9-3 估算信息域值

信 息 域	信息域值			估算计数（平均值）	加权因子	CT 计数
	最佳值	可能值	悲观值			
输入数	20	24	30	24	4	96
输出数	12	15	22	16	5	80
查询数	16	22	28	22	4	88
文件数	4	4	5	4	10	40
外部接口数	2	2	3	2	7	14
总计数值：						318

接着，根据提问方式评价出 14 个复杂度加权因子的取值，并计算复杂度调整因子，表 9-4 给出了各因子的具体取值情况：

表 9-4 计算复杂度调整因子

因子 i	所 提 问 题	F_i 取值
1	备份和复原	4
2	数据通信	2
3	分布式处理	0
4	关键性能	4
5	现有的操作环境	3
6	联机数据登录	4
7	多屏幕输入切换	5
8	信息域值复杂度	5
9	输入、输出、文件、查询复杂度	3
10	内部处理复杂度	5
11	设计成可复用的代码	4
12	设计中的转换及安装	3
13	多次安装	5
14	方便修改的应用设计	5
$\sum F_i$	=	52

由表中数据计算得出复杂度调整因子：$(0.65 + 0.01 \times \sum F_i) = 1.17$

最后，求出 FP 的估算值：

FP 估算值=CT 总计数值 × 复杂度调整因子

$= 318 \times 1.17 = 372$

基于过去已完成类似项目的平均 *FP* 生产率大概是 6.5FP/ PM，如果一个劳动力价格是每月 8000 元，则每个 FP 的成本（=8000/6.5）约为 1230 元。根据 FP 估算及历史生产率数据，总的项目成本估算（=1230×372）是 457560 元，工作量估算（=372/6.5）约为 58 个人月。

采用功能点度量软件开发规模有如下的优点：

- 与程序设计工具无关，适用于各种语言环境，对于面向对象的开发方式尤为有用；
- 由于项目启动时就能基本上确定系统的输入、输出等参数，所以功能点度量适合用于开发初期阶段对软件开发成本进行估算。

采用功能点度量也存在一些缺点：

- 该方法涉及到的主观因素比较多，如 *Fi* 的选取与评估人的经验和态度有较大的关系。
- 计算公式中的 *FP* 值的物理意义不直观，不易理解。

3．专家判定（Deiphi）技术

Rand 公司提出的 Deiphi 技术是一种专家判定技术的实际运用。就是组织多位专家进行项目的成本估算，但为了克服由于某位专家可能会产生的某些偏见，通常情况，会请多位专家参与评估，由此获得多个估算值，而后求平均值或加权平均值，作为统一专家意见的方法。采用 Deiphi 技术一般能得到较为准确的估算值。其具体步骤是：

① 首先，由组织者给每位专家发一份软件系统的规格说明书（隐去名称和单位）以及一张记录估算值的表格，请专家为该软件进行评估。

② 接着，专家们详细研究软件规格说明书的内容，对该软件提出 3 个规模的估算值，即：a_i 为该软件可能的最小规模（最少源代码行数）；m_i 为该软件最可能的规模（最可能的源代码行数）；b_i 为该软件可能的最大规模（最多源代码行数）。

采用无记名方式填写表格，并说明做此估算的理由。在填表的过程中，专家互相不进行讨论，但可以向组织者咨询相关问题。

③ 组织者对专家们填在表格中的答案进行分类整理，同时完成以下处理：

- 计算每位专家（序号为 i，$i=1,2,\cdots,n$，共 n 位专家）的估算期望值 E_i（加权平均），同时求出估算值的期望平均值 E：

$$E_i = \frac{a_i + 4m_i + b_i}{6} \qquad E = \frac{1}{n}\sum_{i=1}^{n} E_i$$

- 对专家的估算结果进行分类摘要。

④ 在综合专家估算结果的基础上，再次组织专家无记名地填写表格。然后比较两次估算的结果。若差异很大，则要通过分析找出差异的原因。

⑤ 上述过程可重复多次。最终可获得一个让多数专家共识的软件规模（源代码

行数）。

最后，将结果与历史资料进行类比，根据过去完成软件项目的规模和成本等信息，推算出该软件每行源代码所需要的成本。然后再乘以该软件源代码行数的估算值，就可得到该软件开发成本的估算值。

此方法虽然准确，但缺点是人们无法利用其他参加者的估算值来调整自己的估算值。宽带 Deiphi 技术克服了这个缺点。在专家正式将估算值填入表格之前，由组织者召集小组会议，专家们与组织者一起对估算问题进行讨论，然后专家们再无记名填表。组织者对各位专家在表中填写的估算值进行综合和分类后，再召集会议，请专家们对其估算值有很大变动之处进行讨论，请专家们重新无记名填表。这样适当重复几次，由于增加了协商的机会，集思广益，使得估算值更趋于合理。

9.1.3 软件成本估算的经验模型

软件开发成本估算的公式化方法是依据开发成本估算模型进行计算的。而开发成本估算模型通常采用经验公式来预测软件项目计划所需要的成本、工作量和进度数据。由于用以支持大多数估算模型的经验数据都是从有限的一些历史项目样本中得到的。所以并没有一种估算模型能够适用于所有的软件类型和开发环境，那么从这些模型中得到的结果虽然具有一定的参考价值，但都带有局限性，应该慎重使用。

1．IBM 模型

1977 年由 IBM 的 Walston 和 Felix 提出的基于源代码行数的软件规模估算公式：

$$E=5.2\times L^{0.91}$$
$$D=4.1\times L^{0.36}=14.47\times E^{0.35}$$
$$S=0.54\times E^{0.6}$$
$$\mathrm{DOC}=49\times L^{1.01}$$

其中，L 为源代码行数（KLOC）；E 为工作量（PM）；D 为项目持续时间（月）；S 为人员需要量（人）；DOC 为文档数量（页）。

在此模型中，一般指一条机器指令为一行源代码。一个软件的源代码行数不包括程序注释、作业命令、调试程序在内。对于非机器指令编写的源程序，例如汇编语言或高级语言程序，应按一定的比例转换成机器指令源代码行数来计算。

IBM 模型只是一个静态单变量模型的经验公式，在应用中并不能作为一个通用的公式，有时需要根据实际具体情况对公式中的参数进行修改。但这种修改必须拥有足够的历史数据，在明确局部的环境定义之后才能做出。

2．Putnam 模型

1978 年由 Putnam 提出的适合于大型软件的估算模型。这是一种可用于软件开发的各个阶段的动态多变量的模型。该模型假定工作量在软件开发的整个生存期中服从特定的分布，并以各个阶段的实测数据为基础，画出大型软件项目开发工作量的 Rayleigh-Norden 分布曲线（图 9-1），曲线所反映的各个阶段的工作量直观明了，便于分析以估算出成本。虽然这种模型是依据在一些大型项目（总工作量达到或超过 30 人年）中收集到

的工作量分布情况而推导出来的，但也可以应用于一些中小型的软件项目中。

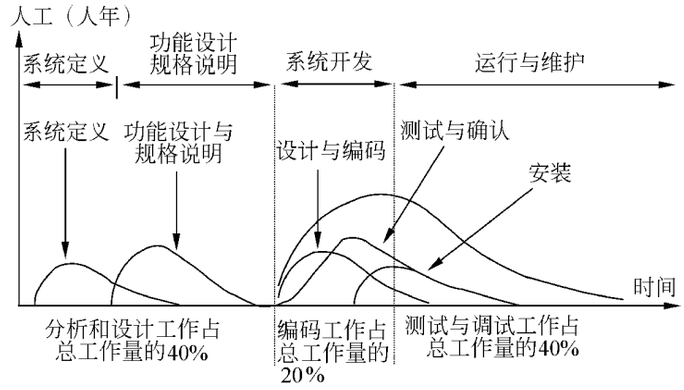


图 9-1　大型软件项目工作量分布示意图

用 Rayleigh-Norden 曲线可以导出一个软件方程式，将已交付的源代码（源语句）行数通过工作量和持续开发时间构成函数关系如下：

$$L=\mathrm{Ck}K^{1/3}\,\mathrm{td}^{4/3}$$

其中，L 为源代码行数（LOC）；K 为软件开发与维护在内的整个生存期所花费的工作量（人年）；td 为开发持续时间（年）；Ck 为技术状态常数，反映出“妨碍程序员进展的限制”，且因开发环境而异。

典型的技术状态常数 Ck 值的选取如表 9-5 所示：

表 9-5　技术状态常数 Ck 取值定义

Ck 的典型值	开发环境	开发环境说明
2000	差	没有系统的开发方法，缺乏文档和复审，批处理方式
8000	好	有合适的系统开发方法，有充分的文档和复审，有交互执行方式
11000	优	有自动开发工具和技术

由上面的方程可推导得到如下估算项目总开发成本的公式：

$$K=L^3/(\mathrm{Ck}^3\,\mathrm{td}^4)$$

$$C=\lambda K$$

其中，C 为项目开发总成本；λ 为每人年的人力成本（一般为人员工资、补助等）。

3．COCOMO 模型（constructive cost model）

1981 年由 TRW 公司开发、BarryBoehm 提出的“构造性成本模型（constructive cost model）”是一种比较精确的软件估算模型的层次体系。

COCOMO 模型可以在静态、单变量模型的基础上进行构造，按其详细程度分成 3 个层次：用于开发初期估算整个系统的工作量和时间的基本 COCOMO 模型。用于估算各个子系统的工作量和开发时间的中间 COCOMO 模型，用于估算相对独立的软部件（模块）的详细 COCOMO 模型。

基本 COCOMO 模型是一个静态单变量模型，用一个以已估算出来的源代码行数

（LOC）为自变量的（经验）函数来计算软件开发工作量。而中间 COCOMO 模型则是在基本 COCOMO 模型的基础上，将已经计算出来的软件开发工作量视为名义工作量，再通过某些影响因素来调整工作量的估算。更进一步就要采用详细 COCOMO 模型了，同时包括中间 COCOMO 模型的所有特性。

鉴于软件项目的分类，故在 COCOMO 模型应用中，同时考虑软件开发环境相关的因素。故此把软件开发项目从总体上分为 3 种类型：组织型（organic）、嵌入型（embadded）和介于上述两类软件之间的半独立型（semidetached）。

在 COCOMO 模型中为方便工作量和进度的计算还引入以下几个基本量：

- DSI（源指令条数）定义为代码或卡片形式的源程序行数。若一行有两条语句，则算做一条指令。作业控制语句和格式语句作为有效语句，但注释语句除开。定义：1KDSI＝1000DSI。
- MM（度量单位为人月）表示开发工作量。定义：1MM=1/12 人年。
- TDEV（度量单位为月）表示开发进度。由工作量决定。

以下针对 3 个层次的 COCOMO 模型计算方法分别进行详细讨论。

1）基本 COCOMO 模型

基本 COCOMO 模型是静态、单变量模型，具有下列形式：

开发工作量：　　$MM = a(KDSI)^b$

开发进度：　　$TDEV = c(MM)^d$

开发成本：　　$C = \lambda E$

其中，KDSI 为项目的源代码行估计值，单位是千行代码（KLOC）；MM 为工作量，单位是人月（PM）；TDEV 为开发时间，单位为月（M）；C 为开发成本，单位是万元；λ 为每人月的人力成本，单位是万元/人月；a、b、c、d 为针对不同软件类型的常数，不同软件类型的 a、b、c、d 取值如表 9-6 所示。

表 9-6　基本 COCOMO 模型 a、b、c、d 常数取值

软件类型	a	b	c	d	适 用 范 围
组织型	2.4	1.05	2.5	0.38	各类应用程序
半独立型	3.0	1.12	2.5	0.35	各类实用程序、编译程序等
嵌入型	3.6	1.20	2.5	0.32	实时处理、控制程序、操作系统

基本 COCOMO 模型的工作量和进度公式具体表述如表 9-7 所示。

表 9-7　基本 COCOMO 模型的工作量和进度公式

软件类型	工 作 量	进　　度
组 织 型	$MM = 2.4\ (KDSI)^{1.05}$	$TDEV = 2.5\ (MM)^{0.38}$
半独立型	$MM = 3.0\ (KDSI)^{1.12}$	$TDEV = 2.5\ (MM)^{0.35}$
嵌 入 型	$MM = 3.6\ (KDSI)^{1.20}$	$TDEV = 2.5\ (MM)^{0.32}$

利用表 9-7 中所给出的公式，可求得软件项目的开发工作量和开发进度，或分阶段求得各软件任务的开发工作量和开发进度。

2）中间 COCOMO 模型

在基本 COCOMO 模型的基础上，将已经计算出来的软件开发工作量视为名义工作量，再用一组“成本驱动因子属性”扩展基本模型。这些成本驱动因子共有 15 个属性，且被划分为 4 个主要类型：产品属性、硬性属性、人员属性及项目属性，可在 6 个级别上取值，根据重要性或价值从“非常低”到“超高”（见表 9-9），通过评估 15 种影响软件工作量的因素，定下某些成本驱动因子的级别，修正 COCOMO 工作量公式和进度公式，便可更合理地调整软件（各阶段）的工作量和进度的估算。

中间 COCOMO 模型的名义工作量与进度公式如表 9-8 所示。

表 9-8　中间 COCOMO 模型的名义工作量与进度公式

软件类型	工 作 量	进　　度
组织型	$MM=3.2\ (KDSI)^{1.05}$	$TDEV=2.5\ (MM)^{0.38}$
半独立型	$MM=3.0\ (KDSI)^{1.12}$	$TDEV=2.5\ (MM)^{0.35}$
嵌入型	$MM=2.8\ (KDSI)^{1.20}$	$TDEV=2.5\ (MM)^{0.32}$

对 15 种影响软件工作量的因素 f_i 按等级打分，如表 9-9 所列。同时，修正工作量计算公式为（单位是人月 PM）：

$$MM = r \times \prod_{i=1}^{15} f_i \times (KDSI)^c$$

其中，r 为由基本 COCOMO 模型计算出来的名义工作量；f_i 为打分确定的影响软件工作量的因素等级分数；$\prod f_i$ 为修正因子；KDSI 为源指令条数（单位千行）；c 为根据软件类型确定工作量运算的幂。

表 9-9　15 种影响软件工作量的因素 f_i 的等级打分表

工作量因素 f_i		非 常 低	低	正　常	高	非 常 高	超　高
产品因素	软件可靠性	0.75	0.88	1.00	1.15	1.40	
	数据库规模		0.94	1.00	1.08	1.16	
	产品复杂性	0.70	0.85	1.00	1.15	1.30	1.65
硬件因素	执行时间限制			1.00	1.11	1.30	1.66
	存储限制			1.00	1.06	1.21	1.56
	虚拟机易变性		0.87	1.00	1.15	1.30	
	环境周转时间		0.87	1.00	1.07	1.15	
人的因素	分析员能力		1.46	1.00	0.86	0.71	
	应用论域实际经验	1.29	1.13	1.00	0.91	0.82	
	程序员能力	1.42	1.17	1.00	0.86	0.70	
	虚拟机使用经验	1.21	1.10	1.00	0.90		
	程序语言使用经验	1.41	1.07	1.00	0.95		
项目因素	现代程序设计技术	1.24	1.10	1.00	0.91	0.82	
	软件工具的使用	1.24	1.10	1.00	0.91	0.83	
	开发进度限制	1.23	1.08	1.00	1.04	1.10	

【例 9-2】　考察一个微机远程通信的嵌入型软件，其规模为 10KDSI，使用中间 COCOMO 模型进行软件成本估算。求程序的名义工作量 MM、程序的实际工作量、开发所用的时

间 TDEV 各为多少？如果软件开发人员的工资都按每月 5000 元计算，则该软件项目开发人员的工资总额是多少？

解：先将规模 KDSI=10 带入表 9-8 中嵌入型软件计算公式得出名义工作量值。

程序名义工作量值： $MM = 2.8 \times (10)^{1.20} = 44.38$（MM）

再考虑 15 种影响软件工作量的因素，并根据表 9-9 评估出 15 种影响软件工作量的因素 f_i 的等级分取值，结果见表 9-10。通过选定的乘法因子，修正 COCOMO 工作量公式和进度公式，可更合理地估算软件（各阶段）的工作量和进度。此时，实际工作量计算公式修正成：

$$MM = r \times \prod_{i=1}^{15} f_i \times (KDSI)^c$$

程序实际工作量： $MM = 44.38 \times \prod_{i=1}^{15} f_i = 44.38 \times 1.17 = 51.5$ (MM)

开发所用时间： $TDEV = 2.5 \times (51.5)^{0.32} = 8.9$（月）

如果开发人员的工资都按每月 5000 元计算，则该项目的工资总额为实际开发工作量（人月）与月工资之积：

$51.5 \times 5000 = 257500$（元）

表 9-10 影响工作量的因素 f_i 的等级打分取值

影响工作量因素 f_i	情 况 说 明	取 值
1 软件可靠性	仅用于局部地区，恢复问题不严重	1.00（正常）
2 数据库规模	20KB	0.94（低）
3 产品复杂性	用于远程通信处理	1.30（很高）
4 时间限制	使用 70%的 CPU 时间	1.10（高）
5 存储限制	使用 71%的存储空间	1.06（高）
6 机器	使用商用微处理机	1.00（额定值）
7 周转时间	平均 2 小时	1.00（额定值）
8 分析员能力	本岗位工作 5 年	0.86（高）
9 工作经验	远程通信工作 3 年	1.10（低）
10 程序员能力	参与项目开发 8 个	0.86（高）
11 工作经验	微型机工作 6 个月	1.00（正常）
12 语言使用经验	12 个月	1.00（正常）
13 使用现代程序设计技术	1 年以上	0.91（高）
14 使用软件工具	基本的微型机软件	1.10（低）
15 工期	9 个月	1.00（正常）
修正因子		$\prod f_i = 1.17$

3）详细 COCOMO 模型

详细 COCOMO 模型的工作量因素分级表（类似于 15 种影响软件工作量因素 f_i 的等级打分表）是按分层、分阶段给出的。而名义工作量公式和进度公式与中间 COCOMO

模型相同。对应每一个影响因素，按划分的层次：模块层、子系统层、系统层，将有 3 张不同的工作量因素分级表，以提供给不同层次的估算使用。每一张表中工作量因素又按开发中各个不同阶段列出。如表 9-11 所示是一个关于软件可靠性（RELY）要求的工作量因素分级表（仅子系统层）。使用这些表格，可以比中间 COCOMO 模型更准确地估算软件开发工作量。

表 9-11　软件可靠性工作量因素分级表（子系统层）

阶段 / RELY 级别	需求和产品设计	详细设计	编码及单元测试	集成及测试	综　合
非常低	0.80	0.80	0.80	0.60	0.75
低	0.90	0.90	0.90	0.80	0.88
正常	1.00	1.00	1.00	1.00	1.00
高	1.10	1.10	1.10	1.30	1.15
非常高	1.30	1.30	1.30	1.70	1.40

4．自动估算工具

目前已有一些自动估算的工具软件实现了以上介绍的经验估算模型。项目管理者能够方便地使用这些工具自动估算出项目的成本和工作量，还可对人员配置和交付日期等科目进行估计。在使用这些工具时通常需要项目管理者提供以下数据：

（1）定量估算软件项目的规模数据，如：总代码行数或功能点数；

（2）定性地说明项目特性，如复杂性、可靠性或事件的关键性等；

（3）开发人员和（或）开发环境的描述。

根据这些数据，自动估算工具即可提供诸如：项目所需的工作量、成本、人员配备、开发进度和相应风险等数据。

目前几种具有代表性的自动估算工具如表 9-12 所示。

表 9-12　自动估算工具

工具软件名称	软件制作公司	工具软件基于原理
BYL	Gordon 集团	基于 COCOMO 模型
WICOMO	Wang 研究所	基于 COCOMO 模型
DECPlan	DEC 公司	基于 COCOMO 模型
SLIM	…	基于 Putnam 模型
SPQR/20	…	基于功能点模型
ESTIMACS	…	基于功能点模型

9.2　效 益 分 析

效益包括有形的经济效益，也包括无形的社会效益。软件系统的经济效益等于因使用新软件而增加的收益加上使用新软件可以节省的运行费用（运行费用包括操作员人数、工作时间、消耗的物资等）。在计算软件系统的经济效益时，应按照货币的时间价值来计算，这是因为对软件项目的投资在前，而软件系统效益的产生在后，这其中有一个延续

较长时间的过程。

针对成本-效益的分析是先将软件项目投资中可能发生的所有成本与效益归纳起来，再利用数量分析方法来计算成本和效益的比值，从而以经济角度去评价开发一个新的软件项目是否可行、是否盈利。通常利用净效益和效益成本比率来分析计算项目投资的经济效益。在成本效益分析中，效益和成本可在现值基础上进行分析对比，也可在等年值的基础上进行分析对比。

9.2.1 几种效益度量方法

1．货币的时间价值

成本估算的目的是要对项目的投资作出评价。但事实是：投资在前，取得效益在后。因此必须考虑货币的时间价值。通常用利率表示货币的时间价值，即货币随着时间的推移可以通过利息来获得增值。假设银行年利率为 i，现在存入 P 元钱，则 n 年后可得钱数为：$F=P(1+i)^n$ 元，这就是 P 元钱不拿去投资而是存于银行，n 年后也能获得的价值。反过来说，若想 n 年后能收入 F 元，那么这些钱现在的价值是：$P=F(1+i)^{-n}$，即现在存入银行 P 元即可在 n 年后收入 F 元。这个公式说明了货币的时间价值，即在有利率的情况下现在的钱是随时间增值的。

2．投资回收期

投资回收期是衡量一个开发项目价值的经济指标。所谓投资回收期就是使累计的经济效益等于最初的投资额所需要的时间。投资回收期越短，就能越快地获得利润，这项开发项目也就越值得投资。

3．纯收入

项目的纯收入是衡量项目工程价值的另一项经济指标。所谓纯收入就是在整个生存期之内系统的累计经济效益（折合成现在值）与投资额之差。这相当于将钱投资到一个待开发的软件项目预期可取得的效益和把钱存在银行里（或贷款给其他企业）到期所取得的收益进行比较，衡量是否合算。如果纯收入为零，则项目的预期效益与在银行存款一样。但投资有风险，存银行没风险，仅从经济角度来看，这个项目可能是不值得投资的。如果纯收入小于零，那么显然这个项目不值得投资。只有当纯收入大于零，才能考虑是否投资。

4．投资回报率

把钱存在银行里，可以用年利率来衡量利息的多少。类似地，也可用投资回报率来衡量投资效益的大小。已知现在的投资额为 P，并且已经估算出将来每年可以获得的经济效益为 F_k，以及软件的使用寿命为 n，$k=1,2,\cdots,n$。则投资回报率 j（相当于利率），可用如下的方程来计算：

$$P=\frac{F_1}{(1+j)^1}+\frac{F_2}{(1+j)^2}+\frac{F_3}{(1+j)^3}+\cdots+\frac{F_n}{(1+j)^n}$$

这相当于现在把数额等于投资额的资金存入银行，而每年从银行取回的利息钱就等于软件每年预期可以获得的效益。当时间等于软件寿命年限时，正好把在存入银行中的钱全部取光。而此时的年利率就等于投资回报率。若此方法获得的回报率大于实际银行

利率，那么此投资是值得考虑的。

【例 9-3】　考察开发某个计算机应用系统的投资额为 3000 元，该系统投入使用后，每年可以节约 1000 元，5 年内可能节约 5000 元。3000 元是现在投资的钱，5000 元是 5 年内节省的钱，两者是不能简单直接判别的。

假定年利率为 12%，利用计算货币现值的公式：$F = P(1+i)^n$，可以算出该系统投入使用后每年预计节省的金额的现在价值（见表 9-13）。

表 9-13　计算机系统每年节省金额的现在价值

年	节省 F（元）	利率 0.12	现在价值 P（元）	按年累计现在价值（元）
1	1000	1.12	892.86	892.86
2	1000	1.25	800.00	1692.86
3	1000	1.40	714.29	2407.15
4	1000	1.57	636.94	3044.09
5	1000	1.76	568.18	3612.27

则求：该系统的纯收入、投资回收期和投资回收率。

分析：纯收入就是在整个生存期之内系统的累计经济效益（折合成现在值）与投资之差。

投资回收期就是使累计的经济效益等于最初的投资额所需的时间。

投资回收率是投入资金所获得的利率。

解：

该计算机应用系统在 5 年中的纯收入为：3612.27–3000 = 612.27（元）。

投资回收期约为：3 +(3000–2407.15)/(3044.09–2407.15) ≈ 3.93 (年)。

投资回收率设为 r，由下列方程式：

$$3000 = 1000/(1+r)+1000/(1+r)^2+1000/(1+r)^3+\cdots+1000/(1+r)^5$$

解得投资回收率为：$r \approx 20\%$。

9.2.2　效益分析方法

效益分析是在估算新软件系统的投资成本基础上，再与可能取得的效益（有形的和无形的）进行比较权衡、分析考察而确定项目的经济利益是否盈亏的方法。相对有形的效益可以用货币的时间价值、投资回收期、纯收入、投资回报率及投入产出比等指标进行度量。而无形的效益一般很难直接进行度量，主要是从性质上、心理上、社会效益上进行衡量。但在某些时候或某些情形下无形的效益也会转化成有形的效益。例如，一个设计先进、稳定可靠的软件能够使用户使用满意，信誉很好，从而影响到其他潜在的用户也对该软件充满好感，一旦需要时就会选择购买该软件，这样就使得无形的心理效益转化成有形的经济效益。

软件系统的效益分析将随其系统的特性而异。例如，大多数数据处理系统的基本目标是开发具有较大的信息容量、更高的质量、更快捷的检索、结构更好的系统。因此，效益集中体现在信息存取和对用户交互环境的影响上面。而对于工程科学计算软件，其

相关的效益在本质上就不同于数据处理系统，更多地体现在对新产品开发或新技术设计的模型支持，并从中获得间接的经济效益。

概括地说，一个软件系统的经济效益应该是使用新软件系统而增加的收入加上使用新软件系统可以节省的费用之和（其中包括人数、工时、耗材等方面的费用节余）。那么进行效益分析可以从下面几方面入手。

1．基于投资收益的分析

在成本效益分析中，效益和成本可以在货币现值基础上进行对比，也可在货币的等年值的基础上进行对比。如果效益和成本在现值基础上对比，则项目整个经济寿命期的效益（包括直接效益和间接效益）和总成本（包括建设期的投资支出和经济寿命期的经营成本）之间的衡量，可用下列两种方式来表示：

（1）现值净效益＝现值总效益－现值总成本

（2）效益成本比率＝现值总效益÷现值总成本

上列公式中：（1）净效益是将效益、成本相减的结果，是绝对投资经济效益指标。当效益大于成本出现净效益（正值）时，表示该投资有经济效益。（2）效益成本比率是效益与成本相比的比率，是个相对投资经济效益指标。在有净效益的情况下，效益成本比率一定大于 1，表示该投资有经济效益。效益成本比率越大，表示投资经济效益越好。

如果效益和成本是在等年值基础上进行对比，则用年效益与等额年成本进行衡量，可用下列方式来表示：

（1）年净效益＝年效益－等额年成本

（2）效益成本比率＝年效益÷等额年成本

上列公式中：等额年成本=等额年投资成本+年经营成本

其中：

等额年投资成本＝（投资支出－回收资产余值×现值系数）× 资金回收系数

分析：在年效益大于等额年成本时出现年净效益（正值），则效益成本率大于 1，表示有投资经济效益。若年净效益为负值，效益成本率小于 1,则表示没有投资效益，甚至是赔本的项目，此时结论就是：从经济的角度衡量，此项目不可投资。

2．基于投资回报率的分析

任何项目的开发都是投资在前，收益在后。倘若未来的投资回报率低于银行利率时，投资就失去了经济意义，也就没什么投资经济效益可谈。另外，投资回报率是有时间要素的，在时间等于软件寿命时，若还不能获取相当于银行存款所得收益，那么同样没有投资的必要。只有在投资回报率大于银行利率的情况下，实施该项目的投资才给予考虑。若能排除项目开发中的各种风险或降低风险至最小，才可以切实投资。当然，投资总是带有一定风险的，就看如何把握、如何运作。从某种意义上说，效益就蕴藏在有效的项目管理之中。

实际上，对于投资回报率也可在软件产品的开发和存在过程中获得增加，主要可以从 3 个方面入手：一是降低成本，节约开支与增加软件的可重用度等措施，都可以达到降低成本的目的；二是利用现有资产来增加软件产品效益（如改善销售业绩）；三是综合以上两项，进行全面减负增效。

3．基于盈亏平衡点的分析

对于软件产品项目，由于有前期的投入，一般需要进行盈亏平衡分析。这是一种根据项目正常生产年份的产品产量（销售量）、固定成本、可变成本、税金等，来研究开发项目的产量、成本、利润之间变化与平衡关系的方法。软件项目也可将每年收益、开发费用与运行年数之间的变化关系套用盈亏平衡点的分析，当项目的收益与成本相等时，即为盈亏平衡点（BEP），此时收支相抵，小于平衡点的年份里属于无盈利，而大于平衡点的年份即有利可图。通过盈亏平衡分析可以展现软件产品项目开发的获利分布情况。

首先对项目中每一项可能发生的费用进行估算，然后用开发费用和运行费用来确定投资的盈亏平衡点和投资回收期。假设新软件每年可节约的总费用的估计值为 96000 元（即每年收益），已支出总开发（或购买）费用为 204000 元，且每年度费用（运行费）估计为 32000 元。根据公式：

年增成本＝总开发费用＋年运行费用×年数

年增效益＝年效益×年数

盈亏平衡点：年增效益＝年增成本（联立求解上两式）

投资回收期：盈亏平衡点所对应的年份

带入以上具体参数到公式中即可计算出投资回收期大约需要 3.2 年（见图 9-2）。实际上，投资的回收期可以用更详细的分析方法来确定，即货币的时间价值、税收的影响及其他潜在的对投资的扰动因素。同时还可把无形的效益也考虑在内进行计算分析，再根据其可能获得的经济效益判断在经济上是否值得开发该软件系统。

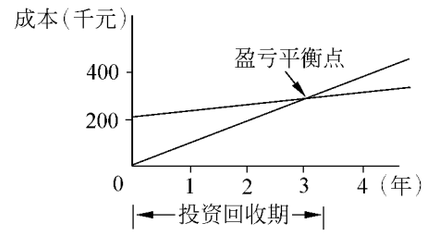


图 9-2　盈亏平衡点例

9.3　项目组织与计划

软件项目组织开启了软件项目从萌动构思到具体实现的大门。随之而来的便是关于项目计划的制定、人员的落实、资源的配置、监控的实施等一系列项目管理过程。

项目启动，计划先行。为了保证软件项目开发成功，必须对软件开发项目的工作范围、可能遇到的风险、需要的资源（人员、硬件、软件）、要实现的任务、经历的里程碑、花费的工作量（成本），以及进度的安排都做到心中有数、执行有序、沟通有谱、检查有据。制定项目计划正是为了达到这样一种目的。

项目计划的制定提供了一系列指导项目实施和跟踪控制的纲领性文件。因为，项目计划是来自高层管理机构批准的关于项目实施的正式文档，所以，项目计划既是对项目参与人员实行目标管理和目标考核的依据，也是软件过程控制的依据。项目计划应该是一种面向大众的相对简短的文档，必须能够做到：①在软件管理者、技术人员和客户之间传达项目范围和资源信息；②定义风险并提出有关风险管理技术的建议；③定义管理复审的成本和进度；④给项目相关人员提供软件开发的整体方法；⑤概述如何保证质量及变更的管理。

计划不是拿来展示的，而是要用来执行的。项目计划必须具有较强的可操作性，达到一定的执行度。如何将工作任务划分出一个个相对独立的可执行单元正是实施计划首先应解决的问题。常用的任务划分方法有：

- 任务分解的层次结构——按项目的实际情况进行自顶向下的结构化分解，形成树型工作任务结构。进一步把各节点的工作内容、所需工作量、预计进度期限逐一规定落实下来。可借助工作分解结构（WBS）的方法进行。
- 任务落实的责任矩阵——在任务分解的基础上，把工作分配给相关的人员。用一个矩阵形表格标定任务的分工和责任。行为工作项目，列为实施人员，对应相交的结点即为某人的具体工作任务。
- 任务阶段的细分部署——首先按照软件生存期，把开发过程划分为若干个工作阶段，并对每一阶段做出工作计划。再把每一阶段的工作分解为若干任务，由此做出任务计划。再把任务细分为若干步骤，最终做出步骤计划。然后落实具体实施。

落实和实施计划的关键因素是人，所以软件项目的组织涉及到人力资源的调配。由于软件项目可以是一个单独的开发项目，也可以与产品项目一同组成一个完整的软件产品项目。如果是单独的开发，则成立软件项目组即可；如果是产品项目开发，则需成立软件项目组（负责软件的设计与开发）和产品项目组（负责市场调研和销售）一同合作组成软件产品项目组。但无论什么组织形式，项目中的人员管理都是主要的管理受体，在制定项目计划和实施项目计划时必须认真考虑和安排。

9.3.1 项目计划的制定

在管理学中，计划，也称为规划，其定义是“组织确定目标、实现目标的战略与手段、步骤、程序的过程”。制定了计划，就有了目标，有了条理，有了把握，有了依据。管理活动就能适时开始、顺利进行。

项目计划是项目管理的第一步，能够展示从软件构思成为软件产品的整个实施过程。项目计划的制定直接关系到项目的好坏与成败。因此，项目计划的合理性与完整性是确保项目能够按计划执行的重要因素。再者，项目计划不仅具有不同的层次和结构，而且具有一定的广泛性和专业性，所以项目计划应该凝结整个项目组成员思想和智慧的结晶。

制定软件项目计划的目标就是要确定：软件开发项目的工作范围是什么、必须部署哪些资源、应该投入多少工作量、需要花费多少成本、以及如何规划进度安排、能否规避风险损害等一系列问题。

对于一个大型的、复杂的软件项目，任务之间的依赖关系可以使用任务网络来定义。任务，有时也称作项目的“工作分解结构（work breakdown structure，WBS）”，其定义可以是针对整个软件产品，也可以是针对某单个功能进行。采用该方法可以由顶向下将一个软件项目的任务分解为易于操作和管理的相对独立的几部分或几个细目，细目再展开形成子细目，任何分支最低层的细目称作工作包。待开发的软件系统可以先按生命周期的阶段展开，然后按照子系统或系统功能点展开，当然也可以直接按各个子系统展开（注意此时不能遗漏整体集成的工作任务）。从而得到层次分明、相对独立的单个工作

集，所有这些工作集就构成了项目的整个范围。

工作分解结构（WBS）随着项目规模的差异所起的作用不尽相同。小的项目只需要很简单的 WBS，层次的划分基本上是一目了然。但软件项目规模越大，WBS 也就越重要，从另外一个角度来讲也就越难做好。对大型项目而言，确定项目的 WBS 往往不可一蹴而就，需要经过多次反馈、修正，最后才能得到一个项目各方都能接受的 WBS。

有了经过 WBS 分解的各项独立任务，即可开始编制项目计划。从形式上看，项目计划实质就是项目管理的设计书。考察一个项目的管理是否完备有序，首先应该从项目计划开始判断。再者，软件过程的里程碑能否正常到达，项目计划也是最直接、最明确的考核标准。显然，项目计划应该具有以下的效能。

（1）能够使项目组和有关管理人员对项目的有关事项，如资源配备、人员安排、时间进度、内外接口、风险化解等达到共识，形成事先约定，做到人人心中有数，事事有章可循。

（2）能够使一些支持性工作以及并行工作得到及时安排，避免因计划不周而造成各子进程之间相互牵掣，影响全局。做到未雨绸缪，防患于未然。

（3）能够使项目相关人员明确自己的职责，便于自我管理、自我监督和自我激励。

（4）能够有效地支持项目管理工作，使项目负责人对开发工作的跟踪和检查有据可依。

（5）能够完善事先准备，使注意力专心于解决问题，而不会分心于下一步该做些什么。

（6）能够提供项目总结评价的依据，对比判断项目实施情况，获取有用的管理信息，通过不断检查计划与运行，从中提炼经验和教训。

项目计划的制定需要经过深入的调查分析，全面的综合考虑并运用相应的处理技术，并且反复修改多次才能得以定稿，而后还必须通过严格的评审，切不可敷衍了事。尤其在制定项目时需要把握准确相关的内容和涉及的任务，通常应该包含以下几个方面。

（1）估算所需要的人力（通常以人月为单位）、项目持续时间（以年份或月份为单位）、成本（以元为单位）。

（2）作出进度安排，资源分配，建立项目组织及任用人员（包括人员的地位、作用职责、规章制度、签订合同等），根据规模和工作量估算分配任务。

（3）进行风险分析，包括风险识别、风险评估、风险优化、风险驾驭策略、风险规避手段和风险监控措施等。而这些步骤将贯穿在整个软件工程过程中。

（4）制定质量管理指标，确定如何识别已定义的任务、如何掌握结束时间和结束标准、如何识别和监控关键路径以确保结束正常、如何度量开发进程与控制质量、以及如何建立分隔任务的里程碑等。

（5）编制项目预算和成本估算、费用支出报告。

（6）准备运行环境和基础设施，配置管理规划等。

项目计划是项目管理工作的中心内容。针对不同工作目标和不同的时间进度，可以有不同类型的软件项目计划，而每一种项目计划都是为完成一个项目管理工作而安排的具体内容，实际编制时应该分门别类：

- 项目实施计划（软件开发计划）——这是软件开发的综合性计划，通常应包括

任务、进度、人力、环境、资源、组织等多个方面管理的内容；

- 质量保证计划——把软件开发的质量要求具体规定为每个开发阶段可以检查的质量保证活动；
- 配置管理计划——说明为支持本项目的开发所需要的各种条件和设施；
- 软件测试计划——规定测试活动的任务、测试方法、测试数据、测试标准、以及测试进度、资源、人员职责等；
- 文档编制计划——规定所开发项目应编制的文档种类、内容范围、以及编制进度、人员职责等；
- 用户培训计划——规定对用户进行培训的目标、达到的要求、以及培训进度、人员职责等；
- 综合支持计划——规定软件开发过程中所需要的支持（硬件与软件、技术与资料、用户与外援等），以及如何获取和利用这些支持；
- 软件发布计划——软件开发项目完成后，如何分发及提供给用户，或如何作市场宣传。

另外，项目开发中还需制订各个相关专题计划，如分合同计划、开发人员培训计划、安全保密计划、系统安装计划等。

总之，编制项目计划的过程就是用文件的形式，把软件项目管理者的管理意识和工作方案表述出来，把软件项目开发过程中各项工作的参与人员、开发进度、经费预算、软硬件条件、风险管理等条目所作的安排记载下来，最终形成《项目开发计划》（参见 GB/T 8567-1988 计算机软件产品开发文件编制指南），以便根据此计划开展和检查该项目的开发实施工作。

9.3.2 项目组人员配备规则

所有软件项目中最关键的因素是人员。合理地配备人员并充分发挥其工作效率是成功地完成软件项目的切实保证。由于软件开发的各个阶段所需要的技术人员的类型、层次和数量均不相同。因此，在整个软件开发期间，应该根据项目进行的不同阶段适时调度开发人员，恰当掌握用人标准，对人员的选择、配置和组织进行适时调整，合理安排。这是关系到软件开发的效率、进度和质量的大问题，必须引起高度的重视。

通常，参与软件过程（即每一个软件项目阶段）的人员可以划分为以下 5 类。

（1）高级管理者：负责确定商业性问题，这些问题往往对项目产生很大影响。

（2）项目管理者（如项目经理）：负责计划、激励、组织和控制软件开发人员，并与客户沟通、协调。保证项目按时、按质、按量完成。以及辅助管理人员负责文档事务处理。

（3）开发人员：负责开发一个软件产品或应用软件部件所需的专门技术人员：系统分析员、程序员（高、中、初级）、测试工程师等。

（4）客户：负责说明待开发软件需求的甲方人员。

（5）最终用户：一旦软件发布成为产品，直接与软件进行交互（打交道）的使用人员。

一般情况，软件项目不同阶段对不同层次的技术人员分配的工作量及需要情况是不一样的。图 9-3 是典型的软件开发人员参与情况曲线。特征显示，在项目前期和后期高级技术人员和管理人员更多地参与工作，而到中期则需要更多的初级技术人员参与。

这就是被称之为工作量分配的“40-20-40 规则”。40%或者更多的工作量分配给前端的分析和设计任务。类似比例的工作量用于后端的测试和系统集成，只有约 20%的工作量用于编码。这种工作量分布只是被当作指导性原则。而各个项目的特点决定了其工作量分布状况有所不同。一般用于项目计划的工作量大概 2%到 3%，需求分析大约占用 10%到 25%的工作量。另有 20%到 25%的工作量用于软件设计。由于在软件设计期间投入了相当的工作量，随后的编码工作就变得相对简单些，15%到 20%的工作量即可完成这一工作。测试和随之而来的系统调试工作将占用 30%到 40%的软件开发工作量。

在一个小型软件开发项目中，可能只需一个人就可以完成需求分析、设计、编码和测试。但随着项目规模的增大，必然涉及到更多的人员参与，不能容忍 10 人年工作量的项目让一个人工作十年来完成，这时，项目开发团队的组织就成为必然。

通过合理有效的团队组织结构可以使项目组最大限度地发挥每个人的技术水平和创造能力，同时通过相互的有机配合与协调运作，使软件项目开发获得更高的工作效率。当然，项目人员的组织结构最终采用什么样的形式才合理、才有效，应该根据软件项目的特点决定，同时也与参加开发的人员素质有关。常见的组织结构模式有 3 种。

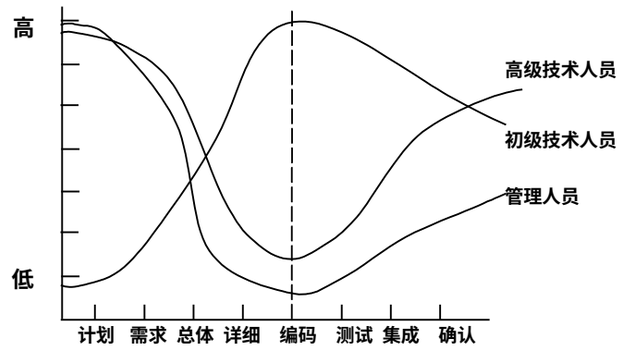


图 9-3　软件开发各阶段人员参与状况图

（1）按课题组划分的模式：把开发人员按课题方向组成小组，小组成员自始至终承担本课题的各项任务。这要求小组成员不仅能够独当一面，而且在其他各方面也有一定专长，具有全面把握的能力。该模式适用于规模不大的项目，每个人都是一专多能。

（2）按功能职责划分的模式：把参与开发项目的软件人员按任务的工作阶段划分为若干工作小组。所开发的软件在每个专业小组完成阶段加工后沿工序流水线向下传递。这种流水作业的方式适用于多项目并行的情况。其特点是每个小组都能把自己那部分工作锤炼得炉火纯青，符合软件工厂的开发模式。

（3）按矩阵形态组合的模式：这种模式是以上两种模式的整合。一方面按工作性质成立多个专门小组，属于按职责划分，另一方面每一个项目都由其项目经理负责，属于按课题组划分。每个软件开发人员既属于某一个专门小组，又在项目经理领导下参加某

一个项目的具体工作。该模型比较适合于规模中、大型的软件项目。

在各种模式下，人员组织结构的底层还可以划分成有效率的程序设计小组，各组相互之间采用多种协调和通信技术，以此支持各组进行高质量的软件开发工作。众所周知，根据软件工程要求，软件产品已不再是程序员个人独立完成的作品，而是团队合作开发的成果。这意味着相互之间沟通联系已必不可少。而且小组内部人员的组织形式对提高开发效率也有着十分重要的影响。那么如何进行小组组织就成为关键要点，常见的程序设计小组组织形式有 3 种，可根据实际情况灵活运用。

（1）民主分权式（democratic decentralized，DD）：这种软件工程小组没有固定的负责人。“任务协调者是短期指定的，之后就由其他协调不同任务的人取代”。问题与其解决方法的确定是由小组讨论决策的。小组成员间的通信是平行的。

实例：民主制小组——在民主制小组中，遇到问题可以在组员之间平等地交换意见，工作组目标的制定以及决定的作出都由全体人员参加。这种组织形式强调发挥每个成员的积极性，并要求每个成员充分发挥主动精神和协作精神。

（2）控制分权式（controlled decentralized，CD）：这种软件工程小组有一个固定的负责人，协调特定的任务及负责子任务的二级负责人关系。问题解决仍是一个群体活动，但解决方案的实现是由小组负责人在小组之间进行划分的。小组和个人间的通信是平行的，但也会发生沿着控制层产生的上下级的通信。

实例：层次式小组——在层次式小组中，组内人员分为 3 级：组长（负责人）一人负责全组工作，直接领导两到三名高级程序员，每位高级程序员通过基层小组，管理若干位程序员。这种结构比较适合于项目本身就是层次结构的课题。

（3）控制集权式（controlled centralized，CC）：顶层的问题解决和内部小组协调是由小组负责人管理的。负责人和小组成员之间的通信是上下级式的。

实例：主程序员制小组——相当于组长负责制，由主程序员和若干个助手（程序员等）组成软件开发小组。主程序员负责规划、协调和审查工程的全部技术活动。程序员负责软件的具体分析和开发。这种组织结构突出主程序员的领导，强调主程序员与其他技术人员的联系。

9.3.3 人员组织与管理

在人力配备问题上，若是调理不当，很容易造成人力资源的浪费，并延误工期。特别是采用恒定人员配备方案时，会在项目的开始和最后都出现人力过剩的情况，而在中期又会出现人力不足的现象。参加软件开发的人员如何组织起来并施加有效的管理，发挥最大的工作效率，对成功地完成软件项目极为重要。

项目人员的组织并不在于人多势众，只要合理的调配，即可获得理想的生产率。倘若认为：“即使进度拖延也总可以增加更多的程序员，并在后期跟上进度”的想法是十分危险的。实际情况是，在项目后期增加人手通常会产生一种破坏性影响，其结果是使进度更加拖延。因为后来增加的人员必须先学习本系统，而培训他们的人员正是原来一直工作着的那些人，当他们在进行教学时，就不能完成任何开发工作，而项目也就进一步被拖延了。除去学习系统所需的时间之外，新加入人员将会增加人员之间通信的路径数量并造成

整个项目中通信的复杂度。虽然通信交流对于一个成功的软件开发项目而言绝对是必不可少的，但是每增加一条新的通信路径就会增加额外的工作量，从而需要更多的时间。

从整个软件项目的人事管理角度来组织调配时，可采用项目经理负责制模式以形成人力资源层次结构，以便于层次型人员管理。首先成立项目管理委员会，项目管理委员会下设项目管理小组、项目评审小组和软件产品项目组，各小组之间的工作既是独立的又是相互制约的，但都必须向项目管理委员会负责。

1）项目管理委员会

项目管理委员会是负责项目管理的最高决策机构，一般由软件开发单位总经理、副总经理组成。主要职责有：①依照项目管理相关制度，管理项目；②监督项目管理相关制度的执行；③对项目立项、项目撤销进行决策；④任命项目管理小组组长、项目评审委员会主任、项目组组长。

2）项目管理小组

项目管理小组对项目管理委员会负责，一般由软件开发单位管理人员组成。主要职责有：①草拟项目管理的各项制度；②组织项目阶段评审；③保存项目过程中的相关文件和数据；④为优化项目管理提出建议。

3）项目评审小组

项目评审小组对项目管理委员会负责，可下设开发评审小组和产品评审小组，一般由软件开发单位技术专家和市场专家组成。主要职责有：①对项目可行性报告进行评审；②对开发计划和阶段性报告进行评审；③对市场计划报告进行评审；④项目结束时，对项目总结报告进行评审。

4）软件产品项目组

软件产品项目组对项目管理委员会负责，下设软件项目组和产品项目组（如果是软件产品项目的话）。软件项目组和产品项目组分别设开发经理和产品经理。成员一般由技术人员和市场人员构成。主要职责是：根据项目管理委员会的安排，软件项目组具体负责项目的软件开发，产品项目组具体负责市场调研及销售工作。

当选定了符合自身特点的项目人员组织结构，又采用了适宜的小组组织形式，那么，在制定责任与权利时应该注意以下原则。

（1）尽早落实岗位职责——抓紧指定专人负责软件开发，赋予其进行管理的权力，并对任务的完成负责。例如，委派项目经理；指定程序小组长等。

（2）尽量减少交互接口——软件开发过程中，人员之间的联系是必不可少的，但必须注意到：人员组织的工作效率是和完成任务中存在的人际联系数目成反比的，过多的交往联系势必降低工作效率。应保持任务的独立性，减少相互交往的频率。

（3）尽力保持责权均衡——软件经理人员所负的责任不应比委任给他的权力还大。明确责任和权利是对等的。

在项目人员管理的实施过程中，还应增强甲方与乙方项目经理之间的沟通、项目经理与项目组成员之间的沟通，以使信息能够及时上传下达，避免错误或问题沉积，这也是出现在人员管理中的重要环节。虽然项目开发中相互联系与通信是必不可少的，但并不提倡在小组成员中频繁的交往，那么在管理上就需要建立良好的沟通机制，保证有效的沟通渠道。通常可以采取以下几种常见的通信方式。

（1）正式非个人方式：如正式会议、正式文件等；

（2）正式个人之间交流：如成员之间的正式讨论、小组会等（一般不形成决议）；

（3）非正式个人之间交流：如个人之间的自由交流、开发经验等；

（4）电子通信：如 E-mail（电子邮件）、BBS（电子公告板系统）等；

（5）成员网络：如小组成员与小组之外或公司之外有经验的相关人员进行交流。

虽然沟通可以增进了解，但通信是要花费时间和代价的，如果说一个软件开发组有 n 个人，每两人之间都需要通信，则总的通信路径有 $n*(n-1)/2$（条）之多，工作效率必然以减去花费在通信上的花费为代价，实际效果就是降低了原有的生产率，所以相互通信必须合理掌握一个度。

在进行人员管理时，不仅要着手招聘、培训、考评、薪资等各个具体内容的操作，而且必须重视其中的风险管理问题。其实，每个项目在人事管理中都可能遇到风险，如招聘失败、新政策引起员工不满、技术骨干突然离职等等，这些事件将影响项目的正常运转，甚至导致项目开发的瘫痪。如何防范这些风险的发生，是应该注重和积极加以防范的问题。因此，必须加强人力资源管理中的风险管理。要有效地规避人力风险，则应该在软件项目的人员配备方案中掌握以下的原则。

（1）注重质量：对于技术性很强的软件项目开发工作，人员的素质与能力尤为重要，哪怕只是任用少量实践经验丰富、独立开发能力强的人员去完成关键性任务，也常常要比使用众多经验不足的人员更加高效。

（2）搞好培训：能力是靠培养练就出来的，实践是一方面，而培训更是一方面。有计划、有投入地培养所需的技术人员和管理人员，是储备人力资源的有效办法。

（3）双轨任用：人员的使用与提升有技术与管理之别，各负其责，各行其是，在不同的专业与业务上各尽其才。

人员配备中除了组织结构上的考虑之外，全面评价一个软件人员的素质与能力，考察其各方面的条件也是十分重要的，以下几点可以作为评判参考。

（1）牢固掌握专业基本知识和开发技能。眼界开阔，思想活跃，接受新事物较快。

（2）思路敏捷、善于分析、具有严密的逻辑思维能力和综合解决问题的能力。

（3）工作踏实、用心细致，遵循标准和规范，具有严格的科学作风。

（4）工作敬业、有较强的耐心、毅力和责任心。具有创新思维和个性风格。

（5）善于听取意见，善于团结协作，善于沟通关系。具有良好的团队合作精神。

（6）行文流畅，言简意赅，具有良好的书面和口头表达能力。

另外，对项目负责人、项目经理等高层人员的能力和素质也相应有更进一步的要求。

（1）具有进行计划、指导、控制和评价项目实施各项活动的能力。

（2）具有组织和领导项目团队的能力，并能协调与项目有关的各部门的工作。

（3）对项目实施过程中出现的问题能准确的做出判断，并能提出解决办法。

（4）对项目实施过程中潜在的问题能及时预测，并能提出预防措施，或化解方法。

（5）善于信息交流和沟通，能处理好各种不同层次的人际关系。

（6）善于计划和利用时间，把有效时间集中于处理最重要和最关键的问题上。

（7）善于沟通与客户的关系，并能处理和协调好与用户、分包单位之间的问题。

（8）项目经理应熟悉项目管理业务，对与项目实施有关的工作有一定程度的了解，

尤其是对项目实施各阶段之间的衔接和联系应做到心中有数。

（9）对项目团队中的每个岗位的职责和分工有充分的了解。

（10）一般应具有较深厚的项目管理工作经验。

9.4　进 度 计 划

项目进度计划是根据项目的目标，在项目确定的范围内、依照确定的需求和质量标准、并在项目进度及成本预算许可的情况下，制定出一个周密的项目活动安排的过程。

制定计划的目标不是幻想消除实现过程中的所有错误，而是要在细心规划下少犯错误或只犯意外的错误，使错误造成的损失降到最小。没有计划是不行的，不具备可执行性、可操作性的计划也是不行的。项目进度计划就是为了减少开发过程中可能出现的时间偏差，确保项目能够按时、按质、按量地完成。

9.4.1　制定开发进度计划

制定项目开发进度计划是在对项目范围定义完成之后，根据工作分解结构（WBS）所划分的具体工作任务进行工作时序安排的过程。主要是规划各项任务何时开始、持续多久、何时结束，以便在时间上跟踪和控制项目的进度，确保项目能够按时完成。

正因为每个项目都可根据一定的原则将其分解为一系列活动（任务），每一活动还可以分解为一系列的子活动（子任务）。由此即可对各个工作包（任务的最小划分）制定实施的进度计划，使每项任务都能有条不紊、按时保质地完成。由于进度计划来自工作任务安排，那么其规模与需求是基本明确的，在此基础之上再进行资源的分配以及进度计划的安排将更加切实可行。

在制定进度计划时需要考虑的一个重要问题是任务的并行性问题。当参加项目开发的人数不止一个人时,为了最大限度地提高工作效率必须采取软件开发并行工作的方式。由于并行任务是同时发生的，所以进度计划应该体现和决定任务之间的从属关系，确定各个任务的先后次序和衔接条件，明确各个任务完成的持续时间与制约关系。另外还应注意处于关键路径上的任务，这样才能确保在项目进度安排中抓住重点，稳步推进。

当项目进度计划的任务信息确定以及相互关系明确后，还要对所有环节进行整合分析、评估，考察各个任务开工和完工之间有无冲突、整个进度计划是否与原始目标的定义相吻合、当工期拖延时采取什么调控方案、阶段性任务完工日期的控制任务等，权衡全局使之达到合理可行的状况。

另外，制定项目开发进度计划，要将跟踪、报警和控制作为进度的保证措施，通过设置警戒线或底线的方法加以实施，以此提示拖延时间可能引起的风险。警戒线上应该设置必要的解决或缓解延时问题的活动。当警戒信号出现后，就应该执行相应应急处理预案。

在项目管理中还必须处理好进度与质量之间的关系。在进度压力下赶任务，其结果往往是以牺牲产品的质量为代价的，这将得不偿失。因此，完善软件项目的开发进度计划本身也是一项重要的任务。

进度安排方法常用的有两种，即甘特图（gantt chart）法和工程网络法〔包括关键路径方法（CPM）及计划评估和复审技术(PERT)。“甘特图”是最常用的时间管理方法，

而“关键路径方法（CPM）”与“计划评估和复审技术（PERT）”则是两种进度安排方法。利用项目管理的工具软件可快速完成进度计划的制定并绘制相关的控制图表，如：Microsoft Project 软件。

9.4.2 甘特图与时间管理

项目时间管理是在项目的执行和实施过程中进行，通过经常性地检查实际进度是否按计划要求完成，来发现和纠正偏差，从中及时找出原因，积极采取补救措施或调整修正原计划。在创建软件项目进度表时，可以从一组任务（工作分解结构 WBS）入手。如果使用自动工具，就可以用任务网络或者任务大纲的方式输入工作分解结构，然后为每一项任务输入工作量、持续时间和开始时间。此外，每一项任务还必须被分配给特定的人员。由此可为整个项目建立一个时间表，即甘特图。

甘特图类似于表示时间关系的横道图，先用水平线段表示任务的工作阶段，线段的起点和终点分别对应着任务的开工时间和完成时间，线段的长度表示完成任务所需的时间。而横坐标方向附加一条可向右移动的纵线，标注所有任务行进到当前时间的位置，并且表明各项任务进展的相对状况。图 9-4 给出了一个具有 5 个任务（A～E）的甘特图。其中 5 条横向线段分别定义 5 项任务的计划时段，而可向右移动的纵线指明了随项目进展已完成（纵线左边）的任务和未完成（纵线右边）的任务。从甘特图上即可清楚地看出各子任务在时间上的对比关系。

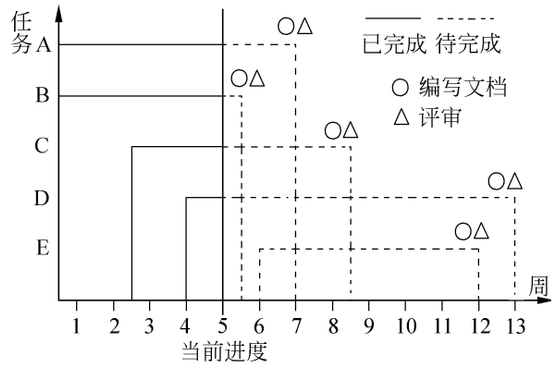


图 9-4 甘特图

在图示方法中，涉及到以下几方面的信息，应该给予文字说明：

- 各个任务的计划开始时间和完成时间；
- 各个任务完成的标志（即○：文档编写；△：评审）；
- 各个任务与参与工作的人数，各个任务与工作量之间的衔接情况；
- 完成各个任务所需的物理资源和数据资源。

甘特图的优点是标明了各任务的计划进度和当前进度，能动态地反映项目开发进展情况。缺点是难以反映多个任务之间存在的复杂的逻辑关系。

时标网状图克服了甘特图的缺点，用具有时标的网状图既表示了各个任务的分解情况，又展示了各个子任务之间在进度上的逻辑依赖关系（见图 9-5）。时标网状图中的（直线、折线）箭头表示各任务间的（先决）依赖关系；箭头上的名字表示任务代号；箭头

的水平长度表示完成该任务的时间；而圆圈表示一个任务结束、另一个任务开始的事件。如：E1 任务能够开始④必须是 B2 任务已结束（先决条件）。

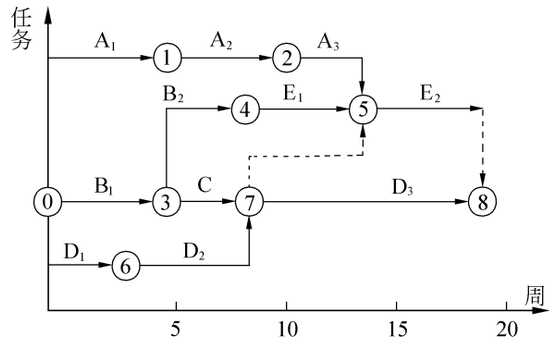


图 9-5　时标网状图

在实际工作中，每一任务完成的标准，并不能以甘特图中该阶段任务结束或下一任务开始为依据，而必须是在交付的阶段性文档通过评审后才决定能否继续往下进行。因此在甘特图中，文档编制与评审是软件开发进度的里程碑和基线（指一个（或一组）配置项在项目生命周期的不同时间点上通过正式评审而进入正式受控的一种状态）。甘特图只能表达工作进度与时间的关系，而不能凭此强行实施下阶段的工作。

9.4.3　工程网络与关键路径

任务之间的依赖关系也可以使用工程网络的方式来定义。其中，关键路径法（CPM）和计划评审技术（PERT）是两种可以用于项目进度安排的常用方法。这两种方法都提供了网络图分析工具，将项目从开始到结束的工作定量划分描述成一个项目的任务网络，以图或表的形式表示出来，可以支持项目进度进行以下处理。

（1）确定关键路径——由此决定项目持续时间的任务链；

（2）开发时间估算——使用统计模型为单个任务建立最可能的开发持续时间值；

（3）计算边界时间——为特定任务定义时间“窗口”，以指导进度安排与调度。

1．关键路径法（CPM）

项目的关键路径是指一系列决定项目最早完成时间的活动链接。虽然这些活动并不一定是项目中最重要活动的集合，但肯定是项目网络图中最长的路径，并且有最少的浮动时间。这是一种用来预测总体项目所经历时间的网络分析工具。

尽管关键路径是最长的路径，但是代表了为完成项目所花费的最短时间。在绘制完成了项目网络图之后，通过计算，能够确定出项目各任务的最早开始时间、最迟开始时间和结束时间。通过最早与最迟时间之差来分析每一工作相对于时间的紧迫程度及工作的重要程度，其中最早开始时间与最迟开始时间的差额称为浮动时间。浮动时间为零的工作即为关键任务，而由这些关键任务组成的一系列日程的衔接就构成了关键路径。简要地说，就是将项目网络图中每条路径所有活动的历时分别相加，其中最长的一条就是

关键路径。可见，关键路径上的每一项任务都是关键任务，这些任务的特点是没有浮动时间可拖延。

一个项目也可能有多条关键路径，并且关键路径也可能因实际情况发生变化。事实上，关键路径只与项目的时间维度有关。这时需要注意多条关键路径的活动执行情况。

关键路径法的主要目的就是要确定项目中的关键任务，而后找出串接这些关键任务的关键路径，以保证项目实施过程中重点突出，线路明确，按期完成（见图 9-6 关键路径法示图）。

在项目行进中总会存在这样一类直接影响项目工期变化的关键任务，这些任务的完成时间直接影响到下一道工序的开始，即每一天都是计划规定必须达到的，没有多余的机动时间。如图 9-6 所示：0 号结点是某个工作网络的起点，8 号结点是工作网络的终点，图中各节点代表不同的任务开始事件，节点之间的边标示了完成该任务所需要的计划时间，而整个工期最长 41 天结束。经分析，其中 0123678 节点就是关键任务的开始，因为这些节点没有多余的时间作为机动，而由这些节点所连接的路径：0－1－2－3－6－7－8 就是关键路径，在这路径之上的任何一项关键任务都不得拖延，否则工期就不能按时完成。那些不在关键路径上的任务为非关键任务，或多或少都可以拖延一定的时间却不会影响整个工期。利用关键路径图可以明确地表达各项任务的计划时间，以及各项任务之间的依赖关系。

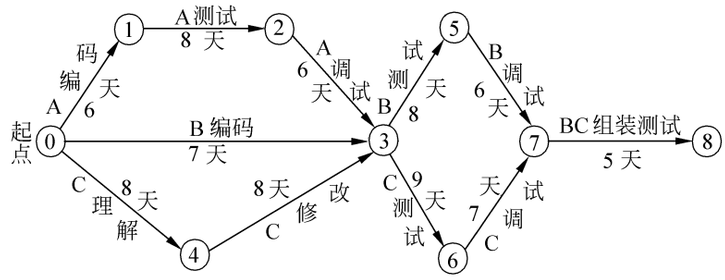


图 9-6　关键路径法

2．计划评审技术（PERT）

计划评审技术是当具体活动的历时估算存在很大的不确定性时用来估计项目历时的网络分析技术。

计划评审技术（PERT）的形式与关键路径法（CPM）基本相同。PERT 通常使用的计算方法也是 CPM 的方法。只是在工作延续时间方面 CPM 仅需要一个确定的工作时间段，而 PERT 需要一个工作的 3 个时间估计，包括最短时间（乐观的）a、最可能时间（常规的）m 及最长时间（悲观的）b，然后采用概率 β 分布来计算工作的期望时间 t，接着再根据这些时间 t 值去做工程网络图，进而再求出关键路径。

通常用两张表来定义网络图。一张表给出与一特定软件项目有关的所有任务（也称为任务分解结构 WBS），另一张表给出应当按照什么样的次序来完成这些任务（也称为限制表）。在组织较为复杂的项目任务时，或是需要对特定的任务作进一步更为详细的计

划时，还可以使用分层的任务网络图。目前甘特图、PERT 和 CPM 都已经在多种项目管理工具软件中得到实现，并能提供有效的进度跟踪、报警功能（如 Microsoft Project 软件）。

9.4.4　项目进度跟踪与控制

项目计划执行过程中，经常会出现一些预先无法想到的情况而使项目的进度早于或晚于计划进度或使项目的实际成本低于或高于计划成本，这时需要对项目计划作出相应的调整，并对近期内即将发生的活动加强控制，以便积极纠正并挽回时间和成本造成的偏差或损失。

项目跟踪指的是在项目运行过程中把实际发生的情况与原来估计的情况进行比较，以检查项目开发是否按照计划正常进行。

实施跟踪需要在项目计划制定结束评审通过时即把这一原始计划保存下来，该原始计划又称为基准。当然要跟踪项目光建立基准不行，还需要了解跟踪项目有关的具体步骤并建立有效的跟踪处理程序。

跟踪项目进度是一项经常实施的重要活动，不仅可以考察项目的进展情况，还可以检查项目的质量状况，在进行这项活动时应该注意以下几点。

（1）安排日程并建立比较基准计划，应该确定项目信息更新日程以达到监督进度所要求的频率。对每个任务的状态进行记录并更新日程中适当的信息。这样可以对比基准计划中的任意信息进行粗略或详细的跟踪。

（2）跟踪项目进度重要的是及时更新项目信息，这样可以及时反映项目的实际运行状况与比较基准计划的差异，以便及时调整和修正计划，达到项目跟踪的真正目的。

（3）更新项目信息既可以针对整个项目，也可以仅更新项目的部分任务，但应该是及时与准确的信息。

（4）项目信息更新的依据必须是项目当前的实际数据。

由于项目跟踪是一种调查研究的过程，具有其经常性和特殊性，推荐通过以下方式及步骤加以实施。

（1）定期举行项目状态会议，由项目组中的各个成员分别报告工作进度和遇到的问题。

（2）评价所有在软件工程过程中必须进行的复审的结果。

（3）确定由项目计划所安排的项目里程碑是否在预定日期内完成。

（4）比较作为基准之一的项目资源表（该表包含任务名称、计划开始、实际开始、计划结束、实际结束、人员分配、工作量分配、附注等记录项）中列出的各项任务的实际开始日期与计划开始日期。

（5）与开发者进行非正式会谈，获取他们对项目进展及可能出现的问题的客观评估。

实际工作中，软件项目管理人员应该利用控制手段来管理项目资源，并指导项目开发人员。如果过程进展顺利（即，项目在预算范围内按进度进行，复审结果表明的确取得了实际进展，达到了各个里程碑），则几乎不必施加控制。但是如果出现问题，项目管理者就必须施加严格控制，以便尽快排解问题。例如，在应用领域中可能需要追加一些

资源；可能需要重新部署人员，可能需要重新调整项目进度等处理。

一旦真的需要对项目计划进行调整，必须注意以下的问题：

- 对近期内即将发生的活动加强控制，积极补救时间和成本；
- 对工期估计较长或预算估计较大的活动应进一步审核其预估根据， 并做好压缩该活动时间和费用的准备工作，因为估计值越大的项目更有压缩的可能；
- 将某些可以再细分的活动再进一步细分，研究细分活动之间并行工作或技术重用的可行性，如可行，则可以有效地压缩时间和费用。

9.5 风 险 管 理

项目风险是指潜在的预算、进度、人力、资源、客户及需求等方面的问题以及这些问题对软件项目的影响。项目风险的存在将会威胁到项目计划的实施。也就是说，如果项目风险变成现实，就很有可能会拖延项目的进度、增加项目的成本甚至毁灭整个项目。可以肯定地说：当软件开发过程或者软件产品本身具有的风险一旦爆发将会造成对软件项目的极大损害或巨大损失。

项目风险事实上肯定存在但管理上又难以把握，这是因为软件项目风险中包含了两个关键特性:

（1）不确定性——刻划风险的事件可能发生也可能不发生；即，没有 100%发生的风险，只有 100%存在着风险。这使人们对软件风险的防范意识受到严峻的考验。

（2）可能损失——如果风险变成了现实，就会产生恶性后果或严重损失。

风险管理关注的是未来将要发生的事情。如果能够提前重视软件项目管理中存在的风险，并且有所防范，就可以最大限度地减少风险的发生，就算是风险已经发生也可以实施有所准备的应对方案，将风险造成的损失减少到最低限度。这其中，进行风险管理是一种必要的、也是行之有效的手段。可见，风险管理在项目管理中的确占据非常重要的地位：

- 有效的风险管理可以提高项目的成功率。在项目早期就应该进行必要的风险分析，并通过规避风险以降低失败概率，避免返工造成的成本增加。
- 提前对风险制定应对预案能够保证合理的处理。在风险发生时迅速作出反应，避免忙中出错造成更大的损失。
- 风险管理可以增加团队的健壮性。与团队成员一起做风险分析可以让大家对困难有充分估计，对各种意外早有心理准备，不至于受挫后士气低落；而项目经理如果心中有数就可以在发生意外时从容应对，有利于提高组员的信心从而稳定队伍。
- 有效的风险管理可以帮助项目经理抓住工作重点，将主要精力集中于重大风险的分析与规避，将工作方式从被动救火式转变为主动防范式。

总之，项目开发中对风险进行有效的控制管理就可以大大提高软件开发的成功率。项目风险管理工作就是要在风险成为影响软件项目成功的威胁之前，认真识别、着手处理并积极消除风险的源头。通常，在规划和实施风险管理时可以简单地分成 5 个步骤：风险识别、风险分析、风险计划、风险跟踪和风险应对。

9.5.1　风险识别与分类

风险识别就是要系统地、明确地找出不确定因素对软件项目计划（估算、进度、资源分配等）构成的威胁，并通过识别已知的或可预测的风险，来达到监控风险与规避风险的目的。

识别风险的先导活动就是用各种不同的方法对风险进行分类。根据风险内容一般可将风险分为以下几类。

（1）项目风险——因项目在预算、进度、人力、资源、顾客、需求等方面的估计有误造成。例如：没有在预算的成本范围内完成任务（成本风险）；没有在预计的时间范围内完成任务（进度风险）；没有按照要求的技术性能和质量水平完成任务（质量风险）；因需求变更造成范围变更的有关影响（范围风险）等。

（2）技术风险——因技术不成熟、不适用等原因造成。由于软件在设计、实现、接口、验证、维护过程中可能存留潜在的问题，或采用了不可靠的技术等对项目造成的危害。

（3）商业风险——因市场问题、销售问题等因素造成。由于软件没有人要、或不知道、或无法推销而造成的损失。

（4）法律风险——因许可权、专利、合同失效、诉讼败诉等造成。

（5）内部非技术风险——因公司管理人员不成熟等原因造成（管理风险）；或由于公司经营战略发生了变化等原因造成（战略风险）。

（6）外部可预测风险——因市场需求、维修需求、日常运作、环境影响、社会影响、货币变动、通货膨胀、税收等造成。

（7）外部不可预测风险：因规章（不可预测的政府干预）、或自然灾害等不可抗力造成。

另外，根据对风险的认知程度（可确定性）还可以将风险分为以下几类。

（1）已知风险——如员工离职等。

（2）可预报风险——从历史经验得出可能有的风险。

（3）不可预知风险——新情况产生的新问题。

风险分类为的是便于风险识别。在风险识别过程中主要是将不确定性错误转变为明确的风险陈述。通常需要完成以下几项任务。

（1）进行风险评估——在项目的初期，以及主要的转折点（里程碑）或重要的项目变更发生时进行风险评估。这些变更通常指成本、进度、范围或人员等方面的改变。

（2）系统地识别风险——采用 3 种简单的方法识别风险：风险检查表，定期会议（周例会），日常状况（每天晨会）。

（3）将已知风险编写为文档——通过编写风险陈述和详细说明来记录已知风险，相应的风险背景描述包括关于风险问题的：何时、何地、何事、何故及如何处理。

（4）交流已知风险——在大家都参加的会议上同时以口头和书面方式交流已知风险，必须将识别出来的风险详细记录到文档中，以便他人查阅和防范。

风险识别的有效方法是建立风险检查条目表（参考下节：风险评估与分析），此表

可以通过列出所有可能的与每一个风险因素有关的提问来帮助项目计划人员了解在项目和技术方面存在哪些风险。主要涉及以下几方面检查及其相关条目。

（1）产品规模风险检查——与待开发或要修改的软件的产品规模（估算偏差、产品用户、需求变更、软件复用、数据库等）相关的风险。因为，项目风险是直接与产品规模成正比的。待开发产品的信息必须与过去的经验加以比较。如果出现了较大的百分比偏差，或者如果数字相近但过去的结果很不令人满意，则风险较高。

（2）业务影响风险检查——与管理和市场所加之的约束（公司收益、上级重视、符合需求、用户水平、产品文档、政府约束、成本损耗、交付期限等）有关的风险。 因为，销售部门是受商业驱动的，而商业考虑有时会直接与技术现实发生冲突。另外来自市场的不确定因素也大大增加了与商业影响相关的常见风险。

（3）与客户相关的风险检查——与客户的素质（人文素质、合作态度、需求理解、技术水平等）以及开发者与客户定期通信（技术评审、交流沟通等）能力有关的风险。因为，对于项目管理者而言，不合作的客户是对项目计划的潜在危险和实际风险。

（4）过程定义风险检查——与软件过程被定义的程度以及被开发组织所遵守的程度相关的风险。因为，如果分析、设计及测试以无序的方式进行；如果每个人都认为质量很重要但又没有人切实地采取行动来保证质量，那么，这个项目就处于风险之中。

（5）技术攻关风险检查——与待开发软件的复杂性及系统所包含技术的“新颖性”相关的风险。因为，突破技术的限制是极具挑战性且令人兴奋的，但这也恰恰是极具风险的。

（6）开发环境风险检查——与用来建造产品的工具（项目管理、过程管理、分析与设计、编译器及代码生成器、测试与调试、配置管理、工具集成、工具培训、联机帮助与文档）的可用性和质量相关的风险。因为，即使是熟练的开发者，不适当的或没有效率的工具也会阻碍工作的进行。如果环境有缺陷，就很可能成为重要的风险源。

（7）与人员经验和模式有关的风险检查——与参与开发的技术人员的总体技术水平（优秀程度、专业把握、效力时段、数量充足等）及项目经验（业务培训、工作基础等）相关的风险。因为，接受过必要培训的优秀开发人员是否能够自始至终地参加整个项目的工作将左右着软件产品开发的成败，人员的离走确实蕴含着极大的风险。

风险检查条目表本身能够以不同的方式进行组织，重要的是表中要能够明确详尽地反应软件项目中可能产生风险的各个条目定义。与上述每个话题相关的问题可以由针对具体项目的调查分析来解答，而这些问题的答案可以使计划者能够估计出风险的种类及其产生的影响。当然也可以采用另一种不同形式的风险检查条目表，仅仅列出与每一个常见风险子类型有关的特性。最后列出一组风险元素（性能、成本、支持、及进度）和驱动因子（不可能、不一定、可能和极可能），以及风险元素和驱动因子发生的概率，供风险分析使用。

经验表明，所有项目风险的 80%（即可能导致项目失败的 80%的潜在因素）能够通过 20%的已识别风险来说明。这不仅诠释了风险识别的重要性，同时也告诉管理者只要抓住那些属于 20%之内的风险（具有最高项目优先级的风险），就能够有效地控制和驾驭风险处理。

9.5.2　风险评估与分析

一旦能够识别风险并进行风险分类之后，针对软件项目去评估可能存在的风险等级及其影响程度、进而分析其特性并实施对其控制就显得尤为重要了。

在风险评估过程中，要进一步审查在风险预测识别阶段所做推测的精确度，试图为所发现的风险排出优先次序，并开始考虑如何控制和避免可能发生的风险。该过程包括以下主要活动：

（1）确定风险的驱动因素。为了很好地消除软件风险，项目管理者需要标识影响软件风险因素的风险驱动因子，这些因素包括：性能、成本、支持和进度；驱动因子包括：不可能、不一定、可能和极可能。

（2）分析风险来源。分析引起风险的根本原因，以便正本清源。

（3）预测风险影响。如果风险发生，就根据可能性及后果程度来评估风险影响。一般将可能性定义为大于 0 而小于 100，并分为 5 个等级（1、2、3、4、5）。而把后果分为 4 个等级（低，中等，高，关键的）。采用风险可能性和后果对风险进行分组，具体为 4 个影响类别：可忽略的、轻微的、严重的及灾难性的。

（4）风险级别排序。对风险按照风险影响进行优先排序，优先级别最高的风险，其风险严重程度等于 1，优先级别最低的风险，其风险严重程度等于 20。针对排序采用优先级算法处理。

对于大多数软件项目而言，前面所讨论的风险因素——性能、成本、支持、及进度——也表示了风险参考的水平值。即，对于性能下降、成本超支、支持困难或进度延迟（或者这 4 种的组合），都有一个阈值的要求，超过阈值就会导致项目被迫终止。如果风险的组合所产生的问题引起一个或多个参考水平值被超过，则工作将被叫停。在软件风险分析中，风险参考水平值就是这样一个存在的点，称为参考点或临界点，在这个点上决定继续进行该项目或立即终止该项目（当问题相当严重时），与其相关的处理结论应该是有理有据、能够被接受的。

风险评估主要从两个方面估计每一种风险：一是估计某个风险发生的可能性；二是估计与风险相关的问题出现后将会产生的结果。通常，项目计划人员与管理人员、技术人员应该一起进行风险估计活动，以建立一个尺度来表明风险发生的可能性，描述风险的后果，估计风险对项目及软件的影响，指明风险估计的准确性以便消除认识上的误解。

一种简单的风险估计方法是采用风险列项排序的手段，将对风险的陈述转变为按优先顺序排列的风险列表（见表 9-14）。（建议采用电子表格，如：Microsoft Excel 来实现风险表定义，这样使得表中的内容易于操纵和排序）

表 9-14　分类前的风险表样本

风 险 条 目	风险类别	发生概率/ %	影响程度	RMMM
规模估算可能非常低	PS	60	2	
用户数量大大超出计划	PS	30	3	
复用程度低于计划	PS	70	2	

续表

风 险 条 目	风险类别	发生概率/ %	影响程度	RMMM
最终用户抵制该系统	BU	40	3	缓解
交付期限将被紧缩	BU	50	2	监控
资金将会流失	CU	40	1	
用户将改变需求	PS	80	2	计划
技术达不到预期的效果	TE	30	1	
缺少对工具的培训	DE	80	3	
人员缺乏经验	ST	30	2	
人员流动比较频繁	ST	60	2	
……				

注：PS：产品/项目规模风险　BU：商业风险　CU：客户特性风险　TE：建造技术风险　DE：开发环境风险　ST：人员经验与经验风险

影响取值：1＝灾难的　2＝严重的　3＝轻微的　4＝可忽略的

RMMM＝风险缓解、监控、及管理计划（risk mitigation, monitoringand management plan）

风险表中第一列尽量详细标出所有可能的、已标识的风险。其定义可以利用风险识别中使用的风险检查条目表来完成。不同的风险在第二列上加以分类（如：PS 指产品规模风险，BU 指商业风险）。每个风险发生的概率则标明在第三列中。 其概率值可以由项目组成员分别估算，然后将这些单个值集中求平均，得到一个有代表性的概率值填入表中。

下一步是评估每个风险所产生的影响。使用表中所述的特性评估每个风险因素，并确定其影响的类别。对 4 个风险因素——性能、支持、成本、及进度——的影响类别求平均可得到一个整体的影响值（如果其中一个风险因素对项目特别重要，也可以使用加权求平均值）。

对于风险表的前四列内容，根据概率及影响来进行排序。高发生概率、高影响性的风险放在表的上方，而低发生概率风险则移到表的下方。这样就完成了第一次风险排序。而后项目经理研究已排序的表，并定义一条水平中止线。该中止线（表中某一项上的一条水平线）表示：只有那些在线之上的风险（发生概率较高）才会得到进一步的关注。而在线之下的风险则需要再次评估以完成第二次分级排序。

风险影响及发生概率从管理的角度来考虑是起着不同作用的。一个具有高影响但发生概率很低的风险因素不应该花费太多的管理时间。而高影响且发生概率为中到高的风险、以及低影响且高概率的风险，应该首先列入管理考虑之中（高概率优先），因为这些风险产生的干扰甚至损害太频繁了，已经到了不能不彻底解决其影响的时候。

RMMM 定义为风险缓解、监控、及管理计划（risk mitigation, monitoring and management plan）。因为，风险管理策略可以包含在软件项目计划中，而且风险管理步骤也可以组织成一个独立的风险缓解、监控和管理计划（RMMM 计划）。这个计划将所有风险分析工作文档化，并由项目管理者作为整个项目计划中的一部分来使用。

一旦建立了 RMMM 计划，且项目开始启动，则风险缓解及监控步骤也就开始了。其实，风险缓解是一种问题规避活动。风险监控则是一种项目跟踪活动，有 3 个主要目

的：①评估一个被预测的风险是否真正发生了；②确保为风险而定义的缓解步骤能够被正确地实施；③收集能够用于未来风险分析的信息。在很多情况下，项目中发生的问题可以追溯到不止一个风险，所以风险监控的另一个任务就是试图在整个项目中确定“风险起源”，即什么风险引起了这样的问题。

如果风险真的发生了所产生的后果，将有3个因素会受到关注：风险的性质、范围及时间。风险的性质是指当风险发生时可能产生的问题。例如，一个定义得很差的与客户硬件的外部接口（属技术风险）会妨碍早期的设计及测试，也有可能导致项目后期阶段的系统集成问题。风险的范围结合了严重性（即风险有多严重）及其整体分布情况（即项目中有多少部分受到影响或有多少用户受到损害）。最后，风险的时间主要考虑何时能够感到风险及风险持续多长时间。

9.5.3　风险策划与管理

风险策划是拟定有组织、有计划的风险管理方法的持续性活动。其主要任务是：制定应对风险策略、规划评估与追踪方案、监控作业项目及职责、鉴别资源配置标准等。

风险会因时间延续和施加干预而改变，当人们尝试去控制或改变风险时，其发生的几率就会变化。在事件发生时，风险影响的判断及处理使得人们必须重新评估并及时作出可能的调整。由于涉及风险的事件是在不断发生和改变着的，所以风险规划过程就得一直持续下去。

被动或主动的风险策略将会影响风险的处理结果。被动策略只是针对可能发生的风险来监督项目，直到风险变成真实的问题时，才会分配资源来处理这些问题。更普遍的情况是，软件项目组对于风险不闻不问，直到发生了错误，这时，项目组才赶紧采取行动，试图迅速地纠正错误。这种行为常常被称为“救火模式”。而主动策略早在技术工作开始之前就已经启动了。事先标识出潜在的风险，评估风险出现的概率及产生的影响，且按重要性加以排序，然后，软件项目组建立一个计划来管理风险。主要的目标是预防风险，但因为不是所有的风险都能够预防，所以，项目组必须建立一个应对意外事件的计划，使其在必要时能够以可控的及有效的方式作出反应。

风险策划过程的活动主要是把按优先级排列的风险列表转变为风险应对预案。该预案包括下列内容。

（1）制定风险应对策略。风险应对策略有接受、避免、保护、减少、研究、储备和转移几种方式。

（2）制定风险行动步骤。风险行动步骤详细说明了所选择的风险应对途径，将详细描述处理风险的步骤。

在实际工作中应尽可能地明确估计各种风险，然后逐一列出并评价风险的程度和级别，然后对这些风险进行密切关注和严格管理。另外，还要对每个风险的表现、范围、时间做出尽量准确的评价判断。应该认识到风险在突发时所呈现出来的特征反应是不一样的，在分析处理时也需要区分风险构成的归类，确定风险属于哪一种形式：

- 性能风险——产品是否能够满足需求且符合其使用目的，其中不确定的程度。
- 成本风险——项目预算是否能够维持生产开发的支出，其中不确定的程度。

- 支持风险——软件产品是否易于排错、适应性强，其中不确定的程度。
- 进度风险——项目进度是否能控制，产品是否能按时交付，其中不确定的程度。

以下的问题是通过调查世界各地有经验的软件项目管理者，从而得到风险数据归纳导出的。这些问题对于项目成功的相对重要性有着不同的影响，根据重要程度顺序排列如下，以便能够准确合理地全面规划和管理项目风险：

- 高层的软件和客户的管理者是否正式地同意支持该项目？
- 终端用户是否热心地支持该项目和将要建立的系统/产品？
- 软件工程组和客户对于需求是否有充分的理解？
- 客户是否充分地参与了需求定义过程？
- 终端用户的期望是否实际？
- 项目的范围是否稳定？
- 软件工程组是否拥有为完成项目所必须的各种技术人才？
- 项目的需求是否稳定？
- 项目组是否具有实现目标软件系统的技术基础和工作经验？
- 项目组中的人员数量对于执行项目的任务是否足够？
- 所有客户是否都一致赞同该项目的重要性并支持将要建立的系统/产品？

只要对这些问题的回答有一个是否定的，就应当制定缓解、监控、驾驭的步骤，即风险规划，以避免项目失败。项目处于风险的程度直接与对这些问题采取否定回答的数目成正比。

在实际管理中，需求的不确定性风险将会导致项目进度、成本、质量、资源、合同等相关风险，使项目容易处于失控状态之中。另外，技术路线风险可能直接导致项目失败。如果项目的目标、范围超过了软件开发所选用技术，尽管软件产品号称拥有高深而实际应用并不成功的分析模型，造成项目组自身对开发技术也不熟悉，无疑会使项目处于毁灭性的风险中。

因此，要合理地进行风险管理并不是一件容易的事情，可从以下 7 个方面进行分析：

- 风险发生的可能性；
- 风险发生的结果（影响）；
- 建立一个尺度表示风险可能性（如，极罕见、罕见、普通、可能、极可能）；
- 描述风险带来的后果；
- 估计风险对软件产品和项目的影响；
- 确定风险评估的正确性；
- 根据影响程度排定有限队列。

对于一个大型项目来说，因为其规模性和复杂性，使得风险管理本身就可能成为一个“项目”。因此，在实际处理上应去伪取真、去粗取精、抓住重点、各个击破。由于整个软件风险的 80%能够由仅仅 20%的已标出风险来说明。因此，对于一些已标出、已评估的风险，若其并不属于具有最高优先级的关键的 20%范围中的风险，就可以采用暂时放弃、来时处理的原则。假如风险管理及意外事件应对计划遭受挑战，且风险变成了现实，例如，项目正在进行之中，有些关键人员宣布将要离开，倘若应急预案落到实处，则有后备人员可以顶上。另外信息已经文档化，有关知识已经在项目组中广泛进行了交

流，从而使得新加入人员能够通过“知识交接模式”很快赶上项目的进度，不至于使软件项目受到人走楼空的过深影响。

9.5.4　风险规避与监控

风险分析活动的目的在于建立处理风险的策略。一个有效的策略最好能规避风险。而风险规避的最好方式是把风险控制在项目启动阶段，把项目损失减小到最小程度。

风险规避是风险应对过程的活动，体现在执行风险行动计划，以求将风险降至可接受程度。主要包括下述内容。

（1）对触发事件的通知作出反应：得到授权的个人必须对触发事件作出反应。适当的反应包括回顾当前现实以及更新行动时间框架，并分派风险行动计划。

（2）执行风险行动计划：应对风险应该按照书面的风险行动计划进行。

（3）对照计划，报告进展：确定和交流对照原计划所取得的进展。定期报告风险状态，加强小组内部交流。小组必须定期回顾风险状态。

（4）校正偏离计划的情况：有时结果不能令人满意，就必须换用其他途径。将校正的相关内容记录下来。

在应对风险的处理过程中，一般存在被动式和主动式两种应对策略。被动策略称为“救火模式”，在处理时风险很可能已经转变成危机了。而明智的策略应该是主动式的，主动策略启动超前，监控及时，评估有序，计划有备。其主要的目标是规避风险，但在风险一旦出现时就能够及时实施处理预案，使风险得到很快的控制，损失降到最低限度。

能够主动出击规避风险的前提条件就是能够跟踪监控并驾驭风险。风险跟踪过程的活动主要包括监视风险状态并及时发出通知，启动风险应对预案并实施消除行动。在这个过程中必须抓住以下环节。

（1）比较阈值和状态：通过项目控制面板来获取。如果指标的值在可接受标准之外，则表明出现了不可接受的异常情况。

（2）对启动风险进行及时通告：对要启动（可能出现）的风险进行预警，在每天的晨会上通报给全组人员，并安排人员负责进行处理。

（3）定期通报风险的情况：在定期的会议上通告相关人员目前的主要风险及状态。

风险的规避与监控主要依靠管理者的经验来实施。例如，某项目开发人员的离职会对该项目造成一定的影响，为了缓解这一风险，项目管理必须建立一个策略来降低人员流动所造成的影响。那么该风险驾驭和监控的策略如下：

- 与现有人员一起探讨人员流动的原因（如工作条件差、收入较低、人才竞争、心理障碍等）；
- 在项目开始前，把缓解这些原因的工作列入管理计划中；
- 当项目启动时，做好会出现人员流动的准备，采取一些有效技术（如编制工作文档），以确保人员一旦离开后项目仍能继续进行；
- 建立良好的项目组织和通信渠道，使大家能够了解开发活动的关键信息；
- 制定文档标准并建立相应机制，以保证能够及时完成文档的建立；
- 对所有工作进行细致的评审，使计划更合理，能够按进度完成；

- 对关键部件的关键人员，必须培养后备力量。

风险规避与控制会带来额外的项目成本。例如，培养关键技术人员的后备力量需要花经费、花时间。因此，当通过某个风险规避步骤而得到的收益被证实其成本超出预算时，要对风险规避部分重新评价，进行传统的成本-效益分析。在考虑风险成本之后，决定是否采用相关策略。不过，花在标识、分析和管理风险上的时间可以从多个方面得到回报，那就是：更加平稳的项目进展过程、较高的跟踪和控制项目的能力以及在问题发生之前已经做了周密计划而产生的信心。通过对风险及其风险的相互作用来评估项目结果的可能范围，可以从成本、进度及性能 3 个方面对风险进行评价，确定哪些风险事件或来源是可以避免的，哪些可以忽略不考虑（包括可以承受），哪些必须采取应对措施。

另外，分析项目风险可以采用一些有效的措施，以达到规避或减小风险的目的。

（1）项目经理同用户最高决策层保持畅通的交流与沟通渠道；

（2）项目开发模型力求采用迭代模型；通过“增量式”的开发方式来及早发现风险，化解风险；

（3）审慎运用尚处于研究阶段的分析技术与模型，选择成熟并适合的技术方法；

（4）采用层次较高、关系合理的项目组织结构；

（5）博采众长，多参考专家意见。

风险规避及追踪监控是一种动态跟踪项目行进的活动。大多数情况下，从项目发生的问题中总能追踪出许多风险。而风险规避与监控工作就是要把“责任”（由什么风险导致问题发生）分配到项目各个具体实施部件中去。

正是因为项目中的风险永远不可能全部消除，所以实际处理中必须具备有效应对方法，一般这种应对分为 3 个层面。

（1）避免：通过分析找出发生风险事件的原因，消除这些原因来避免一些特定的风险事件发生。

（2）减轻：通过降低风险事件发生的概率或得失量来减轻对项目的影响。也可以采用风险转移的方法来减轻风险对项目带来的影响。另外，项目预算中考虑应急储备金也是一种降低风险影响的方法。

（3）接受：通过应急预案及措施接收不可抗拒力等造成的后果。比如：为了避免自然灾害造成的影响，可在一个大的软件项目中考虑异地备份中心。

9.6 软 件 质 量

软件工程的最高目标就是产生高质量的应用软件、系统或产品。随着软件开发规模的扩大，软件质量的问题也就越加突显。虽然无论软件用户或是开发者都追求稳定可靠的软件产品，但如何获得真正意义上的合格软件却不是一件容易的事情。人们往往只重视功能的实现、花费的多少、进度的控制等一系列有形的东西，却容易忽视软件生命周期中应遵循的质量标准和规范。由于软件产品本身的特殊性和现实的需要，使得软件质量控制已不再是仅由软件测试单独处理的问题，而成为在整个软件开发过程中都必须实施的渗透性管理。

早在 1983 年，ANSI/IEEE std729 标准就已给出了软件质量的定义：

软件产品必须满足规定的、隐含的以及与需求能力有关的全部特征和特性。

而（美国）电气电子工程师学会 IEEE 对软件质量的定义则更加具体与清楚：

- 软件产品具备满足给定需求的特性及特征总体的能力；
- 软件拥有所期望的各种属性组合的程度；
- 用户认为软件满足自己综合期望的程度；
- 软件组合特性可以满足用户预期需求的程度。

根据这个定义，在对软件质量进行评测时应该反映出以下 3 方面的特性：

（1）与确定的功能和性能需求的一致性——软件需求是质量度量的基础，缺少与需求的一致性就无质量可言。

（2）与规范化的开发准则的一致性——如果不遵守规范标准的开发准则，软件质量就得不到保证。

（3）与所期望的隐含特性的一致性——针对软件没有显式地提出的隐含需求，加强软件应具备的良好的可维护性。如果忽视软件隐含的需求，软件质量将不可信。

软件质量的保证来自于软件开发中一系列软件过程的规范实施和测试评审。这里有针对软件特性的度量，以此评价软件产品的品质；有公认的质量标准体系，指导怎样实施质量控制管理；有得力的质量保证过程，采用评审加复审严格考核软件产品。现如今，质量保证作为一个软件过程，已明确列入了国际标准“ISO/IEC 12207 信息技术-软件生存期过程”中。实施质量管理的类型也从产品质量检验型管理走到了企业全面质量管理（TQC），进而实行国际质量认证。随着软件技术和软件应用的发展，具有现代软件产业时代特征的以过程为中心的软件开发、生产与质量管理正在实现。

质量形成于过程，高可信、高质量的软件必将依赖对软件过程的管理。下面是目前国际上软件过程质量管理的 3 个典型代表。

（1）CMM /PSP/TSP

- CMM：软件能力成熟度模型（capability maturity model for software），这是由美国卡内基梅隆大学的软件工程研究所（software engineering institute, SEI）受美国国防部评估软件供应商能力的要求，1986 年开始研究制定，并在美国，随后在全世界推广实施的一种软件生产过程评估标准，主要用于软件开发过程和软件开发能力的评估和改进。CMM 标准共分 5 个等级，由低到高分别为：初始级、可重复级、已定义级、定量管理级和优化级。
- PSP：将质量管理的理念和思想向人力资源管理方面扩展而推出的个人软件过程（personal software process, PSP）和团队软件过程（team software process, TSP）以及/人员能力成熟度模型（personal-CMM, P-CMM）。
- 将过程技术与产品线技术融合 :软件产品线 PLP（products line practice）和 COTS（commercial off the shelf）。

（2）ISO 9000 质量标准体系。

（3）ISO/IEC 15504 软件过程评估标准。

在实例软件开发过程中，影响软件产品质量的根源主要有 4 个方面，即：①开发技术、②过程质量、③人员素质、④成本、时间、进度。抓住这些关键部分，就能有效地控制软件质量的提升，从根本上管理好软件质量。

软件工程学者 Robert L Glass 对软件质量的描述给出了一个更加明确的关系式：

用户满意度 = 合格产品 + 好的质量 + 在预算和进度表内交付

这个表达式直观地揭示了：质量是重要的，但如果用户不满意就毫无意义。

9.6.1 软件质量特性与度量

软件质量特性反映了软件的本质，是衡量和评判一个软件质量优劣的依据。定义软件的一系列质量特性就等价于为软件明确了软件应具备的品质，通过与评价准则对比就可以间接地度量软件的质量要素，达到判定软件质量好坏的目的。

通常把影响软件质量的特性用软件质量模型来描述。虽然已有多种关于软件质量模型的设计方案，但共同的特点都是把软件质量特性定义设计为分层结构模型。

1．McCall 质量特征模型

1976 年，Boehm 第一次提出了软件质量度量的层次模型。1978 年，McCall 和 Walters 提出改进的质量模型，明确了软件特性是软件质量的反映，而软件属性可用做评价质量的准则，通过定量化地度量软件属性即可判定软件质量的优劣。同时提出了表明软件质量的 3 类要素共 11 个质量特性（见图 9-7）。

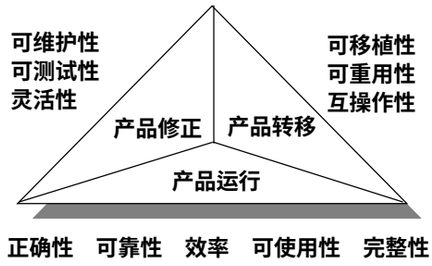


图 9-7 McCall 质量特征模型

（1）表现软件运行特征的要素：正确性、可靠性、效率性、可使用性、完整性。

- 正确性（correctness）——系统满足规格说明和用户目标的程度，即在预定环境下能正确地完成预期功能的程度；
- 可靠性（reliability）—— 软件按照设计要求，在规定时间和条件下不出故障持续运行的程度；
- 效率（efficiency）——为了完成预定的功能，系统所需计算机资源的程度；
- 可使用性（usability）——系统在完成预定应该完成的功能时令人满意的程度；
- 完整性、安全性（integrity, security）——对未经授权的人使用软件或数据的企图，系统能够控制（或禁止）的程度。

（2）表现软件承受被修改的能力方面的要素：可维护性、可测试性、灵活性。

- 可维护性（maintainability）——在运行现场诊断和改正发现的错误所需要的工作量的大小；
- 可测试性（restability）——软件容易检验和测试的程度；

- 适应性（灵活性）（flexibility）——修改或改进正在运行的系统需要工作量的多少。

（3）表现软件对新环境的适应程度方面的要素：可移植性、可重用性、互操作性。

- 可移植性（portability）——把程序从一种硬件配置和（或）软件系统环境转移到另一种配置和环境时，需要工作量的多少；
- 可重用性（reuseability）——在其他应用软件中该程序可以被再次使用的程度（或范围）；
- 互操作性（interperability）——把该系统和另一个系统结合起来使用需要工作量的多少。

以上各个质量特性从高层对软件品质进行了规格要求，但是直接给予度量评价是很困难的。因此，McCall 又定义了一些评价准则以辅助软件质量的评定。评价准则的基础是确定影响软件质量特性要素的属性。且这些属性满足这样两个条件：①能够比较完整、准确地描述软件质量要素；②比较易于量化和测量。在实施质量评价时即可根据具体软件的情况并参照这些评价准则进行分级评判，一般分级范围从 0（最低）到 10（最高），定出级别后，再以此标定软件的质量特性优劣。评价准则（共 29 个）定义如下：

- 可跟踪性——在给定的开发和运行环境下，跟踪设计表示或实际程序部件直到原始需求的（可追溯）能力；
- 完备性——充分实现软件需求的程度；
- 一致性——在整个软件设计与实现过程中技术与记号的统一程度；
- 安全性——防止软件受到意外的或者蓄意的存取、使用、修改、毁坏、攻击或防止泄密的程度，以及防范“计算机病毒”侵害的能力；
- 容错性——也称为健壮性。当系统出错（机器临时发生故障或数据输入非法）时，能以某种预定方式，作出适当处理，得以继续执行和恢复系统的能力；
- 准确性——也称为精确性。表示能达到的计算或控制精度；
- 简单性——以不复杂、可理解的方式定义和实现软件功能的程度；
- 执行效率——为了实现某个功能，提供使用最少处理时间的程度；
- 存储效率——为了实现某个功能，提供使用最少存储空间的程度；
- 存取控制——软件对用户存取权限的控制所采取处理方式能达到的程度；
- 存取审查——软件对用户存取权限的检查程度；
- 操作性——操作该软件的难易程度（通常取决于与软件操作有关的操作规程，以及是否提供有效的输入／输出方法或界面）；
- 易训练性——软件辅助新用户使用系统的能力（这取决于是否提供帮助用户熟练掌握软件系统的方法，如联机帮助。也称为可培训性或培训性）；
- 简明性——软件易读的程度（这个特性可以帮助人们方便地阅读本人或他人编制的程序和文档。也称为可理解性）；
- 模块独立性——软件系统内部接口达到的高内聚、低耦合的程度；
- 自描述性——对软件功能进行自我说明的程度。也称为自含文档性；
- 结构性——软件结构能达到的良好程度；

- 文档完备性——软件文档齐全、描述清楚、满足规范或标准要求的程度；
- 通用性——软件功能覆盖面宽广的程度；
- 可扩充性——软件的体系结构、数据设计和过程设计的可扩充程度；
- 可修改性——软件容易修改，而不致于产生副作用的程度；
- 自检性——软件监测自身操作效果和发现自身错误的能力，也称为工具性；
- 机器独立性——不依赖于某个特定设备及计算机就能工作的程度，也称为硬件独立性；
- 软件独立性——软件不依赖于非标准程序设计语言特征、操作系统特征，或其他环境约束，仅靠自身即能实现其功能的程度，也称为自包含性；
- 通信共享性——使用标准的通信协议、接口和带宽的标准化的程度；
- 数据共享性——使用标准数据结构和数据类型的程度；
- I/O 容量——管理输入输出设备的数量以及缓冲区容量；
- I/O 速率——支持输入输出在单位时间内的传送速度；
- 通信性——提供有效的 I/O 方式的程度。

根据以上评价准则列表对应软件质量特性（见表 9-15），再通过对评价准则的相关款项评分统计，即可衡量出相应质量特性的水平高低，进而综合判定一个软件产品性能的好坏。例如：对可跟踪性、完备性、一致性的评分就可衡量正确性处于何等水平；又如：对执行效率、存储效率的评分即可判定效率程度如何，等等。

表 9-15　McCall 软件评价准则

质量特性 / 评价准则	正确性	可靠性	效率性	可使用性	完整性	可维护性	可测试性	灵活性	可移植性	可重用性	互操作性
可跟踪性	¡Ì										
完备性	¡Ì										
一致性	¡Ì	¡Ì				¡Ì		¡Ì			
安全性		¡Ì			¡Ì						
容错性		¡Ì									
准确性		¡Ì									
简单性		¡Ì				¡Ì	¡Ì	¡Ì			
执行效率			¡Ì								
存储效率			¡Ì								
存取控制					¡Ì						
存取审查					¡Ì						
操作性				¡Ì							
易训练性				¡Ì							
简明性		¡Ì				¡Ì	¡Ì		¡Ì		
模块独立性		¡Ì				¡Ì	¡Ì	¡Ì	¡Ì		¡Ì
自描述性						¡Ì	¡Ì	¡Ì		¡Ì	
结构性						¡Ì					

续表

质量特性 / 评价准则	正确性	可靠性	效率性	可使用性	完整性	可维护性	可测试性	灵活性	可移植性	可重用性	互操作性
文档完备性						¡Ì					
通用性								¡Ì	¡Ì	¡Ì	¡Ì
可扩充性								¡Ì	¡Ì		
可修改性							¡Ì	¡Ì		¡Ì	
自检性				¡Ì	¡Ì		¡Ì				
机器独立性									¡Ì	¡Ì	
软件独立性									¡Ì	¡Ì	
通信共享性											¡Ì
数据共享性											¡Ì
I/O 容量				¡Ì							
I/O 速率				¡Ì							
通信性				¡Ì							

2．ISO 的软件质量评价模型

国际标准化组织发布 ISO/IEC 9126 1991 质量特性国际标准，规定软件质量模型由 3 层组成，共定义 6 个质量特性，21 个子特性（见图 9-8）。

高层：质量特性——软件质量需求评价准则（SQRC）

中层：质量子特性——软件质量设计评价准则（SQDC）

低层：度量——软件质量度量评价准则（SQMC）

在这个三层模型中，第一层为质量特性层，作为软件质量需求评价的准则，定义了 6 个质量特性，即：功能性、可靠性、可维护性、效率、可使用性、可移植性；第二层为质量子特性层，作为软件质量设计评价的准则，推荐了 21 个子特性，如：属于判定功能性的 5 个子特性（适合性、准确性、互操作性、依从性、安全性）；属于判定可靠性的 3 个子特性（成熟性、容错性、易恢复性）；属于判定可使用性的 3 个子特性（易理解性、易学习性、易操作性）；属于判定效率的 2 个子特性（时间特性、资源特性）；属于判定可维护性的 4 个子特性（易分析性、易变更性、稳定性、易测试性）；属于判定可移植性的 4 个子特性（适应性、易安装性、遵循性、易替换性）。这些都是用于衡量质量特性的评价准则，虽不做为执行标准，但可选择其全集或子集作为评判依据；第三层为度量层，作为软件质量评估的衡量，并未统一规定标准，由各使用单位视情况而自定义。实际判定时，可以采用百分制或 5 分制，甚至分级方法以及加权来给质量子特性度量。例如：准确性＝90；容错性＝85，或：准确性＝优；容错性＝良，等等。

在软件的质量特性本身或与质量子特性之间都存在着相互的影响。例如，由于追求高效率，应尽可能采用汇编语言。但是用汇编语言编制出来的程序，其可靠性、可移植性以及可维护性都很差。所以，在进行软件质量设计时，必须全面考虑，权衡利弊，并根据质量要求，适当合理地选择和设计质量特性，并进行综合评价。

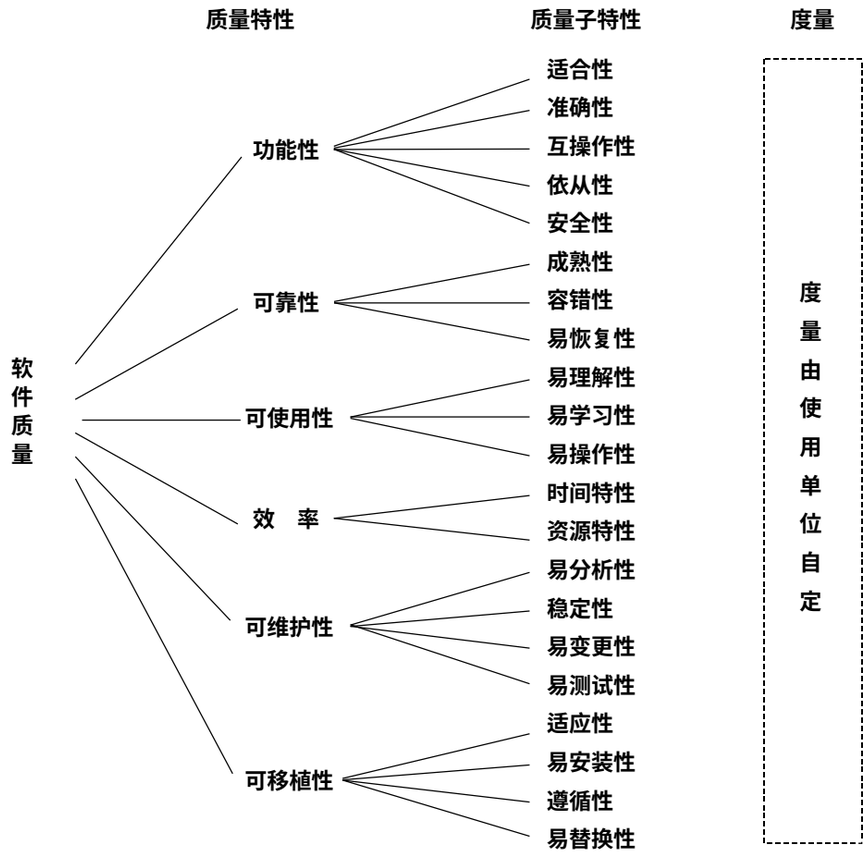


图 9-8 ISO 的软件质量评价模型

在软件质量评价模型中，为了合理表明软件产品质量的优劣，对软件的度量是十分重要的。度量（metric）定义：“对一个系统、构件或过程具有的某个给定属性的度的一个定量测量”。度量的作用就是为了有效地、定量地进行量化的控制管理。对开发过程进行度量的目的是为了改进开发过程，而对产品进行度量的目的正是为了提高产品的质量。

由于质量度量贯穿于软件工程的全过程、甚至延伸到软件交付用户使用之后。因此在软件交付之前所作的度量十分重要。这个度量提供一个定量的根据，用以判断软件设计和测试质量的好坏，以及提交软件产品的可靠度与可信度。质量度量一般针对程序的复杂性、有效的模块性和总的程序规模等。当软件交付后，度量则把注意力集中于还未发现的差错数和系统的可维护性方面。

特别强调的是，软件质量的事后度量应该可回溯以表明软件工程过程的有效性程度，评判前面开发工作完成的好与差。事后度量也是一种广泛使用的质量度量方法，包括正确性、可维护性、完整性和可使用性的测评。

① 正确性：软件能够正确地运行所要求的功能。最一般的度量是每千行代码

（KLOC）的差错数。差错指已被证实是不符合需求的缺陷。在程序交付用户进行广泛使用后由用户报告差错情况，并按标准的时间周期（典型情况是1年）进行统计计数。

② 可维护性：包括当程序中发现错误时，要能够很容易地修正错误；当程序的环境发生变化时，要能够很容易地适应环境；当用户希望变更需求时，要能够很容易地增加需求。由于还没有一种方法可以直接度量可维护性，因此必须采取间接度量。有一种简单的面向时间的度量，叫做平均变更等待时间MTTC（mean time to change），这个时间包括开始分析变更要求、设计合适的修改、实现变更并测试变更、以及把这种变更发送给所有的用户所需要的时间。一般地，一个维护性好的程序比那些维护性差的程序，具有较低的MTTC（对于相同类型的变更而言）。

③ 完整性：这个属性度量一个系统防护对其实施安全性攻击（事故的和人为的）的能力。应考虑涉及软件的所有成分：程序、数据和文档都会遭到攻击。

为了度量完整性，还需要定义两个附加的属性：危险性和安全性。危险性是指特定类型的攻击将在一给定时间内发生的概率，可以被估计或从经验数据中导出。安全性是排除特定类型攻击的概率，也可以被估计或从经验数据中导出。一个系统的完整性可定义为：

$$\text{完整性}=\sum(\text{（1－危险性）}\times\text{（1－安全性）})$$

④ 可使用性：即用户友好性。如果一个软件功能再强，但很难使用，也会被束之高阁。可使用性力图量化“用户友好性”，并依据以下4个特征进行度量：

- 为学习本软件所需要的体力上的和智力上的技能程度；
- 为达到适度有效使用本软件所需要的时间多少；
- 能够适度有效地使用本软件时在生产率方面的净增值；
- 以用户角度对系统的主观评价（可以通过问题调查表得到）。

9.6.2　软件质量体系与控制

对软件质量的重视已经成为共识，但要有效地实施软件质量控制就必须建立、健全一套完整的软件质量管理体系，在执行标准、行为规范、组织结构、实施措施等方面监督控制软件开发全过程，确保软件质量可信与可靠。

质量保证体系主要实现质量管理的组织结构、责任、规程、过程和资源。这就需要确立软件项目各部门之间进行联合与协作的机构,并明确相关的质量保证及各自的业务。遵循并执行各种级别、各种类型的软件质量标准，形成有效运作的软件质量管理体系。这不仅为软件开发者和软件用户落实了关于质量的切实需求和标准依据，而且也在组织行为上完善了软件质量管理，把握了软件质量保证。

软件质量体系提供的规范标准，应作为一种强制性的要求施行于每一项软件项目的开发。在实际工作中实施软件质量管理时，查看是否已经掌握了以下的基本原则：

- 控制及保证所有软件过程的质量；
- 实施过程控制的出发点应是预防不合格；
- 质量管理的中心任务是建立并实施文档化的质量体系；
- 坚持经常性的质量改进；

- 有效的质量体系应满足顾客和组织内部双方的需要和利益；
- 定期评价软件是否遵循合理的质量体系；
- 明确搞好质量管理关键在于领导和健全组织。

为全面开展质量管理，项目组织应使用 ISO 9000 族国际标准，以便在不同的环境下开发、实施和改进质量体系。达到目标和效果的和谐统一，有效地控制软件过程的质量。

1．ISO 9000 质量标准体系

ISO 9000 是《质量管理和质量保证标准》，以一种能够适用于任何行业（不论提供的是何种产品或服务）的一般术语，描述了质量保证的要素。要求“必须使影响产品质量的全部因素在生产全过程中始终处于受控状态”。重点强调了顾客需要的满足、职能职责的确定和评价潜在风险及利益的重要性，在建立并保持一个有效的质量体系和不断改进质量体系时，应考虑所有这些方面。ISO 9000 不仅阐明了与质量有关的基本概念以及这些概念之间的区别和相互联系；也提供了质量管理和质量保证的一族标准。任何一个打算建立和实施一个质量体系的组织，都应参照该指南标准。

2．ISO 9003 软件质量标准

ISO 9003 是“使 9001 适用于软件开发、供应及维护”的“指南”。其核心思想是“将质量融入产品之中”。因为在软件完成编码以后，无论花多大的气力用于测试，质量的提高都是有限度的。事实上，软件产品的质量取决于软件生存期所有阶段的所有活动。

为使软件产品达到质量要求，ISO 9003 规定软件开发机构应当建立质量保证体系，强调软件质量并非在软件测试中才能得到，而是形成于开发生产的全过程。还叙述了需方和供方应如何进行有组织的质量保证活动，才能得到较为满意的软件产品；规定了从双方签订开发合同到设计、实现以至维护整个软件生存期中应当实施的质量保证活动。

3．国标 GB/T 19000 族标准

我国对软件质量体系的建立也十分重视，原则上确定对 ISO 9000 质量体系等同采用，并发布了与其相适应的质量管理国家标准系列 GB/T 19000 族标准，同时积极组织实施和开展质量认证工作。

4．技术支持标准 ISO 10001～ISO 10020

这类标准主要是对实施 ISO 9000 族标准的技术指南，主要针对质量计划、技术状态、审核、测量设备的管理。

5．软件工程-产品质量 ISO/IEC 9126

这是信息技术软件产品质量特性评价及其使用指南。其中：第一部分质量模型 ISQ/IEC 9126-1:2001、第二部分外部度量 ISO/IEC TR 9126-2:2003、第三部分内部度量 ISO/IEC TR 9126-3:2003。我国 1996 年发布 GB/T16260-1996 的等同国家标准。

6．软件质量保证计划 ANSI/IEEE STOL 730-1984，983-1986

在作项目计划时需要有一个软件质量保证计划。目前较常用的是 ANSI/IEEE STOL 730-1984，983-1986 标准，其中包含以下内容：①计划目的；②参考文献；③管理；④文档；⑤标准和约定；⑥评审和审计；⑦测试；⑧问题报告和改正活动；⑨工具、技术和方法；⑩媒体控制；祐供应者控制；烧记录、收集、维护和保密；烧培训；髂风险管理。

我国针对计算机软件质量保证计划规范以及计算机软件配置管理计划规范的国家标准是 GB/T 12504-1990 及 GB/T 12505-1990。

在软件质量体系中，领导是关键，组织是保证，标准是纲领，成效是目的。因此，软件开发相关的各部门都应参与软件质量保证活动的各项工作。并通过制定质量体系保证图或制定质量保证计划来具体落实质量管理的实施步骤。在这个计划中要确定软件质量目标，明确每阶段为追求总目标所必须达到的要求，还要对质量管理进度做出安排，确定所需的人力、资源和成本等等。

详实的质量保证计划在说明各种软件人员的职责时，还要规定为了达到质量目标各种软件人员必须进行哪些活动。其次，要根据制定的质量保证体系图或质量保证计划，建立在各阶段中执行的质量评价和质量检查系统，以及有效运用质量信息的质量信息监控系统，并使其发挥作用。在实际操作中相关人员需要真正掌握以下工作要点：

- 根据计划，明确在何时、何处进行文档检查和程序检查；
- 根据计划，明确应当采集哪些数据，以及如何进行分析处理，例如，在每次评审和测试中发现的错误如何修正；
- 在计划中明确描述希望得到的质量度量；
- 在计划中明确规定在项目的哪个阶段进行评审及如何展开评审；
- 在计划中明确规定在项目的哪个阶段应当产生哪些报告和计划；
- 规定软件产品各方面测试应达到的水平及目标。

在实施软件质量控制时，可采用跟踪评测、调查研究、开会审查、书面报告等多种形式监控质量的偏差，并随时对质量存在问题的相关部门加以提示、警告、修正和备案，使质量保证计划落实到实处。

9.6.3　软件质量保证与评审

1．软件质量保证

软件质量保证是为保证软件产品和技术服务充分满足用户的质量要求而进行的有计划、有组织的活动。质量保证要面向软件的终端用户，站在用户的立场上来把握产品质量。软件质量保证为的是成就高质量的软件，而高质量的软件必须具有 3 个必备的条件：

（1）满足软件需求定义的功能及性能；

（2）所有文档符合事先确定的软件开发标准；

（3）软件的特征和属性遵循软件工程的目标和原则。

软件质量保证的标准是 70 年代首先在军方的软件开发合同中出现的，此后迅速传遍整个商业领域。其主要的组织形式是在多个机构中赋予相关人员负有保证软件质量的责任——包括软件工程小组（软件工程师、项目管理者、客户、销售人员）和供应商质量保证（supplier quality assurance, SQA）小组的成员。SQA 包括：方法和工具有效应用的规程、正式技术复审、测试策略和技术、变化控制规程、保证与标准符合的规程、以及度量和报告机制。

软件质量保证由一系列任务构成，这些任务的参与者有两种人：软件开发人员和质

量保证人员。前者负责技术工作，由软件工程师等人组成；后者负责质量保证的计划、监督、记录、分析及报告工作，由 SQA 小组担当。软件质量保证是在软件过程中的每一步都进行的“保护性活动”。在进行质量保证活动时，主要有以下 3 种策略，采用哪种策略也是相当关键的。

（1）以事后检测为重：软件产品开发完成之后进行检测，这时只能判断软件产品的质量，不能提高软件产品质量。

（2）以过程管理为重：把质量的保证工作重点放在过程管理上，对开发过程中的主要工序都要进行质量控制。

（3）以新产品开发为重：在新产品的开发设计阶段，采取强有力的措施来预先消灭由于设计原因而产生的质量隐患。

一般情况，多以监控过程的管理为主。软件工程小组的软件工程师通过采用可靠的技术方法和措施、执行计划周密的软件测试、进行正式的技术评审来考虑软件质量问题并切实保证软件质量。而 SQA 小组的职责是辅助软件工程小组得到高质量的最终产品。

“SQA 计划”为建立软件质量保证提供了一张行路图。该计划由 SQA 小组和项目组在制定项目计划时共同制定，并由相关部门复审。以此作为每个软件项目中的 SQA 活动的模板。该计划将指导和控制由软件工程小组和 SQA 小组执行的质量保证活动。在该计划中要标识以下几点。

（1）需要进行的评价——指出质量保证所覆盖的软件过程活动。

（2）需要进行的审计和复审——给出各种复审和审计方法的总览。如：软件需求复审、设计复审、软件验证和确认复审、管理复审等。

（3）项目可采用的标准——定义为获得高质量产品所能接受的工作规范的最小集合。列出所有在软件过程中采用的合适的标准和实践方法（例如，文档标准、编码标准和复审指南等）。

（4）错误报告和跟踪过程——定义错误及缺陷的报告、跟踪和解决规程。

（5）由 SQA 小组产生的文档——标识支持 SQA 活动与任务的工具和方法；给出控制变化的软件配置管理过程；定义一种合同管理方法；建立组装、保护、维护所有记录的方法。

（6）为软件项目组提供的反馈数量——标识为满足这一计划所需的培训；定义标识、评估、监控和控制风险的方法。

软件工程小组和 SQA 小组在进行软件质量保证时要根据质量保证计划进行一系列管理活动：

首先，软件工程小组要为正在进行的工作确定一个过程，并对这些软件过程进行定义描述。SQA 小组将复审该过程说明，以保证该过程与组织政策、内部软件标准、外界所订标准（如 ISO 9001）以及软件项目计划的其他部分相符。

其次，在软件过程行进中，定期复审各项软件工程活动，审计指定的软件工作产品、对其是否符合定义好的相应部分进行核实——由 SQA 小组对选出的软件产品进行复审；识别、记录和跟踪出现的偏差，对是否已经改正的状况进行核实；定期将工作结果向项目管理者报告。

最后，确保软件工作及其提交的软件产品中出现的偏差已被记录在案，并根据预定

规程进行处理——偏差可能出现在项目计划、过程描述、采用的标准或技术工作产品中，由 SQA 小组记录所有不符合的部分，并报告给高级管理者——不符合的部分将受到跟踪直至问题得到真正解决。

除进行上述活动之外，SQA 小组还需要协调变化的控制和管理，并帮助收集和分析软件度量信息。作为软件质量保证的 SQA 小组是软件质量的组织保障，其具体包括以下工作任务。

（1）在需求分析阶段对软件质量提出要求，并自顶向下逐步分解为可以识别、度量、控制的质量要素，为以后的软件开发各阶段的软件测量、定量和定性分析打好基础。

（2）研究并确定软件开发的方法和工具。

（3）组织对软件开发的各个阶段进行软件工程的正式技术评审（FTR）。

（4）制定并执行软件测试策略和测试计划。

（5）生成相关软件文档并对文档的改变进行控制。

（6）保证软件的开发过程和选用的软件开发标准相一致。

（7）建立软件质量要素的测量机制。

（8）记录 SQA 活动并生成各种 SQA 报告。

2．软件评审

在所有质量保证的活动中，技术评审是一项以提高软件质量为目的的重要活动。这种从技术角度对软件进行的评审，以发现纠正错误，控制降低成本，保证软件质量为本，把可能出现的质量问题消灭在下个软件过程发生之前。因为在软件开发的各阶段都可能产生错误，所以软件评审并不是在整个项目完毕后才开始进行，而是在软件开发的各个阶段都实施评审。以此形成软件开发过程的一系列“基线”，保证软件生命周期每个阶段的开始都干净利索。

SQA 最为重要的活动之一就是软件评审及复审。软件评审及复审作为软件过程的过滤器，用于“净化”软件过程的各项活动，能够在发现及改正错误的成本还相对较小的时候就排除错误。正式技术评（复）审或走查是一种典型的评审会议，在实践中这种形式对于及时发现错误十分有效。

1）评审目标

- 发现任何形式表现的软件功能、逻辑或实现方面的错误；
- 通过评审验证软件的需求；
- 保证软件按预先定义的标准表示；
- 已获得的软件是以统一的方式开发的；
- 使项目更容易管理。

2）评审准则

- 评审软件产品，而不是评审设计者（不能使设计者有任何压力）；
- 会场要有良好和谐的沟通气氛；
- 建立并维持议事日程（会议不能脱离主题）；
- 限制争论与反驳（评审会不是为了解决问题，而是为了发现问题）；
- 指明问题范围，而不是解决所提到的问题；
- 展示记录（最好有黑板或投影，将问题随时写出来）；

- 限制会议人数和坚持会前准备工作；
- 对每个被评审的产品要列出评审清单（帮助评审人员思考）；
- 对每个正式的技术评审分配资源和时间进度表；
- 对全部评审人员进行必要的培训或明确职责；
- 及早地对自己的评审做评审（对评审准则的评审）。

3）成立评审小组

- 组长——组长是小组的核心，由技术水平较高且没有直接参与本项工程的人担任。

组长的任务是组织和领导技术审查的全过程，如安排会议日程，分发必要的文档资料，主持审查会议，确保审查全面、公正。

- 作者——作者是被审查文档或程序的编写者。如果开发工作由一个小组集体完成，通常由技术小组负责人代表小组参加审查小组。作者的责任是汇报阶段成果，回答技术上的问题。
- 评审员——由技术专家担任。通常一个是前一阶段的技术骨干，另一个是后一阶段的技术骨干。评审员的任务是分别从各自的角度，公正客观地评价被审查的软件阶段产品。

4）软件评审的步骤

①准备；②简要介绍情况；③阅读被评审的文档（如检查表，软件开发各个阶段其检查表的内容不一样）；④开始评审；⑤作出结论；或⑥返工；⑦复审。

5）评审过程

- 召开评审会议：一般应有 3～5 人参加，会前每个参加者做好准备，评审会每次时间不要太长。
- 会议结束时必须做出以下决策之一：接受该产品，不需做修改；暂时接受该产品但需要改进；由于错误严重，拒绝接受该产品。
- 评审报告与记录：所有提出的问题都要进行记录，在评审会结束前产生一个评审问题表，另外必须完成评审纪要或评审简要报告。

总而言之，“软件质量保证就是将质量保证的管理对象和设计原则映射到适用的软件工程管理和技术空间上”。质量保证的能力是成熟的工程学科的量尺。当上述映射成功实现时，其结果就是成熟的软件工程。这也正是达到软件能力成熟度模型 CMM2 的基本要求。

9.6.4 软件配置项及其管理

软件配置管理，简称 SCM（software configuration management），也是一种贯穿于整个软件工程的保护性活动，是用于控制系统一系列变化的学科。由于软件工程项目中的变更和修改总是不可避免的，因此 SCM 活动被设计用于在团队开发中标记变更、控制变更、确保变更的正确实现、落实变更的相互报告等面向变更的一种管理。SCM 通过一系列技术、方法和手段来维护产品的历史，鉴别和定位产品独有的版本，并在产品的开发和发布阶段控制变化，通过有序管理来减少重复性工作。配置管理保证了软件开发的

质量和效率。也是为软件质量保证提供有形支持的重要措施。

现代的软件项目开发是一种多人参加、多头并行、多方合作的庞大而复杂的过程，所涉及到的各个方面人员，各个渠道的信息不仅只是在研发小组的成员之间交流沟通，而且还要在各个研发小组之间通信反馈，甚至还会存在客户与研发者之间的互访。所有这些交流信息、反馈意见都有可能导致对软件的修改，小到只是对某个源文件中的某个变量的定义改动，大到重新设计程序模块甚至是整个需求分析的修改变动。由于软件开发所固有的特征，从这样一些变动中就会形成众多的软件版本，而版本的管理就是软件配置项管理的主要内容。

软件配置管理提供了结构化的、有序化的、产品化的软件工程的方法，涵盖了软件生命周期的所有领域并影响所有文件和过程。

软件配置管理的主要目标是使软件的变更和修改可以更容易地被适应，并减少当变更必须发生时所需花费的工作量。

软件配置是指软件生命周期中各阶段所提交的各种文档和可执行代码的集合，而软件配置项 SCI（software configuration iteam）就是软件配置管理的对象，主要包括以下文件：

- 系统规格说明书；
- 软件需求规格说明书；
- 用户手册初稿；
- 详细设计规格说明书；
- 测试计划；
- 操作手册；
- 软件问题报告；
- 维护请求；
- 软件工程标准；
- 软件项目开发计划；
- 可供使用的原型；
- 总体设计规格说明书；
- 源程序清单；
- 测试报告；
- 用户手册正式稿；
- 可直接运行的目标码程序；
- 工程变更通知；
- 项目开发总结。

IEEE 标准 729-1983 就配置管理的内容进行了规范的定义。在国际标准化组织 ISO9000.3 标准中，对配置管理系统的功能也作了如下规定：

- 惟一地标识每个软件项的版本；
- 标识共同构成一完整产品的特定版本的每一软件项的版本；
- 控制由两个或多个独立工作的人员同时对一给定软件项的更新；
- 按要求在一个或多个位置对复杂产品的更新进行协调；
- 标识并跟踪所有的措施和更改；这些措施和更改是在从开始直到运行期间，由于更改请求或问题引起的。

软件工程过程中某一阶段的变更，均要引起软件配置的变更，这种变更必须严格加以控制和管理，保持修改信息，并把精确、清晰的信息传递到软件工程过程的下一步骤。

变更控制包括建立控制点和建立报告与审查制度。其中“检出”和“登入”处理实现了两个重要的变更控制要素，即存取控制和同步控制。存取控制管理各个用户存取和修改一个特定软件配置对象的权限。同步控制可用来确保由不同用户所执行的并发变更。

在实际开发中是否进行配置管理也与软件的规模有关，软件的规模越大，配置管理就越显得重要、也就愈发需要。因此，配置管理的使用取决于项目规模和复杂性以及风

险水平。

软件配置管理根据其工作范围划分可分为版本管理、问题跟踪和建立管理 3 个部分，其中版本管理是基础，主要完成以下任务：

- 建立一个项目；
- 重构任何修订版的某一项或某一文件；
- 利用加锁技术防止覆盖性重写；
- 当增加一个修订版时要求输入变更描述；
- 提供比较任意两个修订版的使用工具；
- 采用增量存储方式保存修订版；
- 提供对修订版历史和锁定状态的报告功能；
- 提供归并功能；
- 允许在任何时候重构任何版本；
- 权限的设置；
- 晋升模型的建立；
- 提供各种报告。

在对文件和项目的版本实施管理时，严格进行变更跟踪是非常重要的，否则主观与客观不一，文档与现实不同都会引起混乱而导致软件质量无据可依。跟踪主要抓住 3 个要点：

- 版本号：由配置管理维护的内部数码，用户对其没有控制权。每个文件和项目的每个版本都有一个版本号，这些版本号总是定义成惟一的、且是递增的。
- 标签：用户赋给某个项目或文件的某个版本的一个字符串标识。
- 日期/时间戳：给出了一个文件最后被修改的时间信息。

在进入软件配置管理的过程时还需要考虑下面一些问题。

（1）标识配置对象——采用什么样的方式标识和管理众多已存在的文档的各种版本，使其变更能够有效地实现又便于管理？

（2）修改控制——在软件交付用户之前和之后，如何控制变更?

（3）配置审计——谁有权批准变更和对变更安排优先级?

（4）版本控制——如何保证变更得以正确地实施而不会造成混乱?

（5）配置状况报告——利用什么办法估计变更可能引起的其他问题?

实际上，软件配置管理通过以下方法，强化软件的可靠性和质量。

（1）提供用于识别和控制文档、代码、接口、数据库的结构框架，适用于软件开发生命周期的所有阶段；

（2）全面支撑某一特定开发及维护工作方法，能够适应各种类型的需求、标准、政策、组织机构以及相关的管理策略；

（3）针对特定的基线状态、变更控制、测试、发布版本或审查活动，生成相应的管理信息和产品信息。

基线（baseline）是软件生存期中各个开发阶段末尾的特定点，又称里程碑。由正式的技术评审和批准而得到的 SCI 协议和软件配置的正式文本才能成为基线。基线的作用是使各阶段工作的划分更加明确化，使本来连续的工作在这些点上断开，以便于检验和

肯定阶段成果，例如明确规定不允许跨越里程碑修改另一阶段的文档。

基线是软件开发的里程碑，标志是有一个或多软件配置项的交付，并且这些配置项已经经过正式技术复审而获得认可（见图 9-9）。

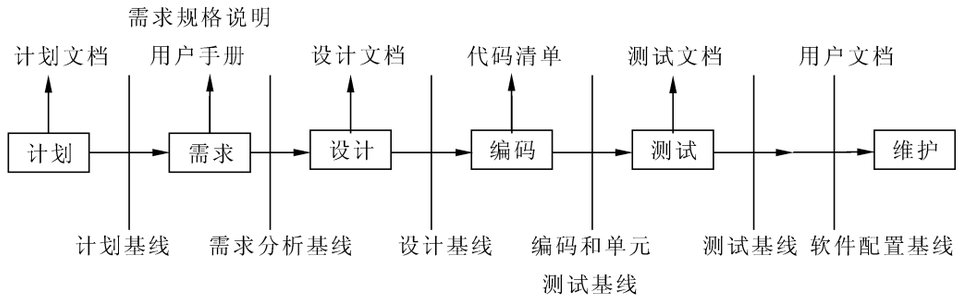


图 9-9　软件生存期基线

SCM 使软件产品和过程的变更成为受控的和可预见的，要求并在适当的工具软件支持下能够控制这样几点：①谁做的变更；②软件有什么变更；③什么时间做的变更；④为何要变更。例如：微软的 Visual SourceSafe （简称 VSS）工具软件。

9.7　软件工程标准

标准是以科学、技术和实践经验的综合成果为基础，经有关方面协商一致，由主管机构批准，以特定形式发布，共同遵守的准则和依据，具有经济性、民主性、科学性和法规性。而标准的制定、发布、出版、实施等整个活动过程则称为标准化。

随着软件工程学的发展，人们对计算机软件的认识已逐渐深入。软件工作的范围也从只是使用程序设计语言编写程序，扩展到整个软件生命周期的所有过程。同时还涉及到许多技术管理工作以及确认与验证工作。为使所有这些专门的工作能够有条不紊地展开，更有利于衡量和验证软件过程的合理性和正确性，更便于软件过程中相互的沟通与交流，更高效地定义和理解软件过程与归档资料，就必须建立相关的工程标准或规范。

软件工程标准的类型也是多方面的。包括了过程标准、产品标准、质量标准、专业标准，以及记法标准、文档标准等等。

9.7.1　软件工程标准化及其意义

一般地说，标准化是现代化大生产的必要条件，是组织专业化生产的前提，是科学管理的重要组成部分，是提高产品质量的技术保证，是合理发展种类的有效措施，是有效利用资源重要途径。

现实工作中，软件项目开发的各开发阶段及各个部门之间都存在着诸多联系和衔接活动。如何把这些错综复杂的关系协调好，需要有一系列统一的约束和规定。再者，在软件开发项目取得阶段成果或最后交付时，都需要进行阶段评审和测试验收。再进一步，

当软件投入实际运行时，其维护工作面临的问题又需回溯到开发环节查询根源。这一切预示着软件项目的管理必须具有统一的行为规范和衡量标准。无论软件生存期的哪一个环节、哪一样工作、哪一项活动都需要有章可循、有据可依、有源可查，这就需要软件过程的工程化和标准化的支持和保障。只要坚持有效地执行软件工程标准化，则可获得良好的管理效果。

（1）在统一标准之下，能够保证软件的可靠性、可维护性和可移植性（这表明软件工程标准化可提高软件产品的质量）；

（2）在标准规程指导下软件人员更能充分发挥技术水平，提高软件的生产率；

（3）有了标准可使软件人员之间的交流与通信提高效率，避免差错和误解；

（4）遵循规范的标准有利于软件管理，增强软件过程的有效性；

（5）实施标准化管理能够降低软件产品的成本和运行维护成本；

（6）全面依照规范的标准有利于缩短软件开发周期。

随着人们对计算机软件的认识不断加深，以及软件行业的迅猛发展，对软件项目的管理提出了更新和更高的要求。在软件工程学的指导下，软件已走出了个人或小作坊的圈子，形成了应用市场广阔、开发前景远大的朝阳产业。在深入探讨软件开发过程如何规范化管理的同时，也相继推出了一系列不同类型、不同层次的软件工程标准，涵盖了从软件项目的形成、需求分析、软件设计、编码实现、测试调试、安装检验、运行维护，直到软件淘汰整个软件生存周期。同时还涉及诸多技术管理工作（如过程管理、资源管理、质量管理、产品管理）以及确认与验证工作（如评审和审核、产品分析、软件测试、管理认证等）。这为程序设计的标准化以及软件过程的标准化、规范化起到了积极的推动作用。

9.7.2 软件工程标准的类型与层次

我国标准按级别可分为：国家标准、行业标准、地方标准和企业标准四级；按标准化工作应用范围可分为：技术标准和管理标准两大类；按标准执行程度又可分为：强制性标准和推荐性标准两种。

对于软件工程标准的分类，我国国家标准 GB／T 15538－1995《软件工程标准分类法》已给出了定义，涵盖了软件项目的过程管理、产品管理、资源管理，以及确认与验证等方面。

软件工程标准的类型主要分为以下几类。

（1）过程标准——涉及方法、技术、度量。

（2）产品标准——涉及需求、设计、部件、描述、计划、报告。

（3）专业标准——涉及职别、道德、准则、认证、特许、课程。

（4）记法标准——涉及术语、表示法、语言。

标准的类型反映出软件工程在各个方面都已有统一的标准及规范，在进行软件项目开发时必须严格遵守，以确保软件过程的顺利实施和提高软件产品的质量。由于软件标准涉及面较广，来自各个国家及组织的不同机构所制定的软件工程标准也就代表了不同层面的规范和要求，其标准适用的范围也有所不同，大致可分为 5 个级别。

1．国际标准

由国际联合机构制定并公布，提供给各国参考的标准。

（1）国际标准化组织 ISO（international standards organization）负责除电工、电子领域之外的所有其他领域的标准化活动。

（2）国际电工委员会 IEC（international electrotechnical commission）主要负责电工、电子领域的标准化活动。

国际上两大重要的标准化组织，具有广泛的代表性和权威性，所公布的各种标准都有较大的影响力，甚至也当作（或等同采用）为某些国家的国家标准。由该组织发布的各种标准文档通常冠有 ISO 或 ISO/IEC 字样。ISO 下设 180 多个标准化技术委员会，已制定标准 10000 余项；IEC 下设 80 多个标准化技术委员会，已制定标准 4000 余项。

针对信息行业，国际标准化组织建立了“计算机与信息处理技术委员会”，简称 ISO／TC97，专门负责与计算机有关的标准化工作。至 1999 年底，ISO/IEC JTC1 已制定出近 40 项软件工程国际标准，如：

① ISO 8631－86 Information processing_program constructs and conventions for their representation《信息处理—程序构造及其表示法的约定》

② ISO/IEC 15504 《信息技术—软件过程评估》国际标准

这是 1998 年 SPICE（software process improvement and capability dtermination）项目组织发表的用于确定自我能力改造和进行软件供应商能力认定的国际标准。

该标准覆盖了过程评估、过程改进和过程能力确认等 9 个方面的指南和模型。

目前，有关组织正在就 15504 标准和 CMMI（cmm integration 软件能力成熟度集成模型）的兼容性问题进行探讨，很有可能 ISO/IEC 15504 标准最终会和 CMMI 兼容，并同时支持和兼容软件过程能力成熟度模型 CMM（capacity maturity model）。

③ ISO/IEC 12207—1995《软件生存周期过程标准》

该标准为软件产业确立了一个软件生存周期过程的通用框架，说明需方在获得一个含软件的系统、一个单独的软件和一项软件服务时，以及供方在供给、开发、操作和维护软件产品时，所涉及的各种必要的过程、以及各过程包含的活动和各活动包含的任务。

该标准将软件生存期的各种活动归入 3 类 17 个过程及其他重要内容。还为软件组织规定了一个用于定义、控制和改进其软件生存周期过程的标准过程。

软件过程一般划分为：

- 软件工程过程：软件开发和生产的过程；如：需求分析、设计、编码、测试等过程。
- 软件管理过程：对软件开发和生产的过程进行管理的过程；如：项目策划过程、跟踪监控过程、质量保证过程。
- 软件支持过程：对软件开发和生产的过程进行支持的过程；如：评审过程、培训过程、质量过程。

2．国家标准

由政府或国家级的机构制定或批准，适用于全国范围的标准。

（1）中国国家标准（简称“国标”GB）

由中华人民共和国国家技术监督局公布实施的标准。我国制定和推行标准化工作的

总原则是向国际标准靠拢，对于能够在中国适用的标准一律按等同采用的原则（使得我国标准与国际标准的技术内容完全相同，仅稍做编辑性修改）。至 1997 年，我国已制定信息技术类国家标准 400 余项，制定软件工程标准 20 多项，为软件开发工程化提供了有效的开发工具和环境，及先进的开发方法和科学管理措施，对提高我国软件生产率和软件质量起到十分重要的作用。这些标准可分为 4 类（见表 9-16）：①基础标准；②开发标准；③文档标准；④管理标准。

表 9-16　国际标准与国家标准对应表（部分）

分类	标准号		标准名称
	国际标准	中国国家标准	
基础标准	ISO 5807—1985	GB 1526—89	信息处理——数据流程图、程序流程图、系统流程图、程序网络图和系统资源图的文件编辑符号及约定
		GB/T11457—1995	软件工程术语
	ISO/DIS 12620—1996	GB/T 16786—1997	术语工作　计算机应用　数据类目
	ISO/DIS 1087—2—2:96	GB/T 17532—1998	术语工作　计算机应用　词汇
		GB/T15189—1994	DOS 中文信息处理系统接口规范
	ANSI/IEEE 1002	GB/T15538—1995	软件工程标准分类法
	ISO 8631	GB 13502—92	信息处理——程序构造及其表示法的约定
	ISO 5806	GB/T15535—1995	信息处理——单命中判定表的规范
	ISO 8790	GB/T14085—1993	信息处理系统——计算机系统配置图符号及其约定
		GB/T13702—1992	计算机软件分类与代码
		GB/T 18221—2000	信息技术 程序设计语言 环境与系统软件接口 独立于语言的数据类型
		GB/T13502—1992	信息处理 程序构造及其表示的约定
开发标准		GB 8566—88	软件开发规范
		GB/T15532—95	计算机软件单元测试
		GB/T15853—1995	软件支持环境
	ISO 6593—1985	GB/T15697—1995	信息处理——按记录组处理顺序文卷的程序流程
		GB/T14079—1993	软件维护指南
		GB/T18491.1—2001	信息技术 软件测量 功能规模测量 第 1 部分: 概念定义
		GB/T 18492—2001	信息技术 系统及软件完整性级别
		GB/T 18493—2001	信息技术 软件生存周期过程指南
		GB/T 19000—1994	质量管理和质量保证标准
		GB/T 19001—1994	质量体系 设计、开发、生产、安装和服务的质量保证模式
		GB/T 19002—1994	质量体系 生产、安装和服务的质量保证模式
		GB/T 19003—1994	质量体系 最终检验和试验的质量保证模式
		GB/T 19004—1994	质量管理和质量体系要素
		GB/T 8566—2001	信息技术 软件生存周期过程

续表

分类	标准号		标准名称
	国际标准	中国国家标准	
文档标准		GB/T 16680—1996	软件文档管理指南
		GB/T 8567—1988	计算机软件产品开发文件编制指南
	ANSI/IEEE 829	GB/T 9385—1988	计算机软件需求说明编制指南
	ANSI/IEEE 830	GB/T 9386—1988	计算机软件测试文件编制规范
		GB/T 16704—1996	计算机软件著作权登记文件格式
管理标准	IEEE 828	GB/T 12505—1990	计算机软件配置管理计划规范
	ISO/IEC 9126—91	GB/T 16260—96	信息技术　软件产品评价　质量特性及其使用指南
	ANSI/IEEE 730	GB/T 12504—1990	计算机软件质量保证计划规范
		GB/T 14394—1993	计算机软件可靠性和可维护性管理
		GB/T 17544—1998	信息技术 软件包 质量要求和测试
		GB/T 15532—1995	计算机软件单元测试
	ISO 9000—3—93	GB/T 19000.3—94	质量管理和质量保证标准　第 3 部分：GB/T 19001—ISO 9001 在软件开发、供应和维护中的使用指南
		GB/T 13423—1992	工业控制用软件评定准则

（2）美国国家标准协会 ANSI（american national standards institute）：这是美国一些民间标准化组织的领导机构。

（3）美国商务部国家标准局联邦信息处理标准：FIPS（NBS）[federal information processing standards（national bureau of standards）]。

（4）英国国家标准：BS（British standard）。

（5）德国标准协会：DIN（Deutsches institute für nor- mung）。

（6）日本工业标准：JIS（Japanese industrial standard）。

3．行业标准

由行业机构、学术团体或国防机构制定，适用于某个业务领域的标准。

（1）石油天然气行业标准 SY；中国国防及军队使用的标准 GJB。

例如：SY/T 6227—1996《石油工业数据库设计规范》；

SY/T 5759—1995《石油计算机网络及其节点名称代码规定与 IP 地址分配方式》；

GJB 437—88《军用软件开发规范》。

近年来中国政府的许多部门也都制定和公布了一些适合于本部门软件标准化工作需要的规范。这些规范大都参考了国际标准或国家标准。

（2）美国电气与电子工程师学会 IEEE（institute of electrical and electronics engineers）：该学会公布的标准常冠有 ANSI 的字头。下设一个软件标准分技术委员会（SESS），负责软件标准化活动。

例如：ANSI／IEEE Str 828—1983《软件配置管理计划标准》。

（3）美国国防部标准 DOD_STD（department of defense _standards）：适用于美国国防部门。

（4）MIL_S（military_standard）——美国军用标准，适用于美军内部。

4．企业标准

一些大型企业或公司，由于软件工程工作的需要，制定适用于本部门的规范。

例如：BZ08CSTC 软件产品登记测试规范；

BZ02CSTC 应用软件产品测试规范。

5．项目规范

由某一科研生产项目组织制定，为该项任务专用的软件工程规范。

例如：计算机集成制造系统（863/CIMS）的软件工程规范。

9.7.3 软件质量标准与认证

质量管理和质量保证标准 ISO 9000 族是由 ISO/TC176（TC176：品质管理和品质保证技术委员会）制定的国际标准，族内一共有 17 个标准。我国对此一方面发布了与其相应的质量管理国家标准系列 GB/T 19000。另一方面积极组织实施和开展质量认证工作。计算机软件行业也和其他领域一样遵循此质量标准体系。

ISO 9000 是关于质量管理和质量保证以及质量体系要素的系列标准，客观地对生产者（也称供方）提出了全面的质量管理要求及质量管理办法，并且还规定了消费者（也称需方）的管理职责，因此得到双方的普遍认同。

1．基本思想

（1）强调质量并不是在产品检验中得到的，而是在生产的全过程中形成的。

（2）为确保产品质量，ISO 9000 要求“在生产的全过程中，影响产品质量的所有因素都要始终处于受控状态”。

（3）可以用 ISO 9000 标准证实“企业具有持续地提供符合要求的产品的能力”。

（4）还可以用 ISO 9000 标准来“持续地改进质量”。

2．适用领域

（1）硬件：具有特定形状的产品，如机械、电子产品，包括计算机硬件。

（2）软件：通过媒体表达的信息所构成的智力产品，包括计算机软件。

（3）流程性材料：将原料转化为某一特定状态的产品。

（4）服务：为满足客户需求的更为广泛的活动。

3．主要内容

ISO 9000 族标准是指由 ISO/TC176 技术委员会制定的所有国际标准。可分为 5 个部分：①术语标准；②使用或实施指南标准；③质量保证标准；④质量管理标准；⑤支持性技术指南。下面是 ISO 9000 族国际标准：

ISO 9000　质量管理和质量保证标准——选择和使用的导则；

ISO 9001　质量体系——设计/开发、生产、安装和服务中的质量保证模式；

ISO 9002　质量体系——生产和安装中的质量保证模式；

ISO 9003　质量体系——最终检验和测试中的质量保证模式；

ISO 9004　质量管理和质量体系要素——导则。

我国等同采用的相应定义为：

GB/T 19000.1—1994　质量管理和质量保证标准　第1部分：选择和使用指南；

GB/T 19000.2—1994　质量管理和质量保证标准；

GB/T 19000.3—1994　质量管理和质量保证标准—第3部分—在软件开发、供应和维护中的使用指南；

GB/T 19001—1994　质量体系设计、开发、生产、安装和服务的质量保证模式；

GB/T 19002—1994　质量体系生产、安装和服务的质量保证模式；

GB/T 19002.2—1994　质量管理和质量体系要素—第2部分—服务指南；

GB/T 19003—1994　质量体系最终检验和试验的质量保证模式；

GB/T 19004.1—1994　质量管理和质量体系要素　第1部分：指南；

GB/T 19004.2—1994　质量管理和质量体系要素　第2部分：服务指南；

GB/T 19004.3—1994　质量管理和质量体系要素　第3部分：流程性材料指南；

GB/T 19004.4—1994　支持工具和技术；

GB/T 19021.1—1993　质量体系审核指南；

GB/T 12504-1990　计算机软件质量保证计划规范；

GB/T 12505-1990　计算机软件配置管理计划规范。

4．ISO 9000—3标准

由于ISO 9000系列标准原本是为制造硬件产品而制定的标准，不能直接用于软件制作。曾经试图将9001改写用于软件开发方面，但效果不佳。最终以ISO 9000系列标准的追加形式，另行制定出ISO 9000—3标准，由此ISO 9000—3成为“使9001适用于软件开发、供应及维护”的“指南”。

5．质量管理万字头标准

这类标准主要是实施ISO 9000族标准的技术指南。目前正式颁布的有以下几项标准：

ISO 10005∶1995　质量管理——质量计划指南；

ISO 10007∶1995　质量管理——技术状态管理指南；

ISO 10011—1∶1990　质量体系审核指南第1部分∶审核；

ISO 10011—2∶1991　质量体系审核指南第2部分∶审核员的资格条件；

ISO 10011—3∶1991　质量体系审核指南第3部分∶审核工作管理；

ISO 10012—1∶1991　测量设备的质量保证要求第1部分∶测量、试验设备的管理；

ISO 10013∶1995　质量手册编制指南。

6．质量体系认证

认证的定义是∶“由可以充分信任的第三方证实某一经鉴定的产品或服务符合特定标准或规范性文件的活动。”

软件质量保证标准所包括的3个标准在内容上ISO 9001完全包含了ISO 9002；ISO 9002又完全包含了ISO 9003。这代表着在具体情况下对供方质量体系要求的3种不同模式，反映了不同复杂程度的产品所要求的质量保证能力的不同，是质量体系认证的依据，所有软件企业申请产品质量认证还是质量体系认证，首先都要贯彻实施GB/T 19000—ISO 9000族标准，建立完善的质量体系。而后认证机构依据以上3个标准中的一个对企业进行审核。也就是软件企业通过质量体系认证只有3种∶ISO 9001体系认证，ISO 9002

体系认证和 ISO 9003 体系认证。

特别地，国际标准 ISO 9003 是 ISO 9000 质量体系认证中关于计算机软件质量管理和质量保证标准的部分，从管理职责、质量体系、合同评审、设计控制、文件和资料控制、采购、顾客提供产品的控制、产品标识和可追溯性、过程控制、检验和试验、检验/测量和试验设备的控制、检验和试验状态、不合格品的控制、纠正和预防措施、搬运/贮存/包装/防护和交付、质量记录的控制、内部质量审核、培训、服务、统计系统等方面对软件质量进行了要求。

当然，软件过程能力成熟度 CMM 的认证同样可以从另一个方面证实软件企业的质量水平及其他方面的能力。

9.7.4 软件文档标准化

软件工程不仅以工程的角度阐述了软件项目开发的过程、方法与工具，同时还明确指出：软件是程序、数据及其文档的完整集合。从而明确强调软件文档（documentation）的重要地位，充分肯定软件文档的重要性。

文档通常以书面或图表的方式对活动、需求、过程及结果进行描述、定义、规定、报告或认证。文档具有很好的可存性、可读性和可依性，能够指导并规范开发过程的行为，同时也能记录和保持开发活动的情况与结果；有利于提高软件开发过程的能见度、增强监控及评审的执行力度；是相互沟通交流、提高开发效率的有力支持。

软件文档的规范编制，并不仅仅是写在纸上的条文和记录，而应该是软件过程的纲领，开发活动的指导，项目行进的基线，产品交付的档案。认真严格地执行这一过程将对软件项目的管理起到很好的保障作用。

（1）使软件开发过程能够从逻辑上被回溯。

（2）检查软件开发进度和测试软件质量有依据。

（3）软件开发过程人人心中有数，事事有章可依。

（4）方便协作交流沟通，大大提高工作效率。

（5）阶段评审核准的里程碑，软件生命周期标志性成果。

（6）推广软件产品，扩大影响，争取潜在用户的法宝。

从管理的角度来说，文档是软件开发规范的体现和指南。在使用工程化的原理和方法来指导软件的开发和维护时，应当充分注意软件文档的编制和管理工作。实际上，能够按照规范要求生成一整套软件文档的过程，也就是按照软件开发规范完成一个软件开发的过程。两者应该是和谐统一、相得益彰的。

从文档的形式上看，软件文档大致可分为两类：一类是工作表格，主要是开发过程中填写和描绘的各种图表；另一类资料文档，主要是必须编写的各种技术资料或技术管理资料。在进行软件文档的编制中，可以用自然语言，或特别设计的形式语言，或介于两者之间的半形式语言（结构化语言），以及各类图形、表格来编制文档。当然，文档的组织形式可以是书写的，也可以在计算机辅助工具中产生，而后打印输出，或直接在机器上浏览查看。但一定要十分注意文档的保存与保管，以免有用的信息被破坏或丢失。

按照软件文档产生和使用的范围，可将其分为 3 类。

（1）开发文档——属于技术指导性文档。为软件开发人员指明实施开发过程的技术相关事宜。其中包括：可行性研究报告、项目开发计划、软件需求说明书、数据要求说明书、系统设计说明书、详细设计说明书等。

（2）管理文档——属于过程监控性文档。对软件开发过程进行有效控制和管理的规定与标准。其中包括：项目开发计划、开发进度月报、测试计划、测试报告等。

（3）用户文档——属于产品应用性文档。使软件产品能够正确使用、正常工作、正当维护的指导要领。其中包括：项目开发总结报告、用户手册、操作手册、维护修改建议等。

软件文档来源于实际的软件项目开发，但文档的编制应该符合相关的国际或国家标准。早在1988年，国家标准局就已发布了《计算机软件开发规范》以及《软件产品开发文件编制指南》等标准，这使得编制软件文档的工作有章可依，而不能任意各行其说。

作为软件开发的准则和规程。国家标准的相关文件给出了在软件项目开发中基本具有并提交的14种文档，及编制这些文档的核心内容提纲。主要内容提纲如下（具体内容提纲参见GB/T 8567—1988《计算机软件产品开发文件编制指南》）。

（1）可行性研究报告——阐述该软件项目在技术、经济和社会各方面的可行性；评述为了合理地达到开发目标而可能选择的各种方案；着重说明并论证所选定的方案。

（2）项目开发计划——描述在开发过程中各项工作的负责人员、开发进度、所需经费预算、所需软、硬件条件等问题作出的安排，以便根据本计划开展并检查本项目的开发工作。

（3）开发进度月报——列出本月内进行的各项主要活动，并且说明本月内遇到的重要事件。

（4）软件需求说明书——详细的说明目标软件的功能、性能、用户界面及运行环境等。使用户和软件开发者双方对该软件的初始规定有一个共同的理解。奠定整个开发工作的基础。

（5）数据要求说明书——给出数据逻辑描述和数据采集的各项要求，分析、制定数据结构，为生成和维护系统的数据文件做好准备。

（6）数据库设计说明书——对概念结构设计、逻辑结构设计、物理结构设计、数据字典设计、安全保密设计作出说明。向准备从事此数据库的生成、测试、维护人员提供专门的指导。

（7）系统设计说明书从总体上描述软件构架，给出目标系统的高层逻辑模型。

（8）详细设计说明书——着重描述每一个模块的实现算法、逻辑流程等。

（9）测试计划——针对组装测试和确认测试，为组织测试的活动制定计划。

（10）测试分析报告——说明测试计划的执行情况。对测试结果加以分析，并提出测试的结论性意见。

（11）操作手册——为操作人员提供软件各种运行情况的有关要点，特别是操作方法细节。

（12）用户手册——详细描述软件的功能、性能和用户界面，使用户知道如何使用该软件。

（13）模块开发卷宗——扼要描述诸模块的功能说明、设计说明和测试说明及复审的结论。

（14）项目开发总结报告——与项目实施计划对照，总结汇报实际执行的情况（资源、成本、人力的投入及进度、成果等）。对本项目作一综合评价。

以上各种文档属于软件生命周期中应该产生和得到的阶段性成果，有着各自的时间顺序。其中，有的仅反映某一个阶段的工作，有的则需跨越多个阶段。依照国标 GB/T 8567—1988《计算机软件产品开发文件编制指南》，在软件生存周期各阶段中部署编制相关文档的时序如表 9-17 所示。

表 9-17　软件生存周期各阶段中文档部署时序

阶段 / 文件	可行性研究与计划阶段	需求分析阶段	设计阶段	实现阶段	测试阶段	运行与维护阶段
数据需求说明书	▬					
项目开发计划		▬				
	▬					
软件需求说明书		▬				
数据需求说明书						
		▬				
测试计划						
		▬				
概要设计说明书			▬			
详细设计说明书						
数据库设计说明书			▬			
模块开发卷宗				▬		
用户手册		▬				
操作手册			▬	▬		
测试分析报告					▬	
开发进度月报						
项目开发总结					▬	

当软件产品最终交付用户时，相关的文档也一同由用户归档。这时产品如何使用和维护在很大程度上要依赖技术文档的说明，因此技术文档应该十分的准确可靠。这意味着要向用户提供高质量、高可信的文档，那么在编制用户相关文档时一定要注意以下问题。

（1）文档读者的针对性：管理文档主要面向管理人员，用户文档主要面向用户，维护文档面向技术人员。

（2）文档行文的精确性：同一课题几个文档的内容应当是统一与一致，不能出现多义性的描述。

（3）文档表达的清晰性：文档编写应力求简明扼要，尽量配以适当的图或表，以增强其清晰性。

（4）文档内容的完整性：任何一个文档都应当是完整的、独立的，可以自成体系。有时需要以内容重复来保证各自独立与完整。

（5）文档编排的灵活性：根据具体软件开发项目的规模和复杂程度，决定编制的文档种类。不要一律看待。

软件项目管理部门应该根据本单位承担的应用软件的专业领域和本单位的管理能力，制定一个对文档编制要求的实施规定。或根据任务的规模、复杂性、对该软件的开发过程及运行环境所需详细程度的判断，确定文档的详细程度。特别是在文档编制完成后，综合检查所有文档是否能够说明和解释清楚下列问题：

- 哪些需求要被满足（what）？即回答“做什么？”。
- 软件在什么环境中实现，所需信息从哪里来（where）？即回答“在何处？”。
- 开发时间如何安排（when）？即回答“何时干？”。
- 开发（或维护）工作打算由谁来做（who）？即回答“谁来干？”。
- 需求应如何实现（how）？即回答“怎样干？”。
- 为什么要进行这些软件开发或维护修改工作（why）？即回答“为何干？”。

文档相伴整个软件生存期，甚至作为最终成果之一。那么文档的管理与维护就成为不可或缺和不可松懈的工作。因为在项目的进程中，各类文件会不断地编制生成、适时修改或补充完善。为了保证软件产品的完整性、统一性，项目管理组织应设一名专职（或兼职）的文档保管员，负责集中管理本项目已有的文档。一般要备份两套主文本：一套可按规定手续办理借阅，一套作为原始资料归档。所有的复本必须与主文本保持一致，在作必要的修改时，也应先修改主文本。开发成员个人只保持与自己工作相关的部分文档。当新文档取代了旧文档时，文档保管员应及时注销旧文档。在文档内容有更动时，也应随时修订主文本，使其及时反映更新了的内容。

关键的是项目开发结束时，文档保管员应收回开发成员的个人文档。若发现个人文档与主文本有差别时，应核实原因，及时解决，或是落实修正主文档，或是查清备份文档，以确保文档内容的一致性。但主文本的修改必须特别谨慎。修改前一定要充分估计修改可能带来的影响，并按照标准过程：提议、评议、审核、批准和实施等步骤加以严格的控制。

总之，文档本身就是软件产品的一部分。从软件工程的角度要求，没有文档的软件，不成其为软件，更谈不上软件产品。软件文档的编制在软件开发工作中占有突出的地位和相当的分量，对于充分发挥软件产品的效益有着重要意义。

本 章 小 结

软件项目管理是软件工程的保护性活动，支持并确保着软件开发能够在规定的时间（按时）、规定的费用（按量）、规定的质量（按质）下完成各项任务。软件项目管理始于任何技术活动之前，止于软件产品交付使用之后，贯穿了整个软件生命周期。其间经

历了：1.项目范围确定；2.项目成本估算；3.项目计划制定；4.项目进度控制；5.项目人员组织；6.项目风险规避；7.项目质量保证；8.项目标准文档等关键环节，最终得以交付一个合格的软件产品，完满地结束软件项目管理工作。

由此可见：软件项目管理揽括了范围管理、成本管理、人员管理、进度管理、风险管理、质量管理、配置管理和文档管理等方面的内容。其目标就是确保软件开发能按时间、按预算、按质量地完成任务。本章也正是从以上这些方面，展开了详细的分析与描述。

在软件项目管理过程中第一个关键的活动是制定项目计划，通过任务分解结构（WBS）确定该软件的范围以此来估算项目成本与时间进度还有人员安排。从而推出了一系列的管理计划，用以指导软件项目管理过程。

在所有软件项目管理要素中最关键的要素是人，选用一种合适的结构将人员组织起来，以达到最高的生产率。采取项目管理控制时通常包括进度控制、人员控制、经费控制和质量控制。

风险是无时无处无事不存在的，识别并分类风险有利于估计风险、监控风险进而规避风险；项目质量保证由质量体系与监督控制支持，其中阶段评审、软件配置管理也是有力的保障。

软件工程标准的类型是多方面的。将其归类在：过程标准、产品标准、专业标准以及记法标准之中。最为关键的要遵循质量管理和质量保证标准。按级别分有：国际标准、国家标准、行业标准、企业规范及项目（课题）规范 5 个等级。

文档编制是体现项目管理水平的标尺，也是软件产品的重要组成部分，一个合格的软件产品不能没有合格的文档。因此，文档的编制必须执行并且要保质保量。

软件项目计划与管理为的就是使软件项目能够按照预定的成本、进度、质量顺利完成，由此而展开了对成本、人员、进度、风险、质量、配置、文档等进行分析和管理的一系列活动。其效果就是让软件项目尤其是大型软件项目的整个生命周期（从分析、设计、编码到测试、维护全过程）都能在管理者的控制之下，确保了软件项目的顺利进行。

习 题 9

9.1 软件质量的含义是什么？

9.2 影响软件质量的因素分哪两大类？

9.3 软件质量的各种特性怎样度量？

9.4 什么是软件质量保证的策略？现代软件质量保证的策略是什么？

9.5 软件质量保证活动有哪 7 个主要方面？如缺少一两个方面行不行？是否还有没有考虑到的方面？

9.6 为什么说软件技术评审是软件质量保证的一个最基本的活动？

9.7 为什么说软件质量保证中最重要的两个方面是：对软件质量保证活动和软件配置的审计；软件质量保证计划的制定和标准的采用？

9.8 为什么说软件测试是软件开发中不可缺少的重要的医患，但不是软件质量保证的“安全网”？

第10章

软件过程能力成熟度模型CMM[22~26]

随着计算机软件技术在信息现代化建设的广泛应用和迅速发展，计算机已渗入社会生活的各方面，软件质量的好坏直接影响到国家的经济效益和生产率，特别是一些国家重点软件开发机构和重点科研项目。由于软件复杂性的提高，如何有序地管理和控制软件开发过程，提高生产效率，确保生产出符合预算和进度要求的高可靠性和可用性软件，已成为各大型企业和各软件开发机构关注的焦点。

目前，国内许多承担软件开发工作的软件机构都开展了质量管理体系的认证工作，引入了国际通行的质量管理体系质量保证模式——ISO 9001（或ISO 9002），但不少软件开发机构在实施ISO 9001时都不同程度地感到ISO 9001（加上ISO 9000—3）与实际活动不是那么相适应，于是国外正在迅速升温的专门针对软件开发机构的软件质量保证模型悄然进入我国一些软件开发机构和有关研究单位，这就是能力成熟度模型（the capability maturity model，CMM）。

10.1 软件过程与软件过程成熟度

软件过程成熟度的起源可以追溯到已经存在了近70年的产品质量管理原则。质量运动最初以产品检查开始，逐渐发展到以统计的方法进行过程控制。卡内基梅隆大学的软件工程研究所（software engineering institute，SEI）在美国国防部的资助下开发了CMM。

10.1.1 软件过程

软件过程是人们用以开发和维护软件及其相关的软件工作产品（如项目计划、设计文档、编程、测试、用户手册等）的一系列活动，包括软件工程活动和软件管理活动，其中必然会涉及有关的方法和技术等。

根据国际标准化组织（ISO）制订的过程标准《ISO/IEC 12207 信息技术－软件生存期过程》，软件过程包括5个基本过程、8个支持过程和4个

组织过程，每个过程包括若干项活动，每项活动又包括若干条具体任务。在具体的项目实施过程中，可以对这些过程进行裁剪和扩充，并将每个必要的过程、活动和任务细化到可操作的程度。

软件过程把软件生产的三大要素（人、资源和技术）有机地结合起来，形成一个高效运作的整体。事实上，由于软件产品是一种富于创造性、创新性的产品，因此，软件开发过程是复杂的思维过程，很大程度上依赖于开发人员高度的智力投入。其生产方式没有固定的模式，产品无形，难以度量控制，工作量难以估计，进度难以衡量，质量难以保证，成本高，修改、维护工作繁重，开发人员的创造性与产品规范化测试要求是一对需要不断协调的矛盾，并直接影响软件产品的质量。同时软件规模和复杂度成指数激增。对于大型系统，成百上千的人共同开发一个系统，需要协调、协同、合作、组织。其次，由于软件行业是高速发展的行业，不断有新的技术、方法涌现，要求软件过程对技术环境具有适应性。用户的需求是不确定、不断变化的，要求软件过程对用户需求具有适应性。因此，软件过程是动态的、变化的、错综复杂的特殊过程。正是软件过程的这些特性，使得软件过程的管理与改进十分困难，软件质量的保证也相当艰难。

10.1.2 软件过程成熟度

软件过程成熟度（software process maturity）：软件过程行为可被定义、预测和控制并被持续性提高的程度，主要用来表明不同项目所遵循的软件过程的一致性。

软件过程成熟度是指一个软件过程被明确定义、管理、度量和控制的有效程度。成熟意味着软件过程能力持续改善的过程，成熟度代表软件过程能力改善的潜力。过程的改善不能跳跃式进行。成熟度等级用来描述某一成熟度等级上的组织特征，每一等级都为下一等级奠定基础，过程的潜力只有在一定的基础之上才能够被充分发挥。例如：一般看来，规划一个工程过程要比规划管理过程更加重要，但实际上如果没有管理的规定，工程过程很容易成为进度和成本的牺牲品。另外，成熟级别的改善需要强有力的管理支持。成熟级别的改善包括管理者和软件从业者基本工作方式的改变，组织成员依据建立的软件过程标准执行并监控软件过程，一旦来自组织和管理上的障碍被清除后，有关技术和过程的改善进程能迅速推进。

10.1.3 软件过程改进框架

软件过程改进的战略应建立在当前软件过程改进环境下的一个整体框架之上，这个整体框架应标识出软件过程改进中必须包括的关键领域。对于创建持续的软件过程改进而言，框架应包含 4 方面的内容：软件过程架构、软件过程规划图、软件过程评估、软件过程改进计划。

1．软件过程架构

为支持过程必须具有两种类型的架构。一个是组织及管理方面的架构，这包括角色与职责；另一个是技术方面的架构，包括技术工具与设备。所有这些都为与过程相关的

活动提供支持，并且维持过程的持续改进活动。

组织及管理方面的架构包括用于建立过程监控和对过程执行实施强制措施的角色与职责，主要包括：负责人角色、管理角色、协调角色和改进团队的角色。

软件过程技术方面的架构包括用于支持软件工程过程组与过程改进团队的技术构架、计算机设备以及工具。该架构应包括全局层次的、项目/团队一级的与过程相关的内容。项目一级的设备应当具有一定的灵活度，可以让不同的项目根据自身的情况选用合适的技术过程支持环境。一个高效灵活的技术过程架构是进行有效的过程改进和过程维护的基础。

2．软件过程改进规划图

软件过程规划图指明了实现软件过程所要经历的阶段，以及为实现这些阶段目标所必经的关键点。软件过程改进规划图可以是公开发表的过程标准，例如能力成熟度模型（CMM）或者是 ISO/IEC15504 成熟度等级，也可以是在上述标准之上根据组织自身情况进行修改后的版本。

3．软件过程评估

通常软件过程评估是根据软件过程规划图而进行的。评估的结果需要进行分析，从而找出当前过程的长处与短处，以便为更好地进行改进提供建议。正是通过评估才能使人们沿着过程改进规划图不断提高过程的成熟度。

评估是起点，不是终点。为了实现软件过程改进，应将评估转化为一个相应的行动计划。制定出行动计划后，应该建立相应的机制，从而使过程制度化。

4．软件过程改进计划

为进行软件过程改进，根据评估所发现的问题，有针对性地制定出相应的改进方案。通过这些改进方案，将会提高现有的软件过程水平，而过程的改进又会提高过程的规范化以及过程的有效性。

构成框架的这四个部分是相互关联的，任何一个软件过程改进的策略都应涵盖这四个部分，缺一不可，否则将会导致软件过程程序中的冲突。从相互关联的角度来看，这是一种循环关系。一个完整的循环首先从对过程架构的评估开始。评估是进行改革的催化剂，通常与软件过程改进规划图相关联，评估结果会记录在软件过程改进的计划中，以便更有针对性地进行改进，从而达到过程改进规划图中更高的软件过程成熟度级别。

10.2　CMM 简 介

软件过程需要不断完善，首先从非工程化的软件开发方式改变为工程化的软件方式，按照软件工程的系统方法进行软件的工程活动，进而不断完善和改进各个软件过程，从而不断提高软件过程能力，特别是完成软件产品在预算、进度、产品的质量等方面的风险也逐步降低。而 CMM 机构正是这些思想和方法的体现，事实上，CMM 给软件机构提供了量度软件过程的尺子，同时，CMM 也是一个指南，起到指导软件机构的作用。

10.2.1 CMM 的发展过程

软件能力成熟度（the capability maturity model for software，CMM）是美国软件工程研究所（software engineering institute，SEI）首先提出的，而 CMM 的基本思想是基于已有 60 多年历史的产品质量原理。希祆特（Walter Shewart）在 20 世纪 30 年代发表了统计质量控制原理，戴明（W.Edwards）和朱兰（Joseph Juran）的关于质量的著作又进一步发展和论证了该原理。

实际上，将质量控制原理变为成熟度框架的思想是克劳斯比（Philip Crosby）。克劳斯比在著作“ Quality is Free ”中首先提出成熟度框架，其质量管理成熟度网络描绘了采用质量实践时的 5 个进化阶段，而该框架后来又由 IBM 的拉迪斯（Rom Radice）和他的同事们在汉弗莱（Watts Humphrey）指导下进一步改进以适应软件过程的需要。1986 年，汉弗莱将此成熟框架带到了 SEI 并增加了成熟度等级的概念，将这些原理应用于软件开发，发展成为软件过程成熟度框架，形成了当前软件产业界正在使用的框架。

汉弗莱的成熟度框架早期版本发表在 1987 年的 SEI 技术报告。该报告中还发表了初步的成熟度提问单,这个提问单作为工具给软件组织提供了软件过程评估的一种方法。1987 年又进一步研制了软件过程评估和软件能力评价两种方法，以便估计软件过程成熟度。自 1990 年以来，SEI 基于几年来将框架运用到软件过程改进方面的经验，进一步扩展和精炼了该模型，目前，软件能力成熟度模型 2.0 版已修订问世。

10.2.2 软件过程成熟度的基本概念

下面是有关软件过程成熟度的基本概念。

（1）软件过程：人们用于开发和维护软件及其相关过程的一系列活动，包括软件工程活动和软件管理活动。

（2）软件过程能力：描述开发组织或项目组遵循其软件过程能够实现预期结果的程度，既可对整个软件开发组织而言，也可对一个软件项目而言。

（3）软件过程性能：表示开发组织或项目组遵循其软件过程所得到的实际结果，软件过程性能描述的是已得到的实际结果,而软件过程能力则描述的是最可能的预期结果，既可对整个软件开发组织而言，也可对一个特定项目而言。

（4）软件过程成熟：一个特定软件过程被明确和有效地定义，管理测量和控制的程度。

（5）软件能力成熟度等级：软件开发组织在走向成熟的途中几个具有明确定义的表示软件过程能力成熟度的平台。

（6）关键过程域：每个软件能力成熟度等级包含若干个对该成熟度等级至关重要的过程域，这些过程域的实施对达到该成熟度等级的目标起到保证作用。这些过程域就称为该成熟度等级的关键过程域，反之非关键过程域对达到相应软件成熟度等级的目标不起关键作用。过程域归纳为：互相关联的若干软件实践活动和有关基础设施的一个集合。

（7）关键实践：对关键过程域的实践起关键作用的方针、规程、措施、活动以及相关基础设施的建立。关键实践一般只描述“做什么”而不强制规定“如何做”。整个软件

过程的改进是基于许多小的、渐进的步骤，而不是通过一次革命性的创新来实现的，这些小的渐进步骤就是通过一些关键实践来实现。

（8）软件能力成熟度模型：随着软件组织定义、实施、测量、控制和改进其软件过程，软件组织的能力也伴随着这些阶段逐步前进，完成对软件组织进化阶段的描述模型。

10.2.3　全面质量管理和 CMM

软件质量管理的目的是建立对项目的软件产品质量的定量了解和实现特定的质量目标。软件质量管理包含确定软件产品的质量目标，制定实现这些目标的计划，并监控及调整软件计划、软件工作产品、活动和质量目标，以满足顾客和最终用户对高质量产品的需要及愿望。软件质量管理的实践基于 3 个关键过程区域，前两个是建立和实施项目、定义项目、定义软件过程的关键过程区域——集成软件管理和软件产品工程，而后一个是定量过程管理关键过程区域，建立对项目定义、软件过程实现所希望的结果能力的定量了解。

基于组织、顾客和最终用户的需要来建立软件产品的质量目标，这些目标才能实现。

为实现质量目标，组织制定战略和计划，项目则具体调整其已定义的软件过程。

1．目标

目标 1　项目的软件质量管理活动是有计划的。

目标 2　软件产品质量的可测目标及其优先级是确定的。

目标 3　实现软件产品的质量目标的实际进程是用数量表示的和受到管理的。

2．执行约定

约定 1　项目遵循书面的用以管理软件质量的组织方针。

该方针一般规定：

（1）项目的软件质量管理活动支持组织对改善软件产品质量的承诺。那些能提高软件产品质量的过程改进具有组织的最高优先级。每个新发行的软件产品应该是可测量地好于前任产品或主要竞争对手的产品。

（2）项目基于项目定义软件过程确定和收集用于软件质量管理的测量数据。

（3）项目确定软件产品的质量目标，并监控其实现质量目标的进程。

（4）确定软件质量管理的职责并将其分派给软件工程组和其他的软件有关组。

软件有关组的例子包括：软件质量保证组；软件配置管理组；文档支持组。建立使得这些组能确定在满足软件产品质量目标上的成功情况的准则。

3．执行能力

能力 1　为管理软件产品的质量提供足够的资源和投资。

（1）可以得到在诸如安全性和可靠性领域有专长的工程师以便帮助设置软件质量目标和评审朝着此目标前进的进程。

（2）使得支持预测、测量、跟踪和分析软件质量的工具合用。

支持工具的例子包括：

—数据采集工具；

—数据库系统；

—电子表格程序；

—软件生存周期仿真器；

—定量分析工具；

—代码检查工具。

能力 2　实施和支持软件质量管理的个人接受在执行其活动方面所要求的培训。

培训的例子有：

—策划产品的质量约定和目标；

—测量产品和过程质量；

—用已定义软件过程去控制产品质量。

能力 3　软件工程组和其他软件有关组的成员接受在软件质量管理方面所要求的培训。

培训的例子包括：

—理解定量管理产品质量的目标和好处；

—收集测量数据；

—理解对于软件过程和产品的质量度量；

—策划和控制软件产品的质量。

4．执行的活动

活动 1　按照已文档化的规程制定和维护项目的软件质量计划。

该规程一般规定：

（1）当合适时，建立对组织、顾客和最终用户的软件质量需求的理解。在这些实践中所指的最终用户是顾客指定的最终用户或最终用户的代表。

测量顾客和最终用户的软件质量需求的方式的例子有：

—调查；

—焦点组（focusgroups）；

—由用户作的产品评价。

（2）组织、顾客和最终用户的软件质量需求及其优先级可以追踪到分配给软件的系统需求及软件质量目标。

追踪这些需求和优先级的方法的一个例子是质量功能展开（quality function deployment，QFD）。

追踪需求和优先级到产品的软件质量目标的一个例子是建立交付后允许出现缺陷数目的目标，随着产品逐渐成熟不断进行预报练习以便评估满足这些目标的可能性。在这些实践中，分配给软件的系统需求称为“分配需求”。

（3）对项目定义软件过程满足软件质量目标的能力加以评估并记入文档。

诸如质量功能展开和关于设计的 Taguchi 方法等技术可用于显示产品质量目标和过程能力的关系。

（4）当合适时，软件质量计划满足组织的质量计划。

（5）当合适时，软件质量计划基于组织中以前项目的或当前项目的计划。

（6）在项目的开头，在主要的项目里程碑处，以及每当分配需求有重大改变时，更新软件质量计划。

（7）软件质量计划经受同行评审。

（8）受影响的组和个人评审软件质量计划。

受影响的组和个人的例子有：

—顾客；

—最终用户；

—软件工程组（包括所有的小组，例如软件设计小组）；

—软件估计组；

—系统工程组；

—系统测试组；

—软件质量保证组；

—软件配置管理组；

—合同管理组；

—文档支持组。

（9）高级管理者评审软件质量计划。

（10）对软件质量计划进行管理的控制。

“进行管理和控制”意味着在给定时间（过去或现在）使用的工作产品的版本是已知的（即版本控制），而且以受控的方式引进更改（即更改控制）。如果希望有比“进行管理和控制”所蕴含的更加正规的手续，则工作产品可置于配置管理的完备纪律之下，正如在软件配置管理关键过程区域中所描述的。

（11）对所有受影响的组和个人，软件质量计划是可以得到的。

活动 2　项目的软件质量计划是项目软件质量管理活动的基础。

该计划包括：

（1）过程中进行软件质量度量的点。

（2）软件产品的有重大影响（high-leverage）的质量目标。

软件产品的有重大影响的质量目标是那些能以最少成本提供最大的顾客满意度的质量目标，或者是顾客或最终用户定的“必须有”的质量目标。

（3）为了在过去质量性能的基础上作出改进，软件项目将实施的行动。

（4）旨在测量软件产品质量的活动。

旨在测量软件产品质量的软件活动的例子有：

—同行评审；

—原型开发；

—产品仿真；

—测试。

（5）当合适时，有关软件工作产品的质量目标。

适合记载在项目的软件质量计划中的软件产品质量目标的例子有：

—计划将实现的特征；

—关键特征，如果不实现这些特征，就会产生顾客或最终用户不希望或不需要的产品。

（6）当预计软件产品质量不满足质量目标时将采取的措施。

活动 3　在整个软件生存周期，确定、监控和修定项目的软件产品的定量质量目标。

（1）识别那些描述软件产品将如何好地运行或如何好地被开发和维护的产品质量

特征。

软件质量特征的例子包括：

—功能性；

—可靠性；

—维护性；

—适用性。

（2）确定用于量化软件产品质量特征的测量。

确定产品质量测量的活动的例子包括：

—评审以前的性能数据和顾客需求；

—开发原型；

—用形式化表示方法描述中间软件产品；

—采用正式的软件工程方法；

—进行测试。

（3）对于每个软件产品质量性，根据所要求的和所希望的值，选择可测的数字值作为产品的质量目标。

在软件产品可靠性方面可能的质量目标的例子有：

—如同在需求中所规定的平均故障间隔时间；

—必须实现的平均故障间隔时间（由分析和实验所确定）；

—计划达到的平均故障间隔时间。

（4）在项目软件质量计划中用文字说明软件产品质量目标。

适合记载在项目软件质量计划中的软件产品质量目标的例子有：

—计划要实现的特征；

—关键特征，如果不实现这些特征，就会生产顾客或最终用户不希望或不需要的产品。

（5）确定每个软件生存周期阶段的质量目标并写成文档。

软件生存周期阶段的例子包括：

—软件需求；

—软件设计；

—编码；

—软件测试。

和软件生存周期阶段有关的质量目标的例子有：

—与每个软件生存周期阶段有关的产品缺陷数将比以前发行的产品的缺陷数降低某个预先确定的百分比；

—预先确定的在测试周期结束时预期将发现缺陷的百分比。

（6）随着对产品的了解和对组织、顾客及最终端用户的需求的了解逐渐增加，修订软件产品和软件生存周期阶段的质量目标。

活动 4　在事件驱动的基础上，对项目软件产品的质量进行测量、分析，并将其与产品的定量质量目标相比较。

（1）计划和执行旨在阐述项目的软件质量目标的软件作业。在软件作业的开头，执行该作业的群组：

—评审软件产品的质量目标；

－确定适用于软件作业的质量目标；

－确定用于实现软件质量目标的计划；

－评审为实现软件质量目标所作的对过程的更改。

更改的一个例子是修订同行评审的检查单以便阐述已发现的逃过同行评审的缺陷。

（2）测量每个软件生存周期阶段的软件工作产品的质量。

测量工作产品质量的方法的例子有：

—同行评审；

—仿真；

—测试。

（3）分析质量测量结果，并将其与软件质量目标作比较以确定是否满足质量目标。

（4）采取合适的与软件质量计划相一致的措施，以便使得产品的质量测量结果与软件质量目标相符合。

（5）当与确定软件质量目标相互矛盾时（即不损害另一个目标就不能实现某个目标），采取解决矛盾的措施。

－分析实现软件质量目标的成本；

－根据长期经营战略和短期优先级，考虑替代的软件质量目标；

－当合适时，顾客和最终用户参与质量权衡决策；

－当合适时，修订软件工作产品和计划以反映权衡的结果。

活动 5　将软件项目的产品定量质量目标恰当地分配给那些向项目交付软件产品的子承包商。

5．测量和分析

测量 1　进行测量并将测量结果用于确定软件质量管理活动的状态。

测量的例子包括：

—质量差的费用（基于已知的质量度量，其精度是能采集到的精度）；

—实现质量目标的成本。

6．验证实施

验证 1　高级管理者定期参与评审软件质量管理的活动。

验证 2　项目经理既定期地，事件驱动地参与评审软件质量管理活动。

验证 3　软件质量保证组评审和（或）审计软件质量管理的活动和工作产品，并报告结果。

至少，评审和（或）审计要查证：项目的软件质量计划的准备工作；用于建立和跟踪软件质量目标的过程。

10.2.4　基于模型改进的优点与风险

使用诸如 CMM 这样的模型作为改进软件过程的框架有很多优点。CMM 有助于让大家都了解机构进行软件过程改进的方法。CMM 为讨论软件过程建立了一个公共语言，并定义了为解决软件问题应做工作的顺序。

CMM 通过提供一个框架来执行可靠的、一致的估价来支持软件过程测量。尽管这些过程估价不排除人为因素，但是 CMM 提供了客观公正的基础。这个测量框架也支持工业领域机构间的比较，比较可通过当前实践状况报告[Humphrey89b、Kitson92 和 Herbsleb94]和软件过程改进实例研究[Humphrey91b、Lipke92、Dion93 和 Wohlwend93]来进行。

最重要的是，CMM 建立了一组有关软件过程和实践的集合，各部门的实际工作人员合作开发了这些过程和实践。在 CMM 建立期间，数百名软件专业人员评审了 CMM。有关 CMM 各类草稿的数千条意见和修改要求，由 CMM 开发和修改小组进行评审和解答，并建立了 CMM 咨询委员会来帮助 SEI 理解和解决由一些团体提出的问题。

当然，基于一个模型的改进工作不是没有风险的。用 George Box 的话说，“所有的模型都是错的，有些模型是有用的”。模型是对模型所代表的真实世界的简化，并且 CMM 并没有穷举软件过程的说明。CMM 不是综合性的，仅涉及到其他非过程因素，比如影响软件项目成功的人员和技术等。

CMM 不是银弹，不可能涉及对成功项目来说都重要的所有问题。例如，CMM 目前不涉及特定领域的专门知识，也不提倡任何特殊软件技术，或者建议如何选择、雇佣、激励以及保留有能力的人才。尽管这些问题对一个项目的成功非常关键，但不是 CMM 的组成部分。

CMM 代表了一种实施软件过程改进的“常识工程”方法。成熟度级别、关键过程域、共同特性和关键实践在软件界被广泛讨论和评审。尽管 CMM 还不完善，但代表了软件界广泛一致的意见，对于指导软件过程改进工作是一个有用的工具。

为了能一致地、有纪律地改进软件产品的管理和开发，CMM 提供了一个概念性结构。CMM 不能保证一定能成功地生成软件产品，也不能保证一定能很好地解决软件工程中的所有问题。但是，基于 CMM 的改进大纲的有关报告显示，可以提高软件机构达到成本、质量和生产率目标的可能性。

CMM 为成熟的软件过程明确了实践，并提供了实践情况（有时，提供技术情况）的例子，但这并不意味着 CMM 包罗万象或至高无上。CMM 明确了有效软件过程的特征，但成熟的机构针对的是与项目成败有关的所有问题，包括人员、技术以及过程。

CMM 是用来帮助软件机构改进其软件过程的一个工具，同时也协助交办机构选择和管理软件承制方。CMM 虽然只是一个工具，但必须灵活地运用才能帮助不同机构处理各自具体的业务需要。CMM 的目的是按成熟度框架中的结构描述优秀的管理与工程实践。

要正确、深入地使用 CMM，需要正确地评估方法。在特定环境下，要运用智慧、经验和知识对 CMM 进行适当、具体的说明。说明应建立在业务需要以及机构和项目的目标基础上。僵化地使用 CMM 会给机构造成隐患，而不是带来益处。

要逐步达到更高的软件过程成熟度级别，就需要长期保证对软件过程实施持续的改进。软件机构可能要花十年或更多时间为持续过程改进提供基础和改善企业文化。尽管大多数软件公司都不会有历时十年的过程改进大纲，但要成为成熟的软件机构必须要有这种持续不断的努力。

10.3　软件过程成熟度框架

软件开发的风险之所以大，是由于软件过程能力低，其中最关键的问题在于软件开发组织不能很好地管理其软件过程，从而使一些好的开发方法和技术起不到预期的作用。而且项目的成功也是通过工作组的杰出努力，所以仅仅建立在可得到特定人员上的成功不能为全组织的生产和质量的长期提高打下基础，必须在建立有效的软件工程实践和管理实践的基础设施方面，坚持不懈地努力，才能不断改进，才能持续地成功。

CMM 提供了一个框架，将软件过程改进的进化步骤组织成 5 个成熟等级，为过程不断改进奠定了循序渐进的基础。这 5 个成熟度等级定义了一个有序的尺度，用来测量一个组织的软件过程成熟和评价其软件过程能力，这些等级还能帮助组织自己对其改进工作排出优先次序。成熟度等级是已得到确切定义的，也是在向成熟软件组织前进途中的平台。每一个成熟度等级为连续改进提供一个台基。每一等级包含一组过程目标，通过实施相应的一组关键过程域达到这一组过程目标，当目标满足时，能使软件过程的一个重要成分稳定。每达到成熟框架的一个等级，就建立起软件过程的一个相应成分，导致组织能力一定程度的增大。

10.3.1　成熟度的 5 个级别

软件能力成熟度等级是软件开发组织在走向成熟途中的几个具有明确定义的、表征软件过程能力成熟度的平台。下面分别介绍 CMM 的 5 个成熟度等级的特性，以说明软件过程在每个等级上的变化。

等级 1——初始级（initial）

在初始级上，软件过程是无序的，有时甚至是混乱的。软件开发组织不能提供开发和维护软件的稳定环境，对软件过程几乎没有定义，进度、预算、功能性和产品质量等都不可预测。实施情况依赖于个人的能力，且随个人固有的技能、知识和动机的不同而变化。所以处于等级 1 的软件开发组织几乎没有稳定的软件过程，只能依靠个人的能力而不是组织的能力去预测软件开发活动的结果。

等级 2——可重复级（repeatable）

在可重复级上，软件开发组织已建立管理软件项目的方针和实施这些方针的规程，基于在类似项目上的经验对新项目进行策划和管理。达到等级 2 的目的是使软件项目的有效管理过程制度化，使软件开发组织能重复以前开发项目时的成功实践。有效过程应具有如下特征：实用、已文档化、曾实施、已培训、可测量和能改进。

等级 2 的软件开发组织中的项目已设置基本的软件管理和控制。项目制定的约定均根据对以前项目的观察结果和当前项目的需求，因而切实可行。项目的软件管理者跟踪软件成本、进度和功能，一旦出现问题能及时识别。对软件需求和为实现需求所开发的工作产品建立基线，并控制其完整性。软件项目的标准均已确定，并且组织能保证准确地执行这些标准。处于等级 2 的软件开发组织的过程能力，可概括为“可重复的”。因为软件项目的计划和跟踪是稳定的，能重复以前的成功。

等级 3——已定义级（defined）

在已定义级上，全组织的软件开发和维护的标准过程均已文档化，包括软件工程过程和软件管理过程，且这些过程被集成为一个有机的整体，称为组织的标准软件过程。用这个标准软件过程来帮助软件管理者和技术人员工作得更有效。组织在使其软件过程标准化时，利用有效的软件工程实践。组织中有专门负责全组织软件过程活动的组，例如软件工程过程组（SEPG）。软件开发组织制定并实施全组织的培训计划，以保证其职工和管理者均具有履行其职责所需的知识和技能。

项目根据其特征裁剪组织的标准软件过程，从而建立起自定义的软件过程，称为项目定义软件过程。一个已定义软件过程包含一组协调的、集成的、妥善定义的软件工程过程和管理过程。妥善定义的过程具有如下特征：具有关于准备就绪的判据、输入、标准、进行工作的规程、验证机制(例如同行专家评审)、输出以及关于完成的判据。因为软件过程已妥善定义，管理者就能洞察所有项目的技术进展情况。

达到等级 3 组织的能力可概括为“标准的和一致的”。因为无论软件工程活动还是管理活动，过程都是稳定的且可重复的。在所建立的产品生产线内，成本、进度和功能性均受控制、对软件质量也进行了跟踪。

等级 4——定量管理级（managed）

在定量管理级上，软件开发组织对软件产品和过程都设置了定量质量目标。对所有项目都测量其重要软件过程活动的生产率和质量。利用全组织的软件过程数据库收集和分析从项目定义软件过程中得到的数据。在等级 4 上的软件过程均必须具有明确定义的和一致的测量方法和手段，使得定量地评价项目的软件过程和产品质量成为可能。项目通过将其过程的实际变化限制在定量的可接受范围之内，从而实现对其产品和过程的控制。

处于等级 4 的软件开发组织的过程能力，可概括为“定量地可预测的”。该等级的过程能力使软件开发组织能在定量限制的范围内预测过程和产品质量方面的趋势。当超过限制范围时，采取措施予以纠正，使软件产品具有可预测的高质量。

等级 5——优化级（optimizing）

在优化级上，整个软件开发组织集中精力进行不断的过程改进。为了预防缺陷出现，组织能有效地识别出软件过程的弱点，并预先加强防范。在采用新技术并建议更改全组织的标准软件过程之前，必须进行费用效益分析。组织应能不断地识别出最好的软件工程实践和技术创新，并在整个组织内推广。各软件项目经分析现有软件过程存在的缺陷，并确定其发生的原因，使经验教训在全组织内共享。

处于等级 5 的软件开发组织的软件过程能力的基本特征可概括为“不断改进的”。因为这些组织为完善其软件过程能力进行着不懈的努力，因而能够不断地改善其项目的过程实效。为了能够不断改进，既采用在现有过程中增量式前进的办法，也采用借助新技术、新方法进行革新的办法。

CMM 中的各成熟度等级描述了在这个成熟度等级上软件开发组织的特征。每一等级均为后继的等级奠定基础，为有效地实施软件过程提供支持。软件能力成熟度等级的提高是一个循序渐进的过程，每个等级形成一个必要的基础，从此基础出发才能达到下一个等级。

表 10-1 给出了 CMM 模型概要，表中的 5 个等级各有其不同的行为特征。要通过描

述不同等级组织的行为特征：即一个组织为建立或改进软件过程所进行的活动，对每个项目所进行的活动和所产生的横跨各项目的过程能力。

表 10-1　CMM 成熟度级别

过程能力等级	特　　点	关键过程域
1 初始级	软件过程是无序的，有时甚至是混乱的，对过程几乎没有定义，成功取决于个人努力。管理是反应式（消防式）的	
2 可重复级	建立了基本的项目管理过程来跟踪费用、进度和功能特性。制定了必要的过程纪律，能重复早先类似应用项目取得成功	需求管理 软件项目计划 软件项目跟踪和监督 软件子合同管理 软件质量保证 软件配置管理
3 已定义级	已将软件管理和工程文档化、标准化，并综合成该组织的标准软件过程。所有项目均使用经批准、剪裁的标准软件过程来开发和维护软件。	组织过程定义 组织过程焦点 培训大纲 集成软件管理 软件产品工程 组织协调 同行专家评审
4 已定量管理级	收集对软件过程和产品质量的详细度量，对软件过程和产品都有定量的理解与控制	定量的过程管理 软件质量管理
5 优化级	过程的量化反馈和先进的新思想、新技术促进过程不断改进	缺陷预防 技术变更管理 过程变更管理

通过表 10-1，可以知道每个成熟度等级的关键过程域以及每个关键过程域包括一系列相关活动，只有全部完成这些活动，才能达到过程能力目标。为了达到这些相关目标，必须实施相应的关键实践。

10.3.2　软件过程的可视性

软件工程师们掌握有关项目状态和性能的第一手材料，所以对项目状态有详细而深入的了解。可是，对大项目而言，软件工程师们的洞察力常来源于在其职责范围内的个人经验。项目以外的不掌握第一手材料的人，例如高级经理，对项目过程缺乏可视性，为得到监控进程所需求的信息只得依靠定期的评审。在过程成熟度每个等级上，后续的成熟等级逐步增加地提供软件过程更好的可视性。

等级 1 的软件过程是一种无组织的整体——一个黑盒，项目过程的可视性是有限的。由于没有很好地定义活动阶段的划分，经理在确定项目过程和活动状态上极为困难。

需求以失控的方式进入软件过程，然后生产出产品。软件开发常被看作为不可知的魔术，特别对不熟悉软件的经理而言更是如此。

等级 2 上，顾客需求和工作产品受到控制，已经建立起基本的项目管理实践。这些管理控制使得在规定的场合项目可视。构造软件的过程可看作为一个接一个的黑盒子，

这使得随着活动在盒子间流动，在各过渡点（项目里程碑）上具有管理可视性。尽管管理者可能不知道盒子内发生事情的细节，但是过程的产品和用以证实过程正在适当进行的检验点是确定且已知的。当问题出现时，管理者能对问题作出反应。

等级 3 上，盒子的内部结构（即项目定义软件过程中的作业）是可视的。内部结构表示组织的标准软件过程用于具体项目的方式。经理们和工程师们都了解自己在过程中的角色和职责，并且知道自己的活动如何在合适的细节层次上相互影响，管理者预先作好应付可能发生风险的准备。由于已定义的过程提供对项目活动很好的可视性，项目外的个人能获得精确的、即时的状态更新信息。

等级 4 上已定义的软件过程被配备上度量，并得到定量地控制。经理们能够度量进程和问题。管理者作决策时，有一个客观的定量的基础。随着过程的变异性越来越小，管理者预测结果的能力越来越成熟。

等级 5 上，为了提高生产率和质量，以受控的方式对构造软件的新的和已改进的方法进行不断的试验。因为无效的或者易出错的活动进行到标识并加以替代或修改，有纪律的更改成为一种生活方式。不仅对现存在过程明察秋毫，而且洞察力扩展到能了解过程潜在变化的影响。经理们能定量地估计更改的影响和有效性。然后跟踪更改。

10.3.3 跳越成熟度级别

CMM 中的成熟度等级描述在一个成熟度等级上组织的特征。每一等级均为接连的几个等级奠定基础，为有效地且效率高地实施过程提供支持。可是，在有益时，组织也可以使用比所在等级高的那些成熟度等级中所描述的过程。例如，虽然 CMM 中在等级 3 以前不讨论工程过程——诸如需求分析、设计、编码和测试，但是甚至等级 1 组织都必须进行这些活动。在有利可图时，等级 1 或等级 2 组织可以进行同行评审（等级 3 的）、Pareto 分析（等级 4 的）或者引入新技术（等级 5 的）。在讨论为了从等级 1 提高到等级 2 一个组织应采取何种步骤时，一个经常性的建议是建立软件工程过程小组，而这是等级 3 组织的属性。虽然度量是等级 4 的关注焦点，但也是较低成熟度等级的必备部分。

可是这些过程的潜力只有在建立适当基础之后才能得到完全的发挥。例如，同行评审不可能完全有效，除非始终一致地实施这种评审，即使对非常紧急的项目也是如此，成熟度等级只描述一个等级上占主导地位的问题。等级 1 组织的占主导地位的问题是管理；策划和管理软件项目的困难掩盖了其他问题。

因为每个等级形成一个必要的基础，从此基础出发才能达到下一个等级，因此，跳越等级是违反生产规律的。CMM 识别几个等级，一个组织必须逐步经历这些等级才能建立优秀的软件工程文化。没有合适基础的过程，在最需要的时刻即处于压力下不起作用，也不提供未来改进的基础。

等级 1 组织，在尚未建立可重复过程（等级 2）之前，试图去实施已定义的过程，通常不会成功，因为项目经理会被进度和成本的压力压垮。这是在关注工程过程之前应首先集中注意力于管理过程的基本原因。看起来定义和实施一个工程过程似乎要比定义和实施管理过程容易（特别在技术人员眼中），但是如果没有管理规定，工程过程会成为进度和成本等压力的牺牲品。

一个尚无已定义过程作为基础就试图实施已管理（等级 4）的组织通常是不成功的，因为没有已定义的过程就没有解释度量的共同基础。虽然对于一个项目能采集数据，但几乎没有什么度量对本项目之外的其他项目有重大意义，也不能显著地增加组织对软件过程的理解。缺少已定义过程时，由于被测过程的变化，要鉴别出有意义的度量是困难的。

一个尚无已管理过程（等级 4）作为基础就试图实施优化过程（等级 5）的组织，由于缺乏对过程更改所产生的后果的了解，多半会失败。在不能控制过程使其处于统计意义上狭窄的范围内（即过程度量中仅有小的变化）的情况下，数据中有太多的噪声以致不能客观地确定某项具体的过程改进是否有效。因为几乎没有定量基础用于作出理性的、有信息依据的决策，决策可能退化为宗教式的争斗。

过程改进工作应该把焦点集中在经营环境的上下文中组织的需要。具有实施较高成熟度等级的过程的能力并不表示可以跳越成熟度等级。

10.4　能力成熟度模型的结构

CMM 为每个软件组织建立和改善软件过程提供了一个阶梯式的过程成熟度框架，这一框架由 5 个成熟度等级构成。成熟度等级反映了一个软件组织进行软件产品开发的能力。除初始级以外，其余的成熟度等级都包含了若干个关键过程区域，每个关键过程区域又包含了若干个关键实践，这些关键实践按照 5 个共同特点加以组织。关键实践是对关键过程区域起重要作用的基础设施或活动，只要认真地执行关键实践，就能实现关键过程区域的目标，进而改善组织的软件过程能力。

10.4.1　成熟度级别的内部结构

图 10-1 是 CMM 内部结构图。除第 1 级外，CMM 的每一级是按完全相同的结构构成的。每一级包含了实现这一级目标的若干关键过程域 KPA（key process area），每个 KPA 都确定了一组目标，将每个 KPA 分成 5 个共同特性，分别是执行约定、执行能力、执行活动、测量和分析、验证执行。每个共同特性进一步又划分成若干关键实践 KP（key process），当这些关键实践都得到实现时，就完成了关键过程域的目标。下面来具体解释这些概念。

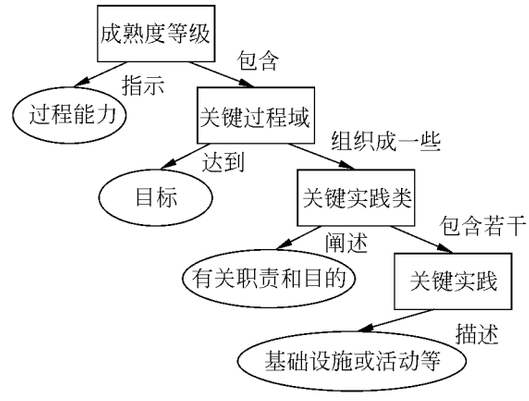


图 10-1　CMM 内部结构图

关键过程域：除第 1 级外，每一级都有多个关键过程域，表明了机构改进其软件过程的方法。关键过程域明确了达到某一成熟度级别必须解决的问题。如处于第 3 级的机构，必须解决第 2 级和第 3 级中所有关键过程域的问题。每个关键过程域明确相应的活动，完成了

这些活动，就提高了过程能力。对满足关键过程域的机构而言，必须实现关键过程域的所有目标。每个关键过程域都用关键实践的概念进行描述。

10.4.2 关键过程域

除等级 1 外，每个成熟度等级均含几个关键过程域（KPA）。KPA 是互相关联的若干软件实践活动和有关基础设施的集合。KPA 的实施对达到该成熟度等级的目标起保证作用。

每个关键过程域只与特定的成熟度等级直接相关，指明一组相关的实践活动，当这些活动全部完成时，就能达到对增强软件过程能力至关重要的若干目标。当其所有项目均已达到一个关键过程域的目标时，该组织才达到了以该关键过程域为表征的软件过程能力。这里的“目标”是用来概括一个关键过程域的关键实践，阐明该关键过程域的范围、边界和目的意图等。为了达到一个成熟度等级，必须实现该等级上的全部关键过程域。

10.4.3 关键实践

关键实践：对关键过程域的有效实施和制度化起最重要作用的基础设施和活动。每个关键实践的描述由两部分组成：前一部分说明关键过程域的基本方针、规程和活动，称为顶层关键实践；后一部分是详细描述，可能包括例子，称为子实例。图 10-2 给出了可重复级软件项目策划关键过程域中的一个关键实践的内部结构的例子。

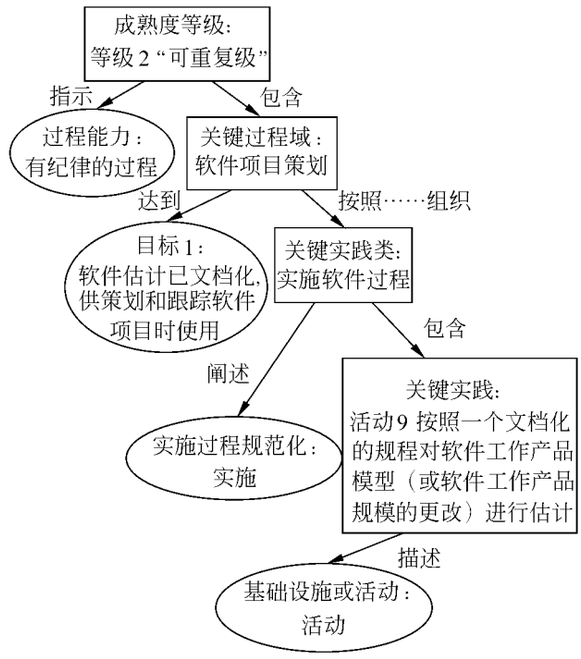


图 10-2　一个关键实践的例子

关键实践描述应该做“什么”，而不是强制要求应该“如何”实现目标。其他可代替的实践也可能实现该关键过程域的目标。

10.4.4　共同特性

不同成熟度级别中的关键过程域执行的具体实践不同。这些实践分别组成关键过程域的 5 个属性，即 5 个共同特性。

共同特性：指明一个关键过程域的执行和制度化是否有效、可重复和可持续。下面是这 5 个特性。

（1）执行约定：机构为确保过程的建立和持续而采取的一些措施。典型内容包括建立机构的策略和领导关系。

（2）执行能力：项目或机构完整地执行软件过程而必须有的先决条件，典型内容包括资源、机构的组织和培训。如建立软件质量保证组 SQA（software quality assurance）。每个关键过程域都描述了对资源和资金的要求，其中包括具体技术、充足资金和具体工具。这里的培训比一般意义上的培训范围要大，包括使用正规的和非正规的手段进行的具体教学和实践。

（3）执行活动：执行一个关键过程域所必需的活动、任务和规程的描述，典型内容包括制定计划和规程，执行和追踪，必要时采取纠正措施。

（4）测量和分析：为确定与过程有关的状态所必需的基本测量实践的描述。这些测量用于控制和改进过程，典型内容包括可能采用的测量实例。

（5）验证执行：为确保活动执行与已建立的过程一致所采取的步骤，典型内容包括管理部门和软件质量保证组实施的评审和审核。

10.5　CMM 的应用

任何单位所实施的软件过程，都可能在某一方面比较成熟，在另一方面不够成熟，但总体上必然属于这 5 个层次中的某一个层次。在某个层次内部，也有成熟程度的区别。在一个较低层次的上沿，很可能与一个较高层次的下沿非常接近，此时由这个较低层次向该较高层次进化也就比较容易。反之，在一个较低层次的下沿向较高层次进化，就比较困难。在 CMM 框架的不同层次中，需要解决带有不同层次特征的软件过程问题。因此，一个软件开发单位首先需要了解自己处于哪一个层次，然后才能够对症下药地针对该层次的特殊要求解决相关问题，这样才能收到事半功倍的软件过程改善效果。任何软件开发单位在致力于软件过程改善时，只能由所处的层次向紧邻的上一层次进化，即软件过程的进化是渐进的，而不能是跳跃的。而且在由某一成熟层次向上一更成熟层次进化时，在原有层次中的那些已经具备的能力还应该得到保持与发扬。

10.5.1　基于 CMM 的估价方法

CMM 的评估要遵循 SEI 的 CAF（cmm assessment frame-work）规范，由 CMU/SEI 授权的主任评估师（lead assessor）领导一个评审小组进行。评估过程包括员工培训（企

业的高层领导也要参加)、问卷调查和统计、文档审查、数据分析、与企业的高层领导讨论和撰写评估报告等，评估结束时由主任评估师签字生效。目前主要有两种基于 CMM 的评估方法，一种是 CBA-SCE（cmm-based appraisal for software capability estimation)，是基于 CMM 对组织的软件能力进行评估，是由组织外部的评估小组对该组织的软件能力进行的评估。另一种是 CBA-IPI(cmm-based appraisal for internal process improvement)，是基于 CMM 对内部的过程改进进行的评估，是由组织内部和外部人员联合组成的评估小组对软件组织本身进行评估以改进质量，结果归组织所有，目的是引导组织不断改进质量。这两种评估均由 CMU/SEI 授权的主任评估师领导，根据 CMM 模型来进行，都要审查正在使用和将来使用的文件/文档，并对不同的组织员工进行采访。SCE 与 IPI 两种评估结果应该一致，评估结果的所有资料都将呈报 CMU/SEI。

CBA-IPI 的评估由几方共同组成：评估小组、公司的管理人员、具体项目的执行人员以及主任评估师。其中评估小组是由经验丰富的软件专业人员组成，还要经过 CMM 和 CBA 评估的培训，使其了解组织的同时，也懂得如何将 CMM 模型及关键实践与组织的要求建立关联。评估过程主要分成三个阶段：准备阶段、评估阶段和报告阶段。第一阶段是准备阶段，包括小组人员培训、计划以及其他必要的评估准备工作。在评估的最初几天，小组成员的主要任务是采集数据，回答 SEI 的 CMM 提问单，文档审阅以及进行交谈，对整个组织中的应用有一个全面的了解；第二阶段进行数据分析，评估员要对记录进行整理，并检验所观察到的一切信息，然后把这些数据与 CMM 模型进行比较；第三阶段给出一个评估报告，在每个评估报告中，必须针对 CMM 的每个关键过程方面，指出这个软件过程在什么地方已经有效地执行了，什么地方还没有有效地执行。只有所有评估人员一致通过的情况下，这个评估报告才有效。在评估报告的基础上，评估小组产生一个评估结果。评估和评级的结果应与有关的关键过程方面和目标相对应。评估报告和结果将送交所有有关的人员并上报 CMU/SEI。

需要说明，软件企业在作评估之前要明确 CMM 的目标和目前有哪些需要亟待解决的问题。同时，企业还需要用一些量化的数据来具体计算在实施 CMM 过程中各方面的利益问题。该利益分析有利于建立共同战略目标，有利于识别动力和障碍，也有利于设计针对性的沟通和培训计划。表 10-2 是某小型企业的利益分析结果。其中：S 表示短期可获得利益（短期利益为一个软件项目周期）；L 表示长期可获得利益；D 表示直接受益者；U表示间接受益者；N表示不受益者。

表 10-2　利益周期分析表

利　益	周　期	公　司	部　门	项　目	成　员	客　户
1．企业财富有效管理	S	D	D	U	U	U
2．提高交付产品质量	L	D	D	U	U	D
3．按计划按时完成项目	L	D	D	D	D	D
4．按计划控制成本	L	D	D	U	U	U
5．团队纪律性	S	D	D	D	U	U
6．减少人员流失带来的损失	S	D	D	D	U	U
7．提供管理及工程方法论	S	D	D	U	U	U
8．为产品化战略提供前提	L	D	D	U	U	U

续表

利　益	周　期	公　司	部　门	项　目	成　员	客　户
9．过程透明	S	D	D	U	U	U
10．建立学习型企业	S	D	D	D	D	U
11．促进沟通	S	D	D	D	D	D
总　计		11	11	5	3	3

从表 10-2 中可以得到获得直接利益最多的是公司和部门，获利最少的是项目组成员和客户。

10.5.2　软件过程评估及软件能力评价

软件过程评估集中关注在组织自身的软件过程中识别出改进的优先级。评估组采用 CMM 对调查发现进行鉴别和优先级排序。这些调查发现与 CMM 中的关键实践所提供的指导一起（例如，软件工程过程小组）被用来规划组织的改进策略。

软件能力评价集中关注识别与一项在进度要求和预算内构造出高质量软件的特定项目或合同相联系的风险。在采购过程中可以对投标者进行软件能力评价。评价的发现，正如 CMM 所设计的那样，可以用于确定在挑选特定承包商方面的风险。也可对现有的合同进行评价以便监控其过程性能，目的是在承包商的软件过程中识别出潜在的可改进之处。

CMM 为进行软件过程评估和软件能力评价建立一个共用的参考框架。虽然这两个方法的目的不同，但均采用 CMM 作为评估软件过程成熟度的根据。图 10-3 概要描述评估和评价中的共同步骤。

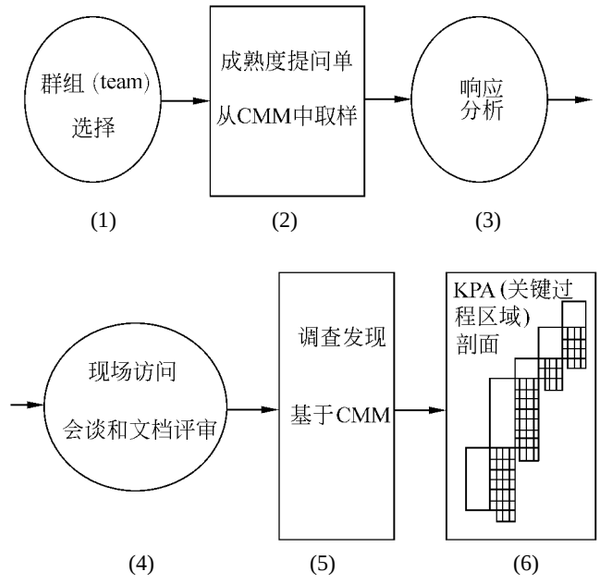


图 10-3　软件过程评估和能力评价的共同步骤

第一步是选择一个群组。该群组应该接受在 CMM 基本概念和评估或评价方法的细节方面的培训。该群组的成员应是具有丰富软件工程和管理方面知识的专业人员。

第二步是让来自待评估或评价单位的代表完成成熟度提问单的填写和其他诊断工具的要求。一旦完成此项活动，评估或评价组就对提问单响应进行分析（第三步），即对提问响应进行统计，并识别必须作进一步探查的区域。待探查的区域与 CMM 的关键过程区域相对应。

现在该群组已准备好去访问被评估或评价单位的现场（第四步）。从响应分析的结果着手，该群组进行会谈和评审文档以便了解该现场所遵循的软件过程。CMM 中的关键过程区域和关键实践对该群组成员在提问、倾听、评审和综合得自会见和文档中的信息等方面提供指导。

在确定现场的关键过程区域的实施是否满足相关的关键区域的目标方面，该群组运用专业性的判断。当 CMM 的关键实践与现场的实践间存有明显差异时，该群组必须用文件记下用于判断此关键过程区域理论依据。

在现场工作阶段结束时，该群组生成一个调查发现清单（第五步），标识出该组织软件过程的强项和弱项。在软件过程评价中，该调查发现成为提出过程改进建议的基础；在软件能力评价中调查发现成为采购单位所作的风险分析的一部分。

最后，该群组作出一份关键过程区域剖面图（第六步），指出该组织已满足和尚未满足关键过程区域目标的那些区域。一个关键过程区域可能是已满足的，但仍有一些相关的调查发现，假定这些调查发现没有识别出能阻止实现该关键过程区域的任何一个目标的主要问题。

总之，软件过程评估和软件能力评价方法两者均有以下特征：

- 采用成熟度提问单作为现场访问的出发点；
- 采用 CMM 作为指导现场调查研究的路线图；
- 利用 CMM 中的关键过程区域生成标识软件过程强项和弱项的调查发现；
- 在对关键过程区域目标满足情况进行分析的基础上，推导出一个剖面；
- 根据调查发现和关键过程区域剖面，向合适的对象提出结论意见。

尽管有这些相似之处，但软件过程评估或软件能力评价的结果可能不同，甚至在相继运用相同的方法接连进行评估（或评价）的情况下也是如此。

一个原因是评估或评价的范围可能变化。首先，必须确定受调查的组织情况。对一个大公司而言，对“组织”有几种不同的定义是完全可能的。范围可以基于有共同的高级管理者、共同的地理位置、指明为一个盈亏中心、共同的应用领域或其他考虑。其次，甚至在似乎是相同的组织中，所选项目的样本也可能影响范围。评估可能对公司中的一个部门进行，那么评估组基于全部门的范围得出其调查发现。后来，可能对该部门中的一条产品线进行评价，此时评价组所得出的调查发现基于窄得多的范围。没有了解上下文就在这些结果间作比较是有问题的。

软件过程评估和软件能力评价在动机、目的、输出和结果上均不同。这些因素导致在会谈目的、询问的范围、所采集的信息和结果的表示方式上有本质的不同。如果考察所采用的详细规程，那么评估和评价的方法大不相同。评估培训并不使得该组做好完成评价工作的准备，反之亦然。

软件过程评估都是在开放、合作的环境中的，评估的成功取决于管理者和专业人员两方面对改进组织的支持。评估目的在于暴露问题和帮助经理和工程师改进组织。尽管提问单在使评估组的注意力集中于成熟度等级问题上是有价值的，但是重点仍应放在作为了解组织软件过程的工具的结构化和非结构化的会谈上。除了识别组织所面临的软件过程问题外，评估的最有价值的结果还有为改进而作的改进，全组织范围关注过程，以及执行改进行动计划上的动力和热情。

另一方面，软件能力评价在更为面向审计的环境中进行。评价目的与金钱密切相关，因为评价群组的推荐将帮助挑选承包商或设置奖金。重点放在已文档化的审计记录上，这些记录能揭示组织实际执行的软件过程。

但是不意味着软件过程评估和软件能力评价不能比较。由于两者均基于 CMM 类似点和不同点应该是明显的且可以解释的。

10.5.3　软件过程改进

软件工业经历了 3 次特别明显的发展浪潮：第一次浪潮以瀑布式生命周期和结构化的方法为特征；第二次浪潮是过程成熟运动；第三次浪潮将是预期中的软件工业化。每次浪潮都是对前一次的改进。

（1）启动软件过程改进阶段

启动软件过程改进必须有 5 个阶段：

起始阶段　将软件过程改进介绍给经理和技术人员，使经理和技术人员相信软件过程改进的好处，这是让软件过程改进工作走上正轨的基础。

实现阶段　制定软件过程软件改进所采取的策略、执行评估、贯彻从评估结果中得出的行动计划。

制度化阶段　确保软件过程改进能够持续进行，并且已经融入企业文化中。

测量阶段　测量获得的收益，确保软件过程改进行为和商业目标一致。

改进阶段　评估测量阶段所获得的结果，制定进一步的过程改进计划以提高过程规范所带来的收益，保持软件过程改进和商业目标一致。

（2）软件过程改进的实现和制度化

软件过程改进的实现和制度化阶段是其生命周期中的两个阶段。在获得评估的建议之后，开始进入实现阶段。一旦软件过程改进的第一个循环已经完成，就需要有规律地重复整个活动，这就要求软件过程改进的制度化。

① 实现阶段

在评估获得的建议的基础上设计出实现评估建议的行动计划，行动计划应该是定义和开始一个或数个软件过程改进项目的基础。软件过程改进的最终目的是改进软件工程师和项目经理的实践，因此是一个改变的计划。对改变策略和技术的所有管理在软件过程改进的实现中都实用。

一个软件过程改进行动的重要工作表现为重新设计或改进当前的过程、设计新的过程、设计或改进过程环境的形式。软件过程改进工作的最终目的是创建过程架构和过程文化，使得过程思维和过程行为成为做事情的自然方式。

过程改进工作细分为几个软件过程改进项目，每个软件过程改进项目与相关的软件过程改进组相对应。每个软件过程改进组负责一个或多个软件过程改进的项目，并且实现软件过程改进行动。对于组织来讲，软件过程改进的最终目的是成功实现软件过程制度化。

软件过程改进周期能够通过高级管理层启动改进工作开始，接着经历评估、建议、行动计划和改进实现阶段，然后重新回到评估。在经历一个完整的过程改进周期之后，软件过程改进实践已经逐渐在组织内部制度化。

② 度量软件过程改进带来的收益

软件过程成熟度的一个主要目的是改进软件过程管理的可见性，获得对产品和过程质量的统计控制。软件过程评价本身是一种定性的度量，来测评目前的有效性。通过反馈机制则能够获得过程执行情况的定量指标，然后可以基于当前的性能设定改进目标(定量的)。

软件过程性能的测评为所有的系统性能改进提供必要的反馈环。反馈环对系统的稳定和调节是必需的。在软件过程环境中，反馈环是达到持续的过程改进状态所必需的，这种反馈环提供的过程性能的数据会提高产品的质量。

度量可以为改进提供反馈信息，任何没有反馈回路的系统会变得不稳定。当设计过程时，应该明确过程目标，将度量计划和目标联系在一起。

(3) 软件过程改进战略策划

通常，战略策划由一个小组负责，小组里要包括参与了过程评审的人员以及其他策划工作的受益人，另外高层经理的参与是非常重要的；策划是在负责人的指导下以讨论方式进行。实践证明，以下步骤非常有效：针对不同的改进点分别制定改进方案 ；方案评价；将改进方案排序 ；估计并制定实施的进度表 ；获得管理层的承诺 。

① 制定过程改进方案

评审结束后，策划组要对评审结果进行分析，筛选出十个左右的改进点；然后将每个改进点都作为一个改进项目，分别制定2～3页的改进方案。方案主要包括以下内容：现状的简单介绍；改进方案介绍；预期收益；实施负责人；对成本、资源和项目周期的估计。方案中还应该说明建议使用的实施方法，例如是否进行试用等。估计成本时要包括：过程定义的时间、试用期间人员培训的成本、处理反馈意见的时间和重新试用的成本。

② 评价各个改进方案

怎么确定改进活动的优先级呢？主要是通过考察 3 方面的因素，即：对商业目标的影响、风险和在 CMM 中的定位。有些公司还会对各方案进行成本/收益分析（例如，考察投资回报率），但是 1 级或 2 级的企业往往没有充分的历史数据，因此无法准确估计过程改进的无形收益；4 级和 5 级的企业通常就能做到这一点，3 级的企业也有可能做到。

③ 对改进方案进行排序

进行了以上分析之后，按照分值对各个改进方案进行排序，总分的计算方法如下：

总分=（权重 1）（对商业目标的影响）+（权重 2）（风险）+（权重 3）（在 CMM 中的定位）

公式中的得分是按上面介绍的步骤进行处理得出的，权重主要是根据策划小组成员的共识确定的，有些公司认为 3 方面的因素同样重要因此赋予相同权重，也有些公司认

为对商业目标的影响的重要性是在 CMM 中定位的 3 倍，而风险因素是在 CMM 中定位的 2 倍。这样，就基本建立了各个改进项目的优先级，分数最高的优先级最高。

④ 估计实施的进度表

排序完成后，就要考虑各个改进点的依赖关系，根据优先级顺序和依赖关系进行总体战略策划，并制定进度表。有趣的是，优先级较低的改进项目往往是优先级较高的项目的先决条件，因此在进度表中就应该靠前。另外，还要考虑实施效果的影响和可视性。例如，对于 1 级的企业，管理层还没有建立起过程改进的威信，过去给人的印象总是言行不一，那么就要选择风险较低，大家都能看到且有不凡收益的改进项目，帮助大家建立信心，即使这些项目优先级较低。

⑤ 获得管理层的承诺

下一步，要完成正式的计划、提交管理层获得认同和承诺。上面说过，高层管理人员的参与确定关键成功因素是非常必要的，这里要再次强调管理层的重要作用，因为管理层要负责批准战略计划、授权启动改进项目并且不断重申对于过程改进的承诺。

过程改进的总体计划通常包括：介绍（说明计划的目标）、制定计划时所使用的方法、对评审结果和推荐措施的总结，主要内容是各个改进项目的方案和策划活动的结果。当然也应该包括：进度表、相关任务、负责人、项目运行指标以及所需的资源，如：人员、资金、软件、硬件工具等。管理人员评审和签字批准，意味着管理层对改进活动人员和资源上的支持和承诺。

10.5.4　使用 CMM

作为一种模型，CMM 实际上是对软件机构工程过程的理论和数据模拟，在对 CMM 的应用中，主要包括软件产品供应方和应用方两大类。

目前，世界范围内已采纳 CMM 标准的企业纷纷以此标准决定软件项目合同的承接与分包。实践中，许多中小企业在接纳 CMM 体系时，采纳了保留企业部分原有工程过程指标并加以修改的办法。

卡内基-梅隆大学软件研究所提出了一套实施 CMM 标准的方法，按照该标准的建议，IDEAL 是企业开始引入 CMM 体系的良好参照模式，包括：

I——启动（initiating），表示开发机构应为 CMM 的引入准备好前期基础设施和程序。

D——诊断（diagnosing），明确机构目前所处的能力水平及目标等级所在。

E——建构（establishing），制定如何实现目标等级的计划。

A——行动（acting），具体实施该计划。

L——学习（learning），积累以往经验并将其用于持续的改进过程之中，同时注意新技术和工具的引入以协助过程实施。

CMM 可以有两个基本用途：即软件过程评估和软件能力评估。评估的目的是理解组织内当前正在使用的软件过程，确定待改进的关键过程域并开始前期的有关活动。

软件过程评估确定一个组织的当前软件过程的状态，确定组织所面临的急需解决的与软件过程有关的问题，进而有步骤地实施软件过程改进，使组织的软件过程能力不断

提高。软件过程评估集中关注一个组织的软件过程所需改进之处及其轻重缓急。评估组靠 CMM 去指导组织进行调查、分析和排优次序。组织可用这些调查结果和 CMM 中的关键实践所提供的指导相结合，来规划本组织软件过程的改进策略。

软件能力评估有时又叫能力测定，是一个受过培训的专业人员组成的小组，针对组织内选中的软件过程进行系统评估，并与指定的目标能力加以比较。同时，集中关注识别一个特定项目在进度要求和预算限制内构造出高质量软件所面临的风险。在采购过程中可以对投标者进行软件能力评价。评价的结果，可用于确定在挑选特定承制方方面的风险，也可对现有的合同进行评价，以监控承制方的过程性能，以便在承制方的软件过程中识别出潜在的可改进之处。可见，软件能力评估是识别合格的能完成软件的承制方，或者监控承制方软件开发工作过程能力进行评价的问题。

10.5.5 CMM 实施工具

一般来说，实施 CMM 需要以下主要工具：软件开发过程框架、需求管理工具、面向对象的分析设计工具、配置管理工具、变更管理工具和软件测试工具。

1．软件开发过程框架

CMM 是一种软件过程控制和评估框架，列出了每个级别需要完成的目标以及判定条件，但并没有叙述如何实现这些目标。软件开发过程框架工具的目标就是为开发团队建立一个清晰的、可重复执行的流程，以帮助团队成员按时完成项目各阶段的工作。

Rational 公司的 RUP（rational unified process）就是这样一个完整的软件开发过程框架，包括 3000 个 HTML 文档、近一百万字的流程指南，其中文版本 RUP-C 已经在中国市场正式发布。

RUP 对 CMM 实施的主要帮助体现在以下几个方面。

（1）凝结了全球软件行业的最佳开发经验，以指南、模板和示例的形式为开发团队提供流程指导。

（2）建立统一的软件开发标准，改善团队成员之间的沟通。

（3）降低软件开发风险，增加软件开发的可预测性。

（4）赋予项目经理对进度和交付期限的控制能力。可以说，RUP-C 是每一个计划实施 CMM 项目组的最佳起点，对 CMM 涉及的每个 KPA 都有帮助。

2．需求管理工具

需求是软件客户的要求，决定了软件系统的工作内容，是整个开发活动的基本出发点和最终目标。在整个项目生命周期内，要想有效地协作，就需要对重要的需求信息提供访问权限，使跨功能团队的所有成员都能掌握必要的详细信息。需求管理的目的是在客户和相应的软件项目之间建立共同的理解，并最终形成估计、策划和跟踪整个软件生命周期内软件项目活动的基础。

需求管理是 CMM2 级（可重复级）的关键过程域之一，其主要工作包括两点：其一，通过与涉众（stakeholder）的交流来获取需求，并进行有效的组织和记录；其二，使客户和项目团队在系统变更需求上达成一致。

一个优秀的需求管理工具可以在保证有效管理需求的前提下提高需求管理工作流

程的自动化程度，使需求管理可以真正在项目实施中得到有效的推行。Rational 公司为需求管理提供了 Analyst Studio 需求工作包，具有以下主要特点。

（1）结合业界认可的 RUP 方法，提供完整的需求分析及管理流程。

（2）以 Web 方式获取反馈，加强团队之间的有效沟通。

（3）用追踪图直观展现需求变化带来的影响。Analyst Studio 除了可用于 CMM2 级的需求管理外，还可以对以下 KPA 提供帮助：软件项目规划、软件项目跟踪与监督、软件子合同管理、软件产品工程、组间协作、同级复审和定量过程管理。

3．面向对象的分析设计工具

在 CMM3 级的软件产品工程（software product engineering）KPA 中，对软件设计提出了明确的要求，要求软件设计遵循一定的设计语言、采用面向对象的方法、使设计结果可复用等。采用面向对象的分析设计方法，主要原因有如下 3 点。

（1）通过分析和设计，使开发者可以先关注问题的领域，再关心具体的设计和编程问题，从而有利于降低整个过程的复杂性，提高分析模型和设计模型的质量。

（2）生成的分析模型和设计模型形成文档的主体，从根本上解决“先写代码、再补文档”的老问题，并能帮助团队规避因人员流动带来的不良影响。

（3）分析模型和设计模型将成为团队内部以及团队之间有效沟通的桥梁，消除误解，进一步解决“系统集成难”的顽症，同时也可以促进团队之间的软件复用。

Rational Rose 是 Rational 公司开发的可视化建模工具，采用“统一建模语言（UML）”的表示方法，在同一个模型中实现业务建模、对象建模和数据建模，使所有参与项目的成员都可以在统一的语言环境中工作于同一个模型之上，有利于改善成员之间的沟通；其次，支持多种语言的代码生成及双向工程，可实现代码和模型的互相转换，并且可以将遗留代码引入模型中；第三，带有对设计元素进行测试的模块工具（quality architect），可以尽早发现设计中的问题，真正实现“质量从头抓起”。

Rational Rose 除了可帮助实施 CMM3 级的软件产品工程外，还可以对组间协作和同级复审 KPA 提供帮助。

4．配置管理和变更管理工具

软件配置管理（SCM）是 CMM2 级中一个非常重要的 KPA，目的是在软件项目的生命周期内建立并维护软件项目产品的完整性。在 CMM 标准中，明确规定了软件配置管理(SCM)以及变更请求管理(CRM)的相关工作，包括以下两方面。

（1）配置管理的主要工作包括通过创建软件配置管理库、定义配置项(包括需求、分析设计模型、代码、文档、测试用例、测试数据等)以及建立和维护软件的基线。

（2）变更请求管理的主要工作包括控制和记录配置项内容的变更，建立和维护一个系统并使其追踪和管理变更请求及问题报告。

Rational 的 ClearCase 和 ClearQuest 是相当成熟的配置管理和变更管理工具，已经连续 4 年被国内数据公司 IDC 评为业界最佳的配置管理工具，并在众多的国内外企业中得到了应用，国内的华为、中兴、大唐、东信等企业都选用了这两个工具。ClearCase 的主要作用体现在 3 个方面。

（1）帮助项目组利用版本对象库(VOB)完整地保存整个项目的开发历史，实现对软件资产的有效管理。

（2）利用版本对象库的安全机制，灵活地控制不同人员对不同配置项的检出和读取的权利，有效地保护企业的核心机密，减少工作阻碍，保证其他开发工作，保证项目进度。

ClearQuest 的主要作用是加强开发团队与外界的沟通，用户、测试人员与市场销售人员可以直接通过 Web 提交变更请求，包括“缺陷”或“功能扩充请求”。

可以毫不夸张地说，配置管理和变更管理是软件工程的基础。ClearCase 和 ClearQuest 除了对 CMM2 级的软件配置管理 KPA 有帮助外，还可以对以下 KPA 提供帮助：需求管理、软件项目跟踪与监督、软件质量保证、软件产品工程以及定量过程管理。

5．测试工具

软件质量保证是 CMM2 级中另一个重要的 KPA，软件测试水平的高低直接影响软件产品质量的好坏与开发周期的长短。综观现阶段诸多软件开发组织在软件测试方面的状况，不难发现，多数组织存在以下问题。

（1）没有进行单元测试，或单元测试不彻底。

（2）由于需求不准确，导致测试缺乏计划。

（3）测试工作大多依赖手工进行，没有有效的自动化测试工具，致使软件测试效率低下，测试周期较长，不能在软件开发的各个周期进行完整的测试。

（4）缺乏有效的跟踪机制跟踪解决软件测试中发现的缺陷，致使有的问题得不到及时、恰当的解决。

针对这些问题，Rational 提供了一系列测试工具，帮助用户解决上述问题。这些工具包括用于单元测试的 RQA（rose quality architect）、Purify、Pure Coverage 和 Quantify 以及用于测试管理、回归测试和性能管理的 Rational Team Test。

目前，对大型软件系统来说，迭代式开发已经成为一种主流的开发模式，在开发的每个迭代周期内对软件功能进行确认，这就是回归测试。如果依靠人工测试的办法，将是一个烦琐、耗时的过程。Rational Team Test 采用面向对象的记录技术，将对系统的功能测试动作记录在测试脚本中，当系统进入下一个迭代周期时，只需回放这个测试脚本，就可以自动地进行软件功能的确认，这种方法可以极大地提高软件测试的效率，保证软件功能测试的完整性。

以上工具可以单独使用，解决个别问题，但是对于一个想在 CMM 实施中获得较高级别评估的软件组织来说，则需要尽可能广泛地进行有效的规划与部署，将这些工具有机地结合起来，满足企业在软件质量方面的较高要求。为此，Rational 提供了一个完整的解决方案——TestStudio，包括用于单元测试的 PQC（purify、quantify、pure coverage），提供全面功能测试和性能测试的 TeamTest，进行差错跟踪的 ClearQuest，以及一些对软件开发管理提供支持的基础工具，并将这些工具紧密地结合在一起。

本 章 小 结

CMM 通过优化企业开发流程，改善企业现有的规范和团队配合工作方法，来弥补软件企业对某个项目经理或开发工程师的单纯的依赖，从而避免了项目中某些“英雄”离职使得整个项目瘫痪的情况。此外，CMM 评估贯穿整个开发体系，从开发团队最底层的工程师到中层项目经理再到高层管理者。不仅如此，软件公司通过 CMM 评估也有

利于塑造整个公司的开发文化。因此，必须在软件企业实施 CMM。但是，CMM 不是万能的，CMM 的成功与否，与一个组织内部有关人员的积极参与和创造性活动是密不可分的，而且 CMM 并未提供实现有关子过程域所需要的具体知识和技能。因此，个体软件过程 PSP（personal software process）和群组软件过程 TSP（team software process）方法的应运而生可以说是由定向软件工程走向定量软件工程的一个标志。总之，单纯实施 CMM，永远不能真正做到能力成熟度的升级，只有将实施 CMM 与实施 PSP 和 TSP 等方法有机地结合起来，才能发挥最大的效力。

习　题　10

10.1　简述软件工程 7 原理与 CMM 的关系?
10.2　CMM 的理论基础是什么，是怎么发展起来的？
10.3　CMM 的主要特点是什么？
10.4　如何描述 CMM 软件能力成熟度模型分级结构及主要特征？
10.5　CMM 的关键过程域是如何划分的，如何将这些过程域在 CMM 进行分类？
10.6　怎样从管理方面、组织方面和工程方面对 CMM 关键过程的域分类?
10.7　成熟过程的特征是什么？
10.8　什么样的过程规范才能减少抑制性，最大限度的发挥创造性？
10.9　为什么软件开发需要规范化过程？
10.10　不同成熟度等级对进度、费用及质量方面有什么样的影响？
10.11　简述基于 CMM 评估的内容、评估过程和评估模型。
10.12　为什么 CMM 在实际项目应用中需要进行过程裁剪？裁剪的依据是什么？软件成熟度的各等级的行为特征是如何描述的？

第11章 软件的可靠性 27~29

如何获得一个可靠性较好的软件，是软件工程的课题之一。同时，软件可靠性也是最重要的软件特征，直接关系到软件完成功能的能力，一般来说，评价软件的可靠性用概率的方法。因此必须通过不断地测试来取得数据，并根据测试结果构造可靠性模型，评价实际的可靠性。本章将从几个方面对软件的可靠性进行讨论。

11.1 软件可靠性基本概念

11.1.1 软件可靠性定义

1．软件可靠性定义

软件在给定的时间间隔内，按照规格说明书的规定成功地完成运行要求的概率。显然，时间间隔是随机的，随着时间的增加，遇到故障的概率会增加，软件的可靠性就会降低。

2．软件可用性

一般说来，出现故障而又可以修复的软件被认为是可用的软件，软件的可用性是在给定的时间内，软件按照规格说明书的规定成功地完成运行要求的概率。

可靠性与可用性的差别在于，可靠性是时间间隔内不许出错，而可用性是瞬间的成功，也可能失败，失败了修复也叫可用。

3．稳态可用性

稳态可用性是正常运行时间占整个运行时间的比例，或者说是概率，定量表示为：

设正常运行时间为 t_{n1}，t_{n2}，…，故障停机时间为 t_{d1}，t_{d2}，…，则 $T_{\text{up}}=\sum t_{ni}$，$T_{\text{down}}=\sum t_{di}$，则稳态可用概率 A_{ss} 为：

$$A_{\text{ss}}=\frac{T_{\text{up}}}{T_{\text{up}}+T_{\text{down}}}$$

11.1.2　软件可靠性的主要指标

软件可靠性的主要指标，是指用定量的方法来衡量软件的可靠性。

1．平均无故障时间 MTTF（mean time to failure）

（1）假设对 n 个相同的系统进行测试，n 个系统故障时间分别是 t_1，t_2，…，t_n，则平均无故障时间 MTTF 定义为：

$$\text{MTTF}=\frac{1}{n}\sum_{i=1}^{n}t_i$$

对于软件系统来说，这相当于同一系统在 n 个不同的环境（测试用例不同）下进行测试，因此，MTTF 是一个描述故障模型的指标量，这个指标的目标值应由用户给出。

（2）平均无故障时间 MTTF 的估算

基本假设：E_T 为测试前故障总数；I_T 为机器指令总数；$f(\tau)$ 为测试调试时间；$E_{d(\tau)}0$ 为 $f(\tau)$ 时间内发现的错误数；$E_{c(\tau)}0$ 为 $f(\tau)$ 时间内修正的错误数。

单位程序长度里的故障数 E_T/I_T 近似于常数 0.5%～2%。

当对发现的故障进行改正且在改正过程中没有植入新的故障，则有：

$$E_{d(\tau)}=E_{c(\tau)}$$

剩余故障数：

$$E_{r(\tau)}=E_T-E_{c(\tau)}$$

单位程序长度剩余故障数：

$$E_{r(\tau)}=E_{r(\tau)}/I_T=（E_T-E_{c(\tau)}）/I_T$$

则平均无故障时间：

$$\text{MTTF}=\frac{1}{KE_{r(\tau)}}=\frac{I_T}{K(E_T-E_{c(\tau)})}$$

其中，K 为经验常数。

2．平均无故障间隔时间 MTBF(mean time between failure)

MTBF 是一个与 MTTF 相关的量，是指两次故障之间的平均时间，实际使用时，对上面的公式而言，通常是指当 n 很大时，系统内第 n 次故障与第 n+1 次故障之间的平均时间。

11.1.3　软件生存期与软件寿命

一切有生命的东西都有“寿命”，也就是从出生开始到死亡为止的时间。作为信息产品的软件也一样存在着“寿命”。与许多有“寿命”的东西不同的是，软件不存在物理磨损。

从软件工程的角度说，软件产品的寿命是指软件的整个生存期，这个生存期包括软件规划、需求分析、设计、编码、测试、运行、维护等。

从用户的角度讲，用户更关心的是软件的使用情况，因而从可靠性的角度理解软件的寿命成分更多一些，或者说用户关心的是软件的寿命，用户把软件产品能够可靠地完成任务作为软件寿命长短的标准。如 MTTF 和 MTBF 可能更是用户关心的。

当然，与软件的寿命有关的因素很多，除了与系统本身及环境有关外，还与评价的标准有关。

由此可见，由于对软件产品的关注内容的不同，决定了软件生存期和软件寿命存在的差异，主要体现在以下 3 个方面。

（1）从软件的过程而言软件生存期主要包括 3 个阶段，即软件定义、软件设计开发、软件运行维护阶段。而软件寿命是描述软件在使用期内执行规定任务的一个供参考的可靠性指标。

（2）软件产品的可靠性直接影响软件的寿命。而可靠性如何是由软件生存期的各阶段的工作质量决定的。从图 11-1 可以看出，在软件生产的各个阶段产生的错误中，设计阶段产生的错误占大多数。同样从图 11-2 可以看出，修正一个软件错误所需的费用随着软件生存期的进展而上升。也就是说，错误发现得越早，修正错误所需的费用就越少。

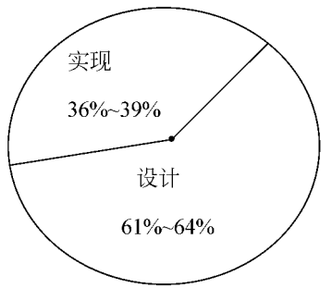


图 11-1　软件生存期错误来源

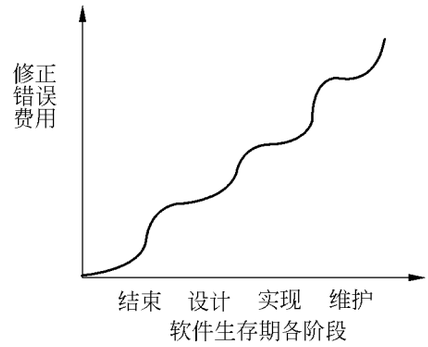


图 11-2　修正错误所需费用

（3）软件生存期是软件工程术语。软件寿命是软件可靠性工程术语，软件生存期中靠软件开发质量管理来提高软件的可靠性，着重开发过程；而软件可靠性工程偏重于可靠性的实际提高和可靠性的评估。

为了保证软件的可靠性，在软件生存期的各个阶段内，都需要采取一些可靠性措施。如分析阶段采用先进的结构化的分析技术，设计阶段采用结构化的设计技术以及结构化的程序设计技术等等。

11.2　软件可靠性评估

如何评估软件的可靠性，是软件工程的一个一直以来没有根本解决的课题，由于涉及的内容以及技术方面存在的问题，使得人们在评估软件可靠性时，显得力不从心，以至于到现在也没有一个可以从根本上解决的办法。在实践中，人们建立了各种模型及相关的模型算法，为可靠性的评估提供了一个有效的途径。

11.2.1　软件可靠性模型

软件的可靠性可看成是随机过程，用概率来描述。软件的可靠性有自己的特点，一

方面，软件本身没有物理损耗，不会随着时间的增加而增加或减少可靠性；另一方面，软件的故障常常是由于系统分析和系统设计引起的，这就使得对软件的可靠性分析变得非常复杂，自第一个软件可靠性模型提出以来，人们在实践中使用的模型已经有几十种，但始终没有一个通用的可靠性模型对所有的软件产品能做出最好的可靠性分析。因而，对软件的可靠性模型研究还有待于继续深入。

下面介绍几个较简单的模型。

1．Jelinski-Moranda 模型

Jelinski 和 Moranda 开发的可靠性模型是最早建立且一直被人们使用的模型之一，模型的基本假设如下：

（1）检测错误率与当前软件的错误数成正比；

（2）在错误发生间隔期间检测错误率为常数；

（3）出现错误即刻修正，不使新的错误引入软件；

（4）软件运行方式与可靠性预测方式相同；

（5）每个错误出现机会相等，且所有错误的严重程度相同；

（6）错误被查出时，失效是独立的。

模型的数据需求实现如下：

软件失效时间间隔分别为：$x_1, x_2, x_3, \cdots, x_n$，那么有：

① 累计失效函数 $m(t)$

$$m(t) = N \times (1 - \exp(-K \times t))$$

② 失效率函数 $\lambda(t)$

$$\lambda(t) = N \times K \times \exp(-K \times t)$$

2．Shooman 模型

Shooman 模型的基本思想是失效率与软件中的错误总数成一定比例，且失效率为常数，而且随着错误的消除，失效率也随着相同比例下降。

模型的基本假设如下：

（1）失效率与软件的潜在的错误成比例；

（2）所有失效发生的概率相同，彼此独立；

（3）所有错误的严重程度相同；

（4）软件运行方式与可靠性预测方式相同；

（5）引起失效的错误被及时修正，在修正错误时没有新的错误被引入。

模型数据：失效时间间隔 x_i 和失效时刻 t_i，其中 $x_i = x_i - t_{i-1}$。

假设：E_T ——测试前程序中故障总数；I_T ——机器指令总数；τ ——时间；$E_r(\tau)$ ——在 τ 时刻潜在的错误数；$E_d(\tau)$ ——在 0～τ 期间的错误数；$E_c(\tau)$ ——在 0～τ 期间修正的错误数。

由模型的基本假设得到：

（1） $E_c(\tau) = E_d(\tau)$ 引起失效的错误被及时修正。

（2） $E_r(\tau) = E_T - E_c(\tau)$ 修正错误时没有新的错误被引入。

令 $\varepsilon_r(\tau) = E_r(\tau) / I_T$，$\varepsilon_c = E_c(\tau) / I_T$ 则：

$$\varepsilon_r(\tau)=E_T/I_T-\varepsilon_c(\tau)$$

（3）$\lambda=\mathrm{k}\times\varepsilon_r(\tau)=\mathrm{k}\times(E_T/I_T-\varepsilon_c(\tau))$，k 为常数。

（4）MTTF=$1/\lambda$。

11.2.2 估算软件中错误的方法

软件可靠性模型有效地解决了软件可靠性评估的问题，但如何估算软件中的错误，方法也是多种多样，这些方法都是人们在实践中的总结，这种实践基于人们在软件系统分析和设计中出现的问题，由于分析和设计的思路不同，使得方法不统一。下面介绍几种估算软件中错误的方法。

1．Shooman 模型估算法

可以利用 Shooman 模型，估算程序中原有错误的总量 E_T。在这里引入故障率 λ，且定义 λ 是 MTTF 的倒数，由 MTTF 的概念有：

$$\mathrm{MTTF}=\frac{I_T}{K(E_T-E_{c(\tau)})}=\frac{1}{\lambda}$$

所以，
$$\lambda=K\left(\frac{E_T}{I_T}-\frac{E_c(t)}{I_T}\right)$$

若设 T 是软件总的运行时间，M 是软件在这段时间的错误次数，则：

$$\frac{T}{M}=\frac{1}{\lambda}=\mathrm{MTTF}$$

现在对程序进行两次不同且相互独立的功能测试，相应的查错时间 $\tau_1<\tau_2$，查出的错误数 $E_c(\tau_1)<E_c(\tau_2)$，则有：

$$\lambda_1=\frac{E_c(\tau_1)}{\tau_1}=\frac{1}{\mathrm{MTTF1}}=K\left(\frac{E_T}{I_T}-\frac{E_c(\tau_1)}{I_T}\right)$$

$$\lambda_2=\frac{E_c(\tau_2)}{\tau_2}=\frac{1}{\mathrm{MTTF2}}=K\left(\frac{E_T}{I_T}-\frac{E_c(\tau_2)}{I_T}\right)$$

解方程组，得到错误数 E_T 的估算值 $\hat{E}_T$ 和 K 的估算值 $\hat{K}$：

$$\hat{E}_T=\frac{E_c(\tau_2)\lambda_1-E_c(\tau_1)\lambda_2}{\lambda_1-\lambda_2}$$

$$\hat{K}=\frac{I_T\lambda_1}{E_T-E_c(\tau_1)}\ \text{或}\ \hat{K}=\frac{I_T\lambda_2}{E_T-E_c(\tau_2)}$$

2．最小二乘法估算法

同样可以利用最小二乘法来计算 E_T 和 K 的值。

由故障率

$$\lambda=K\left(\frac{E_T}{I_T}-\frac{E_c(t)}{I_T}\right)$$

整理得：

$$\frac{E_c(t)}{I_T} = \frac{E_T}{I_T} - \frac{\lambda(t)}{K}$$

在这里看成是关于 t 的函数。

若对程序进行若干次不同的功能测试，得到一系列实验数据如下：

$$(E_c(\tau_i)，\ \lambda(\tau_i))，\ i=1,2,3,\cdots,n$$

画出函数 $E_c(t)$ 的曲线，发现这条曲线近似于一条递减的直线，用各组实验数据去与图形中最相近的点比较，从而估算出 E_T 和 K 的值。

令　　$\frac{E_c(\tau_i)}{I_T}=\varepsilon_i$　　$\lambda(\tau_i)=\lambda_i$　　$-\frac{1}{K}=a$　　$\frac{E_T}{I_T}=b$

如图 11-3 所示，得到一方程组 $\varepsilon_i = a\ \lambda_i + b$

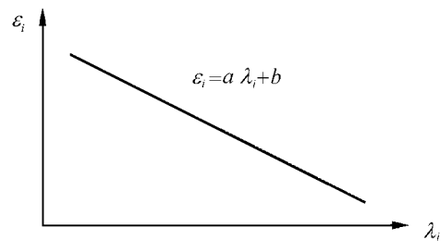


图 11-3　比较曲线

用最小二乘法解此方程，可得到 a，b 的值：

$$\hat{a}=\frac{\begin{vmatrix} XY & Y \\ X & n \end{vmatrix}}{\begin{matrix} YY & Y \\ Y & n \end{matrix}}　　\hat{b}=\frac{\begin{vmatrix} YY & XY \\ Y & X \end{vmatrix}}{\begin{matrix} YY & Y \\ Y & n \end{matrix}}$$

式中　　$XY=\sum_{i=1}^{n}\varepsilon_i\lambda_i$　　$YY=\sum_{i=1}^{n}(\lambda_i)^2$　　$Y=\sum_{i=1}^{n}\lambda_i$　　$X=\sum_{i=1}^{n}\varepsilon_i$

则　　$\hat{K}=-\frac{1}{\hat{a}}$，　　$\hat{E}_T=\hat{b}\times I_T$

3．植入错误估算法

植入错误（也叫播种错误）法是利用测试前人为地植入的错误估算错误的方法。

若设 N_S 是在测试前向程序中植入的错误数，n_s 是经过一段时间测试后发现的植入错误数，n_0 是在测试中新发现的程序固有错误，假设有测试用例发现植入错误和固有错误的能力相同，则程序中固有错误数 $\hat{N}_0$ 为

$$\hat{N}_0=\frac{n_0\times N_S}{n_s}$$

作为对植入错误法的补充，还有一种分别测试法，叫 Hyman 分别测试法。原理是，由两个测试员对同一程序的两个副本进行测试，用 t 表示测试时间，记 $t=0$ 时程序中的错误总数 B_0，$t=t_1$ 时，测试员甲发现的错误数为 B_1，测试员乙发现的错误数为 B_2；其

中两人发现的相同错误数目为 b_1，不同错误数为 b_2。如果 b_2 不是很大，这时有

$$B_0 = \frac{B_1 \times B_2}{b_1}$$

如果 b_2 比较显著，则需要每隔一段时间进行一次，且重复，直至几次估算结果相近。

11.3 软件可靠性技术

解决软件可靠性的第一步是有效地对软件可靠性进行评价。接下来就要利用技术的手段提高软件的可靠性。这涉及两个方面；一是从模型本身入手，模型的质量如何？在很大程度上取决于模型的算法，而算法的模型化使得建模从技术上有了较大的进步。二是从分析和设计等阶段入手，利用技术的手段，使软件在设计开发过程中尽可能少地出现错误或即使出现错误也不会对软件功能产生影响。下面从以上所提到的两个方面入手进行讨论。

11.3.1 算法模型化

对于上面介绍的模型，是针对不同的需求给出的，其算法也是在假定的条件下得到的。前面的讨论中已经给出了一些算法模型，下面给出预测失效时间的算法模型。

1．预测失效时间模型

根据已有的经验数据，有效地预测下次失效时间，对软件的现实意义是不言而喻的。因而将预测算法的模型化就显得较为重要，可以方便地进行预测，从而有效地预防和采取措施。

根据 J.M 模型的假设，可以得到以下两个方程：

$$K = \frac{n}{N \times \left(\sum_{i=1}^{n} x_i \right) - \sum_{i=1}^{n} (i-1) x_i}$$

$$\sum_{i=1}^{n} \frac{1}{\left[N - (i-1) \right]} = \frac{n}{\left\{ N - \left[\sum_{i=1}^{n} (i-1) \times x_i \right] \right\}} \Big/ \sum_{i=1}^{n} x_i$$

解方程组可以得到 N 和 K 的值。

那么，第 n+1 次失效等待时间 MTTF′

$$\mathrm{MTTF}' = \left[(N-n) \times K \right]^{-1}$$

例如：假设对某系统已观测到以下的失效时间间隔：

$x_1 = 7, x_2 = 11, x_3 = 8, x_4 = 10, x_5 = 15, x_6 = 22, x_7 = 20, x_8 = 25, x_9 = 28, x_{10} = 35$ 试求第 11 次失效的失效发生时间间隔。

由本节介绍的方程组求得：N=11.6　K=0.0096，

从而可得到：MTTF′=65.1。

若以小时为单位，则表示下次失效发生在第 10 次失效（当前失效）修复后的 65.1 小时。

2．估算程序中错误总数模型

对于程序中错误总数的估算，在上一节中已经做了介绍，不再重复。

11.3.2　软件容错技术

提高软件的可靠性的技术大致可分为两类：一类是避开错误技术，即在开发过程中不让错误潜入软件的技术；另一类技术是容错技术，即对某些无法避开的错误，使其影响减小到最低程度的技术。避开错误的技术属于质量管理，是实现产品应有质量所必不可少的。但是，无论管理水平多么高超，也无法做到完美无缺和绝对无错误，这就需要采取容错技术使得错误发生时不影响系统的功能和特性。

1．基本概念

什么是容错软件？一般认为，如果在一定程度上软件对自身的错误具有屏蔽能力或能从错误状态自动恢复到正常状态或即使在错误发生时也能完成预期功能，把这种软件称为容错软件。容错软件都具有以下特点。

（1）容错的对象是一个规定功能的软件，这些功能都是由需求规格说明书定义的。

（2）软件的容错能力有一定的限度。

（3）容错的过程一般采用对算法进行的检测，通过冗余的手段实现。

2．容错的一般方法

实现容错的手段主要是冗余。冗余是指所有对于实现系统功能的软件来说，多余的那部分资源，包括硬件、软件、时间、信息等。通常冗余技术分为 4 类。

1）结构冗余

结构冗余是常用的冗余技术，由于工作方式的不同，分为静态冗余、动态冗余、混合冗余。

静态冗余通过表决和比较来屏蔽系统中的错误，常用的方式是一个三维表决系统，如图 11-4 所示，既可以由硬件实现，也可以用软件实现。

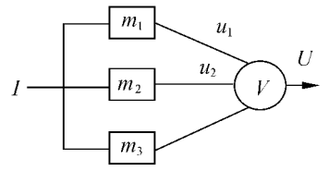


图 11-4　三维表决系统

其输出为：$U=(u_1 \wedge u_2 \wedge) \vee (u_2 \wedge u_3) \vee (u_1 \wedge u_3)$

根据表决的定义，对于软件，3 个功能相同但由 3 个人采用不同的方法开发出来的模块的运行结果通过表决，以多数的结果作为系统的输出结果。显然，当一个模块出错时，这个错误的结果在表决时被其他两个正确的结果“屏蔽”掉了，并不会影响系统输出正确结果。由于无须对错误进行特别的测试，也不必有其他的动作，因而称其为静态冗余。

与静态冗余不同，动态冗余是用多重模块待机储备，相继运行来维持系统的正常工作。当检测到工作的模块出现错误时，就用一个待用模块来替代错误模块重新运行。在这里，系统要进行一系列检测、切换和恢复的过程，故称为动态冗余。结构图如 11-5 所示。

图 11-5 中，M_1，M_2，M_3，…，M_n 是 n 个具有相同功能的不同模块，它们是各自独立设计的，M_1 是主模块，其他称为备用模块。当 M_1 出错时，由 M_2 顶替 M_1 工作，M_3，…，M_n 成为待用模块，只有当 M_1，M_2，M_3，…，M_n 相继出现错误时，系统才失效。

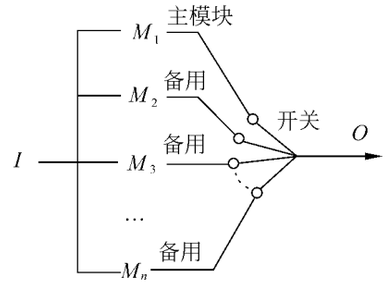


图 11-5　待机储备系统结构

混合冗余兼有静态冗余和动态冗余的优点。系统中的待用模块采用三维表决系统，使得系统的可靠性大大提高，这里不作说明。

2）信息冗余

信息冗余是为了检测或纠正信息在运算或传输过程中的错误而外加的一部分信息，在通信和计算机系统中，信息通常以编码的方式出现。采用奇偶码、定重码、循环码等冗余码制式就可以发现甚至纠正这些错误。其中循环码也叫循环校验码，简称 CRC。下面重点介绍循环码的形成过程。

如果要传送的信息长度为 k 位，则可用 k -1 阶代码多项式 $M(x)$ 表示，指定一个 r 阶多项式，则存在一个校验多项式 $G(x)$。

循环码的算法如下：

（1）用 $M(x)$ 乘以校验多项式 $G(x)$ 的最高次项 $M(x) \times x^r$，也就是在信息的低位端添加 r 个 0 形成 n 位的信息。

（2）用模 2 运算（即 1+1=0，1–1=0，无借位、无进位）实行 $x^r \times M(x)/G(x)$ 得余数多项式 $R(x)$。

（3）用模 2 减法实行 $x^r \times M(x) - R(x)$，令 $T_s(x) = x^r \times M(x) - R(x)$，即为发送信息的码多项式。

显然 $T_s(x)$ 能被 $G(x)$ 除尽（模 2），因为在除法运算中，被除数减去余数后一定能被除数除尽。接收时，将接收到的信息码多项式 $T_r(x)$ 进行同样算法，即接收到的信息码多项式 $T_r(x)/G(x)$。若 $R(x)=0$，则确认数据无差错；若 $R(x)\neq 0$，则表示有错，需要重发。

例如：$M(x)=x^7+x^4+x^2+x$，$G(x)=x^6+x^4+x^2+1$，进行上述运算得到如下算式。

生成多项式：　$G(x)=x^6+x^4+x^2+1$

发送数据：　$M(x)=x^7+x^4+x^2+x$

变形数据：　$x^6\times M(x)=x^{13}+x^{10}+x^8+x^7$

循环码：　$x^6\times M(x)/G(x)$ 的余数多项式 $R(x)=x^3+x^2+x+1$

发送数据：$T_s(x)=x^6\times M(x)-R(x)=x^{13}+x^{10}+x^8+x^7+x^3+x^2+x+1$

接收数据：　$T_r(x)=x^{13}+x^{10}+x^8+x^7+x^3+x^2+x+1$

无差错：　$T_r(x)/G(x)$ 的余式 $R(x)=0$

下面有两个多项式 $G(x)$ 作为国际标准校验码多项式

$$\text{CRC–12}=x^{12}+x^{11}+x^3+x^2+1$$

$$CRC\text{–}16= x^{16} + x^{15} + x^2 + 1$$

3）时间冗余

时间冗余是以重复执行指令或程序来消除瞬时错误带来的影响，其过程描述如图 11-6。

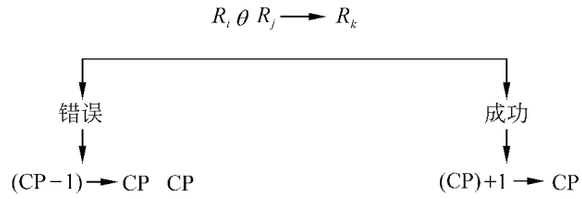


图 11-6　指令重复执行过程

图 11-6 中，R_i 和 R_j 为源地址，R_k 为目的地址，θ 为操作码，CP 为指令计数器。即 R_i 和 R_j 的内容经过 θ 运算后，其结果送入 R_k 中，如果这时有错误恢复请求，则指令计数器内容减 1，重新执行该指令，从而消除错误。如果目的达到（恢复成功），则指令计数器内容加 1，继续执行下一条指令。否则，说明重复不成功。

4）冗余附加技术

冗余附加技术是指为了实现上述冗余技术所须的资源和技术，包括程序、指令、数据等。在屏蔽软件错误的容错系统中，冗余附加技术包括：

（1）独立设计的功能相同的冗余备份程序的存储和调用；

（2）实现错误检测和错误恢复的程序；

（3）为实现容错软件所须的固化程序。

3．容错系统的设计过程

容错系统的设计过程如图 11-7 所示，包括以下设计步骤：

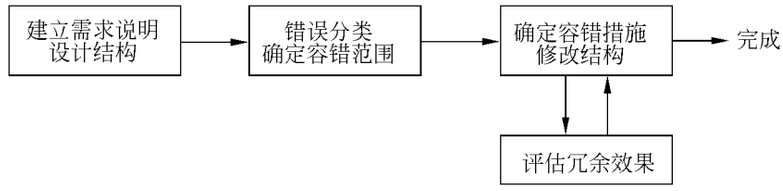


图 11-7　容错系统的设计过程

本 章 小 结

软件的可靠性表明了一个程序按照用户要求和设计目标，执行功能的正确程度。本章从软件可靠性基本概念入手，对软件可靠性模型、估算方法进行了说明，为软件质量分析提供了量化的方法。软件的容错技术是有效提高软件可靠性的技术手段。

习 题 11

11.1 什么是软件的可靠性？度量软件可靠性的主要指标有哪些？

11.2 什么是软件的容错技术？主要方法有哪些？

11.3 结构冗余中，都存在由多人开发且功能相同的模块，请问：这样做会不会使软件成本成倍增加？为什么？

11.4 软件的生存期与软件寿命有什么区别？

11.5 假设有一个过程监控系统，通过试运行观测到以下失效发生时间间隔：

$x_1=10, x_2=8, x_3=4, x_4=17, x_5=15, x_6=22, x_7=19, x_8=25, x_9=27, x_{10}=42$，

试估算下次失效的发生时间。

11.6 假设有甲、乙测试员分别对含有 5 万条程序代码的某管理系统进行测试，其中甲在 4 周内发现了 50 个错误，乙在 8 周内发现了 60 个错误，假定甲乙两人同时开始测试且测试过程互不相关，试估算整个系统中的错误数？平均失效等待时间是多少？

11.7 假设有一段 8 位的信息：10010111，拟采用循环码制式的冗余方法检测和纠正传输过程中可能出现的错误，试写出其循环码。

11.8 思考题：从软件可靠性角度，如何理解“分阶段”的重要性和必要性。

第12章

软件工具及环境 30~33

工欲善其事，必先利其器。在软件开发过程中，软件工具是支持软件工程的重要基础，是软件工程方法的具体实现，是软件工程理论与实践的最终体现。随着人们对软件功能需求的提高，软件开发的复杂度也愈来愈大，因此，高效率的软件开发工具和良好的软件开发环境将为软件生产率的提高和软件质量的改进提供重要的保证。

软件工具的发展经历了单一功能的软件工具阶段、软件开发环境阶段，以及向计算机辅助软件工程（CASE）方向发展的历程。

在20世纪70年代，软件开发与设计方法在由结构化程序设计技术（SP）向结构化设计（SD）技术发展过程中出现了第一代的基于正文的工具，用以支持单个的过程任务，如检查设计的一致性工具，编译系统、语法制导编辑器等，而后软件开发与设计方法又形成了基于结构化分析技术的一整套相互衔接的SA-SD的方法学，与此相应的计算机辅助软件工程技术则主要由开发孤立的软件工具而逐步向程序设计环境的开发和使用方向发展，如支持程序设计的编程环境的IDE（integrated development environment）等。

在20世纪80年代中期与后期，主要是实时系统设计方法以及面向对象的分析和设计方法的发展，克服了结构化技术的缺点。在这期间，人们将独立使用的软件工具集成于一个软件工具平台上，形成具有较强功能的软件开发环境，其特点是支持使用图形表示的结构化方法，如数据流图与结构图。其开发环境表现在提高环境中工具的集成性方面，如“集成的项目支持环境”，将详细的开发信息存放在“项目词典”中，以便在同一环境中其他的软件工具可以共享。

到了1986年正式使用CASE来称谓“支持软件工程方法学的计算机辅助手段”，使得软件开发环境成为用于项目计划、分析、设计、编程、测试和维护的一个工具箱的集合。实现了软件系统系统开发工作的计算机自动化，提高了系统开发的效率与质量、降低投资成本和开发风险。 从认知方法论的角度来看，CASE技术，不仅为软件开发提供工具环境还从软件开发方法上支持软件的开发过程，如支持结构化方法的全面实施、为原型的建立

提供高效率的手段等，实现了系统分析、系统设计、编码及软件测试与维护等开发过程以及项目管理、文档生成管理等开发工作的规范化、工程化和自动化。

12.1 软 件 工 具

软件工具是为了提高软件开发的质量和效率，从软件计划、分析、设计、测试和调试、运行维护和配置管理各方面，为软件开发者提供的计算机程序或软件系统。如用于方法中的符号编辑器、代码生成器、根据方法规则检查系统模型的分析模块以及生成软件文档的报告生成器等；具有代表性的工具有美国 Westem Michigan University 的需求分析工具 PSL/PSA，支持软件设计的 PDL 工具，静态分析工具 DAVE 等。一般来讲，把支持系统分析和设计的软件工具称为高端的软件工具，把仅支持系统实现和测试的工具，如调试程序、测试程序和程序编辑器等称为低端的软件工具。

12.1.1 软件工具的作用与功能

随着软件工具的大量使用，使得软件开发的效率大大提高，软件工具在软件开发工程中的作用愈来愈大，软件工具的功能、对方法的支持和可用性等方面都直接体现了软件生产力发展的程度。

如何正确的选择和使用软件工具，如何确定软件工具在开发过程中的辅助地位，最大限度的发挥软件工具的作用，一直是软件开发过程中所面临的问题；由于软件开发的过程中蕴藏着极大的不确定因素，因而没有任何软件工具能包罗万象，软件开发的方法和技术也远没有达到自动化的程度，对软件工具的评价目前还没有定性的方法来衡量，所以对软件工具的选择和使用要根据具体的软件项目以及所采用的软件开发方法、项目管理机制等因素来确定，合理的使用软件工具，既要看到软件工具的作用又不能夸大软件工具的功能而过分的依赖软件工具。

支持软件工程的工具软件，一般应该具备认识与描述客观系统、存储与管理开发过程的信息、代码的编写与生成、文档的编辑或生成、软件项目的管理等功能。

1．认识与描述客观系统

协助开发人员认识软件工作的环境与要求、合理地组织与管理系统开发的工作过程。

2．存储及管理开发过程中的信息

系统开发中产生大量的信息，结构复杂，数量众多，由工具提供一个信息库和人机界面，有效地管理这些信息。

3．代码的编写、生成、测试

通过各种信息的提供，使用户在较短时间内，半自动地生成所需的代码段，进行测试和修改。

4．文档的编制或生成

包括文字资料、各种报表、图形等文档。

5．软件项目的管理

项目管理包括：进度、资源与费用和质量的管理。

6．配置管理

完成软件配置的标识、版本控制等。

12.1.2　软件工具的分类

软件工具种类繁多且涉及到软件工程的各个开发阶段以及支持不同的软件工程方法，从不同的角度对软件工具进行分类，对帮助开发人员评价、选择和使用软件工具具有较现实的指导意义。工具的分类也引起了人们的研究热潮，出现了较流行的分类结果，如 Reifer 和 Trattner 将软件工具分为：模拟工具、开发工具、测试和评价工具、运行和维护工具、性能测试工具和程序设计支持工具等 6 类；也可以按如下分类。

1．按照用途分

支持项目管理的工具：提供为项目管理人员使用，用于项目进度控制、成本管理、资源分配、质量控制等功能。

支持软件分析与设计的工具：支持需求分析、设计、编码、测试、维护等软件生命周期各个阶段的开发工具和管理工具。

支持程序设计的工具：操作系统、编译程序，解释程序和汇编程序等。

2．按照界面划分

支持字符界面的工具。

支持图形用户界面（graphics user interface，GUI）的工具。

3．按照软件生存周期的阶段划分

系统计划工具——针对软件开发的全过程，跨生命周期地管理软件项目。如版本管理软件、项目进度管理软件等。

需求分析工具——主要用于支持需求分析的工具，包括描述工具和分析工具，如数据字典管理系统、绘制数据流程图的专用工具、绘制系统结构图或 E-R 图的工具等。

系统设计工具——包含系统变换工具和系统描述工具，如用于描述的工具：流程图，判定表，程序描述言语（PDL）等。

支持编码的工具——支持代码生成的软件工具，包括编辑系统、汇编程序、解释和编译系统等。

测试和调试工具——包括测试工具和调试程序，如测试用例生成器、测试系统、调试诊断程序、跟踪程序等。

运行和维护工具——包括系统运行配置工具以及支持系统维护的工具等。

文档管理工具——生成和管理系统文档的工具。

此外，还可以按照系统平台、按照集成度等特征来划分软件工具的种类。

12.2　软件开发环境

“软件开发环境是相关的一组软件工具的集合，支持一定的软件开发方法或按照一

定的软件模型组织而成的”。这是 IEEE 和 ACM 支持的国际工作小组于 1985 年在第 8 届国际软件工程会议上提出的定义。该定义指明了软件开发环境不仅集成了相关的软件工具，还按照一定的开发模型和方法有机的构成，并且贯穿于整个软件生存周期的各个阶段。软件环境的形成能更好的支持软件开发的整个过程，从而进一步提高软件的生产率，更有效的保证软件的质量。

12.2.1 软件开发环境的分类

软件开发环境可根据软件的生存周期、软件开发方法、开发模型等特征作如下分类。

1．按照解决的问题分类

可分为程序设计环境、系统综合环境和项目管理环境。

2．按照软件开发环境的发展趋向分类

可分为以语言为中心的环境、专用工具箱环境和基于方法的环境。

3．按照集成化的程度分类

可分为基于操作系统的环境、基于数据库系统的环境和基于知识库系统的环境。

4．按照集成的机制分类

数据集成——工具间可以交换数据。

界面集成——将工具的界面集成在一起，具有相同的界面和交互方式。

控制集成——同时在不同的工具之间进行各种控制功能。

过程集成——系统嵌入软件工程的知识，根据软件模型来辅助用户的开发。

平台集成——工具运行与相同的软硬平台的融合。

综合集成——在一个环境中全部或部分集成了上述机制。

12.2.2 软件开发环境的特点

软件开发环境就是将以前单一功能的软件工具集成到统一的软件平台之上，以进一步提高各个软件工具之间的信息共享程度、优化软件工具的通用性以及提高使用效率，方便用户的使用。具体来讲，就是在一个软件开发环境中高度集成支持需求分析工具、软件设计工具、编码工具、测试工具、运行维护工具、文档管理工具以及项目管理工具等，更有效地支持软件的开发过程。

软件开发环境必须具有整体性、层次性、扩充性以及移植性等特点。

(1) 整体性：此时的软件工具不再是具有单一功能的个体程序，而是高度集成的相互关联的完整的软件系统，支持软件开发过程中的每一个阶段。

(2) 通用性：软件开发环境具有一定的通用性，特别是针对相同或相近领域的项目，在系统开发过程中的不同层次上以及不同阶段中也具有通用性。

(3) 层次性：软件开发环境有较分明的层次结构，如宿主层、核心层、基本层和应用层等。

(4) 扩充性：软件开发环境中的工具程序在功能上必须具有较强的内聚性，但要保证工具之间的较低的耦合性，使得工具组可以随时地进行功能或性能等方面的修改，使

系统具有一定的灵活性。

（5）移植性：软件开发环境可以方便的在不同的计算机平台之间进行转移。

12.2.3　软件开发环境的组成与结构

软件开发环境应该包括一组软件工具、接口程序、语言支持系统以及环境数据库等部分。欧洲计算机制造商联合会（european computer manufacturers association，ECMA）在 20 世纪 90 年代初提出了软件开发模型的参考模型。该模型建议，软件开发环境通常要提供 5 类核心服务。

（1）数据存储服务——用于命名和管理实体，建立实体间的联系；

（2）数据集成服务——扩展数据存储服务，包括版本管理、配置管理、状态控制和数据交换等服务，以适用于软件开发；

（3）任务管理服务——完成项目定义、任务执行、事务恢复、事件监视、角色管理以及历史信息管理等工作；

（4）消息服务——提供软件工具与软件环境间的通信服务，包括消息传递、工具注册；

（5）用户界面服务——支持用户界面的集成。

ECMA 参考模型如图 12-1 所示。

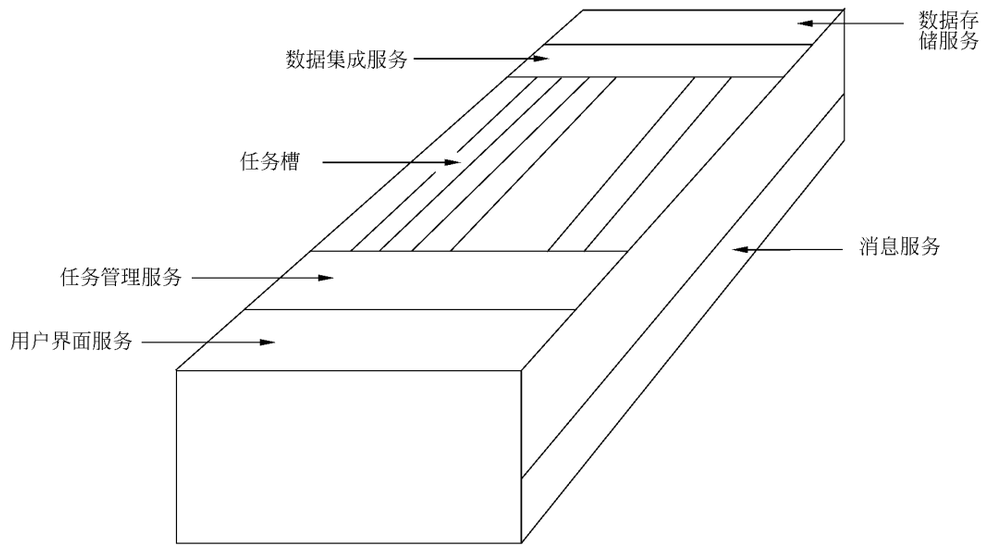


图 12-1　ECMA 参考模型

ECMA 参考模型在我国有成功的原型系统，如北京大学的青鸟系统。

12.3 计算机辅助软件工程（CASE）

软件开发环境为软件项目的开发提供了巨大的支持和保障，软件工具及软件开发环境的不断使用和进步，便形成了一整套研究和开发软件工具的理论、方法和技术，这就是计算机辅助软件工程，简称 CASE（computer-aided software engineering）。

12.3.1 CASE 的概念及现状

由于 CASE 是从计算机辅助编程工具、4GL（第 4 代程序生成语言）发展而来的大型综合计算机辅助软件工程开发环境，因此，CASE 可以进行各种需求分析、功能分析，生成各种结构化图表，如数据流程图、结构图、实体-关系图、层次化功能图和判定表等，并能支持系统开发整个生命周期。软件开发环境的概念也从具体的工具发展成为热门学科，它是一种从开发者的角度支持信息系统各种开发技术和方法（如结构化方法、快速原型法、面向对象方法）的计算机技术。

早在 1986 年正式使用 CASE 称谓“支持软件工程方法学的计算机辅助手段”前，软件工具就已大量出现了。可以说，软件工程从诞生起就面临着“如何组织开发人员进行大规模团队作业、如何逐步利用计算机代替开发人员进行作业”等两大课题。并且首先必须解决前者——建立有序的、高效率的、科学的工程作业方法，然后才能一步步用计算机取代各工程阶段的人工作业。

CASE 的演进是按孤立型→扩充型→接口型→统一型来发展的。

孤立型：20 世纪 80 年代初，包括文档生成自动化，图表生成自动化，分析设计辅助工具。

扩充型：20 世纪 80 年代中，包括分析设计自动化，一致性检查，建立系统信息中心库。

接口型：20 世纪 80 年代后，包括根据设计规格自动生成源代码，程序维护工具，项目管理工具。

统一型：20 世纪 90 年代，包括与 AI（人工智能）技术结合，与重用技术结合，面向整个软件工程。

CASE 环境的演进，刚开始是以工具的形式出现，例如一些图形语言的软件工具、界面与报表的制作工具、数据字典、数据库管理系统、简单的规范语言检查工具及程序生成器等。在近几年来，CASE 的发展已由零星的软体工具发展到整合的软件开发环境。

12.3.2 CASE 技术的功能及组成

由于不同领域中对软件功能需求具有多样性，因而软件系统的结构及复杂性也不尽相同，从而也产生了不同的软件开发方法及开发过程，CASE 环境既要为不同领域的软件开发提供普遍的功能支持，又要为特殊的需求提供具有针对性的功能，以满足不同的开发需要。

CASE 功能通常可分为下列 3 个部分。

（1）CASE 的过程：即软件开发过程模式。

（2）CASE 的方法论：如分析与设计的方法论，像 DFD、DD、ER、STD 和 OO 方法等。

（3）CASE 的环境：即一般的软件开发环境，软件开发环境是一个建立在操作系统上面的软件架构，提供了多种软件工具来支持整个生命周期中的各项软件工程活动。

目前的 CASE 有许多的存在形式，如高端工具、低端工具、生命周期管理工具和系统集成软件开发工具等，具体如下：

高端工具包括软件开发生命周期中系统分析、系统设计各阶段工作所需要的工具。如数据流图（data flow diagram）、Mini 规格说明书（miniI spec）、数据字典（data dictionary）和实体关系图（entity-relationship diagram）等。上述工具提供的图形及文字功能为系统概念的描述和信息的完整性与一致性提供了保证。

低端工具是指支持软件生命周期后半段的编码、测试及维护等阶段的工具。包括系统原型的构建工具（system prototyping）build、代码生成器（program generator）、文档生成工具（document generator）及测试工具（testing tool），上述工具能有效地帮助开发人员设计和自动构造系统程序代码。

生命周期管理工具是指软件开发各阶段的管理工具。包括项目管理工具（project management tool），为项目管理人员提供从不同角度观察项目的现状，作为系统审查和决策的依据；成本估算工具（cost estimate tool）；建构管理工具（configuration management tool），包括需求追踪、变动管理、系统建构管理等内容；版本控制工具（version control tool）及知识库管理工具（repository manager tool）；另外，还有逆向软件工程工具（reverse engineering tool）等。

集成的软件开发工具是指支持软件开发生命周期各阶段的集成化开发环境。包括用户集成，系统开发过程集成，工具集成，开发组织的集成及管理上的集成等。

12.3.3　CASE 工具分类及特点

对 CASE 工具的分类可以帮助开发人员了解不同类型的 CASE 工具的作用及其任务，站在不同的角度可以产生不同的分类方法，下面是两种不同的分类方法：

1．从功能的角度分类

规划工具：PERT 工具、估算工具、电子表格工具。

编辑工具：文本编辑器、图表编辑器、文字处理器。

配置工具：版本管理、系统建立工具。

原型建立工具：高端语言、用户界面生成工具。

变更工具：需求跟踪工具、变更控制系统。

方法支持工具：设计编辑器、数据字典、代码生成器。

语言处理及分析工具：编译及解释系统、静态及动态分析器。

测试工具：测试用例生成工具、调试程序。

文档工具：图像编辑器，文字工具等。

2．从对软件过程支持的广度分类（Fuggetta 于 1993 提出）

工具：用以支持单个任务，如检查设计的一致性、程序编译、比较测试结果等。这些工具既能独立使用又可以集成到系统平台上，如编辑器等。

工作平台：用以支持描述、设计等过程阶段和活动，一般是由集成度不同的工具组成。

环境：用以支持全部软件过程或至少是软件过程的主要部分，通常包括某种方式集成的若干个工作平台。

前面讨论过，CASE 工具首先支持不同的软件开发方法；其次支持软件开发生命周期的各个阶段。通过一系列集成化的软件工具、技术和方法，使整个计算机信息系统的开发自动化。CASE 方法与其他方法相比一般地来说有如下几方面的特点。

（1）提高信息系统的开发效率和开发质量，加快信息系统的开发进程，有效地降低信息系统的开发费用。

（2）解决从客观世界对象到软件系统的直接映射，强有力地支持软件系统开发的全过程，使结构化方法更加实用，使原型法方法和 OO（面向对象）方法易于付诸实施。

（3）实现系统设计的恢复和逆向软件工程的自动化。

（4）自动产生程序代码。

（5）自动进行各类检查和校验。

（6）项目管理和控制实现自动化。

（7）提高了软件复用性和可移植性。

（8）使软件工具自身不断演进，逐步具有 AI 功能。

12.3.4 CASE 与软件工程的关系

随着 CASE 技术的发展，其在软件开发过程中的地位也越来越重要，尽管“软件开发环境与开发技术”已经形成了独立的学科，但 CASE 从诞生起就有明确的辅助软件工程的性质，即与软件工程方法学是密不可分的，软件工程为软件的开发提供正确的方向以及理论和方法上的指导，CASE 只是软件工程理论、方法和技术的具体体现。在各种 CASE 目前的实用水平下，一个成功软件项目的关键仍在于选择正确的软件工程方法，比使用最好的工具更重要。一个集成的 CASE 工作台（或系统）所支持的方法学必须让开发人员掌握和领会，正确运用 CASE 所支持的方法学是正确使用 CASE 的前提。

从另一方面来看，CASE 的发展也已经改变着软件工程的发展。从目前 CASE 的迅速发展来看，CASE 超出了辅助软件工程的范围。近年来，CASE 使软件工程从技术、方法学到观念、认知体系都发生了深刻变化。主要表现在以下几个方面。

（1）简化、缩短了软件工程周期

CASE 从整体上减轻了系统分析人员、设计人员、测试人员以及管理人员的工作负担，把他们从繁琐的低级的劳动中解放出来，平均效率提高 50%以上。

（2）改变了系统分析方法，使得需求分析更为准确

需求分析正确与否是软件工程的出发点和前提。由于系统分析员和用户两方面的客观和主观因素的影响，因而需求分析往往不彻底；系统分析员一般很难准确地理解和描

述用户领域的需求，而用户由于不熟悉计算机系统的功能而不能够在短时间内提出和准确表达自己功能需求。实践表明，引入 CASE 支持下的快速原型开发方法能有效的解决需求分析不充分的问题，对保证软件质量、提高开发效率、降低开发成本有显著效果。

（3）简化了软件维护、丰富了软件维护方法

这里所说的维护是广义的，包括改正性、适应性、完善性和预防性维护。一方面，引入 CASE 后软件开发自动化程度增高，人工介入减少，加上工具智能化，使软件差错大大减少。

本 章 小 结

基于单个程序的软件工具，对于早期的程序设计活动带来了便利，有限的提高了程序设计的效率。将分散的个体化的工具集成起来而形成的软件开发环境则为软件开发人员提供了功能更强大、覆盖面更广的设计环境，软件开发环境不仅支持各阶段的开发，还为软件生命周期的全过程提供支持。

CASE 技术的出现，使得软件的开发进入了一个新的发展阶段，美国国防部在 STARES（software technology for adaptalle reliable system）计划中给 CASE 下的定义为“软件工程环境是一组方法、过程及计算机程序（计算机化的工具）的整体化构件，支持从需求定义、程序生成直到维护的整个软件生存期。”CASE 为软件开发过程提供了自动化程度更高的支持。CASE 不仅支持单个的过程活动，也支持一组相关的活动，更支持整个软件过程中的活动。

习　题　12

12.1　什么是 CASE？如何分类？

12.2　软件开发环境由哪几部分组成的？

12.3　在实际应用中，你使用过哪些软件工具或软件开发环境？举例说明其功能。

12.4　早期的软件工具包括哪些？有何特点？

参 考 文 献

1 张海藩. 软件工程导论（第三版）. 北京：清华大学出版社，1998
2 杨文龙，姚淑珍，吴云. 软件工程. 北京：电子工业出版社，1997
3 陈松乔，任胜兵，王国军. 现代软件工程. 北京：清华大学出版社，2004
4 Ian K. Bray. 需求工程导引. 舒忠梅等译. 北京：人民邮电出版社，2003
5 郭荷清等. 现代软件工程——原理、方法与管理. 广州：华南理工大学出版社，2004
6 Jim Arlow 等. UML 和统一过程：实用面向对象的分析和设计. 方贵宾等译. 北京：机械工业出版社，2003
7 Ian Sommerville. 软件工程. 程成等译. 北京：机械工业出版社，2003
8 Stephen R. Schach. 面向对象与传统软件工程. 韩松等译. 北京：机械工业出版社，2003
9 郑人杰. 软件工程（中级）. 北京：清华大学出版社，1999
10 万建成，卢雷. 软件体系结构的原理、组成与应用. 北京：科学出版社，2002
11 齐治昌，谭庆平，宁洪. 软件工程. 北京：高等教育出版社，1997
12 Roger S.Pressman. 软件工程——实践者的研究方法（第四版）. 黄柏素，梅宏译. 北京：机械工业出版社，1999
13 胥光辉，金风林，丁力. 软件工程方法与实践. 北京：机械工业出版社，2004
14 Penny Grubb，Armstrong A Tankang. 软件维护：概念与实践（第二版）. 韩柯，孟海军译. 北京：电子工业出版社，2004
15 史济民，顾春华，李昌武. 软件工程——原理、方法与应用（第二版）. 北京：高等教育出版社，2002
16 Rex Black. 软件测试过程管理（第二版）. 龚波，但静培，林生等译. 北京：机械工业出版社，2003
17 Brian Marick. 软件子系统测试. 韩柯等译. 北京：机械工业出版社，2003
18 James A.Whittaker. 实用软件测试指南. 马良荔，俞立军译. 北京：电子工业出版社，2003
19 Paul C. Jorgensen. 软件测试（第二版）. 韩柯，杜旭涛译. 北京：机械工业出版社，2003
20 Dirk Huberty 等. 软件质量与软件测试. 马博，赵云龙译. 北京：清华大学出版社，2003
21 Joseph Raynus. CMM 软件过程改进指南. 邱仲潘等译. 北京：电子工业出版社，2002
22 Sami Zahran. 软件过程改进. 陈新，罗劲枫等译. 北京：机械工业出版社，2002
23 郑人杰，王伟，王方德. 基于软件能力成熟度模型（CMM）的软件过程改进——方法与实施. 北京：清华大学出版社，2003
24 Kim Caputo. CMM 实施与软件过程改进. 于宏光，王家锋等译. 北京：清华大学出版社，2003
25 James R. Persse. CMM 实施指南. 王世锦，蔡愉祖译. 北京：机械工业出版社，2003
26 卡耐基-梅隆大学软件工程研究所. 能力成熟度模型（CMM）：软件过程改进指南. 刘孟仁等译. 北京：电子工业出版社，2001
27 郑人杰. 实用软件工程（第二版）. 北京：清华大学出版社，2001
28 王庆育. 软件工程. 北京：清华大学出版社，2004
29 刘润彬、张华. 软件工程简明教程. 大连：大连理工大学出版社，1995
30 Ian Sommerville. Software Engineering(影印版/第 7 版). 北京：机械出版社，2003
31 李刚. 软件工程分析与考试指导. 北京：高等教育出版社，2002
32 邓良松，刘海岩，陆丽娜. 软件工程. 西安：西安电子科技大学出版社，2000
33 周苏，王文. 软件工程教程. 北京：科学出版社，2002
34 钟珞. 现代软件工程学. 北京：国防工业出版社，2004
35 钟珞. 计算机软件基础. 武汉：武汉理工大学出版社，2002
36 钟珞. 系统分析员知识精要与试题分析. 北京：中国物质出版社，2005

读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便于我们更好地对本教材做进一步的改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A **座** 517（100084）**市场部收**

电话：62770175-3506

电子邮件：jsjjc@tup.tsinghua.edu.cn

教材名称：软件工程

ISBN：7-302-11849-3/TP · 7699

个人资料

姓名：_____ **年龄：**_____ **所在院校/专业：**_____

文化程度：_____**通信地址：**_____

联系电话：_____**电子信箱：**_____

您使用本书是作为：□**指定教材** □**选用教材** □**辅导教材** □**自学教材**

您对本书封面设计的满意度：

□**很满意** □**满意** □**一般** □**不满意** **改进建议**_____

您对本书印刷质量的满意度：

□**很满意** □**满意** □**一般** □**不满意** **改进建议**_____

您对本书的总体满意度：

从语言质量角度看 □**很满意** □**满意** □**一般** □**不满意**

从科技含量角度看 □**很满意** □**满意** □**一般** □**不满意**

本书最令您满意的是：

□**指导明确** □**内容充实** □**讲解详尽** □**实例丰富**

您认为本书在哪些地方应进行修改？（可附页）

_____

_____

您希望本书在哪些方面进行改进？（可附页）

_____

_____

电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本书有配套的电子教案，有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案，希望有助于教学活动的开展。相关信息请拨打电话 62770175-3506 **或发送电子邮件至** jsjjc@tup.tsinghua.edu.cn **咨询，也可以到清华大学出版社主页（**http://www.tup.com.cn**）上查询。**