



Mc
Graw
Hill

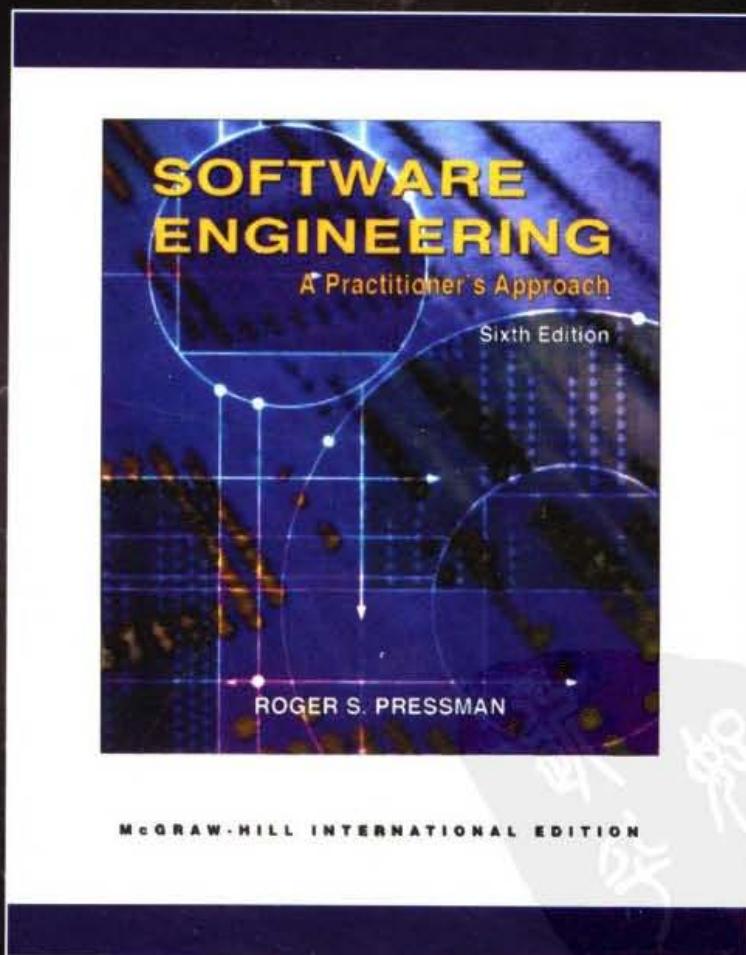
计 算 机 科 学 从 书

原书第6版

软件工程

实践者的研究方法

(美) Roger S. Pressman 著 郑人杰 马素霞 白晓颖 等译



Software Engineering
A Practitioner's Approach
Sixth Edition



机械工业出版社
China Machine Press

软件工程 实践者的研究方法 (原书第6版)

Roger Pressman 编写的这部翔实而全面的软件工程指南，广泛适合软件工程专业的学生及投身软件工程实践或需要参与这种实践的软件开发人员和管理人员。

——《IEEE Software》

这是一本经典的现代教材，叙述清晰而又有权威性。本书包含大量插图、例子、习题和参考资料……如果读者心存疑问：“软件工程是什么？它现在在哪里？”那么最好阅读这本书。

——《ACM Computing Reviews》

作为一名软件工程实践者，我发现此书是无价的。对于我做过的所有项目，本书都有重大的参考价值。

——摘自Amazon.com的评论

20多年以来，《软件工程：实践者的研究方法》一书是最受学生和行业专业人员欢迎的软件工程指南。它在全面而系统、概括而清晰地介绍软件工程的有关概念、原则、方法和工具方面获得了广大读者的好评，在国际软件工程界享有无可质疑的权威地位。

本书第6版在结构和内容上均有不少调整、更新和充实。第6版更加突出了软件过程，增加了敏捷开发方法，论述了很多人们称之为“21世纪工程学科”的重要主题。

对第6版的内容做了如下划分，更便于课堂教学及自学使用：

- 第一部分 软件过程，介绍了惯例模型和敏捷过程模型。
- 第二部分 软件工程实践，介绍了现代分析、设计和测试方法，新的重点放在基于UML的建模方面。
- 第三部分 应用Web工程，是第6版中新增的内容，描述了如何使软件工程实践适应WebApp工程。
- 第四部分 管理软件项目，介绍与计划、管理和控制软件项目有关的主题。
- 第五部分 软件工程高级课题，专门讲述了形式化方法、净室软件工程、基于构件的方法及再工程。

作者简介

Roger S. Pressman

博士是软件过程改善和软件工程技术方面的国际知名的权威。30多年来，他作为软件工程师、管理人员、教授、作者及咨询顾问始终工作在软件工程领域。

Pressman博士著有6部著作，撰写了很多技术文章，是多种行业期刊的固定撰稿人，曾任多种行业杂志的编委，并多年来一直担任《IEEE Software》杂志Manager专栏的编辑。

Pressman博士是知名的演讲者，曾在许多行业会议上演讲，他还是美国计算机协会(ACM)、美国电气与电子工程师协会(IEEE)等组织的成员。



ISBN 7-111-19400-4

封面设计：陈子平



华章图书

上架指导：计算机/软件工程

华章网站 <http://www.hzbook.com>

网上购书：www.china-pub.com

投稿热线：(010) 88379604

购书热线：(010) 68995259, 68995264

读者信箱：hzjsj@hzbook.com

ISBN 7-111-19400-4

定价：69.00 元

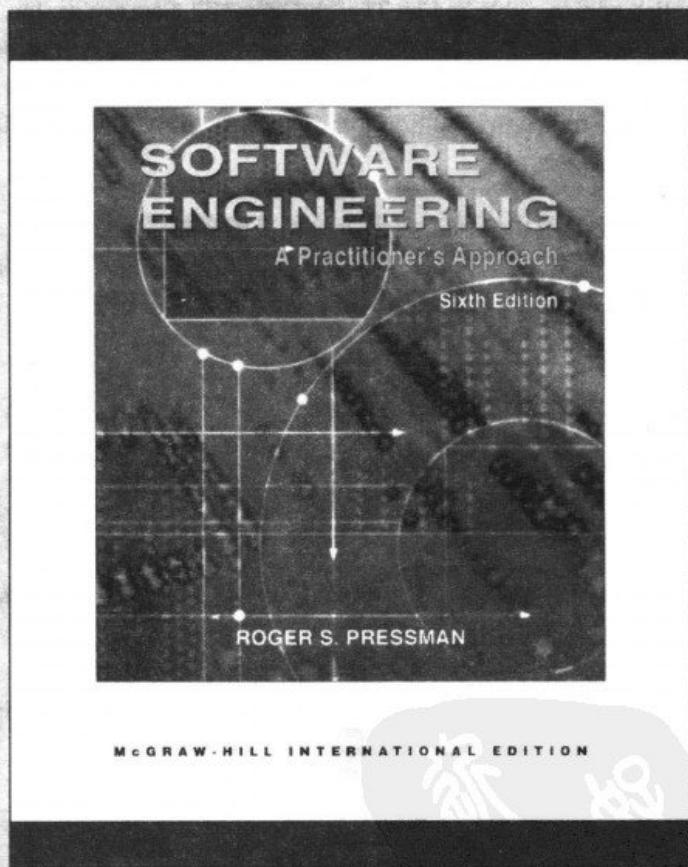


软件工程

实践者的研究方法



(美) Roger S. Pressman 著 郑人杰 马素霞 白晓颖 等译



Software Engineering
A Practitioner's Approach
Sixth Edition



机械工业出版社
China Machine Press

本书自1982年发行第1版以来，一直受到软件工程界的高度重视，成为高等院校计算机相关专业软件工程课的重要教学参考书。20多年来，它的各个后继版本一直都是软件专业人士熟悉的读物，在国际软件工程界享有无可质疑的权威地位。它在全面而系统、概括而清晰地介绍软件工程的有关概念、原则、方法和工具方面获得了广大读者的好评。此外，本书在给出传统的、对学科发展具有深刻影响的方法时，又适当地介绍了当前正在发展的、具有生命力的新技术。

本书第6版在结构和内容上均有不少调整、更新和充实。第6版更加突出了软件过程，增加了敏捷开发方法，更便于阅读。全书包括软件过程、软件工程实践、应用Web工程、管理软件项目及软件工程高级课题五个部分。

本书可作为计算机相关专业本科生和研究生的教材，同时也是软件工程领域专业人员的优秀参考读物。

Roger S. Pressman: *Software Engineering: A Practitioner's Approach, Sixth Edition* (ISBN 0-07-123840-9).

Copyright © 2005, 2001, 1997, 1992, 1987, 1982 by The McGraw-Hill Companies, Inc.

Original English edition published by The McGraw-Hill Companies, Inc. All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

Simplified Chinese translation edition jointly published by McGraw-Hill Education (Asia) Co. and China Machine Press.

本书中文简体字翻译版由机械工业出版社和美国麦格劳-希尔教育（亚洲）出版公司合作出版。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有McGraw-Hill公司防伪标签，无标签者不得销售。

版权所有，侵权必究。

本书法律顾问 北京市展达律师事务所

本书版权登记号：图字：01-2004-4837

图书在版编目（CIP）数据

软件工程：实践者的研究方法（原书第6版）／（美）普雷斯基曼（Pressman, R. S.）著；
郑人杰等译。—北京：机械工业出版社，2007.1

（计算机科学丛书）

书名原文：Software Engineering: A Practitioner's Approach, Sixth Edition
ISBN 7-111-19400-4

I. 软… II. ① 普… ② 郑… III. 软件工程 IV. TP311.5

中国版本图书馆CIP数据核字（2006）第082718号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：刘立卿

北京京北制版印刷厂印刷 新华书店北京发行所发行

2007年1月第1版第1次印刷

184mm×260mm · 45.75印张

定价：69.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall, Addison-Wesley, McGraw-Hill, Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum, Stroustrup, Kernighan, Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T., Stanford, U.C. Berkeley, C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037



专家指导委员会

(按姓氏笔画顺序)

尤晋元	王 珊	冯博琴	史忠植	史美林
石教英	吕 建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范 明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅 宏	程 旭	程时端	谢希仁
裘宗燕	戴 葵			



译者序

作为一部软件工程著作，本书全面、系统地阐述了当前软件工程技术的各个层面。它概括而清晰地介绍了有关的概念、原则、技术、方法和工具。自从20多年前第1版问世以来，它在国际软件工程界发挥了巨大而深远的影响，从而树立了无可质疑的权威形象。

从第1版开始，我一直是本书的忠实读者，它是我在教学工作中的一本重要参考教材，也是我的良师益友。

软件工程是一门工程学科，它告诉我们如何规范化地开发软件。事实一再表明，不规范的软件是不可用、不易修改的软件。本人从20世纪80年代中期开始承担软件工程课的教学任务，深知这门专业课的重要性及教学的难点。如果询问从大学计算机专业毕业多年的从事实际工作的专业人员，在大学哪门课最为实用，许多人会毫不犹豫地回答：是软件工程课。其实，软件工程课不只在计算机行业有用，非计算机专业的学生也会对这门课很有兴趣。十几年前，我所工作的清华大学要求我面向非计算机专业的理工科研究生开设软件工程选修课，结果报名十分踊跃，竟出现700人同堂听课的场面。据学校研究生处称，如果不具备软件工程的概念和知识，这些来自各专业的研究生在各自的研究项目中所开发的软件，因为极不规范而难以理解。对导师来说，整个研究课题前后工作无法很好地衔接，也很难交流。

本书从1982年第1版开始就为我国软件界密切关注，国内至少已出版了第2版、第5版的中译本，对国内大学的软件工程课程产生了巨大的影响。

第6版的特点

1. 在第5版的基础上做了大量的充实和更新，例如，突出了软件过程，增加了敏捷开发方法。
2. 除各章后面提供了大量供进一步阅读的参考文献信息外，针对不同的读者（学生、教师和专业人员）提供了各种形式的材料，范围广泛，内容丰富，且使用方便。
3. 为了方便阅读和理解，除在各章开头给出全章内容简介和关键词外，在文中穿插了许多内容不同的解释框。而且全书贯穿了一个应用实例（住宅安全系统——SafeHome），以对话形式逐步展开这一软件的开发过程，从而提高了严肃的技术书籍的可读性，极大地引发了读者的阅读兴趣。

读者对象

本书前言中已明确指出了三类主要读者，即高校学生、教师和专业技术人员，总体上该书是为教学服务的。教师如果以本书为教材，面对的又是软件工程的初学者，可能需自行补充一些具体的材料，让学生在本书内容的基础上掌握一些具体的、实用的、操作性更强的工作步骤。当然这可结合习题或实习，可以在课内讲解，也可作为指定的补充读物，由学生自学掌握。对于教师自己，或具有一定软件工程基础的研究生和软件专业人员，本书给你的是

全面的、丰富的、代表当前发展水平的软件工程知识。

参加本书翻译工作的主要有清华大学计算机系和华北电力大学计算机系的教师，包括马素霞（第18~20章、第28~32章），白晓颖（第1~3章），董渊（第4~6章），赵冲冲（第7~9章），张志强（第10~12章），戴桂兰（第13~15章），王素琴（第21~23章），谢萍（第24~27章），贾克（第16~17章）等。在翻译过程中，得到清华大学计算机系栗宏、谢继辉，软件学院马秋霞、程明、李海庆、邬双，华北电力大学计算机系周莉梅、杜江的帮助，内蒙古大学的郇失宇老师仔细审阅了第4~6章，在此对他（她）们的辛勤劳动表示感谢。本人将大部分译稿、马素霞教授将后半部分译稿做了仔细审核与修改。尽管翻译工作历时一年，在大家非全时投入又不能集中工作的情况下仍感到时间紧张，某些部分仍不够理想。限于水平，对内容的理解和中文表达难免有不当之处，在此敬请读者批评指正。又，原书中存在的个别问题（包括错、漏及不妥之处）均在译者注中指出。

总之，这是一本非常优秀的软件工程读物，本人也十分高兴地向读者推荐，认真阅读它将会使你受益匪浅。

郑人杰

2006年元旦

译者简介

郑人杰 清华大学教授，清华同方股份有限公司顾问。1961年毕业于清华大学，后留校任教。多年来围绕软件工程领域从事教学、科研工作，近年注重于软件质量及软件过程改进。曾编写《实用软件工程》、《计算机软件测试技术》、《基于软件能力成熟度模型（CMM）的软件过程改进》等。

马素霞 华北电力大学计算机科学与技术系教授。1989年于清华大学计算机系获工学硕士学位；1986年于南开大学计算机系获学士学位。毕业后一直从事计算机软件方面的教学及研究工作。2000年通过国家留学基金委的选拔，2001年作为国家公派访问学者在加拿大卡尔顿（Carleton）大学计算机科学系进修一年，2002年4月回国。主要研究方向为软件工程、分布式地理信息系统。

白晓颖 博士，现任清华大学计算机科学与技术系副教授。2001年于美国亚利桑那州立大学计算机科学与工程系获博士学位，1998年于北京航空航天大学计算机系获硕士学位，1995年于西北工业大学计算机系获学士学位。主要研究方向为软件工程。

前　　言

成功的计算机软件能够很好地满足使用者的要求，能在相当长的时间内无故障地运行，能够非常好用，也容易修改；这样的软件能够也确实会把事情办好。反之，有问题的软件无法让用户满意，容易出错，难于修改，甚至很难使用；这样的软件将会也确实会把事情办糟。我们希望开发出好的软件，把事情办好，避免那些隐藏在背后的糟糕的事情发生。为了获得成功，在设计和开发软件时，我们需要有规范，需要有工程化的方法。

在本书第1版问世以来的25年中，软件工程已经从少数倡导者提出的一些朦胧概念发展成为一门正规的工程学科。如今，软件工程已被公认为是一个值得深入研究、认真学习和热烈讨论的课题。在整个行业中软件工程师已经代替了程序员成为人们优先选择的工作岗位。软件过程模型、软件工程方法和软件工具已在广阔的行业应用领域得到成功的采用。

尽管管理人员和工作在第一线的专业人员都承认，需要有更为规范化的软件方法，但他们却始终在争论着应该采取什么样的规范。许多个人和公司至今仍在杂乱无章地开发着自己的软件，甚至即使他们正在开发的系统服务于目前最为先进的技术，也仍然如此。许多专业人员和学生并不了解现代方法。于是造成了严重的后果，以至所开发的软件质量很差，糟糕的事情时有发生。此外，有关软件工程方法性质的争论一直持续进行着。软件工程的重要地位问题已成为研究课题。人们对待软件工程的态度已有所改变，研究工作已取得了进展，不过要成为一个完全成熟的学科还有大量的工作要做。

本书第6版希望成为推动工程学科走向成熟入门读物。和前五版一样，第6版对学生和专业人员同样具有很强的吸引力。本书既是软件行业专业人员的工作指南，同时也是大学高年级学生以及一年级研究生的综合性参考书。

第6版包含很多新的内容，它绝不是前一版的简单更新。这一版做了不少修订，结构上也有调整，更加强调一些新的和重要的软件工程过程和实践。此外，我们专门为本书开辟了一个网站（www.mhhe.com/pressman），其中提供的“支持系统”（见图0-1）包含了为学生、教师和专业人员提供的大量专业资源，从而丰富和充实了本书的内容。（访问原书网站可能无法获取有些配套资源，需要这些配套资源的教师可联系麦格劳-希尔教育出版公司北京办事处，联系方式参见书后的“教学服务沟通表”。——编者注）

第6版共32章，分为五个部分。这样就把全书分为五个主题，从而有助于教师解决无法在一个学期内讲完书中全部材料的问题。

- 第一部分 软件过程，给出了软件过程的不同论点，考虑到所有重要的过程模型，涉及传统过程和敏捷过程在指导思想上的争论。
- 第二部分 软件工程实践，给出了分析、设计和测试方法，其中突出讨论了面向对象技术和UML建模。鉴于面向对象方法目前在行业中已被广泛采用，原来第5版中第四部分的内容（面向对象软件工程）现已全部纳入到这里了。
- 第三部分 应用Web工程，提供了Web应用系统的分析、设计和测试的全面工程方法。
- 第四部分 管理软件项目，给出的是与计划、管理和控制软件项目的人员有关的问题。

- 第五部分 软件工程高级课题，涉及形式化方法、净室软件工程、基于构件的软件工程、再工程以及未来的发展趋势等问题。

第6版除了对前一版本做了许多更新和重要修改外，还特别增加了120多个框。主要分为以下4种：

- 让读者跟随一个虚构的项目组，跟随他们的工作进程开发一个计算机应用系统。
- 对选择的题目提供补充的讨论。
- 概述反映某些软件工程活动工作流的任务集。
- 推荐与特定章节相关的自动化工具。

第6版分为五部分，这样方便教师根据时间和学习要求安排讲课内容。在一个学期内可进行一个部分，也可进行多个部分。例如“方法课”可能只强调第一和第二部分；而Web开发课会强调第一和第三部分；管理课应把重点放在第一和第四部分。第6版内容这样组织，其意图在于给教师提供多种教学安排的选择。但无论如何选择，第6版的内容都可获得“支持系统”的补充支持（参见图0-1）：

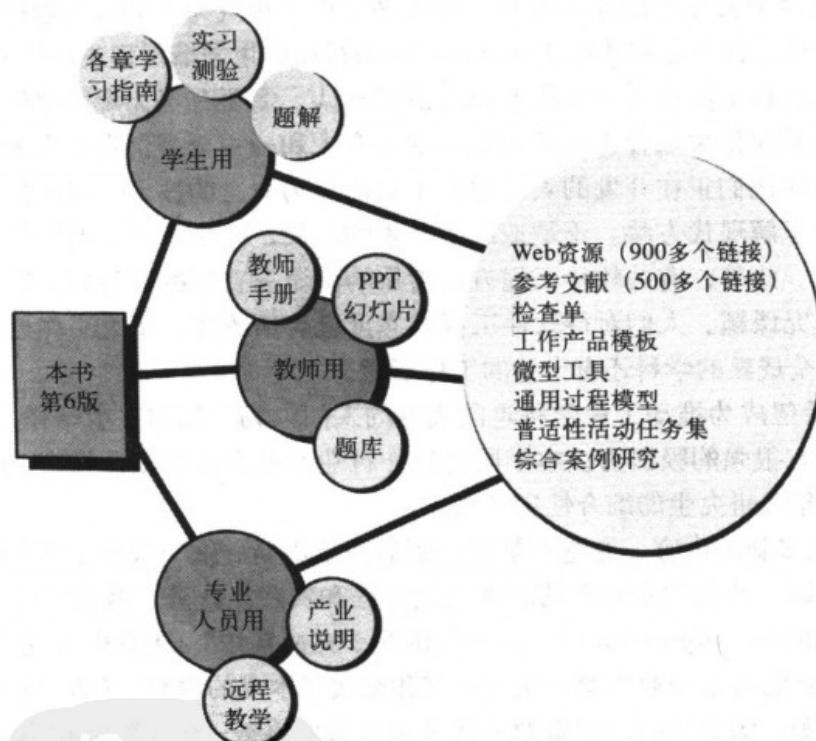


图0-1 本书第6版的支持系统

为学生提供的资源

提供种类繁多、内容丰富的材料供学生学习。包括：在线学习中心可提供学习指南、实习测验以及Web资源（包括软件工程检查单、一套不断演化的微型工具、完整的案例研究以及工作产品模板）。此外，还有900多种网上参考资料可供学生更深入地探讨软件工程问题。

为教师提供的资源

本书第6版为教师提供了广泛的资源，包括综合性在线教师指南（可下载）、教学补充材料、700多幅供讲课用的PowerPoint幻灯片、试题库及试卷模板。此外，参考文献指明了500多篇软件工程的研究论文（分专题组织，且可下载），在讲授高级软件工程课程时可作为专题

讨论课使用。在教师指南中，对各种类型的软件工程课程的教学提出了建议，介绍了与课程配合开展的软件项目以及许多有用的教学辅助工具。

为专业人员提供的资源

有许多资源可供工业界的专业人员（也包括学校师生）使用，包括软件工程文档及其他工作产品的纲要和模板、一套有用的软件工程检查单、CASE工具目录、综合性Web资源，并且还提供了根据具体任务划分的软件工程过程的“通用过程模型”。

本书第6版配有在线支持系统，这样既保证了使用上的灵活性，又保证了内容上的足够深度。这一点是任何单本教科书无法比拟的。

致谢

本书六个版本的出版工作是我一生中持续最久的技术项目。甚至在书稿完成后，我仍然不断地从一些技术文献中提取信息并加以吸收和组织。正是这个原因，我十分感谢这些书籍、文章（包括发表在硬拷贝和电子媒体上）的作者，在过去的25年中是他们给了我不少见解和想法。

特别感谢渥太华（Ottawa）大学的Tim Lethbridge，他非常仔细地审核了第6版，并帮助我开发了UML及OCL实例，还开发了与本书配套的综合案例研究，他的支持和建议非常有价值。特别感谢密歇根大学迪尔本分校（Michigan-Dearborn）的Bruce Maxim，他帮助我开发了与本书配套的Web站点，Bruce负责大部分教学内容。最后，我想感谢第6版的所有评审人员，他们的深入见解和批评非常宝贵：

Mark Ardis <i>Rose-Hulman Institute</i>	Sergiu Dascalu <i>University of Nevada, Reno</i>
Xiaoxia Cao <i>Shanghai University</i>	Harry Delugach <i>University of Alabama, Huntsville</i>
Nimmagadda Chalamaiah <i>Jawaharlal Nehru Technological University</i>	Premkumar Devanbu <i>University of California, Davis</i>
Lipika Dey <i>I.I.T., Delhi</i>	Ahmed Naumaan <i>University of Minnesota</i>
Osama Eljabiri <i>New Jersey Institute of Technology</i>	Joey Paquet <i>Concordia University</i>
Gerald Gannon <i>Arizona State University</i>	James Purtill <i>University of Maryland</i>
David Gustafson <i>Kansas State University</i>	Tong Seng, Jon Quah <i>Nanyang Technological University</i>
Qingchun Hu <i>East China University of Science and Technology</i>	K.V.S.V.N. Raju <i>Andhra University</i>
Shi-Ming Huang <i>National Chung Cheng University</i>	D Janaki Ram <i>Indian Institute of Technology, Madras</i>
Clinton Jeffery <i>New Mexico State University</i>	Ahmed Salem <i>California State University, Salem</i>
Barbara Jennings <i>Colorado School of Mines</i>	Hee Beng Kuan Tan <i>Nanyang Technological University</i>
Venkatesh Kamat <i>Goa University</i>	Chris Teng <i>San Jose State University</i>
Jo Ann Lane <i>San Diego State University</i>	Flora Tsai <i>Nanyang Technological University</i>
Minglu Li <i>Shanghai Jiao Tong University</i>	David Umphress <i>Auburn University</i>
Robert Lingard <i>California State University, Northridge</i>	Liang Wang <i>Renmin University of China</i>
Jiang B. Liu <i>Bradley University</i>	Laura Williams <i>North Carolina State University</i>
	Junmin Ye

WY Liu
City University of Hongkong
Banshidhar Majhi
National Institute of Technology
John D. McGregor
Clemson University
Hong Mei
Peking University

Central China Normal University
Renkun Ying
Tsinghua University

使用本书早期版本的工业界的的专业人员、大学教授及学生塑造了本书第6版的内容，他们花费了很多时间提出建议、批评和想法，在此向他们表示感谢。另外，还要感谢许多工业界的客户，他们教给我的比我能够教给他们的更多。

随着本书版本的更新，我的儿子Mathew和Michael已经长大成人。他们在现实生活中的成熟、品质和成功给了我灵感，没有什么比这更让我感到自豪的了。最后，感谢我的妻子Barbara鼓励我继续出版本书的下一个版本。

Roger S. Pressman



作者简介

Roger S. Pressman是软件过程改进和软件工程技术领域知名的权威。30多年来，他作为软件工程师、管理人员、教授、作者及咨询顾问始终投身于软件工程领域。

Pressman博士曾经以软件产业专业技术人员和管理人员的身份从事先进工程、制造领域的CAD/CAM系统的开发。他也做过科学和系统程序设计方面的工作。

在获得美国康涅狄格大学工程学博士学位后，Pressman博士进入学术界成为布里奇波特(Bridgeport)大学计算机工程系的Bullard副教授，同时担任该校CAD/CAM中心主任。

现在，Pressman博士是R. S. Pressman & Associates, Inc.的总裁，该公司专门从事软件工程方法的咨询和培训业务。作为公司的主要咨询专家，他设计和开发了一套完整的软件工程录像课程“Essential Software Engineering”以及软件过程改进的指导系统“Process Advisor”。这两项产品已为世界上数千家公司采用。最近，他与印度QAI公司合作，开发网上软件工程教学系统“eSchool”。

Pressman博士撰写了许多论文，是多种行业期刊的固定撰稿人，并著有6本技术书。除本书之外，还有：

- 《A Manager's Guide to Software Engineering》(McGraw-Hill)

该书曾获奖。

- 《Making Software Engineering Happen》(Prentice-Hall)

这是涉及软件过程改进的关键管理问题的第一本书。

- 《Software Shock》(Dorset House)

该书论述软件及其对商业和社会的影响。

Pressman博士曾任多种行业杂志的编委，并多年来一直担任《IEEE Software》杂志Manager专栏的编辑。

Pressman博士是知名的演讲者，曾在许多行业会议上作重要讲话，他还是美国计算机协会(ACM)、美国电气与电子工程师协会(IEEE)等组织的成员。

Pressman博士和他的妻子Barbara住在南佛罗里达。他热爱体育运动，擅长网球和高尔夫球。还曾写过两部小说：《The Aymara Bridge》和《The Puppeteer》。

本书使用说明

各章首页

每章开头都包括：

1. 要点浏览。放在封闭的大方框内，包括：

- 概念——说明本章讨论内容的主题，包含哪些方面的工作或工作客体。
- 人员——表明本章讨论的工作由谁来做，介绍工作的主体。
- 重要性——阐明本章工作的重要性和价值所在。
- 步骤——介绍完成本章讨论的工作包含哪些环节，以及其间的关系。
- 工作产品——给出完成本章讨论的工作应得到什么成果。
- 质量保证措施——解释如何说明本章所讨论工作是正确的，用什么方法、采取什么措施能保证它的质量。

2. 关键概念。置于每章开头的左边框内，分层次列出本章涉及的重要术语。

例如，第2章的首页如下所示：

第2章 过程综述

要点浏览

概念：当开发产品或构建系统时，遵循一系列可预测的步骤（即路线图）是非常重要的。它有助于及时交付高质量的产品。软件开发中所遵循的路线图就称为“软件过程”。

人员：软件工程师及其管理人员根据需要调整开发过程，并遵循该过程。除此之外，软件的需求方也需要参与过程的定义、建立和测试。

重要性：软件过程提高了软件工程活动的稳定性、可控性和有组织性，如果没有过程约束，软件活动将失控并变得混乱。但是，现代软件工程方法必须是“灵活”的。也就是要求软件工程活动、控制以及文档的编制适合于项目团队和要开发的产品。

步骤：具体来讲，采用的过程依赖于所构造软件的特点。飞机电子系统的软件与网站的建设可能是两个截然不同的构建过程。

工作产品：从软件工程师的观点来看，工作产品就是过程定义的一系列活动和任务的结果，即程序、文档和数据。

质量保证措施：有大量的软件过程评估机制，开发机构可以评估其软件过程的“成熟度”。然而，评价所采用过程的有效性，最好的指标还是所构建产品的质量、适时性和长期生存能力。

关键概念
CMMI
ISO 9001: 2000
过程评估
过程框架
过程模式
过程技术
PSP
任务集
TSP
普遍性活动

Howard Baetjer, Jr.[BAE98]曾著书从经济学家的角度分析软件和软件工程，书中对软件过程评述如下：

软件同其他资产一样，是知识的具体体现，而知识最初都是以分散的、不明确的、隐藏的且不完整的形式广泛存在的，因此，软件开发是一个社会学习的过程。软件过程是一个对话，在对话中，软件所必需的知识被收集在一起并在软件中实现。过程提供了用户与设计人员之间、用户与不断演化的工具之间以及设计人员与不断演化的工具（技术）之间的交互途径。软件开发是一个迭代的过程，在这个过程中，演化的工具本身就作为沟通的媒介，每新一轮对话都可以从参与的人员中获得更有用的知识。

带图标的简要注释

以下4类注释均位于正文的左侧：

1.  指出该段内容的要点。
2.  指出网上资源的Web站点。
3.  指出实际工作应注意的地方，并给予指导或建议。
4.  此处所提问题的答案即在右侧正文中。

文中插入的说明

这类说明一般排为夹在文中自然段之间的通栏长方框，包括以下4种，其中前3种有明显标识：

1. **SAFEHOME** 该框以对话形式给出软件工程人员在完成相关工作、将软件工程原则和方法应用到实际问题时的考虑。书中是以Safe Home系统（住宅安全系统）为例进行有关讨论。
 2. **INFO** 栏内给出所述内容的进一步补充和扩展信息，从而使读者对相关概念有更广更深的理解。
 3. **SOFTWARE TOOLS** 给出当前最为流行的、有代表性的商品化软件工具，指明工具的开发商及其相关信息。
 4. **TASK SET** 该框内简要地列举出要完成的任务。
5. 文中插入的引语：也被置于自然段之间的通栏长方框内，无任何标识。框内的内容引自有影响的文献，其简明精炼的语句给读者以有益的启发。引语结束后指明所引文献的作者。这种引语很简练，通常只有两行以内的文字。



{ — ° ç Ô s < œ

i { 2 ! > h Ú t X + h t ÿì U ï Y ' q £ \ ÿÔ; h t ô! I F '
 r £ , & i · ÿp " M ä h r u , ÿì p Q ' l p » ÿq Ó{ h 2 % 8 1 . X
 +
 § » öl Ó« . i ï ÿ& ö÷i O u ÷ Ñr : h Ø Ò[v h ßO - . [r : h { ò V Æ
 I 2 i p ßO - ÿ. [! . 3 + / Ø Ò[v K [v = I < { { 2 2 C ÿ{ I Ó! c ' ÿ Õp
 i k , Ät ! I » ÿq Ó{ h i F O L @ K J A P I B ? R > ÿ" \ i Ä i t ð Ö N à i
 p » W i F O L @ K J A P Ä i f f l ÿ_ w i y t »
 r , ÿ AE H { 2 § \ t r ! t b < § h 1 ü + \$! t b § { Ö h
 ÿ w i ' i = O L J A P * i " y Ü! J A P F A A L D L § h Ú h 4 ¶ N _ ð i h r
 ÿ e v y Y u . i ÿ O M H O A N R A N ? O O = F = T v ß [> h H t ß i W Ö ø
 H t ß W " a l , y " » + r W r § " æ ? i ' æ q ? Ú æ
 I ? i ! ') h § J h I ') p s ¥ ÿ x £ ð Ö Ö b r k k ð à ! e
 AE Ú h « J A P i p h Æ F A A ÿ s 7 h Æ = L E { 2 h Ø Ò[v % ð [y 4 i
 ÿ c i r £ , & i · ÿ M ä l » ÿq Ó{ ! . Ä
 " J A P F = R = ÿ N _ ? t Ø ð ! | { Ø Ò[v l " t Õ p l » ! f , » ÿ h Ö h
 ý " t ý ! " h ' I B ? J A P p < Ù ¶ Ö ... § k æ p h f ÿ Ú h . » k Ü t h ! ' r à k à t ! l » p t b ÿ V Æ t b ÿ V ' i J A P ÿ AE 9 Ä Ä Ø e Ù ¶ ÷ t ÷ ð ¥ ü r ð t b § l t à ð Ö' . N _ Ó t b ÿ V Æ Ö k Ü t h . ý B E ÷ ð ÿ h t " ¥ N _ i < h . Ö' Ü ÿ V Æ y W ð Ä f ÿ Ú
 Ö{ ¾ ÿ - w J Ö B ÿ » h ß > ÷ Ñr : O i ÿ t p h x 2 C ÿ{ J ¾ 8 x J
 < S D W ` u U ÿ ÷ Ñr : B > t Ü Ø Ò[v h x J Ü l N _ ' p ÿ V Æ i · k » Ø Ò[v
 ÿ t Ü { h { ò V Æ h x b 2 C E Ö Ö Ø h t { ° ð ÿ -
 B > ! i p » Ö N à W i J A P h F = R = I # b y Ä Ú h # s + h i . [ÿ' , - , h w t °
 ' p h q h I R ? K N I I A : h Ö h Ä ð t ý § ð h Ü l p Ä h Ü i p Ö «
 { h 4 ¶ ' 2 h Æ B ^ R I » i p W l k ð Ö t b E p b
 t Ö p x æ p ' , P , # 2 6 + * 1 1 ' , / " B ¥ h k B > , ÿ » t B h i ' p f p ^ Ö J h h ! & h t ! K ÿ " ÿ l p , ÿ ' , Ü £ , F ÿ V Æ h Ö f
 ÿ i ' , H s C h B B ÿ » i i p h ! l » h Ö Ö p e p ' , f
 N à { 2 E ÷ ð i ! ! [v = L E { 2 h i · Ø F F h i f l . AE ¥ Z V Æ i p ? !
 { 2 » h W i I B ? J A P I Ä s 7 5 7 h w f . B ä Æ i 4 { 2 § Ö h i » 2
 4 ! ÿ 2 C Æ B § Ö > e B > Ö ð i 4 h 4 i y h t Ä h H > h ^ C 4 h ß > ð i
 y h 4 i 4 h £ Ö h § Ö ' i h " 9 , k

{ 2 þs k - h ß>ì J A P I B ? R > I þ . ý ø þ ' þ - %o R h » ý i - h q
‰o R - t ß t k ° ý f , ¥ Àà fi ' - Äe b § k ! ôô h - b
ô ÿ , < 4 - h / ß k - l h ô ¾1 o y N âk Ö= ÷ Ö âý I ¢ U ý
- h } À , ' œ - Y . r k H \ ' Öý p ° . - Ñ] H ' y h n k ^ k < ý i
- h æ g k Ö - Äe b ' Ös ô ý » ý h q Öý Öt { 2 þs ° [h L » þ !
Ö ÀH Öý h w h # ° h ^ Ö # 8 q I F i + t ! ð t ° ¢ ý t i h
· m ‡ Öþý »

I ° ¥ ý ; Öh üyt f ¥ Z h , k ‡ I & i · ý p " h Ý óy , Öl À! , À
ÿ ° ý Ú , h « I & i · ý M ä y ø B l ÖE L @ B , ' s < s 6 2 % , ! 8 1 L @ B
" > D P P L > > O P D A E P D K I A ? K I N A = @ D P I P E @ D P I H
ü I , " Ø , Ö þ ß ý » h t Öh · R { ý ' , h þ » W i e ' ,
X . " ^ h / ý _ À þ ý
» I q Öx J i þ i öl Úh i þ i · E ÷ ÉI » 2 [ý y < , Ö p ý § i o
Ø { 2 ç - . ö þ " o ' + Ø ‡ q + ý - - p I I p ý § t x 8 1 . þ » h w
· ý 2 C ¥ i i ‡ . ý _ h Ý à 6 ól § y h . ý 2 C + À = t i " I
§ ? t þ ! i \ h ¥ ý . i ð l \ ô O ý t , I Y å { Ö Ø , ! 4 ô À
! ^ ! M Ö Ø ‡ h , œ w Ä Ø i I § h À i ý . B Ö / ! % I J
‡ y ý i s ß ý »
+ Ø ‡ x § j > > O P D A E P D K I A ? K I
ß O - i » t - ß ý " \ " ý { h Ö q -
i Ú t p ý § h " E § h x J k -
i Ý x Ö ý " ý h i þ h t À e ^
q + B Ä Ø h v h À i ý . B Ö /
Ä Ø ý k þ ý § h q Ö + k À â . a i q + Ä Ø f ! ý

目 录

出版者的话	2.9 小结	33
专家指导委员会	参考文献	33
译者序	习题与思考题	34
译者简介	推荐读物与阅读信息	34
前言	第3章 过程模型	36
作者简介	3.1 惯例过程模型	37
本书使用说明	3.2 瀑布模型	37
第1章 软件工程介绍	3.3 增量过程模型	38
1.1 软件角色的演化	3.3.1 增量模型	38
1.2 软件	3.3.2 RAD模型	39
1.3 软件特性的变化	3.4 演化过程模型	40
1.4 遗留软件	3.4.1 原型开发	41
1.4.1 遗留软件的质量	3.4.2 螺旋模型	43
1.4.2 软件演化	3.4.3 协同开发模型	44
1.5 软件神话	3.4.4 演化过程模型的最终评述	45
1.6 这一切都是如何开始的	3.5 专用过程模型	46
1.7 小结	3.5.1 基于构件的开发	46
参考文献	3.5.2 形式化方法模型	47
习题与思考题	3.5.3 面向方面的软件开发	47
推荐读物与阅读信息	3.6 统一过程	49
第一部分 软件过程	3.6.1 简史	49
第2章 过程综述	3.6.2 统一过程的阶段	50
2.1 软件工程：一种层次化技术	3.6.3 统一过程工作产品	51
2.2 过程框架	3.7 小结	52
2.3 能力成熟度模型集成	参考文献	53
2.4 过程模式	习题与思考题	54
2.5 过程评估	推荐读物与阅读信息	54
2.6 个人过程模型和团队过程模型	第4章 敏捷视角下的过程	56
2.6.1 个人软件过程	4.1 敏捷是什么	57
2.6.2 团队软件过程	4.2 敏捷过程是什么	58
2.7 过程技术	4.2.1 敏捷开发的立场	59
2.8 产品与过程	4.2.2 人的因素	59
	4.3 敏捷过程模型	60
	4.3.1 极限编程	61

28.1.2 软件开发中的数学	608	推荐读物与阅读信息	639
28.1.3 形式化方法概念	608	第30章 基于构件的开发	640
28.2 数学预备知识	610	30.1 基于构件系统的工程	641
28.2.1 集合与构造性规格说明	610	30.2 CBSE过程	642
28.2.2 集合运算符	611	30.3 领域工程	643
28.2.3 逻辑运算符	613	30.3.1 领域分析过程	643
28.2.4 序列	613	30.3.2 特征化函数	643
28.3 应用数学表示法描述形式化规格 说明	614	30.3.3 结构化建模与结构点	644
28.4 形式化规格说明语言	615	30.4 基于构件的开发	645
28.5 对象约束语言	616	30.4.1 构件合格性检验、适应性 修改与组装	645
28.5.1 OCL语法及语义概述	616	30.4.2 构件工程	648
28.5.2 使用OCL举例	617	30.4.3 复用的分析与设计	648
28.6 Z规格说明语言	618	30.5 构件分类与检索	649
28.6.1 Z语法及语义概述	619	30.5.1 描述可复用构件	649
28.6.2 使用Z举例	620	30.5.2 复用环境	650
28.7 形式化方法的十条戒律	621	30.6 CBSE经济学	651
28.8 形式化方法——未来之路	622	30.6.1 对质量、生产率及成本 的影响	651
28.9 小结	622	30.6.2 运用结构点的成本分析	652
参考文献	623	30.7 小结	653
习题与思考题	623	参考文献	653
推荐读物与阅读信息	624	习题与思考题	654
第29章 净室软件工程	626	推荐读物与阅读信息	655
29.1 净室方法	627	第31章 再工程	657
29.1.1 净室策略	627	31.1 业务过程再工程	658
29.1.2 净室方法的特异之处	629	31.1.1 业务过程	658
29.2 功能规格说明	629	31.1.2 BPR模型	659
29.2.1 黑盒规格说明	630	31.2 软件再工程	660
29.2.2 状态盒规格说明	631	31.2.1 软件维护	660
29.2.3 清晰盒规格说明	631	31.2.2 软件再工程过程模型	661
29.3 净室设计	631	31.3 逆向工程	663
29.3.1 设计求精与验证	632	31.3.1 数据的逆向工程	664
29.3.2 设计验证的优点	634	31.3.2 处理的逆向工程	665
29.4 净室测试	635	31.3.3 用户界面的逆向工程	665
29.4.1 统计使用测试	636	31.4 重构	666
29.4.2 认证	636	31.4.1 代码重构	666
29.5 小结	637	31.4.2 数据重构	667
参考文献	638	31.5 正向工程	668
习题与思考题	638		

4.3.2 自适应软件开发	64	6.6 小结	109
4.3.3 动态系统开发方法	66	参考文献	110
4.3.4 Scrum	66	习题与思考题	110
4.3.5 Crystal	68	推荐读物与阅读信息	111
4.3.6 特征驱动开发	68	第7章 需求工程	112
4.3.7 敏捷建模	69	7.1 连接设计和构造的桥梁	113
4.4 小结	71	7.2 需求工程任务	113
参考文献	71	7.2.1 起始	114
习题与思考题	72	7.2.2 导出	114
推荐读物与阅读信息	73	7.2.3 精化	115
第二部分 软件工程实践			
第5章 软件工程实践综述	76	7.2.4 协商	115
5.1 概念	77	7.2.5 规格说明	115
5.1.1 实践的精髓	77	7.2.6 确认	116
5.1.2 核心原则	78	7.2.7 需求管理	116
5.2 沟通实践	79	7.3 启动需求工程过程	118
5.3 策划实践	82	7.3.1 确认共利益者	118
5.4 建模实践	84	7.3.2 识别多种观点	118
5.4.1 分析建模原则	85	7.3.3 协同合作	119
5.4.2 设计建模原则	86	7.3.4 首次提问	119
5.5 构造实践	89	7.4 导出需求	120
5.5.1 编码原则和概念	89	7.4.1 协同需求收集	120
5.5.2 测试原则	90	7.4.2 质量功能部署	123
5.6 部署	92	7.4.3 用户场景	124
5.7 小结	93	7.4.4 导出工作产品	125
参考文献	94	7.5 开发用例	125
习题与思考题	95	7.6 构建分析模型	129
推荐读物与阅读信息	95	7.6.1 分析模型的元素	129
第6章 系统工程	97	7.6.2 分析模式	132
6.1 基于计算机的系统	98	7.7 协商需求	133
6.2 系统工程层次结构	99	7.8 确认需求	134
6.2.1 系统建模	100	7.9 小结	135
6.2.2 系统仿真	101	参考文献	135
6.3 业务过程工程概述	102	习题与思考题	136
6.4 产品工程概述	103	推荐读物与阅读信息	137
6.5 系统建模	105	第8章 构建分析模型	139
6.5.1 Hatley-Pirbhai建模	105	8.1 需求分析	140
6.5.2 UML系统建模	107	8.1.1 整体目标和原理	140

8.2 分析建模的方法	142	9.3.6 功能独立	186
8.3 数据建模概念	143	9.3.7 求精	186
8.3.1 数据对象	143	9.3.8 重构	187
8.3.2 数据属性	143	9.3.9 设计类	188
8.3.3 关系	144	9.4 设计模型	190
8.3.4 基数和形态	145	9.4.1 数据设计元素	190
8.4 面向对象的分析	146	9.4.2 体系结构设计元素	191
8.5 基于场景建模	147	9.4.3 接口设计元素	191
8.5.1 编写用例	147	9.4.4 构件级设计元素	192
8.5.2 开发活动图	152	9.4.5 部署级设计元素	193
8.5.3 泳道图	153	9.5 基于模式的软件设计	194
8.6 面向流的建模	153	9.5.1 描述设计模式	194
8.6.1 创建数据流模型	154	9.5.2 在设计中使用模式	195
8.6.2 创建控制流模型	156	9.5.3 框架	195
8.6.3 控制规格说明	157	9.6 小结	196
8.6.4 处理规格说明	158	参考文献	196
8.7 基于类的建模	159	习题与思考题	197
8.7.1 识别分析类	160	推荐读物与阅读信息	197
8.7.2 描述属性	162	第10章 进行体系结构设计	199
8.7.3 定义操作	162	10.1 软件体系结构	200
8.7.4 CRC建模	164	10.1.1 什么是体系结构	200
8.7.5 关联和依赖	169	10.1.2 为什么体系结构如此重要	201
8.7.6 分析包	170	10.2 数据设计	201
8.8 生成行为模型	171	10.2.1 体系结构级的数据设计	201
8.8.1 识别用例事件	171	10.2.2 构件级的数据设计	202
8.8.2 状态表现	171	10.3 体系结构风格和模式	203
8.9 小结	174	10.3.1 体系结构风格的简单分类	203
参考文献	175	10.3.2 体系结构模式	206
习题与思考题	175	10.3.3 组织和求精	207
推荐读物与阅读信息	176	10.4 体系结构设计	208
第9章 设计工程	178	10.4.1 系统的环境表示	208
9.1 软件工程中的设计	179	10.4.2 定义原始模型	209
9.2 设计过程和设计质量	180	10.4.3 将体系结构精化为构件	210
9.3 设计概念	183	10.4.4 描述系统实例	211
9.3.1 抽象	183	10.5 评估可选的体系结构设计	212
9.3.2 体系结构	183	10.5.1 体系结构权衡分析方法	212
9.3.3 模式	184	10.5.2 体系结构复杂性	214
9.3.4 模块化	184	10.5.3 体系结构描述语言	214
9.3.5 信息隐蔽	185	10.6 映射数据流到软件体系结构	215

10.6.1 变换流	215	12.3.1 用户分析	260
10.6.2 事务流	215	12.3.2 任务分析和建模	261
10.6.3 变换映射	215	12.3.3 显示内容分析	265
10.6.4 事务映射	221	12.3.4 工作环境分析	265
10.6.5 精化体系结构设计	224	12.4 界面设计步骤	266
10.7 小结	224	12.4.1 应用界面设计步骤	266
参考文献	225	12.4.2 用户界面设计模式	268
习题与思考题	225	12.4.3 设计问题	269
推荐读物与阅读信息	226	12.5 设计评估	272
第11章 构件级设计建模	228	12.6 小结	273
11.1 什么是构件	229	参考文献	274
11.1.1 面向对象的观点	229	习题与思考题	274
11.1.2 传统观点	230	推荐读物与阅读信息	275
11.1.3 过程相关的观点	232	第13章 软件测试策略	276
11.2 设计基于类的构件	233	13.1 软件测试的策略性方法	277
11.2.1 基本设计原则	233	13.1.1 验证与确认	277
11.2.2 构件级设计指导方针	236	13.1.2 软件测试的组织	278
11.2.3 内聚性	236	13.1.3 传统软件体系结构的测试策略	279
11.2.4 耦合性	238	13.1.4 面向对象软件体系结构的 测试策略	280
11.3 实施构件级设计	240	13.1.5 测试完成的标准	281
11.4 对象约束语言	244	13.2 策略问题	281
11.5 设计传统构件	246	13.3 传统软件的测试策略	282
11.5.1 图形化设计表示	246	13.3.1 单元测试	282
11.5.2 表格式设计表示	247	13.3.2 集成测试	284
11.5.3 程序设计语言	248	13.4 面向对象软件的测试策略	289
11.5.4 设计表示方法的比较	249	13.4.1 面向对象环境中的单元测试	289
11.6 小结	250	13.4.2 面向对象环境中的集成测试	289
参考文献	250	13.5 确认测试	290
习题与思考题	251	13.5.1 确认测试准则	290
推荐读物与阅读信息	251	13.5.2 配置评审	290
第12章 完成用户界面设计	253	13.5.3 α 测试与 β 测试	290
12.1 黄金规则	254	13.6 系统测试	292
12.1.1 置用户于控制之下	254	13.6.1 恢复测试	292
12.1.2 减轻用户的记忆负担	255	13.6.2 安全测试	292
12.1.3 保持界面一致	256	13.6.3 压力测试	293
12.2 用户界面的分析与设计	257	13.6.4 性能测试	293
12.2.1 用户界面分析和设计模型	257	13.7 调试技巧	294
12.2.2 用户界面分析和设计过程	258	13.7.1 调试过程	294
12.3 界面分析	260		

13.7.2 心理因素	295	14.9.2 从行为模型中导出的测试	324
13.7.3 调试策略	296	14.10 针对特定环境、体系结构和 应用系统的测试	325
13.7.4 错误改正	298	14.10.1 图形用户界面测试	325
13.8 小结	298	14.10.2 客户/服务器体系结构测试	325
参考文献	299	14.10.3 测试文档和帮助设施	326
习题与思考题	299	14.10.4 实时系统的测试	327
推荐读物与阅读信息	300	14.11 测试模式	328
第14章 测试战术	302	14.12 小结	329
14.1 软件测试基础	303	参考文献	329
14.2 黑盒测试与白盒测试	304	习题与思考题	330
14.3 白盒测试	305	推荐读物与阅读信息	331
14.4 基本路径测试	305	第15章 产品度量	333
14.4.1 流图表示	305	15.1 软件质量	334
14.4.2 独立程序路径	307	15.1.1 McCall的质量因素	334
14.4.3 导出测试用例	308	15.1.2 ISO 9126质量因素	335
14.4.4 图矩阵	310	15.1.3 向量化视图变迁	336
14.5 控制结构测试	310	15.2 产品度量框架	336
14.5.1 条件测试	311	15.2.1 测度、度量和指标	336
14.5.2 数据流测试	311	15.2.2 产品度量的挑战	337
14.5.3 循环测试	311	15.2.3 测量原则	338
14.6 黑盒测试	312	15.2.4 面向目标的软件测量	338
14.6.1 基于图的测试方法	313	15.2.5 有效软件度量的属性	339
14.6.2 等价划分	314	15.2.6 产品度量全景	339
14.6.3 边界值分析	315	15.3 分析模型的度量	341
14.6.4 正交数组测试	315	15.3.1 基于功能的度量	341
14.7 面向对象测试方法	317	15.3.2 规格说明质量的度量	344
14.7.1 面向对象概念的测试用例		15.4 设计模型的度量	344
设计的含义	318	15.4.1 体系结构设计度量	345
14.7.2 传统测试用例设计方法的 可应用性	318	15.4.2 面向对象设计的度量	346
14.7.3 基于故障的测试	318	15.4.3 面向类的度量——CK度量集	347
14.7.4 测试用例与类层次	319	15.4.4 面向类的度量——MOOD 度量集	350
14.7.5 基于场景的测试	319	15.4.5 Lorenz与Kidd提出的面向 对象度量	350
14.7.6 表层结构和深层结构的测试	320	15.4.6 构件级设计度量	351
14.8 类级可应用的测试方法	321	15.4.7 面向操作的度量	352
14.8.1 面向对象的随机测试	321	15.4.8 用户界面设计度量	353
14.8.2 类级的划分测试	322	15.5 源代码的度量	353
14.9 类间测试用例设计	323		
14.9.1 多类测试	323		

15.6 测试的度量	354	17.5.2 评估商业价值的度量	388
15.6.1 应用于测试的Halstead度量	355	17.6 WebApp项目的“最坏实践”	389
15.6.2 面向对象测试的度量	355	17.7 小结	390
15.7 维护的度量	356	参考文献	390
15.8 小结	357	习题与思考题	391
参考文献	357	推荐读物与阅读信息	392
习题与思考题	359	第18章 WebApp分析	393
推荐读物与阅读信息	359	18.1 WebApp的需求分析	394
第三部分 应用Web工程			
第16章 Web工程	362	18.1.1 用户层次	394
16.1 基于Web的系统及应用的特点	363	18.1.2 开发用例	395
16.2 WebApp工程的层次	365	18.1.3 精化用例模型	397
16.2.1 过程	365	18.2 WebApp的分析模型	397
16.2.2 方法	366	18.3 内容模型	398
16.2.3 工具与技术	366	18.3.1 定义内容对象	398
16.3 Web工程过程	366	18.3.2 内容关系与层次	399
16.3.1 定义框架	367	18.3.3 WebApp的分析类	399
16.3.2 精化框架	369	18.4 交互模型	400
16.4 Web工程的最佳实践	369	18.5 功能模型	402
16.5 小结	371	18.6 配置模型	403
参考文献	371	18.7 关系导航分析	404
习题与思考题	371	18.7.1 关系分析——关键问题	404
推荐读物与阅读信息	372	18.7.2 导航分析	405
第17章 开始一个WebApp项目	373	18.8 小结	406
17.1 表达基于Web的系统	374	参考文献	406
17.1.1 表达问题	374	习题与思考题	407
17.1.2 WebApp的需求收集	375	推荐读物与阅读信息	407
17.1.3 分析模型的过渡	379	第19章 WebApp设计	409
17.2 策划Web工程项目	379	19.1 Web工程的设计问题	410
17.3 Web工程团队	380	19.1.1 设计与WebApp质量	410
17.3.1 人员	380	19.1.2 设计目标	412
17.3.2 组建团队	381	19.2 WebE设计金字塔	413
17.4 Web工程的项目管理问题	382	19.3 WebApp界面设计	414
17.4.1 WebApp策划——外包	382	19.3.1 界面设计原则与指导方针	414
17.4.2 WebApp策划——内部Web 工程	385	19.3.2 界面控制机制	418
17.5 Web工程与WebApp的度量	387	19.3.3 界面设计工作流	418
17.5.1 Web工程工作量的度量	387	19.4 美学设计	420

19.5.1 内容对象	421	20.7 配置测试	452
19.5.2 内容设计问题	422	20.7.1 服务器端问题	453
19.6 体系结构设计	423	20.7.2 客户端问题	453
19.6.1 内容体系结构	423	20.8 安全性测试	453
19.6.2 WebApp体系结构	425	20.9 性能测试	455
19.7 导航设计	426	20.9.1 性能测试的目标	455
19.7.1 导航语义	426	20.9.2 负载测试	455
19.7.2 导航语法	427	20.9.3 压力测试	456
19.8 构件级设计	428	20.10 小结	458
19.9 超媒体设计模式	428	参考文献	458
19.10 面向对象的超媒体设计方法	429	习题与思考题	459
19.10.1 OOHDM的概念设计	430	推荐读物与阅读信息	460
19.10.2 OOHDM的导航设计	431		
19.10.3 抽象界面设计与实现	431		
19.11 WebApp的设计度量	431		
19.12 小结	432		
参考文献	433		
习题与思考题	434		
推荐读物与阅读信息	435		
第20章 WebApp测试	436		
20.1 WebApp的测试概念	436		
20.1.1 质量维度	437		
20.1.2 WebApp环境中的错误	438		
20.1.3 测试策略	438		
20.1.4 测试策划	438		
20.2 测试过程概述	439		
20.3 内容测试	442		
20.3.1 内容测试的目标	442		
20.3.2 数据库测试	443		
20.4 用户界面测试	444		
20.4.1 界面测试策略	444		
20.4.2 测试界面机制	445		
20.4.3 测试界面语义	446		
20.4.4 可用性测试	447		
20.4.5 兼容性测试	448		
20.5 构件级测试	449		
20.6 导航测试	451		
20.6.1 测试导航语法	451		
20.6.2 测试导航语义	451		
		第四部分 管理软件项目	
		第21章 项目管理	462
		21.1 管理涉及的范围	463
		21.1.1 人员	463
		21.1.2 产品	463
		21.1.3 过程	464
		21.1.4 项目	464
		21.2 人员	464
		21.2.1 共利益者	465
		21.2.2 团队负责人	465
		21.2.3 软件团队	466
		21.2.4 敏捷团队	468
		21.2.5 协调和通信问题	469
		21.3 产品	470
		21.3.1 软件范围	470
		21.3.2 问题分解	470
		21.4 过程	471
		21.4.1 合并产品和过程	471
		21.4.2 过程分解	472
		21.5 项目	473
		21.6 W ⁵ HH原则	474
		21.7 关键实践	475
		21.8 小结	475
		参考文献	476
		习题与思考题	476
		推荐读物与阅读信息	477

第22章 过程和项目度量	479
22.1 过程领域和项目领域中的度量	480
22.1.1 过程度量和软件过程改进	480
22.1.2 项目度量	481
22.2 软件测量	483
22.2.1 面向规模的度量	483
22.2.2 面向功能的度量	484
22.2.3 调和代码行和功能点的度量方法	485
22.2.4 面向对象的度量	486
22.2.5 面向用例的度量	487
22.2.6 Web工程项目度量	487
22.3 软件质量度量	489
22.3.1 测量质量	489
22.3.2 缺陷排除效率	490
22.4 在软件过程中集成度量	491
22.4.1 支持软件度量的论点	492
22.4.2 建立基线	492
22.4.3 度量收集、计算和评估	492
22.5 小型组织的度量	493
22.6 制定软件度量大纲	494
22.7 小结	496
参考文献	496
习题与思考题	497
推荐读物与阅读信息	498
第23章 估算	500
23.1 对估算的观察	501
23.2 项目策划过程	501
23.3 软件范围和可行性	502
23.4 资源	503
23.4.1 人力资源	503
23.4.2 可复用软件资源	504
23.4.3 环境资源	504
23.5 软件项目估算	505
23.6 分解技术	505
23.6.1 软件规模估算	506
23.6.2 基于问题的估算	506
23.6.3 基于LOC估算的实例	507
23.6.4 基于FP估算的实例	509
23.6.5 基于过程的估算	510
23.6.6 基于过程估算的实例	511
23.6.7 基于用例的估算	511
23.6.8 基于用例的估算实例	512
23.6.9 调和不同的估算方法	513
23.7 经验估算模型	514
23.7.1 估算模型的结构	514
23.7.2 COCOMO II模型	515
23.7.3 软件方程式	516
23.8 面向对象项目的估算	517
23.9 特殊的估算技术	517
23.9.1 敏捷开发的估算	518
23.9.2 Web工程项目的估算	518
23.10 自行开发或购买的决策	519
23.10.1 创建决策树	520
23.10.2 外包	521
23.11 小结	522
参考文献	523
习题与思考题	523
推荐读物与阅读信息	524
第24章 项目进度安排	525
24.1 基本概念	526
24.2 项目进度安排	527
24.2.1 基本原则	528
24.2.2 人员与工作量之间的关系	529
24.2.3 工作量分配	530
24.3 为软件项目定义任务集	531
24.3.1 任务集举例	531
24.3.2 主要任务的求精	532
24.4 定义任务网络	533
24.5 进度安排	533
24.5.1 时序图	535
24.5.2 跟踪进度	536
24.5.3 跟踪OO项目的进展	537
24.6 获得值分析	538
24.7 小结	539
参考文献	540
习题与思考题	540
推荐读物与阅读信息	541

第25章 风险管理	543
25.1 被动风险策略和主动风险策略	544
25.2 软件风险	544
25.3 风险识别	545
25.3.1 评估整体项目风险	546
25.3.2 风险因素和驱动因子	547
25.4 风险预测	548
25.4.1 建立风险表	548
25.4.2 评估风险影响	549
25.5 风险求精	551
25.6 风险缓解、监测和管理	552
25.7 RMMM计划	553
25.8 小结	555
参考文献	555
习题与思考题	555
推荐读物与阅读信息	556
第26章 质量管理	558
26.1 质量概念	559
26.1.1 质量	559
26.1.2 质量控制	560
26.1.3 质量保证	560
26.1.4 质量成本	560
26.2 软件质量保证	561
26.2.1 背景	562
26.2.2 SQA活动	562
26.3 软件评审	563
26.3.1 软件缺陷对成本的影响	564
26.3.2 缺陷放大和消除	564
26.4 正式技术评审	566
26.4.1 评审会议	566
26.4.2 评审报告和记录保存	567
26.4.3 评审指导原则	567
26.4.4 样本驱动评审	568
26.5 SQA的形式化方法	570
26.6 基于统计的软件质量保证	570
26.6.1 一个普通的例子	570
26.6.2 软件工程中的六西格玛	571
26.7 软件可靠性	572
26.7.1 可靠性和可用性的测量	572
26.7.2 软件安全	573
26.8 ISO 9000质量标准	574
26.9 SQA计划	575
26.10 小结	576
参考文献	577
习题与思考题	578
推荐读物与阅读信息	578
第27章 变更管理	580
27.1 软件配置管理	581
27.1.1 SCM场景	581
27.1.2 配置管理系统元素	582
27.1.3 基线	582
27.1.4 软件配置项	583
27.2 SCM中心存储库	584
27.2.1 中心存储库的作用	585
27.2.2 一般特征和内容	585
27.2.3 SCM特征	585
27.3 SCM过程	587
27.3.1 软件配置中对象的标识	587
27.3.2 版本控制	588
27.3.3 变更控制	589
27.3.4 配置审核	592
27.3.5 状态报告	593
27.4 Web工程的配置管理	593
27.4.1 WebApp的配置管理问题	594
27.4.2 WebApp的配置对象	595
27.4.3 内容管理	595
27.4.4 变更管理	597
27.4.5 版本控制	599
27.4.6 审核和报告	599
27.5 小结	600
参考文献	601
习题与思考题	602
推荐读物与阅读信息	602
第五部分 软件工程高级课题	
第28章 形式化方法	606
28.1 基本概念	607
28.1.1 非形式化方法的缺陷	607

第1章 软件工程介绍

要点浏览

概念：计算机软件是由专业人员开发并长期维护的软件产品。完整的软件产品包括了在各种不同容量和体系结构计算机上的可执行程序，运行过程中产生的各种结果，以及以硬拷贝和电子表格等多种方式存在的软件文档。

人员：软件工程师开发软件并提供技术支持，事实上，产业界中几乎每个人都间接或直接地用到它。

重要性：在我们的生活中无所不在，并

且普遍深入到商业、文化和日常生活多个方面。

步骤：就像开发任何成功的产品一样，我们采用灵活可调的软件开发方法，完成可满足使用者需求的高质量的软件产品。

工作产品：从软件工程的观点来看，最终产品是计算机软件，包括程序、内容（数据）和文档；而从用户角度来看，最终产品是可以改善生活和工作质量的信息。

质量保证措施：阅读本书的后面部分，选择切合你工作情况的观点和意见，通过实际应用验证正确与否。

关键概念
应用分类
挑战
退化
演化
失效曲线
历史
遗留系统
神话
软件特点
软件定义

你是否注意到，某些新科技的发明创造会给其他一些看似无关的技术领域、商业企业、公众甚至整个社会文化带来深远而出人意料的影响和作用。这就是所谓的“意外效应法则”（law of unintended consequences）。

今天，计算机软件已经成为世界舞台上最为重要的科技领域，并且是“意外效应法则”的一个最好的体现。在20世纪50年代，没有人曾预料到软件科学会成为今天商业、科学和工程所必需的技术。它促进了新科技的创新（例如基因工程）、现代科技的发展（例如远程通信），以及传统技术（例如印刷业）向现代科技的过渡。软件技术已经成为个人电脑革命的推动力量，消费者可以很容易在附近的商店购买到包装好的软件产品，一家软件公司可以比传统工业时代的许多公司更大、更有影响力。在大量应用软件的驱动下，因特网将迅速发展，并将使人们生活的诸多方面（从图书馆搜索、消费购物到年轻人的约会习惯）产生革命性的变化。

没有人曾想到软件可嵌入到各种系统中，这些系统包括交通运输、医疗、远程通信、军事、工业、娱乐、办公设备等等。如果笃信“意外效应法则”的话，还有很多结果和影响是我们尚未预料到的。

那么最终，随着时间的推移，将有数百万的电脑程序需要进行纠错、适应性调整和优化，这些维护工作将耗费比开发新软件更多的人力、物力。

“创新观念和科技发现是经济增长的推进器。”

——“华尔街日报”

随着软件重要性的日渐凸现，软件业界一直试图开发新的技术，使得高质量计算机程序

的开发和维护更容易、更快捷，成本更低廉。某些技术注重于特殊应用领域（例如：网站设计和实现）；有些技术着眼于科技领域（例如：面向对象系统，面向方面的程序设计）；还有一些覆盖面很宽（例如：像LINUX这样的操作系统）。然而，我们仍需开发一种软件技术，可以实现上述所有需求。虽然未来这种技术产生的可能性很小，但很多人仍坚信这将是一个正确的方向，并为之付出了工作、安全和终生的努力。

本书阐述了一个包含过程、一系列方法和工具的框架，我们称之为软件工程。本书所面向的读者是正在进行软件开发、需要保证软件正确性的软件工程师。

“在当代社会，工程的作用是提供改善人们物质生活的系统和产品，从而使得生活更轻松、健康、安全和惬意。”

——Richard Fairley and Mary Willshire

1.1 软件角色的演化



软件既是产品也是交付产品的载体。

现在的软件技术具有产品和产品生产载体的双重作用。作为一个产品，它显示了由计算机硬件体现的计算能力，更广泛地说，显示的是由一个可被本地硬件设备访问的计算机网络体现的计算潜力。无论是在手机还是在大型计算机中，软件都扮演着信息转换的角色：产生、管理、查询、修改、显示或者传递各种不同的信息——简单如几个比特的传递或复杂如多媒体演示。而作为产品生产的载体，软件提供了计算机控制（操作系统）、信息通信（网络）以及应用程序开发和控制（软件工具和环境）的基础平台。

软件传递了我们这个时代最重要的产品——信息。它转换个人数据（例如个人金融交易）以便信息在一定范围内发挥更大的作用；它通过管理商业信息提升竞争力；它为世界范围的信息网络提供通路（比如因特网），并对各类格式的信息提供不同的查询方式。

计算机软件的地位在50多年的时间中发生了很大的变化。硬件性能的极大提高、计算体系结构的巨大变化、内存和存储容量的扩大、还有种类繁多的输入和输出方法都使得计算机系统的结构变得更加复杂，功能更加强大。如果系统开发成功，复杂的结构和功能可以产生惊人的效果，但是同时复杂性也给系统开发人员带来巨大的挑战。

人们对计算机和软件及其对文化的冲击的理解发生了很大的变化，20世纪70、80年代出版的一些畅销书对此提出了历史性的见解。Osborne[OSB79]称之为“新的工业革命”；Toffler[TOF80]称微电子是人类历史上“第三次浪潮”的一部分；Naisbitt[NAI82]更预言了工业社会到“信息社会”的变革。Feigenbaum和McCorduck [FEI83]认为信息和知识（由计算机控制）将成为21世纪能源的焦点。Stoll[STO89]认为由网络和软件构建的“电子社区”将是世界范围内知识交换的关键。这些作者的观点都是客观正确的。

20世纪90年代伊始，Toffler[TOF90]描述了一种“权力移交”的现象：计算机和软件带来了“知识的民主化”，随着这种变化，传统的高度集中的权力结构（政府，教育，工业，经济，军事）被分化了。Yourdon[YOU92]曾忧虑美国公司会丧失他们在软件相关产业的竞争优势，并预言了“美国程序员的

WebRef

对软件工业的回顾参见 www.softwarehistory.org。



如果有时间，建议可以选读1本或几本经典著作。请注意这些专家在预测未来的事件和技术发展时犯了何种错误。请务必保持一种谦逊的态度，因为没有人会知道我们构建的系统的未来。

衰落和垮台”。Hammer和Champy [HAM93]认为信息技术将会在“公司的重组”中起到关键作用。20世纪90年代中期，计算机和软件的普及却招致“新勒德”分子的鲁莽攻击（在James Brook和Iain Boal的《Resisting the Virtual Life (反对虚拟生活)》，Stephen Talbot的《The Future Does Not Compute (未来不是计算)》等书中），这些作者将计算机刻意妖魔化，片面地强调其合法性，却忽视计算机所带来的深远益处[LEV95]。

“计算机将很多事情变得简单，但是这些事情中很多都是无关紧要的。”

——Andy Rooney

20世纪90年代末期，Yourdon [YOU96]对计算机软件的职业前景重新进行了评估，并认为美国程序员职业将重新“复兴”。随着网络的重要意义逐渐显现，Yourdon的观点转变证明是正确的。直至20世纪末，焦点再次转移，这次是Y2K（千年虫问题）“时间炸弹”的影响（例如[YOU98a], [KAR99]）。虽然那些Y2K宿命论者的可怕预言有点反应过激，但他们流传甚广的论著却使普及软件的观念家喻户晓。

WebRef

对软件相关论题的广泛评论参见
www.yourdon.com。

在21世纪初，Johnson[JOH01]讨论了“层创进化”(emergence)现象，即相对简单的实体相互结合时“自我组织成更为智能、更具有适应性”的系统。Yourdon[YOU02]再次提到了“9·11”这一悲剧事件，以此讨论全球恐怖行为对IT业界的后续冲击。Wolfram[WOL02]发表了一篇关于“新科学”的论文，提出了一种主要基于复杂软件仿真的统一理论。Daconta及其同事[DAC03]就“语义网络”的发展以及如何改变人们在网络世界的交互方式进行了论述。

35

“因为我曾深入到人类将亲眼目睹的未来，我已经见到了世界的前景和将要出现的所有奇迹。”

——Tennyson

现在，庞大的软件产业已经成为工业经济中的主导因素。早期的独立程序员也已经被多个专业的软件开发团队所代替，他们分别关注复杂的应用系统中某一个技术部分。然而同过去的独立程序员一样，开发现代计算机系统时，软件开发人员依然面临同样的问题¹：

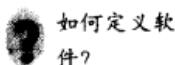
- 为什么软件需要如此长的开发时间？
- 为什么开发成本居高不下？
- 为什么在将软件交付顾客使用之前，我们无法找到所有的错误？
- 为什么维护已有的程序要花费高昂的时间和人力代价？
- 为什么软件开发和维护的过程难以度量？

这种种问题显示了业界对软件以及软件开发方式的关注。这种关注促使了业界对软件工程实践的采纳。

¹ 在一本优秀的关于软件业务的论文集中，Tom DeMarco [DEM95]提出了相反的看法。他认为：“我们更应该总结使得当今的软件开发费用低廉的成功经验，而不是不停地质问为何软件开发成本高昂。这会有助于我们继续保持软件产业的杰出成就。”

1.2 软件

在1970年，顶多有1%的人可以给出“计算机软件”的定义。今天，绝大多数专业人员和许多普通人认为他们对软件大概了解。真的是这样吗？



36

来自教科书的关于软件的定义也许是：软件是（1）指令的集合（计算机程序），通过执行这些指令可以满足预期的特征、功能和性能需求；（2）数据结构，它使得程序可以充分利用信息；（3）描述程序操作和使用的文档。当然，还有更完整的解释，但是我们所需要的并非仅仅一个正式的定义。

为了更好地理解“软件”（实质上是软件工程）的意义，将软件和其他人工产品的特点加以区分是非常重要的。软件是逻辑的系统元素而非物理的系统元素。因此，软件和硬件具有完全不同的特性：

1. 软件是设计开发的，而不是传统意义上生产制造的。

虽然在软件开发和硬件制造间存在某些相同点，但二者根本不同。两者均可通过优秀的设计获得高产品品质，然而硬件制造阶段中的质量问题对于软件来说是不存在的或者易于纠正的；二者都依赖人，但是人员和工作成果之间的对应关系是完全不同的（见第24章）；它们都需要构建产品，但是构建方法不同。软件产品成本主要在于开发设计，因此不能像管理制造项目那样管理软件开发项目。

2. 软件不会“磨损”。

图1-1描述了以时间为变量的硬件的失效率。这个被称为“浴缸曲线”的关系图显示：硬件在早期具有相对较高的失效率（这种失效通常来自设计或生产缺陷）。缺陷被逐个纠正之后，失效率也降低，并在一段时间内保持平稳（理想情况下很低）。然而，随着时间推移，因为灰尘、震动、不当使用、温度超限以及其他环境问题所累积的硬件组件损耗将再次抬高失效率。简而言之，硬件开始“磨损”了。

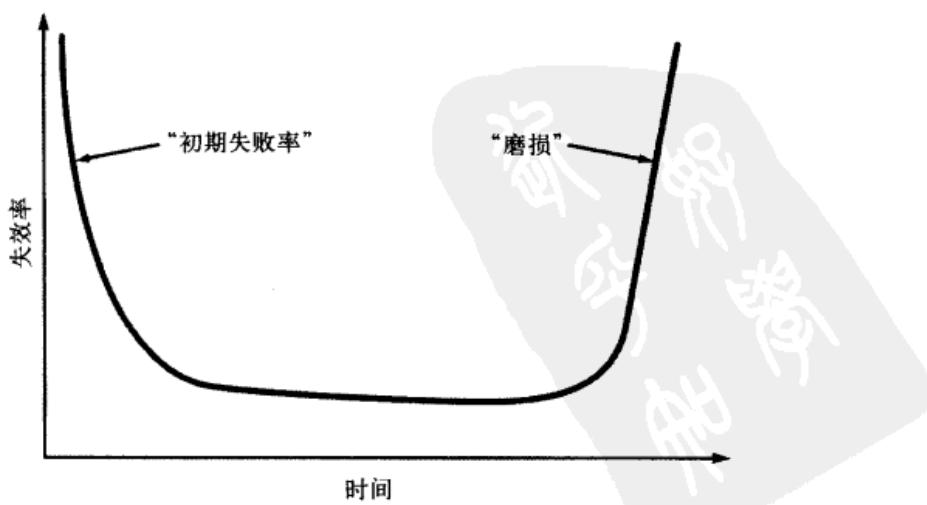


图1-1 硬件失效曲线图

而软件是不会受到引起硬件磨损的环境问题的影响的。因此，从理论上来说，软件的失



若希望降低软件退化，需要改善软件的设计（见第9~12章）

效率曲线应该呈现为图1-2所示的“理想曲线”。未知的缺陷将在程序的生命周期的前期造成高失效率。然而随着错误被纠正（理想情况下，不会引入新的错误），曲线将如图中所示，趋于平缓。“理想曲线”只是软件实际失效模型的粗略简化（更多信息请见第26章）。然而其含义很明显——软件不会磨损，但是软件退化的确存在。

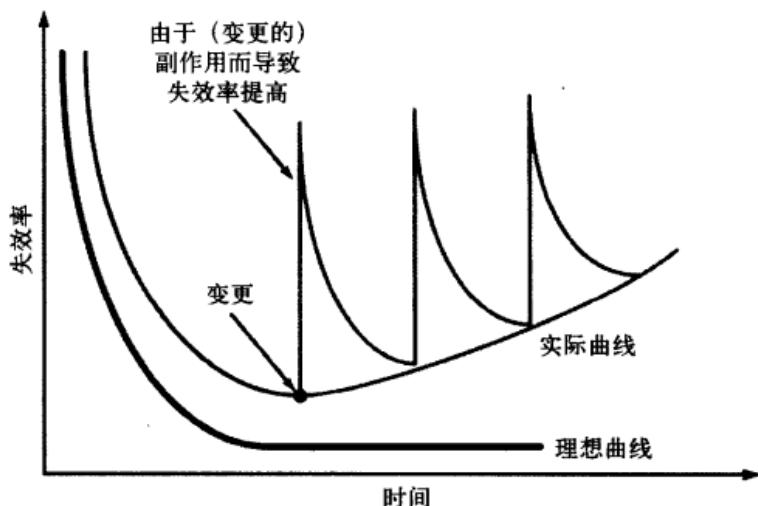


图1-2 软件失效曲线图

POINT
软件工程方法的目的是降低图1-2中突变的幅度及实际失效曲线的斜率。

这个似乎矛盾的现象用图1-2所示的“实际曲线”可以很好地解释。在完整的生存周期里²，软件的每次变更都可能引入新的错误，使得失效率如图1-2所示的那样陡然上升。在曲线回到最初的稳定状态前，新的变更会引起曲线又一次上升。就这样，最小的失效率水平逐渐上升，可以说，不断的变更更是软件退化的根本原因。

38

磨损的另一方面同样说明了软硬件的不同。磨损的硬件部件可以用备用的构件替换；而软件却不存在备用构件。每个软件的错误都暗示了设计的缺陷或者在从设计转化到机器可执行代码的过程中产生的错误。因此，软件维护比硬件维护更为复杂。

3. 虽然整个工业向着基于构件的构造模式发展，然而大多数软件仍是根据实际的顾客需求定制的。

在设计和建造计算机控制硬件时，设计师首先画出数字电路的示意图，通过基本分析确保一定功能的实现，再从器件目录中选取现有的数字电路块。每一块集成电路都有编号、定义好的确认的功能、定义好的功能接口和标准的集成指南。每块器件在选定后都可通过订购方式获取。

POINT
大多数软件还是主要采用用户定制的方式。

工程学科的发展将产生一系列标准的设计构件。标准螺丝及可订购的电子部件只是机械工程师和电子工程师在设计新系统时所使用的上千种标准构件中的两种。可复用构件的使用可以使得工程师专心于设计中真正创新的部分。在硬件设计中，构件复用是工程进程中通用的方法。而在软件设计中，大规模的复用还刚刚开始尝试。

² 事实上，在软件开发开始，早在第一个版本发布之前的很长时间，用户就可能提出变更要求。

“思想的积木构建成思想的大厦。”

——Jason Zebehazy

软件的构件应该设计实现成可在不同程序中复用的组件。现代的可复用构件封装了数据和对数据的处理，使得软件工程师能够利用可复用的构件构建新的应用程序³。例如，现在的用户界面就使用可复用构件构造图形窗口、下拉菜单和各种交互机制，并采用可复用构件库存储和管理构造用户界面所需要的数据结构和处理逻辑。

39

1.3 软件特性的变化

今天，计算机软件可分为七个大类，软件工程正面临持续的挑战。

系统软件。系统软件是一套服务于其他程序的程序。某些系统软件（例如：编译器，编辑器，文件管理实用程序）处理复杂但确定的⁴信息结构；另一些系统应用程序（例如：操作系统构件，驱动程序，网络软件，远程通信处理器）主要处理的是不确定的数据。对于这两种情况，系统软件多具有以下特点：和硬件大量地交互；多用户的使用负荷沉重；需要调度、资源共享和复杂的多进程管理的同步操作；复杂的数据结构；以及多种外部接口。

WebRef

目前最大的shareware/freeware库见：
shareware.cnet.com。

应用软件。应用软件是一些可以满足特定业务需要的独立应用程序。应用软件处理商务或技术数据，以协助业务操作和管理或技术决策。除了传统数据处理的应用程序，应用软件也被用于业务领域的实时控制（例如：销售点的交易处理，实时制造过程控制）。

工程/科学软件。带着“数字处理”算法的标签，工程和科学软件涵盖了广泛的应用领域，从天文学到火山学，从自动压力分析到航天飞机轨道动力学，从分子生物学到自动制造业。不过，当今科学工程领域的应用软件已经不仅仅局限于传统的数值算法。计算机辅助设计、系统仿真和其他的交互性应用程序已经呈现出实时和系统软件的特性。

嵌入式软件。嵌入式软件存在于某个产品或者系统中，可实现和控制面向最终使用者和系统本身特性和功能。嵌入式软件可以执行有限的复杂的功能（例如：微波炉的按键控制）或者提供重要的功能和控制能力（例如：汽车中的燃油控制，仪表板显示，刹车系统等等）。

产品线软件。产品的设计方向是为多个不同用户的使用提供特定功能，关注有限的特定市场（例如：库存控制产品）或者大众消费品市场（例如：文字处理，电子制表软件，电脑绘图，多媒体，娱乐，数据库管理，个人及公司财务应用）。

Web应用软件。“Web应用”的概念涵盖了宽泛的应用程序产品。最简单的可以是一组超文本链接文件，仅仅用文本和有限的图形表达信息。然而，随着电子商务和B2B应用的日益重要，网络应用正在发展为复杂的计算环境，不仅为最终用户提供标准特性、计算功能和内容信息，还与企业数据库和商务应用程序相结合。

人工智能软件。人工智能软件利用非数值算法解决计算和直接分析无法解决的复杂问题。这个领域的应用程序包括机器人、专家系统、模式识别（图像和语音）、人工神经网络、定理证明和博弈等。

³ 第30章介绍基于构件的软件工程。

⁴ 软件的确定性是指系统的输入、处理和输出的顺序及时间是可以预测的；软件的不确定性是指系统的输入、处理和输出的顺序和时间是无法提前预测的。

“没有任何一台计算机具备基本常识。”

——Marvin Minsky

全世界数百万的软件工程师在以上各类项目中努力地工作着。有时是建立一个新的系统；而有时只是对现有应用程序进行纠错、适应性调整和升级。一个年轻的软件工程师所经手的项目比他自己的年龄还大是常有的事。对于我们上述讨论的各类系统，上一代的软件工程师都已经留下了遗留系统。我们希望这代工程师留下的遗留系统可以减轻未来工程师的负担。然而，新的挑战也已经逐渐显现出来。

遍在计算。无线网络的快速发展也许将很快促成真正的分布式计算的实现。软件工程师所面临的挑战将是开发系统和应用软件，以使得小型设备、个人电脑和企业应用可以通过大量的网络设施进行通信。

网络资源。万维网已经快速发展为一个计算引擎和内容提供平台。软件工程师新的任务是构建一个简单（例如：个人理财规划）而智能的应用程序，为全世界的最终用户市场提供服务。

“你无法预测未来，但是可以时刻准备着未来。”

——无名氏

开源软件。开源软件就是将系统应用程序（例如：操作系统、数据库、开发环境）源代码开放，并允许用户自行修改。这种方式正在逐步扩展。软件工程师面对的挑战是，开发可以自我描述的源代码，而更重要的是，开发某种技术，以便用户和开发人员都能够了解已经发生的改动，并且知道这些改动如何在软件中体现出来。41

“新经济”。20世纪90年代后期困扰金融市场的网络经济衰退以及21世纪初随之而来的讨论使得许多商业人士相信，新经济已经衰亡。现在，新经济体系依然健康地生存着，只是发展缓慢。它将朝着多点通信和分布式的方向发展。Andy Lippman [LIP02]谈到：

我们正在进入一个新的通信时代，其特征为不再是特定两个实体（两个人或一个人与一台机器）之间的相互连接，而是多个分布的计算机和分散人群之间的多方通信交流。旧有的电话通信方式是“点到点连接”，而新方法是“多点的相互连接”，例如Napster、即时消息软件、短信系统和BlackBerries。

软件工程师面临的挑战是利用最新形成的理念，开发出便于大众传播、产品发布的应用程序。

所有这些新的挑战毫无疑问将遵守“意外效应法则”并且产生现在无法预测的结果（对商务人员，软件工程师和终端使用者）。然而，软件工程师可以做一些准备工作，采用一个足够灵活应变的过程，以适应未来十年中必将发生的科技和商务的种种巨大变化。

“计算机将实现从分析工具……到具有情感的产品的历史性过渡。”——David Vaskevitch

1.4 遗留软件

如上所述，成千上万的计算机程序可以归于7大类应用领域——系统软件、应用软件、工程科学计算软件、嵌入式软件、产品软件、Web应用软件和人工智能软件（见1.3节）。其中某

些是当今最先进的软件——最近才对个人、产业和政府发布。但是另外一些软件则年代较久，甚至过于久远了。

 什么是遗留软件？ 这些旧的软件——通常称为遗留软件，从上世纪60年代起，就成为持续关注的焦点。Dayani-Fard和他的同事[DAY99]这样描述遗留软件：

42 遗留软件系统……在几十年前诞生，它们不断地被修改以满足商业需要和计算平台的变化。这一系统的繁衍使得大型机构十分头痛，因为它们的维护代价高昂且系统演化风险较高。

Liu和他的同事[LIU98]进一步描述道：“许多遗留软件系统仍然支持核心的业务功能，是业务必不可少的支撑。”因此，遗留软件具有生命周期长以及业务关键性的特点。

1.4.1 遗留软件的质量

 如果遇到质量低下的遗留软件该怎么办？ 然而不幸的是，遗留软件同时存在质量差⁵的特点。通常，遗留系统的设计难以扩展，代码令人费解，文档混乱甚至根本没有，测试用例和结果从未归档，变更的历史管理混乱等等，数不清的问题。然而，这些系统仍然“支撑核心的应用并必不可少”[LIU98]。该如何应对这种情况？

最合理的解释也许就是什么也不做，至少在其不得不进行重大变更之前。如果遗留软件可以满足用户的需求并且可靠运行，则不需要修改它。然而，随着时间的推移，遗留系统经常会由于下述原因发生系统演化：

-  遗留软件都进行哪类改变？
- 软件需要修改其适应性，从而可以满足新的计算环境或者技术的需求。
 - 软件必须根据新的业务需求进行升级。
 - 软件必须扩展以具有与更多现代系统和数据库的协作能力。
 - 软件架构必须进行改建以适应多样化的网络环境。

 所有软件工程师都需认识到：变化是不可避免的。不要反对变化。 当这些变化发生时，遗留系统需要经过再工程（参见第31章）以适应未来的多样性。当代软件工程的目标是“修改在进化论基础上建立的方法论”，即“软件系统不断经历变更，新的软件系统从旧系统中建立起来，并且……新旧所有系统都必须具有互操作性和协作性。”[DAY99]。

1.4.2 软件演化

43 不管其应用领域、软件规模或者复杂性如何不同，计算机软件都将随着时间的推移不断演化。变更（通常是称为软件维护）推动了软件演化，它通常是由以下情况引发的：程序纠错，调整软件以适应新的环境，满足用户新特性和功能的需求，以及对软件实施再工程以便在现代应用中发挥作用。Sam Williams [WIL02]这样描述道：

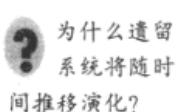
当Windows、Solaris这样的大型程序扩展到3000万~5000万行代码时，成功的项目经理已经认识到，梳理遗留系统的代码与添加新的代码将耗费同样多的时间。简而言之，在十年内，PC微芯片的平均性能已经提高数百倍，然而软件的规模连理性增长都很困难，这已经不再是难言的小秘密，而是整个软件行业的难堪事了。

⁵ 所谓“质量差”是基于现代软件工程思想而言的，这个评判标准对遗留系统有些不公平，因为在遗留软件开发的年代里，现代软件工程的一些概念和原则可能还没有被人们完全理解。

在过去的30年里，Manny Lehman[如LEH97a]和他的同事对工业级的软件和系统进行了详细的分析，试图建立一套“统一软件演化理论”。关于这部分研究的细节不在本书讨论范围内⁶，然而从中衍生出的潜在规律却是值得注意的[LEH97b]。这些规律是：

持续变化规律（1974）。E类型⁷的系统需要不断地进行适应性修改，否则将逐渐偏离需求。

复杂性增长规律（1974）。除非进行维护和精简工作，否则，E类型系统的复杂度将随着系统演化不断增加。



自我调控规律（1974）。E类型系统的发展过程是随着产品发布和过程度量接近于正常，而不断进行自我调控的。

组织稳定性守恒规律（1980）。一个不断演化的E类型系统，其组织在全球范围内的平均有效活动率在产品的生命周期中是保持不变的。

保证通晓性规律（1980）。随着E类型系统的演化，所有相关人员（例如开发人员、销售人员和用户）都必须清楚地了解演化的内容和过程，以便达到满意的演化效果。过度增长会削弱这种控制。因此，在系统的演化过程中，平均增长率保持不变。

持续增长规律（1980）。在整个产品生命周期中，E类型系统的功能性一定是持续增长的，以维持用户满意度。

质量衰减规律（1996）。如果没有严格的维护和适应性调整使之适应运行环境的变化，E类型系统的质量有衰减的趋势。

反馈系统规律（1996）。E类型演化过程可看作是一个多层次、多循环、多代理的反馈系统，必须如此才能在任何合理的基础上，不断取得重大的提高。

Lehman和他的同事们所描述的规律反映了软件工程的内在本质。在本书后面部分，我们将讨论软件过程模型、软件工程方法和管理技术，所有这些都是为了在软件演化过程中保证软件的质量。

1.5 软件神话

软件神话，即关于软件及其开发过程的一些说法被人盲目相信，这可以追溯到信息处理技术发展的初期。神话的一些表象使人们视其为不可捉摸。例如，神话看起来是事实的合理描述（有时的确包含真实的成分），它们符合直觉，并且经常被那些知根知底的实践者拿来宣传。

“在缺少有意义的规范标准的情况下，像软件这样的新兴产业转而依靠民间传说。”

——Tom DeMarco



今天，大多数有见地的软件工程师已经了解到软件神话的本质——它实际上误导了管理者和技术人员对软件开发的态度，从而引发了严重的问题。然而，由于习惯和态度的根深蒂固，至今仍有人对一些软件神话深信不疑。

管理神话。软件项目经理，像所有领域的经理一样，承担着维持预算、保证进度和提高质量的压力。就像溺水人抓住稻草一样，软件经理经常依赖软件神话中的信条，只要它能够减轻以上的压力，即使是暂时性的。

⁶ 读者若有兴趣可阅读[LEH97a]以了解关于软件演化的详细讨论。

⁷ E类型系统指的是在现实世界计算环境下实现的软件，因此需要不断演化。

神话：我们已经有了一本写满软件开发标准和规程的宝典。它无所不包，囊括了我们可能问到的任何问题。

事实：这本宝典也许的确已经存在，但它是否在实际中采用了？我们的软件人员是否了解到它的存在呢？它是否反映了软件工程的现状？是否全面？是否可以适应不同的应用环境？它是否在缩短交付时间的同时还关注保证产品的质量？在很多情况下，问题的答案是否定的。

神话：如果我们未能按时完成计划，我们可以通过增加程序员人数而赶上进度（即所谓的蒙古游牧概念）。

事实：软件开发并不是像机器制造那样的机械过程。Brooks [BRO75]曾说过：“在软件工程中，为赶进度而增加人手，只能使进度更加延误。”初一看来，这种说法似乎与直觉不符。
45 然而，当新人加入到一个软件项目中后，原有的开发人员必须要牺牲本来的开发时间对后来者进行培训。人员的增加应该以有计划、有序的方式进行。

神话：如果我将一个软件外包给另一家公司，则我可以完全放手不管。

事实：如果一个组织对软件项目的内部组织和控制毫无了解，那无一例外地将在外包项目中遇到困难。



在项目开始之前，尽可能努力了解工作内容。也许难以明确所有细节，但你了解得越多，所面临的风险就越低。

用户神话。软件产品的顾客也许是隔壁的某个人，楼下的某个科技公司，市场/销售部门或者签定软件合同的某个公司。多数情况下，顾客相信所谓的软件神话，因为项目经理和工作人员没有纠正他们的错误信息。神话导致顾客错误的期望，最终形成对开发者的不满。

神话：有了对项目目标的大概了解，便足以开始编写程序，我们可以在之后的项目开发过程中逐步了解细节。

事实：虽然通常很难得到综合全面且固定不变的需求描述，但是对项目目标模糊不清的描述将为项目实施带来灾难。清晰的需求描述（经常是逐步变得清晰）要求顾客和开发人员之间不断保持有效的沟通。

神话：虽然项目需求不断变更，但是因为软件是弹性的，因此可以很容易地适应变更。

事实：软件需求的确在随时变更，但随变更引入的时机不同，变更所造成的影响也不同。如果需求变更提出得较早（比如在设计或者代码开发之前），则影响较小⁸；随着时间的后移，变更的代价也迅速增加，因为资源已经被分配，设计框架已经建立，而变更可能会引起剧变，从而需要添加额外的资源或者修改主要设计架构。



每当你认为没有时间采用软件工
程方法时，就再
问问自己，是否
有时间重做整个
软件？

从业者神话。在50多年的编程文化的滋养下，软件开发人员依然深信着各种神话。在软件业发展早期，编程被视为一种艺术。旧有的方式和态度根深蒂固。

神话：当我们完成程序并将其交付使用之后，我们的任务就完成了。

事实：有人曾说过，对于编程来说，开始得越早，耗费的时间就越长。

业界的一些数据显示，60%~80%的工作耗费在软件首次交付顾客使用之后。

神话：直到程序开始运行，才能评估其质量。

事实：最有效的软件质量保证机制之一——正式技术评审，可以从项目启动就开始实行。软件评审（参见第26章）作为“质量过滤器”，已经证明可以比软件测试更能有效地发现多种

⁸ 许多软件工程师采纳了“敏捷”（agile）开发方法，通过增量的方式逐步纳入变更，以便控制变更的影响范围和成本。本书第4章讨论敏捷方法。

类型的软件错误。

神话：对于一个成功的软件项目，可执行程序是惟一可交付的成果。

事实：软件配置包括很多内容，可执行程序只是其中之一。文档是软件工程的基础，更重要的是，为软件支持提供了指导。

神话：软件工程将导致我们产生大量无用文档，并因此降低工作效率。

事实：软件工程并非以创建文档为目的，而是为了保证软件开发的质量。好的质量可以减少返工，从而缩短开发时间。

很多软件专业人员已经认识到软件神话的谬误。然而遗憾的是，即使事实证明需要采用更好的方法，习惯性的态度和方法导致了错误的管理和技术行为。对于软件开发真实情况的正确理解，是系统阐述如何使用软件工程方法解决实际问题的第一步。

1.6 这一切都是如何开始的

每个软件工程项目都来自业务需求——对现有应用程序的纠错，改变原有系统以适应新的商业环境，扩展现有应用程序功能和特性，或者开发某种新的产品、服务或系统。

在软件工程项目的初期，业务需求通常是在简短的谈话过程中非正式地表达出来。以下的一段谈话就是一个典型的例子。

除了一带而过地涉及软件，这段谈话中几乎没有提及软件开发项目。然而，软件将是SafeHome产品线成败的关键。只有SafeHome软件成功，该产品才能成功。只有嵌入其中的软件产品满足顾客的需求（尽管还未明确说明），产品才能被市场所接受。我们将在后面的几章中继续讨论SafeHome中软件工程的话题。

47

SAFEHOME

如何开始一个软件项目

[场景]CPI公司的会议室里。CPI是一个虚构的为家庭和贸易应用生产消费产品的公司。

[人物]Mal Golden，产品开发部高级主管；Lisa Perez，营销经理；Lee Warren，工程经理；Joe Camalleri，业务发展部执行副总裁。

[对话]

Joe: Lee，我听说你们那帮家伙正在开发一个什么通用的无线盒？

Lee: 哦，是的，那是一个很棒的产品，只有火柴盒大小。我们可以把它放在各种感应器上，比如数码相机里，总之任何东西里。采用802.11b无线网络协议，我们可以通过无线连接获得它的输出。我们的任务是它可以带来全新的一代产品。

Joe: Mal，你觉得怎么样呢？

Mal: 我当然同意。事实上，随着这一年来销售业绩的趋缓，我们需要一些新的产品。Lisa和我已经做了一点儿市场调查，我们都认为该系列产品具有很大的潜力。

Joe: 多大，底线是多少？

Mal（避免直接承诺）：Lisa，和他谈谈我们的想法。

Lisa: 这是新一代的家庭管理产品，我们称之为“SafeHome”。产品采用无线接口，

⁹ SafeHome项目将作为一个案例贯穿本书，以便说明项目组在开发软件产品过程中的内部工作方式。公司、项目和人员都是虚构的，但场景和问题是真实的。

给家庭和小型商务使用者提供一个由电脑控制的系统——家庭安全、监督、应用和设备控制。你可以在回家的路上关闭家里的空调，或者如此这类的应用。

Lee (插话)：工程部已经做了相关的技术可行性研究，这个项目可行且制造成本不高。硬件都是现成的，不过软件方面是个问题，但也不是我们不能做的。

Joe：有意思！我想知道底线。

Mal：在美国，60%的家庭拥有电脑。如果我们定价合适，这将成为一个十分成功的产品。到目前为止，只有我们拥有这一无线控制盒的专利，我们将在这个方向上保持两年的领先地位，在第二年的收入大约是3千万到4千万。

Joe (微笑)：我很感兴趣，我们继续讨论一下。

1.7 小结

软件已经成为以计算机为基础的系统和产品中的关键部分，并且成为世界舞台上最重要的技术之一。在过去的50年里，软件已经从解决问题和信息分析的专用工具发展为独立的产业。然而，如何在有限的时间内，利用有限的资金开发高质量的软件仍然是我们所面临的难题。软件——程序、数据和文档，覆盖了科技和应用的很多领域。然而30多年来，软件发展的基本规律保持不变。软件工程的目的就是为开发高质量的软件产品提供一个工程框架。

48

参考文献

- [BRO75] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1975.
- [DAC03] Daconta, M., L. Obrst, and K. Smith, *The Semantic Web*, Wiley, 2003.
- [DAY99] Dayani-Fard, H., et al., "Legacy Software Systems: Issues, Progress, and Challenges," IBM Technical Report: TR-74.165-k, April 1999, available at <http://www.cas.ibm.com/toronto/publications/TR-74.165/k/legacy.html>.
- [DEM95] DeMarco, T., *Why Does Software Cost So Much?*, Dorset House, 1995.
- [FEI83] Feigenbaum, E. A., and P. McCorduck, *The Fifth Generation*, Addison-Wesley, 1983.
- [HAM93] Hammer, M., and J. Champy, *Reengineering the Corporation*, HarperCollins Publishers, 1993.
- [JOH01] Johnson, S., *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*, Scribner, 2001.
- [KAR99] Karlson, E., and J. Kolber, *A Basic Introduction to Y2K: How the Year 2000 Computer Crisis Affects YOU*, Next Era Publications, Inc., 1999.
- [LEH97a] Lehman, M., and L. Belady, *Program Evolution: Processes of Software Change*, Academic Press, 1997.
- [LEH97b] Lehman, M., et al., "Metrics and Laws of Software Evolution—The Nineties View," *Proceedings of the 4th International Software Metrics Symposium (METRICS '97)*, IEEE, 1997, can be downloaded from <http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf>.
- [LEV95] Levy, S., "The Luddites Are Back," *Newsweek*, July 12, 1995, p. 55.
- [LIP02] Lippman, A., "Round 2.0," *Context Magazine*, August 2002, <http://www.contextmag.com/>.
- [LIU98] Liu, K., et al., "Report on the First SEBPC Workshop on Legacy Systems," Durham University, February, 1998, available at <http://www.dur.ac.uk/CSM/SABA/legacy-wksp1/report.html>.
- [OSB79] Osborne, A., *Running Wild—The Next Industrial Revolution*, Osborne/McGraw-Hill, 1979.
- [NAI82] Naisbitt, J., *Megatrends*, Warner Books, 1982.
- [STO89] Stoll, C., *The Cuckoo's Egg*, Doubleday, 1989.
- [TOF80] Toffler, A., *The Third Wave*, Morrow Publishers, 1980.

- [TOF90] Toffler, A., *Powershift*, Bantam Publishers, 1990.
- [WIL02] Williams, S., "A Unified Theory of Software Evolution," *salon.com*, 2002, <http://www.salon.com/tech/feature/2002/04/08/lehman/index.html>.
- [WOL02] Wolfram, S., *A New Kind of Science*, Wolfram Media, Inc, 2002.
- [YOU92] Yourdon, E., *The Decline and Fall of the American Programmer*, Yourdon Press, 1992.
- [YOU96] Yourdon, E., *The Rise and Resurrection of the American Programmer*, Yourdon Press, 1996.
- [YOU98a] Yourdon, E., and J. Yourdon, *Time Bomb 2000*, Prentice-Hall, 1998.
- [YOU98b] Yourdon, E., *Death March Projects*, Prentice-Hall, 1999.
- [YOU02] Yourdon, E., *Byte Wars*, Prentice-Hall, 2002.

习题与思考题

- 1.1 本章1.2节中阐述的关于软件的定义是否可以应用于网站？如果是，请说出网站和传统软件的细微差别。
- 1.2 针对1.1节提出的问题，请给出你的答案，并与同学讨论。
- 1.3 举例说明软件对社会的影响（包括正面影响和负面影响）。仔细阅读1.1节中提到的1990年之前的关于预测软件发展的各种论述，针对其中某一论述，分析作者观点正确与否。
- 1.4 举出至少5个例子来说明“意外效应法则”在计算机软件方面的应用。49
- 1.5 分析1.3节中提出的新挑战（或者本书出版后的其他新问题），任选一种软件技术，写一篇论文（一页左右）来介绍该技术及其给软件工程师所带来的挑战。
- 1.6 用自己的语言描述“保证通晓性规律”（参见1.4.2节）。
- 1.7 在交付最终用户之前，或者第1个版本投入使用之后，许多应用程序都会有频繁的变更。为防止变更引起软件失效，请提出一些有效的解决措施。
- 1.8 仔细阅读Internet新闻组comp.risks，并总结最近讨论的公众所面临的风险。可选阅读材料：“Software Engineering Notes”，ACM出版。
- 1.9 思考1.3节中提到的7个软件分类。请问能否将一个方法应用于所有分类？并就你的答案加以解释。
- 1.10 随着软件的普及，由于程序错误所带来的公众风险已经成为一个愈加重要的问题。设想一个由于软件错误的危害而引起世界末日的真实场景（要涉及经济或者人类社会）。
- 1.11 用自己的语言描述“质量衰减规律”（参见1.4.2节）。
- 1.12 用自己的语言描述“组织稳定性守恒规律”（参见1.4.2节）。

推荐读物与阅读信息¹⁰

在数千本关于软件工程的书中，大多数讨论的是编程语言和软件应用，很少有涉及软件本身的。Pressman和Herron（《Software Shock》，Dorset House, 1991）最早讨论了软件和开发方法的问题。Negroponte的畅销书（《Being Digital》，Alfred A. Knopf, Inc., 1995）提供了关于信息论和其在21世纪发展和影响的观点。DeMarco[DEM95]就软件和开发过程发表了一系列惊人且见解独到的论文。Norman（《The Invisible Computer》，MIT Press, 1998）和

¹⁰ 在每章小结之后，“推荐读物与阅读信息”小节简单介绍了本章相关资料，以便于读者扩展阅读和深入理解本章内容。针对本书，我们已经建立了网站<http://www.mhhe.com/pressman>。网站涉及软件工程的很多主题，并逐章列出了相关的软件工程网站资料作为本书的补充，并给出了每一本书在Amazon.com的链接。

Bergman (《Information Appliances and Beyond》, Academic Press/Morgan Kaufmann, 2000) 在他们的著作中称, 信息应用和普适计算通过新的Internet基础设施将工业界的所有人和应用连接起来, 而个人电脑的大范围的影响将逐渐衰弱。

Minasi在著作中 (《The Software Conspiracy: Why Software Companies Put Out Faulty Products, How They Can Hurt You, and What You Can Do》, McGraw-Hill, 2000) 认为, 现在由于软件缺陷引起的灾难将被消除, 并提出了解决的方法。Compaine (《Digital Divide: Facing a Crisis or Creating a Myth》, MIT Press, 2001) 认为, 在新世纪的头十年里, 是否能够访问到信息资源 (如Web) 的区分将越来越小。

因特网上有很多有关软件和管理问题的信息资源。访问网站<http://www.mhhe.com/pressman> 可以获得许多最新的软件工程资源列表。

第一部分

软件过程

本部分将介绍软件过程，它定义了软件工程各种实践活动的组成和结构。在本部分的各章中将涉及以下问题：

- 什么是软件过程？
- 软件过程中，有哪些共同的、基本的活动？
- 如何建立过程模型？什么是过程模式？
- 什么是惯例过程模型？有哪些优缺点？
- 增量模型的哪些特征使得它们能适应现代的软件项目需求？
- 什么是统一过程？
- 为什么现代软件工程强调“灵活性”？
- 什么是敏捷软件开发？它与传统的过程模型有什么区别？

了解上述问题之后，就可以对软件工程实践的应用环境有更清楚的认识。

第2章 过程综述

要点浏览

概念：当开发产品或构建系统时，遵循一系列可预测的步骤（即路线图）是非常重要的，它有助于及时交付高质量的产品。软件开发中所遵循的路线图就称为“软件过程”。

人员：软件工程师及其管理人员根据需要调整开发过程，并遵循该过程。除此之外，软件的需求方也需要参与过程的定义、建立和测试。

重要性：软件过程提高了软件工程活动的稳定性、可控性和有组织性，如果没有过程约束，软件活动将失控并变得混乱。但是，现代软件工程方法必须是

“灵活”的。也就是要求软件工程活动、控制以及文档的编制适合于项目团队和要开发的产品。

步骤：具体来讲，采用的过程依赖于所构造软件的特点。飞机电子系统的软件与网站的建设可能是两个截然不同的构建过程。

工作产品：从软件工程师的观点来看，工作产品就是过程定义的一系列活动和任务的结果，即程序、文档和数据。

质量保证措施：有大量的软件过程评估机制，开发机构可以评估其软件过程的“成熟度”。然而，评价所采用过程的有效性，最好的指标还是所构建产品的质量、适时性和长期生存能力。

关键概念
CMMI
ISO 9001: 2000
过程评估
过程框架
过程模式
过程技术
PSP
任务集
TSP
普适性活动

Howard Baetjer, Jr.[BAE98]曾著书从经济学家的角度分析软件和软件工程，书中对软件过程评述如下：

软件同其他资产一样，是知识的具体体现，而知识最初都是以分散的、不明确的、隐蔽的且不完整的形式广泛存在的，因此，软件开发是一个社会学习的过程。软件过程是一个对话，在对话中，软件所必需的知识被收集在一起并在软件中实现。过程提供了用户与设计人员之间、用户与不断演化的工具之间以及设计人员与不断演化的工具（技术）之间的交互途径。软件开发是一个迭代的过程，在这个过程中，演化的工具本身作为沟通的媒介，每新一轮对话都可以从参与的人员中获得更有用的知识。

构建计算机软件确实是一个迭代学习的过程，其输出——即Baetjer所称的“软件资本”，是在过程执行中所收集、精练和组织的知识。

但从技术的角度如何确切地定义软件过程呢？本书中将软件过程定义为一个为建造高质量软件所需要完成的任务的框架。过程与软件工程同义吗？答案是“是”，也可以是“否”。软件过程定义了软件开发中采用的方法，但软件工程还包含该过程中应用的技术——技术方法和自动化工具。

更重要的是，软件工程是由有创造力、有知识的人完成的，他们根据产品构建的需要和市场需求，选取成熟的软件过程。

2.1 软件工程：一种层次化技术

尽管有很多作者都各自给出了软件工程的定义，但Fritz Bauer[NAU69]在开创性软件工程主题会议上给出的定义仍是进一步开展讨论的基础：

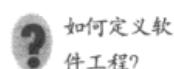
（软件工程是）建立和使用一套合理的工程原则，从而经济地获得可靠的、可以在实际机器上高效运行的软件。

几乎每一个读者都忍不住想在这个定义上增加点什么。它没有提到软件质量的技术层面，它也没有直接谈到用户满意度或按时交付产品的要求，它忽略了测量和度量的重要性，它也没有阐明有效的软件过程的重要性。但Bauer的定义给我们提供了一个基线。什么是可以应用到计算机软件开发中的“合理工程原则”？我们如何“经济地”建立“可靠的”软件？如何才能构建出不是在一个而是在多个不同的“实际机器”上“高效运行”的程序呢？这些都是对软件工程师提出进一步挑战的问题。

“工程不仅仅是一个学科或一个知识体系，它是一个动词，一个行为动词，一个解决问题的方法。”

——Scott Whitmire

IEEE[IEE93]给出了一个更全面的软件工程的定义：



软件工程是：(1) 将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件。(2) 在(1)中所述方法的研究。

然而，不同的软件程序员队伍对于“系统化的、规范的、可量化的”都有不同的定义。我们需要规范，也需要可适应性和灵活性。

软件工程是一种层次化的技术（如图2-1所示）。任何工程方法（包括软件工程）必须以组织对质量的承诺为基础。全面的质量管理、六西格玛和类似的理念培养了不断的过程改进文化，正是这种文化最终引导了人们开发更有效的软件工程方法。软件工程的根基在于质量关注点（quality focus）。

53

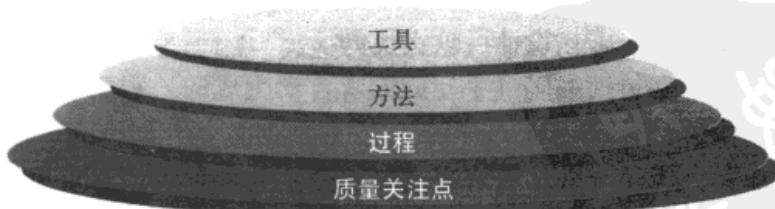


图2-1 软件工程层次图

KEY POINT
软件工程包括
过程、方法和
工具。

软件工程的基础是过程（process）层。软件过程将各个技术层次结合在一起，并实施合理地、及时地开发计算机软件。过程定义一个框架[PAU93]，为有效交付软件工程技术，这个框架必须建立。软件过程构成了软件项目管理控制的基础，并且建立了一个环境以便于技术方法的采用、工作产品（模型、文

档、数据、报告、表格等)的产生、里程碑的建立、质量的保证、正常变更的正确管理。

WebRef

《Cross Talk》

杂志提供了关于过程、方法和工具的许多实际的信息，网站地址是：www.stsc.hill.af.mil。

软件工程方法 (method) 为建造软件提供技术上的解决方法 (“如何做”)。方法覆盖面很广，包括沟通、需求分析、设计建模、编程、测试和支持。软件工程方法依赖于一组基本原则，这些原则涵盖了软件工程所有技术领域，包括建模和其他描述性技术等。

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

2.2 过程框架

过程框架 (process framework) 定义了若干小的框架活动 (framework activity)，为完整的软件开发过程建立了基础。这些框架活动可广泛应用于所有软件开发项目，无论这些项目的规模和复杂性如何。此外，过程框架还包含一些适用于各个软件过程的普适性活动 (umbrella activity)。

54

“过程定义了活动的时间、人员、工作内容和达到预期目标的途径。”

——Ivar Jacobson, Grady Booch, and James Rumbaugh

如图2-2所示，框架中每一个活动都是由一组软件工程动作 (software engineering action) 组成，每一个动作都包括一系列相互关联的任务并产生一个关键的工作产品 (例如，设计 (design) 是一种软件工程动作)。每一个工作任务 (work task) 都完成一部分动作所定义的工作。

55

下面介绍的通用过程框架 (generic process framework) 可适用于绝大多数的软件项目 (该框架是后续章节中过程模型描述的基础)：

沟通 这个框架活动包含了与客户 (和其他共利益者¹) 之间大量的交流和协作，还包括需求获取以及其他相关活动。

策划 指为后续的软件工程工作制定计划。它描述了需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度计划。

建模 它包括创建模型和设计两方面。创建模型有助于客户和开发人员更好地理解软件需求；设计可以实现需求。

构建 它包括编码 (手写的或者自动生成的) 和测试 (测试是为了发现编码中的错误)。

部署 软件 (全部或者完成的部分) 交付到用户，用户对其进行评测并给出反馈意见。

上述五个通用框架活动适用于小程序的开发、大型网络应用程序的建造以及基于计算机的大型复杂系统工程。不同的应用中软件过程的细节可能很不一样，但是框架活动都是一致的。

五个最基本的过程框架活动是什么？

¹ 共利益者 (stakeholder) 就是可在项目成功中分享利益的人，包括业务经理、最终用户、软件工程师、支持人员等。Rob Thomsett曾开玩笑说：“stakeholder就是掌握巨额精明投资 (stake) 的人……如不照看好你的stakeholder，投资将会结束。”

软件过程

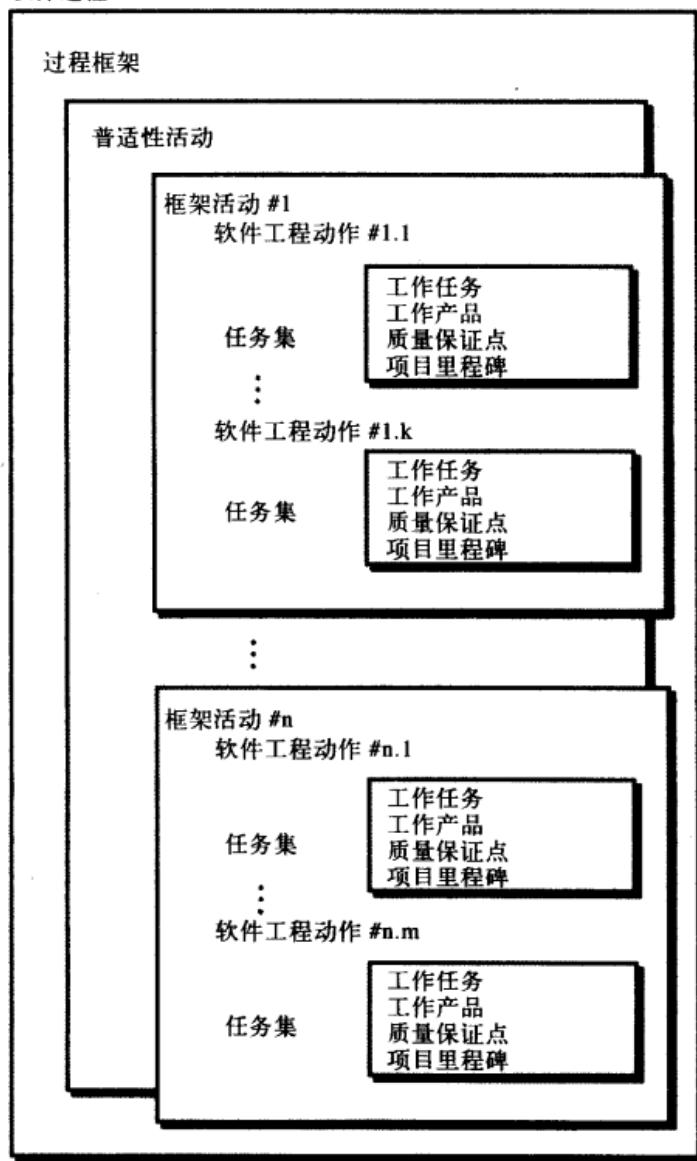


图2-2 软件过程框架

“爱因斯坦认为必然存在着一个对自然界简单的解释，因为上帝既不专制也不喜怒无常。然而软件工程师却无法抱侥幸心理，多数情况下，他必须面对毫无规律的复杂性。”

——Fred Brooks

举例来说，在通用的过程框架中，建模活动包括分析（analysis）和设计（design）两个动作。分析²包括一组工作任务（如需求获取、细化、协商、规格说明和确认），最终产生需求分析模型（和/或需求规格说明）。设计也包括一组工作任务（如数据设计、体系结构设计、接口设计和构件层设计等），最终产生设计模型（和/或设计规格说明）³。

² 软件分析在第7章和第8章详细讨论。

³ 需要明确的是，“建模”这一概念在软件维护的过程中有不同的解释。在某些情况下，分析和设计建模确实存在；但在其他一些维护情况中，建模也许是用于帮助理解遗留系统或是说明对软件的增加和修改。



普适性活动贯穿整个软件过程，主要关注于项目管理、跟踪和控制。



对软件过程的适应性调整是项目成功的关键。

过程模型之间有哪些不同之处？

正式技术评审 评估软件工程产品，尽量在错误传播到下一个动作或活动之前发现并清除错误。

测量 定义和收集过程、项目和产品的度量，以帮助团队在发布软件的时候满足客户要求。同时，测量还可与其他框架和普适性活动协同使用。

软件配置管理 管理整个软件过程中变更所带来的影响。

可复用管理 定义产品复用的标准（包括软件构件），并且建立构件复用机制。

工作产品的准备和生产 包括了创建产品所必需的活动，如建模、文档、日志、表格和列表等。

普适性活动应用于整个软件过程中，本书后面将详细讨论。

所有的过程模型都可以用图2-2所示的过程框架概括。采用任何软件过程模型都需明智地认识到：过程模型的适用性（如软件所需解决的问题、项目特点、开发队伍和组织文化等）是成功的关键。但不同的模型在以下几方面却区别很大：

- 活动和任务的总体流程，以及相互之间的关系。
- 在框架中的每一个活动中任务细化的程度。
- 对所需要提交的工作产品的定义。
- 质量保证活动应用的方式。
- 过程跟踪和控制活动应用的方式。
- 过程描述的详细和严谨程度。
- 客户和共利益者对项目参与的深度。
- 软件项目队伍所赋予的自主权。
- 队伍组织和角色明确程度。

“我认为食谱只是指导方法，一个聪明的厨师每次都会变化出不同的特色。”

——Madame Benoit

58

在过去的30年中，一些过程模型已经在软件工程领域广泛应用，这些模型强调对过程活动和任务的详细定义、识别和应用。应用这些“惯例过程模型”（prescriptive process model）的目的是提高软件质量、项目的可管理性以及对于交付时间和项目费用的可预测性，并对软件工程师构建系统所必需的工作提供指导。但遗憾的是，很多情况下，这些目的并没有达到。如果只是教条地应用这些过程模型，而没有根据实际情况加以调整，那么，在构建基于计算机的系统的过程中，它将增加官僚作风，并为开发人员和用户制造麻烦。

近年来，又提出了一些新的过程模型。这些模型强调项目的灵活性，并在一些基本原则⁴的指导下，提倡弱化软件过程中过于正式的要求（持这种观点的人认为这并不降低过程的有效性）。这些敏捷过程模型（agile process model）强调可操作性和可适应性，对某些类型的项目很适用，尤其是Web应用开发。

哪一种软件过程观点最好？这个问题在软件工程师中引起了广泛的争论，第4章将继续讨

敏捷过程的特点是什么？

⁴ 第4章中将介绍敏捷模型和指导原则。

论。不管怎样，重要的是认识到这两种过程观点虽然采用不同的方法，但都有一个共同的目标，即构建符合用户要求的高质量软件。

2.3 能力成熟度模型集成

美国卡内基·梅隆大学软件工程研究所（Software Engineering Institute, SEI）提出了一个全面的过程元模型，当软件开发组织达到不同的过程能力和成熟度水平时，该模型可用来预测其所开发的系统和软件工程能力。为达到这些能力，SEI认为组织都应建立与能力成熟度模型集成（Capability Maturity Model Integration, CMMI）指南相符合的过程模型（如图2-2所示）[CMM02]。

WebRef

有关CMMI的完整信息可参考
<http://www.sei.cmu.edu/cmmi/>

CMMI用以下两种不同的方式描述过程元模型：（1）作为一个连续（continuous）的模型；（2）作为一个阶段性（staged）的模型。如图2-3所示，连续的CMMI元模型从两个维度描述过程。每个过程域（如项目策划、需求管理）都根据特定的目标和实践要求，进行严格的评估，并根据能力水平评定为以下几级：

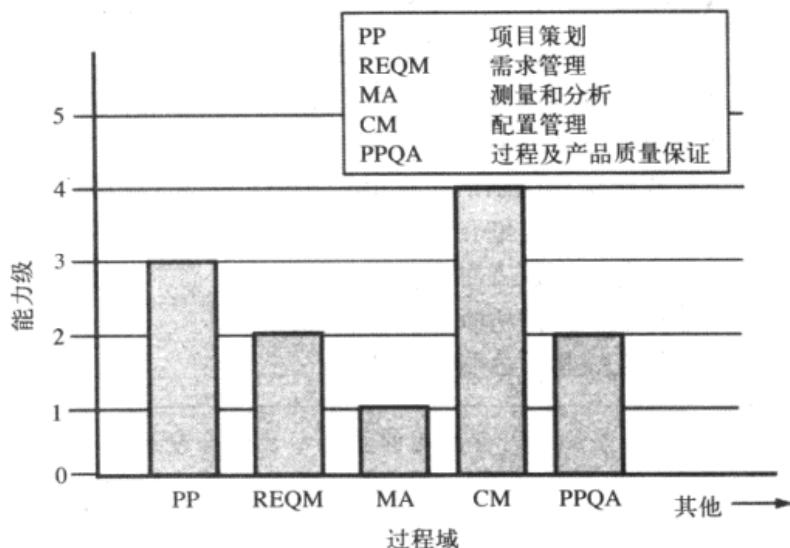


图2-3 CMMI过程能力剖面[PHI02]

第0级：不完全级（Incomplete）。过程域（如需求管理）没有实施，或者已经实施但未达到CMMI 1级成熟度所规定的所有目标。

第1级：已执行级（Performed）。CMMI中定义的所有过程域的特定目标都已经实现。产生规定的工作产品所必需的工作任务都已经执行。

第2级：已管理级（Managed）。所有第1级规定的要求都已经达到。另外，所有与过程域相关的工作都符合组织的规程；工作人员都有足够的资源完成工作；共利益者都积极地参与到要求的过程域；所有工作任务和工作产品都被“监督、控制和评审；并评估是否与过程描述相一致”[CMM02]。

第3级：已定义级（Defined）。所有第2级规定的要求都已经达到。另外，根据组织剪裁准则，对其标准过程进行了剪裁，剪裁过的进程对组织的过程资产增添了新的内容，如工作产品、测量和其他过程改进信息等[CMM02]。

第4级：已定量管理级（Quantitatively Managed）。所有第3级规定的要求都已经达到。另外，通过采用测量和定量的评估等手段，对过程域进行控制和不断改进。“已经建立起来对质量和过程性能的定量指标，并作为过程管理的标准” [CMM02]。

第5级：优化级（Optimized）。所有第4级规定的要求都已经达到。另外，“采用定量（统计）的方法调整和优化过程域，以满足用户不断变更的需求，并持续地提高过程域的有效性 [CMM02]。”

“大部分的软件危机都是自己造成的，正如一个CIO所说：我宁愿错误也不愿延期，因为错误总是有办法修复的。”

——Mark Paulk



每个软件开发机构都应努力达到CMMI的目标。但是，在某些情况下，实现模型的所有方面可能会执行过度。

CMMI定义了每一个过程域的“特定目标”，以及为达到该目标所需的“特定实践”。特定目标明确了如果该过程域的所有活动都有效执行的话，软件项目应具备的特点。特定实践将目标细化成一组过程相关的活动。

60

例如，CMMI定义了“项目管理”的8个过程域⁵，项目策划就是其一，其特定目标（SG）和特定实践（SP）的定义如下[CMM02]：

SG 1 建立估计值

- SP 1.1-1 估计项目的范围
- SP 1.2-1 建立对工作产品和任务属性的估计值
- SP 1.3-1 定义项目的生存周期
- SP 1.4-1 确定工作量和成本估计值

SG 2 制定项目计划

- SP 2.1-1 建立预算和进度计划表
- SP 2.2-1 识别项目风险
- SP 2.3-1 制定数据管理计划
- SP 2.4-1 制定项目资源计划
- SP 2.5-1 制定所需知识技能计划
- SP 2.6-1 制定共利益者参与计划
- SP 2.7-1 建立项目计划

SG 3 获得对计划的承诺

- SP 3.1-1 评审影响项目的计划
- SP 3.2-1 协调工作和资源等级
- SP 3.3-1 获得对计划的承诺

除了每个过程域特定的目标和实践之外，CMMI还规定了对每个过程域通用的5个目标和相关实践。每一个通用的目标与5级能力等级的其中一级相对应。因此，为达到某一级能力等级，与该级相关的通用目标和实践也必须达到。例如，对于项目策划过程域，其通用目标（GG）和通用实践（GP）如下所述 [CMM02]：

⁵ “项目管理”的其他过程域有：过程监控、供方协议管理、IPPD集成项目管理、风险管理、集成团队建立、集成供方管理以及定量项目管理。

GG1 达到特定目标

GP1.1 完成基本实践

GG2 将已管理过程制度化

GP2.1 制定组织方针

GP2.2 策划过程

61

GP2.3 提供资源

GP2.4 职责分配

GP2.5 人员培训

GP2.6 管理配置

GP2.7 识别并吸纳相关的共利益者

GP2.8 监督并控制过程执行

GP2.9 客观评价遵循情况

GP2.10 与高层管理者一起评审状态

GG3 将已定义过程制度化

GP3.1 建立已定义过程

GP3.2 搜集改进信息

GG4 将已定量管理过程制度化

GP4.1 制定过程的量化目标

GP4.2 稳定子过程的性能

GG5 将优化过程制度化

GP5.1 保证过程的持续改进

GP5.2 纠正造成问题的根本原因

分阶段的CMMI和连续的模型定义了同样的过程域、目标和实践。主要区别在于，阶段性模型定义了5个成熟度等级，而不是5个能力等级。为达到成熟度等级，必须达到与那些过程域相关的特定目标和特定实践。成熟度等级与过程域的关系如图2-4所示。

分级的CMMI成熟度连续模型中定义了相同的过程域，目标和实践定义。模型的最基本区别是阶段模型确定了5个成熟度，而不是5个能力层。为了达到一定的成熟度，必须达到其对应的关键目标和一系列过程域的关键实践。成熟度和过程域之间的关系如图2-4所示：

INFO**CMMI，我们该不该使用？**

CMMI是一个过程元模型，它有700多页，定义了如果建立完整的软件过程，软件组织所应具备的过程特征。十多年来争论的一个问题焦点是：CMMI是否执行过度了？如同我们日常生活中（也是软件开发中）的多数问题一样，答案当然不是一个简单的“是”或者“否”。

CMMI的宗旨应该是普遍接纳的。为避免过于简化问题，CMMI认为软件开发过程必须严肃对待——必须详细计划、一致控制、准确跟踪、专业化地执行。必须以项目共利益者的要求、软件工程师的技能和最终产品的质量为核心。任何人都不应该质疑上述观点。

如果有几十人甚至几百人参与，需要在很多月甚至几年的时间内，构建大型复杂的系统，CMMI的详细需求应慎重考虑。CMMI也许只有在下述情况下才适用，即组织文化与

标准的过程模型相一致并且管理者承诺力求过程成功。但是，在另外一些情况下，CMMI过于复杂庞大，组织难以成功地消化吸收。这是否就意味着CMMI不好，或者过于官僚，过时了呢？当然不是。这说明，对某个组织文化适用的并不一定对另一个组织适用。

CMMI是软件工程的伟大成就。在构建计算机软件时到底该采用哪些活动和动作，CMMI对此提供了全面的讨论。尽管软件组织不会采纳CMMI的所有细节，但是每个软件开发队伍都应该接受其主旨，并从其对软件工程过程和实践的讨论中提高认识。

成熟度等级	关注焦点	过程域
优化级	持续的过程改进	组织创新和部署 原因分析和决定
已定量管理级	定量管理	组织过程性能 定量项目管理
已定义级	过程标准化	需求开发 技术解决方案 产品集成 验证 确认 组织过程焦点 组织过程定义 组织培训 集成项目管理 集成供方管理 风险管理 决策分析和决定 组织的集成环境 集成团队建立
已管理级	基本的项目管理	需求管理 项目策划 项目监督和控制 供方合同管理 测量和分析 过程和产品质量保证 配置管理
已执行级		

图2-4 达到成熟度等级所需的过程域

2.4 过程模式

什么是过程模式？

软件过程可以定义为一系列模式的组合，这些模式定义了一系列的软件开发中所需的活动、动作、工作任务、工作产品及其相关的行为[AMB98]。通俗地讲，过程模式提供了一个模板——一种描述软件过程中重要特征的一致性方法。通过模式组合，软件团队可以定义能最好满足项目需求的开发过程。

“模式的重复与软件模块的重复完全不同。事实上，不同的模块是独特的，而模式却是相同的。”

—Christopher Alexander



模式模板提供了
描述模式的一致
性方法。

我们可以在不同抽象层次上定义模式⁶。在某些情况下，模式可以描述一个完整的过程（例如原型开发），在有些情况下，模式可以描述一个重要的框架活动（譬如计划）或者框架活动中的一项具体任务（譬如项目估算）。

Ambler[AMB98]提出了如下所示的描述过程模式的模板：

模式名称。模式名称应能清楚地表述该模式在软件过程中的功能（例如，客户沟通）。

目的。简洁地描述模式的目的。例如，客户沟通的目的是“与客户建立合作关系，共同致力于确定项目范围、业务需求和其他项目约束”。如果需要，还可以通过附加的说明文档和适当的图表对目的进行补充描述。

类型。定义模式类型。Ambler[AMB98]提出了三种类型：

- **任务模式 (task pattern)**，定义过程中软件工程动作或工作任务，以保证成功地执行软件工程实践（例如，需求获取是一个任务模式）。
- **步骤模式 (stage pattern)**，定义过程的框架活动。由于框架活动包括很多工作任务，步骤模式包括与步骤（框架活动）有关的许多任务模式。例如，沟通可能作为一个步骤模式。该步骤模式可能包括需求获取等任务模式。
- **阶段模式 (phase pattern)**，定义在过程中发生的框架活动序列，即使这些活动流本质上是迭代的过程。例如，螺旋模型和原型开发⁷就可能是两种阶段模式。

启动条件。它描述的是模式应用的前提条件。在应用模式之前需要明确：(1) 在此之前，整个开发组织或是开发团队内已经有哪些活动？(2) 过程的进入状态是什么？(3) 已经有哪些软件工程信息或是项目信息？

例如，策划模式（阶段模式）需要的前提条件有：(1) 客户和软件工程师已经建立了合作的交流机制；(2) 客户沟通模式中所定义的一组任务模式已经成功完成；(3) 项目范围、基本业务需求和项目限制条件已经确定。

问题。描述模式将要解决的问题。例如，客户沟通模式需要解决的问题可能描述如下：由于没有建立起信息提取的有效方式，缺乏实用的记录机制，以及没有进行有效的评审等原因，开发人员和用户之间的交流往往不充分。

解决办法。描述模式的实现。这部分主要讨论随着模式的启动，过程的初始状态（模式应用之前）是如何发生改变的。解决方法也描述了随着模式的成功执行，模式启动之前所获得的软件工程信息和项目信息是如何发生改变的。

结束条件。描述模式成功执行之后的结果。模式结束时需要明确：(1) 必须有哪些组织或开发团队相关的活动？(2) 过程的结束状态如何？(3) 产生了哪些软件工程信息或是项目信息？

相关模式。以层次或其他图的方式列举与该模式相关的其他过程模式。例如沟通（步骤模式）包括了一组任务模式：项目团队组织、合作指导原则定义、范围分解、需求获取、约束描述、以及小规格描述/模型创建等。

已知应用实例。介绍该模式应用的具体实例。例如，沟通在每一个软件项目的开始都是必需的，建议在整个软件项目过程中采用；另外，沟通在部署活动中也是必需的。

⁶ 模式的概念广泛应用于软件工程的活动中。第7, 9, 10, 12和14章将分别讨论分析模式、设计模式和测试模式。本书第四部分将讨论项目管理活动中的模式和反模式。

⁷ 第3章将讨论这些阶段模式。

WebRef

可在下面网站
查看过程模式
的全面资源：
www.ambisoft.com/processPatternsPage.html。

过程模式提供了一种有效的机制来描述各种软件过程。模式使得软件工程组织能够从高层抽象（阶段模式）开始，开发层次化的过程描述。高层抽象描述又进一步细化为一系列步骤模式以描述框架活动，然后每一个步骤模式又进一步逐层细化更为详细的任务模式。过程模式一旦建立起来，就可以在过程变体的定义中复用——即软件开发队伍可以将模式作为过程模型的构建模块，定制特定的过程模型。

65

INFO**一个过程模式的例子**

当利益共同体对工作成果有大致的想法，但对具体的软件需求不确认时，下述简化的过程模式描述了可采用的方法。

模式名称。原型开发。

目的。构造一个便于共利益者反复评估的模型（原型），以便识别和确定软件需求。

类型。阶段模式。

启动条件。在模式启动之前必须满足以下四个条件（1）确定共利益者；（2）已经建立起共利益者和软件开发队伍之间的沟通方式；（3）共利益者确定了需要解决的主要问题；（4）对项目范围、基本业务需求和项目约束条件有了初步了解。

问题。需求模糊或者不存在，但都清楚地认识到项目存在问题，且该问题必须通过软件解决。共利益者不确认他们想要什么，即他们无法详细描述软件需求。

解决办法。描述了原型开发过程，详见第3章。

结束条件。共利益者已经开发了一个软件原型，识别了基本的需求（例如，交互模式、计算特征、处理功能等）。随后，可能有两种结果：（1）原型系统可以通过一系列的增量开发成为软件产品；（2）原型系统被抛弃，采用其他过程模式建立了产品软件。

相关模式。以下模式与该模式相关：客户沟通，迭代设计，迭代开发，客户评估，需求抽取。

已知应用实例。当需求不确定时，推荐原型开发方法。

2.5 过程评估

KEY POINT

以改进为目标，
评估力求理解软
件过程的当前状
态。

评估软件过
程有哪些形
式化的技术？

软件过程并不能保证软件按期交付，也不能保证软件满足客户要求，或是具备了长期质量保证所需的技术特点（第26章）。软件过程模型必须与切实的软件工程实践相结合（本书第二部分）。另外，对过程本身也要进行评估，以确认满足了成功软件工程所必需的基本过程标准要求⁸。图2-5显示了软件过程和过程评估及改进方法之间的关系。在过去的几十年中，提出了很多种不同的软件过程评估方法：

用于过程改进的标准CMMI评估方法（Standard CMMI Assessment Method for Process Improvement, SCAMPI）：提供了五步的过程评估模型，包括启动（initiating）、诊断（diagnosing）、建立（establishing）、执行（acting）

⁸ SEI的CMMI[CMM02]丰富翔实地介绍了软件过程的基本特征，以及过程成功的标准。

66

和学习 (learning)。SCAMPI方法采用SEI的CMMI（参见2.3节）作为评估的依据 [SEI00]。

用于组织内部过程改进的CMM评估 (CMM-Based Appraisal for Internal Process Improvement,

CBA IPI)：采用SEI的CMM（CMMI前身，参看2.3节）作为评估的依据[DUN01]，提供了一种诊断方法，用以分析软件或软件开发组织的相对成熟度。

SPICE (ISO/IEC 15504)：该标准定义了软件过程评估的一系列要求。该标准的目的是帮助软件开发组织建立客观的评价体系，以评估定义的软件过程的有效性[SPI99]。

软件ISO 9001:2000：这是一个通用标准，任何开发组织如果希望提高所提供的产品、系统或服务的整体质量，都可采用这个标准。因此，该标准可直接应用于软件组织和公司。

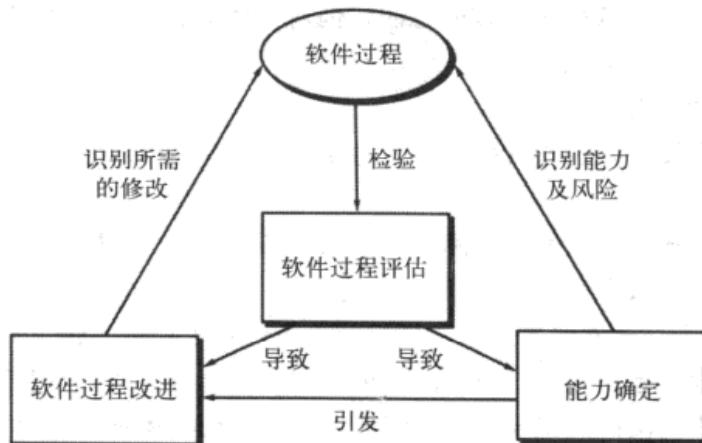


图2-5 软件过程评估与能力确定和过程改进

考虑到ISO 9001:2000在国际范围内广泛采用，我们将在下面简单介绍。

“软件组织在将其从整个项目中所获得的经验转化成资产的过程中存在重大缺陷。”

——NASA

WebRef

对于ISO 9001:2000的一个很好的总结可参考：
<http://praxiom.com/iso-9001.htm>。

67

国际标准化组织 (ISO) 制定了ISO 9001:2000标准[ISO00]。为了提供更高质量的产品而提高用户满意度，该标准定义了质量管理体系的要求（第26章）⁹。

ISO 9001:2000所建议的基本策略表述如下[ISO01]：

ISO 9001:2000强调了组织以下活动的重要性：识别、执行、管理和持续地提高质量管理体系所必需的过程的有效性，并管理上述过程之间的交互以达到组织的目标……

ISO 9001:2000采用“计划—实施—检查—行动”(plan-do-check-act)循环，将其应用于软件项目的质量管理环节。对于软件项目而言，“计划”是为实现高质量的软件产品并最终提高用户满意度，建立必需的过程目标、活动和任务。“实施”是执行软件过程，包括框架活动和普适性活动。“检查”监督并测量过程，以保证所有为质量管理所建立的需求均已实现。“行动”启动软件过程改进活动，并持续地改进过程。

⁹ 软件质量保证 (SQA) 是质量管理中的重要组成部分，已定义为可应用于整个过程框架的普适性活动。第26章将详细讨论。

如果对ISO 9001:2000感兴趣，详细信息可参阅ISO标准或[CIA01], [KET01], [MON01]。

2.6 个人过程模型和团队过程模型

好的软件过程需要贴近于过程的执行人员。如果一个软件过程模型是为社团或是软件组织制定的，那么，只有对其进行充分地改进，满足真正执行软件工程项目的要求，该模型才能有效。理想情况下，每一个软件工程师都会建立最适合他的过程，并同时能够与开发队伍及整个组织的要求相吻合。换句话说，开发团队将制定其自己的过程，同时满足个人小范围的要求和企业大的要求。Watts Humphrey[HUM97] [HUM00]认为，有可能建立“个人软件过程”和/或“团队软件过程”。虽然二者都需要艰苦努力、培训和协调，但都是可以做到的¹⁰。

“成功者不过是养成了成功人士做事的习惯。”

——Dexter Yager

2.6.1 个人软件过程

WebRef

可在下面的网站找到PSP大量的相关资料：
www.ipd.uka.de/PSP/

68

每个开发人员都采用某种过程来开发计算机软件。这种过程也许是混乱的，也许是特定的，可能每天都会改变，可能不够高效、不够有效甚至不成功，但不管怎样，过程都是存在的。Watts Humphrey [HUM97]建议，为了改变无效的个人过程，开发人员必须经过4个阶段，每个阶段都需要培训和认真操作。个人软件过程（personal software process, PSP）强调对产品以及产品质量的个人测量。并且，PSP让第一线工作人员负责项目计划的制定（如估算和进度安排），并授权给他们来控制所有开发的软件产品的质量。

PSP过程模型定义了5个框架活动：策划、高层设计、高层设计评审、开发、后验。

策划。它将需求活动分离出来，并根据需求估算项目的规模和所需资源，并且预测缺陷数目。所有的度量都用工作表或模板记录。最后，识别开发任务，并建立项目进度计划。

高层设计。建立每个构件的外部规格说明，并完成构件设计。如果有不确定的需求，则构建原型系统。所有问题都被记录和跟踪。

高层设计评审。使用形式化验证方法（参见26章）来发现设计中的错误。对所有的重要任务和工作结果都进行度量。

开发。细化和评审构件级设计。完成编码，对代码进行评审，并进行编译和测试。对所有的重要任务和工作结果都进行度量。

后验。根据收集到的度量和测量结果（需要进行大量数据的统计分析），确定过程的有效性。度量和测量结果为提高过程的有效性提供指导。

? PSP中用到了哪些框架活动？

KEY POINT
PSP强调对所犯的错误类型进行记录和分析，以便于制定避免错误的策略。

PSP强调每一个软件工程师都尽早发现错误，并且分析所易犯的错误的种类，后者和前者同样重要。这是通过严格评估软件工程师所提交的所有产品实现的。

PSP代表的是一种严格有序的、基于度量的软件工程方法，这可能对许多第一线工作人员产生文化冲击。但是，如果将PSP恰当地介绍给软件工程师[HUM96]，软件工程的生产率和软件质

¹⁰ 需要指出的是，敏捷方法的支持者（参见第4章）同样认为过程应与团队贴近。他们只不过提出了达到同样目的的另外一种解决方法。

量将大幅度提高[FER97]。然而PSP没有被业界广泛采纳。遗憾的是，其原因更主要的在于人的本性和软件开发组织的惯性，而非PSP方法本身的好坏。PSP是对能力的极大挑战，并且需要得到一定程度的承诺（包括第一线工作人员及经理），这种支持通常是很难得到的。PSP的培训相对时间较长，价格较高。由于文化的关系，对很多软件人员来说，难以达到所需的度量水平。

69 PSP能否用做个人的有效软件过程呢？答案尚不明确。但是，即使PSP没有得到完全采纳，它所引进的许多个人过程改进的理念也值得学习。

2.6.2 团队软件过程

WebRef

关于采用PSP和TSP构建出色团队的信息，可以从下面的网站得到：
www.sei.cmu.edu/tsp/

考虑到很多工业级软件项目都由项目团队开发，Watts Humphrey吸取了引入PSP的经验教训，并提出了团队软件过程（Team Software Process, TSP）方法。TSP的目标是建立一个能够“自我管理”的项目团队，团队能自我组织进行高质量的软件开发。Humphrey [HUM98]为TSP定义了以下目标：

- 建立自我管理团队来计划和跟踪其工作，确定目标，建立团队自己的过程和计划。团队既可以是纯粹的软件开发队伍，也可以是集成的产品队伍（Integrated Product Team, IPT），可以由3~20名工程师组成。
- 指示管理人员如何指导和激励其团队，并保持团队的最佳表现。
- 使CMM第5级的行为常规化，并依此约束员工，这样可加速软件过程改进。
- 为高成熟度的软件组织提供改进指导。
- 协助大学传授工业级团队技能。

一个自我管理的团队对其整体目标有一致的理解。它定义了每个团队成员的角色和责任；跟踪定量项目数据（包括生产率和质量）；确定适合该项目的团队过程和执行该过程的具体策略；定义适合团队软件工程工作的本地标准；持续评估风险并采取风险规避措施；跟踪、管理和报告项目状态。

“找到好的团队成员不难，难的是如何使他们组成一个团队并相互配合。”

——Casey Stengel

70 TSP定义了以下框架活动：项目启动、高层设计、实现、集成、测试以及后验。正如PSP对应的活动（注意这里用的术语是不一样的），这些活动使整个团队按规范的方式计划、设计和构建软件，同时定量地评测软件过程和产品。后验确定过程改进的步骤。

KEY POINT

TSP的脚本定义了团队过程的组成部分，以及过
程中的活动。

TSP使用大量的脚本、表格和标准等来指导其团队成员的工作。脚本（script）定义了特定的过程活动（如项目启动、高层设计、实现、集成、测试和后验），以及作为团队过程一部分的其他更详细的工作职能（如开发计划的制定，需求开发，软件配置管理及单元测试）。为了更好地解释这一过程，我们可以参考下述例子：初始过程活动——项目启动（project launch）。

每个项目都通过一系列的任务（用脚本定义）启动，它使团队项目组为开始项目建立了坚实的基础。以下列出了推荐的启动脚本（只是提纲）[HUM00]：

- 同管理层一起评审项目目标，对团队的目标达成一致并形成文档。
- 确定团队角色。
- 定义团队开发过程。

WebRef

关于PSP和TSP
支持工具的信息可以从下述
网站得到：
processdash.sourceforge.net。

- 制定质量计划，明确质量目标。
- 计划需要的支持设施。
- 制定整体的开发策略。
- 制定整个项目的开发计划。
- 制定下一阶段每个工程师的详细计划。
- 将个人计划合并成团队计划。
- 重新平衡团队工作量，以获得总体上最短开发时间。
- 评估项目风险，并为每个关键风险分配跟踪责任。

需要说明的是，启动活动可以在前面提到的TSP框架活动开始之前应用。这与很多项目迭代开发的本质相吻合，使得项目团队能够适应不断变化的用户需求，并根据前面活动的经验进行调整。

TSP认为，最好的软件团队需要具有自我管理的能力。团队成员制定项目目标，调整过程以满足需要，控制进度计划，并通过度量及其结果分析，不断改进团队的软件工程方法。

与PSP类似，TSP也是一个严格的软件工程过程，在提高生产率和质量方面可以取得明显的和量化的效果。开发团队必须对过程作出全面的承诺，并且经过严格的训练以保证方法得以很好地应用。

2.7 过程技术

前些章节中讨论的通用过程模型必须加以调整才能应用于特定的软件项目团队。为了利于模型调整，开发了过程技术工具（process technology tool）以帮助软件开发组织分析现有过程、组织工作任务、控制并监测过程进度和管理技术质量[NEG99]。

过程技术工具帮助软件开发组织对2.2节中讨论的通用过程框架、任务集和普适性活动建造自动化的模型。模型通常用网络图表示，可以通过分析定义典型的工作流，并识别有可能减少开发时间和费用的其他可选的过程结构。

一旦创建了可接受的过程，其他过程技术工具可用来分配、监测、甚至控制所有过程模型中的软件工程任务。每个项目组成员都可以利用这种工具建立需要完成的工作任务检查单、工作产品检查单和需要执行的质量保证活动检查单。过程技术工具也可以和其他适用于特定工作任务的计算机辅助软件工程工具配合使用。

SOFTWARE TOOLS

过程建模工具

目的：软件组织必须首先理解软件过程，才能对其改进。过程建模工具（也称为过程技术工具或是过程管理工具）用来表示过程的关键环节，以便于更好地理解过程。这些工具也可以提供对于过程描述的链接，以协助过程的参与人员了解并掌握所须完成的操作及工作任务。过程建模工具还提供了与其他工具的链接，以支持对过程活动的定义。

机制：这类工具辅助开发团队定义一个特定过程模型的组成部分（如动作、任务、工作产品、质量保证点等），为每个过程元素的描述和内容定义提供详细的指导，并管理过程执行。在某些情况下，过程技术工具包括标准的项目管理任务如估算、进度计划、跟踪和控制等。

代表性工具¹¹

Igrafx过程工具集，Corel公司出品（www.igrafx.com/products/process），是对软件过程进行映射、度量和建模的一组工具。

Objexis团队门户，Objexis公司出品（www.objexis.com），提供了完整的过程工作流定义和控制。

2.8 产品与过程

如果过程很薄弱，最终产品必将受到影响。但是对于过程的过于依赖也是很危险的。Margaret Davis[DAV95]在一篇短文中如下评述产品和过程的双重性：

大约每十年或是五年，软件界都会对“问题”重新定义，其重点由产品问题转向了过程问题。因此，我们逐步采纳了结构化程序设计语言（产品）、结构化分析方法（过程）和数据封装（产品），到现在的重点是卡内基·梅隆大学软件工程研究所提出的能力成熟度模型（过程）（随后逐步采纳面向对象方法，敏捷软件开发）。

72 钟摆的自然趋势是停留在两个极端的中点，与之类似，软件界的关注点也会不断摆动，当上一次摆动失败时，就会有新的力量加入，促使它摆向另一个方向。这些摆动是非常有害的，因为它们可能根本改变了工作内容及工作方法，对软件工程实践人员造成混乱。而且这些摆动没有解决问题，只要把产品和过程分裂开来而不是作为辩证统一的一体，那就注定要失败。

这种二象性在科学界早有先例，当某一个理论不能对观测到的相互矛盾的结果作出合理解释时，就会出现一种二象性理论。Louis de Broglie于20世纪20年代提出的光的波粒二象性就是一个很好的例子。我相信，我们对软件组成部分和开发过程的观测证明了软件具有过程和产品的二象性。如果仅仅将软件看作一个过程或是一个产品，那就永远都不能正确地理解软件，包括其背景、应用、意义和价值。

所有的人类活动都可以看成一个过程，我们每一个人都从这些活动中获得对自我价值的认识，这些活动所产生的结果可以被多人反复地在不同的情况下使用。也就是说，我们是从我们自己或是他人对我们产品的复用中得到满足的。

因此，将复用目标融入软件开发，这不仅潜在地增加了软件专业人员从工作中获得的满足感，也增加了接受“产品和过程二象性”这一观点的紧迫性。对于一个可复用的工作制品，如果仅仅从产品或仅仅从过程的角度考虑，都不利于软件开发，这种片面的观点或者影响了人们对产品的应用环境和应用方法的认识，或者忽略了该产品还可以作为其他开发活动的输入这一事实。因此，片面地强调某一方面的观点都会极大地降低软件复用的可能性，也会大大减少工作的成就感。

“毫无疑问，如果存在理想系统，那么理想系统应该只需一次编码，编码应非常灵活和准确，使得系统只需说明适用的规则，就可以应用到每一个可能的环境中。但事实上，社会生活太复杂了，人类在能力范围内还难以实现这种理想系统。”——Benjamin Cardozo

¹¹ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

正如从产品获得满足一样，人们在创造性过程中得到了同样的（甚至更大的）成就感。艺术家不仅仅对装裱好的画卷感到高兴，更在每一笔绘画的过程中享受乐趣；作家不仅欣赏出版的书籍，更为每一个苦思得到的比喻而欣喜。一个具有创造性的专业软件人员也应该从过程中获得满足，其程度不亚于最终的产品。

软件人员的工作在今后几年中将发生重大变化。随着由编程到最终软件工程的转变，产品和过程的二象性已经成为保留创造性人才的一个重要因素。

2.9 小结

软件工程是一门涉及软件开发过程、方法、工具的学科。尽管有多种不同的软件工程过程模型，但它们都定义了：一组框架活动，完成每个活动所包含的任务集，任务完成所形成的工作产品，以及一组可应用于整个过程的普适性活动。过程模式可以用来定义过程的特征。[73]

集成能力成熟度模型（CMMI）是一个全面的过程元模型，它描述了成熟软件过程应该具备的特定目标、实践和能力。SPICE和其他的标准定义了指导软件过程评估的要求，ISO 9001:2000标准对过程中的质量管理进行检查。

软件过程的个人和团队模型，都强调了成功软件过程的关键因素：测量、计划和自我管理。本书的后续部分将详尽介绍执行软件过程——即软件工程的原理、概念和方法。

参考文献

- [AMB98] Ambler, S., *Process Patterns: Building Large-Scale Systems Using Object Technology*, Cambridge University Press/SIGS Books, 1998.
- [BAE98] Baetjer, Jr., H., *Software as Capital*, IEEE Computer Society Press, 1998, p. 85.
- [CIA01] Cianfrani, C., et al., *ISO 9001: 2000 Explained*, American Society of Quality, 2001.
- [CMM02] *Capability Maturity Model Integration (CMMI)*, Version 1.1, Software Engineering Institute, March 2002, available at <http://www.sei.cmu.edu/cmmi/>.
- [DAV95] Davis, M., "Process and Product: Dichotomy or Duality," *Software Engineering Notes*, ACM Press, vol. 20, no. 2, April, 1995, pp. 17-18.
- [DUN01] Dunaway, D., and S. Masters, *CMM-Based Appraisal for Internal Process Improvement (CBA IPI Version 1.2 Method Description*, Software Engineering Institute, 2001, can be downloaded at <http://www.sei.cmu.edu/publications/documents/01.reports/01tr033.html>.
- [ELE98] El Emam, K., J. Drouin, and W. Melo (eds.), *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*, IEEE Computer Society Press, 1998.
- [FER97] Ferguson, P., et al., "Results of applying the personal software process," *IEEE Computer*, vol. 30, no. 5, May 1997, pp. 24-31.
- [HUM96] Humphrey, W., "Using a Defined and Measured Personal Software Process," *IEEE Software*, vol. 13, no. 3, May/June 1996, pp. 77-88.
- [HUM97] Humphrey, W., *Introduction to the Personal Software Process*, Addison-Wesley, 1997.
- [HUM98] Humphrey, W., "The Three Dimensions of Process Improvement, Part III: The Team Process," *Crosstalk*, April 1998. Available at <http://www.stsc.hill.af.mil/crosstalk/1998/apr/dimensions.asp>
- [HUM00] Humphrey, W., *Introduction to the Team Software Process*, Addison-Wesley, 2000.
- [IEE93] *IEEE Standards Collection: Software Engineering*, IEEE Standard 610.12-1990, IEEE, 1993.
- [ISO00] *ISO 9001:2000 Document Set*, International Organization for Standards, 2000, <http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html>.
- [ISO01] "Guidance on the Process Approach to Quality Management Systems," Document ISO/TC 176/SC 2/N544R, International Organization for Standards, May 2001.
- [KET01] Ketola, J., and K. Roberts, *ISO 9001: 2000 in a Nutshell*, 2 ed., Paton Press, 2001.
- [MON01] Monnich, H., Jr., and H. Monnich, *ISO 9001: 2000 for Small- and Medium-Sized Businesses*, American Society of Quality, 2001.

- [NAU59] Naur, P., and B. Randall (eds.), *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*, NATO, 1969.
- [NEG99] Negele, H., "Modeling of Integrated Product Development Processes," *Proc. 9th Annual Symposium of INCOSE*, United Kingdom, 1999.
- [PAU93] Paulk, M., et al., *Capability Maturity Model for Software*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.
- [PHI02] Phillips, M., "CMMI V1.1 Tutorial," April 2002, available at <http://www.sei.cmu.edu/cmmi/>.
- [SEI00] SCAMPI, *V1.0 Standard CMMI ® Assessment Method for Process Improvement: Method Description*, Software Engineering Institute, Technical Report CMU/SEI-2000-TR-009, downloadable from <http://www.sei.cmu.edu/publications/documents/00.reports/00tr009.html>.
- [SPI99] "SPICE: Software Process Assessment, Part 1: Concepts and Introduction," Version 1.0, ISO/IEC JTC1, 1999.

习题与思考题

- 2.1 考虑框架活动——沟通。使用2.4节提供的模板设计一个完整的过程模式（一个步骤模式）。
- 2.2 过程评测的目的是什么？为什么将SPICE设计成一个过程评估的标准？
- 2.3 从SEI网站上下载CMMI文档，选择一个非项目计划的过程域。为该过程域列出特定的目标(SG)和相关的特定实践(SP)。
- 2.4 为沟通活动设计一个任务集。
- 2.5 仔细研究CMMI，讨论连续CMMI模型和阶段CMMI模型的优缺点。
- 2.6 用自己的话描述过程框架。当我们谈到框架活动适用于所有的项目时，是否意味着对于不同规模和复杂度的项目，可应用相同的工作任务？请解释。
- 2.7 普适性活动存在于整个软件过程中，你认为它们均匀存在于软件过程中还是会集中在某个或者某些框架活动中？
- 2.8 是否存在软件工程过程中通用活动不适用的情况？如果有，对其进行描述。
- 2.9 在本章的介绍中，Baetjer说过：“软件过程为用户和设计者之间、用户和开发工具之间以及设计者和开发工具之间提供交互的途径[技术]。”设计下面问题：(a) 设计者应该问用户的；(b) 用户应该问设计者的；(c) 用户对将要构建的软件的自问；(d) 设计者对于软件产品和建造该产品采取的软件过程的自问。
- 2.10 图2-1中，基于“质量关注点”指明了软件工程三个层次。这意味着在整个开发组织内采用质量管理活动，如“全面质量管理”。仔细研究，并列出全面质量管理活动中关键原则的大纲。
- 2.11 在软件界，并不是所有人都赞成使用脚本(TSP中所需的机制)。列出脚本的优缺点，分别提出至少两种适合使用脚本和不适合使用脚本的情况。
- 2.12 进一步研究PSP，并简要介绍PSP定量支持的特点。

推荐读物与阅读信息

75 软件工程及软件工程过程的当前发展状况可以通过月刊如《IEEE Software》、《Computer》和《IEEE Transactions on Software Engineering》了解，业界期刊例如《Application Development Trends》和《Cutter IT Journal》通常包含一些关于软件工程的文章。每年，IEEE和ACM会在《Proceeding of the International Conference on Software Engineering》研讨

会上总结当年的学术成果，并且在《ACM Transactions on Software Engineering and Methodology》、《ACM Software Engineering Notes》和《Annals of Software Engineering》等期刊上进一步讨论。当然网上有很多关于软件工程和软件过程的网页。

近年出版了许多关于软件工程和软件过程的书籍，有些是关于整个过程的全局概念，有些则深入讨论过程中一些重要的问题。下面是一些畅销书：

- Abran, A., and J. Moore, *SWEBOk: Guide to the Software Engineering Body of Knowledge*, IEEE, 2002.
- Ahern, D., et al., *CMMI Distilled*, Addison-Wesley, 2001.
- Chrisis, B., et al., *CMMI: Guidelines for Process Integration and Product Improvement*, Addison-Wesley 2003.
- Christensen, M., and R. Thayer, *A Project Manager's Guide to Software Engineering Best Practices*, IEEE-CS Press (Wiley), 2002.
- Glass, R., *Fact and Fallacies of Software Engineering*, Addison-Wesley, 2002.
- Hunter, R., and R. Thayer (eds), *Software Process Improvement*, IEEE-CS Press (Wiley), 2001.
- Persse, J., *Implementing the Capability Maturity Model*, Wiley, 2001.
- Pfleeger, S., *Software Engineering: Theory and Practice*, 2nd ed., Prentice-Hall, 2001.
- Potter, N., and M. Sakry, *Making Process Improvement Work*, Addison-Wesley, 2002.
- Sommerville, I., *Software Engineering*, 6th ed., Addison-Wesley, 2000.

在另一方面，Robert Glass的一本书（《Software Conflict》，Yourdon Press, 1991）中有些关于软件和软件工程过程的有趣且有争论的文章。Yourdon（《Death March Projects》，Prentice-Hall, 1997）讨论了当大型软件项目出问题时会发生的情况以及怎样避免这些错误的发生。

Garmus（《Measuring the Software Process》，Prentice-Hall, 1995）以及Florac和Carlton（《Measuring the Software Process》，Addison-Wesley, 1999）讨论了利用测量方法来统计评测软件过程的效率。

在过去的十年里，发布了大量软件工程标准和规程。IEEE《Software Engineering Standards》包含了许多标准，这些标准几乎涵盖软件技术所有重要方面。ISO 9001:2000文档集为希望改善质量管理活动的软件组织提供了指导。其他的软件工程标准可以从DOD、FAA、政府及非营利组织得到。Fairclough（《Software Engineering Guides》，Prentice-Hall, 1996）提供了关于欧洲宇航局（ESA）制定的软件工程标准的详细索引。

因特网上有很多关于软件工程和软件过程的信息资源。可以在SEPA的网页上找到一个与软件过程相关的网址索引列表：<http://www.mhhe.com/pressman>。

第3章 过程模型

要点浏览

概念：惯例过程模型定义包含活动、动作、任务、里程碑和工作产品在内的明确的集合，是高质量软件开发所必需的。这些过程模型不是完美的，但是确实为软件工程工作提供了有用的路线图。

人员：软件工程师及其经理根据他们的需要采用某一惯例过程模型。同时，软件的需求方也参与了遵循该过程模型的活动。

重要性：惯例模型之所以重要是因为它为活动提供稳定性、控制和有组织性，如果活动失去控制，将会变得非常混乱。有些人把惯例过程模型称为“严格的过程模型”，因为这种模型通常包含了CMMI（参见第2章）所建议的能力要

求。尽管如此，每一种过程模型都应该适当地调整，使得模型能够有效地应用于某一特定的软件项目。

步骤：这些过程通过一套框架活动来指导软件团队，这些框架活动被组织到一个过程流中，这个过程流可能是线性的、增量的或演进的。每种过程模型的术语和细节不同，但是通用的框架活动在一定程度上保持一致。

工作产品：从软件工程师的角度来说，工作产品就是过程定义的一系列任务和活动的结果，即程序、文档和数据。

质量保证措施：有大量的软件过程评估机制，各个组织利用这些机制判定软件过程的成熟度。尽管如此，评价所采用过程的有效性，最好的指标是所构建产品的质量、适时性以及长期生存能力。

关键概念
AOSD模型
CBD模型
协同开发
演进过程
形式化方法
增量过程
惯例模型
原型开发
RAD模型
螺旋模型
统一过程
瀑布模型

最早提出惯例过程模型是为了改变软件开发的混乱状况，使软件开发更加有序。历史证明这些传统模型为软件工程工作增加了大量有用的结构化设计，并为软件团队提供了有效的路线图。尽管如此，软件工程工作和它产生的产品仍然停留在“混乱的边缘”[NOG00]。

在一篇探讨软件世界中有序和混乱之间奇怪关系的论文中，Nogueira和他的同事[NOG00]指出：

混乱的边缘被定义为“有序和混乱之间的一种自然状态，结构化和反常之间的重大妥协”[KAU95]。混乱的边缘可以被视为一种不稳定，部分结构化的状态……它的不稳定是因为它不停地受到混乱或者完全有序的影响。

我们通常认为有序是自然的完美状态。这可能是个误区。研究证实……打破平衡的活动会产生创造力、自我组织过程和更高的回报[ROO96]。完全的有序意味着缺乏可变性，而可变性在某些不可预测的环境下往往是一种优势。变更发生时都是存在一些可以组织变更的结构，但结构还不够严格，无法阻止变更的发生。另一方面，太多的混乱会使协调和一致成为不可能。缺少结构并不意味着无序。

上述这段话对于软件工程的哲学思想有着重要的意义。如果惯例过程模型¹力求实现结构化和有序，那么对于富于变化的软件世界，这一模型是否不合适呢？如果我们抛弃传统过程模型（以及它们带来的秩序），取而代之以一些不够结构化的东西，会导致软件世界失去协调和一致吗？

这些问题没有明确的答案，但是软件工程师有很大的选择余地。本章我们将学习惯例过程模型，这些模型把有序和项目的一致性作为首要目标。在第4章中我们将学习敏捷过程方法，这些方法把自我组织、协作、沟通和可适应性作为过程的主要原则。

3.1 惯例过程模型



惯例过程模型提供了一个过程框架，由对应于软件工程动作的明确的任务集组成。

每一个软件工程组织都应该用一组特定的框架活动（参见第2章）来描述其所采用的软件过程。每个框架活动都由一组软件工程动作组成，每一个动作都由任务集定义，任务集说明了为完成开发目标所要进行的工作（及工作产品）。所得到的过程模型应适当加以调整，以适应每个项目的特征、项目团队人员的特点以及工作环境的特点。不管采用了何种过程模型，软件工程师通常都会选择一个通用的过程框架，这个框架包含以下一些框架活动：沟通、策划、建模、构建、部署。

“向前的路有好多条，但站着不动的路只有一条。”

—Franklin D. Roosevelt



一个过程是惯例并不就意味着它是静止的。惯例过程应该根据团队人员、问题和项目的具体情况加以调整。

在下面小节中，我们将探讨一些惯例过程模型。我们称为“惯例”是因为，它规定了一套过程元素——框架活动、软件工程动作、任务、工作产品、质量保证以及每个项目的变更控制机制。每个过程模型还定义了工作流——也就是说，过程元素之间相互关联的方式。

所有的软件过程模型都支持第2章中描述的通用框架活动，但是每一个模型都对框架活动有不同的侧重，并且定义了不同的工作流如何以不同的方式执行每一个框架活动（以及软件工程动作和任务）。

78

3.2 瀑布模型

有时候，可以相当清楚地了解问题的需求，从沟通到部署，工作以线性的方式进行。这种情况通常发生在需要对一个已有系统进行明确定义的适应性调整或增强的时候（比如政府修改了规则，财务软件需要进行相应修改）；也可能发生在一些新的开发项目上，但是需求必须是准确定义和相对稳定的。

瀑布模型，又被称为经典生命周期，它提出了一个系统的、顺序的软件开发方法²，从用户需求规格说明开始，通过策划、建模、构建和部署的过程，最终提供一个完整的软件并提供持续的技术支持。如图3-1所示。

¹ 惯例过程模型通常称为“传统的”过程模型。

² 尽管对最早Winston Royce [ROY70]提出的瀑布模型进行了改进，加入了“反馈”过程，但绝大多数软件组织在应用该过程模型时都将其视为严格的线性模型。

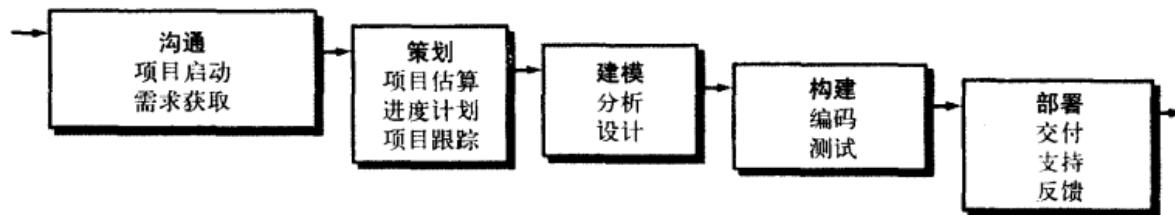


图3-1 瀑布模型

瀑布模型是软件工程最早的范例。尽管如此，在过去的20多年中，对这一过程模型的批评，使它最热情的支持者都质疑它的有效性 [HAN95]。在运用瀑布模型的过程中，人们遇到的问题包括：

为什么瀑布模型有时候会失败？

79

1. 实际的项目很少遵守瀑布模型提出的顺序。虽然线性模型可以加入迭代，但是它是用间接的方式实现的，结果是，随着项目的推进，变更可能带来混乱。
2. 客户通常难以清楚地描述所有的需求。而瀑布模型却需要客户明确需求，因此很难适应在许多项目开始阶段必然存在的不确定性。
3. 客户必须要有耐心，因为只有在项目接近尾声的时候，他们才能得到可执行的程序。对于系统中存在的重大缺陷，如果在可执行程序评审之前没有被发现，将可能造成惨重损失。

在分析一个实际项目时Bradac[BRA94]发现，瀑布模型的线性特性在某些项目中会导致“阻塞状态”，开发团队的一些成员要等待另一些成员完成相关任务。事实上，花在等待上的时间可能超过花在生产上的时间。在线性过程的开始和结束，这种阻塞状态更容易发生。

目前，软件工作是一个高速的、永不停止的变化流，特性、功能和信息内容都会变化，瀑布模型往往并不合适。尽管如此，当需求确定，工作能够采用线性的方式完成的时候，瀑布模型是一个很有用的过程模型。

3.3 增量过程模型

在许多情况下，初始的软件需求有明确的定义，但是整个开发过程却不宜单纯运用线性模型。同时，可能迫切需要为用户迅速提供一套功能有限的软件产品，然后在后续版本中再细化和扩展功能。在这种条件下，需要选用一种以增量的形式生产软件产品的过程模型。

“大多数情况下，软件工作遵从自行车第一定律：不论你去哪，你都会顶风骑上坡路。”

——作者不详

3.3.1 增量模型

KEY POINT
增量模型发布一系列称为增量的版本，随着每个版本交付，逐步为用户提供更多的功能。

增量模型以迭代的方式运用瀑布模型。如图3-2所示，随着时间的推移，增量模型在每个阶段运用线性序列。每个线性序列生产出一个软件的可交付增量[MCD93]。例如，采用增量范型开发的字处理软件，在第1个增量中提供基本的文件管理、编辑和文档生成功能；在第2个增量中提供复杂的编辑和文档生成功能；在第3个增量中提供拼写和语法检查功能；在第4个增量中提供高级页面排版功能。需要注意的是，任何增量的过程流可能使用3.4.1节讨论的原型开发范型。

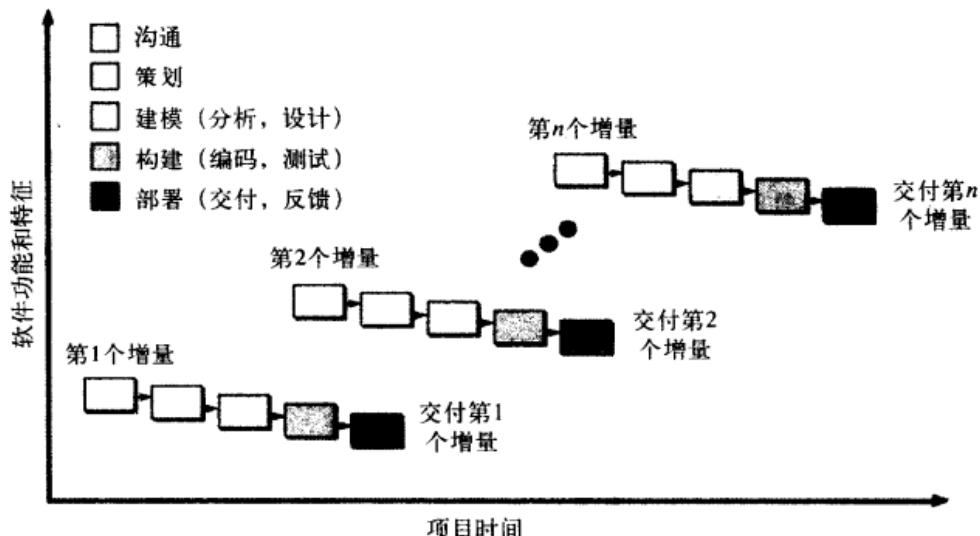


图3-2 增量模型

运用增量模型的时候，第1个增量往往是核心产品。也就是说，满足了基本的需求，但是许多附加的特性（一些是已知的，一些是未知的）没有提供，客户使用该核心产品或者进行仔细的评价，并根据评价结果制定下一个增量计划。这份计划说明了需要对核心产品进行的修改，以便更好地满足客户的要求，也说明了需要增加的特性和功能。每一个增量的交付都会重复这一过程，直到最终产品的产生。



如果你的客户要求你在一个不可能完成的时间提交产品，向他建议只提交一个或几个增量，此后再提交软件的其他增量。

增量过程模型，类似于原型开发和其他演进方法，具有迭代的性质。但和原型开发不同的是，增量模型侧重于每个增量都提交一个可以操作的产品。早期的增量可以看作是最终产品的片段版本，但是它们确实具备了用户服务能力，也为用户的评价提供一个平台³。

如果在项目既定的商业要求期限之前不可能找到足够的开发人员，这种情况下增量模型显得特别有用。早期的增量可以由少量的人员实现。如果核心产品的口碑不错，可以为下一个增量投入更多的人力。同时，增量模型可以规避技术风险。例如，一个系统需要用到某个正在开发的新硬件，而这个新硬件的交付日期不确定。因此可以在早期的增量中避免使用这个硬件，这样可以保证部分功能按时交付给最终用户，不至于造成过分的延期。

3.3.2 RAD模型

快速应用程序开发（Rapid Application Development, RAD）是一种侧重于短暂的开发周期的增量软件过程模型。RAD是瀑布模型的“高速”变体，通过基于构件的构建方法实现快速开发。如果需求被很好地理解了，项目的边界也是固定的⁴，RAD模型能够使开发团队在一段非常短的时间内（比如60~90天）创造出“全功能系统”[MAR91]。

与其他过程模型类似，RAD模型也可以映射到前文提到的一些通用的框架活动中。沟通用来理解业务问题和软件产品必须具有的特征；策划确保多个软件团队能够并行工作于不同

³ 需要注意的是，增量原理也运用在第4章提到的所有敏捷过程模型中。

⁴ 这些条件根本无法保证。事实上，很多软件项目在初期阶段需求定义很不清楚。在这种情况下，原型开发模型和演化模型（参见3.4节）更为适用[REI95]。

的系统功能；建模包括三个主要阶段——业务建模、数据建模和过程建模，这时要建立设计描述，作为RAD构建活动的基础；构建侧重于利用已有的软件构件并应用代码自动生成技术；最后，必要的话部署可以为以后的迭代建立基础 [KER94]。

图3-3显示了RAD过程模型。很显然，严格的时间要求使得RAD项目必须具备“可伸缩范围”[KER94]。如果某个商业应用项目的每个主要功能都能在3个月内完成（采用前文提到的方法），采用RAD是个不错的选择。每个主要功能分配给一个独立的RAD团队，然后再集成成为一个整体。

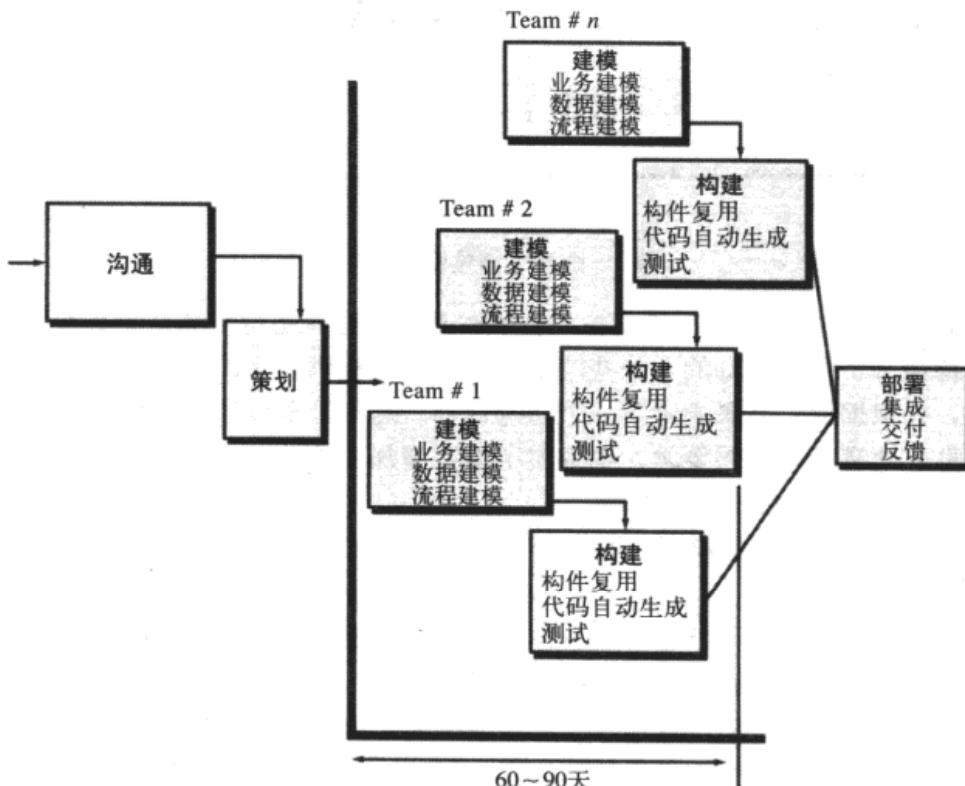


图3-3 RAD模型

82

RAD模型的
缺点是什么？

与其他过程模型类似，RAD模型也有不足之处[BUT94]：(1) 对于大型的可伸缩的项目，RAD需要大量的人力资源来创建多个相对独立的RAD团队；(2) 如果开发者和客户没有为短时间内急速完成整个系统做好准备，RAD项目将会失败；(3) 如果一个系统不能合理地模块化，RAD构件建立会有很多问题；(4) 如果系统需求是高性能，并且需要通过调整构件接口的方式来提高性能，不能采用RAD模型；(5) 技术风险很高的情况下（例如，一个新的应用开发，大量使用新技术），不宜采用RAD。

3.4 演化过程模型

KEY POINT
演化过程模型中，每次迭代产生软件的一个更完整的版本。

软件，类似于其他复杂的系统，会随着时间的推移而演化[GIL88]。在开发的过程中，业务和产品需求经常发生变化，直接导致最终产品难以实现；严格的交付时间使得开发团队不可能圆满完成软件产品，但是必须交付功能有限的版本以应对竞争或商业压力；很好地理解了核心产品和系统需求，但

是产品或系统扩展的细节问题却没有定义。在上述情况和类似情况下，软件工程师需要一种专门应对不断演变的软件产品过程模型。

演化模型是迭代的过程模型，使得软件工程师能够逐步开发出更完整的软件版本。

3.4.1 原型开发

很多时候，客户提出了软件的一些基本功能，但是没有详细定义输入、处理和输出需求。另一种情况下，开发人员可能对算法的效率、操作系统的兼容性和人机交互的形式等情况不确定。在这些情况和类似情况下，原型开发范型是最好的解决办法。



如果你的客户有一个合理的
需求，但是对细节
没有思路，那么
不妨先开发一个
原型。

虽然原型开发可以作为一个独立的过程模型，但是更多的时候是作为一种技术，在本章讨论的其他过程模型中应用。不论人们以什么方式运用它，当需求很模糊的时候，原型开发范型帮助软件工程师和客户更好地理解究竟需要做什么。

原型开发范型（图3-4）开始于沟通。软件工程师和客户进行会晤，定义软件的整体目标，明确已知的需求，并大致勾画出以后再进一步定义的东西。然后迅速策划一个原型开发迭代并进行建模（以快速设计的方式）。快速设计要集中在那些客户和最终用户能够看到的方面，比如人机接口布局或者输出显示格式。快速设计产生一个原型，对原型进行部署，然后由客户或者用户进行评价。根据反馈，进一步细化软件的需求。在原型系统不断调整以满足用户需求的过程中，采用迭代技术，同时也使开发者逐步清楚用户的需求。

83

理想状况下，原型系统提供了定义软件需求的一种机制。当构建了可执行的原型系统时，开发者尝试利用已有的程序片段或应用工具（如报告生成器、窗口管理器等），快速产生可执行的程序。

如果我们的原型达到了上述目的，那么接下来它有什么用呢？Brooks[BRO75]给出了答案：

在大多数项目中，构建的第一个系统很少有可用的，可能太慢了，太大了，很难使用，或者同时具备上述三点。除了重新开始，没有更好的选择。更巧妙的方法是构建一个重新设计的版本，解决上述问题……如果使用了新的系统概念或者新的技术，以前的系统就要丢弃。最好的计划也不能保证第一次就能正确无误。对于管理者来说，不是建立一个试验系统然后扔掉的问题——你应该扔掉这个系统。你需要考虑的仅仅是扔掉这个系统还是把这个系统交给用户。

原型可以作为第一个系统，也就是Brooks推荐我们扔掉的系统。但这可能是一种理想的方式。客户和开发者确实都喜欢原型范型。客户对实际的系统有了直观的认识，开发者也迅速建立了一些东西。但是，原型开发也存在一些问题，原因如下：

84

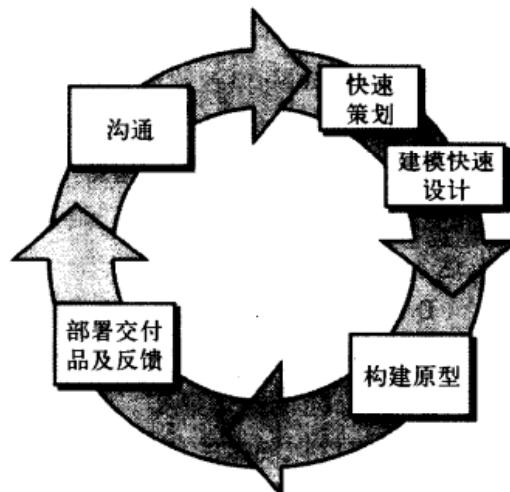


图3-4 原型开发模型



对于要求把一个粗糙的原型系统变为工作产品的压力，建议尽量抵制。这样做的结果往往是产品质量受损害。

1. 客户看到了软件的工作版本，却未察觉到整个软件是用“口香糖和包装带”搭成的，也未察觉到为了尽快完成软件，开发者没有考虑整体软件质量和长期的可维护性。当开发者告诉客户整个系统需要重建以提高软件质量的时候，客户会不愿意，并且要求对软件稍加修改使其变为一个可运行的产品。因此，软件开发管理往往陷入失效。

2. 开发者为了使一个原型快速运行起来，往往在实现过程中采用折衷的手段。他们经常会使用不合适的操作系统或程序设计语言，仅仅因为当时可用和熟悉。他们也经常会采用一种低效的算法，仅为了证明系统的能力。时间长了，开发者可能会适应这些选择，而忽略了这些选择其实并不合适的理由，结果造成并不完美的选择变成了系统的组成部分。

尽管问题经常发生，原型开发对于软件工程来说仍是一个有效的模型。关键是要在游戏开始的时候制定规则，也就是说，客户和开发者必须承认原型是为定义需求服务的。然后丢弃原型（至少是部分丢弃），实际的软件系统是以质量第一为目标的。

SAFEHOME

选择过程模型：第1部分

[场景] CPI公司软件工程部会议室。该公司专注于开发家用和商用的消费产品。

[人物] Lee Warren，项目经理；Doug Miller，软件工程经理；Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员。

[对话]

Lee：我简单说一下。正如我们现在所看到的，我已经花了很多时间讨论SafeHome产品的生产线。毫无疑问，我们做了很多工作定义这个东西，我想请各位谈谈你们打算如何做这个产品的软件部分。

Doug：看起来，我们过去在软件开发方面相当混乱。

Ed：Doug，我不明白，我们总是能成功开发出产品来。

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

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

Doug (笑)：我希望我们的开发方法更专业一些。我上星期参加了一个培训班，学了很多关于软件工程的知识。我们现在需要一个过程。

Jamie (皱眉)：我的工作是编程，不是文书。

Doug：在你反对我之前，请先尝试一下。我想说的是……(Doug开始讲述本书第2章讲述的过程框架和本章到目前为止讲到的过程模型。)

Doug：所以，似乎线性模型并不适合我们……它假设我们此刻明确了所有的需求，而事实上并不是这样。

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

Doug：对。

Ed: 原型开发方法听起来不错，正适合我们现在的处境。

Vinod: 有个问题，我担心它不够结构化。

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

3.4.2 螺旋模型

螺旋模型，最早由Boehm[BOE88]提出，是一种演进式软件过程模型。它结合了原型的迭代性质和瀑布模型的系统性和可控性特点。它具有快速开发越来越完善软件版本的潜力。Boehm[BOE01]这样描述螺旋模型：

螺旋模型是一种风险驱动型过程模型生成器，对于软件集中的系统，它可以指导多个共利益者的协同工作。它有两个显著的特点。一是采用循环的方式逐步加深系统定义和实现的深度，同时降低风险。二是确定一系列里程碑，确保共利益者都支持可行的和令人满意的系统解决方案。

运用螺旋模型，把软件开发为一系列演进版本。在早期的迭代中，软件可能是一个理论模型或原型。在后来的迭代中，会产生一系列逐渐完整的系统版本。

KEY POINT
螺旋模型能运用在应用开发的整个生命周期，从概念开发到维护。

螺旋模型被分割成一系列由软件工程团队定义的框架活动。为了讲解方便，我们使用前文讨论的通用框架活动⁵。如图3-5所示，每个框架活动代表螺旋上的一个片段。随着演进过程开始，从圆心开始顺时针方向，软件团队执行螺旋上的一圈表示的活动。在每次演进的时候，都要考虑风险（第25章）。每个演进过程，还要标记里程碑——沿着螺旋路径达到的工作产品和条件的结合体。

86

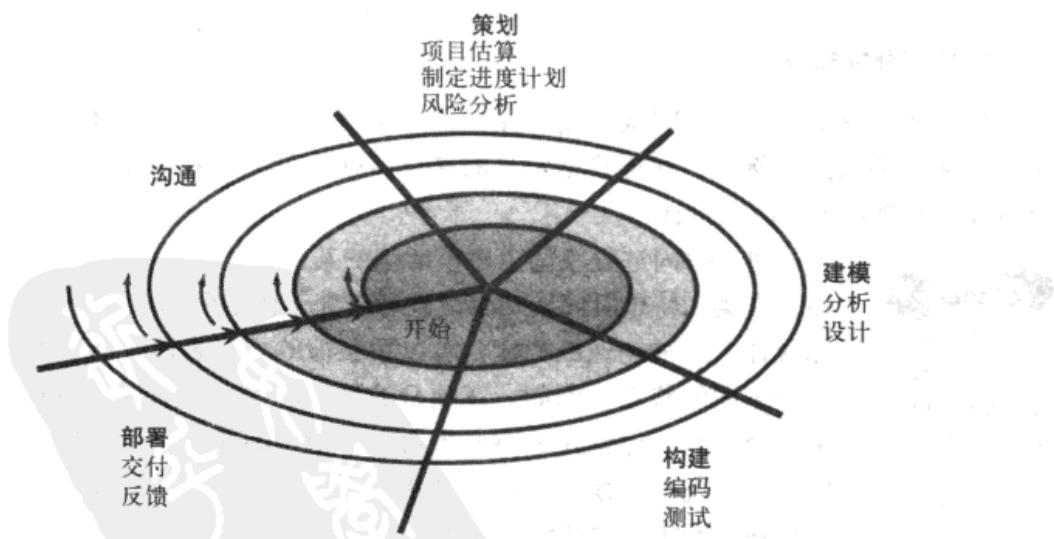


图3-5 典型的螺旋模型

⁵ 这里讨论的螺旋模型有别于Boehm提出的模型。原始的螺旋模型可参见[BOE88]。关于Boehm的螺旋模型的更多讨论可参见[BOE98]。

螺旋的第一圈一般开发出产品的规格说明，接下来开发产品的原型系统，并在每次迭代中逐步完善，开发不同的软件版本。螺旋的每圈都会跨过策划区域，此时，需调整项目计划，并根据交付后用户的反馈调整预算和进度。另外，项目经理还调整完成软件开发需要迭代的次数。

WebRef

可以在如下网址
获得关于螺旋模
型的有用信息
[http://www.sei.
cmu.edu/cbs/
spiral2000](http://www.sei.
cmu.edu/cbs/
spiral2000)

其他过程模型当软件交付后就结束了。螺旋模型则不同，它应用在计算机软件的整个生命周期。因此，螺旋上的第一圈可能表示“概念开发项目”，它起始于螺旋的中心，经过多个迭代⁶，直到概念开发的结束。如果这个概念将被开发成实际的产品，过程模型将继续沿着螺旋向外伸展，此时成为“新产品开发项目”。新产品可能沿着螺旋通过一系列的迭代不断演进。最后，可以用一圈螺旋表示“产品提高项目”。本质上，当螺旋模型以这种方式进行下去的时候，它将永远保持可操作性，直到软件产品的生命周期结束。过程经常会处于休止状态，但每当有变更时，过程总能够在合适的人口点启动（如产品提高）。



如果你的项目要
求固定预算开发
(通常不是一个
好方法)，螺旋
模型会带来问
题：每一圈完成
的时候，都将重
新计划项目开销。

87

螺旋模型是开发大型系统和软件的理想方法。由于软件随着过程的推进而变化，在每一个演进层次上，开发者和客户都可以更好地理解和应对风险。螺旋模型把原型开发作为降低风险的机制，更重要的是，开发者可以在产品演进的任何阶段使用原型开发方法。它保留了经典生命周期中系统逐步细化的方法，但是把它纳入一种迭代框架之中，这种迭代方式与真实世界更加吻合。螺旋模型要求在项目的所有阶段始终考虑技术风险，如果适当地应用该方法，能够在风险变为问题之前化解风险。

与其他模型相似，螺旋模型也并不是包治百病的灵丹妙药。很难说服客户（特别是以合同的形式）演进的方法是可控的。它依赖大量的风险评估专家来保证成功。如果有较大的风险没被发现和管理，肯定会发生问题。

3.4.3 协同开发模型

协同开发模型，有时候叫做协同工程，可以表示为一系列框架活动、软件工程动作和任务以及相应地状态。例如，螺旋模型定义的建模活动由以下一些动作完成：开发原型、建立分析模型和规格说明以及设计⁷。



协同模型更适合
于不同的工程团
队共同开发的系
统工程项目。

对于协同过程模型中建模活动的某一软件工程任务，图3-6给出了一种描述。在某一特定时间，建模活动可能处于图中所示的任何一种状态⁸。其他活动或任务（如沟通或构建）可以用类似的方式表示。所有的活动同时存在并处于不同的状态。例如，在项目的早期，沟通活动（图中并未标明）完成了第一个迭代，停留在等待变更状态。初始沟通完成后，建模活动一直停留在空状态，现在转换到正在开发状态。如果客户要求必须改变需求，建模活动从正在开发状态转换到等待变更状态。

协同过程模型定义了一系列事件，这些事件将触发软件工程活动、动作或者任务的状态

⁶ 沿轴指向中心的箭头区分开了部署和沟通两个区域，表明沿同一个螺旋路径，潜在的局部迭代。

⁷ 需要注意的是，分析和设计都是很复杂的动作，需要进行大量的讨论。本书的第二部分将详细讨论这些问题。

⁸ 状态是外部可见的某种行为模式。

转换。例如，设计的早期阶段（建模活动期间发生的软件工程动作），发现了分析模型中的不一致性，于是产生了分析模型改正事件，该事件将触发从完成状态到等待变更状态的分析动作。

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

3.4.4 演化过程模型的最终评述

我们注意到，现代计算机软件总是在持续改变，这些变更通常要求在非常短的期限内实现，并且要充分满足客户/用户的要求。许多情况下，及时投入市场是最重要的管理需求。如果市场时间错过了，软件项目自身可能会变得毫无意义⁹。

“我今天只走这么远，只有明天才能为我指明方向。”

——Dave Matthews乐队

演化过程模型就是为了解决上述问题的，但是，作为一类通用的过程模型，它们也有缺点。Nogueira和他的同事[NOG00]对此概括如下：

尽管演化软件过程毫无疑问具有一定的优势，但我们还是有一些担忧。首先，原型开发（和其他更加复杂的演化过程）由于构建产品需要的周期数目不确定，给项目策划带来了困难。大多数的项目管理和估算技术是基于线性活动的，所以并不完全适用于演化软件过程。

其次，演化软件过程没有确定演进的最快速度。如果演进的速度太快，完全没有间歇时间，项目肯定会陷入混乱；反之，如果演进速度太慢，则会影响生产率……

再次，软件过程应该侧重于灵活性和可扩展性，而不是高质量。这种说法听起来很惊人。但是，我们必须优先追求开发速度，而不是零缺陷。为了追求高质量而延长开发时间势必造成产品推迟交付，从而失去进入市场的良机。这种对于范型的偏离是由于处在混乱边缘的竞争所造成的。

一个强调灵活性、可扩展性和开发速度而不是高质量的软件过程确实听起来令人震惊。可是，很多广为人们尊重的软件工程专家都这样建议（例如[YOU95], [BAC97]）。

演化模型的初衷是采用迭代或者增量的方式开发高质量软件¹⁰。可是，用演化模型也可以

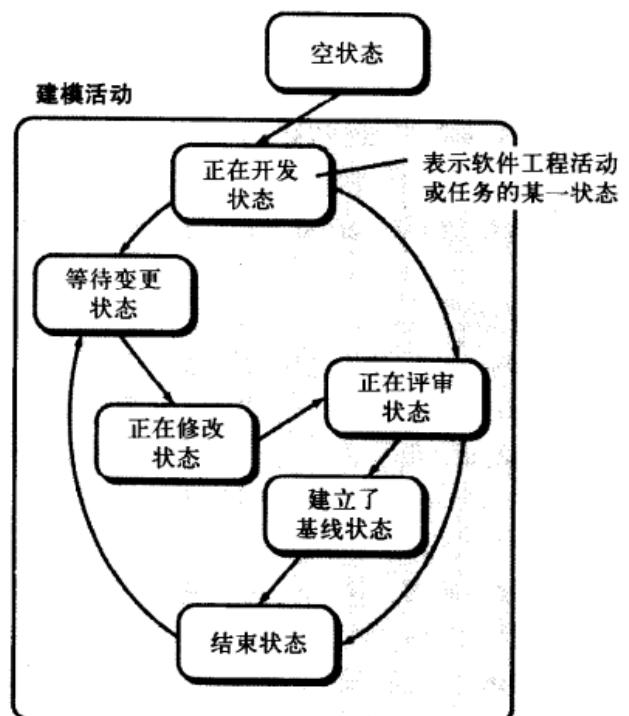


图3-6 协同过程模型的一个组成部分

88

89

⁹ 需要注意的是，尽管及时投入市场很重要，但最早进入市场并不保证一定会成功。事实上，许多非常成功的软件是第二个或第三个进入市场的，他们的成功在于吸取了前人的教训。

¹⁰ 在这里，软件质量的含义非常广泛，不仅仅指客户满意度，也指第26章将要讲到的各种技术指标。

做到强调灵活性、可扩展性和开发速度。软件开发团队及其经理所面临的挑战就是在这些严格项目和产品参数与客户（软件质量的最终仲裁者）满意度之间找到一个合理的平衡点。

SAFEHOME

选择过程模型：第2部分

[场景] CPI公司软件工程部会议室，该公司生产家用和商用消费类产品。

[人物] Lee Warren，项目经理；Doug Miller，软件工程经理；Ed和Vinod，软件团队成员。

[对话]

(Doug介绍演化过程的思想)

Ed：我现在有了一些想法。增量方法挺有意义的，我很喜欢螺旋模型，听起来很实用。

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

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

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

Lee（皱眉）：我同意这种看法，可是高管人员不喜欢这种方式，他们喜欢确定的计划。

Doug（笑）：老兄，你应该给他们上一课。

90

3.5 专用过程模型

专用过程模型具有前面章节中提到的传统过程模型的一些特点，但是，专用过程模型往往应用面较窄，只适用于某些特定的软件工程方法¹¹。

3.5.1 基于构件的开发

商品化成品（commercial off-the-shelf, COTS）软件构件，由厂家作为产品供应，它们可以在构建软件时使用。这些构件通过良好定义的接口提供特定的功能，能够集成到软件中。

WebRef

基于构件开发的相关信息可以参见www.cbd-hq.com。

基于构件开发模型（见30章）具有许多螺旋模型的特点，它本质上是演化模型[NIE92]，需要以迭代方式构建软件。不同之处在于，基于构件开发模型采用预先打包的软件构件开发程序。

建模和构建活动开始于识别可选构件。这些构件有些设计成传统的软件模块，有些设计成面向对象的类或软件包¹²。不考虑构件的开发技术，基于构件开发模型由以下步骤组成（采用演进方法）：

¹¹ 在某些情况下，这些专用过程也许更确切地应该称为技术的集合或方法论，是为了实现某一特定的软件开发目标而制定的。但它们确实也提出了一种过程。

¹² 本书第二部分讨论面向对象技术。在这里，类封装了一组数据以及处理数据的过程。类包是一组共同产生某种结果的相关类的集合。

- 对于该问题领域的基于构件的可用的产品进行研究和评估。
- 考虑构件集成的问题。
- 设计软件架构（第10章）以容纳这些构件。
- 将构件（第11章）集成到架构中。
- 进行充分的测试（第13章和第14章）以保证功能正常。

基于构件开发模型能够使软件复用，软件复用为软件工程师带来极大收益。QSM联合有限公司对复用性的研究报告指出，基于构件开发缩短了70%的开发周期，减少了84%的项目开销，生产率指数可达到26.2，而工业标准值为16.9[YOU94]。尽管这些结果得益于丰富的构件库，但毫无疑问，对软件工程师来讲，基于构件开发模型具有明显的优势。

3.5.2 形式化方法模型

形式化方法模型（第28章）的主要活动是生成计算机软件形式化的数学规格说明。形式化方法使软件工程师可以应用严格的数学符号来说明、开发和验证基于计算机的系统。这种方法的一个变型是净室软件工程[MIL87, DYE92]，净室软件工程目前应用于一些软件开发机构，我们将在第29章讨论这种方法。

“写一个错误的程序比读懂一个正确的程序更容易。”

——Alan Perlis

形式化方法提供了一种机制，使得在软件开发中可以避免一些问题，而这些问题在使用其他软件工程范型时难以克服。使用形式化方法，歧义性问题、不完整问题、不一致问题都能更容易被发现和改正——不是依靠特定的评审，而是应用数学分析的方法。在设计阶段，形式化方法是程序验证的基础，使软件工程师能够发现和改正一些往往会被忽略的问题。

虽然不是一种主流的方法，形式化方法的意义在于可以提供无缺陷的软件。尽管如此，人们还是对在商业环境中应用形式化方法有怀疑，这表现在：

既然形式化方法能够保证软件的正确性，为什么没有被广泛应用呢？

- 目前，形式化模型开发非常耗时，成本也很高。
- 只有极少数程序员具有应用形式化方法的背景，因此需要大量的培训。
- 对于技术水平不高的客户，很难用这种模型进行沟通。

尽管有这些疑虑，软件开发者中还是有很多形式化方法的追随者，比如有人用来开发高度关注安全的软件（如飞行器和医疗设施），或者开发软件出错将导致重大经济损失的软件。

3.5.3 面向方面的软件开发

WebRef
关于AOP的资源和信息可以参见aosd.net。

不论选择什么软件过程，复杂软件无一例外地实现了一套局部化的特征、功能和信息内容。这些局部的软件特性被做成构件（例如，面向对象的类），然后在系统架构中使用。随着现代计算机系统变得更加复杂，某些关注点——客户需要的属性或者技术兴趣点——体现在整个架构设计中。有些关注点是系统的高层属性（例如安全性，容错能力），还有一些关注点影响了系统的功能（例如，业务规则的应用），还有一些是系统的（例如，任务同步或存储器管理）。

如果某个关注点涉及系统多个方面的功能、特性和信息，这些关注点通常称为横切关注点（crosscutting concern）。方面需求（aspectual requirement）定义了那些对整个软件体系结

3.6 统一过程

Ivar Jacobson, Grady Booch和James Rumbaugh[JAC99]在他们关于统一过程的处女作中，讨论了关于建立一种“用例驱动，以架构为核心，迭代并且增量”的软件过程的必要性，阐述如下：

当前，软件朝着更大、更复杂的系统发展。这是由于计算机的计算能力逐年递增，使得人们对其期望值增大。另一方面，还受Internet应用膨胀的影响，要促使各类信息交换……我们从软件版本的提高中学到了如何改进产品，使我们对更复杂软件的胃口越来越大。我们希望软件更好地满足我们的需要，结果导致软件更加复杂。简而言之，我们想要的越来越多。

在某种程度上，统一过程（Unified Process, UP）尝试着从传统软件过程中挖掘最好的特征和性质，但是以敏捷软件开发（第4章）中许多最好的原则来实现。统一过程认识到与客户沟通以及从用户的角度描述系统（即，用例¹⁴）并保持描述的一致性的重要性。它强调软件体系结构的重要作用，并“帮助架构师专注于正确的目标，例如可理解性、对未来变更的可适应性以及复用”[JAC99]。它建立了一种迭代的、增量的过程流，提供了一种演进的特性，这对现代软件开发非常重要。

本节概要介绍了统一过程的一些关键元素。本书的第二部分讨论应用该过程的主要方法，辅助的UML¹⁵建模技术，以及在实际软件工程工作中应用统一过程所需要的表示法。

3.6.1 简史

从20世纪80年代到90年代初期，面向对象（Object-Oriented, OO）方法和程序设计语言¹⁶在软件工程领域中获得了广泛的认可。在这一时期，涌现了大量的面向对象分析（Object-Oriented Analysis, OOA）和面向对象设计（Object-Oriented Design, OOD）方法，并提出了通用的面向对象过程模型（类似于本章中介绍的演化模型）。就像软件工程中大多数新的范型一样，每种OOA和OOD方法的追随者们常常为哪种方法最好争论不休，但是没有哪一种方法占据主流。

20世纪90年代早期，James Rumbaugh [RUM91]、Grady Booch [BOO94]和Ivar Jacobson [JAC92]开始研究“统一方法”，他们的目标是结合各自方法中最好的特点，并吸纳其他OO领域专家提出的其他特点（例如，[WIR90]）。他们的成果就是UML——统一建模语言，这种语言包含了大量用于OO系统建模和开发的符号。到了1997年，UML已经变成了面向对象软件开发的工业标准。同时，Rational（瑞理）公司和其他生产商开发了自动化工具以支持UML方法。

UML提供了支持面向对象软件工程实践必要的技术，但它没有提供指导项目团队应用该技术时的过程框架。接下来的几年中，Jacobson、Rumbaugh和Booch建立了统一过程模型，一

¹⁴ 用例（use case）是一种文字描述或模板，从用户的角度描述系统功能和特性。用例由用户来写，并作为创建更为复杂的分析模型的基础。

¹⁵ UML（Unified Modeling Language，统一建模语言），已经成为分析和设计建模中应用最广泛的表示法。它统一了三种最重要的面向对象表示方法。

¹⁶ 如果不熟悉面向对象方法，可参看本书第8章和第9章中的简要介绍，更详细的介绍参看[REE02], [STI01]或[FOW99]。

3.6 统一过程

Ivar Jacobson, Grady Booch和James Rumbaugh[JAC99]在他们关于统一过程的处女作中，讨论了关于建立一种“用例驱动，以架构为核心，迭代并且增量”的软件过程的必要性，阐述如下：

当前，软件朝着更大、更复杂的系统发展。这是由于计算机的计算能力逐年递增，使得人们对其期望值增大。另一方面，还受Internet应用膨胀的影响，要促使各类信息交换……我们从软件版本的提高中学到了如何改进产品，使我们对更复杂软件的胃口越来越大。我们希望软件更好地满足我们的需要，结果导致软件更加复杂。简而言之，我们想要的越来越多。

在某种程度上，统一过程（Unified Process, UP）尝试着从传统软件过程中挖掘最好的特征和性质，但是以敏捷软件开发（第4章）中许多最好的原则来实现。统一过程认识到与客户沟通以及从用户的角度描述系统（即，用例¹⁴）并保持描述的一致性的重要性。它强调软件体系结构的重要作用，并“帮助架构师专注于正确的目标，例如可理解性、对未来变更的可适应性以及复用”[JAC99]。它建立了一种迭代的、增量的过程流，提供了一种演进的特性，这对现代软件开发非常重要。

本节概要介绍了统一过程的一些关键元素。本书的第二部分讨论应用该过程的主要方法，辅助的UML¹⁵建模技术，以及在实际软件工程工作中应用统一过程所需要的表示法。

3.6.1 简史

从20世纪80年代到90年代初期，面向对象（Object-Oriented, OO）方法和程序设计语言¹⁶在软件工程领域中获得了广泛的认可。在这一时期，涌现了大量的面向对象分析（Object-Oriented Analysis, OOA）和面向对象设计（Object-Oriented Design, OOD）方法，并提出了通用的面向对象过程模型（类似于本章中介绍的演化模型）。就像软件工程中大多数新的范型一样，每种OOA和OOD方法的追随者们常常为哪种方法最好争论不休，但是没有哪一种方法占据主流。

20世纪90年代早期，James Rumbaugh [RUM91]、Grady Booch [BOO94]和Ivar Jacobson [JAC92]开始研究“统一方法”，他们的目标是结合各自方法中最好的特点，并吸纳其他OO领域专家提出的其他特点（例如，[WIR90]）。他们的成果就是UML——统一建模语言，这种语言包含了大量用于OO系统建模和开发的符号。到了1997年，UML已经变成了面向对象软件开发的工业标准。同时，Rational（瑞理）公司和其他生产商开发了自动化工具以支持UML方法。

UML提供了支持面向对象软件工程实践必要的技术，但它没有提供指导项目团队应用该技术时的过程框架。接下来的几年中，Jacobson、Rumbaugh和Booch建立了统一过程模型，一

¹⁴ 用例（use case）是一种文字描述或模板，从用户的角度描述系统功能和特性。用例由用户来写，并作为创建更为复杂的分析模型的基础。

¹⁵ UML（Unified Modeling Language，统一建模语言），已经成为分析和设计建模中应用最广泛的表示法。它统一了三种最重要的面向对象表示方法。

¹⁶ 如果不熟悉面向对象方法，可参看本书第8章和第9章中的简要介绍，更详细的介绍参看[REE02], [STI01]或[FOW99]。

种用UML进行面向对象软件工程的框架。目前，统一过程和UML广泛应用于各种各样的面向对象项目中。统一过程提出的迭代增量模型能够而且应该能够满足特定的项目需要。

运用UML可以生成一系列工作产品（例如模型和文档）。但软件工程师往往省略这些东西，使开发更灵活，更易于响应变更。
95

3.6.2 统一过程的阶段¹⁷

WebRef

关于统一过程的白皮书可以访问
[www.national.com/
products/rup/
whitepapers.jsp](http://www.national.com/products/rup/whitepapers.jsp)。



UP阶段的目的与本书中定义的通用框架活动的目的类似。

我们讨论了5种通用的框架活动，并认为它们可以用来描述任何软件过程模型。统一过程也不例外。图3-7描述了统一过程（UP）的阶段，并将它们与第2章中讨论的通用活动进行了对照。

UP的起始（inception）阶段包括客户沟通和策划活动。通过与客户和最终用户协作，定义软件的业务需求，提出系统大致的架构，并制定开发计划以保证项目开发具有迭代和增量的特性。该阶段识别基本的业务需求，并初步用“用例”描述每一类用户所需要的主要特征和功能。通常，用例描述了某一角色（例如人，机器，或其他系统）在与软件交互的过程中所执行的一系列操作。用例有助于识别项目边界，并为项目策划提供基础。

此时的架构仅是主要子系统及其功能、特性的试探性概括。随后，体系结构将被细化和扩充成为一组模型，以描述系统的不同视图。策划活动识别各种资源，评估主要风险，定义进度计划，并为用于软件增量开发的各个阶段建立基础。

细化（elaboration）阶段包括用户沟通和通用过程模型的建模活动（参见图3-7）。细化阶段扩展了起始阶段定义的用例，并扩展体系结构以包括了软件的五种视图——用例模型、分析模型、设计模型、实现模型和部署模型。在某些情况下，细化阶段建立了一个“可执行的体系结构基线”[ARL02]，是建立可执行系统的“第一步”（first cut）¹⁸。体系结构基线证明了体系结构的可实现性，但没有提供系统使用所需的所有功能和特性。另外，在细化的最终阶段将评审项目计划以确保项目的范围、风险和交付日期合理。在该阶段对项目计划进行修订。

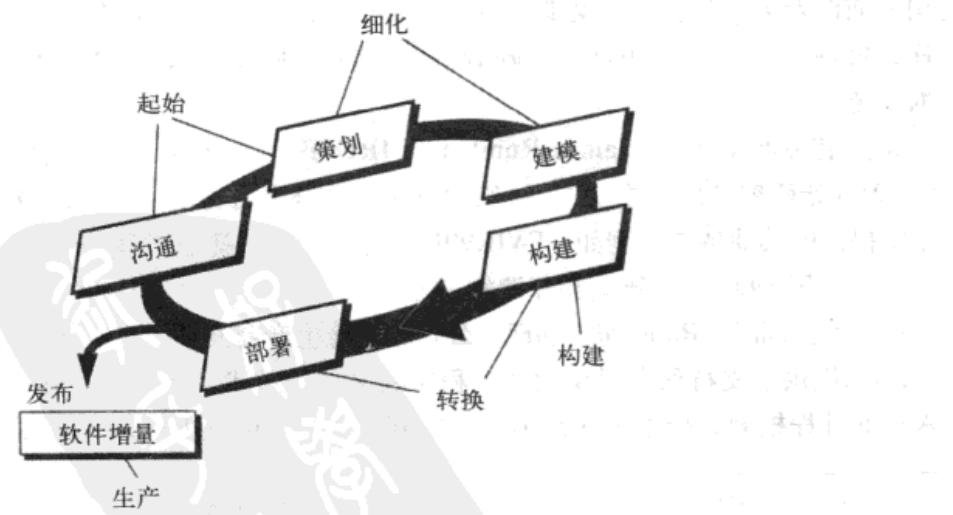


图3-7 统一过程

¹⁷ 统一过程有时也用Rational公司命名，称为Rational统一过程（Rational Unified Process，RUP），该公司是开发和细化该过程的主要投资方，并建立了支持该过程的完整环境（工具及技术）。

¹⁸ 需要指出的是，体系结构基线不是个原型（见3.4.1节），因为它并不被抛弃，而是在下一个UP阶段进一步充实。

WebRef

有关UP的讨论
和说明可参见
www.unified-process.org。

UP的构建（construction）阶段与通用软件过程中的构建活动相同。构建阶段采用体系结构模型作为输入，开发或获取软件构件，使得最终用户能够操作用例。为达到上述目的，需要完善在细化阶段开始的分析和设计模型，以反应出软件增量的最终版本。软件增量（例如发布的版本）所必须要求的特性和功能在原代码中实现。随着构件的实现，要对每一个构件设计并实施单元测试。另外，还实施了其他集成活动（构件组装和集成测试）。用例被用来导出一组验收测试，以便在下一个UP阶段开始前执行。

96

UP的转换（transition）阶段包括通用构建活动的后期阶段以及第一部分通用部署活动。软件被提交给最终用户进行Beta测试¹⁹，用户反馈报告缺陷及必要的变更。另外，软件开发团队创建系统发布所必要的支持信息（如用户手册、问题解决指南及安装步骤）。在转换阶段结束时，软件增量成为可用的发布版本。

UP的生产（production）阶段与通用过程的部署活动一致。在该阶段，监控软件的持续使用，提供运行环境（基础设施）的支持，提交并评估缺陷报告和变更请求。

97

有可能在构建、转换和生产阶段的同时，下一个软件增量的工作已经开始。这就意味着五个UP阶段并不是顺序地进行，而是阶段性地并发进行。

一个软件工程的工作流分布在所有UP阶段。在UP中，工作流类似于任务集（参见第2章定义）。也就是说，工作流识别了完成一个重要的软件工程活动的必要任务，以及在成功完成任务之后所产生的工作产品。需要注意的是，并不是工作流所识别的每一个任务都在所有的项目中应用。软件开发团队根据各自的需要适当地调整过程（动作、任务、子任务及工作产品）。

3.6.3 统一过程工作产品

图3-8说明了UP的4个技术阶段所产生的主要工作产品。在起始阶段，目的是建立项目的整体“愿景”（vision），识别一组业务需求，为软件设想一个业务用例，并定义有可能威胁项目成功的项目风险及业务风险。从软件工程的角度来看，在起始阶段最重要的工作成果是用例模型——一系列用来描述外界用户（人及其他与软件交互的系统）如何与系统交互并从中获益的用例。从根本上说，用例图是一组使用场景，采用标准模板描述，并通过描述交互的前置条件、事件流或场景和后置条件，来说明软件的特性和功能。最初，用例描述业务域层次的需求（即，高层抽象）。但是，随着UP各个阶段的实施，用例模型逐步提炼和细化，并作为创建子任务工作产品的重要输入。在起始阶段，只完成10%~20%的用例模型。细化后，可创建80%~90%的模型。

细化阶段产生了一组工作产品，以进一步细化需求（包括非功能²⁰需求），并产生体系结构描述和初步设计。随着软件工程师开始面向对象分析，首要的目的是定义一组分析类以恰当地描述系统行为。UP分析模型是这一系列活动所产生的工作产品。类及分析包（类的集合）是分析模型的一部分，被逐步细化为设计模型，设计模型识别了设计类、子系统和子系统之间的接口。分析和设计模型都扩展并细化了软件体系结构，是体系结构的演进。另外，细化阶段重新修订风险及项目计划，以保证各项依然有效。

98

¹⁹ Beta测试是受控的测试活动（参见第13章），在此阶段，最终用户操作软件，目的是发现缺陷和不足之处。制定正式的缺陷/不足报告方案，并由软件开发团队评估反馈意见。

²⁰ 不能从用例图中识别的需求。

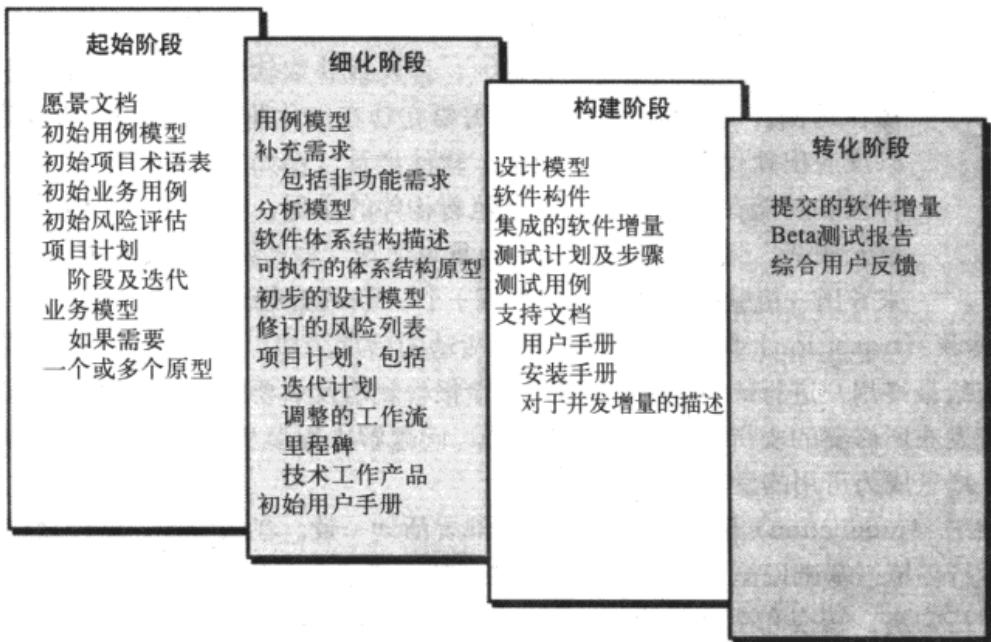


图3-8 在UP每一个阶段产生的主要工作产品

构建阶段产生实现模型，将设计类转化成用以实现系统的软件构件，并产生部署模型将软件构件映射到物理计算环境。最终，测试模型描述了将采用的测试，以保证构建的软件已经适当地实现了用例。

转换阶段产生软件增量，并在用户使用软件的过程中评估工作产品，同时产生了Beta测试的反馈意见以及合格的变更请求。

3.7 小结

惯例软件过程模型已经使用了多年，力图给软件开发带来秩序和结构。每一个传统模型都建议了一种不同的过程流，但都实现了同样的一组通用框架活动：沟通、计划、建模、构建和部署。

瀑布模型建议线形流程的框架活动，这在软件世界里通常与当代软件开发的现实情况不符（例如，持续的变更、演化的系统、紧迫的开发时间）。但瀑布模型确实适用于需求定义清楚且稳定的软件开发。

增量软件过程模型通过一系列的增量发布产生软件。RAD模型是为大型且必须在严格的时间内提交的项目而设计的。

演化过程模型认识到大多数软件工程项目的迭代特性，其设计的目的是为了适应变更。

99 演化模型，例如原型开发及螺旋模型，快速地产生增量的工作产品（或是软件的工作版本）。这些模型可以应用于所有的软件工程活动——从概念开发到长期的软件维护。

基于构件的模型强调构件复用及组装。形式化方法模型提倡采用数学的方法进行软件开发和验证。面向方面的模型目的是解决跨整个软件体系架构的横切关注点。

统一过程模型是一种“用例驱动、以体系结构为核心、迭代及增量”的软件过程框架，由UML方法和工具支持。统一过程是一种增量模型，定义了五个阶段：（1）起始阶段，包括用户沟通和计划活动两个方面，强调定义和细化用例，并将其作为主要模型；（2）细化阶段，

包括用户沟通和建模活动，重点是创建分析和设计模型，强调类的定义和体系结构的表示；(3) 构建阶段，细化设计模型，并将设计模型转化为软件构件实现；(4) 转化阶段，将软件从开发人员传递给最终用户，并由用户完成Beta测试和验收测试；(5) 生产阶段，持续地监控软件的运行，并提供技术支持。

参考文献

- [AMB02] Ambler, S., and L. Constantine, *The Unified Process Inception Phase*, CMP Books, 2002.
- [ARL02] Arlow, J., and I. Neustadt, *UML and the Unified Process*, Addison-Wesley, 2002.
- [BAC97] Bach, J., "Good Enough Quality: Beyond the Buzzword," *IEEE Computer*, vol. 30, no. 8, August 1997, pp. 96–98.
- [BOE88] Boehm, B., "A Spiral Model for Software Development and Enhancement," *Computer*, vol. 21, no. 5, May 1988, pp. 61–72.
- [BOE98] Boehm, B., "Using the WINWIN Spiral Model: A Case Study," *Computer*, vol. 31, no. 7, July 1998, pp. 33–44.
- [BOE01] Boehm, B., "The Spiral Model as a Tool for Evolutionary Software Acquisition," *CrossTalk*, May 2001, available at <http://www.stsc.hill.af.mil/crosstalk/> 2001/05/boehm.html.
- [BOO94] Booch, G., *Object-Oriented Analysis and Design*, 2nd ed., Benjamin Cummings, 1994.
- [BRA94] Bradac, M., D. Perry, and L. Votta, "Prototyping a Process Monitoring Experiment," *IEEE Trans. Software Engineering*, vol. 20, no. 10, October 1994, pp. 774–784.
- [BRO75] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1975.
- [BUT94] Butler, J., "Rapid Application Development in Action," *Managing System Development*, Applied Computer Research, vol. 14, no. 5, May 1994, pp. 6–8.
- [DYE92] Dyer, M., *The Cleanroom Approach to Quality Software Development*, Wiley, 1992.
- [ELR01] Elrad, T., R. Filman, and A. Bader (eds.), "Aspect-Oriented Programming," *Comm. ACM*, vol. 44, no. 10, October 2001, special issue.
- [FOW99] Fowler, M., and K. Scott, *UML Distilled*, 2nd ed., Addison-Wesley, 1999.
- [GIL88] Gilb, T., *Principles of Software Engineering Management*, Addison-Wesley, 1988.
- [GRA03] Gradecki, J., and N. Lesiecki, *Mastering AspectJ: Aspect-Oriented Programming in Java*, Wiley, 2003.
- [GRU02] Grundy, J., "Aspect-Oriented Component Engineering," 2002, <http://www.cs.auckland.ac.nz/~john-g/aspects.html>.
- [HAN95] Hanna, M., "Farewell to Waterfalls," *Software Magazine*, May 1995, pp. 38–46.
- [HES96] Hesse, W., "Theory and Practice of the Software Process—A Field Study and its Implications for Project Management," *Software Process Technology, 5th European Workshop, EWSPT 96*, Springer LNCS 1149, 1996, pp. 241–256.
- [HES01] Hesse, W., "Dinosaur Meets Archaeopteryx? Seven Theses on Rational's Unified Process (RUP)," *Proc. 8th Intl. Workshop on Evaluation of Modeling Methods in System Analysis and Design*, Ch. VII, Interlaken, 2001.
- [JAC92] Jacobson, I., *Object-Oriented Software Engineering*, Addison-Wesley, 1992.
- [JAC99] Jacobson, I., Booch, G., and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
- [JAC99] Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
- [KAU95] Kauffman, S., *At Home in the Universe*, Oxford, 1995.
- [KER94] Kerr, J., and R. Hunter, *Inside RAD*, McGraw-Hill, 1994.
- [KIS02] Kiselev, I., *Aspect-Oriented Programming with AspectJ*, Sams Publishers, 2002.
- [MAR91] Martin, J., *Rapid Application Development*, Prentice-Hall, 1991.
- [McDE93] McDermid, J., and P. Rook, "Software Development Process Models," in *Software Engineer's Reference Book*, CRC Press, 1993, pp. 15/26–15/28.
- [MIL87] Mills, H. D., M. Dyer, and R. Linger, "Cleanroom Software Engineering," *IEEE Software*, September, 1987, pp. 19–25.
- [NIE92] Nierstrasz, O., S. Gibbs, and D. Tsichritzis, "Component-Oriented Software Development," *CACM*, vol. 35, no. 9, September 1992, pp. 160–165.
- [NOG00] Nogueira, J., C. Jones, and Luqi, "Surfing the Edge of Chaos: Applications to Software Engineering," *Command and Control Research and Technology Symposium*, Naval Post

McGraw-Hill, 2001) 的书中介绍了这些传统的模型，并讨论了它们的优点和缺点。Brooks (《The Mythical Man-Month》, second edition, Addison-Wesley, 1995) 在他的书中虽然没有直接讲过程，但是他用一生的学识讲了和过程相关的每一方面。Firesmith和Henderson-Sellers (《The OPEN Process Framework: An introduction》, Addison-Wesley, 2001) 为创建“灵活但有序的软件过程”提出了一个通用的模板，并讨论了过程属性和目的。

Sharpe和McDermott (《Workflow Modeling: Tools for Process Improvement and Application Development》, Artech House, 2001) 介绍了为软件和业务过程建模的工具。Jacobson, Griss 和Jonsson (《Software Reuse》, Addison-Wesley, 1997) 还有McClure (《Software Reuse Techniques》, Prentice-Hall, 1997) 提供了许多关于基于构件开发的有用信息。Heineman和Council (《Component-Based Software Engineering》, Addison-Wesley, 2001) 描述了实现基于构件的系统需要的过程。Kenett和Baker (《Software Process Quality: Management and Control》, Marcel Dekker, 1999) 讨论了质量管理与过程设计之间如何紧密联系。

Ambriola (《Software Process Technology》, Springer-Verlag, 2001), Derniame和他的同事 (《Software Process: Principles, Methodology, and Technology》, Springer-Verlag, 1999)，还有Gruhn和Hartmanis (《Software Process Technology》, Springer-Verlag, 1999) 编辑了与软件过程相关的许多研究和理论的会议论文集。

Jacobson、Booch和Rumbaugh写出了统一过程的第一部著作[JAC99]。Arlow和Neustadt [ARL02]以及Ambler和Constantine[AMB02]的三卷本书提供了大量的补充信息。Kruchten (《The Rational Unified Process》, second edition, Addison-Wesley, 2000) 对统一过程进行了很有价值的介绍。Royce (《Software Project Management: A Unified Framework》, Addison-Wesley, 1998) 详细讲述了如何用统一过程进行项目管理。统一过程的定义性描述由Rational公司开发，

McGraw-Hill, 2001) 的书中介绍了这些传统的模型，并讨论了它们的优点和缺点。Brooks (《The Mythical Man-Month》, second edition, Addison-Wesley, 1995) 在他的书中虽然没有直接讲过程，但是他用一生的学识讲了和过程相关的每一方面。Firesmith和Henderson-Sellers (《The OPEN Process Framework: An introduction》, Addison-Wesley, 2001) 为创建“灵活但有序的软件过程”提出了一个通用的模板，并讨论了过程属性和目的。

Sharpe和McDermott (《Workflow Modeling: Tools for Process Improvement and Application Development》, Artech House, 2001) 介绍了为软件和业务过程建模的工具。Jacobson, Griss 和Jonsson (《Software Reuse》, Addison-Wesley, 1997) 还有McClure (《Software Reuse Techniques》, Prentice-Hall, 1997) 提供了许多关于基于构件开发的有用信息。Heineman和Council (《Component-Based Software Engineering》, Addison-Wesley, 2001) 描述了实现基于构件的系统需要的过程。Kenett和Baker (《Software Process Quality: Management and Control》, Marcel Dekker, 1999) 讨论了质量管理与过程设计之间如何紧密联系。

Ambriola (《Software Process Technology》, Springer-Verlag, 2001), Derniame和他的同事 (《Software Process: Principles, Methodology, and Technology》, Springer-Verlag, 1999), 还有Gruhn和Hartmanis (《Software Process Technology》, Springer-Verlag, 1999) 编辑了与软件过程相关的许多研究和理论的会议论文集。

Jacobson、Booch和Rumbaugh写出了统一过程的第一部著作[JAC99]。Arlow和Neustadt [ARL02]以及Ambler和Constantine[AMB02]的三卷本书提供了大量的补充信息。Kruchten (《The Rational Unified Process》, second edition, Addison-Wesley, 2000) 对统一过程进行了很有价值的介绍。Royce (《Software Project Management: A Unified Framework》, Addison-Wesley, 1998) 详细讲述了如何用统一过程进行项目管理。统一过程的定义性描述由Rational公司开发，可以从www.rational.com在线查看。

因特网上有大量关于软件工程和软件过程的信息。和软件过程有关的因特网链接可以参考SEPA网站<http://www.mhhe.com/pressman>。



第4章 敏捷视角下的过程

要点浏览

概念：敏捷软件工程是哲学理念和一系列开发指南的综合。这种哲学理念推崇让客户满意和软件尽早增量发布；小而高度自主的项目团队；非正式的方法；最小化软件工程工作产品以及整体精简开发。开发方法强调超越设计和分析（尽管并不排斥这类活动）的发布及开发人员和客户之间主动和持续的沟通。

人员：软件工程师和其他项目共利益者（经理、客户、最终用户）共同组成敏捷开发团队，这个团队是自我组织的并掌握着自己的命运。一个敏捷团队能够培养所有参与人员之间的交流与合作。

重要性：孕育了基于计算机系统和软件产品的现代商业环境，正以飞快的节奏不断变化着，敏捷软件工程提出了可用

于特定类型软件和软件项目的不同于传统软件工程的合理方案。事实证明，这一方法可以快速交付成功的系统。

步骤：敏捷开发恰当的称呼应当是“类软件工程”，它保留了基本框架活动：客户沟通、策划、建模、构建、交付和评估，但将其缩减到一个推动项目组朝着构建和交付发展的最小任务集（有人认为这是以牺牲问题分析和方案设计为代价而实现的）。

工作产品：接受敏捷理念的客户和软件工程师有共同的观点：唯一真正重要的工作产品是在合适时间提交给客户的可运行软件增量。

质量保证措施：如果敏捷团队认为过程可行，开发出的可交付软件增量能使客户满意，则表明敏捷方法已经正确实施。

关键概念
敏捷宣言
敏捷建模
敏捷过程模型
敏捷性
敏捷原理
ASD
Crystal
DSDM
XP
FDD
结对编程
政策
重构
Scrum
团队特征

2001年，Kent Beck和其他16位知名软件开发者、软件工程作家以及软件咨询师[BEC01a]（被称为敏捷联盟）共同签署了“敏捷软件开发宣言”。该宣言声明：

我们正在通过亲身实践以及帮助他人实践的方式来揭示更好的软件开发之路，通过这项工作，我们认为：

个体和交互胜过过程和工具

可工作软件胜过宽泛的文档

客户合作胜过合同谈判

响应变化胜过遵循计划

亦即，虽说上述右边的各项很有价值，但我们认为左边的各项具有更大的价值。

一份宣言通常和一场即将发生的破旧立新的政治运动相联系。从某些方面来讲，敏捷开发确实是一场运动。

虽然多年来大家一直都在使用着指导敏捷开发的基本思想，但真正凝聚到一场运动中还是近十年来的事情。本质上讲，敏捷方法¹是为了克服传统软件工程中认识和实践的弱点开发而成的。敏捷开发可以带来多方面好处，但它并不适用于所有的项目、所有的方面、所有的人和所有的情况，它并不完全对立于传统软件工程实践，也不能作为超越一切的哲学理念而用于所有软件工作。

[103] 在现代经济生活中，很难甚至无法预测一个基于计算机的系统（如基于网络的应用）如何随时间推移而演化。市场情况飞快变化，最终用户需求不断变更，新的竞争威胁毫无征兆地出现，因此，在很多情况下我们必须足够敏捷地去响应不断变化、无法确定的商业环境。

难道说认识到现状我们就完全抛弃那些有价值的软件工程原理、概念、方法和工具吗？绝对不是。和其他所有工程原理一样，软件工程也在持续发展着，我们可以通过改进软件工程本身来适应敏捷带来的挑战。

“敏捷：1，其他：0。”

——Tom DeMarco

Alistair Cockburn[COC02a]在他那本发人深省的敏捷软件开发著作中，论证了本书第3章介绍的惯例过程模型中存在的主要缺陷：忘记了开发计算机软件的人员的弱点。软件工程师不是机器人，许多软件工程师在工作方式上有很大差别，在技能水平、主动性、服从性、一致性和责任心方面也有巨大差异。一部分人可以通过书面方式很好地沟通，而有些人则不行。Cockburn论证说：过程模型可以“利用纪律或者宽容来处理人的这一共同弱点[COC02a]”，因而大多数惯例过程模型选择了纪律。他还指出：“不能一致连贯地做同一件事是人性的弱点，因而高度纪律性的方法学非常脆弱[COC02a]。”

要想让过程模型可用，要么必须提供可实现机制来维持必要的纪律，要么必须向软件工程师以某种方式展示宽容。显而易见，宽容实践易于被接受和保持，但是（正如Cockburn所承认）可能产能低下。正像人生中的大多数事情一样，两者必须考虑平衡。

4.1 敏捷是什么

在软件工程工作这个环境下，敏捷是什么？Ivar Jacobson[JAC02]给出一个非常有用的论述：

敏捷已经成为当今描述现代软件过程的时髦用词。每个人都是敏捷的，敏捷团队是能够适当响应变化的灵活团队。变化就是软件开发本身，软件构建有变化、团队成员在变化、使用新技术会带来变化，各种变化都会对开发的软件产品以及项目本身造成影响。我们必须接受“支持变化”的思想，它应当根植于软件开发中的每一件事中，因为这是软件的心脏与灵魂。敏捷团队意识到软件是团队中所有人共同开发完成的，这些人的个人技能和合作能力是项目成功的关键所在。

在Jacobson的观点中，普遍存在的变化是敏捷的基本动力，软件工程师必须加快步伐以适应Jacobson所描述的快速变化。

“敏捷是动态的、内容独特的、勇于接受变化和面对成长的。” ——Steven Goldman等

¹ 敏捷方法有时也指轻量级或者精简的方法。



不要误认为敏捷能赋予你随意操作、做出拙劣产品的权利。过程是需要的，而纪律是必需的。

但是，敏捷不仅仅是有效地响应变化，它还包含着对本章开头所提宣言中哲学观念的信奉。它鼓励能够在组员之间、技术和商务人员之间、软件工程师和经理之间进行更便利沟通的团队结构和协作态度，强调可运行软件的快速交付而不是中间产品（这并不总是好事情），将客户作为开发组成员以消除持续、普遍存在于多数软件项目中的“区分你我”的态度，意识到在不确定的世界里计划是有局限性的，它必须是可以调整的。

敏捷联盟[AGI03]为希望达到敏捷的人们定义了12条原则：

1. 我们最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。
2. 即使在开发的后期，也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。
3. 经常交付可工作软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。
4. 在整个项目开发期间，业务人员和开发人员必须天天都在一起工作。
5. 围绕受激励的个人构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。
6. 在团队内部，最富有效果和效率的信息传递方法是面对面交谈。
7. 可工作软件是进度的首要度量标准。
8. 敏捷过程提倡可持续的开发速度。责任人（sponsor）、开发者和用户应该能够保持一种长期、稳定的开发速度。
9. 不断地关注优秀的技能和好的设计会增强敏捷能力。
10. 简单——使不需要的工作最大化的艺术——是必要的。
11. 好的架构、需求和设计出自于自组织团队。
12. 每隔一定时间，团队会反省如何才能更有效地工作，并相应调整自己的行为。

敏捷可用于任何软件过程，实现要点是将软件过程设计为如下方式：允许项目团队调整并合理安排任务，理解敏捷开发方法的易变性并制定计划，精简并维持最基本的工作产品，强调增量交付策略，快速向客户提供适应产品类型和运行环境的可运行软件。

4.2 敏捷过程是什么

任何一个敏捷过程都可以由所强调的三个关键假设而识别出来，这三个假设可适用于大多数软件项目：

1. 提前预测哪些需求是稳定的和哪些需求会变化非常困难。同样，预测项目进行中客户优先级的变化也很困难。
2. 对很多软件来说，设计和构建是交错进行的。事实上，两种活动应当顺序开展以保证通过构建实施来验证设计模型，而在通过构建验证之前很难估计应该设计到什么程度。
3. 从制定计划的角度来看，分析、设计、构建和测试并不像我们所设想的那么容易预测。

WebRef
关于敏捷过程的文章可参见：
www.aanpo.org/articles/index

给出这三个假设，同时也就提出一个重要的问题：如何建立能解决不可预测性的过程？正如前文所述，答案就在于过程（对于飞快变化的项目、技术条件）的自适应性。也就是说，敏捷过程必须具有自适应性。

但是原地踏步式的连续适应性变化收效甚微，因而，敏捷软件过程必须增量地适应。为了达到这一目的，敏捷团队需要客户的反馈（以做出正确的适应性改变），可执行原型或部分实现的可运行系统是客户反馈的最有效媒介。

因此，应当使用增量式开发策略，必须在很短的时间间隔内交付软件增量（可执行原型或部分实现的可运行系统）来适应（不可预测的）变化的步伐，这种迭代方法允许客户：周期性地评价软件增量，向软件项目组提出必要的反馈，影响能够接受反馈的过程的适应性变化。

“对于开发过程及其产品本身，快速反馈是不可替代的。”

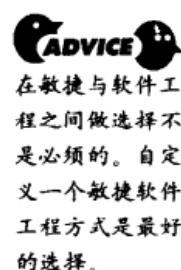
—Martin Fowler

4.2.1 敏捷开发的立场

将敏捷软件开发作为许多传统软件工程的对立面，它们在优越性和适用性方面存在着许多（有时是激烈的）争论。在表达对敏捷拥护者阵营（“敏捷者”）的感想时，Jim Highsmith [HIG02a]（开玩笑地）说明了他那颇为极端的观点：“传统方法学家陷入了误区，乐于生产完美的文档而不是满足业务需要的可运行系统。”而在表述（同样是玩笑性质的）他对传统软件工程阵营的立场时，他则给出完全相反的观点：“轻量级方法者或者说敏捷方法学家是一小撮自以为了不起的黑客，他们妄图将其手中的玩具软件放大到企业级软件而制造出一系列轰动。”

像所有的软件技术争论一样，这场方法学之争有滑向派别之战的危险。一旦争吵发生，理智的思考就消失了，信仰而不是事实主导着各种决定。

没有人反对敏捷。而真正的问题在于“什么是最佳实现途径”？同等重要的还有，如何构建满足用户当前需要的软件，同时展示具有能满足客户长期需求的扩展能力？



对这两个问题，还没有绝对正确的答案。即便在敏捷学派内部，针对敏捷问题，也提出了很多有细微差异的过程模型（见4.3节），每个模型内有一组“想法”（敏捷者们不愿称其为工作任务），显现出和传统软件工程的显著差异。同时，许多敏捷概念是从优秀的软件工程概念修正而来的。其结果是：兼顾两派的优点则双方都能得到很多好处，而相互诽谤则两者都将一无所获。

感兴趣的读者可以参看[HIG01], [HIG02a]和[DEM02]，这些文献针对很多重要技术和方针问题给出了饶有趣味的总结。

107

4.2.2 人的因素

敏捷软件开发的拥护者们花费了很多精力强调“人的因素”在成功敏捷开发中的重要性。正如Cockburn和Highsmith[COC01]所说：“敏捷开发关注个人的才智和技巧，根据特定人员和团队来塑造过程。”这一描述的关键在于“构造可以满足人员及团队需求的过程模型”，而非其他可选的过程模型²。

“对一个团队刚刚够用（的过程）对其他团队要么不够用，要么太多。”

—Alistair Cockburn

如果敏捷开发团队成员希望努力维护所使用过程的特性，则该团队成员及团队本身必须具备以下一些特点：

基本能力。同在传统软件工程中一样，在敏捷开发中，能力一词包含了个人内在才能、特定的软件相关技能以及对所选过程的全局知识。关于过程技能和知识可以而且应该教给敏

² 很多成功的软件工程组织也认识到这个事实，因而忽略他们所选择的过程模型。

捷团队的每一位成员。

 有效的软件
团队，其成
员必须具备哪
些显著特点？

共同目标。虽然敏捷团队成员能为项目执行不同的任务，拥有不同的技能；但是，所有人必须瞄准同一个目标，即在承诺的时间内向客户提交可运行的软件增量。为了实现这一目标，项目组还应当做出或大或小的连续适应性变化，以使过程更适合团队使用。

精诚合作。抛开过程而言，软件工程就是在项目组沟通中评估、分析和使用信息；产生能够帮助客户和其他人了解项目组工作的信息；构建对客户具有业务价值的软件和相关数据库等信息。为了实现这些任务，项目组成员之间，项目组与客户以及项目组与业务经理之间必须精诚合作。

108 决策能力。包括敏捷团队在内，任何一个好的软件项目组必须有能够掌握自身命运的自由。这预示着应当赋予项目组在技术和项目问题上的自主决策权。

模糊问题解决能力。软件项目经理应当认识到：敏捷项目组被迫不断面对不确定的事情，被迫不断和变化作斗争。有时，项目组不得不接受今天正在解决的问题明天变得根本不需解决这样的现实，然而，今后的项目将会从任何解决问题的活动（包括解决错误问题的活动）中获益。

相互信任和尊重。敏捷团队必须成为DeMarco和Lister[DEM98]所说的具有凝聚力的团队（参见第21章），这样的团队展现出的相互信任和尊重使其形成一个强有力组织，确保整体的实力大于各部分实力之和[DEM98]。

 **POINT**
自我组织团队
根据所从事的
工作来安排。
团队自行分派
任务，制定计
划以完成任务。

自我组织。自我组织在敏捷开发中具有三重含义：（1）敏捷团队组织自身以完成工作；（2）团队组织最能适应当前环境的过程；（3）团队组织最好的进度安排以完成软件增量交付。自我组织具有一些技术上的好处，但是更为重要的是它能促进合作，鼓舞士气。实际上，这也就是项目组的自我管理。Ken Schwaber[SCH02]在他的著作中强调以下事情：“团队确定他们预期能在迭代内完成多少工作，并承担这些工作。没有什么能比让别人来分派任务更让团队感到沮丧的，也没有什么能让自己负责以履行承诺更让团队倍感鼓舞了。”

4.3 敏捷过程模型

软件工程的历史是由散乱着的几十个废弃的过程描述和方法学、建模方法和表示法、工具以及技术所构成，每一个都是轰轰烈烈地冒出来，接着又被新的（期望是）更好的所替代。随着敏捷过程模型的大范围推广，每一种模型都在争取得到软件开发界的认可，敏捷运动正在遵循着同样的历史步伐³。

“我们的同行对待方法学就像14岁左右的孩子们对待衣服。”

——Stephen Hawrysh和Jim Ruprecht

109

³ 这不是一件坏事。在一种或多种方法及模型被当做事实标准接受之前，必须努力奋斗以获得软件工程师的青睐和认可。优胜者成为最佳实践，而失败者要么彻底消失，要么融入优胜者的模型中。

在以下的几节中，我们将简要介绍几个敏捷过程模型，这些方法（在理论和实践上）都有许多相似之处，我们试图通过强调每种方法的特点来展现其与众不同之处。一个重要的说明是，所有敏捷方法（或多或少地）都遵循敏捷软件开发宣言以及4.1节中提到的那些原则。

4.3.1 极限编程

WebRef

一篇关于XP规则的优秀的综述
请参见：
www.extremeprogramming.org/rules.html。

虽然极限编程（eXtreme Programming, XP）相关的思想和方法最早出现于20世纪80年代后期，但Kent Beck[BEC99]具有开创意义的著作出版于1999年，其后Jeffries等人[JEF01]的书中讲述了XP技术细节，之后Beck和Fowler[BEC01b]关于XP计划的著作给出了这一方法的详细描述。

XP使用面向对象方法作为推荐的开发范型（见本书第二部分）。XP包含了策划、设计、编码和测试4个框架活动的规则和实践。图4-1描述了XP过程，并指出与各框架活动相关的关键概念和任务。以下段落将总结一下XP关键活动。

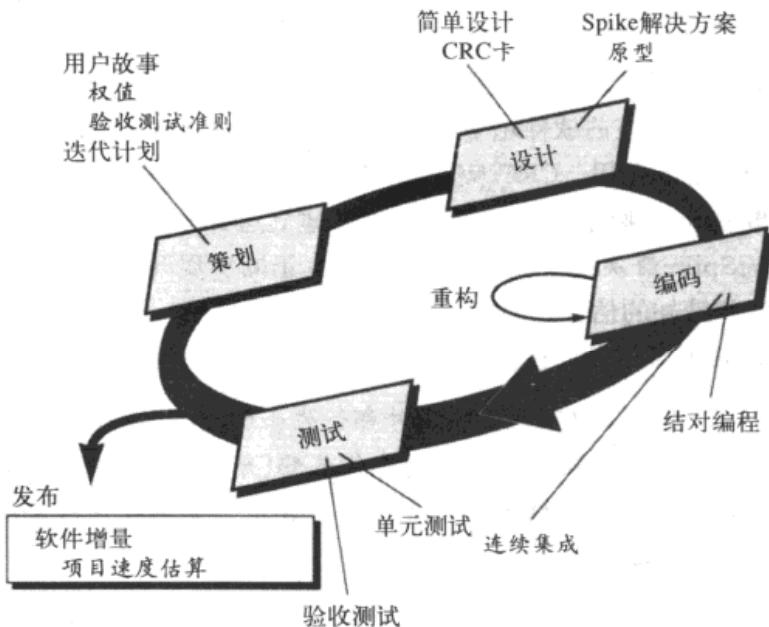


图4-1 极限编程过程



什么是XP中的“故事”？

WebRef

一个有价值的
XP“计划游戏”
参见：c2.com/cgi/wiki?planningGame。

策划。策划活动开始于建立一系列描述待开发软件必要特征与功能的“故事”（也称为“用户故事”）。每个故事（类似于第7、8章讲述的用例）由客户书写并置于一张索引卡上，客户根据对应特征或功能的全局业务价值标明权值（即优先级）⁴。XP团队成员评估每一个故事并给出以开发周数为度量单位的成本。如果某个故事的成本超过了3个开发周，将请客户把该故事进一步细分，重新赋予权值并计算成本。需要说明的是新故事可以在任何时刻书写。

客户和XP团队共同决定如何把故事分组并置于XP团队将要开发的下一个发行版本中（下一个软件增量）。一旦形成关于一个发布版本的基本承诺（就包括的故事、发布日期和其他项目事项达成一致），XP团队将以下三种方式

110

⁴ 一个故事的权值取决于其他相关故事的存在。

之一对有待开发的故事进行排序：(1) 所有选定故事将在几周之内尽快实现；(2) 具有最高价值的故事将移到进度表的前面并首先实现；(3) 高风险故事将首先实现。

项目的第一发行版本（也称为一个软件增量）发布之后，XP团队计算项目的速度。简言之，项目速度是第一个发行版本中实现的用户故事个数。项目速度将用于：(1) 帮助建立后续发行版本的发布日期和进度安排；(2) 确定是否对整个开发项目中的所有故事有过分承诺。一旦发生过分承诺，则调整软件发行版本的内容或者改变最终交付日期。

在开发过程中，客户可以增加故事，改变故事的权值，分解或者去掉故事。接下来由XP团队重新考虑所有剩余的发行版本并相应修改计划。

“极限编程是基于简洁、交流、反馈和勇气的软件开发法则。”

——Ron Jeffries

设计。XP设计严格遵循KIS (Keep It Simple, 保持简洁) 原则，即使用简单而不是复杂的表述。另外，设计为故事提供不多也不少的实现原则，不鼓励额外功能性（因开发者假定以后会用到）设计⁵。

XP鼓励使用CRC卡（第8章）作为有效机制，在面向对象语境中考虑软件、CRC（类责任协作者）卡的确定，组织和当前软件增量相关的对象和类⁶。XP团队使用类似于第8章（8.7.4节）描述的过程来管理设计运用。CRC卡也是作为XP过程一部分的唯一的设计工作产品。

如果在某个故事设计中碰到困难，XP推荐立即建立这部分设计的可执行原型，实现并评估设计原型（被称为Spike解决方案），其目的是在真正的实现开始时降低风险，对可能存在设计问题的故事确认其最初的估计。

XP鼓励既是构建技术又是设计技术的“重构”，Fowler[FOW00]描述“重构”如下：

WebRef

重构技术和工具可以在以下站点找到：www.refactoring.com。

重构是以不改变代码外部行为而改进其内部结构的方式来修改软件系统的过程。这是一种净化代码[并修改/简化内部设计]以尽可能减少引入错误的严格方法。简而言之，重构就是在编码完成之后改进代码设计。

因为XP设计实际上不使用符号并且几乎不产生工作产品，一旦生成除CRC卡和“Spike解决方案”之外的工作产品，则设计会被当作是可以并且应当在构建过程中连续修改的暂时人工产品。重构的目的是控制那些由于提出“可以根本改进设计”的小修改而造成的（代码）改动[FOW00]。然而应当注意的是，重构所需的工作量随着应用软件规模的增长而急剧增长。

WebRef

关于XP的有用信息可以在以下站点获得：www.xprogramming.com。

XP的中心观念是设计在编码开始前后同时发生，重构意味着设计随着系统的构建而连续进行。实际上，构建活动本身将给XP团队提供关于如何改进设计的指导。

编码。XP推荐在故事开发和基本设计完成之后，团队不应直接开始编码，而是开发一系列用于检测本次（软件增量）⁷发布的包括所有故事的单元测试，一旦建立起单元测试，开发者就可以更集中精力于必须实现的内容以通过单

⁵ 虽说复杂的设计符号和术语应该简化，但每一个软件工程方法都要遵循自己的设计准则。

⁶ 面向对象类将在本书第8章及整个第二部分详细讨论。

⁷ 这种方法类似于学生在开始学习之前获知考试题目，从而使得学生很方便地将注意力仅仅集中于那些将要提出的问题。

元测试。不需要加任何额外的东西（KIS，保持简洁）。一旦编码完成，就可以立即完成单元测试，由此可向开发者提供即时的反馈。

什么是结对编程？

XP推荐/建议两个人面对同一台计算机共同为一个故事开发代码。这一方案提供了实时解决问题（两个人总比一个人强）和实时质量保证的机制，同时也使得开发者能集中精力于手头的问题。实施中不同成员担任的角色略有不同，例如，一名成员考虑特定设计的详细编码实现，而另一名成员确保编码遵循特定的标准（XP要求之一），生成的代码符合该故事的接口（界面）设计。[112]

结对的两人完成其工作，所开发代码和其他工作集成。有些情况下，这种集成工作由集成团队按日实施，还有一些情况下，结对者自己负责集成，这种“连续集成”策略有助于避免兼容性和接口问题，建立能及早发现错误的“冒烟测试”环境（第13章）。

测试。正如已经指出的，在编码开始之前建立单元测试⁸是XP方法的关键因素。所建立的单元测试应当使用一个可以自动实施的框架（因此易于并可重复执行），这种方式支持代码修改之后即时的回归测试策略（会经常发生，为XP提供重构支持）。

一旦将个人的单元测试组织到一个“通用测试集”[WEL99]，每天都可以进行系统的集成和确认测试。这可以为XP团队提供连续的进展指示，也可在一旦发生问题的时候及早提出预警。Wells[WEL99]指出，“每个小时修改一些小问题，要比仅仅在最后截止期之前修正大问题要节省时间。”



XP验收测试根据
用户故事得到。

XP验收测试，也称为客户测试，由客户确定，将着眼于客户可见的、可评审的系统级的特征和功能，验收测试根据本次软件发布中所实现的用户故事而确定。

SAFEHOME

考虑敏捷软件开发

[场景] Doug Miller的办公室。

[人物] Doug Miller，软件工程经理；Jamie Lazor和Vinod Raman，软件团队成员。

[对话]

(敲门)

Jamie: Doug，有时间吗？

Doug: 当然，Jamie，什么事？

Jamie: 我们考虑过昨天讨论的过程了……就是我们打算为这个新的SafeHome项目选什么过程。

Doug: 哦？

Vinod: 我和在其他公司的一位朋友聊，他告诉我极限编程。那是一种敏捷过程模型，听说过吗？

Doug: 听说过，有好也有坏。

⁸ 单元测试将在第13章中详细讨论，集中于单个软件构件进行构件接口、数据结构和功能的检查，其目标是发现各构件本身错误。

Jamie: 对，看起来很适合我们。可以使软件开发更快，用结对编程来达到实时质量检查……我想这一定很酷。

Doug: 它确实有很多实实在在的好主意。比如，我喜欢其中的结对编程概念，还有共利益者参加项目组的想法。

Jamie: 哦？你是说市场部将和项目组一起工作？

Doug（点头）：他们也是共利益者，不是吗？

Jamie: 哇，他们会5分钟就提出变更。

Vinod: 不要紧。我的朋友说XP项目有包容变更的方法。

Doug: 所以你俩认为我们应当使用XP？

Jamie: 绝对值得考虑。

Doug: 我同意。既然我们选择了增量模型方法，那就没有理由不利用XP带来的好处。

Vinod: Doug，刚才你说“有好处也有坏处”，坏处是什么？

Doug: 我不喜欢XP不重视分析和设计……简而言之就是直接编码。（团队成员相视而笑。）

Doug: 那你们同意用XP方法吗？

Jamie:（代表二人说）我们干的就是编码！

Doug（大笑）：没错，但我希望看到你花少量时间编码和重新编码，而花多一点时间分析我们应当做什么并设计一个可用系统。

Vinod: 或许我们可以二者兼用，带有一定纪律性的敏捷。

Doug: 我想我们能行，Vinod，实际上我坚信我们能行。

4.3.2 自适应软件开发

自适应软件开发（Adaptive Software Development, ASD）是由Jim Highsmith[HIG00]提出的，它可作为构建复杂软件和系统的一项技术，其基本概念着眼于人员协作和团队自我组织。Highsmith[HIG98]在他的著作中讨论了这一点：

自我组织是复杂自适应系统的一个重要特点，类似于收集人们创意的容器，当烦人的问题出现时产生创造性能量。自我组织发生在个体、独立代理（体内的细胞、生态系统中的物种、特色团队的开发者）合作建立紧急、重要的结果的时候，重要结果超越任何单独个体的能力。例如，大脑中单个神经元并不能完成感知过程，而是共同协作才能够产生感知。我们曾试图将这种共同作用现象看作是偶然发生，或者至少是无规律和不可靠的，自我组织研究表明这种观点是错误的。

WebRef

ASD的有用资源可以在如下站点找到：www.adaptivesd.com

Highsmith认为一个基于协作的敏捷、自适应性开发方法是“我们复杂交互作用中如同纪律和工程的秩序之源”，他给ASD“生命周期”（图4-2）的定义包含思考、协作和学习三个阶段。

思考。思考过程中，启动项目并完成自适应循环策划。自适应循环策划通过使用项目启动信息——客户任务描述、项目约束（如发布日期或用户描述）和基本需求——来确定项目所需的一系列软件增量发布循环⁹。

⁹ 注意，自适应循环计划可以并很可能随着项目和商业情况的变化而进行适应性改变。

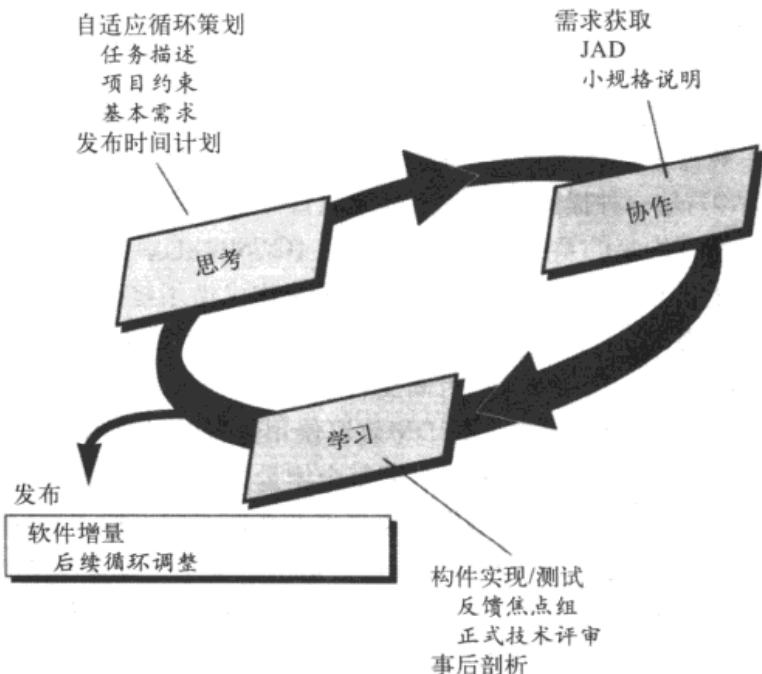


图4-2 自适应软件开发

ASD自适应
循环的特征
是什么？

ADVICE
只有在抛弃了
“我们和他们”的
观念后，有效的
协作才会出现。

协作。受激励的人员以超越其聪明才智和独创的方式共同工作，协作方法是所有敏捷方法中不断重现的主旋律。虽说沟通是协作的一部分，但协作并不容易。它并不是简单的沟通，它也不只是一种团队工作，真正的协作离不开稳定的团队（第21章）。但这并不排斥个人主义，因为个人的创造力在协作思考中起着重要作用。更重要的是信任，一起工作的人们必须相互信任，才能够：（1）毫无恶意地做出评论；（2）毫无怨言地相互帮助；（3）尽最大努力工作；（4）拥有解决手头工作的技能；（5）以能导致有效行动的方式沟通问题和事务。

“我喜欢倾听，我从细心倾听中学到很多东西，而很多人从不倾听别人。”

——Ernest Hemingway

115

学习。当ASD团队成员开始开发作为自适应循环一部分的构件时，其重点是朝着完成循环的方向学习尽可能多的东西。事实上，Highsmith[HIG00]认为软件开发人员常常高估自己（对技术、过程和项目）的理解力，这样的学习将帮助他们改进其真正的理解水平。ASD团队通过以下三种方式学习：

1. 焦点组。客户和/或最终用户对已发布软件增量提供反馈，这些反馈给出产品是否满足业务需求的直接指示。
2. 正式技术评审。ASD团队评审他们开发的软件构件，以便在工作进展的同时提高质量、学习知识。
3. 事后剖析。ASD团队自我反省，着眼于自身的表现和过程（以学习和方法改进为目的）。

需要指出的是：ASD理念无论其使用什么过程模型都具有重要的存在价值。ASD整体上强调软件项目团队具有自我组织的动态性、人与人的协作、个人以及团队的学习，从而使团

队具有高成功的可能性。

4.3.3 动态系统开发方法

WebRef

DSDM的有用资源可参见：
www.dsdm.org。

动态系统开发方法（Dynamic System Development Method, DSDM）[STA97]是一种提供“通过在可控项目环境中使用增量原型开发模式完全满足对时间有约束的系统的构建和维护”[CCS02]的敏捷软件开发方法。某些方面类似于第3章讨论的RAD过程，DSDM建议借用修改版Pareto（佩瑞多）原则的哲学观念。这种情况下，如果交付整个应用系统需用100%时间，那么80%的应用系统可以用20%的时间交付。

像XP和ASD一样，DSDM建议使用迭代软件过程。然而，DSDM方法的每一个迭代都遵循80%原则，即每个增量只完成能够保证顺利进入下一增量的工作，剩余的细节则可以在知道更多业务需求或者提出并同意变更之后完成。

DSDM协会（www.dsdm.org）是一个成员公司集体充当其方法管理者的全球性组织。协会定义了一个称为DSDM生命周期的敏捷过程模型。该生命周期定义了3个不同的迭代循环，前面还加了两个生命周期活动：

可行性研究——建立要开发应用的业务需求和相关约束，并评估该应用采用DSDM过程是否可行。
116

业务研究——建立应用系统提供业务价值所需要的功能和信息需求；同时，确定基本的应用系统架构并识别软件的可维护性需求。

功能模型迭代——为客户开发一系列证明其功能的增量原型（注意：所有DSDM原型都倾向于演化为可发布应用）。这一迭代的意图是通过用户使用原型系统诱导反馈信息以获取额外的需求。

设计和构建迭代——在功能模型迭代中，重新构建原型以确保每一个原型都以工程化方式实现，并能为最终用户提供可操作的业务价值。有些情况下，功能模型迭代、设计和构建迭代可同步进行。

实现——将最终软件增量（一个可操作的原型）置于可操作环境。应当注意：（1）增量不见得100%完成；（2）增量完成之后可能需要改变。在这两种情况下，DSDM开发转向功能模型迭代继续进行。

DSDM和XP可以结合使用，这种组合方法用具体实践（XP）定义固定的过程模型，这些具体实践是构建软件增量所必需的。另外，ASD协作和自我组织团队的概念也可运用于这种组合过程模型。

4.3.4 Scrum

Scrum（得名于橄榄球比赛¹⁰）是Jeff Sutherland和他的团队在20世纪90年代早期发展的一种敏捷过程模型，近年来，Schwaber和Beedle[SCH01]对其做了进一步的发展。Scrum原则[ADM96]和敏捷宣言一致：

- 组织小型团队以达到“沟通最大化，负担最小化，非语言描述、非形式化知识”。
- 过程对技术和业务变化必须具有适应性，以“保证制造具有最好可能的产品”。

¹⁰ 一组队员围绕着球构成圆圈共同努力（有时具有暴力性）将球按到地上。

- 过程生产频繁发布“可检查、可调整、可测试、可文档化、可构建”的软件增量。
- 开发工作和开发人员划分为“清晰的、低耦合的部分或包”。
- 坚持在产品构建过程中进行测试和文档化。
- Scrum过程提供“在任何需要的情况下都能完成产品的能力（因为竞争对手的产生，因为企业需要现金，因为用户/客户需要功能，因为这就是所承诺的……” [ADM96]。

WebRef

有关Scrum的有用信息可在以下站点找到：
www.controlchaos.com。

应用Scrum原则指导过程（过程由“需求、分析、设计、演化和交付”等框架性活动组成）中的开发活动。每一个框架活动中，发生于一个过程模式（在以下段落中讨论）中的工作任务称为一个冲刺。修改并确定冲刺的工作产品（每一个框架活动中的冲刺的数目根据产品复杂度和规模大小而有所不同）以适应手头的问题，该工作产品经常为Scrum团队实时修改。Scrum过程的全局流程如图4-3所示。

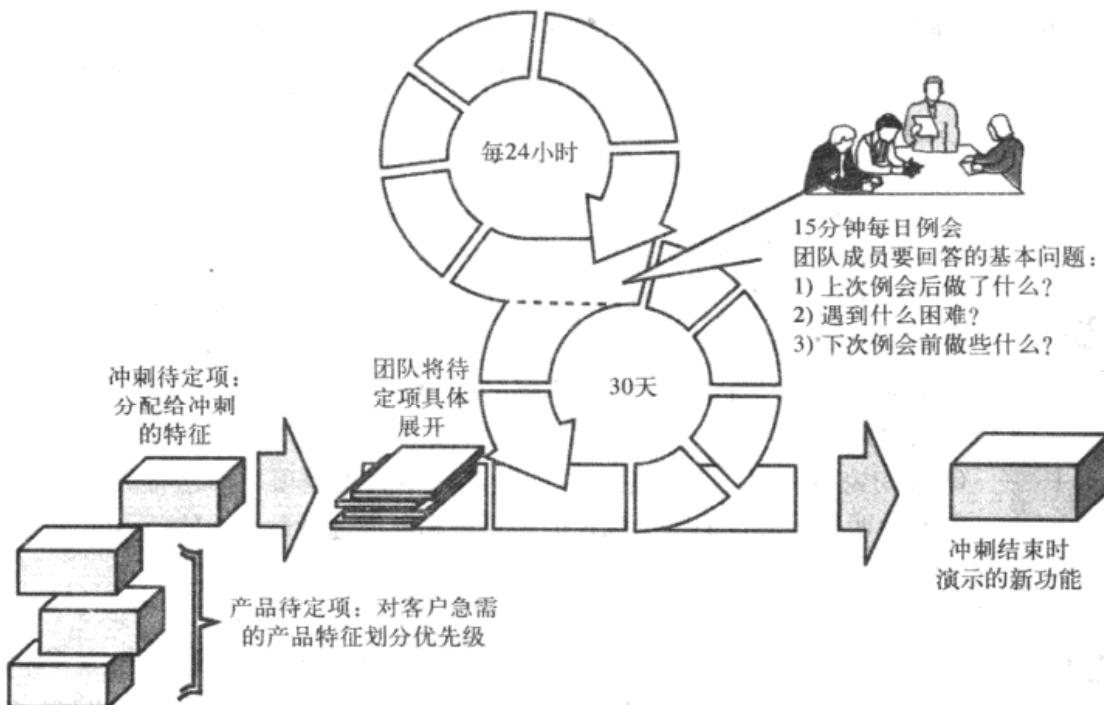


图4-3 Scrum过程流

“Scrum使得我们可以构建更具柔性的软件。”

——Mike Beetle等

KEY POINT
Scrum由一系列强调项目优先级、分离的工作单元、沟通、频繁的客户反馈等过程模式组成。

Scrum强调使用一系列“软件过程模式”[NOY02]，这些过程模式被证实 在时间紧张的、需求变化的和业务关键的项目中是有效的。每一个过程模式 定义一系列开发活动：

待定项（backlog）——一个能为用户提供业务价值的项目需求或特征的优 先级列表。待定项中可以随时加入项目（这就是变更的引入）。产品经理根据 需要评估待定项并更新优先级。

冲刺（sprint）——由一些工作单元组成，这些工作单元是完成待定项中 定义的需求所必需的，必须能在预定的时间段内（一般情况下为30天）完成。 在冲刺过程中，要冻结该冲刺所涉及的具体待定项（也就是说，冲刺过

程中不允许有变更)。因此,冲刺给开发团队成员提供一个短期但稳定的环境。

Scrum例会——每天由Scrum团队参加的短会(一般情况为15分钟),会上所有成员要回答三个问题[NOY02]:

- 上次例会后做了什么?
- 遇到什么困难?
- 下次例会前做些什么?

团队领导,也称为Scrum主持人,主持会议并评价每个团队成员的表现。Scrum会议帮助团队尽早发现潜在的问题。同时,每日例会导致“知识社会化”[BEE99]并进一步促进自我组织团队的建设。

演示——向客户交付软件增量,使用户可以试用并评价所实现的功能。一个重要的需提醒大家的是,演示不需要包含所有计划的功能,但是该时间段内的可交付功能必须完成。

Beedle和他的同事[BEE99]在他们所发表的关于这些模式的综合讨论中说:“Scrum预先假定混乱的存在……”。Scrum过程模式保证软件开发团队在无法消除不确定的世界里能成功地工作。

4.3.5 Crystal

Alistair Cockburn[COC02a]和Jim Highsmith[HIG02b]建立了Crystal¹¹敏捷方法系列,其目的是发展一种提倡“机动性的”软件开发方法,Cockburn将软件开发刻画为:“一种资源有限、合作完成的发明和交流活动,其首要目标是交付有用、可工作的软件,其第二目标是为下一次行动做准备”[COC02b]。

为实现机动性,Cockburn和Highsmith定义了一系列方法学,它们包含具有共性的核心元素,每一个都含有独特的角色、过程模式、工作产品和实践。Crystal家族实际上是一组经过证明、对不同类型项目非常有效的敏捷过程。它的发明使得敏捷团队可以根据其项目和环境选择最合适的Crystal家族成员。

WebRef

关于Crystal的综合讨论可在以下站点找到:
www.crystalmethodologies.org。

119

WebRef

有关FDD的各种文章和描述请参见:www.thecoad-letter.com。

4.3.6 特征驱动开发

特征驱动开发(Feature Driven Development, FDD)最初由Peter Coad及其同事[COA99]作为面向对象软件工程实用过程模型而构思的。Stephen Palmer和John Felsing[PAL02]扩展并增强了Coad的工作,描述了一个可用于中、大型软件项目的适应性敏捷过程。

在FDD环境中,特征“是可以在2周或更短时间实现的具有客户价值的功能”[COA99]。强调特征定义是为了如下好处:

- 特征是小块可发布功能,用户可以方便地描述、轻松地理解其相互关系,更好地评审以发现歧义性错误和遗漏。
- 特征可以组织为具有层次关系的业务相关分组。
- 特征很小,其设计、代码都可以很容易、很有效地检查。
- 项目计划、进度和跟踪都由特征层次驱动,而不是可任意调整的软件工程任务集。

Coad及其同事[COA99]建议使用以下模板定义特征:

<action> the <result> <by | for | of | to> a(n) <object>

¹¹ Crystal(水晶)的名字是从地质学水晶的特征而来,每一种水晶都有自己的颜色、形状和硬度。

这里的<object>是“一个人、地点或事件（包括角色、时刻或时间间隔或者类似类别—项的描述）”。例如，一个电子商务应用的特征可能如下所示：

Add the product to a shopping cart.

Display the technical-specifications of a product.

Store the shipping-information for a customer.

一个特征集将相关特征分在一个业务相关的类别中，定义[COA99]如下：

<action><-ing> a(n) <object>

例如，*Making a product sale*是一个特征集，包含上面提到的及其他特征。

FDD方法定义五种“协作”[COA99]框架活动（FDD中称为“过程”），如图4-4所示。

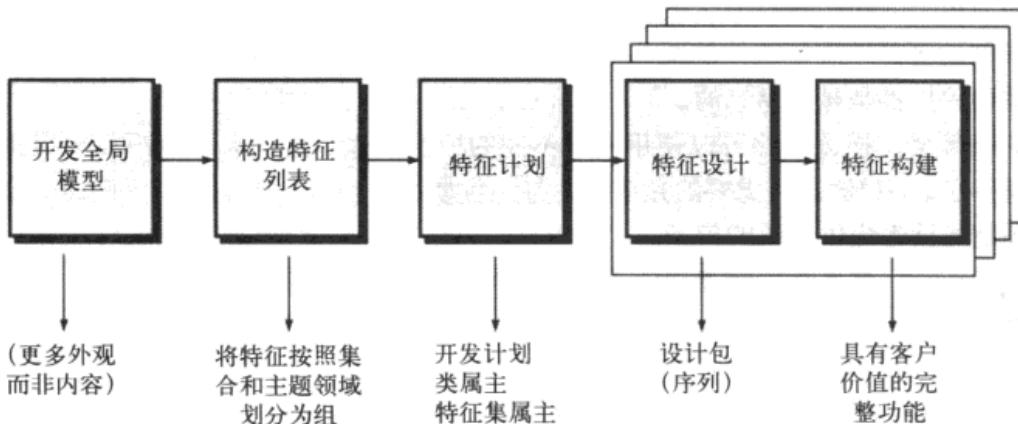


图4-4 特征驱动开发[COA99] (经过授权)

和其他敏捷方法相比，FDD更加强调项目管理原则和技术。随着项目规模和复杂度增长，项目管理常常是不充分的，对于开发者、管理者和客户而言，理解项目状态都非常必要（已经完成了什么，遇到了什么问题），如果截止期压力很大，则确定软件增量（特征）是否能如期完成非常重要。FDD在特征设计和实现阶段定义了6个里程碑以达到上述目标，分别为：“设计走查、设计、设计检查、编码、代码检查、促进构建”[COA99]。

4.3.7 敏捷建模

WebRef

有关敏捷建模的综合信息可在以下站点找到：
www.agilemodeling.com。

很多情况下，软件工程师必须构建大型的、业务关键的系统，这种系统的范围和复杂性必须通过建模方式保证以下事项：(1) 所有参与者可以更好地理解要做什么；(2) 有效地将问题分解给要解决它的人；(3) 对系统工程和构件过程的每一步进行质量评价。

过去的30多年来，人们提出了许多用于分析和设计（架构和构件层面）的软件工程建模方法和符号，这些方法有很重要的价值，但事实证明它们都很难使用也很难维持（尤其在跨多个项目时），部分问题则是建模方法的“负担”，这里的负担指所需符号的数量、所建议采用模型形式化的程度、用于大型项目时模型的大小和变化发生时维护的难度。就算没有其他原因，仅仅考虑项目智力投入的可管理性，分析和设计建模对大型项目就具有重要意义。敏捷方法软件工程建模能否为这个问题提供可选方案呢？

[120]

[121]

在敏捷建模官方网站上, Scott Ambler[AMB02]用以下方式描述敏捷建模 (Agile Modeling, AM) :

AM是一种用于对软件系统有效建模和文档化的实践方法学。AM是可以有效和以轻量级方式用于软件开发项目的软件建模价值、原则和实践的集合。由于敏捷模型只是大致完善, 而不要求完美, 因此敏捷模型比传统的模型还要有效。

作为对敏捷宣言价值的补充, Ambler建议加入勇气和谦逊, 一个敏捷团队必须有做出可能导致否决设计和重新构建等决定的勇气, 也必须有意识到技术不能解决所有问题、应当尊重和采纳业务专家和其他相关人员意见的谦逊作风。

虽然AM提出大致的“核心”和“补充”建模原则, 但其中独具特色的是[AMB02]:

有目的的模型。在构建模型之前, 使用AM的开发者心中应当有明确的目标 (如与客户沟通信息, 或帮助更好地理解软件的某些侧面), 一旦确定模型的目标, 该用哪种类型的符号以及具体的需要程度都是显而易见的。

使用多个模型。描述软件可以使用多种不同的模型和表示法, 大多数项目只用到其中很小的部分就够了。AM建议从需要的角度看, 每一种模型应当表达系统的不同侧面, 应使用能够为那些预期的读者提供价值的模型。



前进灯、是用于所有软件工作的理念, 即仅仅构建能提供价值的模型——不多也不少。

前进灯。随着软件工程工作的进展, 只保留那些能提供长期价值的模型, 抛弃其余的模型。保留下来的每一个工作产品都必须随着变化而进行维护, 这些描述工作将使整个团队进度变慢。Ambler[AMB02]提示说“每次决定保留一个模型, 你都要在团队中以抽象方式使用信息的便利性与敏捷性方面做权衡 (即团队内部、团队与利益者增强沟通)。

内容重于表述形式。建模应当向预期的读者分享信息。一个有用的内容很少, 但语法完美的模型不如一个带有缺陷但能向读者提供有用内容的模型有价值。

理解模型及工具。理解每一个模型及其构建工具的优缺点。

适应本地需要。建模方法应该适应敏捷团队的需要。

122

SOFTWARE TOOLS

敏捷开发

目的: 敏捷开发工具的目标是辅助软件开发一个或多个方面, 强调便于快速构建可执行软件。这些工具也可以用于惯例 (传统) 过程模型 (见第3章) 的开发。

机制: 工具的机制各不相同。通常, 敏捷工具集包括项目计划、用例开发和需求收集、快速设计、代码生成和测试的自动支持。

代表性工具¹²

注: 由于敏捷开发是一个热门话题, 许多工具供应商都声称出售支持敏捷方法的工具。下面工具具有的特性对敏捷项目非常有用。

Actif Extreme, 由Microtool开发 (www.microtool.com), 提供对各种技术活动的敏捷过程管理支持。

¹² 这里记录的工具只是此类工具的例子, 并不代表本书支持这些工具。大多数情况下, 工具的名字由各自的开发者注册为商标。

Ideogramic UML，由Ideogramic开发（www.ideogramic.com），是特别为敏捷过程开发的UML工具集。

Together Tool Set，由Borland发布（www.borland.com或www.togethersoft.com），提供支持XP和其他敏捷过程中许多技术活动的工具包。

4.4 小结

软件工程的敏捷理念强调4个关键问题：具有控制力的自我组织团队对所开展工作的重要性；团队成员之间、开发参与者与客户之间的交流与合作；对“变更代表机遇”的认识；以及强调快速软件交付以让客户满意。敏捷过程模型能解决上述这些问题。

极限编程（XP）是应用最广泛的敏捷过程。按照计划、设计、编码和测试四个框架活动组织，XP建议一系列新颖和有力的技术，保证敏捷团队创建能体现客户指定优先级特征和功能的频繁软件发布。

适应性软件开发（ASD）强调人的合作和团队的组织，按思考、协作和学习三个框架活动组织，ASD使用迭代过程，该过程由自适应循环计划、相对严格的需求收集方法和一个迭代开发循环构成，其中迭代开发循环包括客户焦点组和正式技术评审作为实时反馈机制。动态系统开发方法（DSDM）定义了三种不同的迭代循环：功能模型迭代、设计和构建迭代和实现迭代，另外前面还增加了可行性研究和业务研究两个传统的生命周期活动。DSDM倡导时间调度的使用，认为对每一个软件增量所需的必要工作仅仅是能够方便地进入下一次增量开发。[123]

Scrum强调一系列软件过程模式的使用，这些模式已被证实对时间紧迫、需求变化和业务重要的项目非常有效。每一个过程模式定义一系列开发任务并允许Scrum团队以适应其项目的方式构建过程。

Crystal是一系列敏捷过程模型，可用于具有特定特征的项目。和其他敏捷方法类似，Crystal采用迭代策略，但可以调整过程的严格程度以适应不同规模和复杂度的项目。

特征驱动开发（FDD）某种程度上比其他敏捷方法更“形式化”，而且通过项目开发团队专注于特征（可在2周或更短时间内实现的客户价值功能）的开发来维持敏捷性。FDD比其他敏捷方法更强调项目管理和质量管理。

敏捷建模（AM）认为建模对于所有的系统都是必要的，但是模型的复杂度、类型和规模都必须根据所构建软件来调节。通过提出一系列核心和补充建模原则，AM给实践者的分析和设计任务以非常有用的指导。

参考文献

- [ADM96] Advanced Development Methods, Inc., "Origins of Scrum," 1996, <http://www.controlchaos.com/>.
- [AGI03] The Agile Alliance Home Page, <http://www.agilealliance.org/home>.
- [AMB02] Ambler, S., "What Is Agile Modeling (AM)?" 2002, <http://www.agilemodeling.com/index.htm>.
- [BEC99] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.
- [BEC01a] Beck, K., et al., "Manifesto for Agile Software Development," <http://www.agilemanifesto.org/>.
- [BEC01b] Beck, K., and M. Fowler, *Planning Extreme Programming*, Addison-Wesley, 2001.

- [BEE99] Beedle, M., et al., "SCRUM: An extension pattern language for hyperproductive software development," included in: *Pattern Languages of Program Design 4*, Addison-Wesley Longman, Reading, MA, 1999. Download at http://jeffsutherland.com/scrum/scrum_plop.pdf.
- [BUS00] Buschmann, F., et al., *Pattern-Oriented Software Architecture*, 2 volumes, Wiley, 1996, 2000.
- [COA99] Coad, P., E. Lefebvre, and J. DeLuca, *Java Modeling in Color with UML*, Prentice-Hall, 1999.
- [COC01] Cockburn, A., and J. Highsmith, "Agile Software Development: The People Factor," *IEEE Computer*, vol. 34, no. 11, November 2001, pp. 131-133.
- [COC02a] Cockburn, A., *Agile Software Development*, Addison-Wesley, 2002.
- [COC02b] Cockburn, A., "What Is Agile and What Does It Imply?" presented at the Agile Development Summit at Westminster College in Salt Lake City, March 2002, <http://crystalmethodologies.org/>.
- [CCS02] CS3 Consulting Services, 2002, <http://www.cs3inc.com/DSDM.htm>.
- [DEM98] DeMarco, T., and T. Lister, *Peopleware*, 2nd ed., Dorset House, 1998.
- [DEM02] DeMarco, T., and B. Boehm, "The Agile Methods Fray," *IEEE Computer*, vol. 35, no. 6, June 2002, pp. 90-92.
- [FOW00] Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 2000.
- [FOW01] Fowler, M., and J. Highsmith, "The Agile Manifesto," *Software Development Magazine*, August 2001, <http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm>.
- [FOW02] Fowler, M., "The New Methodology," June 2002, <http://www.martinfowler.com/articles/newMethodology.html#N8B>.
- [HIG98] Highsmith, J., "Life—The Artificial and the Real," *Software Development*, 1998, at <http://www.adaptivesd.com/articles/order.html>.
- [HIG00] Highsmith, J., *Adaptive Software Development: An Evolutionary Approach to Managing Complex Systems*, Dorset House Publishing, 1998.
- [HIG01] Highsmith, J., ed., "The Great Methodologies Debate: Part 1," *Cutter IT Journal*, vol. 14, no. 12, December 2001.
- [HIG02a] Highsmith, J., ed., "The Great Methodologies Debate: Part 2," *Cutter IT Journal*, vol. 15, no. 1, January 2002.
- [HIG02b] Highsmith, J., *Agile Software Development Ecosystems*, Addison-Wesley, 2002.
- [JAC02] Jacobson, I., "A Resounding 'Yes' to Agile Processes—But Also More," *Cutter IT Journal*, vol. 15, no. 1, January 2002, pp. 18-24.
- [JEF01] Jeffries, R., et al., *Extreme Programming Installed*, Addison-Wesley, 2001.
- [NOY02] Noyes, B., "Rugby, Anyone?" *Managing Development* (an on-line publication of Fawcette Technical Publications), June 2002, <http://www.fawcette.com/resources/managingdev/methodologies/scrum/>.
- [PAL02] Palmer, S., and J. Felsing, *A Practical Guide to Feature Driven Development*, Prentice-Hall, 2002.
- [SCH01] Schwaber, K., and M. Beedle, *Agile Software Development with SCRUM*, Prentice-Hall, 2001.
- [SCH02] Schwaber, K., "Agile Processes and Self-Organization," Agile Alliance, 2002, <http://www.aanpo.org/articles/index>.
- [STA97] Stapleton, J., *DSDM—Dynamic System Development Method: The Method in Practice*, Addison-Wesley, 1997.
- [WEL99] Wells, D., "XP—Unit Tests," 1999, <http://www.extremeprogramming.org/rules/unittests.html>.

124

习题与思考题

- 4.1 选择一条4.1节提到的敏捷性原则，讨论本章所描述的各过程模型是否符合该原则。
- 4.2 试着再加上一条“敏捷性原则”，以便帮助软件工程团队更具有机动性。
- 4.3 是否每一个敏捷过程都可以用第2章所提及的通用框架性活动来描述？建一张表，将通用活动和每个敏捷过程所定义的活动对应起来。
- 4.4 为什么迭代过程更容易管理变更？是不是本章所讨论的每一个敏捷过程都是迭代的？只用一次迭代就能完成项目的敏捷过程是否存在？解释你的答案。

- 4.5 用自己的语言描述（用于软件项目的）敏捷性？
- 4.6 阅读本章开头的“敏捷软件开发宣言”，能否想出一种情况，此时4个价值中的一个或多个将为软件开发团队带来问题？
- 4.7 为什么需求变化这么大？人们终究无法确定他们想要什么吗？
- 4.8 许多敏捷过程模型推荐面对面交流，实际上，现在软件开发团队成员及其客户在地理上是相互分散的。你是否认为这意味着这种地理上的分散应当避免？能否想出一个办法克服这个问题。
- 4.9 考虑4.2.2节提到的7种特色，根据你自己对它们的重要性的认识进行排序。125
- 4.10 写一个描述多数网页浏览器所具有的“收藏夹”或“收藏”特征的XP用户故事？
- 4.11 访问敏捷建模官方站点，给出所有核心和补充AM原则的列表。
- 4.12 用自己的语言描述XP的重构和结对编程的概念。
- 4.13 为什么称Crystal是一个敏捷方法家族？
- 4.14 使用第2章描述的过程模式模板，为4.3.4节描述的任一Scrum模式开发过程模式。
- 4.15 使用4.3.6节描述的FDD特征模板，定义Web浏览器的特征集，并为该特征集开发一系列特征。
- 4.16 什么是XP的“Spike解决方案”？

推荐读物与阅读信息

在Ambler（《Agile Modeling》，Wiley，2002）、Beck[BEC99]、Cockburn[CO02]和Highsmith[HIG02b]的书中深入探讨了敏捷软件开发的大致理念和深层原则。

Beck[BEC99]、Jeffries及其同事（《Extreme Programming Installed》，Addison-Wesley，2000）、Succi和Marchesi（《Extreme Programming Examined》，Addison-Wesley，2001）、Newkirk和Martin（《Extreme Programming in Practice》，Addison-Wesley，2001）、Auer及其同事（《Extreme Programming Applied: Play to Win》，Addison-Wesley，2001）的著作中，就如何更好地应用XP给出了关键的有指导意义的讨论。McBreen（《Questioning Extreme Programming》，Addison-Wesley，2003）则以挑剔的眼光审视XP，定义了何时、何地更为适用。对结对编程的深入考虑可阅读McBreen（《Pair Programming Illuminated》，Addison-Wesley，2003）。

Fowler和他的同事（《Refactoring: Improving the Design of Existing Code》，Addison-Wesley，1999）非常详尽地强调了XP中重构的概念，McBreen（《Software Craftsmanship: The New Imperative》，Addison-Wesley，2001）讨论了软件技术，认为敏捷方法可以作为传统软件工程的备选方案。

Highsmith[HIG00]深入强调了ASD。关于DSDM方法的有价值的论述可以在Stapleton（《DSDM: The Method in Practice》，Addison-Wesley，1997）的书中找到。Palmer和Felsing[PAL02]发表了关于FDD的详细研究，Carmichael和Haywood（《Better Software Faster》，Prentice-Hall，2002）发表了另一项关于FDD的研究，其中包含了对该过程机制的一步步的描述。Schwaber及其同事（《Agile Software Development with SCRUM》，Prentice-Hall，2001）发表了对Scrum方法的深入探讨。

Martin（《Agile Software Development》，Prentice-Hall，2003）在强调XP方法的基础上讨

论了敏捷原则、模式和实践。Poppdieck和Poppdieck (《Lean Development: An Agile Toolkit for Software Development Managers, Addison-Wesley, 2003》) 给出管理和控制敏捷项目的指导, Highsmith (《Agile Software Development Ecosystems》, Addison-Wesley, 2002) 提出了一份关于敏捷原则、过程和实践的调查。

关于敏捷软件开发的更为广泛的信息资源可以在Internet网上获得, 经常更新的敏捷过程

[126] 参考文献可以在SEPA网站<http://www.mhhe.com/pressman>上找到。



第二部分

软件工程实践

在这一部分，读者将学到构成软件工程实践的基本原则、概念和方法。在接下来的章节中，将涉及如下问题：

- 指导软件工程实践有哪些原则和概念？
- 如何由系统工程迈向高效的软件工程？
- 什么是需求工程？做好需求分析的基本思想是什么？
- 如何创建分析模型？它包含哪些元素？
- 什么是设计工程？做好设计的基本思想是什么？
- 创建架构、接口和构件级设计都用到了哪些概念、模型和方法？
- 什么样的策略适用于软件测试？
- 设计高效的测试用例有哪些方法？
- 哪些方法和策略可以用来评定分析和设计模型、源代码以及测试用例的质量？

一旦解决了以上问题，准备工作也就已经完成了，就可以马上开始软件工程应用实践了。

第5章 软件工程实践综述

要点浏览

概念：实践是软件计划和开发时需要考虑的方方面面，包括概念、原则、方法和工具等。它表达了一些细节——需要考虑的技术问题以及怎样实现软件过程中的东西，即实际构造高质量软件所必需的东西。

人员：软件工程实践由软件工程师和软件项目经理共同完成。

重要性：软件过程为每个开发计算机系统或产品的人提供了成功抵达目的地的路线图。实践为你提供了沿路驾驶的细节，它会告诉你哪里有桥、哪里有路障、哪里有公路，它帮助你理解一些必须理解的并且必须遵循的快速安全驾驶概念和原则，它指示你如何驾驶、在哪里应

该减速、在哪里应该加速。在软件工程中，实践就是当要把软件由想法转化为现实时你天天应该做的事情。

步骤：不管选择哪种过程模型，都必须运用实践三要素：概念、原则和方法。实践的第四个要素是工具，工具为方法的应用提供支持。

工作产品：实践贯穿于整个技术活动中。这些技术活动开发出由所选软件过程模型所定义的所有工作产品。

质量保证措施：首先，要深刻理解目前工作所用到的概念和原则（例如设计）。然后，确信你已经选定了一个合适方法；确信你理解如何运用适合的方法以及使用适合任务的自动工具，以及坚信为保证工作产品的质量所采用的技术。

关键概念

原则

敏捷建模
分析
编码
沟通
部署
设计
策划
软件工程
测试
问题解决
W⁵HH问题

在一本探讨软件工程师生活和思想的书中，Ellen Ullman[ULL97]通过一个生活片段描述了重重压力下软件工作者的思索：

我对于时间已经没有概念。在这间办公室里，没有窗户也没有时钟，只有红色LED显示屏在闪烁，它不断地闪现着12: 00, 12: 00, 12: 00, 12: 00。Joel和我已经折腾了好几天，有一个bug，一个很难处理的bug。这红色的闪烁脉冲就像我们的大脑，一直在以相同的速率闪动着……

我们在做什么？……具体是什么我现在也不知道。我们可能是在帮助可怜的病人，或者是在分布式数据库协议上调整一组底层例程来验证比特流——这些我并不关心。我应该关心，在其他时候——在此之后，或许当我们从这间到处都是电脑的房间中出来的时候，我会非常关心为什么、为了谁和为了什么目标而开发软件。但是现在不。我已经穿过了一层隔膜，在这里真实世界及其用处都已不再重要，我是软件工程师……

这段文字展现了软件工程实践的黑暗画面，而这种情况很可能和本书大多数读者都有关系。

计算机软件开发人员日常从事的艺术、工艺或者规范性活动¹，就是软件工程。那什么是软件工程“实践”？一般来讲，实践就是软件工程师日常使用概念、原则、方法和开发工具的集合，实践使得项目经理可以管理软件项目，保证软件工程师开发计算机程序，实践利用由必要技术和管理组成的软件过程模型，保证开发工作顺利开展，实践将使一些杂乱的容易被忽视的方法转化为更具组织性、更高效并且更容易获得成功的重要东西。

5.1 概念

WebRef

软件工程实践方面各种深刻的想法都可以在以下网址获得：
www.literateprogramming.com。

在第2章，我们介绍了一种通用的软件过程模型，这种模型由一组活动组成——这些活动确定了软件工程实践的框架。通用的框架活动——沟通、策划、建模、构造和部署——和普适性活动确定了软件工程工作的框架。所有在第3章和第4章讲述的软件过程模型都可以映射到这个框架里。但是怎样能让软件工程的实践也适合这个框架呢？在以下几节里，我们介绍一些应用于这些框架活动的基本概念和原则²。

5.1.1 实践的精髓



你可能会觉得 Polya的方法是很普通的，的确如此。但是它在软件的世界里所表现出的不寻常会让你大吃一惊。

在现代计算机发明之前的一本权威著作《How to Solve it》中，George Polya[POL45]列出了解决问题的本质，这也正是软件工程实践的精髓：

1. 理解问题（交流和分析）。
2. 计划解决方案（建模和软件设计）。
3. 实施计划（代码生成）。
4. 检查结果的精确度（测试和质量保证）。

在软件工程中，这些常识性步骤引发了一系列基本问题（引自POL45）：

理解问题：

- 谁将从问题的解决中获益？也就是说，谁是共利益者？
- 有哪些是未知的？哪些数据、功能、特征和行为是解决问题必需的？
- 问题可以划分吗？是否可以描述更小、更容易理解的问题？
- 问题可以图形化描述吗？可以创建分析模型吗？

计划解决方案：

- 以前曾经见过类似问题吗？在解决方案中是否隐藏着熟悉的模式？是否已经有软件实现了所需要的数据、功能、特征和行为？
- 类似问题是否解决？如果是，解决方案所包含元素是否可以复用？
- 可以定义子问题吗？如果可以，解决方案是否适用于子问题？
- 能用一种可以很快实现的方式来描述解决方案吗？能构建出设计模型吗？

实施计划：

- 成果和计划一致吗？源码是否可追溯到设计模型？
- 成果的每个组成部分是否都正确？设计和代码是否经过评审？或者更进一步，算法是否经过正确性证明？

129

¹ 有些作者认为是其中之一而没有其他内容。实际上，软件工程包含上述三个部分。

² 鼓励读者回顾本章相关小节以明确本书后面将要讨论的软件工程方法和普适性活动。

检查结果：

- 成果的每个部分是否可以进行测试？是否实现了合理的测试策略？
- 解决方案是否产生了与所要求的数据、功能、特征和行为一致的结果？是否按照项目共利益者的需求对软件进行了确认？

130 “在任何问题的解决方案中都会有所发现。”

—George Polya

5.1.2 核心原则

“原则”这个词在字典里的定义是“某种思想体系所需要的重要的根本规则或者假设”。在本书中，我们将讨论一些不同抽象层次上的原则。一部分关注软件工程的整体，另一部分考虑特定的、通用的框架活动（比如客户沟通），还有一些关注软件工程的活动（比如架构设计）或者技术任务（比如用例场景书写）。无论关注哪个层次，原则都可以帮助我们建立一种可以形成扎实软件工程实践的观念。因此，原则非常重要。

David Hooker[HOO96]提出7个关注软件工程整体实践的核心原则，复述如下³。



在开始一个软件项目之前，确保软件具有商业价值并且让用户体会到它的价值。

第1原则：存在价值

一个软件系统因能给用户提供价值而具有存在价值，所有的决定都应该基于这个思想。在确定系统需求之前，在关注系统功能之前，在决定硬件平台或者开发过程之前，问问你自己：这确实能为系统增加真正的价值吗？如果答案是不，那就坚决不做。所有的其他原则都以这条原则为基础。

第2原则：保持简洁

软件设计并不是一种随便的过程，在软件设计中需要考虑许多因素。所有的设计都应该尽可能简洁，但不是简化。这有助于更容易地理解和维护系统。这并不是说那些特征甚至内部的特征应该以“简练”为借口而取消。的确，优雅的设计通常也是简洁的设计，简练并不意味着“快速和粗糙”。事实上，它经常是经过大量思考和多次工作迭代才达到的，所得到的软件更易于维护且更少出错。

131 “简洁比所有巧妙的措词更加美妙。”

—Alexander Pope (1688—1744)

第3原则：维护视图

清晰的视图是软件项目成功的基础。没有视图，项目将会由于它有“两种或者更多种思想”而永远不能结束，没有完整而清晰的概念，系统将会变成由许多不协调的设计补丁堆积在一起的拼凑物……



131 如果软件有价值，那么价值就来源于软件有用的生命，基于这个原因，软件必须是可维护的。

软件系统架构设计的失败将削弱甚至彻底破坏一个设计良好的系统。拥有一个可以掌握系统架构并能监督其他人遵守这个架构的软件架构师将能确保软件项目开发非常成功。

第4原则：生产者要让消费者理解

已具产业规模的软件系统不可能在真空中开发和使用，通常软件系统必

³ 这里的引用得到了作者的授权[HOO96]。Hooker定义这些原则的模式请参见：<http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment>。

定是由开发者以外的人员使用、维护和编制文档，这就必须要让别人理解你的系统。因此，已完成了说明、设计、实现，别人必须要弄懂你在做什么。对于任何一个软件开发产品，其读者群可能都很大。需求说明中时刻关注用户、设计中始终想到实现、编码中想着那些要维护和扩展系统的人。一些人可能会被迫调试你所编写的代码，这使得他们成了你所编写代码的用户，尽可能地使他们的工作简单化会大大提升系统的价值。

第5原则：面向未来

生命期持久的系统拥有更多的价值。在现今的计算环境中，规格说明随时会改变，硬件平台几个月后就会被废弃，软件生命周期都是以月而不是以年来衡量。可是，真正具有“工业实力”的软件系统必须持久耐用。为了能成功地做到这一点，系统必须能适应这样那样的变化，能成功做到这一点的系统都是那些一开始就这样来设计的系统。永远不要把自己的设计局限于一隅，经常问问：“如果出现……应该怎样应对”，构建系统过程中准备所有可能方案以解决普遍问题，而不是仅着眼于某个特定问题的一个方案⁴，这会为将来整个系统的复用提供可能。

第6原则：计划复用

复用既省时又省力⁵。高水平的复用系统是开发软件系统过程中很难实现的一个目标，代码和设计复用是面向对象技术带来的一个重要好处，然而，这种投入的回报也不会自动实现。面向对象（或常规）编程对复用起到了杠杆的作用，它需要从一开始就考虑和计划。系统开发过程中各种层面的复用有多种实现技术，最有名、讨论最多的是详细设计和编码阶段的复用，如许多新文献都讲到的软件设计模式复用。不过，这只是问题的一部分，对复用而言，与团队中的其他人员交流是很重要的，想想看，你怎么可能复用那些你根本就不知道是否存在的东西呢？在复用前先策划可以降低成本，并增加可复用构件以及构件化系统的价值。

132

第7原则：认真思考

这最后一条原则可能是最容易被忽略的。在行动之前清晰、完整的思考通常能产生更好的结果。一旦仔细思考过某件事情，能做好这件事情的可能性就变大了，而且也能获得更多关于如何把它做好的知识。如果仔细思考过某件事情还是把它做错了，那么，这就变成了很有价值的经验。思考还有一个作用是能让你对一无所知的某件事情产生感性认识，这样你慢慢就能找到答案了。当清晰的思想用于系统中时，价值也就产生了。如果能在使用前六条原则时经过认真思考，那么，潜在的收获将数不胜数。

如果每位工程师、每个开发团队都能遵从Hooker这七条简单的原则，那么，开发复杂计算机软件系统时所遇到的许多困难都可以迎刃而解。

5.2 沟通实践

在分析、建模或规格说明之前，客户的需求必须通过沟通活动（也称需求诱导）来收集，任何一个有问题的客户都可能会影响到将来的开发成果。沟通从开发者对客户提出的需求做出回应之时就开始了，但是从沟通到理解这条路并不平坦。

高效的沟通（与其他技术人员的沟通，与客户和其他共利益者的沟通，与项目经理的沟

⁴ 作者注：把这个建议发挥到极致非常危险，设计通用方案会带来性能损失和额外开销。

⁵ 作者注：这对于未来采用复用技术的项目而言是正确的，而设计和实现可复用构件是要增加成本的。研究表明，设计和开发可复用构件比直接开发系统要增加25%~200%的成本，而有些情况下费用的差别很难估计出来。

通) 是一个软件工程师所面临的最具挑战性的工作。在这里, 我们将讨论与客户沟通的原则和概念。不过, 许多原则同样可以应用于软件开发中其他场合的沟通。



133

原则#1: 倾听。尝试仔细倾听讲话者的每一句话, 而不是急于叙述你对观点的看法。如果有什么事情不清楚时要求其澄清, 但是要避免经常打断别人的讲述, 当别人正在陈述的时候不要在言语或动作上表现出异议(比如: 转动眼睛或者摇头)。

原则#2: 有准备的沟通。在与其他人碰面之前花点时间去理解问题。如果必要的话, 做一些调查来理解业务领域的术语。如果你负责主持一个会议, 那么在开会之前准备一个议事日程。

原则#3: 需要有人推动。每个沟通会议都应该有一个主持人(推动者), 其作用是:(1)保持会议向着有效的方向进行;(2)能调解会议中所发生的冲突;(3)能确保遵循我们所说的沟通原则。

原则#4: 最好当面沟通。但是, 如果能把一些相关信息表示出来, 通常可以工作得更好, 例如可以在集中讨论中使用草图或文档草稿。

“简洁的问题和简洁的回答是解决问题最好的办法。”

——Mark Twain

原则#5: 记录所有决定。任何解决方法都可能有缺陷。参与交流的人应该记录下所有要点和决定。

原则#6: 保持通力协作。当项目组成员的想法需要汇集在一起用以阐述一个产品或者某个系统的功能或特征的时候, 就产生了协作与协调的问题。每次小的协调都可能建立起项目成员间的相互信任, 并且为项目组创建一致的目标。

原则#7: 聚焦并协调话题。在任何交流中, 参与的人越多, 话题转移的可能性就越大。最简便的方法就是保持谈话局部化, 只有当某个话题完全解决之后再开始别的话题(参见原则#9)。

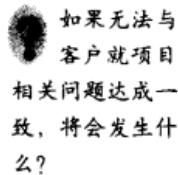
INFO

客户与最终用户之间的差别

软件工程师与许多共利益者交流, 但是客户与最终用户之间在所采用技术上有重大的冲突。有的时候客户与最终用户是一致的, 但是对于许多项目来说, 客户与最终用户是不同的人, 为不同商业组织的不同管理者工作。

客户是这样的人(或组织):(1)最初要求构建软件的人;(2)为软件定义全部业务目标的人;(3)提供基本的产品需求的人;(4)为项目调整资金的人。在产品或者业务系统中, 客户通常负责市场;在某个IT环境下, 客户或许是一个商业单位或部门。

最终用户是这样的人(或组织):(1)将一直使用为了达到某种商业目的而编写的软件;(2)定义软件可操作细节以便可以达到其商业目的。



134

原则#8: 采用图形表示。语言沟通的效果很有限, 当文字无法表述某项工作的时候, 草图或者绘图通常可以让表述变得更为清晰。

原则#9: 继续前进原则。(a)一旦认可某件事情, 继续前进;(b)如果不认可某件事情, 继续前进;(c)如果某项特性或者功能不清晰, 当时无法澄清,

继续前进。交流如同所有其他软件工程活动一样需要时间，与其永无止境地迭代，不如让参与者认识到还有很多话题需要讨论（参见原则#2），“继续前进”有时是最好的交流方式。

原则#10：谈判双赢原则。谈判不是一场竞赛或者一场游戏，谈判双赢时才发挥了谈判的最大价值。很多时候软件工程师和客户必须商讨一些问题，如功能和特征、优先级和交付日期等。如果团队合作得很好，那么各方都有一个共同的目标，因此，谈判需要各方的协调。

SAFEHOME

沟通出现问题

[场景] 软件工程开发团队工作场所。

[人物] Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员。

[对话]

Ed: 对这个SafeHome的项目你们听到什么了？

Vinod: 第一次会议安排在下周。

Jamie: 我已经做了一些调查，但是进行得不那么顺利。

Ed: 你的意思是什么？

Jamie: 是这样，我给Lisa Perez 打了一个电话，她是市场部经理。

Vinod: 然后呢……

Jamie: 我想让她告诉我关于SafeHome的特征和功能……之类的事情。可是，她开始问我一些关于安全系统、监视系统等方面的问题，这些我并不精通。

Vinod: 这对你有何启示？

(Jamie耸了耸肩)

Vinod: 市场需要我们扮演顾问的角色，我们最好在开始第一次会议之前对这个产品领域做一些了解。Doug说过他希望我们与客户“协作”，这样才能更好地了解如何开发。

Ed: 也许你应该去她的办公室，电话不适合做这样的工作。

Jamie: 你们说的都对。我们应该努力做好这方面的工作，做好早期的交流。

Vinod: 我看到Doug在看一本“需求工程”的书，我敢打赌这本书上肯定罗列了一些好的交流原则，我要从他那里借来看看。

Jamie: 好主意，然后你就可以教我们啦。

Vinod (微笑): 哈，没问题。

TASK SET

沟通活动通用任务集

- 识别主要客户和其他共利益者（见7.3.1节）
- 与主要客户会谈“环境无关的问题”（见7.3.4节），以确定：
业务需要和商业价值。
最终用户的特性/需要。
需要的用户可见输出。
业务约束。
- 写一页项目范围（范围往往会修订）的说明（见7.4.1节和21.3.1节）。
- 评审范围说明，并应客户要求做出相应修改。

5. 与客户/最终用户进行协作，确定以下内容：

采用标准格式（见7.5节）记录客户可见的使用场景⁶。

输入和输出。

重要的软件特性、功能和行为。

客户定义的商业风险（见25.3节）。

6. 简要笔头描述（例如一些列表）场景、输出/输入、特性/功能以及风险。

7. 与客户反复交换意见以细化场景、输出/输入、特性/功能以及风险。

8. 为每个用户场景、特性、功能和行为分配客户定义的优先级（见7.4.2节）。

9. 回顾在与客户和其他共利益者交流时搜集的所有信息并且按照要求修订。

10. 为计划活动做准备（第23章与第24章）。

5.3 策划实践

沟通活动可以协助软件开发团队定义其全局目标（主题，当然这会随着时间的推移而变化）。可是，理解这些目标与为达到这些目标的制定计划并不是一回事。策划活动包括一系列管理和技术实践，可以为软件开发团队定义一个便于他们向着战略目标和战术目标前进的路线图。

“在准备战斗时我总是发现计划没用，但是制定计划是必不可少的。”

——Dwight D. Eisenhower

有很多制定计划的哲学。一些人是“最低要求者”，他们认为变化常常会消除详细计划的必要性；另一些人是“传统主义者”，他们认为计划提供了有效的路线图，并且计划的细节越多，团队所失去的将会越少；也有一些人是“活动主义者”，他们认为快速制定计划是必需的，而路线图将会在真正的软件开发工作开始时浮现出来。

要做什么？在许多项目中，“过度计划”是一种时间的浪费（事物有太多的变化），但是“最起码的计划”是制止混乱的良方。就像在生活中的很多事情一样，适度执行计划足以为团队提供有用指导——不多也不少。

无论多么严格地制定计划，都应该遵循以下原则。

原则#1：理解项目范围。如果你不知道要去哪里，就不可能使用路线图。范围可以为软件开发团队提供一个目的地。

136

原则#2：客户参与策划。客户定义了一些优先权，确定了项目的约束。为了适应这种情况，软件工程师必须经常商谈交付的顺序、时间线以及其他与项目相关的问题。

原则#3：采用迭代计划。项目计划不可能一成不变。在工作开始的时候，有很多事情有可能改变，那么计划就必须调整以适应这些变化。另外，迭代式增量过程模型指出了基于用户反馈的（在每个软件增量发布之后）重新制定计划。

原则#4：基于已知估计。估计的目的是基于项目组对已完成工作的当前理解，提供一种关于工作量、成本和任务工期的指标。如果信息是含糊的或者不可靠的，估计也将是不可靠的。

⁶ 使用场景格式将在第8章讨论。

WebRef

有关策划和项目管理的优秀资料可参见：
[www.4pm.com/
repository.html](http://www.4pm.com/repository.html)

原则#5：计划考虑风险。如果团队已经明确了哪些风险最容易发生且影响最大，那么应急计划就是必需的了。另外，项目计划（包括进度）应该可以调整以适应那些可能发生的风险。

原则#6：保持脚踏实地。人们不能每天每时每刻都工作。噪音总能钻入人们的交流之中。生活的现实就是冗长与含糊，变化总是在发生，甚至最好的软件工程师都会犯错误，这些现实的东西都应该在项目计划制定的时候考虑。

“成功更多的是一种协调能力而不是天赋。”

——王安



原则#7：调整计划粒度。粒度主要指项目计划的细节。一个“细粒度”的计划可以提供很有意义的工作任务细节，这些细节是基于短时间增量的计划（这样跟踪和控制就会常常发生）。一个“粗粒度”的计划提供了更宽的长时间工作任务。通常，以当前时间为基准，粒度随项目时间线的远离而从细到粗，在几个星期或几个月的时间里可以详细地策划项目，而在几个月内都不会发生的活动不需要细化（太多的东西将会发生变化）。

原则#8：制定计划确保质量。计划应该确定软件开发团队如何去确保开发的质量。如果要执行正式技术评审⁷的话，应该将其列入进度；如果在编码过程中用到了结对编程（第4章），那么在计划中要明确描述。

137

原则#9：描述如何适应变化。即使最好的策划也有可能被无法控制的变化破坏。软件开发团队应该确定在软件开发过程中有哪些变化是需要去适应的，例如，客户会随时提出变更吗？如果提出了一个变更，团队是不是要立即去实现它？变更会带来怎样的影响和开销？

原则#10：经常跟踪、校正计划。项目有时会落后时间表一天的时间。因此，需要每天都追踪计划的进程，找出计划与实际执行不一致的问题所在，当实际任务有所改变时，计划也要随之做出调整。

最高效的方法是软件开发项目组所有成员都参与到策划活动中来，只有这样，项目组成员才能更好地执行计划。

在一篇关于软件项目与过程的优秀论文中，Barry Boehm [BOE96]说道：“你们需要一个能为简单项目提供简单计划的组织原则。”Boehm提出了一个确定项目目标、里程碑、进度、项目责任、管理以及技术方案和所需资源的方法，他称之为W⁵HH原则。以下问题用来指导定义项目特征及项目计划。

② 开发一个实际项目计划所必须提出和回答的问题是什么？

为什么要开发？所有部门都应陔对软件开发工作的商业目的做出评定。也就是说，是否需要为某个商业目的而做出人力、时间和金钱支出。

要做什么东西？确定要开发的功能和完成工作所需的任务。

什么时候完成？为重要的项目任务确定工作流和时间线，并制定客户所要求的里程碑。

功能由谁负责？项目组每个成员的角色与责任都必须明确。

组织位于哪里？并不是所有角色和责任都在软件开发团队内部。客户、用户以及其他共利益者也有责任。

怎样才能在工作中体现技术和管理？一旦确定了产品范围，项目的管理

⁷ 正式技术评审问题将在第26章讨论。

和技术策略也必须确定。

需要多少资源？这个问题的答案是一种估算（参看第23章），而这种估算基于对上述问题的回答。

Boehm的W⁵HH问题的答案对不同大小、复杂度的项目来讲都很重要。但是策划过程该如何开始呢？

“我们认为软件开发人员忽视了一个重要的事实：大多数的组织并不知道他们在做什么。他们自认为知道在做什么，但实际上他们并不知道。” ——Tom DeMarco

TASK SET

策划活动通用任务集

1. 再评估项目范围（见7.4节和21.3节）。
2. 评估风险（见25.4节）。
3. 确定和细化用户场景（见7.5节与8.5节）。
4. 从用户场景中抽象出功能与特征（见8.5节）。
5. 确定实现软件基本要求的技术功能与特征。
6. 按照用户优先级对用户特有的功能与特征（场景）分组。
7. 制定一个粗粒度的项目计划（见第23、24章）。
 - 确定预计的软件增量个数。
 - 确定整个项目的进度（见第24章）。
 - 为每一个增量确定预计的交付时间。
8. 为当前的迭代制定细粒度项目计划（见第23、24章）。
 - 为每个功能特征制定工作任务（见23.6节）。
 - 对每个工作任务估算工作量（见23.6节）。
 - 对每个工作任务分派责任（见23.4节）。
 - 确定要开发的工作产品。
 - 确定要使用的质量保证方法（见第26章）。
 - 描述处理变更的方法（见第27章）。
9. 定期追踪进度（见24.5.2节）。
 - 注意可能出现的问题（例如，进度延误）。
 - 按需调整进度。

5.4 建模实践

我们可以通过创建模型来更好地理解需要构建的实体。当实体是物理实物（例如：一栋建筑，一架飞机，一台机器）时，我们可以构建在形式和形状上都和实物相同只是比实物缩小了的模型。可是，当实体是一个软件的时候，我们的模型就是另外一种形式了，它必须能够表现出软件所转换的信息、使转换发生的架构和功能、用户要求的特征以及转换发生时系统的行为。模型必须能在不同的抽象层次下完成那些目标——首先从客户的角度描述软件，然后在更侧重于技术方面表述软件。



分析模型表达了客户的需求。设计模型为软件的构造提供了详细的描述。

在软件工程中，要创建两类模型：分析模型和设计模型。分析模型通过在以下三个不同域描述软件来表达客户的需求：信息域、功能域、行为域。设计模型表述了可以帮助开发者高效开发软件的特征：架构（见第10章）、用户界面（见第12章）以及构件细节（见第11章）。

139

下面几节中，我们将描述与分析模型和设计模型相关的基本原则和概念，在以后几章中将描述创建分析和设计模型所使用的技术方法和表达方法。

“工程师在设计时的首要问题是发现真正的问题。”

——作者不详

5.4.1 分析建模原则

在过去的30年里，已经开发出了大量的分析建模方法。研究人员已经弄清了分析中的问题及其出现原因，也开发出了各式各样的建模方法以及相关启发性解决方法。每一个分析方法都有其独立观点。不过，所有的分析方法都具有共同的操作原则。

分析模型集中于软件的三个属性：要处理的信息，要发布的功能和要展现的行为。

原则#1：必须描述并理解问题的信息域。信息域包括了流入系统的数据（从最终用户、其他系统或者外部设备）、流出系统的数据（通过用户接口、网络接口、报告、图形以及其他方式）以及那些收集和组织永久性数据对象的数据（例如：永久存储的数据）。

原则#2：必须确定软件所要实现的功能。软件功能直接为最终用户提供服务并且为用户可见特征提供内部支持。一些功能对流入系统的数据进行转换，有些功能还影响着对软件内部过程或外部系统的控制。功能可以在不同的抽象层次描述，必须涉及从整体意图到过程细节的描述。

原则#3：必须描述软件的行为（作为外部事件的结果）。软件的行为受外部环境交互驱动。最终用户提供的输入，由外部系统提供的控制数据，或者基于网络收集的监控数据都会引起软件的不同行为。

原则#4：描述信息、功能和行为的模型必须以一种能揭示分层（或者分级）细节的方式分解开来。分析建模是解决软件工程问题的第一步。它能使开发者更好地理解问题并且为确定解决方案（设计）准备条件。复杂的问题很难完全解决，基于这样的原因，我们使用“分而治之”的战略。把大的复杂问题划分成很多易于理解的子问题，这就是“分解”的概念，这也是分析建模的关键策略。

140

原则#5：分析任务应该从本质信息转向实现细节。分析建模从最终用户角度描述问题开始。在没有考虑解决方案的前提下描述问题的“本质”。例如，一个视频游戏需要玩家在他所扮演角色进入一个危险的迷阵时控制其角色行动的方向，这就是问题的本质。实现细节（通常作为设计模型的一部分来描述）指出问题的本质将如何去实现，对于视频游戏来说，可能需要用到声音输入。此外，可能使用键盘命令和操纵杆（或者鼠标）来控制角色的行动方向。

TASK SET

通用分析建模任务集

1. 评审业务需求、最终用户特征/要求、用户可见输出、业务约束以及其他一些在客户沟通和策划活动中需要确定的技术需求。
2. 扩展和细化用户场景（见8.5节）。
 - 定义所有的角色。
 - 表达角色如何与系统交互。
 - 从用户场景中抽象出功能和特征。
 - 评审用户场景的完备性和准确性（见26.4节）。
3. 信息域建模（见8.3节）。
 - 描述所有主要信息对象。
 - 定义每个信息对象的属性。
 - 表达信息对象之间的关系。
4. 功能域建模（见8.6节）。
 - 表达出功能如何修改数据对象。
 - 细化功能以提供细节。
 - 书写叙述性的过程来描述每个功能和子功能。
 - 评审功能模型（见26.4节）。
5. 行为域建模（见8.8节）。
 - 确定引起系统行为变化的外部事件。
 - 确定表明每个外部可见行为模型的状态。
 - 描述事件如何引起系统的状态转换。
 - 评审行为模型（见26.4节）。
6. 用户接口分析和建模（见12章）。
 - 实施任务分析。
 - 创建屏幕视图原型。
7. 评审所有模型，考虑完备性、一致性以及是否有遗漏。

5.4.2 设计建模原则

软件设计模型如同建筑师的房屋设计方案。由表达所有需要建造的东西（例如，房屋的三维透视图）开始，然后逐渐进行细化，为每个构建详图（例如，管线分布图）提供指导。类似的，软件设计模型为系统提供了各式各样的不同视图。

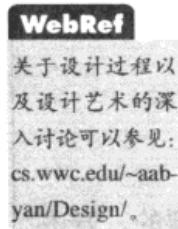
“首先要了解设计是否考虑周全而且是否合理：一经证明是无误的，就要坚持下去；不要因为有人排斥它，就动摇取得最终结果的决心。” ——William Shakespeare

141

现在已经有许多方法可以导出不同软件设计要素。部分方法是数据驱动的，它通过数据结构来得到程序构架和处理构件；另外一部分方法是模式驱动的，使用问题域（分析模型）信息来开发架构风格和处理模式；还有一些方法是面向对象的，使用问题域对象来创建数据

结构以及操作这些数据结构的方法。不过，无论使用什么方法，都使用同一套设计原则。

原则#1：设计可追溯到分析模型。分析模型描述了问题的信息域、用户可见的功能、系统的行为以及一套分析类，分析类把业务对象和为这些对象服务的方法结合在一起。设计模型将这些信息转化为系统架构：一套实现主要功能的子系统以及一套实现分析类的构件级设计。除去与软件基本构造相关的设计外，设计模型的各个元素都应该能在分析模型上找到其相应的部分。



原则#2：经常关注待建系统的架构。软件架构（参看第10章）是系统的骨架，它决定着系统接口、数据结构、程序控制流和行为、测试的方法以及在建系统的可维护性等。基于上述原因，设计应该从考虑架构开始，在架构确定以后才开始考虑构件级设计。

原则#3：数据设计与功能设计同等重要。数据设计是架构设计的基本要素。在设计中数据对象实现方法绝不能忽略，一个好的结构化数据设计可以简化程序流程，让软件构件设计与实现变得更简单，使得处理过程更为高效。

原则#4：必须设计接口（包括内部接口和外部接口）。系统中构件之间数据流的流动方式大大影响着系统的处理效率、误差传播以及设计简单化等方面。一个好的接口设计可以让集成变得更简单并能辅助测试人员进行构件功能测试。

原则#5：用户界面设计必须符合最终用户要求。在任何情况下，界面的设计都强调使用的方便性。用户接口是软件中可见的部分，无论系统的内部功能多么复杂，无论其数据结构多么容易理解，无论系统架构设计有多好，一个不好的界面设计都会使整个软件变得很糟糕。

原则#6：功能独立的构件级设计。功能上独立是软件构件“单一思想”的度量方法。构件提供的功能应该是内聚的——也就是说，它应该关注一个且仅仅一个功能或子功能⁸。

原则#7：构件之间以及构件与外部环境之间松散耦合。耦合可以通过很多方式来实现——构件接口、消息传递、全局数据。随着耦合程度的提高，错误发生的几率也会随之提高，整个软件的可维护性也会降低。因此，应该尽可能地降低构件耦合。

原则#8：设计表述（模型）应该做到尽可能易于理解。设计的目的是向编码人员、测试人员以及维护人员传递信息，如果设计过于复杂而难以理解，就无法成为一种高效的沟通媒介。

原则#9：设计应该迭代式进行。每一次迭代，设计者都应该尽力简化。就像大多数创造性活动一样，设计是迭代发生的，第一次迭代的工作是细化设计并纠正错误，之后的迭代就应该让设计变得尽量简单。

恰当地应用以上设计原则，软件工程师就能创造出兼顾内部高质量和外部高质量的设计。外部质量因素是软件在最终用户使用时所遇到的实际属性（例如：速度、可靠性、正确性、可用性），内部质量因素对与软件工程师来说是很重要的，他们要从技术的角度做出高质量的设计。要想有一个高质量的内部设计，设计师们必须理解设计的基本概念（见第9章）。

⁸ 对于内聚的其他讨论见第9章。

INFO

敏捷建模

Scott Ambler在他的讨论敏捷建模的书中[AMB02]定义了一套原则⁹，这套原则适用于软件的敏捷分析和设计（见第4章）。

原则#1：软件团队首要目标是构建软件而不是创造模型。

原则#2：不要创建任何你不需要的模型。

原则#3：尽量创建能描述问题和表示软件的最简单模型。

原则#4：用能适应模型改变的方式构建模型。

原则#5：明确描述创建每一个模型的目的。

原则#6：调整所开发模型来适应待开发系统。

原则#7：尽量构建有用的模型而不是完美的模型。

原则#8：对于模型的构造方法不要过于死板。模型能成功地传递内容是主要的，表述形式是次要的。

原则#9：如果直觉告诉你模型不准确，尽管理论上看上去很正确，那么就要仔细注意了。

原则#10：尽可能快地获取反馈。

无论选择哪种软件过程模型，无论应用哪种软件工程实践，所有的软件开发团队都希望采用敏捷开发。因此，无论使用哪种软件过程模型，以上原则都可以且应该适用。

143

TASK SET

通用设计任务集

1. 使用分析模型，选择适合软件的架构风格（模式）（见第10章）。

2. 把分析模型划分成若干设计子系统并按照架构分配到各个子系统中（见第10章）。

确定每个子系统都是功能耦合的。

设计子系统接口。

把分析的类或功能分配到各个子系统中。

使用信息域模型，设计合适的数据结构。

3. 设计用户界面（见第12章）。

评审任务分析结果。

确定基于用户场景的活动顺序。

创建界面行为模型。

定义界面对象和控制机制。

评审界面设计并进行必要的修正（见26.4节）。

4. 进行构件级设计（见第11章）。

在较低层抽象级上确定所有的算法。

精化每个构件的接口。

定义构件级的数据结构。

评审构件级设计（见26.4节）。

5. 开发部署模型（见9.4.5节）。

⁹ 在本节所强调的原则都根据本书的目的而进行了缩写与改编。

5.5 构造实践

构造活动包括一系列编码和测试任务，从而为向客户和最终用户交付可运行软件做好准备。在现代软件工程中，编码可能是：(1) 直接生成编程语言源代码；(2) 使用待开发构件的类中间设计表示自动生成源代码；(3) 使用第四代编程语言（例如Visual C++）自动生成可执行代码。

“在生活中，我有一个癖好，喜欢偷看别人写得很糟糕的代码。偶尔我也能发现一些真正有价值的东西，一些结构很好的代码，这些代码书写形式固定，与具体机器无关，每个构件都是那么简单，易于组织且易于修改。”

——David Parnas

144

最初的测试是构件级的，通常称为“单元测试”。其他级别的测试包括：(1) 集成测试（在构建系统的时候进行）；(2) 确认测试——测试系统（或者软件增量部分）是否完全按照需求开发；(3) 验收测试——由客户检验系统是否按照需求实现了所有的功能。

在编码和测试过程中有一套原则，下面就讲述这些原则和概念。

5.5.1 编码原则和概念

这些原则和概念与编程风格、编程语言和编程方法紧密结合。下面陈述一些基本的原则：



避免开发解决错误问题的程序。多放点注意力在第一个准备原则上。

WebRef

关于各种编码规范的链接请看：
www.literateprogramming.com/festyle.html

准备原则：在写下每行代码之前，要确保：

1. 理解了所要解决的问题。
2. 理解基本的设计原则和概念。
3. 选择一种能够满足软件构建以及运行环境要求的编程语言。
4. 选择一种能提供工具以简化工作的编程环境。
5. 构件级编码完成后进行单元测试。

编码原则：在开始编码时，要确保：

1. 遵循结构化编程方法约束算法[BOH00]。
2. 选择能满足设计要求的数据结构。
3. 理解软件架构并开发出与其相符的接口。
4. 尽可能保持条件逻辑简单。
5. 用易于测试的方法开发嵌套循环。
6. 选择有意义的变量名并要符合相关编码标准。
7. 编写注释。
8. 增强代码的可读性（例如：缩排和空行）。

确认原则：在完成第一阶段的编码之后，要确保：

1. 适当进行代码走查。
2. 进行单元测试并改正所发现的错误。
3. 重构代码。

145

讲述编码和编码原则的书有不少，如[KER78]的编程风格入门，[MCC93]的软件构造实践，[BEN99]的编程精华，[KNU99]的编程艺术，[HUN99]的实用编程等。

通用构造任务集

1. 构建基本架构（见第10章）。
 - 评审架构设计。
 - 编码并测试构件以实现基本架构。
 - 获得可复用的架构模式。
 - 测试基本构架以确保接口完整性。
2. 构建软件构件（见第11章）。
 - 评审构件级设计。
 - 对构件进行单元测试（见13.3.1节和14.7节）。
 - 对构件数据结构和接口进行编码。
 - 对内部算法和相关处理功能进行编码。
 - 在编码完成时评审代码（见26.4节）。
 - 检查正确性。
 - 确保遵循编码标准。
 - 确保编码有注释。
3. 构件单元测试。
 - 进行所有的单元测试。
 - 纠正所发现的错误。
 - 重复单元测试。
4. 将已经完成的构件集成到基本构架中。

5.5.2 测试原则

在一本经典软件测试书中，Glen Myers[MYE79]描述了一系列测试规则，这些规则很好地阐明了测试的目标：

- | | |
|-------------------|---|
| ● 软件测试的
目标是什么？ | <ul style="list-style-type: none"> • 测试是一个以查找程序错误为目的的程序执行过程。 • 一个好的测试用例能最大限度地找到尚未发现的错误。 • 一个成功的测试能找到那些尚未发现的错误。 |
|-------------------|---|

这些目标意味着软件开发者们在观念上的一些戏剧性的变化。他们持有与常人相反的观点——常人的观点认为那些找不到一个错误的测试是成功的测试。我们的目标是要设计一些能用最短的时间、最少的工作量来系统地揭示不同类型错误的测试。

[146] Davis[DAV95]提出了一套测试原则¹⁰，这些原则本书做了一些改动：

原则#1：所有的测试都应该可以追溯到用户需求¹¹。软件测试的目标就是要揭示错误。而最严重的错误（从用户的角度来看）是那种导致程序无法满足需求的错误。

原则#2：测试计划应该远在测试开始前就开始着手。测试计划（第13章）在分析模型一

¹⁰ 这里只提到了Davis测试原则中的一小部分，更多的信息请看[DAV95]。

¹¹ 这个原则指的是功能测试等基于需求的测试。结构测试（着眼于构架和逻辑细节的测试）无法直接解决特定的需求。

通用构造任务集

1. 构建基本架构（见第10章）。
 - 评审架构设计。
 - 编码并测试构件以实现基本架构。
 - 获得可复用的架构模式。
 - 测试基本构架以确保接口完整性。
2. 构建软件构件（见第11章）。
 - 评审构件级设计。
 - 对构件进行单元测试（见13.3.1节和14.7节）。
 - 对构件数据结构和接口进行编码。
 - 对内部算法和相关处理功能进行编码。
 - 在编码完成时评审代码（见26.4节）。
 - 检查正确性。
 - 确保遵循编码标准。
 - 确保编码有注释。
3. 构件单元测试。
 - 进行所有的单元测试。
 - 纠正所发现的错误。
 - 重复单元测试。
4. 将已经完成的构件集成到基本构架中。

5.5.2 测试原则

在一本经典软件测试书中，Glen Myers[MYE79]描述了一系列测试规则，这些规则很好地阐明了测试的目标：

- | | |
|-------------------|---|
| ● 软件测试的
目标是什么？ | <ul style="list-style-type: none"> • 测试是一个以查找程序错误为目的的程序执行过程。 • 一个好的测试用例能最大限度地找到尚未发现的错误。 • 一个成功的测试能找到那些尚未发现的错误。 |
|-------------------|---|

这些目标意味着软件开发者们在观念上的一些戏剧性的变化。他们持有与常人相反的观点——常人的观点认为那些找不到一个错误的测试是成功的测试。我们的目标是要设计一些能用最短的时间、最少的工作量来系统地揭示不同类型错误的测试。

[146] Davis[DAV95]提出了一套测试原则¹⁰，这些原则本书做了一些改动：

原则#1：所有的测试都应该可以追溯到用户需求¹¹。软件测试的目标就是要揭示错误。而最严重的错误（从用户的角度来看）是那种导致程序无法满足需求的错误。

原则#2：测试计划应该远在测试开始前就开始着手。测试计划（第13章）在分析模型一

¹⁰ 这里只提到了Davis测试原则中的一小部分，更多的信息请看[DAV95]。

¹¹ 这个原则指的是功能测试等基于需求的测试。结构测试（着眼于构架和逻辑细节的测试）无法直接解决特定的需求。

5.6 部署

正如我们在第2章中提到的，部署活动包括三个动作：交付、支持和反馈。由于现代软件过程模型本质上是演化的，因此，部署活动并不是只发生一次，而是在软件完全开发完成之前要进行许多次。每个交付循环都会向客户和最终用户提供一个可运行的并具有可用功能和特征的软件增量。每个支持循环都会为部署循环中所提到的所有功能和特征提供一些文档和人员帮助。每个反馈循环都会为软件开发团队提供一些重要的引导，以帮助修改软件功能、特征以及下一个增量所用到的方法。

软件增量交付对于任何一个软件项目来说都是重要的里程碑。以下将讲述一些软件开发团队在准备交付一个软件增量时所应该遵从的原则：



在软件增量交付之前，确定客户知道其对软件的期望。否则，客户期望的总是比你交付的东西多。

原则#1：客户对于软件的期望必须得到管理。客户通常并不希望看到软件开发团队对软件的提交作出承诺后又没有实现。这将导致反馈变得无用并且会挫伤软件开发团队的士气。Naomi Kartan[KAR94]在她的关于管理客户期望的书中提到：“管理客户期望首先应该认真考虑你该与客户交流什么与怎样交流。”她建议软件工程师必须认真地处理与客户有冲突的信息。（例如：对不可能在交付时完成的工作作出了许诺；在某次软件增量交付时交付了多于当初许诺要交付的工作，这将使得下次增量所要做的工作随之变少。）

原则#2：完整的交付包应该经过安装和测试。光盘或其他包括了可执行软件的介质以及一些支持数据文件、文档和一些相关的信息必须组装起来，并经过实际用户的完整的 β 测试。所有的安装脚本和其他一些可操作的功能都应该在所有可能的计算机配置（例如：硬件、操作系统、外围设备、网络）下进行过完整的操作。

原则#3：技术支持必须在软件交付之前就确定下来。最终用户希望在问题发生时能得到及时的响应和精确的信息。如果技术支持跟不上或者根本就没有技术支持，那么客户立即会十分不满意。支持应该是有计划的，准备好支持的材料并且建立适当的记录保持机制，这样软件开发团队就能按照支持请求种类进行分类评估。

148

原则#4：必须为最终用户提供适当的说明材料。软件开发团队交付的不仅仅是软件本身，也应该提供培训材料（如果需要的话）和故障解决方案，还应该发布关于“本次增量与以前版本有何不同”的描述¹²。

原则#5：有缺陷的软件应该先改正再交付。迫于时间的压力，一些软件组织会交付一些低质量的增量，还在增量中向客户提出警告：“这些缺陷将在下次发布时解决。”这样做是错误的。在软件商务活动中有一条谚语：“客户在几天后就会忘掉你所交付的高质量软件，但是他们永远忘不掉那些低质量的产品所出现的问题。软件会时刻提醒着问题的存在。”

已交付的软件会为最终用户提供一些好处，同时它也会为软件开发团队提供一些有用的反馈。当一个增量投入使用后，应该鼓励最终用户对软件的功能、特性以及易用性、可靠性和其他特性做出评价。软件开发团队应该收集和记录用户反馈，以便（1）对交付的增量及时做出修改；（2）确定要合并到下一个增量中的变更；（3）针对这些变更相应地修改设计；（4）为下一个增量修订计划以反映变更（包括交付时间表）。

¹² 在沟通活动中，软件开发团队应该确定用户想要什么样的帮助材料。

TASK SET**通用部署任务集**

1. 创建交付介质。
 - 装配和测试所有可执行文件。
 - 装配和测试所有数据文件。
 - 创建和检查所有用户文档。
 - 完成电子版（例如pdf格式）。
 - 完成超文本链接“帮助”文件。
 - 完成故障处理指导。
 - 同小部分用户代表共同测试交付介质。
2. 确定支持人员或团队。
 - 创建文档和（或）计算机所支持的工具。
 - 建立联系机制（例如网站、电话、电子邮件）。
 - 建立问题日志管理机制。
 - 建立问题报告机制。
 - 建立问题（错误）报告数据库。
3. 确立用户反馈机制。
 - 定义反馈过程。
 - 定义反馈的方式（书面的和电子的）。
 - 建立反馈数据库。
 - 定义反馈评估过程。
4. 向用户公布要交付的介质。
5. 建立不断发展的支持功能。
 - 提供安装和初始支持。
 - 提供解决问题的持续支持。
6. 收集用户反馈。
 - 记录反馈。
 - 评估反馈。
 - 与用户就反馈问题进行沟通。

149

5.7 小结

软件工程实践包括概念、原则、方法和在整个软件开发过程中所使用的工具。虽然每个软件工程项目是不同的，但却有着通用的普遍原则和一些与项目或产品无关的适用于每个过程框架活动的任务。

要实施成功的软件工程实践，掌握技术和管理精髓是必需的。技术精髓包括理解需求、建立不确定部分的原型、确定软件架构和计划构件的集成。管理精髓包括确定优先级、确定进度表、管理风险、为项目质量和变更确定适当的项目控制方式。

在开发者与客户进行沟通时，客户沟通原则主要着眼于两点：减少摩擦和扩大双方的交流广度，双方必须互相协作以更好地交流。

策划原则全部着眼于为了使开发整个系统或产品沿着最佳路线前进提供指导。计划应该为每个软件增量而设计，或者为整个项目而制定。无论如何，计划都必须涉及要做什么，谁来完成以及什么时候完成。

建模包括分析和设计，描绘了逐渐细化的软件。建模的目的是加深对所要完成工作的理解并为软件开发人员提供技术指导。

构造包括了编码和测试循环，循环过程包括为每个构件生成源码并对其进行测试和纠错。集成则是将各个单独的构件结合起来以及对集成的模块进行功能性测试。编码原则定义了一些通用动作，这些动作应该发生在编写代码之前，在编码开始之前这些动作应该已经完成了。尽管有许多的测试原则，但是只有一个是最主要的：测试是一个为了发现错误而执行程序的过程。

在软件开发的进化过程中，部署发生在向客户展示每个软件增量的时候。交付的关键原则是满足客户期望并且能为客户提供合适的软件信息支持。支持需要充分准备。反馈允许客户提出一些具有商业价值的变更意见，为开发者的下一个软件工程循环迭代提供输入。
150

参考文献

- [AMB02] Ambler, S., and R. Jeffries, *Agile Modeling*, Wiley, 2002.
- [BEN99] Bentley, J., *Programming Pearls*, 2nd ed., Addison-Wesley, 1999.
- [BOE96] Boehm, B., "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, July 1996, pp. 73–82.
- [BOH00] Bohl, M., and M. Rynn, *Tools for Structured Design: An Introduction to Programming Logic*, 5th ed., Prentice-Hall, 2000.
- [DAV95] Davis, A., *201 Principles of Software Development*, McGraw-Hill, 1995.
- [FOW99] Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [GAR95] Garlan, D., and M. Shaw, "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering*, vol. I (V. Ambriola and G. Tortora, eds.), World Scientific Publishing Company, 1995.
- [HIG00] Highsmith, J., *Adaptive Software Development: An Evolutionary Approach to Managing Complex Systems*, Dorset House Publishing, 2000.
- [HOO96] Hooker, D., "Seven Principles of Software Development," September 1996, available at <http://c2.com/cgi/wikiSevenPrinciplesOfSoftwareDevelopment>.
- [HUN95] Hunt, D., A. Bailey, and B. Taylor, *The Art of Facilitation*, Perseus Book Group, 1995.
- [HUN99] Hunt, A., D. Thomas, and W. Cunningham, *The Pragmatic Programmer*, Addison-Wesley, 1999.
- [JUS99] Justice, T., et al., *The Facilitator's Fieldbook*, AMACOM, 1999.
- [KAN93] Kaner, C., J. Falk, and H. Q. Nguyen, *Testing Computer Software*, 2nd ed., Van Nostrand Reinhold, 1993.
- [KAN96] Kaner, S., et al., *The Facilitator's Guide to Preparatory Decision Making*, New Society Publishing, 1996.
- [KAR94] Karten, N., *Managing Expectations*, Dorset House, 1994.
- [KER78] Kernighan, B., and P. Plauger, *The Elements of Programming Style*, 2nd ed., McGraw-Hill, 1978.
- [KNU98] Knuth, D., *The Art of Computer Programming*, 3 volumes, Addison-Wesley, 1998.
- [MCC93] McConnell, S., *Code Complete*, Microsoft Press, 1993.
- [MCC97] McConnell, S., "Software's Ten Essentials," *IEEE Software*, vol. 14, no. 2, March/April 1997, pp. 143–144.
- [MYE78] Myers, G., *Composite Structured Design*, Van Nostrand, 1978.
- [MYE79] Myers, G., *The Art of Software Testing*, Wiley, 1979.
- [PAR72] Parnas, D. L., "On Criteria to Be Used in Decomposing Systems into Modules," *CACM*, vol. 14, no. 1, April 1972, pp. 221–227.
- [POL45] Polya, G., *How to Solve It*, Princeton University Press, 1945.

- [ROS75] Ross, D., J. Goodenough, and C. Irvine, "Software Engineering: Process, Principles and Goals," *IEEE Computer*, vol. 8, no. 5, May 1975.
- [SHA95a] Shaw, M., and D. Garlan, "Formulations and Formalisms in Software Architecture," *Volume 1000—Lecture Notes in Computer Science*, Springer-Verlag, 1995.
- [SHA95b] Shaw, M., et al., "Abstractions for Software Architecture and Tools to Support Them," *IEEE Trans. Software Engineering*, vol. SE-21, no. 4, April 1995, pp. 314–335.
- [STE74] Stevens, W., G. Myers, and L. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, 1974, pp. 115–139.
- [TAY90] Taylor, D. A., *Object-Oriented Technology: A Manager's Guide*, Addison-Wesley, 1990.
- [ULL97] Ullman, E., *Close to the Machine: Technophilia and its Discontents*, City Lights Books, 1997.
- [WIR71] Wirth, N., "Program Development by Stepwise Refinement," *CACM*, vol. 14, no. 4, 1971, pp. 221–227.
- [WOO95] Wood, J., and D. Silver, *Joint Application Design*, Wiley, 1995.
- [ZAH90] Zahniser, R. A., "Building Software in Groups," *American Programmer*, vol. 3, nos. 7–8, July–August 1990.

[151]

习题与思考题

- 5.1 为沟通活动做一些“推动性”的调查研究（使用已有的或其他的参考资料），并且为此准备一些指导方针。
- 5.2 对于软件工程有没有可推荐的其他的技术“精髓”？对每一个进行陈述并解释你为什么会把它包含进来。
- 5.3 对于软件工程有没有可推荐的其他的管理“精髓”？对每一个进行陈述并解释你为什么会把它包含进来。
- 5.4 一个重要的沟通原则说到“有准备的沟通”。在早期的工作中你都需要做哪些准备？在早期的准备中都应该产生哪些工作产品？
- 5.5 在分析建模的过程中有哪三个“域”是要考虑的？
- 5.6 在沟通活动中需要做一些“协商”方面的调查研究，并且为“协商”准备一些指导方针。
- 5.7 描述在项目时间表中应采取什么样的粒度。
- 5.8 敏捷沟通与传统的软件工程沟通有什么不同？有什么相同？
- 5.9 “继续前进”为什么是必要的？
- 5.10 在软件工程中为什么模型是很重要的？它们总是必需的吗？在你描述其必要性时会用一些修饰的词语吗？
- 5.11 试着用一段话来总结David Hooker的“软件开发的七项原则”（见5.1节）。试着用几句自己的话来提取其主要内容。
- 5.12 试着为5.6节中的编码添加几条附加的原则。
- 5.13 为什么对于软件开发团队来说反馈是至关重要的？
- 5.14 你同不同意下面的说法：“既然我们要向客户交付多个增量，那么为什么我们还要关注早期增量的质量——我们可以在今后的迭代中逐步改善这些问题。”说说你的看法。
- 5.15 什么是成功的测试？

推荐读物与阅读信息

客户沟通在软件工程中是至关重要的活动，然而很少有团队愿意花时间阅读这方面的资料。由Pardee创作的书（《To Satisfy and Delight Your Customer》，Dorset House, 1996）以及

Karten[KAR94]都提供了许多关于客户交互的有效方法和观点。沟通和策划的原则与概念在许多项目管理书籍中都有所涉及。一些比较有用的项目管理书籍包括：Hughes和Cotterell（《Software Project Management》，second edition，McGraw-Hill，1999），Phillips（《The Software Project Manager's Handbook》，IEEE Computer Society Press，1998）， McConnell（《Software Project Survival Guide》，Microsoft Press，1998）和Gilb（《Principles of Software Engineering Management》，Addison-Wesley，1988）。

实际上，每一本软件工程书中都包括了对分析、设计和测试概念及原则的有益讨论。其中的佼佼者有：Endres及其同事（《Handbook of Software and Systems Engineering》，Addison-Wesley，2003），Sommerville（《Software Engineering》，sixth edition，Addison-Wesley，2000），Pfleeger（《Software Engineering: Theory and Practice》，Prentice-Hall，2001）
152 和Schach（《Object-Oriented and Classical Software Engineering》，McGraw-Hill，2001）。软件工程原则在Davis[DAV95]的书中有非常好的整理和评述。

建模概念和原则可以参考很多着眼于需求工程和软件设计的书。Young（《Effective Requirements》，Addison-Wesley，2001）强调客户和开发人员组织联合小组进行协作需求分析。Weigert（《Software Requirements》，Microsoft Press，1999）提出很多关键的需求工程和需求管理实践。Somerville和Kotonya（《Requirements Engineering: Processes and Techniques》，Wiley，1998）讨论需求诱导的概念和技术以及其他需求工程原则。

Norman的著作（《The Design of Everyday Things》，Currency/Doubleday，1990）是每一个设计从业者的必读教材。Winograd及其同事（《Bringing Design to Software》，Addison-Wesley，1996）汇集了很多讨论软件设计实践的短文。Constantine和Lockwood（《Software for Use》，Addison-Wesley，1999）提出了与“以用户为中心的设计”相关的概念。Tognazzini（《Tog on Software Design》，Addison-Wesley，1995）提出一些有价值的哲学讨论，包括关于设计的本质、决策对于软件质量的影响、团队能力对软件的影响等。

很多书籍都关注软件构建活动的一个或多个问题。Kernighan和Plauger[KER78]写了一本关于编码风格的经典书籍，McConnell[MCC93]提出了软件构建实践的实用指导原则，Bentley[BEN99]提出大量关于编码的宝贵建议，Knuth[KNU98]写过一套三卷本关于编码艺术的书，Hunt[HUN99]提出了实用的编程原则。过去10年间，关于测试的文献大量涌现，Myers[MYE79]依旧是经典的，Whittaker（《How to Break Software》，Addison-Wesley，2002）、Kaner和他的同事（《Lessons Learned in Software Testing》，Wiley，2001）以及Marick（《The Craft of Software Testing》，Prentice-Hall，1997）的著作都提出了重要的测试概念、原则和很多实用指导。

在因特网上有大量关于软件工程实践的信息资源，与软件工程相关的最新WWW参考资料可以在SEPA网站<http://www.mhhe.com/pressman>上找到。
153

第6章 系统工程

要点浏览

概念：软件工程化之前必须了解该软件所处的外部“系统”。为此，必须确定系统整体目标，必须识别硬件、软件、人员、数据库、规程和其他系统要素的角色，而且必须对有效的需求进行提取、分析、说明、建模、确认和管理，这些活动都是系统工程的基础。

人员：系统工程师通过和客户、未来用户以及其他共利益者一起工作以理解系统需求。

重要性：不能像古语说的那样“只见树木，不见森林”。在这里，“森林”是系统，而树木是实现系统所需的技术要素（包括软件）。如果在理解系统之前就匆忙构造技术要素，毫无疑问你将犯错误

并让你的客户失望。关注树木之前，必须先了解森林。

步骤：从客户中提取信息以识别目标和详细的运行需求；进行需求分析以便对其清晰性、完整性和一致性进行评估；创建规格说明（经常结合系统模型），并由开发者和客户共同确认；最终管理需求以保证变更得到适当的控制。

工作产品：必须生成可以作为系统工程结果的系统有效表示，这种表示可以是原型、规格说明，甚至是一个符号模型，但它必须表达待建系统的运行、功能和行为特征，并达到对系统架构的透彻了解。

质量保证措施：评审所有系统工程工作产品，考察其清晰性、完整性和一致性。同样重要的是预测系统需求的变更，并使用可靠的方法来管理变更（参看第27章）。

关键概念
业务过程工程
宏要素
产品工程
系统
架构
特征
要素
层次
建模
仿真
模板
UML模型

约500年前，Machiavelli曾经说：“没有什么会比在事物新秩序引入阶段担任领导更加困难，行动更加危险，成功更加渺茫。”最近50年间，基于计算机的系统引入了全新的秩序，尽管自Machiavelli说这句话以来科技有了巨大的飞跃，但是他的话依然是真知灼见。

软件工程从系统工程演变而来。相比以前只专注于软件个体的情况，系统工程关注各种要素的分析、设计，并将其组织成系统，这里的系统可以是产品、服务、信息转换或控制技术。

系统工程过程在不同的应用领域有不同的表现形式：当工作集中于某业务企业时，业务过程工程就会发挥作用；而关注产品（包括从无线电话到空中交通控制系统的任何东西）生产的过程称为产品工程。

无论是业务过程工程还是产品工程，都试图将规则带入到基于计算机的系统的开发中。尽管二者有不同的应用领域，但是它们都力求将软件融入其中。也就是说，业务过程工程与产品工程¹都为计算机软件开辟了发展的空间，同时，在软件应用与基于计算机的系统的其他要素之间建立了紧密的联系。

¹ 实际上，这种情况下经常使用系统工程这个词。不过，出于本书需要，我们将系统工程的概念一般化，使之同时包含了业务过程工程和产品工程。

在这一章中，我们主要关注管理问题和具体过程活动，这两方面保证软件组织能够用正确的方式在正确的时间做正确的事。

6.1 基于计算机的系统

“系统”一词可能是技术词汇中使用频率最多、范围最广的术语。我们常说政治系统、教育系统、飞行控制系统、制造系统、银行系统、地铁系统等等。从这个词只能得到很少的信息，因此需要用形容词来描述“系统”，以便知道这个词的使用环境。《韦氏字典》这样定义系统：

1. 相互联系以形成单一或有机整体的事务集合或排列；
2. 事实、原理、规则等集合按照一种顺序分类排列，来展现连接各个部分的逻辑计划；
3. 分类或排列的方法或计划；
4. 完成某件事的有效途径；方法；规程……

字典中还给出另外5种定义，目前尚未找到一个精确的同义词。系统是一个特殊的词，借鉴韦氏字典的定义，我们将基于计算机的系统定义如下：

组织在一起通过处理信息来实现预定目标的要素集合或排列。

这里的目标可以是支持某些商业运作，也可以是开发一种可以销售并产生商业价值的产品。为了达到这个目标，基于计算机的系统利用各种各样的系统要素：

155 软件。计算机程序、数据结构和一些相关的工作产品，用以实现所需的逻辑方法、规程或控制。

硬件。提供计算能力的电子设备，支持数据流的互连设备（如网络交换机、电信设备），支持外部功能的机电设备（如传感器、发动机、泵等）。

人员。软件、硬件的使用者和操作员。

数据库。一个大型有组织的信息集合体，可以通过软件访问并持久存储。

文档。对系统使用和（或）操作进行描述的信息（如系统模型、规格说明、使用手册、在线帮助文件、Web网站）。

规程。定义每个系统要素或其外部相关流程的具体使用步骤。

这些要素通过各种各样的途径结合起来处理信息。例如，市场营销部门通过处理原始销售数据可以勾画出某产品典型消费者的特征；一个机器人将包含控制指令的命令文件转化为引起特定物理动作的控制信号序列。无论是创建辅助市场营销部门工作的信息系统，还是创建支持机器人的控制软件，它们都需要系统工程。

“不要信赖那些大到不能扔出窗子的计算机。”

—Steve Wozniak



基于计算机的系统的复杂特征在于，组成一个系统的要素还可以表示更大系统中的一个宏要素。宏要素是指基于计算机的系统，它作为更大的基于计算机的系统的一部分。例如，工厂自动化系统本质上是一个分层系统。这个分层系统的底层有数控机床、机器人和数据输入设备，它们各自都是基于计算机的系统。数控机床的要素包括电子和机电设备（例如，处理器、存储器、电机、传感器）、软件（用来交换数据、控制机器）、人员（机器操作员）、

单个数据库（存储NC程序）、文档以及规程，机器人和数据输入设备也可以按照类似方法分解。

这个分层体系的下一层定义了制造单元。制造单元也是基于计算机的系统，它有自己的要素（如计算机、机械夹具），同时也集成了一些作为宏要素的数控机床、机器人和数据输入设备。

总之，制造单元及其宏要素都由一些通用要素组成：软件、硬件、人员、数据、过程和文档。有些情况下宏要素共享某些通用要素，如机器人和数控机床都由一个操作员（人员要素）控制，而在有些情况下通用要素为某个系统独享。156

系统工程师的任务是根据系统总的层次结构为一个特定系统定义要素（宏要素）。在以下部分，我们将阐述计算机系统工程的任务。

6.2 系统工程层次结构

WebRef

国际系统工程委员会 (INCOSE) 提供了许多有用的资源，请参见：www.incose.org

不管其所关注领域，系统工程师围绕一系列自顶向下、自底向上的方法遍历图6-1所示的层次。系统工程过程经常从“全局视图”开始，也就是检查整个业务或者产品领域，以保证可以建立适当的业务或技术环境。细化全局视图，完全集中于所关心的具体领域，在这个确定领域上分析出所需的系统要素（如数据、软件、硬件、人员），最后开展目标系统分析、设计和构建活动。在系统层次顶端确定一个较为广阔的环境，而详细的技术活动通过相关工程学科原则（如硬件或软件工程）在底层具体处理²。

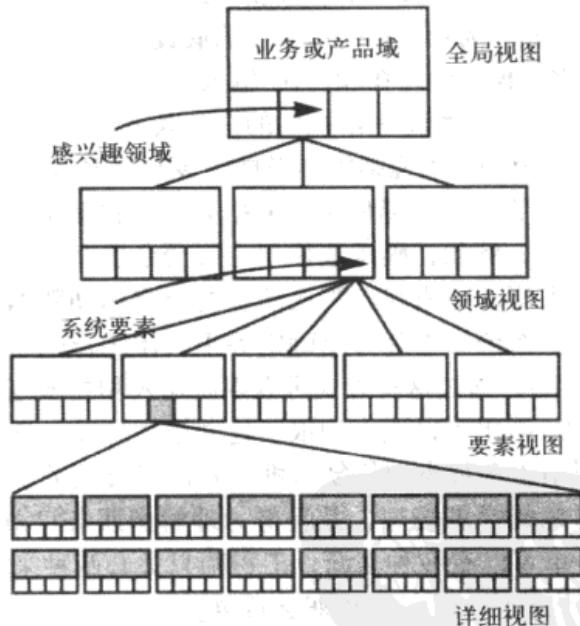


图6-1 系统工程层次图

以略微形式化来看，这种全局视图（world view, WV）由一个领域集合组成 (D_i)，它们各自都是一个系统或是大系统中的子系统。

$$WV = \{D_1, D_2, D_3, \dots, D_n\}$$

² 然而在有些情况下，系统工程必须先考虑单个的系统要素。这种方法中，通过自底向上的方法首先详尽考虑子系统的所有部件。



好的系统工程从对外界环境（全局视图）的清楚了解开始，然后逐步缩小关注点，直至理解技术细节。

每个领域都由特定要素 (E_j) 组成，各自在完成某领域或其组成部分目标的过程中扮演一些角色。

$$D_i = \{E_1, E_2, E_3, \dots, E_m\}$$

最后，每种要素通过完成特定功能的构件 (C_k) 来实现。

$$E_j = \{C_1, C_2, C_3, \dots, C_k\}$$

在软件范畴中，一个构件可以是一段计算机程序、一个可复用的计算机构件、一个模块、一个类或对象，甚至还可以是一个编程语言语句。

“一定要在更大的层次空间内考虑事物的设计——椅子处于房间内，房间处于建筑中，建筑处于某种环境里，环境是城市规划的一部分。”

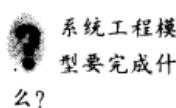
——Eliel Saarinen

157

需要重点说明的是，系统工程师在上述层次中逐步下移，将使工作重点更加集中。而在特定环境中，全局视图描绘一个总体的、功能性的清晰定义，帮助工程师理解该领域并最终理解系统或产品本身。

6.2.1 系统建模

系统建模是系统工程过程中的重要要素。无论重点在全局视图上还是在局部视图上，工程师都要建立如下模型：



158

- 定义在所考虑视图中满足需要的过程。
- 描述过程行为和该行为所依据的假设。
- 明确定义模型的外在和内在³输入。
- 描述有助于工程师理解视图的全部联系（包括输出）。

建立系统模型，工程师需要考虑很多制约因素。

1. 假设。假设能够降低可能的排列和变化数量，使模型能以合理的方式反映问题特征。例如，娱乐业使用三维着色工具建立实际动画，这种产品可以实现三维人形动画表现，真人演员的特定运动数据通过录像或人工绘制图形完成输入。系统工程师可以确定关于人体运行范围的假设条件（比如说腿不能环绕躯干），通过这种方式可以限制输入和处理范围。

2. 简化。简化可以让人们很快建立模型。以一家出售办公用品的公司为例，该公司销售各种类型的复印机、扫描仪及相关设备和服务，系统工程师建立服务组织的需求模型，并且努力理解由服务订单而产生的大量信息流。尽管服务订单可以有多种来源，工程师只关心两种来源：内部要求和外部需求，这种简化的输入能够产生服务订单。

3. 限制。限制用来确定系统的边界。例如，为下一代飞机的控制系统构建模型，由于飞机设计有两个发动机，那么推进监控系统模型应该可以适应最多两台发动机和相关冗余备份系统。

4. 约束。约束将会对建立模型以及实现模型时所采用路线的方式作出指导。例如，上述三维着色系统工具运行于双G5处理器平台，那么，问题的计算复杂性必须限制在这些处理器的处理极限范围内。



³ 外在输入连接给定视图中某一成分与同层或别层的其他成分，内在输入连接特定视图中相关的孤立部件。

5. 偏好。它为全部数据、功能和技术指明首选的架构。首选的解决办法有时会与其他约束条件相抵触。然而，客户满意程度经常以其所首选方法的实现程度为依据。

这样生成的系统模型（在各种视图上）可以称为完整的自动化解决方案、半自动化解决方案或非自动化解决方案。实际上，刻画各种不同类型模型的特征并将其作为处理手头问题的备选方案是非常有可能的。系统工程师基本可以通过简单修改不同系统要素（人员、硬件、软件）的相关内容来得到不同类型的模型。

“简单事情简单处理，复杂事情尽可能简单。”

——Alan Kay

159

6.2.2 系统仿真



如果对一个交互系统来说，没有可行的仿真能力，项目风险就会增加。考虑使用迭代过程模型，它使你能够在第一次迭代后交付一个工作产品，然后再使用其他迭代微调其性能。

许多基于计算机的系统都通过交互方式与现实世界交流。就是说，现实世界的事件都是通过组成计算机系统的硬件和软件来监控，基于这些事件，系统对机器、过程、甚至引发这些事件的人进行控制。实时和嵌入式系统常常都属于交互类型。

许多交互型控制的机器或过程（如商用飞行器或炼油厂）必需具有极高的可靠性要求，一旦系统崩溃将会造成巨大的经济和人员损失。出于这个原因，通过系统建模和模拟工具来消除建立交互系统时可能造成的意外。这些工具应用于系统工程过程中，用以确定硬件、软件、数据库和人员角色，建模和模拟工具可以使系统工程师“测试驱动”系统规格说明。

SOFTWARE TOOLS

系统仿真工具

目的：系统仿真工具给软件工程师提供一种可以在实时系统建立之前预测系统行为的能力。进一步，软件工程师可以利用这些工具开发出实时系统模型，以便客户在系统实际运行完成之前深入地把握系统的功能、操作和实际响应。

机制：这类工具可以用来定义基于计算机系统的各种要素并进行一系列不同的模拟，以便更深入地理解系统的操作特性和整体性能。主要有两类系统仿真工具：(1) 可以对任意计算机系统进行可视化仿真的通用工具；(2) 针对特定应用领域的专用工具（如飞机控制系统、制造系统、电子系统）。

代表性工具⁴

CSIM，由Lockheed Martin Advanced Technology Labs (www.atl.external.lmco.com) 开发，是一种面向模块图系统的通用离散事件仿真器。

Simics，由Virtutech (www.virtutech.com) 开发，是一种可以对基于软件和硬件的系统进行建模及分析的系统仿真平台。

SLX，由Wolverine Software (www.wolverinesoftware.com) 开发，提供通用构造模块用于许多系统的性能建模。

关于各种系统仿真工具的链接可以参看<http://www.idsia.ch/~andrea/simtools.html>。

160

⁴ 这里列出的工具并不代表本书支持这些工具，只是此类工具的示例。大多数情况下，工具的名字由其开发者注册为商标。

6.3 业务过程工程概述

业务过程工程 (business process engineering, BPE) 的目标是定义一个能有效利用信息进行业务活动的体系。从公司信息技术需求的全局角度而言，毫无疑问需要系统工程。不仅需要对合适的计算架构进行说明，而且需开发应用于组织计算资源的特定配置的软件体系结构。业务过程工程为建立实施计算架构的总体计划提供了一种方法。

在业务目标和目的的环境中，必须分析和设计如下3种不同的架构：



BPE中要定义和开发哪些架构？

- 数据架构
- 应用架构
- 技术基础设施

数据架构为业务或业务功能的信息需求提供了框架，单独建立的框架模块是被业务所用到的数据对象。一个数据对象包括用于定义不同侧面的属性集、质量、特征或数据描述符。

一旦数据对象集确定，也就确定了它们之间的关系。关系表明对象之间是如何相联系的。以消费者和产品A为例，这两个对象可以通过购买联系起来，也就是说，消费者购买了产品A或者说产品A被消费者购买了。在业务活动中流动的数据对象（在一个重要的业务活动中也许有成百上千的对象）通过数据库组织起来，为业务需要提供所需的信息。



作为一个软件工程师，你可能永远不会涉及ISP或BAA，然而，如果清楚地知道这些活动还未进行，那么通知共利益者，该项目的风险非常高。

161

应用架构包含那些为了某些业务目的而在数据架构范围内进行转换的系统要素。本书认为应用架构是执行转换的程序（软件）系统。然而，在更广的范围内，应用架构将人员角色（信息的转换者和使用者）和尚未实现自动化的业务规程联系在一起。

技术基础设施为数据架构和应用架构提供基础。基础设施包括用来支持应用和数据的硬件和软件，包括计算机、操作系统、网络、通信链路、存储技术和用于实现这些技术的架构（如客户/服务器）。

为了建立系统架构模型，定义了层次化的业务过程工程活动（如图6-2所示）。

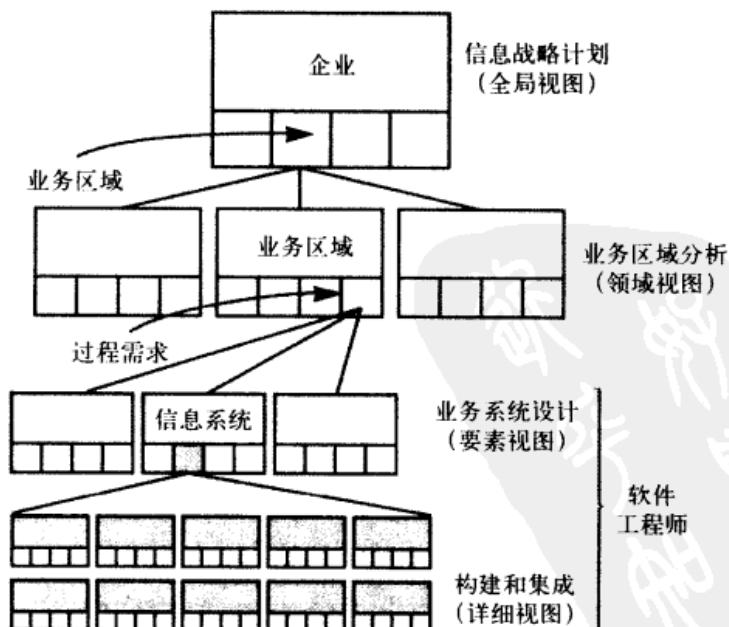


图6-2 业务过程工程层次图[MAR90]

6.4 产品工程概述

产品工程的目的是将用户期望的已定义的一组能力转变成真实产品。为了达到这个目的，产品工程——类似于业务过程工程——必须给出架构和基础设施。这个架构包括四个不同的系统构件：软件、硬件、数据（数据库）以及人员。建立支持基础设施，它包括能够把各种构件联系在一起的技术和用来支持构件的信息（如文档、CD-ROM、视频）。

如图6-3所示，全局视图由需求工程（第7章）得到。全局需求由客户引出，这些需求包括信息和控制要求、产品功能和行为、产品整体性能、设计和接口约束条件以及其他特殊要求。一旦这些需求确定下来，需求工程的工作就是将这些功能和行为分配到上述4个构件中。

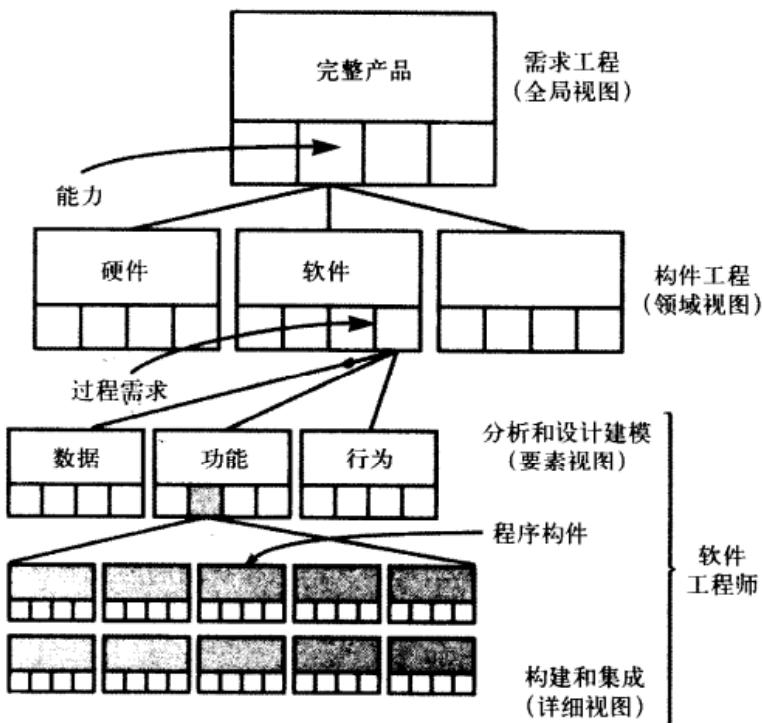


图6-3 产品工程层次图

ADVICE
并发过程模型
(第3章) 经常用
在这种环境，每
个工程师并行地
按规范要求工
作。记住，在按
规范工作的同时
还要加强沟通。

一旦开始分配，系统构件工程就开始了。系统构件工程实际上是一组并发活动，分别处理软件工程、硬件工程、人员工程和数据库工程这些系统构件。每个工程规范都用特定领域的观点来看待，但重要的是必须建立工程规范和维持相互间的积极沟通，需求工程的部分作用是建立便于沟通的接口机制。

产品工程的要素视图是应用于已分配构件的工程规范。对软件工程来说，这意味着需进行分析和设计建模活动（以后几章详细说明），以及包括编码、测试和支持任务在内的构造和部署活动。分析任务模型是将需求分配到数据、功能和行为表示中，设计将分析模型映射到数据设计、结构设计、接口设计和软件构件级设计中。

系统工程初步

[场景] SafeHome项目启动会后，软件工程团队工作地点。

[人物] Jamie Lazar、Vinod Raman、Ed Robbins，软件工程团队成员。

[对话]

Ed: 我感觉进展还算顺利。

Vinod: 是啊……但是我们只看到系统的总体情况，对这个软件来讲还有很多需求收集工作等着我们呢。
163

Jamie: 这就是为什么我们接下来要讨论未来五天计划。还有，我建议请两位客户到这儿和我们一起呆几周，进行真正的沟通和协作。

Vinod: 具体怎么做？

Jamie: 在他们眼里我像疯子一样，但Doug[软件工程经理]喜欢这种敏捷方法，他正在和客户们聊呢。

Ed: 讨论期间我用PDA做了笔记，可以整理出基本功能列表。

Jamie: 太好了，让我看看。

Ed: 我已经给你们两个每人发了一份，你们看一下我们再讨论。

Vinod: 午饭后讨论怎么样？

(Jamie和Vinod从Ed处收到以下内容)

SafeHome结构/功能草案：

- 系统将使用一台或多台PC（个人电脑）、各种壁挂和/或手持控制面板、各种传感器和用具/设备控制器。
- 上述所有设备通过无线协议（如802.11b）通信，将在新房屋或者现有房屋内应用。
- 除了新的无线盒之外，所有硬件都不需要单独定制。

项目启动会上收集到的基本软件功能如下。

住宅安全功能：

- 针对未授权进入（非法闯入），设置标准的门/窗/运动体传感器监测。
- 烟、火、一氧化碳浓度监测。
- 房屋地基水位（如洪水或水暖管爆裂）监测。
- 屋外行人运动监测。
- 通过Internet改变安全设置。

住宅监视功能：

- 连接安装在屋内/屋外的一个或多个摄像头。
- 摄像头全景/缩放控制。
- 设置摄像头监视范围。
- 在PC上显示监视图像。
- 通过Internet查看监视图像。
- 可选择数字化记录摄像内容。
- 摄像内容重放。

住宅管理功能：

- 灯光控制。
- 用具控制。
- HVAC控制。
- 房内所有视频/音频设备控制。
- “度假/外出模式”一键设置功能。
 - 设置相应的用具、灯光、HVAC。
 - 设置自动应答信息。
 - 联系供应商以停止报纸、信件等投递。

通信管理功能：

- 自动应答机功能。
 - 呼叫方列表（以ID方式）。
 - 留言及时间。
 - 通过声音识别系统得到留言文本。
- 电子邮件功能（所有标准的电子邮件功能）。
 - 标准电子邮件显示。
 - 电话访问邮件语音阅读。
- 个人电话本。
- PDA连接。

其他功能：

待定。

所有功能都可以在口令保护方式下通过因特网访问。

6.5 系统建模

由于一个系统可以在不同抽象层次（如全局视图、领域视图、要素视图）上表现，因此系统模型在本质上倾向于分级或分层。在层次的顶端，展示完整（全局视图）的系统模型，展现主要数据对象、处理功能和行为，而不考虑实现全局视图要素的系统构件。随着层次的细化或进一步划分，逐渐完成构件细节（在这里指的是硬件、软件等）建模。最后系统模型逐步演变为工程模型（进一步细化得到），而这个工程模型就是适用的工程规范。

164

6.5.1 Hatley-Pirbhai建模

每个基于计算机的系统都可以按以下方式建模，即利用“输入-处理-输出”模板进行信息转换。Hatley和Pirbhai扩展该视图以包括两个附加的系统特征——用户界面处理、维护和自检处理。尽管这些附加特征并不是在每个基于计算机的系统上都体现，但是它们非常普遍，并且它们的规格说明使系统模型更健壮。

POINT
Hatley-Pirbhai模型描述输入、处理和输出以及用户接口与维护/自检。

利用输入、处理、输出、用户界面处理和自检处理的表达方式，系统工程师可以建立一个系统构件模型，为后面每个工程规范步骤建立良好的基础。

开发系统模型可以使用系统模型模板[HAT87]。系统工程师将系统要素分派到模板内5个处理过程之一：(1) 用户界面，(2) 输入，(3) 系统功能和控制，(4) 输出，(5) 维护和自检。

与应用于系统和软件工程中的许多建模技术一样，分析人员也可以使用系统模型模板建立一个详细的层次等级。系统环境图（system context diagram, SCD）居于层次的上层，环境图“建立待实现系统和系统操作环境之间的信息边界”，也就是说，SCD确定系统所使用信息的所有外部生产者、系统所产生信息的所有外部消费者、所有通过接口交流或者执行维护和自检的实体。

为了阐明SCD的作用，考虑下面传送带分类系统（CLSS）的实际情况：

开发CLSS，使通过传送带传送的箱子在终端能被识别并分别存放到6个箱柜中。这些箱子将穿过分类站，根据印在箱子侧面的标识数字和条形码，将箱子分流到适当的箱柜里存放。箱子随机通过并且均匀排列，传送带缓慢移动。

全部CLSS软件均在分类站的台式机上执行，与条码阅读器交互读取每个箱子上的数字，与传送带检测装置交互获得传送带速度，存储所有分类数据，与分类站操作员交互产生多种报告和诊断，把控制信号送到分流硬件以便给箱子分类，并与中央工厂自动化系统联系。

165

SCD和CLSS如图6-4所示。图中分成5个重要部分，顶端部分代表用户界面处理，左边和右边部分分别描述输入和输出处理，中间部分包含处理和控制功能，底层部分注重维护和自检。图中每个框都代表一个外部实体——即生产者或消费者的系统信息，例如条码阅读器产生的信息输入到CLSS系统。整个系统（或者主要的子系统）由圆角矩形符号表示。因此，在SCD中央表示CLSS的处理和控制区域。SCD中箭头标记表明从外部环境进入到CLSS系统中的信息（数据和控制）。外部实体条码阅读器产生用于标记条码的输入信息。本质上，SCD将任意系统都置于其外部环境中。

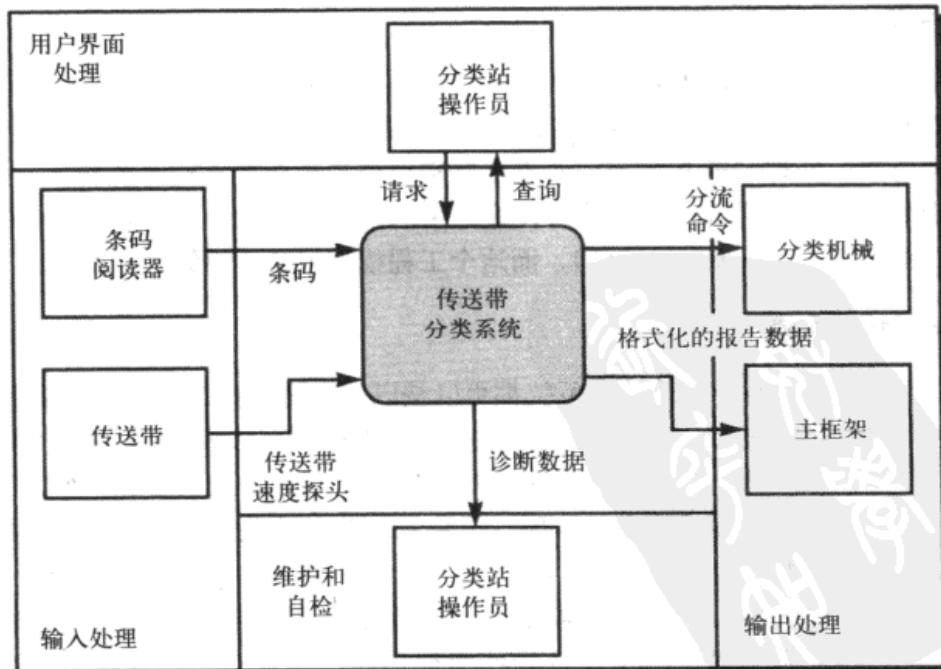


图6-4 CLSS系统环境图

系统工程师通过详细考虑图6-4中带阴影的矩形来细化系统环境图。环境中为SCD所确认的主要子系统保证传送带分类系统能够正常运作，系统流图（system flow diagram, SFD）中定义的主要子系统都从SCD中导出，流经SCD区域的信息流用于指导系统工程师开发系统流图——一个更详细的CLSS描述。系统流图展示主要子系统和重要信息（数据和控制）流。另外，系统模板将子系统处理划分成前面提到的5个区域，在这一层，每个子系统都包含一个或多个由系统工程师分派的系统要素（如硬件、软件、人员）。

初始系统流图成为SFD层次的顶层结点。原始SFD中的每一个圆角矩形都可以扩展到另一个单独展示的架构模板，这个过程在图6-5中描述。每个系统SFD都可以作为（用于描述子系统的）后续工程步骤的起点。

子系统和在子系统之间流动的信息可以指定/绑定到后续工程工作中，每个子系统的详细描述和其间流动的数据的定义是“系统规格说明”的重要要素。

6.5.2 UML系统建模

UML提供了大量图表表示法，它们用于在系统和软件层次进行分析和设计⁵。对CLSS系统来说，需要对4个重要的系统要素进行建模：（1）支持CLSS的硬件；（2）实现数据库访问和产品分类的软件；（3）向系统提交各种请求的操作员；（4）保存相关条码和目的信息的数据库。

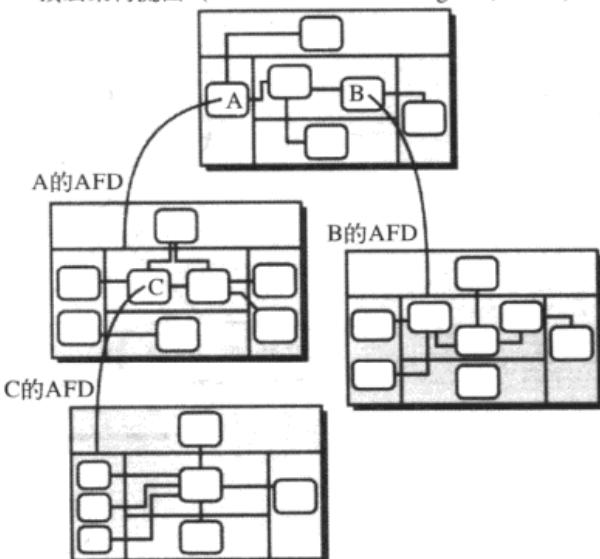
UML部署图建立CLSS在系统层次的硬件模型，如图6-6所示，每个三维方盒描述一个属于系统物理架构的硬件要素。有些情况下，硬件要素需要作为项目的一部分来构建，而更多的情况是采用现成的硬件要素。对于工程师团队来说，最大的挑战是设计良好的硬件接口。

CLSS的软件可以用各种各样的UML图来描述。CLSS软件的操作规程方面可以用类似于流程图的活动图（图6-7）来描述，以表现系统实现各种功能时的具体步骤，圆角矩形表示特定的系统功能，箭头表示系统的流程，菱形表示分支（从菱形出发的箭头都需要标注），水平实线表示并发事件。

WebRef

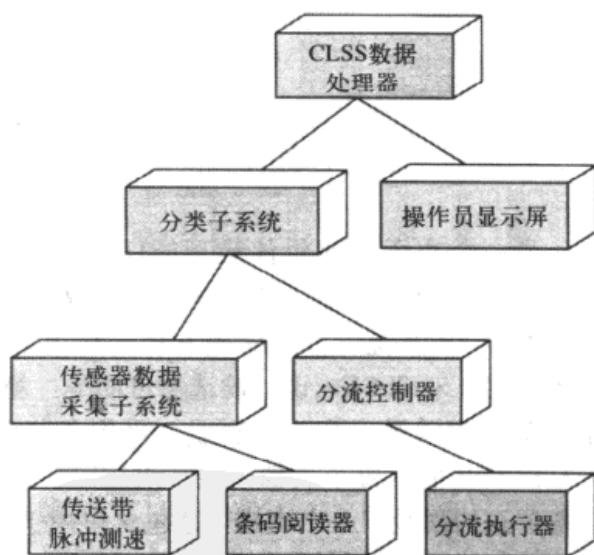
关于UML词法
和语法（将在后
面章节中讨论）
完整的规格说明
可以查找站点：
[www.rational.com/
uml/index.jsp](http://www.rational.com/uml/index.jsp)

顶层架构流图（archecture flow diagram, AFD）



166

图6-5 构建SFD层次图



167

图6-6 CLSS硬件部署图

⁵ 更详细的关于UML图的描述参见第8~11章，有兴趣的读者可以阅读[SCH02]、[LAR01]或[BEN99]。

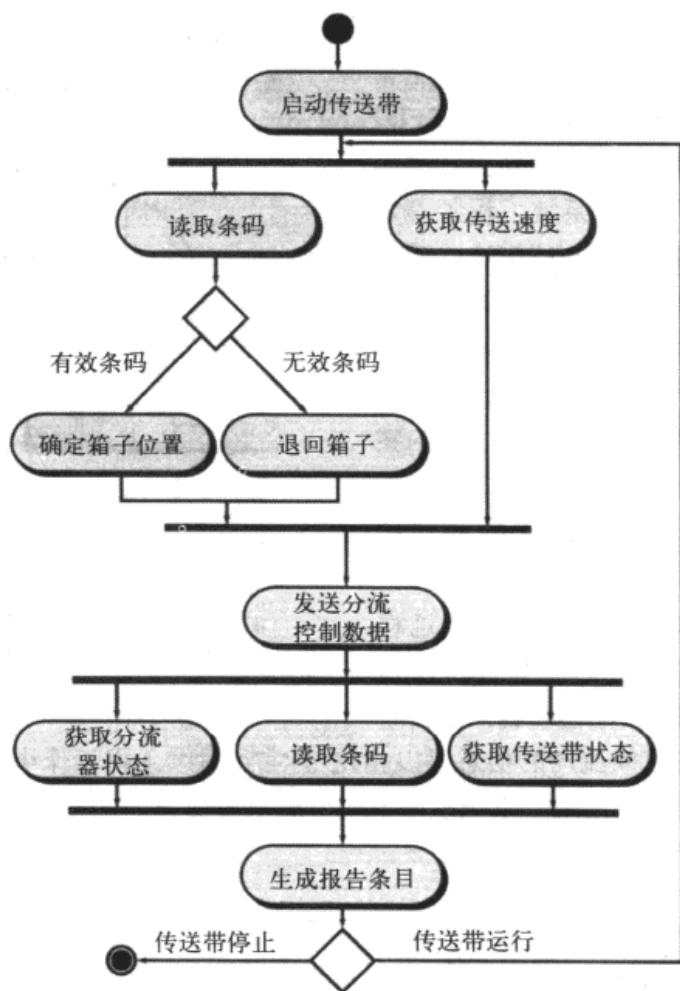


图6-7 CLSS活动图

168

另一个用于构建软件模型的UML表示法是类图(class diagram)(以及本书后面章节会提到各种与类相关的图)。在系统工程师层次上看，类⁶是从问题描述中提取的。对CLSS来说，候选类可能是：箱子、传送带、条码阅读器、分流控制器、操作者请求、报告、产品以及其他。每个类都封装一组用以说明该类所有必备信息的属性，一个类描述还包含应用于CLSS系统环境中的类的一组操作。Box(箱子)类的UML类图如图6-8所示。

CLSS操作员可以用图6-9所示的UML用例图来建模。用例图阐述了一个参与者(这个用例中是操作员，用火柴棍小人表示)与系统的交互行为。方框(表示CLSS系统的边界)里的每个椭圆都表示一个用例，用例用文字来描述角色与系统的交互场景。

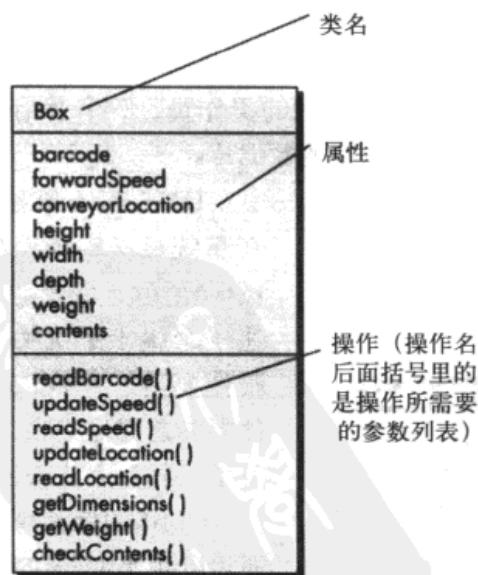


图6-8 CLSS类图

⁶ 前面几章我们曾指出，类描述构成问题域一部分的一组实体。这些实体可以供系统处理或存储，也可以作为系统所产生信息的操作者或使用者。

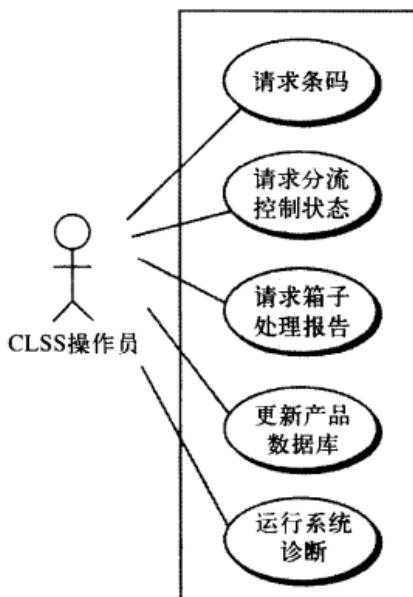


图6-9 CLSS用例图

169
170**SOFTWARE TOOLS****系统建模工具**

目的：系统建模工具使用工具专用的符号提供建模能力，以便软件工程师为基于计算机的系统中的所有要素建模。

机制：这类工具的机制差别很大。一般情况下，这类工具可对以下内容建模：（1）系统所有功能要素的结构；（2）系统静态和动态行为；（3）人机交互界面。

代表性工具⁷

Describe，由Embarcadero Technologies (www.embarcadero.com) 开发，这是一套可用于软件或整个系统的UML建模工具。

Rational XDE和Rose，由Rational Technologies (www.rational.com) 开发，它为基于计算机的系统提供了一套UML建模和开发工具集。

Real-Time Studio，由Artisan Software (www.artisansw.com) 开发，这是一套支持实时系统建模和开发的工具。

Telelogic Tau，由Telelogic (www.telelogic.com) 开发，这是一套支持分析与设计建模的UML建模工具，还能链接到软件构建特征。

6.6 小结

一个高科技系统包含很多要素：硬件、软件、人员、数据库、文档以及操作规程。系统工程师将用户的要求转换成由一个或多个上述要素构成的系统模型。

系统工程开始于“全局视图”。分析业务领域或者产品以建立全部基本业务需求，然后焦点逐渐移到“领域视图”上，在领域视图中独立分析每个系统要素。每个要素都分成一个或

⁷ 这里列出的工具并不代表本书支持这些工具，而只是这类工具的示例。在大多数情况下，工具的名字由其开发者注册为商标。

多个工程构件，然后按照相应工程规范来处理。

业务过程工程是一种系统工程方法，用以确定能够有效使用信息的业务架构。业务过程工程的目的是提供易于理解的数据架构、应用架构，还有满足业务战略和每个业务领域目标的基础设施。

产品工程是从系统分析开始的系统工程，系统工程师确认用户需求，确定经济和技术可行性，将功能和性能分配到软件、硬件、人员、数据库等关键工程构件中。
[171]

参考文献

- [BEN99] Bennett, S., S. McRobb, and R. Farmer, *Object-Oriented Systems Analysis and Design Using UML*, McGraw-Hill, 1999.
- [HAR93] Hares, J. S., *Information Engineering for the Advanced Practitioner*, Wiley, 1993, pp. 12–13.
- [HAT87] Hatley, D. J., and I. A. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, 1987.
- [LAR01] Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2nd ed., Prentice-Hall, 2001.
- [MAR90] Martin, J., *Information Engineering: Book II—Planning and Analysis*, Prentice-Hall, 1990.
- [MOT92] Motamarri, S., "Systems Modeling and Description," *Software Engineering Notes*, vol. 17, no. 2, April 1992, pp. 57–63.
- [SCH02] Schmuller, J., *Teach Yourself UML in 24 Hours*, 2nd ed., Sams Publishing, 2002.
- [SPE93] Spewak, S., *Enterprise Architecture Planning*, QED Publishing, 1993.
- [THA97] Thayer, R. H., and M. Dorfman, *Software Requirements Engineering*, 2nd ed., IEEE Computer Society Press, 1997.

习题与思考题

- 6.1 任选一个你比较熟悉的大型系统或产品。定义描述系统或产品全局视图的一组领域，描述组成该领域的一组要素，对其中一个要素，确定必须工程化的技术构件。
- 6.2 对你熟悉的系统、产品或服务，建立它们的层次系统。层次应该向下扩展到简单系统要素（硬件、软件等），至少得到层次树的一个分支。
- 6.3 尽管这时的信息还比较粗略，请你尽力建立SafeHome系统的UML部署图、活动图、类图和用例图。
- 6.4 业务过程工程试图定义数据架构和应用架构以及技术基础设施。描述每个术语的含义并举例。
- 6.5 系统工程师有3种来源：系统开发人员、用户或一些外部组织。讨论一下每种来源的利与弊。描述一个理想的系统工程师。
- 6.6 你的导师要求大家给出一个基于计算机的系统或产品的高层描述：
 - a. 假定你是一个系统工程师，你会问哪些问题。
 - b. 基于你的问题答案，计划出至少2种系统划分的方案。
 - c. 在课堂上与其他同学比较方案。
- 6.7 任选一个你比较熟悉的系统或产品，描述出建造有效系统模型必需的假设、简化、限制、约束和偏好。
- 6.8 研究文献并写出一篇简短文章描述建模和模拟工具是如何工作的。或者是收集两个或更多的商用建模和模拟工具的文献，并且比较它们的相似处与不同处。
- 6.9 尽你所能找出“系统”的同义词。

- 6.10 有不能在系统工程活动期间建立的系统特性吗？如果有的话，描述其特性并解释一下为什么它们需要考虑的事项要延迟到后面的工程步骤中。
- 6.11 为你选择（或老师指定）的基于计算机的系统开发一个系统环境图。 [172]
- 6.12 有没有这样的情况，正式的系统规格说明可以简化或完全省略。请解释你的理由。

推荐读物与阅读信息

Hatley和他的同事（《Process for Systems Architecture and Requirements Engineering》，Dorset House, 2000）、Buede（《The Engineering Design of Systems: Models and Methods》，Wiley, 1999）、Weiss和他的同事（《Software Product-Line Engineering》，Addison-Wesley, 1999）、Blanchard和Fabrycky（《System Engineering and Analysis》，third edition, Prentice-Hall, 1998）、Armstrong和Sage（《Introduction to Systems Engineering》，Wiley, 1997）以及Martin（《System Engineering Guidebook》，CRC Press, 1996）介绍了系统工程过程（非常强调工程化），并提供了重要的指导。Blanchard（《System Engineering Management》，second edition, Wiley, 1997）和Lacy（《System Engineering Management》，McGraw-Hill, 1992）讨论了系统工程管理问题。

Chorafas（《Enterprise Architecture and New Generation Systems》，St. Lucie Press, 2001）介绍了信息工程和系统体系结构的“下一代”IT解决方案，包括基于Internet的系统。对于信息系统和产品，Wallnau和他的同事（《Building Systems from Commercial Components》，Addison-Wesley, 2001）讨论了基于构件的系统工程问题。Lozinsky（《Enterprise-Wide Software Solutions: Integration Strategies and Practices》，Addison-Wesley, 1998）讨论了如何使用软件包作为解决方案，这种解决方案可以使公司从遗产系统向现代商业过程迁移。Bradley（《Elimination of Risk in Systems》，Tharsis Books, 2002）对风险和系统工程做了重要讨论。

Davis（《Business Process Modeling with Aris: A Practical Guide》，Springer-Verlag, 2001）、Bustard和他的同事（《System Models for Business Process Improvement》，Artech House, 2000）以及Scheer（《Business Process Engineering: Reference Models for Industrial Enterprises》，Springer-Verlag, 1998）描述了企业级信息系统的业务过程建模方法。

Davis和Yen（《The Information System Consultant's Handbook: Systems Analysis and Design》，CRC Press, 1998）以百科全书式的方式全面介绍了信息系统领域中的信息系统分析与设计问题。由Thayer和Dorfman[THA97]编写的优秀的IEEE教程讨论了系统和软件级需求分析问题之间的关系。

Law和他的同事（《Simulation Modeling and Analysis》，McGraw-Hill, 1999）讨论了为多种应用领域进行系统模拟和建模的技术。

对于那些在系统工作中很活跃，或者对系统工作的更高级的处理感兴趣的读者来说，Gerald Weinberg的书（《An Introduction to General System Thinking》，Wiley-Interscience, 1976；《On the Design of Stable Systems》，Wiley-Interscience, 1979）已经成为经典著作，此书对“通用系统思想”进行了出色的讨论，这种系统思想无疑引导了系统分析与设计的通用方法。Weinberg新出版的书（《General Principles of Systems Design》，Dorset House, 1988；《Rethinking Systems Analysis and Design》，Dorset House, 1988）延续了他早期工作的风格。

系统工程和相关主题的大量信息源可在Internet上获得。最新的与系统工程、信息工程、业务过程工程和产品工程相关的WWW参考文献可在SEPA Web站点<http://www.mhhe.com/pressman>上找到。 [173]

第7章 需求工程

要点浏览

概念：需求工程帮助软件工程师更好地理解他们将要解决的问题。其中所包含的一系列任务有助于理解软件将如何影响业务、客户想要什么以及最终用户将如何和软件交互。

人员：软件工程师（有时涉及系统工程师或分析师）和项目共利益者（项目经理、客户、最终用户）都将参与需求工程。

重要性：在设计和开发某个一流的计算机软件时，如果软件解决的问题不对，那么即使软件再精巧也满足不了任何人的要求。这就是在设计和开发一个基于计算机的系统之前，理解客户需求的重要性所在。

步骤：需求工程首先定义将要解决的问

题范围和性质；然后是引导、帮助客户定义需要什么；接下来是精练需求，精确定义和修改基本需求。当客户提出问题后，就要进行协商：优先次序如何定义？什么是必需的？何时需要？最后，以某种形式明确说明问题，再经过评审或确认以保证我们和客户对于问题的理解是一致的。

工作产品：需求工程的目的在于为各方提供关于问题的一个书面理解。这可以通过如下工作产品得到：用户场景、功能和特征列表、分析模型或规格说明。

质量保证措施：客户和最终用户将评审需求工程的工作产品，以确保我们所理解的是他们真正想要的。需要提醒大家的是：即使参与各方均认可，事情也将变化，而且变化可能贯穿整个项目过程。

关键概念
分析模型元素
分析模式
精化
导出
起始
小规格说明
谈判
QFD
需求管理
规格说明
可追溯性
用例
确认

理解问题的需求是软件工程师所面对的最困难的任务之一。第一次考虑需求工程的时候，可能看起来并不困难。但是，客户难道不知道需要什么？最终用户难道对将给他们带来实际收益的特征和功能没有清楚的认识？不可思议的是，很多情况下的确是这样的。甚至即使用户和最终用户清楚地知道他们的要求，这些要求也会在项目的实施过程中改变。因此实施需求工程非常困难。

在Ralph Young[YOU01]的一本关于有效需求实践的书的前言中我写到：

这是最恐怖的噩梦：一个客户走进你的办公室，坐下，正视着你，然后说：“我知道你认为你理解我说的是什么，但你并不理解的是我所说的并不是我想要的。”这种情况总是在项目后期出现，而当时的情况通常是：已经对交付期限作出承诺，声誉悬于一线并且已经投入大量资金。

我们这些已经在系统和软件业中工作多年的人就生活在这样的噩梦中，而且目前还都不知道该怎么摆脱。我们努力从客户那里引导出需求，但是难以理解获取的信息。我们通常采用混乱的方式记录需求，而且总是花费极少的时间检验到底记录了什么。我们容忍变更

控制我们，而不是建立机制去控制变更。总之，我们未能为系统或软件奠定坚实的基础。这些问题中的每一个都是极富挑战性的，当这些问题集中在一起时，即使是最有经验的管理者和实践者也会感到头痛。但是，确实存在解决方法。

把需求工程称为上述挑战的“解决方案”并不诚实，但需求工程确实为我们提供了解决问题的可靠途径。

7.1 连接设计和构造的桥梁

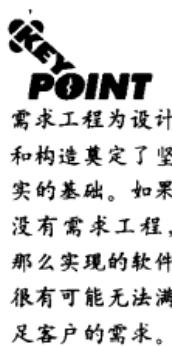
设计和编写软件富有挑战性、创造性和趣味性。事实上，编写软件是如此吸引人，以至于很多软件开发人员在清楚地了解用户需要什么之前就迫切地投入到编写工作中。开发人员认为：在编写的过程中事情总是会变得清晰；只有在检验了软件的早期版本后项目共利益者才能够更好地理解要求；事情变化太快，需求工程是浪费时间；底线是开发一个可运行的程序，其他都是次要的。构成这些论点的原因在于其中也包含了部分真实情况¹，但是这中间的每个论点都存在一些小问题，汇集成整体就可能导致软件项目的失败。

“构建一个软件系统最困难的部分是确定构建什么。其他部分工作不会像这部分工作一样，在出错之后会如此严重地影响随后实现的系统，并且在以后修补竟会如此的困难。”

——Fred Brooks

需求工程和其他软件工程活动类似，必须适应过程、项目、产品和工作人员的要求。从软件过程的角度来看，需求工程（Requirement Engineering, RE）是一个软件工程动作，开始于沟通并持续到建模。175

某些情况下，可能会选择简洁的方法。还有一些情况下，必须严格执行全面的需求工程所定义的每项任务。总的来说，软件团队必须使其方法适应需求工程。但是适应并不意味着放弃，在软件团队尝试解决问题之前进行实际的工作以理解问题的需求，这是十分必要的。



需求工程在设计和构造之间建立起联系的桥梁。桥梁源自何处？有人可能认为开始于项目共利益者（如管理人员、客户、最终用户），也就是在那里定义业务需求，刻画用户场景，描述功能和特性，识别项目约束条件。其他人可能会建议从宽泛的系统定义开始，此时软件只是更大的系统范围中的一个构件（第6章）。但是如果不管起始点在哪里，横跨这个桥梁将把我们带到项目之上更高的层次：由软件团队检查将要进行的软件工作的内容；必须提交设计和构建的特定要求；完成指导工作顺序的优先级定义；将深切影响随后设计的信息、功能和行为。

7.2 需求工程任务

需求工程为以下工作提供了良好的机制[THA97]：理解客户需要什么，分析要求，评估可行性，协商合理的方案，无歧义地详细说明方案，确认规格说明，管理需求以至将这些需求

¹ 对小项目（不超过一个月）和只涉及简单软件工作的更小项目，这确实是正确的。随着软件规模和复杂性的增加，这些论点就开始出问题。



可以在需求阶段做一些设计工作，在设计阶段做一些需求工作。

转化为可运行系统。需求工程过程通过执行七个不同的活动来完成：起始、导出、精化、协商、规格说明、确认和管理。

重要的是注意到，某些需求工程的活动并行发生且要全部适应项目的要求。所有的努力都是为了确定客户想要什么，所有的工作都是为设计和构造客户希望的软件奠定一个坚实的基础。

7.2.1 起始

如何开始一个软件项目？有没有一个独立的事件成为新的基于计算机的系统或产品的催化剂？或是要求会随时间流逝而发展呢？这些问题没有确定的答案。

“大多数软件灾难的种子通常都是在软件项目开始的头三个月内种下的。”

—Capers Jones

[176] 某些情况下，偶然的一次交谈就可能必须投入大量的软件工程工作。但通常都是当确定了商业要求或发现了潜在的新市场、新服务时项目才开始。业务领域的共利益者（如业务管理人员、市场营销人员、产品管理人员）定义业务用例，确定市场的宽度和深度，进行粗略的可行性分析，并确定项目范围的工作说明。所有这些信息都取决于变更（很可能的结果），但是应该充分地与软件工程组织²及时讨论。

在项目起始阶段³，软件工程师会询问一些似乎与项目无直接关系的问题（见7.3.4节）。目的是对问题、方案需求方、期望方案的本质、客户和开发人员之间初步的交流和合作的效果建立基本的谅解。

7.2.2 导出

这无疑似乎非常简单——询问客户、用户和其他人，系统或产品的目标是什么？想要实现什么？系统和产品如何满足业务的要求，最终系统或产品如何用于日常工作？但是，实际上并非如此简单——这非常困难。

Christel和Kang[CRI92]指出了一系列问题，这可以帮助理解为什么导出需求这么困难：

清晰地理解
客户的需求
为什么很困难？

- **范围问题。**系统的边界不清楚，或客户/用户的说明带有多余的技术细节，这些细节可能会混淆而不是澄清系统的整体目标。
- **理解问题。**客户/用户并不完全确定需要什么，对其计算环境的能力和限制所知甚少，对问题域没有完整的认识，与系统工程师在要求沟通上有问题，省略那些他们认为是“明显的”信息，确定的需求和其他客户/用户的需求相冲突，需求说明有歧义或不可测试。
- **易变问题。**需求随时间变化。

为了帮助解决这些问题，需求工程师必须以有条理的方式开展需求收集活动。

² 如果要开发一个基于计算机的系统，讨论将从系统工程开始，涉及关乎各领域的全局以及系统所在的领域活动（参阅本书6.2节）。

³ 第3章的读者可以回忆一下，统一过程定义了更全面的“起始阶段”，包括本章所讨论的起始、导出和精化任务。

7.2.3 精化



精化虽然好，但你必须知道何时停止精化。关键在于以某种能为设计奠定坚实基础的方式说明问题即可。如果工作超出了这一点，那么就是在做设计。

在起始和导出阶段获得的信息将在精化阶段扩展和提炼。该阶段需求工程活动集中于开发一个精确的技术模型，用以说明软件的功能、特征和约束。

177

精化是一个分析建模动作（第8章），由一系列的建模和求精任务构成。当描述最终用户（以及其他角色）如何与系统交互的用户场景创建和求精时，就会发生精化动作，每个用户场景被分解为精炼分析类——最终用户可见的业务域实体。应该定义每个分析类的属性，确定每个类所需要的服务⁴，确定类之间的关联和协作关系，并完成各种UML图作为补充。

精化的最终结果是形成一个分析模型，该模型定义了问题的信息域、功能域和行为域。

INFO

分析建模

假设要求你说明建造一个美食家厨房的所有需求，前提是你知道房间的尺寸、门窗的位置以及可用空间。

为了完整地说明应该建造什么，可能要列出所有的橱柜和厨具（生产厂家、样式、数量、尺寸），然后确定厨房的工作台面（金属板、花岗岩等）、管道附件、地板等。这些列表提供了有用的规格说明，但是都没有提供你想要的完整模型。为了完成模型，应该绘制一张三维透视图展示橱柜和厨具的位置以及相互之间的关系。使用这个模型评估工作流的效果（所有厨房都有的需求）和房间的美观程度（虽是个人观点，但却是非常重要的需求）要相对容易得多。

构造分析模型基本上和为厨房画一张蓝图或三维透视图相同。在形成系统构思时，这对评估系统构件相互之间的关系、确定如何在图片中满足需求、评估系统是否“美观”都非常重要。

7.2.4 协商



在有效的协商中没有赢家也没有输家，而是双赢。这是因为双方合作才是“协议”的坚实基础。

只给定了有限的业务资源，而客户和用户却提出了过高的目标要求，这是常有的事。另一个相当常见的现象是，不同的客户或用户提出了相互冲突的需求，并坚持“我们的特殊要求是至关重要的”。

178

需求工程师必须通过协商过程来调解这些冲突。应该让客户、用户和其他共利益者对各自的需求排序，然后按优先级讨论冲突。识别和分析与每项需求相关联的风险（详细内容参考第25章）；粗略“估算”开发工作量，并用于评估每项需求对项目成本和交付时间的影响；使用迭代的方法，删除、组合或修改需求，以便相关各方均能达到一定的满意度。

7.2.5 规格说明

在基于计算机的系统（和软件）的环境下，术语规格说明（specification）对不同的人有不同的含义。一个规格说明可以是一份写好的文档、一套图形化的模型、一个形式化的数学

⁴ 也可使用术语“操作”和“方法”。

模型、一组使用场景、一个原型或上述各项的任意组合。



规格说明的形式和格式随着待开发软件的规模和复杂度的不同而变化。

有人建议应该开发一个“标准模板”[SOM97]并将之用于规格说明，他们认为这样将促使以一致的从而也更易于理解的方式来表示需求。然而，在开发规格说明时保持灵活性有时是必要的。对大型系统而言，文档最好是采用自然语言描述和图形化模型来编写。而对于技术环境明确的较小产品或系统，使用场景可能就足够了。

规格说明是需求工程师完成的最终工作产品，它将作为软件工程师后续活动的基础，它描述了一个基于计算机系统的功能和性能，以及那些将影响系统开发的约束。

7.2.6 确认



需求确认时的重要的一点是一致性。使用分析模型可以保证需求说明的一致性。

在确认这一步将对需求工程的工作产品进行质量评估。需求确认要检查规格说明以保证：所有的系统需求已被无歧义地说明；不一致性、疏漏和错误已被检测出并被纠正；工作产品符合为过程、项目和产品建立的标准。

正式技术评审（第26章）是最主要的需求确认机制。确认需求的评审小组包括软件工程师、客户、用户和其他共利益者，他们检查系统规格说明，查找内容或解释上的错误，以及可能需要进一步解释澄清的地方、丢失的信息、不一致性（这是建造大型产品或系统时遇到的主要问题）、冲突的需求或不现实的（不能达到的）需求。

INFO

需求确认检查单

通常，按照检查单上的一系列问题检查每项需求非常有用。这里列出其中部分可能会问到的问题：

- 需求说明清晰吗？有没有可能造成误解？
- 需求的来源（如人员、规则、文档）弄清了吗？需求的最终说明是否已经根据或对照最初来源检查过？
- 需求是否用定量的术语界定？
- 其他哪些需求与此需求相关？是否已经使用交叉索引或其他机制清楚地加以说明了？
- 需求是否违背某个系统领域的约束？
- 需求是否可测试？如果可以，能否说明测试（有时称为确认准则）检验了需求？
- 对已经创建的任何系统模型，需求是否可跟踪？
- 对整体系统/产品目标，需求是否可跟踪？
- 规格说明的构造方式是否有助于理解、引用和翻译成更技术性的工作产品？
- 对已创建的规格说明是否建立了索引？
- 和系统性能、行为及运行特征相关的需求的说明是否清楚？哪些需求是隐含出现的？

7.2.7 需求管理

在第6章，我们提到基于计算机的系统其需求会变更，而且变更的要求贯穿于系统的整个

WebRef

可以从www.jiludwig.com网站获取各种和需求管理相关的有用信息。



当系统庞大而复杂时，确定需求之间的关系可能是个棘手的工作。使用跟踪表可以使工作更为方便容易。

生存期。需求管理是用于帮助项目组在项目进展中标识、控制和跟踪需求以及变更需求的一组活动⁵。这类活动中的大部分和第27章中讨论的软件配置管理（SCM）技术是相同的。

需求管理从标识开始。每个需求被赋予唯一的标识符。一旦需求被标识，便开始建立跟踪表，如图7-1所示，每个跟踪表将标识的需求与系统或其环境的一个或多个方面相关联。可以使用的跟踪表有很多，下面是其中的代表：

特征跟踪表。显示需求与重要的、客户可见的系统/产品特征的关系。

来源跟踪表。标识每个需求的来源。

依赖跟踪表。指明需求之间是如何相互关联的。

子系统跟踪表。按照需求所控制的子系统对需求分类。

接口跟踪表。显示需求与内部和外部系统接口的关系。

在很多情况下，这些跟踪表作为需求数据库的一部分来维护，以便可以快速查找它们，从而了解一项需求的变更将如何影响待建系统的不同方面。

180

		系统或其环境的特定方面					All
需求		A01	A02	A03	A04	A05	
R01			✓		✓		
R02	✓		✓				
R03	✓			✓			✓
R04		✓			✓		
R05	✓	✓		✓			✓
Rnn		✓	✓				

图7-1 通用跟踪表

SOFTWARE TOOLS**需求工程**

目的：需求工程工具有助于需求收集、需求建模、需求管理和需求确认。

机制：工具的工作机制多种多样。通常，需求工程工具创建大量的图形化（例如UML）模型来说明系统的信息、功能和行为。这些模型构成了软件过程中其他所有活动的基础。

代表性工具⁶

Atlantic Systems Guide, Inc提供了一个相当全面（也是最新）的需求工程工具的列表，可以在<http://www.systemsguild.com/GuildSite/Robs/retools.html>找到。第8章讨论需求建模工具。下面提到的工具主要侧重需求管理。

EasyRM：由Cybernetic Intelligence GmbH (www.easy-rm.com) 开发，构建项目专用

⁵ 正式的需求管理只应用于大型项目，这种大型项目通常有数百个可标识的需求。对于小项目，需求工程工作可以适当裁减，不太正式也是可以接受的。

⁶ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

的字典/术语，包含详细的需求说明和属性。

OnYourMark Pro: 由Omni-Vista (www.omni-vista.com) 开发，构建需求数据库，建立需求之间的关联并允许用户分析需求和进度/成本之间的关系。

Rational RequisitePro: 由Rational Software (www.rational.com) 开发，允许用户构建需求数据库，描述需求之间的关联、组织、优先级以及跟踪需求。

RTM: 由Integrated Chipware (www.chipware.com) 开发，是一个需求说明和跟踪工具，也支持一定程度的变更控制和测试管理。

应该指出的是，许多需求管理任务可以使用简单的电子表或小型数据库系统完成。

7.3 启动需求工程过程

在理想情况下，客户和软件工程师在同一个小组中工作⁷。在这种情况下，需求工程就只是和组里熟悉的同事进行有意义的交谈。但实际情况往往不是这样。
[181]

客户可能正在不同的城市或国家，对于想要什么可能仅有模糊的想法，对于将要构建的系统可能存在不同的意见，技术知识可能很有限，可能仅有有限的时间与需求工程师沟通。这些事情都是不希望遇到的，但却又是十分常见的，软件开发团队经常被迫在这种环境的限制下工作。

在下面的小节中，将要讨论启动需求工程所必需的步骤——使项目始终走向成功解决方案的方向。

7.3.1 确认共利益者



共利益者是那些对将要开发的系统有直接兴趣或直接从中获益的人。

Sommerville和Sawyer[SOM97]把共利益者定义为“直接或间接从正在开发的系统中获益的人”。可以确定如下几个容易理解的共利益者：业务操作管理人员、产品管理人员、市场营销人员、内部和外部客户、最终用户、顾问、产品工程师、软件工程师、支持和维护工程师以及其他人员。每个共利益者对系统都有不同的考虑，当系统成功开发后所能获得的利益也不相同，同样地，当系统开发失败时所面临的风险也不同。

在开始阶段，需求工程师应该创建一个人员列表，列出那些有助于诱导出需求的人员(7.4节)。最初的人员列表将随着接触的共利益者人数增多而增加，因为每个共利益者都将被询问“你认为我还应该和谁交谈”。

7.3.2 识别多种观点

因为存在很多不同的共利益者，所以系统需求调研也将从很多不同的视角开展。例如，市场营销部门关心能激发潜在市场的、有助于新系统更容易销售的功能和特点；业务经理关注应该在预算内实现特点，并且这些特点应该满足已规定的市场限制；最终用户可能希望系统的功能是他们所熟悉的并且易于学习和使用；软件工程师可能关注软件基础设施如何能够支持更多适于销售的功能和特点；支持工程师可能关注软件的可维护性。

“把三个共利益者请进一个房间，然后问他们想要什么样的系统，你很可能会得到四个或更多的不同观点。”

——作者不详

⁷ 推荐所有项目都使用该方法，而且该方法是敏捷软件开发方法的主要部分。

这些参与者（以及其他人）中的每一个人都将为需求工程贡献信息。当从多个角度收集信息时，所形成的需求可能存在不一致性或相互矛盾。需求工程师的工作就是把所有共利益者提供的信息（包括不一致或矛盾的需求）分类，分类的方法应该便于决策制定者为系统选择一个内部一致的需求集合。

182

7.3.3 协同合作

在前面的章节中，我们通篇提醒客户（和其他共利益者）之间应团结协作（避免内讧），并和软件工程人员团结协作，才能成功实现预定的系统。但是如何实现协作？

需求工程师的工作是标识公共区域（即所有共利益者都同意的需求）和矛盾区域或不一致区域（即某个共利益者提出的需求和其他共利益者的需求相矛盾）。很明显，后一种分类更有挑战性。

INFO

使用“优先点”

有一个方法能够解决相互冲突的需求，同时更好地理解所有需求的相对重要性，那就是使用基于优先点的“投票”方案。所有的共利益者都会分配到一定数量的优先点，这些优先点可以适用于很多需求。每个共利益者在一个需求列表上，通过向每个需求分配一个或多个优先点来表明该需求的相对重要性（从他或她的个人观点）。优先点用过之后就不能再次使用，一旦某个共利益者的优先点用完，他就不能再对需求实施进一步的操作。所有共利益者在每项需求上的优先点总数显示了该需求的综合重要性。

协作并不意味着必须由委员会定义需求。在很多情况下，共利益者的协作是提供他们各自关于需求的观点，而一个有力的“项目领导者”（例如业务经理或高级技术员）可能要对删除哪些需求做出最终判断。

7.3.4 首次提问

本章前面提到，在项目初始时的提问应该是“与环境无关的”[GAU89]。第一组与环境无关的问题集中于客户和其他共利益者、整体目标、收益。例如，需求工程师可能会问：

- 谁是这项工作的最初提出者？
- 谁将使用该解决方案？
- 成功的解决方案将带来什么样的经济收益？
- 存在别的解决方案吗？

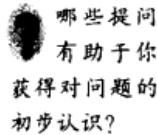
这些问题有助于识别所有对构建软件感兴趣的共利益者。此外，问题还确认了某个成功实现的可度量收益以及定制软件开发的可选方案。

183

“知道问题比知道所有的答案更好。”

——James Thurber

下一组问题有助于软件开发组更好地理解问题，并允许客户表达其对解决方案的看法：



- 如何描述由某成功的解决方案产生的“良好的”输出？
- 该解决方案强调了什么问题？
- 能向我们展示（或描述）解决方案的使用环境吗？

- 存在影响解决方案的特殊性能问题或约束吗？

最后一组问题关注于沟通活动本身的效果。Gause和Weinberg[GAU89]称之为“元问题”。下面给出一些元问题的简单列表：

- 你是回答这些问题的合适人选吗？你的回答是“正式的”吗？
- 我的提问和你想解决的问题相关吗？
- 我的问题是否太多了？
- 还有其他人员可以提供更多的信息吗？
- 还有我应该问的其他问题吗？

“问问题的人是五分钟的傻瓜，而不问问题的人将永远是傻瓜。”

——中国谚语

这些问题（和其他问题）将有助于“打破坚冰”，并有助于交流的开始，而且这样的交流对需求导出的成功至关重要。但是，会议形式的问与答（Q&A）并非一定会取得成功的好方法。事实上，Q&A会议应该仅仅用于首次接触，然后就应该用需求诱导形式（包括问题求解、协商和规格说明）取代。在7.4节中将介绍这类方法。

7.4 导出需求

184

在7.3.4中所说的Q&A形式在获取需求时是有效的，但是该方法在提取更详细的需求时并非一定会取得成功。事实上，Q&A会议的局限性已在上面指出，在下面的小节中将介绍这类方法。

7.4.1 协同需求收集

提出面向团队的需求收集方法是为了鼓励合作，即一个包括共利益者和开发人员的团队共同完成如下任务：确认问题，为解决方案的要素提供建议，协商不同的方法，以及说明初步的解决方案需求集合[ZAH90]。⁸

关于需求收集，现在已经提出了很多不同的协同需求收集方法，适用于稍有不同的场景，而且都应用了以下的基本原则为基础的某些原则：

 举办一个协同需求收集会议的基本原则是什么？

- 会议由软件工程师和客户（连同其他的共利益者）共同举办和参与。
- 制定筹备和参与会议的规则。
- 建议拟定一个会议议程，这个议程既要足够正式，使其涵盖所有的重要点；但也不能太正式，以鼓励思维的自由交流。
- 由一个“主持人（facilitator）”（可以是客户、开发人员或其他外人）控制会议。
- 使用某种“定义机制”（可以是工作表、活动挂图、不干胶贴纸或电子公告牌、聊天室或虚拟论坛）。
- 目的是识别问题，提出解决方案的要素，协商不同的方法以及在有利于完成目标的氛围中刻画初步解决方案的需求问题。

为了更好地理解在会议召开时的事件流，我们提出了一个概略的场景，该场景概述了准

⁸ 该方法有时也称作便利的应用规格说明技术（facilitated application specification techniques, FAST）。

备需求收集会议的事件序列、在会议过程中发生的事件序列以及在会议之后的事件序列。

“我们大量的时间——项目的主要工作量，不是花在实现或测试上，而是花在决定构造什么上。”
——Brian Lawrence

在需求的起始阶段（7.3节），基本问题和问题的答案确定了问题的范围和对解决方案的整体理解。除了这些最初的会议之外，共利益者要写一个1~2页的“产品要求”。此外还要选择会议地点、时间和日期，并选举“主持人”。来自开发组和其他共利益者组织的代表被邀请出席会议，在会议召开之前应将产品要求分发给所有的与会者。

185

WebRef

联合应用程序开发（JAD）是需求收集普遍采用的技术。可以在www.carolla.com/wp-jad.htm中找到详细的说明。

在召开会议评审产品要求的前几天，要求每个与会者列出构成系统周围环境的对象、由系统产生的其他对象以及系统用来完成功能的对象。此外，要求每个与会者列出服务操作或与对象交互的服务（过程或功能）列表。最后，还要开发约束列表（如成本、规模大小、业务规则）和性能标准（如速度、精确度）。告诉与会者，这些列表不要求完备无缺，但要反映每个人对系统的理解。

举一个例子⁹，考虑SafeHome项目中的一个市场营销人员写成的会前文件（摘录）。此人对SafeHome项目的住宅安全功能叙述如下：

我们的研究表明，住宅管理系统市场以每年40%的速度增长。我们推向市场的首个SafeHome功能将是住宅安全功能，因为多数人都熟悉“报警系统”，所以这将更容易销售。

住宅安全功能应该为各种不希望出现的“情况”提供保护，如非法入侵、火灾、漏水、一氧化碳浓度超标等等。该功能将使用无线传感器监控每种情况，户主可以编程控制，并且在发现情况时自动电话联系监控部门。

事实上，其他人在需求收集会议中将补充大量的信息。但是，即使有了补充信息，仍有可能存在歧义性和疏漏，错误也有可能发生。但对此时而论，上面的“功能描述”是足够了。

需求收集团队由营销、软件和硬件工程师以及制造方的代表组成，并使用外来人员作为会议“主持人”。

ADVICE
如果一个系统或产品将要为很多用户提供服务，必须绝对保证需求是从所有用户的代表群中提取的。如果只有一个用户定义所有的需求，那么接受风险将会非常高。

每个人都要给出上面所说的列表。SafeHome描述的对象可能包括：一个控制面板、若干烟感器、若干门窗传感器、若干动态检测器、一个警报器、一个事件（一个已被激活的传感器）、一个显示器、一台计算机、若干电话号码、一个电话等。服务列表可能包括：配置系统、设置警报器、监测传感器、电话拨号、控制面板编程以及读显示器（注意，服务作用于对象）。采用类似的方法，每个与会者都将开发约束列表（例如，当传感器不工作时系统必须能够识别，必须是用户友好的，必须能够和标准电话线直接连接）和性能标准列表（例如，一个传感器事件应在一秒内被识别，应实施事件优先级方案）。

186

⁹ SafeHome例子（有一定的扩展和变动）在下面很多章节中用于说明软件工程的重要方法，练习为该例子举行你自己的需求收集会议并为其开发一组列表。

“事实不会因为被忽略而不存在。”

——Aldous Huxley

当需求收集会议开始时，讨论的第一个话题是新产品的要求和开发理由——每个人均应该认同开发该产品是合理的。一旦达成一致，每个参与者提出自己的讨论列表。这些列表可以用大的纸张钉在墙上或用便签纸贴在墙上或写在墙板上。或者，列表也可以被贴在电子公告牌上或聊天室中，便于会议前的评审。理想情况下，应该能够分别操作每个列表项，以便于组合列表、删除项以及加入新项。在本阶段，严禁批评和争论。



一定要克制攻击客户意见“太昂贵”或“不实际”这样的冲动。这里建议通过协商形成一个大家均接受的列表。为了达到这个目标，必须保持开放的思想。

当某话题域的各个列表被提出后，小组将生成一个组合列表。该组合列表将删除冗余项，并加入在讨论过程中出现的一些新的想法，但是不删除任何东西。在所有话题域的组合列表都生成后，主持人开始讨论协调。组合列表可能会缩短、加长或重新措词，以求更恰当地反映即将开发的产品/系统，目标是为每个话题域（对象、服务、约束和性能）提交一个意见一致的列表，在后面的工作中将用到这些列表。

一旦完成意见一致的列表，团队被分为更小的子团队，每个子团队试图为每个列表中的一个或多个项目编写小规格说明（mini-specification）¹⁰。每个小规格说明是对包含在列表中的单词或短语的精炼，例如，对SafeHome对象控制面板的小规格说明可以如下：

控制面板是一个安装在墙上的单元，尺寸大概是9×5英寸；控制面板和传感器、计算机之间是无线连接；通过一个12键的键盘与用户交互，通过一个2×2的LCD显示器为用户提供反馈信息；软件将提供交互提示、回显以及类似的功能。

然后，每个子团队将他们完成的每个小规格说明提交给所有的与会者讨论，进行添加、删除和进一步细化等工作。在某些情况下，编写小规格说明可能会发现新的对象、服务、约束或性能需求，可以将这些新发现加入到原始列表中。在所有的讨论过程中，团队可能会提出某些在会议中不能解决的问题，将这些问题列表保留起来以便这些意见在以后的工作中发挥作用。

187

在小规格说明完成后，每个与会者应提出一个针对产品/系统的确认标准列表并将该列表提交给团队，然后完成一个意见一致的确认标准列表。最后，应对一个或多个参与方（或外来者）赋予如下任务：以会议的结果为输入，撰写完整的规格说明草案。

SAFEHOME

召开需求收集会议

[场景]一间会议室，进行首次需求收集会议。

[人物] Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人（指向白板）：这是目前住宅安全功能的对象和服务列表。

营销人员：从我们的观点看差不多覆盖了需求。

¹⁰ 也有很多软件团队不是创建小规格说明，而是选择创建用户场景，即所谓的用例（use-case）。这些在7.5节中有详细说明。

Vinod: 没有人提到他们希望通过Internet访问所有的SafeHome功能吗？这应该包括住宅安全功能，不是吗？

营销人员: 是的，这很正确……我们必须加上这个功能以及合适的对象。

主持人: 这还需要加上一些限制吗？

Jamie: 肯定，包括技术上的和法律上的。

产品代表: 什么意思？

Jamie: 我们必须确保外人不能非法侵入系统、废除系统、抢劫甚至更糟。我们的责任非常重。

Doug: 非常正确。

营销人员: 但我们确实需要Internet连接……只是保证能够制止外人接入……

Ed: 说比做容易，而且……

主持人（打断）：我现在不想讨论这个问题。我们把它作为动作项记录下来，然后继续……（Doug作为会议的记录者记下合适的内容）

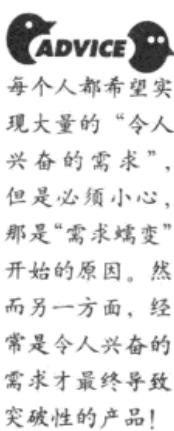
主持人: 我有种感觉，这儿仍存在很多需要考虑的问题。

（小组接下来花费45分钟提炼并扩展住宅安全功能的细节。）

7.4.2 质量功能部署



质量功能部署（Quality Function Deployment, QFD）是一种将客户要求转化成软件技术需求的技术。QFD“目的是最大限度地让客户从软件工程过程中感到满意[ZUL92]”。为了达到这个目标，QFD强调理解“什么是对客户有价值的”，然后在整个工程活动中部署这些价值。QFD确认了三类需求[ZUL92]：



正常需求。这些需求反映了在和客户开会时确定的针对某产品或系统的目标。如果实现了这些需求，将满足客户。这方面的例子如：所要求的图形显示类型、特定的系统功能以及已定义的性能级别。

期望需求。这些需求隐含在产品或系统中，并且可能是非常基础的以至于客户没有显式地说明，但是缺少这些将导致客户明显不满。这方面的例子如：人机交互的容易性、整体的操作正确性和可靠性以及软件安装的简易性。

令人兴奋的需求。这些需求反映了客户期望之外的特点，但是如果实现这些特点的话将会使客户非常满意。例如，文字处理软件要求具有标准的特征，但如果交付的产品包含一些页面布局能力，则将是令人欣喜并出人意料的。

实际上，QFD贯穿于整个软件工程[PAT96]。然而，很多QFD概念对需求提取活动也适用。在下面的段落中，我们仅仅给出这些概念（适用于计算机软件）的概述。

“去掉那些可能失败的，也很可能去掉了那些最有希望的。” ——William Shakespeare

WebRef

关于QDF的有用信息可以查看
www.qfdi.org

在与客户的会议中，功能部署被用于确定系统所需的每个功能的价值；信息部署确认系统必须使用和产生的数据对象和事件。这些都和功能紧密相关联。最后，任务部署在合适的环境下检查系统或产品的行为；价值分析帮

助确定上述三个部署中需求的相对优先级别。

QFD通过客户访谈和观察、调查以及检查历史数据（如问题报告）为需求收集活动获取原始数据。然后把这些数据翻译成需求表——称为客户意见表，并由客户评审。接下来使用各种图表、矩阵和评估方法抽取期望的需求并努力导出令人兴奋的需求[BOS91]。

7.4.3 用户场景

当收集需求时，系统功能和特点的整体愿景开始具体化。但是在软件团队弄清楚不同类型最终用户如何使用这些功能和特征之前，很难转移到更技术化的软件工程活动。为实现这一点，开发人员和用户可以创建一系列的场景——场景可以识别对将要构建的系统的使用线索。场景通常称为用例[JAC92]，它提供了系统将如何被使用的描述。在7.5节中将更详细地讨论用例。
189

SAFEHOME

开发一个初步的用户场景

[场景] 一间会议室，继续首次需求收集会议。

[人物] Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人： 我们已经讨论过通过Internet访问SafeHome功能的安全性，我想考虑得再多些。下面我们开发一个使用住宅安全功能的用户场景。

Jamie： 怎么做？

主持人： 我们可以采用两种不同的方法完成这个工作，但是现在，我想不要太正式吧。请问（他指向一个市场人员），你设想该如何使用这样的系统。

营销人员： 嗯……好的，如果我出门在外，我想我能做的就是必须让某个没有安全码的人比如管家或修理工进入我家。

主持人（微笑）： 这就是你的原因……告诉我们实际上你怎么做？

营销人员： 嗯……首先是我需要一台电脑，登录为所有SafeHome用户提供维护的Web网站，提供我的用户身份证号……

Vinod（打断）： Web页面必须是安全的、加密的，以确保我们安全……

主持人（打断）： 这是个有用信息，Vinod，但这太技术性了，我们还是只关注最终用户将如何使用该功能，好吗？

Vinod： 没问题。

营销人员： 那么，就像我所说的，我会登录一个Web网站并提供我的用户身份证号和两级密码。

Jamie： 如果我忘记密码怎么办？

主持人（打断）： 好想法，Jamie。但是我们先不谈这个。我们先把这种情况作为“异常”记录下来。我确信还有其他的异常。

营销人员： 在我输入密码后，屏幕将显示所有的SafeHome功能。我选择住宅安全功能，系统可能会要求我确认我是谁，要求我的地址或电话号码或其他什么，然后显示一张图片，

包括安全系统控制面板和我能执行的功能列表——安装系统、解除系统、解除一个或多个传感器。我猜还可能会允许我重新配置安全区域和其他类似的东西，但是我不确定。

(当市场营销人员继续讨论时，Doug记录下大量内容。这些构成了最初非正式的用例场景基础。另一种方法是让市场营销人员写下场景，但这应该在会议之外进行。)

7.4.4 导出工作产品

根据将要构建的系统或产品的规模不同，需求导出后产生的工作产品也不同。对于大多数系统而言，工作产品包括：

需求收集后
生成什么样
的信息结果？

- 必要性和可行性陈述。
- 系统或产品范围的界限说明。
- 参与需求导出的客户、用户和其他共利益者的列表。
- 系统技术环境的说明。
- 需求列表（推荐按照功能加以组织）以及每个需求适用的领域限制。
- 一系列使用场景，有助于深入了解系统或产品在不同运行环境下的使用。
- 任何能够更好地定义需求的原型。

190

所有参与需求导出的人员需要评审这些工作产品。

7.5 开发用例

在一本讨论如何编写有效用例的书中，Alistair Cockburn[COC01]写道：“一个用例捕获一个合同……即说明当对某个共利益者的请求响应时，在各种条件下系统的行为”。本质上，用例讲述了能表达主体场景的故事：最终用户（扮演多种可能角色中的一个）如何在一特定环境下和系统交互。这个故事可以是叙述性的文本、任务或交互的概要、基于模板的说明或图表显示。不管其形式如何，用例从最终用户的角度描述了软件或系统。



从参与者的角度
定义用例。参与
者是人员或设备
在和软件交互时
所扮演的角色。

撰写用例的第一步是定义各类故事中所包含的“参与者”。参与者是在将要说明的功能和行为环境内使用系统或产品的各类人员（或设备）。参与者代表了系统运行时人（或设备）所扮演的角色，更为正式的定义就是：参与者是任何与系统或产品通信的事物，且对系统本身而言参与者是外部的。当使用系统时，每个参与者都有一个或多个目标。

要注意的是，参与者和最终用户并非一回事。典型的用户可能在使用系统时扮演了许多不同的角色，而参与者表示了一类外部实体（经常是人员，但并不总是如此），在用例中他们仅扮演一种角色。例如，考虑一个机床操作员（一个用户），他和生产车间（其中布置了许多机器人和数控机床）内的某个控制计算机交互。在仔细考察需求后，控制计算机的软件需要四种不同的交互模式（角色）：编程模式、测试模式、监测模式和纠错模式。因此，4个参与者可定义为：程序员、测试员、监控员和故障检修员。有些情况下，机床操作员可以扮演所有这些角色，而有些情况下，每个参与者的角色可能由不同的人员扮演。

191

因为需求导出是一个逐步演化的活动，因此在第一次迭代中并不能确认所有的参与者。

WebRef
一篇非常好的关
于用例的文章可
以从 [www.rat-
ional.com/produc-
ts/whitepapers/
100622.jsp](http://www.rat-
ional.com/produc-
ts/whitepapers/
100622.jsp) 下载。

在第一次迭代中有可能识别主要的参与者[JAC92]，而对系统了解更多之后，才能识别出次要的参与者。主要参与者直接且经常使用软件，和他们交互可以获取所需的系统功能并导出系统的预期收益。次要参与者支持系统，以便主要参与者能够完成他们的工作。

一旦确认了参与者，就可以开发用例。对于应该由用例回答的问题，Jacobson[JAC92]提出了以下建议¹¹：

 开发一个有效用例必须知道哪些？

- 谁是主要参与者、次要参与者？
- 参与者的目标是什么？
- 故事开始前有什么前提条件？
- 参与者完成的主要工作或功能是什么？
- 按照故事所描述的还可能需要考虑什么异常？
- 参与者的交互中有什么可能的变化？
- 参与者将获得、产生或改变哪些系统信息？
- 参与者必须通知系统外部环境的改变吗？
- 参与者希望从系统获取什么信息？
- 参与者希望得知意料之外的变更吗？

回顾基本的SafeHome需求，我们定义了三个参与者：房主（用户）、配置管理人员（类似房主，但扮演不同的角色）、传感器（附属于系统的设备）和监测子系统（监测SafeHome房间安全功能的中央站）。仅从该例子的目的来看，我们只考虑了房主这个参与者。房主通过使用报警控制面板或计算机等多种方式和住宅安全功能交互：

- 输入密码以便能进行其他交互。
- 查询安全区的状态。
- 查询传感器的状态。
- 在紧急情况时按下应急按钮。
- 激活/关闭安全系统。

192 考虑房主使用控制面板的情况，系统激活的基本用例如下¹²：

1. 房主观察SafeHome控制面板（图7-2），以确定系统是否已准备好接收输入。如果系统未就绪，“not ready”消息将显示在LCD显示器上，房主必须亲自动手关闭窗户/门才能使“not ready”消息消失。（not ready消息暗示某个传感器是开着的，即某个门或窗户是开着的。）
2. 房主使用键盘键入4位密码，该密码和系统中存储的有效密码相比较。如果密码不正确，控制面板将鸣叫一声并自动复位以等待再次输入；如果密码正确，控制面板等待进一步的操作。
3. 房主选择键入“stay”或“away”（见图7-2）启动系统。“stay”只激活外部传感器（内部的运动监测传感器是关闭的）。“away”激活所有的传感器。
4. 当激活时，房主可以看到一个红色的警报灯。

基本的用例从较高层次上给出参与者和系统之间交互的故事。

在很多情况下，进一步细化用例可以为交互提供更详细的说明。例如，Cockburn[COC01]建议使用如下模板详细说明用例：

¹¹ Jacobson的问题已经被扩展到为用例场景提供更完整的视图。

¹² 注意，该用例和通过Internet访问系统的情形不同，该用例的环境是通过控制面板交互而不是使用计算机所提供的GUI。

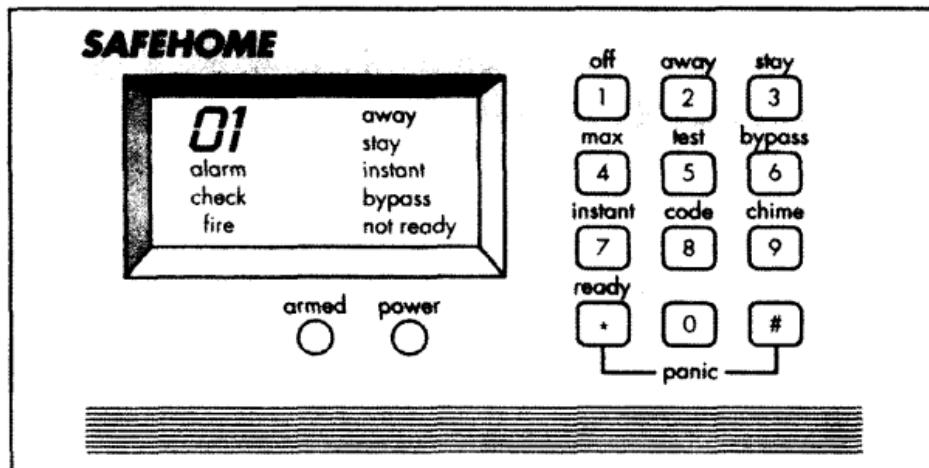


图7-2 SafeHome控制面板



用例通常不是形式化的，但是使用这里介绍的模板可以确保你已经说明了所有关键问题。

用例：初始化监测

主要参与者：房主。

目标：设置系统在房主离开住宅或留在房间内时监测传感器。

前提条件：系统已经输入密码并识别各种传感器。

触发器：房主决定“设置”系统，即打开警报功能。

场景：

1. 房主：观察控制面板。
2. 房主：输入密码。
3. 房主：选择“stay”或“away”。
4. 房主：观察红色报警灯显示SafeHome已经被打开。

异常：

1. 控制面板未就绪：房主检查所有的传感器，确定哪些传感器是开着的（即门窗是开着的）并将其关闭。

2. 密码不正确（控制面板鸣叫一声）：房主重新输入正确的密码。
3. 密码不识别：必须对监测和响应子系统重新设置密码。
4. 选择stay：控制面板鸣叫两声而且stay灯点亮；边界传感器被激活。
5. 选择away：控制面板鸣叫三声并且away灯点亮；所有传感器被激活。

优先级：必须的，必须被实现。

何时可用：首次增量。

使用频率：每天多次。

使用方式：通过控制面板接口。

次要参与者：技术支持人员，传感器。

次要参与者使用方式：

技术支持人员：电话线。

传感器：有线或无线接口。

KEY POINT

共利益者可以访问每个用例，而且可以指定每个用例的相对优先级。

194

未解决的问题：

1. 是否还应该有不使用密码或使用缩略密码激活系统的方式？
2. 控制面板是否还应显示附加的文字信息？
3. 房主输入密码时，从按下第一个按键开始必须在多长时间内输入密码？
4. 在系统真正激活之前有没有办法关闭系统？

可以使用类似的方法开发其他的房主交互用例。重要的是必须认真评审每个用例。如果某些交互元素模糊不清，用例评审将解决这些问题。

SAFEHOME

开发高级的用例图

[场景] 会议室，继续需求收集会议。

[人物] Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员；Doug Miller，软件项目经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

对话

主持人： 我们已经花费了相当多的时间讨论SafeHome住宅安全功能。在休息时间我画了一个用例图，用它来概括重要的场景，这些场景是该功能的一部分。大家看一下。

(所有的与会者注视图7-3。)

Jamie： 我恰好刚刚开始学习UML符号。住宅安全功能是由中间包含若干椭圆的大方框表示吗？而且这些椭圆代表我们已经用文字写下的用例，对吗？

主持人： 是这样的。而且棍型小人代表参与者——和系统交互的人或东西，如同用例中所描述的……哦，我使用矩形标签表示在这个用例中那些不是人而是传感器的参与者。

Doug： 这在UML中合法吗？

主持人： 合法性不是问题，重点是交流信息。我认为使用类似人的棍型小人代表设备可能会产生误导，因此我做了一些改变。我认为这不会产生什么问题。

Vinod： 好的。这样我们就为每个椭圆进行了用例说明，还需要生成更详细的基于模板的说明吗？我们已经阅读过那些说明了。

主持人： 有可能，但这可以等到考虑完其他的SafeHome功能之后。

营销人员： 等一下，我已经看过这幅图，突然间我意识到我们遗漏了什么。

主持人： 哦，是吗。告诉我们遗漏了什么。

(会议继续进行。)

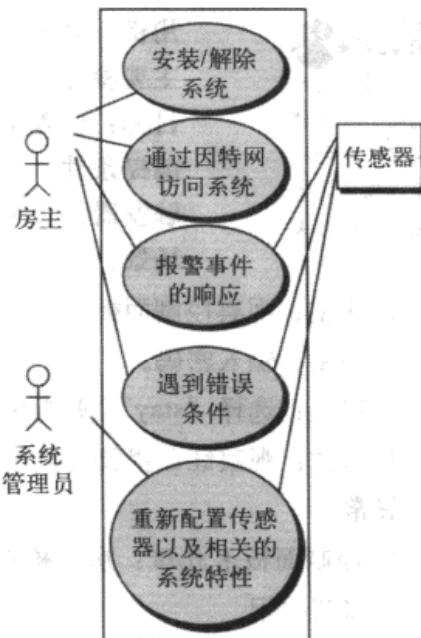


图7-3 SafeHome住宅安全功能的用例图

195

SOFTWARE TOOLS**用例开发**

目的：通过使用能提高访问透明性和一致性的自动化模板和机制，帮助开发用例。

机制：工具的原理各不相同。通常，用例工具为创建有效用例提供填空式的模板。大多数用例功能能嵌入到一系列更宽泛的需求工程功能中。

代表性工具¹³

Clear Requirement Workbench：由LiveSpecs软件公司 (www.livespecs.com) 开发，为创建和评估用例以及其他各种需求工程功能提供自动化支持。

大量的分析建模工具（多数基于UML）：为用例开发和建模提供文字和图形化支持。

Objects by Design：UML工具资源 (www.objectsbydesign.com/tools/umltools_byCompany.html)，提供对该类工具的全面链接。

各种用例模板和数据库可以在[UseCases.org](http://www.usecases.org) (www.usecases.org) 中找到。

7.6 构建分析模型

分析模型的目的是为基于计算机的系统提供必要的信息、功能和行为域的说明。模型应能够动态改造，以便于软件工程师更多地了解将要实现的系统，以便于共利益者更多地了解他们到底需要什么。因此，分析模型是任意给定时刻的需求的快照，我们期望它变化。

当分析模型演化时，某些元素将变得相对稳定，为后续设计任务提供稳固的基础。但是，有些模型元素可能是不稳定的，这表明客户仍然没有完全理解系统的需求。

分析模型及其构建方法将在第8章详细说明，下面小节仅提供简要的概述。

7.6.1 分析模型的元素

对基于计算机的系统，有很多不同的方法考虑需求。某些软件人员坚持最好选择一个表现模式（例如用例）并排斥所有其他的模式。有些专业人士则相信使用许多不同的表现模式来描述分析模型是值得的，不同的表现模式迫使软件团队从不同的角度考虑需求——一种方法更有可能造成需求遗漏、不一致性和歧义性。



把共利益者包括进来通常是个好主意。做到这一点最好的方法之一是让每个共利益者写下用例来描述将如何使用软件。

分析模型的特定元素由将要使用的分析建模方法（第8章）规定。但是，一些普遍的元素对大多数分析模型来说都是通用的。

196

基于场景的元素：使用基于场景的方法可以从用户的视角描述系统。例如，基本的用例（7.5节）及其相应的用例图（图7-3）演化成更精细的基于模板的用例。分析模型的基于场景的元素通常是正在开发的分析模型的第一部分。同样，它们也作为创建其他建模元素时的输入。

还有一个方法和基于场景建模稍有不同，它可以描述在某个处理环境中存在的并且已经被定义为需求导出任务一部分的活动或功能，也就是说，描述某个受限环境中处理过程的活动（也可以使用术语功能或操作）序列被定

¹³ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

义为分析模型的一部分。类似于大多数的分析模型（和其他软件工程模型）元素，活动（功能）可以在很多不同的抽象层面上描述。这类模型可以迭代定义，每次迭代都补充处理的细节。例如，图7-4为导出需求描绘了一个UML活动图¹⁴，显示了三层细节内容。

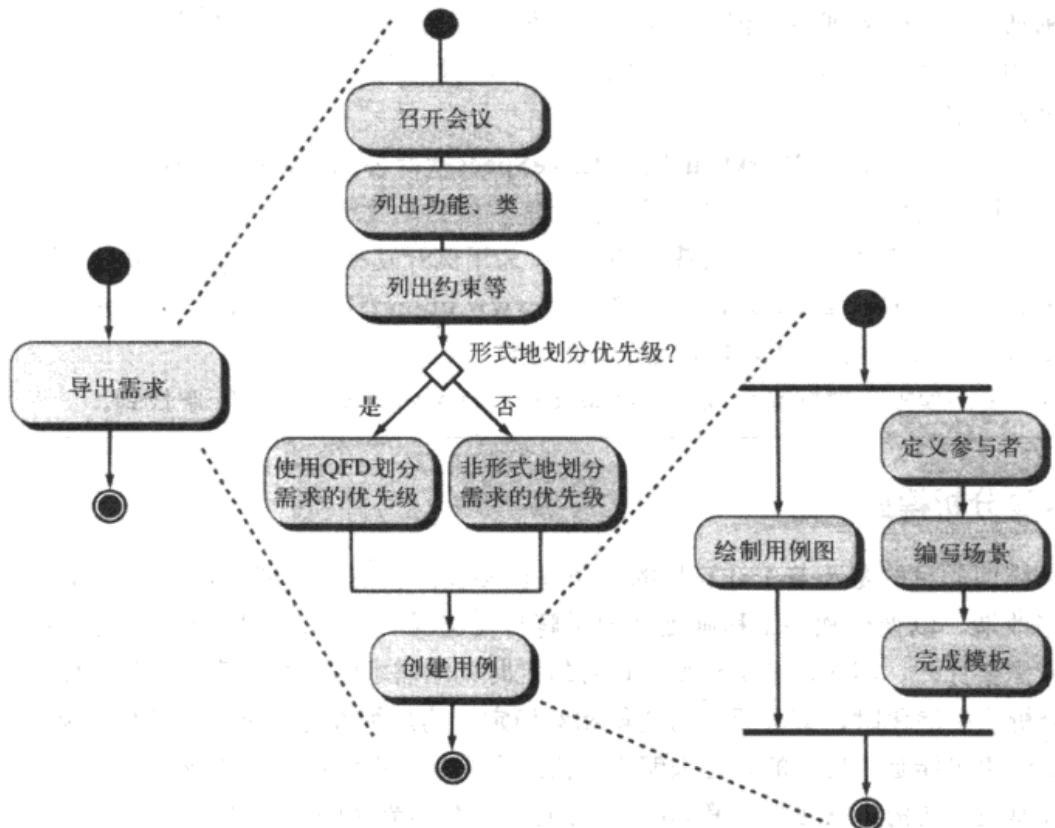


图7-4 导出需求的活动图



一种分离类的方法是查找用例脚本中的叙述性名词。至少某些名词将是候选类。第8章中将给出更详细的说明。



状态是外部可观察到的行为模式，外部激励导致状态间的转换。

基于类的元素：每个使用场景都暗示着当一个参与者和系统交互时所操作的一组“对象”，这些对象被分成类——具有相似属性和共同行为的事物集合。例如，描绘SafeHome安全功能传感器类的类图如图7-5所示。注意，类图列出了传感器的属性（如name/id, type）和可以用于修改这些属性的操作（例如identify()、enable()）。除了类图，其他分析建模元素描绘了类相互之间的协作以及类之间的关联和交互。在第8章中将有更详细的讨论。

行为元素：基于计算机的系统行为能够对所选择的设计和所采用的实现方法产生深远的影响。因此，分析模型必须提供描述行为的建模元素。

状态图（第8章）是一种表现系统行为的方法，该方法描绘系统状态以及导致系统改变状态的事件。状态是任何可以观察到的行为模式。另外，状态图还指明了在某个特殊事件后采取什么动作（例如激活处理）。

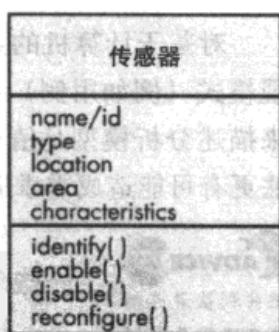


图7-5 传感器的类图

¹⁴ 活动图和流程图非常相似——用图形化图表描述控制流序列和逻辑（第11章）。

为了更好地说明状态图，考虑一个办公室复印机的读指令状态。UML状态图表示如图7-6所示，一个矩形框代表一个状态。矩形被分为三个区域：(1) 状态名（如读指令），(2) 状态变量，指出状态如何向外界显示自己，(3) 状态活动，指出如何进入状态(**entry/**)以及当处在该状态时可以调用的动作(**do:**)。

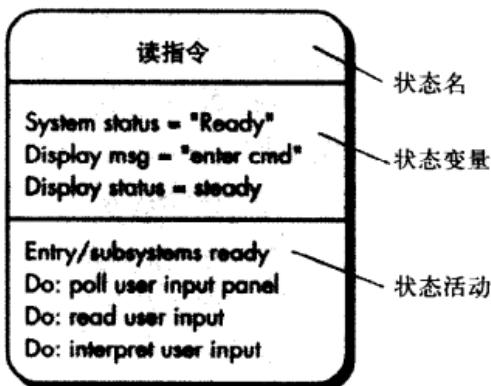


图7-6 UML状态图表示

SAFEHOME

初步的行为建模

[场景] 会议室，继续需求会议。

[人物] Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员；Doug Miller，软件项目经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人： 我们刚才差不多已经讨论完SafeHome住宅安全功能。但是在结束之前，我希望讨论一下功能的行为。

营销人员： 我不太理解你所说的行为意味着什么。

Ed（大笑）：那就是如果产品行为错误就让它“暂停”。

主持人： 不太准确，让我解释一下。

（主持人向需求收集团队解释行为建模的基本知识。）

营销人员： 这看起来有点技术性，我不敢确定能不能在这里帮上忙。

主持人： 你当然可以。你从用户的角度观察到什么行为？

营销人员： 嗯……好的，系统将监测传感器，将从房主那里读指令，将显示其状态。

主持人： 看到了吧，你是可以帮上忙的。

Jamie：还应该使用计算机确定是否有任何输入，例如基于Internet的访问或配置信息。

Vinod：是的，实际上，配置系统是其权利内的一个状态。

Doug：你这家伙开始灵光了，让我们多想一些……有方法把这个画出来吗？

主持人： 有方法，但让我们等到会后再开始吧。

面向信息流的元素： 信息在基于计算机的系统中流动时会被转换，系统接受多种形式的输入；使用函数转换输入；生成多种形式的输出。输入可以是由变频器发送的控制信号，可以是操作人员输入的数字，也可以是网络上传送的信息包或从备份存储器取回的庞大数据库文件。转换可能由单一的逻辑比较、复杂的数学算法或专家系统的规则推理方法构成，

输出可能是点亮一个发光二极管或是生成一份200页的报告。实际上，我们可以为任何基于计算机的系统（不管其规模大小和复杂度）创建信息流模型。第8章将更详细地讨论流的建模。

7.6.2 分析模式

任何有一些软件项目需求工程经验的人都开始注意到，在特定的应用领域内某些事情在所有的项目中重复发生¹⁵。这些可以被称为分析模式[FOW97]，它们在特定应用领域内表示某些在建模时可以被重复使用的东西（如类、功能、行为）。

Geyer-Schulz和Hahsler[GEY01]提出了使用分析模式的两个优点：

首先，分析模式提高了抽象分析模型的开发速度，通过提供可重复使用的分析模型捕获具体问题的主要需求，例如关于优点和约束的说明。其次，通过建议的设计模式和可靠的通用问题解决方案，分析模式有利于把分析模型到设计模型的转变。

WebRef
可以在hillside.net/patterns上找到一个非常有用的模式库和很多其他的模式资源。

通过参考模式名，分析模式被集成到分析模型中，并且也被保存在库中，以便需求工程师能够使用搜索工具查找和复用。

关于分析模式的信息可以使用标准的模板来表现，模板采用的格式如下[GEY01]¹⁶：

模式名：描述符（即模式名）应抓住模式的本质。当参考模式时描述符将用在分析模型中。

目的：在某个应用领域环境中，描述完成或表现的模式和（或）着手解决的问题。

动机：说明如何使用模式解决问题的场景。

外因和环境：说明能够影响如何使用模式的外部问题（外因）和当使用模式时必须解决的外部问题。外部问题可能包含和业务相关的主题、外部技术约束以及和人员相关的问题。

解决方案：说明模式如何被用于解决问题，重点在结构性和行为方面的问题。

结论：专用于说明当使用模式后将发生什么以及在其应用过程中存在哪些折衷。

设计：讨论如何通过使用已知的设计模式完成分析模式。

已知应用：在实际系统范围内的应用举例。

相关模式：和指定模式相关的一个或多个分析模式，因为该分析模式
(1) 通常和指定模式一起使用，(2) 和指定模式结构相似，(3) 是指定模式的变形。

在第8章将举例说明分析模式，并将针对这个主题展开更多的讨论。

¹⁵ 在某些情况下，事情重复发生而不管应用领域是什么。例如，不管所考虑的应用领域是什么，用户接口的特点和功能都是共有的。

¹⁶ 文献中提出了各种各样的模式模板，感兴趣的读者可以参阅[FOW97]、[BUS96]和[GAM95]。

INFO**模式**

事实上我们在日常生活中接触到的每件事物上都可以看到模式。

考虑一下动作冒险电影——更特殊的带有喜剧元素的动作冒险侦探电影。我们可以为主角和同伴、管主角的上司、心情愉快的罪犯以及其他更多角色定义模式。

例如，管主角的上司总是年长的、打领带（主角不打）、经常冲主角和同伴喊叫的人，他们通常提供滑稽的安慰，或者可能扮演更恶毒的角色，在主角和同伴前进的路上设置官僚的或自私的障碍。这样就建立了一个戏剧性的模式。

举个稍微更技术性的例子，比如在使用移动电话时如下模式是显而易见的：打电话、查找电话号、从多个消息中获取信息。这些模式中的每一个都可以说明一次，然后在任何移动电话软件中重复使用。

7.7 协商需求

在一个理想的需求工程情境中，起始、导出和精化工作能确保得到足够详细的客户需求，以开展后续的软件工程步骤。但不幸的是，这几乎不可能发生。实际中，客户和开发人员将进入协商的过程，这时要让客户以成本和产品投放市场的时间为背景，平衡功能、性能和其他的产品或系统特性。这个协调过程的目的是保证所开发的项目计划，在满足客户需求的同时反映软件团队所处真实世界的限制（如时间、人员、预算）。

“折衷是这样一种艺术：采用某种方式分蛋糕，让每个人感觉自己得到了最大的一块。”

——Ludwig Erhard

最好的协商是争取“双赢”的结果¹⁷，即客户的“赢”在于获得满足大多数需要的系统或产品，软件团队的“赢”在于按照实际情况，在可实现的预算和时间期限内完成工作。

201

WebRef

可以从sunset.usc.edu/~aegyed/publications/Software_Requirements_Negotiation-Some_lessons-Learned.html下载关于软件需求协商的简短文章。

Boehm[BOE98]定义了每个软件过程迭代启动时的一系列协商活动。不是定义单一的客户交流活动，而是定义了如下一系列活动：

1. 识别系统或子系统关键的共利益者。
2. 确认共利益者“赢”的条件。
3. 就共利益者“赢”的条件进行协商，以便使其与所有涉及人（包括软件团队）的一系列双赢条件一致。

这些初始化步骤的成功实施可以实现双赢的结果，这是继续开展后续软件工程活动的关键。

INFO**协商的艺术**

学习如何有效地协商可以帮助你更好地度过人生或技术生涯。如下指导原则非常值得考虑：

1. 认识到这不是竞争。为了成功，为了获得双赢，双方将不得不妥协。

¹⁷ 有许多关于谈判技巧的书籍（如[LEW00]、[FAR97]、[DON96]）。这是年轻的（或年长的）软件工程师或管理人员应该学习的重要事情之一，快读一本吧。

2. 制定策略。决定你希望得到什么；对方希望得到什么；你将如何行动以使得这两方面的希望都能实现。
3. 主动地听。不要当对方说了之后你才做出程式化的响应。听，是为了获取信息，这些信息有助于在磋商中更好地说明你的立场。
4. 关注对方的兴趣。如果想避开冲突，就不要太过于坚持自己的立场。
5. 不要进行人身攻击。应集中于需要解决的问题。
6. 要有创新性。当处于僵局时不要害怕而应考虑如何摆脱困境。
7. 随时准备作出承诺。一旦已经达成一致，不要闲聊胡扯，马上作出承诺然后继续进行。

SAFEHOME

开始协商

[场景] Lisa Perez的办公室，在第一次需求收集会议之后。

[人物] Doug Miller，软件工程经理；Lisa Perez，市场营销经理。

[对话]

Lisa：我听说第一次会议进行得很好。

Doug：确实是这样，你派了几个有经验的人参加会议……他们确实贡献了很多。

Lisa（微笑）：是的，他们的确告诉我他们融入了会议，而且会议卓有成效。

Doug（大笑）：下次再见面时我一定要脱帽致敬……看，Lisa，我想在你们主管所说的日期内获取所有的住宅安全功能可能会有问题。我知道现在还早，但是我们已经有一些落后于原定计划并且……

Lisa：我们必须在那个时间获得产品，Doug。你说的是什么功能？

Doug：我认为我们可以在截止日期前完成所有的住宅安全功能，但是必须把Internet访问功能推迟到第二个版本中。

Lisa： Doug，Internet访问是SafeHome最引人注目之处，我们正在围绕这一点开发我们整个的营销活动。我们必须拥有它！

Doug：我理解你的处境，我确实理解。问题在于为了向你提供Internet访问，我们将需要一整套Web站点安全防护措施，这将花费时间和人力。我们还必须在第一个版本中开发很多的附加功能……我认为我们不能在现有资源下完成这些。

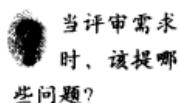
Lisa（皱眉）：我知道，但你必须找到实现方法，Internet访问对住宅安全功能非常关键，对其他功能也很关键……其他功能可以等到下一版……我同意这样。

很明显Lisa和Doug陷入了僵局，而且他们必须协商出一个解决办法。他们能够“双赢”吗？如果你扮演调解人的角色，有什么建议？

7.8 确认需求

当分析模型的每个元素都已创建时，需要检查一致性、是否有遗漏以及歧义性。模型所表现的需求由客户划分优先级并组合成一个整体，该需求整体将实现为一系列软件增量并交付给客户。分析模型的评审将提出如下问题：

- 每个需求都和系统/产品的整体目标一致吗？



当评审需求时，该提哪些问题？

- 所有的需求都已经在恰当的抽象层上说明了吗？换句话说，是否有一些需求是在技术细节过多的层次上提出的，并不适合当前的阶段。
- 需求是真正必需的，还是另外加上去的，有可能不是系统目标所必需的特性吗？
- 每个需求都有界定且无歧义吗？
- 每个需求都有归属吗？换句话说，是否每个需求都标记了来源（通常是一个单独的详细说明）？
- 有需求和其他需求相冲突吗？
- 在系统或产品所处的技术环境下每个需求都能够实现吗？
- 一旦实现后，每个需求是可测试的吗？
- 需求模型恰当地反映了将要构建系统的信息、功能和行为吗？
- 需求模型是否已经使用合适的方式“分割”，能够逐步地揭示详细的系统信息？
- 已经使用需求模式简化需求模型吗？所有的模式都已经被恰当地确认了吗？所有的模式都和客户的需求一致吗？

203

应当提出这些问题和其他一些问题并使问题得到回答，以确保需求模型精确地反映客户的需求并确保为设计奠定坚实的基础。

7.9 小结

在开始设计和构建一个基于计算机的系统之前，了解需求是必要的。为实现这一点，需要实施一系列的需求工作。需求工程发生在与客户沟通和为一般的软件过程定义的建模活动过程中，七个不同的需求工程功能——起始、导出、精化、协商、规格说明、确认和管理——由软件团队成员指导实施。

在项目起始阶段，开发人员和客户（以及其他共利益者）建立基本的问题需求，定义最重要的项目约束以及陈述主要的特征和功能，必须为系统表现出来这些特征和功能以满足其目标。该信息在导出阶段得到提炼和延伸，在此阶段中利用有主持人的会议、QFD和用户场景开发进行需求收集活动。

精化阶段进一步把需求扩展为分析模型——基于场景、基于活动、基于类、行为和面向信息流的模型元素的集合。为了创建这些元素可以使用各种建模符号。模型可能会参考分析模式（模式即可在不同的应用系统中看到的重复出现的问题域特征）。

当确定需求并且创建分析模型时，软件团队和其他项目共利益者协商优先级、可用性和每条需求的相对成本。协商的目标是开发一个理想的项目计划。此外，将按照客户需求确认每个需求和整个分析模型，以确保将要构建的系统是正确的。

参考文献

- [BOE98] Boehm, B., and A. Egyed, "Software Requirements Negotiation: Some Lessons Learned," *Proc. Intl. Conf. Software Engineering*, ACM/IEEE, 1998, pp. 503–506.
- [BOS91] Bossert, J. L., *Quality Function Deployment: A Practitioner's Approach*, ASQC Press, 1991.
- [BUS96] Buschmann, F., et al., *Pattern-Oriented Software Architecture: A System of Pattern*, Wiley, 1996.

- [COC01] Cockburn, A., *Writing Effective Use-Cases*, Addison-Wesley, 2001.
- [CRI92] Christel, M. G., and K. C. Kang, "Issues in Requirements Elicitation," Software Engineering Institute, CMU/SEI-92-TR-12 7, September 1992.
- [DON96] Donaldson, M. C., and M. Donaldson, *Negotiating for Dummies*, IDG Books Worldwide, 1996.
- [FAR97] Farber, D. C., *Common Sense Negotiation: The Art of Winning Gracefully*, Bay Press, 1997.
- [FOW97] Fowler, M., *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
- [GAM95] Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [GAU89] Gause, D. C., and G. M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.
- [GEY01] Geyer-Schulz, A., and M. Hahsler, *Software Engineering with Analysis Patterns*, Technical Report 01/2001, Institut für Informationsverarbeitung und-wirtschaft, Wirtschaftsuniversität Wien, November 2001, downloaded from: http://wwwai.wu-wien.ac.at/~hahsler/research/virlib_working2001/virlib/.
- [JAC92] Jacobson, I., *Object-Oriented Software Engineering*, Addison-Wesley, 1992.
- [LEW00] Lewicki, R. D. Saunders, and J. Minton, *Essentials of Negotiation*, McGraw-Hill, 2000.
- [PAR96] Pardee, W., *To Satisfy and Delight Your Customer*, Dorset House, 1996.
- [SOM97] Somerville, I., and P. Sawyer, *Requirements Engineering*, Wiley, 1997.
- [THA97] Thayer, R. H., and M. Dorfman, *Software Requirements Engineering*, 2nd ed., IEEE Computer Society Press, 1997.
- [YOU01] Young, R., *Effective Requirements Practices*, Addison-Wesley, 2001.
- [ZAH90] Zahniser, R. A., "Building Software in Groups," *American Programmer*, vol. 3, nos. 7-8, July-August 1990.
- [ZUL92] Zultner, R., "Quality Function Deployment for Software: Satisfying Customers," *American Programmer*, February 1992, pp. 28-41.

习题与思考题

- 7.1 为如下活动之一开发一个完整的用例：
- 在ATM提款。
 - 在餐厅使用信用卡付费。
 - 使用一个在线经纪人账户购买股票。
 - 使用在线书店搜索书（某个指定主题）。
 - 你的指导老师指定的一个活动。
- 7.2 讨论一下当需求必须从三、四个不同的客户中提取时会发生什么问题。
- 7.3 在需求起始阶段情境下讨论时，“可行性分析”意味着什么？
- 7.4 你负责从一个客户处导出需求，而他告诉你太忙了没时间见面，这时你该怎么做？
- 7.5 为什么大量的软件开发人员没有足够重视需求工程？以前有没有什么情况让你可以跳过需求工程？
- 7.6 开发一个促进需求收集的“工具包”。工具包应包含一系列的需求收集会议指导原则，用于促进列表创建的材料以及其他任何可能有助于定义需求的条款。
- 7.7 另外想出三个以上在需求起始阶段可能要问共利益者的“上下文无关的问题”。
- 7.8 让我们设想你已经说服客户（你是一个绝好的销售人员）同意你作为一个开发人员所提出来的每一个要求，这让你成为一个高明的协商人员吗？为什么？
- 7.9 你的指导老师将把班级分成四或六人的小组，组中一半的同学扮演市场部的角色，另一半将扮演软件工程部的角色。你的工作是定义本章所介绍的SafeHome安全功能的需求，并使用本章所提出的指导原则引导需求收集会议。

- 7.10 贯穿本章，我们提及“客户”，请为信息系统开发人员、基于计算机的产品构建者、系统构建者说明“客户”的含义。这里要注意：该问题可能比你想像得更复杂！
- 7.11 为什么我们说分析模型表现了系统的时间快照？ 205
- 7.12 你认为当需求确认揭示了一个错误时将发生什么？谁将参与修正错误？
- 7.13 使用7.6.2节说明的模板，为指导老师建议的应用提出一个或多个分析模式。
- 7.14 用你自己的话描述一个分析模式。
- 7.15 用例“异常”代表什么？
- 7.16 在需求工程活动的协商情境中，“双赢”意味着什么？
- 7.17 简短地讨论一个分析模型的每个元素，指出每个元素对模型的贡献，每个元素为什么是唯一的以及每个元素所表示的概要信息。

推荐读物与阅读信息

因为需求工程是成功创建任何复杂的基于计算机的系统的关键，所以大量的书籍都在讨论需求工程。Hull和她的同事（《Requirements Engineering》，Springer-Verlag，2002），Bray（《An Introduction to Requirements Engineering》，Addison-Wesley，2002），Arlow（《Requirements Engineering》，Addison-Wesley，2001），Gilb（《Requirements Engineering》，Addison-Wesley，2000），Graham（《Requirements Engineering and Rapid Development》，Addison-Wesley，1999），Sommerville和Kotonya（《Requirement Engineering: Processes and Techniques》，Wiley，1998），等等，这些书都讨论了该主题。Dan Berry (<http://se.uwaterloo.ca/~dberry/bib.html>) 发表了关于需求工程主题的各种思路的文章。

Lauesen（《Software Requirements: Styles and Techniques》，Addison-Wesley，2002）全面概述了需求分析方法和表示方法。Weigers（《Software Requirements》，Microsoft Press，1999）、Leffingwell及其同事（《Managing Software Requirements: A Unified Approach》，Addison-Wesley，2000）提出了一系列有用的需求最佳实践，并为需求工程过程的很多方面提供了实用指导原则。

Robertson和Robertson（《Mastering the Requirements Process》，Addison-Wesley，1999）提供了一个详细的案例学习，帮助解释软件需求分析和分析模型的所有方面。Kovitz（《Practical Software Requirements: A Manual of Content and Style》，Manning Publications，1998）讨论了一个逐步实施的需求分析方法，并为那些必须开发需求规格说明的人员提供了风格指南。Jackson（《Software Requirements Analysis and Specification: A Lexicon of Practices, Principles and Prejudices》，Addison-Wesley，1995）按字母A~Z的顺序概括了该主题。Ploesch（《Assertions, Scenarios and Prototypes》，Springer-Verlag，2003）讨论了开发软件需求的先进技术。

Windle和Abreo（《Software Requirements Using the Unified Process》，Prentice-Hall，2002）从统一过程和UML符号的角度讨论了需求工程。Alexander和Steven（《Writing Better Requirements》，Addison-Wesley，2002）提出了一套简短的指导原则，目的是指导编写清楚的需求，使用场景表现需求并评审最终结果。

用例建模通常在创建分析模型的所有其他方面时使用，该主题在Bittner和Spence（《Use-Case Modeling》，Addison-Wesley，2002）、Cockburn[COC01]、Armour和Miller（《Advanced

Use-Case Modeling: Software Systems》, Addison-Wesley, 2000)、Kulak和同事(《Use Cases: Requirements in Context》, Addison-Wesley, 2000)、Schneider和Winters(《Applying Use Cases》, Addison-Wesley, 1998)的著作中都有详细讨论。

在Internet上有大量的、丰富的关于需求工程和分析的信息资源。与需求工程和分析相关的最新的Web引用列表可以在SEPA的Web站点<http://www.mhhe.com/pressman>上找到。
[206]



第8章 构建分析模型

要点浏览

概念：文字记录是极好的交流工具，但并不必然是表达计算机软件需求的最好方式。分析建模使用文字和图表的综合形式，以相对容易理解的方式描绘需求的数据、功能和行为，更重要的是，可以更直接地评审它们的正确性、完整性和一致性。

人员：软件工程师（有时被称作分析师）使用从客户那里提取的需求构建模型。

重要性：为了确认软件需求，你需要从不同的视角检验需求。分析建模从多个“维度”表现需求，这样就增加了查明错误、消除不一致性、发现遗漏的几率。

步骤：可以使用很多不同格式的图表为信息、功能和行为需求建模。基于场景

的建模从用户的角度表现系统；面向流的建模在说明数据对象如何通过处理函数进行转换方面提供了指示；基于类的建模定义了对象、属性和关系；行为建模描述了系统状态、类和事件在这些类上的影响。一旦创建了模型的雏形，就将不断改进，并分析评估其清晰性、完整性和一致性。最终的分析模型将由所有的共利益者确认。

工作产品：可以选择大量的图表格式用于分析模型，每种表现方法都提供了一个或多个系统元素的视图。

质量保证措施：必须评审分析建模工作产品的正确性、完整性和一致性，必须反映所有共利益者的要求并建立一个可以从中导出设计的基础。

关键概念
分析建模
行为的
基于类的
控制规格说明
类-职责-协作者
数据
面向流的
基于场景的
类
数据流图
数据对象
域分析
结构化分析
单凭经验的方法

在技术层面上，软件工程开始于一系列的建模工作，最终生成待开发软件的需求规格说明和全面的设计表示。分析模型实际上是一组模型，是系统的第一技术表示。

在一本关于分析建模方法的开创性书籍中Tom DeMarco[DEM79]这样描述该过程：

回顾已确认的问题和分析阶段的过失，我建议对分析阶段的目标进行以下的增补：

- 分析的结果必须是高度可维护的，尤其是要将此结果应用于目标文档[软件需求规格说明]。
- 必须使用一种有效的分割方法解决规模问题，维多利亚时代小说式的规格说明是不行的。
- 尽可能使用图形符号。
- 考虑问题必须区分逻辑的[本质]和物理的[实现]。
无论如何，我们至少需要……
- 某种帮助我们划分需求的方法，并在规格说明前用文档记录该划分……

- 某种跟踪和评估接口的手段……
- 使用比叙述性文本更好的新工具来描述逻辑和策略……

尽管DeMarco在25年前就写下了关于分析建模的特点，但他的意见仍然适用于现代的分

207 析建模方法和表示方法。

8.1 需求分析

需求分析产生软件操作特征的规格说明，指明软件和其他系统元素的接口，建立软件必须满足的约束。需求分析让软件工程师（有时这个角色也被称作分析师或建模师）细化在前期需求工程工作中建立的基础需求，并建立模型描述用户场景、功能活动、问题类和类之间的关系、系统和类行为以及数据流。



一旦软件完成
后，分析模型和
需求规格说明将
成为提供评估软
件质量的手段。

需求分析向软件设计者提供信息、功能和行为的表示，这些表示可以被转化为结构、接口和构件级的设计。最终，在软件完成后，分析模型和需求规格说明就为开发人员和客户提供了评估软件质量的手段。

在整个分析建模过程中，软件工程师的主要关注点集中在“做什么”而不是“怎么做”方面。包括：系统处理什么对象？系统必须执行什么功能？系统显示什么行为？定义什么接口？有什么约束¹？

在前面的章节中，我们注意到在该阶段要得到完整的需求规格说明是不可能的。客户也许无法精确地确定想要什么，开发人员也许无法确定能恰当地实现功能和性能的特定方法，208 这些现实都削弱了迭代需求分析和建模方法的效果。分析师将为已经知道的内容建模，并使用该模型作为软件进一步扩展的设计基础²。

8.1.1 整体目标和原理

分析模型必须实现三个主要目标：(1) 描述客户需要什么；(2) 为软件设计奠定基础；(3) 定义在软件完成后可以被确认的一组需求。分析模型在系统级描述（第6章）（说明系统在软件、硬件、数据、人员和其他系统元素发挥作用后应达到的整体功能）和软件设计（第9章）（说明软件的应用程序结构、用户接口、构件级的结构）的差距之间建立桥梁。这个关系如图8-1所示。

“值得进攻的问题总是通过反击证明其价值。”

—Piet Hein

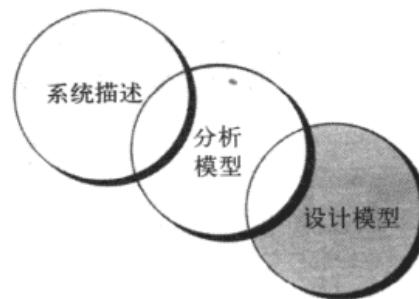


图8-1 分析模型在系统描述和设计模型之间建立桥梁

重要的是要注意到在系统描述中给出（在更高的抽象层）分析模型的某些元素，并且需求工程的工作实际上是作为系统工程的一部分开始的。此外，分析模型的所有元素都可以直接跟踪到设计模型。通常难以清楚地区分这两个重要的建模活动之间的设计和分析工作，有些设计总是作为分析的一部分进行，而有些分析将在设计中进行。

¹ 应注意到，当客户在技术上更有经验时，会有向既表明“做什么”又表明“怎么做”方式规格说明发展的趋势。但是，主要的工作仍应集中于“做什么”。

² 另一种选择是，软件团队创建一个原型（第3章）以便更好地理解系统需求。

8.1.2 分析的经验原则

Ariow和Neustadt[ARL02]提出了大量有价值的经验原则，在创建分析模型时应该遵循这些经验原则：

- 模型应关注在问题域或业务域内可见的需求，抽象的级别应该相对高一些。“不要陷入细节” [ARL02]，即不要试图解释系统将如何工作。
- 分析模型的每个元素都应能增加对软件需求的整体理解，并提供对信息域、功能和系统行为的深入理解。
- 关于基础结构和其他非功能的模型应推延到设计阶段再考虑。例如，可能需要一个数据库，但是只有在已经完成问题域分析之后才应考虑实现数据库所必需的类，访问数据库所需的函数，以及使用时所表现出的行为。
- 最小化整个系统内的关联。表现类和功能之间的联系非常重要，但是，如果“互联”的层次非常高，应该想办法减少互联。
- 确认分析模型为所有共利益者都带来价值。对模型来说，每个客户都有自己的使用目的。例如，利益相关的业务人员将使用模型确认需求；设计人员将使用模型作为设计的基础；质量保证人员将使用模型帮助规划验收测试。
- 尽可能保持模型简洁。如果没有提供新的信息，不要添加附加图表；如果一个简单列表就够用，不要使用完整的符号表。

8.1.3 域分析

WebRef

很多关于域分析的有用信息可以在www.iturls.com/English/SoftwareEngineering/SE_mod5.asp中找到。

在需求工程讨论中（第7章），我们注意到分析模型通常在特定业务领域内的很多应用中重复发生。如果用一种方式对这些模式加以定义和分类，让软件工程师或分析师识别并复用这些模式，将促进分析模型的创建。更重要的是，应用可复用的设计模式和可执行的软件构件的可能性将显著增加。这将把产品投放市场的时间提前，并减少开发费用。

但问题是，首先如何识别分析模式？由谁来对分析模式进行定义和分类，并为随后的项目准备好分析模式？这些问题的答案在域分析中。Firesmith [FIR93]这样描述域分析：

210

软件域分析是识别、分析和详细说明来自某个特定应用领域的公共需求，特别是那些在该应用领域内被多个项目重复使用的……[面向对象的域分析是]在某个特定应用领域内，根据通用的对象、类、部件和框架，识别、分析和详细说明公共的、可复用的能力。

WebRef

可以在www.sei.cmu.edu/str/descriptions/deda.html上找到一篇有价值的关于域工程和分析的论述。

“特定应用领域”的范围从航空电子设备到银行业，从多媒体视频游戏到医疗设备中的嵌入式软件。域分析的目标很简单，就是：查找或创建那些分析类和（或）能够广泛应用的、共有的功能和特点，这样就可以复用³。

在某种程度上，域分析师的角色类似于重型机械制造业中一名优秀的刀具工的角色。刀具工的工作是设计并制造工具，这些工具可被很多人用来进

³ 一个域分析的补充观点是：“包括为域建模，因此软件工程师和其他的共利益者可以更好地学习……不是所有域的类都必然导致可复用的类” [LET03]。

行类似的而不一定是同样的工作。域分析师⁴的角色是发现和定义可复用的分析模式、分析类和相关的信息，这些也是可用于类似但不要求必须是完全相同的应用。

“最好的学习技巧是一点一点地理解。”

——John Locke

图8-2[ARA89]说明了域分析过程的关键输入和输出。应该调查领域知识的来源以便于确定可以在不同领域内复用的对象。

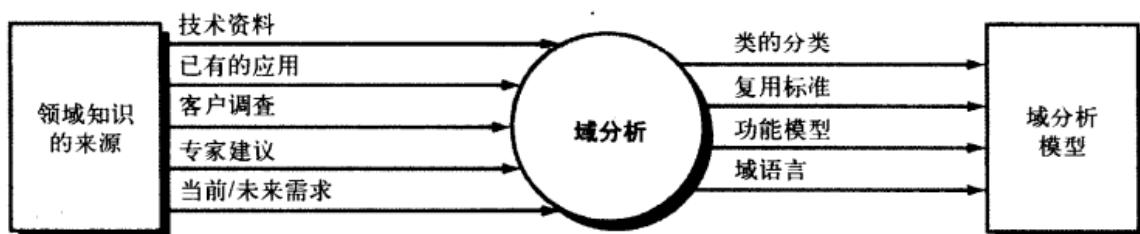


图8-2 域分析的输入和输出

8.2 分析建模的方法

一种考虑数据和处理的分析建模方法被称作结构化分析，其中数据作为独立实体转换。

211 数据对象建模定义了对象的属性和关系，操作数据对象的处理建模应表明当数据对象在系统内流动时处理如何转换数据。

分析建模的第二种方法称作面向对象的分析，这种方法关注于定义类和影响客户需求的类之间的协作方式。UML和统一过程（第3章）主要是面向对象的。

“分析容易使人灰心丧气，全都是复杂的人际关系、不确定的和困难的东西。总而言之，分析是迷人的。一旦沉迷，原来轻松构建系统的快乐将难以令你满足。”

——Tom DeMarco

尽管本书中我们提出的分析模型综合了两种方法的特点，但是软件团队往往选择一种方法并排斥另一种方法中的所有表示手段。问题不是哪一种方法最好，而是怎么组合这些表示手段才能够为共利益者提供最好的软件需求模型和过渡到软件设计的最有效方法。

分析模型将生成图8-3所示的每个建模元素的派生类。然而，不同项目之间，每个元素（即用于构建元素和模型的图表）的特定内容可能因项目而异。就像我们在本书中多次提到的那样，软件团队必须想办法保持模型的简单性。只有那些为模型增加价值的建模元素才能使用。

“我们为什么应该构建模型？为什么不直接构建系统本身？答案是我们可以按照如下方式构建模型：突出或强调某些关键的系统特征，同时削弱系统的其他方面。”

——Ed Yourdon

⁴ 不要认为因为有域分析师在工作，软件工程师就不需要理解应用的领域。软件团队的每个成员都应该对软件将要使用的领域有一定的了解。

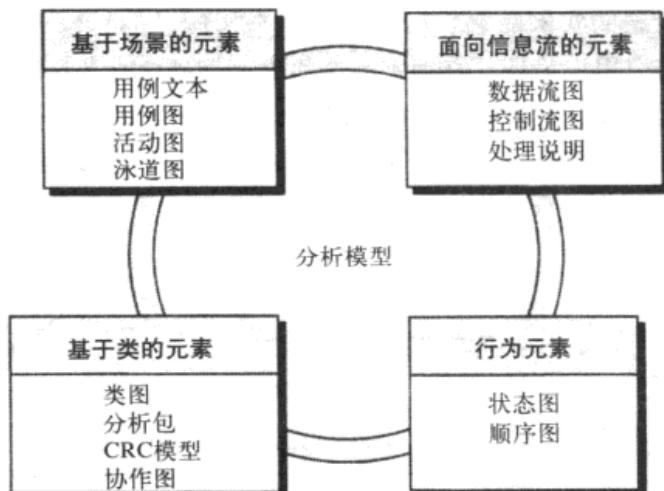


图8-3 分析模型的元素

212

8.3 数据建模概念

WebRef

在www.datamodel.org上可以找到数据建模的有用信息。

分析建模通常开始于数据建模。软件工程师或分析师需要定义在系统内处理的所有数据对象、数据对象之间的关系以及其他与这些关系相关的信息。

8.3.1 数据对象

数据对象是几乎任何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的特征或属性的事物。因此，“宽度”（单个的值）不是有效的数据对象，但是“维度”（包括宽度、高度和深度）可以被定义为一个对象。

数据对象如何在一个应用的环境内表现自己？

数据对象可能是外部实体（例如产生或使用信息的任何东西）、事物（例如报告或显示）、发生（例如电话呼叫）或事件（例如警报）、角色（例如销售人员）、组织单位（例如财务部）、地点（例如仓库）或结构（例如文件）。例如，人或车可以被认为是数据对象，因为他们可以用一组属性来定义。“数据对象描述”包括了数据对象及其所有属性。

POINT
数据对象是被计算机软件处理的任意复合信息的表示。

数据对象只封装数据——在数据对象内没有对作用于数据的操作的引用⁵。因此，数据可以表示为如图8-4所示的一张表，表头反映了对象的属性。在这个例子中，car是通过make、model、ID number、body type、color和owner定义的。该表的表体表示了数据对象的特定实例。例如，Chevy Corvette是数据对象car的一个实例。

213

8.3.2 数据属性

数据属性定义了数据对象的性质，可以具有三种不同的特性之一。它们可以用来：(1) 为

⁵ 这种区别将数据对象与面向对象方法中的“类”或“对象”区分开来。



属性命名某数据对象，描述其特征，并在某些情况下引用另一个对象。

数据对象的实例命名；(2) 描述这个实例；(3) 建立对另一个表中的另一个实例的引用。另外，必须把一个或多个属性定义为标识符——也就是说，当我们要找到数据对象的一个实例时，标识符属性成为一个“键”。在某些情况下，标识符的值是唯一的，但不是必须的。在数据对象car的例子中，ID号可以作为一个合理的标识符。

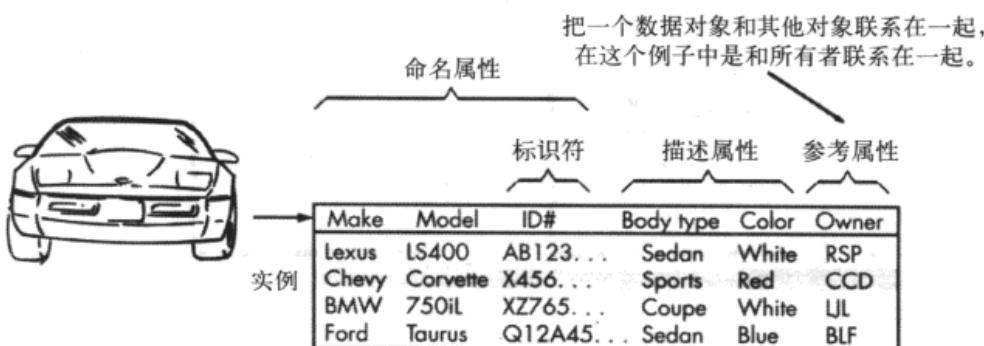


图8-4 数据对象的表格表示

WebRef

对那些希望进行彻底的数据建模的人来说，一种称为“标准化”的概念非常重要，可以在 www.datamodel.org 中找到这方面的有用信息。

通过对问题环境的理解可以恰当地确定特定数据对象的一组属性。car的属性可以很好地用于汽车运输部门的应用系统，但这些属性对于汽车制造公司来说是无用的，汽车公司需要制造控制软件。在后一种情况下，car的属性可能也包括ID number、body type和color，但为了使car在制造控制环境中成为一个有用的对象，必须增加一些与生产相关的其他的属性（如interior code、drive train type、trim package designator、transmission type）。

INFO

数据对象和OO类——它们是同一个东西吗？

当讨论数据对象时会出现一个常见的问题：数据对象和面向对象的类是同一个东西吗？答案是否定的。

数据对象定义了一个复合的数据项，也就是说合并一组独立的数据项（属性）并为数据项集合取个名字（数据对象名）。一个OO类封装了数据属性但也合并了对这些属性所定义数据进行处理的操作。另外，类的定义暗示了一个作为面向对象软件工程方法一部分的全面的基础设施。类之间通过消息通信，它可以按层次关系组织并为类的实例——对象——提供继承属性。

8.3.3 关系



关系指明数据对象相互“连接”的方式。

数据对象可以以多种不同的方式互相连接。考虑两个数据对象：person和car。这些对象可以使用图8-5a所示的标记表示。在person和car之间可以建立联系，因为这两个对象之间是相关的。但这个关系是什么呢？为确定答案，我们必须理解在将要构建的软件的环境中人（在这里是指车主）和车的角色。我们可以用一组“对象/关系对”来定义有关的关系，例如：

- 有车的人。
- 汽车保险投保人。

关系owns和insured to drive定义了person和car之间的相关连接。图8-5b以图形方式说明了这些对象/关系对，并提供了关于关联方向的重要信息，这一方向信息通常可以减少不确定性或误解。

8.3.4 基数和形态

数据建模的基本元素——数据对象、属性和关系——为理解问题的信息域提供了基础。然而，还必须理解与这些基本元素相关的其他元素。

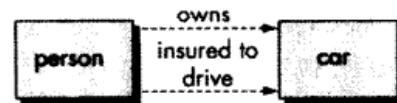
我们已经定义了一组对象并表示了与之绑定的对象/关系对。但是，就软件工程的目的而言，简单地说对象X与对象Y相关并没有提供足够的信息。我们必须理解对象X的多少次出现和对象Y的多少次出现相关，这引出了被称为基数的数据建模概念。

一个数据对象与另一个数据对象的多次出现相关，该如何处理这种情况？

数据模型必须能够表示在一个给定的关系中对象出现的次数。Tillmann [TIL93]以如下的方式定义了对象/关系对的基数：“基数是关于一个（对象）的出现次数可以与另一个（对象）的出现次数相关联的规格说明”。例如，一个对象只能和一个其他对象关联（1：1关系）；一个对象可以和很多对象关联（1：N关系）；一个对象的多次出现可以和另一个对象的多次出现关联（M：N关系）⁶。基数也可以定义“能够参与一个关联的最大对象数”[TIL93]。但是，它没有指示出一个特定的数据对象是否必须参与在关系中。为说明该信息，数据模型为“对象/关系”对增加了形态。



a) 数据对象之间的基本连接



b) 数据对象之间的关系

图8-5 数据对象之间的关系

实体-关系图

对象/关系对是数据对象的基石。可以使用实体/关系图（ERD）⁷图形化地表示这些对象/关系对。ERD最初是由Peter Chen[CHF77]为关系数据库系统设计提出的，并由其他人进行了扩展。ERD识别了一组基本元素：数据对象、属性、关系以及各种类型的指示符。使用ERD的主要目的是表示数据对象及其关系。

已经介绍过基本的ERD符号，带标记的矩形表示数据对象，连接对象的带标记的线表示关系。在ERD的某些变形中，连接线包含一个带有关系标记的菱形。数据对象的连接和关系使用各种指示基数和形态的特殊符号来建立。

关于数据建模和实体关系图的更多信息，感兴趣的读者可以参考[THA00]。

INFO

对于关系的出现，如果没有明确的必要性或关系是可选的，那么关系的形态是0；如果关系必须出现一次，那么形态是1。

⁶ 例如，一个叔叔可以有多个侄子，而一个侄子也可以有多个叔叔。

⁷ 尽管ERD仍然在某些数据库设计应用中使用，但UML符号在现在的数据设计中更通用。

“为了使一个信息系统有用、可靠、适应性强和经济实用，首先必须基于可靠的数据建模，其次只能基于过程分析……因为数据结构本质上和实际相关，而过程和技术相关。”

——Duncan Dwelle

SOFTWARE TOOLS

数据建模

目的：数据建模工具为软件工程师提供表现数据对象、数据对象的特点和数据对象的关系的能力。主要用于大型数据库应用系统和其他信息系统项目，数据建模工具以自动化的方式创建全面的实体-关系图、数据对象字典以及相关模型。

机制：该类型的工具帮助用户描述数据对象及其关系。在某些情况下，工具使用ERD符号；有些情况下工具使用其他一些原理为关系建模。该类工具通过为公共数据库管理系统（DBMS）生成数据库模式帮助创建数据库模型。

代表性工具⁸

AllFusion ERWin，由Computer Associates (www3.ca.com) 开发，辅助设计数据库的数据对象、恰当的结构和关键元素。

ER/Studio，由Embarcadero Software (www.embarcadero.com) 开发，支持实体-关系建模。

Oracle Designer，由Oracle Systems (www.oracle.com) 开发，为业务处理、数据实体和关系建模，并转化成可以生成完整应用系统和数据库的设计。

MetaScope，由Madrone Systems (www.madronesystems.com) 开发，是一个支持图形化显示数据的低成本数据建模工具。

ModelSphere，由Magna Solutions GmbH (www.magnasolutions.com) 开发，支持各种相关的建模工具。

Visible Analyst，由Visible Systems (www.visible.com) 开发，支持包括数据建模在内的各种分析建模功能。

216

8.4 面向对象的分析

任何关于面向对象分析的讨论，都必须先解释术语面向对象（Object-oriented, OO）。什么是面向对象的观点？某个方法为什么被认为是面向对象的？什么是对象？在上世纪80年代和90年代，就在OO赢得了大量拥护者的同时，关于这些问题的正确答案还存在很多不同的意见（如[BER93]、[TAY90]、[STR88]、[BOO86]）。今天，对于OO已经形成了一致的观点。

面向对象的分析（OOA），其目的是定义与即将解决的问题相关的所有类（以及与其相关的关系和行为）。为实现这一点，必须完成如下一些工作：

1. 在客户和软件工程师之间必须对基本的用户需求进行交流（第7章）。
2. 必须确定类（也就是说，定义属性和方法）。
3. 定义类的层次结构。

⁸ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

4. 表现对象与对象的关系（对象连接）。
5. 必须为对象行为建模。
6. 上述1~5的工作步骤重复迭代直至模型完成。

与使用更传统的输入—处理—输出（信息流）模型或从多层次信息结构中衍生出的专有模型不同，OOA构建的是面向类的模型，这依赖于对OO概念的理解。

INFO

面向对象的概念

面向对象的概念在软件工程界已经发展得很完善。下面简要地说明在分析建模中经常遇到的重要的OO概念。和软件设计相关的其他更多的OO概念将在第10章中说明。

属性——说明一个类的数值集合。

类——封装数据和过程的抽象，这些是说明某些真实世界中的实体的内容和行为所必需的。换种方式说，类是一组相似对象的概括说明（如，模板、模式或蓝图）。

对象——某个特定类的实例。对象继承类的属性和操作。

操作——也称作方法和服务，表现类的某个行为。

子类——超类的特化，子类可以从超类继承属性和操作。

超类——也称作基类，是一组相关类的泛化。

8.5 基于场景建模

尽管可以用多种方式度量基于计算机的系统或产品，用户的满意度仍是其中最重要的。如果软件工程师了解最终用户（和其他参与者）希望如何与系统交互，软件团队将能够更好地、更准确地刻画系统特征，完成更有针对性的分析和设计模型。因此，使用UML分析建模，将从开发用例、活动图和泳道图形式的场景开始。

8.5.1 编写用例



在某些情况下，用例成为最主要的需求工程机制，但是，这不意味着你应该放弃在第7章中讨论的概念和技术。

用例捕获信息的产生者、使用者和系统本身之间发生的交互。在本节，我们检验如何开发用例，这是分析建模活动的一部分⁹。

用例的概念（第7章）比较容易理解——从某个特定参与者的角度用简单易懂的语言说明一个特定的使用场景¹⁰。但是我们如何知道：（1）编写什么？（2）写多少？（3）编写说明应该多详细？（4）如何组织说明？如果想让用例像一个分析建模工具那样提供价值，那么必须回答这些问题。

“[用例]只是帮助定义系统之外（参与者）存在什么以及系统应完成什么（用例）。”

——Ivar Jacobson

⁹ 用例是用户接口的分析建模中特别重要的一部分，将在第12章详细讨论接口分析。

¹⁰ 参与者不是一个确定的人员，而是一个人员（或设备）在特定的环境内所扮演的角色，参与者“呼叫系统并由系统提供一种服务”[COC01]。

“为了使一个信息系统有用、可靠、适应性强和经济实用，首先必须基于可靠的数据建模，其次只能基于过程分析……因为数据结构本质上和实际相关，而过程和技术相关。”

——Duncan Dwelle

SOFTWARE TOOLS

数据建模

目的：数据建模工具为软件工程师提供表现数据对象、数据对象的特点和数据对象的关系的能力。主要用于大型数据库应用系统和其他信息系统项目，数据建模工具以自动化的方式创建全面的实体-关系图、数据对象字典以及相关模型。

机制：该类型的工具帮助用户描述数据对象及其关系。在某些情况下，工具使用ERD符号；有些情况下工具使用其他一些原理为关系建模。该类工具通过为公共数据库管理系统（DBMS）生成数据库模式帮助创建数据库模型。

代表性工具⁸

AllFusion ERWin，由Computer Associates (www3.ca.com) 开发，辅助设计数据库的数据对象、恰当的结构和关键元素。

ER/Studio，由Embarcadero Software (www.embarcadero.com) 开发，支持实体-关系建模。

Oracle Designer，由Oracle Systems (www.oracle.com) 开发，为业务处理、数据实体和关系建模，并转化成可以生成完整应用系统和数据库的设计。

MetaScope，由Madrone Systems (www.madronesystems.com) 开发，是一个支持图形化显示数据的低成本数据建模工具。

ModelSphere，由Magna Solutions GmbH (www.magnasolutions.com) 开发，支持各种相关的建模工具。

Visible Analyst，由Visible Systems (www.visible.com) 开发，支持包括数据建模在内的各种分析建模功能。

216

8.4 面向对象的分析

任何关于面向对象分析的讨论，都必须先解释术语面向对象（Object-oriented, OO）。什么是面向对象的观点？某个方法为什么被认为是面向对象的？什么是对象？在上世纪80年代

上面讨论的SafeHome住宅监视功能（子系统）确定了如下由参与者房主执行的功能（简化列表）：

- 通过Internet访问摄像头监视功能。
- 选择将要查看的摄像头。
- 提供所有摄像头的缩略视图。
- 在计算机的窗口中显示摄像头视图。
- 控制某个特定摄像头的镜头转动和缩放。
- 可选择地记录摄像头的输出。
- 回放摄像头的输出。

随着和共利益者（扮演房主的人）更多地交谈，需求收集团队为每个标记的功能开发用例。通常，用例首先用非正式的描述性风格编写。如果需要更正式一些，可以使用类似于第7章中提出的某个结构化的形式重新编写同样的用例，在本节的后面将重新生成用例。

为了举例说明，考虑功能“访问摄像头监视设备——显示摄像头视图（access camera surveillance-display camera views, ACS-DCV）”，扮演参与者房主的共利益者可能会编写如下说明：

用例：访问摄像头监视设备——显示摄像头视图（ACS-DCV）

参与者：房主

如果我位于远方，我可以使用任何计算机上的合适的浏览器软件登录SafeHome产品网站。输入我的用户ID和两级密码，一旦被确认，我可以访问已安装的SafeHome系统的所有功能。为取得某个摄像头视图，从显示的主功能按钮中选择“监视”，然后选择“选取摄像头”，将会显示房屋的平面设计图，再选择感兴趣的摄像头。另一种可选方法是，通过选择“所有摄像头”，可以同时从所有的摄像头查看缩略视图快照。当选择了某个摄像头时，可以选择“查看”，然后以每秒一帧速度显示的图像就可以在窗口中显示。如果希望切换摄像头，选择“选取摄像头”，原来窗口显示的信息消失，并且再次显示房间的平面设计图，然后就可以选择感兴趣的摄像头，以便显示新的查看窗口。

描述性用例的一种变形是通过用户活动的顺序序列表现交互，每个活动由声明性的语句表示。再以ACS-DCV功能为例，我们可以写下：

用例：访问摄像头监视设备——显示摄像头视图（ACS-DCV）

参与者：房主

1. 房主登录SafeHome产品网站。
2. 房主输入用户ID。
3. 房主输入两个密码（每个至少八个字符长度）。
4. 系统显示所有的主要功能按钮。
5. 房主从主要功能按钮中选择“监视”。
6. 房主选择“选取摄像头”。
7. 系统显示房屋的平面设计图。
8. 房主从平面设计图中选择某个摄像头图标。

9. 房主选择“查看”按钮。
10. 系统显示一个由摄像头ID确定的查看窗口。
11. 系统在查看窗口内以每秒一帧的速度显示视频输出。

重要的是注意到，这个连续的陈述没有考虑其他可能的交互（描述更加自由随意而且确实表达了一些其他选择）。这种类型的用例有时被称作主场景[SCH98]。

“用例可以被用于很多[软件]过程，我们感兴趣的是迭代和风险驱动过程。”

——Geri Schneider和Jason Winters

当然，要完整地理解所描述的功能，必需说明可能的交互。因此，主场景中的每个步骤将通过如下提问得到评估[SCH98]：

- | | |
|---|---|
|  在开发场景时，如何检查动作的可选择的过程？ | <ul style="list-style-type: none"> • 在这一点，参与者能进行一些其他活动吗？ • 在这一点，参与者有没有可能遇到一些错误的情况？ • 在这一点，参与者有没有可能遇到一些其他的行为（如由一些参与者控制之外的事件调用）？如果有，是什么？ |
|---|---|

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

例如，考虑前面描述的主场景的第6步和第7步：

6. 房主选择“选取摄像头”。
7. 系统显示房屋的平面设计图。

在这一点上，参与者能进行一些其他活动吗？答案是肯定的。参考非正式的描述说明，参与者可以选择同时查看所有摄像头的缩略视图。因此，一个次场景可能是“查看所有摄像头的缩略视图”。

在这一点，参与者有没有可能遇到一些错误的情况？作为基于计算机的系统操作，可能出现许多错误情况。在这里，我们仅仅考虑在第6步和第7步中说明的活动的直接错误条件，问题的答案还是肯定的。带有摄像头图标的房屋平面图可能还没有形式，这样选择“选取摄像头”就导致错误情况：“没有为该房屋配置平面设计图”¹²。该错误情况就成为一个次场景。

在这一点，参与者有没有可能遇到一些其他的行为？问题的答案再一次是肯定的。当第6步和第7步发生时，系统可能遇到报警。这将导致系统显示一个特殊的报警通知（类型、地点、系统动作），并向参与者提供和报警性质相关的一组操作。因为这个次场景可以在所有的实际交互中发生，所以不会成为ACS-DCV用例的一部分。而且，将开发一个单独的用例——“遇到报警条件”，这个用例可以根据需要被其他用例引用。

次场景将被表现为说明ACS-DCV的基本序列的异常，所引用的正式的用例模板将在补充内容中显示。

¹² 在该例子中，另一个参与者，系统管理员必须配置平面设计图、安装并初始化（如分配设备ID）所有的摄像头，并进行测试，确保每个摄像头都可以通过系统和通过平面设计图访问到。

SAFEHOME

监视功能的用例模板

用例：访问摄像头监视设备——显示摄像头视图（ACS-DCV）。

主参与者：房主。

环境目标：从任何远程地点通过Internet查看遍布房间的摄像头输出。

前提条件：系统必须被完整配置；必须获得正确的用户ID和密码。

触发器：房主出门在外时决定查看房屋内部。

场景：

1. 房主登录SafeHome产品网站。
2. 房主输入用户ID。
3. 房主输入两个密码（每个都至少有8个字符的长度）。
4. 系统显示所有的主要功能按钮。
5. 房主从主要功能按钮中选择“监视”。
6. 房主选择“选取摄像头”。
7. 系统显示房屋的平面设计图。
8. 房主从房屋的平面设计图中选择某个摄像头的图标。
9. 房主选择“视图”按钮。
10. 系统显示一个由摄像头ID确定的查看窗口。
11. 系统在查看窗口中每秒一帧地显示视频输出。

异常：

1. ID或密码不正确或不被识别——阅读用例：“确认ID和密码”。
2. 没有为该系统配置监视功能——系统显示恰当的错误消息；阅读用例：“配置监视功能”。
3. 房主选择“查看所有摄像头的缩略视图快照”——阅读用例：“查看所有摄像头的缩略视图快照”。
4. 平面设计图不可用或还没有配置——显示恰当的错误消息，阅读用例：“配置平面设计图”。
5. 满足报警条件——阅读用例：“满足报警条件”。

优先级：中等优先级，必须在基础功能之后实现。

何时有效：第三个增量。

使用频率：不经常。

参与者的联系渠道：通过基于电脑的浏览器和Internet连接到SafeHome网站。

次要参与者：系统管理员，摄像头。

次要参与者的联系渠道：

1. 系统管理员：基于计算机的系统。
2. 摄像头：无线连接。

未解决的问题：

1. 有什么机制保护公司雇员在未授权的情况下使用该功能？
2. 足够安全吗？黑客入侵该功能将代表最主要的个人隐私受侵。
3. 在给定摄像头视图所要求的带宽下，可以接受系统通过Internet的响应吗？
4. 当可以使用高带宽的连接时，能开发比每秒一帧更快速度的视频提供能力吗？

WebRef

什么时候结束编写用例？关于该话题的论述可以参阅`ootips.org/use-cases-done.html`。

在很多情况下，不需要创建使用场景的图形化表示。然而，图形化的表示可以促进理解，尤其是当场景比较复杂时。正如我们在第7章提到的，UML的确为用例提供了图形化表现的能力。图8-6为SafeHome产品描述了一个初步的用例图，每个用例由一个椭圆表示。本节只详细讨论了ACS-DCV用例。

8.5.2 开发活动图



一个UML活动图表现了实施某个功能时发生的活动和判定。

223

UML活动图（在第6章和第7章中曾简要讨论过）通过提供特定场景内交互流的图形化表示来补充用例。类似于流程图，活动图使用两端为半圆形的矩形表示一个特定的系统功能，箭头表示通过系统的流，判定菱形表示判定分支（每个箭头从标记的菱形发出），水平线意味着并行发生的活动。ACS-DCV功能的活动图如图8-7所示。应注意到活动图增加了额外的细节，而这些细节是用例不能直接描述的（是隐含的）。例如，用户输入用户ID和密码的次数可能希望受到限制，这可以通过重新输入提示之下的判定菱形来体现。

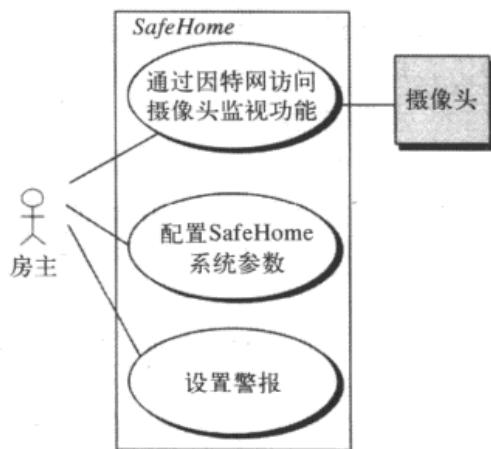


图8-6 SafeHome系统的初步的用例图

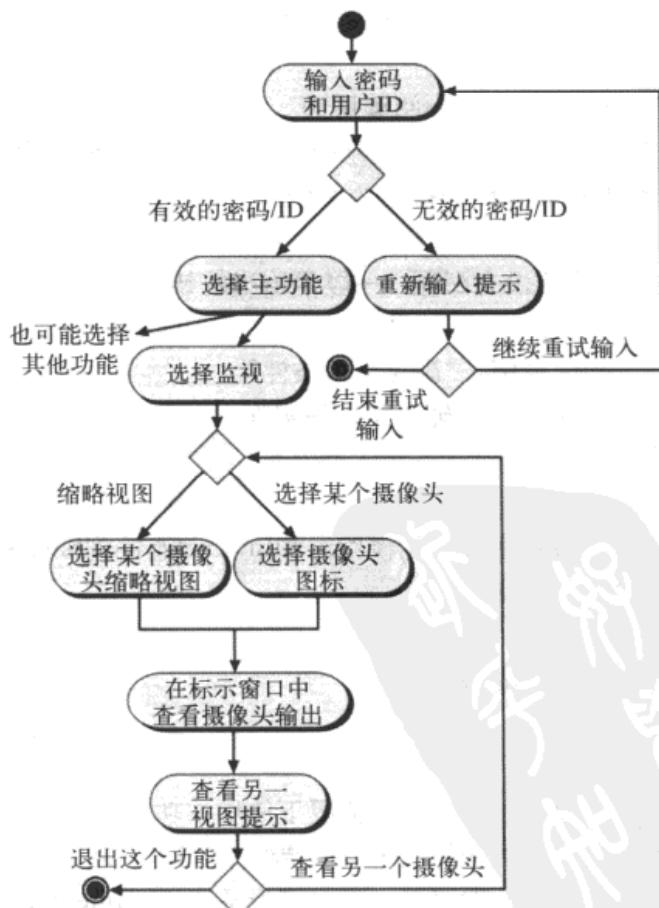


图8-7 ACS-DCV功能的活动图

8.5.3 泳道图

UML泳道图是活动图的一种有用的变形，可让建模人员表示用例所描述的活动流，同时指示哪个参与者（如果在某个特定功能中涉及了多个参与者）或分析类对活动矩形所描述的活动负责。职责由纵向分割图的并列条形部分表示，就像游泳池中的泳道。

POINT
UML泳道图表现了活动流和一些判定，并指明由哪个参与者实施。

三种分析类——房主、摄像头和接口——在图8-7所表示的活动图中具有直接或间接的责任。参看图8-8，活动图被重新排列，这样和某个特殊分析类相关的活动按类落入相应的泳道中。例如，接口类表示房主可见的用户接口。活动图标出对接口负责的两个提示——重新输入提示和查看另一视图提示。这些提示以及与此相关的判定都落入了接口泳道。从该泳道发出的箭头返回到房主泳道，这是因为房主的活动在房主泳道中发生。

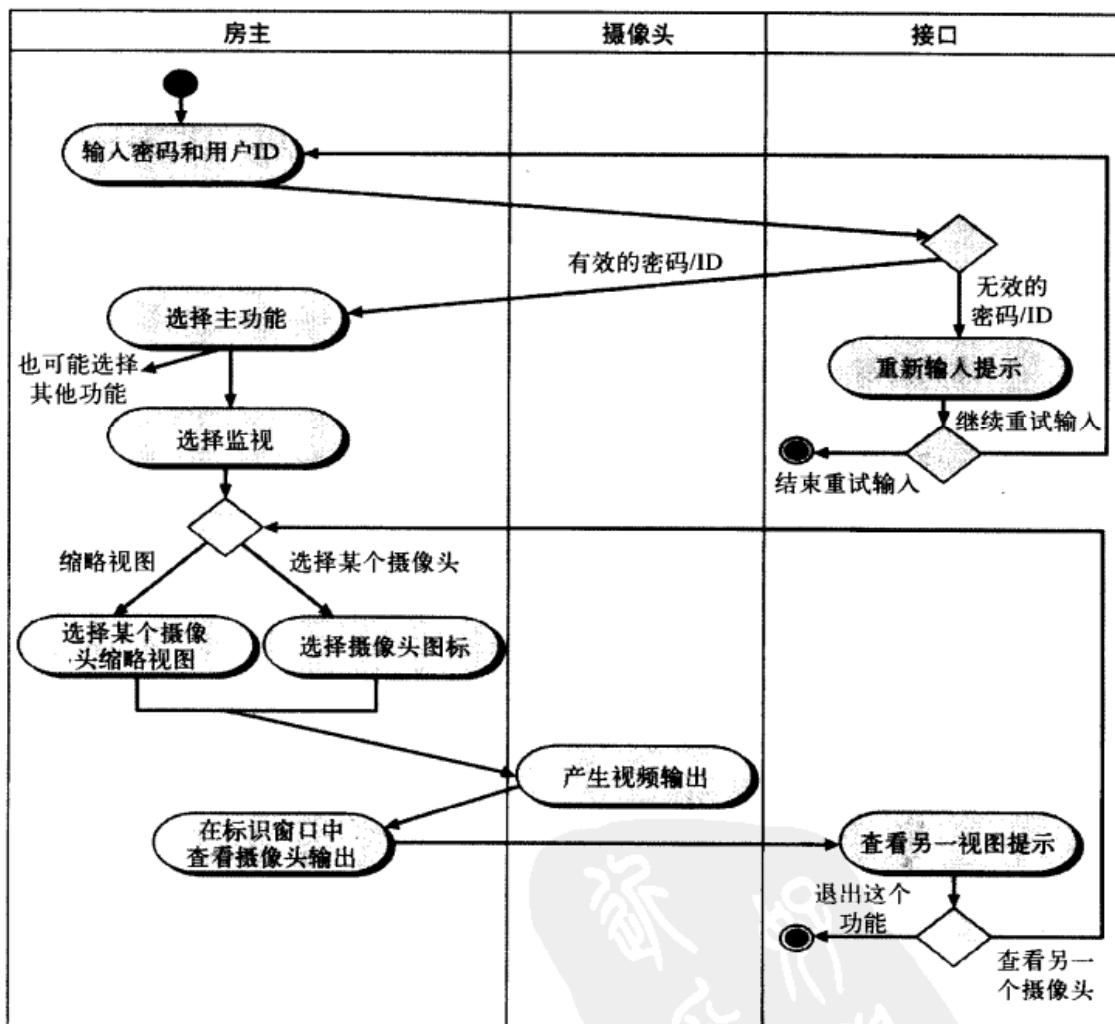


图8-8 ACS-DCV功能的泳道图

8.6 面向流的建模

面向流的数据建模至今仍是最广泛使用的分析表式法之一¹³。尽管数据流图（Data Flow

¹³ 数据流建模是结构化分析中的核心建模活动。

Diagram, DFD) 及相关的图和信息不是UML的正式组成部分,但是它们可以补充UML图并提供对系统需求和流的补充认识。

DFD采取了系统的输入–处理–输出观点,也就是说,流入软件的数据对象,经由处理元素转换,最后以结果数据对象的形式流出软件。数据对象由带标记的箭头表示,转换由圆圈(也称作泡泡)表示。DFD使用分层的方式表示,即第一个数据流模型(有时也称作第0层DFD或环境图)从整体上表现系统,随后的数据流图改进环境图,提供每个后续层增加的细节。

“数据流图的目的是在用户和系统开发人员之间提供语义的桥梁。”

——Kenneth Kozar

8.6.1 创建数据流模型



某些人可能认为DFD是“保守派”,在当前的实践中没有位置。该看法在分析层面上排斥潜在有用的表现模式。但如果有帮助,也可以使用DFD。

226

数据流图有助于软件工程师开发信息域的模型,并同时开发功能域的模型。当DFD被改进到非常详细的程度时,分析师同时也就完成了系统功能分解。并且,当进入使应用具体化的处理时,DFD的求精导致了数据的相应求精。

导出数据流图时有一些很有用的简单的指导原则:(1)第0层的数据流图应将软件/系统描述为一个泡泡;(2)主要的输入和输出应被仔细地标记;(3)通过把在下一层表示的候选处理过程、数据对象和数据存储分离,开始求精过程;(4)应使用有意义的名称标记所有的箭头和泡泡;(5)当从一个层转到另一个层时要保持信息流连续性¹⁴;(6)一次精化一个泡泡。存在一种自然而然的趋势,即数据流图过于复杂。当分析师试图过早地显示过多的细节或在信息流中表示软件流程方面的内容时,就会发生这种情况。

为了举例说明DFD和相关符号的使用,我们再来考虑SafeHome的安全功能。安全功能的环境层DFD如图8-9所示。主要的外部实体(方框)产生系统所使用的信息并使用系统产生的信息,带标记的箭头代表数据对象或数据对象类型的层次。例如,“用户指令和数据”包括了所有的配置命令、所有的激活/撤销命令、所有各式各样的交互以及所有限定或扩展某命令的输入数据。

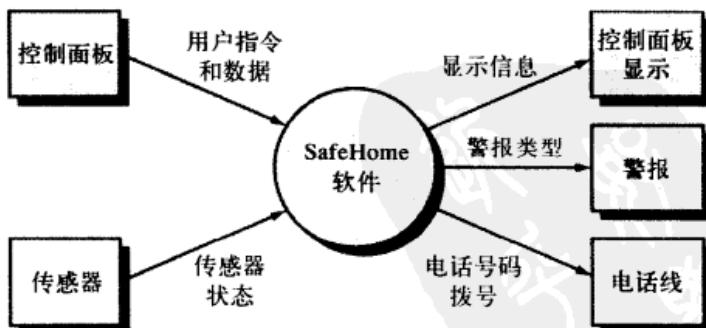


图8-9 SafeHome安全功能的环境层DFD

第0层的DFD现在要扩展到第1层,但我们应如何进行呢?一个简单而有效的方法是对描述

¹⁴ 也就是说,流入系统或流入一层上任一转换的数据对象必须和在更细化的层上流入该转换的是相同的数据对象(或其组成部分)。



当每一层DFD被精化时，必须维护信息流的连续性。这意味着在某一层的输入和输出必须和精化后的层的输入和输出相同。



语法分析并非绝对可靠。但是，如果你正在努力地定义数据对象和变换，它将提供极好的起跳准备。



要确定所分析的处理叙述是在完全相同的抽象层面上编写的。

环境层泡泡的叙述进行“语法分析”[ABB83]，即将第一次需求收集会议中获得的SafeHome处理叙述¹⁵中的所有名词（和名词短语）与动词（和动词短语）分离开来。为了具体说明，考虑如下处理叙述文字，对其中第一次出现的名词加上下划线，第一次出现的动词采用斜体¹⁶。

SafeHome安全功能帮助房主在安装时配置安全系统，监测所有连接到安全系统的传感器，通过Internet、计算机或控制面板和房主交互。

在安装中，SafeHome计算机被用于设计和配置系统，每个传感器被分配一个编号和类型，主人密码被用以控制启动和关闭系统，而且当传感器事件发生时拨打输入的电话号码。

当识别出一个传感器事件时，软件激活附于系统上的可发声的警报，在一定的延迟时间（由房主在系统配置活动中指定）后，软件拨打监测服务的电话号码并提供关于位置的信息，报告被检测到的事件的性质，电话号码将每20秒重拨一次，直至电话接通。

房主通过控制面板、计算机或浏览器这些统称为接口的设施接收安全信息。接口在控制面板、计算机或浏览器窗口中显示提示信息和系统状态信息。房主的交互采用如下形式……

这里提到的语法分析，开始形成一种模式。动词是SafeHome处理，即，它们最终将被表示为后来的DFD中的泡泡；名词是外部实体（方框）、数据或控制对象（箭头）、数据存储（双横线）。进一步观察可以注意到名词和动词之间可以互相连接起来。例如，每个传感器被分配一个编号和类型，这样编号和类型就是数据对象——传感器的属性。因此，通过对任何DFD层次中某个泡泡的处理叙述文字进行语法分析，可以产生许多关于如何精化到下一个层次的有用信息。使用这些信息，第1层的DFD如图8-10所示。如图8-9所示的环境层的处理，已经被扩展为6个处理，这些处理来自于语法分析检查。类似地，第1层处理之间的信息流也通过分析获得。此外，在第0层和第1层之间要保持信息流的连续性。

在DFD第1层中表示的处理可以被进一步精化到更低的层次。例如，监测传感器处理可以被精化为如图8-11所示的第2层DFD。再次提醒大家，在这两层之间保持了信息流的连续性。

DFD的求精持续地进行，直到每个泡泡都执行一个简单的操作，也就是说，直至每个泡泡所代表的处理都执行一个功能，并且该功能可以很容易地被程序实现。在第9章，我们将讨论内聚的概念，这个概念可以用来评估给定功能的简单性。现在，我们只是尽力精化，直到每个泡泡都是简洁的、功能单一的。

¹⁵ “处理叙述”类似于用例的风格，但是目标稍有不同。“处理叙述”提供了将要开发的功能的整体说明，而不是从某个参与者的角度写下的场景。

¹⁶ 注意，同义的或与建模过程没有直接关系的名词或动词被忽略掉。还应注意到，当考虑8.7节中的基于类的建模时，将使用类似的话语分析。

227

228

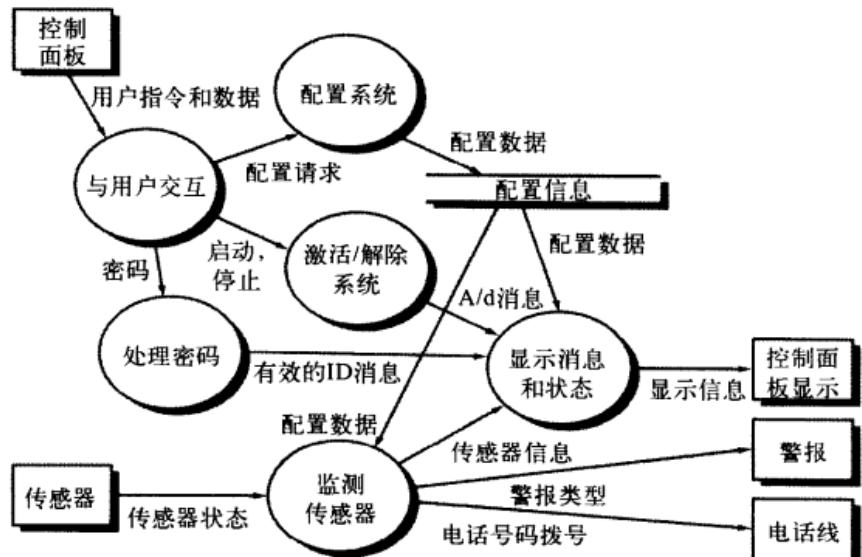


图8-10 SafeHome安全功能的第1层DFD

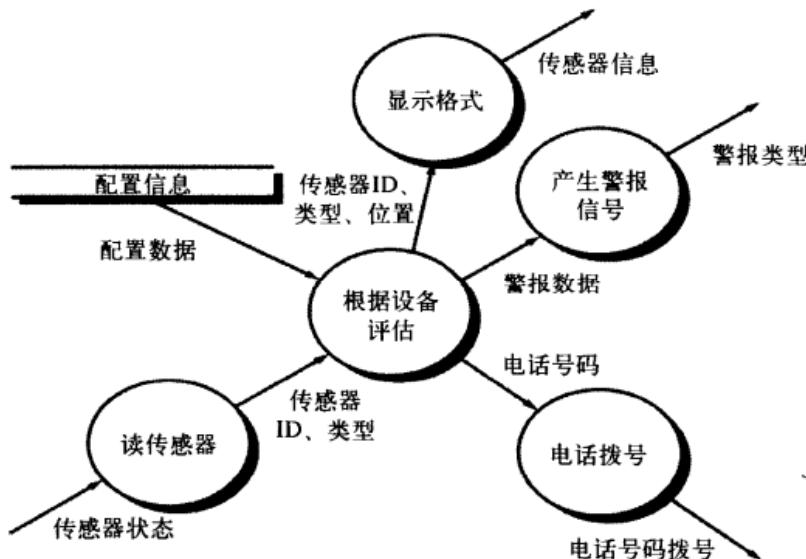


图8-11 精化监测传感器处理的第2层DFD

8.6.2 创建控制流模型

对于很多类应用问题来说，为了获得关于软件需求的有益理解，使用数据模型和数据流图是很有必要的。然而，就像我们已经提到的，有一大类应用问题是事件驱动的而不是数据驱动的；这类问题产生控制信息而不是报告或显示信息，并且，处理信息时非常关注时间和性能。这样的应用除了数据流建模外还需要使用控制流建模。

我们已经提到，事件或控制项可以实现为布尔值（例如，true或false，on或off，1或0）或条件的离散列表（空、拥挤、满）。为了选择潜在的候选事件，建议使用如下的指导原则：

- 列出所有被软件“读”的传感器。
 - 列出所有的中断条件。
 - 列出操作人员能够启动的所有“开关”。
 - 列出所有的数据条件。

- 回顾对处理叙述所进行的名词/动词语法分析，考察所有可能作为控制流输入/输出的“控制项”。
- 通过标识其状态来描述系统的行为，标识这些状态是如何达到的，并定义状态间的迁移。
- 关注可能的疏忽——刻画控制中非常普遍的错误；例如，提问“有什么其他途径可以达到或离开这个状态吗？”

8.6.3 控制规格说明

控制规格说明（Control Specification, CSPEC）使用两种不同的方式表现系统的行为（在被引用的层次上）¹⁷。CSPEC包含一个状态图，该图是行为的序列说明；也可能包括程序激活表——行为的组合说明。

图8-12为SafeHome的第一层控制流模型描述了一个初步的状态图¹⁸，图中显示了当系统在这个层次上定义的四个状态之间来回移动的时候，系统如何对事件作出响应。通过评审状态图，软件工程师可以确定系统的行为，而且更重要的是可以确定被描述的行为中是否存在“空洞”（holes）。



图8-12 SafeHome安全功能的状态图

例如，状态图（图8-12）指明：当系统被重置、激活或断电时，可能会发生Idle（空闲）状态的转换；如果系统被激活（也就是打开报警系统），将会转换到MonitoringSystemStatus状态（监测系统状态），显示信息也将变化，并调用处理monitorAndControlSystem。源自MonitoringSystemStatus状态的转换有两种：（1）当系统撤销激活时，发生回到idle状态的转

¹⁷ 更多的行为建模符号将在本章的后面部分介绍。

¹⁸ 这里使用的状态图符号遵照UML表示法。在结构化分析中“状态迁移图”也是有效的，但是UML格式在信息的内容和表现上更优秀。

换；（2）当传感器被触发时，发生到ActingOnAlarm状态的转换。在评审中将考虑所有的转换和所有状态的内容。

[230] CSPEC描述系统的行为，但是没有提供关于处理的内部工作信息，其实，这些处理作为行为的结果被激活。在8.6.4节中将讨论提供这种信息的建模符号。

SAFEHOME

数据流建模

[场景] Jamie的小房间，在已经召开的上一次需求收集会议之后。

[人物] Jamie、Vinod和Ed，SafeHome软件工程团队的成员。

[对话]

(Jamie已经勾画出图8-9到图8-12所示的模型草图，并向Ed和Vinod展示。)

Jamie：我在学校里上过软件工程的课并学到些东西，教授说这是有点陈旧的方法了，但是你知道吗？这能帮助我阐明事物。

Ed：太酷了。但是我在那里看不到任何类或对象。

Jamie：不……这只是带有一点点行为元素的流模型。

Vinod：也就是说这些DFD表现了软件的I-P-O视图，对吗？

Ed：I-P-O？

Vinod：就是Input-Process-Output（输入-处理-输出），DFD确实非常直观……如果你观察一会，就会看到数据对象如何在系统中流动以及在流动中是如何变化的。

Ed：看起来似乎我们能把每个泡泡转化为一个可执行的构件……至少在最低层的DFD是如此。

Jamie：这就是最酷的部分，你可以这样做。事实上，有一种方法可以把DFD转化为设计架构。

Ed：真的吗？

[231] Jamie：是的，而这是我们开发一个完整的分析模型首先必须做的，不是吗？

Vinod：对的，这是第一步，尽管状态图也能完成同样的工作，但是我们必须开始说明基于类的元素和行为方面。

Ed：我们有那么多的活要做，但是我们却没有那么多的时间。

(Doug——软件工程经理——走进小房间。)

Doug：那么接下来的几天将开发分析模型，有问题吗？

Jamie（骄傲地看着）：我们已经开始了。

Doug：好样的。我们有那么多的活要做，但是我们却没有那么多的时间。

（三个软件工程师互相看了一眼，会心地笑了。）

8.6.4 处理规格说明

处理规格说明（Process Specification, PSPEC）用于描述出现在求精过程中最终层次的所有流模型的处理。处理规格说明的内容可以包括叙述性正文、处理算法的程序设计语言（Program Design Language, PDL）描述¹⁹、数学方程、表、图或图表。通过为流模型中的每个

¹⁹ PDL把编程语言的语法和叙述性正文混合在一起，目的是提供处理的流程设计细节。PDL将在第11章中讨论。



在DFD的最低求精层的每个转换都是一个“小规格说明”。

泡泡提供PSPEC，软件工程师创建了“小规格说明”(mini-spec)，小规格说明可以作为实现处理的软件构件的设计指南。

为了说明如何使用PSPEC，考虑在SafeHome的流模型中表示的“处理密码”转换(图8-10)，该功能的PSPEC可能采用如下形式：

PSPEC：处理密码（在控制面板）。“处理密码”转换完成SafeHome安全功能中控制面板上的密码确认。“处理密码”从“与用户交互”功能接收4位密码，将该密码首先和存储在系统中的主密码进行比较，如果主密码匹配，则向“显示消息和状态”功能传送“valid id message = true”。如果主密码不匹配，则把四位密码和次密码表（这些密码可能会被赋予房子的客人和（或）在房主不在家时需要进入房子的工人）比较。如果密码和表中的某项匹配，将向“显示消息和状态”功能传送“valid id message = true”；如果不匹配，则向“显示消息和状态”功能传送“valid id message = false”。

如果在此阶段需要更多的算法细节，程序设计语言描述也可以作为PSPEC的一部分包含在其中。然而，很多人认为，PDL版本应该推迟到构件设计开始时。

[232]

SOFTWARE TOOLS

结构化分析

目的：结构化分析工具让软件工程师能够以有助于一致性和连续性检查、易于编辑和扩展的方式创建数据模型、流模型和行为模型。使用这些工具创建的模型可以为软件工程师提供对分析模型的理解，并帮助他们在错误扩散到设计甚至更糟的是扩散到实现之前就消除错误。

机制：此类工具使用“数据字典”作为说明所有数据对象的中心数据库。一旦字典里的条目被定义，可以创建实体-关系图并开发对象的多层结构。数据流图的特点有助于容易地创建这个图形化的模型，而且也为创建PSPEC和CSPEC提供一些细节。分析工具还帮助软件工程师使用状态图作为操作表示符号来创建行为模型。

代表性工具²⁰

AxiomSys，由STG，Inc. (www.stgcase.com) 开发，提供完整的结构分析工具套件，包括实时系统建模的Hatley-Pirbhai扩展。

MacA&D，WinA&D，由Excel Software开发 (www.excelsoftware.com)，提供一系列物美价廉的Macs和Windows设备的分析和设计工具。

MetaCASE Workbench，由MetaCase Consulting开发 (www.metacase.com)，用于定义分析或设计方法（包括结构化分析），包括其概念、规则、符号、生成器。

System Architect，由Popkin Software开发 (www.popkin.com)，提供广泛的分析和设计工具，包括数据建模和结构化分析工具。

8.7 基于类的建模

我们如何开发基于类的分析模型的元素——类和对象、属性、操作、包、CRC模型和协

²⁰ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

作图？下面的几小节中将提供一系列有助于识别和表示这些元素的非正式指导原则。

8.7.1 识别分析类

如果环顾房间，就可以发现一组容易识别、分类和定义（就属性和操作而言）的物理对象。但当你“环顾”软件应用的问题空间时，了解类（和对象）就没那么容易了。

“真正困难的问题是首先发现什么才是正确的对象（类）。”

——Carl Argila

[233] 通过检查问题的陈述，或通过对为系统开发的用例或处理叙述进行“语法分析”（使用本章前面早已使用的术语），可以开始类的识别。带有下划线的每个名词或名词词组可以确定为类，并将这些名词输入到一个简单的表中，同义词应被标注出。如果要求某个类实现一个解决方案，那么这个类就是解决方案的一部分；否则，如果某个类只需要说明一个解决方案，那么这个类就是问题空间的一部分。一旦所有的名词都被分离出来，我们该寻找什么？分析类以如下方式之一表现：

分析类如何
把自己表现
为解决方案空间
的元素？

- 外部实体（例如，其他系统、设备、人员），产生或使用基于计算机的系统的信息。
- 事物（例如，报告、显示、字母、信号），问题信息域的一部分。
- 发生或事件（例如，所有权转移或机器人的一组移动完成），在系统操作环境内发生。
- 角色（例如，经理、工程师、销售人员），由和系统交互的人员扮演。
- 组织单元（例如，部门、组、团队），和某个应用相关。
- 场地（例如，制造车间或码头），建立问题的环境和系统的整体功能。
- 结构（例如，传感器、四轮交通工具、计算机），定义了对象的类或与对象相关的类。

这种分类只是文献中已提出的大量分类之一²¹。例如，Budd[BUD96]建议了一种类的分类法，包括数据产生者（源）和数据使用者（汇点）、数据管理者、查看（或观察者）类以及辅助类。

还需要特别注意的是：什么不能是类或对象。通常，决不应该用“强制性的过程的名称”为类命名[CAS89]。例如，如果医疗图像系统的软件开发人员使用名字“**InvertImage**”甚至用“**ImageInversion**”定义对象，就可能犯下一个小小的错误。从软件获得的**Image**当然可能是一个类（这是信息域中的一部分），图像的翻转是适用于该类的一个操作，很可能在**Image**类中定义操作**inversion()**，但是不可能定义单独的类来暗示“图像翻转”。如Cashman[CAS89]所言：“面向对象的目的是包装，但仍保持独立的数据以及对数据的操作。”

[234] 为了说明在建模的早期阶段如何定义分析类，我们返回到SafeHome安全功能。在8.6.1节，我们对安全功能的“处理叙述”²²进行“语法分析”，提取名词，可以获得如下表所示的一些潜在的类：

²¹ 另一个重要的分类——定义实体、边界和控制类——在8.7.4节讨论。

²² 很重要的一点是注意到，该技术也应用于作为需求收集（导出）活动一部分的每个用例。也就是说，用例可以从语法上解析以提取潜在的分析类。

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

这个表应不断完善，直到处理叙述中所有的名词都已经考虑到。注意，我们称列表中的每一项为潜在的对象，在进行最终决定之前还必须对每一项深思熟虑。

“阶级斗争，一些阶级胜利了，一些阶级消灭了……”

——毛泽东

Coad和Yourdon[COA91]建议了6个选择特征，分析师在考虑每个潜在的类是否应该包含在分析模型中时应使用这些特征：

如何确定某个潜在的类是否应该真的成为分析类？

- 保留信息。只有当相关信息必须被记录才能保证系统正常工作时，潜在类在分析过程中才是有用的。
- 所需服务。潜在类必须具有一组可确认的、能用某种方式改变类的属性值的操作。
- 多个属性。在需求分析过程中，焦点应在于“主”信息；事实上，只有一个属性的类可能在设计中有用，但是在分析活动阶段，把它作为另一个类的某个属性可能更好。
- 公共属性。可以为潜在类定义一组属性，这些属性适用于类的所有实例。
- 公共操作。可以为潜在类定义一组操作，这些操作适用于类的所有实例。
- 必要需求。在问题空间中出现的外部实体，或者任何系统解决方案的运行所必需的信息，几乎都被定义为需求模型中的类。

235

考虑包含在需求模型中的合法类，潜在类应全部（或几乎全部）满足这些特征。判定潜在类是否包含在分析模型中多少有点主观，而且后面的评估可能会舍弃或恢复某个类。然而，基于类的建模的首要步骤就是定义类，因此必须进行决策（即使是主观的）。以此为指导，我们对列出的潜在的SafeHome类根据上述选择特征进行筛选，如下表所示：

潜在类	所适用的特征（编号）
房主	拒绝：6适用但是1、2不符合
传感器	接受：所有都适用
控制面板	接受：所有都适用
安装	拒绝
系统（别名安全系统）	接受：所有都适用
编号，类型	拒绝：3不符合，是传感器的属性

(续)

潜在类	所适用的特征(编号)
主密码	拒绝: 3不符合
电话号码	拒绝: 3不符合
传感器事件	接受: 所有都适用
发声警报	接受: 2、3、4、5、6适用
监测服务	拒绝: 6适用但是1、2不符合

应注意到: (1) 上面的表是不全面的——必须添加其他类以使模型更完整; (2) 某些被拒绝的潜在类将成为那些被接受的类的属性(例如, 编号和类型是传感器的属性, 主密码和电话号码可能成为系统的属性); (3) 问题的不同陈述可能导致作出不同的“接受或拒绝”决定(例如, 如果每个房主都有个人密码或通过声音确认, 房主类可能满足特征1和2并有可能被接受)。

8.7.2 描述属性

KEY POINT 属性描述已经选择的、包含在分析模型中的类。实质上, 属性是定义类——明确类在问题空间的环境下意味着什么。

为了给分析类开发一个有意义的属性集合, 软件工程师可以再次研究用例并选择那些合情合理的“属于”类的“东西”。此外, 每个类都应回答如下问题: 什么数据项(组合的和/或基本的)在问题的环境内能够完整地定义这个类?

为了说明这个问题, 考虑为SafeHome定义的**System**(系统)类。我们已经提到过, 房主可以配置安全功能以反映传感器信息、报警响应信息、激活/关闭信息、识别信息等等。我们可以用如下方式表现这些组合数据项:

```
identification information = system ID + verification phone number + system status
alarm response information = delay time + telephone number
activation/deactivation information = master password + number of allowable tries +
temporary password
```

等式右边的某些数据项可以进一步地精化到基础级, 但是考虑到我们的目标, 可以为**System**类组成一个合理的属性列表(图8-13中的阴影部分)。

传感器是整个SafeHome系统的一部分, 但是并没有列为图8-13中的数据项或属性。**Sensor**(传感器)已经被定义为类, 多个**Sensor**对象将和**System**类关联。通常, 如果多个(超过一个)项和类相关联, 就应避免把这个项定义为属性。

8.7.3 定义操作

操作定义了某个对象的行为。尽管存在很多不同类型的操作, 但通常可以粗略地划分为四种类型: (1) 以某种方式操作数据(例如, 添加、删除、重新格式化、选择); (2) 执行计算的操作;

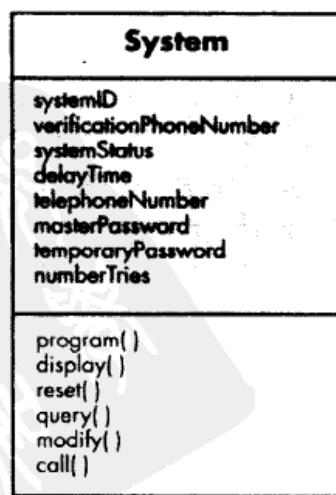


图8-13 **System**类的类图



当为分析类定义操作时，集中于面向问题的行为而不是实现所要求的行为。

(3) 请求某个对象的状态的操作；(4) 监视某个对象发生某个控制事件的操作。这些功能通过在属性和/或相关属性（8.7.5节）上的操作实现。因此，操作必须“理解”类的属性和相关属性的本质。

因为第一次迭代要导出一组分析类的操作，分析师可以再次研究“处理叙述”（或用例）并合理地选择属于该类的操作。为了实现这个目标，可以再次研究语法分析并分离动词，这些动词中的一部分将是合法的操作并能够容易地连接到某个特定类。例如，从本章前面提出的SafeHome处理叙述中可以看到，“传感器被分配编号和类型”、“主密码用于激活和解除系统”，这些短语暗示了一些东西：

- assign()操作和Sensor类相关联。
- program()操作被System类封装。
- arm()和disarm()适用于System类。

在更进一步的调查基础上，program()操作很可能被划分为一些配置系统所需要的更具体的子操作。例如，program()含着电话号码、配置系统特性（例如，创建传感器表、输入报警特征值）、输入密码。但是我们暂时把program()指定为一个单独的操作。

SAFEHOME

类模型

[场景] Ed的小房间，开始分析建模。

[人物] Jamie、Vinod和Ed，SafeHome软件工程团队的成员。

[对话]

（对于访问摄像头监视设备——显示摄像头视图（ACS-DCV）（本章8.5.1节已有介绍），Ed已经在从用例模板中提取类方面做了一些工作，并向他的同事展示已经提取的类。）

Ed: 那么当房主希望选择一个摄像头的时候，他/她可能从一个平面设计图中选择。我已经定义了一个FloorPlan类，这是图。

（他们查看图8-14。）

Jamie: 那么FloorPlan这个类把墙（包括墙段、门窗）以及摄像头都组织在一起，这就是那些带标记的线条的意义，是吗？

Ed: 是的，它们被称作“关联”，一个类根据已知的关联和另一个类联系（在8.7.5节中讨论关联）。

Vinod: 那么实际的平面设计图是由墙构成的，并包含摄像头和放置在那些墙中的传感器。平面设计图如何知道在哪里放置那些对象？

Ed: 这个不能，但是其他类知道。察看属性WallSegment，该属性用于构建墙，墙段具有起点坐标和终点坐标，draw()操作完成其他的。

Jamie: 这些也适用于门窗。看起来摄像头有一些额外的属性。

Ed: 是的，它们需要提供转动信息和缩放信息。

Vinod: 我有个问题，为什么摄像头有ID而其他的没有？

Ed: 显示功能要求识别每个摄像头。

Jamie: 说的有道理，但是我的确还有些其他问题。

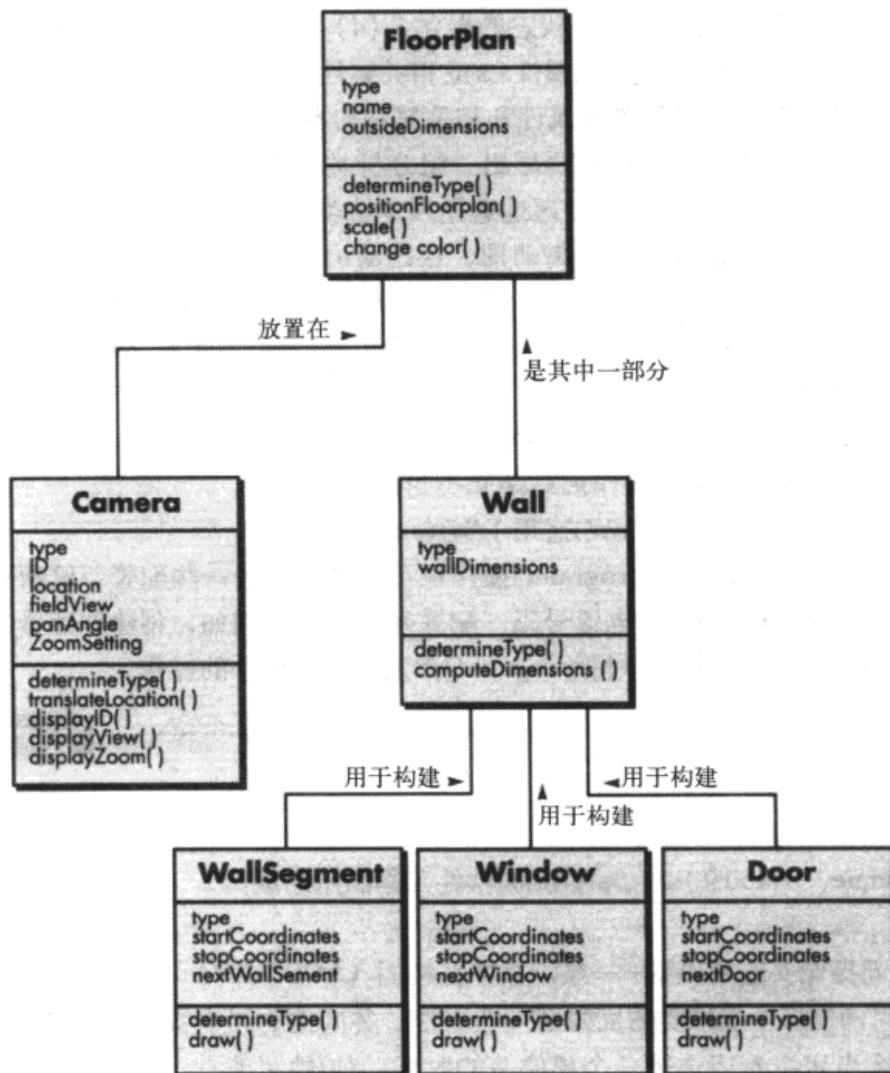


图8-14 FloorPlan类的类图

(Jamie问了一些问题，因此有一些小的修改。)

Vinod: 你有每个类的CRC卡吗？如果有，我们应该进行角色演练，以确保没有任何遗漏。

Ed: 我不太清楚如何做。

Vinod: 这不难，而且确实有用，我给你演示一下。

239

8.7.4 CRC建模

CRC (Class-Responsibility-Collaborator, 类-职责-协作者) 建模[WIR90]提供了一个简单方法，可以识别和组织与系统或产品需求相关的类。Ambler[AMB95]用如下文字解释了CRC建模：

CRC模型实际上是表示类的标准索引卡片的集合。这些卡片被分为三部分，顶部写类名，下面左侧部分列出类的职责，右侧部分列出类的协作关系。

事实上，CRC模型可以使用真的或虚拟的索引卡，目的是开发类的有组织的表示。职责

是和类相关的属性和操作，简单地说，职责就是“类所知道或能做的任何事” [AMB95]。协作者是那些提供完成某个职责所需要信息的类。通常，协作意味着信息请求或动作请求。

“CRC卡的一个作用是避免早期失败、频繁失败，即使失败也花费不多，整理一堆卡片要比重新组织大量源代码代价小得多。” ——C. Horstmann

—C. Horstmann

FloorPlan类的一个简单CRC索引卡如图8-15所示。CRC卡上所列出的职责只是初步的，可以添加或修改。在职责栏右边的**Wall**和**Camera**是需要协作的类。

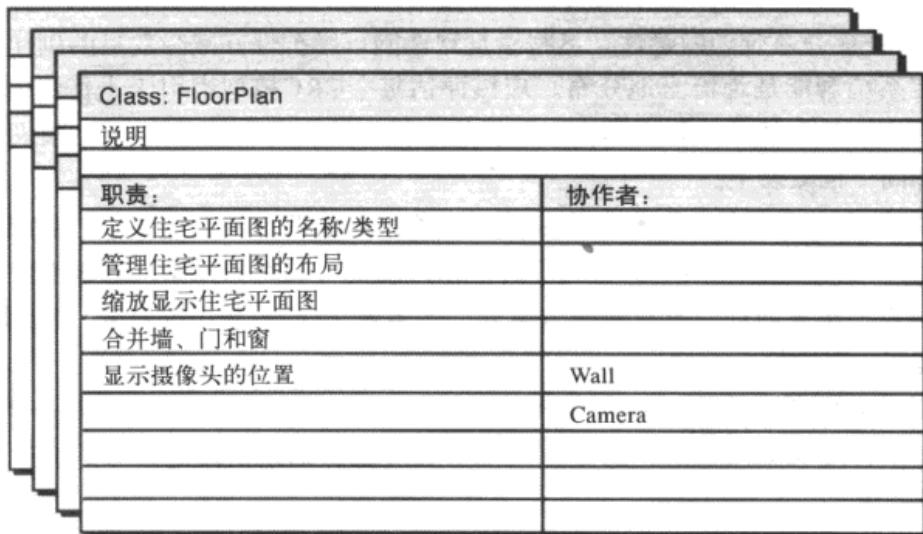


图8-15 CRC模型索引卡

240

类。在本章前面已经说明识别类和对象的基本原则。8.7.1节所说的类的分类方法可以通过如下分类扩展：

WebRef

关于这些类的类型的精彩讨论，可以参阅 www.theumlcafe.com/a0079.htm

- 实体类是从问题的说明中直接提取出来的（例如FloorPlan和Sensor），也被称作模型或业务类。这些类一般代表保存在数据库中的和贯穿应用程序（除非被明确删除）的事物。
 - 边界类用于创建用户可见的和交互的接口（例如交互屏幕或打印的报表）。实体类包含对用户来说很重要的信息，但是并不显示这些信息。边界类被设计成负责管理实体对象对用户的表示方式。例如，一个被称作**CameraWindow**的边界类可能负责显示SafeHome系统的监视摄像头输出。
 - 控制类自始至终管理“工作单元”[UML03]。也就是说，控制类可以管理（1）实体类的创建或更新；（2）当边界类从实体对象获取信息后的实例化；（3）对象集之间的复杂通信；（4）确认对象间交换的数据或用户和应用程序间交换的数据。通常，直到设计开始时才开始考虑控制类。

“对象可以科学地分为三个主要类别：不工作的、损坏的和捣乱的。”

—Russell Baker

职责。在8.7.2节和8.7.3节中已经说明了识别职责（属性和操作）的基本原则。Wirfs-Brock和她的同事[WIR90]在给类分配职责时建议了以下五个指导原则：

给类分配职责时可以采用什么指导原则?

241

1. 系统智能应分布在所有类中以求最佳地解决问题的需求。每个应用程序都包含一定程度的智能，也就是系统所知道的以及所能完成的。智能在类中可以有多种分布方式。“哑 (dumb)”类（那些几乎没有职责的类）可以被建模为一些“灵巧 (smart)”类（那些有很多职责的类）的仆从。尽管该方法使得系统中的控制流简单易懂，但同时有如下缺点：(a) 把所有的智能集中在少数类，使得变更更为困难；(b) 将会要求更多的类，因此需要更多的开发工作。

如果系统智能更平均地分布在应用程序的类中，每个对象只了解和执行一些事情（通常是适度集中），并提高系统的内聚性，这将增加软件的可维护性并减少变更的副作用影响。

为了确定系统智能是否恰当地分布，应该评估每个CRC模型索引卡上标记的职责，以确定某个类是否应该具有超长的职责列表，如果是这样，就表明智能太集中²³。此外，每个类的职责应表现在同一抽象层上。

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

3. 信息和与之相关的行为应在同一个类中。这实现了OO思想中的封装，数据和操作数据的处理应包装在一个内聚单元中。

4. 某个东西的信息应局限于一个类中而不要分布在多个类中。通常应由一个单独的类负责保存和操作某特定类型的信息，而不应由多个类负责。如果信息是分布的，软件将变得更加难以维护和测试。

5. 必要时，职责应由相关类共享。很多情况下，各种相关对象必须在同一时刻展示同样的行为。例如，考虑一个视频游戏，必须显示如下类：**Player**, **PlayerBody**, **PlayerArms**, **PlayerLegs**, **PlayerHead**。每个类都有各自的属性（例如，**position**, **orientation**, **color**, **speed**）并且所有这些都必须在用户操纵游戏杆时更新和显示。因此，职责**update()**和**display()**必须被每个对象共享。**Player**知道在什么时候某些东西变化了并且需要**update()**操作，它和其他对象协作以刷新到新的位置或方向，但是每个对象控制各自的显示。

242 协作。类有一种或两种方法实现其职责：(1) 类可以使用其自身的操作控制各自的属性，从而实现特定的职责；(2) 类可以和其他类协作。

Wirfs-Brock和她的同事[WIR90]定义协作如下：

协作从客户职责实现的角度表现从客户到服务器的请求。协作是客户和服务器之间契约的具体实现……如果我们说某个对象和其他对象协作以实现某个职责，就需要向其他对象发送消息。单独的协作是单向流——表示从客户到服务器的请求。从客户的角度看，每个协作都和服务器的某个职责实现相关。

KEY POINT
如果一个类不能实现其所有的职责，那么就需要一个协作。

协作识别类之间的关系。当一组类相互协作以实现某个需求时，这些类可以组织为子系统（一个设计问题）。

通过确认类本身是否能够实现每个职责，可以识别协作。如果不能实现每个职责，那么需要和其他类交互，因此就存在协作。

例如，考虑SafeHome的安全功能。作为活动流程的一部分，Control-

²³ 在这种情况下，可能需要将一个类分成多个类或子系统，以便更有效地分布智能。

Panel对象必须确定是否启动所有的传感器，定义名为determine-sensor-status()的职责。如果传感器是启动的，**ControlPanel**必须设置属性status为“not ready”。传感器信息可以从每个**Sensor**对象获取，因此只有当**ControlPanel**和**Sensor**协作时才能实现determine-sensor-status()职责。

为识别协作者，分析师可以检查类之间三种不同的通用联系[WIR90]：(1) *is-part-of*（是……一部分）联系；(2) *has-knowledge-of*（有……的知识）联系；(3) *depends-upon*（依赖……）联系。在下面的段落中将简单地分别说明这三种通用联系。

属于某个聚合类一部分的所有类通过*is-part-of*联系和聚合类连接。考虑前面提到的视频游戏中所定义的类，**PlayerBody** *is-part-of* **Player**，**PlayerArms**、**PlayerLegs**和**PlayerHead**也类似。在UML中，这些联系使用聚合表示，如图8-16所示。

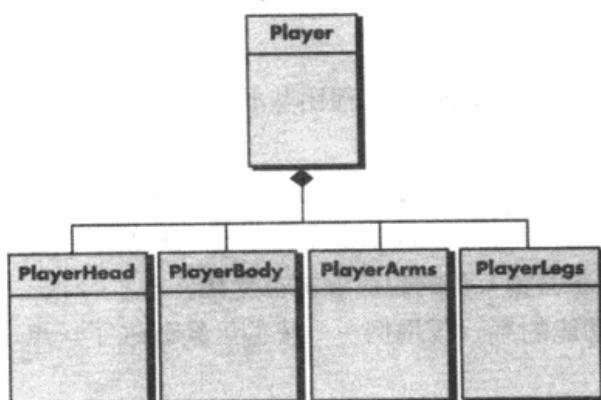


图8-16 聚合类

当一个类必须从另一个类中获取信息时，就建立了*has-knowledge-of*关联。前面所说的determine-sensor-status()职责就是*has-knowledge-of*联系的一个例子。

*depends-upon*关联意味着两个类之间具有*has-knowledge-of*和*is-part-of*不能实现的依赖关系。例如，**PlayerHead**通常必须连接到**PlayerBody**（除非视频游戏特别暴力），然而每个对象并没有其他对象的直接信息。**PlayerHead**对象的center-position属性由**PlayerBody**的中心位置确定，该信息通过第三方对象**Player**获得。因此，**PlayerHead depends-upon PlayerBody**。

所有情况下，协作类的名称记录在CRC模型索引卡上，紧靠在职责旁边。因此，索引卡包含一个职责列表以及相关的能够实现这些职责的协作（图8-15）。

当一个完整的CRC模型被开发出来时，客户代表和软件工程组织的代表可以使用如下方法评审模型[AMB95]：

1. 所有参加CRC模型评审的人员拿到一部分CRC模型索引卡，卡片按照协作分开（也就是说每个评审员不得有两张存在协作关系的卡片）。
2. 所有的用例场景（以及相关的用例图）将被分类管理。
3. 评审组长细致地阅读用例。当评审组长看到一个已命名的类时，给拥有相应类索引卡的人员一个令牌。例如，SafeHome的一个用例包含如下描述：

房主观察SafeHome控制面板以确定系统是否已经准备接收输入。如果系统没有准备好，房主必须手工关闭窗户（门）以使得系统指示器就绪。（未就绪指示器意味着某个传感器是开启的，也就是说某个门或窗户是打开的。）

situationPanel——在按钮上提供信息，用户可以使用这些信息选择环境。

FloorPlan——类似于监视对象，但这个用来显示设备。

deviceIcons——图标上的信息，代表灯、家用设备、HVAC等等。

devicePanels——模拟家用设备或设备控制面板，允许控制。

操作：

displayControl(), selectControl(), displaySituation(), selectSituation(), accessFloorplan(),
selectDeviceIcon(), displayDevicePanel(), accessDevicePanel()……

类：HomeManagementInterface

职责

协作者

displayControl	OptionsPanel (类)
selectControl	OptionsPanel (类)
displaySituation	SituationPanel (类)
selectSituation	SituationPanel (类)
accessFloorplan	FloorPlan (类)
.....

Ed：这样，当调用accessFloorPlan()操作，将和**FloorPlan**对象协作，这个对象类似我们为监视所开发的。等一下，我这里有它的说明（他们查看图8-14）。

Vinod：确实如此。如果我们希望评审整个类模型，我们可以从这个索引卡开始，然后到协作者的索引卡，再到协作者的协作者的索引卡，依此类推。

Ed：这真是个发现遗漏和错误的好方法。

Vinod：的确如此。

8.7.5 关联和依赖



关联定义了类之间的联系，多样性定义了一个类和另一个类之间的联系数量关系。

在很多例子中，两个分析类以某种方式相互关联，如两个数据对象可能相互关联（8.3.3节）。在UML中，这些联系被称作关联（associations）。参考图8-14，通过识别**FloorPlan**和两个其他类（**Camera**和**Wall**）之间的一组联系确定类**FloorPlan**。类**Wall**和三个构成墙的类（**WallSegment**, **Window**和**Door**）相关联。

在某些情况下，关联可以更进一步地指出多样性（术语“基数”已经在本章前面部分使用过）。参考图8-14，一个**Wall**对象可以由一个或多个**WallSegment**对象构成。此外，**Wall**对象可能包含0或多个**Window**对象以及0或多个**Door**对象。这些多样性限制如图8-17所示，其中“一个或多个”使用 $1..*$ 表示，“0或多个”使用 $0..*$ 表示。在UML中，星号表示范围无上界²⁴。

在很多情况下，两个分析类之间存在客户—服务器联系。此时，客户类在某些方面依赖于服务器类并且建立了依赖联系。依赖由一个构造型（stereotype）定义。在UML中，构造型是一个“可扩展机制”[ARL02]，允许软件工程师定义特殊的建模元素，这些建模元素的语义是由用户自定义的。在UML中，构造型由一对尖括号表示（如《stereotype》）。

²⁴ 其他的多样性关系，比如一对一、多对多、一对某个范围（指定上下限）以及其他，可以看作关联的一部分。

situationPanel——在按钮上提供信息，用户可以使用这些信息选择环境。

FloorPlan——类似于监视对象，但这个用来显示设备。

deviceIcons——图标上的信息，代表灯、家用设备、HVAC等等。

devicePanels——模拟家用设备或设备控制面板，允许控制。

操作：

displayControl(), selectControl(), displaySituation(), selectSituation(), accessFloorplan(),
selectDeviceIcon(), displayDevicePanel(), accessDevicePanel()……

类：HomeManagementInterface

职责

协作者

displayControl	OptionsPanel (类)
selectControl	OptionsPanel (类)
displaySituation	SituationPanel (类)
selectSituation	SituationPanel (类)
accessFloorplan	FloorPlan (类)
.....

Ed：这样，当调用accessFloorPlan()操作，将和**FloorPlan**对象协作，这个对象类似我们为监视所开发的。等一下，我这里有它的说明（他们查看图8-14）。

Vinod：确实如此。如果我们希望评审整个类模型，我们可以从这个索引卡开始，然后到协作者的索引卡，再到协作者的协作者的索引卡，依此类推。

Ed：这真是个发现遗漏和错误的好方法。

Vinod：的确如此。

8.7.5 关联和依赖



关联定义了类之间的联系，多样性定义了一个类和另一个类之间的联系数量关系。

在很多例子中，两个分析类以某种方式相互关联，如两个数据对象可能相互关联（8.3.3节）。在UML中，这些联系被称作关联（associations）。参考图8-14，通过识别**FloorPlan**和两个其他类（**Camera**和**Wall**）之间的一组联系确定类**FloorPlan**。类**Wall**和三个构成墙的类（**WallSegment**, **Window**和**Door**）相关联。

在某些情况下，关联可以更进一步地指出多样性（术语“基数”已经在本章前面部分使用过）。参考图8-14，一个**Wall**对象可以由一个或多个**WallSegment**对象构成。此外，**Wall**对象可能包含0或多个**Window**对象以及0或多个**Door**对象。这些多样性限制如图8-17所示，其中“一个或多个”使用 $1..*$ 表示，“0或多个”使用 $0..*$ 表示。在UML中，星号表示范围无上界²⁴。

在很多情况下，两个分析类之间存在客户—服务器联系。此时，客户类在某些方面依赖于服务器类并且建立了依赖联系。依赖由一个构造型（stereotype）定义。在UML中，构造型是一个“可扩展机制”[ARL02]，允许软件工程师定义特殊的建模元素，这些建模元素的语义是由用户自定义的。在UML中，构造型由一对尖括号表示（如《stereotype》）。

²⁴ 其他的多样性关系，比如一对一、多对多、一对某个范围（指定上下限）以及其他，可以看作关联的一部分。

下面举例说明SafeHome监视系统中的简单依赖。一个**Camera**对象（本例中的服务器类）向一个**DisplayWindow**对象（本例中的客户类）提供视频图像。这两个对象之间的联系不是简单的联系，而是存在依赖联系。在监视用例（没有列出来）中，建模者知道必须提供特殊的密码才能查看指定摄像头的位置。为实现这一点，一种方法是让**Camera**请求密码，然后在获得**DisplayWindow**的允许后显示视频。这可以由图8-18表示，其中《access》意味着通过密码控制使用摄像头的输出。

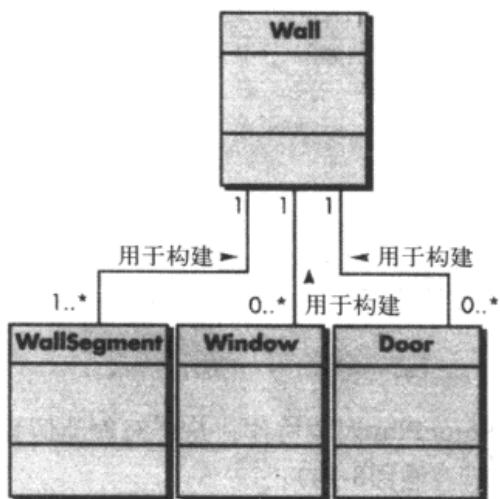


图8-17 多样性

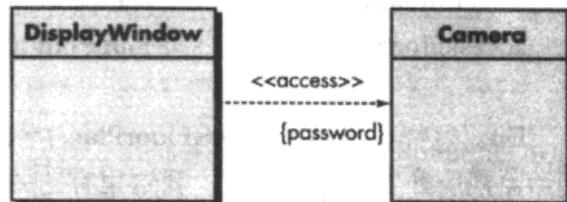


图8-18 依赖

8.7.6 分析包

分析建模的一个重要部分是分类，也就是将分析模型的各种元素（如用例、分析类）分组打包——称作分析包，并为其取一个有代表性的名称。

POINT 分析包用来组合一组相关的类。为了说明分析包的使用，考虑我们前面所说的视频游戏。假设视频游戏的分析模型已经建成，并且已经生成大量的类。一些类关注于游戏环境——用户游戏时所看到的可视场景。诸如**Tree**、**Landscape**、**Road**、**Wall**、**Bridge**、**Building**、**VisualEffect**类等可能就属于游戏环境类。另一些类关注于游戏内的特点，说明游戏的物理特点、动作和限制。也可能定义了**Player**（前面提到的）、**Protagonist**、**Antagonist**、**SupportingRoles**类。还需要一些类来说明游戏的规则——游戏者如何穿越环境，如在这里可以选择**Rules of Movement**类和**Constraints-OnAction**类。还可能存在很多其他类，这些类可以由图8-19中的分析包表示。

每个包中分析类名字前的加号表示该类是公共可见的，因此可以从其他包访问。包中的任何元素之前可以添加其他符号——尽管目前的图中没有显示。负号表示该元素对其他包是隐藏的，#号表示该元素只能由指定包中的类访问。

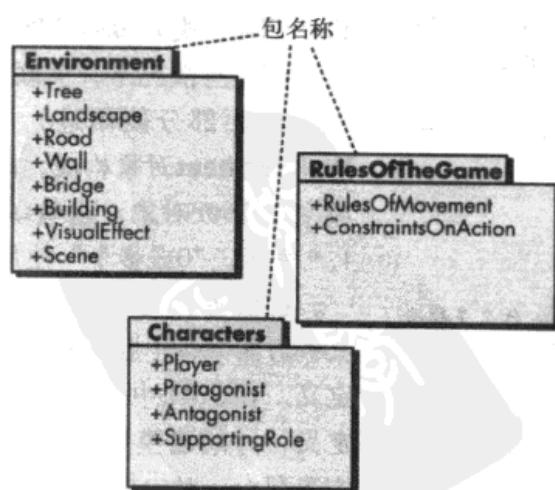


图8-19 包

8.8 生成行为模型

如何建模说明软件对某些外部事件的响应？

8.7节讨论的类图、CRC索引卡和其他面向对象模型表现了分析模型中的静态元素，现在是转变到系统或产品的动态行为的时候了。要实现这一点，我们必须把系统的行为表现为特定事件和时间的函数。

248

行为模型显示了软件如何对外部事件或激励作出响应。要生成模型，分析师必须按如下步骤进行：

 **POINT**
要想通过解析用例来定义事件，可以通过从信息交换的角度检查用例来达到。

1. 评估所有的用例，以保证完全理解系统内的交互序列。
2. 识别驱动交互序列的事件，并理解这些事件如何和具体的类相互关联。
3. 为每个用例生成序列。
4. 创建系统状态图。
5. 评审行为模型以验证准确性和一致性。

在下面的几小节中将讨论每个步骤。

8.8.1 识别用例事件

就像我们在8.5节中提到的，用例表现了涉及的参与者和系统的活动顺序。一般而言，只要系统和参与者之间交换了信息就发生事件。回忆我们前面在8.6.3节中讨论的行为建模，应该注意到事件不是被交换的信息，而是信息已被交换的事实。

为了说明如何从信息交换的角度检查用例，考虑SafeHome安全功能中的一小部分用例。

房主使用键盘键入4位密码，该密码和保存在系统中的验证密码相比较。如果密码不正确，控制面板将鸣叫一声并复位以等待下一次输入；如果密码正确，控制面板等待进一步的操作。

用例场景中加下划线的部分表示事件。应确认每个事件的参与者，应标记交换的所有信息，而且应列出每个条件或限制。

举个典型事件的例子，考虑用例中加下划线的“房主使用键盘键入4位密码”。在分析模型的环境下，对象**Homeowner**²⁵向对象**ControlPanel**发送一个事件，这个事件可以称作“输入密码”。传输的信息是组成密码的4位数字，但这不是行为模型的本质部分。重要的是注意到某些事件对用例的控制流有明显的影响，而其他的事件对控制流没有直接的影响。例如，事件“输入密码”不会显式地改变用例的控制流，但是事件“比较密码”（从与事件“该密码和保存在系统中的验证密码相比较”的交互中得到）的结果将直接影响到SafeHome软件的信息流和控制流。

249

一旦确定了所有的事件，这些事件将被分配到所涉及的对象，对象负责生成事件（例如，**Homeowner**生成“输入密码”事件）或识别已经在其他地方发生的事件（例如，**ControlPanel**识别“比较密码”事件的二元结果）。

8.8.2 状态表现

在行为建模的场合下，必须考虑两种不同的状态描述：（1）系统执行其功能时每个类的状态；（2）系统执行其功能时从外部观察到的系统状态²⁶。

²⁵ 在这个例子中，我们假设和SafeHome交互的每个用户（房主）都有正确的密码，因此是合法的对象。

²⁶ 8.6.3节介绍的状态图描述了系统的状态。本节我们的讨论将集中在分析模型中的每个类的状态。

KEY POINT

系统状态可以表现特定的外部可观察的行为，类的状态可以表现当系统执行功能时的行为。

类状态具有被动和主动两种特征[CHA93]。被动状态很简单，是某个对象所有属性的当前状态。例如，类Player（在前面讨论的视频游戏应用程序中）将包含当前的position和orientation属性，以及和游戏相关的Player的其他特性（例如，用来显示magic wishes remaining的属性）。一个对象的主动状态指的是对象进行持续变换和处理时的当前状态。类Player可能具有如下的主动状态：移动、休息、受伤、疗伤、被捕、失踪等等。必须发生事件（有时被称为触发器）才能迫使对象做出从一个活动状态到另一个活动状态的转移。

下面将讨论两种不同的行为表现形式。第一种显示独立的类如何基于外部事件变化状态，第二种以时间函数的形式显示软件的行为。

分析类的状态图。行为模型的组成之一是UML状态图，UML状态图为每个类表现活动状态和导致这些活动状态变化的事件（触发器）。图8-20举例说明了SafeHome安全功能中ControlPanel类的状态图。

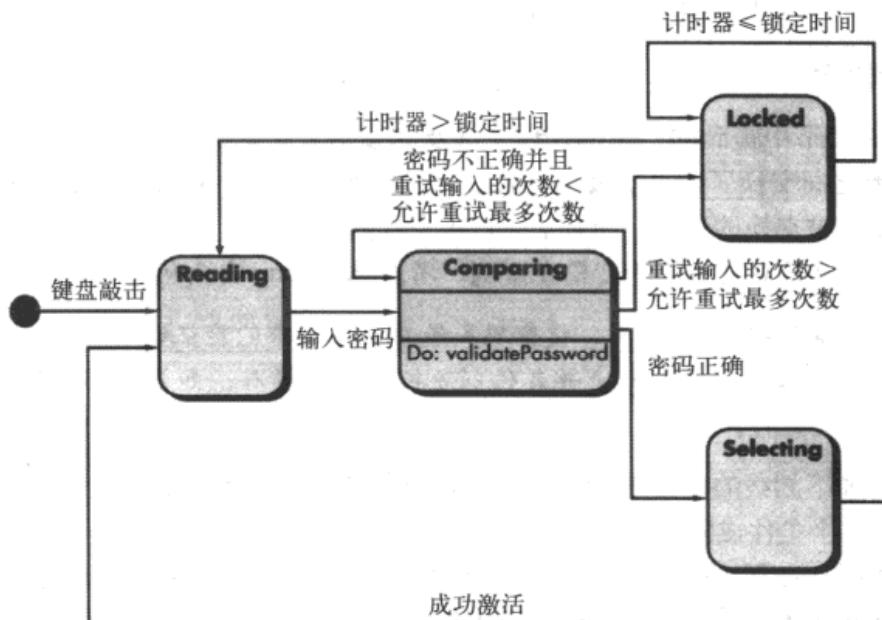


图8-20 ControlPanel类的状态图

图8-20中显示的每个箭头都表示某个类从一个活动状态转移到另一个活动状态。每个箭头上的标注都表现了触发状态转移的事件。尽管活动状态模型在提供类的“生命历史”信息方面非常有用，但也能说明可加深理解类的行为的补充信息。除了说明导致转换发生的事件外，分析师还可以说明保护和动作[CHA93]。保护是为了保证转变发生而必须满足的一个布尔条件。例如，图8-20中从“Reading”状态到“Comparing”状态的转移可以由检查用例来确定：

if (password input = 4 digits) then compare to stored password

一般而言，转移的保护通常依赖于某个对象的一个或多个属性值。换句话说，保护依赖于对象的被动状态。

动作和状态转移同时发生或作为状态转移的结果，而且通常动作包含对象的一个或多个操作（职责）。例如，和“输入密码”事件（图8-20）相关联的动作是validatePassword()操作，

该操作访问password对象并通过执行按位比较来验证输入的密码。



与不注明相关类表现行为的状态图不同，顺序图通过说明类如何从一个状态转移到另一个状态来表现行为。

顺序图。第二种表现行为的方式在UML中称作顺序图，说明事件如何引发从一个对象到另一个对象的转移。一旦通过检查用例确认了事件，建模人员就创建了一个顺序图——用时间函数表现事件如何引发流从一个对象到另一个对象。事实上，顺序图是用例的速记版本。它表现了导致行为从一个类流到另一个类的关键类和事件。

图8-21列举了SafeHome安全功能的部分顺序图。每个箭头代表了一个事件（源自用例）并说明了事件如何引导SafeHome对象之间的行为。时间纵向（向下）度量，窄的纵向矩形表示处理某个活动所用的时间。沿着纵向的时间线可以展示出对象的状态。

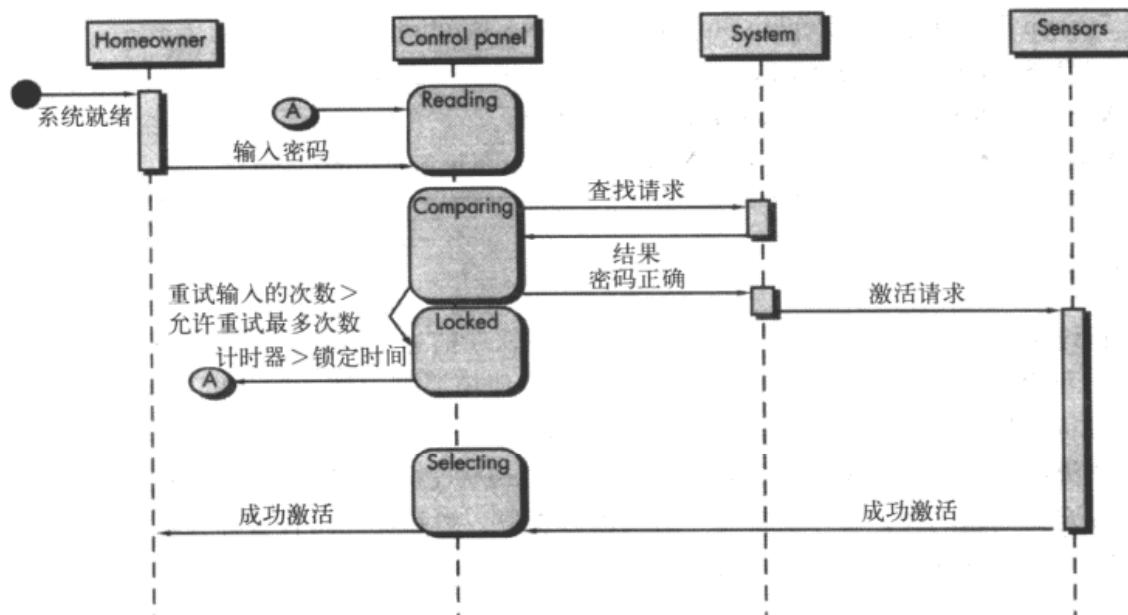


图8-21 SafeHome安全功能的顺序图（部分）

第一个事件“系统就绪”来自于外部环境并且把行为引导到对象Homeowner。房主输入一个密码，“查找请求”事件被发送到System，System在一个简单的数据库中查找密码并向ControlPanel（现在处在Comparing状态）返回结果（找到或没有找到）。有效的密码将促使系统产生“密码正确”事件，该事件通过“激活请求”事件激活传感器。最终，控制通过事件“成功激活”返回到房主。

一旦完成了完整的顺序图，所有导致系统对象之间转移的事件都可以被整理为输入事件集合和输出事件集合（从一个对象）。对于将要构建的系统而言，这些信息对于创建有效的设计非常有用。

SOFTWARE TOOLS

UML中的通用分析建模

目的：分析建模工具可以使用UML标记语言开发基于场景的模型、基于类的模型和行为模型。

机制：支持构建分析模型所需的全部UML图（这些工具也支持设计建模）。除了制图，工具还要求：（1）为所有的UML图执行一致性和正确性检查；（2）为设计和代码生成提

供链接；（3）构建数据库，以实现管理和评估复杂系统所需的大型UML模型。

代表性工具²⁷

如下工具支持分析建模所需的全部UML图：

ArgoUML，开源工具 (argouml.tigris.org)。

Control Center，由TogetherSoft开发 (www.togethersoft.com)。

Enterprise Architect，由Sparx Systems开发 (www.sparxsystems.com.cn)。

Object Technology Workbench (OTW)，由OTW Software开发 (www.otwsoftware.com)。

PowerDesigner，由Sybase开发 (www.sybase.com)。

Rational Rose，由Rational Corporation开发 (www.rational.com)。

System Architect，由Popkin Software开发 (www.rational.com)。

UML Studio，由Pragsoft Corporation开发 (www.pragsoft.com)。

Visio，由Microsoft开发 (www.microsoft.com)。

Visual UML，由Visual Object Modelers开发 (www.visualuml.com)。

8.9 小结

分析建模的目标是创建各种表现形式，以描述软件信息、功能和行为的需求。为实现这一目标，可以采用两种不同的（但潜在是互补的）建模原则：结构化分析和面向对象分析。结构化分析把软件看作信息转换器，辅助软件工程师识别数据对象、对象间的联系以及这些数据对象流经软件处理功能时转换的方式。面向对象分析就是检查定义为一组用例的问题域，尽量提取定义问题的类。每个类都有一组属性和操作，类和类之间有多种关联方式，并使用UML图建模。分析模型由四种建模元素构成：基于场景的模型、流模型、基于类的模型和行为模型。

基于场景的模型从用户的角度描述软件需求。用例——参与者和软件之间的某个交互的叙述或模板驱动说明——是主要的建模元素。在需求获取过程中提取的用例，定义了特定功能或交互的关键步骤。用例的形式化和详细程度各不相同，但是最终结果为所有的其他分析建模活动提供了必需的输入。还可以使用活动图说明场景——一种类似于流程图的图形表现形式，描述在特定场景中的处理流。泳道图显示了处理流如何分配给不同的用户或类。

流模型关注当数据对象通过处理函数转换时的流动。在结构化分析的过程中提取的流模型使用数据流图，即一种建模符号，描述当数据对象在系统中移动时输入如何转变成输出。使用处理规格说明（或叙述）来说明转换数据的每一个软件功能。除了数据流，该建模元素还说明了控制流，这是显示事件如何影响系统行为的一种表现方式。

基于类的建模使用从基于场景和面向流的建模元素中提取的信息确定分析类。可以使用语法分析从文本叙述中提取候选类、属性和操作，并制定了用于定义类的标准。CRC索引卡可以用于定义类之间的联系。此外，可以使用各种UML建模符号定义类之间的层次、联系、关联、聚合和依赖。使用分析包可以将类分类和分组，在某种意义上为大型系统提供了更好

²⁷ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

的管理。

前面三种分析建模元素提供了软件的静态视图，而行为建模描述了动态行为。行为模型使用基于场景、面向流和基于类的元素作为输入，从整体上表现分析类和系统的状态。要实现这一点，要识别状态，定义导致类（或系统）做出状态转移的事件，以及确认当转移完成时所发生的行为。状态图和顺序图是用于行为建模的UML表达方式。

参考文献

- [ABB83] Abbott, R., "Program Design by Informal English Descriptions," *CACM*, vol. 26, no. 11, November 1983, pp. 892-894.
- [AMB95] Ambler, S., "Using Use-Cases," *Software Development*, July 1995, pp. 53-61.
- [ARA89] Arango, G., and R. Prieto-Diaz, "Domain Analysis: Concepts and Research Directions," *Domain Analysis: Acquisition of Reusable Information for Software Construction*, (Arango, G. and R. Prieto-Diaz, eds.), IEEE Computer Society Press, 1989.
- [ARL02] Arlow, J., and I. Neustadt, *UML and the Unified Process*, Addison-Wesley, 2002.
- [BER93] Berard, E. V., *Essays on Object-Oriented Software Engineering*, Addison-Wesley, 1993.
- [BOO86] Booch, G., "Object-Oriented Development," *IEEE Trans. Software Engineering*, vol. SE-12, no. 2, February 1986, pp. 211ff.
- [BUD96] Budd, T., *An Introduction to Object-Oriented Programming*, 2nd ed., Addison-Wesley, 1996.
- [CAS89] Cashman, M., "Object-Oriented Domain Analysis," *ACM Software Engineering Notes*, vol. 14, no. 6, October 1989, p. 67.
- [CHA93] de Champeaux, D., D. Lea, and P. Faure, *Object-Oriented System Development*, Addison-Wesley, 1993.
- [CHE77] Chen, P., *The Entity-Relationship Approach to Logical Database Design*, QED Information Systems, 1977.
- [COA91] Coad, P., and E. Yourdon, *Object-Oriented Analysis*, 2nd ed., Prentice-Hall, 1991.
- [COC01] Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, 2001.
- [DAV93] Davis, A., *Software Requirements: Objects, Functions and States*, Prentice-Hall, 1993.
- [DEM79] DeMarco, T., *Structured Analysis and System Specification*, Prentice-Hall, 1979.
- [FIR93] Firesmith, D. G., *Object-Oriented Requirements Analysis and Logical Design*, Wiley, 1993.
- [LET03] Lethbridge, T., personal communication on domain analysis, May, 2003.
- [OMG03] Object Management Group, OMG Unified Modeling Language Specification, version 1.5, March 2003, available from <http://www.rational.com/uml/resources/documentation/>.
- [SCH02] Schmuller, J., *Teach Yourself UML in 24 Hours*, 2nd ed., SAMS Publishing, 2002.
- [SCH98] Schneider, G., and J. Winters, *Applying Use Cases*, Addison-Wesley, 1998.
- [STR88] Stroustrup, B., "What Is Object-Oriented Programming?" *IEEE Software*, vol. 5, no. 3, May 1988, pp. 10-20.
- [TAY90] Taylor, D. A., *Object-Oriented Technology: A Manager's Guide*, Addison-Wesley, 1990.
- [THA00] Thalheim, B., *Entity Relationship Modeling*, Springer-Verlag, 2000.
- [TIL93] Tillmann, G., *A Practical Guide to Logical Data Modeling*, McGraw-Hill, 1993.
- [UML03] The UML Café, "Customers Don't Print Themselves," available at <http://www.theumlcafe.com/a0079.htm>, May, 2003.
- [WIR90] Wirfs-Brock, R., B. Wilkerson, and L. Weiner, *Designing Object-Oriented Software*, Prentice-Hall, 1990.

254

习题与思考题

- 8.1 简单用几句话尝试说明结构化分析和面向对象分析的主要差别。
- 8.2 有没有可能不完成如图8-3所示的所有四种元素就开发出一个有效的分析模型？解释一下。
- 8.3 有没有可能在分析模型创建后立即开始编码？解释你的答案，然后说服反方。
- 8.4 域分析的目的是什么？域分析如何与需求模式概念相联系？
- 8.5 一个单凭经验的分析原则是：模型“应该关注于那些在问题域或业务域内可见的需求”。

在这些域中哪些类型的需求是不可见的？提供一些例子。

8.6 构建如下系统中的一个：

- a. 你所在大学的基于网络的课程注册系统。
- b. 一个计算机商店的基于Web的订单处理系统。
- c. 一个小企业的简单发票系统。
- d. 取代Rolodex并安装在无线电话上的软件。
- e. 内置于电磁灶或微波炉的自动食谱。

选择你感兴趣的系统并说明数据对象、关系和属性。

8.7 选出问题8.6中列出的五个系统之一，为其画出环境级模型（第0层DFD），并为该系统编写环境级的“处理叙述”。
255

8.8 什么是分析包？如何使用分析包？

8.9 为你在问题8.6中选择的系统开发CSPEC和PSPEC，尽可能让你的模型完整。

8.10 某个大城市的公共工程部决定开发基于Web的路面坑洼跟踪和修补系统（PHTRS）。说明如下：

市民可以登录Web站点报告路面坑洼的地点和严重程度。当路面坑洼被上报时，它被记入“路面坑洼跟踪和修补系统”中并分配一个标识号，保存时还有如下信息：街道地址、大小（比例从1到10）、位置（中央、路边等）、地区（由街道地址确定）以及修补优先级（由坑洼大小确定）。工作订单数据和每个坑洼有关，数据包含坑洼位置和大小、维修组标识号、维修组内人员数量、分配的设备、修复时间长度、坑洼状态（正在处理中、已修复、临时修复、未修复）、使用的填充材料数量以及修复成本（从修复时间长度、人员数量、材料和使用的设备计算）。最后，生成损失文件以便保存该坑洼所造成的损失信息，并包含公民的姓名、地址、电话号码、损失类型、损失钱数。PHTRS是基于Web的系统，可交互地进行所有的查询。

使用结构化分析符号为PHTRS开发分析模型。

8.11 说明面向对象的术语封装和继承。

8.12 使用在问题8.7中开发出的环境级的DFD，开发第1层和第2层的数据流图。以对环境级的处理叙述进行“语法分析”为开始，要记住通过在泡泡之间标记所有的箭头来说明所有的信息流，为每个转换使用富有意义的名字。

8.13 分析类的状态图与整个系统的状态图有何不同？

8.14 为问题8.10所提出的PHTRS系统开发一个类模型。

8.15 为问题8.10所提出的PHTRS系统开发一组完整的CRC模型索引卡。

8.16 指导你的同事一起评审CRC索引卡，评审结果增加了多少类、职责和协作者？

8.17 说明某个分析类的“关联”和“依赖”之间的差别。

8.18 为问题8.10所提出的PHTRS系统画出UML用例图，你必须对用户和系统的交互方式做一些假设。

8.19 为8.7.4节中非正式描述的SafeHome住宅管理系统编写基于模板的用例。

推荐读物与阅读信息

关于结构化分析已经出版了很多著作，大多数都很好地涵盖了该主题，但其中只有一些

确实进行了真正出色的工作。DeMarco和Plauger（《Structured Analysis and System Specification》，Pearson，1985）是一部经典著作，至今都是基本符号表示法的优秀入门书。Kendall和Kendall（《Systems Analysis and Design》，第5版，Prentice-Hall，2002）以及Hoffer等人（《Modern Systems Analysis and Design》，Addison-Wesley，第3版，2001）的著作也值得推荐。Yourdon的著作（《Modern Structured Analysis》，Yourdon-Press，1989）至今仍是市面上关于该主题覆盖内容最全的著作。

256

Allen（《Data Modeling for Everyone》，Wrox Press，2002）、Simpson和Witt（《Data Modeling Essentials》，第2版，Coriolis Group，2000）、Reingruber和Gregory（《Data Modeling Handbook》，Wiley，1995）详细地介绍了创建工业品质数据模型的指南。Hay（《Data Modeling Patterns》，Dorset House，1995）的著作提出了一些典型的数据模型模式，在很多不同的业务中都能遇到这些模式。关于行为建模的详细论述可以在Kowal（《Behavior Models: Specifying User's Expectations》，Prentice-Hall，1992）的著作中查阅。

用例构成面向对象分析的基础。Bittner和Spence（《Use-Case Modeling》，Addison-Wesley，2002）、Cockburn[COC01]、Armour和Miller（《Advanced Use-Case Modeling: Software Systems》，Addison-Wesley，2000）、Rosenberg和Scott（《Use-Case Driven Object Modeling with UML: A Practical Approach》，Addison-Wesley，1999）的著作为创建和使用这种重要的需求提取和表现机制提供了有价值的指导。

关于UML有意义的讨论可以在下面著作中找到：Arlow和Neustadt[ARL02]、Schmuller[SCH02]、Fowler和Scott（《UML Distilled》，第2版，Addision-Wesley，1999）、Booch和他的同事（《The UML User Guide》，Addison-Wesley，1998）、Rumbaugh和他的同事（《The Unified Modeling Language Reference Manual》，Addison-Wesley，1998）。

在下面著作中对支持统一过程的基础分析和设计方法有所讨论：Larman（《Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process》，第2版，Prentice-Hall，2001）、Dennis和他的同事（《System Analysis and Design: An Object-Oriented Approach with UML》，Wiley，2001）、Rosenberg和Scott（《Use-Case Driven Object Modeling with UML》，Addison-Wesley，1999）。Balcer和Mellor（《Executable UML: A Foundation for Model Driven Architecture》，Addison-Wesley，2002）讨论了UML的全部语义、可以创建的模型以及将UML看作一种可执行语言的方法。Starr（《Executable UML: How to Build Class Models》，Prentice-Hall，2001）为创建有效的分析和设计类提供了有益的指导原则和详细的建议。

在Internet上可以找到大量的关于分析建模的信息资源。在SEPA的Web站点<http://www.mhhe.com/pressman>可以找到和分析建模有关的最新Web参考列表。

257

第9章 设计工程

要点浏览

概念：实际上，每个工程师都希望做设计工作，因为这里有创造性——客户需求、业务要求和技术限制都在某个产品或系统中得到集中的体现。设计创建了软件的表达或模型，但与分析模型（关注于说明必需的数据、功能和行为）不同，设计模型提供了软件数据结构、体系结构、接口和构件的细节，而这些都是实现系统所必需的。

人员：软件工程师处理所有的设计工作。

重要性：设计要让软件工程师为将要构建的系统或产品建立模型。在生成代码、进行测试以及在涉及大量最终用户之前，可能要评估该模型的质量并进行改进。设计是确立软件质量的关键步骤。

步骤：设计可以采用很多不同的方法描

绘软件。首先，设计必须体现系统或产品的体系结构；其次，为各类接口建模，这些接口在软件和最终用户、软件和其他系统及设备以及软件和自身组成的构件之间起到联系作用；最后，设计用于构建系统的软件构件。每个视图表现了不同的设计活动，但是都要遵循一组基本的设计概念，这些设计概念指导着所有的软件设计工作。

工作产品：在软件设计过程中，包含体系结构、接口、构件和部署表示的设计模型是主要的工作产品。

质量保证措施：软件团队从以下诸方面来评估设计模型：确定设计模型是否存在错误、不一致或遗漏，是否存在更好的方案可供选择，设计模型是否可以在已经设定的限制、时间进度和花费下实现。

关键概念
抽象
体系结构
设计类
模型元素
质量
功能独立性
信息隐藏
模块化
模式
重构
求精

设计工程包括一套原理、概念和实践，可以指导高质量的系统或产品开发。设计原理（在第5章中讨论）建立了最重要的原则，用以指导设计师工作。在运用设计实践的技术和方法之前，必须先理解设计概念，而且设计实践本身会导致产生各种软件设计表示，这些表示将指导随后的构建活动。

虽然设计工程目前还不是软件工程中通用的阶段，但它必将成为一个通用阶段。设计是一项核心的工程活动。在20世纪90年代早期，Lotus 1-2-3的发明人Mitch Kapor在《Dr. Dobbs杂志》上发表了“软件设计宣言”：

什么是设计？设计是你身处两个世界——技术世界和人类的目标世界——而你尝试将这两个世界结合在一起……

罗马建筑批评家Vitruvius提出了这样一个观念：设计良好的建筑应该展示出坚固、适用和令人赏心悦目的特点。对好的软件来说也是如此。所谓坚固，是指程序应该不含任何妨碍其功能的缺陷。适用是要程序符合开发的目标。赏心悦目则是要求使用程序的体验应是愉快的。本章，我们开始介绍软件设计理论。

设计工程的目标是创作出坚固、适用和赏心悦目的模型或设计表示。为此，设计师的做法必须是先实现多样化再行聚合。Belady[BEL81]指出“多样化是指要获取多种方案和设计的原始资料，包括目录、教科书和头脑中的构件、构件方案和知识”。在各种信息汇聚在一起之后，设计师应从其中挑选这样的元素，即能够满足需求工程（第7章）和分析模型（第8章）所定义的需求。此时，设计工程师在经取舍后，进行聚合，使之成为构件的某种特定的配置，于是便得到最终的产品”[BEL81]。

多样化和聚合需要直觉和判断力，其质量取决于构造类似实体的经验、一系列指导模型演化方式的原则和（或）启发、一系列质量评价的标准以及导出最终设计表示的迭代过程。

计算机软件设计工程随着新的方法、更好的分析和更广泛的理解的演化而不断变化。实际上，当前大多数软件设计方法都缺少那些更经典的工程设计学科所具有的深度、灵活性和定量性。然而，软件设计的方法是存在的，设计质量的标准是可以获得的，设计表示法也是能够应用的。在本章中，我们将探讨可以应用于所有软件设计的基本概念和原则、设计模型的元素以及模式对设计过程的影响。在第10~12章中，我们考察应用于体系结构、接口和构件级设计的各种各样的设计方法。

9.1 软件工程中的设计

软件设计在软件工程过程中处于技术核心，并且它的应用与所使用的软件过程模型无关。对软件需求进行分析和建模开始之后，软件设计是建模活动的最后一个软件工程活动，接着便要进入构造阶段（生成代码和测试）。

259

“软件工程最常见的奇迹是从分析到设计以及从设计到代码的变迁。” ——Richard Due



设计工程通常应
开始于数据的一致性——这是其他所有设计元素的基础。在基础建立后，必须导出体系结构。只有这样，才能继续执行其他设计工作。

分析模型（第8章）的每个元素都提供了创建四种设计模型所需的信息，这四种设计模型是完成完整的设计规格说明所必需的。软件设计过程中的信息流如图9-1所示。由基于场景的元素、基于类的元素、面向流的元素和行为元素所表明的分析模型是设计任务的输入。使用后续章中讨论的设计表示法和设计方法，将得到数据/类设计、体系结构设计、接口设计和构件设计。

数据/类设计将分析类模型（第8章）转化为设计类的实现以及软件实现所要求的数据结构。CRC索引卡（8.7.4节）定义的类和关系、类属性和其他表示法刻画的详细数据内容为数据设计活动提供了基础。在和软件体系结构设计连接中可能会有部分的类设计，更详细的类设计在设计每个软件构件时进行。

体系结构设计定义了软件的主要结构元素之间的联系、可用于达到系统所定义需求的体系结构风格和设计模式以及影响体系结构实现方式的约束[SHA96]。体系结构设计表示——基于计算机系统的框架——可以从系统规格说明、分析模型和分析模型中定义的子系统的交互导出。

260

接口设计描述了软件和协作系统之间、软件和使用人员之间是如何通信的。接口就意味着信息流（如数据流和/或控制流）和特定的行为类型。因此，使用场景和行为模型为接口设计提供了所需的大量信息。

构件级设计将软件体系结构的结构元素变换为对软件构件的过程性描述。从基于类的模型、流模型和行为模型获得的信息将作为构件设计的基础。

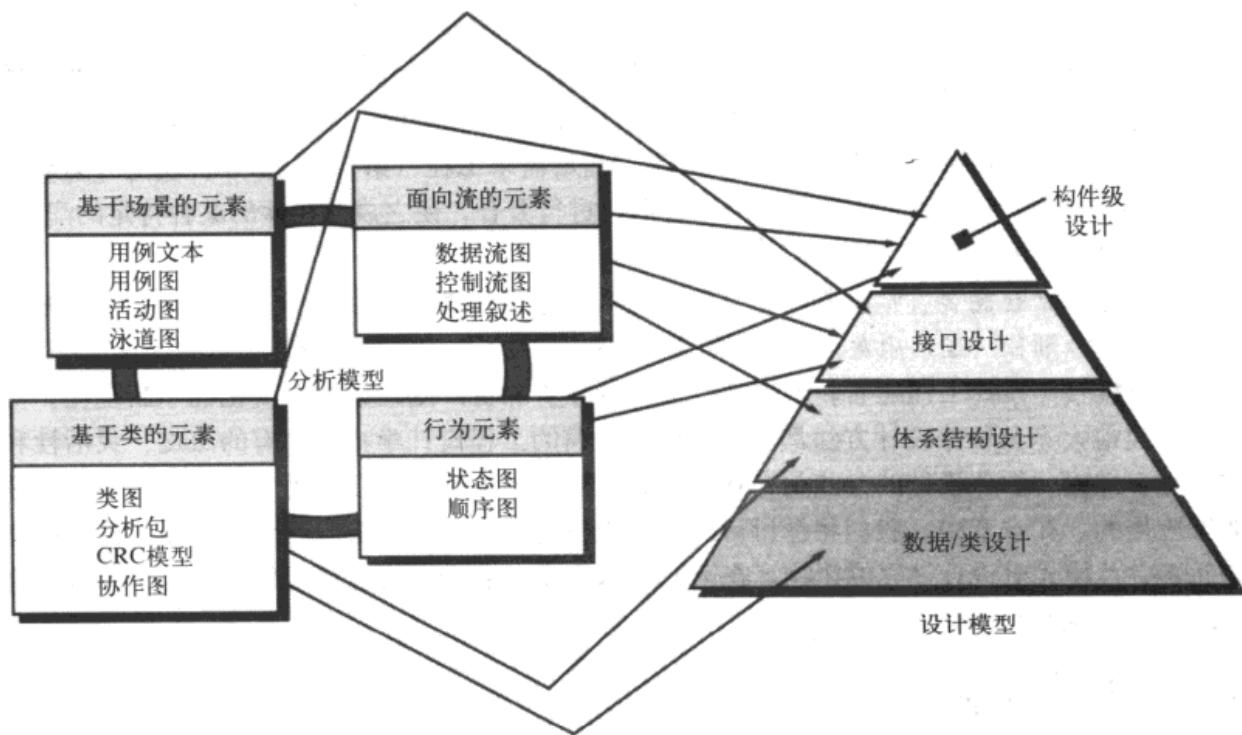


图9-1 从分析模型到设计模型的转化

“有两种构造软件设计的方式：一种是使其尽可能简单以致明显地没有不足；另一种是使其尽可能复杂以致明显地没有不足。第一种方式更为困难。” ——C.A.R Hoare

我们在设计过程中所做出的决定将最终影响软件构建的成功，而且重要的是会影响软件维护的难易程度。但是，设计为什么如此重要呢？

软件设计的重要性可以用一个词来表达——质量。设计是软件工程中形成质量的地方，设计为我们提供了可以用于质量评估的软件表示，设计是我们能够将用户需求准确地转化为软件产品或系统的唯一方法。软件设计是所有软件工程活动和随后的软件支持活动的基础。没有设计，我们将冒构造不稳定系统的风险，这样的系统稍做改动就无法运行，而且难以测试，直到软件工程过程的后期才能评估其质量，而那时时间已经不够并且已经花费了大量经费。

9.2 设计过程和设计质量

软件设计是一个迭代的过程，通过设计过程，需求被变换为用于构建软件的“蓝图”。初始时，蓝图描述了软件的整体视图，也就是说，设计是在高抽象层次上的表达——在该层次上可以直接跟踪到特定的系统目标和更详细的数据、功能和行为需求。随着设计迭代的开始，后续的精化导致更低抽象层次的设计表示。这些表示仍然能够跟踪到需求，但是连接更加错综复杂了。

在整个设计过程中，使用第26章中讨论的一系列正式技术评审或设计走查来评估设计演化的质量。McGlaughlin[MCG91]提出了可以指导评价良好设计演化的三个特征：

- 设计必须实现所有包含在分析模型中的明确需求，而且必须满足客户期望的所有隐含需求。
- 对于那些生成代码的人和那些进行测试以及随后维护软件的人而言，设计必须是可读的、可理解的指南。
- 设计必须提供软件的全貌，从实现的角度说明数据域、功能域和行为域。

以上每一个特征实际上都是设计过程的目标，但是如何达到这些目标呢？

“编写一段能工作的灵巧的代码是一回事；而设计能支持某个长久业务的东西则完全是另一回事。”

—C. Ferguson

质量指导原则。为了评估某个设计表示的质量，我们必须建立优秀设计的技术标准。在本章的后面部分，我们将讨论设计概念，这些概念也可以作为软件质量的评价标准。现在，我们给出如下的指导原则：

优秀设计的
特征是什么？

1. 设计应展示出这样一种结构：(a) 已经使用可识别的体系结构风格或模式创建；(b) 由展示出良好设计特征的构件（将在本章后面讨论）构成；(c) 能够以演化的方式实现¹，从而便于实现和测试。
2. 设计应该模块化；也就是说，软件应按照逻辑划分为元素或子系统。
3. 设计应该包含数据、体系结构、接口和构件的清楚的表示。
4. 设计应导出数据结构，这些数据结构适于要实现的类，并由可识别的数据模式提取。
5. 设计应导出显示独立功能特征的构件。
6. 设计应导出接口，这些接口降低了构件之间以及与外部环境连接的复杂性。
7. 设计的导出应根据软件需求分析过程中获取的信息采用可重复使用的方法进行。
8. 应使用能够有效传达其意义的表示法来表达设计。

达到这些设计原则不能靠偶然，设计工程应通过应用基本的设计原则、系统化的方法学和严格的评审来保证良好的设计。

262 INFO

评估设计质量——正式技术评审

设计之所以重要在于它允许一个软件团队在软件实现前评估软件的质量²——此时修正错误、遗漏或不一致既容易且代价不高。但是我们如何在设计过程中评估质量？此时，不可能去测试软件，因为还没有可执行的软件，那么怎么办？

在设计阶段，通过开展一系列的正式技术评审（Formal Technical Review, FTR）来评估质量。FTR在第26章³中有详细的讨论，这里先概述一下该技术。FTR是由软件团队成员召开的会议。通常，根据将要评审的设计信息的范围，选择两人、三人或四人参与，每人扮演一个角色：评审组长策划会议、拟定议程并主持会议；记录员记录笔记以保证没有遗漏；制作人是指其工作产品（例如某个软件构件的设计）被评审的人。在会议之前，每个与会者都会收到一份设计工作产品的拷贝并要求阅读，寻找错误、遗漏或含糊不清的地方。当会议开始时，目的是注意到工作产品中的所有问题，以便能够在开始实现之前修正这些问题。FTR通常持续90分钟到两个小时。在FTR结束时，评审小组要确认在设计工作产品能够被认可为最终设计模型的一部分之前，是否还需要进一步的行动。

¹ 对于更小的系统，设计有时候可以线性地进行。

² 在第15章中讨论的质量因子有助于评审团队评估质量。

³ 这时也可以参阅26.4节，FTR是设计过程中的重要部分，是获得高质量设计的重要机制。

“质量不是那些被束之高阁的观赏品，也不是像圣诞树上的闪亮金箔那样的装饰品。”

——Robert Pirsig

质量属性。Hewlett-Packard[GRA87]开发了一系列的软件质量属性，并取其首字母组合为FURPS，其中各字母分别代表功能性(functionality)、易用性(usability)、可靠性(reliability)、性能(performance)、可支持性(supportability)。FURPS质量属性体现了所有软件设计的目标：



263

软件设计人员倾向于关注要解决的问题。只是不能忘记FURPS属性毕竟是问题的一部分，因而必须考虑。

- 功能性评估程序的特征集和能力、所提交功能的普遍性以及整个系统的安全性。
- 易用性通过考虑人为因素(第12章)、整体美感、一致性和文档来评估。
- 可靠性通过测量故障的频率和严重性、输出结果的精确性、故障平均时间(Mean-Time-To-Failure, MTTF)、故障恢复能力和程序的可预见性来评估。
- 性能度量处理速度、响应时间、资源消耗、吞吐量和效率。
- 可支持性综合了扩展程序(可扩展性)、适应性和耐用性这三方面的能力(这三个属性体现了一个更通用的术语：可维护性)，此外还包括可测试性、兼容性、可配置性(组织和控制软件配置元素的能力，第27章)、系统安装的简易性和问题定位的简易性。

在将要开发的软件中，并不是每个软件质量属性都具有相同的分量。有的应用问题可能强调功能性，特别突出安全性。还有的应用问题可能要求性能，特别突出处理速度。还有的可能关注于可靠性。抛开权重不谈，重要的是要注意到必须在设计开始时就考虑这些质量属性，而不是在设计完成后和已经开始构建时才考虑。

TASK SET

通常设计任务集

1. 检查信息域模型，并为数据对象及其属性设计恰当的数据结构。
2. 使用分析模型，选择一个适于软件的体系结构类型(模式)。
3. 将分析模型分割为若干个设计子系统，并在体系结构内分配这些子系统。要确定每个子系统是功能内聚的。

设计子系统接口。

为每个子系统分配分析类或功能。

4. 创建一系列的设计类或构件。将每个分析类说明转化为设计类。

据设计标准检查每个设计类，考虑继承问题。

定义与每个设计类相关的方法和消息。

评估设计类或子系统并为这些类或子系统选择设计模式。

评审设计类，并在需要时修改。

5. 设计外部系统或设备所需要的所有接口。

6. 设计用户接口。

评审任务分析的结果。

基于用户场景详细说明活动序列。

创建接口的行为模型。

- 定义接口对象、控制机制。
- 评审接口设计，并在需要时修改。
- 7. 进行构件级设计。
 - 在相对较低的抽象层次上详细地说明所有算法。
 - 精化每个构件的接口。
 - 定义构件级的数据结构。
 - 评审每个构件并修正所有已发现的错误。
- 8. 开发部署模型。

264

9.3 设计概念

在软件工程的历史进程中发展了一系列基本的软件设计概念。尽管多年来对于每一种概念的关注程度不断变化，但它们都经历了时间的考验。每一种概念都为软件设计者提供了应用更加复杂设计方法的基础。

M. A. Jackson[JAC75]曾经说过：“软件工程师的智慧源于认识到使程序工作和使程序正确之间的差异。”而基础的软件设计概念为“使程序正确”提供了必要的框架。

9.3.1 抽象

当我们考虑某一问题的模块化解决方案时，可以给出许多抽象级。在最高的抽象级上，使用问题所处环境的语言以概括性的术语描述解决方案。在较低的抽象级上，将提供更详细的解决方案说明。

“抽象是人类处理复杂问题的基本方法之一。”

—Grady Booch



设计人员努力地获取过程抽象和数据抽象，这两类抽象都将为手边的问题提供服务。如果它们能够服务于整个问题域，那就更好。

当我们在不同的抽象级间移动时，我们力图创建过程抽象和数据抽象。过程抽象是指具有明确和有限功能的指令序列。过程抽象的命名暗示了这些功能，但是隐藏了具体的细节。过程抽象的例子如单词“开”门。“开”隐含了一长串的过程性步骤（例如，走到门前，伸出手并抓住把手，转动把手并拉门，离开打开的门等）⁴。

数据抽象是描述数据对象的冠名数据集合。在过程抽象“开”的情形下，我们可以定义一个名为门的数据抽象。同任何数据对象一样，门的数据抽象将包含一组描述门的属性（例如，门的类型、转动方向、开门机构、重量和尺寸）。因此，过程抽象“开”将利用数据抽象门的属性中所包含的信息。

9.3.2 体系结构

软件体系结构意指“软件的整体结构和这种结构为系统提供概念上完整性的方式”[SHA95a]。从最简单的形式来看，体系结构是程序构件（模块）的结构或组织、这些构件交互的形式以及这些构件所用数据的结构。然而在更广泛的意义上，构件可以被推广，用于代

265

⁴ 但是应注意到，只要过程抽象所蕴含的功能保持相同，一个系列的操作可以被另一个替换。因此，如果是自动门并且连接了一个传感器，实现“开”所需的步骤明显会有所不同。

表主要的系统元素及其交互。

“软件体系结构是在质量、进度和成本方面有最高投资回报的工作产品。”——Len Bass等

WebRef

关于软件体系结构的更有深度的讨论可以参考
www.sei.cmu.edu/ata/ata_init.html。



不要让体系结构仅仅保持最初的形式。如果这样做了，剩余的项目将不得不强行配合设计。应该明确清楚地设计体系结构。

软件设计的目标之一是导出系统的体系结构透视图，该透视图作为一个框架，将指导更详细的设计活动。一系列的体系结构模式使得软件工程师能够复用设计级的概念。

体系结构设计可以使用大量的一种或多种模型来表达[GAR95]。结构模型将体系结构表示为程序构件的一个有组织的集合。通过确定类似应用中遇到的可复用的体系结构来设计框架，框架模型可以提高设计抽象级别。动态模型强调程序体系结构的行为方面，指明结构或系统配置作为外部事件的函数将如何变化。过程模型注重系统必须提供的业务设计或技术流程设计。最后，功能模型可以用来表示系统的功能层次结构。在第10章将对体系结构设计进行讨论。

9.3.3 模式

Brad Appleton定义设计模式如下：“模式是冠名的洞察力财宝，对于竞争事件中某确定环境下重复出现的问题，它承载了已证实的解决方案的精髓。”[APP98]。换句话说，设计模式描述了在某个特定场景与可能影响模式应用和使用方式的“影响力”中解决某个特定的设计问题的设计结构。

“每个模式都描述了一个在我们所处环境内反复发生的问题，然后描述该问题的核心解决方案，你可以几百万次地重复使用该解决方案而根本不需要同样的方式重复工作两次。”
 ——Christopher Alexander

266

每个设计模式的目的都是提供一个描述，以使得设计人员能够确定：(1) 模式是否适合当前的工作；(2) 模式是否能够复用（因此，节约设计时间）；(3) 模式是否能够用于指导开发一个类似但是功能或结构不同的模式。在9.5节中针对设计模式将有更详细的讨论。

9.3.4 模块化

软件体系结构和设计模式表现为模块化；也就是说，软件被划分为独立命名的、可寻址的构件，有时被称为模块，把这些构件集成到一起可以满足问题的需求。

有人提出“模块化是软件的单个属性，它使程序能被理性地管理”[MYE78]。软件工程师难以掌握单块软件（即由一个单独模块构成的大程序）。对于大型程序，其控制路径的数量、引用的跨度、变量的数量和整体的复杂度使得理解这样的软件几乎是不可能的。为说明这一点，考虑下面的论据，这些论据基于对人类解决问题规律的观察。

考虑两个问题， p_1 和 p_2 ，如果 p_1 的理解复杂度大于 p_2 的理解复杂度，那么解决 p_1 所需的工作量大于解决 p_2 所需的工作量。作为普遍的情况，这个结果直观上是显而易见的，因为解决

困难问题的确需要花费更多的时间。

另一个结果是两个问题结合时的理解复杂度通常要大于每个问题各自的理解复杂度之和。这就引出了“分而治之”的策略——将一个复杂问题分解成可以管理的若干块，这样能够更容易地解决问题。这对于模块化和软件是意义重大的，事实上，这就是模块化的论据。



不要过度模块化。每个模块的简单性将被集成的复杂性所掩盖。

可以得出结论：如果我们无限制地划分软件，那么开发它所需的工作量会变得小到可以忽略！不幸的是，其他因素开始起作用，导致这个结论是不成立的（十分遗憾）。如图9-2所示，开发某个独立软件模块的工作量（成本）的确随着模块数增加而下降，给定同样的需求，更多的模块意味着每个模块的规模更小。然而，随着模块数量增加，集成模块的工作量（成本）也在增加。这些特性形成了如图所示的总体成本或工作量曲线。事实上，的确存在一个模块数量 M ，这个数量可以带来最小的开发成本。但是，我们缺乏必要的技术来精确地预测 M 。

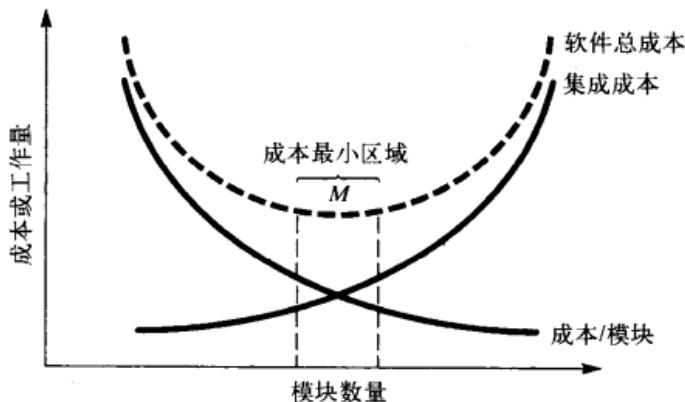


图9-2 模块化和软件成本

267

当考虑模块化时，图9-2所示的曲线确实提供了有益的指导。我们应该进行模块化，但是应注意保持在 M 附近。应该避免过低或过高的模块化。但是我们如何知道 M 的附近在哪呢？我们应该怎样将软件划分成模块呢？回答这些问题需要理解本章后面提出的其他设计概念。

我们做模块化设计（以及由其产生的程序）可以更容易地计划开发；可以定义和交付软件增量；更容易实施变更；能够更有效地开展测试和调试；可以开展长期维护而没有严重的副作用。

9.3.5 信息隐蔽



信息隐蔽的目的
是在一个模块接
口后面隐藏数据
结构的细节和过
程的处理，模块
的用户不需要知
道细节。

模块化概念让每个软件设计师面对一个基本问题：我们将如何分解一个软件解决方案以求获得最好的模块集合？信息隐蔽[PAR72]原则建议模块应该“具有的特征是：每个模块对其他所有模块都隐蔽自己的设计决策”。换句话说，模块应该详细说明且精心设计以求在某个模块中包含的信息（算法和数据）不被不需要这些信息的其他模块访问。

隐蔽意味着通过定义一系列独立的模块可以得到有效的模块化，独立模块相互之间只交流实现软件功能所必需的那些信息。抽象有助于定义构成软件的过程（或信息）实体。隐蔽定义并加强了模块内的过程细节和模块所使用的任何局部数据结构的访问约束[ROS75]。

把信息隐蔽用做模块化系统的一个设计标准，在测试和随后的软件维护过程中，在需要修改时将提供最大的益处。因为大多数数据和程序对软件的其他部分是隐蔽的，因此在修改过程中，无意地引入错误并传播到软件其他地方的可能性会很小。

9.3.6 功能独立

功能独立的概念是模块化、抽象概念和信息隐蔽的直接结果。在关于软件设计的一篇里程碑式的文章中，Wirth[WIR71]和Parnas[PAR72]间接提到增强模块独立性的精化技术。Stevens、Myers和Constantine[STE74]之后又充实了这一概念。

通过开发具有“专一”功能和“避免”与其他模块过多交互的模块，可以实现功能独立。
[268] 换句话说，我们希望软件设计时要使每个模块仅涉及需求的某个特定子功能，并且当从程序结构的其他部分观察时，每个模块只有一个简单的接口。一个很合理的问题是：独立为什么重要。

具有有效模块化（也就是独立模块）的软件更容易开发，这是因为功能被分隔而且接口被简化（考虑由一个团队进行开发时的结果）。独立模块更容易维护（和测试），因为修改设计或修改代码所引起的副作用被限制，减少了错误扩散，而且模块复用也成为可能。概括地说，功能独立是优秀设计的关键，而设计又是软件质量的关键。



POINT
内聚性定性地指示一个模块只关注一个事情的程度。

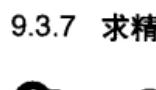
独立性可以使用两条定性的标准评估：内聚性和耦合性。内聚性显示了某个模块相关功能的强度；耦合性显示了模块间的相互依赖性。

内聚性是9.3.5节说明的信息隐蔽概念的自然扩展。一个内聚的模块执行一个独立的任务，与程序的其他部分构件只需要很少的交互。简单地说，一个内聚的模块应该（理想情况下）只完成一件事情。



POINT
耦合性定性地指示一个模块和其他模块以及和外部世界联系的程度。

耦合性表明软件结构中多个模块之间的相互连接。耦合性依赖于模块之间的接口复杂性、引用或进入模块所在的点以及什么数据通过接口传递。在软件设计中，我们将尽力得到尽可能低的耦合。模块间简单的连接性使得软件易于理解并减少“涟漪效果”（ripple effect）的倾向[STE74]，“涟漪效果”是指在某个地方发生错误时导致扩散到整个系统。



ADVICE
存在这样一种倾向：跳过求精步骤而直接转移到最底层的细节。这将导致错误和疏忽，并使设计更加难以评审。还是进行逐步求精为好！

逐步求精是由Niklaus Wirth[WIR71]最初提出的一种自顶向下的设计策略。通过连续精化层次结构的程序细节来实现程序的开发，层次结构的开发将通过逐步分解功能的宏观陈述（过程抽象）直至形成程序设计语言的语句。

求精实际上是一个细化的过程。我们从在高抽象级上定义的功能陈述（或数据描述）开始，也就是说，该陈述概念性地描述了功能或信息，但是没有提供有关功能内部工作的信息或数据内部结构的信息。精化促使设计者在原始陈述上细化，并随着每个精化（细化）的持续进行将提供越来越多的细节。

抽象和精化是互补的概念。抽象使得设计人员能够明确说明过程和数据而同时忽略低层细节；精化有助于设计人员在设计过程中揭示低层的细节。这两个概念均有助于设计人员在设计演化中构造出完整的设计模型。

“我没有失败，我只是发现了10 000种行不通的道路。”

——Thomas Edison

9.3.8 重构

WebRef
www.refactoring.com有关于重构的非常好的资源。

很多敏捷方法（第4章）都建议一种重要的设计活动——重构，重构是一种重新组织的技术，可以简化构件的设计（或代码）而无需改变其功能或行为。Fowler[FOW99]这样定义重构：“重构是使用这样一种方式改变软件系统的过程：不改变代码[设计]的外部行为而是改进其内部结构。”

当重构软件时，检查现有设计的冗余性、没有使用的元素、低效的或不必要的算法、拙劣的或不恰当的数据结构以及其他设计不足，修改这些不足以获得更好的设计。例如，第一次设计迭代可能得到一个构件，表现出很低的内聚性（即，执行三个功能但是相互之间仅有有限的联系）。设计人员可以决定将构件重构为三个独立的构件，每个都表现出较高的内聚性。这样的处理结果将实现一个更容易集成、更容易测试和更容易维护的软件。

SAFEHOME

设计概念

[场景] Vinod的房间，设计建模开始。

[人物] Vinod、Jamie和Ed，SafeHome软件工程团队成员；Shakira，是团队的新成员。

[对话]

（这四个团队成员上午都参加了一位本地计算机科学教授举行的名为“应用基本的设计概念”的研讨会，他们刚从会上回来。）

Vinod：你们从研讨会学到什么没有？

Ed：大部分的东西我都已经知道，但我想重温一遍总不是什么坏事。

Jamie：我在计算机科学专业学习时，我从没有真正理解信息隐蔽为什么像他们说得那么重要。

Vinod：因为……底线……这是减少错误在程序内扩散的一种技术。实际上，功能独立做的也是同样的事。

Shakira：我不是计算机科学专业的，因此教授提到的很多东西对我而言都是新的。我能生成好的代码而且速度快，我不明白这个东西为什么这么重要。

Jamie：我了解你的工作，Shak，你要知道，其实你是在自然地做这些事情……这就是为什么你的设计和编码很有效。

Shakira（微笑）：就是，我通常的确是尽量将代码分割，让分割后的代码关注于一件事，保持接口简单而且有约束，在任何可能的时候重用代码……就是这样。

Ed：模块化、功能独立、隐蔽、模式……知道吧。

Jamie：我至今还记得我上的第一堂编程课……他们教我们迭代地精化代码。

Vinod：设计可以采用同样的方式，你知道的。

Ed：唯一一个我以前从未听说过的概念是“重构”。

Shakira：我记得她说那是用在极限编程中的。

Ed：是的。其实它和精化并没有太大不同，它只是在设计或代码完成后进行。我认为，这是软件开发过程中的一种优化。

Jamie：让我们回到SafeHome设计。我觉得在我们开发SafeHome的设计模型时，我们应该将这些概念用在评审检查单上。

Vinod：我同意。但重要的是，我们都要做到在做设计时想一想这些概念。

9.3.9 设计类

在第8章，我们提到分析模型定义了一组完整的分析类。这些类中的每一个都关注于用户或客户可见的问题，描述问题域中的某些元素。分析类的抽象级相对较高。

设计人员可以创建什么类型的类？

在设计模式演化时，软件团队必须定义一组设计类：(1) 通过提供设计细节精化分析类，这些设计细节将促成类的实现；(2) 创建一组新的设计类，该设计类实现了软件的基础设施以支持业务解决方案。[\[AMB01\]](#)建议了五种不同类型的设计类，每一种都表现了设计体系结构的一个层次：

- 用户接口类定义人—机交互(Human-Computer Interaction, HCI)所必需的所有抽象。在很多情况下，HCI出现在隐喻的环境（例如，支票簿、订单表格、传真机），而接口的设计类可能是这种隐喻元素的形象表示。
- 业务域类通常是早期定义的分析类的精化。这些类识别实现某些业务域元素所必需的属性和服务（方法）。
- 过程类实现完整管理业务域类所必需的低层业务抽象。
- 持久类代表将在软件执行之外持续存在的数据存储（例如，数据库）。
- 系统类实现软件管理和控制功能，使得系统能够运行并在其计算环境内与外界通信。

271 在设计模型演化时，软件团队必须为每个设计类开发一组完整的属性和操作。随着每个分析类转化为设计表示，抽象级就降低。也就是说，分析类使用业务域的专门用语描述对象（以及对象所用的相关服务）；设计类更多地表现技术细节，将作为实现的指导。

Arlow和Neustadt[\[ARL02\]](#)建议应评审每个设计类以确保设计类是“组织良好的”(well-formed)。他们为组织良好的设计类定义了四个特征：

什么是组织良好的设计类？

完整性与充分性。设计类应该完整地封装所有的，可以合理预见（根据对类名的理解）会存在于类中的属性和方法。例如，为视频编辑软件定义的**Scene**类，只有当它包含与创建视频场景相关的所有合理的属性和方法时，才是完整的。充分性确保设计类只包含那些“对实现这个类的目的足够”的方法，不多也不少。

原始性。和某个设计类相关的方法应该关注于实现类的某个服务。一旦服务已经被某个方法实现，类就不应该再提供另外一种完成同一事情的方法。例如，视频编辑软件的**VideoClip**类，可能用属性**start-point**和**end-point**指定剪辑的起点和终点（注意，加载到系统的原始视频可能比要用的部分长）。方法**setStartPoint()**和**setEndPoint()**为剪辑提供了设置起点和终点的唯一手段。

高内聚性。一个内聚的设计类具有小的、集中的职责集合，并且专注于使用属性和方法来实现那些职责。例如，视频编辑软件的**VideoClip**类可能包含一组用于编辑视频剪辑的方法。只要每个方法只关注于和视频剪辑相关的属性，内聚性就得以维持。

低耦合性。在设计模型内，设计类之间相互协作是必然的。但是，协作应该保持在一个可以接受的最小范围内。如果设计模型高度耦合（所有的设计类都和其他所有的设计类有协作关系），那么系统就难以实现、测试，并且维护也很费力。通常，一个子系统内的设计类对其他子系统中的类应仅有有限的了解。该限制被称作Demeter定律[\[LIE03\]](#)，该定律建议一个方法应该只向周围类中的方法发送消息⁵。

272

⁵ 关于Demeter定律不太严谨的一种解释是“每个单元应该只和它的朋友交谈，而不要和陌生人交谈”。

SAFEHOME

将分析类精化为设计类

[场景] Ed的办公室，继续设计建模。

[人物] Vinod和Ed，SafeHome软件工程团队成员。

[对话]

(Ed正进行**FloorPlan**类（参考8.7.4节的讨论以及图8-14）的工作，并进行设计模型的精化。)

Ed：你还记得**FloorPlan**类吗？这个类用作监视和住宅管理功能的一部分。

Vinod（点头）：是的，我好像想起来我们把它用作住宅管理CRC讨论的一部分。

Ed：确实如此。不管怎样，我们要把设计精化，希望显示出我们将如何真正地实现**FloorPlan**类。我的想法是把它实现为一组链表（一个专用数据结构）。像这样……我已经精化分析类**FloorPlan**（图8-14），而且实际上是简化后的。

Vinod：分析类只显示问题域中的东西，也就是说，在电脑屏幕上实际显示的、最终用户可见的那些东西，对吗？

Ed：是的。但对于**FloorPlan**设计类来说，我已经开始添加一些实现中特有的东西。需要说明的是**FloorPlan**是墙段——就是**Segment**类——的聚集，**Segment**类由墙段、窗户、门等的列表组成。**Camera**类和**FloorPlan**类协作，这很显然，因为在平面图中可以有很多摄像头。

Vinod：咳，让我们看看新的**FloorPlan**设计类图。

(Ed向Vinod展示如图9-3所示的图。)

Vinod：好的，我看出来你想做什么了。这样你能够容易地修改平面图，因为新的东西可以在列表中添加或删除——聚集——而不会有任何问题。

Ed（点头）：确实，我想会是这样的。

Vinod：我也赞同。

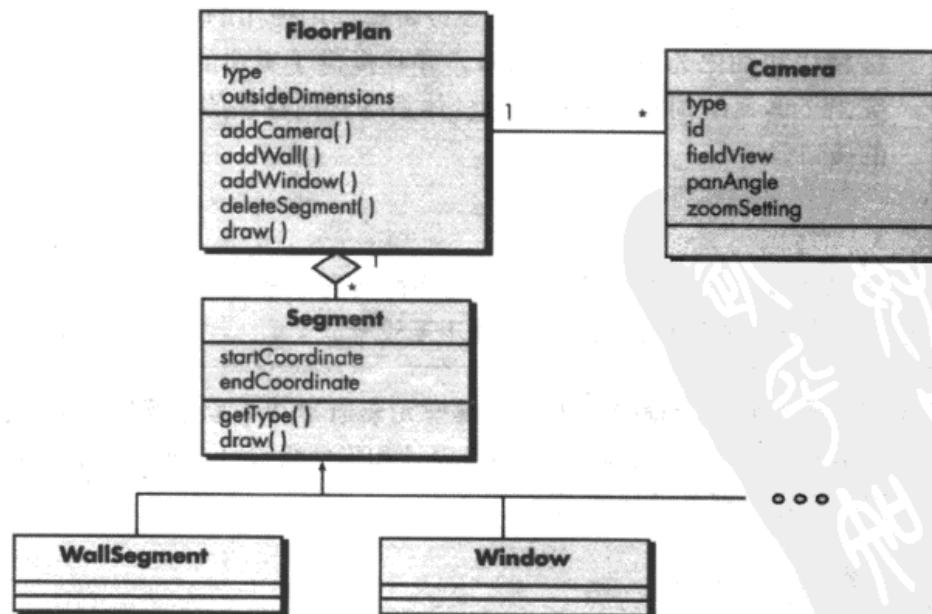


图9-3 FloorPlan的设计类和类的复合聚集

9.4 设计模型

设计模型可以从两个不同的维度观察，如图9-4所示。过程维度表示设计模型的演化，设计工作作为软件过程的一部分被执行；抽象维度表示过程的详细级别，分析模型的每个元素转化为一个等价的设计，然后迭代精化。参考图9-4，虚线表示分析模型和设计模型之间的边界。在某些情况下，分析模型和设计模型之间可能存在清晰的差别；而有些情况下，分析模型慢慢地融入设计模型而没有明显的差别。

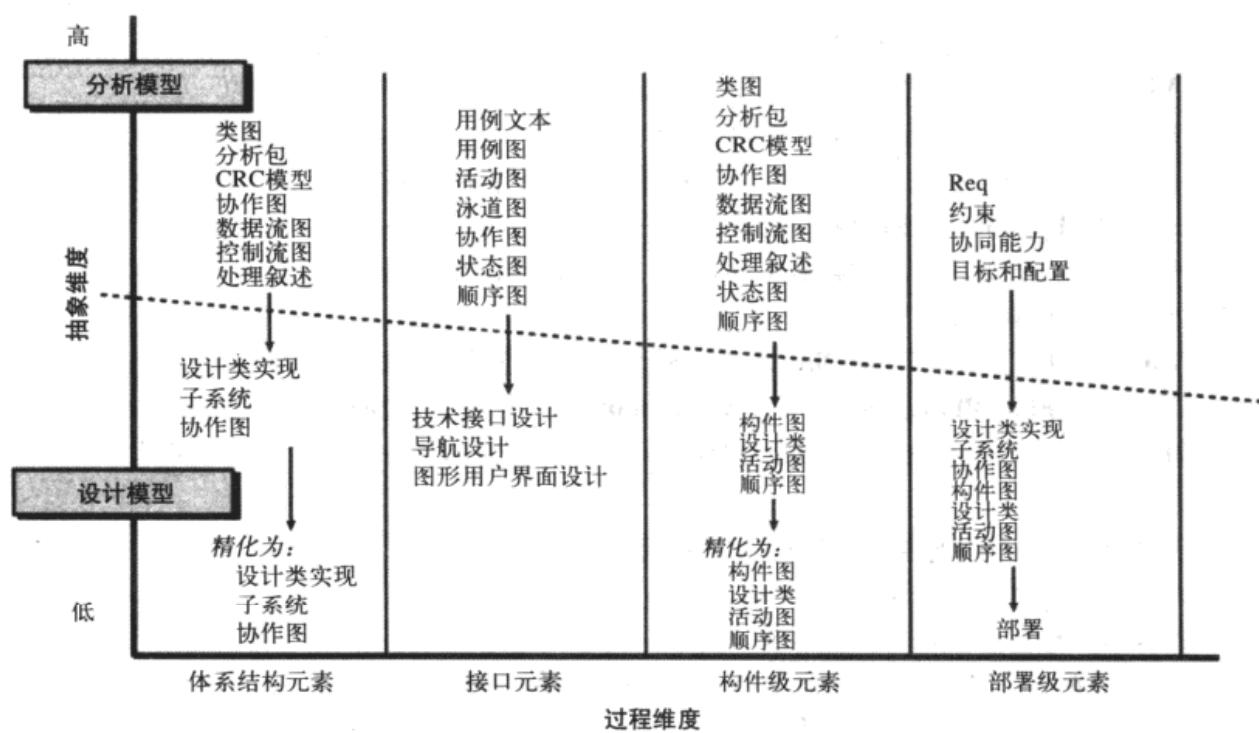


图9-4 设计模型的维度

设计模型的元素很多都是在分析模型中使用的UML图。差别在于这些图被精化和细化为设计的一部分，并且提供了更多的与实现相关的特殊细节，突出了体系结构的结构和风格、体系结构内存在的构件以及构件和外界之间的接口。

设计模型有四个主要元素：数据，体系结构，构件和接口。

“提出设计是否是必要的或能否负担得起这样的问题非常离题，因为设计是不可避免的。不是好的设计就是坏的设计，根本不可能不要设计。” ——Douglas Martin

然而，重要的是要看到，沿横坐标表示的模型元素通常并不总是顺序开发的。大多数情况下，初步的体系结构设计是基础，随后是接口设计和构件级设计（通常是并行进行）。通常直到设计全部完成后才开始部署模型的工作。

9.4.1 数据设计元素

和其他软件工程活动一样，数据设计（有时也称为数据体系结构设计）创建在高抽象级



在体系结构（应用程序）级，数据设计关注于文件或数据库；在构件级，数据设计考虑实现本地数据对象所必需的数据结构。

上（以客户/用户的观点）表示的数据模型和/或信息模型。然后，数据模型被精化为越来越和实现相关的特定表示，亦即基于计算机的系统能够处理的表示。在很多软件应用中，数据体系结构对必须处理该数据的软件的体系结构有深远的影响。

数据结构通常是软件设计的重要部分。在程序构件级，数据结构设计以及相关的处理这些数据的算法对于创建高质量的应用程序是至关重要的。在应用程序级，数据模型（源自于需求工程）到数据库的转变是实现系统业务目标的关键。在业务级，收集存储在不同的数据库中的信息并重新组织为“数据仓库”要使用数据挖掘或知识发现技术，这些技术影响业务本身的成功。在各种情况下，数据设计都发挥了重要作用。在第10章中将更详细地讨论数据设计。

9.4.2 体系结构设计元素

软件的体系结构设计等效于房屋的平面图。平面图描绘了房间的整体布局，包括各房间的尺寸、形状、相互之间的联系，能够进出房间的门窗。平面图为我们提供了房屋的整体视图；而体系结构设计元素为我们提供了软件的整体视图。

“你可以在绘图桌上使用橡皮或在建筑工地现场使用大铁锤。”——Frank Lloyd Wright

体系结构模型源于应用域、分析模型和有效的风格和模式。

体系结构模型[SHA96]从以下三个来源获得：(1) 关于将要构建的软件的应用域信息；(2) 特定的分析模型元素，例如数据流图或分析类、现有问题中它们的关系和协作；(3) 体系结构模式（9.5节）和风格（第10章）的可获得性。

275

9.4.3 接口设计元素

软件的接口设计相当于一组房屋的门、窗和外部设施的详细绘图（以及规格说明）。这些绘图描绘了门窗的尺寸和形状、门窗的工作方式、设施（例如，水、电、气、电话）连接入室的方式和在平面图上的室内布置。图纸可以告诉我们门铃在哪、是否使用内部通信以通知有客来访以及如何安装保安系统。门、窗、外部设施的详细图纸（以及规格说明）大体上告诉我们事件和信息如何流入和流出住宅以及如何在平面图的房间内流动。类似地，软件接口设计元素告诉我们信息如何流入和流出系统以及被定义为体系结构一部分的构件之间是如何通信的。

“相比较于优秀设计，公众更熟悉拙劣的设计。实际上，公众更习惯于拙劣的设计，因为这是实际的生活。新的有危险，而旧的更让人安心。” ——Paul Rand

接口设计有三个重要的元素：(1) 用户界面（UI）；(2) 和其他系统、设备、网络或其他的信息生产者或使用者的外部接口；(3) 各种设计构件之间的内部接口。这些接口设计元素允许软件和外部通信，并使得在软件体系结构内存在的构件之间能够内部通信和协作。

UI设计是软件工程中的主要活动，这会在第12章中详细地考虑。UI设计包含美学元素

KEY POINT

接口设计元素有三部分：用户界面；与外部系统的接口；应用程序内部构件的接口。

(例如布局、颜色、图形、交互机制)、人机工程元素(例如信息布局、隐喻、UI导航)和技术元素(例如UI模式、可复用构件)。通常，UI是整个应用体系统结构内独一无二的子系统。

外部接口的设计需要关于发送和接收信息的实体的确定信息。在各种情况下，这些信息应在需求工程(第7章)中收集，并且一旦开始进行接口设计，还要检验这些信息⁶。外部接口设计应包括错误检查和(在必要时)适当的安全特征。

276 内部接口的设计和构件级的设计(第11章)紧密相关。分析类的设计实现体现了包含如下内容的方案：在各种类的运作之间实现通信和协作所必需的所有操作和消息发送模式。每个消息的设计必须提供必不可少的信息传递和已被请求的操作的特定功能需求。

WebRef

可以在www.us-eit.com上找到关于UI设计的非常有价值的信息。

在有些情况下，接口建模的方式和类所用的方式几乎一样。UML定义接口如下[OMG01]：“接口是类、构件或其他分类(包括子系统)的外部可见的[公共的]操作说明，而没有内部结构的规格说明。”更简单地说，接口是一组描述类的部分行为的操作，并提供了那些操作的访问方法。

例如，SafeHome安全功能使用控制面板，即允许房主控制安全功能的某些方面。在系统的高级版本中，控制面板的功能可能会通过无线PDA或移动电话实现。

ControlPanel类(图9-5)提供了和键盘相关的行为，因此必须实现操作readKeyStroke()和decodeKey()。如果这些操作需要由其他类提供(在此例中是**WirelessPDA**和**MobilePhone**)，定义如图9-5所示的接口是非常有用的。名为**KeyPad**的接口表示为<<Interface>>构造型，或用一个带有标识且用一条线和类相连的小圆圈表示，定义接口时并没有实现键盘行为所必需的属性和操作集合。

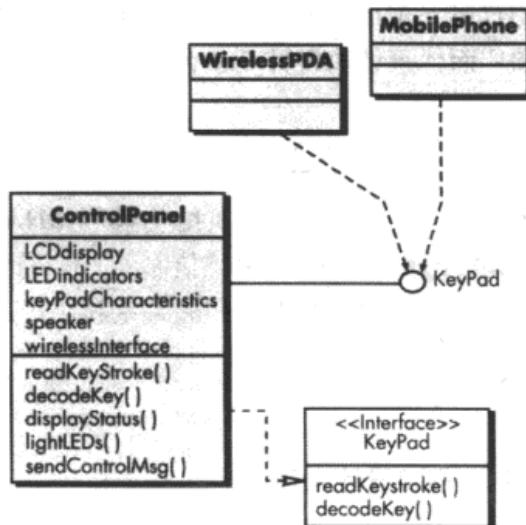


图9-5 ControlPanel(控制面板)
的UML接口设计表示

“在设计简单得完全不会错的东西时常犯的一个错误是低估傻瓜的聪明。”

——Douglas Adams

277

在控制面板的右边带有三角箭头的虚线(图9-5)表示**ControlPanel**类提供了**KeyPad**操作作为其行为的一部分。在UML中，这被刻画为实现(realization)。也就是说，**ControlPanel**行为的一部分将通过实现**KeyPad**操作来实现。这些操作将被提供给那些访问该接口的其他类。

9.4.4 构件级设计元素

软件的构件级设计完全类似于某个房屋中每个房间的一组详细绘图(以及规格说明)。这

⁶ 接口特点随时间变化是很少见的。因此，设计人员应确保接口的规格说明随时更新。

些绘图描绘了每个房间内的布线和管道、电器插座和开关、水龙头、水池、浴室、浴盆、下水道、壁橱和储藏室的位置，还说明了所使用的地板、装饰以及和房间相关的任何细节。软件的构件级设计完整地描述了每个软件构件的内部细节。为此，构件级设计为所有本地数据对象定义数据结构，为所有在构件内发生的处理定义算法细节，并定义允许访问所有构件操作（行为）的接口。

“细节不仅仅是细节，细节构成设计。”

——Charles Eames

在面向对象的软件工程中，使用UML图表现的构件如图9-6所示。图中表示的构件名为**SensorManagement** (SafeHome安全功能的一部分)。虚线箭头连接了构件和名为**Sensor**的类。**SensorManagement**构件完成所有和SafeHome传感器相关功能，包括监测和配置传感器。第11章中将进一步讨论构件图。

构件的细节设计可以在很多不同的抽象级下建模。活动图可以用于表示处理逻辑，构件详细的处理流可以使用伪代码表示（类似于编程语言的表示方法，在第11章中讨论），也可以使用一些图形（例如活动图或流图）来表示。

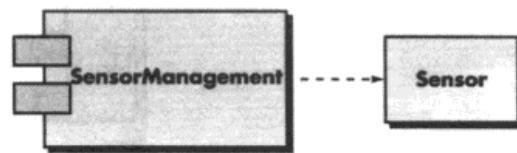


图9-6 SensorManagement的UML构件图

9.4.5 部署级设计元素

部署级设计元素指明软件功能和子系统将如何在支持软件的物理计算环境内分布。例如，SafeHome产品元素被配置为在三个主要的计算环境内运行——基于住宅的PC、SafeHome控制面板和位于CPI公司的服务器（提供基于Internet的系统访问）。

在设计过程中，开发的UML部署图以及随后的精化如图9-7所示。图中显示了三个计算环境（实际上，还可能包括传感器、摄像头和其他的环境）。图中标识出了每个计算元素中还有子系统（功能）。例如，个人计算机中有完成安全、监视、住宅管理和通信功能的子系统。此外，还设计了一个外部访问子系统以管理外界资源对SafeHome系统的访问。细化每个子系统，用以说明该子系统所实现的构件。

图9-7所示的图使用描述符形式，这意味着部署图显示了计算环境但并没有明确地说明配置细节。例如，“个人计算机”并没有进一步地明确，它是一台“Wintel”PC还是一台Macintosh、Sun工作站或Linux系统。在后面的阶段或构建开始时，应该用实例形式重新为部署图提供这些细节，明确每个实例的部署（专用的称为硬件配置）。

POINT
部署图先从描述符形式开始，这种形式使用通用的术语描述部署环境。随后，使用实例形式并显式地描述配置元素。

278

279

“时不时地放下问题，再放松一点点，这样当你重新回来工作时，你的判断将会更可靠。走开一定的距离，问题就会显得更小一些，然后一次就能看到该问题的更多部分，从而更容易看到协调和比例的不足。”

——Leonardo DaVinci

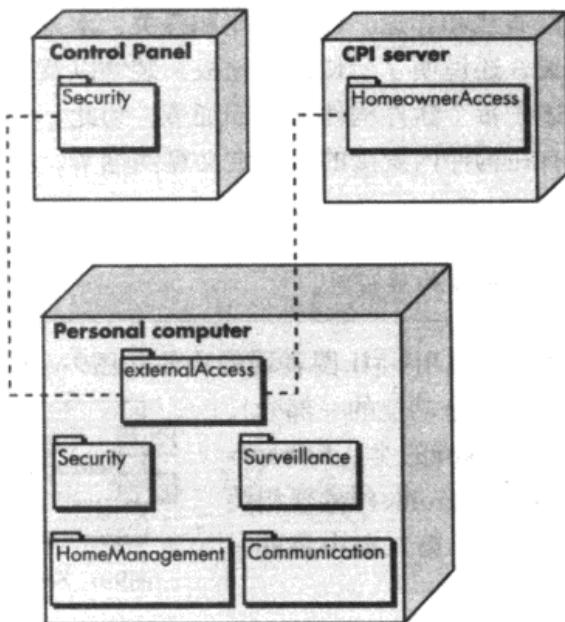


图9-7 SafeHome的UML部署图

9.5 基于模式的软件设计

WebRef

如果你需要查找某个设计（或其他）模式，可以访问www.patternsdepot.com/pages/。

任何领域的顶级设计人员都具有一种不可思议的能力，就是看出其中的模式，即刻画出问题和相应的多个模式的特点，组合这些模式就可以形成解决方案。贯穿整个设计过程，软件设计人员都应寻找每个机会重用现有的设计模式（当它们满足设计需要时）而不是创建一个全新的。

9.5.1 描述设计模式

成熟的工程学科利用成千上万的设计模式。例如，机械工程师使用两级键锁转轴作为设计模式。模式中固有的东西是属性（转轴的直径、键槽的尺寸等）和操作（例如转轴的旋转、转轴的连接）。电气工程师使用集成电路（极其复杂的设计模式）解决新问题所特有的元素。设计模式可以使用以下框内的模板[MAI03]来描述。

INFO

设计模式模板

模式名——使用简短却富有表现力的名称描述模式的本质。

含义——描述模式并说明该模式是什么。

其他名称——列出该模式的所有同义词。

动机——提供一个问题的示例。

适用性——记录能够应用该模式的特殊的设计环境。

结构——描述实现该模式所必需的类。

参与者——描述实现该模式所必需的类的职责。

协作——描述参与者如何协作以便实现其职责。

因果关系——说明“设计的影响因素”，即影响模式的因素和模式在实施时必须考虑的潜在的权衡。

相关模式——相关设计模式的交叉索引。



设计影响因素是那些解决方案的问题和属性的特点。这些特点限制了设计开发的方式。

设计模式的描述还可以考虑一系列的设计影响因素。设计影响因素描述与应用该模式的软件相关的非功能性的需求（例如维护的简易性、可移植性）。此外，影响因素定义了可能限制设计实现方式的一些约束。本质上，设计影响因素描述了使用设计模式所必需的环境和条件。模式特征（类、职责和协作）指出了设计的属性，调整这些属性可以帮助模式适应各种各样的问题[GAM95]。这些属性表现了设计的特点，可以通过搜索（例如通过数据库）找到一个恰当的模式。最终，与使用设计模式相关的指导原则指出了设计决策的结果。

280

“模式是不成熟的——意味着你通常必须自己完成并使之适应你自己的环境。”

——Martin Fowler

应该慎重选择设计模式的名称。软件复用中的一个关键问题是当存在成百上千个候选模式时，不能找到合适的可复用模式。一个有意义的模式名称将有助于查找“恰当”的模式。

9.5.2 在设计中使用模式

在整个软件设计中都可以使用设计模式。一旦开发了分析模型（第8章），设计人员可以检查待求解问题的详细表示和该问题所带来的限制。在各种抽象级上检查问题说明，以确定如下类型的设计模式中的一个或多个是否适合该问题。

对软件工程师来说，什么类型的设计模式是有效的？

体系结构模式：这些模式定义软件的整体结构，体现了子系统和软件构件之间的关系，并定义了说明体系结构元素（类、包、构件、子系统）之间关系的规则。

设计模式：这些模式解决设计中特有的元素，例如解决某些设计问题中的构件聚集、构件间的联系或影响构件到构件之间通信的机制。

习惯用语：有时被称为编码模式，这些编程语言所特有的模式通常实现了构件、特定的接口协议或构件之间通信机制的算法元素。

这些模式类型的不同不仅在于每一个模式都是在不同的抽象级上表示，还在于为软件过程中构建活动（这里是指编码）提供直接指导的抽象等级也不同。

9.5.3 框架

框架是一个代码骨架，可以使用为解决问题而设计的特定类或功能来填充这个代码骨架，使之丰满。

在某些情况下，可能需要为设计工作提供和实现相关的特定的骨架基础设施（被称作框架）。也就是说，设计人员可能选择一个“可复用的小架构，该架构为一族软件抽象及其环境提供通用的结构和行为……从而在给定的域中表明它们的协作和使用”[APP98]。

框架不是体系结构模式，而是一个带有“插入点”（也被称作钩和孔）集合的骨架，插入点使得体系结构能够适应特定的问题域。插入点使得设计人员能够在骨架内集成与问题相关的特定类。在面向对象的环境内，框架是相互合作的类的集合。

281

实质上，框架的设计人员会坚持说一个可复用的小架构适用于在某一有限的应用域中开发的所有软件。为了实现最有效，框架的使用没有变化，但可以添加附加的设计元素，即设

计人员只有通过插入点来充实、丰满框架的骨架。

9.6 小结

在需求工程的首次迭代有结论时开始设计工程，软件设计的目的是应用一系列能够引导高质量的系统或产品的开发原则、概念和实践。设计的目标是创建一个软件模型，该模型将正确地实现所有的客户需求并为那些使用软件的人带来快乐。设计工程师必须从大量可供选择的设计中筛选并最终集中于一个最适合项目共利益者需要的解决方案。

设计过程从软件的“宏观”视图向微观视图转移，后者定义了实现系统所必需的细节。设计过程开始时关注于体系结构，然后定义子系统，建立子系统之间的通信机制，识别构件，开发每个构件的详细描述，另外还要设计外部接口、内部接口和用户接口。

设计思想在软件工程的前半个世纪不断发展。它们描述计算机软件的属性（表现这些属性时不应考虑所选择的软件工程过程）；描述所使用的设计方法；或描述所使用的编程语言。

设计模型包含四种不同的元素。随着每个元素的开发，逐渐形成更全面的设计视图。体系结构元素使用各种信息以获得软件、软件子系统和构件的完整的结构表示，这些信息来自于应用域、分析模型和模式与风格的分类。接口设计元素为外部和内部的接口以及用户接口建模。构件级元素定义体系结构中的每一个模块（构件）。最后，部署级设计元素划分体系结构、构件和容纳软件的物理配置的接口。

基于模式的设计可以复用那些在过去已经被证明是成功的设计元素。在针对某个特定的应用进行评估时，应将每个体系结构模式、设计模式或习惯用语分类、全面文档化和细致地考虑。框架作为模式的扩展，为某个特定应用域内完整的子系统设计提供体系结构骨架。

282

参考文献

- [AMB01] Ambler, S., *The Object Primer*, Cambridge Univ. Press, 2nd ed., 2001.
- [APP98] Appleton, B., "Patterns and Software: Essential Concepts and Terminology," downloadable at <http://www.enteract.com/~bradapp/docs/patterns-intro.html>.
- [ARL02] Arlow, J., and I. Neustadt, *UML and the Unified Process*, Addison-Wesley, 2002.
- [BEL81] Belady, L., Foreword to *Software Design: Methods and Techniques* (L.J. Peters, author), Yourdon Press, 1981.
- [FOW00] Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 2000.
- [GAM95] Gamma, E., et al., *Design Patterns*, Addison-Wesley, 1995.
- [GAR95] Garlan, D., and M. Shaw, "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering*, vol. I (V. Ambriola and G. Tortora, eds.), World Scientific Publishing Company, 1995.
- [GRA87] Grady, R. B., and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
- [JAC75] Jackson, M. A., *Principles of Program Design*, Academic Press, 1975.
- [LIE03] Lieberherr, K., "Demeter: Aspect-Oriented Programming," May 2003, available at: <http://www.ccs.neu.edu/home/lieber/LoD.html>.
- [MAI03] Maiorilollo, J., "What Are Design Patterns and Do I Need Them?," developer.com, 2003, available at <http://www.developer.com/design/article.php/1474561>.
- [MCG91] McGlaughlin, R., "Some Notes on Program Design," *Software Engineering Notes*, vol. 16, no. 4, October 1991, pp. 53-54.
- [MYE78] Myers, G., *Composite Structured Design*, Van Nostrand, 1978.
- [OMG01] Object Management Group, *OMG Unified Modeling Language Specification*, version 1.4, September 2001.
- [PAR72] Parnas, D. L., "On Criteria to be used in Decomposing Systems into Modules," *CACM*,

- vol. 14, no. 1, April 1972, pp. 221–227.
- [ROS75] Ross, D., J. Goodenough, and C. Irvine, "Software Engineering: Process, Principles and Goals," *IEEE Computer*, vol. 8, no. 5, May 1975.
- [SCH02] Schmuller, J., *Teach Yourself UML*, SAMS Publishing, 2002.
- [SHA96] Shaw, M., and D. Garlan, *Software Architecture*, Prentice-Hall, 1996.
- [STA02] "Metaphor," *The Stanford HCI Learning Space*, 2002, <http://hci.stanford.edu/hcils/concepts/metaphor.html>.
- [STE74] Stevens, W., G. Myers, and L. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, 1974, pp. 115–139.
- [WIR71] Wirth, N., "Program Development by Stepwise Refinement," *CACM*, vol. 14, no. 4, 1971, pp. 221–227.

习题与思考题

- 9.1 如果软件设计不是程序（它确定不是），那么它是什么？
- 9.2 当你“编写”程序时你设计软件吗？软件设计和编码有什么不同吗？ [283]
- 9.3 用你自己的话说明软件体系结构。
- 9.4 访问一个设计模式存储库（在Web上）并花上几分钟浏览这些模式。选择一个并用在你的类中。
- 9.5 举出三个数据抽象的例子和可以用来操作这些数据抽象的过程抽象的例子。
- 9.6 应用“逐步求精方法”为下列程序中的一个或多个开发三种不同级别的过程抽象：(a) 开发一个账单打印程序，给出一个数字的美元总价，该程序将用填写账单所必需的正常文字打印出该总价；(b) 为某个超越方程迭代求解；(c) 为一个操作系统开发一个简单的任务调度算法。
- 9.7 应在什么时候把模块设计实现为单块集成软件？如何实现？性能是实现单块集成软件的唯一理由吗？
- 9.8 为你每天都能遇到的东西（例如家用电器、汽车、设备）推荐一个设计模式，使用在9.5节中提供的模板完整地记录该模式。
- 9.9 讨论作为有效模块化属性的信息隐蔽的概念和模块独立性概念之间的联系。
- 9.10 是否存在一种情况：复杂问题需要较少的工作去解决？这样的情况对模块化观点有什么影响？
- 9.11 柔合性的概念如何与软件可移植性相关联？举例支持你的论述。
- 9.12 检查为设计所描述的任务集，质量评估应在任务集中的什么地方进行？如何实施？
- 9.13 对极限编程进行一些研究，就敏捷软件开发过程写一篇关于使用重构方法的简短论文。
- 9.14 我们如何评估软件设计的质量？

推荐读物与阅读信息

Donald Norman编写了两本书（《The Design of Everyday Things》，Doubleday，1990，和《The Psychology of Everyday Things》，HarperCollins，1988），这两本书已经成为设计文学中的经典著作，任何人想设计被人类所使用的任何东西，都“必须”阅读这些著作。Adams（《Conceptual Blockbusting》，第3版，Addison-Wesley，1986）编写的著作对那些希望扩宽思路的设计人员极为重要。最后，Polya在编写的一本经典著作（《How to Solve It》，Princeton University Press，第2版，1988）中提供了通用的问题解决流程，在软件设计人员面对复杂问题时能够提

供帮助。

遵循同样的传统，Winograd等人（《Bringing Design to Software》，Addison-Wesley，1996）讨论了有用的、无用的软件设计及其理由。Wixon和Ramsey编写了一本令人着迷的书（《Field Methods Casebook for Software Design》，Wiley，1996），书中建议使用领域搜索方法（和人类学家所使用的那些方法非常类似）理解最终用户如何完成所必须完成的工作，并为软件设计提供指导以满足用户的需要。Beyer和Holtzblatt（《Contextual Design: A Customer-Centered Approach to Systems Designs》，Academic Press，1997）提供了软件设计的另一种视图，即将客户/用户集成到软件设计流程的各个方面。

McConnell（《Code Complete》，Microsoft Press，1993）就设计高质量的计算机软件的实践方面进行了非常精彩的论述。Robertson（《Simple Program Design》，第3版，Boyd and Fraser Publishing，1999）介绍性地论述了软件设计，对那些刚开始这方面学习的人非常有益。Fowler和他的同事（《Refactoring: Improving the Design of Existing Code》，Addison-Wesley，1999）讨论了软件设计增量优化的技术。

在过去的10年中，已经编写了大量基于模式设计的书籍可供软件工程师选择。Gamma和他的同事[GAM95]曾经编写了一本关于这个主题的开创性的书籍。Douglass（《Real-Time Design Patterns》，Addison-Wesley，2002）、Metsker（《Design Patterns Java Workbook》，Addison-Wesley，2002）、Juric等人（《J2EE Design Patterns Applied》，Wrox Press，2002）、Marinescu和Roman（《EJB Design Patterns》，Wiley，2002）以及Shalloway和Trott（《Design Patterns Explained》，Addison-Wesley，2001）等书中，讨论了在特定应用和语言环境下的设计模式。此外，建筑师Christopher Alexander所编写的经典著作（《Notes on the Synthesis of Form》，Harvard University Press，1964，和《A Pattern Language: Towns, Buildings, Construction》，Oxford University Press，1977）是每个希望全面了解设计模式的软件设计师必须阅读的。

关于设计工程的信息在Internet上有大量的各种资源。在SEPA网站（<http://www.mhhe.com/pressman>）上可以找到和软件设计和设计工程相关的最新WWW参考信息。

第10章 进行体系结构设计

要点浏览

概念：体系结构设计描述了建立计算机系统所需的数据结构和程序构件。它需要考虑系统采取的体系结构风格，系统组成构件的结构、性质，以及所有体系结构构件之间的相互关系。

人员：尽管软件工程师能够设计数据和体系结构，但是在建造大型复杂系统的时候，数据和体系结构的设计往往由专家来完成。数据库或者数据仓库设计者为系统创建数据体系结构。“系统体系结构设计师”为系统工程和软件需求分析中导出的需求选择合适的体系结构风格。

重要性：人们不能在没有图纸的情况下建房子，同样也不能通过勾画出房子的管道布局而开始绘制房屋的蓝图。

在开始考虑细节之前，需要关注整体视图——房子本身。这就是体系结构设计需要做的事情——它为你提供整体的视图并保证得到正确的理解。

步骤：体系结构设计始于数据设计，然后导出系统体系结构的一个或者多个表示。对可选的体系结构风格或模式进行分析，以导出最适合于客户需求和质量属性的结构。一旦选定，使用体系结构设计方法对体系结构进行精化。

工作产品：在体系结构设计过程中，将创建一个包括数据架构和程序结构的体系结构模型。此外，还需描述构件的性质以及交互关系。

质量保证措施：在每个阶段，对于软件设计的工作产品要进行评审，以确保工作产品与需求以及工作产品彼此之间的清晰性、正确性、完整性和一致性。

关键概念
ADL
原始模型
体系结构（架构）
评估
复杂度
环境图
定义
设计
映射
模式
ATAM
构件
数据设计
分解
风格

设计通常被描述为一个多步过程，其主要任务是从需求信息中综合出数据的表示、程序结构、接口特征和过程细节。Freeman[FRE80]对此概念描述如下：

设计是一项关注于如何做出重要决策的活动，这些决策通常都与结构有关。设计和编程都要考虑抽象信息的表示和处理顺序，但是在详细程度上两者有很大的不同。设计就是构建一个一致的、良好计划的程序表示，主要描述高层各部分的相互关系和低层所包括的逻辑操作……

正如我们在第9章提到的，设计是由信息驱动的。软件设计方法都是通过仔细考虑分析模型的三个域而得到的。因此，信息、功能和行为三个域是创建软件设计的指南。

本章将讨论建立设计模型数据和体系结构层的“内聚的、良好设计的表示”所需的方法。目标是提供一种导出体系结构设计的系统化方法，而体系结构设计是构建软件的初始蓝图。

10.1 软件体系结构

Shaw和Garlan[SHA96]在他们划时代的著作中以如下方式讨论了软件的体系结构：

从第一个程序被划分成模块开始，软件系统就有了体系结构。同时，程序员已经开始负责模块间的交互和模块装配的全局属性。从历史的观点看，体系结构隐含了很多内容——实现的偶然事件或先前遗留系统。好的软件开发人员经常采用一个或者多个体系结构模式作为系统组织策略，但是他们只是非正式地使用这些模式，并且在最终系统中没有将这些模式清楚地体现出来。

现在，有效的软件体系结构及其明确的描述和设计，已经成为软件工程领域中重要的主题。

“系统的体系结构是一个关于系统形式和结构的综合框架，包括系统构件和构件的整体。”
——Jerrold Grochow

10.1.1 什么是体系结构



当我们谈论建筑物的体系结构时，脑海中会出现很多不同的属性。在最简单的层面上，我们会考虑物理结构的整体形状。但是在实际中，体系结构还包含更多的方面。它是各种建筑构件集成一个有机整体的方式，是建筑适合其所在环境并与其邻近的其他建筑相互协调的方式，它反映了建筑满足规定的用途和满足主人要求的程度。体系结构是对结构的一种美学观感（建筑的视觉效果）——纹理、颜色和材料结合在一起创建外部门面和内部“居住环境”。体系结构还是很多微小的细节——灯光设计、地板类型、壁挂布置，等等。总而言之，它是艺术。

287

但是，什么是软件体系结构？Bass、Clements 和Kazman[BAS03]对于这个难懂的概念给出了如下的定义：

一个程序和计算系统软件体系结构是指系统的一个或者多个结构。结构中包括软件的构件，构件的外部可见属性以及它们之间的相互关系。

体系结构并非可运行软件。确切地说，它是一种表达，使软件工程师能够：(1) 分析设计在满足规定需求方面的有效性；(2) 在设计变更相对容易的阶段，考虑体系结构可能的选择方案；(3) 降低与软件构造相关联的风险。

“体系结构忙中建，闲时则悔。”

——Barry Boehm

WebRef

关于软件体系结构的很多有用信息可以从
www2.umassd.edu/SECenter/SAResources.html获得。

上面的定义强调在任意体系结构表述中“软件构件”的角色。在体系结构设计的环境中，软件构件可以简单到程序模块或者面向对象的类，也可以扩充到包含数据库和能够完成客户与服务器网络配置的“中间件”。

在本书中，软件体系结构的设计考虑了设计金字塔（图9-1）中的两个层次——数据设计和体系结构设计。在前面的讨论中，数据设计使我们表示出传统系统中体系结构的数据构件和面向对象系统中类的定义（封装了属性和操作），体系结构设计则主要关注软件构件的结构、属性和交互作用。

10.1.2 为什么体系结构如此重要

在一本关于软件体系结构的书中，Bass和他的同事[BAS03]给出了软件体系结构之所以重要的三个关键原因：



体系结构模型提供了系统的整体视图，允许软件工程师把系统作为一个整体来检验。

- 软件体系结构的表示有助于对计算机系统开发感兴趣的各方（共利益者）开展交流。
- 体系结构突出了早期设计决策，这些决策对随后的所有软件工程工作有深远的影响，同时对系统作为一个可运行实体的最后成功有重要作用。
- 体系结构“构建了一个相对小的，易于理解的模型，该模型描述了系统如何构成以及其构件如何一起工作”[BAS03]。

体系结构设计模型和包含其中的体系结构模式都是可以传递的，也就是说，体系结构的风格和模式（10.3.1节）可以被应用于其他系统的设计中，并且表示了一组使软件工程师能以可预见的方式描述体系结构的抽象。

288

10.2 数据设计

数据设计是把在分析模型（第8章）中定义的数据对象转化成软件构件级的数据结构，并且在必要时转化为应用程序级的数据库体系结构。在某些情况下，必须为一个新系统专门设计和建立数据库。然而，有些时候系统仍然使用现有的一个或多个数据库。

10.2.1 体系结构级的数据设计

当今时代，大大小小的业务均充斥着数据，甚至一个中型规模企业拥有为多个应用系统提供服务的几十个数据库（包含数千亿字节的数据）也不是什么新鲜事。问题在于如何从这样庞大的数据环境中提取有用的信息，特别当需要的信息是功能交叉时（例如，仅当特定的市场数据与产品工程数据交叉相关时才能获得的信息）。

“数据仓库与数据垃圾堆之间的差别在于数据的质量。”

——Jarrett Rosenberg

WebRef

有关数据仓库技术方面的信息可以从下面的网址获取：www.datawarehouse.com。

为了解决上述挑战，IT界开发出了数据挖掘技术，也称为数据库中的知识发现（knowledge discovery in database，KDD），该技术遍历现有的数据库以试图抽取出合适的业务级信息。然而，多个数据库（它们的结构不同，其中包含的细节程度不同）的存在和其他多种因素的影响，致使在现有的数据库环境下进行数据挖掘比较困难。另一种可选的解决方案称为数据仓库（data warehouse），该技术为数据体系结构增加了一个附加层。

数据仓库是一个独立的数据环境，它不直接和日常的应用系统结合，但包含了某业务使用的所有数据[MAT96]。从某种意义上讲，数据仓库是一个庞大的、独立的数据库，它能够访问存于数据库中的数据，为某项业务所需要的一组应用系统服务。

关于数据结构、数据库和数据仓库设计的详细讨论参见相关专题的书籍（如[DAT00]、[PRE98]和[KIM98]）。感兴趣的读者可以参见本章后面的“推荐读物与阅读信息”。

289

SOFTWARE TOOLS

数据挖掘和数据仓库

目的：数据挖掘工具有助于识别那些描述特定数据对象或一组数据对象的属性之间的重要联系。数据仓库工具有助于数据仓库数据模型的设计。

机制：一般说来，挖掘工具以大量数据作为输入并且允许用户查询数据，从而使用户更好地理解各种数据项间的联系。用于设计的数据仓库工具可以提供实体联系或其他建模能力。

代表性工具¹**数据挖掘：**

Business Objects，由Business Objects开发，SA（www.business.object.com）是用于支持数据集成、查询、报告、分析和解析的数据设计工具。

SPSS，由SPSS开发（www.spss.com），提供了大量的统计功能，能够对大数据集进行分析。

数据仓库：

Industry Warehouse Studio，由Sybase（www.sybase.com）开发，提供了一个数据仓库基础设施包，它能够进行跳跃式数据仓库设计。

IFW Business Intelligence Suite，由Modelware（www.modelwarepl.com）开发，是一套模型、软件工具和提供快速数据仓库设计和数据超市设计与实现的数据库设计集合。

在“数据仓库信息中心”（www.dwinfocenter.org）可以找到关于数据挖掘和数据仓库工具和资源的完整列表。

10.2.2 构件级的数据设计

构件级的数据设计关注于那些被一个或者多个软件构件直接访问的数据结构的表示。Wasserman[WAS80]提出了一些原则，这些原则能够用来说明和设计这种数据结构。在实际应用中，数据设计在创建分析模型期间就已经开始了。考虑到需求分析和设计经常会重叠，我们主要考虑以下数据规格说明原则（摘自[WAS80]）：

可应用于数据设计的原则有哪些？

1. 应用于功能和行为的系统分析原则也可应用于数据。同样应该开发和评审数据流和数据内容的表示，标识数据对象，还应该考虑其他可选的数据组织结构，评估数据模型对软件设计的影响。
2. 标识所有数据结构及其完成的操作。设计一个高效的数据结构，必须考虑其上的操作。把属性和操作封装在一个类中满足这个原则。
3. 应该建立定义数据对象内容的机制，并且用于定义数据及其操作。类图（第8章）定义包含在类中的数据项（属性）和应用到这些数据项上的方法（操作）。
4. 低层的数据设计决策应该延迟到设计过程的后期。数据设计可以采用逐步求精的过程，也就是说，所有的数据组织可以在需求分析阶段定义，在数据设计工作中精化，在构件级设计阶段刻画细节。

¹ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。在大多数情况下，工具的名字由各自的开发者注册为商标。

5. 只有那些直接使用数据结构内部数据的模块才能够看到该数据结构的表示。信息隐蔽概念以及相关的耦合概念（第9章）为软件设计质量的评估提供了依据。

6. 应该开发一个由有用的数据结构及其操作组成的库。类库即可实现这个目标。

7. 软件设计和程序设计语言应该支持抽象数据类型的规格说明和实现。如果没有办法对所选用于实现的编程语言中的结构进行直接说明，那么复杂数据结构的实现将变得非常困难。

这些原则为构件级数据设计方法提供了基础，可将这些方法纳入到分析和设计活动中。

10.3 体系结构风格和模式

当一个建筑师用短语“殖民式中厅”(center hall colonial)来描述某房子时，大多数熟悉美国房子的人将能够获得对房子的整体画面，对其建筑主平面图可能像什么样子也会有所了解。建筑师使用体系结构风格（建筑风格）作为描述机制，将该房子和其他风格（如A框架、Cope Cod等）的房子区分开来。但是更重要的是，体系结构风格也是建筑的样板。必须进一步规定房子的细节，具体说明它的最终尺寸，进一步给出定制的特征，确定建筑材料等。实际上是建筑风格（殖民式中厅）指导了建筑师的工作。

“每位艺术家的思想背后都蕴涵着某种体系结构的模式或者类型。”——G. K. Chesterton

什么是体系
结构风格？

为计算机系统建造的软件也展示了众多体系结构风格中的一种。每种风格描述一种系统类别，包括：(1) 一组构件（比如：数据库、计算模块）完成系统需要的某种功能；(2) 一组连接器，它们能使构件间实现“通信、合作和协调”；(3) 约束，定义构件如何集成成为一个系统；(4) 语义模型，它能使设计者通过分析系统的构成成分的性质来理解系统的整体性质[BAS03]。

291

一种体系结构风格就是一种加在整个系统设计上面的变换。它的目的就是为系统的所有构件建立一个结构。在对已有体系结构进行再工程（第31章）时，强制采用一种体系结构风格会导致软件结构的根本性改变，包括对构件功能的再分配[BOS00]。

与体系结构风格一样，体系结构模式也对体系结构的设计施加一种变换。然而，体系结构模式与体系结构风格在许多基本方面存在不同：(1) 体系结构模式涉及的范围要小一些，它更多集中在体系结构的某一局部而不是体系结构的整体；(2) 模式在体系结构上施加规则，描述了软件是如何在基础设施层次（例如，并发）[BOS00]上处理某些功能性方面的问题；(3) 体系结构模式倾向于在系统结构的环境中处理特定的行为问题，例如，一个实时应用系统如何处理同步和中断。模式可以与体系结构风格结合起来，用于建立整个系统结构的外形。在下面的小节中，我们将讨论经常用到的软件体系结构风格和模式。

WebRef

基于属性的体系
结构风格(ABAS)
可以用作软件体
系结构的构造
块。相应信息可
以从www.sei.
cmu.edu/ata/abas.
html获得。

10.3.1 体系结构风格的简单分类

在过去的50年中，尽管已经创建了数百万的计算机系统，但是，绝大多数都可以被归类为少数几种体系结构风格之一（见[SHA96]、[BUS96]、[BAS03]）：

以数据为中心的体系结构。数据存储（如文件或数据库）驻留在这种体系结构的中心，其

他构件会经常访问该数据存储，并对存储中的数据进行更新、增加、删除或者修改。图10-1描述了一个典型的以数据为中心的体系结构风格，其中客户软件访问中心存储库。在某些情况下存储库是被动的，也就是说，客户软件独立于数据的任何变化或其他客户软件的动作而访问数据。该方法的一个变种是将中心存储库变换成“黑板”，当用户感兴趣的数据发生变化时，它将通知客户软件。

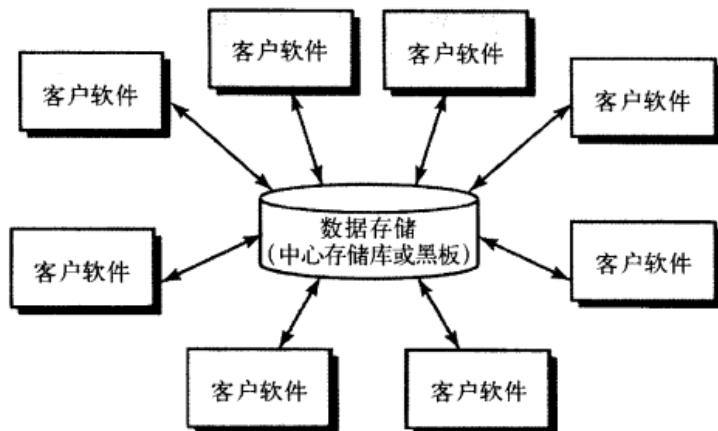


图10-1 以数据为中心的体系结构

以数据为中心的体系结构提升了可集成性（integrability）[BAS03]，也就是说，现有的构件可以被修改而且新的客户构件可以加入到系统结构之中，而无需考虑其他的客户（因为客户构件是独立运作的）。另外，数据可以在客户间通过“黑板”机制传送（即黑板构件负责协调信息在客户间的传递），客户构件独立地执行过程。

292 数据流体系结构。当输入数据经过一系列的计算和操作构件的变换形成输出数据时，可以应用这种体系结构。管道和过滤器结构（图10-2）拥有一组被称为过滤器的构件，这些构件通过管道连接，管道将数据从一个构件传送到下一个构件。每个过滤器独立于其上游和下游的构件而工作，过滤器的设计要针对某种形式的数据输入，并且产生某种特定形式的数据输出（到下一个过滤器）。然而，过滤器没有必要了解与之相邻的过滤器的工作。

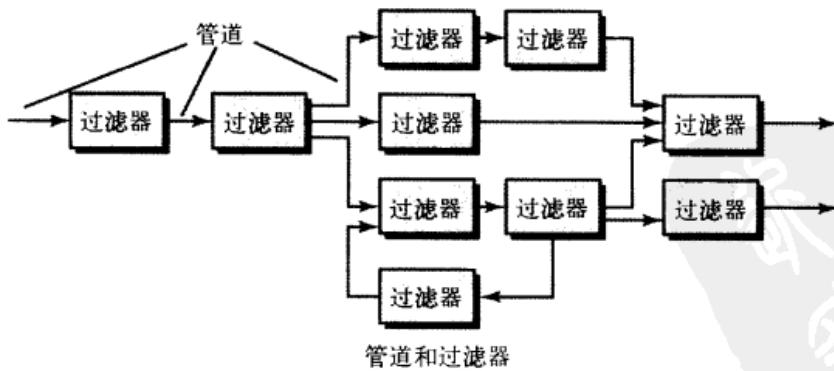


图10-2 数据流体系结构

“设计模式和风格的运用在工程学科中无处不在。” ——Mary Shaw和David Garlan

293

如果数据流退化成单线的变换，则称为批序列（batch sequential）。这种结构接收一批数

据，然后应用一系列连续的构件（过滤器）变换它。

调用和返回体系结构。该体系结构风格能够让软件设计师（系统架构师）设计出一个相对易于修改和扩展的程序结构。在此类体系结构中，存在两种子风格[BAS03]：

- **主程序/子程序体系结构。**这种传统的程序结构将功能分解为一个控制层次，其中“主”程序调用一组程序构件，这些程序构件又去调用别的程序构件。图10-3描述了该种系统结构。
- **远程过程调用体系结构。**主程序/子程序体系结构的构件分布在网络的多个计算机上。

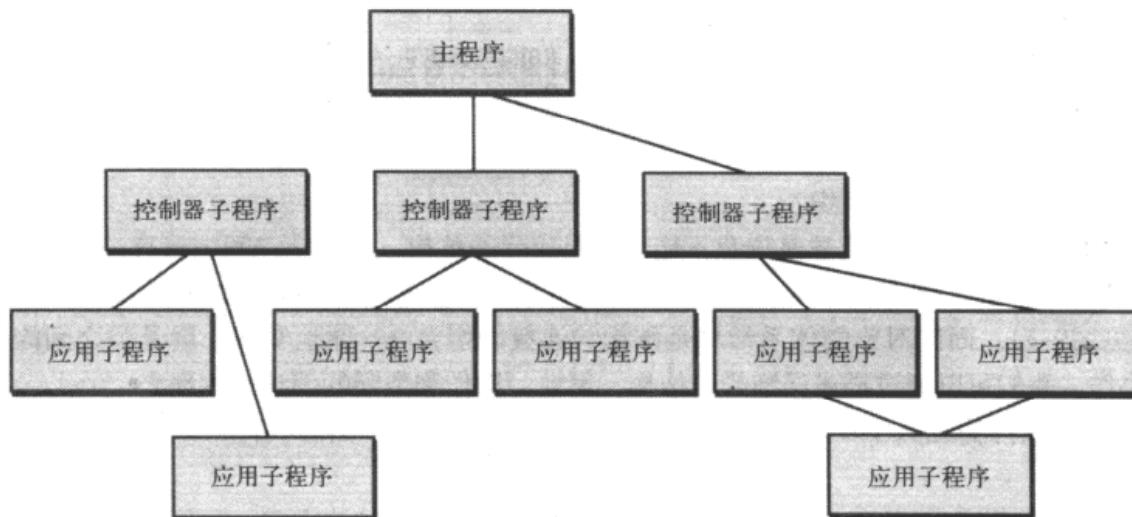


图10-3 主程序/子程序体系结构

面向对象体系结构。系统的构件封装了数据和必须应用到该数据上的操作，构件间通过信息传递进行通信与合作。

层次体系结构。层次体系结构的基本结构如图10-4所示。其中定义了一系列不同的层次，每个层次各自完成操作，这些操作不断接近机器的指令集。在最外层，构件完成用户界面的操作；在最内层，构件完成与操作系统的连接；中间层提供各种实用程序服务和应用软件功能。

这些体系结构风格仅仅是软件设计师可用风格中的一小部分²。一旦需求工程揭示了待构建系统的特征和约束，就可以选择最适合这些特征和约束的体系结构风格或者风格的组合。在很多情况下，会有多种风格是适合的，需要对可选的体系结构风格进行设计和评估。例如，在很多数据库应用中，层次体系结构（适合大多数系统）可以与以数据为中心的体系结构结合起来。

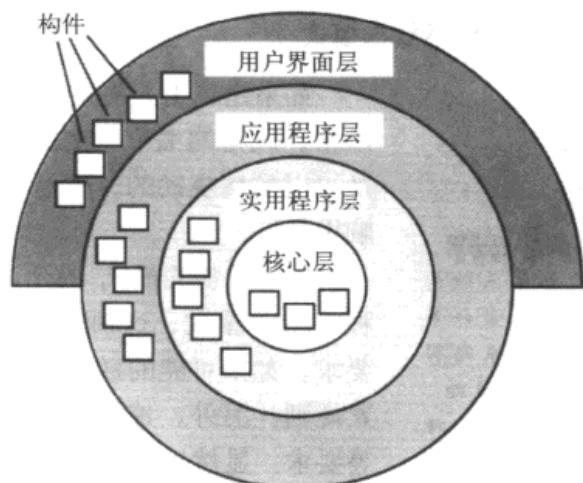


图10-4 层次体系结构

² 参见[BOS00]、[HOF00]、[BAS03]、[SHA97]、[BUS96]和[SHA96]等文献可以找到对体系结构和模式的更详细的讨论。

选择一种体系结构风格

[场景] Jamie的房间，设计模型还在继续进行。

[人物] Jamie和Ed，SafeHome软件工程团队的成员。

[对话]

Ed (皱着眉)：我们已经用UML设计了安全功能模型……类、关系等都是其中的基本材料，所以我觉得面向对象体系结构应该是正确的办法³。

Jamie：但是……

Ed：但是……我对于面向对象体系结构理解起来有些困难，我比较熟悉调用和返回体系结构——一种传统的过程层次。但是面向对象……我不了解，它看起来属于无组织的一类。

295

Jamie (微笑)：无组织？

Ed：是的……我的意思是说 I 不能想像出实际的结构，在设计空间中只有设计类。

Jamie：哦，那不对。存在类的层次……想想我们为**FloorPlan**对象设计的层（聚集）(参见图9-3)。面向对象的体系结构是结构与连接的组合——你也知道，就是类之间的相互协作。我们可以通过描述详细的类结构、属性、操作和类间的消息来体现它。

Ed：我打算花一个小时制定一个调用和返回体系结构，然后我再考虑面向对象体系结构。

Jamie：Doug对此不会有什问题，他说我们应该考虑其他方案。顺便说一句，这两种体系结构彼此结合是绝对没有问题的。

Ed：我知道了。

10.3.2 体系结构模式



软件体系结构可能包含许多涉及体系结构模式的问题，如并发性、持久性和分布性。

如果建筑工人决定建构一个“殖民式中厅”，那么只能应用一种体系结构风格。风格的细节（例如，壁炉的数量、房屋外观、门和窗户的布置）内容是可以酌情变动的，但是一旦确定了房子的整体体系结构，这个风格就会影响设计⁴。

体系结构的模式⁵有所不同。例如，房子（以及房子的建筑风格）采用一种**Kitchen**模式，这种**Kitchen**模式规定了厨房基本用具的放置、水池、橱柜等要求，如果可能的话，也规定了房间中与完成做饭流程相关的这些厨具的布置规则。另外，该模式还可能指明柜台面、灯、墙上的插座、中心岛、地板等要求。显然，对于厨房有不止一种的设计，但是每种设计都应该在**Kitchen**

³ 对于SafeHome的体系结构，我们可以在一个比上述介绍的体系结构更高的层次上来考虑。SafeHome有很多种子系统——住宅监测功能、公司的监控站点，以及运行在PC上的子系统。在这些子系统中也普遍存在着并发过程（例如，那些监控传感器）和事件处理。在系统或产品建立过程中，会在此层次上做出一些体系结构上的决策（第6章），但是软件工程中的体系结构设计必须考虑上述问题。

⁴ 这将暗示着有一个大厅（foyer）和走廊（hallway），房间将被布置到大厅的左右两侧，房子将有两层（或更多），卧室将在楼上等。一旦决定采用殖民式中厅风格，这些规则将会影响到决策。

⁵ 需要注意的是对这个术语还没有达成普遍一致。一些人（例如，[BUS96]）将术语风格（style）和模式（pattern）作为同义词使用，而有些人（如本节中这样）则精细地区分这两个词。

模式提供的解决方案环境下来构思完成。

正如我们前面提到的，软件的体系结构模式定义了处理系统某些行为特征的方法。Bosch[BOS00]定义了一系列的体系结构模式域。在下面的内容中将介绍这方面的典型例子。

并发性：很多应用系统必须以一种模拟并行的方式来操作多个任务（即，一个单独的处理器管理多个并行任务或构件时就会发生这种情况）。在一个应用系统中有很多不同的方法处理并发性，而且每种方法都可以由不同的体系结构模式来呈现，例如，一种方法是使用“操作系统进程管理”模式，该模式提供了一些内置的操作系统特征，这些特征允许构件并发执行。这个模式同时还结合了操作系统中那些管理进程通信、调度的功能以及其他完成并发所需要的功能。还有一种方法是在应用层上定义一个任务调度器。“任务调度器”模式包括一组含有`tick()`操作的活动对象[BOS00]。调度器定期唤醒每个对象的`tick()`操作，该操作在控制权返回调度器之前完成它负责的功能，接着调度器唤醒下一个并发对象的`tick()`操作。

296

持久性：如果数据从创建它的进程执行以来一直存在，则该数据是持久性存在的数据。持久数据被存储在数据库中或者文件里，并且可以在稍后的时间里被其他进程读取和修改。在面向对象的环境中，持久对象的概念对持久性概念做了一些扩展，所有对象属性的值、对象的状态以及其他附加信息都被存储起来，以备今后的存取和使用。一般说来，可以采用两种体系结构模式获得持久性：一个是数据库管理系统模式，该模式将DBMS的存储和存取能力用于应用系统的体系结构中；另一个是应用级的持久模式，此种模式在应用体系结构中建立了持久性特征（例如，字词处理软件管理着自己专用的文档结构）。

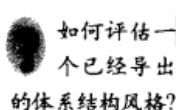
分布性：分布性问题强调系统或系统中构件在一个分布的环境中相互通信的方式。分布性问题有两个元素：(1) 实体间连接方式；(2) 实体间通信的特性。解决分布性问题最普通的体系结构模式是代理（broker）模式。代理在客户端构件和服务器构件之间充当“中间人”。客户端向代理发出一条信息（包含所有使通信有效的信息），代理完成（与服务器的）连接。CORBA（第30章）是代理模式的一个范例。

在选择前面提及的任一种体系结构模式之前，必须评估其对于应用和整个体系结构风格的适应性，也就是它的可维护性、可靠性、安全性和性能。

10.3.3 组织和求精

由于设计过程经常给软件工程师留下一系列可供选择的体系结构，建立一组用于评估所导出的体系结构设计的设计标准是非常重要的。下面的问题[BAS98]有助于对导出的体系结构风格提供深层次的考察。

297



如何评估一个已经导出的体系结构风格？

- **控制：**在体系结构中如何管理控制？是否存在一个不同的控制层次？如果是，构件在控制层次中担当什么角色？构件如何在系统中传递控制？构件间如何共享控制？控制拓扑结构（即控制呈现的几何形状）是什么样？控制是否同步或构件是否异步操作？
- **数据：**构件间如何进行数据通信？数据流是否是连续的，或数据对象是否是零散地传递给系统？数据传递的模式是什么（也就是说，数据是从一个构件传递到另一个构件，还是系统中构件可以全局共享数据）？是否存在数据构件（如黑板或中心存储库）？如果存在，它们的角色是什么？功能构件如何和数据构件交互？数据构件是被动的还是主动的（即数据构件是否主动地和系统中其他构件交互）？数据和控制如何在系统中交互？

这些问题可以帮助设计者对设计质量进行早期评估，也为更详细地对体系结构进行分析奠定了基础。

10.4 体系结构设计

在体系结构设计开始的时候，软件必须放在所处环境进行开发，也就是说，设计应该定义与软件交互的外部实体（其他系统、设备、人）和交互的特性。一般在分析模型阶段可以获得这些信息，而所有其他的信息都是在需求工程阶段获得的。一旦建立了软件的环境模型，并且描述出所有的外部软件接口，那么设计师就可以通过定义和求精实现体系结构的构件来描述系统的结构。这个过程不停地迭代，直到获得一个完善的体系结构。

“一个医生可以掩盖他的错误，但是一个建筑师只能建议他的用户种植松树加以弥补。”

——Frank Lloyd Wright

10.4.1 系统的环境表示

KEY POINT
体系结构环境描述了软件如何与其边界实体。

在第6章中，我们提到系统工程师必须对环境进行建模。系统环境图（图6-4）通过描述系统的出入信息流、用户界面和相关的支持处理等来实现这一需求。在体系结构设计层，软件架构师用体系结构环境图（architectural context diagram, ACD）对软件与外部实体交互方式进行建模。图10-5中给出了体系结构环境图的通用结构。

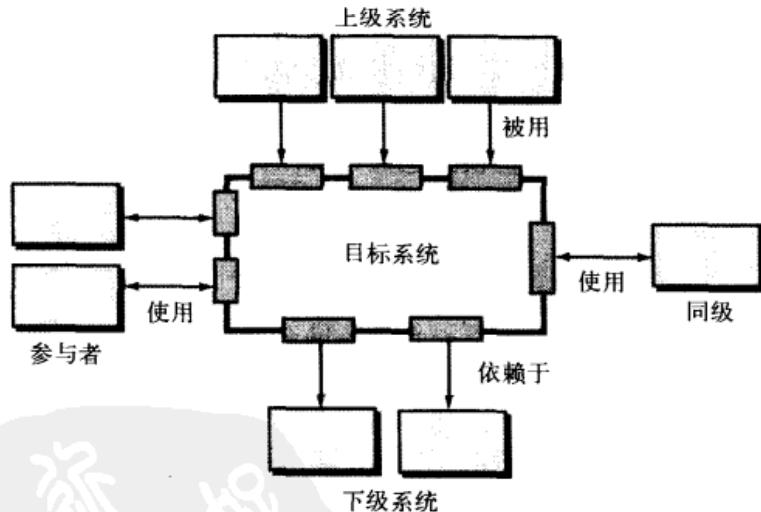


图10-5 体系结构环境图（摘自[BOS00]）

298

根据图中所示，与目标系统（即开发体系结构的系统）交互的系统可以表示为：

系统如何与其他部分进行通信？

- 上级系统——这些系统把目标系统作为某些高层处理方案的一部分。
- 下级系统——这些系统被目标系统使用，并为了完成目标系统的功能提供必要的数据和处理。
- 同级系统——这些系统在对等的基础上相互作用（例如，信息要么由目标系统和同级系统产生，要么被目标系统和同级系统消耗）。

- 参与者——是指那些通过产生和消耗必不可少的处理所需的信息，实现与目标系统交互的实体（人、设备）。

每个外部实体都通过某一接口（带阴影的小矩形）与目标系统进行通信。

为了说明ACD的使用，我们再次考虑SafeHome产品的住宅安全功能。这个SafeHome产品的控制器和基于因特网的系统对于安全功能来说都处于上一级，在图10-6中它们在上方。监视功能是一个同级系统并且在以后的产品版本中还要使用住宅安全功能（或被住宅安全功能使用）。户主和控制面板都是参与者，它们既是安全软件所用信息的生产者，又是安全软件所供信息的消费者。最后，传感器为安全软件所使用，并且在图中显示为下一级。

作为体系结构设计的一部分，必须说明图10-6中每个接口的细节。目标系统所有的流入和流出数据必须在这个阶段标识出来。

299

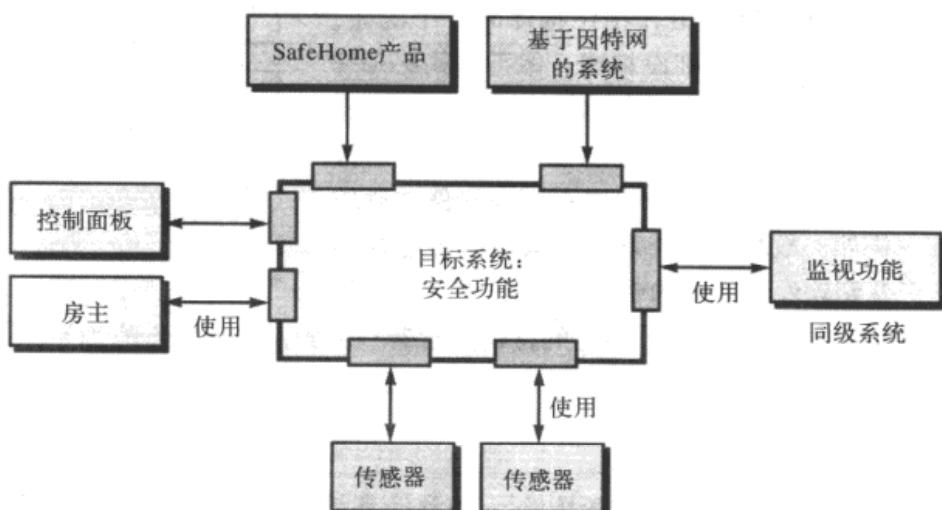


图10-6 SafeHome安全功能的体系结构环境图

10.4.2 定义原始模型



原始模型（archetype）是一个类或者一个模式，描述了一个目标系统体系结构设计的核心抽象。一般来讲，只需要设计相对较小的原始模型集合，即使系统相对比较复杂。目标系统的体系结构由这些原始模型组成，这些原始模型表示体系结构中稳定的元素，但是这些元素基于系统行为可用多种方式加以说明。

在很多情况下，可以通过检验作为分析模型一部分的分析类来获得原始模型。继续关于SafeHome住宅安全功能的讨论，我们可能会定义如下的原始模型：

结点：表示住宅安全功能的输入和输出元素的内聚集，例如，一个结点可能由如下元素构成：(1) 各种传感器；(2) 多种警报指示器。

探测器：对所有为目标系统提供信息的传感设备的一个抽象。

指示器：表示当警报条件发生时，对所有报警机械装置（例如警报汽笛、闪灯、响铃）的一个抽象。

控制器：对允许结点发出警报或者撤销警报的机械装置的抽象。如果控制器安装在网络上，那么它们应该具有相互通信的能力。

图10-7给出了使用UML符号描述上述每一个原始模型的结果。

300

考虑那些构成体系结构基础的原始模型，但是随着体系结构设计的进行，这些抽象必须被进一步精化，例如，探测器可以被精化为一个传感器的类层次。

10.4.3 将体系结构精化为构件



软件体系结构构件主要来自于三个方面：应用领域、基础设施领域和界面领域。由于分析模型不涉及基础设施，所以需要分配充分的时间对其进行仔细考虑。

当软件体系结构精化为构件时，系统的结构开始显现。但是，如何选择这些构件呢？为了回答这个问题，体系结构设计师先从分析模型⁶中所描述的类开始。这些分析类表示那些软件体系结构中必定涉及的应用（业务）领域内的实体。因此，应用领域是导出和精化构件的一个源泉。另一源泉是基础设施域。体系结构必须容纳很多基础设施构件使应用构件能够运作，但是这些基础设施构件与应用领域没有业务联系，例如，内存管理构件、通信构件、数据库构件和任务管理构件往往归并到软件体系结构中。

体系结构环境图中（10.4.1节）描述的接口隐含着一个或者多个特定的构件，这些构件处理穿过接口的数据。在某些情况下（例如，图形用户界面），需要设计一个完整的包含众多构件的子系统体系结构。

301

“软件系统的结构提供一种‘生态环境’，在这个环境里代码诞生、成长和死亡。一个经过良好设计的栖息地考虑了软件系统需要的所有构件的成功进化。” ——R. Pattis

继续SafeHome住宅安全功能的例子，我们可以定义完成下列功能的顶层构件集合：

- 外部通信管理——协调安全功能与外部实体（如基于Internet的系统、外部报警通知）的通信。
- 控制面板处理——管理所有控制面板功能。
- 探测器管理——协调对系统所有探测器的访问。
- 警报处理——审核所有报警条件并执行相应动作。

每一个顶层构件都必须经过反复的精化迭代，然后被安置在SafeHome体系结构中。为每个构件都定义相应的设计类（包含必要的属性和操作）。然而，需要注意的是，在进行构件级设计之前，不要说明所有属性和操作的设计细节（第11章）。

图10-8描述了整个的体系结构（由UML构件图表示）。处理SafeHome图形用户界面和Internet接口的构件的事务一进入，就被外部通信管理获得。这个信息由用于选择合适产品功能（安全功能）的SafeHome执行者构件来管理。控制面板处理构件与户主交互来实现安全功能的安装/解除（arm/disarm）。探测器管理构件选取探测器检测报警条件，而当检测到警报时，警报处理构件将产生相应的输出。

302

⁶ 如果选择一种传统的方法（非面向对象），则可以从数据流模型中导出构件。我们将在10.6节讨论这种方法。

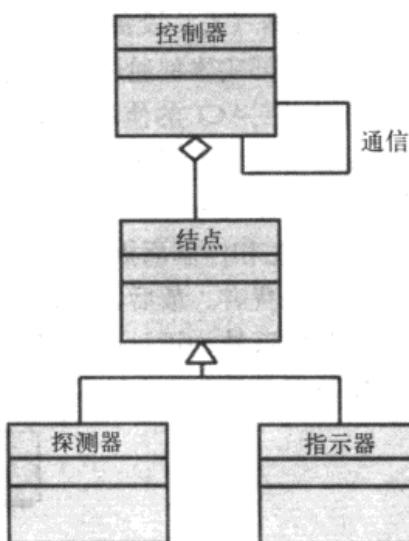


图10-7 SafeHome安全功能原始模型的UML关系图（摘自[BOS00]）

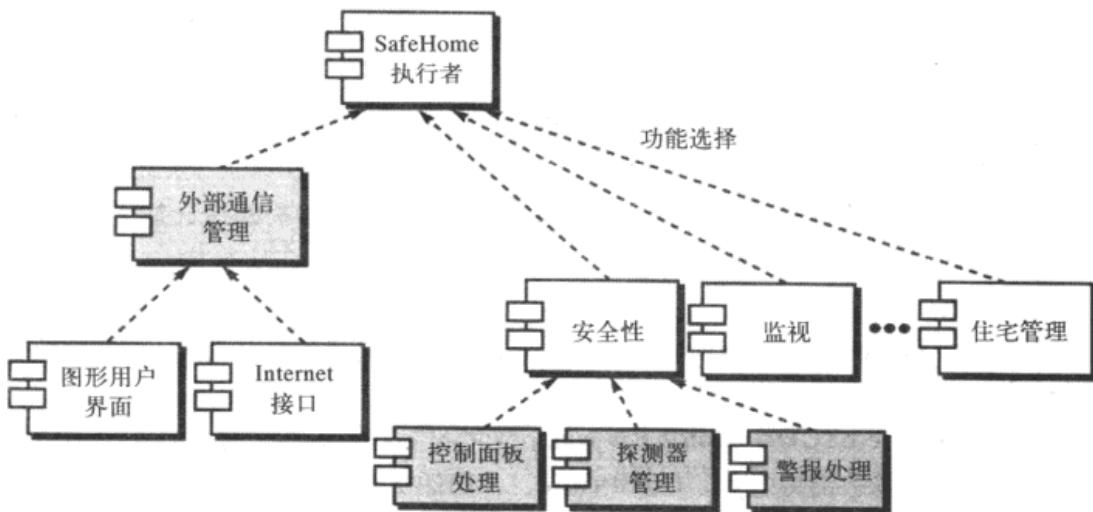


图10-8 带有高层构件的SafeHome整体体系结构

10.4.4 描述系统实例

至此所建立起来的体系结构设计依然处于比较高的层次。系统的环境已经表示出来了，预示问题域中重要抽象的原始模型也被定义出来，系统的整个结构已经显现出来，并且主要的软件构件也都定义出来了。然而，更进一步的精化（回想一下所有的设计都是迭代的）仍旧是必要的。

为了完成体系结构设计，要开发一个体系结构的实际实例。此时，我们的用意是将体系结构应用到一个特定的问题上，目的是证明结构和构件都是合理的。

图10-9描述的是关于安全系统的SafeHome体系结构的一个实例。图10-8中构件被进一步精化以显示更多的细节，例如，探测器管理构件与调度器基础设施构件相互作用，此基础设施构件实现并发地选取安全系统使用的每个传感器对象。对于图10-8中的每一个构件也做了类似的细节刻画。

303

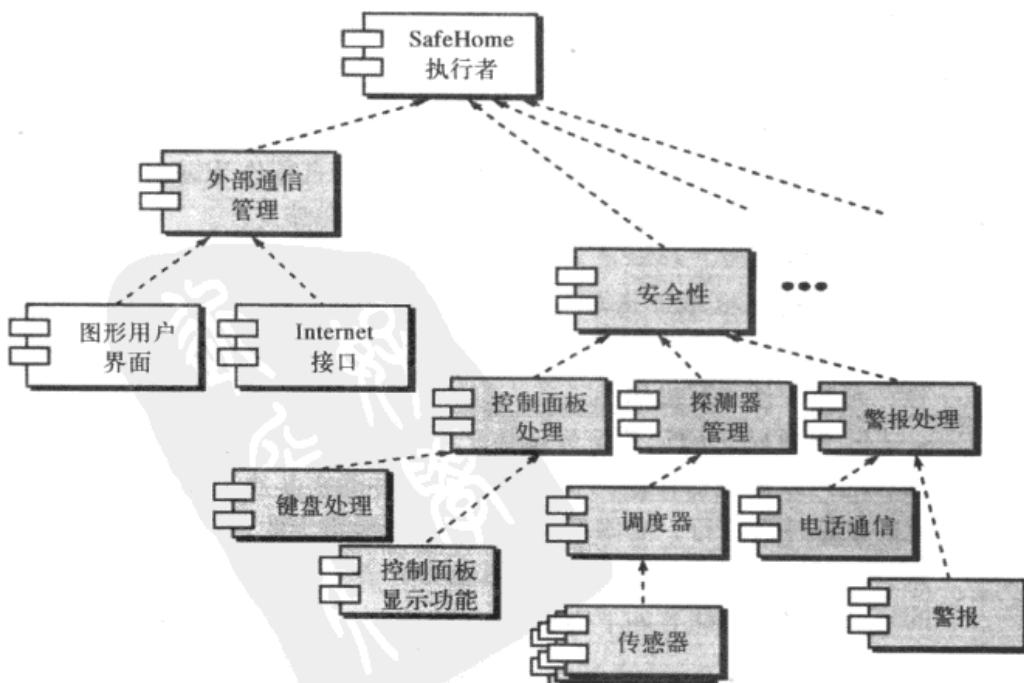


图10-9 带有构件详尽细节的安全功能实例

SOFTWARE TOOLS**体系结构设计**

目的：体系结构设计工具通过描述构件接口、依赖与联系以及交互作用来建立整体软件结构模型。

机制：工具采用的机制多种多样。在大多数情况下，体系结构的设计能力是分析和设计建模自动化工具的一部分功能。

代表性工具⁷

Adalon，由Synthis公司（www.synthis.com）开发，是设计和构建专门基于Web的构件体系结构的专用设计工具。

ObjectiF，由microTOOL GmbH（www.microtool.com）开发，是一个基于UML的设计工具，它产生的体系结构（如，Coldfusion、J2EE和Fusebox等）符合基于构件的软件工程（第30章）。

Rational Rose，由Rational开发（www.rational.com），是基于UML的设计工具，它支持体系结构设计中的所有方面。

10.5 评估可选的体系结构设计

设计会导致多种可供选择的候选体系结构，其中每一种候选体系结构都需要评估，以确定哪种体系结构最适合要解决的问题。在下面的几节中，我们将讨论如何评估可选的体系结构设计。

“也许它在地下室，让我上楼查看一下。”

——M. C. Escher

10.5.1 体系结构权衡分析方法

卡内基·梅隆大学软件工程研究所（SEI）开发了一种体系结构权衡分析方法（Architecture Trade-off Analysis Method, ATAM）[KAZ98]，该方法建立了一个迭代的软件体系结构评估过程。下面的设计分析活动是迭代进行的。

WebRef

有关ATAM的进一步信息可从
www.sei.cmu.edu/ata/ata_method.html获得。

304

1. 收集场景。开发一系列的用例（第7、8章），从用户的角度描述系统。
2. 诱导需求、约束和环境描述。这些信息是作为需求工程的一部分而需要的，并用于保证所有共利益者的关注点都会涉及。
3. 描述那些已经被选用于解决场景和需求的体系结构风格/模式。
4. 通过孤立地考虑每个属性来评估质量属性。体系结构设计评估的质量属性包括：可靠性、性能、安全性、可维护性、灵活性、可测试性、可移植性、可复用性和互操作性。
5. 针对特定的体系结构风格，弄清质量属性对各种体系结构属性的敏感性。这可以通过对体系结构做小的变更并确定某质量属性（如性能）对该变更的敏感性而进行。受体系结构的变更影响很大的属性称为敏感点。
6. 使用在第5步中进行的敏感性分析鉴定候选体系结构（在第3步中开发的）。SEI对该方法有如下描述[KAZ98]：

⁷ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。在大多数情况下，工具的名字由各自的开发者注册为商标。

一旦确定了体系结构敏感点，寻找权衡点就简单了，只需识别出对多个属性敏感的体系结构元素。例如，客户/服务器体系结构的性能可能对服务器数量是高敏感的（在一定范围内，增加服务器的数量可使性能提高）……这样，服务器数量就是该体系结构的一个权衡点。

这6个步骤描述了第一次ATAM迭代。基于第5步和第6步的结果，某些候选体系结构可能被删除，剩余的一个或多个体系结构可能被修改和进一步细化，然后，ATAM步骤被再次应用⁸。

SAFEHOME

体系结构评估

[场景] Doug Miller的办公室，体系结构设计建模正在进行。

[人物] Vinod、Jamie、Shakira和Ed，SafeHome软件工程团队成员；Doug Miller，软件工程团队经理。

[对话]

Doug: 我看到大家为SafeHome这个产品设计了两个不同的体系结构，这是一件好事。我的问题是，你们如何选择最好的体系结构呢？

Ed: 我正在设计一个调用和返回风格的体系结构，然后Jamie或我将设计一个面向对象的体系结构。

Doug: 好的，但是我们如何选择呢？

Shakira: 我在高年级时学习了一门设计课程，我记得有很多选取办法。

Vinod: 是有一些，但是它们都太理论化了。听着，我认为我们可以自己评估并使用用例和场景选择正确的体系结构。

Doug: 这不是一回事吗？

Vinod: 当谈论有关体系结构评估的时候，它们不是一回事。我们已经有了一个完整的用例集合，因此，我们将每一个用例都应用到这两个体系结构中，查看系统的反映——即在用例的环境中构件和连接器是如何工作的。

Ed: 这是个好主意！可以确保我们不遗漏任何东西。

Vinod: 当然，它还能告诉我们体系结构设计是不是令人费解的，系统是不是必须转换到它的分支上来完成工作。

Jamie: 场景仅仅就是用例的别名吗？

Vinod: 不是的，在这里场景具有不同的意义。

Doug: 你们谈论的是质量场景或者变化场景，是吗？

Vinod: 是的，我们要做的就是回到共利益者那里，问问他们SafeHome在接下来的三年中可能会有什么改动，如，新版本、特征等这类变化。我们创建了一套变化场景，也开发了一套质量场景，这些场景定义了在软件体系结构中要看到的属性。

Jamie: 我们把它们运行到体系结构中。

Vinod: 是的，那个能更好地处理用例和场景的体系结构就是我们的最终选择。

305

⁸ 相对于ATAM，软件体系结构分析方法（Software Architecture Analysis Method, SAAM）也是一个选择，并且对于那些对体系结构分析感兴趣的读者来说，后者是一个值得探讨的方法。从如下地址可以下载一篇关于SAAM的文章：<http://www.sei.cmu.edu/publications/articles/saam-metho-propert-sas.html>。

10.5.2 体系结构复杂性

对体系结构的整体复杂性进行评估，一种很有用的技术是考虑体系结构中构件间的依赖关系，这些依赖关系是由系统中的信息/控制流驱动的。Zhao[ZHA98]建议了三种类型的依赖：

共享依赖表示在使用相同资源的消费者间或为相同消费者生产的生产者之间的依赖关系。例如，对两个构件u和v，如果u和v引用相同的全局数据，则在u和v之间存在共享依赖关系。

流依赖表示资源的生产者和消费者间的依赖关系。例如，对两个构件u和v，如果u在控制流入v（先决条件）之前完成，或如果u和v通过参数通信，则在u和v之间存在流依赖关系。

约束依赖表示在一组活动间相关控制流上的约束。例如，对两个构件u和v，u和v不能同时执行（互斥），则在u和v之间存在约束依赖。

Zhao提到的共享依赖和流依赖在某些方面类似于第9章讨论过的耦合概念。耦合是可应用在体系结构级和构件级的重要设计概念，第15章中讨论了用来评估耦合的简单度量。

10.5.3 体系结构描述语言

306 房屋的建筑师拥有一套标准的工具和符号，能够以一种明确的、可理解的方式来描述设计。尽管软件架构师也能够使用UML符号、其他的图表和一些相关的工具，但还是需要更加形式化的方法来描述体系结构设计的规格说明。

体系结构描述语言（architectural description language, ADL）为描述软件体系结构提供一套语义和语法。Hofmann和他的同事[HOF01]建议ADL应该使得设计者具有分解体系结构构件、将单独构件组合成大的体系结构块，以及描述构件之间的接口（连接机制）的能力。一旦体系结构设计使用的描述的、基于语言的技术被建立起来，那么很有可能随着设计的进化而建立起体系结构的有效评估方法。

SOFTWARE TOOLS

体系结构描述语言

下面这些重要的ADL总结来自Rickard Land[LAN02]，下面的摘录得到作者的允许。需要说明的是，前5个ADL是为了研究的目的而开发的，它们不是商业产品。

Rapide (poset.stanford.edu/rapide/) [LUC95]建立在偏序集的概念之上。

UniCon (www.cs.cmu.edu/~UniCon) [SHA96]以一种对设计者十分有用的形式定义软件体系结构。

Aesop (www.cs.cmu.edu/~able/aeop/) [GAR94]主要用于处理风格的复用问题。

Wright (www.cs.cmu.edu/~able/wright/) [ALL97]使用谓词的方式来规范体系结构风格，从而允许静态检查以确定体系结构的完整性和一致性。

Acme (www.cs.cmu.edu/~acme/) [GAR00]是第二代的ADL。

UML (www.uml.org/) 包括很多体系结构描述需要的人工制品，但是没有其他ADL那样完整。

10.6 映射数据流到软件体系结构

10.3.1节中讨论的体系结构风格描述了本质上不同的体系结构，因此，并不存在一种能够实现从分析模型到各种体系风格转换的全面映射。事实上，对某些体系结构风格来说没有实用的映射，设计者必须采用10.4节讨论的技术将需求转换到这些风格的设计上。

为了描述体系结构映射的方法，我们考虑“调用和返回”体系结构的映射技术——这种体系结构是非常常见的结构，很多类型的系统都采用这种结构。这种映射技术使得设计者能够从分析模型的数据流图中导出相当复杂的“调用和返回”体系结构。这种技术有时也称为结构化设计，在Myers[MYE78]及Yourdon和Constantine[YOU79]的书中有详细的描述。

结构化设计经常被刻画为面向数据流的设计方法，因为它提供了方便的从数据流图（第8章）到软件体系结构的变换。信息流的类型决定了映射方法。[307]

10.6.1 变换流

信息必须以“外部世界”信息的形式进出软件。例如，键盘输入的数据、电话线上的语音信号以及多媒体应用中的视频图像都是外部世界信息的形式。为了处理方便，外部的数据形式必须转化成为内部的形式。信息沿着各种将外部数据变换为内部形式的路径进入系统，这些路径被标识为输入流，在软件的核心进行转换。输入数据通过“变换中心”，并沿着各种路径流出软件，这些流出的数据称为输出流。整个的数据流动以一种顺序的方式沿着一条或仅仅很少的几条“直线”路径进行⁹。当一部分数据流图展现了这些特征时，就表明了变换流的存在。

10.6.2 事务流

信息流经常被描述为单个数据项——称为事务，它可以沿多条路径中的一条触发其他数据流。如果数据流采用类似于图10-10的形式，则就是事务流。

事务流通过数据沿某输入路径的移动来呈现其特征，该输入路径将外部信息转换成一个事务。对事务进行评估，并且根据其值启动其中的一条动作路径流。发射出很多动作路径的信息流中心被称为事务中心。

需要指出的是，在一个大系统的DFD中，变换流和事务流可能会同时出现，例如，在一个面向事务的流中，动作路径上的信息流可能会体现出变换流的特征。

10.6.3 变换映射

变换映射是一组设计步骤，可以将具有变换流特征的DFD映射为某个特定的体系结构风

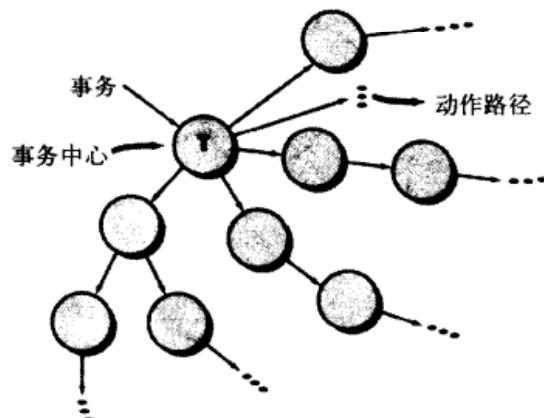


图10-10 事务流

[308]

⁹ 这种类型信息流的一个明显映射就是10.3.1节中描述的数据流体系结构。然而，很多情况下，对于复杂的系统来说，数据流体系结构可能不是最佳的选择。例如，随着时间呈现很多变化的系统，或者在其中与数据流相关的处理不一定是顺序的系统。



只要此时对DFD
进一步精化，就
能导出那些展示
高内聚性的变换
泡泡。

格。为了说明这个方法，我们再次考虑SafeHome的安全功能¹⁰。分析模型的元素之一是一组数据流图，这些数据流图描述带有安全功能的信息流。为了把这些数据流图映射成体系结构，可以以下面的设计步骤进行。

步骤1：评审基本系统模型。基本系统模型或者环境图把安全功能描述为一个单一的变换，描述了流入和流出安全功能的数据的生产者和消费者。图10-11刻画了一个0层模型，图10-12描述了初步精化后的安全功能数据流。

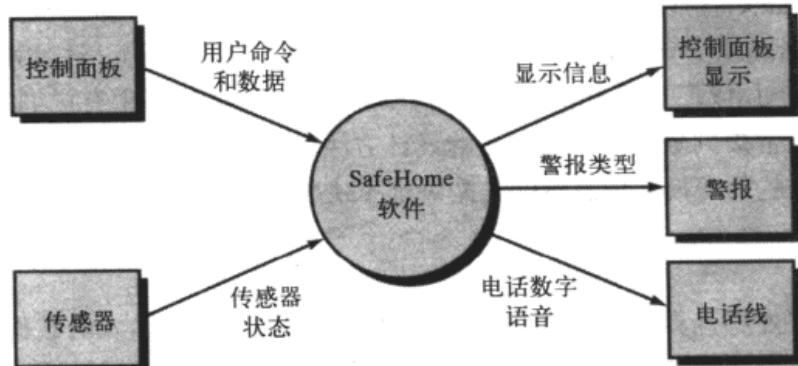


图10-11 SafeHome安全功能的环境级DFD

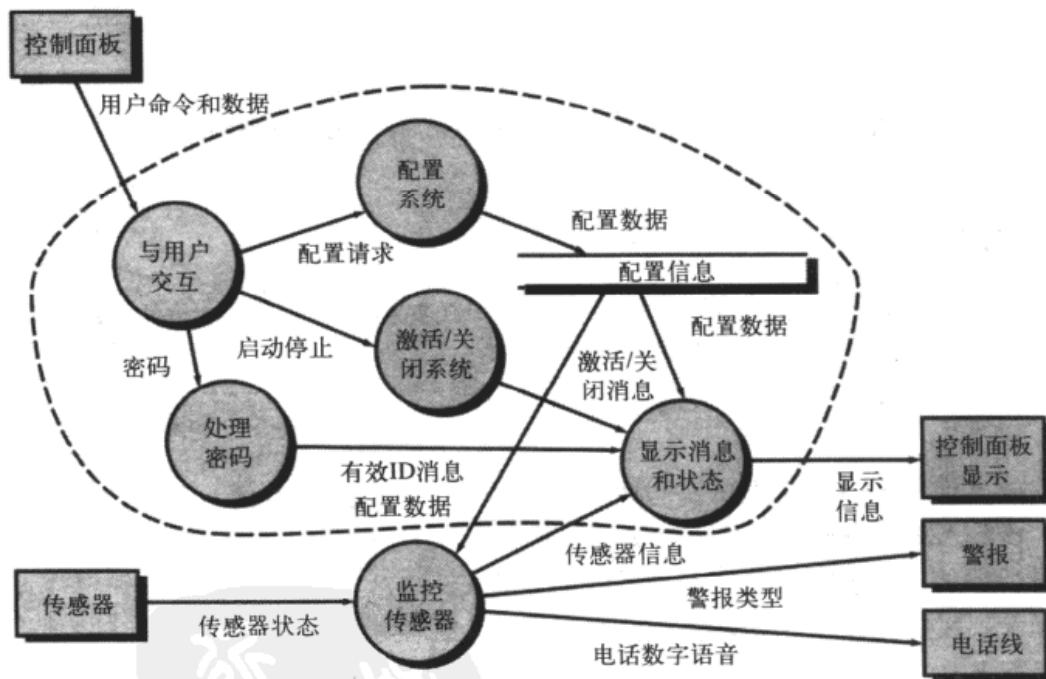


图10-12 SafeHome安全功能的第一层DFD

步骤2：评审和精化软件的数据流图。对从分析模型获得的信息进行精化，以获得更多的细节。例如，检查第2层监控传感器的DFD（图10-13），并导出第3层数据流图（图10-14）。在第3层，数据流图中的每个变换都展示了高内聚性（第9章），也就是说，变换所包含的过程完成单一的、清楚的功能，该功能可被实现为SafeHome软件中的一个构件。因此，图10-14中的DFD包含了设计监控传感器子系统体系结构所需的细节信息，不需要再进一步精化。

¹⁰ 我们这里只考虑SafeHome安全功能中使用控制面板的那一部分，本书和本章前面所讨论其他特征这里不予以考虑。

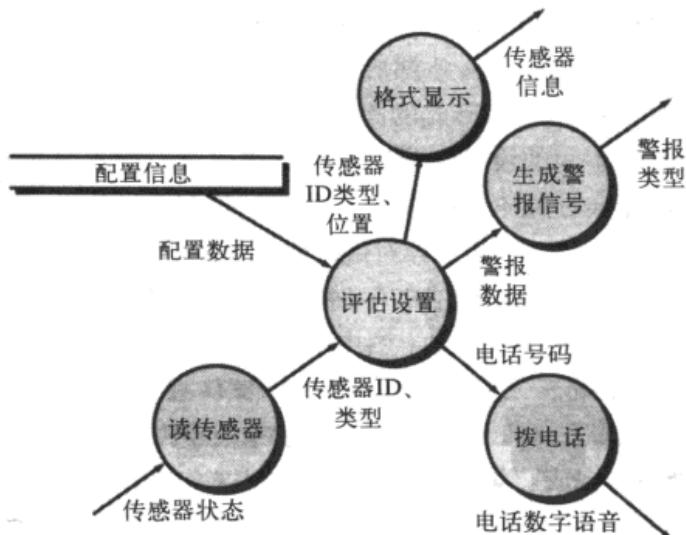


图10-13 精化“监控传感器”变换的第2层DFD

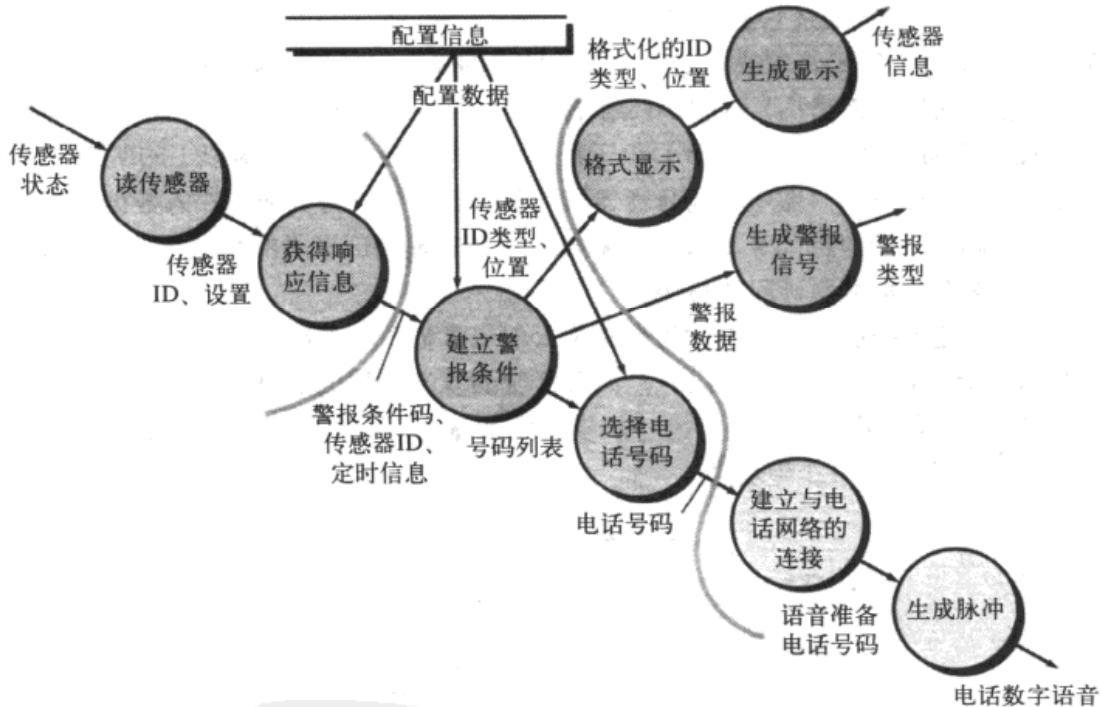


图10-14 具有流边界的监控传感器的第3层DFD

POINT
在一些面向流模型中这两种类型的数据流都会经常遇到。这些流被分割成段，并且使用适当的映射获得程序结构。

步骤3：确定DFD是否含有变换流或事务流特征。总的来说，系统里的信息流总可以表示为变换流，但当遇到一个明显的事务流特征（图10-10）时，最好采用另一种设计映射。在这一步骤中，设计人员根据DFD的主要性质选择全局（整个软件范围）的流特征。此外，也应隔离出变换流和事务流的局部区域，这些子流可以用于改进由前面全局特征导出的程序体系结构。在此，我们只关注图10-14描述的监控传感器子系统的数据流。

通过评估DFD（图10-14），我们可以看出，数据通过一条输入路径进入软件，沿三条输出路径流出，很显然没有明显的事务中心（虽然“建立警报条件”变换可能被看作事务中心）。因此，信息流将呈现出一个从头到尾的变换特征。



边界流位置的多样性展示了多种候选程序结构。这用不了多少时间，并能提供了重要的见解。

311



在这个阶段不要教条。根据所构建系统的复杂程度，可能需要建立两个或更多个输入处理或计算控制器。

312

当遇到变换流时，DFD将被映射成一个能为信息的输入、变换和输出处理提供控制的特定结构（“调用和返回”体系结构）。图10-15给出了对监控传感器子系统进行第一级分解的结果，主控制器（图中称为监控传感器执行者）位于程序结构的顶端，负责协调以下的从属控制功能：

- 输入信息处理控制器（图中称为传感器输入控制器）负责协调所有输入数据的接收。
- 变换流控制器（图中称为警报条件控制器）负责管理内部形式的数据上的所有操作（例如，调用多个数据变换过程的模块）。
- 输出流信息处理控制器（图中称为警报输出控制器）负责管理输出信息的产生。

虽然图10-15蕴涵了三叉结构，但是，大型系统的复杂数据流图可能会要求为上述的每个类属控制功能提供两个或多个模块。第一层模块的数量应限定在既能完成控制功能又能维持良好的独立特征所要求的最少模块数。

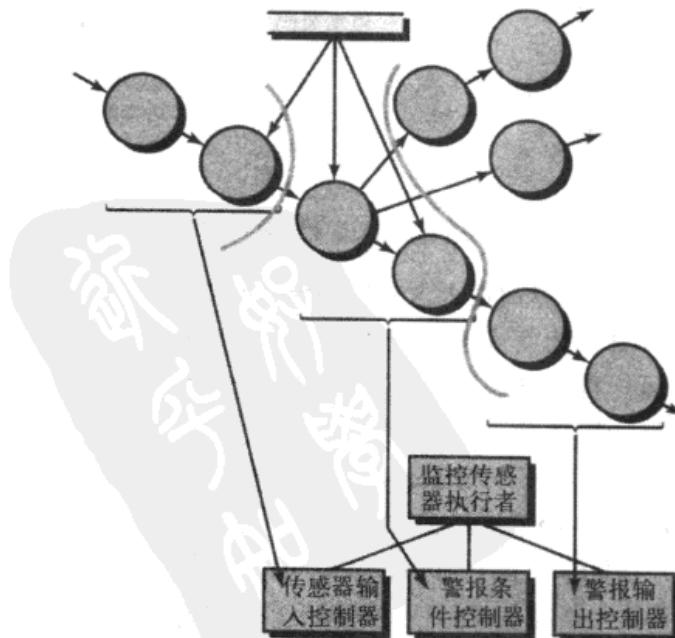


图10-15 监控传感器的第一级分解

步骤4：通过确定输入和输出流的边界，分离出变换中心。前面已经指出，输入流被描述为信息从外部形式变换为内部形式的路径，而输出流是信息从内部形式变换为外部形式的路径。但是输入流和输出流的边界还有待说明，也就是说，不同的设计人员在选择流边界时可能不尽相同。事实上，不同的流边界选择会导致不同的设计方案。尽管在选择流边界时要加以注意，但沿流路径若有一个泡泡的差异对最终程序结构的影响并不会太大。

本例子中的流边界由图10-14中两条纵向穿过流的阴影曲线给出。选定变换中心的变换（泡泡）位于图中两条从顶到下的边界曲线之间。也可以调整边界（例如，将输入流的边界放置在“读传感器”和“获得响应信息”之间）。本设计步骤的重点在于选择合理的边界，而不是花时间反复考虑边界的位置。

步骤5：完成“第一级分解”。使用这个映射导出的程序体系结构导致了自顶向下的控制分布。分解的作用是得到一个程序结构，其中顶层模块做决策；低层模块完成大多数输入、计算和输出工作；中层模块既完成一部分控制，又完成适量的工作。

当遇到变换流时，DFD将被映射成一个能为信息的输入、变换和输出处理提供控制的特定结构（“调用和返回”体系结构）。图10-15给出了对监控传感器子系统进行第一级分解的结果，主控制器（图中称为监控传感器执行者）位于程序结构的顶端，负责协调以下的从属控制功能：

- 输入信息处理控制器（图中称为传感器输入控制器）负责协调所有输入数据的接收。
- 变换流控制器（图中称为警报条件控制器）负责管理内部形式的数据上的所有操作（例如，调用多个数据变换过程的模块）。
- 输出流信息处理控制器（图中称为警报输出控制器）负责管理输出信息的产生。

虽然图10-15蕴涵了三叉结构，但是，大型系统的复杂数据流图可能会要求为上述的每个类属控制功能提供两个或多个模块。第一层模块的数量应限定在既能完成控制功能又能维持良好的独立特征所要求的最少模块数。



保持“第一线工作者”模块在程序结构的底层。这将导致一个易于修改的体系结构。

步骤6：完成“第二级分解”。第二级分解是将DFD中的每个变换（泡泡）映射到程序结构中的相应模块。从变换中心的边界开始，沿输入路径和输出路径向外，将变换依次映射到软件结构的从属层。图10-16描述了第二级分解的一般方法。

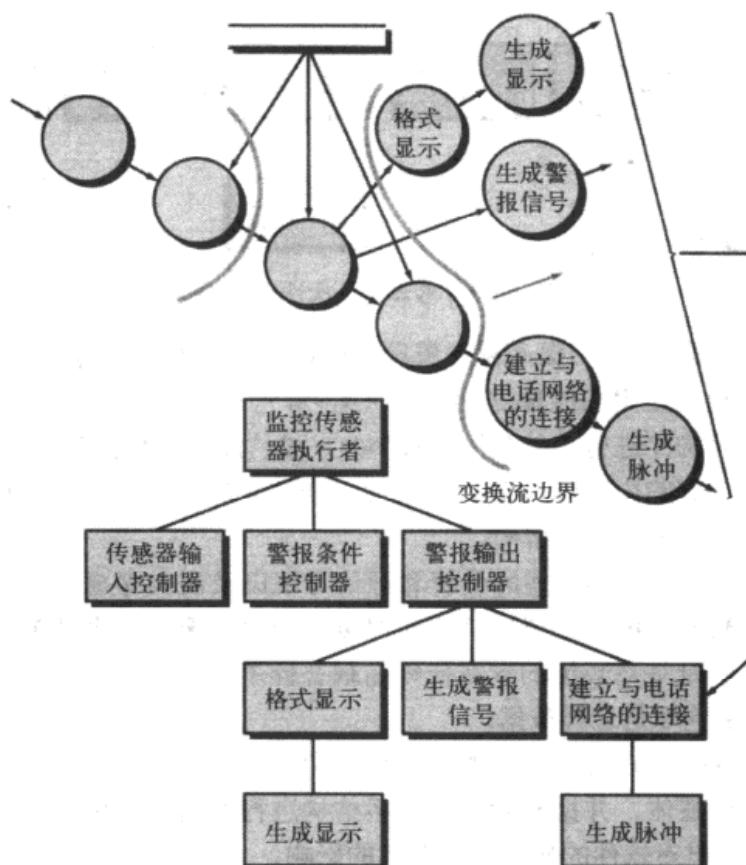


图10-16 监控传感器的第二级分解



删除冗余的控制模块，也就是，如果一个控制模块除了控制另一个模块之外不具具备其他任何功能，则其控制功能应该结合（内爆）到更高层次的模块中。

虽然图10-16描述了DFD变换和软件模块间的一对一映射，但其他的映射方式也经常采用。两个甚至三个变换可以合并在一起表示为一个构件，或者一个单独的变换也可以扩展成两个或者多个构件。现实考虑和设计质量的权衡决定着第二级分解的输出结果。评审和精化可能会导致这个结构的改变，但这个结构仍然应作为第一次迭代设计。

对于输入流的第二级分解遵循同样的方式，从输入流一侧的变换中心边界开始向外移动。监控传感器子系统软件的变换中心映射略有不同，DFD变换部分的每个代表数据转换或计算的变换都被映射为变换控制器的从属模块。图10-17给出了一个完整的第一次迭代的体系结构。

按以上方法映射出来的构件（图10-17）描述了软件体系结构的一个初始设计。虽然构件的名字已经可以体现其功能，但我们仍然需要为每个构件提供简要的处理叙述（可以修改分析建模时创建的PSPEC得到）。

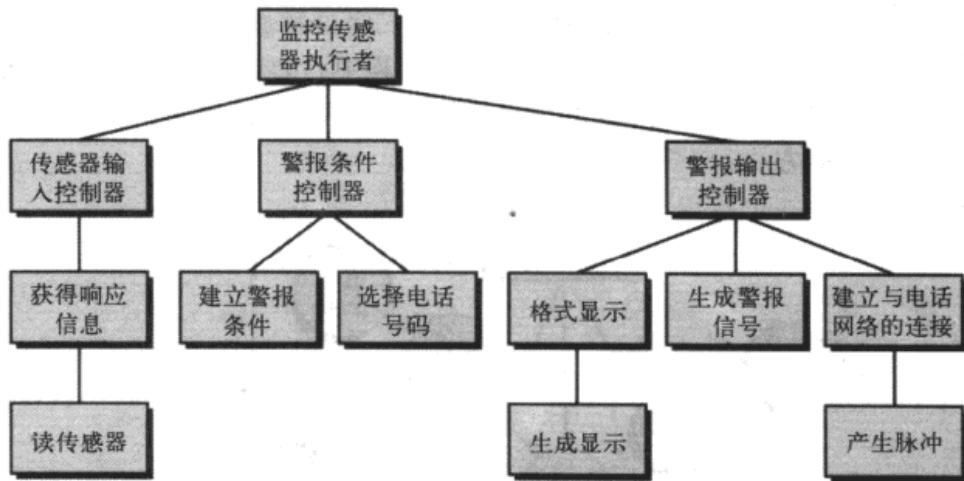


图10-17 监控传感器第一次迭代后的结构

313
314

关注已导出模块的功能独立性。
高内聚和低耦合
应该是工作目标。

步骤7：使用提高软件质量的设计启发式方法，精化第一次迭代得到的体系结构。应用功能独立性概念（第9章），能够精化第一次迭代得到的体系结构。对构件进行“外爆”（explode）或“内爆”（implode），可以得到合理的分解、好的内聚性、低的耦合性，最重要的是获得易于实现、可系统性测试和易于维护的程序结构。

求精往往是在10.5节中简要描述的分析和评估方法以及现实考虑和常识等指导之下进行的。例如，有时输入数据流的控制器完全没有必要，有时需要在一个从属于变换控制器的构件中完成输入处理，有时全局数据的存在使得高耦合性不可能避免，有时不能达到优化结构特征，软件需求加人工判断是最终依据。

以上七个步骤的目的是开发一个软件的体系结构表示，也就是说，一旦结构被定义，我们就可以将其视为一个整体，并据此对软件体系结构进行评估和精化。虽然以后的修改仍然需要一些额外的工作，但对软件的质量具有深远的影响。

读者可以停下来考虑一下以上介绍的设计方法与编写程序过程之间的区别。如果只有代码作为软件的表示，开发人员就很难在全局的层次上对软件进行评估或精化，容易导致“只见树木不见森林”。

SAFEHOME

精化初级体系结构

[场景] Jamie's的房间，正在进行设计建模。

[人物] Jamie和Ed，SafeHome软件工程团队成员。

[对话]

(Ed刚刚完成监控传感器子系统的初级设计。他停下来征求Jamie的意见。)

Ed：这是我设计的体系结构。(Ed给Jamie看图10-17，该图花费了她几个月的时间。)

Jamie：不错，但是我认为可以将其简化一些，那样就更好了。

Ed：例如？

Jamie：嗯，你为什么要用“传感器输入控制器”这个构件呢？

Ed：因为需要一个映射的控制器。

Jamie: 不是这样的，由于我们管理的是输入数据的单一路径，所以控制器就没有必要做那么多事情了。我们可以省掉这个控制器而不会有任何不良影响。

Ed: 我明白了，我将修改并且……

Jamie (微笑): 先等等！我们还可以结合“建立警报条件”和“选择电话号码”两个构件。你(在图中)给出的转换控制器没有必要，少许减少内聚性是允许的。

Ed: 简单，哈哈。

Jamie: 是的，而且当你开展精化的时候，将“格式显示”和“生成显示”两个构件结合起来是一个好主意。控制面板的显示格式很简单，我们可以定义新的模块，称之为“产生显示”。

Ed (草草地画着): 这就是你认为我们应该做的吗？(他给Jamie看图10-18。)

Jamie: 这只是个开始罢了。

315

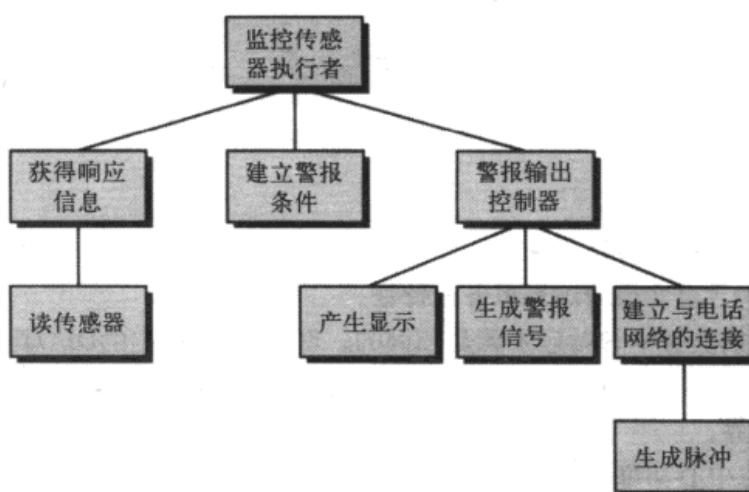


图10-18 监控传感器精化后的程序结构

10.6.4 事务映射

在许多软件应用中，单独的数据项会触发多条信息流中的一条，这些数据流将实现该数据项隐含的功能。这个数据项被称为事务，其相应的流特征在10.6.2节中已经讨论过。本节我们主要考虑将事务流映射到体系结构的设计步骤。

我们将通过考虑SafeHome安全功能的用户交互子系统来说明事务映射。该子系统的第1层数据流如图10-12所示。对此图数据流进行精化，得到第2层数据流图，如图10-19所示。数据对象“用户命令”流入系统，沿三条动作路径之一产生其他信息流，单个数据项“命令类型”引发数据流从中心向外扇出，因此，整个数据流的特征是面向事务的。

值得注意的是，沿着三个动作路径中的两条路径上的信息流可以满足额外的输入流(例如，“系统参数和数据”是动作路径“配置”的输入)。每个动作路径都流入单个变换“显示消息和状态”。

事务映射的设计步骤与变换映射的步骤相类似，甚至在某些情况下是相同的(参见10.6.3节)。其主要的区别在于DFD到软件体系结构的映射。

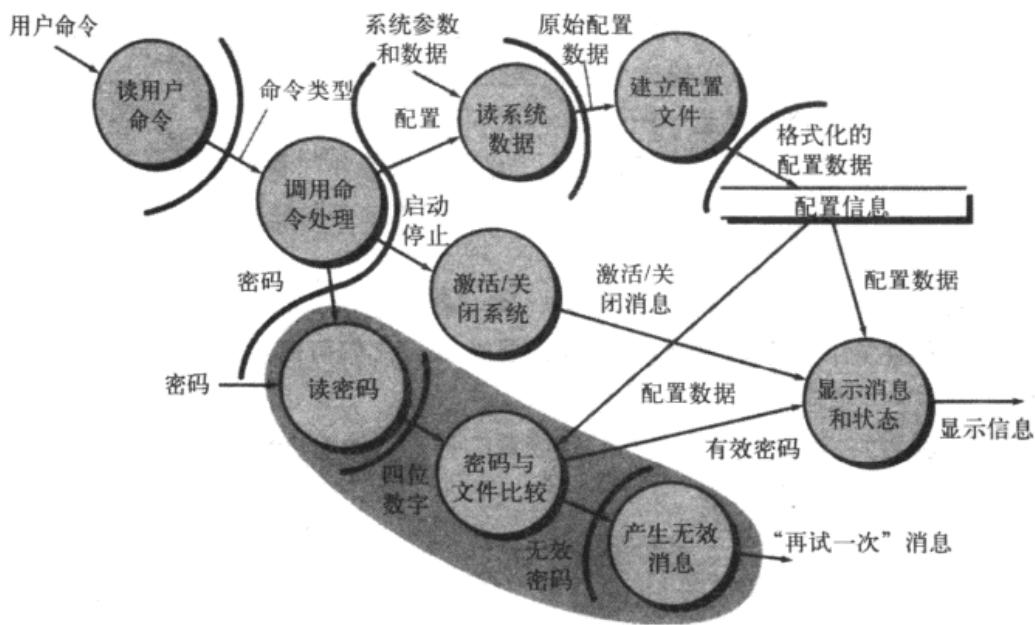


图10-19 用户交互子系统的第2级DFD

步骤1：评审基本系统模型。

316

步骤2：评审和精化软件的数据流图。

步骤3：确定DFD含有变换流还是事务流特征。

这三个步骤与变换映射中的对应步骤是相同的。图10-19中给出的DFD具有典型的事务流特征，但从变换“调用命令处理”发出的两条动作路径上的流具有变换流的特征，因此必须为这两种流建立流边界。

步骤4：标识事务中心和每条动作路径上的流特征。事务中心的位置可以从DFD上直接识别出来，事务中心位于几条动作路径的起始点上。例如在图10-19中，变换“调用命令处理”就是事务中心。



POINT
第1级分解分出
软件控制层，第
2级分解在合适的
控制器下分布
“第一线工作者”
模块。

输入路径（就是接收事务的流路径）和所有的动作路径都必须被隔离出来。评估每条动作路径，以确定它们自己的流特征。例如，路径“密码”（在图10-19中用阴影标出）具有变换流特征，需要给出输入、变换和输出流的边界。

步骤5：将DFD映射到一个适合于进行事务处理的程序结构上。事务流被映射到一个包含输入分支和调度分支的程序结构上。输入分支结构的开发与变换流映射的方法相类似，从事务中心开始，沿输入路径的变换都被映射成模块。调度分支结构又包含一个调度器模块，它控制下面所有的动作模块。DFD的每一个动作流路径都被映射成与其自身的流特征一致的结构。图10-20描述了该过程。

考虑用户交互子系统的数据流，步骤5的第1级分解如图10-21所示。变换“读用户命令”和“激活/关闭系统”可以直接映射到程序结构中，无需中间的控制模块。事务中心“调用命令处理”直接映射成同名的调度器模块；建立系统配置和密码处理的控制器，如图10-21所示。

步骤6：分解并精化事务结构和每条动作路径的结构。数据流图的每条动作路径都有自己的信息流特征，我们知道，它可以是变换流也可以是事务流。与动作路径相关的“子结构”可以根据本节和上节讨论的设计步骤进行开发。

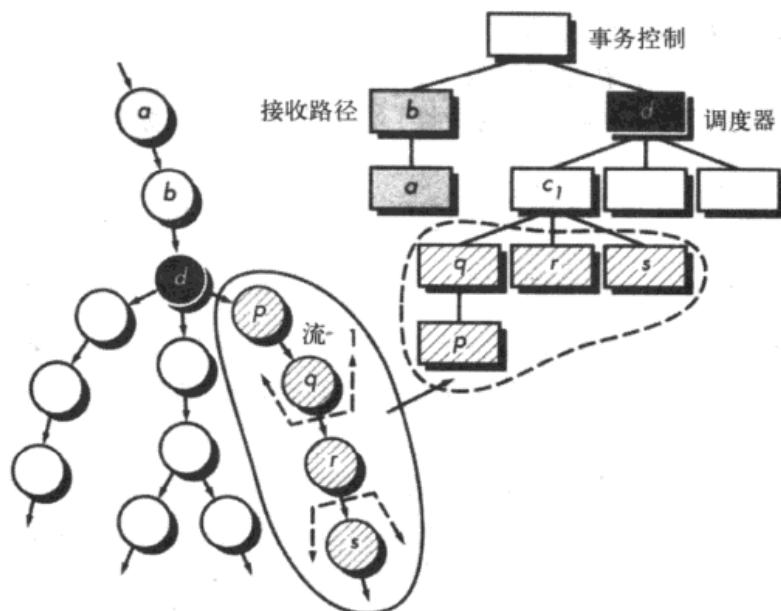


图10-20 事务映射

作为一个例子，考虑图10-19所示的“密码处理”信息流（阴影部分），它具有典型的变换流特征。一个密码作为输入（输入流），它被送到变换中心与以前存储的密码进行比较，如果比较不匹配的话，将产生警报和警告消息（输出流）。“配置”路径也是类似地按变换映射处理。图10-22给出了最后产生的软件体系结构。

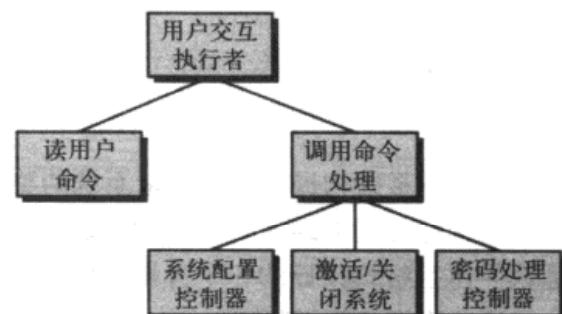


图10-21 用户交互子系统的第1级分解

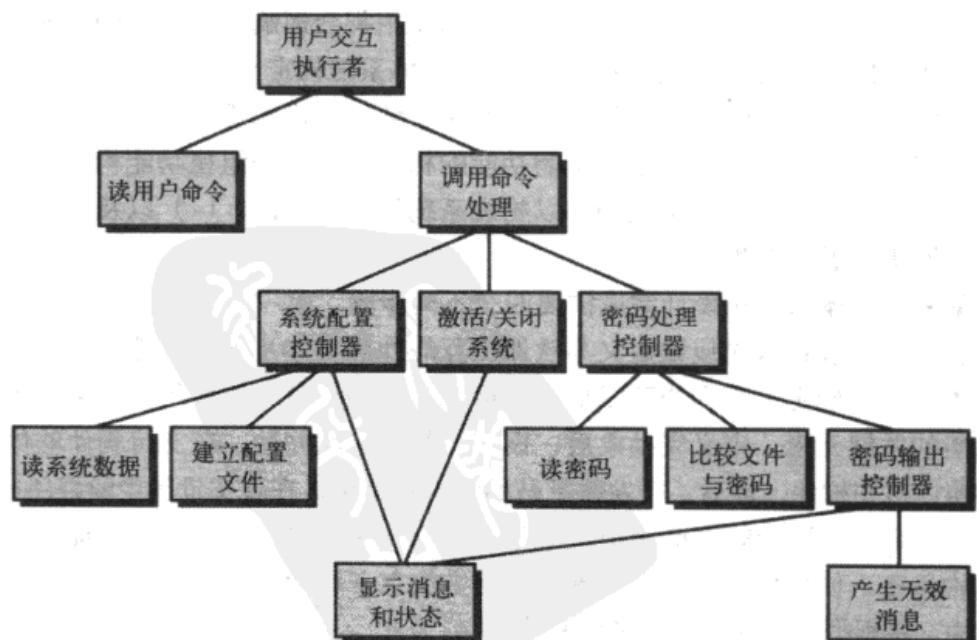


图10-22 用户交互子系统的第一次迭代体系结构

318
319

步骤7：使用提高软件质量的设计启发式方法，精化第一次迭代得到的体系结构。这与变换映射的对应步骤是一样的。在这两种设计方法中，当提出修改程序结构时，必须仔细考虑模块的相关性、实用性（实现和测试的有效性）以及可维护性的标准。

“使其尽可能的简单，但不是简化……”

——Albert Einstein

10.6.5 精化体系结构设计

在讨论设计求精之前，应首先遵循这句话：“记住，不能工作的‘优化设计’是很值得怀疑的。”基于设计度量和启发式方法，软件设计人员应该始终考虑开发一个满足所有功能和性能需求以及值得信任的软件表示。

应该鼓励在设计早期阶段就对软件体系结构进行精化。正如本章前面讨论过的，对选择的体系结构风格导出、精化和评估以达到“最好”。这种优化方法也是开发软件体系结构表示的一个重要收益。

重要的是注意，结构上的简单往往反映出程序的优雅和高效。设计求精应在满足模块化要求的前提下尽量减少构件的数量，在满足信息需求的前提下尽量减少复杂的数据结构。

10.7 小结

软件体系结构提供了待建造系统的整体视图，它描述软件构件的结构和组织、构件的性质以及构件之间的连接。软件构件包括程序模块和程序操作的各种数据表示，因此数据设计是软件体系结构设计的一个有机部分。体系结构关注于早期设计决策，并提供了考虑多个可选系统结构优点的机制。

数据设计将在分析模型中定义的数据对象转换成驻留于软件中的数据结构。描述数据对象的属性、数据对象间的关系以及它们在程序中的使用都会影响数据结构的选择。在更高的抽象层次上，数据设计可能导致数据库或数据仓库的体系结构定义。

对软件工程师而言，可以使用一系列不同的体系结构风格和模式。每个风格描述了一个系统类别，它包含：(1) 一组完成系统所需功能的构件；(2) 一组使功能构件间通信、协调及合作的连接器；(3) 定义如何集成构件以构成系统的约束条件；(4) 以及使得设计者能够理解系统整体性质的语义模型。

320

总体感觉来说，体系结构设计完成需要四个不同步骤。第一步，系统必须表示在相应的环境中，也就是说，设计人员应该定义与软件交互的外部实体及其交互性质。一旦环境得到说明，设计人员应该确定一系列的顶层抽象，称之为原始模型，该原始模型可以表示本系统行为或者功能的关键元素。定义完抽象后，设计开始向实现移动。在支持构件的体系结构环境中标识和描述这些构件。最后，开发体系结构的特定实例，在现实世界中验证所得设计。

作为一个体系结构设计的简单例子，本章介绍的映射方法使用数据流特征来导出一个常用的体系结构风格。使用变换映射或者事务映射方法之一将数据流图映射为程序结构。一旦得到体系结构，将根据质量标准对其进行细化和分析。

参考文献

- [AHO83] Aho, A. V., J. Hopcroft, and J. Ullmann, *Data Structures and Algorithms*, Addison-Wesley, 1983.
- [ALL97] Allen R., "A Formal Approach to Software Architecture," Ph.D. Thesis, Carnegie Mellon University, Technical Report Number: CMU-CS-97-144 1997.
- [BAR00] Barroca, L. and P. Hall (eds.), *Software Architecture: Advances and Applications*, Springer-Verlag, 2000.
- [BAS03] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003.
- [BOS00] Bosch, J., *Design & Use of Software Architectures*, Addison-Wesley, 2000.
- [BUS96] Buschmann, F., *Pattern-Oriented Software Architecture*, Wiley, 1996.
- [DAT00] Date, C. J., *An Introduction to Database Systems*, 7th ed., Addison-Wesley, 2000.
- [DIK00] Dikel, D., D. Kane, and J. Wilson, *Software Architecture: Organizational Principles and Patterns*, Prentice-Hall, 2000.
- [FRE80] Freeman, P., "The Context of Design," in *Software Design Techniques*, 3rd ed. (P. Freeman and A. Wasserman, eds.), IEEE Computer Society Press, 1980, pp. 2-4.
- [GAR94] Garlan D., R. Allen, and J. Ockerbloom, "Exploiting Style in Architectural Design Environments," in *Proceedings of SIGSOFT '94 Symposium on the Foundations of Software Engineering*, 1994.
- [GAR00] Garlan D., R. T. Monroe, and D. Wile, "Acme: Architectural Description of Component-Based Systems," in *Foundations of Component-Based Systems*, G. T. Leavens and M. Sitarman, eds. Cambridge University Press, 2000.
- [HOF00] Hofmeister, C., R. Nord, and D. Soni, *Applied Software Architecture*, Addison-Wesley, 2000.
- [HOF01] Hofmann, C., et al., "Approaches to Software Architecture," downloadable from: <http://citeseer.nj.nec.com/84015.html>.
- [KAZ98] Kazman, R., et al., *The Architectural Tradeoff Analysis Method*, Software Engineering Institute, CMU/SEI-98-TR-008, July 1998.
- [KIM98] Kimball, R., L. Reeves, M. Ross, and W. Thornthwaite, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*, Wiley, 1998.
- [LAN02] Land R., A Brief Survey of Software Architecture, Technical Report, Dept. of Computer Engineering, Mälardalen University, Sweden, February, 2002.
- [LUC95] Luckham D. C., et al., "Specification and Analysis of System Architecture Using Rapide," *IEEE Transactions on Software Engineering*, issue "Special Issue on Software Architecture," 1995.
- [MAT96] Mattison, R., *Data Warehousing: Strategies, Technologies and Techniques*, McGraw-Hill, 1996.
- [MYE78] Myers, G., *Composite Structured Design*, Van Nostrand, 1978.
- [PRE98] Preiss, B. R., *Data Structures and Algorithms: With Object-Oriented Design Patterns in C++*, Wiley, 1998.
- [SHA96] Shaw, M., and D. Garlan, *Software Architecture*, Prentice-Hall, 1996.
- [SHA97] Shaw, M., and P. Clements, "A Field Guide to Boxtology: Preliminary Classification of Architectural Styles for Software Systems," *Proc. COMPSAC*, Washington, DC, August 1997.
- [WAS80] Wasserman, A., "Principles of Systematic Data Design and Implementation," in *Software Design Techniques* (P. Freeman and A. Wasserman, eds.), 3rd ed., IEEE Computer Society Press, 1980, pp. 287-293.
- [YOU79] Yourdon, E., and L. Constantine, *Structured Design*, Prentice-Hall, 1979.
- [ZHA98] Zhao, J., "On Assessing the Complexity of Software Architectures," *Proc. Intl. Software Architecture Workshop*, ACM, Orlando, FL, 1998, pp. 163-167.

321

习题与思考题

- 10.1 使用数据流程图和处理叙述，描述一个具有明显数据流特征的计算机系统。使用10.6.3节介绍的技术定义流边界，并将DFD映射成软件体系结构。
- 10.2 写一篇3~5页的论文，给出基于问题性质来选择数据结构的指南。首先描述在软件中经常用到的一些经典数据结构，然后针对特定问题类型说明选择数据结构的标准。

- 10.3 解释一下服务于一个或者多个传统商业应用的数据库与数据仓库之间的不同。
- 10.4 使用数据流图和处理叙述，描述一个具有明显事务流特性的计算机系统。使用10.6.4节介绍的技术定义流边界，并将DFD映射成软件结构。
- 10.5 10.3.1节提到的一些体系结构风格在本质上是分层的，有一些却不是。请给出每一种类型的风格列表。这些非分层的体系结构风格是如何实现的？
- 10.6 如果还没有做，完成习题8.10。使用本章介绍的设计方法，为PHTRS开发一个软件体系结构。
- 10.7 研究ATAM（使用SEI网站）对10.5.1节给出的6个步骤进行详细讨论。
- 10.8 选出你熟悉的一个应用系统，回答10.3.3节中针对控制和数据提出的每一个问题。
- 10.9 一些设计人员认为所有的数据流都可以当做变换流。试讨论当事务流被当成变换流时，会对导出的软件体系结构有什么影响。请使用例子来说明要点。
- 10.10 在软件体系结构的讨论中经常提及体系结构风格、体系结构模式和框架等术语。（利用Web）做些研究，并描述一下这些术语之间的差别。
- 10.11 为10.3.1节中提到的每个体系结构风格给出两个或三个应用实例。
- 10.12 以房子或建筑的体系结构作比喻，与软件体系结构进行对比。传统的建筑体系结构学科和软件体系结构有何相似之处？有何不同之处？

322

推荐读物与阅读信息

在过去的十年里，涌现出大量软件体系结构方面的著作。Fowler（《Patterns of Enterprise Application Architecture》，Addison-Wesley, 2003）、Clements和他的同事（《Documenting Software Architecture: View and Beyond》，Addison-Wesley, 2002）、Schmidt和他的同事（《Pattern-Oriented Software Architectures》，two volumes, Wiley, 2000）、Bosch[BOS00]、Dikel和他的同事[DIK00]、Hofmeister和他的同事[HOF00]、Boss和Clements及Kazman[BAS03]、Shaw和Garlan[SHA96]以及Buschmann等人[BUS96]的著作对该主题提供了深入的讨论。Garlan（《An Introduction to Software Architecture》，Software Engineering Institute, CMU/SEI-94-TR-021, 1994）的早期工作很好地介绍了软件体系结构。Clements和Northrop（《Software Product Lines: Practices and Patterns》，Addison-Wesley, 2001）论述了支持软件产品线的体系结构设计。对于给定的问题域，Clements和他的同事（《Evaluating Software Architectures》，Addison-Wesley, 2002）考虑了如何对选择的体系结构进行评估及体系结构的最佳选择等问题。

体系结构的专门实现方面的书籍讲述了在特定的开发环境或技术条件下的体系结构设计问题。Wallnau和他的同事（《Building Systems from Commercial Components》，Addison-Wesley, 2001）介绍了构造基于构件的体系结构的方法。Pritchard（《COM and CORBA Side-by-Side》，Addison-Wesley, 1999）、Mowbray（《CORBA Design Patterns》，Wiley, 1997）及Mark等人（《Object Management Architecture Guide》，Wiley, 1996）为CORBA分布式应用支撑框架提供了详细的设计指南。Shanley（《Protected Mode Software Architecture》，Addison-Wesley, 1996）为设计基于PC的实时操作系统、多任务操作系统或设备驱动程序提供了体系结构设计指南。

会议论文集《Proceedings of the International Workshop on Software Architecture》（由

ACM和其他计算组织资助)和《Proceedings of the International Conference on Software Engineering》每年都记载当前软件体系结构的研究现状。Barroca和Hall[BAR00]介绍了最新研究方面的有效调查数据。

数据建模是良好数据设计的先决条件。Teory (《Database Modeling and Design》, Academic Press, 1998)、Schmidt (《Data Modeling for Information Professionals》, Prentice-Hall, 1998)、Bobak (《Data Modeling and Design for Today's Architectures》, Artech House, 1997)、Silverston 和Graziano及Inmon (《The Data Model Resource Book》, Wiley, 1997)、Data[DAT00]以及Reingruber和Gregory (《The Data Modeling Handbook: A Best-Practice Approach to Building Quality Data Models》, Wiley, 1994) 的著作包括数据建模符号、启发式规则和数据库设计方法的详细介绍。近年来, 数据仓库的设计已变得越来越重要。Humphreys和Hawkins及Dy (《Data Warehousing: Architecture and Implementation》, Prentice-Hall, 1999)、Kimball等人 [KIM98]及Inmon[INM95]的著作非常详细地覆盖了该主题。

在大多数软件工程书籍中都可以找到软件设计的全面论述及体系结构与数据设计问题的讨论。在Feijs (《A Formalization of Design Methods》, Prentice-Hall, 1993)、Witt等人 (《Software Architecture and Design Principles》, Thomson Publishing, 1994) 及Budgen (《Software Design》, Addison-Wesley, 1994) 的著作中, 可以找到对该主题更严格的论述。

在Myers[MYE78]、Yourdon和Constantine[YOU79]及Page-Jones (《The Practical Guide to Structured Systems Design》, 2nd ed., Prentice-Hall, 1988) 的著作中, 可以找到面向数据流设计的完整描述。这些著作致力于设计, 并提供数据流方法的全面指导。

大量的体系结构设计的信息资源可以在因特网上获得。最新的与体系结构设计相关的WWW参考文献可在SEPA Web站点<http://www.mhhe.com/pressman>上找到。

323



第11章 构件级设计建模

要点浏览

概念：一套完整的软件构件是在体系结构设计过程中定义的。但是没有在接近代码的抽象级上表示内部数据结构和每个构件的处理细节。构件级设计定义了数据结构、算法、接口特征和分配给每个软件构件的通信机制。

人员：软件工程师完成构件级设计。

重要性：必须能够在建造软件之前就确定该软件是否可以工作。为了保证设计的正确性，以及与早期设计表示（即数据、体系结构和接口设计）的一致性，构件级设计需要以一种可以评审设计细节的方式来表示软件。它提供了一种评估数据结构、接口和算法是否能够工作的方法。

步骤：数据、体系结构和接口的设计表示构成了构件级设计的基础。每个构件的类定义或者处理叙述都转化为一种详细设计，该设计采用图形或基于文本的形式来详细说明内部的数据结构、局部接口细节和处理逻辑。设计符号包括UML图和一些辅助表示。通过使用一系列结构化编程结构来说明程序的设计。

工作产品：每个构件的设计都以图形的、图表的或基于文本的方式表示，这是构件级设计阶段产生的主要工作产品。

质量保证措施：采用设计走查或审查机制。对设计执行检查以确定数据结构、接口、处理顺序和逻辑条件等是否都正确，并且给出早期设计中与构件相关的数据或控制变换。

关键概念
内聚性
构件
传统的
面向对象的
过程相关的
耦合性
设计
指导准则
图形表示法
原则
任务
中间件
OCL
封装原理
PDL
结构化编程

体系结构设计第一次迭代完成之后，就应该开始构件级设计。在这个阶段，全部的数据和软件的程序结构都已经建立起来。其目的是把设计模型转化为运行软件。但是现有设计模型的抽象层次相对较高，而可运行程序的抽象层次相对较低。这种转化具有挑战性，因为可能会在软件过程后期阶段引入难于发现和改正的微小错误。Edsger Dijkstra（有助于我们理解软件设计技术的主要贡献者）在其著作[DIJ72]中写道：

“软件似乎不同于很多其他产品，对那些产品而言，一条规则是：更高的质量意味着更高的价格。那些想要真正可靠软件的人们，将发现他们必须找到某种方法来避免开始时的大多数错误，结果，程序设计过程将变得更加容易……高效率的程序员……不应该将他们的时间浪费在调试上——他们在开始时就不应该引入错误。”

尽管这段话是在很多年以前说的，但现在仍然适用。当设计模型被转化为源代码时，必须遵循一系列设计原则，以保证不仅能够完成转化任务，而且还能保证不在开始时就引入错误。

用编程语言表示构件级设计是可以的。其实，程序的创建是以体系结构设计模型作为指

南的。一种可供考虑的方法是使用某些能够容易转化为代码的中间表示（如图形的、表格的或基于文本的）来表示构件级设计。无论采用何种机制来表示构件级设计，定义的数据结构、接口和算法应该遵守各种已经精心规定好的设计指导准则，以避免在过程设计（procedural design）演化中犯错误。本章将讨论这些设计指导准则和完成设计的方法。

11.1 什么是构件

通常来讲，构件是计算机软件中的一个模块化的构造块。再正式一点，OMG UML规范[OMG01]中将构件定义为“系统中某一定型化的、可配置的和可替换的部件，该部件封装了实现并暴露一系列接口”。

正如第10章的讨论，构件存在于软件体系结构中，因而构件在完成所建系统的需求和目标中起了重要作用。由于构件驻留于软件体系结构的内部，它们必须与其他的构件和存在于软件边界以外的实体（如其他系统、设备和人员）进行通信和合作。

“细节不仅是细节，它们构成了设计。”

——Charles Eames

术语“构件”的实际意义，不同软件工程师可能见解不同。在接下来的几节中，我们将了解三个关于“什么是构件以及在设计建模中如何使用构件”的重要观点。

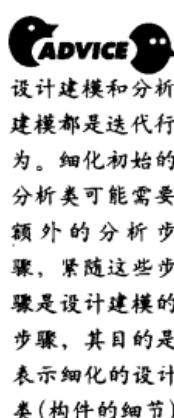
325

11.1.1 面向对象的观点



在面向对象软件工程环境中，构件包括一个协作类集合¹。构件中的每一个类都被详细阐述，包括所有的属性和与其实现相关的操作。作为细节设计的一部分，所有与其他设计类相互通信协作的接口（消息）必须予以定义。为了完成这些，设计师从分析模型开始，详细描述分析类（对于构件而言该类与问题域相关）和基础类（对于构件而言该类为问题域提供了支持性服务）。

为了说明设计细化过程，考虑为一个高级印刷车间构建软件。软件的目的是为了收集前台的客户需求，对印刷业务进行定价，然后把印刷任务交给自动生产设备。在需求工程中得到了一个被称为**PrintJob**的分析类。分析过程中定义的属性和操作在图11-1的左上方给出了注释。在体系结构设计中，**PrintJob**被定义为软件体系结构的一个构件，用简化的UML符号表示的该构件显示在图11-1中部靠右的位置。由图11-1所示，**PrintJob**有两个接口：**computeJob**和**initiateJob**。**computeJob**具有对任务进行定价的功能，**initiateJob**能够把任务传给生产设备。这两个接口在图下方的左边给出（即所谓的棒棒糖式符号）。



构件级设计将由此开始。必须对**PrintJob**构件的细节进行细化，以提供指导实现的充分信息。通过不断补充作为构件**PrintJob**的类的全部属性和操作，来逐步细化最初的分析类。正如图11-1右下部分的描述，**PrintJob**类的细化设计包含更多的属性信息和构件实现所需要的更广泛的操作描述。**computeJob**和**initiateJob**接口隐含着与其他构件（图中没有显示出来）的通信和协作。例如，**computePageCost()**（**computeJob**接口的组成部分）操作可能

¹ 在一些情况下，一个构件可以包含单独一个类。

与包含任务定价信息的**PricingTable**构件进行协作。`checkPriority()` (`initiateJob`接口的组成部分) 操作可能与**JobQueue**构件进行协作，用来判断当前等待产生的任务类型和优先级。

定义为体系结构设计组成部分的每一个构件都要实施细化。细化一旦完成，要对每一个属性、每一个操作和每一个接口进行更进一步的细化。适合每个属性的数据结构必须予以详细说明。另外还要说明实现与操作相关的处理逻辑的算法细节，在这一章的后半部分将要对这种过程 (procedural) 设计活动进行讨论。最后是实现接口所需机制的设计。对于面向对象软件，还包含对实现系统内部对象间消息通信机制的描述。

326

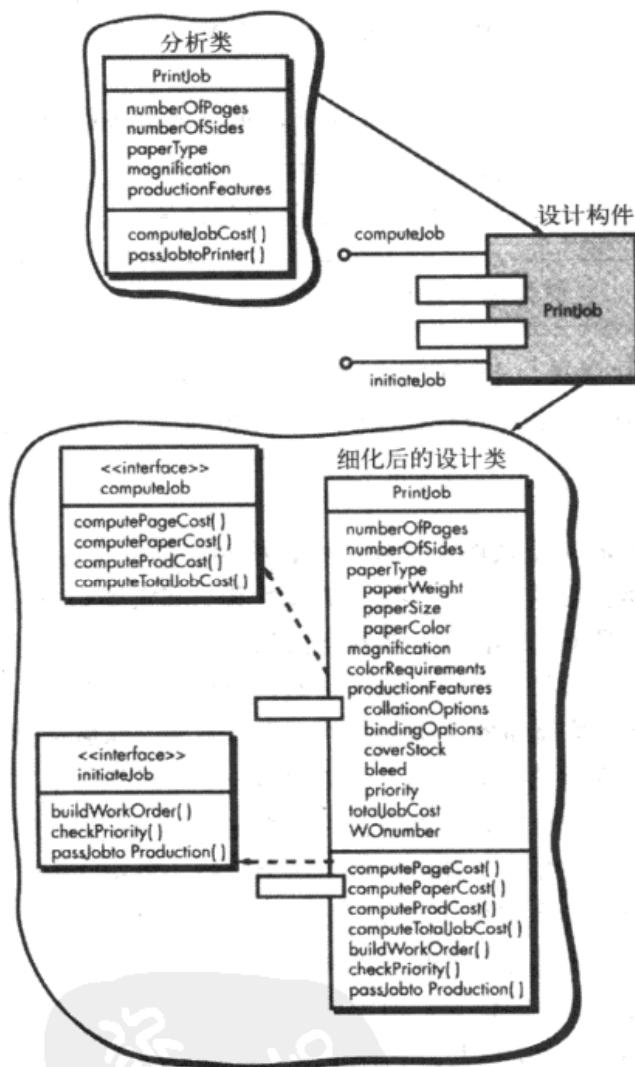


图11-1 设计构件的细化

11.1.2 传统观点

在传统软件工程环境中，一个构件就是程序的一个功能要素，程序由处理逻辑及实现处理逻辑所需的内部数据结构以及能够保证构件被调用和实现数据传递的接口构成。传统构件也被称为模块，作为软件体系结构的一部分，它承担如下三个重要角色之一：(1) 控制构件，协调问题域中所有其他构件的调用；(2) 问题域构件，完成部分或全部用户的需求；(3) 基础设施构件，负责完成问题域中所需相关处理的功能。

与面向对象的构件相类似，传统的软件构件也来自于分析模型。不同的是在该种情况下，是以分析模型中的数据流要素作为导出构件的基础。数据流图（第8章）最低层的每个变换（泡泡）都被映射为某一层次上的模块（10.6节）。一般来讲，控制构件（模块）位于层次结构（体系结构）顶层附近，而问题域构件则倾向位于层次结构的底层。为了获得有效的模块化，在构件细化的过程中采用了功能独立性的设计概念（第9章）。

“我们总是发现可运转的复杂系统都是由可运转的简单系统发展而来。” ——John Gall

为了说明传统构件的细化过程，我们再来考虑为一个高级影印中心构建软件。在分析模型建立过程中导出一组数据流图。假设这些数据流图已经被映射到图11-2中所显示的体系结构（10.6节）中。图中每个方框都表示一个软件构件。带阴影的方框相当于11.1.1节讨论的**PrintJob**类定义的操作功能。然而，在这种情况下，每个操作都被表示为如图11-2所示的能够被调用的单独模块。其他的模块用来控制处理过程，也就是前面提到的控制构件。

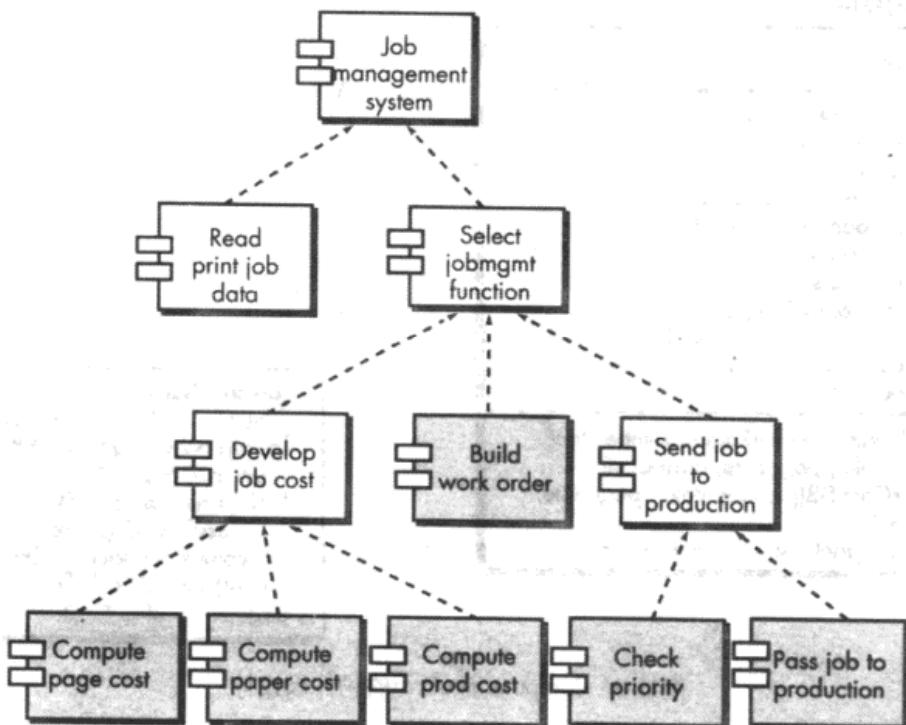


图11-2 一个传统系统的结构图

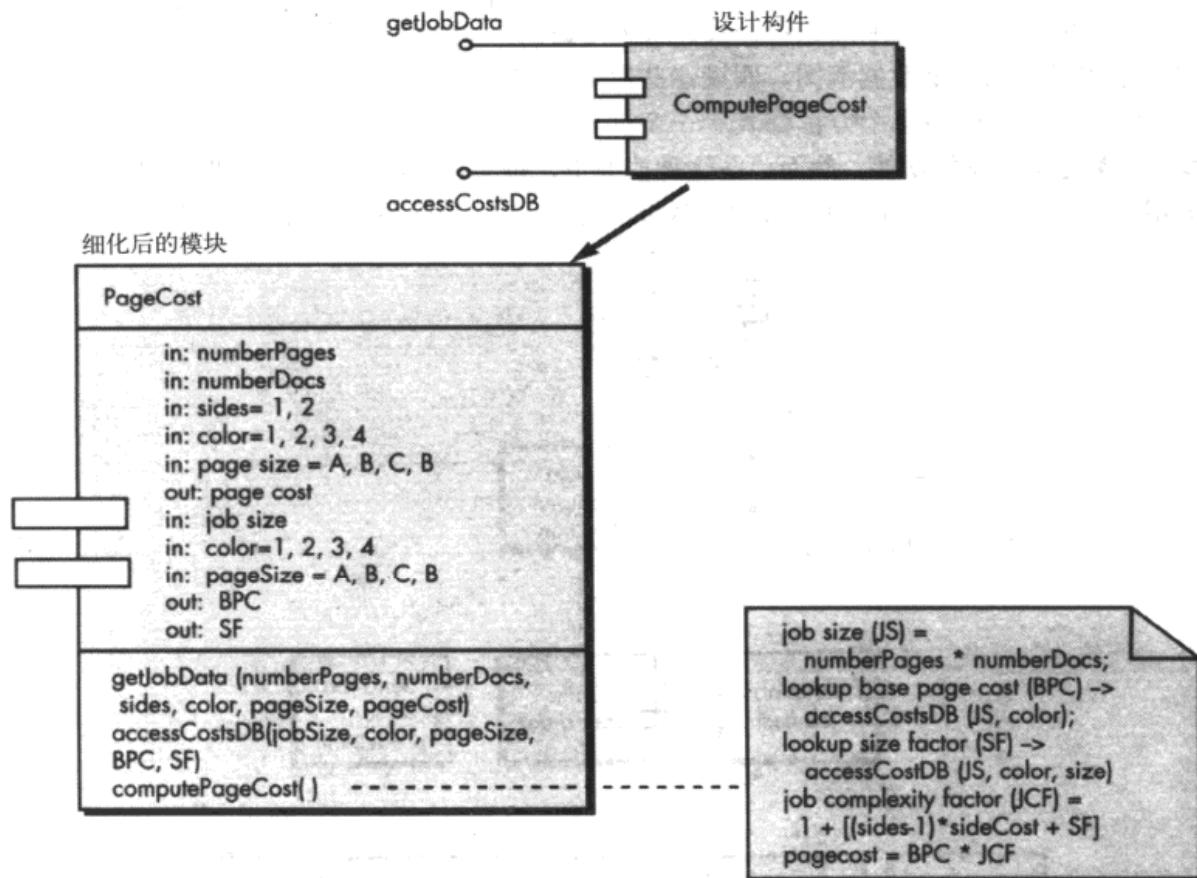
ADVICE
在软件构件设计细化中、设计的焦点转向特定数据结构和操作数据结构的过程设计上。然而，还不要忘了体系结构容纳构件或者服务于多数构件的全局数据结构。

在构件级设计中，图11-2中的每个模块都要被细化。需要明确定义模块的接口，即每个经过接口的数据或控制对象都需要明确加以说明。定义模块内部使用的数据结构。采用第9章讨论的逐步求精方法设计完成模块中相关功能的算法。有时候需要用状态图表示模块行为。

为了说明这个过程，考虑**ComputePageCost**模块。该模块的目的在于根据用户提供的规格说明来计算每页的印刷费用。为了实现该功能需要以下数据：文档的页数，文档的印刷份数，单面或者双面印刷，颜色，纸张大小。这些数据通过该模块的接口传递给**ComputePageCost**。**ComputePageCost**根据任务量和复杂度，使用这些数据来决定一页的费用——这是一个通过接口将所有数据传递

给模块的功能。每一页的费用与任务的大小成反比，与任务的复杂度成正比。

图11-3给出了使用UML建模符号描述的构件级设计。其中ComputePageCost模块通过调用getJobData模块（它允许所有相关数据都传递给该构件）和数据库接口accessCostDB（它能够使该模块访问存放所有印刷费用的数据库）来访问数据。接着，对ComputePageCost模块进一步细化，给出算法和接口的细节描述（图11-3）。其中算法的细节可以由图中显示的伪代码或者UML活动图来表示。接口被表示为一组输入和输出的数据对象或者数据项的集合。设计细化的过程需要一直进行到获得构件的导向结构为止。



329

图11-3 ComputePageCost的构件级设计

11.1.3 过程相关的观点

上面提及的关于构件级设计的面向对象观点和传统观点，都假定从头开始设计构件。也就是说，设计者必须根据从分析模型中导出的规格说明创建新构件。当然，还存在另外一种方法。

在过去的10年间，软件工程已经开始强调使用已有构件来构造系统的必要性。实际上，软件工程师在设计过程中可以使用已经过验证的设计或代码级构件的目录。当软件体系结构设计完后，就可以从目录中选出构件或者设计模式，并用于组装体系结构。由于这些构件是根据复用思想来创建的，所以其接口的完整描述、要实现的功能和需要的通信与协作等对于设计者来说都是可以得到的。在第30章将对基于构件的软件工程进行更细致的讨论。

SOFTWARE TOOLS

中间件和基于构件的软件工程

导致中间件和基于构件的软件工程 (Component-Based Software Engineering, CBSE) 成功或者失败的重要因素之一就是中间件的可用性。中间件是一组基础构件的集合，这些基础构件使得问题域构件与其他构件通过网络进行通信，或者在一个复杂的系统中通信。对于想用基于构件的软件工程方法进行开发的软件工程师来讲，他们可以使用如下三个完整的标准：

OMG CORBA (<http://www.corba.org/>)。

Microsoft COM (<http://www.microsoft.com/com/tech/complus.asp>)。

Sun JavaBeans (<http://java.sun.com/products/ejb/>)。

上面提到的网站上提供了大量的教材、白皮书、工具和关于这些重要中间件标准的广泛资源。在第30章，将会看到更多关于CBSE的信息。

11.2 设计基于类的构件

正如前面提到的，构件级设计利用了分析模型（第8章）开发的信息和体系结构模型（第10章）表示的信息。当选择了面向对象软件工程方法之后，构件级设计主要关注分析类的细化（特定的问题域类）和基础类的定义和精化。这些类的属性、操作和接口的详细描述是开始构建活动之前所需的设计细节。[330]

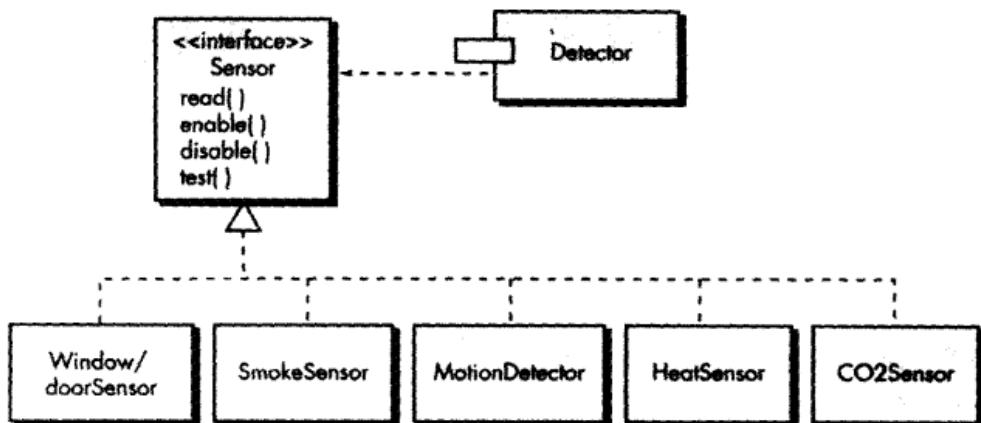
11.2.1 基本设计原则

有四种适用于构件级设计的基本设计原则，这些原则在使用面向对象软件工程方法时被广泛采用。使用这些原则的根本动机在于，使得产生的设计在发生变更时能够适应变更并且减少副作用的传播。设计者以这些原则为指导进行软件构件的开发。

开关原则 (The Open-Closed Principle, OCP)。“模块应该对外延具有开放性，对修改具有封闭性” [MAR00]。这段话似乎有些自相矛盾，但是它却体现出一个优秀的构件级设计应该具有的最重要特征。简单地说，设计者应该采用一种无需对构件自身内部（代码或者内部逻辑）做修改就可以进行扩展（在构件所确定的功能域内）的方式来说明构件。为了达到这个目的，设计者在那些可能需要扩展的功能与设计类本身之间分离出一个缓冲区。

例如，假设SafeHome的安全功能使用了对各种类型安全传感器进行状态检查的**Detector**类。随着时间的推移，安全传感器的类型和数量将会不断增长。如果内部处理逻辑采用if-then-else的结构顺序来实现，其中每个这样的结构都负责一个不同的传感器类型，那么对于新增加的传感器类型，就需要增加额外的内部处理逻辑（依然是if-then-else结构），而这显然违背OCP原则。

图11-4中表明了一种遵循OCP原则实现**Detector**类的方法。对于各种不同的传感器，接口都向**Detector**构件呈现一致的视图。如果要添加新类型的传感器，那么对**Detector**类（构件）无需进行任何改变。这个设计遵守了OCP原则。



[331]

图11-4 遵循OCP原则

SAFEHOME**OCP的应用**

[场景] Vinod的小房间。

[人物] Vinod和Shakira, SafeHome 软件工程团队成员。

[对话]

Vinod: 我刚刚接到Doug (团队经理) 的一个电话, 他说市场营销人员想增加一个新的传感器。

Shakira (假笑): 哎呀, 别在加了。

Vinod: 是啊……你永远不会相信这些家伙都提出了什么。

Shakira: 确实令我很吃惊。

Vinod (大笑): 他们称之为小狗焦虑传感器 (doggie angst sensor)。

Shakira: 那是什么装置?

Vinod: 这个装置是为了那些想把宠物留在彼此相邻很近的公寓、门廊或房子里的人设计的。狗叫致使邻里生气和抱怨, 有了这种传感器, 如果狗的叫声超过一定时间 (比如一分钟), 传感器就会向主人的手机发送特殊的报警信号。

Shakira: 你在骗我吗?

Vinod: 不是, Doug想知道在安全功能中加入这个功能需要多长时间。

Shakira (想了想): 不用多长时间……瞧。(她给Vinod看图11-4。) 我们分离出在**sensor**接口背后的实际的传感器类。只要我们有小狗传感器的规格说明, 那么把它加入其中就是一件简单的事情了。我们要做的就是为其创建一个控制构件……哦, 类。根本不用改变**Detector**构件。

Vinod: 好的, 我将告诉Doug这不是什么大问题。

Shakira: 告诉Doug, 直到下一个版本发布之前, 我们都要集中精力完成小狗焦虑传感器的事情。

Vinod: 这不是件坏事, 而如果他想让你做你可以马上实现吗?

Shakira: 是啊, 我们的接口设计使得我可以毫无困难地完成它。

Vinod (想了想): 你听说过“开关”原则吗?

Shakira (耸了耸肩膀): 没有。

Vinod (微笑): 不成问题。

Liskov替换原则 (Liskov Substitution Principle, LSP)。“子类可以替换它们的基类” [MAR00]。最早提出该设计原则的Barbara Liskov[LIS88]建议，将子类传递给构件来代替基类时，使用基类的构件应该仍然能够正确完成其功能。LSP原则要求源自基类的任何子类必须遵守基类与使用该基类的构件之间的隐含约定。在这里的讨论中，“约定”既是前置条件——构件使用基类前必须为真；又是后置条件——构件使用基类后必须为真。当设计者创建了导出子类，则这些子类必须遵守前置条件和后置条件。



如果你不做设计
只是开发代码，
单单记得代码是
最终的开发实体，
那么你就违背了DIP原则。

依赖倒置原则 (Dependency Inversion Principle, DIP)。“依赖于抽象，而非具体实现” [MAR00]。正如我们在OCP中讨论的那样，抽象可以比较容易地对设计进行扩展，又不会导致大的混乱。构件依赖的具体构件（不是依赖抽象类，如接口）越多，其扩展起来就越困难。

接口分离原则 (Interface Segregation Principle, ISP)。“多个用户专用接口比一个通用接口要好” [MAR00]。多个客户构件使用一个服务器类提供的操作的实例有很多。ISP原则建议设计者应该为每一个主要的客户类型都设计一个特定的接口。只有那些与特定客户类型相关的操作，才应该出现在该客户的接口说明中。如果多个客户要求相同的操作，则这些操作应该在每一个特定的接口中都加以说明。

332

例如，假设**FloorPlan**类用在SafeHome的安全和监督功能中。对于安全功能，**FloorPlan**只有在配置活动中使用，并且使用placeDevice()、showDevice()、groupDevice()和removeDevice()等操作实现在建筑平面图中放置、显示、分组和删除传感器。SafeHome监督功能除了需要这四个有关安全的操作之外，还需要特殊的操作showFOV()和showDeviceID()来管理摄像头。因此，ISP建议为来自SafeHome功能的两个客户端构件定义特殊的接口。安全接口应该只包括palceDevice()、showDevice()、groupDevice()和removeDevice()四种操作。监视接口应该包括placeDevice()、showDevice()、groupDevice()、removeDevice()、showFOV()和showDeviceID()六种操作。

尽管构件级设计原则提供了有益的指导，但构件自身不能够独立存在。在很多情况下，单独的构件或者类被组织进子系统或包中。于是我们很自然地就会问这个包会有怎样的活动。在设计过程中如何正确组织这些构件？Martin在[MAR00]中给出了在构件级设计中可以应用的另外一些打包原则：



为了设计可复用的构件，需要很多好的专业设计。同时，也需要有效的配置控制机制(第27章)。

发布复用等价性原则 (Release Reuse Equivalency Principle, REP)。“复用的粒度就是发布的粒度” [MAR00]。当类或构件被设计用以复用时，在可复用实体的开发者和使用者之间就建立了一种隐含的约定关系。开发者承诺建立一个发布控制系统，用来支持和维护实体的各种老版本，同时用户缓慢地将其升级到最新版本。明智的方法是将可复用的类分组打包成能够管理和控制的包作为一个更新的版本，而不是对每个类分别进行升级。

共同封装原则 (Common Closure Principle, CCP)。“一同变更的类应该合在一起” [MAR00]。类应该根据其内聚性进行打包。也就是说，当类被打包成设计的一部分时，它们应该处理相同的功能或者行为域。当域的一些特征必须变更时，只有那些包中的类才有可能需要修改。这样可以进行更加有效的变更控制和发布管理。

[333] 共同复用原则 (Common Reuse Principle, CRP)。“不能一起复用的类不能被分到一组” [MAR00]。当包中的一个或者多个类变更时，包的发布版本数量也会发生变更。所有那些依赖于已经发生变更的包的类或者包，都必须升级到最新的版本，并且都需要进行测试以保证新发布的版本能够无故障运转。如果类没有根据内聚性进行分组，那么这个包中与其他类无关联的类有可能会发生变更，而这往往会导致进行没有必要的集成和测试。因此，只有那些一起被复用的类才应该包含在一个包中。

11.2.2 构件级设计指导方针

除了11.2.1节中讨论的原则之外，在构件级设计的进程中还可以使用一系列实用的设计指导方针。这些指导方针可以应用于构件、构件的接口，以及对于最终设计有着重要影响的依赖和继承特征等方面。Ambler[AMB02]给出了如下的指导方针：

 我们在命名
构件的时候，
应该考虑些
什么呢？

构件。对那些已经被确定为体系结构模型一部分的构件应该建立命名约定，并对其进行进一步的细化和精化，使其成为构件级模型的一部分。体系结构构件的名字来源于问题域，并且应该能够被查看体系结构模型的所有共利益者理解。例如，无论技术背景如何，FloorPlan这个类的名称对于任何阅读它的人来说都是有意义的。另一方面，基础构件或者细化后的构件级类应该能够反映其实现意义的名称来命名。如果对一个作为FloorPlan实现一部分的链表进行管理时，操作manageList()是一个合适的名称，因为即使是技术人员也不会曲解它²。

在详细设计层面使用构造型帮助识别构件的特性也很有价值。例如，<<infrastructure>>可以用来标识基础构件；<<database>>可以用来标识服务于一个或多个设计类或者整个系统的数据库；<<table>>可以用来标识数据库中的一个表。

接口。接口提供关于通信和协作的重要信息（也可以帮助我们实现OCP原则）。然而，接口表示的随意性会使构件图趋于复杂化。Ambler[AMB02]建议：(1) 当构件图变得复杂时，在较正式的UML框和虚箭头记号方法中使用接口的棒棒糖式记号（这里指的是类似图11-1中的接口表示。——译者注）；(2) 为了保持一致，接口都放在构件框的左边；(3) 即使其他的接口也适用，也只表示出那些与构件相关的接口。这些建议意在简化UML构件图，使其易于查看。

[334] **依赖与继承。**为了提高可读性，依赖关系是自左向右，继承关系是自下（导出类）而上（基类）。另外，构件之间的依赖关系通过接口来表示，而不是采用“构件到构件”的方法来表示。遵照OCP的思想，这种方法使得系统更易于维护。

11.2.3 内聚性

在第9章中，我们把内聚性描述为构件的专诚性 (single-mindedness)。在为面向对象系统进行构件级设计中，内聚性 (cohesion) 意味着构件或者类只封装那些相互关联密切，以及与构件或类自身有密切关系的属性和操作。Lethbridge和Lagani  re[LET01]定义了许多不同类型的内聚性（按照内聚性的级别排序³）：

² 那些市场营销或者客户组织的人员（非技术类型）不大会查看详细设计信息。

³ 一般来讲，内聚性的级别越高，构件就越容易实现、测试和维护。



尽管理解各种级别的内聚性很有益，但是更重要的是在设计构件的时候对内聚性有全面的理解，尽可能保持高内聚性。

功能内聚。主要通过操作来表现，当一个模块完成一种且只一种运算并返回结果时发生这个级别上的内聚。

分层内聚。由包、构件和类来表现。高层能够访问低层的服务，但低层不能访问高层。例如，如果警报响起，SafeHome的安全功能需要打出一个电话。可以定义如图11-5所示的一组分层包，带阴影的包中包含基础构件。访问都是从Control panel包向下的。

通信内聚。访问相同数据的所有操作被定义在一个类中。一般说来，这些类只着眼于数据的查询、访问和存储。

那些体现出功能、层和通信等内聚性的类和构件，相对来说易于实现、测试和维护。设计者应该尽可能获得这些级别的内聚性。然而当遇到下面较低的内聚级别时，有很多实例（可供参考）。

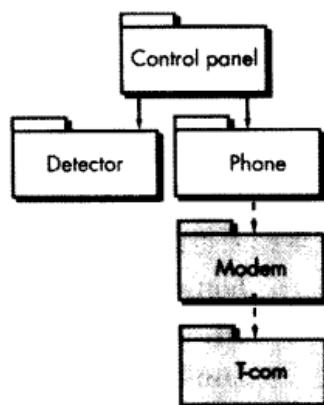
顺序内聚。将构件或者操作按照前者为后者提供输入的方式组合，目的在于实现一个操作的序列。

过程内聚。构件或者操作的组合方式是，允许在调用前面的构件或操作之后，马上调用后面的构件或操作，即使两者之间没有数据进行传递。

暂时内聚。操作的执行是为了反映某一指定的行为或状态，例如在启动时要执行的某一操作或者在错误检测时所要执行的全部操作。

实用内聚。在一类里，但是在其他方面不相关的构件、类或操作被分成一组。例如，如果Statistics类包含计算6个简单统计度量所需的所有属性和操作，那么这个类就表现出实用内聚性。

当存在其他设计选择时，上述级别的内聚性是不希望看到和应该避免的。然而，需要强调的是，实际的设计和实现问题有时会迫使设计者选择低级别的内聚性。



335

图11-5 分层内聚

内聚性的应用

SAFEHOME

[场景] Jamie的小房间。

[人物] Jamie和Ed，SafeHome 软件工程团队成员，正在实现监视功能。

[对话]

Ed: 我已经完成了**camera**（摄像头）构件的初步设计。

Jamie: 希望快速评审一下吗？

Ed: 我想……但实际上，你最好输入一些信息。（Jamie示意他继续。）

Ed: 我们起初为**camera**构件定义了五种操作。看……（给Jamie看一张列表。）

determineType() 给出用户摄像头的类型。

translateLocation() 允许用户删除设计图上的摄像头。

displayID() 得到摄像头ID并将其显示在摄像头图标附近。

displayView() 以图形化方式给出用户摄像头的视角范围。

displayZoom() 以图形化方式给出用户摄像头的放大率。

Ed: 我分别进行了设计，并且它们非常易于操作。所以我认为，把所有的显示操作集成到一个称为**displayCamera()** 的操作中——它可以显示ID、视角和放大率等，是一个不

错的主意。你不这样认为吗?

Jamie (扮了个鬼脸): 我不敢肯定。

Ed (皱眉): 为什么? 所有这些小操作是很令人头疼的!

Jamie: 问题是当将它们集成到一起后, 我们将失去内聚性。你知道的, `displayCamera()` 操作不是专诚性的。
336

Ed (略有不快): 那又怎样? 这样做使得我们最多只需不到100行的源代码。我想实现起来也比较简单。

Jamie: 如果销售人员决定更改我们显示视角域的方式怎么办?

Ed: 我马上跳到`displayCamera()`操作并制作模块。

Jamie: 那么引起的副作用怎么办?

Ed: 什么意思?

Jamie: 也就是说你做了改正, 但是不经意之间产生了ID的显示问题。

Ed: 我没有那么笨。

Jamie: 可能没有, 但是如果两年以后某些支持人员必须构建模块怎么办。他可能并不像你一样理解这个操作, 谁知道呢, 也许他很粗心。

Ed: 所以你反对?

Jamie: 你是设计师……这是你的决定……只要确信你理解了低内聚性的后果。

Ed (想了想): 可能我们要设计一个单独的显示操作。

Jamie: 好决定。

11.2.4 耦合性

在前面关于分析和设计的讨论中, 我们知道通信和协作是面向对象系统中的基本要素。然而, 这个重要(必要)特征存在一个黑暗面。随着通信和协作数量的增长(也就是说, 随着类之间的联系程度越来越强), 系统的复杂性也随之增长了。同时, 随着系统复杂度的增长, 软件实现、测试和维护的困难也随之增大。

耦合是类之间彼此联系程度的一种定性度量。随着类(构件)相互依赖越来越多, 类之间的耦合程度亦会增加。在构件级设计中, 一个重要的目标就是尽可能保持低耦合。

有多种的方法来展现类之间的耦合。Lethbridge和Laganiére[LET01]定义了如下耦合分类:

内容耦合。当一个构件“暗中修改其他构件的内部数据”[LET01]时, 就会发生此种类型的耦合。这违反了基本设计概念当中的信息隐蔽原则。

共用耦合。当大量的构件都要使用同一个全局变量时发生此种耦合。尽管有时候这样做是必要的(例如, 设立一个在整个应用系统中都可以使用的缺省值), 但是这种耦合当进行变更时, 能够导致不可控制的错误蔓延和不可预见的副作用。

控制耦合。当操作A调用操作B, 并且向B传递了一个控制标记时, 就会发生此种耦合。接着, 控制标记将会指引B中的逻辑流程。此种形式耦合的主要问题在于B中的一个不相关变更, 往往能够导致A所传递控制标记的意义也必须发生变更。如果忽略这个问题, 就会引起错误。

印记耦合。当类B被声明为类A某一操作中的一个参数类型时会发生此种



在软件构件设计细化中, 设计的焦点转向特定数据结构和操作数据结构的过程设计上。然而, 还不要忘了体系结构容纳构件或者服务于多数构件的全局数据结构。
337

耦合。由于类B现在作为类A定义的一部分，所以修改系统就会变得更为复杂。

数据耦合。当操作需要传递较长的数据参数时就会发生此种耦合。随着类和构件之间通信“带宽”的增长以及接口复杂性的增加，测试和维护就会越来越困难。

例程调用耦合。当一个操作调用另外一个操作时就会发生此种耦合。这种级别的耦合很常见，并且常常是必要的。然而，它也确实增加了系统的连通性。

类型使用耦合。当构件A使用了在构件B中定义的一个数据类型时会发生此种耦合（例如，只要“一个类将其某个实例变量或者局部变量声明为另一个类的类型”[LET01]，都会发生类型使用耦合）。如果类型定义发生了改变，每个使用该定义的构件也必须随之改变。

包含或者导入耦合。当构件A引入或者包含一个构件B的包或者内容时，就会发生此种耦合。

外部耦合。当一个构件和基础设施构件（如，操作系统功能、数据库能力、无线通信功能等）进行通信和协作时会发生此种耦合。尽管这种类型的耦合是必要的，但是在在一个系统中应该尽量将此种耦合限制在少量的构件或者类范围中。

软件必须进行内部和外部的通信，因此，耦合是必然存在的。然而，在不可避免出现耦合的情况下，设计者应该尽力降低耦合性，并且要充分理解高耦合的后果。

SAFEHOME

耦合的应用

[场景] Shakira的小房间。

[人物] Vinod和Shakira，SafeHome软件工程团队成员，正在实现安全功能。

[对话]

Shakira: 我曾经有一个非常好的想法……之后我又考虑了一下，好像并没有那么好。最后我还是放弃了，不过我想最好和你讨论一下。

Vinod: 当然可以，是什么想法？

Shakira: 好的，每个传感器能够识别一种警报条件，对吗？

Vinod（微笑）：这就是我们称它为传感器的一个原因啊，**Shakira**。

Shakira（恼怒）：Vinod你讽刺我！你应该好好学习一下处理人际关系的技巧。

Vinod: 你刚才说？

Shakira: OK，我指的是……为什么不为每个传感器都创建一个称为makeCall()的操作，该操作能够直接和**OutgoingCall**（外出呼叫）构件协作，也就是通过**OutgoingCall**构件的接口实现协作。

Vinod（沉思着）：你的意思是让协作发生在像**ControlPanel**诸如此类的构件之外？

Shakira: 是的……但接着我又对自己说，这将意味着每个传感器对象都会与**OutgoingCall**构件相关联，而这意味着与外部世界的间接耦合……我想这样会使事情变得复杂。

Vinod: 我同意，在这种情况下，让传感器接口将信息传递给**ControlPanel**，并且让其初始化外出呼叫是一个比较好的主意。此外，不同的传感器将导致不同的电话号码。在信息改变时，你并不想让传感器存储这些信息，因为如果发生变化……

Shakira: 感觉不太对。

Vinod: 耦合设计方法告诉我们是不太对。

Shakira: 无论如何……

11.3 实施构件级设计

在本章的前半部分，我们已经知道构件级设计本质上是细化的。设计者必须将分析模型和架构模型中的信息转化为一种设计表示，这种表示提供了用来指导构建（编码和测试）活动的充分信息。当应用到面向对象系统中时，下面的步骤表示出构件级设计的典型的任务集。



如果你正在设计一个非面向对象的系统，前三步关注于数据对象的精化和处理功能（变换）的标识，这些是分析模型的一部分。

步骤1：标识出所有与问题域相对应的设计类。使用分析模型和架构模型，正如11.1.1节中所描述的那样，每个分析类和体系结构构件都要细化。

步骤2：确定所有与基础设施域相对应的设计类。在分析模型中并没有描述这些类，并且在体系结构设计中也经常忽略这些类，但是此时必须对它们进行描述。如前所述，这种类型的类和构件包括GUI构件、操作系统构件、对象和数据管理构件等。

步骤3：细化所有不能作为复用构件的设计类。详细描述实现类需要的所有接口、属性和操作。在实现这个任务时，必须考虑采用设计试探法（如构件的内聚和耦合）。

步骤3a：在类或构件的协作时说明消息的细节。分析模型中用协作图来显示分析类之间的相互协作。在构件级设计过程中，某些情况下通过对系统中对象间传递消息的结构进行说明，来表现协作细节是必要的。尽管这是一个可选的设计活动，但是其可以作为接口规格说明的前提，这些接口显示了系统中构件通信和协作的方式。

图11-6给出了前面提到印刷系统的一个简单协作图。**ProductionJob**、**WorkOrder**和**JobQueue**三个对象相互协作为生产线准备印刷作业。图中的箭头表示对象间传递的消息。在分析建模时，消息说明如图中所示。然而，随着设计的进行，消息通过下列方式的扩展语法来细化[BEN02]：

```
[guard condition] sequence expression (return value) :=
  message name (argument list)
```

其中**[guard condition]**采用对象约束语言（Object Constraint Language，OCL）⁴来书写，并且说明了在消息发出之前应该满足什么样的条件集合；**sequence expression**是一个表明消息发送顺序的整数（或其他样式的表明发送顺序的指示符，如3.1.2）；**(return value)**是由消息唤醒操作返回的信息名；**message name**表示唤醒的操作，**(argument list)**是传递给操作的属性列表。

步骤3b：为每一个构件确定适当的接口。在构件级设计中，一个UML接口是“一组外部可见的（即公共的）操作。接口不包括内部结构，没有属性，没有关联……”[BEB02]。更正式地讲，接口就是某个抽象类的等价物，该抽象类提供了设计类之间的可控连接。图11-1中给出了接口细化的实例。实际上，为设计类定义的操作可以归结为一个或者更多的抽象类。抽象类内的每个操作（接口）应该是内聚的；也就是说，它应该展示那些关注于一个有限功能或者子功能的处理。

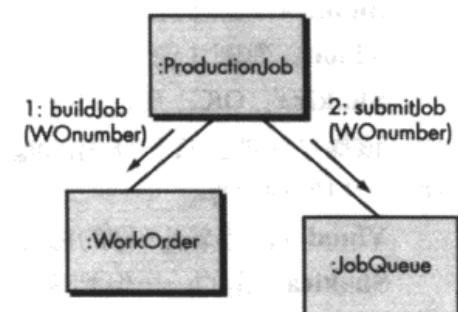


图11-6 带消息的协作图

⁴ 在11.4节和第28章中对OCL进行了简要的描述。

参照图11-1，由于initiateJob接口没有展现出足够的内聚性而受到争议。实际上，其完成了三个不同的子功能：建立工作单，检查任务的优先级，并将任务传递给生产线。接口设计应该重构。一种方法就是重新检查设计类并定义一个新类**WorkOrder**，该类的作用就是处理与装配工作单相关的所有活动。buildWorkOrder()操作成为该类的一部分。类似地，我们可能要定义一个合并了操作checkPriority()的**JobQueue**类。**ProductionJob**类包括传递给生产线的生产任务的所有相关信息。initiateJob接口将采用图11-7所示的形式。**initiateJob**现在是内聚的，集中在一个功能上。与**ProductionJob**、**WorkOrder**和**JobQueue**相关的接口都是近乎专诚的。

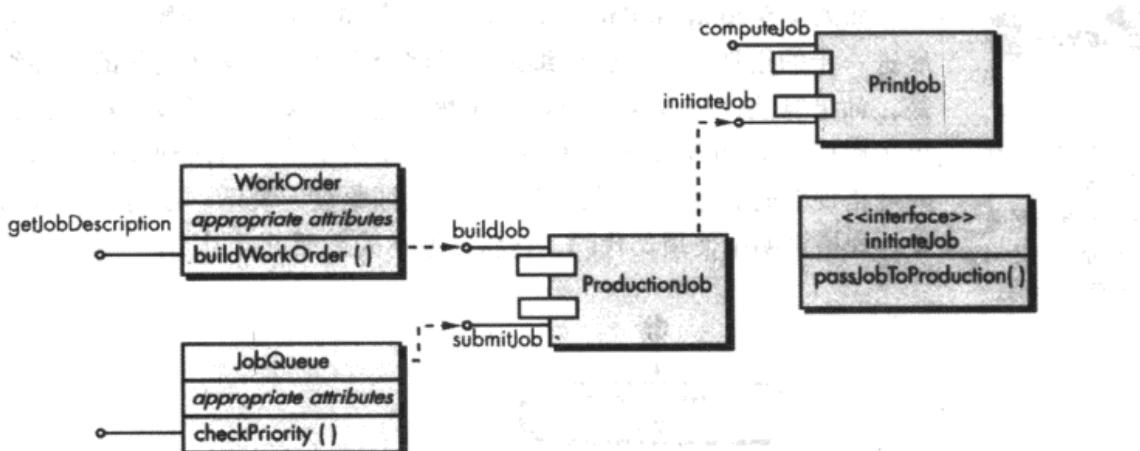


图11-7 为PrintJob重构接口和类定义

步骤3c：细化属性并且定义相应的数据类型和数据结构。一般地，描述属性的数据类型和数据结构都需要在实现时所采用的程序设计语言中进行定义。UML采用下面的语法来定义属性的数据类型：

```
name : type-expression = initial-value {property string}
```

其中**name**是属性名，**type expression**是数据类型；**initial-value**是创建对象时属性的值；**property string**用于定义属性的特征或特性。

在构件级设计的第一轮迭代中，属性通常用名字来描述。再次参考图11-1，**PrintJob**的属性列表只列出了属性名。然而，随着设计的进一步细化，使用UML的属性格式注释来定义每一个属性。例如，以下列方式来定义**paperType-weight**：

```
paperType-weight: string = "A" { contains 1 of 4 values - A, B, C, or D }
```

这里将**paperType-weight**定义为一个字符串变量，初始值为A，**paperType-weight**的取值为集合{A, B, C, D}中的一个。

如果某一属性在多个设计类中重复出现，并且其自身具有比较复杂的结构，那么最好是为这个属性创建一个单独的类。

步骤3d：详细描述每个操作中的处理流。这可能需要由基于程序设计语言的伪代码(11.5.5节)或者由UML活动图来完成。每个软件构件都需要应用逐步求精概念(第9章)通过大量的迭代进行细化。

第一轮迭代中，将每个操作都定义为设计类的一部分。在任何情况下，操作应该采用确保高内聚性的方式来刻画；也就是说，一个操作应该完成单一的目标功能或者子功能。接下去的一轮迭代，只是完成对操作名的详细扩展。例如，图11-1中的操作computePaperCost()可

以采用如下方式进行扩展：

`computePaperCost (weight, size, color): numeric`

这种方式说明`computePageCost()`要求属性`weight`、`size`和`color`作为输入，并返回一个数值（实际的钱数）作为输出。

“如果我有更多的时间，我将写一封更短的信。”

—Blaise Pascal



构件级设计细化
应该是逐步细化。
通常都会有这样
的问题：“是否存
在一种既能够简
化步骤又能达到
同样效果的方法。”

如果实现`computePaperCost()`的算法简单而且易于理解，则就没有必要开展进一步的设计细化。软件编码人员将会提供实现这些操作的必要细节。但是，如果算法比较复杂或者难于理解，此时则需要进行设计细化。图11-8给出了操作`computePaperCost()`的一个UML活动图。当活动图用于构件级设计的规格说明时，通常都在比源码更高的抽象级上表示。还有一种方法是，在设计规格说明中使用伪代码，这将在本章的后面进行讨论。

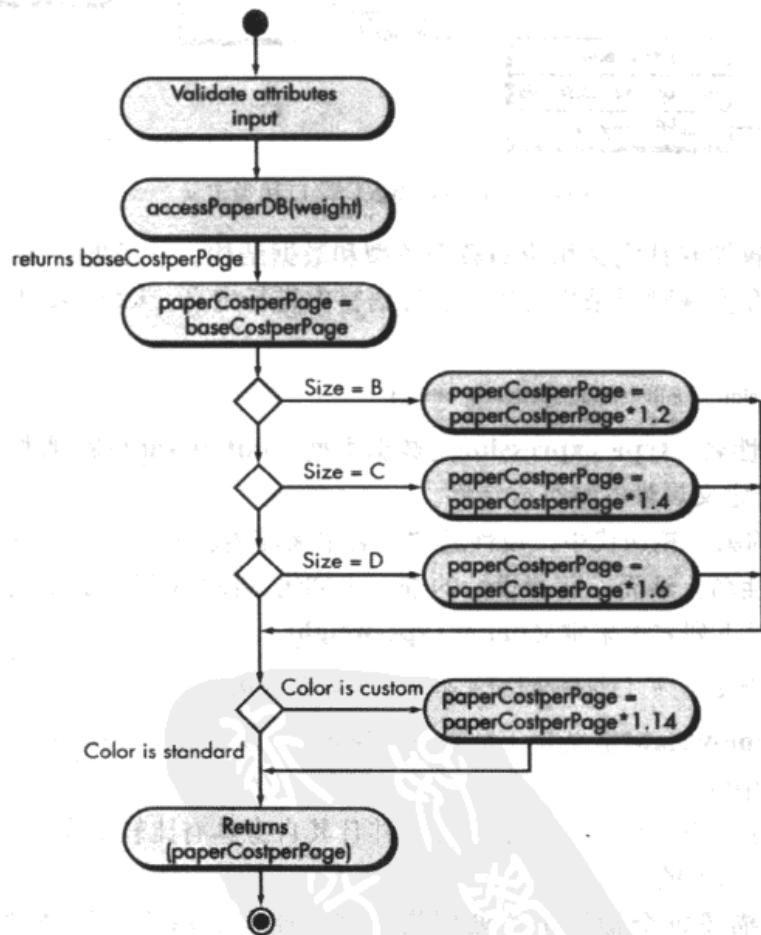


图11-8 `computePaperCost()`操作的UML活动图

步骤4：说明持久数据源（数据库和文件）并确定管理数据源所需要的类。数据库和文件通常都凌驾于单独的构件设计描述之上。在多数情况下，这些持久数据存储起初都被指定为体系结构设计的一部分，然而，随着设计细化过程的不断深入，提供关于这些持久数据源的

结构和组织等额外细节常常是有用的。

步骤5：开发并且细化类或构件的行为表示。UML状态图被用作分析模型的一部分，以表示系统的外部可观察的行为和更多的分析类个体的局部行为。在构件级设计过程中，有些时候对设计类的行为进行建模是必要的。

对象（程序执行时的设计类实例）的动态行为受到外部事件和对象当前状态（行为方式）的影响。为了理解对象的动态行为，设计者必须检查设计类生命周期中所有相关的用例，这些用例提供的信息可以帮助设计者描绘影响对象的事件，以及随着时间流逝和事件的发生对对象所处的状态。图11-9描述了使用UML状态图[BEN02]表示的状态之间的转换（由事件驱动）。

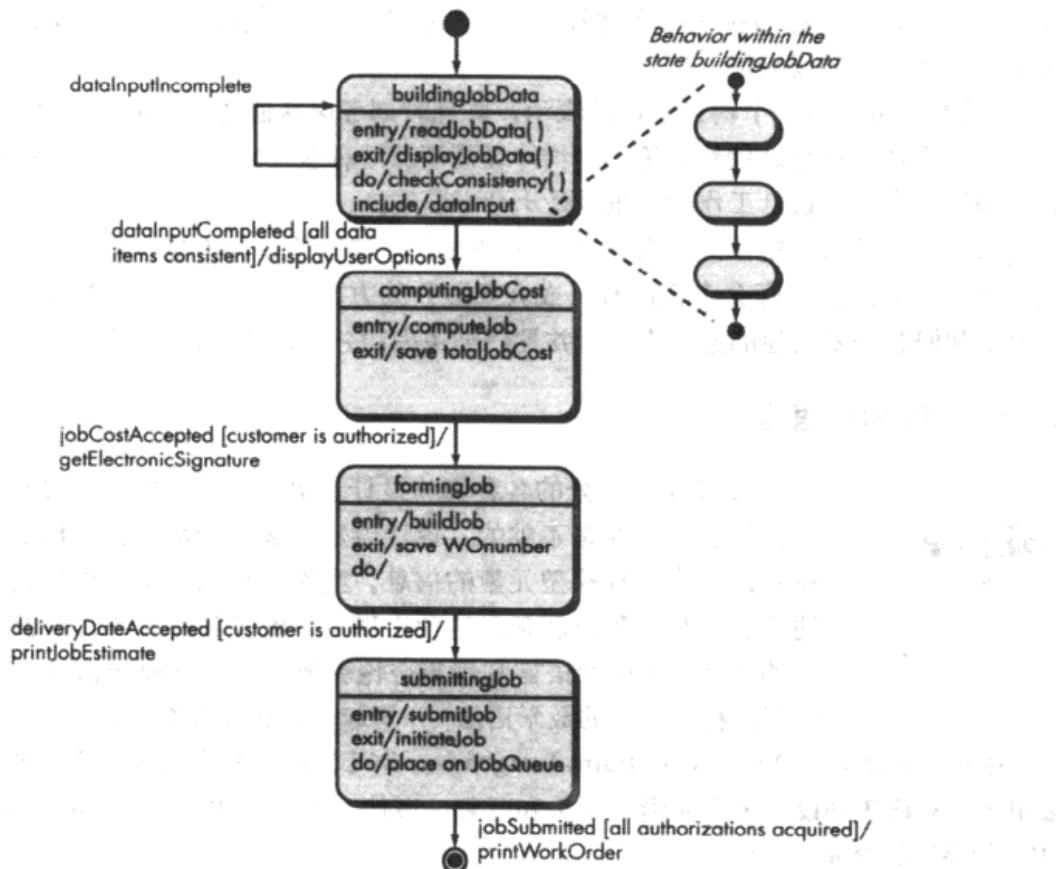


图11-9 PrintJob类的状态图

从一种状态到另一种状态的转换（用圆角矩形来表示），都表示为如下形式的事件序列：

event-name (parameter-list) [guard-condition] / action expression

其中**event-name**表示事件；**parameter-list**包含了与事件相关的数据；**guard-condition**采用对象约束语言书写，并描述了一个在事件发生前必须满足的条件，**action expression**定义了一个状态转换时发生动作。

参照图11-9，针对状态的进入和离开两种情形，每个状态都可以定义entry/ 和exit/ 两个动作。在多数情况下，这些动作与正在建模的类的相关操作相对应。do/ 指示符提供了一种机制，用来显示伴随此种状态的相关活动；而include/ 指示符则提供了通过在状态定义中嵌入更多状态图细节的方式细化的手段。

需要注意的重要一点是，行为模型经常包含一些在其他设计模型中不明显的信息。例如，

通过仔细查看图11-9中的状态图可以知道，当得出印刷任务的价钱和进度数据时，PrintJob类的动态行为取决于用户对此是否认可。如果没有同意（警戒条件（guard condition）确保用户有权审核），印刷工作就不能提交，因为不可能到达submittingJob状态。

步骤6：细化部署图以提供额外的实现细节。部署图（第9章）用作体系结构设计的一部分，并且部署图采用描述符形式来表示。在这种表示形式中，主要的系统功能（经常表现为子系统）都表示在容纳这些功能的计算环境中。

在构件级设计过程中，部署图应该被细化以表示主要构件包的位置。然而，构件一般在构件图中不被单独表示，目的在于避免图的复杂性。某些情况下，部署图在这个时候被细化成实例形式。这意味着指定的硬件和要使用的操作系统环境应加以说明，而构件包在这个环境中的位置等也需要指出。

步骤7：考虑每一个构件级设计表示，并且时刻考虑其他选择。纵观全书，我们始终强调设计是一个迭代过程。创建的第一个构件级模型总没有迭代N次之后得到的模型那么全面、一致或精确。在进行设计工作时，重构是十分必要的。

另外，设计者不能眼光狭隘。设计中经常存在其他的设计方案，在没有决定最终设计模型之前，最好的设计师会考虑所有（或大部分）的方案，运用第5章、第9章和本章介绍的设计原则和概念开发其他的选择方案，并且仔细考虑和分析这些方案。

11.4 对象约束语言



OCL提供了描述构件级设计元素的形式化文法和语法。

作为UML一部分的各种图为设计者提供了丰富的设计模型表示形式。然而，图形表示往往是不够的，设计者还需要一种机制，以明确和规范地表示那些可以约束设计模型元素的信息。当然，可以使用自然语言（如英语）来描述它，但是这种方法常常导致出现不一致性和歧义性。因此，一种更加形式化的语言——这种语言利用集合论和形式化规格说明语言（第28章）而较少使用编程语言中的数学语法——看起来应该比较合适。

345 对象约束语言（Object Constraint Language, OCL）通过允许软件工程师使用形式化的语法和文法构建各种设计模型元素（如类和对象、事件、消息、接口），利用这些无歧义描述语句作为UML的补充。

最简单的OCL语言语句由四个部分组成：（1）语境（context）定义了哪些情况语句是正确的；（2）特性（property）描述语境的一些特征（如，如果语境是一个类，那么特性可能就是一个属性）；（3）操作（operation）用来操纵和限定一个特性（如，算术运算、集合运算等）；（4）关键字（keyword）（如if, then, else, and, or, not, implies）用于说明条件表达式。

作为OCL表达式的一个简单例子，假设在PrintJob类的状态图（图11-9）中，在一个能够引起computingJobCost与formingJob之间状态转换的jobCostAccepted事件上有一个警戒条件。在图中，警戒条件以自然语言的形式进行表达，并且意味着只有当客户有权批准任务的报价时这个授权才会发生。在OCL中，可以采用如下方式来描述这个表达式：

```
customer
  self.authorizationAuthority = 'yes'
```

Customer类（实际是类的一个特定实例）的一个布尔属性authorizationAuthority必须被设为yes，才满足警戒条件。

WebRef

可以从www.omg.org下载到完整的OCL说明。

在创建设计模型时，经常有一些实例（如11.2.1节）在完成设计规定的动作之前必须满足它们的前置条件或后置条件。OCL提供了一个强有力地工具，以形式化的方式说明前置和后置条件。例如，假设印刷车间系统要进行扩展（本章中一直在讨论的例子），在说明其他印刷任务特性的同时，客户提供了印刷任务价格上限和交付的最后期限。如果印刷价格与交付时间超过其上限，任务就不能被提交并且必须通知客户。在OCL中，可以采用如下方式来说明一系列前置条件与后置条件：

```
context PrintJob::validate(upperCostBound : Integer, custDeliveryReq : Integer)
    pre: upperCostBound > 0
        and custDeliveryReq > 0
        and self.jobAuthorization = 'no'
    post: if self.totalJobCost <= upperCostBound
        and self.deliveryDate <= custDeliveryReq
    then
        self.jobAuthorization = 'yes'
    endif
```

该OCL语句定义一个不变量——也就是必须在一些行为之前（**pre**）和之后（**post**）存在的条件。开始的时候，前置条件建立了一个由客户指定的价格上限和交付期限，并且任务授权必须设置为“no”。确定价格上限和交付期限之后，将应用后置条件。还需要注意的是，表达式**self.jobAuthorization='yes'**并不是用来设置“yes”值，而是声明在操作结束时必须将**jobAuthorization**设置为“yes”。

346

对OCL的完整描述已经超出了本书的范围⁵，感兴趣的读者可以参看[WAR98]和[OMG01]以获得更多详细的内容。

SOFTWARE TOOLS**UML/OCL**

目的：很多UML工具都可以用来帮助设计师完成各个级别的设计，其中一些设计工具提供了OCL支持。

机制：这一类工具能够使设计者创建所有的UML图（构建完整设计模型所必须的）。更重要的是很多工具提供严格的语法和语义检查，以及版本控制和变更控制管理（第27章）。当工具提供了OCL支持，设计者就能够用它创建OCL表达式，并且在某些情况下，可以将它们编辑起来进行各种类型的评估和分析。

代表性工具⁶

ArgoUML，由Tigris.org发布（<http://argouml.tigris.org>），它支持全部的UML和OCL，并且包括各种辅助设计工具，除了能够生成UML图和OCL表达式之外，还提供更多功能。

Dresden OCL toolkit，由Dresden技术大学的Frank Finger开发（<http://dresdenocl.sourceforge.net/>），该工具基于一个包含了语法分析、类型检查和形式化OCL约束等模块

⁵ 但是，在第28章将进一步讨论OCL（以一种规范化的方法来描述）。

⁶ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。在大多数情况下，工具的名字由各自的开发者注册为商标。

的OCL编译器开发。

OCL parser，由IBM采用Java语言开发（<http://www-3.ibm.com/software/ad/library/standards/ocldownload.html>），为鼓励那些使用UML建模的人员使用OCL，该工具对于面向对象团体是免费的。

11.5 设计传统构件

传统软件构件⁷的构件级设计基础在20世纪60年代早期已经形成，在Edsger Dijkstra及其同事的著作（[BOH66], [DIJ65], [DIJ76]）中又得到了进一步的完善。20世纪60年代末，Dijkstra等人提出，所有的程序都可以建立在一组限定好的逻辑构造（construct）之上，这一组逻辑构造强调了“对功能域的支持”，其中每一个逻辑结构有可预测的逻辑结构（structure），从顶端进入，从底端退出，读者可以很容易地理解过程流（procedural flow）。

347



POINT
结构化编程是局限于逻辑流的设计技术，它有三种构造：顺序型、条件型和重复型。

这些逻辑构造包括顺序型、条件型和重复型。顺序型实现了任何算法规格说明中的核心处理步骤；条件型允许根据逻辑情况选择处理的方式；重复型提供了循环。这些逻辑构造是结构化编程的基础，而结构化编程是构件级设计的一种重要技术。

结构化的构造使得软件的过程设计只采用少数可预测的操作。复杂性度量（见第15章）表明，使用结构化的构造减少了程序复杂性，从而增加了可读性、可测试性和可维护性。使用有限数量的逻辑构造也符合心理学家所谓的人类“成块”理解过程。要理解这一过程，可以考虑阅读这一页的方式。读者不是阅读单个字母，而是辨认由单词或短语构成的模式或字母块。结构化的构造就是一些逻辑块，读者可以用它来辨认模块的过程成分，而不必逐行阅读设计或代码，遇到容易辨认的逻辑模式就能提高理解度。

11.5.1 图形化设计表示

在第7章、第8章和本章的前半部分我们已经讨论了UML活动图，活动图可以让设计者表示顺序、条件和重复构造（结构化编程的所有元素），它由早期的形象化设计表示（称为流程图，至今仍被广泛使用）派生而来。

348

ADVICE
结构化程序构造应该使设计更容易理解。如果使用它导致了不必要的复杂性，那就不必遵循它了。

如同活动图一样，流程图画起来很简单，方框表示处理步骤，菱形表示逻辑条件，箭头表示控制流。图11-10表示了三种结构化的构造。顺序型由两个表示处理的方框以及连接两者的控制线表示；条件型也称为if-then-else结构，由一个菱形表示，如果值为真则执行then部分，如果值为假，则调用else部分；重复型可由两种略有不同的结构表示，do while结构首先检验条件，然后重复执行循环任务，只要检验条件为真就不停止。repeat until结构首先执行循环任务，然后再检验条件，只要不满足检验条件就不停止。图中的选择型（也称为select-case）构造实际上是if-then-else的扩展，参数被连续地检验，直到有一次条件为真，其对应的case部分处理路径才被执行。

一般来说，如果需要从一组嵌套的循环或条件中退出，完全依赖结构化

⁷ 传统软件构件实现了处理元素，涉及完成问题域中的功能或子功能，或完成基础设施域中的一些能力，常常被称为模块、过程或子程序，传统的构件并不能像OO构件那样封装数据。

的构造将导致效率降低。更重要的是，退出路径上的复杂逻辑检验将会使软件的控制流不清晰，增加出错的可能，降低可读性和可维护性。如何解决这一问题呢？

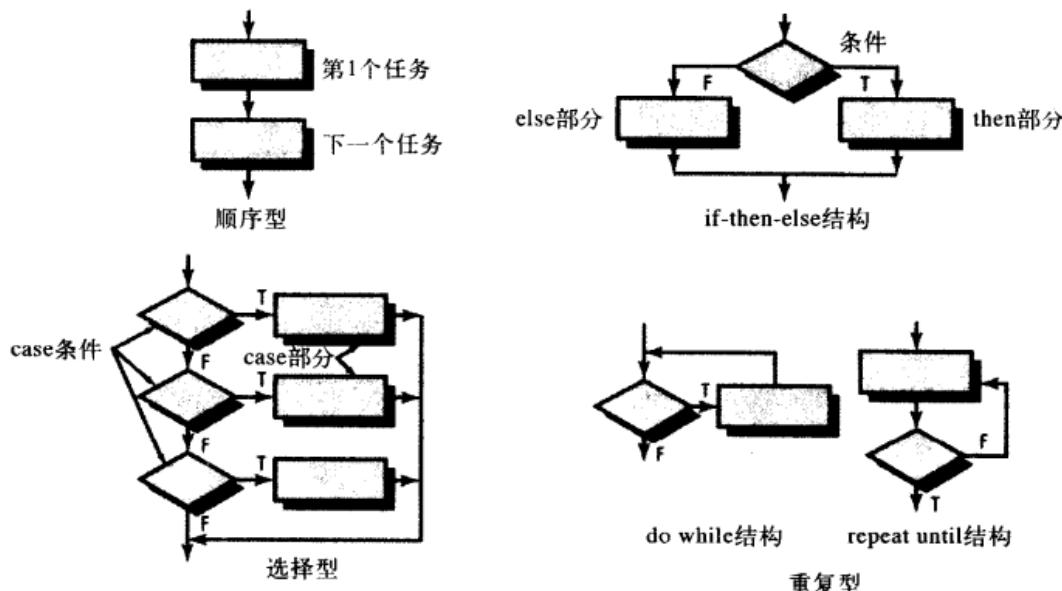


图11-10 流程图构造

设计人员有两种选择：(1) 重新设计过程表示，保证内层嵌套的控制流中不需要退出分支；(2) 以受控方式突破结构化的构造，即设计一条从嵌套层内退出的路径。第一种选择显然是最理想的，但第二种选择也不违反结构化编程的精神。

11.5.2 表格式设计表示

ADVICE
当某一构件中包含一系列复杂的条件和动作时，应该使用决策表。

在许多软件应用系统中，模块需要对复杂的组合条件求值，并根据这些条件选择要执行的动作。决策表[HUR83]提供了一种表示方法，可以将（在处理叙述中描述的）动作和条件翻译成表格，该表格很难被误解，并且能够作为某一个表驱动算法的机器可识别的输入。

决策表分为四个部分，左上部列出了所有的条件，左下部列出了所有可能的动作，右半部构成了一个矩阵，表示条件的组合以及特定条件组合对应的动作，因此矩阵的每一列可以解释成一条处理规则。下面是开发决策表的步骤：

1. 列出特定过程（或模块）相关的所有动作。
2. 列出执行该过程时的所有条件（或所做的决策）。
3. 将特定的条件组合与特定的动作相关联，消除不可能的条件组合；或者找出所有可能的条件排列。
4. 定义规则，指出一组条件应对应哪个或哪些动作。

为了说明决策表的使用，考虑下面一段从一个印刷车间系统中摘录出的非形式化用例：

我们规定了三类会员：普通会员、白银会员、黄金会员（这些类别由客户一年中在本印刷车间的业务数量决定）。普通会员享有常规的印刷和配送服务；白银会员享有8%的优惠价并且比普通会员在等待印刷中享有优先权；黄金会员享有15%的优

惠价，并且比普通会员和白银会员更具优先权。根据管理制度，除了必要的打折优惠外，管理人员还可让每个用户享受不等的折价优惠。

图11-11是上述处理信息的决策表。6条规则中的每一个都代表6个可执行的条件之一。一般，可以有效地使用决策表来补充其他方式的过程设计表达法。

11.5.3 程序设计语言

程序设计语言（Program Design Language，PDL）也称为结构化的英语或伪代码，它是“一种混合语言，采用一种语言（即英语）的词汇和另一种语言的语法（即结构化编程语言）”[CAI75]。在本章中，PDL作为设计语言仅作一般性引用。

PDL看起来像现代的编程语言，其区别在于PDL允许在自身的语句间嵌入叙述性文字（如英语），由于在有语法含义的结构中嵌入了叙述性文字，PDL不能被编译，但当使用工具可以将PDL翻译成程序设计语言“骨架”和（或）设计的图

350 形表示（如流程图），并生成嵌套图、设计操作索引、交叉引用表以及其他一些信息。



以你使用的程序语言作为PDL的基础是一个好办法。当你进行设计时，你能够创建代码骨架，该骨架附有叙述性文字。

程序设计语言是某种语言的简单转化，如Ada、C或Java，基本的PDL语法应该包括用于定义构件的构造、接口描述、数据声明、块结构、条件构造、循环构造和I/O构造。值得注意的是PDL包括多任务处理和（或）并发处理、中断处理、交互同步及其他特征的关键字。用于应用设计的PDL应该规定设计语言的最终形式，某些PDL构造的格式和语义在下面的例子中给出。

为了说明PDL的使用方法，我们以前面提到的SafeHome安全系统的过程设计为例。SafeHome系统监控火、烟、盗贼、水（洪水）和温度（比如冬天房主外出时炉子熄灭）等方面的警报；产生警报信号；调用监控服务，发出合成语音信息。在下面的PDL中，我们可以找到某些在前面几节中提到的重要构造。

考虑到PDL不是编程语言，设计人员可以随便修改而不必担心语法错误。然而，监控软件的设计应该在编写代码之前进行评审（你发现问题了吗）并精化。下面的PDL⁸提供了对早期警报管理构件过程设计版本的一个精化。

```

component alarmManagement;
The intent of this component is to manage control panel switches and input from sensors by
type and to act on any alarm condition that is encountered.
    set default values for systemStatus (returned value), all data items
    initialize all system ports and reset all hardware
    check controlPanelSwitches (cps)
        if cps = "test" then invoke alarm set to "on"
        if cps = "alarmOff" then invoke alarm set to "off"

```

条件	规则					
	1	2	3	4	5	6
普通会员	T	T				
白银会员			T	T		
黄金会员					T	T
特别折扣	F	T	F	T	F	T
动作						
无折扣		✓				
实行8%折扣			✓	✓		
实行15%折扣					✓	✓
实行特殊的x%折扣	✓		✓		✓	

图11-11 决策表

⁸ PDL描述的细节不作统一的定义。有些人选择更加面向自然语言的描述，而有些人则选择更面向代码的描述。

```

    :
    :

default for cps = none
reset all signalValues and switches
do for all sensors
    invoke checkSensor procedure returning signalValue
    if signalValue > bound [alarmType]
        then phone.message = message [alarmType]
        set alarmBell to "on" for alarmTimeSeconds
        set system status = "alarmCondition"
    parbegin
        invoke alarm procedure with "on", alarmTimeSeconds;
        invoke phone procedure set to alarmType, phoneNumber
    paren
    else skip
    endif
enddofor
end alarmManagement

```

351

注意，警报管理构件设计人员使用了parbegin…paren结构，该结构定义了一个并行模块，在parbegin模块内定义的所有任务都可以并行执行。在这里我们不考虑具体的实现。

SOFTWARE TOOLS

PDL工具

目的：尽管使用PDL或伪代码的大多数软件工程师所开发的版本往往根据自己打算用于实现的编程语言改写，但实际上存在着大量的PDL工具。

机制：在某些情况下，存在源码的逆向工程工具（一个糟糕的事实是，在实际中某些程序根本没有文档），有些情况下，工具允许设计师创建PDL并给予自动帮助。

代表性工具⁹

PDL/81，由Caine、Farber和Gordon开发 (<http://www.cfg.com/pdl81/lpd.html>)，支持使用一个已定义的PDL版本来创建设计。

DocGen，由软件改进组（Software Improvement Group）发布 (<http://www.software-improvers.com/DocGen.html>)，是一个逆向工程工具，可以将Ada和C代码生成为类PDL的文档。

PowerPDL，由Iconix开发 (<http://www.iconixsw.com/SpecSheets/PowerPDL.html>)，允许设计者建立基于PDL的设计并把伪代码转化为产生其他设计表示的形式。

11.5.4 设计表示方法的比较

设计表示方法应该能够导致一个易于理解和评审的过程表示。此外，这些表示方法还应该增强“面向代码”的能力，以便代码确实能够成为设计的一个自然副产品。最后，设计表示方法必须易于维护，使得设计总是正确地表示程序。

在讨论设计表示方法时一个自然的问题是：“在具有上面提到的属性的情况下，哪一种表

⁹ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。在大多数情况下，工具的名字由各自的开发者注册为商标。

352 示方法确实是最好的？”对这个问题的任何回答无疑都是主观的和值得争论的。然而，似乎程序设计语言提供了这些特征的最好组合。PDL可以被直接地嵌入到源程序列表中，以改善文档并减少设计维护的难度。可以由任何文本编辑程序或字处理系统完成编辑，自动处理器已经存在，并具有“自动代码生成”的潜力。

然而，这并不说明其他的设计表示方法就比PDL差，活动图和流程图的图形化特征使得设计人员更容易看清控制流。决策表的准确内容，是表驱动应用系统的理想工具。本书没有涉及其他的设计表示（如Petri网），这些描述各有其优点。总之，对设计工具的选择可能更取决于人为因素而不是技术因素。

11.6 小结

构件级的设计过程由一个慢慢降低软件描述抽象层次的活动序列组成。构件级设计最终在一个非常接近于代码的抽象层次上描述软件。

根据所开发软件的特点，可以从两种不同的角度进行构件级设计。面向对象的观点注重细化来自于问题域和基础设施域的设计类。传统的观点细化三种不同的构件或模块：控制模块、问题域模块和基础设施模块。在这两种观点中，都需要应用那些导致高质量软件的基本设计原则和概念。当从过程角度考虑时，构件级设计采用了可复用的软件构件和设计模式，这些是基于构件级的软件工程的关键要素。

面向对象的构件级设计是以类为基础的。在进行类的细化时，有许多重要的原则和概念可以指导软件工程师。如，开关原则、依赖倒置原则，以及耦合性和内聚性概念等都可以指导工程师构造具有可测性、可实现性和可维护的软件构件。在这种情况下，为了实施构件级设计，需要细化类，这通过以下方式达到：详细描述消息发送方式，确定合适的接口，细化属性并定义实现它们的数据结构，描述每个操作的处理流，表示类或构件级的行为等。在任何情况下，设计迭代都是必要的活动。

传统构件级设计需要充分表示出数据结构、接口和程序块的算法以指导编程语言源码的产生。为达到这一要求，设计者采用设计表示方法来表示构件细节，可以使用图形、表格或者文本格式。

353 结构化编程是一种过程设计思想，它限制描述算法细节时使用的逻辑构造数量和类型。结构化编程的目的是帮助设计师定义简单的算法，因而也就易于阅读、测试和维护。

参考文献

- [AMB02] Ambler, S., "UML Component Diagramming Guidelines," available at <http://www.modelingstyle.info/>, 2002.
- [BEN02] Bennett, S., S. McRobb, and R. Farmer, *Object-Oriented Analysis and Design*, 2nd ed., McGraw-Hill, 2002.
- [BOH66] Bohm, C., and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *CACM*, vol. 9, no. 5, May 1966, pp. 366–371.
- [CAI75] Caine, S. and K. Gordon, "PDL—A Tool for Software Design," in *Proc. National Computer Conference*, AFIPS Press, 1975, pp. 271–276.
- [DIJ65] Dijkstra, E., "Programming Considered as a Human Activity," in *Proc. 1965 IFIP Congress*, North-Holland Publishing Co., 1965.
- [DIJ72] Dijkstra, E., "The Humble Programmer," 1972 ACM Turing Award Lecture, *CACM*, vol. 15, no. 10, October, 1972, pp. 859–866.
- [DIJ76] Dijkstra, E., "Structured Programming," in *Software Engineering, Concepts and Tech-*

- niques*, (J. Buxton et al., eds.), Van Nostrand-Reinhold, 1976.
- [HUR83] Hurley, R. B., *Decision Tables in Software Engineering*, Van Nostrand-Reinhold, 1983.
- [LET01] Lethbridge, T., and R. Laganiere, *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*, McGraw-Hill, 2001.
- [LIS88] Liskov, B., "Data Abstraction and Hierarchy," *SIGPLAN Notices*, vol. 23, no. 5, May 1988.
- [MAR00] Martin, R., "Design Principles and Design Patterns," downloaded from <http://www.objectmentor.com>, 2000.
- [OMG01] OMG *Unified Modeling Specification*, Object Management Group, version 1.4, September, 2001.
- [WAR98] Warmer, J., and A. Klepp, *Object Constraint Language: Precise Modeling with UML*, Addison-Wesley, 1998.

习题与思考题

- 11.1 什么是警戒条件？何时使用？
- 11.2 为什么传统软件当中必要的控制构件在面向对象的软件当中一般是不需要的？
- 11.3 用自己的话来说明开关原则。为什么为构件间创建抽象的接口是重要的？
- 11.4 问题域构件不会存在外部耦合的说法有道理吗？如果你认为没有道理，那么哪种类型的构件存在着外部耦合？
- 11.5 用自己的话说明什么是依赖倒置原则。如果某个设计人员过分依赖于具体构件，那么会出现什么情况？
- 11.6 给出你最近开发的三个构件，它们之间存在着多种内聚。如果你为了获得高内聚不得不定义基本的元素，应该是哪些呢？
- 11.7 给出你最近开发的三个构件，它们之间存在着多种耦合。如果你为了获得较低的耦合不得不定义基本的元素，应该是哪些呢？
- 11.8 对基础设施构件做些研究并给出其典型分类的列表。
- 11.9 术语“构件”有时是非常难定义的。请首先给出一个一般性定义，然后给出面向对象软件和传统软件更明确的定义，最后选择三种你熟悉的编程语言说明怎样定义一个构件。[354]
- 11.10 在构件级设计中经常用到术语公有属性和私有属性。你认为它们是什么意思？它们强调了什么样的设计理念？
- 11.11 选择已有程序的某一小部分（大概50~70行源代码）。通过在源代码周围画框隔离出结构化编程构造。程序片段是否存在违反结构化编程原则的构造？如果存在，重新设计代码以使其遵守结构化编程的构造。如果不违反，对于画的框你应该注意些什么？
- 11.12 什么是持久的数据源？
- 11.13 逐步求精和分解是一回事吗？如果不是，它们有什么区别？
- 11.14 完成：(1) 一个细化的设计类；(2) 接口描述；(3) 该类中包含的某一操作的活动图；(4) 前几章讨论过的SafeHome类中的一个详细的状态图。
- 11.15 思考一下，给出11.4节中没有提及的三个或四个OCL结构或者操作符。
- 11.16 在基于类的构件级设计中，接口担当何种角色？

推荐读物与阅读信息

使用面向对象方法设计类和构件的设计原则、概念、指导方针和技巧，在很多关于面向对象软件工程和面向对象分析与设计的论著中都有涉及。其中包括：Bennett和他的同事[BEN02]、

Larman (《Applying UML and Patterns》, Prentice-Hall, 2001)、Lethridge和Laganiere[LET01]、Nicola及其同事 (《Streamlined Object Modeling: Patterns, Rules and Implementation》, Prentice-Hall, 2001)、Schach (《Object-Oriented and Classical Software Engineering, fifth edition, McGraw-Hill, 2001》)、Dennis及其同事 (《Systems Analysis and Design: An Object-Oriented Approach with UML》, Wiley, 2001)、Graham (《Object-Oriented Methods: Principles and Practice》, Addison-Wesley, 2000)、Richter (《Designing Flexible Object-Oriented Systems with UML》, Macmillan, 1999)、Stevens和Pooley (《Using UML: Software Engineering with Objects and Components, revised edition》, Addison-Wesley, 1999)、Riel (《Object-Oriented Design Heuristics》, Addison-Wesley, 1996)。

“按合约设计”概念是一个很有用的设计范型。Mitchell和McKim的著作 (《Design by Contract by Example》, Addison-Wesley, 2001)、Jezequel及其同事的著作 (《Design Patterns and Contracts》, Addison-Wesley, 1999) 都针对这个主题给出了详细介绍。Metsker (《Design Patterns Java Workbook》, Addison-Wesley, 2002)、Shalloway和Trott (《Design Patterns Explained: A New Perspective on Object-Oriented Design》, Addison-Wesley, 2001) 则考虑了设计模式对软件构件设计的影响。设计迭代对于获得高质量的设计是非常重要的。Fowler (《Refactoring: Improving the Design of Existing Code》, Addison-Wesley, 1999) 在设计演化过程中具有指导作用。

Linger、Mills和Witt的著作 (《Structured Programming—Theory and Practice》, Addison-Wesley, 1979) 仍是设计方面的权威著作，里面包含很好的PDL，以及关于结构化编程分枝的细节讨论。其他的书籍，如Robertson (《Simple Program Design, third edition》, Course Technology, 2000)、Farrell (《A Guide to Programming Logic and Design》, Course Technology, 1999)、Bentley (《Programming Pearls, second edition》, Addison-Wesley, 1999)、Dahl (《Structured Programming》, Academic Press, 1997) 等，都集中在传统系统的过程设计问题方面。

最近的一些相关书籍，已经开始单独地致力于构件级设计。一般来讲，编程语言书籍或多或少关注于过程设计，但总以书中所介绍的语言为上下文，这方面的资料有很多。

关于构件级设计的一个更加广泛的信息资源就是因特网，在SEPA的站点<http://www.mhhe.com/>

355 pressman上可以找到最新的WWW上相关文献的列表。



第12章 完成用户界面设计

要点浏览

概念：用户界面（UI）设计在人与计算机之间搭建了一个有效的交流媒介。遵循一系列的界面设计原则，定义界面对象和界面动作，然后创建构成用户界面原型基础的屏幕布局。

人员：软件工程师通过迭代过程来设计用户界面，这个过程采纳了被广泛接受的设计原则。

重要性：不管软件提供了什么样的计算能力和提供了什么样的功能，如果软件不方便使用，使用它常导致犯错，或者使用它不利于完成目标，你是不会喜欢这个软件的。由于界面影响用户对于软件的感觉，因此，它必须是令人满意的。

步骤：用户界面设计首先要识别用户、任务和环境需求。一旦用户任务被确定，则通过创建和分析用户场景来定义一组用户界面对象和动作。这是创建屏幕布局的基础。屏幕布局描述了图形设计和图标位置，描述性屏幕文本的定义，窗口的规格说明和命名，以及主要的和次要的菜单项规格说明。使用工具来开发原型并最终实现设计模型，另外为了保证质量需要对结果进行评估。

工作产品：创建用户场景，构建产品屏幕布局，以迭代的方式开发和修改界面原型。

质量保证措施：原型的开发是通过用户测试驱动的，测试驱动的反馈将用于原型的下一次迭代修改。

关键概念
可访问性
设计步骤
黄金规则
帮助设施
界面
分析
一致性
评估
模型
国际化
对象细节
模式
任务分析
可用性
工作流分析

对于房子的蓝图（其体系结构设计）来讲，如果不画出门、窗、水管、电线和电话线（更不要说有线电视的电缆），则整个设计是不完整的。对于计算机软件来说，这些“门、窗和布线”就构成了系统的接口设计。

接口设计主要包括三个方面：(1) 软件构件间的接口设计；(2) 软件与除人以外的其他非人类信息生产者和消费者（比如其他外部实体）的接口设计；(3) 人（如用户）与计算机间的界面设计。在本章，我们只着重于第三个方面的设计——用户界面设计。

Ben Shneiderman[SHN90]在其关于用户界面设计的经典著作前言中指出：

对于许多计算机化的信息系统用户来说，挫折和焦虑是他们日常生活的一部分。他们努力地学习那些被认为能够帮助他们更好地完成工作的命令语言和菜单选择系统。有些人甚至对计算机、终端和网络产生了紧张和恐惧，因而刻意回避使用计算机化的系统。

Shneiderman提到的问题并不是危言耸听。事实上图形用户界面、窗口、图标和鼠标点击已经消除了很多可怕的界面问题。但是，即使在“窗口世界”中，我们也都遇到过难学、难用、令人迷惑、不可原谅，以及在很多情形下使人感到沮丧的用户界面。然而，仍然有人花

费时间和精力去开发这样的界面，看起来，他们并不是有意地制造这些问题。

用户界面的设计要求在研究技术问题的同时对人加以研究。用户是什么样的人？用户怎样学习与新的计算机系统进行交互？用户怎样解释系统产生的信息？用户对系统有哪些期望？这些问题仅仅是在用户界面设计时必须询问和回答的问题中的一部分。

12.1 黄金规则

Theo Mandel在其关于界面设计的著作[MAN97]中提出了三条“黄金规则”：

1. 置用户于控制之下。
2. 减少用户的记忆负担。
3. 保持界面一致。

这些黄金规则实际上构成了指导用户界面设计活动的基本原则。

12.1.1 置用户于控制之下

在重要的、新的信息系统的需求收集阶段，曾经征询一位关键用户对于窗口图形界面相关属性的意见。该用户严肃地说：“我真正喜欢的是一个能够理解我想法的系统，它在我需要去做以前就知道我想做什么，并使得我可以非常容易地完成它。这就是我所想要的，我也仅此一点要求。”

我的第一反应是摇头和微笑，但是，我沉默了一会儿，认为该用户的想法绝对没有什么错。她想要一个对其要求能够作出反应并帮助她完成工作的系统。她希望去控制计算机，而不是计算机控制她。

设计者施加的大多数界面约束和限制都是为了简化交互模式。但是，这是为谁呢？在很多情况下，设计者为了简化界面的实现可能会引入约束和限制，其结果可能是界面易于构建，但会妨碍使用。

357 Mandel[MAN97]定义了一组设计原则，允许用户操作控制：

以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式。交互模式就是界面的当前状态。例如，如果在字处理菜单中选择拼写检查，则软件将转移到拼写检查模式。如果用户希望在这种情形下进行一些文本编辑，则没有理由强迫用户停留在拼写检查模式，用户应该能够几乎不需做任何动作就进入和退出该模式。

提供灵活的交互。不同的用户有不同的交互偏好，应该提供选择机会。例如，软件可能允许用户通过键盘命令、鼠标移动、数字化笔或语音识别命令等方式进行交互。但是，每个动作并非要受控于每一种交互机制之下。例如，考虑使用键盘命令（或语音输入）来画一个复杂形状是有一定难度的。

允许用户交互被中断和撤销。即使当陷入到一系列动作之中时，用户也应该能够中断动作序列去做某些其他事情（而不会失去已经做过的工作）。用户也应该能够“撤销”任何动作。

当技能级别增长时可以使交互流线化并允许定制交互。用户经常发现他们重复地完成相同的交互序列，因此，值得设计一种“宏”机制，使得高级用户能够定制界面以方便交互。

使用户与内部技术细节隔离开来。用户界面应该能够将用户移入到应用的虚拟世界中，用户不应该知道操作系统、文件管理功能或其他神秘的计算技术。其实，界面不应该要求用户在机器内部层次上进行交互（例如，不应该要求用户在应用软件中输入操作系统命令）。

设计应允许用户与出现在屏幕上的对象直接交互。当用户能够操纵完成某任务所必需的对象，并且以一种该对象好像是真实物理存在的方式来操纵它时，用户就会有一种控制感。例如，某应用界面可允许用户“拉伸”某对象（增大其尺寸）即是直接操纵的一种实现。

“我一直希望计算机能像电话一样易于使用。我的希望已经实现了，我不再想知道如何使用我的电话了。”

——Bjarne Stronstrup (C++的发明人)

358

12.1.2 减轻用户的记忆负担

用户必须记住的东西越多，和系统交互时出错的可能性也就越大。因此，一个经过精心设计的用户界面不会加重用户的记忆负担。只要可能，系统应该“记住”有关的信息，并通过能够帮助回忆的交互场景来帮助用户。Mandel[MAN97]定义了一组设计原则，使得界面能够减少用户的记忆负担：

减少对短期记忆的要求。当用户陷于复杂的任务时，短期记忆的要求将会很大。界面的设计应该尽量不要求记住过去动作和结果。可行的解决办法是通过提供可视的提示，使得用户能够识别过去动作，而不是必须记住它们。

建立有意义的缺省。初始的缺省集合应该对一般的用户有意义，但是，用户应该能够说明个人的偏好。然而，“reset”（重置）选项应该是可用的，使得可以重新定义初始缺省值。

定义直观的快捷方式。当使用助记符来完成系统功能时（如用Alt+P激活打印功能），助记符应该以容易记忆的方式被联系到相关动作（例如，使用被激活任务的第一个字母）。

界面的视觉布局应该基于真实世界的象征。例如，一个账单支付系统应该使用支票簿和支票登记簿来指导用户的账单支付过程。这使得用户能够依赖于能很好理解的可视提示，而不是记住复杂难懂的交互序列。

以不断进展的方式揭示信息。界面应该以层次化方式进行组织，即关于某任务、对象或某行为的信息应该首先在高抽象层次上呈现。更多的细节应该在用户用鼠标点击表明兴趣后再展示。例如，很多字处理应用中都十分常见的一个功能是加下划线，该功能本身是“文本风格”菜单下多个功能中的一个。然而，每种加下划线的能力并未列出，用户必须选择加下划线，然后所有加下划线选项（例如，加单下划线，加双下划线，加虚下划线）才被展示出来。

SAFEHOME

违反用户界面的“黄金规则”

[场景] Vinod的办公室，用户界面设计开始在即。

[人物] Vinod和Jamie，SafeHome软件工程团队成员。

[对话]

Jamie：我已经考虑过监控功能的界面了。

Vinod（微笑）：思考是好事。

Jamie：我认为我们可以将其简化。

Vinod：什么意思？

Jamie：如果我们完全忽略住宅平面图会怎么样？它倒是很华丽，但是要花费很多开发精力。我们只要询问用户他们要查看的指定摄像头，然后在视频窗口显示视频就可以了。

359

Vinod: 家庭成员如何记住有多少个摄像头并且它们都安装在什么地方呢?

Jamie: 家庭成员他应该知道。

Vinod: 但是如果他不知道呢?

Jamie: 他应该知道。

Vinod: 这不是问题的关键……如果他忘记了呢?

Jamie: 哦, 我应该提供一张可操作的摄像头及其位置的清单。

Vinod: 那也有可能, 但是他为什么要有一张清单呢?

Jamie: OK, 无论他是不是有这方面的要求我们都提供一张清单。

Vinod: 好的。至少他不必特意记住我们给他的东西了。

Jamie (想了一会儿): 但是你喜欢住宅平面图, 不是吗?

Vinod: 哈哈。

Jamie: 你认为市场营销人员会喜欢哪一个?

Vinod: 你在开玩笑, 是吗?

Jamie: 不。

Vinod: 哦……华丽的那个……他们喜欢迷人的……他们对简单的不感兴趣。

Jamie: (叹口气) OK, 也许我应该为两者都设计一个原型。

Vinod: 好主意……我们就让客户来决定。

12.1.3 保持界面一致

用户应该以一致的方式展示和获取信息, 这意味着: (1) 按照贯穿所有屏幕显示的设计标准来组织可视信息; (2) 将输入机制约束到有限的集合, 在整个应用中得到一致地使用; (3) 从任务到任务的导航机制要一致地定义和实现。Mandel[MAN97]定义了一组帮助保持界面一致性的设计原则。

“看起来不同的事物产生的效果应该不同, 看起来相同的事物产生的效果也应该相同。”

——Larry Marine

允许用户将当前任务放入有意义的环境中。很多界面使用数十个屏幕图像来实现复杂的交互层次。提供指示器(如, 窗口题目、图标、一致的颜色编码)帮助用户知道当前工作环境是十分重要的。另外, 用户应该能够确定他来自何处以及存在什么途径转换到新任务。

在应用系统家族内保持一致性。一组应用系统(或一套产品)都应实现相同的设计规则, 以保持所有交互的一致性。

如果过去的交互模型已经建立起了用户期望, 除非有不得已的理由, 否则不要改变它。一个特殊的交互序列一旦已经变成事实上的标准(如使用Alt+S来存储文件), 则用户在遇到的每个应用中均是如此期望。如果改变(如使用Alt+S来激活缩放比例)将导致混淆。

在这里和前面几节讨论的界面设计原则为软件工程师提供了基本指南。在下面几节中, 我们将考察界面设计过程。

INFO**可用性**

在一篇关于可用性方面的有深刻见解的论文中，Larry Constantine[CON95]提出了一个与可用性主题非常相关的问题：“用户究竟想要什么？”他给出如下回答：“用户真正想要的是好的工具。所有的软件系统，从操作系统和语言到数据录入和决策支撑应用，都是工具。最终用户希望从为其设计的工具中得到的与我们希望从所使用工具中得到的是一样的。他们想要易于学习并能够帮助他们工作的系统。同时，他们想要的系统应该能提高工作效率，不会欺骗他们或使其糊涂，不会使他们易于犯错误或难于完成工作。”

Constantine指出，系统的可用性并非取决于体系架构、交互技术的发展水平，或者内置的界面智能等方面；而是，当界面的架构适合于将要使用这些界面的用户需求时，才获得可用性。

正式的可用性定义往往令人有些迷惑。Donahue和他的同事[DON 99]给出了如下的定义：“可用性是一种衡量计算机系统好坏的度量……便于学习；帮助初学者记住他们已经学到的东西；降低犯错的可能；使得用户更加有效率；并且使得他们对系统感到满意。”

确定你所建系统是否可用的唯一办法就是进行可用性评估和测试。观察用户与系统的交互，同时回答下列问题[CON95]：

- 在没有连续的帮助或用法说明的情况下，系统是否便于使用？
- 交互规则是否能够帮助一个知识渊博的用户工作得更加有效率？
- 系统是否已经过调试，使之适应其运行的物理环境和社会环境？
- 用户是否意识到系统的状态？在工作期间内，用户是否能够知道她所处的位置？
- 界面是否是按照一种合理并且一致的方式来构建？
- 交互机制、图标和程序是否与界面相一致？
- 交互是否能够提前发现错误并帮助用户修正它们？
- 界面是否能够容错？
- 交互过程是否简单？

如果上述每个问题的回答都是Yes，那么我们可以认为这个系统是可用的。

可用性系统带来的诸多好处在于[DON99]：提高销售量和用户满意度、具有竞争优势、在媒体中获得良好的评论、获得良好的口碑、降低支持的费用、减少文档开销、减少来自不满意用户的投诉。

12.2 用户界面的分析与设计

WebRef

在www.useit.com
上可以找到关于UI
设计的丰富资源。

用户界面的分析和设计全过程始于创建不同的系统功能模型（从外部看对系统的感觉）。用以完成系统功能的任务被分为面向人的和面向计算机的；考虑那些应用到界面设计中的各种设计问题；各种工具被用于建造原型和最终实现设计模型；最后由最终用户从质量的角度对结果进行评估。

361

12.2.1 用户界面分析和设计模型

分析和设计用户界面时要考虑四种模型：工程师（或者软件工程师）建立用户模型；软

件工程师创建设计模型；最终用户在脑海里对界面产生的映像，称为用户的心理模型或系统感觉；系统的实现者创建实现模型。不幸的是，这四种模型可能会相差甚远，界面设计人员的任务就是消解这些差距，导出一致的界面表示。

“如果用户界面有一点瑕疵，那么整个用户界面就被破坏。” ——Douglas Anderson

用户模型确立了系统最终用户的轮廓（profile）。为了建立有效的用户界面，“开始设计之前，必须对预期用户加以了解，包括年龄、性别、身体状况、教育、文化和种族背景、动机、目标以及性格” [SHN90]。此外，用户可以分类为：



即使用户是新手，也会有使用快捷键的需求；即使是经常使用的用户，有时候也需要帮助。都要满足他们的需要。

- 新手。对系统没有任何语法知识的了解¹，并且对应用或计算机的一般用法几乎没有掌握什么语义知识²。
- 对系统有部分了解的中级用户。掌握适度的应用语义知识，但对使用界面所必需的语法信息的了解还比较少。
- 对系统有了解的经常用户。对应用有很好的语义知识和语法知识了解（这经常导致“强力用户综合征”），这些用户经常寻找捷径和简短的交互模式。

整个系统的设计模型包括对软件的数据、体系结构、界面和程序上的表示，需求规格说明可以建立一定的约束来帮助定义系统的用户，但是界面的设计往往是设计模型的附带结果³。

用户的心理模型（系统感觉）是最终用户在脑海里对系统产生的印象，例如，请某个特定页面风格布局系统的用户描述其操作，那么系统感觉将会引导用户的回答，准确的回答取决于用户的经历（新手只能做简要的回答）和用户对应用领域软件的熟悉程度。一个对页面布局有深刻了解，但只使用这种系统一次的用户可能比已经使用该系统好几个星期的新手回答得更详细。

“注意用户的行为而不是他们的言语。”

——Jakob Nielsen

实现模型组合了计算机系统的外在表现（界面的感觉），结合了所有用来描述系统语法和语义的支撑信息（书、手册、录像带、帮助文件等）。当系统实现模型和用户心理模型相一致的时候，用户通常就会对软件感到很舒服，使用起来就很有效。为了将这些模型融合起来，开发的设计模型必须包含用户模型中的一些信息，实现模型必须准确地反映界面的语法和语义信息。

这里描述的模型是“对用户在使用交互式系统时的所做所想，或其他人所做所想的抽象” [MON84]。从本质上讲，这些模型使得界面设计人员满足用户界面设计中最重要原则的关键元素就是：“了解用户，了解任务。”

12.2.2 用户界面分析和设计过程

用户界面的分析和设计过程是迭代的，可以用类似于第3章讨论过的螺旋模型表示。如

¹ 这里的“语法知识”是指有效地使用界面所需要的交互机制。

² “语义知识”是指应用程序的内在含义，即对执行的功能、输入输出的含义、系统目标的理解。

³ 这不应该是实际中要采用的方式。在很多情况下，用户界面设计与体系结构和构件级设计是一样重要的。



362
用户的心理模型影响着用户对界面的理解以及UI是否满足用户的需求。

362

图12-1所示，用户界面分析和设计过程包括4个不同的框架活动[MAN97]：

1. 用户、任务和环境分析及建模。
2. 界面设计。
3. 界面构造（实现）。
4. 界面确认。

图12-1中的螺旋意味着每一个活动都将多次出现，每绕螺旋一周表示需求和设计的进一步精化。在大多数情况下，实现活动涉及原型开发——这是唯一实用的确认设计结果的方式。

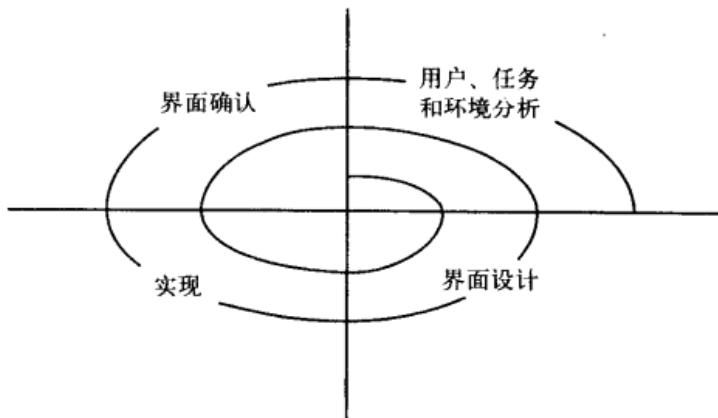


图12-1 用户界面设计过程

界面分析活动的重点在于那些与系统交互的用户的轮廓。记录技能级别、业务理解以及对新系统的一般感悟，并定义不同的用户类型。对每一个用户类别，进行需求诱导。本质上，软件工程师试图去理解每类用户的系统感觉（12.1.2节）。

“依照用户的习惯来设计，比矫正用户的习惯要好。”

——Jon Meads

一旦定义好了一般需求，将进行更详细的任务分析。标识、描述和细化（通过绕螺旋的多次迭代）那些用户为了达到系统目标而执行的任务。12.3节将对任务分析进行更详细的讨论。

用户环境的分析着重于物理工作环境。需要问的问题有：

我们开始进行用户界面设计时需要了解哪些环境呢？

- 界面的物理定位如何？
- 用户是否将坐着、站着或完成其他与界面无关的任务？
- 界面硬件是否适应空间、光线或噪音约束？
- 是否存在由环境因素驱动的特殊人性因素考虑？

作为分析活动的一部分而收集的信息被用于创建界面的分析模型。使用该模型作为基础，开始设计活动。

界面设计的目标是定义一组界面对象和动作（以及它们的屏幕表示），它们使得用户能够以满足系统所定义的每个使用目标的方式完成所有定义的任务。界面设计将在12.4节详细讨论。

正常情况下，构造活动始于创建可评估使用场景的原型。随着迭代设计过程的继续，用户界面开发工具（12.4节）可用来完成界面的构造。

确认着重于：(1) 界面正确地实现每个用户任务的能力，适应所有任务变更的能力以及达到所有一般用户需求的能力；(2) 界面容易使用和学习的程度；(3) 用户把界面作为其工作中

363

364

有用工具的接受程度。

如我们已经提到的，本节描述的活动是以迭代方式开展的。因此，不需要在第一轮就试图刻画所有细节（对分析或设计模型而言）。后续的过程将细化界面的任务细节、设计信息和运行特征。

12.3 界面分析⁴

所有软件工程过程模型的一个重要原则是：在试图设计一个解决方案之前最好对问题有更好的理解。在用户界面的设计中，理解问题就意味着了解：(1) 通过界面和系统交互的人（最终用户）；(2) 最终用户为完成工作要做的任务；(3) 作为界面的一部分而显示的内容；(4) 任务处理的环境。在接下来的几节中，为了给设计任务建立牢固基础，我们来检查界面分析的每一个因素。

12.3.1 用户分析

 我们怎样获
知用户希望
从用户界面上得
到什么呢？

365

 首先，花费精力与实际用户交谈，但要仔细认真。一个强烈的建议并不一定意味着大多数的用户都同意。

之前，我们提到每个用户对于软件都存在心理想像或者系统感觉，而这可能与其他用户开发的心理想像存在着差别。另外，用户的心理想像可能与软件工程师的设计模型相距甚远。设计师能够将得到的心理想像和设计模型聚合在一起的唯一办法就是努力了解用户，同时了解这些用户是如何使用系统的。为了完成这个任务，可以利用从各种途径获得的信息：

用户访谈：最直接的办法，访谈时要有软件团队的代表，他们与最终用户的会面可以更好地让他们了解用户的需求、动机、企业文化和其他的问题。可以是一对一的会议方式，也可以是群体讨论的形式。

零售输入：销售人员与客户和用户定期见面，能够收集到有助于软件团队对用户进行分类和更好理解用户需求的信息。

市场输入：在市场部分的定义中，市场分析是非常重要的，它提供了对市场每个部分使用软件的细微差别理解。

支持输入：技术支持人员与用户交谈，这使他们很容易获得“应该做什么，不应该做什么，用户喜欢什么，不喜欢什么，哪些特征会产生问题，哪些特征易于使用”等问题的信息。

下列一组问题（摘自[HAC98]）将有助于界面设计师更好地理解系统的用户：

- 用户是经过训练的专业人员、技术员、办事员，还是制造业工人？
- 用户平均正规教育水平如何？
- 用户是否具有学习书面资料的能力或者是否渴望接受学校教育？
- 用户是否是专业录入人员还是键盘恐惧者？
- 用户群体的年龄范围如何？
- 是否需要考虑用户的性别差异？
- 如何为用户完成的工作提供报酬？
- 用户是否在正常的办公时间内工作或者一直干到工作完成？

 我们怎么知道终端用户的特征和人数？

⁴ 这一节放到第8章也是合理的，因为那里讨论需求分析问题。在这里指出的是，界面的分析与设计是相互紧密关联的，而且两者之间的界限是很模糊的。

- 软件是用户所完成工作中的一个集成部分，还是偶尔使用一次？
- 用户群中使用的主要交流语言是什么？
- 如果用户在使用软件的过程中犯错，结果会怎么样？
- 用户是否是系统所解决问题领域的专家？
- 用户是否想了解界面背后的技术？

这些问题和类似问题的答案将帮助设计师了解最终用户是什么人，什么可能令他们感到愉悦，如何对用户进行分类，他们对系统的心理模型是什么样子，如何刻画用户界面使其满足用户的需求。

366

12.3.2 任务分析和建模



用户的目的是通过用户界面来完成一个或多个任务。为了实现这一点，用户界面必须提供用户达到目标的机制。

任务分析的目标就是给出下列问题的答案：

- 在指定环境下用户将完成什么工作？
- 当用户工作时将完成什么任务和子任务？
- 在工作中用户将处理什么特殊的问题域对象？
- 工作任务的顺序如何——工作流程？
- 任务的层次关系如何？

为了回答这些问题，软件工程师必须利用第7章和第8章讨论的分析技术，只不过在此种情况下，要将这些技术应用到用户界面。

用例。在前面几章中我们提到过用例描述了参与者（在用户界面设计中，参与者通常是某一个人）和系统的交互行方式。作为任务分析的一部分，设计用例用来显示最终用户如何完成指定的相关工作任务。在大多数情况下，用例采用第一人称以非正式形式（简单一段话）来书写。例如，假如一家小的软件公司想专门为公司室内设计师开发一个计算机辅助设计系统。为了更好地理解他们是如何工作的，实际的室内设计工程师应该描述特定的设计功能。在室内设计师被问到“如何确定家具在室内摆放位置”的时候，室内设计师写下了如下非正式的用例描述：

我从勾画房屋的建筑平面图、窗户与门的尺寸和位置开始设计。我非常关心射入房间的光线，关心窗外的风景（如果它很漂亮，就会吸引我的注意力），关心无障碍墙的长度，关心房间的活动流程。我接下来会查看客户和我选取的家具清单——桌子、椅子、沙发、壁橱，以及重要的饰物清单——灯、挂毯、油画、雕塑、盆景、小布块和我关于客户放置要求的注释。然后，我再用建筑平面图的模板来画出清单中所列的每一项。我标记每一项，由于经常需要移动这些项，所以使用了铅笔。我考虑了多种放置方案，并且确定我最喜欢的方案。接着，我会画出一个房屋的透视图（三维图画）给客户，让客户了解一个房屋看起来应该是什么样的。

这个用例给出了计算机辅助设计系统中一项重要工作任务的基本描述。从这个描述中，软件工程师能够提炼出任务、对象和整个的交互流程。另外，系统额外的一些能够使得室内设计师感到愉悦的特征也能被构思出来。例如，可以将房屋中每一扇窗户的风景都拍摄一个数码相片。当画房屋透视图时，通过每扇窗户就可以看到窗外的真实景象。

367

用户界面设计的用例

[场景] Vinod的办公室，用户设计正在进行。

[人物] Vinod和Jamie，SafeHome软件工程团队成员。

[对话]

Jamie: 我拦住我们的市场部联系人，让她写了一份监视界面的用例。

Vinod: 用什么角色来写？

Jamie: 当然是房子的主人，还会有谁？

Vinod: 还有系统管理员这个角色。即使是房子的主人担任这个角色，这也是一个不同的视角。“管理员”制定系统方案，准备材料，设计建筑平面图，安置摄像头……

Jamie: 我让销售人员扮演想看视频的家庭主人的角色。

Vinod: 好的，这只是监视功能界面的一个重要行为。但是，我们也应该调查一下系统管理员的行为。

Jamie（似乎受到刺激，有些不悦）：你是对的。

（Jamie离开去找销售人员。她几个小时以后回来了。）

Jamie: 我真走运，找到了市场部联系人，我们一起完成了系统管理员的用例。我们应该把“管理”定义为可以应用所有其他SafeHome功能的一个功能。这是我们提出的用例。

（Jamie给Vinod看用例信息。）

非正式用例：我想能够在任何时候设置和编辑系统的布置方案。当我建立起系统，我选择某一个管理功能。系统询问我是否要建立一个新的系统布置方案，或者询问我是否编辑已有的方案。如果我选择了一个新建方案，系统呈现一个绘画屏幕，在网格上可以画出建筑平面图来。为了绘画简便，应该提供墙壁、窗户和门的图标。我只是将图标伸展到合适的长度。系统将把长度显示为英尺或者米（我可以选择系统的度量系统）。我能够从传感器和摄像头库中进行选择，并且把它们放置在建筑平面图中。我标记每一个传感器和摄像头，或者系统自动进行标记。我可以通过合适的菜单来配置传感器和摄像头。如果选择编辑，就可以移动传感器和摄像头，添加新的传感器和摄像头，删除传感器和摄像头，编辑建筑平面图并编辑摄像头和传感器的配置。在每一种情形下，我希望系统能够进行一致性检查并且帮助我避免出错。

Vinod（看完脚本之后）：OK，绘画程序可能有一些有用的设计模式或可重用构件，可以来画图形用户界面。我打赌通过这些工具可以完成一些或大部分的管理员界面。

Jamie: 同意！我马上进行查看。

任务细化。在第9章中，我们讨论了逐步细化（也称为功能分解或者逐步求精），把它作为一种精细化处理任务的机制，而这些任务是软件完成某些期望功能所要求的。界面设计的任务分析采用了一种详细阐述的办法来辅助理解用户界面必须容纳的用户活动。

任务分析可以采用两种方法来实现。正如我们所知道的，经常用交互的计算机系统替代手工或者半自动化的行为。为了理解实现活动目标而必须完成的任务，工程师⁵必须明白人们当

⁵ 在很多情况下，这一节描述的任务由软件工程师来完成。理想的情况是，该工程师受过人类工程学和用户界面设计方面的训练。



任务细化是非常有用的，但也是很危险的。这是因为仅仅细化了任务，没有设想不存在其他的方法，而当实现用户界面的时候将要尝试另一种方法。

前所执行的任务（在使用手工方法的时候），并且将这些任务映射到一组类似的（不必完全一致）、在用户界面的环境中完成的任务集合上。另一种方法是，工程师研究已有的基于计算机的解决方案的规格说明，并且得到一个适应于用户模型、设计模型和系统感觉的用户任务集合。

不管任务分析的整体方法如何，工程师必须首先定义和划分任务。已经知道的一种方法就是逐步细化方法。例如，假设一个小型的软件公司打算专门为室内设计师开发一个计算机辅助设计系统。通过观察工作中的室内设计师，软件工程师了解到，室内设计由一系列的主要活动组成：家具布置（在前面用例设计中提到过）、建筑物和材料的选择、墙壁和窗户装饰物的选择、展示（对于用户而言）、计算成本、购物。其中任何一个都可以被细化成一系列的子任务。例如，使用用例中的信息，可以将家具布置任务细化为下面的子任务：(1) 根据房屋的尺寸画出建筑平面图；(2) 将门窗安置在合适的位置；(3a) 使用家具模型在平面图上描绘相应比例的家具轮廓；(3b) 使用饰件模板（accent template）在平面设计图上勾勒相应比例的饰件；(4) 移动家具和饰件轮廓线到达理想的位置；(5) 标记所有的家具和饰件轮廓；(6) 画上尺寸以显示其位置；(7) 为用户勾画透视图。其他的主要任务也可以应用类似的方法进行细化。

上面1~7的每一个子任务都可以进一步精化。其中1~6的子任务可以通过操纵界面中的信息和完成各种动作来完成。另一方面，子任务7可以在软件中自动完成，并且几乎不用直接与用户交互⁶。界面的设计模型应该以一种与用户模型（某一典型室内设计师的轮廓图）和系统感觉（一个室内设计师期望系统自动提供）相一致的方式来配合这些任务。



尽管对象细化非常有用，但是不能作为一个孤立的方法来使用。在任务分析中必须考虑来自用户的声音。

对象细化。软件工程师这时不是着眼于用户必须完成的任务，而是需要检查用例和来自用户的其他信息，并且提取室内设计师需要使用的物理对象。这些对象可以被分为不同的类。每个类的属性都要被定义下来，并且通过对每个对象上面动作的评估为设计师提供了一个操作列表。例如，家具模板可能被转换成一个称为Furniture的类，这个类包括size、shape和location等属性。室内设计师将会从Furniture类中选择对象，把它移到建筑平面图（在此处的上下文中，建筑平面图是另一个对象）中的某个位置上，拖曳家具的轮廓，依此类推。任务“选择”（select）、“移动”（move）、“拖曳”（draw）等都是操作。用户界面分析模型不能对任何一种操作都提供文字实现。然而，随着设计的不断细化，每个操作的细节都将被定义出来。

工作流分析。当大量扮演着不同角色的用户使用某一个用户界面时，有时候除了任务分析和对象细化之外，还有必要进行工作流分析。该技术使得软件工程师可以很好地理解在包含多个成员（角色）时，一个工作过程是如何完成的。假如某一个公司打算把处方药的开方和给药过程全部自动化。全部过程⁷将围绕着一个基于Web的应用进行循环，医生（或者他们的助手）、药剂师和病人等都可以访问这个应用系统。用UML泳道图（活动图的一种变形）能够有效地表示工作流。

369

⁶ 然而，事实可能不是这样。室内设计师可能想要给出所画透视图、缩放比例、色彩的运用和其他的信息。与透视渲染相关的用例将提供解决这些问题的信息。

⁷ 这个例子来自于文献[HAC98]。

下面只考虑工作过程中的一个小部分：当病人请求处方时发生的情形。图12-2给出了一个泳道图，该图表明了上面提及的三个角色的任务和决定。这些信息可以通过交谈或每个角色书写的用例获取。不管怎样，事件流（图12-2中显示的）使得界面设计师认识到三个关键的界面特征：

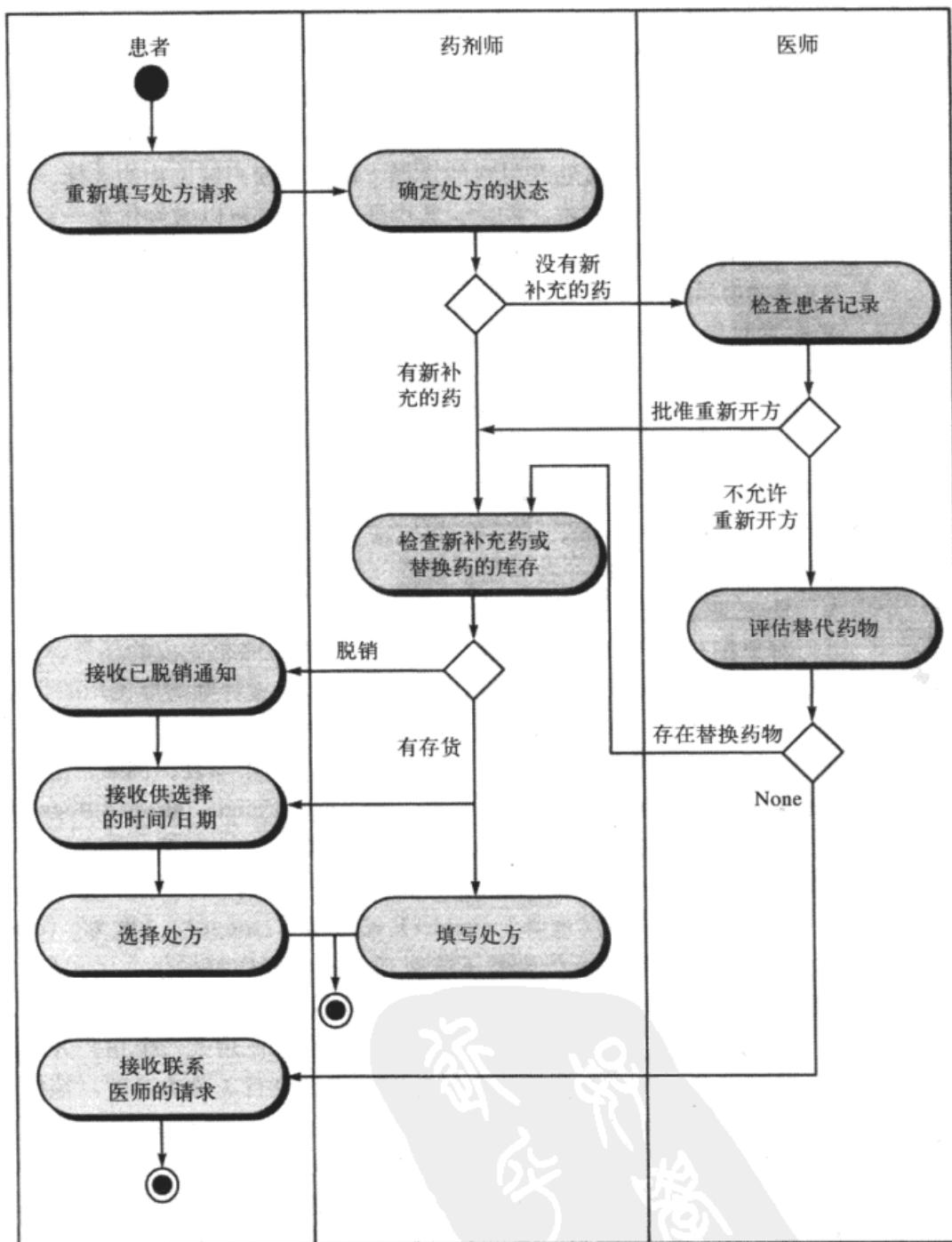


图12-2 处方再填写功能泳道图

1. 每个用户通过界面实现不同的任务；因此，为病人设计的界面在视觉和感觉上将与为药剂师和医生设计的界面有所不同。
2. 为医生和药剂师设计的界面应该能够访问和显示来自第二信息源的信息（例如，药剂

师应能够访问库存详细清单，而医生应能够访问其他可选择药物信息)。

3. 泳道图中的很多活动都可以通过采用任务分析和/或对象求精使其进一步细化（例如，“填写处方”隐含着邮购支付、访问药房，或者访问一个特殊药品分发中心）。

层次表示。在界面分析时，会产生相应的细化过程。一旦建立了工作流，为每个用户类型都能定义一个任务层次。该任务层次来自于为用户定义的每项任务的逐步细化。例如，考虑一个“请求重新填写处方”的用户任务，经过开发后得到如下的任务层：

重新填写处方请求

- 提供辨识信息
 - 指定姓名
 - 指定用户的ID
 - 指定PIN和密码
- 指定处方序号
- 指定重新填写处方所需的日期

370
371

为了完成填写处方的填写任务，定义了三个子任务。其中的第一个“提供辨识信息”子任务，可以进一步细化成三个子子任务。

“使技术适应用户要比用户适应技术好。”

——Larry Marine

12.3.3 显示内容分析

前面小节中标识出的任务导致需要对各种各样不同类型的内容进行描述。对于现代应用问题，界面显示内容包括文字报告（例如电子数据表）、图形化显示（柱状图、三维模型、个人图片），或者特殊形式的信息（语音和视频文件）。在第8章中讨论过的分析模型技术标识出由应用产生的输出数据对象。这些数据对象可能：(1) 由应用系统其他部分构件（与界面无关）生成；(2) 由应用系统所访问数据库中存储的数据获得；(3) 从系统外部传递到正在讨论的应用系统。

在界面分析步骤中，要考虑内容（当它要显示在界面上的时候）的格式和美感。其中要问和回答的问题包括：

对于图形化用户界面显示的内容，如何决定其格式和美感。

- 不同类型的数据是否要放置到屏幕上固定的位置（例如，照片一般显示在右上角）？
- 用户能否定制内容的屏幕位置？
- 是否对所有内容赋予适当的屏幕标识？
- 为了便于理解，应如何划分长篇报告？
- 对于大集合的数据，是否存在直接移动到摘要信息的机制？
- 输出图形的大小是否需要适合所使用显示设备的限制？
- 如何使用颜色来增强理解？
- 出错信息和警告应如何呈现给用户？

在回答这些（和其他的）问题时，就会建立起内容表示的需求。

372

12.3.4 工作环境分析

Hackos和Redish[HAC98]在讨论工作环境分析的重要性时这样写道：

“人们不能孤立地完成任务。他们会受到周围活动的影响，如工作场所的物理特征，使用设备的类型，与其他人的工作关系等。如果你设计的产品不适应环境，那么它一定是难于使用或者使用起来不方便。”

在一些应用中，计算机系统的用户界面被放在“用户友好”的位置（例如，适合的照明、良好的显示高度、简单方便的键盘操作），但有些地方（如，工厂的地板和飞机座舱）光照可能不是很适合，噪音也可能是个问题，也许不能选择使用键盘或鼠标，显示布置也不甚理想。界面设计师可能会受到有些因素的限制，这些因素会减弱易用性。

除了物理的环境因素之外，工作场所的文化氛围也起着作用。可否采用一些方式（如，每次交互所用时间、交互的准确性）来度量系统的交互？在提供一个输入前，两个或多个人员是否一定要共享信息？如何为系统用户提供支持？在界面设计师设计开始前，应该对上述问题和更多的相关问题给予回答。

12.4 界面设计步骤

一旦界面分析完成，最终用户要求的所有任务都已经被详细确定下来，界面设计活动就正式开始了。与所有软件工程设计一样，界面设计是一个迭代的过程。每个用户界面设计步骤都要进行很多次，每次细化和精化的信息都来源于前面的步骤。

尽管已经提出了很多不同的用户界面设计模型（如，[NOR86]和[NIE00]），但它们都建议结合以下步骤：

1. 使用界面分析中获得的信息（12.3节），定义界面对象和行为（操作）。
2. 定义那些导致用户界面状态发生变化的事件（用户动作）。对这个行为建模。
3. 描述每一个界面状态，就像最终用户实际看到的那样。
4. 简要说明用户如何从界面提供的界面信息来解释系统状态。

在某些情况下，界面设计师可以从每个界面状态草图开始（如，在各种环境下用户界面看起来是个什么样子），然后回头定义对象、动作和其他重要的设计信息。不管设计任务的顺序如何，设计师必须：（1）一直遵循12.1节讨论的黄金规则；（2）模拟界面是如何实现的；（3）考虑将要使用的环境（如，显示技术、操作系统、开发工具）。

“交互设计是绘画艺术、技术和心理学的无缝结合。”

——Brad Wieners

12.4.1 应用界面设计步骤

界面设计的一个重要步骤是定义界面对象和作用于之上的行为。为了完成这个目标，需要使用与第8章中介绍的方法相类似的方法来分析用例，也就是说，撰写用例的描述。名词（对象）和动词（行为）被分离出来形成对象和行为列表。

一旦对象和动作完成定义和迭代细化，就可以将它们按类型分类。目标、源和应用对象都被标识出来。把源对象（如报告图标）拖放到目标对象（如打印机图标）上，这意味着该动作要产生一个硬拷贝的报告。应用对象代表着应用特有的数据，它们并不作为屏幕交互的一部分被直接操纵。例如，一个邮件列表被用于存放邮件的名字。该列表本身可以进行排序、合并或清除（基于菜单的动作），但是，它不会通过用户的交互而拖动和删除。

当设计者满意地认为已经定义了所有的重要对象和动作（对一次设计迭代而言）时，开

始进行屏幕布局。和其他界面设计活动一样，屏幕布局是一个交互过程，其中包括：图标的图形设计和放置、屏幕描述性文字的定义、窗口的规格说明和标题，以及各类主要和次要菜单项的定义等。如果一个真实世界的隐喻适合于该应用，则在此时进行说明，并以补充隐喻的方式来组织布局。

为了对上面的设计步骤提供简明的例证，我们考虑SafeHome系统（在前面几章讨论过）的一个用户场景。下面是界面的初步用例描述：

初步用例：我希望通过Internet在任意的远程位置都能够访问SafeHome系统。使用运行在笔记本电脑上的浏览器软件（当正处于工作或者旅行状态时），我可以决定警报系统的状态，启动或关闭系统，重新配置安全区以及通过预先安置的摄像头观察住宅内的不同房间。

为了远程访问SafeHome，我提供了一个标识符和一个密码，这些定义了访问的级别（如并非所有用户都可以重新配置系统）并提供安全保证。一旦确认身份，我（具有全部访问权限）检查系统状态并通过启动或关闭SafeHome系统改变状态。通过显示住宅的建筑平面图，观察每个安全传感器，显示每个当前配置区域以及修改区域（必要时），可以重新配置系统。通过策略地放置摄像头观察房子内部。可以摆动和变焦每个摄像头而提供房子内部的不同视角。

基于这个用例，确定出以下住宅主人的任务、对象和数据项：

- 访问SafeHome系统。
- 输入ID和密码实现远程访问。
- 检查系统状态。
- 启动或关闭SafeHome系统。
- 显示建筑平面图和传感器位置。
- 显示建筑平面图上的区域。
- 改变建筑平面图上的区域。
- 显示建筑平面图上的视频摄像头位置。
- 选择用于观察的视频摄像头。
- 观察视频图像。
- 摆动或变焦摄像头。

374

对象（黑体）和动作（斜体）从这个住宅主人的任务清单中抽取出来。所提到的大部分对象是应用对象。然而，视频摄像头位置（源对象）被拖放到视频摄像头（目标对象）以创建视频图像（显示视频的窗口）。



尽管自动化的工具在开发布局原型的时候非常有用，但是有些时候铅笔和纸还是需要的。

为视频监控设计的屏幕布局初步草图如图12-3所示⁸。为了调用视频图像，需选择显示在监控窗口中的建筑平面图上的视频摄像头位置图标C。在这种情况下，起居室（LR）中的摄像头位置被拖放到屏幕左上部分的视频摄像头图标，此时，视频图像窗口出现，显示来自位于起居室中的摄像头的流视频。变焦和摆动控制条用于控制视频图像的放大和方向。为了选择来自另一个摄像头的图像，用户只需简单地将另一个不同的摄像头位置图标拖放到屏幕左

⁸ 这里与前面章节对这些特征的实现稍有不同。这可以被认为是第一个草案，它描述了一个可以参考的选择方案。

上区域的摄像头图标上即可。

所显示的布局草图需以菜单条上每个菜单项的扩展来补充，指明视频监控模式（状态）有哪些可用的动作。在界面设计过程中将创建用户场景中提到的每个房主任务的一组完整草图。

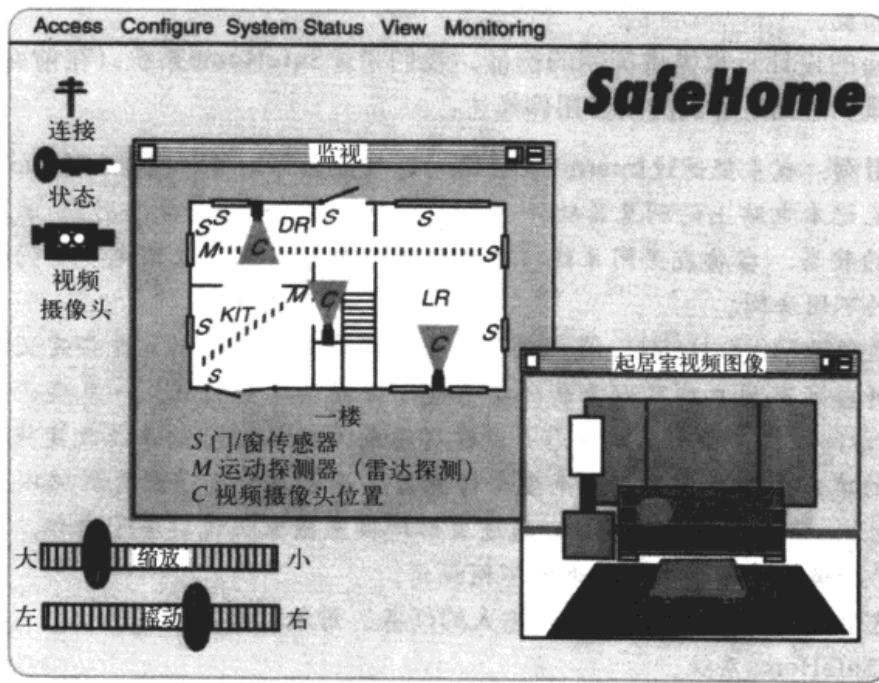


图12-3 基本的屏幕布局

12.4.2 用户界面设计模式

WebRef

375

已经提出了各式各样的UI设计模式。访问www.hic-patterns.org可以获得很多模式。

INFO

用户界面模式

在过去的十年间，人们已经提出了数以百计的用户界面设计模式。Tidwell [TID02]和van Welie[WEL01]对用户界面设计模式给出了分类⁹，一共分为10类。以下描述了每一类的模式示例。

1. 完整用户界面。为高层结构和导航提供设计指导。

模式：高层导航

简要描述：提供高层菜单，通常带有一个图标或者定义一个图像，能够直接跳转到任何一个系统主要功能。

2. 页面布局。负责页面概要组织（用于站点）或者清楚的屏幕显示（用于需要进行交互的应用系统）。

⁹ 在[TID02]和[WEL01]中可以找到完整的模式描述（还有几十个其他模式）。

模式：层叠

简要描述：呈现层叠状的标签卡，伴随着鼠标每一下点击的选择，显示指定的子功能或者分类内容。

3. 表格和输入。考虑了完成表格（form）级输入的各种设计方法。**模式：填充空格（fill-in-the-blanks）**

简要描述：允许在“文本框”中填写文字与数字数据。

4. 表（table）。为创建和操作各种列表数据提供设计指导。**模式：有序表（sortable table）**

简要描述：用来显示长记录列表，可以在任意一列上选择排序机制进行排序。

5. 直接数据操作。解决数据编辑、数据修改和数据转换问题。**模式：面包屑（bread crumbs）**

简要描述：当用户工作于复杂层次结构的页面或者屏幕显示时，提供完全的导航路径。

6. 导航。辅助用户在层级菜单、Web页面和交互显示屏幕上航行。**模式：现场编辑（edit-in-place）**

简要描述：为显示位置上的特定类型内容提供简单的文本编辑能力。

7. 搜索。对于网站上的信息或保存在可以通过交互应用访问的持久存储中的数据，能够进行特定内容的搜索。**模式：简单搜索**

简要描述：提供在网站或者持久数据源中搜索由字符串描述的简单数据项的能力。

8. 页面元素。实现Web页面或者显示屏的特定元素。**模式：向导（wizard）**

简要描述：通过一系列的简单窗口显示来指导完成任务，使得用户能够一步一步地完成某个复杂的任务。

9. 电子商务。主要针对于站点，这些模式实现了电子商务应用中的重现元素。**模式：购物车（shopping cart）**

简要描述：提供一个要购买的项目清单。

10. 其他。模式不能简单地归类到前面所述的任一类中，在某些情况下，这些模式具有领域的依赖性或者只对特定类别的用户适用。**模式：进展指示器（progress indicator）**

简要描述：为某一正在进行的操作提供进展指示。

对于用户界面设计的全面讨论超出了本书的范围，有兴趣的读者可以参看[DUY02]、[BOR01]、[WEL01]和[TID02]等获得进一步的信息。

12.4.3 设计问题

在进行用户界面设计时，几乎总会遇到以下四个问题：系统响应时间、用户帮助设施、错误信息处理和命令标记。不幸的是，许多设计人员往往很晚的时候才注意到这些问题（有时在操作原型已经建立起来后才发现有问题），这往往会导致不必要的反复、项目滞后和用户的挫折感，最好的办法是在设计的初期就将这些作为设计问题加以考虑，因为此时修改比较容易，代价也低。

377

“当试图设计一些十分简单的东西时，人们经常犯的共性错误就是低估了笨人的灵活性。”

——Douglas Adams

系统响应时间。系统响应时间不能令人满意是交互式系统用户经常抱怨的问题。一般来说，系统响应时间是指从用户开始执行动作（比如按回车键或点击鼠标）到软件以预期的输出和动作形式给出响应。

系统响应时间包括两方面的属性：时间长度和可变性。如果系统响应时间过长，用户就会感到焦虑和沮丧。系统时间的可变性是指相对于平均响应时间的偏差，在很多情况下这是最重要的响应时间特性。即使响应时间比较长，响应时间的低可变性也有助于用户建立稳定的交互节奏。例如，稳定在1秒的命令响应时间比从0.1秒到2.5秒不定的响应时间要好。在可变性到达一定值时，用户往往比较敏感，他们总是关心界面背后是否发生了异常。

帮助设施。几乎所有的计算机交互式系统的用户都时常需要帮助。有时一个简单的问题问一下同事就可以解决问题，但更复杂的问题则需要细致地研究用户手册。在多数情况下，现代的软件均提供联机帮助，用户可以不离开用户界面就解决问题。

考虑到帮助设施，需要在设计时解决如下问题[RUB88]：

- 在进行系统交互时，是否在任何时候对任何系统功能都能得到帮助？有两种选择：提供部分功能与动作的帮助和提供全部功能的帮助。
- 用户怎样请求帮助？有三种选择：帮助菜单、特殊功能键或HELP命令。
- 如何表达帮助？有三种选择：在另一个窗口中指示参考某个已印刷的文档（不是理想方式）或在屏幕特定位置给出一行或两行的简单提示。
- 用户如何回到正常的交互方式？可做的选择包括屏幕上显示返回按钮、功能键或控制序列。
- 如何构造帮助信息？有三种选择：平面式（所有信息均通过一个关键词来访问）、分层式（用户可以进一步查询得到更详细的信息）和超文本链接式。

378

错误处理。出错信息和警告是指出问题时系统反馈给用户的“坏消息”。如果做不好的话，出错信息和警告会给出无用和误导的信息，反而增加了用户的沮丧感。很少有用户没有遇到下面这种形式的错误信息：“Application XXX has been forced to quit because an error of type 1023 has been encountered”。应该在某些地方对1023错误给出解释；否则，为什么设计者要指出这个错误信息呢？但是，错误信息没有指出什么出错了或者在何处可以找到进一步的信息。这类错误信息既不能减轻用户的焦虑也不能解决任何问题。

“来自地狱的界面——‘修正这个错误并且继续进行，请输入任一个11位的素数……’”

——作者不详

“好的”出
错消息应该
具有什么样的特
征呢？

通常，交互式系统给出的出错消息和警告应具备以下特征：

- 消息以用户可以理解的语言描述问题。
- 消息应提供如何从错误中恢复的建设性意见。
- 消息应指出错误可能导致哪些不良后果（比如破坏数据文件），以便用户检查是否出现了这些情况或帮助用户进行改正。

- 消息应伴随着视觉或听觉上的提示。也就是说，显示消息时应该伴随警告声或者消息用闪烁方式显示，或以明显表示错误的颜色来显示。
- 消息不应是裁判性的，即不能指责用户。

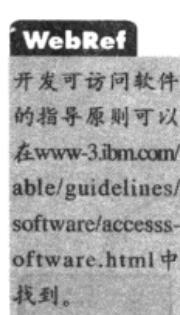
由于没有人喜欢坏消息，无论出错消息代表了什么都很少有用户喜欢出错消息，但从道理上讲，在出现问题时有效的出错消息能够提高交互式系统的质量、减少用户的沮丧感。

菜单和命令标记。键入命令曾经是用户和系统交互的主要方式，并广泛用于各种应用程序中。现在，面向窗口的界面采用点击（point）和拾取（pick）方式减少了用户对键入命令的依赖，但许多高级用户仍然喜欢面向命令的交互方式。在提供命令交互方式时，必须考虑以下问题：

- 每一个菜单选项是否都有对应命令？
- 以何种方式提供命令？有三种选择：控制序列（比如Alt+P）、功能键或键入命令。
- 学习和记忆命令的难度有多大？命令忘了怎么办？
- 用户是否可以定制和缩写命令？
- 在界面环境中菜单标签是否是自解释性的？
- 子菜单是否与主菜单项所指功能相一致？

379

本章前面我们提到，应该建立跨所有应用系统的命令使用约定。如果在一个应用系统中Alt+D表示复制一个图形对象，而在另一个应用系统中Alt+D表示删除一个图像，这就会使用户感到困惑并往往会导致错误，犯错误的可能性是显而易见的。



应用的可访问性。随着计算型应用系统变得无处不在，软件工程师必须确保界面设计中包含使得有特殊要求的用户易于访问的机制。对于那些实际上面临挑战的用户（和软件工程师）来说，由于道义、法律和业务等方面的原因，可访问性是必不可少的。多种可访问性指南（如[W3C03]）——很多都是为Web应用设计的，但这些方针经常也能应用于所有软件——为设计界面提供了详细的建议，以使界面能够达到各种级别的可访问性。其他的指南（如[APP03], [MIC03]）对于“辅助技术”提供了专门的指导，这些技术用来解决那些在视觉、听觉、活动性、语音和学习等方面有障碍的人员的需要。

国际化。软件工程师和他们的经理往往会低估建立一个适应不同国家和不同语言需要的用户界面所需的努力和技能。用户界面经常是为一个国家和一种语言所设计的，在面对其他国家时只好应急对付。设计师面临的挑战就是设计出“全球化”的软件。也就是说，用户界面应该被设计成能够容纳需要交付给所有软件用户的核心功能。本地化特征使得界面能够针对特定的市场进行定制。

软件工程师有多种国际化指导方针（如[IBM03]）可以使用。这些方针一方面解决了很多的设计问题（如，在不同的市场情况下屏幕布局可能是不同的），以及离散实现问题（如，不同的字母表可能生成特定的标识和间距需求）。如何管理几十种具有成百上千字母和字符的自然语言，已经提出的Unicode标准[UNI03]就是用来解决这个挑战性问题的。

SOFTWARE TOOLS**用户界面开发**

目的：用户界面开发工具可以使得软件工程师只需做有限的定制开发就可以建立一个复杂的图形用户界面。这些工具提供了对可复用构件的访问，并且通过选择工具上预定义的功能就可以建立一个用户界面。

380

机制：现代的用户界面由一组可复用的构件组成，这些构件与一些提供特殊特性的定制构件相结合。大多数用户界面的开发工具能够通过使用“拖放”功能来完成界面的设计。换句话说，开发人员通过选择预定义的功能（如，表格构造器，交互机制，命令处理），并将这些功能放置在所创建界面的环境中。

代表性工具¹⁰

Macromedia Authorware，由Macromedia Inc.公司（www.macromedia.com/software/）开发，主要是用来建立e-learning界面和环境。它使用了复杂的构建功能。

Motif Common Desktop Environment，由Open Group（www.osf.org/tech/desktop/cde/）开发，是用于开放系统桌面计算的集成的图形用户界面。它对数据、文件和应用系统的管理提供了单一的、标准的图形化界面。

PowerDesigner/PowerBuilder，由Sybase（www.sybase.com/products/internetappdevtools）开发，是一个全面的CASE（计算机辅助软件工程）工具集合，其中包含很多功能，可用来设计和建立GUI（图形用户界面）。

12.5 设计评估

一旦建立好可操作的用户界面原型，必须对其进行评估，以确定是否满足用户的需求。评估可以从非正式的“测试驱动”（比如用户可以临时提供一些反馈）到正式的设计研究（比如按照统计学的方法向一定数量的最终用户发放评估问题表）。

381

用户界面评估的周期如图12-4所示。完成初步设计后就开始建立第一级原型；用户对该原型进行评估¹¹，用户直接向设计者提供有关界面功效的建议，如果采用正式的评估技术（比如使用提问单、分级评分表），设计者需要从调查结果中得到需要的信息（比如80%的用户不喜欢其中保存数据文件的机制）；针对用户的意見对设计进行修改，完成下一级原型。评估过程不断进行下去，直到不需要再修改为止。

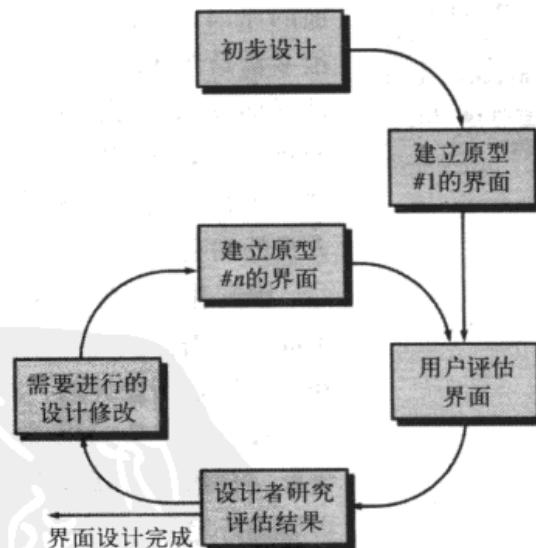


图12-4 界面设计评估周期

¹⁰ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

¹¹ 这里需要注意的是人类环境学和界面设计专家也可能参与界面的评审。这些评审被称为启发式评估或认识走查（cognitive walkthrough）。

原型开发方法是有效的，但是否可以在建立原型以前就对用户界面的质量进行评估呢？如果能够及早地发现和改正潜在的问题，就可以减少评估周期执行的次数，从而缩短开发时间。界面设计模型完成以后，就可以运用下面的一系列评估标准[MOR81]对设计进行早期评审：

1. 系统及其界面的书面规格说明的长度和复杂性在一定程度上表示了用户学习系统的难度。
2. 命令（或动作）的个数以及命令的平均参数个数（或动作中各项操作的数量）在一定程度上表示了系统交互的时间和系统总体的效率。
3. 设计模型中动作、任务和系统状态的数量反应了用户学习系统时所要记忆的内容的多少。
4. 界面风格、帮助设施和错误处理协议在一定程度上表示了界面的复杂度和用户的接受程度。

一旦第一个原型完成以后，设计者就可以收集到一些定性和定量的数据帮助进行界面评估。为了收集定性的数据，可以把提问单分发给原型用户，每项提问的答案可以是（1）简单的“是”或“否”；（2）数字；（3）程度（主观的）；（4）喜欢程度（例如，强烈同意，勉强同意）；（5）百分比（主观的）；或（6）对答案不加限制。

如果需要得到定量数据，就必须进行某种形式的定时研究分析。观察用户与界面的交互，记录以下数据：在标准时间间隔内正确完成任务的数量，使用动作的频度，动作顺序，观看屏幕的时间，出错的数目，错误的类型和错误恢复时间，使用帮助的时间，标准时间段内查看帮助的次数。这些数据可以用于指导界面修改。[382]

有关用户界面评估方法的详细论述已超出了本书的范围，有兴趣的读者可以参考[LEA88]、[MAN97]和[HAC98]等文献。

12.6 小结

用户界面可以说是计算机系统或产品的最重要元素。如果界面设计得很糟糕，可能会严重地阻碍用户使用系统的计算能力。事实上，即使应用系统具有良好设计和可靠实现，差的界面也可能导致该系统失败。

三个重要的原则可用于指导有效的用户界面设计：（1）置用户于控制之下；（2）减少用户的记忆负担；（3）保持界面一致性。为了得到符合这些原则要求的界面，必须实施有组织的设计过程。

用户界面的开发开始于一系列的分析任务，包括用户确认，任务和环境分析/建模。用户分析确定了各类最终用户的概况，并且使用了从各种业务和技术资源收集来的信息。任务分析定义了用户任务和行为，其中使用了细化或面向对象的方法、用户用例的应用、任务和对象的细化、工作流分析和层级任务表示等，来获得对人机交互的充分理解。环境分析标识出了界面必须操作的物质和社会结构。

一旦任务被标识，就可以通过创建和分析用户场景来定义一组界面对象和动作。这为创建屏幕布局提供了基础，屏幕布局描述了图标的图形设计和放置、描述性屏幕文字的定义、窗口的规格说明和标题以及主、子菜单项规格说明。诸如响应时间、命令和动作结构、错误处理和帮助设施等设计问题应该在精化设计模型时考虑。很多实现工具可以用于建造供用户评估的原型。

参考文献

- [APP03] Apple Computer, *People with Special Needs*, 2003, available at <http://www.apple.com/disability/>.
- [BOR01] Borchers, J., *A Pattern Approach to Interaction Design*, Wiley, 2001.
- [CON95] Constantine, L., "What DO Users Want? Engineering Usability in Software," *Windows Tech Journal*, December, 1995, available from <http://www.forUse.com>.
- [DON99] Donahue, G., S. Weinschenk, and J. Nowicki, "Usability Is Good Business," Compuware Corp., July, 1999, available from <http://www.compuware.com>.
- [DUY02] vanDuyne, D., J. Landay, and J. Hong, *The Design of Sites*, Addison-Wesley, 2002.
- [HAC98] Hackos, J., and J. Redish, *User and Task Analysis for Interface Design*, Wiley, 1998.
- 383 [IBM03] IBM, *Overview of Software Globalization*, 2003, available at <http://oss.software.ibm.com/icu/userguide/i18n.html>.
- [LEA88] Lea, M., "Evaluating User Interface Designs," *User Interface Design for Computer Systems*, Halstead Press (Wiley), 1988.
- [MAN97] Mandel, T., *The Elements of User Interface Design*, Wiley, 1997.
- [MIC03] Microsoft Accessibility Technology for Everyone, 2003, available at <http://www.microsoft.com/enable/>.
- [MON84] Monk, A., (ed.), *Fundamentals of Human-Computer Interaction*, Academic Press, 1984.
- [MOR81] Moran, T. P., "The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems," *Intl. Journal of Man-Machine Studies*, vol. 15, pp. 3-50.
- [NIE00] Nielsen, J., *Designing Web Usability*, New Riders Publishing, 2000.
- [NOR86] Norman, D. A., "Cognitive Engineering," in *User Centered Systems Design*, Lawrence Erlbaum Associates, 1986.
- [RUB88] Rubin, T., *User Interface Design for Computer Systems*, Halstead Press (Wiley), 1988.
- [SHN90] Shneiderman, B., *Designing the User Interface*, 3rd ed., Addison-Wesley, 1990.
- [TID99] Tidwell, J., "Common Ground: A Pattern Language for HCI Design," available at http://www.mit.edu/~jtidwell/interaction_patterns.html, May 1999.
- [TID02] Tidwell, J., "IU Patterns and Techniques," available at <http://time-tripper.com/uipatterns/index.html>, May 2002.
- [UNI03] Unicode, Inc., *The Unicode Home Page*, 2003, available at <http://www.unicode.org/>.
- [W3C03] World Wide Web Consortium, *Web Content Accessibility Guidelines*, 2003, available at <http://www.w3.org/TR/2003/WD-WCAG20-20030624/>.
- [WEL01] vanWelie, M., "Interaction Design Patterns," available at <http://www.welie.com/patterns/>, 2001.

习题与思考题

- 12.1 举例说明为什么响应时间可变性会成为一个问题?
- 12.2 试给出两个附加的“降低用户记忆负担”的设计原则。
- 12.3 在12.3.3节提供的内容分析列表中再添加至少5个问题。
- 12.4 试给出两个附加的“保持界面一致性”的设计原则。
- 12.5 假设你被邀请开发一个基于Web的家庭银行系统。请给出用户模型、设计模型、心理模型和实现模型。
- 12.6 针对习题12.5中的系统，开发一组带有主菜单和子菜单项定义的屏幕布局。
- 12.7 试给出两个附加的“置用户于控制之下”的设计原则。
- 12.8 针对习题12.5中的系统，使用细化方法或面向对象方法完成详细任务分析。
- 12.9 针对SafeHome系统，开发一组带有主菜单和子菜单项的屏幕布局，可以选择一种不同于图12-3所示的方法。
- 12.10 接着习题12.8，定义应用的接口对象和行为，确定每个对象的类型。

- 12.11 针对习题12.5中所完成的任务分析，描述你自己的用户帮助设施方法。
- 12.12 说说你操作过的最好和最不好的系统界面，采用本章中介绍的相关概念对其进行评价。 [384]
- 12.13 开发一种能自动集成错误消息和用户帮助设施的方法。即系统能自动识别错误类型，并提供帮助窗口，给出改正错误的建议。进行合理完整的软件设计，其中要考虑到合适的数据结构和算法。
- 12.14 开发一个界面评估提问单，其中包括20个适用于大多数界面的通用问题。由10名同学完成你们所有人使用的交互系统的提问单。汇总你们的结果并向班上汇报。

推荐读物与阅读信息

尽管Donald Norman的著作（《The Design of Everyday Things》，reissue edition, Currency/Doubleday, 1990）不是专门阐述人机界面的，但是其中涉及了可以应用于用户界面有效设计的哲学。对于那些非常关心高质量用户界面设计的人员，我们推荐此读物。

图形用户界面在现代计算世界中是无处不在的，无论是在ATM、移动电话、PDA、Web站点，或者是在一个商业应用中，用户界面都为软件提供了一个窗口。正因如此，关于界面设计的书籍有很多，如Galitz（《The Essential Guide to User Interface Design》，Wiley, 2002）、Cooper（《About Face 2.0: The Essentials of User Interface Design》，IDG Books, 2003）、Beyer和Holtzblatt（《Contextual Design: A Customer Centered Approach to Systems Design》，Morgan-Kaufmann, 2002）、Raskin（《The Humane Interface》，Addison-Wesley, 2000）、Constantine和Lockwood（《Software for Use》，ACM Press, 1999）、Maythew（《The Usability Engineering Lifecycle》，Morgan-Kaufmann, 1999）等都讨论了可用性、用户界面设计的概念、原则和设计技巧，并且包含了很多有用的例子。

Johnson（《GUI Bloopers: Don'ts and Do's for Software Developer and Web Designers》，Morgan Kaufmann, 2000）对那些通过看反例来实现高效学习的人来说具有很好的指导意义。Cooper（《The Inmates Are Running the Asylum》，Sams Publishing, 1999）讨论了为什么高技术产品常令我们精神失常，以及如何设计一个令人愉悦的产品。

任务分析和建模是界面设计活动的关键。Hackos和Redish[HAC98]撰写的书中涉及这方面的主题，并且提供了一个任务分析的详细方法。Wood（《User Interface Design: Bridging the Gap from User Requirements to Design》，CRC Press, 1997）考虑了界面的分析活动以及设计任务的转换。

评估活动主要关注可用性方面。Rubin（《Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests》，Wiley, 1994）和Nielsen（《Usability Inspection Methods》，Wiley, 1994）的书都是关于这个主题的，并且给出了相当多的细节描述。

Murphy（《Front Panel: Designing Software for Embedded User Interfaces》，R&D Books, 1998）的书对设计人员很有意义，书中提供了在嵌入式系统中进行界面设计的指导方针，并且关注于控制、大型机械处理以及医疗和运输系统的界面中有关固有安全危险的处理。Garrett（《Advanced Instrumentation and Computer I/O Design: Real-Time System Computer Interface Engineering》，IEEE, 1994）也讨论了嵌入式系统的界面设计问题。

关于用户界面设计的一个更加广泛的信息资源就是因特网，在SEPA的站点<http://www.mhhe.com/pressman>上可以找到最新的关于用户界面设计的文献列表。 [385]

第13章 软件测试策略

要点浏览

概念：软件测试的目的是为了发现软件设计和实现过程中的疏忽所造成的错误。但是，如何进行测试？是否应该制定正式的测试计划？是应该将整个程序作为一个整体来测试，还是应该只测试其中的一部分？当向一个大型系统加入新的构件时，是否需要重新测试已经测过的部分？什么时候需要客户介入测试工作？当制定测试策略时，就需要回答上述及其他一些问题。

人员：软件测试策略由项目经理、软件工程师以及测试专家来制定。

重要性：测试所花费的工作量经常比其他任何软件工程活动都多。若测试是无计划地进行，既浪费时间，又浪费不必要的劳动。甚至更糟的是，错误会依然存在。因此，为测试软件建立系统化的

测试策略是合情合理的。

步骤：测试从“小规模”开始，进展到“大规模”。这意味着，早期的测试关注单个构件或相关的一小组构件，利用测试发现构件中的数据和处理逻辑错误。当单个的构件被测试完后，需要将构件集成直到建成整个系统。这时，执行一系列的高阶测试（high-order test）以发现在满足顾客需求方面的错误。随着错误的发现，必须利用调试过程进行诊断和纠正。

工作产品：测试规格说明是将软件测试团队的测试具体作法文档化。这主要包括制定描述整体策略的计划、定义特定测试步骤的规程以及规定将要进行的测试。

质量保证措施：通过在测试进行之前评审测试规格说明，可以评估测试用例以及测试任务的完整性。有效的测试计划和规程将导致软件的有规则地构造，并且能够发现构造过程中各个阶段引入的

关键概念
α/β测试
调试
完整性标准
常规方法
集成测试
ITG
面向对象方法
回归测试
冒烟测试
系统测试
测试规格说明
单元测试
V&V
确认测试

软件测试策略将软件测试用例的设计方法集成到一系列经周密计划的步骤中去，从而使软件构造成功地完成。测试策略提供以下方面的路径图：描述将要进行的测试步骤，这些步骤计划和执行的时机，需要多少工作量、时间和资源。因此，任何测试策略都必须包含测试计划、测试用例设计、测试执行以及测试结果数据的收集与评估。

软件测试策略应该具有足够的灵活性，以便促进测试方法的定制；同时，它必须足够严格，以便在项目进行过程中对项目进行合理地策划和追踪管理。Shooman[SHO83]对这些问题进行了讨论：

在许多测试方法中，测试是一个独立的过程，并且同软件开发方法一样，测试类型有多种多样。多年以来，对付程序出错的唯一武器就是谨慎的设计和程序员的个人智慧。目前，有很多现代设计技术（和正式技术评审）帮助我们减少代码中存在的初始错误。类似地，不同的测试方法正在开始聚合成几种不同的途径和思想。

这些途径和思想就是我们所称的“策略”。本书的第14章将介绍软件测试技术，本章主要讨论软件测试策略。

13.1 软件测试的策略性方法

WebRef

有关软件测试的有用资料可以在下列站点找到：
[www.mtsu.edu/~
storm/](http://www.mtsu.edu/~storm/)

测试是可以事先计划并可以系统地进行的一系列活动。因此，应该为软件过程定义软件测试模板，即将特定的测试用例设计技术和测试方法放在一系列的测试步骤中去。

文献中已经提出了许多软件测试策略。这些策略为软件开发人员提供了测试模板，且具备下述的一般特征：

- 为完成有效的测试，软件团队应该进行有效的、正式的技术评审。通过评审，许多错误可以在测试开始之前排除。
- 测试开始于构件层，然后向外“延伸”到整个基于计算机系统的集成。
- 不同的测试技术适用于不同的时间点。
- 测试由软件开发人员和（对大型项目而言）独立的测试组执行。
- 测试和调试是不同的活动，但任何测试策略中都必须包括调试。

软件测试策略必须提供用来验证小段源代码是否正确实现的必要的低级测试，以及用来确认系统的主要功能是否满足用户需求的高级测试。软件测试策略必须为专业人员提供工作指南，同时，为管理者提供一系列的里程碑。由于测试策略的步骤是在软件完成的最后期限的压力已逐步呈现的时候才开始进行的，因此，测试的进度必须是可测量的，应该让问题尽可能早地暴露。

13.1.1 验证与确认

软件测试是通常所讲的更为广泛的主题——验证与确认 (Verification and Validation, V&V) 的一部分。验证是指确保软件正确地实现某一特定功能的一系列活动。确认则指的是确保开发的软件可追溯到用户需求的另外一系列活动¹。Boehm[BOE81]用另一种方式说明了这两者的区别：

验证：我们在正确地构造产品吗？

确认：我们在构造正确的产品吗？

验证与确认的定义还包含了一些软件质量保证 (Software Quality Assurance, SQA) 中的活动，详细内容参见第26章。

验证与确认包含了广泛的SQA活动，其中包括正式技术评审、质量和配置审核、性能监控、仿真、可行性研究、文档评审、数据库评审、算法分析、开发测试、易用性测试、合格性测试以及安装测试[WAL89]。虽然测试在验证与确认中发挥重要作用，但其他的一些活动也是必不可少的。

“测试是开发软件系统中每项可靠的工作都不可避免的部分。” ——William Howden

¹ 应该注意到，哪些类型的测试构成“确认”，对此观点存在极大的分歧。一些人认为所有的测试都是验证，而确认是当需求被评审并得到认可之后，或者更晚当系统可运行时由用户进行的。另外一些人将单元测试和集成测试（13.3.1节和13.3.2节）看成验证，而将高阶测试（将在本章后面讨论）看作确认。



不要轻易地将测试看成是一个安全网，认为它能捕捉由不良的软件工程实践所引起的所有错误。它不是。注重质量和错误检测贯穿整个软件过程。

测试确实为软件质量的评估（更实际地说是错误的发现）提供最后的防线。但是，测试不应当被看作安全网。正如人们所说的那样：“你不能测试质量。如果开始测试之前质量不佳，那么当你完成测试时质量仍然不会改进。”在软件工程的整个过程中，质量体现在软件之中。在测试过程中，方法和工具的正确运用、有效的正式技术评审、稳固的管理与测量，都有助于得到让人认可的质量。

Miller[MIL77]将软件测试和质量保证联系在一起，他认为：“无论是大规模系统还是小规模系统，程序测试的根本动机都是使用经济且能有效应用的方法来认可软件质量。”

13.1.2 软件测试的组织

388 对每个软件项目而言，在测试开始时总会在不同人员之间存在认识上的差异。这时要求开发软件的人员对该软件进行测试。这似乎并无恶意！谁能比开发者本人更了解程序呢？遗憾的是，这些开发人员特别感兴趣的是急于显示他们所开发的程序是无错误的，是按照用户的需求开发的，而且能按照预定的进度和预算完成。在充分进行测试的情况下，这些认识上的差异逐渐消除。

“乐观主义是编程的职业障碍；测试是治疗良方。”

——Kent Beck

从心理学的观点来看，软件分析和设计（连同编码）是建设性的任务。软件工程师分析、建模，然后编写计算机程序及其文档。与其他任何建设者一样，软件工程师也为自己的“大厦”感到骄傲，而蔑视企图拆掉大厦的任何人。当测试开始时，有一种微妙的但确实存在的要摧毁软件工程师已建立的东西的企图。以开发者的观点来看，测试可以认为（心理学上）是破坏性的。因此，开发者精心地设计和执行测试，试图证明其程序的正确性，而不是注意发现错误。遗憾的是，错误是存在的，而且，如果软件工程师没有找到错误，用户也会发现。

上述的讨论会引起人们产生如下误解：(1) 软件开发人员根本不应该做测试；(2) 应当让那些无情地爱挑毛病的陌生人做软件测试；(3) 测试人员仅在测试步骤即将开始时参与项目。这些想法都是不正确的。

KEY POINT
独立的测试组没有软件建设者所经历的“认识差异”。

软件开发人员总是要负责程序的个别单元（构件）的测试，确保每个单元完成其功能或显示出被设计的行为。在多数情况下，开发者也进行集成测试。集成测试是其中一个测试步骤，它将给出整个软件体系结构的构造（和测试）。只有在软件体系结构完成后，独立的测试组才开始介入。

独立测试组（Independent Test Group, ITG）的作用是为了避免开发人员进行测试所引发的固有问题。独立测试可以消除可能存在的认识差异。独立测试组的成员毕竟是依靠找错误来获得报酬的。

然而，软件开发人员并不是将程序交给独立测试组就可以一走了之。在整个软件项目中，开发人员和测试组密切配合以确保测试严格地进行。在测试进行的过程中，必须随时可以找到开发人员，以便及时修改发现的错误。

“人们犯的第一个错误是认为测试团队对保证质量负责。”

——Brian Marick



若组织中不存在ITG，当你进行测试时，必须持有试图破坏软件的观点。

从分析与设计开始到计划和制定测试规程，ITG参与整个项目过程。从这种意义上讲，ITG是大型软件开发项目组的一部分。然而，在多数情况下，ITG直接对软件质量保证组织负责，由此获得一定程度的独立性。若ITG是软件工程组织的一部分，则这种独立性是不可能获得的。

13.1.3 传统软件体系结构的测试策略

软件过程可以看作图13-1所示的螺旋。开始时系统工程定义软件的角色，从而引出软件需求分析，在需求分析中建立软件的信息域、功能、行为、性能、约束和确认标准。沿着螺旋向内，经过设计阶段，最后到达编码阶段。为开发计算机软件，沿着流线螺旋前进，每走一圈都会降低软件的抽象层次。

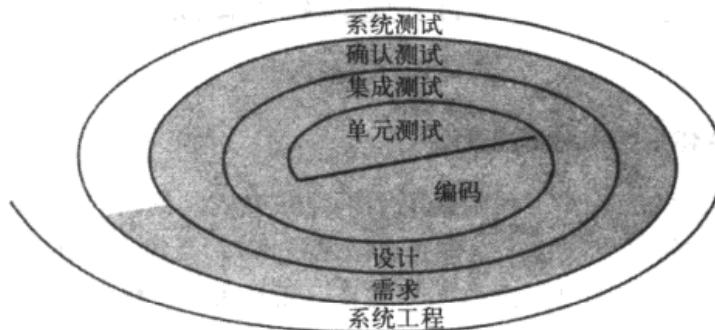


图13-1 测试策略

什么是软件
测试的总体
策略？

软件测试策略也可以放在螺旋模型中来考虑（图13-1）。单元测试起始于螺旋的旋涡中心，侧重于以源代码形式实现的每个单元（即构件）。沿着螺旋向外，就是集成测试，这时的测试重点在于软件体系结构的设计和构造。沿着螺旋向外再走一圈，就是确认测试，在这个阶段，依据已经建立的软件，对需求（作为软件需求分析的一部分而建立）进行确认。最后到达系统测试阶段，将软件与系统的其他成分作为一个整体来测试。为了测试计算机软件，沿着流线向外螺旋前进，每转一圈都拓宽了测试范围。

WebRef
有关软件测试人
员的有用资源可
在站点www.SQA-
tester.com中找到。

以过程的观点考虑整个测试过程，软件工程环境中的测试实际上就是按顺序实现四个步骤，如图13-2所示。最初，测试侧重于单个构件，确保它完全作为一个单元来起作用，因此称之为单元测试。单元测试充分利用测试技术，检查构件中每个控制结构的特定路径以确保完全覆盖，并最大可能地发现错误。接下来，组装或集成各个构件以形成完整的软件包。集成测试处理并验证与程序构造相关的问题。在集成过程中，普遍使用关注输入和输出的测试用例设计技术（尽管也使用检验特定程序路径的测试用例设计技术来保证主要控制路径的覆盖）。在软件集成（构造）完成之后，要执行一系列的高阶测试，必须评估确认准则（需求分析阶段建立的）。确认测试为软件满足所有的功能、行为和性能需求提供最后的保证。

390

最后高阶测试这一步已经超出软件工程的边界，进入更为广泛的计算机系统工程的范围。软件一旦确认，就必须与其他系统成分（如硬件、人、数据库）结合在一起。系统测试验证所有的成分能合适地结合在一起，且能满足整个系统的功能/性能需求。

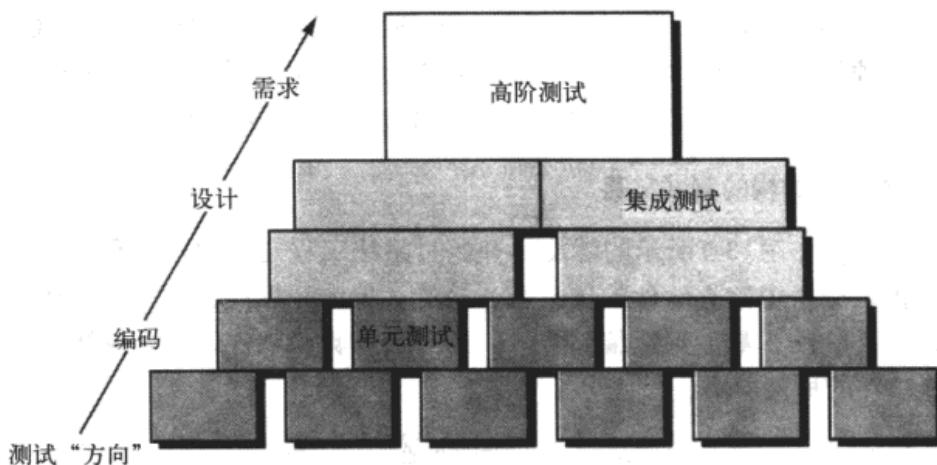


图13-2 软件测试步骤

13.1.4 面向对象软件体系结构的测试策略

面向对象系统的测试为软件工程师提出了不同的挑战。测试的定义必须拓宽以包括运用分析和设计模型中的错误发现技术（如正式技术评审）。当建立面向对象表示时，必须评估其完整性和一致性。单元测试已失去其部分意义，而且集成策略也发生了很大变化。总而言之，测试策略和测试战术（第14章）都必须考虑面向对象软件的特有特征。
[391]

KEY POINT

与传统的测试一样，面向对象测试始于“小型测试”。然而，在多数情况下，被测试的最小成分是类或相互协作类的程序包。

面向对象软件测试的总体策略在思想上与用于传统体系结构的策略是一致的，但测试方法是不同的。从“小型测试”开始，逐步走向“大型测试”。然而，当进行“小型测试”时，重点由（传统意义上的）单个模块测试变换为包含属性和操作且隐含着通信与协作的类的测试。随着将类集成为一个面向对象体系结构，运行一系列的集成测试（原书此处误为regression test（回归测试）。——译者注）以发现由于类（构件）间的通信与协作产生的错误以及新类（构件）的加入而引起的副作用。最后，作为一个整体来测试系统，以确保发现需求中的错误。

准备测试

SAFEHOME

[场景] Doug Miller的办公室，继续构件级设计，并开始某个构件的构造。

[人物] Doug Miller，软件工程经理；Vinod、Jamie、Ed和Shakira，SafeHome软件工程团队成员。

[对话]

Doug: 在我看来，似乎没有花费太多时间讨论测试。

Vinod: 是，我们都有点忙。另外，我们一直在考虑这个问题……实际上，远不止考虑。

Doug（微笑）：我知道，大家都在超负荷地工作，不过我们还得全面考虑。

Shakira: 在开始编码任何构件之前，我喜欢设计单元测试，因此我一直尽力在做测试。我有一个相当大的测试文件，一旦我的构件编码工作完成就运行这个测试文件。

Doug: 那是一种极限编程（一个轻量的软件开发方法，见第4章）概念，不是吗？

Ed: 是的。尽管我没有亲自使用极限编程，但我断定在建立构件之前设计单元测试

是一个好的思想。设计会给我们需要的所有信息。

Jamie: 我一直在做同样的事情。

Vinod: 我负责集成，因此，每次某个家伙传构件给我，我就将其集成到部分已集成的程序中并运行一系列的集成测试（原书此处误为回归测试。——译者注）。我一直忙于为系统中的每个功能设计适当的测试集。

Doug (对Vinod): 你多长时间运行一次测试？

Vinod: 每天……直到系统被集成……嗯，直到我们计划交付的软件增量被集成。

Doug: 你们已经走在我前面了。

Vinod (大笑): 在软件业务中，期待就是一切，老板。

13.1.5 测试完成的标准

什么时候我们完成测试？

每当讨论软件测试时，就会引出一个标准问题：测试什么时候才算做完？怎么知道我们已做了足够的测试？非常遗憾的是，这个问题没有权威性的答案，只是有一些实用的答复和早期的尝试可作为经验指导。

对上述问题的一个答复是：你永远也不能完成测试，这个担子只会从你（软件工程师）身上转移到你的客户身上。客户/用户每次运行计算机程序时，程序就在经受测试。这个严酷的事实突出了其他软件质量保证活动的重要性。

另一个答复（有点讽刺意味，但无疑是准确的）是：当你的时间或资金不够时，测试就完成了。

尽管少数实践者会对这些答复产生异议，但是软件工程师需要更严格的标准以确定什么时候已经做完充足的测试。Musa和Ackerman[MUS89]提出了一个基于统计标准的答复：“不，我们不能绝对地认为软件从不失败，但相对于一个理论上正确的、经过实践验证的统计模型而言，如果在按照概率方法定义的环境中，1000个CPU小时内无故障运行的概率大于0.995，那么我们就有95%的信心说已经进行了足够的测试”。利用统计建模和软件可靠性理论，可以建立以执行时间为函数的软件故障（在测试过程中发现的错误）模型（见[MUS89]、[SIN99]或[IEE01]）。

通过在软件测试过程中收集度量数据并利用现有的软件可靠性模型，对回答“测试何时做完”这种问题提出有意义的指导原则是可能的。毫无疑问，在测试的量化规则建立之前，还有很多工作要做，但是，现有的经验方法还是要比原始的直觉好得多。

13.2 策略问题

在本章的后面几节，我们探讨系统化的软件测试策略。然而，如果忽视了一些重要问题，即使最好的策略也会失败。Tom Gilb[GIL95]提出，如果想实现一个成功的软件测试策略，必须解决下述问题：

什么样的指导原则导致成功的软件测试策略？

早在开始测试之前，就要以量化的方式规定产品需求。尽管测试的主要目的是查找错误，但是一个好的测试策略也能评估其他质量特性，例如：可移植性、可维护性和易用性（第15章）。这些都应该以可测量的方式加以规定，从而保证测试结果的无歧义性。

显式地陈述测试目标。测试的特定目标应该用可测量的术语进行陈述。例如，测试的有效性、测试的覆盖率、故障出现的平均时间、发现和修正缺陷的成本、剩余缺陷的密度或出

[392]

现频率、每次回归测试的工作时间，这些都应当在测试计划中清晰地描述[GIL95]。

了解软件的用户并为每类用户建立用户轮廓。为每类用户描述交互场景的用例，侧重于测试产品的实际使用，可以减少整个测试的工作量。

WebRef

393

相当好的测试资源汇编可在www.io.com/~wazmo/qa/中找到。

建立强调“快速周期测试”的测试计划。Gilb[GIL95]建议软件工程团队“对客户有用的功能增量和（或）质量改进，学会以快速周期（2%的项目工作量）进行测试”。从这些快速周期测试中得到的反馈可用于控制质量的等级和相应的测试策略。

建立能够测试自身的“健壮”软件。软件应该利用防错技术（13.3.1）进行设计。也就是说，软件应该能够诊断某类错误。另外，软件设计应该包括自动化测试和回归测试。

测试之前，利用有效的正式技术评审作为过滤器。在发现错误方面，正式技术评审（第26章）与测试一样有效。因此，评审可以减少生产高质量软件所需的测试工作量。

实施正式技术评审以评估测试策略和测试用例本身。正式技术评审能够发现测试方法中的不一致、遗漏和明显的错误。这节省了时间，也提高了产品质量。

为测试过程建立一种持续的改进方法。测试策略应该是可以测量的。测试过程中收集的度量数据应当作为软件测试的统计过程控制方法的一部分。

“仅对最终用户需求进行测试就像由内部装饰设计师检查一个建筑物，他会以牺牲地基、大梁和管道设备为代价。”

—Boris Beizer

13.3 传统软件的测试策略

许多策略可用于测试软件。其中一个极端是，软件团队等到系统完全建成后对整个系统执行测试以期望发现错误。这种方法尽管比较受欢迎，但效果不好；可能给出的是包含许多缺陷的软件，致使客户和最终用户感到失望。另一个极端是，无论系统任何一部分何时建成，软件工程师每天都在进行测试。这种方法尽管不受多数人欢迎，但确实很有效。遗憾的是，多数软件开发者对使用后一种方法感到犹豫。到底应该怎么做呢？

多数软件团队选择介于这两者之间的测试策略。这种策略以渐进的观点对待测试，以个别程序单元的测试为起点，逐步转移到方便于单元集成的测试，最后以实施整个系统的测试而告终。以下将对这几种不同的测试进行描述。

13.3.1 单元测试

394

单元测试侧重于软件设计的最小单元（软件构件或模块）的验证工作。利用构件级设计描述作为指南，测试重要的控制路径以发现模块内的错误。测试的相对复杂度和这类测试发现的错误受到单元测试约束范围的限制。单元测试侧重于构件中的内部处理逻辑和数据结构。这种类型的测试可以对多个构件并行执行。

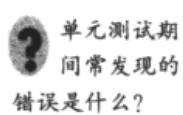
单元测试考虑。作为单元测试一部分的测试如图13-3所示。



图13-3 单元测试

测试模块的接口是为了保证被测程序单元的信息能够正常地流入和流出；检查局部数据结构以确保临时存储的数据在算法的整个执行过程中能维护其完整性；走遍控制结构中的所有独立路径（基本路径）以确保模块中的所有语句至少执行一次；测试边界条件以确保模块在到达边界值的极限或受限处理的情形下仍能正确执行。最后，要对所有的错误处理路径进行测试。

对穿越模块接口的数据流的测试要在任何其他测试开始之前进行。若数据不能正确地输入输出，其他的测试都是没有意义的。另外，应当测试局部数据结构，可能的话，在单元测试期间确定对全局数据的局部影响。



单元测试期间常发现的错误是什么？

在单元测试期间，选择测试的执行路径是最基本的任务。测试用例的设计目的旨在发现因错误计算、不正确的比较或不适当的控制流而引起的错误。计算中常见的错误有：(1) 被误解的或不正确的算术优先级；(2) 混合模式操作；(3) 不正确的初始化；(4) 不精确的精度；(5) 表达式有不正确的符号表示。比较与控制流是紧密地耦合在一起的（即，控制流的转移常常是在比较之后发生的）。测试用例应当能发现以下错误：(1) 不同数据类型的比较；(2) 不正确的逻辑运算符或不正确的优先级；(3) 应当相等但因精度的错误而不能相等；(4) 不正确的变量比较；(5) 不正确的循环终止或循环不能终止；(6) 遇到分支循环时不能退出；(7) 对循环变量的不适当的修改。

395

边界测试是单元测试中一项最重要的任务。软件通常在边界处出错，也就是说，错误行为往往出现在处理 n 维数组的第 n 个元素，或者 i 次循环的第 i 次调用，或者遇到允许出现的最大、最小数值时。使用刚好小于、等于或大于最大值和最小值的数据结构、控制流和数值作为测试用例就很有可能发现错误。

好的设计要求能够预置出错条件，且设置异常处理路径以便当错误确实出现时重定路径或彻底中断处理。Yourdon[YOU75]称这种方法为防错技术（antibugging）。遗憾的是，在软件中引入异常处理是一种趋势，然而却从未对其进行测试。下面的一个真实故事可以说明这个问题。

根据合同开发了一个计算机辅助设计系统。在其中一个事务处理模块中，一个恶作剧者在调用各种控制流分支的一系列条件测试之后加入异常处理信息：“ERROR! THERE IS NO WAY YOU CAN GET HERE”。这个“出错信息”是一个客户在“用户培训”期间发现的。



确信已经设计了执行每个异常处理路径的测试。若没有，当它被调用时可能失败，从而加重了危险的形势。



如果没有进行全面测试的资源，选择关键模块和环复杂度高的模块，仅对它们做单元测试。

当评估异常处理时，下述的潜在错误应能被测试：(1) 错误描述是难以理解的；(2) 记录的错误与真正遇到的错误不一致；(3) 在异常处理之前，错误条件就引起了操作系统的干预；(4) 不正确的异常条件处理；(5) 错误描述未提供充足的信息以便找到错误产生原因。

单元测试规程。单元测试通常被认为是编码阶段的附属工作。单元测试的设计可以在编码开始之前（作为一种可取的敏捷方法）或源代码生成之后完成。设计信息的评审可以指导建立发现前面所讨论的各类错误的测试用例，每个测试用例都应与一组预期结果联系在一起。

由于构件并不是独立的程序，因此，必须为每个测试单元开发驱动软件和（/或）桩软件。单元测试环境如图13-4所示。在大多数应用中，驱动程序只是一个“主程序”，它接收测试用例数据，将这些数据传递给（将要测试的）构件并打印相关结果。桩程序的作用是替换那些从属于将要测试的构件或被

其调用的构件。桩程序或“伪程序”使用从属模块的接口，可能做少量的数据操作，提供入口的验证，并将控制返回到被测模块。

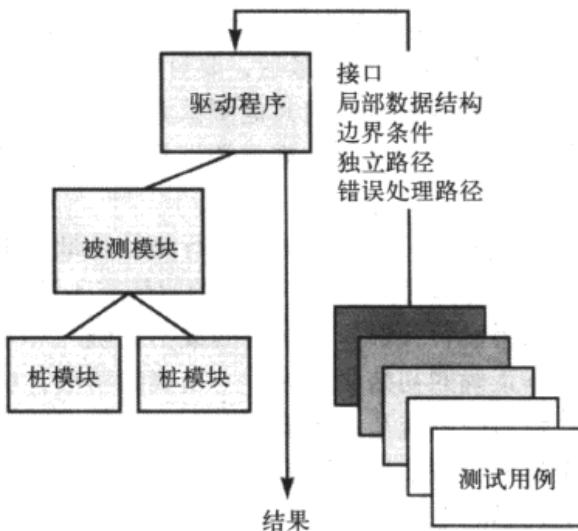


图13-4 单元测试环境

驱动程序和桩程序代表着额外的工作。也就是说，两者都是必须要写（正式的设计通常并不包括）但并不与最终的软件产品一起交付的软件。若驱动程序和桩程序比较简单，则实际的额外开销就会相对比较低。遗憾的是，使用“简单”的额外软件，许多构件是不能完成充分的单元测试的。在这种情况下，完整的测试可以延迟到集成测试这一步（这里也要使用驱动程序和桩程序）。

当构件具有高内聚性时，可简化单元测试。当一个构件只强调一个功能时，测试用例数就会降低，且比较容易预见和发现错误。

13.3.2 集成测试

软件界的初学者一旦完成所有模块的单元测试之后可能会问一个似乎很合理的问题：如果每个模块都能单独工作得很好，那么为什么要怀疑将它们放在一起时的工作情况呢？当然，这个问题涉及“将它们放在一起”——接口。数据可能在穿过接口时丢失；一个模块可能对另一个模块产生负面影响；子功能接合在一起并不能达到预期的功能；单个模块中可以接受的不精确性在连接起来之后可能会扩大到无法接受的程度；全局数据结构产生问题，等等。

集成测试是构造软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试。其目标是利用已通过单元测试的构件建立设计中描述的程序结构。



采取一步到位的集成方法是一个懒惰的策略，它注定会失败。当你进行测试时，应该采用增量集成。

常常存在一种非增量集成的倾向，即利用“一步到位”的方式来构造程序。所有的构件都事先连接在一起，全部程序作为一个整体进行测试，其结果往往是一片混乱！会出现一大堆错误。由于在整个程序的广阔区域中分离出错的原因非常复杂，因此，改正错误比较困难。一旦改正了这些错误，可能又会出现新的错误。这个过程似乎会以一个无限循环的方式继续下去。

增量集成与“一步到位”的集成方法相反。程序以小增量的方式逐步进行构造和测试，这样错误易于分离和纠正，更易于对接口进行彻底测试，而且可以运用系统化的测试方法。下面将讨论一些不同的增量测试策略。

当选择自顶向下集成方法时，可能会遇到什么问题？

自顶向下策略相对来说似乎并不复杂，但实际上可能出现逻辑上的问题。最普遍的问题出现在处理较低层次时要求对较高层进行充分测试。在自顶向下测试开始时，桩模块代替低层次的模块，因此，没有重要的数据在程序结构中向上传递。测试者只有三种选择：（1）许多测试延迟到用实际模块替换桩模块之后；（2）模拟实际模块，开发实现有限功能的桩模块；（3）利用自底向上的方式集成软件。

第一种方法（许多测试延迟到用真正模块替换桩模块之后）使我们失去了特定测试与特定模块引入间对应性的控制。这不仅会为确定错误产生原因带来一定的困难，而且会违背自顶向下方法高度受限的本质特征。第二种方法虽然可行，但随着桩模块越来越复杂，可能会产生很大的额外开销。第三种方法——自底向上测试——将在下面讨论。
399

自底向上集成。自底向上集成测试，顾名思义，就是从原子模块（程序结构的最底层构件）开始进行构造和测试。由于构件是自底向上集成的，在处理时所需要的从属于给定层次的模块总是存在的，因此，没有必要使用桩模块。自底向上集成策略可以利用以下步骤来实现：

当选择自顶向下集成方法时，可能会遇到什么问题？

自顶向下策略相对来说似乎并不复杂，但实际上可能出现逻辑上的问题。最普遍的问题出现在处理较低层次时要求对较高层进行充分测试。在自顶向下测试开始时，桩模块代替低层次的模块，因此，没有重要的数据在程序结构中向上传递。测试者只有三种选择：（1）许多测试延迟到用实际模块替换桩模块之后；（2）模拟实际模块，开发实现有限功能的桩模块；（3）利用自底向上的方式集成软件。

第一种方法（许多测试延迟到用真正模块替换桩模块之后）使我们失去了特定测试与特定模块引入间对应性的控制。这不仅会为确定错误产生原因带来一定的困难，而且会违背自顶向下方法高度受限的本质特征。第二种方法虽然可行，但随着桩模块越来越复杂，可能会产生很大的额外开销。第三种方法——自底向上测试——将在下面讨论。

399 **自底向上集成。**自底向上集成测试，顾名思义，就是从原子模块（程序结构的最底层构件）开始进行构造和测试。由于构件是自底向上集成的，在处理时所需要的从属于给定层次的模块总是存在的，因此，没有必要使用桩模块。自底向上集成策略可以利用以下步骤来实现：

自底向上的集成步骤是什么？

1. 连接低层构件以构成完成特定子功能的簇（有时称之为build）；
2. 写驱动程序（测试的控制程序）以协调测试用例的输入和输出；
3. 测试簇；
4. 去掉驱动程序，沿着程序结构向上逐步连接簇。

遵循这种模式的集成如图13-6所示。连接相应的构件形成簇1、簇2和簇3，利用驱动程序（图中的虚线框），对每个簇进行测试。簇1和簇2中的构件从属于模块M_a，去掉D₁和D₂，将这两个簇直接与M_a相连。与之相类似，在簇3与M_b连接之前去掉D₃。最后将构件M_a和M_b与M_c连接在一起，依此类推。

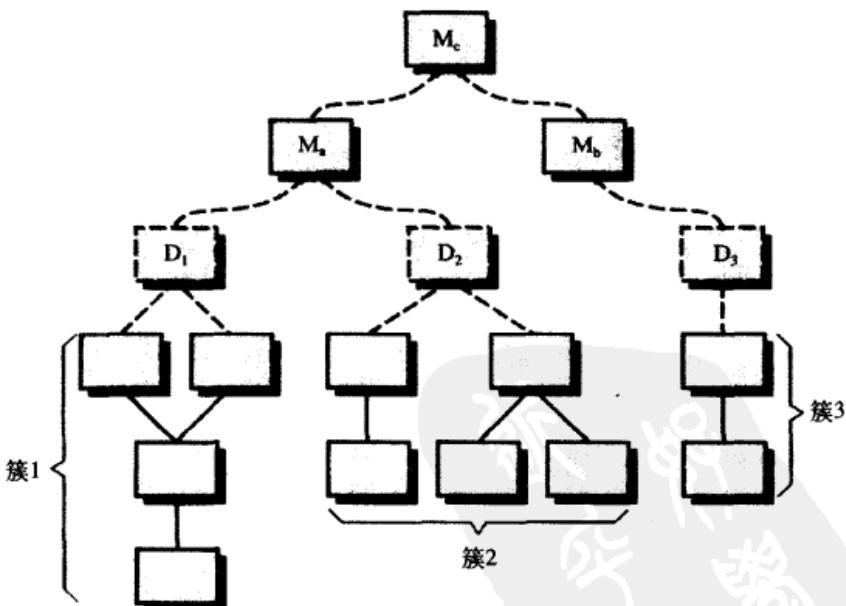


图13-6 自底向上集成

随着集成向上进行，对单独测试驱动程序的需求减少。事实上，若程序结构的最上两层是自顶向下集成的，驱动程序的数量可以大大减少，而且簇的集成得到明显简化。

回归测试。每当加入一个新模块作为集成测试的一部分时，软件发生变更，建立了新的



回归测试是减少“副作用”的重要方法。每次对软件做重要变更时（包括新构件的集成），要进行回归测试。

数据流路径，出现新的I/O，以及调用了新的控制逻辑。这些变更可能会使本来可以正常工作的功能产生问题。在集成测试策略的环境下，回归测试是重新执行已进行测试的某个子集，以确保变更没有传播不期望的副作用。

在较广的环境下，（任何种类的）成功的测试都能发现错误，且错误必须被改正。无论什么时候修正软件，软件配置的某些方面（程序、它的文档或支持数据）也发生变更。回归测试能保证变更（由于测试或其他原因）不引入无意行为或额外的错误。

回归测试可以通过重新执行所有测试用例的子集手工进行，或者利用捕捉/回放工具自动进行。捕捉/回放工具使软件工程师能够为后续的回放与比较捕捉测试用例和测试结果。回归测试套件（将要执行的测试子集）包含以下三种测试用例：

- 能够测试软件所有功能的有代表性的测试样本；
- 额外测试，侧重于可能会受变更影响的软件功能；
- 侧重于已被改变的软件构件测试。

随着集成测试的进行，回归测试的数量可能变得相当庞大，因此，回归测试套件设计应该只包括涉及每个主要程序功能的一个或多个错误类的测试。一旦发生变更，对每个软件功能重新执行所有的测试是不切实际的，而且也是低效率的。



烟幕测试被称为是一种滚动的（rolling）集成测试方法。重建软件（有新构件加入），每天执行烟幕测试。

冒烟测试。当开发软件产品时，冒烟测试是一种常用的集成测试方法，是时间关键性项目的步进机制，它让软件团队频繁地对项目进行评估。本质上，冒烟测试方法包括下列活动：

1. 将已经转换为代码的软件构件集成为“build”。一个“build”包括所有的数据文件、库、可复用的模块以及实现一个或多个产品功能所需的工程化构件；
2. 设计一系列测试以暴露影响build正确地完成其功能的错误。其目的是为了发现极有可能造成项目延迟的业务阻塞（show stopper）错误；
3. 每天该“build”与其他“build”及整个软件产品（以其当前的形式）集成起来进行冒烟测试。这种集成方法可以是自顶向下，也可以自底向上。

401

每天对整个产品进行测试可能使一些读者感到奇怪。然而，频繁的测试让管理者和专业人员对集成测试进展做出实际的评估。McConnell[MCO96]描述冒烟测试如下：

冒烟测试应该对整个系统进行彻底的测试。它不一定是穷举的，但应能暴露主要问题。冒烟测试应该足够彻底，以使得若build通过测试，则可以假定它足够稳定以致能经受更彻底的测试。

当应用于复杂的、时间关键的软件工程项目时，冒烟测试提供了下列好处：

- 降低了集成风险。由于冒烟测试是每天进行的，能较早地发现不相容性和业务阻塞错误，从而降低了因发现错误而对项目进度造成严重影响的可能性；
- 提高最终产品的质量。由于这种方法是面向构造（集成）的，因此，冒烟方法既有可能发现功能性错误，也有可能发现体系结构和构件级设计错误。若较早地改正了这些错误，则将得到较好的产品质量。
- 简化错误的诊断和修正。与所有的集成测试方法一样，冒烟测试期间所发现的错误可能

与新的软件增量有关，也就是说，刚加入“build”的软件可能是新发现错误的来源。

- 易于评估进展状况。随着时间的推移，更多的软件被集成，更多地展示出软件的工作情况。这提高了团队的士气，并使管理者对项目进展有较好的把握。

“将每天的build看作项目的心跳脉搏。若没有心跳脉搏，项目将死亡。”

——Jim McCarthy

WebRef

有关测试策略的评论可以在
www.qalinks.com 找到。

402

策略的选择。有关自顶向下和自底向上测试策略的优缺点有许多讨论（如[BEI84]）。一般来讲，一种策略的优点可能就是另一种策略的缺点。自顶向下方法的主要缺点就是需要桩以及桩带来的测试难题。有关桩的问题可以通过较早地测试主要控制功能这一优点来弥补。自底向上集成测试方法的主要缺点在于：直到加入最后一个模块，一直没有一个作为实体的程序[MYE79]。这个缺点可以通过测试用例设计简易和无需桩模块来弥补。

集成策略的选择依赖于软件的特征，有时也与项目的进度安排有关。一般来讲，组合方法（有时称之为三明治测试方法），即，用自顶向下方法测试程序结构较高层，用自底向上方法测试其从属层，这可能是最好的折衷。

什么是“关键”模块，为什么应该标识它？当执行集成测试时，测试人员应能标识关键模块。关键模块具有下述一个或多个特征：(1) 涉及几个软件需求；(2) 含有高层控制（位于程序结构相对高的层次）；(3) 是复杂的或易错的；(4) 有明确的性能需求。关键模块应尽可能早地测试。另外，回归测试应侧重于关键模块的功能。

集成测试文档。软件集成的总体计划和特定的测试描述应该在测试规格说明中文档化。这个文档包含测试计划和测试规程，它是软件过程的工作产品，也是软件配置的一部分。

测试计划描述集成的总体策略。测试可以分为若干阶段和强调软件特定功能和行为特征的“build”。例如，对于CAD系统的集成测试可以划分为以下几个测试阶段：

- 用户交互（命令选择、图形生成、显示表示、出错处理与表示）；
- 数据加工和分析（符号生成、维数转换、旋转和物理特征的计算）；
- 显示的处理和生成（二维显示、三维显示、图和表）；
- 数据库管理（访问、更新、完整性和性能）。

每个阶段和子阶段（在括号中说明的）刻划了软件内部广泛的功能类别，而且通常与软件体系结构中特定的领域相关，因此，对应于每个阶段建立了相应的程序build（模块集）。下列准则和相应的测试可应用于所有的测试阶段：

接口一致性。当每个模块（或簇）引入程序结构中时，要对其内部和外部接口进行测试；

功能有效性。执行的测试旨在发现功能错误；

信息内容。执行的测试旨在发现与局部或全局数据结构相关的错误；

性能。执行的测试旨在验证软件设计期间建立的性能边界。

集成测试的进度、附加的开发以及相关问题也在测试计划中讨论。确定每个阶段的开始和结束时间，定义单元测试模块的“可用性窗口”。附加软件（桩模块及驱动模块）的简要描述侧重于专门进行的工作的特征。最后，描述测试环境和资源。特殊的硬件配置、特殊的仿真器和专门的测试工具或技术也是需要讨论的问题。

紧接着需要描述的是实现测试计划所必需的详细测试规程。描述集成的顺序以及每个集成

403

步骤中对应的测试，其中也包括所有的测试用例列表（带注释以便后续参考）和期望的结果。

实际测试结果、问题或特例的历史要记录在测试记录中，需要的话可附在测试规格说明后面。这部分包含的信息在软件的维护期间很重要。也要给出适当的参考文献和附录。

与软件配置的其他成分一样，测试规格说明的格式可以根据软件工程组织的局部要求进行剪裁。然而，需要指出的是，集成策略（包含在测试计划中）和测试细节（在测试规程中描述）是最基本的成分，因此必须要有。

“最好的测试人员不是找到最多隐错的人……最好的测试人员是让最多的隐错得到修复的人。”

——Cem Kaner等

13.4 面向对象软件的测试策略

简单地说，测试目标就是在现实的时间范围内利用可控的工作量尽可能多地找到错误。对于面向对象软件，尽管这个基本目标是不变的，但面向对象软件的本质特征改变了测试策略和测试战术（第14章）。

13.4.1 面向对象环境中的单元测试

当考虑面向对象软件时，单元的概念发生了变化。封装导出了类的定义。这意味着每个类和类的实例（对象）包装有属性（数据）和处理这些数据的操作（函数）。封装的类常是单元测试的重点，然而，类中包含的操作是最小的可测试单元。由于类中可以包含一些不同的操作，且特殊的操作可以作为不同类的一部分存在，因此，必须改变单元测试的战术。

 **POINT**
面向对象软件的类测试类似于传统软件的模块测试。对操作进行孤立测试是不可取的。

我们不再孤立地对单个操作进行测试（传统的单元测试观点），而是将其作为类的一部分。为便于说明，考虑一个类层次结构，在此结构内对超类定义某操作X，并且一些子类继承了操作X。每个子类使用操作X，但它应用于为每个子类定义的私有属性和操作的环境内。由于操作X应用的环境有细微的差别，在每个子类的环境中测试操作X是必要的。这意味着在面向对象环境中，以独立的方式测试操作X往往是无效的。

面向对象软件的类测试等同于传统软件的单元测试。不同的是传统软件的单元测试侧重于模块的算法细节和穿过模块接口的数据，面向对象软件的类测试是由封装在该类中的操作和类的状态行为驱动的。

13.4.2 面向对象环境中的集成测试

由于面向对象软件没有明显的层次控制结构，因此，传统的自顶向下和自底向上集成策略（13.3.2）已没有太大意义。另外，由于类的成分间的直接或间接相互作用，每次将一个操作集成到类中（传统的增量集成方法）往往是不可能的[BER93]。

面向对象系统的集成测试有两种不同的策略[BIN94]。一是基于线程的测试（thread-based testing），集成响应系统的一个输入或事件所需的一组类。每个线程单独地集成和测试。应用回归测试以确保没有副效应产生。另一种方法是基于使用的测试（use-based testing），通过测试很少使用服务类（如果有的话）的那些类（称之为独立类）开始构造系统，独立类测试完

KEY POINT

面向对象软件集成测试的一个重要策略是基于线程的测试。线程是对一个输入或事件作出反应的类集合。基于使用的测试侧重于那些不与其他类进行频繁协作的类。

405

后，利用独立类测试下一层次的类（称之为依赖类）。继续依赖类的测试直到完成整个系统。

当进行面向对象系统的集成测试时，驱动程序和桩程序的使用也发生变化。驱动程序可用于测试低层中的操作和整组类的测试。驱动程序也可用于代替用户界面以便在界面实现之前就可以进行系统功能的测试。桩程序可用于在需要类间的协作但其中的一个或多个协作类仍未完全实现的情况下。

簇测试（cluster testing）是面向对象软件集成测试中的一步。这里，利用试图发现协作中的错误的测试用例来测试（通过检查CRC和对象-关系模型所确定的）协作的类簇。

13.5 确认测试

KEY POINT

与所有其他测试步骤相类似，确认测试尽力发现错误，但是它侧重于需求级的错误，即那些对最终用户是显而易见的错误。

确认测试始于集成测试的结束，那时已测试完单个构件，软件已组装成完整的软件包，且接口错误已被发现和改正。在确认测试或系统级测试时，传统软件与面向对象软件的差别已经消失，测试便集中于用户可见的动作和用户可识别的系统输出。

确认可用几种方式进行定义，但是，其中一个简单（尽管粗糙）的定义是当软件可以按照用户认为的合理的预期方式工作时，确认就算成功。在这点上，喜欢吹毛求疵的软件开发人员可能会提出异议：谁或者什么是合理预期的裁决者呢？

合理的预期在软件需求规格说明中定义，软件规格说明是描述软件用户可见属性的文档。该规格说明中包含了称之为确认准则的部分，其中包含的信息形成了确认测试方法的基础。

13.5.1 确认测试准则

软件确认是通过一系列表明已符合软件需求的测试而获得的。测试计划列出将要执行的测试类，测试规程定义了测试用例。测试计划和规程都是用于确保满足所有的功能需求，具有所有的行为特征，达到所有的性能需求，文档是正确的、可用的，且满足其他需求（如：可移植性、兼容性、错误恢复和可维护性）。

执行每个确认测试用例之后，存在下面两种可能条件之一：（1）功能或性能特征符合需求规格说明，因而被接受；（2）发现了与规格说明的偏差，创建缺陷列表。在项目的这个阶段发现的错误或偏差很难在预定的交付期之前得到修复。此时往往必须与用户进行协商，确定解决这些缺陷的方法。

13.5.2 配置评审

确认过程中的一个重要成分是配置评审。评审的目的是确保所有的软件配置元素已正确开发、编目，且具有支持软件生命周期支持阶段的必要细节。配置评审有时称之为审核（audit），将在第27章详细讨论。

13.5.3 α 测试与 β 测试

对软件开发者而言，预见用户如何实际使用一个程序几乎是不可能的。软件使用指南

406

(使用手册) 可能会被错误理解; 可能会经常使用令用户感到奇怪的数据连接; 测试者看起来很明显的输出对于工作现场的用户却是难以理解的。

当为用户开发定制软件时, 执行一系列验收测试能使用户确认所有的需求。验收测试是由最终用户而不是软件工程师进行的, 它的范围从非正式的“测试驱动”直到有计划地、系统地进行一系列测试。实际上, 验收测试的执行可能超过几个星期甚至几个月, 因此, 可以发现长时间以来影响系统的累积错误。

“给予足够的关注, 所有的隐错都是容易找到的 (例如: 给予足够多的 β 测试人员和相关的开发人员, 几乎每个问题都能很快地捕获到, 并容易地修改)。” ——E. Raymond

若软件作为一个产品由多个用户使用, 让每个用户都进行正式的验收测试是不切实际的。多数软件开发者使用称之为 α 测试与 β 测试的过程, 以期查找到似乎只有终端用户才能发现的错误。

α 测试是由最终用户在开发者的场所进行。软件在自然的环境下使用, 开发者站在典型用户的后面观看, 并记录错误和使用问题。 α 测试在受控的环境下进行。

β 测试在最终用户场所执行。与 α 测试不同, 开发者通常不在场, 因此, β 测试是在不为开发者控制的环境下软件的“现场”应用。最终用户记录测试过程中遇见的所有问题(现实存在或想像的), 并将其定期地报告给开发者。接到 β 测试的问题报告之后, 软件工程师进行修改, 然后准备向最终用户发布软件产品。

SAFEHOME

准备确认

[场景] Doug Miller的办公室, 构件级设计正继续进行, 并开始某个构件的建立。

[人物] Doug Miller, 软件工程经理; Vinod、Jamie、Ed和Shakira; SafeHome软件工程团队成员。

[对话]

Doug: 第一个增量的确认将在三个星期内准备好, 怎么样?

Vinod: 大概可以吧。集成进展得不错。我们每天执行冒烟测试, 找到了一些bug, 但到目前为止, 我们不能很好地处理任何事情。

Doug: 跟我谈谈确认。

Shakira: 可以。我们将使用所有的用例作为测试设计的基础。目前我还没有开始, 但我将为我负责的所有用例开发测试。

Ed: 我这里也一样。

Jamie: 我也一样。但是我们已经将确认测试与 α 测试和 β 测试一起考虑, 不是吗?

Doug: 是, 事实上我一直考虑请外包商帮我们做确认测试。在预算中我们有这笔钱……它将给我们新的思路。

Vinod: 我认为我们已经让确认测试在控制之中。

Doug: 我确信是这样, 但ITG(独立测试组)能用另一种眼光来看这个软件。

Jamie: 我们严格地把握着项目的进度, Doug, 我没有时间培训新人来做这个工作。

Doug: 我知道，我知道。但ITG仅根据需求和用例来工作，并不需要太多的培训。

Vinod: 我仍认为我们已经让确认测试处在控制之中。

Doug: 我知道，Vinod，但在这方面我将强制执行。计划这周的后几天与ITG见面。让他们开始工作并看他们有什么意见。

Vinod: 好的，或许这样做可以减轻工作负荷。

13.6 系统测试

在本书的开始，我们就强调过：软件只是基于计算机的大系统的一部分。最终，软件要与其他系统成分（如，硬件、人和信息）相结合，并执行一系列的集成测试和确认测试。这些测试已超出软件过程的范围，而且不仅仅由软件工程师执行。然而，软件设计和测试期间所采取的步骤可以大大提高在大系统中成功地集成软件的可能性。

“与死亡和税收一样，测试既是令人不愉快的，也是不可避免的。” ——Ed Yourdon

一个传统的系统测试问题是“相互指责”。这种情况出现在发现一个错误时，每个系统成分的开发人员都因为这个问题抱怨别人。其实大家都不应该陷入这种无味的争论之中，软件工程师应该预见潜在的接口问题，以及：(1) 设计出错处理路径，用以测试来自系统其他成分的所有信息；(2) 在软件接口处执行一系列模拟不良数据或其他潜在错误的测试；(3) 记录测试结果，这些可作为“相互指责”出现时的“证据”；(4) 参与系统测试的计划和设计，以保证软件得到充分的测试。

系统测试实际上是对整个基于计算机的系统进行一系列不同考验的测试。虽然每个测试都有不同的目的，但所有测试都是为了验证系统成分已正确地集成在一起且完成了指派的功能。408 在下面的几节，我们将讨论几种对基于软件的系统有价值的系统测试[BEI84]。

13.6.1 恢复测试

多数基于计算机的系统必须从错误中恢复并在一定的时间内重新运行。在有些情况下，系统必须是容错的，也就是说，处理错误绝不能使整个系统功能都停止。而在有些情况下，系统的错误必须在特定的时间内或严重的经济危害发生之前得到改正。

恢复测试是通过各种方式强制地让系统发生故障并验证其能适当恢复的一种系统测试。若恢复是自动的（由系统自身完成），则对重新初始化、检查点机制、数据恢复和重新启动都要进行正确性评估。若恢复需要人工干预，则估算平均恢复时间（mean-time-to-repair, MTTR）以确定其是否在可接受的范围之内。

13.6.2 安全测试

任何管理敏感信息或能够对个人造成不正当伤害（或带来好处）的计算机系统都是非礼或非法入侵的目标。入侵包括广泛的活动：黑客为了娱乐而试图入侵系统，不满的雇员为了报复而试图破坏系统，不良分子在非法利益驱使下试图入侵系统。

安全测试验证建立在系统内的保护机制是否能够实际保护系统不受非法入侵。引用Beizer[BEI84]的话来说：“系统的安全必须经受住正面的攻击——但是也必须能够经受住侧面

和背后的攻击。”

在安全测试过程中，测试者扮演试图攻击系统的角色。测试者可以试图通过外部手段获取密码；可以通过瓦解任何防守的定制软件来攻击系统；可以“制服”系统使其无法对别人提供服务；可以有目的地引发系统错误以期在其恢复过程中入侵系统；可以通过浏览非保密数据，从中找到进入系统的钥匙，等等。

只要有足够的时间和资源，好的安全测试最终将能够入侵系统。系统设计人员的作用是使攻破系统所付出的代价大于攻破系统之后获取信息的价值。

13.6.3 压力测试

本章前面所讨论的软件测试步骤，能够对正常的程序功能和性能进行彻底的评估。压力测试的目的是使软件面对非正常的情形。本质上，进行压力测试的测试人员会问：“能将系统折腾到什么程度而不会出错呢？”

压力测试是以一种要求反常数量、频率或容量的方式执行系统。例如：(1) 当平均每秒出现1~2次中断的情形下，可以设计每秒产生10次中断的测试用例；(2) 将输入数据的量提高一个数量级以确定输入功能将如何反应；(3) 执行需要最大内存或其他资源的测试用例；(4) 设计可能产生内存管理问题的测试用例；(5) 创建可能会过多查找磁盘驻留数据的测试用例。从本质上来说，压力测试者是试图破坏程序。

409

“若你正在尽力查找实际系统的隐错，且还没有为你的软件提供实际的压力测试，那么现在应该是你立即开始的时候了。”

——Boris Beizer

压力测试的一个变体称之为敏感性测试。在一些情况下（最常见的是在数学算法中），包含在有效数据界限之内的一小部分数据可能会引起极端处理情况，甚至是错误处理或性能的急剧下降。敏感性测试试图发现可能会引发系统不稳定或者错误处理的有效输入类中的数据组合。

13.6.4 性能测试

对于实时和嵌入式系统，提供所需功能但不符合性能需求的软件是不能接受的。性能测试用来测试软件在集成环境中的运行性能。性能测试可以发生在测试过程的所有步骤中。即使在单元测试中，也可以在执行测试时评估单个模块的性能。然而，只有当整个系统的所有成分都集成在一起之后，才能搞清楚系统的真实性能。

性能测试经常与压力测试一起进行，且常需要硬件和软件相配合。也就是说，在一种苛刻的环境中衡量资源（如，处理器周期）的利用往往是必要的。外部的测试设备可以监测执行间歇，当有事件（如：中断）发生或取样机定时给出信息时记录下来。通过检测系统，测试人员可以发现导致效率降低和系统故障的情形。

SOFTWARE TOOLS

测试计划与管理

目的：这些工具有助于软件团队对所选测试策略作出计划以及进行测试过程的管理。

机制：这类工具提供测试计划、测试存储量、管理与控制、需求追踪能力、集成、错误追踪和报告生成。项目经理利用这些工具计划测试活动，以及在测试进行时控制信息的流动。

410

代表性工具²

OTF (Object Testing Framework)，由MCG Software, Inc. (www.mcsoft.com) 开发，为管理Smalltalk对象的测试套件提供了一个框架。

QADirector，由Compuware Corp. (www.compuware.com/qacenter)，为管理测试过程的各个阶段提供单点控制。

TestWorks，由Software Research, Inc. (www.soft.com/Products/index.html) 开发，包含一个完整的、集成的成套测试工具，包括测试管理与测试报告。

13.7 调试技巧

软件测试是一种能够系统地加以计划和说明的活动，可以进行测试用例设计，定义测试策略，根据预期的结果评估测试结果。

调试 (debugging) 出现在成功的测试之后，也就是说，当测试用例发现错误时，调试是致使错误消除的行为。尽管调试可以是、也应该是一个有序的过程，但它仍然需要很多的技巧。当评估测试结果时，软件工程师经常面对软件问题表现出的“症状”，即，错误的外部表现与其内在原因没有明显的关系。调试就是查找问题症状与其产生原因之间的联系尚未得到很好理解的智力过程。

“只要开始编程，就会惊奇地发现并不是像我们想像地那样容易让程序正确。必须要做调试。我能记起那一刻——意识到从那时起我将花费大部分精力去查找自己程序中的错误。”

——Maurice Wilkes，“发现调试”，1949

13.7.1 调试过程

调试并不是测试，但总是发生在测试之后³。参看图13-7，调试过程开始于测试用例的执行，评估测试结果，而且期望的性能与实际性能之间缺少对应关系。在多数情况下，这种不一致表明还有隐藏的问题。调试试图将症状与原因相匹配，从而使错误得到修正。

411

调试的结果是以下两者之一：(1) 发现问题的原因并将其改正；(2) 未能找到问题的原因。在后一种情况下，调试人员可以假设一个原因，设计一个或多个测试用例来帮助验证这个假设，重复此过程直到改正错误。

为什么调试
如此困难？

为什么调试如此困难？在很大程度上，人类心理（参见下一节）与这个问题的答案间的关系比软件技术更密切。然而，错误的以下特征提供了一些线索：

1. 症状与原因出现的地方可能相隔很远。也就是说，症状可能在程序的一个地方出现，而原因实际上可能在很远的另一个地方。高度耦合的程序结构（第11章）加剧了这种情况的发生；

² 这里提到的工具并不表示本书支持这些工具，而只是此类工具的例子。在多数情况下，工具名称由各自的开发者注册为商标。

³ 此处我们考虑最广义的测试，不仅包括软件发布之前开发人员的测试，也包括用户每次使用软件时对软件的测试。

2. 症状可能在另一个错误被改正时（暂时）消失；
3. 症状实际上可能是由非错误因素（如：舍入误差）引起的；
4. 症状可能是由不易追踪的人为错误引起的；
5. 症状可能是计时问题的结果，而不是处理问题的结果；
6. 重新产生完全一样的输入条件是困难的（如：输入顺序不确定的实时应用）；
7. 症状可能是时有时无的，这在软硬件耦合的嵌入式系统中尤为常见；
8. 症状可能是由分布运行在不同处理器上一些任务引起的。

在调试过程中，我们遇到错误的范围从恼人的小错误（如不正确的输出格式）到灾难性故障（如系统失败，造成严重的经济或物质损失）。错误越严重，查找错误原因的压力也就越大。通常情况下，这种压力会使软件开发人员在修改一个错误的同时引入两个甚至更多的错误。412

“每个人都知道调试的难度是首次写程序的两倍。因此，如果你像写它时一样的聪明，那么将如何对它进行调试呢？”

——Brian Kernighan

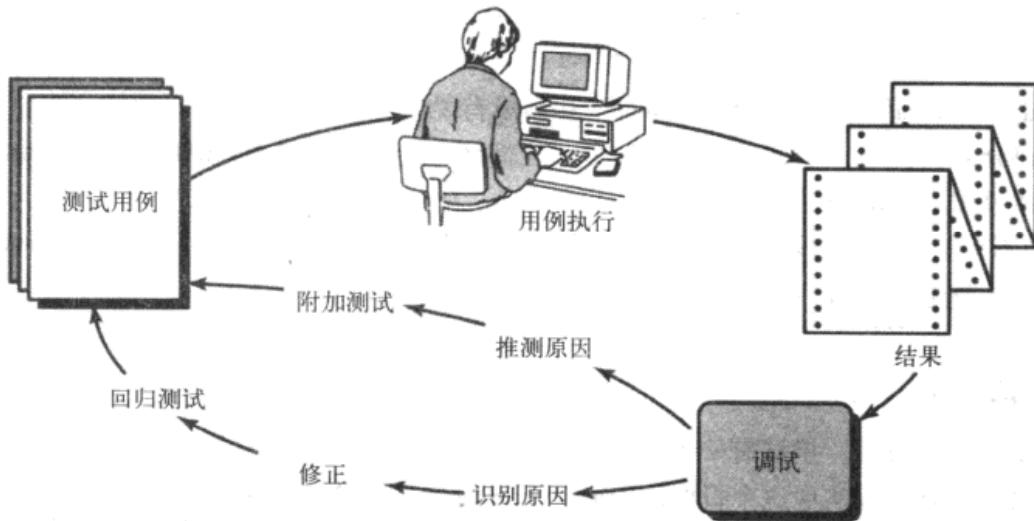


图13-7 调试过程

13.7.2 心理因素

遗憾的是，有证据表明，调试本领属于一种个人天赋。一些人精于此道，而另一些人则不行。尽管有关调试的实验证据可以有多种解释，但对于具有相同教育和经验背景的程序员来说，他们的调试能力是有很大差别的。

Shneiderman [SHN80]对调试的人为因素评论如下：

调试是编程过程中比较容易让人感到受挫的工作之一。它包含解决问题或智力测验的成分，加之不情愿承认自己犯的错误。焦虑和不情愿等原因提高了任务的难度。幸运的是，当隐错最终被修改时，调试人员松了一口气。

尽管学会调试可能比较困难，但可以提出一些解决问题的方法。这些方法将在下一节中讨论。

代表性工具²

OTF (Object Testing Framework)，由MCG Software, Inc. (www.mcsoft.com) 开发，为管理Smalltalk对象的测试套件提供了一个框架。

QADirector，由Compuware Corp. (www.compuware.com/qacenter)，为管理测试过程的各个阶段提供单点控制。

TestWorks，由Software Research, Inc. (www.soft.com/Products/index.html) 开发，包含一个完整的、集成的成套测试工具，包括测试管理与测试报告。

13.7 调试技巧

软件测试是一种能够系统地加以计划和说明的活动，可以进行测试用例设计，定义测试策略，根据预期的结果评估测试结果。

调试 (debugging) 出现在成功的测试之后，也就是说，当测试用例发现错误时，调试是致使错误消除的行为。尽管调试可以是、也应该是一个有序的过程，但它仍然需要很多的技巧。当评估测试结果时，软件工程师经常面对软件问题表现出的“症状”，即，错误的外部表现与其内在原因没有明显的关系。调试就是查找问题症状与其产生原因之间的联系尚未得到很好理解的智力过程。

“只要开始编程，就会惊奇地发现并不是像我们想像地那样容易让程序正确。必须要做调试，我能调试那一切。——麦金利从那时起我将古典文学全部忘却了，而且忘得一干二净。”



设置一个时间限制，比如说：一个小时，在这个时间限制内，你尽力独自调试程序，然后，求助。

法都失败的情况下，我们才使用这种方法。利用“让计算机自己找错误”的思想，进行内存转储，激活运行时跟踪，以及在程序中加载大量的输出语句。希望在所产生的大量信息里可以让我们找到错误原因的线索。尽管大量的信息可能最终导致成功，但更多的情况下这样做只是浪费精力和时间。首先必须进行思考！

回溯法是比较常用的调试方法，可以成功地应用于小程序中。从发现症状的地方开始，向后追踪（手工）源代码，直到发现错误的原因。遗憾的是，随着源代码行数的增加，潜在的回溯路径可能会增加到难以控制的地步。

原因排除法是通过演绎或归纳并引入二分法的概念来实现。对与错误出现相关的数据加以组织，以分离出潜在的错误原因。假设一个错误原因，利用前面提到的数据证明或反对这个假设。或者，先列出所有可能的错误原因，再执行测试逐个进行排除。若最初的测试显示出某个原因假设可能成立的话，则要对数据进行细化以定位错误。

自动调试。以上的调试方法都可以使用辅助调试工具。当试用调试策略时，调试工具为软件工程师提供半自动化的支持。Hailpern 与 Santhanam[HAI02]总结这些工具的状况时写道：“……已提出许多新的调试方法，许多商业调试环境是可用的。集成开发环境（IDE）提供了无需编译就可以捕捉特定语言预置错误（如语句结束符的丢失、变量未定义，等等）的方法。”一个引起工业界重视的领域是将必要的基本程序结构的可视化作为程序的分析手段[BAE97]。可用的工具包括各种调试编译器、动态调试辅助工具（“跟踪器”）、测试用例自动生成器和交互引用映射工具。然而，工具不能替代基于完整设计模型和清晰源代码的仔细评估。

SOFTWARE TOOLS

调试

目的：这些工具为那些必须调试软件问题的人提供自动化的帮助。其目的是洞察那些用手工调试可能难以捕捉的问题。

机制：大多数调试工具是针对特定编程语言和环境的。

代表性工具⁴

Jprobe ThreadAnalyzer，由Sitraka (www.sitraka.com) 开发，有助于线程问题的评估——死锁、延迟以及能对Java应用系统的性能产生严重影响的竞争条件。

C++Test，由Parasoft (www.parasoft.com) 开发，是一个单元测试工具，对C和C++代码的测试提供完全的支持。调试功能有助于已发现错误的诊断。

CodeMedic，由NewPlanet Software (www.newplanetsoftware.com/medic/) 开发，为标准的Unix调试器gdb提供图形界面，且实现了它的最重要特征。gdb目前支持C/C++、Java、PalmOS、各种嵌入式操作系统、汇编语言、FORTRAN和Modula-2。

BugCollector Pro，由Nesbitt Software Corp. (www.nesbitt.com/) 开发，实现了一个多用户的数据库，有助于软件团队追踪已报告的bug、维护要求，并管理调试工作流。

GNATS，一个免费应用软件 (www.gnu.org/software/gnats/)，是一组用于追踪bug报告的工具。

⁴ 这里提到的工具并不代表本书支持这些工具，而只是此类工具的举例。在多数情况下，工具名称由各自的开发者注册为商标。

415

人为因素。若不提到强有力的助手——其他人，有关调试方法和调试工具的任何讨论都是不完整的。一个新颖的观点：每个人都可能有为某个错误一直头痛的经历⁵。因此，调试的最终箴言应该是：“若所有的方法都失败了，就应该去问问别人！”。

13.7.4 错误改正

一旦找到错误，就必须纠正。但是，我们已提到过，修改一个错误可能会引入其他错误，因此，做得过多将弊大于利。Van Vleck[VAN89]提出，在进行“修改”以排除错误发生原因之前，每个软件工程师应该问以下三个问题：

何时纠正错误，我们应该问自己哪些问题？

1. 这个错误的原因在程序的另一部分也产生过吗？在多数情况下，程序的错误是由错误的逻辑模式引起的，这种逻辑模式可能会在别的地方出现。仔细考虑这种逻辑模式可能有助于发现其他错误。
2. 进行修改可能引发的“下一个错误”是什么？在改正错误之前，应该仔细考虑源代码（最好包括设计）以评估逻辑与数据结构之间的耦合。若要修改高度耦合的程序段，则应格外小心。
3. 为避免这个错误，我们首先应当做什么呢？这个问题是建立统计软件质量保证方法的第一步（第26章）。若我们不仅修改了过程，还修改了产品，则不仅可以排除现在的程序错误，还可以避免程序今后可能出现的错误。

13.8 小结

软件测试在软件过程中所占的技术工作量比例最大。然而，我们才刚刚开始理解系统化测试的计划、执行和控制的一些皮毛。

软件测试的目的是发现错误。为达到这个目标，需要计划和执行一系列的测试步骤——单元测试、集成测试、确认测试和系统测试。单元测试和集成测试侧重于验证模块的功能以及将模块集成到程序结构中去；确认测试用来展示软件需求的可追踪性；系统测试在软件集成成为较大的系统时才进行。

每个测试步骤都是通过有助于测试用例设计的一系列系统化测试技术来完成的。在每一步测试中，所考虑的软件的抽象层次都提高了。

与测试不同的是，调试必须被看作一种技术。从问题的症状显示开始，调试活动要去追踪错误的原因。在调试过程中可以利用的众多资源中，最有价值的是其他软件工程师的建议。

416

高质量的软件需要更系统的测试方法。引用Dunn和Ullman[DUN82]的论点：

高质量的软件需要更系统的测试方法。引用Dunn和Ullman[DUN82]的论点：

所需要的是贯穿整个测试过程的整体策略，而且在方法学上应当与基于分析、设计和编码的系统化软件开发一样，进行周密的计划。

在本章中，我们已详细探讨了测试策略问题，考虑了达到主要测试目标最可能需要的步骤：以一种有序的和有效的方法来发现和纠正错误。

⁵ 在设计和编码软件时，成对编程（第4章讨论过极限编程过程模型，建议成对编程作为它的一部分）的概念提供了一种调试机制。

参考文献

- [BAE97] Baecker, R., C. DiGiano, and A. Marcus, "Software Visualization for Debugging," *Communications of the ACM*, vol. 40, no. 4, April 1997, pp. 44–54, and other papers in the same issue.
- [BEI84] Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand-Reinhold, 1984.
- [BER93] Berard, E., *Essays on Object-Oriented Software Engineering*, vol. 1, Addison-Wesley, 1993.
- [BIN94] Binder, R., "Testing Object-Oriented Systems: A Status Report," *American Programmer*, vol. 7, no. 4, April 1994, pp. 23–28.
- [BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981, p. 37.
- [BRA85] Bradley, J. H., "The Science and Art of Debugging," *Computerworld*, August 19, 1985, pp. 35–38.
- [CHE90] Cheung, W. H., J. P. Black, and E. Manning, "A Framework for Distributed Debugging," *IEEE Software*, January 1990, pp. 106–115.
- [DUN82] Dunn, R., and R. Ullman, *Quality Assurance for Computer Software*, McGraw-Hill, 1982, p. 158.
- [GIL95] Gilb, T., "What We Fail to Do in Our Current Testing Culture," *Testing Techniques Newsletter*, (on-line edition, ttn@soft.com), Software Research, January 1995.
- [HAI02] Hailpern, B., and P. Santhanam, "Software Debugging, Testing and Verification," IBM Systems Journal, vol. 41, no. 1, 2002, available at <http://www.research.ibm.com/journal/sj/411/hailpern.html>.
- [IEE01] *Software Reliability Engineering, 12th International Symposium*, IEEE, 2001.
- [MCO96] McConnell, S., "Best Practices: Daily Build and Smoke Test," *IEEE Software*, vol. 13, no. 4, July 1996, pp. 143–144.
- [MIL77] Miller, E., "The Philosophy of Testing," in *Program Testing Techniques*, IEEE Computer Society Press, 1977, pp. 1–3.
- [MUS89] Musa, J. D., and A. F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, May 1989, pp. 19–27.
- [MYE79] Myers, G., *The Art of Software Testing*, Wiley, 1979.
- [SHO83] Shooman, M. L., *Software Engineering*, McGraw-Hill, 1983.
- [SHN80] Schneiderman, B., *Software Psychology*, Winthrop Publishers, 1980, p. 28.
- [SIN99] Singpurwalla, N., and S. Wilson, *Statistical Methods in Software Engineering: Reliability and Risk*, Springer-Verlag, 1999.
- [VAN89] Van Vleck, T., "Three Questions About Each Bug You Find," *ACM Software Engineering Notes*, vol. 14, no. 5, July 1989, pp. 62–63.
- [WAL89] Wallace, D. R., and R. U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software*, May 1989, pp. 10–17.
- [YOU75] Yourdon, E., *Techniques of Program Structure and Design*, Prentice-Hall, 1975.

习题与思考题

- 13.1 列出一些可能与独立测试组 (ITG) 的创建相关的问题。ITG与SQA小组由相同的人员组成吗?
- 13.2 用自己的话描述验证与确认的不同。两者都要使用测试用例设计方法和测试策略吗? [417]
- 13.3 为什么具有较高耦合度的模块难以进行单元测试?
- 13.4 谁应该完成确认测试——是软件开发人员还是软件使用者, 说明你的理由。
- 13.5 使用13.1.3节中描述的测试步骤来建立测试软件的策略总是可能的吗? 对于嵌入式系统来说, 可能会有哪些复杂性呢?
- 13.6 作为一个班级项目, 为你的安装开发调试指南。这个指南应该提供面向语言和面向系统的建议。这些建议是通过总结学校学习过程中所遇到的挫折得到的。从一个经过全班和老师评审过的大纲开始。并将这个指南在局部范围内发布。
- 13.7 项目的进度安排是如何影响集成测试的?

测试应该灌输内疚感吗？测试真的是摧毁性的吗？这些问题的回答是“不”！

本章讨论软件测试用例设计技术，该技术重点关注的是一系列符合总体测试目标及第13章所述测试策略的测试用例创建技术。

14.1 软件测试基础

of Software》, Prentice-Hall, 2002) 描述了面向对象系统的缺陷的管理技术。Beizer[BEI84]描述了有趣的“bug 的分类”，它可以产生有效的测试计划。Ball (《Debugging Embedded Microprocessor Systems》, Newnes Publishing, 1998) 详述了嵌入式微处理器系统的特定调试特征。

从网上可以获得大量的有关软件测试策略的信息资源。与软件测试策略有关的最新的 WWW 资源列表可在 SEPA Web 站点 <http://www.mhhe.com/pressman> 找到。

419

第14章 测试战术

要点浏览

概念：一旦生成了源代码，必须测试软件，以便在交付给客户之前尽可能多地发现（和改正）错误。目标是设计一组测试用例，它们发现错误的可能性很大。但是，如何做呢？这就是软件测试技术发挥作用的地方。这些技术为测试设计提供系统化的指导：（1）执行每个软件构件的内部逻辑和接口；（2）测试程序的输入和输出域以发现程序功能、行为和性能方面的错误。

人员：在测试的早期阶段，软件工程师完成所有的测试。然而，随着测试过程的进展，测试专家可能介入。

重要性：评审和其他软件质量保证活动可以且确实能够发现错误，但它们是远远不够的。每次执行程序，用户都在测试它。因此，在程序交付给客户之前，就必须以发现并消除错误为目的来执行它。为尽可能多地发现错误，必须系统化地执行测试，而且必须利用严格的技术来设计测试用例。

步骤：对于传统的应用系统，从两个不同的视角测试软件：利用“白盒”测试用例设计技术执行程序内部逻辑；利用“黑盒”测试用例设计技术确认软件需求。对于面向对象应用，源代码存在之前就开始“测试”，而源代码一旦生成，就设计一系列的测试以检查类的操作以及在该类与其他类进行协作的过程中是否存在错误。当集成类以形成一个子系统时，结合基于故障的测试方法，运用基于使用的测试以对整个相互协作的类进行彻底测试。最后，用例可辅助测试设计在软件确认阶段发现错误。在每种情况下，其基本意图都是以最少的工作量和时间来发现最大数量的错误。

工作产品：一组针对内部逻辑、接口、构件协作和外部需求的测试用例被设计出来并文档化；对期望结果进行定义；并记录实际结果。

质量保证措施：当开始测试时，改变视角，努力去“破坏”软件！规范化地设计测试用例并对测试用例进行周密的评审。另外，评估测试覆盖率并追踪错误检测活动。

关键概念

BVA
环复杂性
等价分割
流图
模式
可测试性
测试
基本路径
黑盒
类层次
控制结构
基于故障
循环
面向对象
基于场景
白盒

测试为软件工程师展示出有趣的异常现象。测试要求开发者首先抛弃“刚开发的软件是正确的”这一先入为主的观念，然后努力去构造测试用例来“破坏”软件。Beizer[BEI90]对这种情况有效地描述道：

有这样一个神话：若我们确实擅长编程，就应当不会有错误。若我们确实很认真，且每个人都使用结构化程序设计方法，使用自顶向下设计和决策表；若程序是用SQUISH写的；若我们有合适的银弹，就不会有错误。这个神话也就存在。神话中讲道，由于我们并不擅长所做的事，因此有错误存在。若不擅长，就应当感到内疚。因此，测试和测试用例的设计是对失败的承认，它注入了一针内疚剂。测试的枯燥是对我们错误的处罚。为什么被罚？为人？为什么内疚？为没能达到非人的完美境界？为没能区分另一个程序员所想的和所说的？为没有心灵感应？为没有解决人类四千年来尚未解决的交流问题？

参考文献

- [BAE97] Baecker, R., C. DiGiano, and A. Marcus, "Software Visualization for Debugging," *Communications of the ACM*, vol. 40, no. 4, April 1997, pp. 44–54, and other papers in the same issue.
- [BEI84] Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand-Reinhold, 1984.
- [BER93] Berard, E., *Essays on Object-Oriented Software Engineering*, vol. 1, Addison-Wesley, 1993.
- [BIN94] Binder, R., "Testing Object-Oriented Systems: A Status Report," *American Programmer*, vol. 7, no. 4, April 1994, pp. 23–28.
- [BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981, p. 37.
- [BRA85] Bradley, J. H., "The Science and Art of Debugging," *Computerworld*, August 19, 1985, pp. 35–38.
- [CHE90] Cheung, W. H., J. P. Black, and E. Manning, "A Framework for Distributed Debugging," *IEEE Software*, January 1990, pp. 106–115.
- [DUN82] Dunn, R., and R. Ullman, *Quality Assurance for Computer Software*, McGraw-Hill, 1982, p. 158.
- [GIL95] Gilb, T., "What We Fail to Do in Our Current Testing Culture," *Testing Techniques Newsletter*, (on-line edition, ttn@soft.com), Software Research, January 1995.
- [HAI02] Hailpern, B., and P. Santhanam, "Software Debugging, Testing and Verification," IBM Systems Journal, vol. 41, no. 1, 2002, available at <http://www.research.ibm.com/journal/sj/411/hailpern.html>.
- [IEE01] *Software Reliability Engineering, 12th International Symposium*, IEEE, 2001.
- [MCO96] McConnell, S., "Best Practices: Daily Build and Smoke Test," *IEEE Software*, vol. 13, no. 4, July 1996, pp. 143–144.
- [MIL77] Miller, E., "The Philosophy of Testing," in *Program Testing Techniques*, IEEE Computer Society Press, 1977, pp. 1–3.
- [MUS89] Musa, J. D., and A. F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, May 1989, pp. 19–27.
- [MYE79] Myers, G., *The Art of Software Testing*, Wiley, 1979.
- [SHO83] Shooman, M. L., *Software Engineering*, McGraw-Hill, 1983.
- [SHN80] Schneiderman, B., *Software Psychology*, Winthrop Publishers, 1980, p. 28.
- [SIN99] Singpurwalla, N., and S. Wilson, *Statistical Methods in Software Engineering: Reliability and Risk*, Springer-Verlag, 1999.
- [VAN89] Van Vleck, T., "Three Questions About Each Bug You Find," *ACM Software Engineering Notes*, vol. 14, no. 5, July 1989, pp. 62–63.
- [WAL89] Wallace, D. R., and R. U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software*, May 1989, pp. 10–17.
- [YOU75] Yourdon, E., *Techniques of Program Structure and Design*, Prentice-Hall, 1975.

习题与思考题

- 13.1 列出一些可能与独立测试组 (ITG) 的创建相关的问题。ITG与SQA小组由相同的人员组成吗?
- 13.2 用自己的话描述验证与确认的不同。两者都要使用测试用例设计方法和测试策略吗? [417]
- 13.3 为什么具有较高耦合度的模块难以进行单元测试?
- 13.4 谁应该完成确认测试——是软件开发人员还是软件使用者, 说明你的理由。
- 13.5 使用13.1.3节中描述的测试步骤来建立测试软件的策略总是可能的吗? 对于嵌入式系统来说, 可能会有哪些复杂性呢?
- 13.6 作为一个班级项目, 为你的安装开发调试指南。这个指南应该提供面向语言和面向系统的建议。这些建议是通过总结学校学习过程中所遇到的挫折得到的。从一个经过全班和老师评审过的大纲开始。并将这个指南在局部范围内发布。
- 13.7 项目的进度安排是如何影响集成测试的?

422 该设计一个测试集来测试鼠标，以试图显示鼠标位置识别中的错误。

2. 好的测试是不冗余的。测试时间和资源是有限的，执行与另一个测试有同样目标的测试是没有意义的。每个测试应该有不同的目标（即使是细微的差别）。

3. 好的测试应该是“最佳品种”[KAN93]。在一组具有类似目的的测试中，时间和资源的有限性可能只影响这些测试的一个子集的运行。在这种情况下，应该使用最有可能发现所有类的错误的测试。

4. 好的测试应该既不太简单也不太复杂。尽管将一系列测试连接为一个测试用例有时是可能的，但潜在的副作用会掩盖错误。通常情况下，应该独立执行每个测试。

SAFEHOME

设计独特的测试

[场景] Vinod的工作间。

[人物] Vinod与Ed，SafeHome软件工程团队成员。

[对话]

Vinod: 这些是你打算用于测试操作passwordValidation的测试用例。

Ed: 是的，它们应该能覆盖用户进入时所有可能输入的密码。

Vinod: 让我看看……你提到真正的密码是8080，对吗？

Ed: 嗯。

Vinod: 你指定密码为1234和6789是要测试在识别无效代码方面的错误？

Ed: 对，我也测试与正确密码相接近的密码，如……8081和8180。

Vinod: 那是可行的。但是我并不认为既运行1234，又运行6789有太大意义。那是重复的……测试同样的事情，不是吗？

Vinod: 对。但若运行1234不能发现错误……换句话说……操作passwordValidation指出它是无效密码，6789也不可能让我们发现新的东西。

Ed: 我明白你的意思。

Vinod: 我不是吹毛求疵……只是想限制我们做测试的时间，因此，运行最有可能找到新错误的测试是一个好主意。

Ed: 没问题……让我再好好想想。

14.2 黑盒测试与白盒测试

任何工程化的产品（以及大多数其他东西）都可以采用以下两种方式之一进行测试：

(1) 了解已设计的产品所完成的指定功能，可以执行测试以显示每个功能是可操作的，同时，查找在每个功能中的错误；(2) 了解产品的内部运行情况，可以执行测试以确保“所有齿轮吻合”——即内部操作依据规格说明执行，而且对所有的内部构件已进行了充分测试。第一种测试方法称为黑盒测试，第二种方法称为白盒测试²。

黑盒测试暗指在软件接口处执行测试。黑盒测试检查系统的基本方面，很少关心软件的内部结构。软件的白盒测试是基于过程细节的封闭检查。通过提供检查特定条件集和（或）

² 术语功能测试和结构测试有时分别用于代替黑盒测试和白盒测试。

循环的测试用例，测试贯穿软件的逻辑路径和构件间的协作。

“设计测试用例的唯一规则：覆盖所有特征，但并不创建太多测试用例。”

——Tsuneo Yamaura



仅在构件设计
(或有源代码) 之
后设计白盒测试。
程序的逻辑细节
必须可以得到。

乍一看，好像是全面的白盒测试将获得“百分之百正确的程序”。需要我们做的只是识别所有的逻辑路径，开发相应的测试用例并执行它们，评估结果；即，生成测试用例以逐个地测试程序逻辑。遗憾的是，穷举测试提出某种逻辑问题（见下文讨论）。然而，不应该觉得白盒测试不切实际而抛弃这种方法。可以选择并测试有限数量的重要逻辑结构，检测重要数据结构的有效性。

INFO

穷举测试

考虑100行的C语言程序。一些基本的数据声明之后，程序包含两个嵌套循环，依靠输入指定的条件，每个循环从1到20次。在内部循环中，需要4个if-then-else结构。这个程序中大约有 10^{14} 个可能的执行路径！

为了说明这个数字代表的含义，我们假想一个神奇的测试处理器（“神奇”意味着没有这样的处理器存在），在1毫秒内，处理器可以开发一个测试用例并执行它及评估相应的测试结果。若处理器每天工作24小时，每年工作365天，则穷举测试这个程序需要3170年。不可否认，这将对开发进度造成巨大的障碍。

因此，可以肯定地说，对于大型软件系统，穷举测试是不可能的。

14.3 白盒测试

白盒测试，有时也称为玻璃盒测试，是一种测试用例设计方法，它利用作为构件层设计的一部分而描述的控制结构来生成测试用例。利用白盒测试方法，软件工程师设计的测试用例可以：(1) 保证一个模块中的所有独立路径至少被执行一次；(2) 对所有的逻辑值均需测试真(true) 和假(false)；(3) 在上下边界及可操作的范围内执行所有的循环；(4) 检验内部数据结构以确保其有效性。

“错误隐藏在角落里，集聚在边界处。”

——Boris Beizer

424

14.4 基本路径测试

基本路径测试是由Tom McCabe[MCC 76]首先提出的一种白盒测试技术。基本路径测试方法使测试用例设计者产生一种过程设计(procedural design)的逻辑复杂性测度(logical complexity measure)，这种测度为执行路径的基本集的定义提供指导。执行该基本集所生成的测试用例保证程序中的每一条语句至少执行一次。

14.4.1 流图表示

在介绍基本路径方法之前，必须介绍一种简单的控制流表示方法³，即流图(或程序图)。

³ 事实上，不使用流图也可以执行基本路径测试方法，但是，流图是理解控制流和说明方法的一种有用表示。

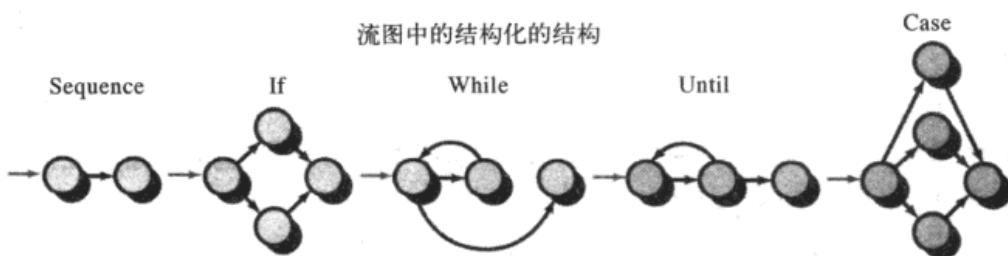


仅当一个构件的逻辑结构是复杂的情况下，才应该画流图。这个图可以更迅速地实现程序路径的追踪。

425

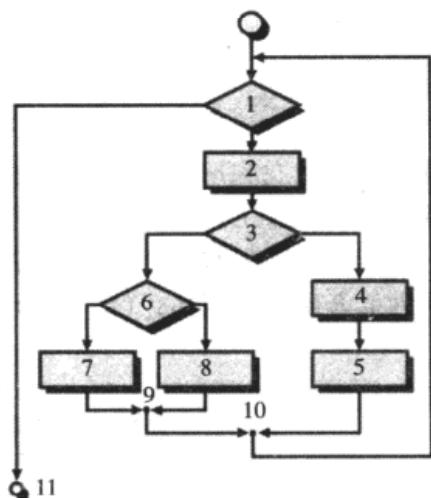
流图利用图14-1所示的表示描述逻辑控制流。每种结构化的结构（第11章）有相应的流图符号。

为说明流图的使用，我们考虑图14-2a所示的过程设计表示。这里，流程图用于描述程序控制结构。图14-2b将这个流程图映射为相应的流图（假设流程图的菱形判定框中不包含复合条件）。在图14-2b中，圆称为流图结点（flow graph node），表示一个或多个过程语句（procedural statement）。处理框序列和一个菱形判定框可以映射为单个结点。流图中的箭头称为边或连接，表示控制流，类似于流程图中的箭头。一条边必须终止于一个结点，即使该结点并不代表任何过程语句（如图14-1中if-else-then结构的流图符号）。由边和结点限定的区间称为域。当计算域时，将图形的外部作为一个域⁴。



图中，每个圆圈表示一个或多个无分支的PDL语句或源代码语句

图14-1 流图表示



a) 流程图

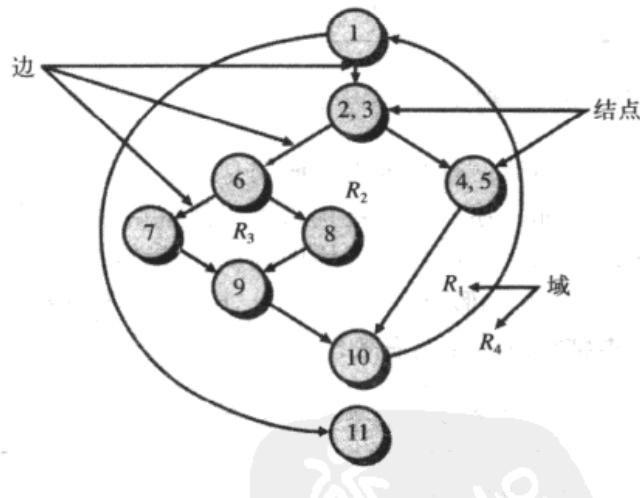


图 14-2

当在过程设计中遇到复合条件时，流图的生成变得稍微复杂一些。当一个条件语句中存在一个或多个布尔运算符（逻辑OR、AND、NAND和NOR）时，复合条件出现。图14-3给出了一段PDL程序及其对应的流图。注意，分别为条件语句“IF a OR b”的每个条件（a和b）创建不同的结点。包含条件的结点称为判定结点，用两条或多条由它发射出的边来描述。

⁴ 14.6.1节详细讨论了图及其在测试中的用法。

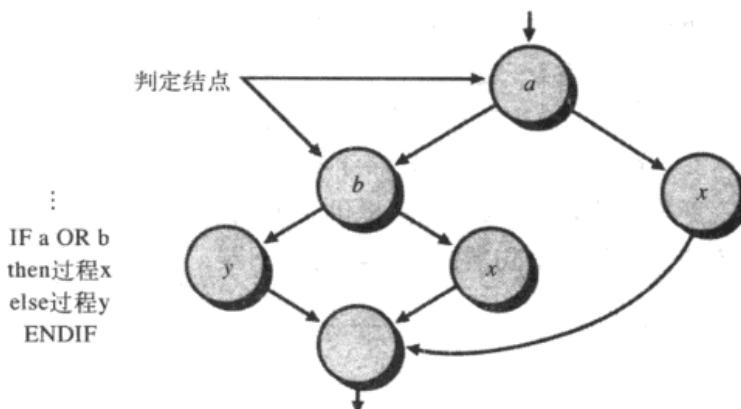


图14-3 复合逻辑

14.4.2 独立程序路径

独立路径是任何贯穿程序的、至少引入一组新的处理语句或一个新的条件的路径。当按照流图进行描述时，独立路径必须沿着至少一条边移动。这条边在定义该路径之前未被遍历。例如，图14-2b所示的流图的一组独立路径如下：

路径1: 1-11

路径2: 1-2-3-4-5-10-1-11

路径3: 1-2-3-6-8-9-10-1-11

路径4: 1-2-3-6-7-9-10-1-11

注意，每条新的路径引入一条新边，路径

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

不是一条独立路径，因为它不过是已提到路径的简单连接，而没有引入任何新边。



在预见易于出错的模块方面，环复杂性是一种有用的度量。利用它制定测试计划及进行测试用例设计。

如何计算环复杂性？

路径1、2、3和4构成图14-2b所示流图的基本集合。也就是说，若设计测试强迫执行这些路径，则可以保证程序中的每条语句至少执行一次，且执行每个条件值为真和为假。应该注意到，基本集合不是唯一的。事实上，对给定的过程设计，可以产生一些不同的基本集合。

如何知道要找出多少路径？环(cyclomatic)复杂性的计算提供了这个答案。环复杂性是一种软件度量，它为程序的逻辑复杂度提供一个量化的测度。当用在基本路径测试方法的环境下，环复杂性的值是用基本集合定义程序的独立路径数，它为我们提供了保证所有语句被执行一次所需测试数量的上限。

环复杂性以图论为基础，可以通过以下三种方法之一来计算：

1. 域的数量与环复杂性相对应。
2. 对流图G，环复杂性V(G) 定义如下：

$$V(G) = E - N + 2$$

其中E为流图的边数，N为流图的结点数。

3. 对流图G，环复杂性V(G) 也可以定义如下：

$$V(G) = P + 1$$

其中P为包含在流图G中的判定结点数。

426

427



环复杂性提供保证程序中每条语句至少执行一次所需测试用例数的上限。

再回到图14-2b中的流图，环复杂性可以通过上述三种算法来计算：

1. 该流图有4个域。
2. $V(G) = 11(\text{边数}) - 9(\text{结点数}) + 2 = 4$ 。
3. $V(G) = 3(\text{判定结点数}) + 1 = 4$ 。

更重要的是， $V(G)$ 的值提供了组成基本集的独立路径的上界，并由此得出覆盖所有程序语句所需测试数量的上界。

SAFEHOME

使用环复杂性

[场景] Shakira的工作间。

[人物] Vinod和Shakira，SafeHome软件工程团队成员，他们正在为安全功能的测试准备测试计划。

[对话]

Shakira: 看……我知道应该对安全功能的所有构件进行单元测试，但是，如果你考虑所有必须测试的操作的数量，工作量太大，我不知道……可能我们应该放弃白盒测试，将所有的构件集成在一起，开始执行黑盒测试。

Vinod: 你估计我们没有足够的时间做构件测试、检查操作，然后集成，是不是？

Shakira: 第一次增量测试的最后期限离我们很近了……是的，我有点担心。

Vinod: 你为什么不对最有可能出错的操作执行白盒测试呢？

Shakira（愤怒地）：我怎么能够准确地知道哪个是最易出错的呢？

Vinod: 环复杂性。

Shakira: 噢？

Vinod: 环复杂性。只要计算每个构件中每个操作的环复杂性。看看哪些操作 $V(G)$ 具有最高值。那些操作就是最有可能出错的操作。

Shakira: 怎么计算 $V(G)$ 呢？

Vinod: 那相当容易。这里有本书说明了怎么计算。

Shakira（翻看那几页）：好了，这计算看上去并不难。我试一试。具有最高 $V(G)$ 值的就是要做白盒测试的候选操作。

Vinod: 但还要记住，这并不是绝对的，那些 $V(G)$ 值低的构件还是可能有错的。

Shakira: 好吧。但这至少缩小了必须进行白盒测试的构件数。

14.4.3 导出测试用例

基本路径测试方法可以应用于过程设计或源代码。在本节中，我们将基本路径测试描述为
428 一系列步骤。以图14-4中用PDL描述的过程average为例，说明测试用例设计方法中的各个步骤。注意，尽管过程average是一个非常简单的算法，但却包含了复合条件与循环。下列步骤可用于生成基本测试用例集：

1. 以设计或源代码为基础，画出相应的流图。利用14.4.1节给出的符号和构造规则创建一个流图。参见图14-4中过程average的PDL描述，将那些PDL编号语句映射到相应的流图结点，以此来创建流图。图14-5给出了相应的流图。

```

PROCEDURE average;
  * This procedure computes the average of 100 or fewer
  numbers that lie between bounding values; it also computes the
  sum and the total number valid.

  INTERFACE RETURNS average, total.input, total.valid;
  INTERFACE ACCEPTS value, minimum, maximum;

  TYPE value[1:100] IS SCALAR ARRAY;
  TYPE average, total.input, total.valid;
    minimum, maximum, sum IS SCALAR;
  TYPE i IS INTEGER;

  ① { i = 1;
    total.input = total.valid = 0; ②
    sum = 0;
    DO WHILE value[i] <> -999 AND total.input < 100 ③
      ④ increment total.input by 1;
      IF value[i] >= minimum AND value[i] <= maximum ⑥
        THEN increment total.valid by 1;
        sum = sum + value[i]
      ELSE skip
      ⑧ } ENDIF
      increment i by 1;
    ⑨ ENDDO
    IF total.valid > 0 ⑩
      ⑪ THEN average = sum / total.valid;
    ⑫ ELSE average = -999;
    ⑬ ENDIF
  END average

```

图14-4 已标识结点的PDL

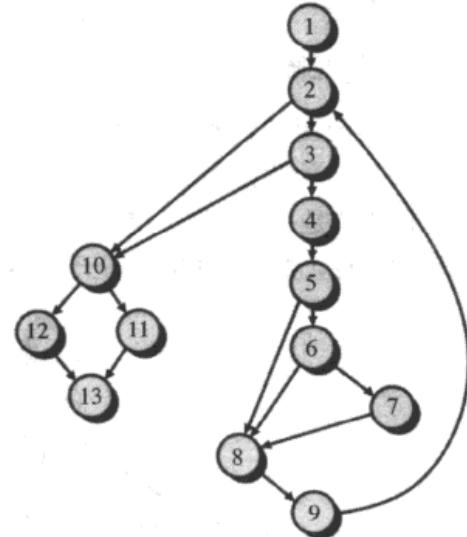


图14-5 过程average的流图

2. 确定所得流图的环复杂性。通过运用14.4.2节中描述的算法来确定环复杂性 $V(G)$ 的值。应该注意到，确定 $V(G)$ 可以不建立流图，而通过计算PDL中条件语句的数量（过程average的复合条件语句计数为2）然后加1也可以得到。在图14-5中，

$$V(G) = 6(\text{域数})$$

$$V(G) = 17(\text{边数}) - 13(\text{结点数}) + 2 = 6$$

$$V(G) = 5(\text{判定结点数}) + 1 = 6$$

“犯错误的是人，找到错误的是神。”

—Robert Dunn

429

3. 确定线性独立路径的基本集合。 $V(G)$ 的值提供了程序控制结构中线性独立路径的数量。在过程average中，我们指定了6条路径：

路径1: 1-2-10-11-13

路径2: 1-2-10-12-13

路径3: 1-2-3-10-11-13

路径4: 1-2-3-4-5-8-9-2-...

路径5: 1-2-3-4-5-6-8-9-2-...

路径6: 1-2-3-4-5-6-7-8-9-2-...

路径4、5和6后面的省略号 (...) 表示可加上控制结构其余部分的任意路径。在设计测试用例的过程中，经常通过识别判定结点作为导出测试用例的辅助手段。本例中，结点2、3、5、6和10为判定结点。

4. 准备测试用例，强制执行基本集合中每条路径。测试人员应该选择测试数据，以便在

测试每条路径时适当地设置判定结点的条件。执行每个测试用例并将结果与期望值进行比较。一旦完成了所有的测试用例，测试人员可以确信程序中所有的语句至少已被执行一次。

重要的是应该注意到，某些独立路径（本例中的路径1）不能单独进行测试。也就是说，遍历路径所需的数据组合不能形成程序的正常流。在这种情况下，这些路径作为另一个路径的一部分进行测试。

430

14.4.4 图矩阵

导出流图甚至确定基本路径集合的过程都可以机械化。开发有助于基本路径测试的软件工具，有一种数据结构——称为图矩阵（graph matrix）——相当有用。

图矩阵是一个方阵，其大小（即行与列的数量）等于流图的结点数。每行和每列都对应于已标识的结点，矩阵中的项对应于结点间连接（边）。图14-6给出了一个简单流图及相应的图矩阵。

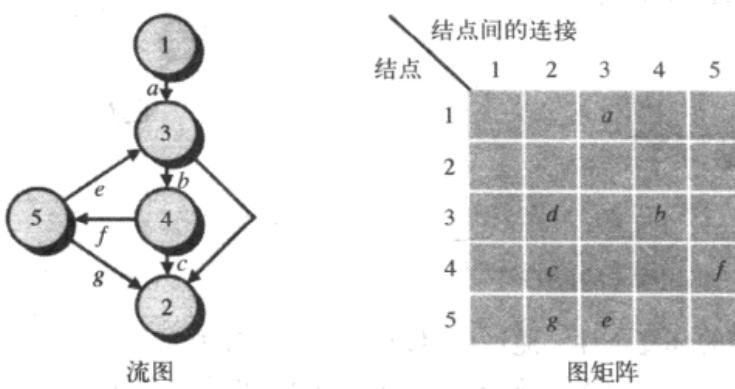


图14-6 图矩阵

如图14-6所示，流图的每个结点用数字标识，而每条边用字母标识。矩阵中的每个字母对应于纵横方向两结点间的连接，例如，边b连接结点3和结点4。

什么是图矩阵，如何对其进行扩展以用于测试？

从这种意义上讲，图矩阵只是流图的表格表示。然而，通过将每个矩阵项加入一个连接权值（link weight），图矩阵成为测试期间评估程序控制结构的一个强有力的工具。连接权值提供了有关控制流的附加信息。最简单的情况下，连接权值是1（存在连接）或0（不存在连接），但是，连接权值可以赋予其他更有意义的特征：

- 执行连接（边）的概率。
- 遍历连接的处理时间。
- 遍历连接时所需要的内存。
- 遍历连接时所需要的资源。

Beizer[BEI90]提供了可用于图矩阵的其他数学算法的全面讨论。利用这些技术，设计测试用例时所需进行的分析可以部分或完全自动化。

431

“将运行测试比设计测试看得更重要是一个典型的错误。”

——Brian Marick

14.5 控制结构测试

14.4节所描述的基本路径测试是控制结构测试技术之一。尽管基本路径测试是简单且有效

的，但其本身并不充分。本节简单讨论控制结构测试的变体，这些技术拓宽了测试的覆盖率并提高了白盒测试的质量。

14.5.1 条件测试



错误在逻辑条件附近比在顺序处理语句中更常见。

条件测试[TAI89]通过检查程序模块中包含的逻辑条件进行测试用例设计。简单条件是一个布尔变量或前面可能带有NOT (\neg) 算符的关系表达式。关系表达式的形式如下：

$$E_1 < \text{关系算符} > E_2$$

其中， E_1 和 E_2 是算术表达式，而 $<$ 关系算符 $>$ 是下列之一：“ $<$ ”、“ \leq ”、“ $=$ ”、“ \neq ”（不等于）、“ $>$ ”或“ \geq ”。复合条件由两个或多个简单条件、布尔算符和括号组成。假定可用于复合条件的布尔算符包括OR (\mid)、AND ($\&$) 和NOT (\neg)。不含关系表达式的条件称为布尔表达式。因此，组成条件的元素中可能的类型包括：布尔算符、布尔变量、一对括号（括住简单或复合布尔条件）、关系算符或算术表达式。

若条件不正确，则至少有一个条件元素不正确。因此，条件中错误类型包括布尔算符错误（不正确/遗漏/额外的布尔算符）、布尔变量错误、括号错误、关系算符错误以及算术表达式错误。条件测试方法侧重于测试程序中的每个条件以确保其不包含错误。

14.5.2 数据流测试

数据流测试方法就是根据变量的定义和使用位置来选择程序测试路径的测试方法。为说明数据流测试方法，假设程序的每条语句都赋予了独特的语句号，而且每个函数都不改变其参数或全局变量。对于语句号为 S 的语句，

$$\begin{aligned}\text{DEF}(S) &= \{X \mid \text{语句 } S \text{ 包含 } X \text{ 的定义}\} \\ \text{USE}(X) &= \{S \mid \text{语句 } S \text{ 包含 } X \text{ 的使用}\}\end{aligned}$$

若语句 S 是一个if或loop循环的语句，它的DEF集为空而USE集取决于 S 的条件。若存在 S 到 S' 的路径且该路径不含 X 的其他定义，则称变量 X 在语句 S 处的定义在语句 S' 仍有效。432



假设数据流测试广泛地用于大型系统的测试中是不太实际的。然而，它可以以特定的方式用于值得怀疑的软件区域。

变量 X 的定义-使用链（或称DU链）的形式为 $[X, S, S']$ ，其中 S 和 S' 为语句号， X 在 $\text{DEF}(S)$ 和 $\text{USE}(S')$ 中，且在语句 S 中定义的 X 在语句 S' 中有效。

一个简单的数据流测试策略要求每个DU链至少覆盖一次，我们称之为DU测试策略。已经证明DU测试并不能保证覆盖程序的所有分支。然而，DU测试不覆盖每个分支仅在if-then-else中的then没有定义变量且不存在else部分的情况下。在这种情况下，if语句的else分支并不需要由DU测试覆盖。

有关数据流测试的研究和比较已做了不少工作（如：[FRA88]、[NTA88]、[FRA93]）。有兴趣的读者可以参看其他参考文献。

“好的测试人员是一注意到‘奇怪的事情’就会对它采取行动的大师。”

——Brian Marick

14.5.3 循环测试

循环是大多数软件实现算法的重要部分。然而，我们在进行软件测试时往往却很少关注它。

循环测试是一种白盒测试技术，它仅侧重于循环构成元素的有效性。可以定义四种不同的循环[BEI90]：简单循环、串接循环、嵌套循环和非结构化循环（如图14-7所示）。

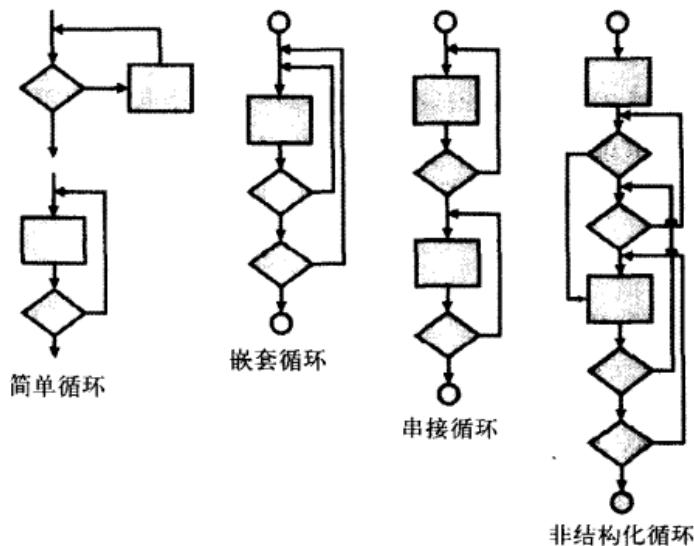


图14-7 循环的类别

433

简单循环。下列测试集可用于简单循环，其中， n 是允许通过循环的最大次数。

1. 跳过整个循环；
2. 只有一次通过循环；
3. 两次通过循环；
4. m 次通过循环，其中 $m < n$ ；
5. $n - 1, n, n + 1$ 次通过循环。

嵌套循环。若将简单循环的测试方法扩展应用于嵌套循环，则可能的测试数将随着嵌套层次的增加而成几何级数增长。这将导致不切实际的测试数量。Beizer[BEI90]提出了一种有助于减少测试数的方法：

1. 从最内层循环开始，将其他循环设置为最小值；
2. 对最内层循环执行简单循环测试，而使外层循环的迭代参数值（即循环计数）最小，并对范围以外或不包括在内的值增加其他测试；
3. 由内向外构造下一个循环的测试，但使其他外层循环具有最小值并使其他嵌套循环为“典型”值；
4. 继续上述过程，直到测试完所有的循环。



不能对非结构化循环进行有效测试，需要对它们

方法。然而，若两个循环串接起来，且第一个循环的循环计数为第二个循环的初始值，则这两个循环并不独立。若循环不独立，建议使用嵌套循环的测试方法。

串接循环。若串接循环的每个循环彼此独立，则可以使用简单循环测试

非结构化循环。若有可能，应该重新设计这类循环以反映结构化程序结

构的使用（第11章）。

14.6 黑盒测试

黑盒测试，也称行为测试，侧重于软件的功能需求。即，黑盒测试使软件工程师能设计

出将测试程序所有功能需求的输入条件集。黑盒测试并不是白盒测试的替代品，而是作为发现其他类型错误的辅助方法。

黑盒测试试图发现以下类型的错误：(1) 功能不正确或遗漏；(2) 接口错误；(3) 数据结构或外部数据库访问错误；(4) 行为或性能错误；(5) 初始化和终止错误。

与白盒测试不同，白盒测试在测试过程的早期执行，而黑盒测试倾向于应用在测试的后期阶段（第13章）。黑盒测试故意不考虑控制结构，而是侧重于信息域。设计黑盒测试要回答下述问题：

POINT
哪些是黑盒
测试必须回
答的问题？

- 如何测试功能的有效性？
- 如何测试系统的行为和性能？
- 哪种类型的输入会产生好的测试用例？
- 系统是否对特定的输入值特别敏感？
- 如何分离数据类的边界？
- 系统能承受什么样的数据率和数据量？
- 特定类型的数据组合会对系统的运行产生什么样的影响？

通过运用黑盒测试技术，可以生成满足下述准则的测试用例集[MYE79]：(1) 能够减少达到合理测试所需的附加测试用例数；(2) 能够告知某些错误类型是否存在，而不是仅仅知道与特定测试相关的错误。

14.6.1 基于图的测试方法

POINT
图表示数据对象
与程序对象间的关系，它使我们能够设计查找与这些关系有关错误的测试用例。

黑盒测试的第一步是理解软件中建模的对象⁵及这些对象间的关系。一旦这步完成，第二步就是定义一系列验证“所有对象之间具有预期的关系”的测试[BEI95]。换而言之，软件测试首先是创建重要对象及其关系图，然后设计一系列测试用例以检查对象关系并发现错误。

为完成这些步骤，软件工程师首先创建图，其中结点表示对象，连接表示对象间的关系，结点权值描述结点的属性（例如：具体的数据值或状态行为），连接权值描述连接的某些特征。

图的符号表示如图14-8a所示。节点用圆表示。而连接有几种形式，有向关系（用箭头表示）表示这种关系只在一个方向存在，双向连接（也称对称连接）表示关系适用于两个方向，并行连接表示图结点间有几种不同的关系。

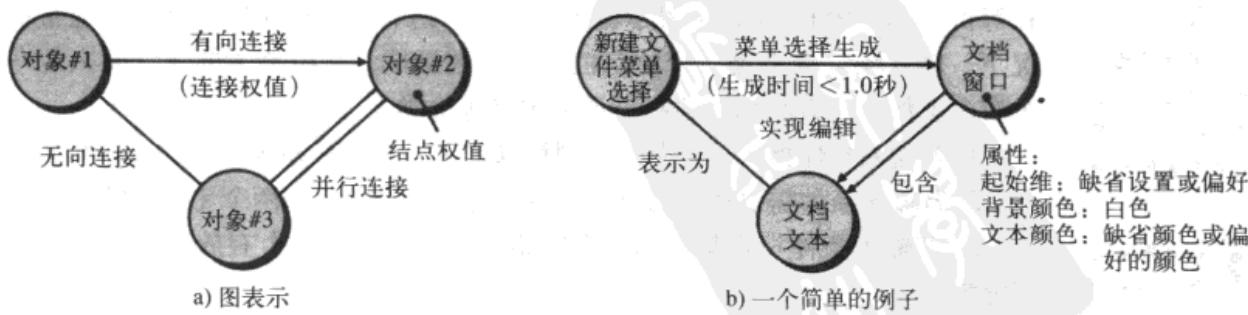


图 14-8

⁵ 在这里，我们在最广泛的环境下考虑术语“对象”。它包括数据对象、传统的构件（模块）以及计算机软件的面向对象元素。

考虑一个简单的例子，字处理应用图的一部分，如图14-8b所示，其中

对象#1=新建文件（菜单选择）

对象#2=文档窗口

对象#3=文档文本

该图中，选择菜单“新建文件”生成一个“文档窗口”。“文档窗口”的节点权值提供窗口生成时预期的属性集。连接权值表明必须在1.0秒之内生成。一条无向连接在“新建文件”菜单选择和“文档文本”之间建立对称关系。并行连接显示“文档窗口”与“文档文本”间的关系。事实上，设计测试用例还需要更详细的图描述。然后软件工程师通过遍历图并覆盖图中所示的关系来设计测试用例。这些测试用例用于发现各种关系中的错误。

Beizer[BEI95]描述了几种使用图的行为测试方法：

436 事务流建模。节点表示事务的步骤（例如：利用联机服务预订机票所需的步骤）。连接表示这些步骤间的逻辑连接。数据流图（第8章）可用于辅助创建这种类型的图。

有限状态建模。节点表示用户可见的不同软件状态（例如：订票人员处理订票时的各个屏幕）。连接代表状态间的转换。状态图（第8章）可用于辅助创建这种图。

数据流建模。节点表示数据对象，而连接为一个数据对象转换为其他数据对象时发生的变换。例如，节点FICA tax withheld(FTW) 由gross wages(GW) 利用关系 $FTW = 0.62 \times GW$ 计算出来。

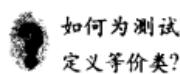
时间建模。节点为程序对象，连接是对象间的顺序连接。连接权值用于指定程序执行时所需的执行时间。

基于图测试方法的详细讨论超出了本书的范围。感兴趣的读者看[BEI95]可以对其有全面的了解。

14.6.2 等价划分

等价划分是一种黑盒测试方法，它将程序的输入划分为若干个数据类，从中生成测试用例。理想的测试用例是可以单独地发现一类错误（例如：所有字符数据处理不正确），否则在观察到一般的错误之前需要运行许多测试用例。等价划分试图定义一个测试用例以期发现一类错误，由此减少所需设计测试用例的总数。


在软件过程中的相对早期阶段，输入类就是已知的。为此，随着设计的创建，开始考虑等价划分问题。

 如何为测试定义等价类？

等价划分的测试用例设计是基于对输入条件的等价类的评估。利用上节引入的概念，若对象可以由具有对称性、传递性和自反性的关系连接，则存在等价类[BEI95]。等价类表示输入条件的一组有效的或无效的状态。通常情况下，输入条件要么是一个特定值、一个数据域、一组相关的值，要么是一个布尔条件。可以根据下述指导原则定义等价类：

1. 若输入条件指定一个范围，则可以定义一个有效和两个无效的等价类；
2. 若输入条件需要特定的值，则可以定义一个有效和两个无效的等价类；
3. 若输入条件指定集合的某个元素，则可以定义一个有效和一个无效的等价类；
4. 若输入条件为布尔值，则可以定义一个有效和一个无效的等价类。

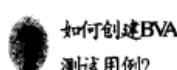
通过运用设计等价类的指导原则，可以为每个输入域数据对象设计测试用例并执行它。选择测试用例以便一次测试一个等价类的尽可能多的属性。

14.6.3 边界值分析

多数错误出现在输入域的边界处，而不是在其“中间”。这是将边界值分析（boundary value analysis, BVA）作为一种测试技术的原因。边界值分析选择一组测试用例检查边界值。



通过侧重考虑一个等价类“边界”处的数据，BVA扩展了等价类划分。



如何创建BVA
测试用例？

边界值分析对“等价划分”测试用例设计技术是一种补充。BVA不是选择等价类的任何元素，而是选择等价类“边缘”的测试用例。BVA不是仅仅侧重于输入条件，也从输出域中生成测试[MYE79]。

BVA的指导原则在很多方面与等价划分的原则相类似：

1. 若输入条件指定为以 a 和 b 为边界的范围，则测试用例应该包括 a 、 b 、略大于 a 和略小于 b ；
2. 若输入条件指定为一组值，则测试用例应当执行其中的最大值和最小值，以及略大于最小值和略小于最大值的值；
3. 指导原则1和2也适用于输出条件。例如，工程分析程序要求输出温度和压强的对照表，测试用例应该能够创建表项所允许的最大值和最小值的输出报告；
4. 若内部程序数据结构有预定义的边界值（如：数组有100项），要在其边界处测试数据结构。

大多数软件工程师会在某种程度上凭直觉完成BVA。通过运用这些指导原则，边界测试会更加完全，从而更有可能发现错误。

“Ariane 5型火箭在起飞时发生爆炸，仅仅是由于这样一个软件缺陷——将64位的浮点值转换为16位的整数值。这个火箭及其四个卫星未投保且价值5亿美元。综合系统测试本应该能够发现这个错误，但由于预算原因而被否决。”

——一篇新闻报道

438

14.6.4 正交数组测试

许多应用程序的输入域是相对有限的。也就是说，输入参数的数量不多，且每个参数可取的值有明显的界定。当这些数量非常小时（例如：三个输入参数，取值分别为三个离散值），则考虑每个输入排列并对所有的输入域进行测试是有可能的。然而，随着输入值数量的增加且每个数据项的离散值数量的增加，穷举测试是不切实际或不可能的。

正交数组测试（orthogonal array testing）可以应用于输入域相对较小但对穷举测试而言又过大的问题。正交数组测试方法对于发现区域错误（region fault）（有关软件构件内部错误逻辑的一类错误）尤其有效。



正交数组测试使
你能够设计测试
用例，它具有合
理的测试用例数，
且能提供最
大的测试覆盖。

为说明正交数组测试与更传统的“一次一个输入项”方法之间的区别，考虑有三个输入项X、Y和Z的系统。每个输入项有三个不同的离散值。这样可能有 $3^3 = 27$ 个测试用例。Phadke[PHA97]提出一种几何观点来组织与X、Y和Z相关的测试用例。如图14-9所示，一次一个输入项可能沿着每个输入轴在顺序上有变化。这导致相对有限的输入域覆盖率（图14-9中左图立方体所示）。

当使用正交数组测试时，创建测试用例的一个L9正交数组。L9正交数组具有“平衡特性[PHA97]”，即测试用例（图中表示为黑点）“均匀地分散在整个测试域中”，如图14-9中右图立方体所示。整个输入域的测试覆盖会更充分。

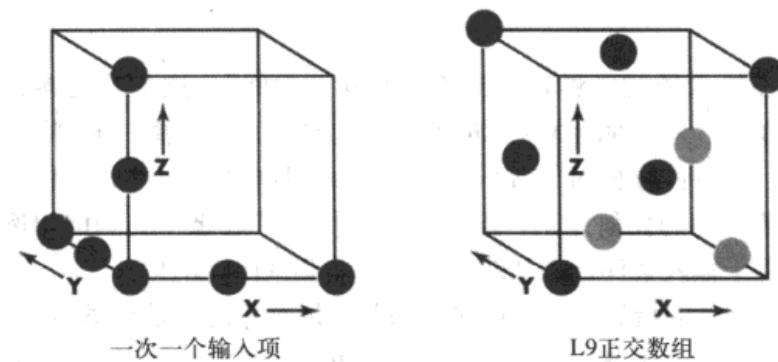


图14-9 测试用例的几何视图

439

为说明L9正交数组的使用，考虑传真应用系统的send函数。向函数send传递四个参数P1、P2、P3和P4。其中每个参数取三个不同的值。例如，P1的取值：

P1=1，现在发送

P1=2，一小时后发送

P1=3，半夜12点后发送

P2、P3和P4也分别取值为1、2和3，表示其他发送功能。

如果选择“一次一个输入项”的测试策略，测试(P1, P2, P3, P4)的测试序列如下：(1,1,1,1), (2,1,1,1), (3,1,1,1), (1,2,1,1), (1,3,1,1), (1,1,2,1), (1,1,3,1), (1,1,1,2), (1,1,1,3)。Phadke[PHA97]评估这些用例如下：

只有确信这些测试参数不相交时，上述的测试用例才是有用的。它们可以侦测出一个参数值使软件出现故障的逻辑错误。这种错误称之为单模式错误 (single mode fault)。这种方法不能查出两个或多个参数同时取某个值时使软件出现故障的错误，也就是说，它不能查出任何相互影响。因此，它的发现错误的能力是有限的。

给定相对少量的输入参数和离散值，穷举测试是可能的。所需要的测试数为 $3^4 = 81$ ，虽比较大，但仍是容易做到的。可以发现所有与数据项排列相关的错误，但所需的工作量相对较大。

正交数组测试方法使我们可以利用比穷举测试少得多的测试用例而提供好的测试覆盖。send函数的L9正交数组如图14-10所示。

440

Phadke[PHA97]对利用L9正交数组测试方法的测试结果评价如下：

检测和分离所有单模式错误。单模式错误是任意单个参数在任意级别上的一致性问题。例如，若因子P1=1的所有测试用例产生一个错误条件，则它是一个单模式错误。在这个例子中，测试1、2和3(图14-10)将显示错误。通过分析有关哪些测试显示错误的信息，可以识别哪个参数值产生错误。通过注释测试1、2和3产生错误，可以将其分离(有关“现在发送(P1=1)”的逻辑处理)为错误源。这

测试用例	测试参数			
	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

图14-10 send函数的L9正交数组

样的错误分离对错误的修改是很重要的。

检测所有双模式错误。若当两个参数的特定级别一起出现时存在一致性问题，则称之为双模式错误。实际上，双模式错误是成对不相容问题或两个测试参数间的有害干扰问题的指示。

多模式错误。正交数组（所显示类型的）仅可以保证单模式和双模式错误的检测。然而，许多多模式错误也可以通过这些测试检测出来。

正交数组测试的详细讨论见[PHA89]。

SOFTWARE TOOLS

测试用例设计

目的：辅助软件团队设计完整的黑盒测试和白盒测试用例集。

机制：这些工具可分为静态测试与动态测试两大类。在产业界，有三种不同类型的静态测试工具——基于代码的测试工具、测试专用语言和基于需求的测试工具。基于代码的测试工具所接收的输入为源代码，完成一系列分析，最后生成测试用例。测试专用语言（例如：ATLAS）使软件工程师能够写详细的测试用例规格说明，在规格说明中，描述每个测试用例及其执行逻辑。基于需求的测试工具将特定用户需求进行分离，对设计检查需求的测试用例（或测试类）提出建议。动态测试工具与执行的程序进行交互，检查路径覆盖，测试特定变量值的断言，或监测程序的执行流。

代表性工具⁶

McCabe Test，由McCabe & Associates (www.mccabe.com) 开发，它实现了一些从环复杂性评估和其他软件度量中派生的路径测试技术。

Panorama，由International Software Automation, Inc. (www.softwareautomation.com) 开发，它包含了一套完整的面向对象软件开发工具，包括辅助测试用例设计与测试计划的工具。

TestWorks，由Software Research, Inc. (www.soft.com/Products) 开发，它是一套完整的自动化测试工具，有助于C、C++和Java软件的测试用例的设计，并为回归测试提供支持。

T-Vec Test Generation System，由T-VEC Technologies (www.t-vec.com) 开发，它是支持单元测试、集成测试和确认测试的工具集。通过使用面向对象需求规格说明中的信息，辅助测试用例的设计。

441

14.7 面向对象测试方法

面向对象体系结构导致包括相互协作类的一系列分层子系统的产生。每个系统成分（子系统与类）完成有助于满足系统需求的功能。有必要在不同的层次上测试面向对象系统，以发现错误。在类相互协作以及子系统跨体系结构层次通信时可能出现这些错误。

在方法上，面向对象测试与传统系统测试相类似，但它们的测试战术是不同的。由于面向对象分析与设计模型在结构和内容上与所得的面向对象程序相类似，因此，“测试”可以起始于对这些模型的评审。一旦产生了代码，真正的面向对象测试以设计一系列检验类操作的“小型测试”以及当一个类与其他类进行协作时是否出现错误开始。当集成类形成一个子系统

⁶ 这里提到的工具并不表示本书支持这些工具，而只是该类工具的举例。多数情况下，工具的名称由各自的开发者注册为商标。

时，结合基于故障的方法，运用基于使用的测试对相互协作的类进行完全检查。最后，利用用例发现软件确认层的错误。

传统的测试用例是通过软件的输入—处理—输出视图或单个模块的算法细节来设计的。面向对象测试侧重于设计适当的操作序列以检查类的状态。

14.7.1 面向对象概念的测试用例设计的含义

WebRef

一些极好的有关面向对象测试的论文集和资源可在 www.rbsc.com 找到。

类经过分析模型到设计模型的演变，它成为测试用例设计的目标。由于操作和属性是封装的，从类的外面测试操作通常是徒劳的。尽管封装是面向对象的本质特征之一，但它可能成为测试的一个小障碍。如 Binder [BIN94]所述：“测试需要对象的具体和抽象状态”。然而，封装使获取这些信息有些困难，除非提供内置操作来报告类的属性值。

继承也为测试用例设计者提出了额外的挑战。我们已经注意到，即使已取得复用，每个新的使用环境也需要重新测试。另外，由于增加了所需测试环境的数量，多重继承⁷使测试进一步复杂化[BIN94]。

若从超类中派生的子类实例使用在相同的问题域，则当测试子类时，使用超类中生成的测试用例集是可能的。然而，若子类用在一个完全不同的环境中，则超类的测试用例将具有很小的可应用性，因而必须设计新的测试用例。

442

14.7.2 传统测试用例设计方法的可应用性

前面描述的白盒测试方法可以应用于类中定义的操作。基本路径、循环测试或数据流技术有助于确保测试一个操作的每条语句。然而，许多类操作的简洁结构使某些人认为：用于白盒测试的工作量最好直接用于类层次的测试。

与利用传统的软件工程方法所开发的系统一样，黑盒测试方法也适用于面向对象系统。如我们在本章的前面提到的，用例可为黑盒测试和基于状态的测试设计提供有用的输入。

14.7.3 基于故障的测试⁸

KEY POINT

基于故障测试的策略是假设一组似乎可能出现的故障，然后派生测试用例证明每个假设。

在操作调用和消息连接中遇到哪些类型的故障？

在面向对象系统中，基于故障测试的目标是设计最有可能发现似乎可能故障（以下称为似然故障）的测试。由于产品或系统必须符合用户需求，完成基于故障的测试所需的初步计划是从分析模型开始的。测试者查找似然故障（即系统的实现有可能产生错误的方面）。为确定这些故障是否存在，设计测试用例以检查设计或代码。

当然，这些技术的有效性依赖于测试人员如何感知似然故障。若在面向对象系统中真正的故障被感知为“没有道理”的，则这种方法实际上并不比任何随机测试技术好。然而，若分析和设计模型可以洞察有可能出错的事物，则基于故障的测试可以花费相当少的工作量而发现大量的错误。

集成测试（当应用于面向对象环境时）寻找操作调用或信息连接中的似然

⁷ 应极小心地使用的一个面向对象概念。

⁸ 14.7.3节到14.7.6节是从Brain Marick贴在因特网新闻组comp.testing上的文章中摘录的，在这里引用已得到作者的允许。有关该主题的详细信息见[MAR94]。应该注意到，14.7.3节到14.7.6节讨论的技术也适用于传统软件。

错误。在这种环境下，有三种错误可以发现：非预期的结果、错误的操作/消息使用，以及不正确的调用。为确定函数（操作）调用时的似然故障，必须检查操作的行为。

集成测试适用于属性与操作。对象的“行为”通过赋予属性值来定义。测试应该检查属性以确定不同类型的对象行为是否存在合适的值。

集成测试试图发现用户对象而不是服务对象中的错误，注意到这一点很重要。用传统的术语来说，集成测试的重点是确定调用代码而不是被调用代码中是否存在错误。利用操作调用为线索，是找出检查调用代码的测试需求的一种方式。

443

“如果你盼望一个程序能正常运行，很可能你也看到它在正常运行，这时你会忽视它可能出现的故障。”

——Cem Kaner等

14.7.4 测试用例与类层次

继承并不能排除对所有派生类进行全面测试的需要。事实上，它确实使测试过程更复杂。考虑下列情形，类Base包含了操作inherited()和redefined()，类Derived重定义redefined()以用于某个局部环境中。毫无疑问，Derived::redefined()必须被测试，因为它表示一个新的设计和新的代码。但是，Derived::inherited()需要重新测试吗？



若Derived::inherited()调用redefined()，而redefined()的行为已经发生变化，Derived::inherited()可能具有新的行为，因此，它需要新的测试，尽管设计与代码没有发生变化。然而，重要的是要注意，只有Derived::inherited()的所有测试的一个子集需要执行。若inherited()的部分设计和代码与redefined()无关（即，不调用它，也不间接调用它），则在派生类中的代码不需要重新测试。

Base::redefined()和Derived::redefined()是具有不同规格说明和实现的两个操作。它们各自有一组从其规格说明和实现中生成的测试需求。那些测试需求探查似然故障：集成故障、条件故障、边界故障，等等。但操作有可能是类似的，它们的测试需求集将重迭。面向对象设计得越好，重迭就越多。新的测试仅需要从不被Base::redefined()测试满足的Derived::redefined()的需求中生成。

总而言之，Base::redefined()测试可应用于类Derived的对象。测试输入可能同时满足基类和派生类。但是，在派生类中预期的结果可能有所不同。

14.7.5 基于场景的测试

基于故障测试忽略了两种主要错误类型：(1) 不正确的规格说明；(2) 子系统间的交互。当出现了与不正确的规格说明相关的错误时，产品并不做客户希望的事情，它有可能做错误的事情或漏掉重要的功能。但是，在这两种情况下，质量（对需求的符合性）均受到损害。当一个子系统的操作创建的环境（如：事件、数据流）使另一个系统失效时，则与子系统交互相关的错误出现。



基于场景的测试关心用户做什么，而不是产品做什么。这意味着捕获用户必须完成的任务（通过用例），然后在测试时使用它们及其变体。

场景可以发现交互错误。为了达到这个目标，测试用例必须比基于故障的测试更复杂且更切合实际。基于场景的测试倾向于用单一测试检查多个子系统（用户并不限制自己一次只用一个子系统）。

444

以设计文本编辑器的基于场景的测试为例，用例的非形式化描述如下：

用例：修改最终草稿

背景：人们经常碰到以下情景：打印“最终”草稿，阅读它，却发现了一些屏幕上不易察觉的恼人的错误。该用例描述了当这种情况发生时出现的事件序列：

1. 打印整个文档；
2. 在文档中移动，修改某些页；
3. 当修改各页时，打印它；
4. 有时打印多页。

这个场景描述了两件事情：一个测试和特定的用户需要。用户的需要是明显的：(1) 打印单页的方法；(2) 打印第 m 页到第 n 页的方法。当进行测试时，需要在打印后测试编辑（以及编辑后测试打印）。测试人员希望发现打印功能使编辑功能产生错误。也就是说，两个软件不是完全独立的。



尽管基于场景的
测试有优点，但
是，当用例作为
分析模型的一部
分时，花时间对
其进行评审将可
获得较高的回报。

用例：打印一个新副本

背景：某人向用户要文档的一个新副本，必须打印它。

1. 打开文档；
2. 打印文档；
3. 关闭文档。

另外，这种测试方法是相当明显的，除非文档突然消失。它是在一个早
期的任务中创建的，那个任务对这个任务有影响吗？

在现代很多编辑器中，文档记住最后一次打印时的状况。缺省情况下，下一次用相同的方式打印。在“修改最终草稿”场景之后，只需要选择菜单中的“打印”，点击对话框的“打印”按钮就可以对上次的正确页面再打印一次。这样，根据编辑器，正确的场景应该如下：

用例：打印一个新副本

1. 打开文档；
2. 在菜单中选择“打印”；
3. 检查是否将连续打印若干页，如果是，点击以打印整个文档；
4. 按“打印”按钮；
5. 关闭文档。

445

但是，这个场景指明了一个潜在的规格说明错误。编辑器没有做用户合理期望它做的事情。用户经常忽略了对上面第3步的检查。当他们走到打印机前，发现只有一页，而他们要100页，这使他们感到恼火。生气的用户报告规格说明错误。

测试用例设计者可能忽略了测试用例设计中的这种依赖性，但是，测试期间这种问题有可能出现。测试人员则必须应付可能的反应：“这就是它应该工作的方式！”

14.7.6 表层结构和深层结构的测试

表层结构是指面向对象程序的外部可观察结构，即，对最终用户是显而易见的结构。许多面向对象系统用户可能不是完成某个功能，而是得到以某种方式操纵的对象。但是，无论接口是什么，测试仍然是基于用户任务进行的。捕捉这些任务涉及理解、观察以及与有代表性的用户（很多非代表性的用户也值得考虑）进行交谈。



细节上确实存在某些差异，例如，在使用面向命令接口的传统系统中，用户可以使用所有命令列表作为测试检查表。若没有测试场景检查某个命令，测试有可能忽略某些用户任务（或具有无用命令的界面）。在基于对象的接口中，测试人员可以使用所有的对象列表作为测试检查表。

当设计人员以一种新的或非传统的方式看待系统时，则可以设计出最好的测试。例如，若系统或产品有基于命令行的接口，测试用例设计人员假设操作是与对象无关的，则可以设计更彻底的测试。问一些这样的问题：“当用打印机时，用户希望使用这个操作（仅用于“扫描仪”对象）吗？”不管界面风格怎样，检查表层结构的测试用例设计应该使用对象和操作为线索导向被忽视的任务。

深层结构指面向对象程序的内部技术细节，即通过检查设计和（或）代码来理解的数据结构。设计深层结构测试来检查面向对象软件设计模型中（第9章至第12章）的依赖关系、行为和通信机制。

分析模型和设计模型用作深层结构测试的基础。例如，UML协作图或分布模型描述了对象和子系统间对外不可见的协作关系。那么测试用例设计者会问：我们已经捕获了某些在协作表中记录的协作任务吗？若没有？为什么没有？

“不要为错误感到羞愧，从而使错误变成遗憾。”

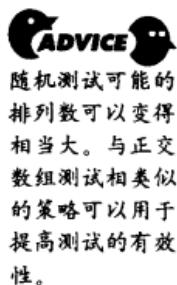
——Confucius

446

14.8 类级可应用的测试方法

在第13章，我们提到软件测试从“小型”测试开始，慢慢进展到“大型”测试。“小型”测试侧重于单个类及该类封装的方法。面向对象测试期间，随机测试和分割是可以用于检查类的方法[KIR94]。

14.8.1 面向对象的随机测试



为提供这些方法的简要说明，考虑一个银行应用，其中Account类有下列操作：open()、setup()、deposit()、withdraw()、balance()、summarize()、creditLimit()和close()[KIR94]。其中，每个操作均可应用于Account类，但问题的本质隐含了一些限制（例如，账号必须在其他操作可应用之前打开，在所有操作完成之后关闭）。即使有了这些限制，仍存在很多种操作排列。一个Account对象的最小行为的生命历史包含以下操作：

open * setup * deposit * withdraw * close

这表示Account的最小测试序列。然而，大量其他行为可以在这个序列中发生：

open * setup * deposit * [deposit | withdraw | balance | summarize | creditLimit] * withdraw * close

一些不同的操作序列可以随机产生，例如：

测试用例 r_1 : open * setup * deposit * deposit * balance * summarize * withdraw * close

测试用例 r_2 : open * setup * deposit * withdraw * deposit * balance * creditLimit * withdraw * close

执行这些序列和其他随机顺序测试以检查不同的类实例的生命历史。

类测试

[场景] Shakira的工作间。

[人物] Jamie与Shakira, SafeHome软件工程团队成员, 为安全功能设计测试用例。

[对话]

Shakira: 我已经为类Detector (图11-4) 开发了一些测试, 你知道, 这个类允许访问安全功能的所有Sensor对象。你对它熟悉吗?

Jamie (笑): 当然, 它允许你加“小狗焦虑症”传感器。

Shakira: 而且是唯一的一个。不管怎么样, 它包含四个操作的接口: `read()`、`enable()`、`disable()`和`test()`。在传感器可读之前, 它必须被激活。一旦激活, 可以读和测试它, 且随时可以中止它, 除非正在处理警报条件。因此, 我定义了检查其行为生命历史的简单测试序列。(向Jamie展示下述序列。)

#1: `enable•test•read•disable`

Jamie: 不错。但你还得做更多的测试!

Shakira: 我知道。这里有我提出的其他测试序列。

(她向Shakira展示下述序列)

#2: `enable•test•[read]n•test•disable`

#3: `[read]n`

#4: `enable•disable•[test | read]`

Jamie: 看我能否理解这些测试序列的意图。#1通过一个正常的生命历史, 属于常规使用。#2重复`read()`操作 n 次, 那是一个可能出现的场景。#3在传感器激活之前尽力读取它……那应该产生某种错误信息, 对吗? #4激活和中止传感器, 然后尽力读取它, 这与#3不是一样的吗?

Shakira: 实际上不一样。在#4中, 传感器已经被激活, #4测试的实际上是`disable()`操作是否像其预期的一样有效工作。`disable()`之后, `read()`或`test()`应该产生错误信息。若没有, 则说明`disable()`操作中有错误。

Jamie: 太棒了, 记住这四个测试必须应用于每一类传感器, 因为所有这些操作根据其传感器类型可能略有不同。

Shakira: 不用担心, 那是计划之中的事。

14.8.2 类级的划分测试

与传统软件的等价划分 (14.6.2) 基本相似, 划分测试 (partition testing) 减小测试特定类所需的测试用例数量。对输入和输出进行分类, 设计测试用例以检查每个类。但划分类别是如何得到的呢?

在类层次上,
什么测试选
项可供使用?

基于状态划分就是根据它们改变类状态的能力对类操作进行分类。再考虑**Account**类, 状态操作包括`deposit()`和`withdraw()`, 而非状态操作包括`balance()`、`summarize()`和`creditLimit()`。将改变状态的操作和不改变状态的操作分开, 分别进行测试, 因此:

测试用例 p_1 : open * setup * deposit * deposit * withdraw * withdraw * close

测试用例 p_2 : open * setup * deposit * summarize * creditLimit * withdraw * close

测试用例 p_1 检查改变状态的操作，而测试用例 p_2 检查不改变状态的操作（除了那些最小测试序列中的操作）。

基于属性划分就是根据它们所使用的属性进行划分类操作。对于类Account，属性balance和creditLimit可用于定义划分。操作可分为三类：(1) 使用creditLimit的操作；(2) 修改creditLimit的操作；(3) 既不使用也不修改creditLimit的操作。然后为每个划分设计测试用例。

基于类别划分就是根据每个操作所完成的一般功能进行划分类操作。例如，在类Account中，操作可分为初始化操作——open()、setup()，计算操作——deposit()、withdraw()，查询操作——balance()、summarize()、creditLimit()，以及终止操作——close()。

448

14.9 类间测试用例设计

当开始集成面向对象系统时，测试用例的设计变得更为复杂。在这个阶段必须开始类间协作的测试。为说明“类间测试用例生成”[KIR94]，我们扩展14.8节中讨论的银行例子，让它包括图14-11中的类与协作。图中箭头的方向指明消息传递的方向，标注则指明作为消息隐含的协作的结果而调用的操作。

与单个类的测试相类似，类协作测试可以通过运用随机和划分方法、基于场景测试以及行为测试来完成。

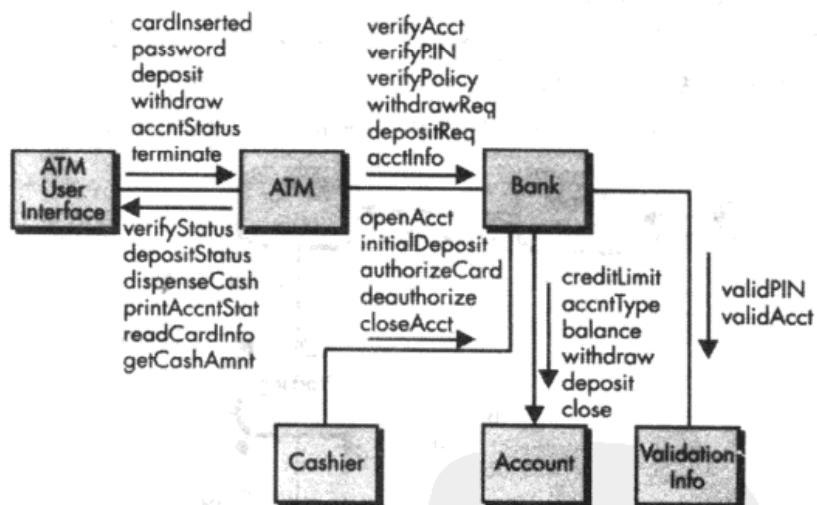


图14-11 银行应用的类协作图（摘自[KIR94]）

14.9.1 多类测试

Kirani和Tsai[KIR94]提出了利用下列步骤生成多类随机测试用例的方法：

- 对每个用户类，使用类操作列表来生成一系列的随机测试序列。这些操作将向其他服务类发送消息；
- 对生成的每个消息，确定协作类和服务对象中的相应操作；
- 对服务对象中的每个操作（已被用户对象发送的消息调用），确定它所发送的消息；
- 对每个消息，确定下一层被调用的操作并将其引入到测试序列中。

449

为便于说明[KIR94]，考虑类Bank相对于类ATM的操作序列（图14-11）：

$\text{verifyAcct} \cdot \text{verifyPIN} \cdot [[\text{verifyPolicy} \cdot \text{withdrawReq}] \mid \text{depositReq} \mid \text{acctInfoREQ}]^n$

类Bank的一个随机测试用例可以是：

测试用例 $r_3 = \text{verifyAcct} \cdot \text{verifyPIN} \cdot \text{depositReq}$

为考虑涉及该测试的协作者，考虑与测试用例 r_3 中提到的操作相关的消息。类Bank必须与类ValidationInfo协作以执行verifyAcct()与verifyPIN()，类Bank必须与类Account协作以执行depositReq()，因此，检查这些协作的新测试用例为：

测试用例 $r_4 = \text{verifyAcctBank}[\text{validAcctValidationInfo}] \cdot \text{verifyPINBank} \cdot$

$[\text{validPinValidationInfo}] \cdot \text{depositReq} \cdot [\text{depositaccount}]$

多个类的划分测试方法与单个类的划分测试方法（14.8.2节）类似。然而，可以扩展这个测试序列以包括那些为发送给协作类的消息而激活的操作。另一种划分测试方法是基于特殊类的接口的。如图14-11所示，类Bank从类ATM和类Cashier中接收消息，因此，类Bank中的操作可以通过划分为服务于类ATM的操作和服务于类Cashier的操作来测试。基于状态划分（14.8.2节）可用于进一步细化上述划分。

14.9.2 从行为模型中导出的测试

在第8章中，我们已讨论过用状态图表示类的动态行为模型。类的状态图可用于辅助生成检查类（以及与该类的协作类）的动态行为的测试序列。图14-12[KIR94]给出了前面讨论的类Account的状态图。根据该图，初始变换经过了“Empty acct”状态和“Setup acct”状态，该类实例的绝大多数行为发生在“Working acct”状态。最终的Withdrawal和结束账户操作使得类Account分别向“Nonworking acct”状态和“Dead acct”状态发生变换。

450

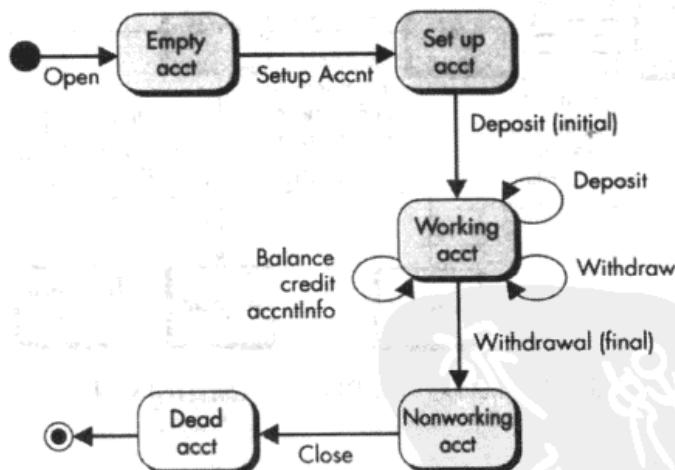


图14-12 类Account的状态变换图（摘自[KIR94]）

将要设计的测试应该覆盖所有的状态[KIR94]，也就是说，操作序列应该导致类Account的变换穿越所有可允许的状态。

测试用例 $S_1: \text{open} \cdot \text{setupAcct} \cdot \text{deposit (initial)} \cdot \text{withdraw (final)} \cdot \text{close}$

应该注意到，这个序列与14.9.1节所讨论的最小测试序列相同。将其他测试序列加入最小测试序列中：

测试用例 S_2 : open • setupAcnt • deposit(initial) • deposit • balance • credit • withdraw
(final) • close

测试用例 S_3 : open • setupAcnt • deposit(initial) • deposit • withdraw • acntInfo • withdraw
(final) • close

可以设计更多的测试用例以保证该类的所有行为已被充分检查。在该类的行为导致与一个或多个类协作的情况下，用多个状态图来追踪系统的行为流。

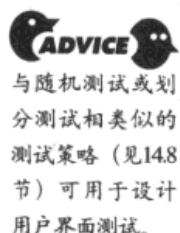
可以通过“广度优先”的方式来遍历状态模型[MGR94]。在这里，广度优先意味着一个测试用例检查单个变换，且测试新的变换时，仅使用前面已经测试过的变换。

考虑银行系统中的一个CreditCard对象。CreditCard的初始状态为undefined（即未提供信用卡号）。在销售过程中一读到信用卡号，对象进入defined状态，即属性card number、expiration date以及银行专用的标识符被定义。当信用卡被发送以请求授权时，它被提交(submitted)，当授权被接收，信用卡被核准(approved)。可以通过设计使变换发生的测试用例来测试CreditCard从一个状态到另一个状态的变换。对这种类型测试的广度优先方法在检查undefined和defined之前，不会检查submitted状态。若这样做了，它就使用了尚未经过测试的变换，从而违反了广度优先准则。451

14.10 针对特定环境、体系结构和应用系统的测试

前面几节所讨论的测试方法普遍可应用于所有环境、体系结构和应用系统，但是，有时还需要专门的测试指导原则和方法。在本节中，我们讨论软件工程师常见的特定环境、体系结构和应用系统的测试指导原则。

14.10.1 图形用户界面测试



图形用户界面（GUI）为软件工程师提出了有趣的挑战。由于提供可复用的构件作为GUI开发环境的一部分，用户界面的开发已变得更加省时且更精确（第12章）。但是，与此同时，GUI的复杂性也提高了，从而使设计和执行测试用例变得更加困难。

由于现代许多GUI具有相同的观感，因此可以设计一系列的标准测试。有限状态建模图可以生成一系列测试，这些测试针对与GUI有关的特定数据和程序对象。

由于存在大量与GUI操作相关的排列，测试应该使用自动化测试工具进行。在过去几年里，已有大量的GUI测试工具出现在市场上。进一步的讨论见第12章。

14.10.2 客户/服务器体系结构测试



客户/服务器（C/S）体系结构向软件测试人员提出了巨大的挑战。客户/服务器环境的分布特征、与事务处理相关的性能问题、不同硬件平台存在的可能性、网络通信的复杂性、中心数据库（或分布式）服务多个客户的需要以及服务器的协调需求，这些问题结合在一起使得C/S体系结构比独立应用的测试要困难得多。事实上，最近的研究成果表明，开发C/S环境显著增加了测试时间和成本。

452

“测试的主题是传统系统与C/S系统之间存在大量共同点的区域。” —— Kelley Bourne

一般来说，C/S软件测试发生在三个不同的层次上：(1)以“断开”模式测试单个客户应用，不考虑服务器的操作和支撑的网络；(2)以协同方式测试客户软件和有关服务器应用，但未明显检查网络操作；(3)测试整个C/S体系结构，包括网络操作和性能的测试。

尽管在每个层次上执行不同类型的测试，但对于C/S应用，常用的测试方法如下：



C/S系统执行什么类型的测试？

应用功能测试。利用本章前面讨论的方法测试客户应用系统的功能。本质上，以独立的形式测试应用系统。

服务器测试。测试服务器的协作和数据管理功能。也考虑服务器性能(总的反应时间和数据吞吐率)。

数据库测试。测试由服务器存储的数据的精确性和完整性。检查客户应用系统委派的事务以保证数据适当地存储、更新和恢复，同时也测试归档。

事务测试。创建一系列的测试以保证每类事务依据需求处理。测试侧重于处理的正确性及性能问题(例如，事务处理时间和事务量)。

网络通信测试。这些测试验证网络节点间的通信是正确的，消息传递、事务以及有关网络通信没有错误。网络安全测试也可以作为这些测试的一部分。

为完成上述测试，Musa[MUS 93]建议开发由C/S使用场景生成的运行剖面(operational profiles)⁹。运行剖面显示不同类型的客户如何与C/S系统进行互操作，即，当设计和执行测试时，运行剖面提供可应用的“使用模式”。

14.10.3 测试文档和帮助设施

453

软件测试一词造成一种假象：大量的测试用例是为检查计算机程序和它们所管理的数据做准备的。回忆本书第1章描述的软件定义，应该注意到测试应该扩充到软件配置的第三个元素——文档。

文档中的错误与数据或源代码中的错误一样，非常影响程序的验收。没有什么比完全按照用户指南但得到的结果却与文档描述不符更让人生气的了。因此，文档测试也是每个软件测试计划中有意义的一部分。

文档测试可分为两个阶段进行。第一阶段为评审和审查(第26章)，检查文档编辑的清晰性；第二阶段是现场测试(live testing)，结合实际程序的使用而使用文档。

文档测试

INFO

在测试文档和(或)帮助设施时，应该回答下列问题：

- 该文档精确地描述了如何完成每种使用模式吗？
- 每种交互序列的描述是精确的吗？
- 实例精确吗？
- 术语、菜单描述以及系统的响应与实际程序相一致吗？
- 在文档中找到指导比较容易吗？

⁹ 应该注意到，运行剖面可用于所有类型的系统结构的测试中，而不仅仅是C/S结构。

- 利用该文档可以容易地完成疑难解答吗？
- 该文档的目录和索引是精确和完整的吗？
- 该文档的设计（布局、字体、缩进、图表）有助于信息的理解与快速吸收吗？
- 所有显示给用户的软件错误信息在该文档中有更详细的描述吗？对看到错误信息后所采取的行动有明确的描述吗？
- 如果提供超级文本链接，链接是精确和完整的吗？
- 如果提供超级文本链接，导航设计适合获取信息吗？

回答这些问题唯一可行的方法是让独立的第三方（如：选定的用户）在程序使用的环境下测试该文档。应该记录所有的差异，确定模糊或薄弱的地方以方便可能的重写。

14.10.4 实时系统的测试

许多实时应用系统的时间依赖性和异步特征给测试带来了新的困难——时间。测试用例设计者不仅必须考虑传统的测试用例，而且要考虑事件处理（如：中断处理）、数据的定时、处理数据任务（即，进程）的并行。在许多情况下，实时系统在一个状态下提供的测试数据导致正常处理，而在另一个状态下提供同样的数据将会出现错误。

例如，控制复印机的实时软件在机器正在复印时（copying状态）接收操作员中断（即，机器操作员按控制键如RESET或DARKEN）不会产生错误。若同一操作员中断出现在机器处于jammed状态时，显示的诊断代码（指明卡纸的位置）将丢失（一个错误）。

此外，实时系统的软件和硬件环境之间的密切关系也会导致测试问题。软件测试必须考虑硬件故障对软件处理的影响。这种故障是很难实时仿真的。

实时系统的综合测试用例设计方法有待进一步发展。然而，可以提出以下的四步策略：



实时系统的
有效测试策
略是什么？

任务测试。实时软件测试的第一步是单独测试每个任务。也就是说，对每个任务设计并执行传统的测试。在测试期间，每个任务单独执行。任务测试可以发现逻辑和功能错误，但不能发现时间或行为错误。

行为测试。利用通过自动化工具创建的系统模型，是可以模拟实时系统的行为并按照外部事件序列检查其行为的。这些分析活动可以作为测试用例设计的基础。当实时软件建成时，执行这些测试用例。

任务间测试。一旦单个任务和系统行为中的错误已经分离出来，测试就要转向与时间相关的错误。用不同的数据率和处理负载来测试任务间的异步通信，以确定任务间是否发生同步错误。另外，通过消息队列和数据存储进行通信用任务的测试以发现这些数据存储区域大小方面的错误。

系统测试。集成软件与硬件并进行全范围的系统测试（第13章）以发现软件/硬件接口处的错误。多数实时系统处理中断，因此，测试布尔事件的处理尤其重要。利用状态图和控制规格说明（第8章），测试人员开发所有可能的中断和中断处理列表，然后设计测试以评估下列系统特征：

- 是否正确赋予和处理中断优先级？
- 每个中断的处理是否正确？
- 中断处理过程的性能（如：处理时间）是否符合需求？
- 关键时刻有大量中断，是否会导致功能和性能上的问题？

另外，作为中断处理一部分用来传输信息的全局数据区域也应该测试，以评估潜在的副作用。

454

455

14.11 测试模式

WebRef

70多个测试模式
可在www.rbsc.com找到。

KEY POINT

测试模式有助于软件团队对测试进行更有效的交流以及对导致特定测试方法的动机有一个较好的理解。

在前面几章我们已经讨论过，模式可以作为一种机制描述软件基本构造块（building block）或软件工程情况（situations）。当构建不同的应用系统或进行不同的项目时，这些基本构造块或情况会重复出现。像它们在分析和设计中的对应物一样，测试模式描述经常出现的基本构造块或情况。当软件测试人员测试新系统或修正后的系统时，能够复用这些基本构造块或情况。

测试模式不仅在测试活动开始时为软件工程师提供有用指导，而且也提供另外三个益处，Marick[MAR02]描述这三个益处如下：

1. 测试模式为问题求解提供一个词汇表。“嘿，你知道，我们应该用Null对象。”
2. 测试模式注意的重点是隐藏在问题之后的力量。这允许（测试用例）设计人员对什么时候和为什么运用一种解决方案提供较好的理解。
3. 测试模式有助于采用迭代的方式进行思考。每个解决方案创建一个可以解决新问题的新环境。

尽管这些益处是“软的”，但不应该忽视它们。甚至在过去10年间，大多数软件测试已是一项专门的活动。如果测试模式有助于软件测试团队对软件测试进行更有效的交流、理解采用特定测试方法的动机，以及将测试用例的设计处理为一种发展的活动，则测试模式已经达到了预期的目的。

测试模式可以采用与分析模式和设计模式（第7章和第9章）同样的方式描述。文献（例如：[BIN99]、[MAR02]）中已提出了几十种测试模式。下面的三种模式（仅以摘要的形式给出）提供了有代表性的例子：

WebRef

描述测试组织、效力、策略以及问题求解的模式
可在www.agcs.com/supportv2/techpapers/patterns/papers/systestp.htm中找到。

456

模式名：成对测试

摘要：一种面向过程的模式，成对测试描述了一种与成对编程（第4章）相类似的技术。在这种测试模式中，两个测试人员一起设计并执行一系列可以应用于单元测试、集成测试或确认测试的活动。

模式名：独立测试接口

摘要：在面向对象系统中需要对每个类进行测试，包括“内部类”（即，不向使用它们的外部成分暴露任何接口的类）。独立测试接口模式描述如何创建“一个测试接口，该测试接口可用于描述一些类（这些类仅对某个内部成分是可见的）的特定测试”[LAN01]。

模式名：场景测试

摘要：一旦已经执行了单元测试与集成测试，需要确定软件是否以让用户满意的方式执行。场景测试描述一种从用户的角度运行软件的技术。在这个层次上的失败表明软件不能满足用户的可见需求[KAN01]。

对测试模式的全面讨论超出了本书的范围。关于这个重要主题，有兴趣的作者应该看[BIN99]和[MAR02]。

14.12 小结

测试用例设计的主要目标是设计有可能发现软件错误的测试用例集。为达到这个目标，可采用两个不同的测试用例设计技术：白盒测试和黑盒测试，这些技术也可应用于传统的软件系统和面向对象系统。

白盒测试侧重于程序控制结构。设计测试用例以保证测试期间程序中所有的语句至少被执行一次，且所有的逻辑条件被检查。基本路径测试是一种白盒测试技术，利用程序图（或图矩阵）生成保证覆盖率的线性无关的测试集。条件和数据流测试进一步检查程序逻辑，循环测试补充其他的白盒测试技术，检查不同复杂度的循环。

黑盒测试用来确认功能需求，而不考虑程序的内部结构。黑盒测试技术侧重于软件的信息域，通过划分程序的输入域和输出域来设计测试用例以提供完全的测试覆盖。等价划分将输入域划分为有可能检查软件特定功能的数据类。边界值分析则检查程序处理边界数据的能力。正交数组测试提供了一种高效的、系统的、使用少量的输入参数的测试方法。

尽管面向对象测试的总体目标（利用最少的工作量找到最大数量的错误）与传统软件的测试目标是相同的。但是，面向对象软件测试策略和战术稍微有些不同。测试的视角放宽以包括分析模型和设计模型的评审。另外，测试的重点由子程序构件（模块）转向类。类的测试设计使用多种方法：基于故障测试、随机测试和划分测试。其中每种方法检查类中封装的操作。设计测试序列以保证检查有关操作。通过检查类的状态（由属性值表示）以确定是否存在错误。[457]

集成测试可以通过基于使用的策略来完成。基于使用策略从那些不使用服务类的类开始，以分层的方式构造系统。集成测试用例设计方法也使用随机测试和划分测试。另外，基于场景和从行为模型中生成的测试可用于测试类及其协作关系。测试序列追踪类间协作的操作流。

特定测试方法包括广泛的软件能力和应用领域。图形用户界面、客户/服务器体系结构、文档与帮助设施以及实时系统，每种都需要特定的测试指导和测试技术。

有经验的软件人员经常说：“测试永不停止，它只不过是从软件工程师转移到用户。客户每次使用程序，都是一次测试。”通过运用测试用例设计，软件工程师可以取得更完全的测试，由此在客户的测试开始之前，发现和校正最大可能数量的错误。

参考文献

- [AMB95] Ambler, S., "Using Use Cases," *Software Development*, July 1995, pp. 53–61.
- [BEI90] Beizer, B., *Software Testing Techniques*, 2nd ed., Van Nostrand-Reinhold, 1990.
- [BEI95] Beizer, B., *Black-Box Testing*, Wiley, 1995.
- [BIN94] Binder, R. V., "Testing Object-Oriented Systems: A Status Report," *American Programmer*, vol. 7, no. 4, April 1994, pp. 23–28.
- [BIN99] Binder, R., *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley, 1999.
- [DEU79] Deutsch, M., "Verification and Validation," in *Software Engineering* (R. Jensen and C. Tonies, eds.), Prentice-Hall, 1979, pp. 329–408.
- [FRA88] Frankl, P. G., and E. J. Weyuker, "An Applicable Family of Data Flow Testing Criteria," *IEEE Trans. Software Engineering*, vol. SE-14, no. 10, October 1988, pp. 1483–1498.
- [FRA93] Frankl, P. G., and S. Weiss, "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow," *IEEE Trans. Software Engineering*, vol. SE-19, no. 8, August 1993, pp. 770–787.
- [KAN93] Kaner, C., J. Falk, and H. Q. Nguyen, *Testing Computer Software*, 2nd ed., Van Nostrand-

Reinhold, 1993.

[KAN01] Kaner, C., "Pattern: Scenario Testing," (draft), 2001, available at <http://www.testing.com/test-patterns/patterns/pattern-scenario-testing-kaner.html>.

[KIR94] Kirani, S., and W. T. Tsai, "Specification and Verification of Object-Oriented Programs," Technical Report TR 94-64, Computer Science Department, University of Minnesota, December 1994.

[LAN01] Lange, M., "It's Testing Time! Patterns for Testing Software, June, 2001, downloadable from <http://www.testing.com/test-patterns/patterns/index.html>.

[LIN94] Lindland, O. I., et al., "Understanding Quality in Conceptual Modeling," *IEEE Software*, vol. 11, no 4, July 1994, pp. 42-49.

[MAR94] Marick, B., *The Craft of Software Testing*, Prentice-Hall, 1994.

[MAR02] Marick, B., "Software Testing Patterns," 2002, <http://www.testing.com/test-patterns/index.html>.

[MCC76] McCabe, T., "A Software Complexity Measure," *IEEE Trans. Software Engineering*, vol. SE-2, December 1976, pp. 308-320.

[MGR94] McGregor, J. D., and T. D. Korson, "Integrated Object-Oriented Testing and Development Processes," *CACM*, vol. 37, no. 9, September 1994, pp. 59-77.

[MUS93] Musa, J., "Operational Profiles in Software Reliability Engineering," *IEEE Software*, March 1993, pp. 14-32.

[MYE79] Myers, G., *The Art of Software Testing*, Wiley, 1979.

[NTA88] Ntafos, S. C., "A Comparison of Some Structural Testing Strategies," *IEEE Trans. Software Engineering*, vol. SE-14, no. 6, June 1988, pp. 868-874.

[PHA89] Phadke, M. S., *Quality Engineering Using Robust Design*, Prentice-Hall, 1989.

[PHA97] Phadke, M. S., "Planning Efficient Software Tests," *Crosstalk*, vol. 10, no. 10, October 1997, pp. 11-15.

[TAI89] Tai, K. C., "What to Do Beyond Branch Testing," *ACM Software Engineering Notes*, vol. 14, no. 2, April 1989, pp. 58-61.

458

习题与思考题

- 14.1 说明、设计和实现一个软件工具，使其能够对你所选的程序设计语言计算环复杂性。在你的设计中，利用图矩阵作为有效的数据结构。
- 14.2 给出至少三个例子，在这些例子中，黑盒测试能够给人“一切正常”的印象，而白盒测试可能发现错误。再给出至少三个例子，在这些例子中白盒测试可能给人“一切正常”的印象，而黑盒测试可能发现错误。
- 14.3 阅读Beizer[BEI95]并确定在习题14.1中开发的程序如何扩展以适应各种连接权值。扩展你的工具以处理执行概率或连接处理时间。
- 14.4 选择一个你最近设计和实现的构件。设计一组测试用例，保证利用基本路径测试所有的语句都被执行。
- 14.5 若现有类已进行了彻底的测试，为什么我们必须对从现有类中实例化的子类进行重新测试？我们可以使用为现有类设计的测试用例吗？
- 14.6 你可以想到除14.1节讨论之外的测试特征吗？
- 14.7 设计一个自动化测试工具，使其能够识别循环并按照14.5.3节中的方法分类。
- 14.8 Myers[MYE79]用以下的程序作为对测试能力的自我评估：某程序读入三个整数值，这三个整数值表示三角形的三条边。该程序打印信息表明三角形是不规则的、等腰的或等边的。开发一组测试用例测试该程序。
- 14.9 设计并实现习题14.8描述的程序（适当时使用异常处理）。从该程序中导出流图并用基本路径测试方法设计测试以保证所有的语句都被执行。执行测试用例并显示结果。

- 14.10 穷举测试（即便对非常小的程序）是否能够保证程序100%正确？
- 14.11 用自己的话，描述为什么在面向对象系统中，类是最小的合理的测试单元。
- 14.12 扩展习题14.7中描述的工具，为曾经遇到的每个循环类生成测试用例。与其他测试人员交互地完成这个功能是有必要的。
- 14.13 将随机测试和划分方法运用到设计SafeHome系统时定义的三个类。产生展示操作调用序列的测试用例。
- 14.14 运用多类测试及从SafeHome设计的行为模型中生成的测试。
- 14.15 测试你经常使用的某个应用的用户手册（或帮助设施）。在文档中找到至少一个错误。

推荐读物与阅读信息

在许多介绍测试用例设计方法的书中，Craig和Kaskiel（《Systematic Software Testing》，Artech House, 2002）、Tamres（《Introducing Software Testing》，Addison-Wesley, 2002）、Whittaker（《How to Break Software》，Addison-Wesley, 2002）、Jorgensen（《Software Testing: A Craftman's Approach》，CRC Press, 2002）、Splaine和他的同事（《The Web Testing Handbook》，Software Quality Engineering Publishing, 2001）、Patton（《Software Testing》，Sams Publishing, 2000）以及Kaner和他的同事（《Testing Computer Software》，second edition, Wiley, 1999）都是不错的。另外，Hutcheson（《Software Testing Methods and Metrics: The Most Important Tests》，McGraw-Hill, 1997）和Marick（《The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing》，Prentice-Hall, 1995）对测试方法和策略进行了讨论。

Myers[MYE79]留下了经典的文稿，非常详细地覆盖了黑盒技术。Beizer[BEI90]全面覆盖了白盒技术，引入了在其他测试方法中通常被忽略的数学精确标准。他后来出版的书[BEI95]介绍了重要方法的简明处理。Perry（《Effective Methods for Software Testing》，Wiley-QED, 1995）、Friedman和Voas（《Software Assessment: Reliability, Safety, Testability》，Wiley, 1995）对测试策略和战术进行了很好的介绍。Mosley（《The Handbook of MIS Application Software Testing》，Prentice-Hall, 1993）讨论了大型信息系统的测试问题，Marks（《Testing Very Big Systems》，McGraw-Hill, 1992）讨论了测试主要的程序设计系统时一定要考虑的特殊问题。

Sykes和McGregor（《Practical Guide for Testing Object-Oriented Software》，Addison-Wesley, 2001）、Bashir和Goel（《Testing Object-Oriented Software》，Springer-Verlag, 2000）、Binder（《Testing Object-Oriented Systems》，Addison-Wesley, 1999）、Kung和他的同事（《Testing Object-Oriented Software》，IEEE Computer Society Press, 1998）、Marick（《The Craft of Software Testing》，Prentice-Hall, 1997）以及Siegel与Muller（《Object-Oriented Software Testing: A Hierarchical Approach》，Wiley, 1996）描述了测试面向对象系统的策略和方法。

软件测试是一种资源密集型的活动。为此，许多组织为部分测试过程提供了自动化支持。Dustin、Rashka和Poston（《Automated Software Testing: Introduction, Management, and Performance》，Addison-Wesley, 1999）以及Graham和她的同事（《Software Test Automation》，Addison-Wesley, 1999）、Poston（《Automating Specification-Based Software Testing》，IEEE Computer Society, 1996）讨论了自动化测试的工具、策略和方法。

很多书考虑了特定应用领域的测试方法和测试策略。Gardiner（《Testing Safety-Related

Software: A Practical Handbook》, Springer-Verlag, 1999) 编辑了一本书, 讲述安全关键系统的测试。Mosley (《Client/Server Software Testing on the Desk Top and the Web》, Prentice-Hall, 1999) 讨论了客户机、服务器和网络构件的测试过程。Rubin (《Handbook of Usability Testing》, Wiley, 1994) 为对用户界面焦虑的那些人提供了有用的指导。

Binder[BIN99]描述了大约70个测试模式, 包括方法测试、类/簇测试、子系统测试、可复用的构件测试、框架测试、系统测试、测试自动化及特定数据库的测试。这些模式清单可在 www.rbsc.com/pages/TestPatternList.htm 中找到。

从网上可以获得大量的有关测试用例设计方法的信息源。最新的WWW资源列表可在

[460] SEPA Web站点<http://www.mhhe.com/pressman>中找到。

第15章 产品度量

要点浏览

概念：从本质上讲，工程是一个量化的学科。工程师用数字来帮助他们设计和评估要制造的产品。到目前为止，软件工程师在其工作中还鲜有定量指标——但是，情况正在发生变化。产品度量有助于软件工程师对其设计和构造的软件获得深层次的了解。与作为整体应用于项目（或过程）的项目度量和过程度量不同，产品度量侧重于软件工程工作产品的特定属性，它在执行技术任务（分析、设计、编码与测试）的过程中收集。

人员：软件工程师利用产品度量来帮助他们构造高质量的软件。

重要性：对计算机软件开发而言，定性要素总是存在的。问题是定性评估可能是不够的。软件工程师需要客观标准以帮助指导数据、体系结构、界面和构件的设计。测试人员需要定量指标以帮助

选择测试用例及其目标。产品度量为分析、设计、编码和测试能更客观地执行和更定量地评估提供基础。

步骤：测量过程的第一步是设计适合于所考虑软件的测度和度量。其次，收集导出公式化度量所需要的数据。一旦完成计算，基于预先建立的指导原则和过去的数据来分析适当的度量。解释分析结果以获得对软件质量的理解，且解释的结果将导致分析模型、设计模型、源代码或测试用例的修改。有些情况下，它也有可能导致软件过程本身的修改。

工作产品：使用从分析模型与设计模型、源代码和测试用例中收集的数据进行计算而得到产品的度量。

质量保证措施：应该在开始收集数据之前建立测量的目标，并以无歧义的方式定义每个产品的度量。仅仅定义不多的度量，然后使用它们去获得对软件工程工作产品质量的理解。

关键概念
功能点
GQM范型
指示器
McCall's因子
测量
属性
原理
测度
度量
分析模型
代码
设计模型
维护
面向对象
测试
质量

测量是任何工程过程的一个关键环节。我们使用测度以较好地理解所创建模型的属性，评估所制造工程产品或系统的质量。但是，与其他工程学科不同，软件工程并不是建立在基本的物理定量定律上。直接测度（如：电压、质量、速度或温度）在软件世界是不常见的。由于软件测度与度量经常是间接得到的，因此有广泛的争论空间。Fenton[FEN91]在讨论这个问题时说道：

测量是根据明确的规则来定义，将数字或符号赋给现实世界实体属性的过程……在物理学、医学、经济学以及近代社会科学中，我们能够测量原来认为不能测量的属性……当然，这些测量并不像物理中一些测量那样精练……但是，这些测量确实存在（并且常常根据这些测量做出重要的决策）。我们有责任尝试去测量不可测的东西，以便提高对特殊实体的理解。这在软件工程中与在其他学科中显得同样重要。

461 但是，软件业界的一些人始终在争论：软件是不可测量的，或者说，测量的尝试应该推迟，直到我们对软件以及用于描述软件的属性有较好的理解。这种说法是错误的！

虽然计算机软件产品度量往往并不是绝对的，但产品度量为我们提供了一种基于一组明确规定了规则来评价质量的系统方法。同时，产品度量为软件工程师对软件产品提供了现场而不是事后的理解。这使软件工程师能够在潜在问题变成灾难性的错误之前发现和纠正它们。

在本章，我们考虑当产品正在开发时，能用于评估产品质量的测度。这些内部产品属性的测度为软件工程师提供了以下方面的实时指示：分析模型、设计模型以及代码模型的功效，测试用例的有效性，以及待开发软件的总体质量。

15.1 软件质量

高质量的软件是一个重要目标，即使不喜欢软件工作的人也会同意这一点。但是，如何定义质量呢？在最一般的意义上，软件质量是对明确陈述的功能和性能需求、明确记录的开发标准以及对所有专业化软件开发应具备的隐含特征的符合度。

毫无疑问，可以修改或扩展以及无休止地讨论上述定义存在的一些问题。针对本书的目的，该定义强调了以下三个方面：

- 462**
1. 软件需求是质量测量的基础。不符合需求就是没有质量¹。
 2. 特定标准定义了一组用以指导软件开发方式的准则。若未能遵守准则，则肯定质量成问题。
 3. 有一组未被提及的隐式需求（如：对易使用性的期望）。若软件符合显式需求，但未能满足其隐式需求，则软件质量仍是值得怀疑的。

软件质量是多种因素的复杂混合体。这些因素随着应用和用户的不同而不同。以下几小节给出了软件质量因素的不同说法，并且描述了获取它们所需开展的活动。

15.1.1 McCall的质量因素



McCall的质量因素至今仍与它们在20世纪70年代首次提出时同样有效，因此可以合理地断言影响软件质量的因素不会随着时代的变化而变化。

463

影响软件质量的因素可分为两大类：(1) 可以直接测量的因素（如：测试期间发现的缺陷）；(2) 只能间接测量的因素（如：易用性和可维护性）。所有情况下，测量都必须发生。我们必须对软件（程序、数据和文档）和某个数据相比并获得质量指标。

McCall、Richards和Walters[MCC77]提出了影响软件质量因素的一种有用的分类。这些软件质量因素侧重于软件产品的三个重要方面：操作特性、承受变更的能力以及对新环境的适应能力，如图15-1所示。McCall及他的同事提供了如下描述：

正确性。程序满足其需求规格说明和完成用户任务目标的程度。

可靠性。期望程序以所要求的精度完成其预期功能的程度。[需要提醒大家注意的是，还有更完整的可靠性定义（见第26章）。]

效率。程序完成其功能所需的计算资源和代码的数量。

完整性。对未授权的人员访问软件或数据的可控程度。

¹ 注意，质量可扩展到分析模型、设计模型的技术特性以及这些模型的编码实现。高质量的模型（从技术的角度看）将导致高质量的软件（从用户的角度看）。

- 易用性。对程序学习、操作、准备输入和解释输出所需要的工作量。
- 可维护性。定位和修复程序中的一个错误所需要的工作量。
- 灵活性。修改一个运行的程序所需的工作量。
- 可测试性。测试程序以确保它能完成预期功能所需要的工作量。
- 可移植性。将程序从一个硬件和（或）软件系统环境移植到另一个所需要的工作量。
- 可复用性。程序（或程序的一部分）可以在另一个程序中使用的程度。这与程序所完成功能的包装和范围相关。
- 互操作性。将一个系统连接到另一系统所需要的工作量。

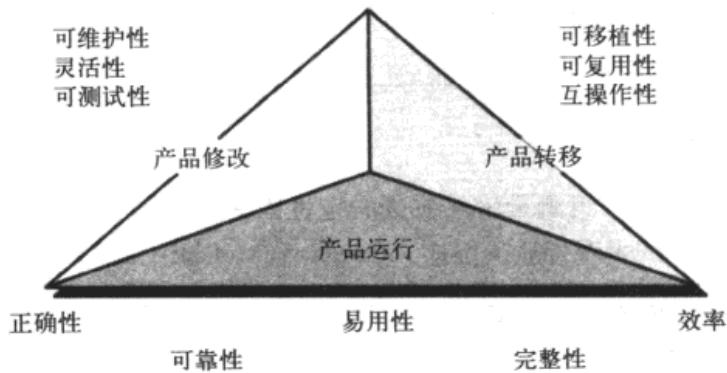


图15-1 McCall的软件质量因素

“评价产品的质量与它将世界向更好的方向改变了多少有关。”

—Tom DeMarco

开发这些质量因素的直接测度²是困难的，且在有些情况下是不可能的。事实上，由McCall等人定义的度量仅能主观地测量。度量可以用检查单形式为软件特定属性划分等级[CAV78]。

15.1.2 ISO 9126质量因素

ADVICE

利用这些质量因子建立你自己的检验表。首先，为每个项目指定一个相对重要性，然后，将工作产品分成等级以便评估正在开发软件的质量。

ISO 9126标准的制定是试图标识计算机软件的质量属性。这个标准标识了六个关键的质量属性：

功能性。软件满足所陈述需要的程度，由以下子属性指示：适宜性、准确性、互操作性、依从性和安全性。

可靠性。软件可用的时间长度，由以下子属性指示：成熟性、容错性和可恢复性。

易用性。软件容易使用的程度，由以下子属性指示：可理解性、易学习性和可操作性。

效率。软件优化使用系统资源的程度，由以下子属性指示：时间表现和资源表现。

可维护性。软件易于修复的程度，由如下子属性指示：可分析性、可修改性、稳定性和可测试性。

² 直接测度意味着存在一个简单可计算的值，它为被考察的属性提供直接的指标。例如，程序的“规模”可以通过计算代码的行数直接测量。

可移植性。软件可以从一个环境移植到另一个环境的容易程度，由以下子属性指示：适应性、可安装性、符合性和可替换性。

与第9章和15.1.1节所讨论的软件质量因素一样，ISO 9126中的质量因素不一定有助于直接测量。然而，它们确实提供了有价值的间接测量的基础和评估系统质量的优秀检查表。（译者注：针对ISO 9126的不足之处，近年来国际标准化组织对其做了修订和扩充，形成了4个新标准，即ISO/IEC 9126-1至ISO/IEC 9126-4。此外，还开发了软件产品评价的6个新标准，即ISO/IEC 14598-1至ISO/IEC 14598-6。）

“当工作者有意将事情做对或做得更好时，任何活动将变得富有创造性。”

——John Updike

15.1.3 向量化视图变迁

在前面几节中，讨论了一组测量软件质量的定性因素。我们设法开发软件质量的精确的测度，但有时又会为活动的主观性质所困惑。Cavano和McCall[CAV78]讨论了这种情形：

质量的确定是日常事件 [葡萄酒品尝比赛、运动赛事（例如：体操）、智力竞赛等等] 中的一个关键因素。在这些情况下，质量是以最基本、最直接的方式来判断的：在相同的条件和预先决定的概念下将对象进行一对一的对比。葡萄酒的质量可以根据清澈度、颜色、酒花和味道等来判断。然而，这种类型的判断是很主观的，最终的取值必须由专家给出。

主观性和特殊性也适用于软件质量的确定。为了帮助解决这个问题，需要对软件质量有一个更精确的定义，同样，为了客观分析，需要产生软件质量的定量方法……

在接下来的几节中，我们检查一组可应用于软件质量定量评估的软件度量。在所有的情况下，这些度量表示间接的测度。也就是说，我们从不真正测量质量，而是测量质量的一些表现。复杂因素在于所测量的变量和软件质量间的精确关系。

“就像温度测量一样，开始使用食指，慢慢地使用复杂的尺度、工具和技术，软件测量的成熟过程也是如此。”

——Shari Pfleeger

465

15.2 产品度量框架

如我们在本章前面提到的，测量将数字或符号赋给现实世界中的实体属性。为达到这个目标，需要一个包含统一规则的测量模型。尽管测量理论（例如：[KYB84]）及其在计算机软件中的应用（例如：[DEM81]、[BRI96]和[ZUS97]）等主题超出了本书的讨论范围，但是，为测量软件的产品度量建立一个基本框架和一组基本原则是值得的。

15.2.1 测度、度量和指标

尽管测量（measurement）、测度（measure）和度量（metrics）这三个术语常常可以互换使用，但注意到它们之间的细微差别是很重要的。由于“测度”一词可用作名词，也可用作

动词。因此，这个术语的定义是令人费解的。在软件工程中，“测度”为产品或过程的某些属性的程度、数量、维数、容量或大小提供量化的指示（这里只解释了“测度”作为名词的一部分含意。有时它作为动名词，特别是作为动词时其含意与measurement相同，我们也译为“测量”。——译者注）。而“测量”是确定测度的动作。度量在《IEEE标准词汇表》[IEE93]中的定义为：度量是一个系统、构件或过程具有给定属性的量化测量程度。

当收集了一个数据点（例如：在一个软件构件中发现的错误数），就已建立了一个测度。收集一个或多个数据点（例如：一些构件评审、调查单元测试以收集每个单元测试错误数的测度），由此产生测量。软件的度量以某种方式（例如：每次评审发现错误的平均数，或每个单元测试所发现错误的平均数）与单个测度相关。

软件工程师收集测度并开发度量以便获得指标。一个指标是一个度量或多个度量的组合，它提供了对软件过程、软件项目或产品本身的深入理解。指标提供的理解使项目经理或软件工程师能够调整过程、项目或产品以使其变得更好。

“一门科学与其测量工具一样成熟。”

——Louis Pasteur

15.2.2 产品度量的挑战

在过去30年中，许多研究人员试图开发一种能为软件复杂性提供全面测量的度量。Fenton[FEN94]将这种研究描绘为“寻找不可能的圣杯”。尽管已提出了许多复杂性测量[ZUS90]，但每种方法都对什么是复杂性以及哪些系统属性导致复杂性持有不同的看法。作为类比，我们考虑用来评价汽车吸引力的度量，一些观察者可能强调车身设计，另一些会强调机械特性，还有一些鼓吹价格、性能、燃料经济性或汽车丢弃时的回收能力。由于这些特性中的任意一个都有可能和其他特性不一致，因此，很难为“吸引力”制定一个单一值。对计算机软件而言，会出现同样的问题。

WebRef

有关产品度量的
大量信息由Horst
Zuse收集在
irb.cs.tuberlin.de
/~zuse/。

466

然而，仍有必要去测量和控制软件的复杂度。若这个质量度量的单一值难以获取的话，针对不同内部程序属性（例如：有效模块化、功能独立性和在第9章至第12章讨论的其他属性）开发测度应该是可能的。这些测量和由此产生的度量可用作分析模型和设计模型的独立指标。但是，这里出现了新问题，Fenton[FEN94]说道：“尝试去寻找刻画许多不同属性的测度是危险的，其危险性在于，测度不可避免地不得不满足有冲突的目标。这与测量的代表性理论是相反的。”尽管Fenton所说的是正确的，但许多人提出：在软件过程的早期阶段执行的产品测量为软件工程师评估质量提供一致和客观的机制。

然而，公平地说人们有理由知道产品度量的有效性。也就是说，产品度量与基于计算机系统的长期可靠性和质量的符合程度如何？Fenton[FEN91]用下面的方式回答了这个问题：

尽管软件产品的内部结构[产品度量]与其外部产品和过程属性之间直觉上存在联系，但事实上几乎没有科学地尝试去为它们建立特定的关系。这有许多原因，其中最普遍的说法是进行相关的实验是不实际的。

上述每个挑战都是值得警惕的原因，但它们并不构成摒弃产品度量的理由³。若要获得质量，则测量是必需的。

15.2.3 测量原则

在介绍一系列的产品度量之前，重要的是理解基本的测量原则。产品度量的作用主要包括：(1) 辅助分析模型和设计模型的评估；(2) 提供过程设计（procedural designs）和源代码复杂性的指示；(3) 方便更有效测试的设计。Roche[ROC94]提出一个能用以下五个活动描述的测量过程：

467

有效测量过
程的步骤是
什么？

公式化。导出适合于所考虑软件表示的测量和度量。

收集。用于导出公式化度量所需数据的积累机制。

分析。度量的计算和数学工具的使用。

解释。为获得对表示的质量的理解而评价度量。

反馈。从对递交给软件团队的产品度量的解释中获得建议。

ADVICE 软件度量仅当被有效地特征化和确认以证明它们的价值时才是有用的。提出了许多度量特征化和确认原则，其中一些有代表性的原则如下[LET03]：

度量应该具有引人的数学特性。也就是说，度量的值应该在有意义的范围之内（例如：0至1，其中0意味着不存在，1表示最大值，0.5表示中间点）。同时，所谓在合理范围内的度量不应该仅由顺序测量的成分组成。

当度量代表一个软件特征时，当正向品质出现时特征值提高，当不理想品质出现时特征值下降。度量值应该以同样的方式增加或减小。

每种度量在发布或用于做决策之前，应该在广泛的环境中根据经验加以确认。度量应该测量重要的、与其他因素无关的因素。它应该扩展到大系统中，并能在许多程序设计语言和系统领域中行得通。

尽管公式化、特征化和确认是关键的，但收集和分析是驱动测量过程的活动。Roche[ROC94]为这些活动提供了以下指导原则：(1) 只要有可能，数据的收集与分析应能自动化地进行；(2) 应该使用有效的统计技术以建立内部产品属性与外部质量特性之间的关系（例如：体系结构的复杂程度是否与产品使用报告中的缺陷数相关）；(3) 应该为每个度量建立解释性指导原则和推荐建议。

15.2.4 面向目标的软件测量

WebRef

GQM 的有益讨论可在www.the-dacs.com/Gold-Practices/practices/gqma.html找到。

468

目标/问题/度量（Goal/Question/Metric, GQM）范型是由Basili和Weiss[BAS84]开发的，这种技术用于为软件过程的任何部分识别出有意义的度量。GQM强调了以下要求：(1) 确定特定过程活动的明确的测量目标或将要评估的产品特性；(2) 定义一组必须回答的问题以达到目标；(3) 确定被良好公式化的度量以帮助回答这些问题。

目标定义模板[BAS94]可用于定义每个测量目标。模板采取以下形式：

在…{进行测量的环境}…环境中，从…{对测量感兴趣的人⁴}…的角度，关于…{活动或属性}

³ 尽管文献中对特定度量的评论是常见的，许多评论集中在深奥的主题上而忽略了现实世界中度量的主要目标：帮助软件工程师建立一个系统的客观的方法来获得对其工作的洞察且最终用来提高产品的质量。

⁴ van Solingen 和Berghout[SOL99]提出：目标几乎总是“理解、控制或改进”过程活动或产品属性。

被考虑的方面}…方面，为…{分析的总体目标}…目的，分析…{将要测量的属性和活动名}…。

作为一个例子，考虑SafeHome的目标定义模板：

分析SafeHome软件体系结构，目的是评估体系结构构件，涉及方面为使SafeHome具有较强可扩展性的能力，视角为完成该工作的软件工程师，环境为后续三年的产品改进。

明确定义了测量目标之后，形成一组问题。回答这些问题有助于软件团队（或其他共利益者）确定是否已达到测量目标。可能会问到的问题如下：

Q1：体系结构构件是否将以功能与数据分开的方式描述？

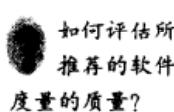
Q2：每个构件的复杂性是限定在一定的范围内以便于修改与扩展吗？

每个问题都应该利用一个或多个测度和度量以量化的方式回答。例如，度量若给出了体系结构构件内聚性（第9章）指标可能对回答Q1有用。15.4.1节或15.4.2节讨论的环复杂性及度量可能有助于对Q2的理解。

实际上，一些测量目标可能具有相关的问题和度量。在每种情况下，所选择（或派生）的度量应该与15.2.3节所讨论的测量原则和15.2.5节所讨论的测量属性相符。对于GQM的进一步信息，有兴趣的读者应该看[SHE98]或[SOL99]。

15.2.5 有效软件度量的属性

尽管已提出数百种计算机软件度量，但不是所有的度量都为软件工程师提供了实用的支持。一些度量所要求的测量太复杂，一些太深奥以致现实世界很少专业人员能够理解，另外一些则违反了高质量软件直觉上的基本概念。



经验表明，只有产品度量是直观的且易于计算时才会被使用。若需要大量的“计数”，且需要复杂的计算，则该度量不可能得到广泛采纳。

Ejiogu[EJI91]定义了一组有效软件度量所应具有的属性。导出的度量及导出度量的测度应该是：

- 简单的和可计算的。学习如何导出度量应该是相对容易的，且其计算不应该需要过多的工作量或时间。
- 在经验上和直觉上有说服力。度量应该满足工程师直觉上对所考虑的产品属性的概念。
- 一致的和客观的。度量应该总是产生无歧义的结果。
- 单位与量纲的使用是一致的。度量的数学计算应该使用不会导致奇异单位组合的测度。
- 编程语言的独立性。度量应该基于分析模型、设计模型或程序结构本身。
- 高质量反馈的有效机制。度量应该导致较高质量的最终产品。

469

尽管大多数软件度量满足这些属性，但一些常用的度量可能不满足其中某种属性。例如，对于功能点方法（在15.3.1节讨论）——一种软件交付功能的测量。有人提出⁵它不满足一致性和客观性这一属性，因为，独立的第三方与其同事就算用相同的软件信息也不可能导出同样的功能点值。难道我们应该由此拒绝使用功能点度量吗？当然不能！功能点提供了有用的理解且由此提供了独特的值，尽管它不能很好地满足一个属性。

15.2.6 产品度量全景

已提出了各种各样的度量分类法，下面概要地描述最重要的度量领域：

⁵ 可以提出积极有力的等价反向论据。这是软件度量的特征。

被考虑的方面}…方面，为…{分析的总体目标}…目的，分析…{将要测量的属性和活动名}…。

作为一个例子，考虑SafeHome的目标定义模板：

分析SafeHome软件体系结构，目的是评估体系结构构件，涉及方面为使SafeHome具有较强可扩展性的能力，视角为完成该工作的软件工程师，环境为后续三年的产品改进。

Jamie: 你说呢?

Vinod: 返工……正是这样。如果我们使用的一个度量有助于我们避免一个主要的或一个中等的问题，那它就节省了返工一部分系统所需要的时间，我们节省了时间。不是吗?

Ed: 我想这有可能，但你能保证某个产品度量能帮助我们找到问题吗?

Vinod: 你能保证它不能帮助我们找到问题吗?

Jamie: 那你计划怎么办?

Vinod: 我认为我们应该选择一些设计度量，可能是面向类的，将它们作为我们所开发的每个评审过程的一部分。

Ed: 我确实不太熟悉面向类的度量。

Vinod: 我花一些时间查查，然后给一些建议……怎么样，你们这些家伙?

(Ed与Jamie淡漠地点点头)

15.3 分析模型的度量

尽管文献中很少出现分析和规格说明度量，但对项目估算度量进行改造并将其应用于这个环境中是有可能的。这些度量以预测结果系统的规模为目的而检查分析模型。规模有时是(但不总是)设计复杂性的指示器，而且总是编码、集成和测试工作量增加的指示器。

15.3.1 基于功能的度量

WebRef

有关功能点的更多的有用信息可在www.ifpug.org和www.functionpoints.com找到。

功能点度量(FP)由Albrecht[ALB79]首次提出，可用做测量系统交付功能的有效手段⁶。利用历史数据，功能点度量可用于：(1)估算设计、编码和测试软件所需开销或工作量；(2)预计测试期间将遇到的错误数；(3)预测实现系统中的构件数和(或)预计的源代码行数。

利用基于软件信息域可数的(直接)测度和软件复杂性评估的经验关系来计算功能点。信息域的值用下列方式定义⁷：

外部输入数。每个外部输入(EI)源于一个用户，或从另一个应用系统中传送过来；它提供了面向应用系统的不同数据或控制信息。输入常用于更新内部逻辑文件(ILF)。输入应该与独立计数的查询相区分开来。

外部输出数。每个外部输出(EO)从应用系统中导出，并为用户提供信息。在这种情况下，外部输出指的是报告、屏幕、错误消息等。不对报告中的单独数据项进行分开计数。

外部查询数。一个外部查询(EQ)定义为一个在线输入。其结果是以在线输出(经常从ILF中得到)的方式产生某个即时软件响应。

内部逻辑文件数。每个内部逻辑文件(ILF)是驻留在应用系统边界之内的数据逻辑分组，它通过外部输入来维护。

外部接口文件数。每个外部接口文件(EIF)是驻留在应用系统外部的数据逻辑分组，但它为该应用系统提供有用的数据。

⁶ 自Albrecht最早的研究成果发布之后，出现了大量有关FP的书、论文和文章。一些有价值的书目可以在[IFP03]中找到。

⁷ 事实上，信息域值的定义及其计算的方式有点复杂。对于详细内容，有兴趣的读者应该参看[IFP01]。

一旦收集了这些数据，图15-2所示的表就完成了，且复杂度的值与每个计数相关。利用功能点方法的组织制定标准以确定表内某个特定的栏目是简单的、一般的还是复杂的。不过，复杂度的确定毕竟有一定的主观性。

为计算功能点 (FP)，利用下面的关系式：

$$FP = \text{总计} \times ([0.65 + 0.01 \times \sum(F_i)]) \quad (15-1)$$

其中，“总计”是从图15-2中所得到的所有FP项的总数。

信息域值	计数	加权因子			=
		简单的	中等的	复杂的	
外部输入	×	3	4	6	=
外部输出	×	4	5	7	=
外部查询	×	3	4	6	=
内部逻辑文件	×	7	10	15	=
外部接口文件	×	5	7	10	=
总计					→

图15-2 功能点的计算

F_i ($i=1 \sim 14$) 是值调整因子 (VAF)，它基于对下列问题的回答确定 [LON02]：

1. 系统需要可靠的备份和恢复吗？
2. 需要专门的数据通信从应用系统中传输信息或将信息传输到应用系统吗？
3. 存在分布处理功能吗？
4. 性能是关键的吗？
5. 系统将运行在一个现有的、紧张使用的操作环境吗？
6. 系统需要在线数据项吗？
7. 在线数据项需要对多个屏幕或操作建立输入事务吗？
8. ILF被在线更新吗？
9. 输入、输出、文件或查询是复杂的吗？
10. 内部处理是复杂的吗？
11. 所设计的代码是可复用的吗？
12. 转换与安装包括在设计中吗？
13. 系统是为不同组织中的多个安装而设计的吗？
14. 应用系统是为便于变更和易于为用户使用而设计的吗？

每个问题可用从0（不重要或不适用）到5（绝对必需）间的数值来回答。

式 (15-1) 中的常量值和应用于信息域计数的加权因子是由经验确定的。

为说明FP度量的使用，我们考虑一个简单的分析模型，如图15-3所示。图中描述了SafeHome软件的一个功能的数据流图 (第8章)。该功能管理用户交互，接收一个用户密码来启动或关闭系统，并且允许对安全区状态和不同安全传感器进行查询。该功能显示了一系列的提示信息且发送合适的控制信号到安全系统的不同构件。



473

值调整因子用于
提供问题复杂性的
指示。

WebRef

一个在线的FP
计算器可在irb.cs.
unimagdeburg.de/
sw-eng/us/java/
fp/找到。

474

为确定用以计算功能点度量所需的一组关键信息域测度，需要对数据流图加以评估。图中显示了三个外部输入——密码、紧急按钮和启动/关闭；以及两个外部查询——区域查询与

一旦收集了这些数据，图15-2所示的表就完成了，且复杂度的值与每个计数相关。利用功能点方法的组织制定标准以确定表内某个特定的栏目是简单的、一般的还是复杂的。不过，复杂度的确定毕竟有一定的主观性。

为计算功能点 (FP)，利用下面的关系式：

15.3.2 规格说明质量的度量



通过测量规格说明的特性，获得对其确定性和完备性的定量理解是可能的。

如何评价分析模型和相应的需求规格说明的质量呢？Davis与其同事[DAV93]就此提出了一系列特征：确定性（无歧义性）、完整性、正确性、可理解性、可验证性、内部与外部一致性、可完成性、简洁性、可跟踪性、可修改性、精确性和可复用性。另外，作者[DAV93]提到，高质量的规格说明是电子存储的、可执行的，或至少是可解释的、对比较重要之处加了注释的，另外还应是版本化的、有条理的、附有交叉索引的并且是适度说明的。

尽管在本质上上述的许多特征似乎是可以定性的，Davis等人[DAV93]建议每个特征可以用一个或多个度量来表示。例如，假设在一个规格说明中有 n_r 个需求，便有

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

其中， n_f 为功能需求数， n_{nf} 为非功能（如：性能）需求数。

为确定需求的确定性（无歧义性），Davis等人提出了一种度量，这种度量基于评审者对每个需求的解释的一致性：

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

其中， n_{ui} 是所有的评审者都有相同解释的需求数目。 Q_1 的值越接近1，规格说明的歧义性越低。

功能需求的完整性可以通过下列表达式来确定：

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

其中， n_u 为独特功能需求的数目， n_i 是由规格说明定义或隐喻的输入（刺激）的个数， n_s 是所指定状态的个数。 Q_2 测量了已为系统指定的必要功能的百分比。

然而，它并没有考虑非功能性需求。为了把这些非功能性需求结合到整体度量中以求完整性，我们必须考虑需求已经被确认的程度：

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

其中， n_c 是已经确认为正确的需求数， n_{nv} 是尚未确认需求的个数。

“测量可测量的东西，且使不可测量的东西可测量。”

——Galileo

15.4 设计模型的度量

很难想像一架新飞机、一个新计算机芯片或一个新办公楼可以在没有定义设计测度、没有确定设计质量各方面的度量的指导下开展设计。然而，基于复杂软件系统的设计，事实上通常是在并没有测量的情况下进行的。更讽刺的是，软件的设计度量是可用的，而很大一部分软件工程师却一直忽视了它们的存在。

与所有其他软件度量一样，计算机软件的设计度量并不是完美的。关于它们的功效及其应用方式一直存在争论。许多专家提出：在设计测度可以使用之前，需要进一步实验。然而，选择没有测量的设计是难以接受的。

15.4.1 体系结构设计度量



度量可以提供有关体系结构设计的结构化数据与系统复杂性的理解。

体系结构设计度量侧重于程序体系结构的特征，它强调体系结构（第10章）的结构和体系结构内模块或构件的有效性。这些度量从某种意义上讲是“黑盒的”，它并不需要一个特定软件构件的内部运作知识。

Card与Glass[CAR90]定义了三种软件设计复杂性测量：结构复杂度、数据复杂度和系统复杂度。

对于层次体系结构（例如，调用与返回体系结构），模块*i*的结构复杂度的定义方式如下：

$$S(i) = f_{\text{out}}^2(i) \quad (15-2)$$

其中， $f_{\text{out}}(i)$ 是模块*i*的扇出数⁸。

数据复杂度 $D(i)$ 提供了模块*i*的内部接口复杂度的指示，定义如下：

$$D(i) = v(i) / [f_{\text{out}}(i) + 1] \quad (15-3)$$

其中， $v(i)$ 是传入传出模块*i*的输入和输出变量的个数。

最后，系统复杂度 $C(i)$ 定义为结构复杂度和数据复杂度的总和：

$$C(i) = S(i) + D(i) \quad (15-4)$$

其中任何一个复杂度值上升，系统的整体体系结构复杂度也随之增加。这样集成与测试工作量增加的可能性也较大。

Fenton[FEN91]提出一些简单的形态（即，外形）度量，使不同的程序体系结构能够用一组简单的尺度进行比较。参考图15-5中调用与返回体系结构，可以定义下述度量：

$$\text{规模} = n + a$$

其中， n 为结点数， a 为弧数。对于图15-5所示的体系结构：

规模 = 17 + 18 = 35；

深度 = 4，深度是从根结点到叶结点的最长路径；

宽度 = 6，宽度是体系结构任一层次的最多结点数；

弧与结点的比例， $r = a/n$ 。

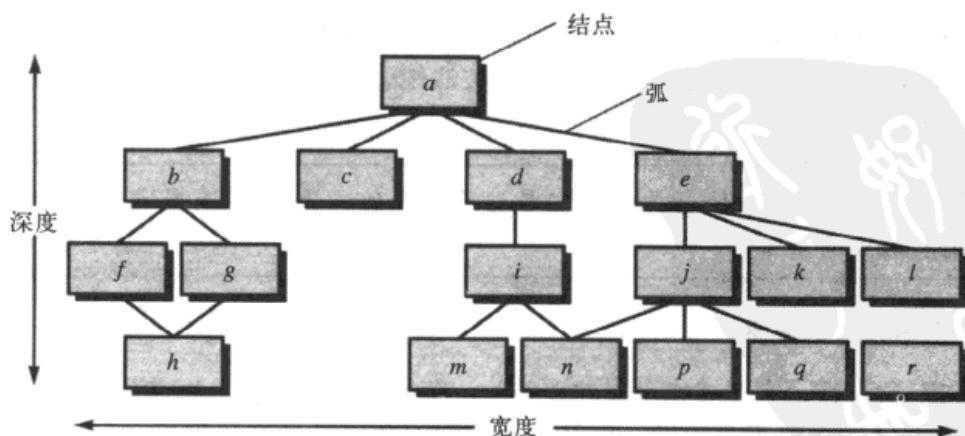


图15-5 形态度量

⁸ 扇出定义为直接从属于模块*i*的模块数，即，模块*i*直接调用的模块数。扇入定义为直接调用模块*i*的模块数。

这一比例给出了体系结构的连接密度，且对体系结构的耦合性提供了一个简单的指示。对于图15-5（原书该图中右下角I框和r框间漏掉一纵向连线。——译者注）中的体系结构， $r = 18/17 = 1.06$ 。

美国空军司令部[USA87]基于计算机程序可测量的设计特征，开发了一组软件质量指标。

478 利用类似于IEEE Std.982.1-1988[IEE94]标准中提出的概念，使用从数据和体系结构设计中取得的信息，导出了一个范围从0到1的设计结构质量指标 (design structure quality index, DSQI)。为计算DSQI的值，必须先搞清楚下列值[CHA89]：

S_1 = 程序体系结构中定义的模块总数

S_2 = 其正确功能依赖于数据源输入或产生用于其他地方的数据的模块数（通常控制模块不计入 S_2 之内）

S_3 = 其正确功能依赖于前面处理的模块数

S_4 = 数据库中的条目数（包括所有数据对象及定义对象的所有属性）

S_5 = 数据库不重复的数据项总数

S_6 = 数据库段（不同记录或个体对象）的数目

S_7 = 具有单个人口和出口（异常处理不看作多重出口）的模块数

一旦计算机程序的 S_1 至 S_7 的值确定之后，以下中间值可以计算出来：

程序结构： D_1

其中 D_1 的定义如下：若体系结构的设计是用一种独特的方法（如，面向数据流设计或面向对象设计）来开发的，则 $D_1 = 1$ ，否则 $D_1 = 0$ 。

模块独立性： $D_2 = 1 - (S_2/S_1)$

模块不依赖于前面处理： $D_3 = 1 - (S_3/S_1)$

数据库规模： $D_4 = 1 - (S_5/S_4)$

数据库项的划分： $D_5 = 1 - (S_6/S_4)$

模块入口/出口特性： $D_6 = 1 - (S_7/S_1)$

当这些中间值确定后，DSQI用下列方式来计算：

$$\text{DSQI} = \sum w_i D_i \quad (15-5)$$

其中， i 的值为1至6， w_i 是相对权值，考虑了每个中间值的重要性，且 $\sum w_i = 1$ （若所有 D_i 的权值相等，则 $w_i = 0.167$ ）。

过去设计的DSQI值可以确定下来，并与目前正在开发的设计相比较。若DSQI明显低于平均值，意味着需要进一步做设计工作与评审。类似地，若对现有的设计做重要变更，则那些变更对DSQI的影响可以计算出来。

479 “测量可以看作是一条绕行的路，它是必不可少的，因为[没有量化的支持]多数人不能作出明确和客观的决策。”

——Horst Zuse

15.4.2 面向对象设计的度量

关于面向对象设计，有很多东西是主观的——有经验的设计者“知道”如何去刻画面向对象系统以使之有效地实现用户需求。但是，当面向对象设计模型的规模和复杂性增加时，更客观地看待设计特征对有经验的设计者（他可获得更为深入的理解）和新手（他可以获得质量指标，否则这些指标是得不到的）都是有益的。

在讨论面向对象系统的软件度量时，Whitmire[WHI97]描述了面向对象设计的9个独特的、可测量的特性：

当评估一个面向对象模型设计时，可以测量什么特性？

规模。规模可以从四个方面来定义：总数量、容量、长度和功能。总数量通过对面向对象实体（如类或操作）的静态计数来测量。容量测度与总数量测度相同，但是，它是针对给定时间点动态收集的。长度是对互连的设计元素链的测度（例如：继承树的深度是一种长度测度）。功能度量间接指示出交付给客户的某个面向对象应用的价值。

复杂性。与规模一样，存在很多不同的软件复杂性观点[ZUS97]。Whitmire通过检查面向对象设计的类如何互相关联来看待结构化特征方面的复杂性。

耦合性。面向对象设计的成分间的物理连接（例如：类间的合作数量或对象间传递的消息数）表示了一个面向对象系统的耦合性。

充分性。Whitmire将充分性定义为“从当前应用的角度看，一个抽象拥有其所需特征的程度，或一个设计构件拥有其抽象特征的程度。”换句话说：该抽象（类）若对我们有用，需要具备哪些特性？[WHI97]。本质上，一个设计构件（例如：一个类）是充分的，只要它完全反映了所建模的应用域对象的所有性质，即该抽象（类）拥有它需要的性质。

“过去必须依靠民间传说和神话所做的许多决策，现在可以利用量化数据来做了。”

——Scott Whitmire

完备性。完备性与充分性唯一的不同在于，“我们用于比较抽象或设计构件的特征集”[WHI97]。充分性从当前应用的角度来比较抽象。完备性则考虑多个角度，它要问：“为了完全表示问题域对象，需要什么特性？”由于完备性的标准考虑了不同的角度，它间接隐含了抽象或设计构件可以复用的程度。

内聚性。与它在传统软件中的对应项一样，面向对象构件应该采取的设计方式是：使所有的操作一起工作，以达到单一的、定义良好的目标。类的内聚性是通过检查“它所拥有的性质集合是问题域或设计域的一部分”[WHI97]的程度来确定的。

原始性。这是与简单性相类似的特征，原始性（应用于操作和类）是指某操作的原子性程度，即操作不能由包含在类中的其他操作序列构造而成。一个展示高原始性程度的类仅仅封装原始操作。

相似性。两个或多个类在其结构、功能、行为或目的方面的相似程度。

易变性。在本书的前面我们已经看到，当修改需求或系统的其他部分时，设计变更可能发生，从而导致设计构件的适应性修改。面向对象设计的易变性测量了将发生变更的可能性。

实际上，面向对象系统的产品度量不仅可以应用于设计模型，也可以应用于分析模型。在下面的几节中，我们将探讨在类层次和操作层次上提供质量指标的度量。

480

15.4.3 面向类的度量——CK度量集

类是面向对象系统的基本单位，因此，测量和度量个体类、类层次和类协作，对必须评估设计质量的软件工程师没有多少价值。在前面几章中，我们已了解类封装操作（处理）和属性（数据），类经常是子类（有时称为子女）的“父辈”，子类继承父类的属性和操作，类

常与其他类协作。每种特征都可以用作测量的基础⁹。



481
类的方法数及其复杂性与测试该类所需工作量直接相关。

其中一个被广泛引用的面向对象软件度量集是由Chidamber和Kemerer [CHI94]提出的。他们提出了六个面向对象系统的基于类设计的度量，通常称为是CK度量集¹⁰。

每个类的加权方法 (weighted methods per class, WMC)。假定为类C的n个方法定义的复杂度分别为 c_1, c_2, \dots, c_n ，所选择的特定复杂性度量（例如：环复杂度）应该规范化，以便方法的额定复杂度取值为1.0。

$$WMC = \sum c_i \quad (i=1, \dots, n)$$

方法的数目及其复杂度是实现和测试类所需工作量的合理指标。此外，方法数目越多，继承树越复杂（所有的子类继承其父类的方法）。最后，对于给定类，随着方法数目的增长，它可能越来越成为特定于应用的，由此限制了潜在的复用。因此，WMC应保持合理的低值。

虽然对类的方法的计数似乎是相当直接的，但问题比实际看上去要复杂。对方法的计数应该采用一致的方法[CHU95]。



482
继承的功能非常强大，但如果小心使用，可能会陷入麻烦。使用DIT和其他度量可以理解类层次的复杂性。

继承树的深度 (depth of the inheritance tree, DIT)。这个度量为“从结点到树根的最大长度”[CHI94]。参看图15-6，所显示类层次的DIT值为4。随着DIT的增大，低层次的类有可能将继承很多方法。当试图预测类行为时，将带来潜在困难。一个深的类层次（DIT值大）也导致较大的设计复杂性。从正面来讲，大的DIT值意味着许多方法可以复用。

子女的数量 (number of children, NOC)。在类层次中，直接从属于某类的子类称为子女。如图15-6所示，类C₂有3个子女——子类C₂₁、C₂₂和C₂₃。随着子女数量的增长，复用也增加。但是，当NOC增大时，父类所表示的抽象可能削弱，有些子女不是父类的合适成员。当NOC增大时，测试（需要在其运行环境中检查每个子女）的工作量将也随之增加。

对象类之间的耦合 (coupling between object classes, CBO)。CRC模型（第8章）可用于确定CBO的值。本质上，CBO是在CRC索引卡上所列出类的协作的数量¹¹。当CBO增大时，类的可复用性将有可能减小。CBO的高值也使修改与随之而来的测试变得更为复杂。通常，每个类的CBO值应该保持适当的低值。这与传统软件中减少耦合和内聚的概念均可应用于传统软件和OO软件。保持类的低耦合及类和操作的高内聚。

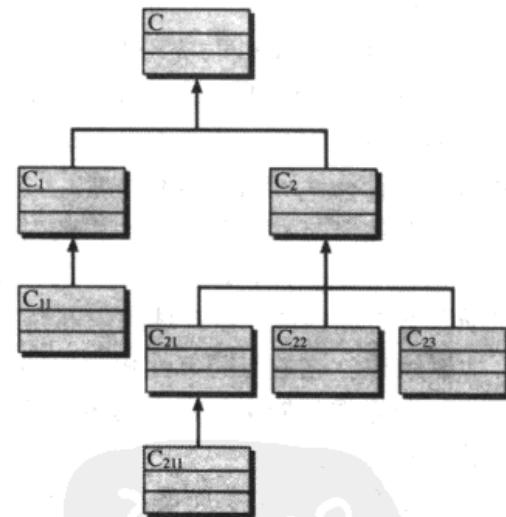


图15-6 类层次

⁹ 应该注意，在本章讨论的某些度量的有效性当前在技术文献中仍有争论。那些支持测量理论的人们要求具有某些OO度量无法提供的某种程度的形式化。然而，说这里提到的所有度量为软件工程师提供了有益的理解是不为过的。

¹⁰ Chidamber与Kemerer利用术语“方法”而不是“操作”。本节该术语采纳他们的用法。

¹¹ 若CRC索引卡由手工开发，在CBO被可靠地确定之前，必须评估其完整性和一致性。

合性的一般性指导原则是一致的。

对类的响应 (reponse for a class, RFC)。类的响应集是“该类的某对象接收到消息，作出响应而可能执行的一组方法” [CHI94]。RFC是响应集中的方法数。当RFC增大时，由于测试序列(第14章)增加，测试的工作量也随之增加。同样，当RFC增大时，类的整体复杂性也随之增加。

方法中缺少内聚 (lack of cohesion in methods, LCOM)。类C中的每个方法访问一个或多个属性(也称为实例变量)，LCOM是访问一个或多个相同属性的方法的数量¹²。若没有方法访问相同的属性，则 $LCOM = 0$ 。为说明 $LCOM \neq 0$ 的情形，考虑一个具有6个方法的类，其中4个方法有一个或多个属性是共同的(即它们访问共同属性)，因此 $LCOM = 4$ 。若LCOM的值高，则方法可能通过属性相互耦合，这增加了类设计的复杂性。虽然有些情况下LCOM取高值有其理由，但是，总是希望保持高内聚，即保持LCOM的低值¹³。

SAFEHOME

CK度量的应用

[场景] Vinod的工作间。

[人物] Vinod、Jamie、Shakira和Ed, SafeHome软件工程团队成员，他们正在进行构件级设计和测试用例设计。

[对话]

Vinod: 你们看过我周三发给你们的关于CK度量集的描述吗，做过一些测量吗？

Shakira: 不是太复杂。正如你建议的那样，我退回我的UML类和顺序图，并得到DIT、RFC和LCOM的粗略值。我没找到CRC模型，因此不能计算CBO。

Jamie (笑)：你没找到CRC模型是因为它在我这里。

Shakira: 这就是我喜欢这个团队的原因——大家能很好地沟通。

Vinod: 我计算……你们为CK度量算过数吗？(Jamie和Ed肯定地点点头)。

Jamie: 我有CRC卡，所以我看了看CBO。大部分类看上去相当一致，有一个例外我已经做了标记。

Ed: 有些类的RFC值相当高，相对于平均值……或许我们应该看看能否对它们进行简化。

Jamie: 可能行，也可能不行。我仍然担心时间，我不想修改那些能正常工作的类。

Vinod: 我同意这个观点。或许我们应该查找至少有两个或更多的CK度量值不太好的类，有两项不利就得改。

Shakira (查看Ed的具有较高RFC值的类列表)：看这个类，它的LCOM和RFC的值都高，是两项不利吧？

Vinod: 我认为是这样……由于复杂性，实现和测试都是困难的。也许值得设计两个不同的类来实现同样的行为。

Jamie: 你认为修改它将节省时间吗？

Vinod: 从长远的眼光来看，是这样。

483

¹² 其形式化的定义更为复杂，详细内容见[CHI94]。

¹³ 在一些情况下，LCOM测度提供有益的理解，但在另一些情况下可能会产生误导。例如，让耦合封装在一个类中在整体上提高了系统的内聚性。因此，至少在这种意义上，较高的LCOM确实说明类具有较高的内聚性，而不是较低的内聚性。

15.4.4 面向类的度量——MOOD度量集

Harrison、Counsell和Nithi[HAR98]提出了一组面向对象设计的度量，这组度量提供了面向对象设计特征的定量指标。以下给出MOOD度量的部分样例：

方法继承因子 (method inheritance factor, MIF)。一个面向对象系统的类体系结构利用方法（操作）和属性继承的程度可定义为：

$$\text{MIF} = \sum M_i(C_i) / \sum M_a(C_i)$$

这里是对*i*从1到*T_c*求和。*T_c*定义为体系结构中类的总数，*C_i*是体系结构中的一个类，而且，

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

其中，*M_a(C_i)*为可在*C_i*关联中被调用的方法的总数；*M_d(C_i)*为类*C_i*中声明的方法的总数；*M_i(C_i)*为类*C_i*中继承的（未被覆写的）方法的数量。

MIF的值（属性继承因子——attribute inheritance factor, AIF——以类似的方式定义）

484 指示了继承对面向对象软件的影响。

“随着[面向对象]范型的进一步普及，为评估其质量所做的面向对象软件的分析变得越来越重要。”

—Rachel Harrison等

耦合因子 (coupling factor, CF)。本章前面提到，耦合是对面向对象设计中元素间连接的指示。MOOD度量定义耦合如下：

$$\text{CF} = \sum_i \sum_j \text{is_client}(C_i, C_j) / (T_c^2 - T_c)$$

这里，对*i*从1到*T_c*和*j*从1到*T_c*求和。函数`is_client = 1`，当且仅当用户类*C_c*与服务类*C_s*间存在关系，且*C_c ≠ C_s*，否则，`is_client = 0`。

尽管许多因素影响软件复杂性、可理解和可维护性，但是，可以合理地得出这样的结论：随着CF值的增大，面向对象软件的复杂性将随之增加，由此，可理解和可维护性和潜在的可复用性将受到损害。

Harrison与他的同事[HAR98]对MIF、CF及其他度量进行了详细分析，并检查了它们用在设计质量评估中的有效性。

15.4.5 Lorenz与Kidd提出的面向对象度量

在关于面向对象度量的书籍中，Lorenz与Kidd[LOR94]将基于类的度量分为与构件级设计相关的四类：规模、继承、内部和外部。面向对象设计类的面向规模的度量侧重于对单个类的属性和操作的计数以及面向对象系统的整体平均值。基于继承的度量侧重于整个类层次中操作被复用的方式。类的内部度量考察内聚与面向代码的问题。外部度量检查耦合性与复用。Lorenz与Kidd提出的度量样本如下：

类的规模 (class size, CS)。类的整体规模可以用下面的测度来确定：

- 封装在类中的操作的总数（包括继承来的和私有的实例操作）；
- 封装在类中的属性的总数（包括继承来的和私有的实例属性）。

Chidamber和Kemerer（15.4.3节）提出的WMC度量也是类规模的加权测

ADVICE
在分析模型的评审中，CRC索引卡将提供对CS期望值的合理指示。如果遇到一个拥有大量职责的类，可以考虑将其分解。

485

量。正如我们前面提到的，大的CS值指明类可能有太多的职责。这将减小该类的可复用性且使实现和测试更复杂。一般来讲，在确定类的规模时，继承的和公有的操作与属性应该有较大的加权值[LOR94]。使私有操作和属性特化且在设计中更加局部化。也可以计算类属性和操作数量的平均值。CS的平均值越低，类在本系统中能被广泛复用的可能性越高。

由子类加入操作的数量 (number of operations added by a subclass, NOA)。子类通过加入操作和属性来特化。随着NOA值的增加，子类越远离超类中隐含的抽象。一般来讲，当类层次的深度增加 (DIT值变大) 时，在类层次中低层的NOA的值将下降。

15.4.6 构件级设计度量

传统软件构件的构件级设计度量侧重于软件构件的内部特性，其包括模块的“三C”——内聚性 (cohesion)、耦合性 (coupling) 和复杂度 (complexity) ——的测度。这些测度有助于软件工程师判断构件级设计的质量。

在需要考虑模块内部运作知识的意义上讲，本节讨论的度量是“玻璃盒”，一旦开发了过程设计 (procedural design)，构件级设计度量就可以应用。另外，它们也可以延迟到有源代码时才用。



可以计算一个构件的功能独立性 (耦合和内聚) 的度量并用其评估一个设计的质量。

内聚性度量。 Bieman和Ott[BIE94]定义了一组为模块内聚性提供指示的度量。该度量用五个概念和测度来定义：

数据切片。简单地说，数据切片是对模块进行回溯走查，它寻找当走查开始时影响模块状态的数据值。应该注意到，程序切片（侧重于语句和条件）和数据切片都可以定义。

数据记号。为模块定义的变量可以定义为该模块的数据记号。

胶合记号。存在于一个或多个数据切片中的数据记号的集合。

超胶合记号。在一个模块中，由每个数据切片公用的数据记号。

粘度。一个胶合结点的相对粘度是与所绑定的数据切片的数目直接成比例的。

Bieman和Ott开发了强功能内聚性 (SFC)、弱功能内聚性 (WFC) 和依附性 (胶合记号绑定数据切片的相对程度) 的度量。这些度量可以用下列方式解释[BIE94]：

486

所有这些内聚性度量的取值范围都在0到1之间。当一个过程 (procedure) 有多于一个输出且没有展示任何由特定度量指示的内聚属性时，内聚性度量取值为0。没有超胶合记号的过程，即过程中没有为所有数据切片共用的记号，这种过程具有0强功能内聚——没有影响所有输出的数据记号。没有胶合记号的过程，即过程中没有为一个以上数据切片共用的记号（在有多个数据切片的过程中），这种过程展示0弱功能内聚和0依附性——没有影响一个以上输出的数据记号。

当Bieman和Ott度量取最大值时，出现强功能内聚性和依附性。

耦合度量。模块耦合为一种模块与其他模块、全局数据和外部环境的连接提供指示。在第9章，以定性的方式讨论了耦合性。

Dhama[DHA95]提出了一个包含数据与控制流耦合、全局耦合和环境耦合的模块耦合度量。计算模块耦合性所需要的测度可以按照上述三种耦合类型来定义。对于数据与控制流耦合，有：

d_i =输入数据参数的个数

c_i =输入控制参数的个数

d_o =输出数据参数的个数

c_o =输出控制参数的个数

对于全局耦合，有：

g_d =用作数据的全局变量的个数

g_c =用作控制的全局变量的个数

对于环境耦合，有：

w =被调用模块的个数（扇出）

r =调用所考虑模块的模块数（扇入）

利用这些度量，模块耦合指标 m_c 可用下式定义：

$$m_c = K/M$$

其中， K 为比例常数，且

$$M = d_i + (a \times c_i) + d_o + (b \times c_o) + g_d + (c \times g_c) + w + r \quad (15-6)$$

K, a, b, c 的值必须根据经验导出。

随着 m_c 值的增大，整体模块的耦合性降低。为了让耦合度量随着耦合程度的上升而上升，对耦合度量加以改进，定义为：

$$C = 1 - m_c$$

其中，式(15-6)的测度增加时，耦合度的值随之增加。

复杂性度量。可以计算多种软件度量以确定程序控制流的复杂度，其中许多度量是基于流图的。如第14章所讨论的，图作为一种表示方法，它由结点和连接（边）构成。当连接（边）有向时，流图为有向图。

McCabe和Watson[MCC94]为复杂性度量指出了一些重要应用：

复杂性度量可用来在源代码的自动分析[或过程设计信息]中预计有关可靠性和可维护性的关键信息。同时，复杂性度量也在软件项目中提供反馈以有助于控制[设计活动]。

在测试和维护期间，它们提供了软件模块的详细信息以有助于指出潜在的不稳定区域。

POINT 计算机软件最广泛使用（且被争论）的复杂性度量是环复杂度，它最先是由Thomas McCabe([MCC76]、[MCC89])提出的，在第14章有详细的讨论。

Zuse([ZUS90]、[ZUS97])讨论了许多不同的软件复杂性度量，多达18种以上。作者为每种度量给出了基本定义（例如，有许多环复杂性度量的变体），并对其进行了分析和评论。Zuse的著作是迄今为止最为全面的。

15.4.7 面向操作的度量

由于类是面向对象系统中最主要的单元，因此，已提出的度量很少是针对类中操作的。Churcher和Shepperd[CHU95]讨论这个问题时说：“近来的研究结果表明，根据语句数和逻辑复杂性[WIL93]，方法倾向于小，并提出系统的连接结构可能比单个模块的内容更重要。”然而，通过检查方法（操作）的平均特征，可以获得一些了解。由Lorenz与Kidd[LOR94]提出的

三个简单度量比较适当：

平均操作规模 (average operation size, OS_{avg})。尽管代码的行数可用作操作规模的指示器，但LOC测度受到第22章所讨论的一系列问题的影响。因此，操作所发送的消息数为判断操作的尺寸提供一个选择。当单一操作所发送的消息数增加时，类中的职责有可能未能很好地分配。

操作复杂性 (operation complexity, OC)。任何一种为传统软件提出的复杂性度量，都可以用来计算操作的复杂性[ZUS90]。由于操作应该限于特定的职责，设计者应该保持OC的值尽可能的低。488

每个操作参数的平均数 (average number of parameters per operation, NP_{avg})。操作的参数越多，对象间的合作越复杂。一般来讲，应该保持 NP_{avg} 的值尽可能的低。

15.4.8 用户界面设计度量

尽管在人机界面设计方面有许多重要文献（第12章），但是，有关界面质量和易用性度量的信息却比较少。

Sears[SEA93]提出：布局恰当性（layout appropriateness, LA）是人机界面设计的一个有价值的度量。典型的图形用户界面（GUI）使用布局实体（图标、文本、菜单、窗口等）帮助用户完成任务。为了用GUI完成给定任务，用户必须从一个布局实体移动到另一个实体。每个布局实体的绝对位置和相对位置、使用频率以及从一个布局实体迁移到另一个的“开销”，这些都影响着界面的恰当性。

“从为洗碗机上添加碗盘这项工作中你至少能学会一条用户界面设计原则，就是在机器上堆得太多，那就会使哪个碗盘都洗不干净。”

——作者不详



界面设计度量是精细的，但最重要的是，确信你的最终用户喜欢该界面，并对所需的交互感到舒服。

Kokol与他的同事[KOK95]为UI（用户界面）屏幕定义了内聚性度量，它测量了一个屏幕内容与另一个屏幕内容的相对连接。若展示在屏幕上的数据（或其他内容）属于单个主要数据对象（像分析模型中定义的），则该屏幕UI的内聚性是高的。若展现多种不同类型的数据或内容且这些数据与不同的数据对象相关，则UI的内聚性是低的。作者为内聚性提供了经验模型[KOK95]。

此外，UI交互的直接测度可以侧重于以下方面：获得某特定场景或操作所需时间，从错误情况中恢复所需时间，获得一个用例所需的特定操作或任务的数量，展现在屏幕上的数据或内容对象的数量，文本密度与规模，以及一些其他测量。然而，这些直接测量必须加以组织，从而建立有意义的UI度量，这样才能改进UI质量和（或）易用性。

值得指出的是，GUI设计的选择可以用度量（如LA或UI屏幕内聚性）作指导，但终裁者应该是基于GUI原型的用户输入。Nielsen和Levy[NIE94]说道：“若一个人仅基于用户观点选择界面[设计]，他就有相当大的机会获得成功。用户的平均任务性能和他们对GUI的主观满意度是紧密相关的。”

15.5 源代码的度量

489

Halstead的“软件科学”理论[HAL77]提出了计算机软件的第一个分析“定律”¹⁴。

¹⁴ 应该指出，Halstead的“定律”已经产生很大的争议，并不是每个人都同意他的基本理论。然而，对一些编程语言的实验验证已完成（如：[FEL89]）。

Halstead利用可以在代码生成后导出的或一旦设计完成之后可以估算得到的一组基本测度，指出了用于计算机软件开发的定量定律。软件科学使用的一组基本测度如下：

n_1 =在程序中出现的不同操作符的数量

n_2 =在程序中出现的不同操作数的数量

N_1 =出现的操作符总数

N_2 =出现的操作数总数



操作符包括所有的控制流结构、条件和数学运算。操作数包括所有的程序变量和常量。

Halstead利用这些基本测度开发了一些表达式，这些表达式可用于度量整个程序的长度、算法的最小潜在信息量、实际信息量（指定一个程序所需的“位”数）、程序层次（一种软件复杂性测度）、语言级别（对给定语言为一常量）和其他特征（如：开发工作量、开发时间，甚至软件中的预计缺陷数）。

Halstead表明，长度 N 可以估算如下：

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

而程序的信息量可以定义为：

$$V = N \log_2 (n_1 + n_2)$$

应该注意到， V 随着编程语言的不同而不同，它表示了说明一个程序所需要的信息量（以“位”计数）。

“人脑遵守一套比较严格的规则【用于开发算法】，这些规则比已知的规则严格得多。”

—Maurice Halstead

理论上，一个特定算法必定存在一个最小信息量。Halstead将信息量比率 L 定义为程序最简洁形式的信息量与实际程序的信息量之比。实际上， L 一定总是小于1的。根据基本测度，信息量比率可以表示为：

$$L = 2/n_1 \times n_2/N_2$$

Halstead的工作有必要通过实验证，而且大量的研究已针对软件科学进行。进一步的信

490

息见[ZUS90]、[FEN91]和[ZUS97]。

15.6 测试的度量

尽管已有不少著作讨论软件测试度量（如：[HET93]），但提出的大部分度量都侧重于测试过程而不是测试的技术特征本身。一般来讲，测试者必须依靠分析、设计和代码度量来指导测试用例的设计与执行。

基于功能的度量（15.3.1节）可以用作整体测试工作量的预报器。以往项目的各种项目级特征（例如：测试工作量与时间、未发现的错误以及所产生的测试用例数）可以收集起来，将这些与项目组开发的功能点数关联起来考虑。然后，项目组可为当前项目预计这些特征的“期望”值。



测试度量可分为两大类：(1) 试图预测在各个测试层次所需测试数量的度量；(2) 对于给定的构件，侧重于测试覆盖率为度量。

体系结构设计度量提供了与集成测试相关的难易信息以及对专用测试软件（如：桩模块和驱动程序）的需求。环复杂度（一种构件级设计度量）是基本路径测试（第14章描述的测试用例设计方法）的核心。此外，环复杂度还可用来定位要进行广泛单元测试的候选模块。环复杂度高的模块可能比环复杂度低的模块更易于出错。因此，测试人员应该在将这些模块集成到系统之前花费超过平均值的工作量以发现该模块中的错误。

15.6.1 应用于测试的Halstead度量

从Halstead测度（15.5节）导出的度量也可用来估算测试工作量。利用程序信息量定义 V ，可以计算程序层次 PL 以及Halstead工作量 e ：

$$PL = 1/[(n_1/2) \times (N_2/n_2)] \quad (15-7a)$$

$$e = V/PL \quad (15-7b)$$

分配给模块 K 的工作量占整体测试工作量的百分比可以用下式进行估算：

$$\text{测试工作量百分比 } (K) = e(K)/\sum e(i) \quad (15-8)$$

其中， $e(K)$ 是利用式（15-7）计算的模块 K 的测试工作量，式（15-8）的求和是系统所有模块的Halstead工作量的总和。

15.6.2 面向对象测试的度量

15.4节提到的面向对象设计度量为设计质量提供了一种指示，它也为检查一个面向对象系统所需要的测试工作量提供了通用的指示。

Binder[BIN94]提出了一组对面向对象系统的可测试性具有直接影响的设计度量。该度量考虑了封装和继承方面。采用以下的量化方法：



OO测试可能是相当复杂的。度量可以帮助你基于所测量的特征来锁定值得“怀疑”的线程、场景和类包。

方法中缺少内聚（lack of cohesion in methods, LCOM）¹⁵。LCOM的值越高，为保证方法不会产生副作用，需要被测试的状态越多。

公有与保护属性的百分比（percent public and protected, PAP）。该度量指明类的公有属性或保护属性的百分比。PAP的高值增加了类间副作用的可能性，由于公有和保护属性导致较高的潜在耦合（第9章）¹⁶。必须设计保证发现这些副作用的测试。

对数据成员的公有访问（public access to data members, PAD）。这个度量指明可以访问另一个类的属性的类（或方法）的数量，这违背了封装。PAD的高值导致类间的潜在副作用，必须设计保证发现这些副作用的测试。

根类的数量（number of root classes, NOR）。该度量是在设计模型中描述的不同类层次的计数。必须为每个根类和相应的类层次开发一组测试。当NOR增大时，测试工作量也随之增加。

¹⁵ 参见15.4.3节中LCOM的描述。

¹⁶ 一些人支持这样的观点：设计中没有属性是公有的或私有的，即PAP=0。这意味着，在其他类中的所有属性必须通过其方法来访问。

扇入 (fan-in, FIN)。当用于面向对象环境时，扇入是多继承的指示。FIN>1指示类从多个一个根类中继承属性和操作。应该尽可能避免FIN>1的情况。

子女数 (number of children, NOC) 和**继承树的深度** (depth of the inheritance tree, DIT)¹⁷。如第14章所讨论的，对每个子类，必须重新测试超类的方法。

15.7 维护的度量

本章所介绍的所有软件度量均可用于新软件的开发和现有软件的维护。然而，人们已提出了专门针对维护活动的度量。

492 IEEE Std.982.1-1988[IEE94]提出了一种软件成熟度指标 (software maturity index, SMI)，它提供了对软件产品稳定性的指示 (基于每个产品发布的变更)。可以确定以下信息：

M_T =当前发布的模块数量；

F_c =当前发布中已变更的模块数量；

F_a =当前发布中已增加的模块数量；

F_d =当前发布中已删除前一发布中的模块数量。

软件成熟度指标用下列方式计算：

$$SMI = [M_T - (F_a + F_c + F_d)] / M_T$$

当SMI的值接近1时，产品开始稳定。SMI也可用于软件维护活动策划的度量。产生软件产品的某个发布的平均时间可以与SMI联系起来，且可以为维护工作量开发一个经验模型。

SOFTWARE TOOLS

产品度量

目的：帮助软件工程师开发有意义的度量，用以评估分析与设计建模、源代码生成以及测试期间所产生的工作产品的质量。

机制：这类工具涉及广泛的度量。它们要么作为独立的形式出现，要么附在分析与设计、编码或测试工具中（这种比较常见）。在多数情况下，度量工具分析软件的一种表示（例如：UML模型或源代码），并由此形成一种或多种度量。

代表性工具¹⁸

Krakatau Metrics，由Power Software (www.powersoftware.com/products) 开发，计算C/C++和Java的复杂性度量、Halstead度量及相关度量。

Metrics4C，由+1Software Engineering (www.plus-one.com/Metrics4C_fact_sheet.html) 开发，计算各种体系结构度量、设计度量、面向代码度量以及面向项目度量。

Rational Rose，由Rational Corporation (www.rational.com) 开发。它是一种综合的UML建模工具集，为特征分析引入了大量的度量。

RSM，由M-Squared Technologies (msquaredtechnologies.com/m2rsm/index.html) 开发，它为C、C++和Java计算各种面向代码的度量。

Understand，由Scientific Toolwork, Inc. (www.scitools.com) 开发，它为各种编程语言计算面向代码的度量。

¹⁷ 参见15.4.3节中NOC和DIT的描述。

¹⁸ 这里提到的工具并不表示本书支持这些工具，而只是此类工具的举例。在多数情况下，工具的名字由各自的开发者注册为商标。

15.8 小结

软件度量为产品内部属性的质量评估提供了一种定量方法，从而可以使软件工程师在产品开发出来之前进行质量评估。度量为创建有效的分析模型、设计模型、可靠的代码和完全的测试提供必要的理解。

为在现实世界中有用，软件度量必须是简单的和可计算的、有说服力的、一致的和客观的。它应该是与程序设计语言无关的，且为软件工程师提供有效的反馈。

分析模型的度量侧重于分析模型的三个成分：功能、数据和行为。设计度量考虑体系结构、构件级设计和界面设计问题。体系结构设计度量考虑设计模型的结构方面；构件级设计度量通过为内聚性、耦合性和复杂性建立直接的测度，指示了模块质量情况；用户界面设计度量为GUI的易用性提供指示。

面向对象系统的度量侧重于能应用于类与设计特征的测量：局部化、封装、信息隐藏、继承及对象抽象技术。这些特征使类具有唯一性。

Halstead提供了一组令人感兴趣的源代码级度量。利用代码中出现的操作符和操作数的数量，为评估程序的质量开发了各种度量。

很少有产品度量是直接针对软件测试和维护提出的。然而，许多其他产品度量可用于指导测试过程，且作为评估计算机程序可维护性的机制。已提出大量的面向对象度量，它们可用于评估面向对象系统的可测试性。

参考文献

- [ALB79] Albrecht, A. J., "Measuring Application Development Productivity," *Proc. IBM Application Development Symposium*, Monterey, CA, October 1979, pp. 83–92.
- [ALB83] Albrecht, A. J., and J. E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Engineering*, November 1983, pp. 639–648.
- [BAS84] Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Software Engineering*, vol. SE-10, 1984, pp. 728–738.
- [BER95] Berard, E., "Metrics for Object-Oriented Software Engineering," an Internet posting on comp.software-eng, January 28, 1995.
- [BIE94] Bieman, J. M., and L. M. Ott, "Measuring Functional Cohesion," *IEEE Trans. Software Engineering*, vol. SE-20, no. 8, August 1994, pp. 308–320.
- [BIN94] Binder, R. V., "Object-Oriented Software Testing," *CACM*, vol. 37, no. 9, September 1994, p. 29.
- [BRI96] Briand, L. C., S. Morasca, and V. R. Basili, "Property-Based Software Engineering Measurement," *IEEE Trans. Software Engineering*, vol. SE-22, no. 1, January 1996, pp. 68–85.
- [CAR90] Card, D. N., and R. L. Glass, *Measuring Software Design Quality*, Prentice-Hall, 1990.
- [CAV78] Cavano, J. P., and J. A. McCall, "A Framework for the Measurement of Software Quality," *Proc. ACM Software Quality Assurance Workshop*, November 1978, pp. 133–139.
- [CHA89] Charette, R. N., *Software Engineering Risk Analysis and Management*, McGraw-Hill/Intertext, 1989.
- [CHI94] Chidamber, S. R., and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. Software Engineering*, vol. SE-20, no. 6, June 1994, pp. 476–493.
- [CHI98] Chidamber, S. R., D. P. Darcy, and C. F. Kemerer, "Management Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Trans. Software Engineering*, vol. SE-24, no. 8, August 1998, pp. 629–639.
- [CHU95] Churcher, N. I., and M. J. Shepperd, "Towards a Conceptual Framework for Object-Oriented Metrics," *ACM Software Engineering Notes*, vol. 20, no. 2, April 1995, pp. 69–76.

- [CUR80] Curtis, W., "Management and Experimentation in Software Engineering," *Proc. IEEE*, vol. 68, no. 9, September 1980.
- [DAV93] Davis, A., et al., "Identifying and Measuring Quality in a Software Requirements Specification," *Proc. First Intl. Software Metrics Symposium*, IEEE, Baltimore, MD, May 1993, pp. 141-152.
- [DEM81] DeMillo, R. A., and R. J. Lipton, "Software Project Forecasting," in *Software Metrics* (A. J. Perlis, F. G. Sayward, and M. Shaw, eds.), MIT Press, 1981, pp. 77-89.
- [DEM82] DeMarco, T., *Controlling Software Projects*, Yourdon Press, 1982.
- [DHA95] Dhami, H., "Quantitative Models of Cohesion and Coupling in Software," *Journal of Systems and Software*, vol. 29, no. 4, April 1995.
- [EJI91] Ejiogu, L., *Software Engineering with Formal Metrics*, QED Publishing, 1991.
- [FEL89] Felician, L., and G. Zalateu, "Validating Halstead's Theory for Pascal Programs," *IEEE Trans. Software Engineering*, vol. SE-15, no. 2, December 1989, pp. 1630-1632.
- [FEN91] Fenton, N., *Software Metrics*, Chapman and Hall, 1991.
- [FEN94] Fenton, N., "Software Measurement: A Necessary Scientific Basis," *IEEE Trans. Software Engineering*, vol. SE-20, no. 3, March 1994, pp. 199-206.
- [GRA87] Grady, R. B., and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
- [HAL77] Halstead, M., *Elements of Software Science*, North-Holland, 1977.
- [HAR98] Harrison, R., S. J. Counsell, and R. V. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Trans. Software Engineering*, vol. SE-24, no. 6, June 1998, pp. 491-496.
- [HET93] Hetzel, B., *Making Software Measurement Work*, QED Publishing, 1993.
- [IEE93] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1993.
- [IEE94] *Software Engineering Standards*, 1994 edition, IEEE, 1994.
- [IFP01] *Function Point Counting Practices Manual*, Release 4.1.1, International Function Point Users Group, 2001, available from <http://www.ifpug.org/publications/manual.htm>.
- [IFP03] Function Point Bibliography/Reference Library, International Function Point Users Group, 2003, available from <http://www.ifpug.org/about/bibliography.htm>
- [KOK95] Kokol, P., I. Rozman, and V. Venuti, "User Interface Metrics," *ACM SIGPLAN Notices*, vol. 30, no. 4, April 1995, can be downloaded from: <http://portal.acm.org/>.
- [KYB84] Kyburg, H. E., *Theory and Measurement*, Cambridge University Press, 1984.
- [LET03] Lethbridge, T., private communication of software metrics, June, 2003.
- [LON02] Longstreet, D., "Fundamental of Function Point Analysis," Longstreet Consulting, Inc, 2002, available at <http://www.ifpug.com/fpafund.htm>.
- [LOR94] Lorenz, M., and J. Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, 1994.
- [MCC76] McCabe, T. J., "A Software Complexity Measure," *IEEE Trans. Software Engineering*, vol. SE-2, December 1976, pp. 308-320.
- [MCC77] McCall, J., P. Richards, and G. Walters, "Factors in Software Quality," three volumes, NTIS AD-A049-014, 015, 055, November 1977.
- [MCC89] McCabe, T. J., and C. W. Butler, "Design Complexity Measurement and Testing," *CACM*, vol. 32, no. 12, December 1989, pp. 1415-1425.
- [MCC94] McCabe, T. J., and A. H. Watson, "Software Complexity," *Crosstalk*, vol. 7, no. 12, December 1994, pp. 5-9.
- [NIE94] Nielsen, J., and J. Levy, "Measuring Usability: Preference vs. Performance," *CACM*, vol. 37, no. 4, April 1994, pp. 65-75.
- [ROC94] Roche, J. M., "Software Metrics and Measurement Principles," *Software Engineering Notes*, ACM, vol. 19, no. 1, January 1994, pp. 76-85.
- [SEA93] Sears, A., "Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout," *IEEE Trans. Software Engineering*, vol. SE-19, no. 7, July 1993, pp. 707-719.
- [SHE98] Sheppard, M., *Goal, Question, Metric*, 1998, available at <http://dec.bournemouth.ac.uk/ESERG/mshepperd/SEMGQM.html>.
- [SOL99] van Solingen, R., and E. Berghout, *The Goal/Question/Metric Method*, McGraw-Hill, 1999.
- [UEM99] Uemura, T., S. Kusumoto, and K. Inoue, "A Function Point Measurement Tool for UML Design Specifications," *Proc. of Sixth International Symposium on Software Metrics*, IEEE, November 1999, pp. 62-69.

- [USA87] *Management Quality Insight*, AFCSP 800-14 (U.S. Air Force), January 20, 1987.
- [WHI97] Whitmire, S., *Object-Oriented Design Measurement*, Wiley, 1997.
- [WIL93] Wilde, N., and R. Huitt, "Maintaining Object-Oriented Software," *IEEE Software*, January 1993, pp. 75-80.
- [ZUS90] Zuse, H., *Software Complexity: Measures and Methods*, DeGruyter, 1990.
- [ZUS97] Zuse, H., *A Framework of Software Measurement*, DeGruyter, 1997.

习题与思考题

- 15.1 开发一个软件工具用以计算某编程语言模块的环复杂度，可以任选一种语言。
- 15.2 McCall的质量因素是在20世纪70年代提出的。自它们提出以来，几乎计算的各个方面改动都很大，而且McCall的因素仍然适用于现代软件。你能根据这个事实得出什么结论吗？
- 15.3 试着从日常生活中提出一些测度或度量，它们不符合15.2.5节定义的有效的软件度量属性。
- 15.4 类X具有12个操作。在面向对象系统中，所有操作的环复杂度已计算出来，模块复杂度的平均值为4。对于类X，操作1至12的复杂度分别为5,4, 3,3, 6,8, 2,2, 5,5, 4,4。计算每个类的加权方法。
- 15.5 一个系统具有12个外部输入，24个外部输出，30个字段不同的外部查询，管理4个内部逻辑文件，与6个不同的遗产系统相连接（6EIF）。所有这些数据属于平均复杂度，且整个系统相对简单。计算该系统的FP。
- 15.6 测量理论是与软件度量密切相关的高级主题。利用[ZUS97]、[FEN91]、[ZUS90]和[KYB84]或其他资源，写一篇短文概括测量理论的主要原则。个人项目：关于这个主题准备一个演讲并在班上交流。
- 15.7 为什么不能为程序复杂性或程序质量开发一种单一的、全包容的度量？
- 15.8 某个主要的信息系统有1140个模块，其中96个模块完成控制和协调功能，490个模块的功能依赖于前面的处理。该系统大约处理220个对象，每个对象平均有三个属性。存在140个唯一的数据库项和90个不同的数据库段。且600个模块有单一的入口和出口点。计算这个系统的DSQI值。
- 15.9 一个遗产系统有940个模块，最新版本中需要变更其中的90个模块，此外，加入40个新模块，移除12个旧模块。计算这个系统的软件成熟度指标。
- 15.10 开发一个小型软件工具，用它对你选择的编程语言源代码进行Halstead分析。
- 15.11 软件系统X有24个功能需求和14个非功能需求。什么是需求的确定性及完备性？

496

推荐读物与阅读信息

关于软件度量方面有大量的书籍，其中大部分都侧重于过程与项目度量，而不是产品度量。Kan (《Metrics and Models in Software Quality Engineering》, Addison-Wesley, second edition, 2002)、Fenton与Pfleeger (《Software Metrics: A Rigorous and Practical Approach》, Brooks-Cole Publishing, 1998) 以及Zuse[ZUS97]对产品度量进行了全面的讨论。

Card与Glass[CAR90]、Zuse[ZUS90]、Fenton[FEN91]、Ejiogu[EJI91]、Moeller与Paulish (《Software Metrics》, Chapman and Hall, 1993) 和Hetzl[HET93]都比较详细地讨论了产品度

- [USA87] *Management Quality Insight*, AFCSP 800-14 (U.S. Air Force), January 20, 1987.
- [WHI97] Whitmire, S., *Object-Oriented Design Measurement*, Wiley, 1997.
- [WIL93] Wilde, N., and R. Huitt, "Maintaining Object-Oriented Software," *IEEE Software*, January 1993, pp. 75-80.
- [ZUS90] Zuse, H., *Software Complexity: Measures and Methods*, DeGruyter, 1990.
- [ZUS97] Zuse, H., *A Framework of Software Measurement*, DeGruyter, 1997.

习题与思考题

- 15.1 开发一个软件工具用以计算某编程语言模块的环复杂度，可以任选一种语言。
- 15.2 McCall的质量因素是在20世纪70年代提出的。自它们提出以来，几乎计算的各个方面改动都很大，而且McCall的因素仍然适用于现代软件。你能根据这个事实得出什么结论吗？
- 15.3 试着从日常生活中提出一些测度或度量，它们不符合15.2.5节定义的有效的软件度量属性。
- 15.4 类X具有12个操作。在面向对象系统中，所有操作的环复杂度已计算出来，模块复杂度的平均值为4。对于类X，操作1至12的复杂度分别为5,4, 3,3, 6,8, 2,2, 5,5, 4,4。计算每个类的加权方法。
- 15.5 一个系统具有12个外部输入，24个外部输出，30个字段不同的外部查询，管理4个内部逻辑文件，与6个不同的遗产系统相连接（6EIF）。所有这些数据属于平均复杂度，且整个系统相对简单。计算该系统的FP。
- 15.6 测量理论是与软件度量密切相关的高级主题。利用[ZUS97]、[FEN91]、[ZUS90]和[KYB84]或其他资源，写一篇短文概括测量理论的主要原则。个人项目：关于这个主题准备一个演讲并在班上交流。
- 15.7 为什么不能为程序复杂性或程序质量开发一种单一的、全包容的度量？
- 15.8 某个主要的信息系统有1140个模块，其中96个模块完成控制和协调功能，490个模块的功能依赖于前面的处理。该系统大约处理220个对象，每个对象平均有三个属性。存在140个唯一的数据库项和90个不同的数据库段。且600个模块有单一的入口和出口点。计

第三部分

应用Web工程

本书的这一部分，将了解开发高质量Web应用的原理、概念和方法。接下来的几章会介绍以下几方面的问题：

- WebApp（Web应用）是否不同于其他类型的软件？
- 什么是Web工程？它采用软件工程实践的哪些元素？
- Web工程过程的元素有哪些？
- 如何系统地阐述并计划一个Web工程项目？
- 如何对WebApp需求进行分析和建模？
- 指导从业人员进行WebApp设计的概念和原理是什么？
- 如何为WebApp建立体系结构、界面和导航设计？
- 应用哪些构造技术实现设计模型？
- 哪些测试概念、原理和方法适用于Web工程？

一旦解决了这些问题，也就为开发高质量的Web应用做好了较充分的准备。

[499]

第16章 Web工程

要点浏览

概念：基于Web的系统和应用（WebApp）向广大的最终用户发布一组复杂的内容和功能。Web工程（WebE）是用来创建高质量WebApp的过程。WebE不是软件工程的完全复制，但是，它借用了很多软件工程的基本概念和原理。除此以外，WebE过程也强调与软件工程相似的技术和管理活动。虽然在这些活动的管理方式方面存在微妙的不同，但是，指导开发基于计算机系统的规范化方法的主要思想是相同的。

人员：由Web工程师和非技术性内容的开发者负责创建WebApp。

重要性：随着WebApp越来越多地被集成到小公司和大公司的业务策略中（例如，电子商务），对构造可靠的、可用的和自适应的系统的需求变得越来越重要。这

就是在WebApp开发中使用规范化方法的必要性原因。

步骤：同任何工程学科一样，WebE使用与特定的策略、技巧和方法相调和的一般途径。在WebE过程中，首先对WebApp要解决的问题进行系统地阐述；然后，对WebE项目进行策划，并为WebApp的需求和设计建模；使用与Web相关的特定技术和工具构造系统；将WebApp发送给最终用户，同时使用技术标准和商业标准对其进行评估。因为WebApp不断演化，必须建立配置控制、质量保证及运行支持机制。

工作产品：将产生一系列的WebE工作产品。最终的产品是可运行的WebApp。

质量保证措施：直到最终用户使用WebApp，否则很难确定。然而，可以用SQA实践来评估WebE模型的质量、全部系统内容和功能、可用性、性能及安全性。

关键概念

基本问题

最佳实践

过程框架

质量标准

WebApp

属性

分类

Web工程

方法

过程

工具

万维网（Word Wide Web）和因特网（Internet）无疑是计算史上最重要的发展。这些技术将我们（最终将有数十亿的跟随者）带入了信息时代。在21世纪初期，网络已经成为日常生活中不可缺少的部分。

我们中的很多人还能记起没有Web的世界，让我们回顾一下另一个时代（软件的早期时代）技术无序增长的情况。那是一个缺少规范的时代，但却拥有巨大的热情和创造；那是一个程序员经常胡乱拼凑系统（某些好，某些糟糕）的时代，普遍存在的态度似乎是“快速完成系统并投入运行，随着开发的进展，我们会清楚将要开发的系统（或更好地了解我们真正想要建造什么）”。这些听起来是不是很熟悉？

在《IEEE Software》[PRE98]中公布的虚拟圆桌会议上，我明确表示出我是如何看待Web工程的：

对我而言，几乎任何重要的产品或系统都值得应用工程学。在开始建造以前，最好充分地理解问题，设计一个可行的解决方案，采用可靠的方式来实现，并进行充分的测试。可能还应该在工作中控制对系统的变更，并用某种机制保证最终结果

的质量。很多Web开发者不同意这种观点，他们认为他们的世界确实是不同了，传统的软件工程方法不可能简单地应用。

500

这给我们提出了一个关键问题：能否将软件工程的原理、概念和方法应用到Web开发中？很多是可以的，但应用时可能需要某些不同的调整。

但是，如果Web开发坚持使用非规范化方法会怎样呢？在缺少开发基于Web的系统的规范化过程时，我们越来越担心实现这些系统的成功开发、部署和维护会面临严重的问题。从本质上讲，当我们迈入新世纪时，今天正在创建的应用系统的基础设施可能会导致“混乱的Web”，这就意味着开发不良的Web应用具有很高的失败率。更糟糕的是，当基于Web的系统变得更加复杂时，一个地方的失效会将各种问题传播到很多地方。当这个问题发生时，人们将会动摇对整个Internet的信心。同样糟糕的是，这可能会导致不必要的和构想拙劣的政府法规，对这些独特的技术产生不可挽回的损害。

为了避免混乱的Web，并在开发和应用大型复杂的基于Web的系统中取得更大成功，我们在开发、部署和评估基于Web的系统和应用时，更加需要规范化途径及新的方法和工具。这些方法和技术必须充分考虑到新媒介的特点、操作环境和用户的多样性，这些都对基于Web的应用系统的开发提出了更多的挑战。

Web工程采用“可靠科学的原则、工程化的原则和管理原则，以及规范、系统的手段，以期获得高质量的基于Web的系统和应用的成功开发、部署和维护。”[MUR99]

16.1 基于Web的系统及应用的特点

WWW的早期（大约从1990年到1995年），“Web站点”仅包含链接在一起的少量超文本文件，这些文件使用文本和有限的图标来表示信息。随着时间的推移，一些开发工具（例如，XML、Java）使HTML得到扩展，这些开发工具使得Web工程师在向客户提供信息的同时也能提供计算功能。基于Web的系统和应用¹（我们将这些总称为WebApp）诞生了。今天，WebApp已经发展成为成熟的计算工具，这些工具不仅可以为最终用户提供独立的功能，而且已经同公司数据库和业务应用集成在一起了。

“当我们看到任何程度的稳定性时，Web就会变得完全不同了。” ——Louis Monier

毫无疑问WebApp不同于第1章中讨论的很多其他类型的计算机软件。Powell总结了主要的不同，他认为基于Web的系统“涉及印刷出版和软件开发之间、市场和计算之间、内部通信和外部关系之间以及艺术和技术间的混合”[POW98]。在绝大多数WebApp中都可以遇到下列属性：

网络密集性 (network intensiveness)：WebApp驻留在网络上，服务于不同客户群体的需求。WebApp可以驻留在Internet上（这就使得世界范围内的通信成为可能）。或者说，一个应用可以放置在内联网（实现组织范围内的通信）或外联网（网际间通信）上。

并发性 (concurrency)：在同一时间可能有大量用户使用WebApp。很多情况下，最终用

¹ 在本章中，“Web应用”（WebApp）这个术语包含了很多事物，从一个简单的帮助消费者计算汽车租借费用的网页，到为商业人员和度假者提供全套旅游服务的大型复杂的Web站点。其中包括一些完整的Web站点、Web站点的专门功能以及在Internet、Intranet或ExtraNet上的信息处理应用。



在第1章介绍的任何一个软件领域中的传统应用都能够表现出这些属性，然而，WebApp几乎总是具有这些属性。

502

户的使用模式存在很大差异。

无法预计的负载量 (unpredictable load)：WebApp的用户数量每天都可能会有数量级的变化。周一显示有100个用户使用这个系统，周二就可能会有10 000个用户。

性能 (performance)：如果一位WebApp用户必须等待很长时间（访问、服务器端处理、客户端格式化及显示），该用户就会转向其他地方。

可得性 (availability)：尽管期望百分之百的可得性是不切实际的，但是大多数的WebApp用户通常要求“24/7/365”（全天候）的可访问性。澳大利亚或亚洲的用户可能在北美传统的国内应用软件脱机维护时要求访问。

数据驱动 (data driven)：许多WebApp的主要功能是使用超媒体向最终用户提供文本、图片、音频及视频内容。除此以外，WebApp一般用来访问那些存储在数据库中的信息，这些数据库最初并不是基于Web的环境的整体组成部分（例如，电子商务或金融应用）。

内容敏感性 (content sensitive)：内容的质量和艺术性仍然在很大程度上决定了WebApp的质量。

持续演化 (continuous evolution)：传统的应用软件是随一系列规划好的时间间隔发布而演化的，而Web应用则持续地演化。对某些WebApp（特别是WebApp的内容）而言，以分钟为单位进行更新，或者对每个请求进行独立运算是司空见惯的事。有人认为，WebApp的持续演化使得在其上完成的工作类似于园艺。Lowe[LOW99]在讨论这个问题时写道：

工程是采用一致的、科学的方法，在特定的实践环境中实施系统或应用的开发和运行。Web站点开发通常更多地是创建一个基础设施（布置好花园），然后“照管”在该花园中生长和开花的“信息”。随着时间的推移，花园（即Web站点）将持续地演化、变化和生长。好的初始体系结构将允许Web站点以一种可控制的和一致的方式增长……

通过不断地“照看”和“饲养”，使一个Web站点成长（在健壮性和重要性方面），但是，和花园不同的是，Web应用必须服务于（并适应）比园丁更多的需求。

即时性 (immediacy)：尽管即时性（也就是将软件尽快推向市场的迫切需要）是很多应用领域的特点，然而将WebApp投入市场可能只是几天或几周的事情²。Web工程师们必须使用经过修改的计划、分析、设计、实现和测试的方法以满足WebApp开发所要求的紧迫的时间进度安排。

保密性 (security)：由于WebApp是通过网络访问来使用的，要限制访问的最终用户的数量，即使能也非常困难。为了保护敏感的内容，并提供保密的数据传输模式，在支持WebApp的整个基础设施上和应用本身内部必须实现较强的保密措施。

美学性 (aesthetics)：WebApp具有吸引力的一个不可否认的部分是其观感。当要面向市场推销产品或想法时，与技术设计相比，美学可能同样事关该应用的成功。

503

这些一般属性是所有WebApp都具有的，但其影响程度会有所不同。

² 应用现代工具，我们只需几小时就能完成一个复杂网页的开发。



在第1章介绍的任何一个软件领域中的传统应用都能够表现出这些属性，然而，WebApp几乎总是具有这些属性。

502

户的使用模式存在很大差异。

无法预计的负载量 (unpredictable load)：WebApp的用户数量每天都可能会有数量级的变化。周一显示有100个用户使用这个系统，周二就可能会有10 000个用户。

性能 (performance)：如果一位WebApp用户必须等待很长时间（访问、服务器端处理、客户端格式化及显示），该用户就会转向其他地方。

可得性 (availability)：尽管期望百分之百的可得性是不切实际的，但是大多数的WebApp用户通常要求“24/7/365”（全天候）的可访问性。澳大利亚或亚洲的用户可能在北美传统的国内应用软件脱机维护时要求访问。

数据驱动 (data driven)：许多WebApp的主要功能是使用超媒体向最终用户提供文本、图片、音频及视频内容。除此以外，WebApp一般用来访问那些存储在数据库中的信息，这些数据库最初并不是基于Web的环境的整体组成部分（例如，电子商务或金融应用）。

内容敏感性 (content sensitive)：内容的质量和艺术性仍然在很大程度上决定了WebApp的质量。

持续演化 (continuous evolution)：传统的应用软件是随一系列规划好的时间间隔发布而演化的，而Web应用则持续地演化。对某些WebApp（特别是WebApp的内容）而言，以分钟为单位进行更新，或者对每个请求进行独立运算是司空见惯的事。有人认为，WebApp的持续演化使得在其上完成的工作类似于园艺。Lowe[LOW99]在讨论这个问题时写道：

工程是采用一致的、科学的方法，在特定的实践环境中实施系统或应用的开发和运行。Web站点开发通常更多地是创建一个基础设施（布置好花园），然后“照管”在该花园中生长和开花的“信息”。随着时间的推移，花园（即Web站点）将持续地演化、变化和生长。好的初始体系结构将允许Web站点以一种可控制的和一致的方式增长……

通过不断地“照看”和“饲养”，使一个Web站点成长（在健壮性和重要性方面），但是，和花园不同的是，Web应用必须服务于（并适应）比园丁更多的需求。

即时性 (immediacy)：尽管即时性（也就是将软件尽快推向市场的迫切需要）是很多应用领域的特点，然而将WebApp投入市场可能只是几天或几周的事情²。Web工程师们必须使用经过修改的计划、分析、设计、实现和测试的方法以满足WebApp开发所要求的紧迫的时间进度安排。

保密性 (security)：由于WebApp是通过网络访问来使用的，要限制访问的最终用户的数量，即使能也非常困难。为了保护敏感的内容，并提供保密的数据传输模式，在支持WebApp的整个基础设施上和应用本身内部必须实现较强的保密措施。

美学性 (aesthetics)：WebApp具有吸引力的一个不可否认的部分是其观感。当要面向市场推销产品或想法时，与技术设计相比，美学可能同样事关该应用的成功。

503

这些一般属性是所有WebApp都具有的，但其影响程度会有所不同。

² 应用现代工具，我们只需几小时就能完成一个复杂网页的开发。

- (1) 包含变化；(2) 鼓励独创性、开发团队的独立性以及同WebApp的共利益者密切沟通；
- (3) 采用小的开发团队构造系统；(4) 强调使用短开发周期演化或增量开发 [MCD01]。

16.2.2 方法

WebE方法大体上包括一组技术性任务，这些任务使Web工程师能够理解并把握WebApp的特点，从而开发出高质量的WebApp。WebE方法（在第18~20章详细讲述）按以下方式分类：



值得注意的是，很多WebE方法直接采纳了软件工程中的对应方法；有些正处于成型阶段。其中有些方法会继续使用，有些会由于出现了更好的方法而被淘汰。

505

沟通方法——定义方法以方便Web工程师和其他WebApp共利益者沟通（例如，最终用户、业务客户、问题域专家、内容设计者、团队领袖、项目管理者）。在需求收集及任何时候评估WebApp增量时，沟通技术都显得尤为重要。

需求分析方法——为理解下面的问题提供了基础：WebApp要发布的内容，为最终用户提供功能，以及当通过WebApp导航时各类用户所需的交互模式。

设计方法——包括一系列的设计技术来描述WebApp的内容、应用和信息体系结构、界面设计和导航结构。

测试方法——包括对内容和设计模型的正式技术评审，以及一组针对构件级和体系结构问题的测试技术，包括导航测试、可用性测试、保密性测试和配置测试。

值得注意的是，尽管WebE方法采用了许多与本书第二部分描述的软件工程方法相同的基本概念和原理，但是，必须对这些分析、设计和测试技巧进行适应性修改，才能满足WebApp的特殊特点。

除了上面简要描述的技术性方法外，一系列的普适性活动（和相关的方法）对于Web工程的成功也是很重要的。其中包括项目管理技术（例如，评估、进度安排、风险分析）、软件配置管理技术及评审技术。

16.2.3 工具与技术

WebRef

WebE技术的优秀资源可以在webdeveloper.com和www.eborcom.com/webmaker找到。

随着WebApp逐渐成熟及普遍，在过去的10年中，已经开发出大量的工具和技术。这些技术包括许多内容描述和建模语言（例如，HTML、VRML、XML）、编程语言（例如，Java）、基于构件的开发资源（例如，CORBA、COM、ActiveX、.NET）、浏览器、多媒体工具、网站创建工具、数据库连接工具、保密工具、服务器和服务器实用程序、网站管理及分析工具。

对Web工程工具和技术的全面讨论不在本书的范围内，感兴趣的读者可以访问下面的一些网站：“Web Developer’s Virtual Encyclopedia” (www.wdlv.com)，“WebDeveloper” (www.webdeveloper.com)，“Developer Shed” (www.devshed.com)，“Webknowhow.net” (www.webknowhow.net)，或者“WebReference” (www.webreference.com)。

16.3 Web工程过程

基于Web的系统和应用的特点对所选的WebE过程有着深远的影响。在第3章中我们曾经提到，一个软件工程要基于待开发软件的特点选择过程模型，这对于Web工程师来说也同样成立。

如果说即时性和持续演化是WebApp的重要特点的话，Web工程团队可能选择快速发布

WebApp的敏捷过程模型（第4章）。另一方面，如果WebApp要开发很长一段时间（例如，重要的电子商务应用），就有可能选择增量过程模型（第3章）。

“Web开发还处于青年时期……正如大多数的青年人一样，在他试图离开他的父母时，他想要像成年人那样被接受。如果Web开发要完全成长起来，就必须向更老练的软件开发世界学习。”
——Doug Wallace等

此领域中应用系统的网络密集特性意味着应用系统要面对大量的、多种多样的用户（因此产生对需求导出和建模的特殊要求），需要采用高度特殊化的应用结构（因此产生设计要求）。由于WebApp通常是内容驱动的，并强调美学，因此在WebE过程中可能需要有并行的开发活动，并包括由技术人员和非技术人员构成的开发团队（例如，广告撰写人，图形设计者）。

506

16.3.1 定义框架

我们在第4章提到的任何一个敏捷过程模型（例如，极限编程、适应性软件开发、SCRUM）都可以成功地应用于WebE过程。这里提到的过程框架是我们在第4章中讨论的原理和思想的结合体。

为了取得有效性，任何一个工程过程必须是自适应的。也就是说，这个项目团队的组织、团队成员间交流的模式、工程活动和需要实现的任务、收集和创建的信息以及用来生产高质量产品的方法，都必须适合做这些工作的人；必须符合项目进度安排和限制；必须适合要解决的问题。在我们为WebE定义一个过程框架之前，必须认识到：



WebE过程模型
以三点为基础：
增量发布、不断
变更和短期限。

1. WebApp通常是增量交付的。即开发并发布每个增量时，框架活动会重复出现。
2. 变更会频繁发生。对所交付增量的评估或业务环境发生变化都会引起变更发生。

3. 期限短。这样就减轻了庞大的工程文档的创建和评估，但是这并不排除一个简单的事实，那就是必须按照某种方式记录重要的分析、设计和测试。

除此以外，应该应用那些作为“敏捷软件开发宣言”（第4章）组成部分而定义的原则。然而，这些原则并不是十条戒律。采用这些原则的思想，而并不要求严格遵守宣言，这样做有时是合理的。

考虑到这些问题，下面我们讨论通用过程框架（第2章讨论过的）内的WebE过程。



通用过程模型
(在第2章介绍) 也
适用于Web工程。

客户沟通：在WebE过程中，客户沟通的特点在于它有两个主要任务：业务的分析和表达。业务分析为WebApp定义业务或组织的环境。除此以外，确定共利益者，预测业务环境或业务需求方面的可能变更，定义WebApp同其他业务应用、数据库和功能的集成。表达是一个涉及所有共利益者的需求收集活动。其目的是使用收集到的最有价值的信息描述那些WebApp要解决的问题（以及WebApp的基本需求）。除此之外，要力图识别出那些不确定的领域和可能出现变更的地方。

507

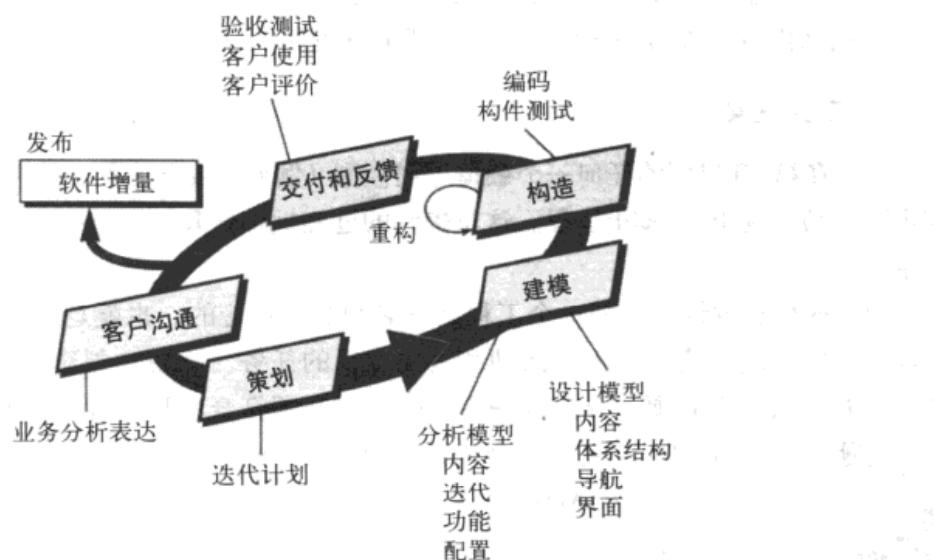
策划：需要为WebApp增量创建项目计划。这个计划包括任务定义和WebApp增量开发所需的时间段（通常以周为单位计算）的时间进度表。

建模：传统软件工程的分析和设计任务也适用于WebApp的开发，能够合并及融入到WebE建模活动中（第18、19章）。其目的是开发用来定义需求的“快速的”分析和设计模型，同时提供一个满足需求的WebApp。

构造：应用WebE工具和技术构造已建模的WebApp。一旦构造出WebApp的增量，就会执行一系列快速测试，以确保发现设计方面（即内容、体系结构、界面、导航）的错误，并对WebApp的其他特性进行附加测试。

部署：需要对WebApp的运行环境进行配置，交付给最终用户后，评估阶段就开始了。将评估结果反馈给WebE团队，必要时修改增量。

这5个WebE框架活动应用在一个增量过程流中，如图16-1所示。



508

图16-1 WebE过程

INFO

Web工程——基本问题

任何产品的工程都包括一些微妙的地方，如果没有相当工作经验是不容易看出来的。由于WebApp的特性，使得Web工程师不得不在早期框架活动中解决各种各样的问题。在表达阶段涉及与业务需求和产品目标相关的策略问题，在分析建模阶段必须考虑与特征和功能相关的需求问题，在设计模型演化中必须考虑与WebApp体系结构、界面特性及导航等广泛的相关设计问题。最后，有关用户与WebApp的实际交互方式的一系列人的问题会经常遇到。

随着分析和设计的进展，有些一定要考虑的问题，Susan Weinshenk[WEI02]对其给出了建议。这里记录的是一小部分（适合的部分）：

- 一个网站的主页有多重要？它应该包括有用的信息，还是应该提供一个简单的链表，引导用户进入更深一层查看更详细的内容？
- 什么是最有效的网页布局（例如，菜单在上面、左面、还是右面）？它是否因正在开发的WebApp的类型的不同而不同？
- 选择哪种媒体具有最佳效果？图示是不是比文本更有效？视频（或音频）是不是有效选择？什么时候应该选择不同媒体？

- 当用户在查找信息时，我们需要用户做多少工作？人们希望点击多少次？
- 当WebApp很复杂时，导航帮助有多重要？
- 用户能够容忍多复杂的表格输入？如何实现快速表格输入？
- 查询能力有多重要？使用直接浏览的用户比例是多少，使用特殊查找的用户比例是多少？按照一定方式（设想从外部资源提供一个链接）构建每个页面有多重要？
- 是否需要将WebApp设计成能够被身体或其他方面有残疾的人使用的方式？

对于上面这些问题，我们没有绝对正确的答案，不过，在WebE过程中，这些问题必然会涉及到。我们会在第17~20章中考虑可能的答案。

16.3.2 精化框架

我们已经讲到WebE过程模型必须是适应性强的。也就是说，精化每个框架活动所需要的工程任务的定义留给WebE团队决定。有些情况下，可以非正式地执行框架活动；而有些情况下，需要明确定义一系列的任务，并由团队成员执行。任何情况下，团队都有责任在分配的时间内开发出高质量的WebApp增量。

值得强调的是，可以基于问题、产品、项目及WebE团队成员的特点，对与WebE框架活动相关的任务进行修改、排除或扩充。

“我们当中的很多人相信软件开发的最佳实践是很有用的，并且值得去实现。也有一些人认为最佳实践是学院派感兴趣的事情，并不适合真实的世界。非常感谢！”

——Warren Keuffel

509

16.4 Web工程的最佳实践

是否每个WebApp开发者都使用在16.3节中定义的WebE过程框架和任务集？可能不是。有时在巨大的时间压力下，Web工程团队会设法走捷径（即使这些做法我们并不提倡，这样做会导致更多的开发工作量，而不是更少）。但是，如果要建立符合行业质量标准的WebApp，就应该应用一组基本的最佳实践（取自本书第二部分讨论的软件工程实践）。



1. 即使WebApp的一些细节是模糊的，也需要花时间去理解业务要求和产品目标。很多WebApp开发商错误地认为，模糊的需求（非常普遍）使他们不需要确定将要开发的系统具有合理的商业用途。最终的结果是（通常是）好的技术工作为了错误的理由、错误的用户构造了错误的系统。如果共利益者不能清楚地说明WebApp的业务需求，就需要加倍小心，只有共利益者能够确定产品（WebApp）的一系列明确的目标，才继续开发。

2. 使用基于场景的方式来描述用户如何同WebApp交互。共利益者一定要坚持开发用例（在本书第二部分讨论过），通过用例来反映各种不同角色如何同WebApp交互。然后这些场景可被用于：(1) 项目计划制定和追踪；(2) 指导分析和设计建模；(3) 作为测试设计的重要输入。

3. 制定一个项目计划，即使这个计划很简短。将计划建立在所有共利益者都能接受的预先确定的过程框架之上。由于项目时间很短，进度表中时间

段的任务安排应该精细；也就是，很多情况下工程的计划和追踪都是以天为单位的。

4. 不管将要构造什么，都应该花一些时间建模。通常，在WebE工程中不建立全面的分析和设计模型。然而，UML类、顺序图及其他UML符号表示法（例如，状态图）对于问题的理解可以提供非常宝贵的帮助。

5. 对模型的一致性和质量进行评审。正式技术评审（第26章）应该贯穿整个WebE项目。在评审上花一些时间是值得的，因为它通常能避免重复工作，并得到高质量的WebApp，从而提高客户的满意度。

6. 使用那些能够让你尽可能多地使用可复用构件来构造系统的工具和技术。有很多WebApp
[510] 工具可以用来完成WebApp构造的各个方面。其中，很多工具容许Web工程师使用可复用构件来建造应用程序的重要部分。

7. 不要依赖早期用户的意見来调试WebApp——设计一个全面的测试，并在发布系统之前执行这些测试。WebApp的用户通常只给WebApp一次机会，一旦WebApp执行错误，用户就会离开，永远不会再回来。正是因为这个原因，“先测试，后部署”是高于一切的指导原则，即使必须拖延最后的部署期限。

INFO

WebApp的质量准则或质量指导原则

WebE一直努力开发高质量的WebApp。但是，这里的“质量”是指什么？要获得高质量，我们应该遵循什么指导原则？Quibeldey-Cirkel[QUIO1]在他的Web站点质量保证方面的论文中，给出了阐述这个问题的较全面的一组在线资源：

- W3C: Style Guide for Online Hypertext (www.w3.org/Provider/Style)
- The Sevloid Guide to Web Design (www.sev.com.au/webzone/design/guide.asp)
- Web Pages That Suck (www.webpagesthatsuck.com/index.html)
- Resources on Web Style (www.westegg.com/unmaintained/badpages)
- Gartner's Web Evaluation Tool (www.gartner.com/ebusiness/website-ings)
- IBM Corp: Web Guidelines (www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/572)
- World Wide Web Usability (ijhcs.open.ac.uk)
- Interface Hall of Shame (www.iarchitect.com/mshame.htm)
- Art and the Zen of Web Sites (www.tlc-systems.com/webtips.html)
- Designing for the Web: Empirical Studies (www.microsoft.com/usability/webconf.htm)
- Nielsen's useit.com (www.useit.com)
- Quality of Experience (www.qualityofexperience.org)
- Creating Killer Web Sites (www.killersites.com/core.html)
- All Things at Web (www.pantos.org/atw)
- SUN's New Web Design (www.sun.com/980113/sunonnet)
- Tognazzini, Bruce: Homepage (www.asktog.com)
- Webmonkey (hotwired.lycos.com/webmonkey/design/?tw=design)
- World's Best WebSites (www.worldbestwebsites.com)
- Yale University: Yale Web-Style Guide (Info.med.yale.edu/caim/manual)

16.5 小结

基于Web的系统和应用的影响无疑是计算史上一个最重要的事件。随着WebApp重要性的增长，规范化的Web工程方法（改自软件工程的原理、概念、过程和方法）已经开始发展。

WebApp不同于其他种类的计算机软件，WebApp是网络密集的、内容驱动的及持续演化的。其他区别因素还有：驱动WebApp发展的即时性，在WebApp运行中对保密性的强烈要求，对美学的要求，以及功能性内容的发送。同其他类型的软件一样，可以使用不同的质量标准来评估WebApp，包括可用性、功能性、可靠性、高效性、可维护性、保密性、实用性、可伸缩性及投入市场时间。511

WebE可以被描述成三层——过程、方法和工具/技术。WebE过程采用了敏捷开发思想，此开发思想强调“瘦”的、可以增量交付所建造系统的工程化方法。通用过程框架（沟通、计划、建模、构造及部署）同样适用于WebE。这些框架活动被精炼为一组适合每个项目要求的WebE任务集合。同那些在软件工程工作中应用的普适性活动类似，这样的活动（SQA、SCM、项目管理）也可应用于所有的WebE项目。

参考文献

- [AOY98] Aoyama, M., "Web-Based Agile Software Development," *IEEE Computer*, November/December, 1998, pp. 56–65.
- [DAR99] Dart, S., "Containing the Web Crisis Using Configuration Management," *Proc. First ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999. (*The Proceedings of the First ICSE Workshop on Web Engineering* are published on-line at <http://fistserv.macarthur.uws.edu.au/san/icse99-WebE/ICSE99-WebE-Proc/default.htm>).
- [FOW01] Fowler M., and J. Highsmith, "The Agile Manifesto," *Software Development Magazine*, August 2001, <http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm>.
- [MCD01] McDonald, A., and R. Welland, *Agile Web Engineering (AWE) Process*, Department of Computer Science, University of Glasgow, Technical Report TR-2001-98, 2001, downloadable from <http://www.dcs.gla.ac.uk/~andrew/TR-2001-98.pdf>.
- [MUR99] Murugesan, S., *WebE Home Page*, <http://fistserv.macarthur.uws.edu.au/san/WebEHome>, July, 1999.
- [NOR99] Norton, K., "Applying Cross Functional Evolutionary Methodologies to Web Development," *Proc. First ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999.
- [POW98] Powell, T. A., *Web Site Engineering*, Prentice-Hall, 1998.
- [PRE98] Pressman, R. S. (moderator), "Can Internet-Based Applications Be Engineered?" *IEEE Software*, September 1998, pp. 104–110.
- [QUI01] Quibeldey-Cirkel, K., "Checklist for Web Site Quality Assurance," *Quality Week Europe*, 2001, downloadable from www.fbi.fh-darmstadt.de/~quibeldey/Projekte/QWE2001/Paper_Quibeldey_Cirkel.pdf.
- [WEI02] Weinschenk, S., "Psychology and the Web: Designing for People," 2002, <http://www.weinschenk.com/learn/facts.asp>.

习题与思考题

- 16.1 用一个实际Web站点作为例子，列举WebApp“内容”的不同表示。
- 16.2 研究并撰写一篇2~3页的文章，对16.2.3节提到的某一项技术进行总结。
- 16.3 如何判断Web站点的“质量”？请按优先级次序列出10个你认为最重要的质量特性。
- 16.4 是否有其他的一般属性以区分WebApp和更传统的软件应用？设法找出2~3个。512
- 16.5 回顾第4章提到的“敏捷软件开发宣言”。其中12条原则中的哪些可以很好地应用到2年

期项目（包括几十名员工）？此项目是为一家汽车公司构建一个重要的电子商务系统。这12条原则中的哪些可以很好地应用到2个月的项目？此项目是为小型房地产公司建立一个信息站点。

- 16.6 如果开发一个新的电子商务应用，直接通过Web销售移动电话和提供服务，列出可能出现的“风险”。
- 16.7 回顾第3章和第4章中描述的软件工程过程，是否有另外的一个或一些过程（除了敏捷过程模型）可以应用到Web工程中？如果有，指出是哪个或哪些，并说明为什么？

推荐读物与阅读信息

近几年来讨论Web工程主题的书已经出版了数百本，但很少有讲述Web工程的所有方面的书。Sarukkai（《Foundations of Web Technology》，Kluwer Academic Publishers，2002）对WebE所需的技术进行了有价值的汇编。Murugusan和Deshpande（《Web Engineering：Managing Diversity and Complexity of Web Development》，Springer-Verlag，2001）编辑出版了一本WebE方面的有用论文集。关于Web工程和Web信息系统工程的国际会议学报每年由IEEE Computer Society Press发行。

Flor（《Web Business Engineering》，Addison-Wesley，2000）讨论了能让Web工程师更清楚地理解客户需求的业务分析过程和相关内容。Bean（《Engineering Global E-Commerce Sites》，Morgan Kaufmann，2003）为全球WebApp的开发提出了指导方针。Lowe和Hall（《Hypermedia and the Web：An Engineering Approach》，Wiley，1999）以及Powell[POW98]提供了相当全面的指导。Umar（《Application Re-engineering：Building Web-Based Applications and Dealing with Legacy Systems》，Prentice-Hall，1997）解决了WebE中最难的问题之一——遗留系统的再工程，使其与基于Web的系统兼容。IEEE Std.2001-1999定义了基本的WebE实践。

关于Web工程的大量信息可在Internet上获得。与Web工程有关的最新WWW参考文献列表可在SEPA Web站点<http://www.mhhe.com/pressman>找到。

尽管有很多人声称Web代表了在新规则下定义的新范型，专业开发人员正在认识到Internet出现之前的软件开发中得到的教训仍然适用。Web页面也是用户界面，HTML编程也是编程，浏览器部署的应用也是软件系统，它们都可以利用基本的软件工程原理。

514 软件工程的基本原则之一是：在你解决某个问题之前，要先理解问题，并确保你的解决方案是客户真正需要的。这条原则是表达的基础，也是Web工程的首要活动。另外一条基本原则是：先策划，后开发。这条原则是项目策划的基础。

17.1 表达基于Web的系统

基于Web的系统和应用的表达描述了一系列的Web工程活动，首先标识业务需求，进而描述WebApp的目标，定义主要特性和功能，进行需求收集，并开发分析模型。通过表达可以使共利益者和Web工程团队为WebApp的构造建立一组共同的目标。同时，表达也确定了开发的范围，并提供了决定产品是否成功的方法。表达活动之后的一项技术活动是分析活动，这项活动标识WebApp的数据、功能和行为需求。



在我们进一步详细讨论表达之前，有必要问一句：表达的结束点和需求分析的开始点在什么地方？这个问题不容易回答。表达关注于“宏观”——关注业务需求、目标及相关信息。然而，维持在这一抽象层次上几乎是不可能的。共利益者和Web工程师希望定义所需的内容，讨论特定的功能，列举特定的特征，并确定最终用户同WebApp的交互方式。这是表达？还是需求收集？答案是两者都是。

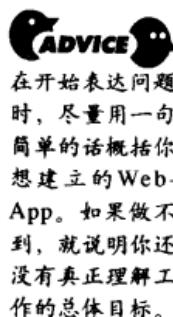
17.1.1 表达问题

在表达开始时，Powell[POW98]提出了一系列应该提问及回答的问题：

- 515**
- WebApp的主要动因（业务需求）是什么？
 - WebApp必须完成的目标是什么？
 - 谁将使用WebApp？

对这些简单问题的回答都应该尽可能简洁。例如，假定SafeHome¹的制造商决定建立一个电子商务Web站点，直接向消费者销售产品，可能会用下面的话来描述这个WebApp的目标：

SafeHomeAssured.com将允许消费者配置和购买安装一套家庭/商务管理系统所需的所有部件。



值得注意的是，在这段描述中没有涉及细节。目的是定义WebApp的总体目标，并将其包含在合理的业务内容中。

在与各种共利益者讨论之后，得出第二个问题的答案：

SafeHomeAssured.com能够让我们向消费者直接销售产品，以此消除中间销售成本，并提高利润。它也能使我们增加销售额，预计可增加当前年销售额的25%，同时能使我们将产品销售到那些我们还没有建立销售点的地域。

¹ 在本书前两部分已经用过SafeHome产品的例子。

最后，公司得出该WebApp的用户统计情况：“预计SafeHomeAssured.com的用户是家庭和小型商业单位。”

上面的回答蕴涵了SafeHomeAssured.com Web站点的特定目标。通常，存在两类目标[GNA99]：

- **信息目标：**指明向最终用户提供特定的内容和（或）信息的意图。
- **应用目标：**指明在WebApp中完成某项任务的能力。

在SafeHomeAssured.com WebApp中，一个信息目标可能是：

站点为用户提供详尽的产品说明，包括技术描述、安装说明和价格信息。

通过考察上面问题的答案，可以陈述应用目标如下：

SafeHomeAssured.com将向用户询问与要保护的设施（即房子、办公室/零售店空间）有关的一些信息，并给出所用产品和配置的定制建议。

一旦确定了所有的信息目标和应用目标，就可以建立用户轮廓了。用户轮廓捕获“与潜在用户相关的特性，包括他们的背景、知识、爱好以及其他更多内容”[GNA99]。在SafeHome-Assured.com的例子中，用户轮廓将确定购买安全系统的人一般应具有的特点（这种信息将由市场部提供）。

“如果你正准备开发[WebApp]，你的想法可能是‘准备，开火，瞄准’；如果你非常担心出错，那么你的指导思想应该是‘准备，瞄准，开火’。”

——作者不详

516

一旦确定了目标和用户概貌，表达活动将着重于描述该WebApp的范围。在很多情况下，已经得到的目标被集成到范围描述中。此外，指出希望得到的该WebApp的集成度也是有用的。也就是说，经常需要将现有的信息系统（例如，现有的数据库应用）和一个基于Web的前端集成在一起，在这个阶段应考虑连接性问题。

17.1.2 WebApp的需求收集

我们在第7章中讨论过需求收集的方法。尽管Web工程的需求收集活动会被简化，但是，软件工程中提出的总体需求收集目标没有改变。为了使其适合于WebApp，将这些需求目标调整如下：

- 明确内容需求。
- 明确功能需求。
- 对各类用户定义不同的交互场景。

 WebApp需
要用到哪些
需求收集步骤。

可以使用下面的需求收集步骤来实现这些目标：

1. 要求共利益者定义用户种类，并对每一类用户进行描述。
2. 与共利益者进行交流，明确WebApp的基本需求。
3. 分析收集到的信息，并不断同共利益者交流这些信息。
4. 定义用例（第8章），为每一类用户描述交互场景。

定义用户种类：可以证明WebApp的复杂性与使用该系统的用户种类数成正比。为确定用户的种类，就必然会涉及以下一系列基本问题：



在所有的软件工程工作中，了解用户的背景、动机和目标是很重要的。如果你在不了解这些内容的情况下开发WebApp，你的工作就会面临危险。

[517]

- 用户使用WebApp的总体目标是什么？例如，一个SafeHomeAssured.com电子商务网站的用户可能对收集家庭管理产品方面的信息感兴趣，另外一个用户可能想做一个价格比较，而第三个用户想购买SafeHome产品。每个人都代表了一类用户群体；每一类用户将有不同的需求，并需要不同的导航来浏览WebApp。可能第四个用户已经拥有了SafeHome产品，正在寻找技术支持，或者想购买另外一些传感器或附件。
- 与WebApp的内容和功能相关的用户背景和经验是什么？如果一个用户具有技术背景和丰富的经验，基本内容和功能对这类用户不会有帮助。另一方面，对于一个新手，这些基本内容和功能就很有用，否则他会感到非常迷惑。
- 用户如何访问WebApp？是从其他的Web站点（在内容和功能方面同WebApp相似）提供的链接访问WebApp？还是使用更受约束的方式访问WebApp？
- 用户喜欢或不喜欢哪些一般的WebApp特性？不同类型的用户有截然不同的、可预见的喜好。这就值得我们想办法去判断他们是否会喜欢我们的WebApp。在很多情况下，可以通过询问他们喜欢或不太喜欢的WebApp来得到这个问题的答案。

通过分析这些问题的答案，我们可以描述出最小的、合理的用户种类集合。在进行需求收集时，被定义的每一类用户都必须被考察到。

SAFEHOME

WebApp的需求收集

[场景] Doug Miller的办公室。

[人物] Doug Miller，软件工程团队经理；Vinod Raman，SafeHome软件工程团队成员；随后是三名市场部人员。

[对话]

Doug: 管理层已经决定，我们要建立电子商务网站来销售SafeHome。

Vinod: 什么，Doug！我们没有时间来做那个……我们已经深陷软件开发的工作中。

Doug: 我知道，我知道……所以我们准备将开发外包给一个擅长构建电子商务网站的公司。他们说会加快速度，在不到一个月的时间内完成……使用大量的可复用构件。

Vinod: 嗯，好的……那为什么让我来？

Doug: 为了使事情加快速度——他们需要我们帮助收集站点的需求，我想让你会晤各类共利益者，收集他们的想法，并将这些想法加入到基本需求中。

Vinod（被激怒了）：Doug……你没有听到我说的……我们正处于最紧张的时候，而且，这事……

Doug（打断）：只需要花你一天的时间，Vinod。同市场部的人会晤，帮助他们确定这个项目的基本内容和功能，你也知道，这是通常的做法。

Vinod（听天由命地）：好吧！我会给他们打个电话，安排一下明天的事情，但是，你从不让我轻松。

Doug (微笑): 这就是为什么给你的薪水最高的原因。

Vinod: 没错。

(第二天, Vinod会见三名市场部人员。)

Vinod: 你一直在告诉我用户的目标和背景。

市场部人员甲: 正如我所说的, 我们需要用户能够熟悉整个SafeHome系统, 你知道, 这些包括挑选传感器、控制面板、特点和功能, 然后自动生成一份“材料清单”, 获得定价, 然后通过Web网站购买这个系统。

市场部人员乙: 我们假设这个用户是一个家庭用户(不是技术人员), 所以我们需要一步一步地引导他(或她)完成这个过程。

市场部人员丙: 我不是技术人员, 但是, 除了基本的电子商务方面, 我担心我们需要做的专业方面的东西。

Vinod (对丙): 你的意思是?

市场部人员丙: 最困难的部分可能是使用一种简单、完全的方式引导客户完成“定制过程”, 实际的电子商务应该是非常简单的。

市场部人员甲: 我们已经为那些自己不想定制的人提供了800服务电话。

市场部人员丙: 我同意。

Vinod: 好的, 我们需要仔细讨论一下, 作为售前活动, 你希望如何进行产品定制? 不过我们先暂时搁下这个问题, 我还有几个其他问题需要讨论。

Vinod (看着市场部人员乙): 你刚才说你要引导客户完成这个过程, 有没有特别的方法?

市场部人员乙: 我想看到一个逐步的过程, 对于那些基本需求问题的回答由客户填写, 选择菜单等等。其中每一步也是一个窗口, 在进入下一步之前, 对每个窗口中的数据进行验证。

Vinod: 你向典型客户验证了你的想法吗?

市场部人员乙: 还没有, 但我会做的。

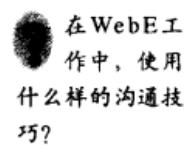
Vinod: 另外一个问题……用户如何才能找到我们的网站?

市场部人员甲: 我们正在同一个广告公司洽谈, 这个公司将在杂志、邮件快递及搜索引擎的广告部分刊登我们的网址www.SafeHomeAssured.com, 甚至还可以在电视和广播中做广告。

Vinod: 我的意思是……他们通常会直接通过主页进入。

市场部人员丙: 这也是我们希望的。

Vinod: 好, 现在我们开始工作, 让我们详细研究一下, 你希望如何在线定制系统。



同共利益者和最终用户交流: 大多数的WebApp都有大量的最终用户。尽管对用户进行分类会更容易地确定用户的需求, 但是, 仅仅使用从一两个用户那里收集到的信息作为表达和分析的基础是不明智的, 一定要考虑到更多用户(更多意见或观点)。

可以使用下面的一种或多种技巧来实现沟通[FUC02a]:

- 传统焦点组——一个经过专门训练的主持人同一小群(通常少于10人)具有代表性的最

终用户（或者内部的共利益者扮演最终用户的角色）会晤。目的是讨论要开发的WebApp，除了讨论，还为了更好地理解系统需求。

- 519**
- **电子焦点组**——同一群具有代表性的最终用户和共利益者进行适当的电子形式的讨论。参加者的人数可以更多一些。因为所有用户都能够在同一时间加入讨论，这样就可以在短时期内收集到更多的信息。由于所有的讨论都是基于文本形式的，可以自动生成同期讨论的记录。
 - **反复调查**——使用一系列的简明调查，通过Web网站或电子信箱发给具有代表性的用户，要求他们对一些有关WebApp的问题进行回答。需要仔细分析这些反馈，并用于完善下次调查。
 - **探索调查**——一个基于Web的调查，这个调查是关于一个或更多个WebApp的，这些WebApp拥有同要开发的WebApp相似的一些用户。让用户链接到这些调查，并对一系列问题做出回答（参与调查的用户通常会得到一些奖励）。
 - **设定场景**——向挑选出来的用户进行咨询，创建非正式的用例来描述客户同WebApp的特定交互。



对内容对象和操作的评估可以推迟到分析建模开始之前。在这个时候最重要的是收集信息，而不是评估信息。



本书的第二部分已经详细讨论了用例。很多人提倡开发冗长的用例，甚至一个非正式的用例描述也有用处。说服用户去写用例。

分析收集到的信息：完成信息收集之后，就可以根据用户种类和业务类型对这些信息进行分类，然后进行相关的评估，目的是要列出包含以下内容的目录：内容对象，在特定的用户交易中应用于内容对象的操作，WebApp提供给最终用户的功能（例如，信息的、计算的、逻辑的和基于帮助的），以及在沟通活动中谈到的非功能性需求等。

Fuccella和Pizzolato[FUC02b]提出了一个简单的（低技术水平的）方法来理解内容和功能是如何被组织的，这个方法就是为内容对象、应用于内容对象的操作、WebApp功能和其他非功能性的需求建立一个“卡片”栈，将这些卡片打乱，按随机顺序排列，然后分发给每一类用户的代表。要求用户将卡片分组，按照他们希望的那样将内容和功能组织在WebApp中。然后要求用户对每一组进行描述，并解释这组卡片为什么重要。一旦每个用户完成了这项工作，Web工程团队就会在不同类型的用户中寻找公共的组及对特定的用户类独特的其他组。

然后，WebE团队开发标签列表，用来指向由卡片分出来的各组信息。将卡片分给不同类型的用户，然后请他们给每个标签分配内容和功能。这样做的目的是确定标签（实际WebApp内的链接）应该在什么时候恰到好处地为客户提供其希望在标签下找到的内容和功能。重复这个步骤，直到取得一致意见。

“如果你不能将你所做的事情描述成一个过程的话，你就不知道你究竟在做什么。”

——W. E. Deming

开发用例：用例²描述一个特定的用户类型（称为一个角色）如何同WebApp交互来完成一个特定的行为。这个行为可能简单到只是获取定义的内容，也可能复杂到对客户选择的记录（由在线数据库维护）进行详细的客户指导分析。用例从用户的角度描述了这个交互。

² 开发用例的技术已经在第7章及第8章详细讨论过了。

尽管开发和分析用例要花费时间，但是用例能够：(1) 帮助开发人员明白用户希望如何同WebApp交互；(2) 为创建有效的分析模型提供详细的必要信息；(3) 帮助划分WebE工作；(4) 为那些测试WebApp的人提供重要的指导。

TASK SET

客户沟通（分析/表达）

1. 明确商业共利益者。确切地说，要明确谁是WebApp的“客户”？哪些业务人员可以作为专家和具有代表性的最终用户？谁能够在团队中起积极作用？
2. 表达业务内容。WebApp如何能适合更广泛的业务策略？
3. 为WebApp确定关键业务目标。如何定性和定量地度量WebApp的成功？
4. 定义指示性的且可适用的目标。要为最终用户提供哪些类型的内容？当使用WebApp时，要完成哪些功能或任务？
5. 明确问题。WebApp要解决哪些特定的问题？
6. 收集需求。使用WebApp完成哪些客户任务？要开发哪些内容？要使用什么样的交互比喻？WebApp要提供哪些计算功能？如何对WebApp进行网络资源配置？期望什么样的导航模式？

17.1.3 分析模型的过渡



对于小项目，可以在UML模型中维护一个简单的“需求数据库”（用电子表），这样就能够使WebE团队的所有成员跟踪交付的内容和功能需求，并能更好地控制那些可能出现的、不可避免的一连串变更。

正如本章前面提到的，将Web工程团队从表达引导到分析建模的活动表现出了连续性。实质上，在表达初期考虑的抽象层次是业务决策。然而，当进一步表达时，就需要讨论战略上的细节，并提出特定的WebApp需求。最终，对这些需求进行建模（使用用例和UML符号）。

已讨论过的软件需求分析的概念和原理（第7章和第8章）可无需修改地应用于Web工程分析活动。在分析阶段，对表达活动中定义的范围进行精化，以创建完整的WebApp分析模型。在WebE中要进行4种不同类型的分析：内容分析、交互分析、功能分析、配置分析。其中，每一个分析任务和与之相关的建模技术将在第18章讨论。

521

“失败的准备，其结果就是准备失败。”

——Benjamin Franklin

17.2 策划Web工程项目

基于WebApp的即时性，我们有理由问：我们真的需要花时间策划并管理一个WebApp吗？难道我们不能让WebApp自然演化，而只需少量或不需要明确的管理？一些Web开发人员选择少量管理或者无管理方式，但是这样做并不会让他们成功！

WebRef

项目管理者的辅助工具可以在www.e-project.com找到。

图17-1给出了一个自Kulik和Samuelson的[KUL00]改编的图表，它描述了“e项目”（WebApp项目的术语）如何与传统软件项目相比较。由图可见，传统的软件项目和大多数的e项目有很多相似之处。由于项目管理是应用于传统项目的，因此我们理所当然认为它也适用于大多数的e项目。小规模的e项目与传统

项目具有不同的特点。然而，即使在小规模的e项目中，为了避免出现迷惑、沮丧和失败的情况，也必须进行策划，考虑风险，制定进度表并定义控制措施。

	传统项目	小型e项目	大型e项目
需求收集	严格的	受限制的	严格的
技术规格说明	健壮：模型，规格说明	总体描述	健壮：UML模型，规格说明
项目持续时间	以月或者年为度量单位	以天、周或月为度量单位	用月或年为度量单位
测试和质量保证	致力于取得质量目标	致力于风险控制	在第26章描述的SQA
风险管理	明确的	内部的	明确的
可交付使用的期限	18个月或更长	3~6个月或更短	6~12个月或更短
发布过程	严格的	快速的	严格的
发布后客户的反馈	需要大量的主动工作	从用户交互中自动获得	自动获得及通过请求反馈获得

522

图17-1 传统项目和e项目的区别（根据[KUL00]改编）

17.3 Web工程团队

一个成功的Web工程团队融合了大量不同的人才，他们必须在高度紧张的项目环境中作为一个团队来工作。时间期限很短，变更很残酷，而技术又在不断更新，因此，组建一支团结的团队（见第21章）不是一件简单的事情。

“在今天以网络为中心、拥有Web的世界中，对于很多事情，一个人都需要了解很多。”
——Scott Tilley和Shihoug Huang

17.3.1 人员

创建成功的Web应用系统需要多方面的技能。Tilley和Huang[TIL99]在讨论这个问题时说道：“对[Web]应用软件而言，存在如此多的不同方面，以至于有文艺复兴式的人物（重新）出现，这种人能够自如地工作于多个学科之中……”。虽然该作者是绝对正确的，但是，“文艺复兴式”人才是相对短缺的，如果给定了大型WebApp开发项目的相关要求，所需的多种多样的技能最好分布在整个Web工程团队中。

Web工程团队的组织方式与传统软件团队（第21章）的组织方式相同，但是参与者及他们的角色通常有很大差别。整个WebE团队成员必须具备的技能包括：基于构件的软件工程、网络、体系结构设计和导航设计、Internet标准/语言、人机界面设计、图形设计、内容布局及WebApp测试。

WebE团队的成员中应该有人扮演下面的角色³：



人们在WebE
团队中扮演
什么角色？

内容开发者或提供者：因为WebApp在本质上是内容驱动的，WebE团队成员的角色必须着重于内容的生成和/或收集。由于内容涉及的数据对象非常广泛，内容开发者或提供者可能来自不同的专业（非软件的）背景。

Web出版者：必须对内容开发者或提供者生成的多种多样的内容进行组织，使这些内容包含在WebApp中。此外，某些人必须充当开发WebApp的技

³ 这些角色改编自Hansen及其同事的著作[HAN99]。

技术人员和非技术内容开发者或提供者之间的联络人，这个角色就由Web出版者承担，他必须了解内容和WebApp技术。

Web工程师：Web工程师参与WebApp开发过程中的一系列活动，包括需求导出、分析建模、体系结构、导航和界面设计、WebApp实现及测试。Web工程师也应该对构件技术、客户/服务器体系结构、HTML/XML和数据库技术等有扎实的理解，同时对多媒体概念、硬件或软件平台、网络安全和Web站点支持等应用知识有很好的了解。

业务领域专家：一个业务领域专家应该能够回答与业务目标和WebApp需求相关的所有问题。

支持专家：应该将这个角色分配给长期负责WebApp支持的人员。因为WebApp持续地演化，支持专家负责站点的纠错、适应性修改和增强，包括内容的更新、新的规程和表单的实现以及导航模式的改变。

管理者：通常称之为“Web站长”，他负责WebApp的日常运作，包括：为WebApp运行而制定的开发和实现政策，支持和反馈规程的建立，安全过程和访问权限的实现，Web站点流量的度量和分析，变更控制规程的协调（第27章），以及同支持专家协调。管理者也可能参与到Web工程师和支持专家进行的技术活动中。

17.3.2 组建团队



这些特点是那些已经采纳了敏捷思想（第4章）的协作的、自我组织的团队的象征。你的团队做的越好，你就越能生产出好的软件产品。

第21章详细讨论了组建成功的软件工程团队的指导原则，但是这些原则是否适合那些有巨大压力的WebApp项目呢？答案是肯定的。

很久以前，Tracy Kidder[KID00]在他编写的计算机界最畅销的书中讲述了这样一个故事：一家计算机公司勇敢地去尝试组建一台计算机，来迎接更大竞争者的新产品的挑战⁴。这个故事形象地描述了团队工作、领导能力，以及当某个重要的项目没有按计划顺利进行时，技术专家所遇到的难以忍受的压力。

Kidder书中的结论几乎是不正确的，但是当一个组织建立Web工程团队时，下面这些要点[PIC01]就非常中肯了：

应该建立一系列的团队准则。这些准则包括希望团队中每个成员怎样做，如何处理问题，当项目继续时存在哪些机制来提高团队效率。

必须有坚强的领导层。团队的领导必须为整个团队做出榜样，并不断同员工交流。他必须显示一定的热情，使其他团队成员发自内心地投入到他们面对的工作中。

尊重个人的才能很重要。并不是每个人都精通每件事情。最好的团队能够充分发挥个人的能力。最好的团队领导允许个人具有尝试自己好的想法的自由。

团队每个成员都应该负责任。Kidder书中的主角称此为“签约”。

一开始很简单，但是一直保持动力却很难。最好的团队从来不会让“不可战胜的”困难难倒他们。团队的成员会想出“足够好的”解决办法，并继续下去，从长远观点来看，希望向前发展，这样的动力会促使人们找到更好的解决办法。

⁴ Kidder的书《The Soul of a New Machine》最早出版于1981年，强烈推荐那些希望从事或已经从事计算机技术的人阅读这本书。

17.4 Web工程的项目管理问题

一旦完成了表达，基本的WebApp需求就确定了，企业必须从下面两个Web工程选项中挑选一种：(1) 将WebApp外包，Web工程由拥有技术和能力但缺少业务来源的第三方供应商来完成。(2) 内部开发，WebApp由企业雇佣的Web工程师开发。另外还有一个选择就是内部开发一部分Web工程工作，将其他工作外包。

“正如17世纪Thomas Hobbs观察到的，暴力统治下的生活是孤独的、贫穷的、肮脏的、粗野的和短暂的。在运行不良的软件项目下的生活同样也是孤独的、贫穷的、肮脏的、粗野的，也几乎永远是短暂的。” ——Steve McConnell

不管WebApp是外包、内部开发，还是由外部供应商和内部员工共同分工完成，要完成的工作是一样的。但是，需求的沟通、技术活动的分配、共利益者和开发人员之间的交互程度以及无数其他极其重要的问题都需要有所改变。

图17-2说明了外包同内部开发WebApp在组织上的不同。内部开发（图17-2a）直接将Web工程团队的所有成员集中（虚线圆表示集中）在一起，使用正规的组织方式进行交流。对于外包（图17-2b），如果让自己的内部成员（例如，内容开发者、共利益者、内部Web工程师）直接同外包供应商交流，而没有通过供应商的联络人来协调和控制的话，这样是不实际的，也是不明智的。在下面的小节中，我们将详细策划外包和内部开发。

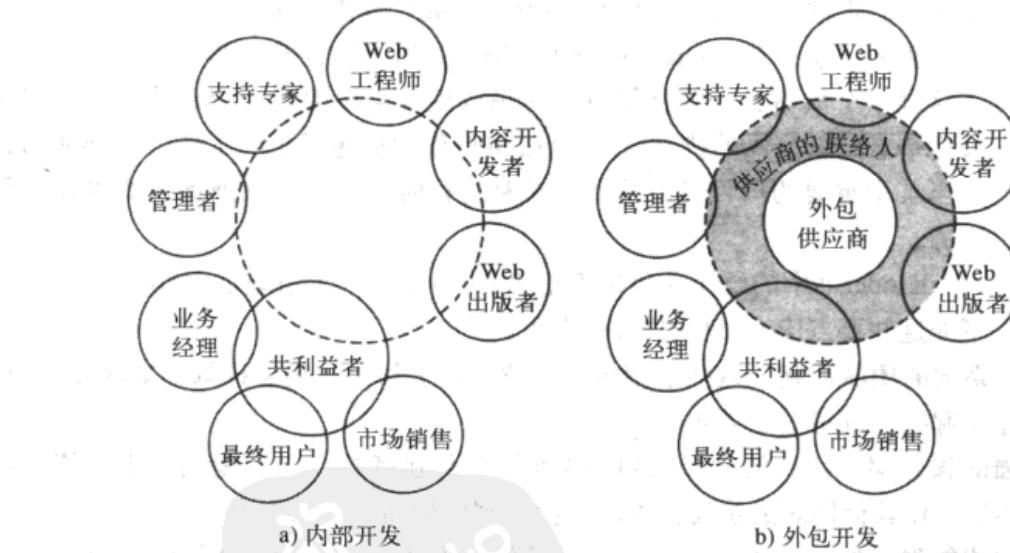


图17-2 外包和内部开发组织上的不同

17.4.1 WebApp策划——外包

绝大多数的WebApp都会外包给专门从事基于Web的系统和应用开发的供应商⁵。在这种情况下，企业（客户）应向两个或多个供应商询问WebApp开发的固定报价，对这些竞争报价进行比较，然后选择一个供应商来完成该工作。但是，签约的组织要寻求什么？如何确定

⁵ 尽管很难找到可靠的工业数据，但是可以肯定地说，WebApp外包的比例比传统软件外包的比例高得多。在第23章可以找到关于外包的其他讨论。



不要认为由于已经将WebApp外包，你的责任就减少了。事实上，很有可能需要更多的关照和管理，而不是更少。

WebApp供应商的竞争力？如何知道报价是否合理？当组织（及其外包签约方）开始着手主要的WebApp开发工作时，期望在何种程度上进行策划、安排进度和风险评估？

“很多500强的企业已经将软件视为一种服务模式[外包]，在企业的内部和外部使用相似的模式。”

—Nick Evans

这些问题并不容易回答，但下面几条指导原则值得考虑。

526

启动项目：如果WebApp开发选择了外包策略，一个机构在寻求外包供应商来完成此工作前，必须完成下面的一系列任务：

1. 在17.1.3节（和第18章）中讨论的很多分析任务应该在内部完成。明确WebApp的用户，列出可能对该WebApp感兴趣的共利益者，定义和评估WebApp的整体目标，刻画该WebApp提供的信息/服务，记录与其竞争的Web站点，并确定对一个成功的WebApp进行定性和定量“测度”的标准。这些信息应该写入提供给外包供应商的产品规格说明中。



有些人认为不需要“概要设计”。不过我们可以将其视为“第一次提议”，外包供应商可以在此基础上进行修改并提高。至少概要设计表达了你对最终结果的看法。

2. WebApp的概要设计应该在内部进行。显然，专业的Web开发人员会创建完整的设计，但是，如果外包供应商已经确认了该WebApp的一般观感（在项目的初始阶段总是要被修改），则可以节省时间和成本。设计应该包括WebApp要展示的内容的类型和容量，以及将要完成的交互处理的类型（例如，表格、订单条目）。应该将这些信息加入到产品规格说明中。

3. 应该开发出粗略的项目进度表，不仅包括最终的交付日期，还应该包括主要的里程碑日期。在WebApp的演化过程中，里程碑应该附属于其交付版本（增量）。

4. 为内部组织和外包供应商列出一个责任列表。实质上，这项工作描述了双方所需要的信息、联系和其他资源。

5. 明确签约组织和供应商间的监督和交互的程度。这应该包括供应商联络人的任命和联络人责任和权利的明确，当进行开发时质量评审点的确定，以及供应商在机构之间的交流中应负的责任。

应该将在上面步骤中所取得的所有信息组织在报价申请书中，并传递给候选的供应商⁶。

当考虑不同的外包供应商时，我们应使用什么指导原则？

挑选外包供应商：近几年，出现了数千家“Web设计”公司来帮助商业机构建立Web站点或从事电子商务。很多已经在WebE过程方面相当熟练，但是，也有很多公司和黑客没有太多差别。为了选择候选的Web开发者，签约方必须在以下方面付出努力：(1) 访问过去的客户，以确定Web供应商的职业道德，满足承诺的进度和成本的能力，以及有效沟通的能力；(2) 确定在过去成功的项目中供应商的首席Web工程师的名字（然后，通过合同的约束使他参与到你的项目中）；(3) 仔细检查供应商完成的样板，该样板和将要签约的WebApp在观感（和业务领域）方面应具有相似性。甚至在提供报价申请书之前，面对面的会议能够使签约方和供应商更好地了解彼此之间的“符合”程度。

527

⁶ 如果WebApp的开发工作将在内部完成，就不需要改动，还是按照原来的方式启动项目。

“抛砖引玉。”

—George Peppard playing Col.John “Hannibal” Smith on The A-Team (20世纪80年代的一个电视节目)

评估报价的正确性和估算的可靠性：因为在这方面几乎没有历史数据保留下，并且WebApp范围的易变性众所周知，估算在本质上就具有风险。为此，某些供应商将一定的保险额度加到项目的成本报价中，这是可以理解的，也是合适的。问题不是“我们的付出是否得到了最好的回报？”而应该是：

- WebApp系统的报价成本是否能够提供直接或间接的投资回报，从而证明项目是合算的？
- 已经提供报价的供应商是否具备我们所要求的职业道德和经验？

如果对这些问题的回答是“肯定的”，则该报价是公正的。

理解你期望或实现的项目管理程度：与项目管理任务（由供应商和签约方共同完成）相关的手续的正式程度直接与WebApp的规模、成本和复杂度成正比。对于大型、复杂的项目，应该制定详细的项目进度表，包括定义工作任务、SQA检查点、工程工作产品、客户评审点及主要里程碑。供应商和签约方应该一同对风险进行评估，并制定计划以减轻、监控和管理那些被认为是重要的风险。应该以书面形式明确定义质量保证和变更控制的机制。应该建立签约方和供应商之间高效沟通的方法。

确定开发进度：因为WebApp开发进度跨越的时间段较短（每次发布增量间隔经常小于1或2个月），开发进度中时间段的任务安排应该精细，也就是说，工作任务和小的里程碑应该按日来安排。这种精细的安排使得签约方和供应商能够在最终完成日期受到威胁之前认识到进度偏差。

528



为了管理范围，在一个增量中进行的工作被冻结。变更被延迟到下一个Web-App增量进行。

管理范围：由于在WebApp项目的开发过程中，其范围很有可能发生变更，所以WebE过程模型应该是可修改的和增量的，这种情况允许供应商的开发团队“冻结”某个增量的范围，使其可以创建一个可运行的WebApp版本。可能在下一个增量再执行前面增量的评审中所建议的范围变更，但是，一旦第二个增量开始，范围又再次被暂时冻结。这个方法使得WebApp队伍可以正常工作，而不必去刻意顺从连续的变更流；但是，我们仍能看到大多数WebApp的持续演化特征。

上面建议的指导原则并不是开发低成本、及时的WebApp产品的保证，然而，它们可以帮助签约方和供应商双方在最小误解的情况下顺利地起动工作。

SAFEHOME

外包准备

[场景] Doug Miller的办公室。

[人物] Doug Miller (SafeHome软件工程团队经理) 和Sharon Woods——电子商务系统的雇员，也是SafeHome电子商务网站的外包供应商，同时也是将要开发这个网站的Web工程团队的经理。

[对话]

Doug: Sharon, 见到你很高兴！我们会在今后的几个月一同工作。

Sharon (微笑): 是的，但是，你的员工们好像也要跟我们一起工作。Vinod已经给

了我们一个网站规格说明的草稿，也明确了大部分重要的内容和网站功能。

Doug: 好，其他还需要什么？

Sharon: 电子商务的功能很容易。我担心的是前端……就是在购买之前让客户按照自己的意愿定制产品。

Doug: Vinod给了你一些基本的过程，不是吗？

Sharon: 是的，他给了，但是我想与一些真实的客户核实一下。我们也希望与你的内容开发者取得联系，以便能够正确地描述每个传感器、图片、价格、界面或相互连接等信息。

Doug: Vinod有时间为你画一幅粗略的定制过程的流程图吗？

Sharon: 他现在正在做。他说他不得不放下产品方面的工作，他清楚这件事情很重要……他说他会在明天早晨发E-mail给我。

Doug: 好……你瞧，我很高兴能够参与到这个项目中来。我们能不能建立某些程序来监管我们的目标？我不希望干扰你们，但是……

Sharon: 没问题，我很高兴让我们的客户参与进来。

Dough: 我会在这个项目中担任联络人的角色。所有的联络都将通过我或者某个我指定的人，比如Vinod。因为我们的关系很紧，我想建立一个以日为单位的进度表，每天同你谈论或发E-mail来讨论我们完成的成果、遇到的问题等等。我知道这项工作比较麻烦，但是我认为这样做很合适。

Sharon: 这样很好。

Doug（从他办公桌上抽出几份文件递给Sharon）：我已经起草了一个包括里程碑日期在内的粗略进度表，你认为如何？

Sharon（看完进度表后）：嗯，我不能确定这个是否适合我们，我想略加修改一下，今天稍后发E-mail给你。

Doug: 好的！

529

17.4.2 WebApp策划——内部Web工程

当WebApp逐渐普遍深入和更具有业务策略性时，很多公司倾向于内部控制开发。内部WebE的管理同外包相比有些不同，这并不奇怪。

“如果昨天你想让人帮你做一个Web网站，你会做什么？”

—James Lewin



认识到本节讨论的步骤能够很快完成很重要。无论如何，针对这种规模的项目，WebE策划的工作量不能超过整个项目工作量的5%。

中小规模（即持续时间小于3~5个月）的WebE项目的管理需要敏捷方法，这种方法不再强调项目管理，但不排除制定计划的必要。那些基本的项目管理原则（第21章）仍然适用，但是总体方法会变瘦，而且不太正式。然而，随着WebApp项目的规模逐渐增大，WebE项目管理也变得越来越像软件工程管理（本书第四部分）。建议使用下面的步骤对中小规模的WebE项目进行管理：

理解范围、变更的维数和项目的限制。没有一个项目（不管时间限制多么紧张）能够在项目团队还没理解要做什么之前就开始。需求收集和客户沟通是有效的WebApp策划的重要前提。

定义增量的项目策略。我们已经提到过WebApp一直在演化。如果演化失去控制，并且很混乱时，最终获得成功的可能性就很小了。然而，如果团队建立了项目策略，定义了为最终用户提供有用的内容和功能的WebApp增量(版本)，工作量就能够更有效地集中在焦点上。

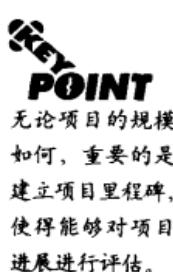
完成风险分析。在第25章详细讨论了传统软件工程项目的风险分析⁷。所有风险管理的任务也适用于WebE项目，只是对方法进行了简化。

大多数的Web工程团队的注意力集中在进度风险和技术风险。团队必须考虑和解决的很多与风险相关的问题是：计划好的WebApp增量能否在已规定的时间期限发布？当额外的增量被开发后，它们会为最终用户提供有价值的东西吗？变更请求如何影响发布进度？团队理解了所需要的Web工程方法、技术和工具吗？现在可用的技术适合现有的工作吗？有可能出现的变更需要引入新技术吗？

进行快速估算。大多数Web工程项目估算的重点是宏观问题，而不是微观问题。WebE团队评估所策划的WebApp增量能否在已有的进度限制下，利用已有的资源完成开发。这就需要将每个增量的内容和功能作为一个整体来考虑，这样才能完成估算。通常不对增量的微观功能或工作的细目分类，也不对多个数据点计算进行估算（见第23章）。

选择任务集（过程描述）。使用过程框架（第16章）选择适合问题、产品、项目以及WebE团队成员的特点的一组Web工程任务。也必须认识到这组任务可能需要修改以适应每个开发增量。

制定进度表。WebE项目需要有适合短期任务的较精细的进度表，也需要有适合以后时期（当另外的增量被发布之后）的粗略进度表。也就是说，根据要开发的增量的项目期限分配Web工程任务。预定的增量发布后，才对后面的WebApp增量进行任务分配。



建立项目跟踪机制。在敏捷开发环境中，可运行的软件增量的发布通常是测量项目进展的主要方法。但是，当软件还远远没有达到可发布之前，Web工程师通常不可避免地遇到一个问题：我们现在处在什么位置？在传统的软件工程工作中，项目进展情况通常是根据已经到达了哪个里程碑（例如，对一个工作产品的成功评估）来确定。对于中小规模的Web工程项目，可能没有很好地定义里程碑，也不重视正式的质量保证活动。因此，一种方法是让整个WebE团队投票决定已经完成了哪些框架活动，以此来决定项目进度，然而这种方法是不可靠的。另外一种方法是确定有多少用例已经完成，还有多少用例（对于一个给定的增量来说）将要完成，这就大致给出了项目增量相对的“完成”程度。

“进展是由纠正正在进展中产生的错误而取得的。”

—Claude Gibb

建立变更管理方法。由于有了适用于WebApp的增量开发策略，变更管理就很方便了。由于增量开发的时间很短，一个变更的引入通常可能被拖延到下一个增量，因此，必须尽快完成变更，以减少由于变更所造成的拖延。WebApp的配置和内容管理在第27章介绍。

⁷ 那些对基本的风险管理术语和实践不熟悉的读者，这时应该查阅第25章。

SOFTWARE TOOLS**WebE项目管理**

目的：辅助Web工程团队进行策划、管理、控制和跟踪Web工程项目。

机制：项目管理工具使得WebE团队能够建立一系列的工作任务，为每个任务指定工作量和明确的责任，建立任务依赖关系，定义进度，并跟踪控制项目活动。这类中的很多工具是基于Web的。

代表性工具⁸

Business Engine，由Business Engine (www.businessengine.com) 开发，是一套基于Web的工具，为WebE和传统软件项目提供了丰富的项目管理功能。

iTeamwork，由iTeamwork.com (www.iteamwork.com) 开发，“是一个通过Web浏览器使用的免费的、在线的、基于Web的团队项目管理应用系统。”

OurProject，由Our Project (www.ourproject.com) 开发，是一套应用于WebE和传统软件项目的项目管理工具。

Proj-Net，由Rational Concepts, Inc. (www.rationalconcepts.com) 开发，完成了一个“协作和通信的虚拟项目办公室（VPO）”。

StartWright (www.startwright.com/project1.htm) 开发的WebE和传统软件项目管理工具和信息是最全面的Web资源之一。

我们应该强调一下，很多传统的项目管理工具（本书第四部分）也能有效地用于WebE项目。

17.5 Web工程与WebApp的度量



通常，你收集到的WebE度量的数目和它们整体的复杂度应该正比于要开发的WebApp的规模。

当Web工程师开发复杂的系统时，正如完成此项任务的其他技术人员一样，他们应该使用度量来改善Web工程过程和产品。在第15章，我们讨论了在软件工程范畴内软件度量在策略上和战术上的应用，这些用法也适用于Web工程。

总之，软件度量为改进软件过程、增加项目估算的准确性、增强项目跟踪、提高软件质量提供了基础。如果能够合理地描述这些特征，WebE度量能够获得所有的益处，还能提高可用性、WebApp的性能和用户满意度。

在Web工程的范畴内，度量具有三个主要目的：(1) 从技术角度提供了WebApp的质量指标；(2) 提供了工作量估算基础；(3) 从业务角度提供了WebApp成功的标识。

在本节，我们概括了WebApp的普通工作量和复杂度的度量⁹。这些信息可以用于开发工作量估算的历史数据库。另外，复杂度估算最终能使你定量地评估一个或多个我们在第16章所讨论的WebApp的技术特点。

532

17.5.1 Web工程工作量的度量

当一个WebApp演化时，Web工程师耗费人力劳动来完成各种工作任务。Mendes和他的同

⁸ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。在大多数情况下，工具的名字由各自的开发者注册为商标。

⁹ 值得注意的是，WebE度量还处于起步阶段。

事们[MEN01]提出了很多可能的WebApp工作量的度量方法。WebE团队可以把其中的一些或全部信息记录下来，用于将来建立估算用的历史数据库（第23章）。

应用创作和设计任务

建议的度量	描述
构造工作量	构造WebApp和（或）建立体系结构的时间
互连工作量	将页面链接起来构建WebApp结构的时间
界面策划	策划WebApp界面的时间
界面构建	实现WebApp界面的时间
链接测试工作量	测试WebApp所有链接的时间
媒体测试工作量	测试WebApp中所有媒体的时间
总工作量	构造工作量 + 互连工作量 + 界面策划 + 界面构建 + 链接测试工作量 + 媒体测试工作量

页面创作

建议的度量	描述
文本工作量	页面的创作或文本复用的时间
页面链接工作量	创作页面链接的时间
页面构造工作量	构造页面的时间
总页面工作量	文本工作量 + 页面链接工作量 + 页面构造工作量

媒体创作

建议的度量	描述
媒体工作量	创作或复用媒体文件的时间
媒体数字化工作量	将媒体进行数字化的时间
总媒体工作量	媒体工作量 + 媒体数字化工作量

程序创作

建议的度量	描述
编程工作量	创作HTML、Java或者相关语言实现的时间
复用工作量	复用/修改现有程序的时间

533

17.5.2 评估商业价值的度量

WebRef

与Internet业务有关的很多主题的极好资料可以在www.internet.com找到。

有趣的是，业务人员已经在WebApp的开发、收集及使用度量方面明显地超过了Web工程师（例如，[STE02]，[NOB01]）。通过统计最终用户的数量和了解他们的使用模式，一个公司或组织就能够取得更有意义的WebApp内容、更有效的市场营销以及更高的商业利润。

收集有商业价值的数据需要何种机制，这通常由Web工程团队来完成；但是对所产生的数据和作用的评估是由其他支持者来完成的。例如，假设页面的浏览量是根据每个访问者来计算的，根据所收集的度量，来自搜索引擎X的访问者平均有9个页面浏览量，而来自门户网站Y的访问者只有2个页面的浏览量。市场部门可用这些平均数来分配标语广告预算（根据收集的度量可以看出，搜索引擎X上的广告浏览量比门户网站Y上更多）。

关于商业价值度量的收集和使用（包括正在进行的关于个人隐私的讨论）的深入讨论超出了本书的范围。感兴趣的读者可以查阅[INA02]、[EIS02]、[PAT02]或[RIG01]。

SOFTWARE TOOLS**Web度量**

目的：评估使用WebApp的方法、用户的类型，以及WebApp的可用性。

机制：一旦WebApp在线运行，大多数的Web度量工具都能够捕获有用的信息。这些工具可以提供大量数据，可以用这些数据来评估WebApp的哪些元素使用最多，如何使用它们，以及谁使用它们。

代表性工具¹⁰

Clicktracks，由clicktracks.com (www.clicktracks.com) 开发，是一个日志文件分析工具，它将Web站点的访问者的行为直接显示在Web站点上。

Marketforce，由Coremetrics (www.Coremetrics.com) 开发，是众多工具中的典型代表，它收集的数据可用来评估电子商务WebApp是否成功。

Web Metrics Testbed，由NIST (zing.ncsl.nist.gov/WebTools/) 开发，是一套基于Web的工具，用来评估WebApp的可用性。

WebTrends，由netIQ (www.NetIQ.com) 开发，为所有类型的WebApp收集广泛的有用数据。

17.6 WebApp项目的“最坏实践”

有时学习正确做某事的最好的方法就是仔细分析为何不那么做！在过去的10年中，遭到失败的WebApp并非少数，其主要原因是：(1) 忽略了项目，而且改变了管理原则（然而是非正式的），导致Web工程团队“碰壁”；(2) WebApp开发使用了特殊的方法，结果不能产生可工作的系统；(3) 在需求的收集和分析中使用了草率的方式，从而不能开发出满足客户要求的系统；(4) 在设计时采用了不合适的方法，从而不能开发出可用的、功能性的、可扩展的（可维护的）及可测试的WebApp；(5) 使用不受重视的（unfocused）方法进行测试，在方法引入之前，不可能产生能工作的系统。

534

记住这些事实的同时，我们有必要去考虑一系列Web工程的“最坏实践”，这些实践改编自Tom Bragg[BRA00]的一篇文章。如果你的e项目表现出它们中的任何一个实践，就有必要立即采取补救措施了。

最坏实践#1：因为我们已经有了一个伟大的想法，所以现在我们就开始开发这个WebApp吧！不用操心去考虑这个WebApp业务是否正确，用户是否的确需要使用它，是否已经真正理解了业务需求。时间很短，我们不得不马上开始。

事实：花费几个小时或者几天的时间为WebApp做一份业务案例，确定你的想法能够得到投资方和使用方的认可。

最坏实践#2：员工们会不断地修改不合适的地方，因此，就没有必要努力理解WebApp的需求。从来不记录任何东西（浪费时间），紧紧依靠口头交流。

事实：的确，WebApp需求会随着Web工程活动的进行而演化。口头传递信息既快又简单；然而，需求收集和分析中的草率方式是造成更多（不必要的）修改的催化剂。

¹⁰ 这里记录的工具只是此类工具的例子，并不代表本书支持这些工具。在大多数情况下，工具的名字由各自的开发者注册为商标。

最坏实践#3：那些在传统软件开发中具有领域经验的开发者能够立即开发WebApp，不需要对其进行新的培训。毕竟，WebApp也只是软件，不是吗？

事实：WebApp不同于传统的软件，必须熟练地应用非常广泛的方法、技术和工具，这方面的培训及经验是必不可少的¹¹。

最坏实践#4：官僚作风。坚持沉重的过程模型、工作时间表、大量没必要的“进展”会议，并让从来没有管理过WebApp项目的人担任项目领导，这些都是官僚作风。

事实：鼓励敏捷的过程，强调有经验的Web工程团队的能力和创造性。排除干扰，让他们把精力集中在工作上。如果必须收集与项目相关的信息（由于法定的原因或者度量计算），那么数据条目或数据集合应该尽可能简单，不唐突。

最坏实践#5：测试？何必那么麻烦？我们会将WebApp项目提供给几个最终用户，然后让他们告诉我们什么工作，什么不工作。
[535]

事实：的确，随着时间的推移，最终用户会完成对WebApp的彻底“测试”，但是，他们会因为系统的不可靠性和糟糕的性能而感到不安，以至于在问题得到纠正之前他们就远远离开了（甚至永远也不会回来）。

在接下来的几章中，我们会讨论Web工程方法，以帮助读者避免这些错误。

17.7 小结

“表达”是一项同客户交流的活动，它明确了WebApp将要解决的问题，同时，业务要求、项目目标、最终用户的分类、主要功能和特征以及同其他应用系统的互操作程度都需要得到定义。随着获得的详细技术信息越来越多，表达就变成了需求分析。

WebE团队由一群技术和非技术成员组成，这些成员以相当的自主性和灵活性组织在一起。在Web工程中同样需要项目管理，但是项目管理的任务被简化了，而且不像应用于传统软件工程项目的项目管理那样正式。很多WebApp项目被外包，但是更多的项目倾向于内部开发。针对每种方法，项目管理在策略和战术上也是不一样的。

Web工程度量还处于起步阶段，但是可以想到的是，它能够用来明确WebApp的质量，提供工作量估算的基础，并且从业务角度提供WebApp成功的指标。

参考文献

- [BRA00] Bragg, T., "Worst Practices for e-Business Projects: We Have Met the Enemy and He Is Us!" *Cutter IT Journal*, vol. 13, no. 4, April 2000, pp. 35-39.
- [CON02] Constantine, L., and L. Lockwood, "User-Centered Engineering for Web Applications," *IEEE Software*, vol. 19, no. 2, March/April 2002, pp. 42-50.
- [EIS02] Eisenberg, B., "How to Interpret Web Metrics," *ClickZ Today*, March 2002, available at <http://www.clickz.com/sales/traffic/article.php/992351>.
- [FUC02a] Fuccella, J., J. Pizzolato, and J. Franks, "Finding Out What Users Want from your Web Site," IBM developerWorks, 2002, <http://www-106.ibm.com/developerworks/library/moderator-guide/requirements.html>.
- [FUC02b] Fuccella, J., and J. Pizzolato, "Giving People What They Want: How to Involve Users in Site Design," IBM developerWorks, 2002, <http://www-106.ibm.com/developerworks/library/design-by-feedback/expectations.html>.

¹¹ 很多大型的WebE项目需要同传统的应用和数据库集成在一起。在这种情况下，那些只具有传统软件开发经验的人员可以加入，也应当加入。

- [GNA99] Gnado, C., and F. Larcher, "A User-Centered Methodology for Complex and Customizable Web Applications Engineering," *Proc. First ICSE Workshop in Web Engineering*, ACM, Los Angeles, May 1999.
- [HAN99] Hansen, S., Y. Deshpande, and S. Murugesan, "A Skills Hierarchy for Web Information System Development," *Proc. First ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999.
- [INA02] Inan, H., and M. Kean, *Measuring the Success of Your Web Site*, Longman Publishing, 2002.
- [KID00] Kidder, T., *The Soul of a New Machine*, Back Bay Books (reprint edition), 2000.
- [KUL00] Kulik, P., and R. Samuels, "e-Project Management for a New e-Reality," Project Management Institute, December, 2000, <http://www.seeprojects.com/e-Projects/>
- [LOW98] Lowe, D., and W. Hall, eds., *Hypertext and the Web—An Engineering Approach*, Wiley, 1998.
- [MEN01] Mendes, E., N. Mosley, and S. Counsell, "Estimating Design and Authoring Effort," *IEEE Multimedia*, January–March 2001, pp. 50–57.
- [NOB01] Nobles, R., and K. Grady, *Web Site Analysis and Reporting*, Premier Press, 2001.
- [PAT02] Patton, S., "Web Metrics That Matter," *CIO*, November 15, 2002. available at <http://www.computerworld.com/developmenttopics/websitemgmt/story/0,10801,76002,00.html>.
- [PIC01] Pickering, C., "Building an Effective E-Project Team," *E-Project Management Advisory Service*, Cutter Consortium, vol. 2, no. 1, 2001, <http://www.cutter.com/consortium>.
- [POW98] Powell, T.A., *Web Site Engineering*, Prentice-Hall, 1998.
- [RIG01] Riggins, F., and S. Mitra, "A Framework for Developing E-Business Metrics through Functionality Interaction," January 2001, download from <http://digitalenterprise.org/metrics/metrics.html>.
- [STE02] Sterne, J., *Web Metrics: Proven Methods for Measuring Web Site Success*, Wiley, 2002.
- [TIL99] Tilley, S., and S. Huang, "On the Emergence of the Renaissance Software Engineer," *Proc. 1st ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999.

536

习题与思考题

- 17.1 思考17.5.1节讨论的对Web工程工作量的度量。试为一种或多种类型开发出5个或更多的其他度量。
- 17.2 除了17.1.1节提到的三个基本的表达问题，你认为是否还有任何其他问题要问？如果有，这些问题是什么，为什么你会问这些问题？
- 17.3 使用Web站点导航的方便程度是WebApp质量的一个重要指标。开发2个或3个能够用来说明导航方便程度的度量。
- 17.4 用自己的语言说明如何分析在同客户交流中收集到的信息，并说出这个活动的结果是什么。
- 17.5 描述与外包WebApp开发相关的5个风险。
- 17.6 考虑在本章中讨论的SafeHome电子商务网站，你会使用哪种同客户沟通的机制来导出系统需求，为什么？
- 17.7 复习图17-1所列出的表，再加上3行或者更多行来进一步区分传统项目和e项目。
- 17.8 用自己的语言描述Web出版者的角色。
- 17.9 复习第4章介绍的敏捷开发团队的特点。你认为一个敏捷团队组织适合WebE吗？你能对这个WebApp开发的组织做哪些修改？
- 17.10 将用例的开发作为需求收集活动的一部分有什么好处？
- 17.11 描述与内部WebApp开发相关的5个风险。
- 17.12 在需求收集中，什么是“用户类别”？举出一个在线图书销售的三种用户类别的例子。
- 17.13 如何区分表达与需求收集？如何区分表达与需求分析和分析建模？

[537] 17.14 使用17.5.2节中的参考书目之一，讨论如何使用商业价值度量辅助实际的业务决策。

推荐读物与阅读信息

传统的应用软件有一些表达和需求收集方法，通过修改这些类似的方法，可以得到WebApp表达和需求收集方法。第7章和第8章推荐的阅读材料包括很多对Web工程师有用的信息。

Flor (《Web Business Engineering》，Addison-Wesley，2000) 讨论了业务分析，以及能够让Web工程师更好地理解用户要求的相关内容。WebApp的可用性是一个概念，这个概念是很多信息（被定义为表达和需求收集的一部分）的基础。Krug和Black (《Don't Make Me Think: A Common Sense Approach to Web Usability》，Que Publishing，2000) 包括很多指导原则和例子，可帮助Web工程师将用户需求转化为有效的WebApp。

WebE项目的项目管理方法很多都借鉴了应用于传统软件项目的原理和概念。然而，敏捷是一个格言。Wallace (《Extreme Programming for Web Projects》，Addison-Wesley，2003) 描述了敏捷开发如何被应用到WebE中，并包括了一些关于项目管理问题的有用讨论。Shelford和Remillard (《Real Web Project Management》，Addison-Wesley，2003)、O'Connell (《How to Run Successful Projects in Web Time》，Artech House，2000)、Freidlein (《Web Project Management》，Morgan Kaufman，2000) 及Gilbert (《90 Days to Launch: Internet Projects on Time and on Budget》，Wiley 2000) 讨论了广泛的WebE项目管理问题。Whitehead (《Leading a Software Development Team》，Addison-Wesley，2001) 提出了很多有用的指导原则，可以对这些原则进行修改以适用于Web工程团队。

在Sterne[STE02]、Inan[INA02]、Nobles[NOB01]、Menasce和Almeida (《Capacity Planning for Web Services: Metrics, Models and Methods》，Prentice-Hall，2001) 的书中提出了一些如何使用Web度量进行业务决策的技术。在Ferry和Ferry (《77 Sure-Fire Ways to Kill a Software Project》，iUniverse.com，2000) 的书中讨论了“最坏实践”。

关于Web工程的表达和策划方面的广泛的信息源可以在Internet上得到。最新的关于表达和策划的WWW参考文献列表可以在SEPA Web站点<http://www.mhhe.com/pressman>找到。



第18章 WebApp分析

要点浏览

概念：Web应用分析重点在于解决三个重要问题：(1) 表达或处理的信息或内容是什么？(2) 为最终用户提供的是什么功能？(3) 当WebApp表达内容和执行功能时，它表现出来的行为是什么？可将这些问题的回答表示为分析模型的一部分，分析模型包含了各种UML表示。

人员：Web工程师、非技术性内容开发者和共利益者参与分析模型的创建。

重要性：我们在整本书中都强调：在解决问题之前，一定要先理解问题。分析模型很重要，并不是因为它能使Web工程团队开发出具体的WebApp需求模型（由于需求的变化太频繁，这种期望不太现

实），而是因为分析模型能使Web工程师定义问题的基本方面——问题是不可改变的（在短期内）。只有理解了软件项目的基本内容、功能和行为，设计和构造才能变得容易。

步骤：分析模型集中于问题的4个基本方面——内容、交互、功能和配置。内容分析确定内容类和协作；交互分析描述了用户交互的基本元素、导航及最终发生的系统行为；功能分析定义了为用户提供WebApp功能及最终的处理顺序；配置分析确定了WebApp所处的操作环境。

工作产品：分析模型由一系列UML图和描述内容、交互、功能和配置的文本组成。

质量保证措施：评审分析建模工作产品的正确性、完整性和一致性。

关键概念
分析模型
配置模型
内容模型
内容关系
数据树
功能模型
交互模型
导航分析
关系分析
RNA
用例
用户层次

当我们在Web工程的范畴内开始考虑分析建模时，会感觉存在明显的矛盾。毕竟，我们已经谈到WebApp具有直接性和灵活性（第16章），从而减轻了分析阶段或设计阶段详细建模的负担。而且，敏捷原理（一个对许多Web工程项目都适合的过程模型）表明：无论构建什么模型，如果我们不重视分析建模，就会导致设计建模的局限性。Franklin[FRA02]注意到了这一点，他写道：

网站具有典型的复杂性和高度的动态性。为了能使产品尽快推出，并运转起来，往往开发周期很短。通常，开发者在没有真正理解他们要努力做什么或怎么去做的情况下，就直接进入编码阶段。服务器端编码通常很特别，在需要的时候才增加数据库表，体系结构有时也以随意的方式发展。但是某种形式的建模和严格的软件工程可以使软件开发过程更加顺利，并且能够保证Web系统将来更易于维护。

同时达到这两个目的可能吗？在直接性和灵活性支配的世界中，我们在进行“某种建模和严格的软件开发”的同时，仍然能够保持工作效率吗？在某种条件下，回答是肯定的。

在以下大多数或所有情况下，Web工程团队应该采用分析建模：

- 将要构建的WebApp非常庞大和（或）复杂。
- 共利益者的人数很多。
- Web工程师和其他参与者的人数很多。
- WebApp的目的和目标（在表达期间决定）将影响业务的基本要求。
- WebApp的成功对商业成功具有重要意义。

如果这些条件都不存在，可以不采用分析模型，而将表达期间获得的信息和收集的需求（第17章）作为创建WebApp设计模型的基础。在这种条件下，可能会创建有限的分析模型，但分析模型会与设计模型合为一体。

18.1 WebApp的需求分析

WebApp的需求分析包括三个主要任务：表达、需求收集¹ 和分析建模。在表达期间，要确定WebApp的动机（目的）和目标，并定义用户的种类。随着需求收集的开始，Web工程团队和WebApp共利益者（例如，顾客、最终用户）之间的交流会逐渐增强。应首先列出内容和功能需求，并从最终用户的角度来开发交互的场景（用例），其目的是对为什么要建立WebApp有一个基本的了解，明确谁将使用它？它将为用户解决哪些问题？

“不管是设计前的策划阶段，还是构造前的设计阶段，软件工程的原则都经受了每个重要的技术转变，它们也必将战胜这种转变。”

——Watts Humphrey

540

18.1.1 用户层次



建立一个用户层次是一个好主意。它使你对用户数量有了一个简单印象，并可以反复查对，以确保每个用户的需求都已提出。

确定与WebApp交互的最终用户的种类是表达和需求收集任务的一部分。在大多数情况下，用户种类是非常有限的，因此没有必要用UML来表示。但是，当用户的种类增多时，有时候建立如图18-1所示的用户层次图是明智的。这幅图描述了在第16章和第17章所谈论的SafeHomeAssured.com电子商务网站的用户。

图18-1所示的用户种类（通常称为角色）表明了WebApp提供的功能，以及为用户层次中的最终用户（角色）建立用例的需求。在此图中，SafeHomeAssured.com的用户处在用户层次的顶端，代表了最一般的用户类（种类），并在下面的层次中对其进行细化。客人是可以访问网站的用户，但没有注册，这种用户通常可以查找一般信息，比较商品价格，或者对网站上“免费”的内容或功能感兴趣。注册用户要花时间提供联系信息（以及表格中要求填写的其他统计数据）。注册用户的子类包括：

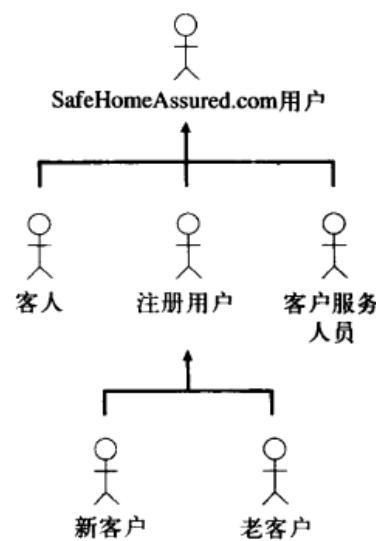


图18-1 SafeHomeAssured.com的用户层次

¹ 第17章详细讨论了表达和需求收集。

- 新客户——注册用户，他们想要定制并购买SafeHome部件（因此，必须与WebApp的电子商务功能交互）。
- 老客户——已经购买了SafeHome部件的用户，他们正在使用WebApp (1) 购买另外的部件；(2) 获得技术支持信息；(3) 联系客户支持。

客户服务人员是一种特殊的用户，当他们帮助需要SafeHome客户支持的客户时，也可以与SafeHomeAssured.com的内容和功能交互。

18.1.2 开发用例

Franklin [FRA01]称用例为“功能包”，这个词反映了分析建模技术的重要本质²。用例是为用户层次上的每一用户种类开发的。在Web工程环境中，用例本身相对来说是非正式的——可以用一段叙述性的文字描述用户与WebApp之间的特定交互³。

图18-2表示“新客户”类（图18-1）的UML用例图。图中的每个椭圆代表一个用例，描述了新客户和WebApp之间的一次具体交互。例如，第一次交互的用例是“登录SafeHome-Assured.com”，这个普通的交互只需要用一段文字来描述即可。

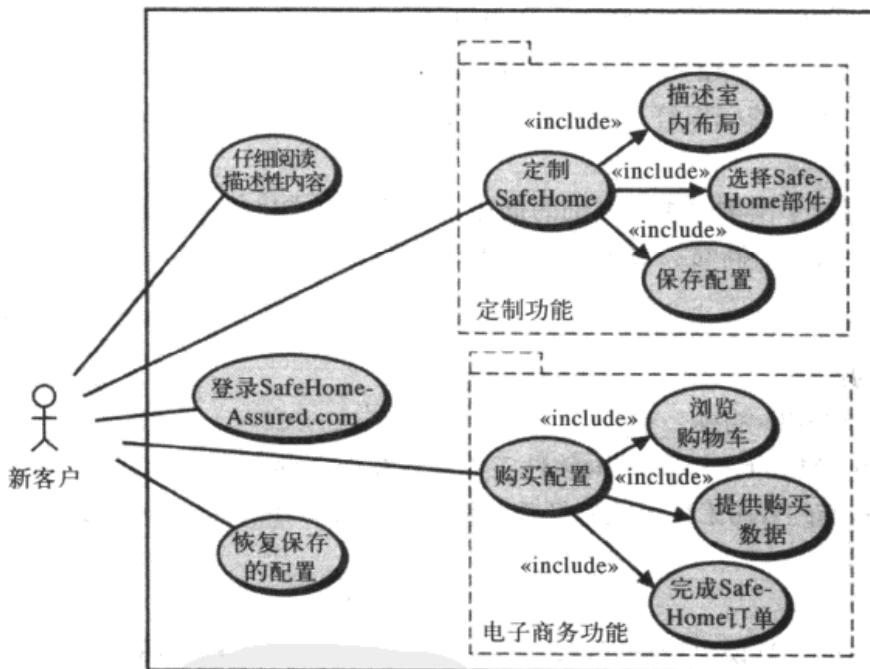


图18-2 新客户的用例图

在图18-2中的虚线框里记录了WebApp的主要功能（和与功能相关的用例）。这些就是UML中提到的“包”，代表了特定的功能。记录的两个包指的是定制（customization）功能包和电子商务（e-commerce）功能包。

作为例子，我们考虑用例图中的定制功能包。一个新客户必须描述要安装SafeHome的室内环境。为了做到这一点，新客户可以启动用例“描述室内布局”、“选择SafeHome部件”及“保存配置”。下面从一个新客户的角度来考虑这些初步用例：

² 开发用例的技术已经在本书前面章节详细讨论过了（见第7章和第8章）。

³ 尽管可以建立更正式的用例描述，但是WebE对敏捷性要求更高，通常不赞成这样做。



随着WebApp规模的增长及分析建模变得更加严格，不得不对这里介绍的初步用例进行扩展，以便与8.5节建议的格式更加接近。

用例：描述室内布局

WebApp会问一些有关计划安装的SafeHome所处的环境方面的一般性问题——房间的数目、房间的大小、房间的类型、楼层的数目、外面的门和窗户的数目。通过将每层房间的概貌集中到一起，WebApp可以使我们建立一个粗略的平面布置图。给平面布置图取一个名字，并保存起来供将来参考（见用例：保存配置）。

用例：选择SafeHome部件

WebApp会推荐产品部件（例如，控制面板、传感器、摄像头）及每个房间和外部入口的其他特征（例如，用软件实现的基于PC的功能）。当要求选择时，如果选择的部件存在的话，WebApp就会提供，我们会得到每一个产品部件的描述信息和价格信息。选择了不同的部件之后，WebApp会创建并显示一份材料清单。可以给材料清单取一个名字，并保存起来供将来参考（见用例：保存配置）。

用例：保存配置

WebApp允许保存定制的数据，使得以后能够返回来访问它。也可以保存房间布局及为这种布局所选择的SafeHome材料清单。为了做到这一点，需要为房间布局和材料清单提供唯一的标识符，并设置一个特殊的配置密码。在访问已保存的信息时，必须使用密码。

尽管能够为每一个用例提供更详细的描述，但是非形式化的文本描述对理解用例很有用。在图18-2中，应该为每个椭圆（用例）编写类似的描述。

SAFEHOME

精化WebApp的用例

[场景] Doug Miller的办公室。

[人物] Doug Miller，SafeHome软件工程团队经理；Sharon Woods，SafeHome电子商务网站外包经销商的Web工程团队的经理；Sam Chen，SafeHomeAssured.com客户支持部经理。

[对话]

543

Doug：Sharon，听说事情进展得很顺利，真让人高兴。分析建模差不多完成了吧？

Sharon（微笑）：我们正在努力。只剩下用户层次[图18-1]中的“客户服务人员”的用例未开发完。

Doug（看着Sam）：Sam，你做完了吗？

Sam：做完了。Sharon，我已经用e-mail传给你了，Doug，还抄送给了你。这是它的打印件。（他将几页纸递给了Doug和Sharon。）

Sam：我们看它，是想在顾客通过电话订购时，使用SafeHomeAssured.com网站作为支持工具。我们的电话接线员将填写所有需要的表单，并为顾客处理订单。

Doug：为什么不引导顾客去访问网站呢？

Sam（微笑）：你认为每个人都习惯使用网站，其实不是！许多人仍然喜欢电话，所以我们不得不给他们提供这种选择。但是，大多数功能部件都已经放到网站上了，我们不

想建立一个单独的订购处理系统。

Sharon: 有道理。

(所有组都阅读用例[下面是一个例子])

用例: 描述室内布局[注意, 它不同于“新客户”类别的同名用例。]

我会请顾客(通过电话)描述房屋的每个房间, 并在专门为客户支持人员设计的大表单上记录房屋尺寸和其他特征。一旦输入了房屋数据, 就可以把数据保存在客户的姓名或电话号码下。

Sharon: Sam, 你的初步用例描述得有点简洁。我认为需要使它们充实一点。

Doug (点头): 我同意。

Sam (皱眉): 怎样做?

Sharon: 噢……你提到的“专门为客户提供人员设计的大表单”应该更详细一些。

Sam: 我的意思是我们没有必要让(电话)接线员像在线顾客那样一步一步地完成这个过程, 一个大表单就可以做这些事情。

Sharon: 让我们拟定一下表单的样子吧!

各组都提供了充分的细节, 以便Sharon的团队可以有效地使用用例。

18.1.3 精化用例模型

当为每一类用户建立了用例图时, 从外部可观察到的WebApp需求的最高层次的视图就建立了。将用例组织到功能包中, 之后对每个包进行评估[CON00], 以确保每个包是:

我们如何访问被用户分成不同功能的用例包?

- 可理解的——所有的共利益者理解此包的目的。
- 内聚的——包中涉及的功能彼此密切相关。
- 松散耦合的——包内的功能或类相互之间合作, 但让包外的合作保持最少。
- 层次浅的——深的功能层次很难导航, 最终用户也很难理解, 因此, 用例层次的数目应该尽可能地少。

由于需求分析和建模活动是反复迭代的, 可能会有新用例添加到已定义的包中, 已有的用例会被精化, 特定的用例可能会被重新分配到不同的包中。

544

18.2 WebApp的分析模型

WebApp分析模型是由信息驱动的, 这些信息包含在为应用系统开发的用例中。对用例描述进行分析, 确定可能的分析类及与每个类相关的操作和属性。WebApp所表示的内容是确定的, 执行的功能是从用例描述里抽取出来的。最后, 实现具体的需求, 以便建立支持WebApp的环境和基本设施。

在一个Web-App的建模过程中, 发生了哪些类型的分析活动?

- 内容分析确定WebApp所提供的全部内容, 包括文本、图形和图像、视频和音频数据。
- 交互分析描述用户与WebApp交互的方式。
- 功能分析定义将对WebApp内容所进行的操作, 并描述了独立于内容、却对最终用户很必要的其他处理功能。
- 配置分析描述WebApp所处的环境和基本设施。

对于这四种分析任务期间收集的信息，应该根据需要进行检查、修改，然后将这些信息组织在一个模型里，传给WebApp的设计者。

模型本身包含结构元素和动态元素。结构元素确定分析类和内容对象，用来创建满足共利益者要求的WebApp；分析模型的动态元素描述了结构元素之间以及与最终用户是如何相互作用的。

“成功的WebApp允许客户自己更好、更快、更廉价地满足他们的需求，而不是通过[一个公司的]雇员终端用户。”

——Mark McDonald

18.3 内容模型

545 内容模型包含结构元素，它们为WebApp的内容需求提供了一个重要的视图。这些结构元素包含内容对象（例如，文本、图形图像、照片、视频图像、音频），这些内容对象是WebApp的一部分。另外，内容模型还包括所有分析类——用户可见的实体，当用户与WebApp交互时，这些实体被创建或操作。分析类包含描述它的属性、实现类所需要的行为的操作以及允许这个类与其他类通信的协作。

像分析模型的其他元素一样，通过仔细检查为WebApp所开发的用例，可以得到内容模型；通过对用例进行分析，可以抽取出内容对象和分析类。

18.3.1 定义内容对象



内容对象是展现给最终用户的内聚性信息中的任意一项。内容对象一般是从用例中抽取出来的。

Web应用系统将已有的信息（称为内容）介绍给用户。内容的类型和形式很复杂。内容的开发可能发生在WebApp实现之前、WebApp构建之中，或者WebApp投入运行以后的很长一段时间。在每种情况下，它都通过导航链接合并到WebApp的总体结构中。内容对象可能是产品的文本描述、描述新闻事件的一篇文章、在运动中拍摄的一张动作照片、公司标志（logo）的一种生动表示、一个演讲的简短视频，或者是幻灯片集合的一个音频插曲。

通过检查直接或间接引用内容的场景描述，可以从用例中抽取出内容对象。例如，在用例“选择SafeHome部件”中，我们看到下面这样的句子：

我将会得到每个产品部件的描述信息和价格信息。

尽管没有直接引用内容，但引用内容是隐含存在的。Web工程师会与用例的作者会晤，更加详细地理解“描述信息和价格信息”的含义。在这种情况下，用例的作者可能指出“描述信息”包括：(1) 部件的一段概要描述；(2) 部件的一张照片；(3) 部件的多段技术描述；(4) 一个部件的示意图，展示它怎样适应一个典型的SafeHome系统；(5) 一段极短的视频，展示了怎样在一个典型的家庭环境中安装部件。

值得注意的是，必须开发（通常是内容开发者，而不是Web工程师）或获得每一个内容对象，并集成到WebApp的体系结构中。

“开发Web——那么多的内容，那么少的时间。”

——作者不详

18.3.2 内容关系与层次



数据树代表了内容对象的层次结构。

在许多实例中，用一个简单的列表就可以列出内容对象，并给出每个对象的简短描述。对于定义那些必须被设计和实现的内容需求来说，这已经是足够了。但是，在某些情况下，内容模型可能包括实体关系图（第8章）或数据树 [SRI01]，数据树描绘了内容对象之间的关系和（或）WebApp维护的内容层次。

546

考虑图18-3中为SafeHome部件创建的数据树，该树体现了一种信息层次，这种信息层次描述了部件（后面我们会看到，一个SafeHome部件实际上是这个应用系统的一个分析类）。不带阴影的长方形表示简单或复合数据项（一个或多个数据值），带阴影的长方形表示内容对象。在此图中，**description**（描述）由5个内容对象（有阴影的长方形）定义。在某些情况下，随着数据树的扩展，这些对象中的一个或多个将会得到进一步细化。

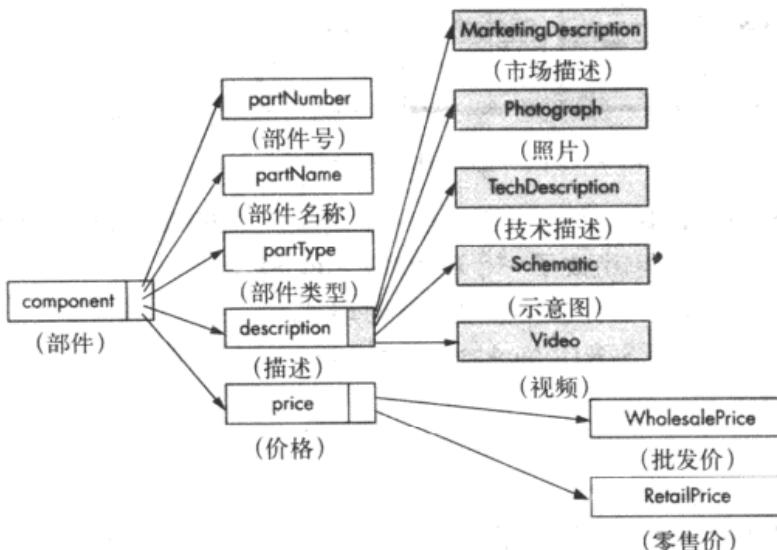


图18-3 SafeHome部件的数据树

18.3.3 WebApp的分析类⁴

正如我们已经注意到的，通过分析每个用例可以得到分析类。例如，考虑在18.1.2节介绍过的初步用例：选择SafeHome部件。

用例：选择SafeHome部件

WebApp会推荐产品部件（例如，控制面板、传感器、摄像头）及每个房间和外部人口的其他特征（例如，用软件实现的基于PC的功能）。当要求选择时，如果选择的部件存在的话，WebApp就会提供，我们会得到每一个产品部件的描述信息和价格信息。选择了不同的部件之后，WebApp会创建并显示一份材料清单。可以给材料清单取一个名字，并保存起来供将来参考（见用例：保存配置）。

对用例进行快速语法分析，可以识别出两个候选类（加下划线的）：**ProductComponent**（产品部件）和**BillOfMaterials**（材料清单）。对每个类的简单描述如图18-4所示。

547

类**ProductComponent**包含所有可以购买的SafeHome部件，以便为特殊的安装定制产品。它是**Sensor**、**Camera**、**ControlPanel**和**SoftFeature**的广义代表。每一个**ProductComponent**

⁴ 对识别和表示分析类机制的详细讨论请看第8章，如果你还没有看，请现在就复习第8章。

对象包含了与图18-3中这个类的数据树所对应的信息。有些类属性是单一的或复合的数据项，而其他的是内容对象（见图18-3）；与类相关的操作也显示出来了。

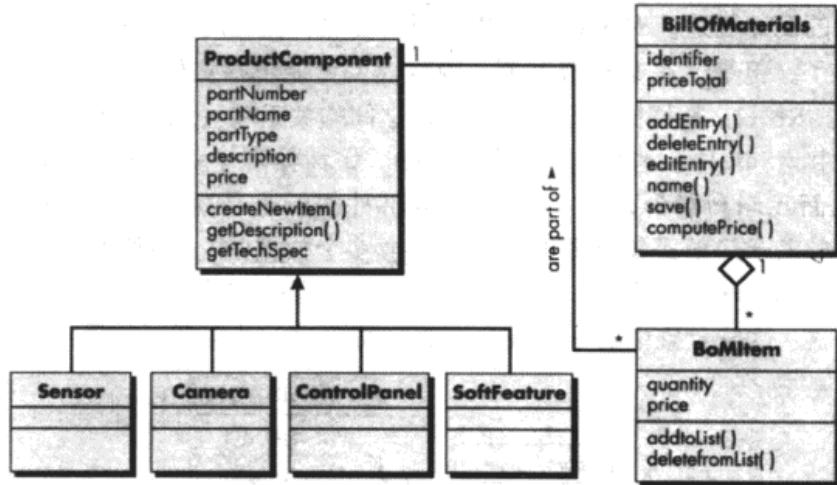


图18-4 用例“选择SafeHome部件”的分析类

类BillOfMaterials包括新客户已选部件的一个列表。BillOfMaterials实际上是BoMItem的聚合（BoMItem的许多实例组成了一个BillOfMaterials）——一个类，它建立了一个由被购买的每个部件组成的列表，部件的特殊属性见图18-4。

为了分析对象，就要剖析SafeHomeAssured.com确定的每个用例，为每个用例开发类似于本节所描述的类模型。

18.4 交互模型

绝大多数的WebApp都能够使最终用户与应用系统的功能、内容及行为“会话”。这种交互模型由四种元素组成：(1) 用例；(2) 顺序图；(3) 状态图⁵；(4) 用户界面原型。除了这些表示，也可以在导航模型中表示交互（18.7节）。

548



与任务分析(第12章)相关的技术可以帮助定义用户交互模型。

用例：用例是WebApp的交互模型中处于支配地位的元素。当分析、设计和构造大型复杂的WebApp时，描述100个或更多的用例是很平常的。但是，其中有一小部分用例描述了不同种类（角色）的最终用户和系统之间的主要交互；而其他用例则细化了交互，提供了指导设计和构造的必要的分析细节。

顺序图：UML顺序图提供了用户动作（由用例定义的系统动态元素）与分析类（类图定义的系统结构元素）之间协作行为的速记表示法。由于分析类是从用例描述中抽取出来的，因此有必要确保已定义类与描述系统交互的用例之间的可追溯性。

在前面的章节中，我们注意到顺序图提供了用例中描述的行为与分析类（结构实体）之间的连接关系。Conallen[CON00]是这样描述的：“在模型的可追溯性中，[分析]模型的动态元素和结构元素的合并是关键连接，应该高度重视。”

用例“选择SafeHome部件”的顺序图如图18-5所示。图的垂直轴描述了在用例中定义的行为，水平轴确定了分析类。当进一步描述用例时，会用到这些类。例如，一个新客户必须首先描述房子内的每个房间（紧随“描述房



在某些情况下，可将来自用例正文描述本身的内容复制到左栏(用户下面)，使得能够表示出直接可追溯性。

⁵ 这些都是重要的UML表示法，在第8章给出了描述。

间”后的星号表示这个行为是重复的)。因此,新客户要回答关于房间的大小、门和窗户等问题。一旦定义了一个房间,就可以将其放到房子的平面布置图里。然后,新客户描述下一个房间,或继续下一个动作(保存平面布置图配置)。顺序图中的动作可以把每个分析类绑定到用例行为上。如果图上遗漏了某个用例行为,Web工程师必须重新评价分析类的描述,确定是否有被遗漏的类。一旦为用例定义了分析类,就要为每个用例创建顺序图。

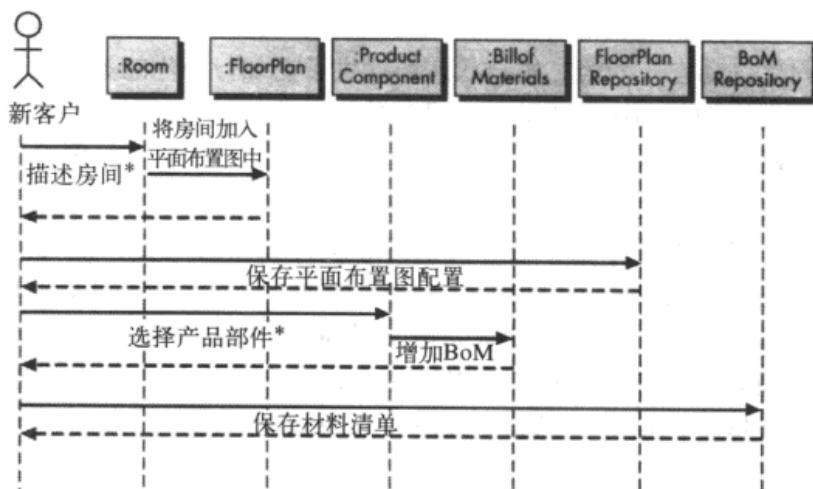


图18-5 用例“选择SafeHome部件”的顺序图

状态图:当交互发生时, UML状态图(第8章)提供了WebApp动态行为的另外一种表示法。像在Web工程(或软件工程)中使用的大多数建模表示法一样,可以在不同的抽象层次上表示状态图。图18-6截取了顶层(高抽象层次)状态图的一部分,描述了一个新客户和SafeHomeAssured.com WebApp之间的交互。

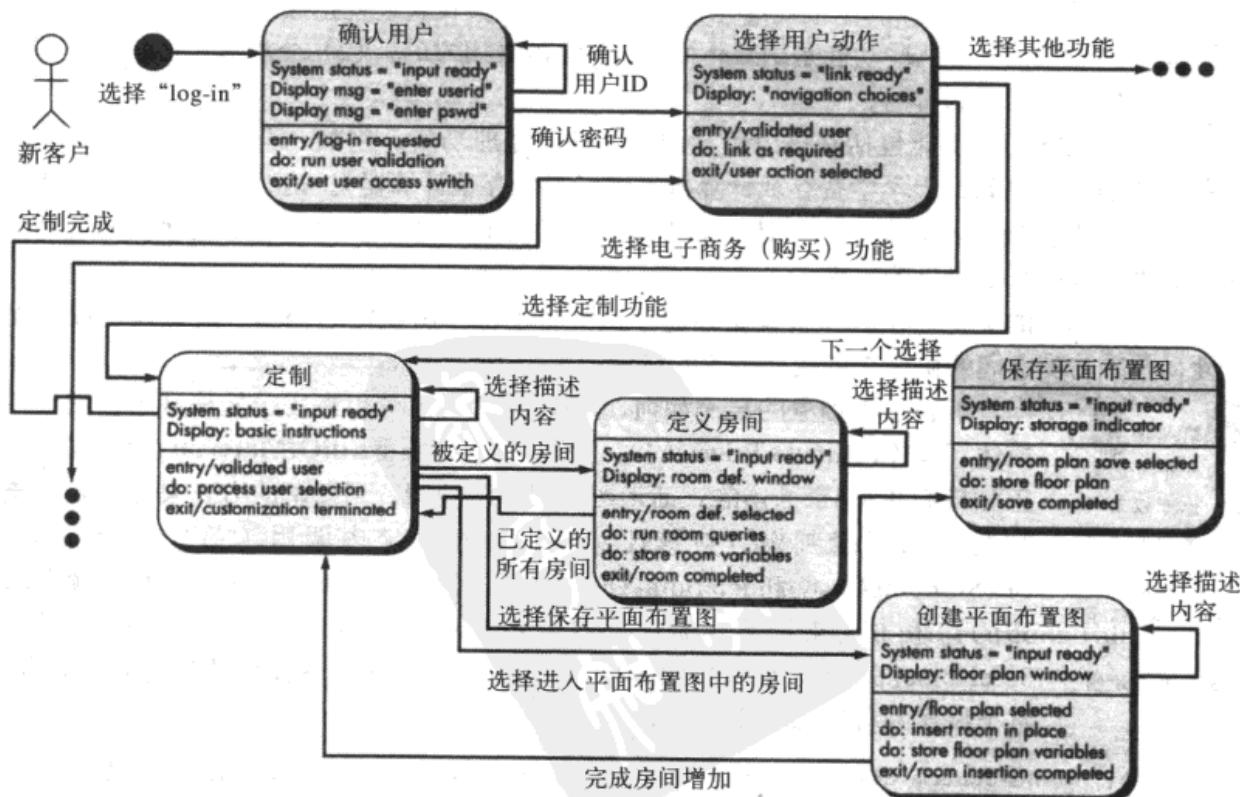


图18-6 新客户交互的部分状态图

在所示的状态图中，确定了6个从外部可以观察到的状态：确认用户、选择用户动作、定制、定义房间、创建平面布置图和保存平面布置图。状态图指明了使新客户从一个状态转移到另一个状态所需要的事件，当进入一个状态时显示的信息，在一个状态内发生的处理，以及引起从一个状态向另一个状态转移的退出条件。

550 用例、顺序图和状态图都表示相关的信息，这三种图都有必要吗？在某些情况下，它们并不都是必要的，只要有用例可能就足够了。然而，用例只提供了交互的一维视图；实际上，顺序图表示了更过程化（动态）的第二维；状态图提供了更加行为化的第三维，它包含了可能的导航路径的信息，而用例和顺序图不能提供导航信息。当三种维度的视图同时使用时，有些冗余和矛盾在一维视图中可能发现不了，但是，当检查第二（或第三）维时，冗余和矛盾就变得非常明显了。因此，包含这三种表示法的交互模型对于分析庞大复杂的WebApp很有用。



有些Web工程师更喜欢使用铅笔和纸质的情节串联图板来描述主要的WebApp页面（屏幕）。尽管可以快速开发这些情节串联图板，但是导航流不如可以运行的原型那样明显。

用户界面原型：用户界面的布局、表示的内容、实现的交互机制以及用户与WebApp连接上的总体审美感都与用户的满意度和WebApp的总体接受能力密切相关。虽然可以认为用户界面原型的创建是一种设计活动，但在分析模型的创建期间创建用户界面原型是一个好办法。对用户界面的物理表示评估得越早，越有可能满足最终用户的需求。用户界面的分析和设计已经在第12章详细讨论过。

WebApp的开发工具种类繁多，而且相对便宜、功能强大，因此最好使用这些工具创建界面原型。原型应该实现主要的导航链接，采用构造它的方式表现总体的屏幕布局。

18.5 功能模型

功能模型描述WebApp的两个处理元素，每个处理元素代表过程抽象的不同层次：（1）用户可观察到的功能——这些功能是由WebApp传递给最终用户的；（2）分析类中的操作——这些操作实现与类相关的行为。

用户可观察到的功能包括直接由用户启动的任何处理功能。例如一个金融网站可以执行多种财政功能（例如，大学学费存款计算器，或者退休存款计算器）。这些功能实际上可能要使用分析类中的操作才能完成，但是从最终用户的角度看，这些功能（更正确地说，是这些功能提供的数据）是可见的结果。

在过程抽象的更低层次，分析模型描述了由分析类操作所执行的处理。这些操作操纵类属性，并参与类之间的协作来完成所需要的行为。



作为一个可选方法，你可以写一个简单的处理描述，或用程序设计语言表示（第11章）。但是，许多人更喜欢图形表示法。

不管过程抽象的层次如何，UML的活动图可用来表示处理细节。图18-7描述了操作computerPrice()的活动图，它是分析类**BillOfMaterials**⁶的一部分。正如我们在第8章描述的，活动图类似于流程图，用图解的方式来说明处理流和其中的逻辑判定。应该注意到，在过程流内调用了另外两个操作：calcShippingCost()和determineDiscount()。操作calcShippingCost()根据顾客选择的运输方法来计算运费，而操作determineDiscount()为选择购买的SafeHome部件确定任何特殊的折扣。构造细节指出这些操作是如何被调用的，每个操作的接口细节要在WebApp设计开始时才考虑。

⁶ 对分析类**BillOfMaterials**评审时可能会这样决定：为了符合内聚性的缘故，computerPrice()操作最好放在类**Invoice**中。这个建议是有好处的。但是，针对本例的目的，我们把它留在了分析类**BillOfMaterials**中。

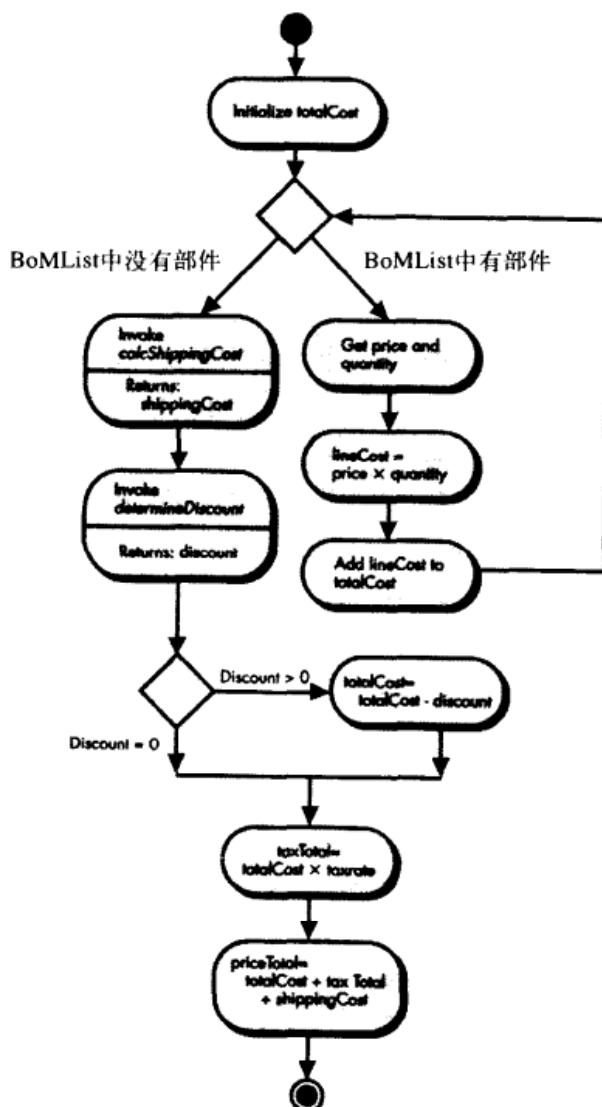


图18-7 computerPrice()操作的活动图

552

18.6 配置模型



尽管考虑所有可能使用的配置是非常重要的，但是要记住：必须对WebApp进行设计，以服务于它的最终用户，而不是适应特定浏览器的特性。

在某种意义上，WebApp的设计和实现一定要适应服务器端和客户端⁷的多种环境。WebApp可以位于服务器上，通过因特网、企业内部网和企业外部网对服务器进行访问。必须详细说明服务器硬件和操作系统环境，此外，还应该考虑服务器端的互操作性。如果WebApp必须访问大型数据库，或者与位于服务器端的公司应用程序进行互操作，则必须详细说明合适的接口、通信协议和相关的协作信息。

客户端软件提供了基础设施，它使得用户可以从所在的位置访问WebApp。一般而言，浏览器用来显示从服务器下载的WebApp内容和功能。尽管有标准，但是每个浏览器有它自己的特性。因此，必须针对每一种浏览器配置（作为

⁷ 服务器端是WebApp及所有相关的系统特性的主机，这些特性使多个用户能够通过网络访问WebApp。客户端提供了一个软件环境(例如，浏览器)，此环境使最终用户能够在用户的桌面上与WebApp交互。

配置模型的一部分详细说明), 对WebApp进行彻底测试。

在某些情况下, 配置模型只不过是服务器端和客户端的属性列表。但是, 对更复杂的WebApp来说, 多种配置的复杂性(例如, 多服务器之间的负载分配、高速缓存的体系结构、远程数据库、同一网页上服务于不同对象的多个服务器)可能对分析和设计产生影响。在必须考虑复杂配置体系结构的情况下, 可以使用UML部署图(第10章)。

18.7 关系导航分析

前面章节中所描述的分析模型的元素确定了内容元素、功能元素及用户执行交互的方式。当由分析发展成设计时, 这些元素就变成了WebApp体系结构的一部分。在Web应用中, 每个体系结构的元素都有可能与所有其他体系结构元素相链接。但是, 随着链接数目的增加, WebApp导航的复杂性也增加了。因而, 问题是如何在内容对象与提供用户所需能力的功能之间建立适当的链接。

“[导航]不仅仅是从一页跳到另一页的行为, 而是在信息空间移动的思想。”

——A. Reina和J. Torres

553

关系导航分析(relationship-navigation analysis, RNA)提供了一系列分析步骤, 用以确定元素之间的关系, 这些元素是在创建分析模型时发现的, 也是分析模型的一部分⁸。Yoo和Bieber[YOO00]是这样描述RNA的:

RNA为系统分析提供了一种系统技术, 可以用来决定一个应用系统的关系结构, 帮助发现应用领域内的所有潜在的有用关系。这些关系后来用链接来实现。RNA也在这些链接上帮助确定合适的导航结构。RNA通过扩大和深化领域的概念模型, 增强系统开发者对应用领域的理解。因而, 开发者可以通过加入附加的链接、元信息和导航使实现得到增强。

RNA方法可以组织成5步:

- 共利益者分析——确定不同的用户种类(正如18.1节描述的), 并建立适当的共利益者层次。
- 元素分析——确定内容对象和最终用户感兴趣的功能元素(正如18.3节和18.5节描述的)。
- 关系分析——描述WebApp元素之间的关系。
- 导航分析——检查用户怎样访问单个元素和成组元素。
- 评估分析——考虑与实现早期定义的关系有关的实际问题(例如, 成本效益比)。

RNA方法中的前两步已经在本章前面讨论过了。在以后的部分, 我们考虑内容对象和功能之间关系的建立方法。

18.7.1 关系分析——关键问题

Yoo和Bieber[YOO00]提出了一个问题列表, Web工程师或系统分析者应该对分析模型中确定的每个元素(内容对象或功能)都提出问题。下面是一个适合WebApp的有代表性的列表[YOO00]:

⁸ 应该注意到, RNA通常可以应用于任何信息系统。RNA最初是为超媒体系统而开发的, 然而, 它也可以很好地适用于Web工程。

 我们如何通过访问分析模型的元素来理解它们之间的关系？

- 此元素属于更广泛的元素种类吗？
- 为此元素确定了哪些属性和参数？
- 有关此元素的描述性信息是否已经存在？如果存在，这些信息在哪里？
- 在WebApp中，此元素在不同的位置出现吗？如果是，这些位置在哪里？
- 此元素包含其他更小的元素吗？如果是，这些更小的元素是什么？
- 此元素是更大的元素集合中的成员吗？如果是，这个更大的集合是什么？它具有什么样的结构？
- 此元素被某个分析类描述了吗？
- 其他元素与此元素相似吗？如果相似，可能将它们合并为一个元素吗？
- 此元素是在其他元素的特殊顺序中使用吗？它的出现依赖其他元素吗？
- 此元素一出现，总是跟着出现另外一个元素吗？
- 此元素必须满足的前置条件和后置条件是什么？
- 有特殊的用户类别使用该元素吗？不同的用户类别使用该元素的方式也会不同吗？如果是这样，怎样不同？
- 能够将这个元素与一个具体的、明确表达的目的或目标联系在一起吗？还是能够与一个具体的WebApp需求联系在一起？
- 当其他元素出现时，这个元素总是会同时出现吗？如果会同时出现，其他元素是什么？
- 这个元素总是和其他元素出现在同一个地方（例如，屏幕或页的相同位置）吗？如果是，其他元素是什么？

554

对这些问题和其他问题的回答有助于Web工程师定位WebApp内不确定的元素，并建立元素之间的关系。

可以建立一种关系分类方法，针对每一个显著的问题，对每个识别出来的关系进行归类。感兴趣的读者请参考[YOO00]了解更多细节。

18.7.2 导航分析

一旦建立了分析模型内定义的元素之间的关系，Web工程师必须考虑一些需求，这些需求规定了用户种类如何从一个元素（例如，内容对象）导航到另一个元素。导航机制被定义为设计的一部分。在这个阶段，开发人员应该考虑总体的导航需求，可以提出并回答以下问题：

 为了更好地了解导航需求，应该提出哪些问题？

- 某些元素比其他元素更容易到达吗（需要更少的导航步骤）？表示的优先级别是什么？
- 为了迫使用户在自己的方向上导航，应该强调某些元素吗？
- 应该怎样处理导航错误？
- 导航到相关的元素组的优先权应该高于导航到某个具体元素的优先权吗？
- 应该通过链接、基于搜索的访问还是其他手段来实现导航？
- 根据前面的导航行为，某些确定的元素应该展现给用户吗？
- 应该为用户维护导航日志吗？
- 在一次用户交互中，一个完整的导航地图或菜单（与单个“返回”链接或有方向的指针相比）在每点都可用吗？

555

- 导航设计应该由大多数普遍期望的用户行为来驱动，还是由已定义的WebApp元素可感知的重要性来驱动？
- 为了加速将来的使用，一个用户能否“存储”他以前对WebApp的导航？
- 应该为哪类用户设计最佳导航？
- 应该如何处理WebApp外部的链接？应该覆盖现有的浏览器窗口，作为一个新的浏览器窗口，还是作为一个单独的框架？

提出并回答这些问题及其他一些问题是导航分析的一部分。

Web工程团队和它的共利益者必须决定导航的总体需求。例如，提供的“站点图”会让用户全面纵览整个WebApp的结构吗？用户会使用“导游”吗？“导游”将突出显示可以使用的最重要的元素（内容对象和功能）吗？用户能够访问内容对象或者由那些元素的属性所决定的功能吗（例如，用户可能想要访问一个特定建筑的所有图片，或者允许计算重量的所有功能）？

18.8 小结

表达、需求收集和分析建模是WebApp需求分析的一部分。这些活动的目的是：(1) 描述WebApp的基本动机（目的）和目标；(2) 定义用户的种类；(3) 记录WebApp的内容和功能需求；(4) 初步了解为什么要构建WebApp，谁将使用它以及它为用户解决哪些问题？

用例是所有需求分析和建模活动的催化剂。可以将用例组织为功能包，并且对每个包进行评估，以确保它是可理解的、内聚的、松散耦合的和层次浅的。

556 创建一个完整的分析模型需要4种分析活动：内容分析确定WebApp的完整的内容范围，交互分析描述了用户与WebApp的交互行为；功能分析定义了对WebApp的内容所进行的操作，描述了独立于WebApp的内容然而却对最终用户很有必要的其他处理功能；配置分析描述了WebApp所处的环境和基础设施。

内容模型描述了那些要合并到WebApp中的内容对象的范围。开发或获得这些内容对象，并将它们合并到WebApp的体系结构中。数据树可以用来代表内容对象的层次结构。分析类（来自用例）提供了另外一种方法来表示WebApp将要操作的关键对象。

交互模型由用例、UML顺序图和UML状态图组成，它描述了用户和WebApp之间的“会话”。除此之外，可以构造一个接口原型帮助我们设计布局和导航需求。

功能模型使用UML活动图来描述用户可观察到的功能和类操作。配置模型描述了WebApp在系统的服务器端和客户端所需要的环境。

关系导航分析确定了在分析模型中所定义的内容元素和功能元素之间的关系，并且建立了在整个系统中定义恰当的导航链接的需求。提出一系列问题可以帮助建立关系，并确定那些对导航设计产生影响的特征。

参考文献

- [CON00] Conallen, J., *Building Web Applications with UML*, Addison-Wesley, 2000.
- [FRA01] Franklin, S., "Planning Your Web Site with UML," *webReview*, available at http://www.webreview.com/2001/05_18/developers/index01.shtml.
- [SRI01] Sridhar, M., and N. Mandyam, "Effective Use of Data Models in Building Web Applications," 2001, available at <http://www2002.org/CDROM/alternate/698/>.

- [YOO99] Yoo, J., and M. Bieber, "A Systematic Relationship Analysis for Modeling Information Domains," 1999, download from <http://citeseer.nj.nec.com/312025.html>.
- [YOO00] Yoo, J., and M. Bieber, "Toward a Relationship Navigation Analysis," *Proc. 33rd Hawaii Conf. On System Sciences*, vol. 6., IEEE, January 2000, download from www.cs.njit.edu/~bieber/pub/hicss00/INWEB02.pdf.

习题与思考题

- 18.1 对用例或用例包进行评估，确保它们是可理解的、内聚的、松散耦合的和层次浅的。用你自己的话描述这些项的含义。
 - 18.2 如果你被迫进行“轻量级的分析建模”（也就是最简单的分析建模），在Web工程活动中，你将定义哪些表示、图和信息？
 - 18.3 利用Web上的大量敏捷软件开发资源做一点分析研究，并得出反对WebApp分析建模的理由。你认为你的理由适合所有情况吗？
 - 18.4 一个用例包代表什么？
 - 18.5 使用类似于图18-1所示的图，分别为下面两个网站建立用户层次：
 - (a) 一个金融服务站点；
 - (b) 一个售书站点。
- [557]
- 从以下种类中选择一个你定期访问的WebApp：(a) 新闻或体育；(b) 娱乐；(c) 电子商务；(d) 游戏；(e) 与计算机相关的内容；(f) 你导师推荐的一个WebApp。
- 完成习题18.6~18.12所描述的活动。
- 18.6 建立一个或多个用例来描述WebApp特定的用户行为。
 - 18.7 选择一个内容对象或功能（是WebApp体系结构的一部分），并回答18.7.1节列出的关系导航问题。
 - 18.8 建立一个UML顺序图和一个UML状态图，来描述WebApp内的一次具体交互。
 - 18.9 考虑已有的WebApp接口，对此接口进行修改，使其得到改进（你认为的），并对修改后的接口开发原型。
 - 18.10 考虑已有的WebApp，回答18.7.2节列出的关系导航问题。
 - 18.11 描述一个局部的内容层次，并为此WebApp至少定义3个分析类。
 - 18.12 选择WebApp提供的用户可观察到的功能，并用UML活动图对它建模。

推荐读物与阅读信息

许多书都致力于传统软件的分析建模（特别强调用例和UML表示法），包含了许多有用的信息，Web工程师很容易对这些信息进行适应性修改。用例构成了WebApp分析建模的基础。Kulak和他的同事（《Use Cases: Requirements in Context, second edition》，Addison-Wesley, 2004）、Bittner和Spence（《Use Case Modeling》，Addison-Wesley, 2002）、Cockburn（《Writing Effective Use Cases》，Addison-Wesley, 2001）、Armour和Miller（《Advanced Use-Case Modeling: Software Systems》，Addison-Wesley, 2000）、Rosenberg和Scott（《Use Case Driven Object Modeling with UML: A Practical Approach》，Addison-Wesley, 1999）以及Schneider、Winters和Jacobson（《Applying Use Cases: A Practical Guide》，Addison-Wesley, 1998）的书对重要的需求表示机制的创建和使用提供了有价值的指导。Arlow和Neustadt

(《UML and the Unified Process》, Addison-Wesley, 2002)、Schmuller (《Teach Yourself UML》, Sams Publishing, 2002)、Booch和他的同事 (《The UML User Guide》, Addison-Wesley, 1998) 以及Rumbaugh和他的同事 (《The Unified Modeling Language Reference Manual》, Addison-Wesley, 1998) 的书对UML进行了有价值的讨论。

致力于网站设计的书籍通常用一章或两章来讨论分析问题 (尽管是粗略的讨论)。以下书籍包括Web工程环境中的分析问题的一个或多个方面: Van Duyne和他的同事 (《The Design of Sites》, Addison-Wesley, 2002)、Rosenfeld和Morville (《Information Architecture for the World Wide Web》, O'Reilly & Associates, 2002)、Wodtke (《Information Architecture》, New Riders Publishing, 2002)、Garrett (《The Elements of User Experience: User Centered Design for the Web》, New Riders Publishing, 2002)、Niederst (《Web Design in a Nutshell》, O'Reilly & Associates, 2001)、Lowe和Hall (《Hypertext and the Web: An Engineering Approach》, Wiley, 1999) 以及Powell (《Web Site Engineering》, Prentice Hall, 1998) 的书提供了相当全面的论述。Norris、West和Watson (《Media Engineering: A Guide to Developing Information Products》, Wiley, 1997)、Navarro和Khan (《Effective Web Design: Master the Essentials》, Sybex, 1998) 以及Fleming和Koman (《Web Navigation Designing the User Experience》, O'Reilly & Associates, 1998) 的书对分析与设计提供了额外的指导。

可以在Internet上获得关于Web工程分析建模的各种信息资源。一个最新的WWW参考列表可在SEPA Web站点<http://www.mhhe.com/pressman>的“software engineering resources”栏目下找到。



第19章 WebApp设计

要点浏览

概念：WebApp设计包括技术性活动和非技术性活动。内容的外观和感觉是美术设计的一部分；用户界面的美学设计是界面设计的一部分；WebApp的技术结构是体系结构设计和导航设计的一部分。在每一个实例中，应该在构造开始之前创建设计模型，但是好的网络工程师认识到：在构造WebApp的同时，设计工作将会随着更多地了解共利益者的需求而不断进化。

人员：Web工程师、美术设计师、内容开发者及其他共利益者都参加Web工程设计模型的创建。

重要性：设计工作允许Web工程师创建模型，可以对该模型进行质量评估，并且可以在内容和编码生成之前、在测试开始之前、以及在最终用户大量参与之前对该模型进行改进。通过设计达到WebApp质量要求。

步骤：WebApp设计包括6个主要步骤，这些步骤由分析建模阶段所获取的信息驱动。内容设计利用从分析模型中获取的信息作为设计内容对象及它们之间关系的基础；美学设计（也称为美术设计）建立了最终用户所关注的外观和感觉；结构设计重点关注所有内容对象和功能的总体超媒体结构；界面设计创建了定义用户界面的总体布局和交互机制；导航设计定义了最终用户是怎样对超媒体结构进行导航的；构件设计表示了WebApp功能元素的详细内部结构。

工作产品：设计模型包括内容、美学、体系结构、界面、导航及构件级设计问题，它是Web工程设计的主要工作产品。

质量保证措施：Web工程团队（及选择的共利益者）对设计模型的每一个元素进行评估，尽力发现错误、不一致或者遗漏的地方。另外，考虑可选的解决方案，并对当前设计模型有效实现的程度进行评估。

关键概念
美学设计
体系结构设计
构件级设计
内容体系结构
内容设计
界面设计
MVC体系结构
导航设计
OOHDM
度量
模式
质量属性

Jakob Nielsen[NIE00]在他的有关Web设计的权威著作中这样写道：“本质上，设计有两种基本的途径：表达你自己的艺术设想和为客户解决问题的工程设想。”在Web发展的头10年，艺术设想是很多开发者选择的方式。以一种特别的方式进行设计，并且通常是在生成HTML时才进行设计。设计从艺术想像力发展而来，而艺术想像力本身又是随着WebApp结构的出现而发展的。

即使在今天，敏捷软件开发（第4章）的大多数“极端”支持者仍然使用Web应用系统向孩子们宣传“有限设计”。他们认为WebApp的直接性和易变性削弱了形式化设计，即设计是随着对应用系统进行构造（编码）而进化的，并且应该花费较少的时间创建详细设计模型。这种论述有其优点，但是只适用于相对简单的WebApp。当内容和功能变得复杂时，当WebApp的规模包含成百上千的内容对象、函数和分析类时，当WebApp的成功对于业务成功具有直接影响时，就不能轻视设计，也不该轻视设计。

这种情况将使我们考虑Nielsen提出的第二种途径——“为客户解决实际问题的工程设想。”
[559] Web工程采纳了这一思想，并且一种更严格的Web设计方法使开发人员能够实现这种设想。

19.1 Web工程的设计问题

当把设计应用到Web工程环境中时，不论是一般问题，还是特殊问题，我们都必须考虑。一般认为，用设计所产生的模型来指导WebApp的构造。无论设计模型的形式如何，它都应该包括足够的信息来反映共利益者的需求（在分析模型中定义）是如何转化为内容和可执行代码的。但是设计又必须有其特殊性，它必须以某种方式来说明关键的WebApp特性，即能够让Web工程师有效地创建和测试的方式。

19.1.1 设计与WebApp质量

在前面几章中，我们谈到设计是产生高质量产品的工程活动。这使我们回到了在各个工程学科都会碰到并且会反复出现的问题：什么是质量？在本节中，我们在Web工程的背景下回答这一问题。

每个上网或者用过内部网的人都对什么是“好的”WebApp有自己的看法，大家看待这一问题的角度也相差甚远。有些用户喜欢闪烁的图片，有些人则喜欢简单的文本；有些用户想看到丰富的内容，而有些人只是渴望看到简略的陈述；有些人喜欢高级的分析工具或者数据库访问，而有些人只是需要一些简单应用。实际上，用户对于“好的WebApp”的理解（基于这种理解而接受或拒绝WebApp），比起从技术角度讨论WebApp的质量，前者可能更重要。

如何认识WebApp的质量呢？具有哪些特性的WebApp才能得到最终用户的好评？同时，在质量方面，具有哪些技术特点才能使Web工程师长期对应用系统进行修正性维护、适应性维护、增强性维护及支持？
[560]

实际上，在第9章、第15章和第26章讨论的软件质量的所有一般特性都适用于WebApp。然而，其中一些最相关的特性（可用性、功能性、可靠性、效率及可维护性）为评估基于Web的系统的质量提供了有用基础。

“如果产品设计的目的是为了更好地符合人类行为的自然趋势，人们将会更满足、更有成就、更多产。”

——Susan Weinschenk

 WebApp质量的主要属性是什么？

Olsina和他的同事[OLS99]设计了一个“质量需求树”，它定义了一组可产生高质量WebApp的技术属性，包括可用性、功能性、可靠性、效率和可维护性¹。图19-1总结了他们的工作。在图中提及的标准对于必须长期从事设计、构造和维护WebApp的工程师是有帮助的。

Offutt[OFF02]又补充了下面的属性，对图19-1描述的5个质量属性进行了扩展。

安全性：WebApp已经和重要的公司及政府数据库高度集成。电子商务应用系统提取敏感的客户信息，然后将这些信息存储起来。由于这些及许多其他原因，WebApp的安全性在很多情况下变得极为重要。安全性的关键度量标准是WebApp和服务器环境拒绝非授权访问和（或）

¹ 这些质量属性与第9章、第15章和第26章所描述的质量属性非常相似。暗示：质量特性对于所有软件都是通用的。

阻挡恶意攻击的能力。WebApp安全的详细讨论超出了本书的范围，感兴趣的读者可以参考[MCC01]、[NOR02]或[KAL03]。

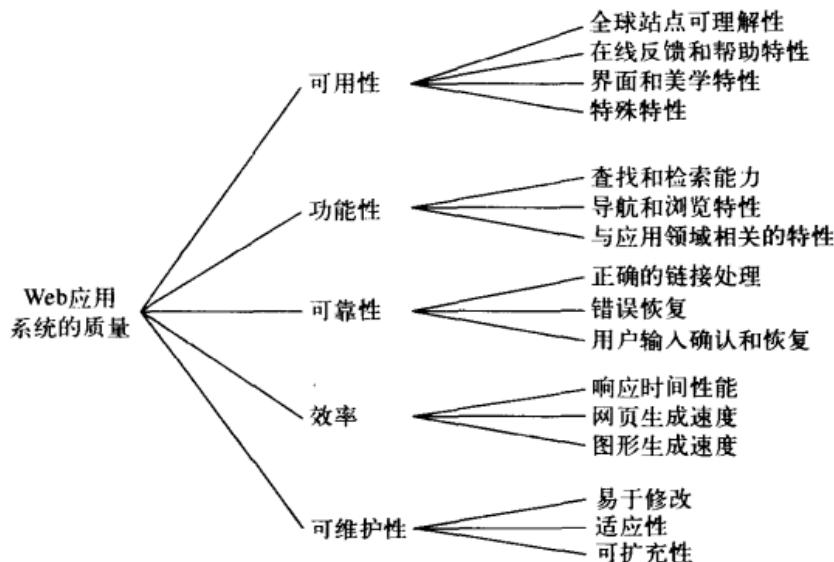


图19-1 质量需求树[OLS99]

561

可得性：如果不可得的话，即使是最好的WebApp也不能满足用户的需求。从技术的角度说，可得性是对WebApp的可用时间占总时间的百分比的一种度量。一般最终用户期望WebApp一天24小时、一周7天、一年365天都是可得的（常简写为24/7/365），对可得性的任何减少都是不可接受的²。但是，“正常运行时间”并不是可得性的唯一指标。Offutt[OFF02]认为：“使用仅限于在一种浏览器或平台上可得的特性”会使WebApp在那些具有不同的浏览器或平台的配置中变得不可得，用户会毫无例外地转向其他地方。

可伸缩性：WebApp和其服务环境能否伸缩来处理100个、1000个、10000个或100000个用户？WebApp和为其提供接口的系统能不能承受访问数量上的巨大波动？响应速度是否会因此而剧减（或者完全停止）？开发成功的WebApp是远远不够的，开发能够成功调节负载（相当多的最终用户）的WebApp同样重要，而且会变得越来越重要。

面市时间：虽然面市时间并不是真正的、技术方面的质量属性，仅仅是从商业角度考虑的一种质量度量。但是，市场上的第一个WebApp往往能够吸引非常多的最终用户。

INFO

WebApp设计质量清单

下面的清单是根据Webreference.com上所展示的信息修改而来的，清单中列出了一组问题来帮助Web工程师和最终用户评估总体的WebApp质量。

- 内容、功能和（或）导航选项能否按照用户的喜好而定制？
- 内容和（或）功能能否按照用户通信所用的带宽进行定制？
- 图形和其他非文本媒体能否正确使用？是否出于显示效率方面的考虑而对图形文件的大小进行了优化？
- 是否用可以理解的、有效显示的方式来组织表格，并按大小进行排序？

² 当然，这种期望是不现实的。大多数的WebApp必须安排“停机时间”以进行安装和升级。

- 是否对HTML进行优化来消除低效率?
- 总体页面设计是否容易阅读和导航?
- 是否所有的指针(链接)都提供了指向用户感兴趣信息的链接?
- 是否大部分的链接在Web中都具有持久性?
- WebApp是否提供了站点管理工具?包括使用跟踪、链接测试、本地搜索和安全性工具。

在WWW上查找信息的人们可以获得数亿的网页。即使很好地命中目标的Web查找也会得到大量内容。要从这么多的信息源中选择需要的信息,用户如何评价WebApp所展示的内容的质量(例如,准确性、精确性、完整性、适时性)呢?Tillman[TIL00]提出了评价内容质量的一组有用标准:

562

-  在评价内容质量时,我们应该考虑什么?
- 能否很容易地判断内容的范围和深度,确保满足用户的需求?
 - 是否容易确定内容作者的背景和权威性?
 - 能否决定内容的通用性?最后的更新时间及更新内容是什么?
 - 内容和位置是否稳定(即它是否一直保存在引用的URL处)?

除了这些与内容相关的问题,下面的一些问题也需要考虑:

- 内容是否可信?
- 内容是否独特?也就是说,WebApp能否给使用它的用户带来一些特别的好处?
- 内容对于目标用户群体是否有价值?
- 内容的组织是否合理?是否有索引?是否容易存取?

这一部分提及的清单只是设计WebApp时应该考虑的问题中的一小部分。Web工程的一个重要目标是开发对所有与质量相关的问题都能提供肯定回答的系统。

“仅仅只是因为你能做,并不意味着你应该这么做。”

——Jean Kaiser

19.1.2 设计目标

在Web设计的定期专栏中,Jean Kaiser[KAI02]提出了下面的设计目标,无论应用的领域、规模和复杂度如何,这些目标实际上可以适用于任何WebApp。

简单性:虽然可能有些过时,但是格言“一切都要适度”也适用于WebApp。设计者们倾向于给用户提供“太多的东西”——详尽的内容、完美的视觉效果、插入的动画、大量的网页等等,但是最好还是尽量做到适度和简单。

一致性:这一设计目标几乎适用于设计模型的每一个元素。内容构造应该一致(例如,在所有相关的文档文件中,文本格式和字体风格都应该保持一致;图形应该有统一的外观、颜色配置和风格)。美术设计(美学方面)应该在WebApp的各部分有统一的外观。体系结构设计应该建立一个能够产生一致的超媒体结构的模板。界面设计应该定义一致的交互、导航和内容显示模式。应该在所有的WebApp元素中一致地使用导航机制。

563

相符性(identity):WebApp的美学、界面和导航设计必须与将要构造的应用系统所处的领域保持一致。毫无疑问,嬉蹦乐组织(hip-hop group)的网站与提供财务服务的公司主页在外观和感觉上肯定不同。WebApp的体系结构会完全不同,界面会被构造成适合不同的用户

种类，导航会被组织为完成不同的目标。Web工程师（以及其他设计参与者）应该通过设计来建立WebApp的相符合性。

健壮性：在已经建立的相符合性的基础上，WebApp通常会给用户明确的“承诺”。用户期待与他们的要求相关的内容和功能，如果这些元素遗漏或不足，WebApp很可能会失败。

导航性：我们已经在前面提及了导航应该简单和一致，也应该以直观的和可预测的方式来设计。也就是说，用户不必搜索导航链接和帮助就知道如何使用WebApp。

视觉吸引：在所有类型的软件中，Web应用系统毫无疑问是最具有视觉效果的、最生动的，也是最具有审美感的。美丽的外观（视觉吸引）无疑是最吸引观看者的眼球的，然而许多设计特性（例如，内容的外观、界面设计、颜色协调、文本布局、图片和其他媒体、导航机制）也会对视觉吸引产生影响。

兼容性：WebApp会应用于不同的环境（例如，不同的硬件、Internet连接类型、操作系统、浏览器），并且必须互相兼容。

“对于有些人来说，Web设计注重视觉效果……而对于另外一些人来说，Web设计是通过文档空间来构造信息和导航，还有些人可能会把Web设计看成是一种构造交互式Web应用系统的技术。实际上，设计应该包括所有这些内容，并且可能比这些还多。”

——Thomas Powell

19.2 WebE设计金字塔

Web工程环境中的设计是指什么？这个问题比想像的更难于回答。设计产生了一个模型，此模型包含了美学、内容和技术的适当混合，这种混合会由于WebApp性质的不同而不同，因而着重强调的设计活动也会有区别。



WebE包括6种不同类型的设计。每种设计都对WebApp的整体质量有影响。

图19-2描述了Web工程的设计金字塔，金字塔的每一层都表示一种设计活动：

- **界面设计**——描述了用户界面的结构和组织形式，包括屏幕布局、交互模式定义和导航机制描述。
- **美学设计**——也称为美术设计，描述了WebApp的“外观和感觉”，包括颜色配置，几何图案设计，文字大小、字体和位置，图形的使用及相关的美学决策。
- **内容设计**——针对作为WebApp组成部分的所有内容，定义其布局、结构和轮廓，建立内容对象之间的关系。
- **导航设计**——针对所有的WebApp功能，描述内容对象之间的导航流程。
- **体系结构设计**——确定WebApp的所有超媒体的结构。
- **构件设计**——开发实现功能构件所需要的详细处理逻辑。

在下面几节中将详细讨论以上这些设计活动。

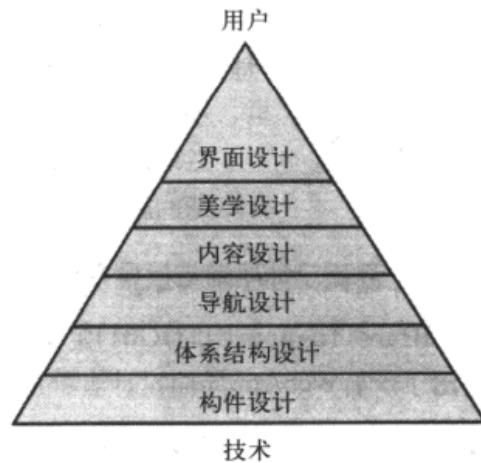


图19-2 WebE设计金字塔

19.3 WebApp界面设计³

无论是为WebApp、传统的软件应用、消费产品设计的用户界面，还是为工业设备设计的用户界面，每个用户界面都应该展示出这样的特性：容易使用，容易学习，容易导航，直观、一致、有效，没有错误，并且功能性强。它还应该为最终用户提供令人满意的和有益的经验。
565 界面设计的概念、准则和方法给Web工程师提供了实现这些特性的工具。

在第12章，我们谈到界面设计并不首先考虑技术方面，而是首先对最终用户进行认真考察。在Web工程的分析建模（第18章）过程中，要针对不同层次的用户，每一类用户可能会有稍微不同的要求，可能会以不同的方式要求与WebApp交互，或者要求独特的功能和内容。这种信息在需求分析中产生，但在界面设计的第一步会再次考虑。

“如果一个站点非常好用，但却缺少美观、合适的设计风格，同样会失败。”

——Curt Cloninger



如果用户可能在
内容层次上的不
同位置和不同层
次进入你的
WebApp，就要
确保设计的每
一个页面都具有导
航特性，引导用
户到达其他感兴
趣的地方。

Dix[DIX99]认为Web工程师设计的界面必须能够为最终用户回答3个主要问题：

我在哪里？界面应该：(1) 为访问过的WebApp提供指示⁴；(2) 提示用户当前在内容层次中所处的位置。

我现在能做什么？界面应该总是能够帮助用户理解当前的选项——哪些功能可以使用？哪些链接是可用的？哪些内容是相关的？

我去过哪里？我将去哪里？界面必须能够辅助导航，因而必须提供一张“地图”（以容易理解的方式实现），这张地图显示了用户在WebApp中去过哪里，还能沿着哪些路径去WebApp的其他地方？

当最终用户通过内容和功能导航时，一个有效的WebApp界面必须回答所有这些问题。

19.3.1 界面设计原则与指导方针

Bruce Tognazzi [TOG01]定义了所有界面都应该具有的一组基本特性，为了做到这一点，提出了所有WebApp界面设计者都应该遵循的指导原则。



好的WebApp界
面是易于理解的
和宽容的，让用
户有一种控制感。

有效的界面在视觉效果上是明显的、宽容的，并且慢慢地给用户灌输控制感。用户能够很快地看到他们的选择范围，领会怎样达到他们的目标，然后做他们的工作。

有效的界面使用户不必关心系统的内部操作。工作被谨慎而连续地保存，从而使用户有充分的选择余地，可以在任何时刻取消任何活动。

有效的应用和服务从用户那里要求最少的信息，而完成最多的工作。

为了设计具有这些特性的界面，Tognazzi[TOG01]确定了一组非常重要的设计准则⁵：

³ 在第12章中提出的指导原则，即使不是全部也是大多数都适用于WebApp的界面设计。如果还没有阅读第12章，最好读一下这一章。

⁴ 我们都曾经给网页标过书签，只是为了稍后再访问，书签并没有网站或网页上下文的指示（也无法连接到网站的其他位置）。

⁵ 本书对Tognazzi的最初准则进行了修改与扩展，这些原则的进一步讨论见[TOG01]。



WebApp界面的设计应符合这里提到的一系列准则。

预测——对WebApp进行设计，使其能够预测出用户的下一个步骤。例如，考虑一种客户支持的WebApp，它是由计算机的打印机制造商开发的。假设用户已经请求了一个内容对象，此对象显示出针对最新发布的操作系统的打印机的驱动程序的信息。WebApp的设计者应该预测出用户可能会请求下载该驱动程序，并且应该提供下载的导航辅助，而无需用户查找这一功能。

传达——界面应该能够传达由用户启动的任何活动的状态。传达可以是直接的（例如，一条文本消息），也可以是间接的（例如，在打印机中移动的纸张表明打印机正在工作）。界面也应该传达用户的状态（例如用户的身份）及在WebApp内容层次中的位置。

一致——导航控制、菜单、图标和美学风格（例如，颜色、形状和布局）的使用应该在整个WebApp中保持一致。例如，如果带有蓝色下划线的文本表示链接，内容中不应该出现不表示链接的蓝色下划线。界面的每一个特征的反应都应该使用与用户的期望相一致的方式⁶。

自治——界面应该辅助用户在整个WebApp中移动，但是应该坚持使用已经为应用建立起来的导航习惯，以这样的方式来辅助用户。例如，对WebApp保密部分的导航应该受到用户ID和密码的控制，而不应该提供能使用户改变这种控制的导航机制。

效率——WebApp的设计和界面应该优化用户的工作效率，而不是优化设计与构造WebApp的Web工程师的效率，也不是优化运行WebApp的C/S环境的效率。Tognazzi[TOG01]在讨论这一问题时写道：“这个简单的事实就是，为什么对于参与软件项目的每一个人来说，认识到提高用户生产率目标的重要性及理解开发有效率的系统和提高用户效率的根本区别非常重要。”

灵活性——界面应该足够灵活，既能够使其中一些用户直接完成任务，也能够使另一些用户以一种比较随意的方式浏览WebApp。在每一种情况下，界面能够使用户认识到自己在哪里，并且给用户提供撤销错误及从选错的导航路径返回的功能。

集中——WebApp界面（及界面表示的内容）应该集中在用户正在完成的任务上。在所有的超媒体中，经常倾向于将用户引导到并不紧密相关的内容。为什么会这样呢？因为这很容易做到！问题是，在支持信息的很多层次中，用户会很快迷失方向，并且会错过最希望看到的最初内容所在的站点。

WebRef

在Web上进行查找会发现很多可用的库，例如，java.sun.com上的Java API包、接口和类，或者msdn.Microsoft.com上的COM、DCOM和类型库。

Fitt规则——“到达目标所用的时间是到达目标的距离和目标规模的函数”[TOG01]。在上个世纪50年代的研究基础上[FIT54]，Fitt规则“是对快速和有目的的活动进行建模的有效方法，在这一活动中，一个附属肢体（如同手）在某个起始位置开始，并移动到目标区域停下来”[ZHA02]。如果一个用户任务定义了选项或标准化输入的顺序（选项有很多不同的排列顺序），第一个选择（例如，鼠标选择）物理上应该与下一个选择挨在一起。例如，考虑销售消费电器的电子商务网站上的WebApp主页的界面。

每一个用户选项暗示了一组顺序执行的用户选项或活动。例如“购买商品”的选项需要用户首先输入产品的类别，然后是产品名。一旦选择了“购买商品”，产品类别（例如，音响设备、电视机、DVD播放器）就以下拉菜单的形式显示，因而，下一个选择将立即出现（在附近），获得此选项所用的时间是可以忽略的。另一方面，如果该选项出现在屏幕另一边的菜单上，用户获得选项（并做出选择）的时间就会变得很长。

⁶ Tognazzi[TOG01]谈到：确保用户的期望被正确理解的唯一方式是通过全面的用户测试（第20章）。

人机接口对象——大量可复用的人机接口对象库已经开发出来。使用这些对象库，一些被最终用户所能“看到的、听到的、接触到的以及别的方式感知到的”[TOG01]接口对象能从大量对象库中获得。

缩短等待时间——WebApp不应该让用户等待内部操作的完成（例如，下载一个复杂的图形图像），而应该利用多任务处理方式，从而使用户继续他的处理工作，看起来就像前面的操作已经完成一样。除了减少等待时间，如果有延迟事件发生，则必须通知用户，从而使用户知道正在发生的事情，包括：(1) 在选中选项后，如果WebApp没有立即做出响应，则应该提供声音反馈（例如，点击声或者铃声）；(2) 显示一个动态时钟或进度条表示处理工作正在进行中；(3) 当处理过程很长时，提供娱乐活动（例如，动画或文本演示）。

“最好的旅程应该有最少的步骤，能够缩短用户和他们要到达的目标之间的距离。”

——作者不详

568

学习能力——应该设计WebApp的界面，将学习时间减到最少，并且一旦已经学习过了，当再次访问此WebApp时，将所需要的再学习时间减到最少。一般而言，界面应该侧重于简单、直观的设计，将内容和功能分类组织，这样对于用户来说很直观。



比喻是一种出色的想法，因为比喻能够反映现实世界的经验。只是要确保你选择的比喻是最终用户所熟悉的。

比喻——当比喻适合应用系统和用户的时候，使用交互比喻的界面更容易学习和使用。比喻应该采用用户熟悉的图片和概念，但是并不要求是现实生活的精确再现。例如，为金融机构实现自动账单支付的电子商务网站使用支票簿（不会令人感到惊讶）来协助用户详细说明和安排账单支付活动。在用户“填写”支票时，他不用输入完整的收款人的名字，而是从提供的收款人的名单中选择，或者通过输入名字的前几个字母就能得到系统的选择提示。虽然比喻的方式没什么变化，但是用户从WebApp中得到了帮助。

保持工作产品的完整性——工作产品（例如用户填写的表单，用户专用数据清单）必须自动保存，使得在有错误发生时数据不会丢失。我们都有过这样不愉快的经历：填写完一个冗长的WebApp表单，最终却由于错误（我们自己的错误、WebApp的错误，或者客户端到服务器端的传输错误）出现了内容丢失。为了避免这种情况，应该将WebApp设计成能够自动保存用户的所有专用数据。

易读性——界面展示的所有信息对于老人和年轻人都应该是易读的。界面设计者应该着重选择易读的字型式样、字体大小以及可以增强对比效果的背景颜色。

跟踪状态——在合适的时候，应该跟踪和保存用户状态，使得用户能够退出系统，稍后返回系统时又能回到退出的地方。一般而言，可设计cookies来存储状态信息。然而，cookies是一种备受争议的技术，别的设计方案也许对某些用户来说更合适。

可见的导航——设计合理的WebApp界面提供了这样的设想，“即用户呆在同一个地方，工作被带到他们面前”[TOG01]。当使用这种方法时，导航就不再是用户关心的事情，用户检索内容对象，并选择功能，这些功能都是通过界面显示并执行的。

SAFEHOME**界面设计评审**

[场景] Doug Miller的办公室。

[人物] Doug Miller, SafeHome软件工程团队经理; Vinod Raman, SafeHome产品软件工程团队成员。

[对话]

Doug: Vinod, 你和你的团队是否有可能评审SafeHomeAssured.com电子商务的界面原型?

Vinod: 是的……我们所有人都从技术角度对它进行了仔细检查, 而且我还做了一些记录。昨天我将这些记录E-mail给了Sharon (SafeHome电子商务网站外包供应商Web工程团队的经理)。

Doug: 你和Sharon可以在一起详细讨论一下……给我一份重要问题的总结。

Vinod: 总的来说, 他们已经做得很好了, 没有遇到什么阻力。但是, 这是一个典型的电子商务界面, 具有高雅的美学设计、合理的布局设计。他们已经完成了所有重要功能……

Doug (可怜地微笑): 但是?

Vinod: 是的, 有些小的问题。

Doug: 例如……

Vinod (给Doug看界面原型的序列情节故事板): 这是一些显示在主页上的主要功能菜单。

学习SafeHome

描述你的住宅

获得SafeHome构件建议

购买SafeHome系统

获得技术支持

问题并不在于这些功能, 它们都没有问题, 但是抽象级别不太合适。

Vinod: 没错。但是有这样一个问题……你可以通过输入构件列表来购买系统……如果你不想描述房子, 就没有必要描述。我建议在主页上创建4个菜单选项:

学习SafeHome

确定你所需要的SafeHome系统

购买SafeHome系统

获得技术支持

当你选择了“**确定你所需要的SafeHome系统**”时, 你会有下面的选项:

选择SafeHome构件

获得SafeHome构件建议

如果你是一个知识渊博的用户, 将从一组分好类的下拉菜单中选择构件, 包括传感器、摄像头、控制面板等。如果需要帮助, 可以请求系统提供建议, 那时系统需要你描述一下你的房间。我认为这样更有逻辑性。

Doug: 我同意。关于这个问题你和Sharon谈过了么?

Vinod: 没有, 我想先和市场部讨论一下, 然后我会给她打电话。

Nielsen和Wagner[NIE96]提出了一些实际可行的界面设计指导原则（基于他们对重要WebApp的重新设计），很好地实现了本节前面提出的准则：

- 对电脑屏幕的阅读速度比对书本的阅读速度要慢大约25%，因此不要迫使用户阅读大量的文本信息，特别是当文本的内容是解释WebApp的操作，或者是辅助导航时。⁶
- 避免“正在建设中”的标志——用户满怀希望而来，但是却看到了这么一个不必要的链接，必然很失望。
- 用户不喜欢使用滚动操作。重要的信息应该放在一般浏览器窗口都可显示的范围内。
- 导航菜单和标题条应该设计得一致，并且出现在用户可用的所有页面中。设计不应该依赖于浏览器的功能来辅助导航。
- 美学效果绝不应该取代功能性。例如，比起一个漂亮的、但内容不明确的图片或图标，简单的按钮可能是更好的导航选择。
- 导航选项应该是明显的，即使是一些临时的用户，也不要让他们满屏幕地搜寻怎样链接到其他内容和服务。

570

好的界面设计能够提高用户对网站提供的内容和服务的理解程度，它并不一定要有闪烁的动画，但是应该是结构合理及功效健全的。

“人们对于那些设计糟糕的WWW站点几乎没有什么耐心。”

——Jakob Nielsen和Annette Wagner

19.3.2 界面控制机制

WebApp界面的目标是：(1) 将界面上所提供的内容和功能组织在一个一致的窗口中；(2) 指导用户完成一系列与WebApp的交互；(3) 组织用户可访问的导航选项和内容。为了实现一致的界面，设计者必须首先运用美学设计(19.4节)建立一致的界面“外观”，这包括很多特性，但是必须把重点放在导航机制的布局和形式上。为了指导用户交互，界面设计者应该利用合适的比喻⁷，使用户能够获得对界面的感性认识。为了实现导航选项，设计者可从很多交互机制中选择一种：

 WebApp设计者可以使用哪些交互机制？

- 导航菜单——关键字菜单（水平或者垂直排列）列出了关键的内容和功能。实现这些菜单以便用户能够从副标题层次中进行选择，副标题是在主标题选项被选中时显示出来的。
- 图标——能够使用户选择属性或者做出决定的按钮、开关及类似的图片。
- 图片——用户可以选择的某种图形表示，它能够实现到某个内容对象或者WebApp功能的链接。

值得注意的是，应该在内容层次的每一级中引入一个或多个这种控制机制。

19.3.3 界面设计工作流

虽然WebApp界面设计的深入讨论最好留给专门的书籍（例如，[GAL02]、[RAS00]或[NIE00]），但我们仍然有必要简要回顾一下重要的设计任务。在第12章中，我们提到用户界

⁷ 在这里，比喻是一种可以在界面的环境中被模仿的表示法（来自于用户在现实世界中的经历），一个简单的例子是用来控制.mpg文件播放音量的滑动条。

面设计首先要确定用户、任务和环境需求。一旦确定了用户任务，就可以创建和分析用户场景（用例），并定义一组界面对象和活动。这项工作是第18章所讨论的WebApp分析模型的一部分。

下面的工作代表了WebApp界面设计的基本工作流程：

1. 回顾那些在分析模型中的信息，并根据需要进行优化。
2. 开发WebApp界面布局的草图。界面原型（包含布局）可能已经作为分析建模活动中的一部分而得以开发。如果布局已经存在，应该根据需要对其进行检查和优化；如果还没有开发界面布局，此时Web工程团队应该与共利益者合作完成开发。图19-3显示的是一张示意性的布局草图。
3. 将用户目标映射到特定的界面行为。对于大多数WebApp来说，用户的主要目标相对比较少（一般在4~7个之间）。应该将这些目标映射到特定的界面行为，如图19-3所示。

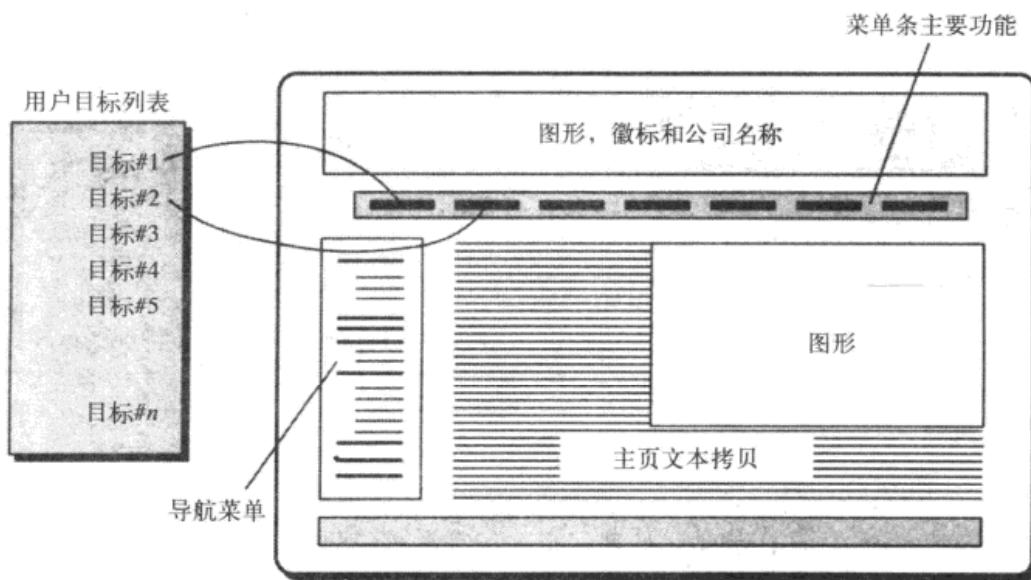


图19-3 将用户目标映射到特定的界面行为

4. 定义与每一个行为相关的一组用户任务。每一个界面行为（例如，购买一个商品）与一组用户任务相联系。在分析建模的过程中已经确定了这些任务。在设计期间，它们必须与明确的交互对象建立对应关系，这些交互对象包括导航事件、内容对象和WebApp功能。

5. 为每一个界面行为设计情节故事板屏像（storyboard screen images）。当考虑每一种行为时，应该创建序列情节故事板图像（屏像），来描述界面是怎样响应用户的交互行为的。应该明确内容对象（即使它们还没有被设计和开发），展示WebApp的功能及导航链接。572

6. 利用从美学设计中的输入来优化界面布局和情节故事板。粗略的布局和情节故事板是由Web工程师完成的，但是重要商业网站的美学外观通常是由专业艺术家完成的，而不是技术专家。

7. 明确实现界面功能的界面对象。这一任务可能会需要在现有对象库中搜索，找到那些适合WebApp界面的可复用对象（类）。另外，在此时定义任何需要的自定义类。

8. 开发用户与界面交互的过程表示。这一可选的任务利用UML顺序图和（或）活动图（在第18章讨论）表示用户与WebApp交互时的活动流程。

9. 开发界面的行为表示法。这一可选的任务利用UML的状态图（在第18章讨论）表示状

态转换和引起状态转换的事件，并定义控制机制（即通过用户可用的对象和行为改变WebApp的状态）。

10. 描述每一种状态的界面布局。利用在任务2和任务5中开发的设计信息，把确定的布局和屏像与任务9中描述的每一个WebApp状态联系起来。

11. 优化和评审界面设计模型。界面的评审应该以可用性为重点。

值得注意的是，Web工程团队所选择的最终任务集必须适合待构建的应用系统的特殊需求。

19.4 美学设计



不是每一个Web工程师(或者软件工程师)都有艺术(美学)天赋。如果你没有艺术天赋，那就聘请一位有经验的美术设计师来进行美学设计工作。

573

美学设计，又称美术设计，是一种艺术工作。它是对Web工程技术方面的补充。没有它，WebApp可能有强大的功能，但是却不能吸引人；有了它，WebApp能够将它的用户带入以用户为核心的充满智慧的世界。

但是什么是美学？有这么一句谚语：“美丽存在于能够发现美丽的人的眼中”。当考虑WebApp的美学设计时，这句话就更加贴切了。为了进行有效的美学设计，我们再一次回到作为分析模型的一部分而开发的用户层次（第18章），并且提出这样的问题：谁是WebApp的用户？他们希望什么样的“外观”？

“我们发现人们仅仅通过视觉设计的效果来快速评估一个网站。”

——指导Web可信性设计的Stanford指导原则

19.4.1 布局问题

每一个网页中能够用来支持非功能性的美学设计、导航特征、信息内容及指导用户功能的“空间”是有限的，应该在美学设计期间对这种空间的“开发”进行规划。

像所有的美学问题一样，当设计屏幕布局时，没有绝对的规则。但是，很多一般的布局指导原则还是值得考虑的：

不要害怕留下空白。我们不建议把网页中每一寸空间都排满信息。如果非要这样做，用户寻找有用信息或要素会很困难，并会造成很不舒服的视觉混乱。

重视内容。毕竟，内容是用户浏览网页的根本原因。Nielsen[NIE00]建议，典型的Web页的80%应该是内容，剩余的资源为导航和其他要素。

按照从左上到右下的顺序组织布局元素。绝大多数用户浏览网页的方式与看书没有什么不同——从左上到右下⁸。如果布局元素有特定的优先级，应该将高优先级的元素放在页面空间的左上部分。

按照页面区域组织导航、内容和功能。在几乎所有的事情中，人们都会寻找存在的模式。如果Web页中没有可辨别的模式，用户的挫败感会增加（由于需要对所需要的信息进行不必要的查找）。

不要通过滚动条扩展空间。虽然滚动经常是需要的，但大多数的研究表明，用户还是不喜欢用滚动条。最好是减少网页内容，或者通过多页显示必要的内容。

⁸ 确实有一些文化和语言上的例外。但是这一准则对于大多数用户是适合的。

在设计布局时，考虑解决方案和浏览器窗口的尺寸。设计应该能够确定布局元素占用可用空间的百分比，而不是在布局中定义固定的尺寸[NIE00]。

19.4.2 美术设计问题

美术设计需要考虑到WebApp外观的各个方面。美术设计过程从布局（19.4.1节）开始，并在设计过程中考虑全局颜色配置、字体、字号、风格、样式、补充媒体（例如，音频、视频、动画）的使用，以及一个应用系统的所有其他美学元素。感兴趣的读者能够从很多专业网站（例如，www.graphic-design.com、www.grantasticdesigns.com、www.wpdfd.com）或一些可打印的资源（例如，[BAG01]、[CLO01]或[HEI02]）中获得设计技巧和指导原则。

INFO

设计良好的网站

有时候，要理解好的WebApp设计，最好方法就是看几个例子。在Marcelle Toor的文章“*The Top Twenty Web Design Tips*”中（<http://www.graphic-design.com/Web/feature/tips.html>），他建议将下面的网站作为美术设计的范例：

www.primo.com——以Primo Angeli为首的设计公司。

www.workbook.com——这个网站展示了插图画家和设计者的工作。

www.pbs.org/riverofsong——针对公众电视的电视连续剧和关于美国音乐的无线电广播。

www.RKDINC.com——提供在线代表作品选和设计技巧的设计公司。

www.commarts.com/career/index.html——《Communication Arts》杂志，一本针对美术设计师的行业期刊，介绍了很多其他设计良好的网站。

www.btdnyc.com——以Beth Toudreau为首的设计公司。

19.5 内容设计

内容设计关注两个不同的设计主题，每一个人都可以使用不同的技巧描述每个主题。内容设计为内容对象创建一种设计表示，并且表示了一种机制，在用实例说明内容对象之间的关系时需要这种机制。这项设计工作由Web工程师完成。

另外，内容设计也关注特定的内容对象内的信息表示方法——一项设计活动，由广告文案写作人员、美术设计人员以及产生WebApp内容的其他人员完成。

“好的设计人员能够使混乱的状态正常化，他们能够通过组织和管理文字和图片清楚地表达自己的想法。”

—Jeffery Veen

19.5.1 内容对象

内容对象被定义成WebApp分析模块的一部分（例如，图18-3），内容对象和代表内容的设计对象之间的关系类似于分析类和第11章描述的设计构件之间的关系。在Web工程的环境中，内容对象与传统软件中的数据对象关系更加紧密。内容对象具有属性，包括内容特定信息（通常在WebApp分析建模期间定义）的属性和被指定为设计成分的实现属性。

举个例子，考虑为SafeHome电子商务系统开发的分析类**ProductComponent**（在第18章

575 开发), 其描述如图19-4所示。在第18章, 我们提到了属性description, 在这里, 这个属性被描述为一个设计类, 名为**CompDescription**。这个类包括5个内容对象: **MarketingDescription**、**Photograph**、**TechDescription**、**Schematic**和**Video**, 如图中阴影部分所示。内容对象所包含的信息被标注成对象的属性。例如, **Photograph** (一个.jpg格式的图片) 有属性horizontal dimension、vertical dimension和border style。

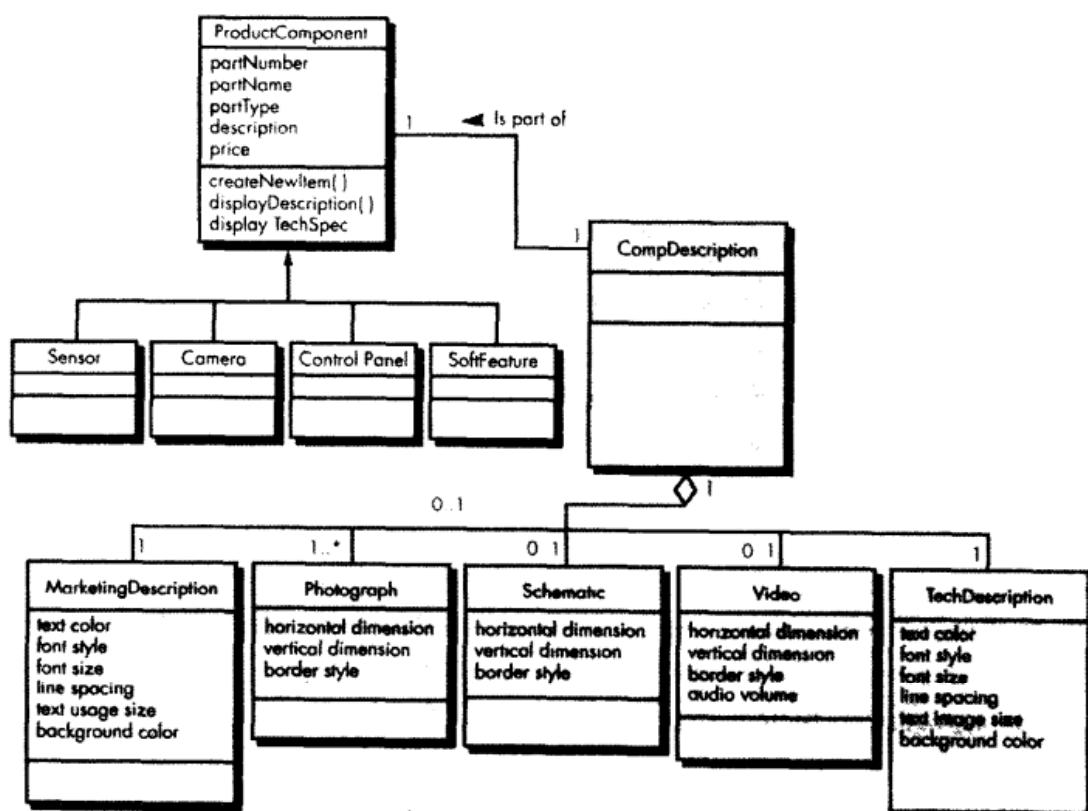


图19-4 内容对象的设计表示

UML的关联和聚合符号⁹可以用来表示内容对象 (实例) 之间的关系。例如, 图19-4所示的UML关联表明一个**CompDescription**类对象用于描述一个**ProductComponent**类实例; 一个**CompDescription**类实例由所示的5个内容对象组成。然而, 所示的多重性符号表明**Schematic**类实例和**Video**类实例是可选的 (重数可能为0), 一个**MarketingDescription**类实例和一个**TechDescription**类实例是必需的, 会用到一个或多个**Photograph**类实例。

19.5.2 内容设计问题



576

与水平滚动条相比, 用户更容易接受垂直滚动条, 所以应避免宽页面格式。

对所有的内容对象建模之后, 就必须编写对象传递的信息, 然后对其格式化, 最大程度地满足用户的要求。内容编辑是专家的工作, 通过提供所传递信息的概要描述和用来传递信息的一般内容对象的类型说明 (例如, 描述性文本、图片、照片) 来设计内容对象, 可能也会应用美学设计 (19.4节) 为内容设计合适的外观。

当设计内容对象时, 将内容对象“分块” [POW00], 然后形成WebApp页

⁹ 这两种表示方法已经在第8章讨论过。

面。集成在一个页面的内容对象的数量与用户需求、网络连接的下载速度和用户能够忍受的滚动次数有关。

19.6 体系结构设计

体系结构设计与已建立的WebApp的目标、展示的内容、将要访问它的用户和已经建立的导航原则紧密相关。体系结构设计者必须确定内容体系结构和WebApp体系结构。内容体系结构¹⁰着重于内容对象（诸如网页的组成对象）的表现和导航的组织方式。WebApp体系结构描述应用系统将以什么样的组织方式来管理用户交互、操纵内部处理任务、实现导航及展示内容。

“设计良好的网站，其体系结构不总是对用户透明的——也不应该这样。”

—Thomas Powell

在大多数情况下，体系结构设计与界面设计、美学设计和内容设计并行进行。由于WebApp的体系结构对导航的影响很大，所以在设计活动中做出的决定会影响导航设计阶段的工作。

19.6.1 内容体系结构

内容体系结构的设计着重于对WebApp的所有超媒体结构进行定义。在进行设计时，可以在下面4种不同的内容结构中进行选择[POW00]：

线性结构。当内部交互的可预测顺序（包含一些变量和转换）很常见时，常选择线性结构（图19-5）。帮助文档的展示可能是一个典型的例子，在有必备的前提信息后，信息页连同相关的图片、简短的视频、音频才能随之出现。内容展示的顺序是预先定义的，而且通常是线性的。还有一个例子是产品订单的输入顺序，必须以特殊的顺序对特殊的信息进行详细说明。在这种情况下，图19-5所示的结构是很合适的。当内容和处理过程变得越来越复杂时，图左边所示的纯粹的线性流程将被更复杂的线性结构所取代，在此结构中，可替换的内容可以被触发或者发生转换，来获取补充的内容（图19-5右边所示的结构）。

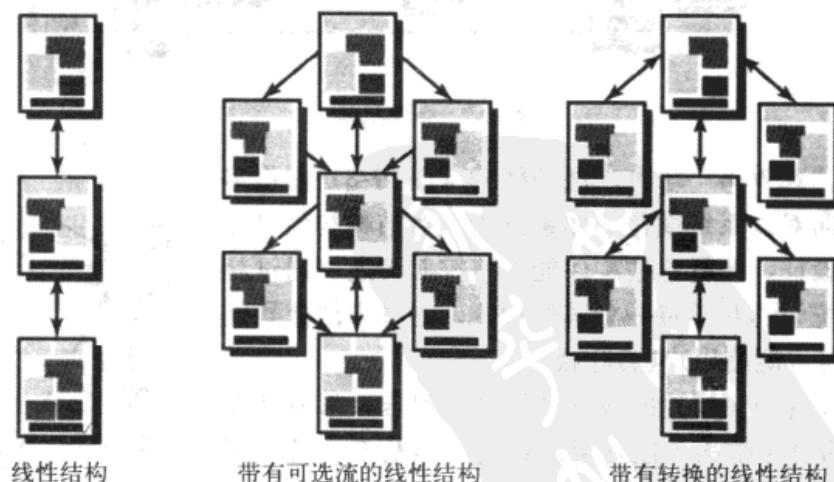


图19-5 线性结构

¹⁰ 术语信息体系结构也用来表示那些能够使组织、标记、导航和内容对象的查找方面表现更好的结构。

网格结构。网格结构（图19-6）是另一种体系结构，当WebApp的内容按类别组织成二维（或更多维）时，可以采用这种结构。例如，考虑这样的情景：一个电子商务网站销售高尔夫球棒。网格的水平方向代表要出售的球棒种类（例如木制棒、金属棒、楔形棒、轻击棒）；垂直方向代表不同的球棒制造厂商所提供的产品。因此，用户可能沿着网格的水平方向找到轻击棒所在的列，然后在竖直方向上检查销售轻击棒的厂家提供的产品。这种WebApp结构只有在内容十分规则的情况下才会使用[POW00]。

578

分层结构。分层结构（图19-7）毫无疑问是最常见的WebApp体系结构。与第10章讨论的分区的软件层次有所不同，在第10章，只有在层次的垂直分支上鼓励控制流程，而WebApp的层次结构可以设计成使控制流水平地穿过垂直分支（通过超文本分支）的方式。因此，在分层结构中最左边展示的内容可以通过超文本链接与中间或者右边分支的内容相连。然而，我们应该注意，虽然这种分支结构能够实现WebApp内容的快速导航，但同时有可能给用户带来困惑。

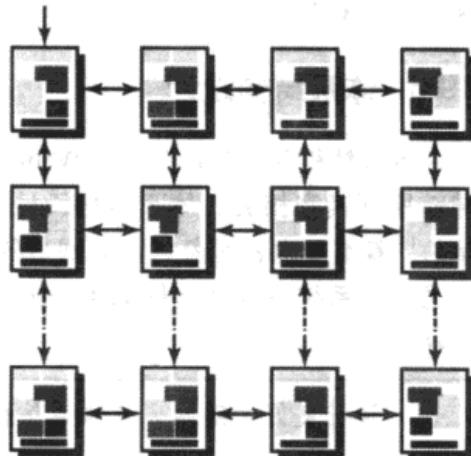


图19-6 网格结构

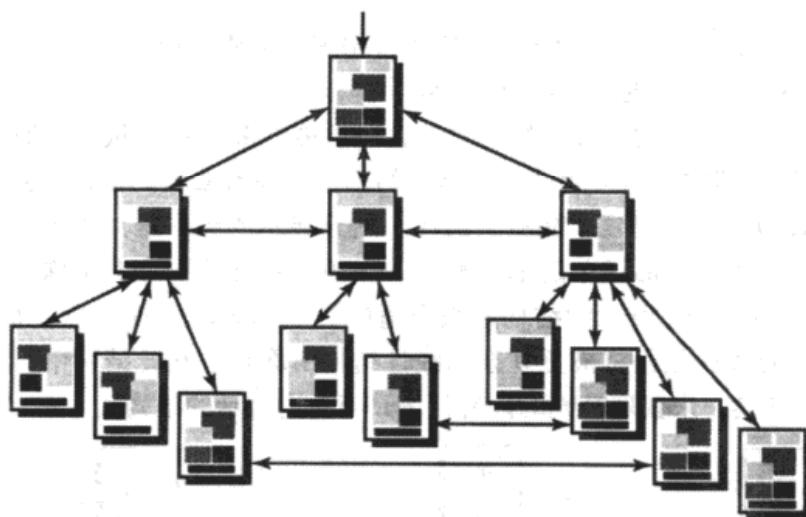


图19-7 分层结构

网络结构或“纯Web”结构。这种结构（图19-8）在很多方面类似于面向对象系统演化的体系结构。对结构构件（此处为网页）进行设计，使得这些构件能够将控制传递（通过超文本链接）到系统中的几乎每一个其他构件。这种方法使导航相当灵活，但同时可能使用户感到困惑。

可以将前面段落中所讨论的设计结构进行组合，形成复合结构。WebApp的总体结构可能是层次结构，但是部分结构可能会展示出线性特性，而结构的另一部分可能是网络结构。结构设计人员的目标就是使WebApp的结构和展示的内容及实现的过程相匹配。



图19-8 网络结构

19.6.2 WebApp体系结构

WebApp体系结构描述了使基于Web的系统或应用达到其业务目标的基础结构。Jacyntho和他的同事[JAC02]对这一结构的基本特性做了如下描述：

579

创建应用程序应该考虑到不同层所关注的方面不一样，特别是，应用数据应该与网页的内容（导航节点）分开，反过来，网页的内容也要与界面的外观（页面）清楚地分离。

作者建议采用3层设计结构，使界面与导航及应用行为相分离，并且认为使界面、应用和导航分离可以简化实现，并能够增加复用性。

KEY POINT
MVC结构将用户界面与WebApp功能及信息内容分离。

模型—视图—控制器（Model-View-Controller, MVC）结构[KRA88]¹¹是许多建议的WebApp基础结构模型之一，它将用户界面与WebApp功能及信息内容分离。模型（有时称“模型对象”）包括应用系统的所有详细内容和过程逻辑，包括所有的内容对象、对外部数据或信息源的访问，以及应用系统的特定处理功能；视图包括所有界面的特定功能，并能够表示内容和处理逻辑，包括所有内容对象、对外部数据或信息源的访问，以及最终用户所需要的所有处理功能；控制器管理对模型和视图的访问，并协调两者间的数据流。在WebApp中，“视图由控制器进行更新，更新数据来自基于用户输入的模型”[WMT02]。MVC体系结构的示意图如图19-9所示。

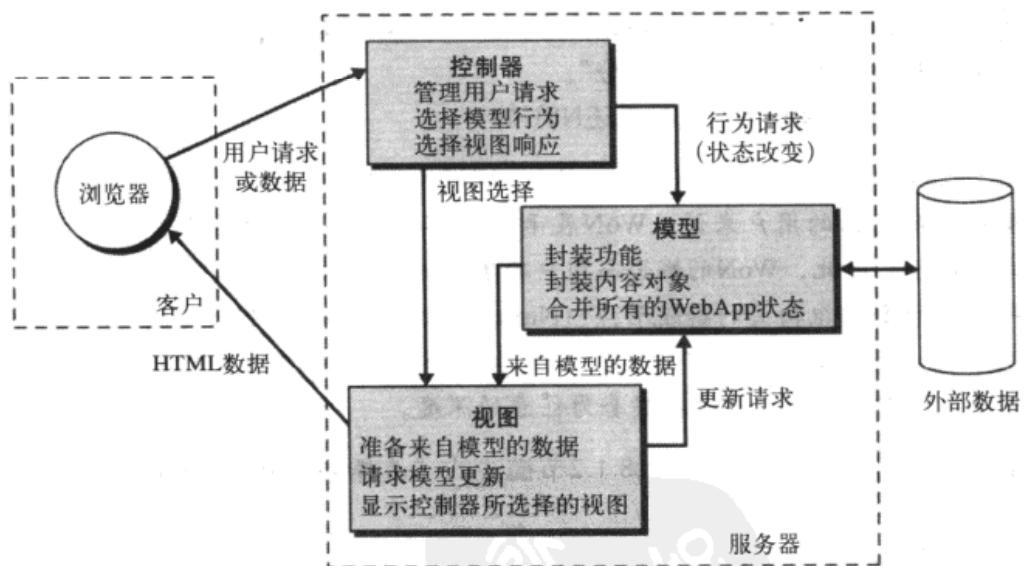


图19-9 MVC结构（改自[JAC02]）

根据此图，用户请求或数据由控制器处理。控制器也可以根据用户请求选择合适的视图对象。一旦确定了请求的类型，就将行为请求传递给模型，模型实现功能，或者检索满足用户请求所需要的内容。模型对象可以访问存储在公司数据库中的数据，被访问的数据可以与本地数据存储在一起，或者作为一些独立文件单独存储。模型创建的数据必须由合适的视图对

580

¹¹ 应该注意到，MVC实际上是为Smalltalk环境创建的设计模式(见 http://www.cetus-links.org/oo_smalltalk.html)，并可用于任何交互式应用。

象对其进行格式化和组织，然后从应用服务器传回到客户端浏览器，并显示在客户的电脑上。

很多情况下，在实现应用系统的开发环境中定义WebApp的体系结构（例如，ASP.net、JWAA或J2EE）。感兴趣的读者请查阅[FOW03]，其中深入讨论了现代开发环境及其在Web应用体系结构设计中的作用。

19.7 导航设计

一旦建立了WebApp的体系结构及确定了体系结构的构件（页面、脚本、applet和其他处理功能），设计人员就应定义导航路径，使用户可以访问WebApp的内容和功能。为了完成这一任务，设计者应该：（1）为网站的不同用户确定导航语义；（2）定义实现导航的机制（语法）。

“Gretel，等到月亮升起来的时候，我们就能看到我撒落的面包屑。这些面包屑会指给我们回家的路。”

——引自Hansel和Gretel

19.7.1 导航语义

像很多其他Web工程活动一样，在进行导航设计时，首先考虑用户层次和为每一类用户（角色）创建的相关用例（第18章）。每一类角色使用WebApp的方式或多或少会有所区别，因而会有不同的导航要求。另外，为每一类角色设计的用例会定义一组类，这组类包含一个或多个内容对象，或者包含WebApp功能。当用户和WebApp进行交互时，会接触到一系列的导航语义单元（Navigation Semantic Unit，NSU）——“信息和相关的导航结构的集合，它们相互协作共同完成相关的用户请求的一部分”。

Gnaho 和 Larcher[GNA99]是这样描述NSU的：

NSU结构包括一组称作导航方式（Way of Navigating，WoN）的导航子结构。对于那些有明确轮廓的用户来说，WoN展示了达到他们期望的目标或子目标的最佳导航方式或路径。因此，WoN的概念与用户轮廓的概念是相关联的。

WoN结构由一组相关的导航节点（Navigational Node，NN）组成，这些导航节点通过导航链接连接起来，有时候还包括其他NSU。这就意味着NSU可以被组合起来形成更高级别的NSU，或者被嵌套为任意的深度。

为了举例说明NSU的开发，考虑18.1.2节描述的“选择SafeHome部件”用例，这里重复描述如下：

用例：选择SafeHome部件

WebApp会推荐产品部件（例如，控制面板、传感器、摄像头）及每个房间和外部入口的其他特征（例如，用软件实现的基于PC的功能）。当要求选择时，如果选择的部件存在的话，WebApp就会提供，我们会得到每一个产品部件的描述信息和价格信息。选择了不同的部件之后，WebApp会创建并显示一份材料清单。可以给材料清单取一个名字，并保存起来供将来参考（见用例：保存配置）。

在用例描述中标有下划线的项代表了类和内容对象，它们将被合并为一个或多个NSU，使得一个新客户可以执行“选择SafeHome部件”用例中描述的场景。

图19-10描述了用例“选择SafeHome部件”所隐含的导航的部分语义分析。使用前面介绍

KEY POINT

NSU描述用例的导航需求。本质上，NSU显示了角色是怎样在内容对象和Web-App功能之间移动的。

的术语，此图也显示了SafeHomeAssured.com WebApp的导航方式（WoN）。重要的问题域类与选中的内容对象（在此例中，名为**CompDescription**的内容对象包是**ProductComponent**类的属性）一同显示，这些项就是导航节点。每一个箭头代表一个导航链接¹²，并且采用用户触发行为进行标注，这些行为使链接发生。

WebApp设计者为每个与用户角色相关的用例都创建一个NSU[GNA99]。例如，一个新客户（图18-1）可能有3个不同的用例，这3个用例都将访问不同的信息及WebApp功能，这就需要为每个用例都创建一个NSU。

582

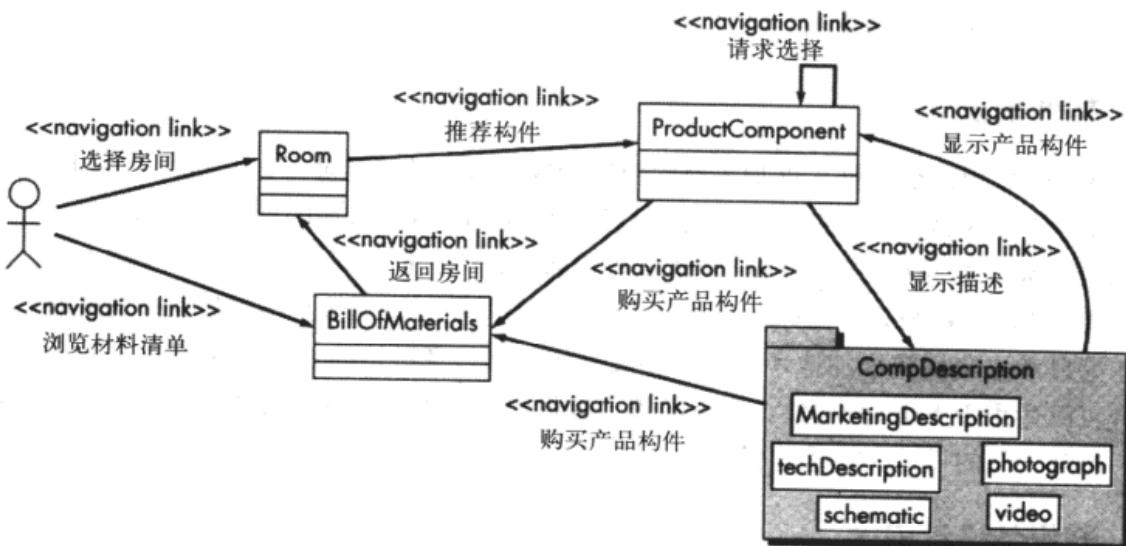


图19-10 创建一个NSU

在导航设计的初始阶段，应对WebApp的内容体系结构进行评估，为每个用例确定一个或多个WoN。如上面所说的那样，一个WoN标识了导航节点（例如内容）和使它们之间能够导航的链接，然后将WoN组织到NSU中。

“网站导航问题是概念的、技术的、空间的、哲学的及逻辑的问题。因此，解决办法要求艺术、科学和组织心理学的复杂的即兴组合。”

—Tim Horgan

19.7.2 导航语法

ADVICE
在大多数情况下，应选择垂直或水平导航机制，但不要两者都选。

在进行设计时，需要定义导航机制。其中有多种可能的选择：

- **单独的导航链接**——基于文本的链接、图标、按钮、开关及图形比喻。
- **水平导航条**——在包含合适链接的工具条中列出重要的内容或功能类别，一般列出4~7类。
- **垂直导航列**——（1）列出重要的内容和功能类别；（2）列出WebApp中几乎所有的主要内容对象。如果选择了第2个选项，这种导航列可以进行“扩充”，将内容对象描述为层次的一部分。

¹² 有时将这些导航链接称为导航语义链接（Navigation Semantic Link，NSL）[CAC02]。

- 标签——只是导航条和导航列的变种，代表内容或功能类别，在需要链接时作为标签页被选中。

583 • 网站地图——向包含在WebApp中的所有内容对象和功能提供完整的导航内容表格。



从每一页都应该可以访问站点地图。应该对地图本身进行组织，使得WebApp信息的结构十分清楚。

除了选择导航机制以外，设计人员还应该建立合适的导航习惯和帮助。例如，为了使图标和图形链接呈现“可点击”的状态，图标和图形的边缘应成斜角，使其呈现出三维效果。应该考虑设计听觉和视觉反馈，提示用户导航选项已被选择。对于基于文本的导航，应该用颜色来显示导航链接，并给出链接已经被访问的提示。在进行用户友好的导航设计时，这些仅仅是许多设计习惯中的几种。

19.8 构件级设计

现代Web应用系统提供了更加成熟的处理功能，这些功能能够：(1) 执行本地化的处理，从而动态地产生内容和导航能力；(2) 提供了适于WebApp业务领域的计算或数据处理；(3) 提供了高级的数据查询和访问；(4) 建立了与外部系统的数据接口。为了实现这些（及许多其他）能力，Web工程师必须设计和创建程序构件，这些构件在形式上与传统软件构件相同。

在第11章，我们较详细地考虑了构件级设计。第11章讨论的设计方法几乎不需要任何修改，就可以适用于WebApp构件。实现环境、编程语言、可复用模式、框架和软件可能会有些不同，但是，总的设计方法是一样的。

19.9 超媒体设计模式

应用于Web工程的设计模式主要包括两类：(1) 适用于所有软件类型的通用设计模式（例如[BUS96]和[GAM95]）；(2) 针对WebApp的超媒体设计模式。通用设计模式已经在第9章讨论过。很多超媒体模式的目录和知识库可以通过Internet获得¹³。

“每一种模式都包含3个部分，表示确定的上下文、问题和解决方案之间的关系。”

—Christopher Alexander

如我们在本书前面所讲到的那样，设计模式是一种解决某些小的设计问题的通用方法，这种方法也适合更大范围的一些特别的问题。在基于Web的系统中，German和Cowan[GER00]提出了以下模式类别：

体系结构模式。体系结构模式辅助内容和WebApp体系结构的设计。19.6.1节和19.6.2节讲述了内容和WebApp体系结构的结构模式。另外，对于要在多种业务领域设计WebApp的Web工程师来说，可以使用很多相关的结构模式（例如，java.sun.com/blueprints/中的Java Blueprints）。

构件构造模式。构件构造模式建议采用某些方法把WebApp构件（例如，内容对象、功能）合并成复合构件。当在WebApp中需要数据处理功能时，可应用由[BUS96]、[GAM95]及其他文献资料所提出的体系结构和构件级设计模式。

¹³ 参见下文“超媒体设计模式资源库”的补充讨论。

导航模式。导航模式辅助WebApp的NSU设计、导航链接设计和总体导航流程。

表示模式。当通过界面将内容展示给用户时，表示模式可以辅助内容的展示。这类模式描述了如何组织用户界面控制功能，以达到更好的可用性；如何显示接口行为和其影响的内容对象之间的关系；如何建立有效的内容层次及很多其他方面的功能。

行为/用户交互模式。行为模式辅助人机交互的设计。这类模式描述界面如何将特定行为产生的结果通知用户，用户如何基于使用环境和用户期望扩充内容，如何更好地描述链接所暗示的目的地，如何将正在进行的交互行为和其他行为的状态通知用户。

超媒体设计模式的信息源在近几年发展迅速，感兴趣的读者可以查阅[GAR97]、[PER99]和[GER00]。

SOFTWARE TOOLS

超媒体设计模式资源库

IAWiki网站 (<http://iawiki.net/WebsitePatterns>) 是大家讨论信息结构的地方，包含很多有用的资源。其中很多链接指向有用的超媒体模式目录和资源库，描述了成百上千的设计模式：

超媒体设计模式资源库 (<http://www.designpattern.lu.unisi.ch/>)。

Tom Erickson提出的交互模式 (http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html)。

Martijn van Welie提出的Web设计模式 (<http://www.welie.com/patterns/>)。

利用导航模式改进Web信息系统 (<http://www8.org/w8-papers/5b-hypertext-media/improving/improving.html>)。

HTML 2.0模式语言 (<http://www.anamorph.com/dos/patterns/default.html>)。

公共基础 (http://www.mit.edu/~jtidwell/interaction_patterns.html)。

个人网站的模式 (<http://www.rdrop.com/~half/Creations/Writings/Web.patterns/index.html>)。

索引模式语言 (<http://www.cs.brown.edu/~rms/InformationStructures/Indexing/Overview.html>)。

585

19.10 面向对象的超媒体设计方法

在过去的10年中，虽然提出了很多Web应用系统的设计方法，但至今还没有哪种方法获得了统治地位。面向对象的超媒体设计方法（Object-Oriented Hypermedia Design Method, OOHDM）是讨论最广泛的WebApp设计方法之一，在这一节中，我们对OOHDM进行简要介绍¹⁴。

OOHDM最初是由Daniel Schwabe和他的同事[SCH95, SCH98]提出来的。OOHDM由四种不同的设计活动组成，即概念设计、导航设计、抽象接口设计和实现。这些设计活动的简要描述如图19-11所示，在接下来的几节中将简单讨论它们。

¹⁴ Koch对10种超媒体设计方式进行了全面比较 [KOC99]。

	概念设计	导航设计	抽象界面设计	实现
工作产品	类, 子系统, 关系, 属性	节点链接, 访问结构, 导航上下文, 导航变换	抽象界面对象, 外部事件响应, 变换	可执行的 WebApp
设计机制	分类, 组合, 聚合, 泛化, 特殊化	建立概念对象和导航对象的对应关系	建立导航和可感知对象的对应关系	目标环境提供的资源
设计重点	应用领域的建模语义学	考虑用户轮廓和任务, 倾重于可感知的方面	可感知对象建模, 实现选择的比喻, 描述导航对象的界面	正确性, 应用性能, 完备性

图19-11 OOHDM方法一览（根据[SCH95]改写）

19.10.1 OOHDM的概念设计

OOHDM的概念设计创建定义WebApp应用领域的子系统、类及关系的表示。可以使用UML¹⁵来创建合适的类图、聚合类及组合类的表示、协作图和描述应用领域的其他信息（更详细的信息见本书第二部分）。

作为OOHDM概念设计的一个简单例子，我们再次考虑SafeHomeAssured.com电子商务应用。SafeHomeAssured.com的部分“概念模式”如图19-12所示。作为WebApp分析的一部分而开发的类图、聚合及相关信息可以在概念设计期间得到复用，并可用来表示类之间的关系。

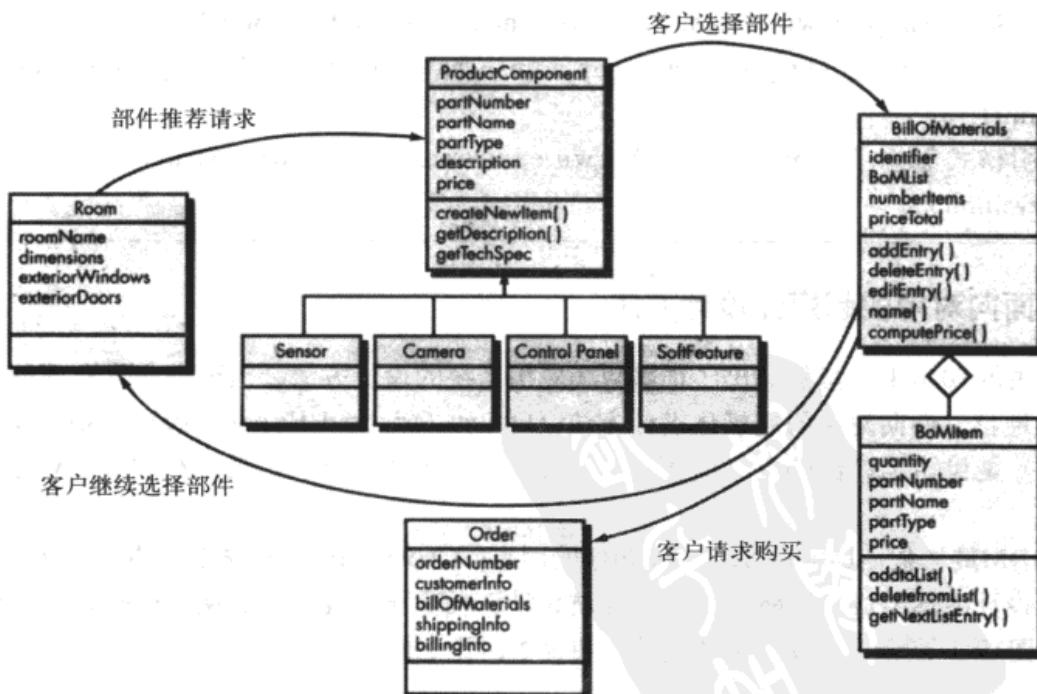


图19-12 SafeHomeAssured.com的部分概念模式

¹⁵ OOHDM并没有制定专门的符号，但是应用这种方法时常常使用UML。

19.10.2 OOHDM的导航设计

导航设计确定了一组对象，这些对象来自概念设计中所定义的类。为了封装这些对象，将定义一系列的“导航类”或“节点”。可以使用UML创建合适的用例图、状态图和顺序图——所有这些表示法都可以帮助设计者更好地理解导航需求。此外，当进行设计时，可以使用导航设计的设计模式。OOHDM使用预定义的一组导航类——节点、链接、锚和访问结构 [SCH98]。访问结构会越来越详细，并且包括WebApp索引、站点地图或导航等机制。

一旦定义了导航类，OOHDM“通过把导航对象分成多个集合（称为上下文）来组织导航空间” [SCH98]。Schwabe这样描述上下文：

除了上下文本身包含的元素，每一个上下文定义还包括内部导航结构的详细描述、入口点、在用户类和操作方面的访问限制及相关的访问结构。

创建上下文模板（类似于第8章讨论的CRC卡），并且用此模板跟踪在OOHDM中定义的不同上下文中各类用户的导航需求。在这一过程中，形成了特定的导航路径（我们在19.7.1节称之为WoN）。

19.10.3 抽象界面设计与实现

抽象界面设计活动确定了当用户与WebApp进行交互时看到的界面对象，界面对象的正式模型，称为抽象数据视图（ADV），用来描绘界面对象和导航对象的关系及界面对象的行为特点。

ADV模型定义了“静态布局” [SCH98]，这种布局描述界面比喻，包含界面中导航对象的描述，以及对辅助导航和交互的界面对象（例如菜单、按钮、图标）的详细描述。此外，ADV模型包含行为构件（类似UML的状态图），用来说明外部事件是怎样“触发导航，以及当用户与应用程序交互时，哪个界面需要发生变化” [SCH01]。对于ADV的详细讨论，感兴趣的读者可以参考[SCH98]和[SCH01]。

OOHDM实现活动描述了特定于WebApp运行环境的设计迭代。在客户/服务器环境、操作系统、支持软件、编程语言及与问题相关的其他环境特性的构造方式上，类、导航和界面具有各自的特点。

19.11 WebApp的设计度量

设计度量的特点是为Web工程师提供实时的质量指数。实际上，一系列有用的测度和度量方法为下面问题提供了定量的答案：

- 用户界面是否能够提高可用性？
- WebApp的美学设计对于应用领域是否合适？是否能够得到用户的认可？
- 是否能够以最小的工作量提供最多信息的方式来设计内容？
- 导航是否有效和直接？
- WebApp体系结构设计是否能够满足WebApp用户的特殊目的和目标、内容结构、功能结构及有效使用系统所需要的导航流程？
- 构件是否是以减少过程复杂性以及增加正确性、可靠性和性能的方式来设计的？

现在，可以定性地表达这些问题¹⁶，但是可以提供定量回答的有效度量还不存在。

¹⁶ 参考第16章（16.4节）和19.1.1节对WebApp质量的定性讨论。

587

588

WebApp设计的度量尚处在初级阶段，已经被广泛验证的度量几乎还没有。感兴趣的读者可以参考[IVO01]和[MEN01]提出的WebApp设计度量方面的例子。

SOFTWARE TOOLS

WebApp的技术度量

目的：帮助Web工程师开发有意义的WebApp度量，这种度量能够衡量应用系统的总体质量。

机制：工具的机制各异。

代表性工具¹⁷

Netmechanic Tools，由Netmechanic (www.netmechanic.com) 开发，是一个帮助提高网站性能的工具集合，更多关注实现方面的问题。

NIST Web Metrics Testbed，由National Institute of Standards and Technology (zing.ncsl.nist.gov/WebTools/) 开发。包括以下有用的工具集合（可以下载）：

Web Static Analyzer Tool (WebSAT) ——对照典型的可用性指导原则检查网页HTML。

Web Category Analysis Tool (WebCAT) ——使工程师构造和管理Web分类分析。

Web Variable Instrumenter Program (WebVIP) ——给网站装备工具，截获用户交互的日志。

Framework for Logging Usability Data (FLUD) ——实现文件格式化工具和解析器来描述用户交互日志。

VisVIP Tool ——在通过网站的用户导航路径上产生三维可视化效果。

TreeDec ——在网页上添加导航帮助。

19.12 小结

WebApp的质量是根据其可用性、功能性、可靠性、效率、可维护性、安全性、可伸缩性及面市时间定义的，它在设计的过程中被引入。为了实现这些质量属性，好的WebApp设计应该展现出简单性、一致性、相符性、健壮性、导航性和视觉吸引力。

界面设计描述了用户界面的结构和组成，包括屏幕布局的表示、交互方式的定义和导航机制的描述。
589

美学设计也称美术设计，描述了WebApp的“外观和感觉”，包含颜色配置、几何布局、文本字号、字体和位置、图形的使用及相关的美学决策。一系列美术设计指导原则为设计方法提供了基础。

内容设计为所有内容定义了布局、结构和外观轮廓，这些内容是WebApp的一部分，并建立了内容对象之间的关系。在内容设计时，首先对内容对象、它们之间的关联和关系进行描述。一组浏览原语建立了导航设计的基础。

结构设计确定了WebApp的总体超媒体结构，并且包含内容结构和WebApp结构。内容的结构风格包括线性结构、网格结构、层次结构和网络结构。WebApp结构描述了使基于Web的系统或应用达到业务目标的基础结构。

导航设计描绘了内容对象之间的导航流和为所有的WebApp功能建立的导航流。通过描述

¹⁷ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的一些样例。

一组导航语义单元来定义导航。每一个单元由导航路径、导航链接和节点组成。导航语法机制用于实现导航，导航也是语义描述的一部分。

构件设计制定了实现WebApp功能构件所需要的详细处理逻辑。第11章所描述的设计技术可应用于WebApp的构件工程。

WebApp设计模式包括可以适用于所有软件类型的通用设计模式及与WebApp相关的超媒体模式。目前已经提出的设计模式包括结构模式、导航模式、构件模式、表示模式、行为/用户设计模式。

面向对象的超媒体设计方法（OOHDM）是提出的许多WebApp设计方法中的一种。OOHDM建议的设计过程包括概念设计、导航设计、抽象接口设计和实现。

Web工程的设计度量正处在初级阶段，并且还没有被充分验证。然而，已经提出了多种测度和度量方法来描述本章所讨论的设计活动。

参考文献

- [AME96] Amento, B., et al., "Fitt's Law," *CS 5724: Models and Theories of Human-Computer Interactions*, Virginia Tech, 1996, available at <http://ei.cs.vt.edu/~cs5724/g1/>.
- [BAG01] Baggerman, L., and S. Bowman, *Web Design That Works*, Rockport Publishers, 2001.
- [BUS96] Buschmann, F., et al., *Pattern-Oriented Software Architecture*, Wiley, 1996.
- [CAC02] Cachero, C., et al., "Conceptual Navigation Analysis: a Device and Platform Independent Navigation Specification," *Proc. 2nd Intl. Workshop on Web-Oriented Technology*, June 2002, download from www.dsic.upv.es/~west/iwwost02/papers/cachero.pdf.
- [CLO01] Cloninger, C., *Fresh Styles for Web Designers*, New Riders Publishing, 2001.
- [DIX99] Dix, A., "Design of User Interfaces for the Web," *Proc. Of User Interfaces to Data Systems Conference, September 1999*, download from <http://www.comp.lancs.ac.uk/computing/users/dixa/topics/webarch/>.
- [FIT54] Fitts, P., "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement," *Journal of Experimental Psychology*, vol. 47, 1954, pp. 381-391.
- [FOW03] Fowler, M., et al., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.
- [GAL02] Galitz, W., *The Essential Guide to User Interface Design*, Wiley, 2002.
- [GAM95] Gamma, E. et al., *Design Patterns*, Addison-Wesley, 1995.
- [GAR97] Garrido, A., G. Rossi, and D. Schwabe, "Patterns Systems for Hypermedia," 1997, download at www.inf.puc-rio.br/~schwabe/papers/PloP97.pdf.
- [GER00] German, D., and D. Cowan, "Toward a Unified Catalog of Hypermedia Design Patterns," *Proc. 33rd Hawaii Intl. Conf. on System Sciences*, IEEE, vol. 6, Maui, Hawaii, June 2000, download from www.turingmachine.org/~dmg/research/papers/dmg_hicss2000.pdf.
- [GNA99] Gnaho, C., and F. Larcher, "A User-Centered Methodology for Complex and Customizable Web Engineering," *Proc. 1st ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999.
- [HEI02] Heinicke, E., *Layout: Fast Solutions for Hands-On Design*, Rockport Publishers, 2002.
- [IVO01] Ivory, M., R. Sinha, and M. Hearst, "Empirically Validated Web Page Design Metrics," *ACM SIGCHI '01*, Seattle, WA, April 2001, available at <http://www.rashmisinha.com/articles/WebTangoCHI01.html>.
- [JAC02] Jacyntho, D., D. Schwabe, and G. Rossi, "An Architecture for Structuring Complex Web Applications," 2002, available at <http://www2002.org/CDROM/alternate/478/>.
- [KAI02] Kaiser, J., "Elements of Effective Web Design," About, Inc., 2002, available at <http://webdesign.about.com/library/weekly/aa091998.htm>.
- [KAL03] Kalman, S., *Web Security Field Guide*, Cisco Press, 2003.
- [KOC99] Koch, N., "A Comparative Study of Methods for Hypermedia Development," Technical Report 9905, Ludwig-Maximilians Universitat, Munich, Germany, 1999, download from <http://www.dsic.upv.es/~west2001/iwwost01/files/contributions/NoraKoch/hypdev.pdf>.
- [KRA88] Krasner, G., and S. Pope, "A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, no. 3, August/September 1988, pp. 26-49.

- [LOW98] Lowe, D., and W. Hall, eds., *Hypertext and the Web—An Engineering Approach*, John Wiley & Sons, 1998.
- [MCC01] McClure, S., J. Scambray, and G. Kurtz, *Hacking Exposed*, McGraw-Hill/Osborne, 2001.
- [MEN01] Mendes, E., N. Mosley, and S. Counsell, "Estimating Design and Authoring Effort" *IEEE Multimedia*, January–March 2001, pp. 50–57.
- [MIL00] Miller, E., "The Website Quality Challenge," Software Research, Inc., 2000, <http://www.soft.com/eValid/Technology/White.Papers/website.quality.challenge.html>.
- [NIE96] Nielsen, J., and A. Wagner, "User Interface Design for the WWW," *Proc. CHI '96 Conf. On Human Factors in Computing Systems*, ACM Press, 1996, pp. 330–331.
- [NIE00] Nielsen, J., *Designing Web Usability*, New Riders Publishing, 2000.
- [NOR02] Northcutt, S., and J. Novak, *Network Intrusion Detection*, New Riders Publishing, 2002.
- [OFF02] Offutt, J., "Quality Attributes of Web Software Applications," *IEEE Software*, March/April, 2002, pp. 25–32.
- [OLS98] Olsina, L., "Building a Web-Based Information System Applying the Hypermedia Flexible Process Modeling Strategy," *Proc. 1st Intl. Workshop on Hypermedia Development*, 1998.
- [OLS99] Olsina, L. et al., "Specifying Quality Characteristics and Attributes for Web Sites," *Proc. 1st ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999.
- [PER99] Perzel, K., and D. Kane, "Usability Patterns for Applications on the World Wide Web," 1999, download at http://jerry.cs.uiuc.edu/~plop/plop99/proceedings/Kane/perzel_kane.pdf.
- [POW00] Powell, T., *Web Design*, McGraw-Hill/Osborne, 2000.
- [RAS00] Raskin, J., *The Humane Interface*, Addison-Wesley, 2000.
- [RHO98] Rho, Y., and T. Gedeon, "Surface Structures in Browsing the Web," *Proc. Australasian Computer Human Interaction Conference*, IEEE, December, 1998.
- [SCH95] Schwabe, D., and G. Rossi, "The Object-Oriented Hypermedia Design Model," *CACM*, vol. 38, no. 8, August 1995, pp. 45–46.
- [SCH98] Schwabe, D., and G. Rossi, Developing Hypermedia Applications Using OOHDMD, *Proc. Workshop on Hypermedia Development Process, Methods and Models, Hypertext '98*, 1998, download from http://citeseer.nj.nec.com/schwabe_98developing.html.
- [SCH01] Schwabe, D., G. Rossi, and S. Barbosa, "Systematic Hypermedia Application Design Using OOHDMD, 2001, available at <http://www-di.inf.puc-rio.br/~schwabe/HT96WWW/section1.html>.
- [TIL00] Tillman, H. N., "Evaluating Quality On the Net," Babson College, May 30, 2000, available at <http://www.hopetillman.com/findqual.html#2>.
- [TOG01] Tognozzi, B., "First Principles," *askTOG*, 2001, available at <http://www.asktog.com/basics/firstPrinciples.html>.
- [WMT02] Web Mapping Testbed Tutorial, 2002, available at <http://www.webmapping.org/vcgdocuments/vcgTutorial/>.
- [ZHA02] Zhao, H., "Fitt's Law: Modeling Movement Time in HCI," *Theories in Computer Human Interaction*, University of Maryland, October 2002, available at <http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/fitts.html>.

591

习题与思考题

- 19.1 你曾经访问过的具有最佳美学效果的网站是哪个？为什么？
- 19.2 你为一个远程学习公司设计WebApp。你想实现一个基于Internet能够将学习内容传递给学生的“学习引擎”。此学习引擎为传递任何科目的学习内容（内容设计者将准备合适的内容）提供了基础结构。为学习引擎开发界面设计原型。
- 19.3 为SafeHomeAssured.com WebApp设计一个界面原型。尽量去创新，但同时要确保界面符合好的界面设计原则。
- 19.4 复习19.3.1节讨论的Tognozzi的界面设计原则，考虑你熟悉的正在运行的WebApp的每一项原则。按照其达到这种原则的程度对WebApp划分等级（使用等级A、B、C、D或F），解释每一等级划分的理由。

- 19.5 再考虑19.2题描述的“学习引擎”，选择一种适合于WebApp的内容结构，讨论一下为什么要这样选择？
- 19.6 内容结构和WebApp结构之间的区别是什么？
- 19.7 本章我们讨论了WebApp的总体质量特性，选择你认为是最重要的三个质量特性，给出论据说明为什么每一个质量特性都应该在Web工程设计工作中受到重视。
- 19.8 你是否遇到过与19.3.2节所描述的不同的界面控制机制？如果是这样，给出简要描述。
- 19.9 在19.1.1节描述的“WebApp设计质量清单”中，至少再增加5个问题。
- 19.10 当构建现代WebApp时，为什么仅将“艺术理念”作为设计准则是不够的？是否存在一种情况，在这种情况下艺术理念就是要遵循的全部准则？
- 19.11 考虑内容对象order（订购），一旦SafeHomeAssured.com的用户完成了所有部件的选择，并决定购买时，就产生此内容对象。为order及所有合适的设计表示开发UML描述。
- 19.12 当设计19.2题所描述的“学习引擎”时，用UML为将要遇到的内容对象开发三种或四种表示法。592
- 19.13 对MVC体系结构做一些研究，针对19.2题描述的“学习引擎”，看看这种结构是否是合适的WebApp体系结构。
- 19.14 导航语法与导航语义的区别是什么？
- 19.15 做一些研究，并在班里介绍两种或三种完全的超媒体设计模式。
- 19.16 为SafeHomeAssured.com定义两个或三个NSU，对于每一个NSU，给出较详细的描述。

推荐读物与阅读信息

虽然“Web设计”方面的书籍已经写了成百上千种，但很少有书讨论适合设计工作的有意义的技术方法。介绍最多的是各种WebApp设计的有用的指导原则，展示网页及Java程序设计的范例，并讨论对实现现代WebApp很重要的技术细节。这方面的很多书籍中，Powell[POW00]的百科全书式的讨论值得考虑。另外，由Galitz[GAL02]、Heinicke[HEI02]、Schmitt（《Designing CSS Web Pages》，New Riders Publishing，2002）、Donnelly（《Designing Easy-to-Use Websites》，Addison-Wesley，2001）及Nielsen[NIE00]编写的书提供了非常有用的操作。

Wallace和他的同事（《Extreme Programming for Web Projects》，Addison-Wesley，2003）介绍了WebApp设计（和其他主题）的敏捷视图。Conallen（《Building Web Applications with UML》，second edition，Addison-Wesley，2002）、Rosenberg和Scott（《Applying Use-Case Driven Object Modeling with UML》，Addison-Wesley，2001）介绍了使用UML对WebApp建模的详细例子。

Van Duyne和他的同事（《The Design of Sites：Patterns，Principles and Processes》，Addison-Wesley，2002）已经编写了一本很优秀的书籍，此书覆盖了Web工程设计过程的大多数重要方面，并包括设计过程模型及设计模式的详细内容。Wodtke（《Information Architecture》，New Riders Publishing，2003）、Rosenfeld和Morville（《Information Architecture for the World Wide Web》，O'Reilly & Associates，2002）以及Reiss（《Practical Information Architecture》，Addison-Wesley，2000）讲述了内容体系结构和其他主题。

在撰写的关于特定的开发环境的书籍中，也提及了设计技术，感兴趣的读者应该查阅J2EE、Java、ASP.NET、CSS、XML、Perl方面的书籍及多种构建WebApp的应用程序（Dreamweaver、HomePage、FrontPage、GoLive、MacroMedia Flash等），来获得有价值的设计技术。

Web工程设计方面的大量信息源可以在Internet上获得，最新的WWW引用列表可以在SEPA站点<http://www.mhhe.com/pressman>找到。593

第20章 WebApp测试

要点浏览

概念：WebApp测试是一组相关的活动，这些活动都具有共同的目标：发现WebApp的内容、功能、可用性、导航性、性能、容量及安全方面存在的错误。为实现这个目标，要将同时包括评审及可运行测试的测试策略应用于整个Web系统的开发过程中。

人员：所有参加WebApp测试的Web工程师和其他项目共利益者（经理、客户、最终用户）。

重要性：如果最终用户遇到错误，就会动摇他们对WebApp的信心，他们会转向其他地方寻找需要的内容及功能，WebApp就会失败。因此，Web工程师一定要在WebApp投入使用前尽可能多地排除错误。

步骤：在进行webApp测试时，首先关注用户可见的方面，之后进行技术及基础结构方面的测试。要进行7个步骤的测试：内容、界面、导航、构件、配置、性能及安全测试。

工作产品：在某些情况下，需要制定WebApp测试计划。在每种情况下，都需要为每一个测试步骤开发一组测试用例，并且要对记录了测试结果的文档进行维护，以备将来使用。

质量保证措施：虽然从来都不能保证已经完成了需要的每一项测试，但是，能够肯定的是，测试已经发现了错误（并且已经改正）。此外，如果你已经制定了一份测试计划，就可以对照测试计划进行检查，以确保所有计划的测试都已经完成。

关键概念
错误特点
质量维度
策略
测试
兼容性
构件级
配置
内容
数据库
负载
性能
导航
压力
可用性
用户界面

594

Web工程过程通常都很紧迫。在进行表达、策划、分析、设计及构造时，其利益者由于担心来自其他WebApp的竞争，迫于客户的要求，并担心失去市场，因而迫使WebApp仓促上线。其结果是，在Web工程过程中，技术活动通常开始较晚，例如，有时给WebApp测试所剩的时间很短，这可能是一个灾难性的错误。为了避免这种错误的发生，Web工程团队一定要确保每个WebE项目的工作产品都具有高质量。Wallace和他的同事[WAL03]在谈到这一点时讲道：

不应该等到项目完成时才进行测试。在你写第一行代码之前，就开始测试。进行持续及有效的测试，你将开发出更耐用的Web站点。

由于不能在传统意义上对分析及设计模型进行测试，除了可运行的测试，Web工程团队还应该进行正式技术评审（第26章），目的是在WebApp交付最终用户使用之前发现并改正错误。

20.1 WebApp的测试概念

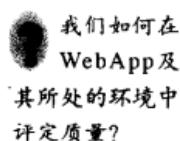
在第13章，我们讲到测试是为了发现（并最终改正）错误而运行软件的过程。这个基本

原理对WebApp也是一样的。事实上，由于基于Web的系统及应用位于网络上，并与很多不同的操作系统、浏览器（或其他诸如PDA或移动电话等接口设备）、硬件平台、通信协议及“暗中的”应用系统进行交互作用，错误的查找对于Web工程师是一个重大的挑战。

为了了解Web工程环境中的测试目标，我们必须考虑WebApp质量的多种维度¹。在此讨论中，我们讨论与Web工程测试工作特别相关的质量维度，同时，我们也讨论作为测试结果所碰到的错误的特性以及为发现这些错误所采用的测试策略。

20.1.1 质量维度

良好的设计应该将质量集成到Web应用系统中。通过对设计模型中的不同元素进行一系列的技术评审，并应用本章所讨论的测试过程，对质量进行评估。评估和测试都要检查下面质量维度中的一项或多项[MIL00]：



- 内容。在语法及语义层对内容进行评估。在语法层，对基于文本的文档进行拼写、标点及文法方面的评估；在语义层，（所表示的信息的）正确性、（整个内容对象及相关对象的）一致性及清晰性都要评估。
- 功能。对功能进行测试，以发现与客户需求不一致的错误。对每一项WebApp功能，评定其正确性、不稳定性和与相应的实现标准（例如，Java或XML语言标准）的总体符合程度。
- 结构。对结构进行评估，以保证它正确地表示WebApp的内容及功能，是可扩展的，及支持新内容、新功能的增加。
- 可用性。对可用性进行测试，以保证接口支持各种类型的用户，各种用户都能够学会及使用所有导航语法及语义。
- 导航性。对导航性进行测试，以保证检查所有的导航语法及语义，发现任何导航错误（例如，死链接、不合适的链接、错误链接）。
- 性能。在各种不同的操作条件、配置及负载下，对性能进行测试，以保证系统响应用户的交互并处理极端的负载情况，而且没有出现不可接受的操作上的性能降低。
- 兼容性。在客户端及服务器端，在各种不同的主机配置下通过运行WebApp对兼容性进行测试，目的是发现针对特定主机配置的错误。
- 互操作性。对互操作性进行测试，以保证WebApp与其他应用系统和（或）数据库有正确接口。
- 安全性。对安全性进行测试，通过评定可能存在的弱点，试图对每一个弱点进行攻击。任何成功的突破尝试都被认为是一个安全漏洞。

595

已经制定了WebApp测试的策略及多种战术，用来测试这些质量维度，在本章后面将对这些进行讨论。

“创新对于软件测试人员是苦乐参半的事情。正当我们似乎知道如何测试一项特定技术时，却出现了一项新技术[WebApp]，以前的好办法都不灵了。” ——James Bach

¹ 第19章也讨论了WebApp的质量。

20.1.2 WebApp环境中的错误

我们已经谈到，在任何软件环境下，测试的主要目的是发现错误（并改正）。对于成功的WebApp测试，它们所遇到的错误具有很多独特的特点[NGU00]：

为什么说在
WebApp运
行中遇到的错误
不同于传统软件
中遇到的错误？

596

1. WebApp测试发现的很多类型的错误都首先表现在客户端（即通过在特定浏览器、PDA或移动电话上实现的接口），Web工程师往往看到了错误的征兆，而不是错误本身。
2. WebApp是在很多不同的配置及不同的环境中实现的，要在最初遇到错误的环境之外再现错误，可能是很困难的，或是不可能的。
3. 虽然许多错误是不正确的设计或HTML（或其他程序设计语言）编码的结果，很多错误的原因都能够追溯到WebApp配置。
4. 由于WebApp位于客户/服务器体系结构之中，在三层体系结构（客户、服务器或网络本身）中追踪错误是很困难的。
5. 某些错误应归于静态的操作环境（即进行测试的特定配置），而有些错误则归于动态的操作环境保护（即瞬间的资源负载或与时间相关的错误）。

上述5个错误特点说明：在Web工程过程中发现的所有错误的诊断中，环境起着非常重要的作用。在某些情况（例如，内容测试）下，错误的位置是明显的；但对于很多其他类型的WebApp测试（例如，导航测试、性能测试、安全测试），错误的根本原因很难确定。

20.1.3 测试策略

WebApp测试策略采用所有软件测试所使用的基本原理（第13章），并建议使用面向对象系统所使用的策略（第14章）。下面的步骤对此方法进行了总结：

KEY POINT
WebApp测试的
总体策略在这里
可以总结为10个
步骤。

597

1. 对WebApp的内容模型进行评审，以发现错误。
2. 对接口模型进行评审，保证适合所有的用例。
3. 评审WebApp的设计模型，发现导航错误。
4. 测试用户界面，发现表现机制和（或）导航机制中的错误。
5. 对选择的功能构件进行单元测试。
6. 对贯穿体系结构的导航进行测试。
7. 在各种不同的环境配置下，实现WebApp，并测试WebApp对于每一种配置的兼容性。
8. 进行安全性测试，试图攻击WebApp或其所处环境的弱点。
9. 进行性能测试。
10. 通过可监控的最终用户群对WebApp进行测试；对他们与系统的交互结果进行评估，包括内容和导航错误、可用性、兼容性、WebApp的可靠性及性能等方面的评估。

由于很多WebApp在不断进化，所以WebApp测试是Web支持人员的一项持续活动，他们使用回归测试，这些测试是从首次开发WebApp时所开发的测试中导出的。

20.1.4 测试策划

对某些Web开发人员来说，策划一词的使用（在任何上下文中）是一种诅咒。如我们在

前面章节中讲述的那样，这些开发人员只是抱着这样的想法开始工作——希望开发一个招人喜爱的WebApp。而Web工程师则认识到策划工作建立了所有工作遵循的路线图，这种努力是值得的。

Splaine和Jaskiel[SPL01]在他们关于WebApp测试的书中说道：

除了最简单的网站，对某种测试策划的需要很快就变得很明显。通常，在特别测试中发现的最初的错误数量很大，以至于在第一次检测到时不能对所有的错误进行定位，这就增加了测试Web站点或Web应用系统的人员的负担。他们不仅要幻想虚构的新的测试，还必须记住以前的测试是如何运行的，以对Web站点或应用系统进行可靠的重复测试，并确保已知的错误都被排除，同时没有引入新的错误。



测试计划确定了
测试任务集、将被开发的工作产品，以及结果评估、记录及重用的方式。

每位Web工程师面临的问题是：我们如何“幻想虚构的新的测试”，这些测试应该集中在什么地方？对这些问题的回答包含在测试计划中。

WebApp的测试计划确定了：(1) 当开始测试时所应用的任务集²；(2) 执行每一项测试任务所生产的工作产品；(3) 以何种方式对测试结果进行评估、记录，以及在进行回归测试时如何重用。在某些情况下，测试计划被集成到项目计划中；而有些情况下测试计划是一个独立的文档。

20.2 测试过程概述

在对Web工程进行测试时，首先测试最终用户能够看到的内容和界面。随着测试的进行，再对体系结构及导航设计的各个方面进行测试。用户可能知道这些WebApp元素，也可能不知道。最后，测试的焦点转到测试技术能力——WebApp基础设施及安装或实现方面的问题，这些方面对于最终用户并不总是可见的。

“一般情况下，在其他应用中使用的软件测试技术[第13、14章]与在基于Web的应用系统中使用的软件测试技术相同……两种测试类型的不同之处在于Web环境中的技术变量增加了。”

——Hung Nguyen

598

图20-1将WebApp的测试过程与第19章讨论的设计金字塔相并列。需要注意的是，当测试流从左到右、从上到下移动时，首先测试WebApp设计中的用户可见元素（金字塔的顶端元素），之后对内部结构的设计元素进行测试。

内容测试（及评审）试图发现内容方面的错误。这项测试活动在很多方面类似于对已写文档的审稿。事实上，大型网站会征召专业审稿人员来发现排字错误、语法错误、内容一致性错误、图形表示错误、交叉引用错误。除了检查静态内容方面的错误，这项测试步骤还要考虑从数据库系统（数据库系统已经被集成到WebApp中）所维护的数据（这些数据是数据库系统的一部分）中导出的动态内容。

界面测试验证用户界面的交互机制及美学方面，目的是发现由于实现糟糕的交互机制而导致的错误，或者由于不小心而产生的遗漏、不一致或歧义性。

² 第2章对任务集进行了讨论。在本书中也使用相关的术语——工作流——来描述完成软件工程活动所需要的一系列任务。

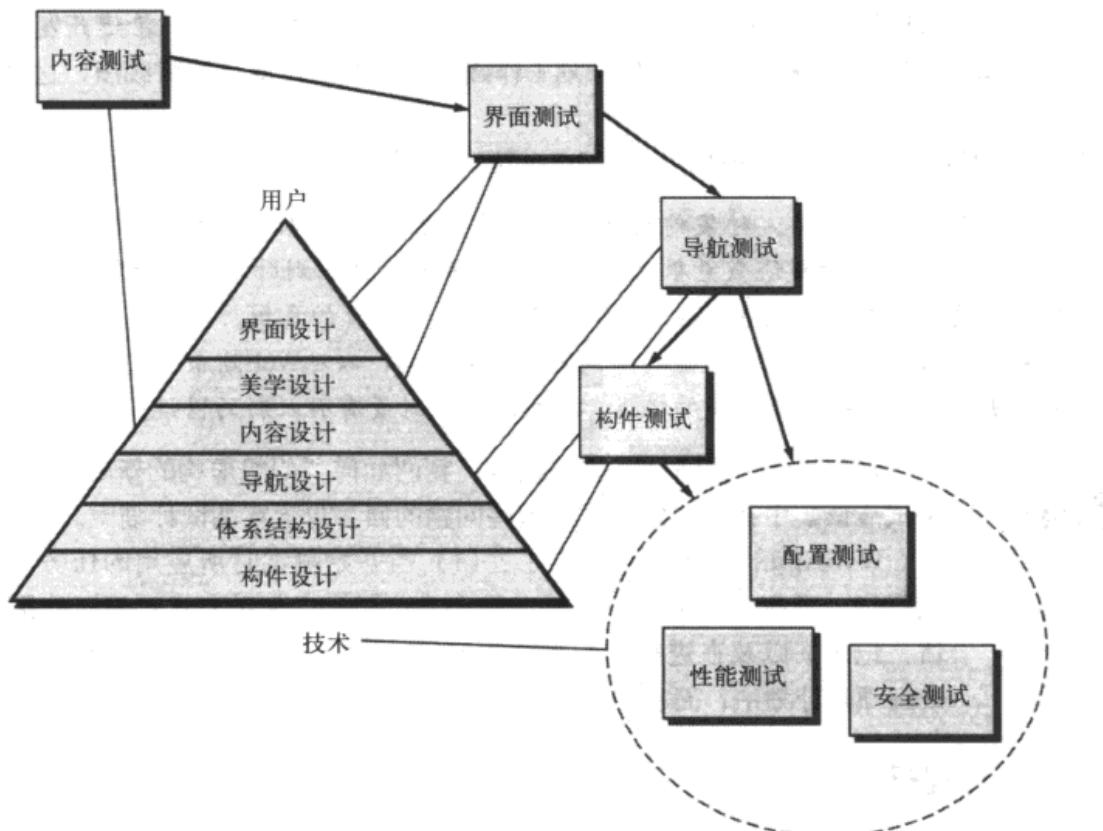


图20-1 测试过程

599

导航测试在测试用例的设计中使用从分析活动中所导出的用例，这些测试用例对照导航设计检查每一个使用场景。对照用例及NSU（第19章），对在界面布局中实现的导航机制（例如，菜单条）进行测试，以保证识别及改正阻止用例完成的任何错误。

构件测试检查WebApp中的内容及功能单元。当考虑WebApp时，单元的概念（在第13章介绍）发生了变化。在内容体系结构中（第19章）“单元”是Web页。每一个Web页包括内容、导航链接及处理元素（表单、脚本、Java小程序）。在WebApp体系结构中的“单元”可能是定义的功能构件（直接向最终用户提供服务）或基础结构构件（使得WebApp达到它的所有能力）。每一个构件的测试方法与传统软件中单个模块的测试方法是一样的，在大多数情况下，使用面向黑盒的测试。然而，如果处理程序复杂，也可能使用白盒测试³。除了功能测试，也对数据库能力进行测试。

KEY POINT
集成测试策略依赖于在设计阶段所选择的WebApp的体系结构。

随着WebApp体系结构的建立，导航测试和构件测试被用于集成测试。集成测试的策略依赖于所选择的内容体系结构及WebApp体系结构（第19章）。如果把内容体系结构设计成线性、网格或简单层次结构，像集成传统软件中的模块那样集成Web页是可能的。然而，如果使用了混合层次或网络（Web）体系结构，集成测试类似于在OO系统中所使用的方法。基于线程的测试（第14章）可用来集成响应一个用户事件所需要的Web页集合（可以使用NSU来定义合适的集合）。每一个线程都得到集成，并且对其实施单独测试。应用回归测试确保没有产生副作用。聚类测试（cluster testing）集成一系列协作的

³ 黑盒测试及白盒测试技术在第14章讨论。

600

页（通过检查用例及NSU来决定）。导出测试用例，以发现协作中的错误。

应该在可能的范围内对WebApp体系结构的每一个元素进行单元测试。例如，在MVC体系结构（第19章）中，对模型、视图及控制器构件进行单独测试。在集成的基础上，对穿过这些元素的控制流及数据进行详细评估。

配置测试试图发现特定的客户或服务器环境中的错误。生成一个交叉引用矩阵，定义所有可能的操作系统、浏览器⁴、硬件平台及通信协议，然后进行测试，发现与每一种可能配置相关的错误。

安全性测试将一系列设计的测试合并起来，攻击WebApp及其环境中的弱点，目的是证明安全性破坏是可能的。

性能测试包括一系列测试，设计这些测试用来评估：（1）WebApp的响应时间及可靠性如何受增长的用户通信量的影响？（2）哪些WebApp构件与性能降级有关？哪些使用特点造成了降级的发生？（3）性能降级是如何影响整个WebApp的目标及需求的？

TASK SET

WebApp测试

1. 评审共利益者的需求。标识关键用户目标，对每类用户的用例进行评审。
2. 建立优先级，以确保每一个用户目标都将被适当地测试。
3. 根据要实施的测试类型（20.2节）的描述，定义WebApp测试策略。
4. 开发一个测试计划。
 - 规定测试进度，并对每个测试分配职责。
 - 指定自动化测试工具。
 - 对每一类测试，规定验收标准。
 - 详细说明缺陷跟踪机制。
 - 定义问题报告机制。
5. 进行“单元”测试。
 - 评审内容的语法及语义错误。
 - 评审内容的许可性。
 - 测试接口机制的正确操作。
 - 测试每一个构件（例如，脚本），确保正确的功能。
6. 进行“集成”测试。
 - 对照用例，测试界面的语义。
 - 实施导航测试。
7. 进行配置测试。
 - 评估客户端的配置兼容性。
 - 评估服务器端的配置。
8. 进行性能测试。
9. 进行安全性测试。

⁴ 对于HTML和Javascript的翻译，浏览器在实现其自身微妙的不同“标准”方面很糟糕。

20.3 内容测试



虽然正式技术评审不是测试的组成部分，但应执行内容评审，以确保内容的质量。

WebApp内容中的错误可以小到简单的印刷错误，或大到不正确的信息、不合适的组织，或者违背知识产权法。内容测试试图在用户碰到这些问题及很多其他问题之前就发现它们。

内容测试结合了评审和可运行的测试用例的生成。采用评审来发现内容中的语义错误（在20.3.1节讨论）。可运行的测试用于发现内容错误，这些错误可被跟踪到动态导出的内容（这些内容由从一个或多个数据库中获取的数据驱动）。

20.3.1 内容测试的目标



601

内容测试的目标是：(1)发现内容中的语法错误；(2)发现语义错误；(3)发现结构错误。

内容测试具有三个重要的目标：(1)发现基于文本的文档、图形表示和其他媒体中的语法错误（例如，打字错误、文法错误）；(2)发现当导航发生时所展现的任何内容对象中的语义错误（即信息的精确性和完备性方面的错误）；(3)发现展示给最终用户的内容的组织或结构方面的错误。

为了达到第一个目标，可以使用自动拼写和语法检查工具。然而，很多语法上的错误会逃避这种工具的检查，而必须由审查人员（测试人员）人为发现。如我们在前面章节中所讲的那样，文本编辑是发现语法错误的唯一最好方法。

语义测试关注于每一个内容对象所显示的信息方面。评审人员（测试人员）必须回答以下问题：

要发现内容中的语义错误，应该询问和回答哪些问题？

- 信息确实准确吗？
- 信息简洁扼要吗？
- 内容对象的布局对于用户来说容易理解吗？
- 嵌入在内容对象中的信息易于被发现吗？
- 对于从其他地方导出的所有信息，是否提供了合适的引用？
- 显示的信息是否是内部一致的？与其他内容对象中所显示的信息是否一致？
- 内容是否具有攻击性？是否容易误解？或者是否会引发诉讼？
- 内容是否侵犯了现有的版权或商标？
- 内容是否包括补充现有内容的内部链接？链接正确吗？
- 内容的美学风格是否与界面的美学风格相矛盾？

对于大型的WebApp（包含成百上千个内容对象）来说，要获得所有这些问题的答案可能是一项令人畏惧的任务。然而，不能发现语义错误将动摇用户对WebApp的信任，并且会导致基于Web的应用系统的失败。

内容对象存在于具有特定风格的体系结构之中（第19章）。在内容测试期间，要对内容体系结构的结构及组织进行测试，以确保将所需要的内容以合适的顺序和关系展现给最终用户。例如，SafeHomeAssured.com WebApp⁵显示了关于传感器的多种信息，其中传感器是安全和

⁵ SafeHomeAssured.com WebApp作为贯穿本书第三部分的一个例子。

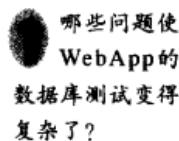
监视产品的一部分。内容对象提供描述信息、技术规格说明、照片和相关的信息。SafeHome-Assured.com内容体系结构的测试试图发现这种信息的表示方面的错误（例如，用传感器Y的照片来描述传感器X）。

20.3.2 数据库测试

现代的Web应用系统要比静态的内容对象做更多的事情。在很多应用领域中，WebApp要与复杂的数据库管理系统连接，并构建动态的内容对象，这种对象是使用从数据库中获取的数据实时创建的。

例如，用于金融服务的WebApp能够产生某种特殊产权（例如，股票或共有基金）的复杂的文本信息、表格信息和图形信息。当用户已经申请了某种特殊的产权信息之后，就会自动创建表示这种信息的复合内容对象。为了完成此任务，需要下面的步骤：(1) 查询大型产权数据库；(2) 从数据库中抽取相关的数据；(3) 一定将抽取的数据组织为一个内容对象；(4) 将这个内容对象（代表由某个最终用户请求的定制信息）传送到客户环境显示。每一个步骤的结果都可能发生错误，并且一定会发生。数据库测试的目标是发现这些错误。

WebApp的数据库测试会由于以下多种原因而变得复杂：



1. 客户端请求的原始信息很少能够以能被输入到数据库管理系统(DBMS)中的形式（例如，结构化查询语言SQL）表示出来。因此，应该设计测试，用来发现在将用户的请求翻译成能够被这些DBMS处理的格式的过程中所产生的错误。

2. 数据库可能离装载WebApp的服务器很远。因此，应该设计测试，用来发现WebApp和远程数据库之间的通信所存在的错误⁶。

3. 从数据库中获取的原始数据一定要传递给WebApp服务器，并且这些原始数据要被正确地格式化，以便随后传递给客户端。因此，应该设计测试，用来证明WebApp服务器接收到的原始数据的有效性，并且还要生成另外的测试，证明转换的有效性，将这种转换应用于原始数据，能够生成有效的内容对象。

4. 动态内容对象一定要以能够显示给最终用户的形式传递给客户端。因此，应该设计一系列的测试，用来发现内容对象格式方面的错误；以及测试与不同的客户环境配置的兼容性。

考虑这四种因素，对图20-2中记录的每一“交互层”[NGU01]，都应该应用测试用例的设计方法。测试应该保证：(1) 有效信息通过界面层在客户与服务器之间传递；(2) WebApp正确地处理脚本，并且正确地抽取或格式化用户数据；(3) 用户数据被正确地传递给服务器端的数据转换功能，此功能将合适的查询格式化（例如，SQL）；(4) 查询被传递到数据管理层⁷，此层与数据库访问程序（很可能位于另一台机器）通信。

通常使用可复用的构件来构造图20-2所示的数据转换层、数据管理层和数据库访问层，这些可复用的构件都分别进行了验证，并且被打成一个包。如果是这种情况，WebApp的测试则集中在图20-2所示的客户层与头两个服务器层（WebApp和数据转换）之间交互的测试用例的设计。

⁶ 当遇到分布式数据库，或者需要访问数据仓库时，这些测试可能变得非常复杂。

⁷ 数据管理层一般合并了SQL调用层接口(SQL-CLI)，如Microsoft OLE/ADO或Java数据库连接(JDBC)。

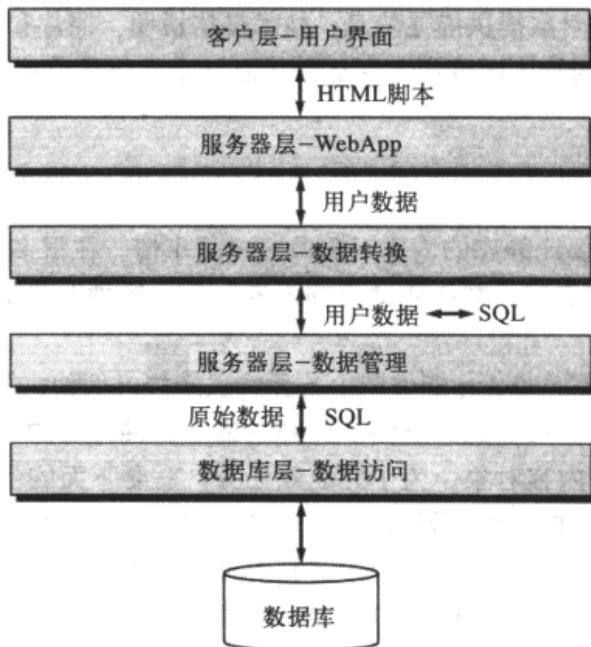


图20-2 交互层

应该对用户界面层进行测试，确保对每一个用户查询都正确地构造了HTML脚本，并且正确地传输给了服务器端。还应该对服务器端的WebApp层进行测试，确保能够从HTML脚本中正确地抽取出用户数据，并且正确地传输给服务器端的数据转换层。

应该对数据转换功能进行测试，确保创建了正确的SQL，并且传给合适的数据管理构件。

604 对合理地设计这些数据库测试需要了解的主要技术进行详细讨论超出了本书的范围，感兴趣的读者应参看[SCE02]、[NGU01]和[BRO01]。

“作为e-顾客（无论是商家，还是消费者），对于频繁停机、在事务处理的中途停止或可用性感觉不好的网站，人们不可能有信心。因此，测试在整个开发过程中具有关键作用。”

——Wing Lam

20.4 用户界面测试

在Web系统开发过程中，需要在三个阶段对WebApp的用户界面进行验证与确认。在表达和需求分析阶段（第17、18章），对界面模型进行评审，确保与用户需求及分析模型的其他元素相一致；在设计阶段（第19章），对界面设计模型进行评审，确保已经达到了为所有用户界面建立的通用质量标准，并且正确描述了特定于应用系统的界面设计问题；由于用户交互是通过界面的语法和语义来表示的，在测试阶段，重点转移到特定于应用系统的用户交互方面的执行。另外，测试提供了对可用性的最终评估。

20.4.1 界面测试策略

界面测试的总体测试策略是：(1)发现与特定的界面机制相关的错误（例如，未能正确执行菜单链接的错误，或者输入数据格式的错误）；(2)发现界面实现导航语义方式的错误、WebApp的功能性错误或内容显示错误。为了实现此策略，必须达到以下一些目标：



除了面向WebApp的详细设计说明书以外，这里记录的界面策略可应用于所有类型的客户/服务器软件。

- 对界面要素进行测试，确保设计原则、美学和相关的可视化内容对用户有效，且没有错误。要素包括字体、颜色、框架、图片、边界、表以及WebApp运行中所产生的相关元素。
- 采用与单元测试类似的方式测试单个界面机制。例如，设计测试用例对所有的表单、客户端脚本、动态HTML、CGI脚本、流动内容及应用系统的特定界面机制（例如，电子商务应用系统中的购物车）进行测试。在很多情况下，测试可以专门集中在这些机制中的一种（“单元”），而不包括其他界面要素及功能。
- 对于特殊的用户类别，在用例或NSU（第19章）的环境中测试每一种界面机制。这种测试方法与集成测试（第13章）类似，因为当界面机制被集成到一起使得用例或NSU执行时，才能够进行测试。605
- 与选择用例及NSU有所不同，此方法要对全部的界面进行测试，发现界面的语义错误。这种测试方法类似于确认测试（第13章），因为其目的是证明与特定的用例或NSU语义相一致。正是在这个阶段，进行一系列的可用性测试。
- 在多种环境（例如，浏览器）中对界面进行测试，确保其兼容性。实际上，可以将这一系列测试看成是配置测试的一部分。

20.4.2 测试界面机制

当用户与WebApp交互时，通过一种或多种界面机制发生交互，在下面的段落中，对每一种界面机制，我们简要介绍一下测试时需要考虑的内容[SPL01]：



外部链接测试应该贯穿WebApp的整个生命期，支持策略中应该包括有规律的、定期的链接测试。

链接。对每一个导航链接进行测试，确保获得了正确的内容对象或功能⁸。Web工程师构建与界面布局（例如，菜单条、索引项）相联系的所有链接列表，然后分别运行每一个链接。另外，一定要检查每一个内容对象内的链接，以发现错误的URL或者到不正确的內容对象或功能的链接。最后，应该对链接到外部WebApp的链接进行精确性测试，并且对其进行估计，决定随着时间的推移这些链接将变得无效的风险有多大。

表单。在宏观层次上进行测试，以确保：(1) 对表单中的标识域给出正确标记，并且为用户可视化地标识出强制域；(2) 服务器接收到了表单中包括的所有信息，并且在客户端与服务器之间的传输过程中没有数据丢失；(3) 当用户没有从下拉菜单或按钮组中进行选择时，使用合适的缺省项；(4) 浏览器功能（例如，“回退”箭头）没有破坏输入到表单中的数据；(5) 对输入数据执行错误检查的脚本工作正常，并且提供了有意义的错误标志信息。

在更具体的层次上，测试应该确保：(1) 表单域有合适的宽度和数据类型；(2) 表单建立了合适的安全措施，防止用户输入的文本字符串长度大于某预先定义的最大值；(3) 对下拉菜单中的所有合适的选项进行详细说明，并按照对最终用户有意义的方式排序；(4) 浏览器“自动填充”特性不会导致数据输入错误；(5) tab键（或其他键）能够使输入焦点在表单域之间正确移动。

⁸ 这些测试可以作为界面测试和导航测试的一部分。



每当流行的浏览器的新版本发布时，都应该重新进行客户端脚本的测试及与动态HTML相关的测试。

606

客户端脚本。当脚本运行时，使用黑盒测试发现处理中的一些错误。由于脚本输入通常来自作为表单处理组成部分而提供的数据，这些测试通常与表单测试联合进行。应该进行兼容性测试，确保所选择的脚本语言在支持WebApp的环境配置中工作正常。另外，还要测试脚本本身。Splaine和Jaskiel[SPL01]建议：“应该保证公司的[WebApp]标准明确了客户端（和服务器端）脚本使用的首选语言和脚本语言的版本。”

动态HTML。运行包含动态HTML的每个网页，确保动态显示正确。另外，应该进行兼容性测试，确保动态HTML在支持WebApp的环境配置中工作正常。

弹出窗口⁹。进行一系列测试，以确保：(1) 弹出窗口具有合适的大小和位置；(2) 弹出窗口没有覆盖原始的WebApp窗口；(3) 弹出窗口的美学设计与界面的美学设计相一致；(4) 附加到弹出窗口上的滚动条和其他控制机制被正确定位，并具有所需的功能。

CGI脚本。一旦已经接收到经过验证的数据，黑盒测试的侧重点集中在数据的完整性（当数据被传递给CGI脚本）和脚本处理。此外，进行性能测试，确保服务器端的配置符合CGI脚本多重调用的处理要求[SPL01]。

流动内容。测试应该证明流动数据是最新的，并且显示正确，能够无错误地暂停，而且很容易重新启动。

cookie。服务器端的测试和客户端的测试都需要。在服务器端，测试应该确保一个cookie被正确构造（包含正确的数据），并且当请求特定的内容和功能时，此cookie能够被正确地传输到客户端。此外，测试此cookie是否具有合适的持续性，确保有效日期正确。在客户端，通过测试来确定WebApp是否将已有的cookie正确地附到了特定的请求上（发送给服务器）。

特定于界面机制的应用。测试是否与界面机制定义的功能和特性清单相符合。例如，Splaine和Jaskiel[SPL01]为电子商务应用中所定义的购物车功能提出了下面的检查单：

- 对能够放置到购物车中的物品的最小数量和最大数量进行边界测试（第14章）。
- 对一个空的购物车的“结账”请求进行测试。
- 测试从购物车中正确地删除一件物品。
- 测试一次购买操作是否清空了购物车中的内容。
- 测试购物车内容的持续性（这一点应该作为客户需求的一部分详细说明）。
- 如果用户请求显示已经保存的内容，测试WebApp将来是否能够记起购物车的内容（假设没有购买活动发生）。

607

20.4.3 测试界面语义

一旦对每一个界面功能都已经进行了“单元”测试，就可以将界面测试的焦点转移到界面的语义。界面的语义测试“评价设计在照顾用户、提供清楚的指导、传递反馈并保持语言与方法的一致性方面做得如何”[NGU01]。

⁹ 弹出窗口的使用已经变得很普遍，并且很多用户都对它们很敏感。应该明智地使用它们，否则，就不要使用。

彻底复习一下界面设计模型，我们就能够得到前面的段落中所蕴涵的问题的部分答案。然而，一旦实现了WebApp，就应该对每一个用例场景（针对每一类用户）进行测试。本质上，用例就变成了设计测试序列的输入。测试序列的目的是发现那些妨碍用户获得与用例相关的目标的错误。

就如同要对每一个用例进行测试一样，Web开发团队需要维护一份检查单，确保每一个菜单项都至少被运行一次，并且内容对象中的每一个嵌入的链接都已经被使用。此外，测试序列应该包括不适当的菜单选择和链接使用，目的是确定WebApp是否提供了有效的错误处理和恢复。

20.4.4 可用性测试

WebRef

可用性测试的有价值的指导可以在www.ahref.com/guides/design/199806/0615jef.html找到。

可用性测试也评价用户在多大程度上能够与WebApp进行有效交互，WebApp在多大程度上指导用户行为，提供有意义的反馈，并坚持一致的交互方法。从这种意义上说，可用性测试与界面语义测试是相似的（20.4.3节）。可用性检查和测试不是集中在某交互目标的语义上，而是要确定WebApp界面在多大程度上使用户的生活变得轻松¹⁰。

可用性测试可以由Web开发团队设计，但是测试本身由最终用户进行。在测试时，可以采用下面的步骤[SPL01]：

1. 定义一组可用性测试类别，并确定每类测试的目标。
2. 设计测试，使其能够评估每个目标。
3. 选择将执行测试的参与者。
4. 当进行测试时，指导参与者与WebApp的交互。
5. 开发一种机制来评估WebApp的可用性。

[608]

可用性测试可能发生在多种不同的抽象级别：（1）对特定的界面机制（例如，表单）的可用性进行评估；（2）对所有网页（包括界面机制、数据对象及相关的功能）的可用性进行评估；（3）考虑整个WebApp的可用性。

可用性测试的第一步是确定一组可用性类别，并对每一类别建立测试目标。下面的测试类别和目标（以问题的形式描述）举例说明了这种方法¹¹：

可用性的哪些特性变成了测试的焦点？描述了哪些特殊的目标？

交互性——交互机制（例如，下拉菜单、按钮、指针）容易理解和使用吗？

布局——导航机制、内容和功能放置的方式是否能让用户很快地找到它们？

可读性——文本是否很好地编写，并且是可理解的¹²？图形表示是否容易理解？

美学——布局、颜色、字体和相关的特性是否使WebApp易于使用？用户对WebApp的外观是否“感觉舒适”？

显示特性——WebApp是否使屏幕的大小和分辨率得到了最佳使用？

时间敏感性——是否能够及时使用或获取重要的要素、功能和内容？

¹⁰ 这种问题常用术语“用户友好性”来表示。当然，问题在于用户对于“友好”界面的感觉可能每个人是完全不同的。

¹¹ 对于其他的可用性问题，参见第12章的“可用性”讨论。

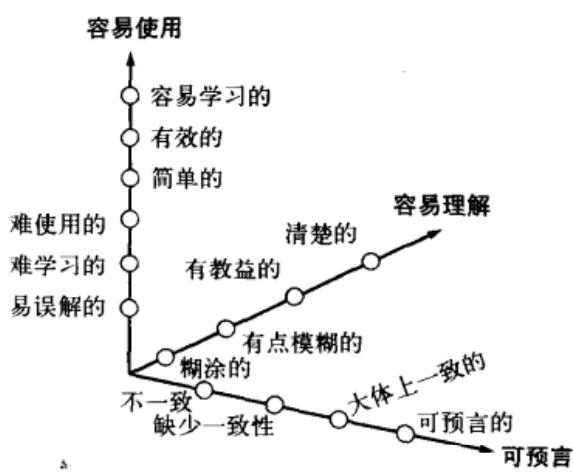
¹² FOG可读性指数和其他工具可以用来定量地评估可读性。更多的细节参见<http://developer.gnome.org/documents/usability/usability-readability.html>。

个性化——WebApp是否能适应多种用户或个别用户的特殊要求?

可访问性——残疾人是否可以使用该WebApp?

对于上面每一种可用性，都需要设计一系列测试。在某些情况下，“测试”可以是对网页的可视化审查；而在有些情况下可以重新执行界面的语义测试，但在下面的实例中，可用性是极为重要的。

作为一个例子，我们考虑对交互和界面机制进行可用性评估。Constantine和Lockwood [CON03]建议应该对下列界面要素进行可用性评审和测试，它们是：动画、按钮、颜色、控制、对话、域、表单、框架、图形、标签、链接、菜单、消息、导航、页、选择器、文本和工具条。
[609]当评估每个要素时，可以由执行测试的用户对其进行定性分级。图20-3描述了用户可能选择的一系列评估“级别”。这些级别可以应用于每个单独的要素、所有的网页或者整个WebApp。



20.4.5 兼容性测试

KEY POINT
一定要在不同的环境中运行WebApp。不同的计算机、显示设备、操作系统、浏览器和网络连接速度都会对WebApp的运行造成很大的影响。每一种计算配置都可能使客户端的处理速度、显示分辨率和连接速度有所不同。操作系统反复无常的行为可能导致WebApp的处理问题。不管WebApp中HTML标准化程度如何，不同的浏览器有时会产生稍微不同的结果。特殊的配置所需要的插件可能容易获得，也可能不容易获得。

在某些情况下，小的兼容性问题显得不很严重；而在有些情况下，就可能遇到严重的错误。例如，下载速度可能变得让人无法接受，缺少所需要的插件可能使内容难以获得，浏览器的不同可能会戏剧性地改变页面的布局，字型可能会被改变且变得难以辨认，或者表单可能被错误地组织。兼容性测试试图在WebApp上线前发现这些问题。

兼容性测试的第一步是定义一组“通常遇到”的客户端计算配置和它们的变型。实际做法是创建一种树结构，并在上面标识每一种计算平台、典型的显示设备、此平台支持的操作系统、可用的浏览器、可靠的Internet连接速度及类似信息。下一步，Web工程团队导出一系列的兼容性确认测试，可以从现有的界面测试、导航测试、性能测试和安全性测试中导出。这些测试的目的是发现由于配置差异所导致的错误和运行问题。

SAFEHOME

WebApp测试

[场景] Doug Miller的办公室。

[人物] Doug Miller, SafeHome软件工程团队经理; Vinod Raman, SafeHome软件工程师团队成员。

[对话]

Doug: 对于SafeHomeAssured.com电子商务WebApp 0.0版, 你是怎样认为的?

Vinod: 外包供应商已经做了很好的工作。Sharon[供应商的开发经理]告诉我, 他们正在按我们说的进行测试。

Doug: 我希望你和团队的其他人能够对电子商务网站做一点非正式的测试。

Vinod (作苦相): 我想我们将雇用第三方测试公司对WebApp进行确认测试。我们仍在忙于推出产品软件。

Doug: 我们将雇用测试供应商进行性能测试和安全性测试, 并且我们的外包供应商已经在进行测试了。只是想从另外一种角度看是否会有帮助, 况且, 我们想控制成本, 所以……

Vinod (叹息): 你在期待什么?

Doug: 我想确信界面和所有的导航都是可靠的。

Vinod: 我想我们可以从针对每个主要界面功能的用例开始:

学习SafeHome**详细说明你需要的SafeHome系统****购买一套SafeHome系统****取得技术支持**

Doug: 很好。但是, 要走通所有的导航路径, 才能得出结论。

Vinod (浏览记录用例的笔记本): 是的, 当你选择“**详细说明你需要的SafeHome系统**”时, 此时将让你:

选择SafeHome构件**获得SafeHome构件建议**

我们要当心每一条路径的语义。

Doug: 当你测试时, 需要检查出现在每一个导航节点的内容。

Vinod: 当然……还有功能元素。谁在测试可用性?

Doug: 哦……测试供应商将配合可用性测试。我们已经雇用了一个市场调查公司列出20个进行可用性研究的典型用户, 但是, 如果你发现了任何可用性问题……

Vinod: 我会转给他们。

Doug: 谢谢! Vinod。

20.5 构件级测试

构件级测试也称功能测试, 它集中于一系列的测试, 试图发现WebApp功能方面的错误。每一个WebApp功能都是一个软件模块(用多种程序设计语言或脚本语言中的一种实现的), 并且可以用第14章讨论的黑盒(及在某些情况下的白盒)技术对其进行测试。

611 构件级测试用例通常受表单级的输入驱动。一旦定义了表单数据，用户就可以选择按钮或其他控制机制来启动运行。下面是典型的测试用例设计方法（第14章）：

- 等价类划分——将功能的输入域划分为输入（种）类，可以从这些输入类中导出测试用例。通过对输入表单进行评估，可以决定哪些数据类与功能有关。对于每个输入类，都导出它的测试用例，并运行，而其他的输入类保持不变。例如，一个电子商务应用系统可能实现一个计算运输费用的功能。在通过表单提供的多种运输信息中，有用户的邮政编码。就可以设计测试用例，通过详细说明邮政编码的值试图发现邮政编码处理中的错误，这种方法可以发现不同的错误类（例如，不完整的邮政编码，不正确的邮政编码，不存在的邮政编码，错误的邮政编码格式）。
- 边界值分析——对表单数据的边界进行测试。例如，前面提到的运费计算功能需要指出产品运输所需要的最大天数，在表单中记录的最少天数是2天，最大天数是14天。然而，边界值测试可能输入值0、1、2、13、14和15，来确定功能如何对有效输入边界之内、之外及边界点的数据做出反应¹³。
- 路径测试——如果功能的逻辑复杂性较高¹⁴，可以使用路径测试来确保程序中的每条独立路径都已经被执行。

除了这些测试用例设计方法，还可以使用称为强制错误测试（forced error testing）[NGU01]的技术导出测试用例，这些测试用例故意使WebApp构件进入错误条件，目的是发现在错误处理过程中发生的错误（例如，不正确或不存在的错误提示信息，由于错误的发生导致WebApp失效，由错误的输入而导致的错误的输出，与构件处理有关的副作用）。

每个构件级测试用例详细说明了所有的输入值和由构件提供的预期的输出。可以将测试过程中产生的实际输出数据记录下来，以供将来的支持和维护阶段参考。

在很多情况下，WebApp功能的正确运行依赖于与数据库的正确接口，其中数据库可能位于WebApp的外部。因此，数据库测试是构件测试中不可分割的一部分。Hower[HOW97]对此进行讨论时这样写道：

WebRef

多种测试资源可以在www.pantos.org/atw/xref.html找到。

数据库驱动的网站可能包括Web浏览器、操作系统、插件应用、通信协议、Web服务器、数据库、[脚本语言]程序……强化安全及防火墙之间的复杂交互。这种复杂性使得不可能测试每种可能的依赖和使站点发生错误的每件事情。而且，一般网站开发项目的时间进度安排都很紧张，所以最好的测试方法是进行风险分析，以确定测试工作该集中在哪里？风险分析应该考虑到测试环境与实际产品环境匹配的紧密程度……风险分析中其他方面的考虑一般还包括：

- 网站中的哪种功能对于其目的是最关键的？
- 网站的哪些区域与数据库的交互量最大？
- 站点的CGI、applet、ActiveX等构件的哪些方面最复杂？
- 哪些类型的问题会导致最多的抱怨或最坏的宣传？
- 网站的哪些部分将最受欢迎？
- 网站的哪些方面的安全性风险最高？

¹³ 在这种情况下，一个较好的输入设计会排除潜在的错误。最大天数可以从下拉菜单中选择，从而排除用户指定禁止的输入。

¹⁴ 逻辑复杂性可以通过计算算法的环路复杂性来确定。详细内容见第14章。

当为WebApp构件及相关的数据库功能设计测试用例时，应该考虑Hower所讨论的与风险相关的所有问题。

20.6 导航测试

用户在WebApp中游历与访问者在商店或博物馆中漫步很相似。有很多路径可走，可以有很多站，有很多事情去学习和观看，启动很多活动，并且可以做决策。如我们所讨论的那样，每个访问者到来时都有一系列的目标，在这种意义上，这种导航过程是可预测的。同时，导航过程又可能是无法预测的，因为访问者受到他所看到的或学到的某事的影响，可能选择一条路径或启动一个动作，而这对于最初的目标来说并不是典型的路径或动作。导航测试的工作是：(1) 确保允许WebApp用户经由WebApp游历的机制都是功能性的；(2) 确认每个导航语义单元（NSU）都能够被合适的用户类获得。

“我们没有迷路，我们面临定位挑战。”

—John M. Ford

20.6.1 测试导航语法

实际上，导航测试的第一个阶段在界面测试期间就开始了。应对导航功能进行测试，确保每个导航都执行了预计的功能。Splaine和Jaskiel[SPL01]建议应该对下面的每个导航功能进行测试：

- **导航链接**——WebApp中的内部链接，到其他WebApp的外部链接及特定网页中的锚都应 613 该被测试，确保选择链接时，能够获得正确的内容和功能。
- **重定向**——当用户请求一个不存在的URL，选择一个目标地址已经被移走或者名字已经被改变的链接时，就会用到这些重定向的链接。应该给用户显示一条提示信息，并且将导航重定向到另一页（例如，主页）。通过请求不正确的内部链接或外部URL，并且评价WebApp如何处理这些请求，来对重定向进行测试。
- **书签**——虽然书签是浏览器功能，还是应该对WebApp进行测试，确保当创建一个书签时，能够抽取出有意义的页标题。
- **框架和框架集**——每个框架包含特定的网页内容；一个框架集包含多个框架，并且可以使多个网页同时显示。由于框架和框架集彼此之间可以嵌套，应该对这些导航和显示机制进行内容的正确性、合适的布局和大小、下载性能和浏览器性能方面的测试。
- **站点地图**——对入口进行测试，确保链接引导用户到达合适的内容和功能。
- **内部搜索引擎**——复杂的WebApp通常包括成百上千的内容对象。内部搜索引擎允许用户在WebApp中搜索关键字，来发现所需要的内容。搜索引擎测试确认搜索的精确性和完备性、搜索引擎的错误处理特性及高级的搜索特性（例如，在搜索域中布尔操作符的使用）。

前面已经提到的某些测试可以由自动工具执行（例如，链接检查），而有些要手工设计和执行。导航测试的目的始终是确保在WebApp上线之前发现导航功能方面的错误。

20.6.2 测试导航语义

在第19章，我们将导航语义单元（NSU）定义为“一组信息和相关的导航结构，在完成

相关的用户需求的子集时，这些导航结构会相互协作” [CAC02]。每个NSU由一系列连接导航节点（例如，网页、内容对象或功能）的导航路径（称为“导航的路”）定义。作为一个整体，每个NSU允许用户获得特殊的需求，这种特殊的需求是针对某类用户，由一个或多个用例定义的。导航测试应检查每个NSU，以确保能够获得这些需求。

在测试NSU时，Web工程团队一定要回答下面的问题：

614

 在测试NSU时，必须提问和回答哪些问题？

- 此NSU是否没有错误地全部完成了？
- 在为此NSU定义的导航路径的上下文中，（为一个NSU定义的）每一个导航节点是否都是可达的？
- 如果使用多条导航路径都能完成此NSU，每一条相关的路径是否都已经被测试？
- 如果使用用户界面提供的指导来帮助导航，当导航进行时，方向正确并可理解吗？
- 是否具有返回到前一个导航节点及导航路径开始位置的机制（不同于浏览器的“回退”箭头）？
- 大型导航节点（即一个长的网页）中的导航机制工作正常吗？
- 如果一个功能在一个节点上运行，并且用户选择不提供输入，NSU的剩余部分能完成吗？
- 如果一个功能在一个节点上运行，并且在功能处理时发生了一个错误，NSU能完成吗？
- 在到达所有节点之前，是否有办法终止导航？然后又能返回到导航被终止的地方，并从那里继续？
- 从站点地图可以到达每一个节点吗？节点的名字对最终用户有意义吗？
- 如果可以从某外部的信息源到达NSU中的一个节点，推移到导航路径的下一个节点可能吗？返回到导航路径的前一个节点可能吗？
- 当运行NSU时，用户知道他在内容体系结构中所处的位置吗？

 如果在Web工程的分析和设计中没有创建NSU，可以将用例应用于导航测试用例的设计，需要提问和回答的一系列问题是一样的。

如同界面测试和可用性测试，导航测试应该由尽可能多的不同的支持者进行。测试的早期阶段由Web工程师进行，但后来的测试应该由其他的项目共利益者、独立的测试团队进行，最后应该由非技术用户进行，目的是彻底检查WebApp导航。

20.7 配置测试

615

配置的可变性和不稳定性是使Web工程面临挑战的重要因素。硬件、操作系统、浏览器、存储容量、网络通信速度和多种其他的客户端因素对每个用户都是难于预料的。另外，某个用户的配置可能会有规律地改变（例如，操作系统升级、新的ISP和连接速度），其结果可能是客户端环境容易出错，这些错误既微妙又重要。如果两个用户不是在相同的客户端配置中工作，一个用户对WebApp的印象及与WebApp的交互方式可能与另一个用户的体验有很大不同。

配置测试的工作不是去检查每一个可能的客户端配置，而是测试一组很可能的客户端和服务器端配置，确保用户在所有配置中的体验都是一样的，并且将特定于特殊配置的错误分离出来。

20.7.1 服务器端问题

在服务器端，设计配置测试用例来验证所计划的服务器配置（即WebApp服务器、数据库服务器、操作系统、防火墙软件、并发应用系统）能够支持WebApp，而不会发生错误。实质上，WebApp被安装在服务器端环境，并进行测试，目的是发现与配置有关的错误。

当设计服务器端的配置测试时，Web工程师应该考虑服务器配置的每个构件。在服务器端的配置测试期间，需要询问及回答以下问题：

 当进行服务器端配置测试时，必须询问和回答哪些问题？

- WebApp与服务器操作系统完全兼容吗？
- 当WebApp运行时，系统文件、目录和相关的系统数据是否被正确创建？
- 系统安全措施（例如，防火墙或加密）允许WebApp运行，并对用户提供服务，而不发生冲突或性能下降吗？
- 是否已经对所选择的具有分布式服务器配置¹⁵（假如存在一种配置）的WebApp进行了测试？
- 此WebApp是否与数据库软件进行了适当的集成？是否对数据库的不同版本敏感？
- 服务器端的WebApp脚本运行正常吗？
- 系统管理员错误对WebApp运行的影响是否已经被检查？
- 如果使用了代理服务器，在站点测试时，是否已经明确这些代理服务器在配置方面的差异？

20.7.2 客户端问题

在客户端，配置测试更多地集中在WebApp与配置的兼容性，这些配置包括下面构件的一种或多种改变[NGU01]：

- 硬件——CPU、内存、存储器和打印设备。
- 操作系统——Linux、Macintosh操作系统、Microsoft Windows、基于移动的操作系统。
- 浏览器软件——Internet Explorer、Mozilla/Netscape、Opera、Safari及其他浏览器。
- 用户界面构件——Active X、Java applets等。
- 插件——QuickTime、RealPlayer等。
- 连接性——电缆、DSL、常规的调制解调器、T1。

除了这些构件，其他配置变量包括网络软件、ISP的难以预测的变化及并发运行的应用系统。

为了设计客户端配置测试，Web工程团队必须将配置变量的数量减少到可管理的数目¹⁶。为了实现这一点，要对每一类用户进行评估，以确定此类用户可能遇到的配置。此外，工业市场上的共享数据可以用来预测最多的可靠构件组合，然后，在这些环境中测试WebApp。

20.8 安全性测试

WebApp的安全性测试是一个复杂的主题，在有效地完成安全性测试之前，必须要对该主

¹⁵ 例如，可能使用单独的应用服务器和数据库服务器，两台机器之间通过网络连接进行通信。

¹⁶ 在每种可能的配置构件的组合中运行测试是非常耗费时间的。

题有充分的了解¹⁷。WebApp和其所处的客户端和服务器端环境对于外部的电脑黑客、对单位不满的员工、不诚实的竞争者以及其他想偷窃敏感信息、恶意修改内容、降低性能、破坏功能或者给个人、组织或业务制造麻烦的任何人都是一个有吸引力的攻击目标。

“对于管理业务和存储资产来说，Internet是一个危险的地方。电脑黑客、解密高手、窥探者、骗子……小偷、故意破坏者、病毒发布者和无赖程序承办商都可以自由行动。”

——Dorothy和Peter Denning

应该设计安全性测试用例去探查在某些方面所存在的弱点，比如客户端环境、当数据从客户端传到服务器并从服务器再传回客户端时所发生的网络通信以及服务器端环境。这些领域中的每一个都可能会受到攻击。发现可能会被怀有恶意的人利用的弱点，这是安全性测试人员的任务。

617



如果WebApp是业务关键的，用来维护敏感的数据，或者很可能成为电脑黑客的目标，则将安全性测试外包给擅长于此的供应商是一个好主意。

在客户端，弱点通常可以追溯到早已存在于浏览器、电子邮件程序或通信软件中的缺陷。Nguyen[NGU01]描述了一个典型的安全漏洞：

经常提到的缺陷之一是缓冲区溢出，这种缺陷使得恶意代码能够在客户端机器上运行。例如，向浏览器中输入的URL长度远远大于为URL分配的缓冲区容量，如果浏览器没有错误探测代码来确认输入的URL的长度，则会导致内存重写（缓冲区溢出）错误。经验丰富的电脑黑客能够聪明地利用这种缺陷，通过写一个带有可运行代码的很长的URL，使浏览器毁坏或改变安全性设置（从高到低），在最坏的情况下甚至会破坏用户数据。

对客户端的另一个可能的攻击是对放置在浏览器中的cookie的未被授权的访问。怀有恶意创建的站点能够获取包含在合法的cookie中的信息，并且用此信息危害用户的隐私，或者更糟糕的是为偷窃行为设置舞台。

客户和服务器之间通信的数据易受电子欺骗行为的攻击，当通信路径的一端被怀有恶意的实体暗中破坏时，电子欺骗行为就发生了。例如，用户会被恶意的网站所欺骗，它看起来好像是合法的WebApp服务器（与合法的WebApp服务器具有相同的外观），其目的是窃取密码、私有信息或信用数据。

在服务器端，攻击包括拒绝服务攻击和恶意脚本，这些恶意脚本可以被传到客户端，或者用来使服务器操作丧失能力。另外，服务器端数据库能够在没有授权的情况下被访问（数据窃取）。

为了防止这些（和很多其他）攻击，可以实现以下一种或多种安全机制[NGU01]：



安全性测试应该被设计为检验防火墙、鉴定、加密和授权。

- 防火墙（firewall）——是硬件和软件相结合的过滤机制，它检查每一个进来的信息包，确保信息包来自合法的信息源，阻止任何可疑的数据。
- 鉴定（authentication）——确认所有客户和服务器身份的一种验证机制，只有当两端都通过了检验才允许通信。
- 加密（encryption）——保护敏感数据的一种编码机制，通过对敏感数据进行某种方式的修改，使得怀有恶意的人不能读懂。通过使用数字证书（digital

¹⁷ 由Trivedi[TRE03]、McClure和他的同事[MCC03]、以及Garfinkel和Spafford[GAR02]编写的书籍提供了关于此主题的有用信息。

certificate)，加密得到了增强，因为数字证书允许客户对数据传输的目标地址进行检验。

- 授权 (authorization) ——一种过滤机制，只有那些具有合适的授权码（例如，用户ID和密码）的人，才允许访问客户或服务器环境。

安全性测试的目的是揭露这些安全机制中的漏洞，这些漏洞能够被怀有恶意的人所利用。在设计安全性测试时，需要深入了解每一种安全机制的内部工作情况，并充分理解所有网络技术。在很多情况下，应将安全性测试外包给擅长这些技术的公司。[618]

20.9 性能测试

你的WebApp要花好几分钟下载内容，而竞争者的站点下载相似的内容只需几秒钟，没有什么比这更让人灰心的了；你正设法登录到一个WebApp，却收到“服务器忙”的信息，建议你过一会再试，没有什么比这更让人烦恼的了；WebApp对某些情形能够立即做出反应，而对有些情形却似乎进入了一种无限等待状态，没有什么比这更让人感到惊慌的了。所有这些事件每天都在Web上发生，并且所有这些都是与性能相关的。

使用性能测试来发现性能问题，这些问题可能是由以下原因产生的：服务器端资源缺乏，不合适的网络带宽，不适当的数据库容量，不完善或不牢固的操作系统能力，设计糟糕的WebApp功能，以及可能导致客户-服务器性能下降的其他硬件或软件问题。性能测试的目的是双重的：(1) 了解系统如何对负载（即用户的数量、事务的数量或总的数据量）做出反应；(2) 收集度量数据，这些数据将促使修改设计，从而使性能得到改善。

20.9.1 性能测试的目标

设计性能测试来模拟现实世界的负载情形。随着同时访问WebApp的用户数量的增加，在线事务数量或数据量（下载或上载）也随之增加，性能测试将帮助回答下面的问题：



至少在被最终用户察觉到以前，WebApp性能的很多方面是难于测试的，包括网络负载、网络接口硬件的反复无常的行为及类似的问题。

- 服务器响应时间是否降到了值得注意的或不可接受的程度？
 - 在什么情况下（就用户、事务或数据负载来说），性能变得不可接受？
 - 哪些系统构件应对性能下降负责？
 - 在多种负载条件下，对用户的平均响应时间是多少？
 - 性能下降是否影响系统的安全性？
 - 当系统的负载增加时，WebApp的可靠性和精确性是否会受影响？
 - 当负载大于服务器容量的最大值时，会发生什么情况？
- 为了得到这些问题的答案，要进行两种不同的性能测试：
- 负载测试——在多种负载级别和多种组合下，对真实世界的负载进行测试。
 - 压力测试——将负载增加到强度极限，以此来确定WebApp环境能够处理的容量。

在下面的两节中将分别考虑每一种测试的策略。

20.9.2 负载测试

负载测试的目的是确定WebApp和其服务器环境如何响应不同的负载条件。当进行测试时，下面变量的排列定义了一组测试条件：

N ，并发用户的数量；

[619]

T , 每用户、每单位时间的在线事务数量;

D , 每次事务服务器处理的数据负载。



如果一个Web-App使用多个服务器提供巨大容量，则必须在多服务器环境中进行负载测试。

每一种情况下，在系统的正常操作范围内定义这些变量。当每一种测试条件运行时，收集下面的一种或多种测量数据：平均用户响应时间，下载标准数据单元的平均时间，或者处理一个事务的平均时间。Web工程团队对这些测量进行检查，以确定性能的急剧下降是否与 N 、 T 和 D 的特殊组合有关。

负载测试也可以用于为WebApp用户估计建议的连接速度。以下面的方式计算总的吞吐量 P :

$$P = N \times T \times D$$

作为一个例子，考虑一个大众体育新闻站点。在某一给定的时刻，2万个并发用户平均每两分钟提交一次请求（事务 T ）。每一次事务需要WebApp下载一篇平均长度为3KB的新文章，因此，可如下计算吞吐量：

$$\begin{aligned} P &= [20\,000 \times 0.5 \times 3\text{KB}] / 60 = 500\text{KB/s} \\ &= 4 \text{ Mb/s} \end{aligned}$$

因此，服务器的网络连接将不得不支持这种数据传输速度，应对其进行测试，确保它能够达到所需要的数据传输速度。

20.9.3 压力测试



压力测试的目的
是为了更好地理解当系统所承受的压力超过它的操作极限时，系统是如何失效的？

620

压力测试（第13章）是负载测试的继续，但是，在压力测试中，我们强迫变量 N 、 T 和 D 满足操作极限，然后超过操作极限。这些测试的目的是回答下面的问题：

- 系统“逐渐”降级吗？或者，当容量超出时，服务器停机吗？
- 服务器软件会给出“服务器不可用”的提示信息吗？更一般地说，用户知道他们不能访问服务器吗？
- 服务器队列请求增加资源吗？一旦容量要求减少，会释放队列所占用的资源吗？
- 当容量超出时，事务会丢失吗？
- 当容量超出时，数据完整性会受到影响吗？
- N 、 T 和 D 的哪些值迫使服务器环境失效？如何来证明失效了？自动通知会被发送到位于服务器站点的技术支持人员那里吗？
- 如果系统失效，需要多长时间才能回到在线状态？
- 当容量达到80%或90%时，某些WebApp功能（例如，计算密集的功能、数据流动能力）会被停止吗？

有时将压力测试的变种称为脉冲/回弹测试（spike/bounce testing）[SPL01]。在这种测试中，增加负载，达到最大容量，然后迅速回落到正常的操作条件，然后再增加。通过回弹系统负载，测试者能够确定服务器如何调度资源来满足非常高的需求，然后当一般条件再现时释放资源（以便为下一次脉冲做好准备）。

SOFTWARE TOOLS**WebApp测试工具分类介绍**

Lam[LAM01]在他的电子商务测试方面的论文中介绍了自动化工具的分类法，这可以直接受用于Web工程环境中的测试。对每一种类我们都提供了代表性工具¹⁸。

配置和内容管理工具：对WebApp的内容对象和功能构件进行版本管理和变更控制。

代表性工具：在www.daveeaton.com/scm/CMTools.html上有全面的列表。

数据库性能工具：测量数据库的性能，诸如执行所选择的数据库查询的时间。这些工具帮助进行数据库优化。

代表性工具：BMC Software (www.bmc.com)。

调试器：调试器是典型的程序设计工具，可发现和解决代码中的软件缺陷。它们是大多数现代应用系统开发环境的一部分。

代表性工具：Accelerated Technology (www.acceleratedtechnology.com)；IBM VisualAge Environment (www.ibm.com)；JDebugTool (www.debugtools.com)。

缺陷管理系统：记录缺陷，追踪它们的状态及解决方案。有些缺陷管理系统还包括报告工具，提供缺陷传播及缺陷解决率方面的管理信息。

代表性工具：EXCEL Quickbugs (www.excelsoftware.com)；McCabe TRUETrack (www.mccabe.com)；Rational ClearQuest (www.rational.com)。

网络监测工具：监视网络阻塞的级别。它们对识别网络瓶颈及测试前端系统和后端系统之间的连接很有用。

代表性工具：在www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html上有全面的列表。

回归测试工具：存储测试用例和测试数据，并且在连续的软件变更之后，可以重复使用这些测试用例。

代表性工具：Compuware QARun (www.compuware.com/products/qacenter/qarun)；Rational VisualTest (www.rational.com)；Seque Software (www.seque.com)。

站点监测工具：通常从用户的角度监测站点的性能。使用这些工具编辑统计表，诸如端到端的响应时间和吞吐量，并周期性地检查某个站点的有效性。

代表性工具：Keynote Systems (www.keynote.com)。

压力工具：在高水平的运行使用状态下，帮助开发者探测系统的行为，并发现系统的极限。

代表性工具：Mercury Interactive (www.merc-int.com)；Scapa Technologies (www.scapatech.com)。

系统资源监视器：系统资源监视器是大多数OS服务器和Web服务器软件的一部分，它们监视资源，如磁盘空间、CPU使用和内存。

代表性工具：Successful Hosting.com (www.successfulhosting.com)；Quest Software Foglight (www.quest.com)。

测试数据产生工具：辅助用户产生测试数据。

代表性工具：在www.softwareqa-test.com/qatweb1.html上有全面的列表。

¹⁸ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的一些样例。另外，工具的名字被注册为所提及公司的商标。

测试结果比较器：帮助将一个测试集合的结果与另一个测试集合的结果进行比较。用这些比较器来检查代码的改变没有对系统行为造成不利影响。

代表性工具：在www.aptest.com/resources.html上可找到有用工具的列表。

事务监视器：测量大量事务处理系统的性能。

代表性工具：QuotiumPro (www.quotium.com)；Software Research eValid (www.soft.com/eValid/index.html)。

网站安全性工具：帮助探测潜在的安全性问题。你可能经常安装安全性探查和监视工具，按你的计划安排运行这些工具。

代表性工具：在www.timberlinetechnologies.com/products/www.html中有全面的列表。

20.10 小结

WebApp测试的目标是对每一种WebApp质量维度进行检查，发现可能导致质量失效的错误或问题。应该对内容、功能、结构、可用性、导航性、性能、兼容性、互操作性、容量和安全性方面进行重点测试，在设计WebApp时进行的评审也属于测试的范围。

WebApp测试策略检查每一个质量维度，从考察内容、功能或导航“单元”开始。一旦单独的单元都已经被确认，重点就转到测试整个WebApp。为了完成这项工作，很多测试来自于用户的看法，并由用例所包含的信息所驱动。应编写Web工程测试计划，并确定测试步骤、工作产品（例如，测试用例），以及测试结果的评价机制。测试过程包括7个不同的测试类型。

内容测试（和评审）主要对内容的多种分类进行测试，目的是发现语义或语法上的错误，这些错误会影响内容的精确性，或对展示给最终用户的方式有影响。界面测试检查用户与WebApp之间通信的交互机制，并对界面的美学方面进行确认，目的是发现由于实现糟糕的交互机制、遗漏、不一致或界面语义不明确所导致的错误。

导航测试使用从分析活动中得出的用例，在测试用例的设计中，测试用例对照导航设计检查每一个使用场景。对导航机制进行测试，确保识别并改正妨碍用例完成的任何错误。构件测试检查WebApp中的内容和功能单元。每一个网页封装内容、导航链接和处理元素，它们形成了WebApp体系结构中的“单元”，因此，必须对这些单元进行测试。

配置测试试图发现针对特殊的客户或服务器环境的错误和（或）兼容性问题，然后进行测试，发现与每一种可能配置有关的错误。安全性测试包括一系列测试，探测WebApp及其环境中存在的弱点，目的是发现安全漏洞。性能测试包括一系列测试，当对服务器端资源容量的要求增加时，评估WebApp的响应时间及可靠性。

参考文献

- [BRO01] Brown, B., *Oracle9i Web Development*, McGraw-Hill, 2nd ed., 2001.
- [CAC02] Cachero, C., et al., "Conceptual Navigation Analysis: A Device and Platform Independent Navigation Specification," *Proc. 2nd Intl. Workshop on Web-Oriented Technology*, June 2002, download from www.dsic.upv.es/~west/iwwost02/papers/cachero.pdf.
- [CON03] Constantine, L., and L. Lockwood, *Software for Use*, Addison-Wesley, 1999; see also <http://www.foruse.com/>.
- [GAR02] Garfinkel, S., and G. Spafford, *Web Security, Privacy and Commerce*, O'Reilly & Associates, 2002.
- [HOW97] Hower, Rick, "Beyond Broken Links," *Internet Systems*, 1997 available at

- <http://www.dbmsmag.com/9707i03.html>
- [LAM01] Lam, W., "Testing E-Commerce Systems: A Practical Guide," *IEEE IT Pro*, March/April 2001, pp. 19–28.
- [MCC03] McClure, S., S. Shah, and S. Shah, *Web Hacking: Attacks and Defense*, Addison-Wesley, 2003.
- [MIL00] Miller, E., "WebSite Testing," 2000, available at <http://www.soft.com/eValid/Technology/White.Papers/website.testing.html>.
- [NGU00] Nguyen, H., "Testing Web-Based Applications," *Software Testing and Quality Engineering*, May/June, 2000, available at <http://www.stqemagazine.com>.
- [NGU01] Nguyen, H., *Testing Applications on the Web*, Wiley, 2001.
- [SCE02] Sceppa, D., *Microsoft ADO.NET*, Microsoft Press, 2002.
- [SPL01] Splaine, S., and S. Jaskiel, *The Web Testing Handbook*, STQE Publishing, 2001.
- [TRE03] Trivedi, R., *Professional Web Services Security*, Wrox Press, 2003.
- [WAL03] Wallace, D., I. Raggett, and J. Aufgang, *Extreme Programming for Web Projects*, Addison-Wesley, 2003.

623

习题与思考题

- 20.1 安全性测试的目标是什么？谁来进行这种测试活动？
- 20.2 用自己的话，讨论在Web工程环境下测试的目标。
- 20.3 兼容性是一项重要的质量维度，必须对哪些方面进行测试，才能确保一个WebApp具有兼容性？
- 20.4 哪些错误更加严重，是客户端错误还是服务器端错误？为什么？
- 20.5 是否存在应该完全忽视WebApp测试的任何情况？
- 20.6 开发正式书写的测试计划是否总是必要的？为什么？
- 20.7 是否可以公平地说，总的WebApp测试策略开始于用户可见的元素，然后移向技术元素？这种策略存在例外吗？
- 20.8 假设你正在开发一个满足老年人需要的在线药房（CornerPharmacy.com）。药房提供了典型的功能，但也为每一位客户维护一个数据库，使得它可以提供药品信息和潜在的药品互作用警告。讨论针对此WebApp的任何特殊的可用性测试。
- 20.9 描述与WebApp的数据库测试有关的步骤。数据库测试是否对客户端和服务器端活动有影响？
- 20.10 测试WebApp可能在服务器端遇到的每一种配置，这样做可能吗？在客户端呢？如果不可能，Web工程师如何选择有意义的配置测试集合？
- 20.11 WebApp的哪些元素可以进行“单元测试”？哪些类型的测试只能在WebApp元素被集成之后进行？
- 20.12 CornerPharmacy.com（习题20.8）已经取得了广泛成功，在运行的头两个月，用户数量剧增。对于固定的服务器端资源集合，画一张图，将可能的响应时间描述为用户数量的函数。对图进行标注，指出“响应曲线”的兴趣点。
- 20.13 导航语法测试和导航语义测试之间的区别是什么？
- 20.14 与界面机制相关的测试和界面语义测试之间的区别是什么？
- 20.15 内容测试是传统意义上的真正测试吗？给出解释。
- 20.16 假设你已经为CornerPharmacy.com（习题20.8）实现了一个药品互作用检查功能。讨论不得不进行的构件级测试的类型，以确保这项功能工作正常。（注意：实现这项功

能将不得不使用数据库。)

- 20.17 为适应其成功, CornerPharmacy.com (习题20.8) 已经实现了一个特殊的服务, 单独处理重新填写处方。平均情况下, 1000个并发用户每两分钟提交一次重填申请, WebApp 下载500B的数据块来响应。此服务器需要具有的吞吐量是多少Mb/s?
- 20.18 负载测试和压力测试之间的区别是什么?

推荐读物与阅读信息

WebApp测试方面的文献仍在发展之中。这方面最新出版的书籍中, 由Ash (《The Web Testing Companion》, Wiley, 2003)、Dustin和他的同事 (《Quality Web Systems》, Addison-Wesley, 2002)、Nguyen[NGU01]以及Splaine和Jaskiel[SPL01]撰写的书论述最全面。Mosley (《Client-Server Software Testing on the Desktop and the Web》, Prentice-Hall, 1999) 论述了客户端和服务器端的测试问题。

Stottlemeyer (《Automated Web Testing Toolkit》, Wiley, 2001) 介绍了WebApp测试策略和方法方面的有用信息, 并对自动化测试工具进行了有价值的讨论。Graham和他的同事 (《Software Test Automation》, Addison-Wesley, 1999) 介绍了自动化工具方面的其他资料。

Nguyen和他的同事 (《Testing Applications for the Web》, second edition, Wiley, 2003) 已对[NGU01]进行了重要更新, 并为测试移动应用提供了独特的指导。虽然微软出版的一本书 (《Performance Testing Microsoft .NET Web Applications》, Microsoft Press, 2002) 主要集中在它的.NET环境, 它对性能测试的论述值得对该主题感兴趣的人阅读。

Splaine (《Testing Web Security》, Wiley, 2002)、Klevinsky和他的同事 (《Hack I.T.: Security through Penetration Testing》, Addison-Wesley, 2002)、Chirillo (《Hack Attacks Revealed》, second edition, Wiley, 2003) 以及Skoudis (《Counter Hack》, Prentice-Hall, 2001) 为必须设计安全性测试的人提供了非常有用的信息。

在Internet上可以获得多种测试Web工程的信息资源。WWW引用的最新列表可以在SEPA 站点<http://www.mhhe.com/pressman>找到。



第四部分

管理软件项目

在

本书的这一部分将考虑计划、组织和监控软件项目所需要的管理技术。在后面几章中会涉及下列问题：

- 在软件项目进行期间，为什么要对人员、过程和问题进行管理？
- 如何使用软件度量来管理软件项目和软件过程？
- 如何估算软件项目的工作量、成本和项目的工期？
- 采用什么技术来正式评估影响项目成功的风险？
- 软件项目经理如何选择软件工程工作任务集？
- 如何编制项目进度计划？
- 什么是质量管理？
- 为什么正式技术评审那么重要？
- 在计算机软件开发过程中以及交付给客户使用后，如何进行变更管理？

回答了这些问题后就为管理软件项目做好了较充分的准备，便可以在某种程度上按时交付高质量的产品。

第21章 项目管理

要点浏览

概念：虽然很多人在悲观的时候接受Dilbert的“管理”观点，但是在构造基于计算机的系统和产品时管理仍然是一项非常必要的活动。项目管理涉及对人员、过程和在软件从初始的概念演化为可运行的实现的过程中发生的事件的计划和监控。

人员：在软件项目中，每个人或多或少都做着“管理”工作。但是，管理活动的范围各不相同。软件工程师管理他的日常活动、计划和监控技术任务。项目经理计划和监控软件工程师团队的工作。高级管理者协调业务和软件专业人员之间的关系。

重要性：构造计算机软件是一项复杂的任务，尤其是当它涉及很多人员长期共同工作的时候。这就是为什么软件项目需要管理的原因。

步骤：理解4P——人员 (People)、产品

(Product)、过程 (Process) 和项目 (Project)。必须将人员组织起来以有效地完成软件工作。必须和客户及其他共利益者很好地沟通，以便了解产品的范围和需求；必须选择适合于人员和产品的过程；必须估算完成工作任务的工作量和工作时间，从而制定项目计划，包括：定义工作产品、建立质量检查点以及确定一些机制以监控计划所规定的工作。

工作产品：在管理活动开始时，必须首先制定项目计划。该计划定义将要进行的过程和任务，安排工作人员，确定评估风险、控制变更和评价质量的机制。

质量保证措施：在按时并在预算内交付高质量的产品之前，你不可能完全肯定项目计划是正确的。不过，作为项目经理，鼓励软件人员协同工作形成一支高效的团队，并将他们的注意力集中到客户需求和产品质量上，这肯定是正确的。

关键概念
敏捷团队
协调
关键实践
人员
问题分解
过程
项目
软件范围
软件团队
共利益者
团队负责人
W³HH原则

Meiler Page-Jones [PAG85]在其关于软件项目管理的论著的序言中给出了以下一段陈述，这引起了许多软件工程顾问的共鸣：

我拜访了很多商业公司——好的和不好的，我又观察了很多数据处理管理者的业绩——好的和不好的。我常常恐惧地看到，这些管理者徒劳地与恶梦般的项目斗争着，在根本不可能完成的最后期限的压力下苦苦挣扎，或者是在交付了用户极为不满意的系统之后，又继续花费大量的时间去维护它。

Page-Jones所描述的正是源于一系列管理和技术问题而产生的症状。不过，如果在事后再剖析一下每个项目，很有可能发现一个共同的问题：项目管理太弱。

在本章和以后的六章中，我们将探讨进行有效的软件项目管理的关键概念。本章考虑基本的软件项目管理的概念和原则。第22章将阐述过程和项目度量，这是做出有效管理决策

的基础。第23章和第24章将讨论用于估算成本、编制实际的进度表并制定有效项目计划的技术。第25章阐述了进行有效的风险监测、风险缓解和风险管理的管理活动。最后，第26章和第27章将探讨在项目开发过程中保证质量的技术以及在应用的整个生命周期中控制变更的技术。

628

21.1 管理涉及的范围

有效的软件项目管理集中于四个P上，即人员、产品、过程和项目。它们的顺序不是任意的，任何管理者如果忘记了软件工程工作是人的智力密集的劳动，他就永远不可能在项目管理上取得成功；任何管理者如果在项目开发早期没有鼓励共利益者之间的广泛交流，他就冒着为错误的问题构造了“良好的”解决方案的风险；对过程不在意的管理者可能冒着把有效的技术方法和工具插入到真空中的风险；没有建立可靠的项目计划就开始工作的管理者将危及产品的成功。

21.1.1 人员

从20世纪60年代起就开始讨论，要培养有创造力的、高技术水平的软件人员（如[COU80]、[WIT94]、[DEM98]）。事实上，“人的因素”非常重要，美国卡内基·梅隆大学的软件工程研究所（SEI）专门开发了一个人员管理能力成熟度模型（PM-CMM），旨在“通过吸引、培养、激励、部署和聘用那些改进软件组织软件开发能力所需要的人才，提高软件组织承担日益复杂的应用问题的能力”([CUR94])。

人员管理成熟度模型中针对软件人员定义了以下的关键实践区域：招募、选择、业绩管理、培训、报酬、个人事业发展、组织和工作设计以及团队精神或企业文化培养。在人员管理上达到较高成熟度的组织，更有可能完成有效的软件工程实践。

PM-CMM与软件能力成熟度集成模型（CMMI）相伴而生（第2章），两者均可指导一个组织创建一个成熟的软件过程。与软件项目的人员管理及人员结构相关的问题将在本章后面的内容中详细论述。

629

21.1.2 产品



在这里的上下文中，术语“产品”是指包括任何应他人需要而开发的软件。其中，不仅包括软件产品，也包括基于计算机的系统、嵌入式软件、Web应用软件以及问题求解软件（如针对工程/科学问题求解的程序）。

在制定项目计划之前，应该首先确定产品的目标和范围，考虑可选的解决方案，识别技术和管理上的限制。如果没有这些信息，就不可能进行合理的（准确的）成本估算，也不可能进行有效的风险评估和适当的项目任务划分，更不可能制定可管理的项目进度计划来给出意义明确的项目进展标志。

软件开发者和客户必须一同定义产品的目标和范围。在很多情况下，这项活动是作为系统工程或业务过程工程的一部分开始的（第6章），并一直持续到作为软件需求工程的第一步（第7章）。确定产品的目标只是标识出产品的总体目标（从客户的角度），而不用考虑这些目标如何实现。而确定产品的范围，要标识出产品的主要数据、功能和行为特性，而且更为重要的是，应以量化的方式界定这些特性。

了解了产品的目标和范围之后，就要开始考虑备选的解决方案了。虽然这一步并不讨论细节，但可以使管理者和参与开发的人员根据给定的约束条

件选择“最好”的方案，其中，约束条件包括产品交付的期限、预算的限制、可用的人员、技术接口以及其他各种因素。

21.1.3 过程



坚持敏捷过程方法（第4章）的人指出敏捷过程比其他过程更简单，这也许是正确的。但是敏捷过程仍然有一个过程，敏捷软件工程依然需要规则。

软件过程（第2, 3, 4章）提供了一个框架，在该框架下可以制定软件开发的综合计划。一小部分框架活动适用于所有软件项目，不用考虑其规模和复杂性。多种不同的任务集合——每一种集合都由任务、里程碑、工作产品以及质量保证点组成——使得框架活动适合于不同软件项目的特性和项目团队的需求。最后是普适性活动——如软件质量保证、软件配置管理和测量，这些活动覆盖了过程模型。普适性活动独立于任何一个框架活动，且贯穿于整个过程之中。

21.1.4 项目

我们实施有计划的、可控制的软件项目的主要理由是：这是我们管理复杂事物的唯一方法。然而，我们仍需要努力。1998年，产业数据表明26%的软件项目彻底失败，46%的项目成本和进度超出预定[REE99]。虽然软件项目的成功率已有所提高，但项目的失败率仍然高于它的应有值¹。

“项目如同旅行。有些项目是简单的、常规性的，就像在白天开车去购物；但是值得做的大多数项目就像在夜晚驾驶一辆卡车离开大路驶入大山中一样。”

——Cem Kaner、James Bach和Bret Pettichord

为了避免项目失败，软件项目经理和开发产品的软件工程师必须留意一些常见的警告信号，了解实施成功的项目管理的关键因素，还要确定计划和监控项目的一目了然的方法。这些问题将在21.5节及以后的章节中讨论。

21.2 人员

在IEEE[CUR88]发表的一项研究中提到，当向三个大型技术公司中主管工程的三位副总裁问及一个成功的软件项目中最重要的因素是什么时，他们的回答如下：

第一位：我想如果必须在我们的环境中挑出一项最重要的因素，我要说它不是我们所用的工具，而是人员。

第二位：一个项目成功的最重要的因素是有聪明的人……我想不出其他的因素……一个项目最重要的事情是选择人员……软件开发组织的成功与其招募优秀人才的能力密切相关。

第三位：我在管理上唯一的准则是确保拥有优秀的人员——真正优秀的人员，同

¹ 看到这些统计数据，人们自然会问，若是这样计算机的影响又为何持续呈指数增长。我认为，部分原因是：相当数量的“失败”项目在刚开始时就是构想拙劣的，客户很快就失去了兴趣（因为他们所需要的并不像他们最初想像的那样重要），进而取消了这些项目。

时我也培养优秀的人员，我提供培养优秀人员的良好环境。

确实，这是对软件工程过程中人员重要性的强有力的证明。不过，我们所有的人，从高级工程副总裁到最低层的开发人员，常常认为人员是不成问题的。虽然管理者常常表态说（就像上面所说的那样）人员是最重要的，但有时他们言行并不一致。

本节我们将要分析参与软件过程的人员，并且研究组织人员的方式，以实现有效的软件工程。

21.2.1 共利益者

参与软件过程（及每一个软件项目）的共利益者可以分为以下五类：

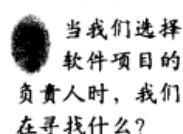
1. 高级管理者——负责定义业务问题，这些问题往往对项目产生很大影响。
2. 项目（技术）管理者——必须计划、激励、组织和控制软件开发人员。
3. 开发人员——拥有开发产品或应用软件所需技能的人员。
4. 客户——详细描述待开发软件需求的人员以及关心项目成败的其他共利益者。
5. 最终用户——一旦软件发布成为产品，最终用户就是直接与软件进行交互的人。

[631]

每一个软件项目都有上述人员的参与²。为了获得高效率，项目团队必须以能够最大限度地发挥每个人的技术和能力的方式进行组织，这是团队负责人的任务。

21.2.2 团队负责人

项目管理是人员密集型的活动，因此，胜任开发的人却常常有可能是拙劣的团队负责人，他们完全不具备管理人员的技能。正如Edgemon所说：“很不幸但却经常是这样，人们似乎碰巧落在项目经理的位置上，也就意外地成为了项目经理” [EDG95]。



在一本关于领导能力的优秀论著中，Jerry Weinberg[WEI86]提出了领导能力的MOI模型（MOI—Motivation（激励）、Organization（组织）、Idea or Innovation（思想或创新））。

激励：（通过“推”或“拉”）鼓励技术人员发挥其最大才能的一种能力。

组织：形成能够将最初概念转换成最终产品的现有过程（或创造新的过程）的能力。

思想或创新：即使必须在特定软件产品或应用的约束下工作，也能鼓励人们去创造并让人感到有创造性的一种能力。

Weinberg提出，成功的项目负责人应采用一种解决问题的管理风格。也就是说，软件项目经理应该注重理解要解决的问题、把握住涌现的各种意见、同时让项目团队的每一个人知道（通过言语，更重要的是通过行动）质量很重要，不能妥协。

“用最简单的话来说，负责人是这样一种人：他知道他想去哪里，并站起来就朝那里走。”

—John Erskine

关于一个具有实战能力的项目经理应该具有什么特点，另一种观点[EDG95]则强调了以下四种关键品质：

解决问题。具有实战能力的软件项目经理能够准确地诊断出最为密切相关的技术问题和

² 当开发Web应用软件时（本书第三部分），在内容创作方面需要其他非技术人员的参与。

632 组织问题：能够系统地制定解决方案，适当地激励其他开发人员来实现该方案；能把在过去项目中学到的经验应用到新环境中；如果最初的解决方案没有结果，能够灵活地改变方向。

管理者的特性。优秀的项目经理必须能够掌管整个项目。必要的时候要有信心进行项目控制，同时还要允许优秀的技术人员能够按照他们的本意行事。

成就。为了优化项目团队的生产效率，项目经理必须奖励那些工作积极主动并且做出成绩的人，并通过自己的行为表明出现可控风险并不会受到惩罚。

影响和队伍建设。具有实战能力的项目经理必须能够“理解”人。他必须能理解语言和非语言的信号，并对发出这些信号的人的要求做出反应。项目经理必须能在高压力的环境下保持良好的控制能力。

21.2.3 软件团队

几乎可以说有多少开发软件的组织，就有多少种软件开发人员组织结构。不管怎么说，组织结构不能轻易改变。至于组织改变所产生的实际的和行政上的影响，并不在软件项目经理的责任范围内。但是，软件项目中所直接涉及的人员的组织，则是项目经理的职责。

“并非每个小组都是团队，并非每个团队都是有效的。”

——Glenn Parker

“最好的”团队结构取决于组织的管理风格、团队里的人员数目与技能水平，以及问题的总体难易程度。Mantei[MAN81]提出了规划软件工程团队结构时应该考虑的七个项目因素：

- 待解决问题的难度。
- 开发程序的规模，以代码行或者功能点来度量（第22章）。
- 团队成员需要共同工作的时间（团队生存期）。
- 能够对问题做模块化划分的程度。
- 待开发系统的质量要求和可靠性要求。
- 交付日期的严格程度。
- 项目所需要的友好交流的程度。

633

“如果你要做得更好，那就竞争。如果你要做得非常好，那就要合作。” ——作者不详

Constantine[CON93]提出了软件工程团队的四种“组织范型”：

1. 封闭式范型。按照传统的权利层次来组织团队。当开发与过去已经做过的产品相似的软件时，这种团队十分有效。但在这种封闭式范型下难以进行创新性的工作。

2. 随机式范型。松散地组织团队，团队工作依赖于团队成员个人的主动性。当需要创新或技术上的突破时，按照这种随机式范型组织的团队很有优势。但当需要“有次序地执行”才能完成工作时，这种团队就会陷入困境。

3. 开放式范型。试图以一种既具有封闭式范型的控制性，又包含随机式范型的创新性的方式来组织团队。工作是大家相互协作完成的。良好的沟通和根据团队整体的意见做出决策是开放式范型的特征。开放式范型的团队结构特别适合于解决复杂的问题，但可能不像其他类型团队那么有效率。

选择软件团
队的结构
时，应该考虑哪
些因素？

定义软件团
队的结构
时，我们有哪些
选择？

4. 同步式范型。依赖于问题的自然划分，组织团队成员各自解决问题的一部分，他们之间没有什么主动的交流。

“与人打交道是困难的，但不是不可能的。”

——Peter Drucker

从历史的角度看，最早的软件团队组织是封闭式范型结构，最初称为主程序员团队。这种结构首先由Harlan Mills提出，并由Baker[BAK72]描述出来。团队的核心成员包括：一个高级工程师（“主程序员”），负责计划、协调和评审团队的所有技术活动；技术人员（一般是2~5人），进行分析和开发活动；一个后备工程师，支持高级工程师的活动，并可以在项目进行过程中以最小的代价接替高级工程师的工作。

主程序员可以有多人配合他的工作，包括一个或多个专家（如电信专家，数据库设计人员）、支持人员（如技术文档写作人员，行政人员）和软件资料员。

Constantine[CON93]提出的随机式范型是主程序员团队结构的一个变种，主张建立具有独立创新性的团队，其工作方式可恰当地称为创新的无政府状态。尽管自由的软件工作方式是有吸引力的，但在绩效良好的团队中必须将创新能力作为软件工程组织的中心目标。为了建成一支绩效良好的团队：

- 团队成员必须相互信任。
- 团队成员的技能分布必须适合于要解决的问题。
- 如果要保持团队的凝聚力，必须将坚持个人己见的人员排除于团队之外。

无论什么类型的团队，每一个项目经理的目标都是帮助建立一支有凝聚力的团队。DeMarco和Lister[DEM98]在其论著《Peopleware》中，讨论了这个问题：

在商业界，我们往往随便使用团队这个词。任何被分配在一起工作的一组人都可以称为“团队”。但很多这样的小组看起来并不像团队，它们没有统一的对成功的定义，没有任何鲜明的团队精神。它们所缺少的是一种很珍贵的东西，我们称之为凝聚力。

一个有凝聚力的团队是一组团结紧密的人，他们的整体力量大于个体力量的总和……

一旦团队开始具有凝聚力，成功的可能性就大大提高。这个团队可以变得不可阻挡，成为成功的象征……他们不需要按照传统的方式进行管理，也不需要去激励。他们已经有了动力。

何谓有凝聚力的团队？

为什么有些团队没有凝聚力？

DeMarco和Lister认为，同一般的团队相比，有凝聚力的团队成员具有更高的生产率和更大的动力。他们拥有共同的目标和共同的文化，而且在很多情况下，“精英意识”使得他们独一无二。

但是，并非所有的团队都具有凝聚力。事实上，很多团队都受害于Jackman[JAC98]称之为“团队毒性”的东西。她定义了5个“培育潜在含毒团队环境”的因素：(1) 狂乱的工作氛围；(2) 引起团队成员间产生摩擦的重大挫折；(3)“碎片式的或协调很差”的软件过程；(4) 在软件团队中没有清晰的角色定义；(5)“接连不断地重蹈覆辙”。

为了避免狂乱的工作环境，项目经理应该确保团队可以获取完成工作所需的所有信息；而且，主要目标一旦确定下来，除非绝对必要，否则不应该修改。如果给予软件团队尽可能

634



敏捷团队是自组织的，拥有制定计划和做技术决定的自主权。

很多敏捷过程模型（如Scrum）给予敏捷团队相当大的自主权进行项目管理，可以因工作需要做出技术决定。将计划制定工作压缩到最低程度，并且允许团队选择自己适用的手段（如：过程、方法和工具），只受业务需求和组织标准的限制。在项目进行过程中，自组织团队关注的是在特定的时间点使项目获益最大的个人能力。为了做到这一点，敏捷团队可以召开简短的日常团队会对当天必须完成的工作进行调整——以使其同步进行。

基于在团队会中取得的信息，团队能使他们所采用的手段不断适应持续增加的工作。当每一天过去的时候，连续的自组织和协作使团队朝着软件逐步增长的完工的方向发展。

21.2.5 协调和通信问题

使软件项目陷入困境的原因很多。许多开发项目规模很大，导致复杂性高、混乱、难以协调团队成员间的关系。不确定性是经常存在的，它会引起困扰项目团队的一连串的变更。互操作性已经成为许多系统的关键特性。新的软件必须与已有的软件通信，并遵从系统或产品所施加的预定义约束。

现代软件的这些特征——规模、不确定性和互操作性——确实都是存在的。为了有效地处理这些问题，软件工程团队必须建立切实可行的方法来协调工作人员之间的关系。为了做到这一点，需要建立团队成员之间以及多个团队之间的正式的和非正式的交流机制。正式的交流机制是通过“文字、各级会议及其他相对而言非交互的和非个人的交流渠道”[KRA95]来实现的。非正式的交流机制则更加个人化。软件团队的成员在遇到特别情况时交流意见，出现问题时请求帮助，而且在日常工作中彼此之间互相影响。

637

SAFEHOME

团队结构

[场景] SafeHome软件项目启动之前，Doug Miller的办公室。

[人物] Doug Miller，SafeHome软件工程团队经理；Vinod Raman、Jamie Lazar及其他团队成员。

[对话]

Doug：你们看过市场营销部准备的有关SafeHome的基本信息了吗？

Vinod（一边看着队友，一边点头）：是的，但我们有很多问题。

Doug：过一会儿再讨论这些问题。我们先来讨论一下应该如何组织一个团队，哪些人应该负责……

Jamie： Doug，我对敏捷方法非常感兴趣，我想我们应该是一个自组织的团队。

Vinod：我同意。给定一个我们都能胜任的严格的期限和某些不确定性（笑），看起来是一种正确的方式。

Doug：我赞同，但是你们知道如何操作吗？

Jamie（边笑边说，就像在背诵什么）：我们做出战术决定，确定由谁做、做什么、什么时间做。但按时交付产品是我们的责任。

Vinod：还有质量。

Doug：很正确，但是记住还有约束。市场部决定要生产的软件的增量，当然这要征



敏捷团队是自组织的，拥有制定计划和做技术决定的自主权。

很多敏捷过程模型（如Scrum）给予敏捷团队相当大的自主权进行项目管理，可以因工作需要做出技术决定。将计划制定工作压缩到最低程度，并且允许团队选择自己适用的手段（如：过程、方法和工具），只受业务需求和组织标准的限制。在项目进行过程中，自组织团队关注的是在特定的时间点使项目获益最大的个人能力。为了做到这一点，敏捷团队可以召开简短的日常团队会对当天必须完成的工作进行调整——以使其同步进行。

基于在团队会中取得的信息，团队能使他们所采用的手段不断适应持续增加的工作。当每一天过去的时候，连续的自组织和协作使团队朝着软件逐步增长的完工的方向发展。

21.2.5 协调和通信问题

使软件项目陷入困境的原因很多。许多开发项目规模很大，导致复杂性高、混乱、难以协调团队成员间的关系。不确定性是经常存在的，它会引起困扰项目团队的一连串的变更。互操作性已经成为许多系统的关键特性。新的软件必须与已有的软件通信，并遵从系统或产品所施加的预定义约束。

现代软件的这些特征——规模、不确定性和互操作性——确实都是存在的。为了有效地处理这些问题，软件工程团队必须建立切实可行的方法来协调工作人员之间的关系。为了做到这一点，需要建立团队成员之间以及多个团队之间的正式的和非正式的交流机制。正式的交流机制是通过“文字、各级会议及其他相对而言非交互的和非个人的交流渠道”[KRA95]来实现的。非正式的交流机制则更加个人化。软件团队的成员在遇到特别情况时交流意见，出现问题时请求帮助，而且在日常工作中彼此之间互相影响。

637

SAFEHOME

团队结构

[场景] SafeHome软件项目启动之前，Doug Miller的办公室。

[人物] Doug Miller，SafeHome软件工程团队经理；Vinod Raman、Jamie Lazar及其他团队成员。

[对话]

Doug：你们看过市场营销部准备的有关SafeHome的基本信息了吗？

Vinod（一边看着队友，一边点头）：是的，但我们有很多问题。

Doug：过一会儿再讨论这些问题。我们先来讨论一下应该如何组织一个团队，哪些人应该负责……

Jamie： Doug，我对敏捷方法非常感兴趣，我想我们应该是一个自组织的团队。

Vinod：我同意。给定一个我们都能胜任的严格的期限和某些不确定性（笑），看起来是一种正确的方式。

Doug：我赞同，但是你们知道如何操作吗？

Jamie（边笑边说，就像在背诵什么）：我们做出战术决定，确定由谁做、做什么、什么时间做。但按时交付产品是我们的责任。

Vinod：还有质量。

Doug：很正确，但是记住还有约束。市场部决定要生产的软件的增量，当然这要征

求我们的意见。

Jamie: 还有?

Doug: 还有, 要使用UML作为我们的建模方法。

Vinod: 但要保持无关的文档减到最少。

Doug: 那谁和我联络?

Jamie: 我们确定Vinod作为技术负责人, 因为他的经验最丰富。因此, Vinod是你的联络人, 但你应该自由地与每个人交流。

Doug: 别担心, 我会的。

21.3 产品

从软件工程项目一开始, 软件项目经理就面临着进退两难的局面。需要定量地估算成本和有组织地计划项目的进展, 但却没有可靠的信息可以使用。虽然对软件需求的详细分析可以提供必要的估算信息, 但需求分析常常需要数周甚至数月的时间才能完成。更糟糕的是, 需求可能是不固定的, 随着项目的进展经常会发生变化。不过, 无论如何计划总是“眼前”就需要的!

因此, 从软件工程项目一开始, 我们就要研究应该开发哪些产品以及要解决哪些问题。至少, 我们要建立和界定项目的范围。

21.3.1 软件范围

软件项目管理的第一项活动是确定软件范围。软件范围是通过回答下列问题来定义的:



638

如果你不能把握待开发软件的某个特征, 就将该特征作为一个项目风险列出(第25章)。

项目环境。要开发的软件如何适应于大型的系统、产品或业务环境, 该环境下要施加什么约束?

信息目标。软件要产生哪些客户可见的数据对象(第8章)来作为输出?需要什么数据对象作为输入?

功能和性能。软件要执行什么功能才能将输入数据转换成输出数据?软件需要满足什么特殊的性能要求吗?

软件项目范围在管理层和技术层都必须是无歧义的和可理解的。对软件范围的描述必须是界定的。也就是说, 要明确给出定量的数据(如并发用户数目、邮件列表的大小、允许的最大响应时间);说明约束和(或)限制(如产品的成本要求会限制内存的大小), 并且描述其他的缓解因素(如期望的算法能被很好地理解, 并且可以采用C++)。

21.3.2 问题分解



为了制定合理的项目计划, 必须对问题进行分解。可以使用一系列功能、用例、敏捷工件或者用户经历来完成。

问题分解, 有时称为问题划分或问题细化, 它是软件需求分析(第7章和第8章)的核心活动。在确定软件范围的活动中, 并不试图去完全分解问题, 只是分解其中的两个主要方面:(1)必须交付的功能;(2)所使用的过程。

在面对复杂的问题时, 人们常常采用分而治之的策略。简单地说, 就是将一个复杂的问题划分成若干较易处理的小问题。这是项目计划开始时所采用的策略。在开始估算(第23章)前, 必须对软件范围内所描述的软件功能

进行评估和精化，以提供更多的细节。因为成本和进度估算都是面向功能的，所以对功能进行某种程度的分解是很有益处的。

例如，考虑这样一个项目，要开发一个新的字处理产品。该产品的独特功能包括：连续的语言和键盘输入，高级的“自动复制编辑”功能，页面布局功能，自动建立索引和目录的功能等等。项目经理必须首先写出软件范围陈述来界定这些功能（以及其他通用的功能，如编辑、文件管理、文档生成等等）。例如，连续的语言输入功能是否需要用户对产品进行“训练”？复制编辑功能提供了哪些能力？页面布局功能要高级到什么程度？

随着软件范围陈述的进展，自然产生了第一级划分。项目团队研究了市场部与潜在客户的交谈资料，发现“自动复制编辑”还应该具有下列功能：(1) 拼写检查；(2) 语句文法检查；(3) 大型文档的引用检查（例如，对参考书的引用是否能在参考书目列表中找到）；(4) 大型文档中章节引用的确认。[639]

其中每一项都是软件要实现的子功能。同时，如果分解可以使制定计划更容易，则每一项又可以进一步细化。

21.4 过程



可以使用项目进度安排自动生成工具来创建“任务网络”（第24章）。任务网络包括估算资源需求、估计开始/结束日期以及其他有关日期。这个包含资源的网络可以用于项目跟踪和控制。

刻画软件过程的框架活动（第2章）适用于所有软件项目。问题是项目团队要选择一个适合于待开发软件的过程模型。

项目经理必须决定哪一个过程模型最适合于：(1) 需要该产品的客户和从事开发工作的人员；(2) 产品本身的特性；(3) 软件项目团队工作的项目环境。当选择一个过程模型以后，项目团队可以基于一系列过程框架活动来制定一个初步的项目计划。一旦确定了初步计划，过程分解就开始了。也就是说，必须制定一个完整的计划，来反映框架活动中所需要的工作任务。在下面的几小节中，我们将对这些活动进行简要的讨论，更详细的信息将在第24章中给出。

21.4.1 合并产品和过程

项目计划开始于产品和过程的合并。软件项目团队要完成的每一项功能都必须通过针对软件组织而定义的一系列框架活动来完成。假定软件组织采用了下列框架活动（第2章）：沟通、策划、建模、构建和部署。



过程框架建立了项目计划的纲要，并通过分配适合项目的一系列任务对其进行调整。

完成任何一项功能的项目团队成员都要将各个框架活动应用于该功能的实现上。实质上，产生了一个类似于图21-1所示的矩阵。每个主要的产品功能（图中列出了前面提到的字处理软件的各项功能）显示在第一列，框架活动显示在第一行。软件工程工作任务（针对每个框架活动）列在紧接下来的行中⁴。项目经理（和其他团队成员）的工作是估算每一个矩阵单元的资源需求，与每个单元相关的任务的开始和结束日期，以及每项任务所产生的工作产品。这些问题将在第24章中探讨。[640]

⁴ 应该注意：工作任务必须适合于项目的特殊需要。



图21-1 合并产品和过程

21.4.2 过程分解

软件团队在选择最适合项目的软件过程模型时，应该具有很大的灵活性。一旦选定了过程模型，项目团队可以根据需要灵活地确定过程模型中应包含的软件工程任务。较小的项目如果与以前已开发过的项目相似，可以采用线性顺序模型。如果时间要求很紧，而且问题能够被很好地划分，RAD模型（见3.3.2节）可能是正确的选择。如果时间要求太紧，不可能完成所有功能时，增量模型可能是最合适的选择。同样地，如果项目具有其他特性（如需求的不确定性、突破性的新技术、难以相处的客户、明显的复用潜力等），可能就会选择其他的过程模型⁵。

一旦选定了过程模型，就要根据所选的过程模型对过程框架做适应性修改，但在所有情况下前面讨论过的通用框架活动——沟通、策划、建模、构建和部署——都可以使用。过程框架既适用于线性模型，也适用于迭代和增量模型、演化模型，甚至是并发模型或构件组装模型。过程框架是不变的，是软件组织所进行的所有软件工作的基础。

但实际的工作任务是不同的。当项目经理问到：“我们如何完成这个框架活动”时，就意味着过程分解开始了。例如，一个小型的比较简单的项目在沟通活动中可能需要完成下列工作任务：

1. 列出需澄清的问题清单。
2. 与客户见面说明需澄清的问题。
3. 共同给出范围陈述。
4. 和所有相关人员一起评审范围陈述。
5. 根据需要修改范围陈述。

这些事件可能在不到48小时内发生。它是一种过程分解方式，这种方式适用于小型的比较简单的项目。

现在，我们考虑一个比较复杂的项目，它具有更广的范围和更重要的商业影响。这样

⁵ 回忆一下，项目特性对项目团队的结构也有相当大的影响，见21.2.3节。

个项目在沟通活动中可能需要完成下列工作任务：

1. 评审客户要求。
2. 计划并安排与客户召开正式的、有人主持的会议。
3. 研究如何详细说明推荐的解决方案和已有的方法。
4. 为正式会议准备一份“工作文档”和议程。
5. 召开会议。
6. 共同制定能够反映软件的数据、功能和行为特性的小规格说明。或者，从用户的角度出发建立描述软件的用例。
7. 评审每一份小规格说明或用例，确认其正确性、一致性和无歧义性。
8. 将这些小规格说明组装起来形成一份范围文档。
9. 和所有相关人员一起评审范围文档或用例集。
10. 根据需要修改范围文档或用例。

两个项目都执行了我们称之为沟通的框架活动，但第一个项目团队的软件工程工作任务只有第二个项目团队的一半。

21.5 项目

为了成功地管理软件项目，我们必须了解可能会出现什么问题（以便能够避免错误）。在一篇关于软件项目的优秀论文中，John Reel[REE99]定义了10个表示信息系统项目正处于危险状态的信号：

 那些信号
表明软件
项目正处于危
险状态？

1. 软件人员不了解其客户的要求。
2. 产品范围定义得很糟糕。
3. 没有很好地管理变更。
4. 选择的技术发生了变化。
5. 业务需求发生变化（或未被很好地定义）。
6. 最后期限是不切实际的。
7. 客户抵制。
8. 失去赞助（或从来没有真正得到过赞助）。
9. 项目团队缺乏具有合适技能的人员。
10. 管理者（和实践者）没有很好地利用已学到的最佳实践和教训。

642

在讨论特别困难的软件项目时，疲惫不堪的从业人员常常提及（半开玩笑地）90-90规则：系统前面90%的任务会花费所分配工作量和时间的90%，系统最后10%的任务也会花费所分配工作量和时间的90%[ZAH94]。导致该90-90规则的根源包含在上面列出的“信号”中。

“我们没有时间停下来空谈，我们已经来不及了。”

——M. Cleron

这太消极了！管理者如何避免上面提到的这些问题呢？Reel[REE99]提出了以下针对软件项目的易于理解的方法，包含5个部分：

1. 在正确的基础上开始工作。通过以下两点来完成：首先努力（非常努力）地正确理解要解决的问题，然后为每个参与项目的人员设置现实的目标和期望。这一点又通过组建合适

的开发团队（21.2.3节）以及给予团队完成工作所需的自由、权力和技术而得到加强。

2. 保持动力。很多项目的启动都有一个良好的开端，但是，后来慢慢地开始瓦解。为了维持动力，项目经理必须采取激励措施使人员变动量保持绝对最小，项目团队应该强调完成的每个任务的质量，而高层的管理应该尽可能不干涉团队的工作⁶。

3. 跟踪进展。对于一个软件项目，当工作产品（例如，模型、源代码、测试用例集）被生产出来和被批准（通过正式技术评审）时，跟踪项目进展要作为质量保证活动的一部分。此外，可以收集软件过程和项目测量（第22章）数据，然后对照软件开发组织的平均数据来评估项目的进展。

4. 做出聪明的决策。总体上，项目经理和软件团队的决策应该“保持项目的简单性”。只要有可能，就使用商用成品软件或现有的软件构件，在可以采用标准的方法时就避免定制接口，识别并避免显而易见的风险，以及分配比你认为的时间更多的时间来完成复杂或有风险的任务（你将需要每一分钟）。

5. 进行事后分析。建立统一的机制，从每个完成的项目中获取可学习的经验。评估计划的进度和实际的进度，收集和分析软件项目度量数据，从项目团队成员和客户处获取反馈，并记录下所有的发现。

643

21.6 W⁵HH原则

Barry Boehm[BOE96]在其关于软件过程和项目的优秀论文中指出：“你需要一个组织原则，可对它进行缩减来为简单的项目提供简单的（项目）计划。”Boehm提出了一种方法，该方法强调项目目标、里程碑和进度、责任、管理和技术方法以及需要的资源。他称之为W⁵HH原则。这种方法通过提出一系列问题，来导出对关键项目特性以及项目计划的定义：

- 为什么（Why）要开发这个系统？对这个问题的回答，可以使所有参与者评估软件工作的商业理由的有效性。换句话说，该系统的商业目的值得花费这些人员、时间和金钱吗？
- 将要做什么（What）？对这个问题的回答将制定完成项目所需的任务清单。
- 什么时候（When）做？就是标识出何时开展项目任务和何时达到里程碑，对这个问题的回答能够帮助团队安排好项目进度。
- 某功能由谁（Who）负责？在本章的前面，我们提到必须规定软件团队的每个成员的角色和责任。对这个问题的回答将帮助完成该项规定。
- 他们的机构组织位于何处（Where）？并非所有角色和责任均属于软件团队，客户、用户和其他共利益者也有责任。
- 如何（How）完成技术工作和管理工作？一旦确定了产品范围，必须定义项目的管理策略和技术策略。
- 每种资源需要多少（How much）？对这个问题的回答，是在对前面问题回答的基础上，通过估算（第23章）而得到的。

 我们如何定义关键的项目特性？

Boehm的W⁵HH原则可适用于任何规模和复杂度的软件项目。给出的问题为项目经理和软

⁶ 这句话的意思是：将官僚主义减少到最低程度，取消无关的会议，不要教条地依附于过程和项目规则。团队应该是自组织的，拥有自治权。

件团队提供了很好的计划大纲。

21.7 关键实践

Airlie Council⁷ 提出了一组“基于性能管理的关键软件实践”。这些实践“被高度成功的软件项目和组织（它们的“底线”性能大大优于产业界的平均水平）一致地使用，并被认为的确是关键的” [AIR99]。 [644]

关键实践⁸ 包括：基于度量的项目管理（第22章），经验成本和进度估计（第23和24章），获得价值跟踪（第24章），正式的风险管理（第25章），根据质量目标跟踪缺陷（第26章），人员计划管理（第21.2节）。每一个关键实践都贯穿于本书的第四部分。

SOFTWARE TOOLS

项目管理的软件工具

在此列出的工具都是通用的，适用于大部分项目管理活动。而特定的项目管理工具，如计划工具、评估工具和风险分析工具，将在后续章节中讲述。

代表性工具⁹

软件计划管理人员网 (www.spmn.com) 开发了一个简单的工具，称为“Project Control Panel”，为项目经理提供关于项目状态的直接指标。该工具有标尺，更像一个仪表板，是用Microsoft Excel实现的。下载地址是http://www.spmn.com/products_software.html。

Gantthead.com已经为项目经理开发了一套实用的检查表，下载地址是<http://www.gantthead.com/>。

Ittoolkit.com (www.ittoolkit.com) 收集了很多计划指南、过程模板和精巧的工作表，可以从光盘上获取。

21.8 小结

软件项目管理是软件工程的普适性活动。它先于任何技术活动之前开始，且持续贯穿于整个计算机软件的定义、开发和维护之中。

4个P——人员、产品、过程和项目，对软件项目管理具有重大的影响。必须将人员组织成有效率的团队，激励他们完成高质量的软件工作，并协调他们实现有效的沟通。产品需求必须在客户与开发者之间进行交流，划分（分解）成各个组成部分，并分配给软件团队。过程必须适合于人员和问题。选择通用过程框架，采用合适的软件工程范型，并挑选工作任务集合来完成项目的开发。最后，必须采用确保软件团队能够成功的方式来组织项目。

在所有软件项目中，最关键的因素是人员。可以按照很多不同的团队结构来组织软件工程师，从传统的控制层次到“开放式范型”团队。可以使用多种协调和沟通技术来支持项目团队的工作。一般而言，正式评审和非正式的人员间交流对开发者最有价值。 [645]

⁷ Airlie Council是由美国国防部组织的一个软件工程专家小组，其目的是为了开发在软件项目管理和软件工程中的最佳实践指南。

⁸ 这里只讨论与“项目完整性”相关的关键实践。

⁹ 在这里提到的工具只是此类工具的样例，并不表示本书支持它们。大多数情况下，工具的名字由各自的开发者注册为商标。

项目管理活动包含测量和度量、估算和进度安排、风险分析、跟踪和控制。这些主题将在以后几章中进一步探讨。

参考文献

- [AIR99] Airlie Council, "Performance Based Management: The Program Manager's Guide Based on the 16-Point Plan and Related Metrics," Draft Report, March 8, 1999.
- [BAK72] Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, vol. 11, no. 1, 1972, pp. 56–73.
- [BOE96] Boehm, B., "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, July 1996, pp. 73–82.
- [COC01] Cockburn, A., and J. Highsmith, "Agile Software Development: The People Factor," *IEEE Computer*, vol. 34, no. 11, November 2001, pp. 131–133.
- [CON93] Constantine, L., "Work Organization: Paradigms for Project Management and Organization," *CACM*, vol. 36, no. 10, October 1993, pp. 34–43.
- [COU80] Cougar, J., and R. Zawacki, *Managing and Motivating Computer Personnel*, Wiley, 1980.
- [CUR88] Curtis, B., et al., "A Field Study of the Software Design Process for Large Systems," *IEEE Trans. Software Engineering*, vol. SE-31, no. 11, November 1988, pp. 1268–1287.
- [CUR94] Curtis, B., et al., *People Management Capability Maturity Model*, Software Engineering Institute, 1994.
- [DEM98] DeMarco, T., and T. Lister, *Peopleware*, 2nd ed., Dorset House, 1998.
- [EDG95] Edgemon, J., "Right Stuff: How to Recognize It When Selecting a Project Manager," *Application Development Trends*, vol. 2, no. 5, May 1995, pp. 37–42.
- [FER98] Ferdinandi, P. L., "Facilitating Communication," *IEEE Software*, September 1998, pp. 92–96.
- [JAC98] Jackman, M., "Homeopathic Remedies for Team Toxicity," *IEEE Software*, July 1998, pp. 43–45.
- [KRA95] Kraul, R., and L. Streeter, "Coordination in Software Development," *CACM*, vol. 38, no. 3, March 1995, pp. 69–81.
- [MAN81] Mantei, M., "The Effect of Programming Team Structures on Programming Tasks," *CACM*, vol. 24, no. 3, March 1981, pp. 106–113.
- [PAG85] Page-Jones, M., *Practical Project Management*, Dorset House, 1985, p. vii.
- [REE99] Reel, J. S., "Critical Success Factors in Software Projects," *IEEE Software*, May, 1999, pp. 18–23.
- [WEI86] Weinberg, G., *On Becoming a Technical Leader*, Dorset House, 1986.
- [WIT94] Whitaker, K., *Managing Software Maniacs*, Wiley, 1994.
- [ZAH94] Zahniser, R., "Timeboxing for Top Team Performance," *Software Development*, March 1994, pp. 35–38.

习题与思考题

- 21.1 SEI的人员管理能力成熟度模型 (PM-CMM) 定义了培养优秀软件人员的“关键实践域”。你的老师将为你指派一个关键实践域，请你分析和总结该域。
- 21.2 描述三种现实生活中的实际情况，其中客户和最终用户是相同的人。也描述三种他们是不同的人的情况。
- 21.3 基于本章给出的信息和自己的经验，列举出能够增强软件工程师能力的“十条戒律”。即，列出10条指导原则，使得软件人员能够在工作中发挥其全部潜力。
- 21.4 高级管理者所做的决策会对软件工程项目团队的效率产生重大的影响。请列举5个实例来说明这一点。
- 21.5 你被指派为一个大型软件产品公司的项目经理。你的工作是管理该公司已被广泛使用的字处理软件的新版本的开发。因为必须获得新的收益，已经规定了紧迫的最后期限，

并对外公布。你会选择哪种团队结构？为什么？你会选择哪（些）种软件过程模型？为什么？

- 21.6 给出21.3.2节中讨论的页面布局功能的第一级分解。
- 21.7 要求开发一个小型应用软件，它的作用是分析一所大学开设的每一门课程，并输出课程的平均成绩（针对某个学期）。写出该问题的范围陈述。
- 21.8 你被指派为一个小型软件产品公司的项目经理。你的工作是开发一个有突破性的产品，该产品结合了虚拟现实的硬件和高超的软件。因为家庭娱乐市场的竞争非常激烈，完成这项工作的压力很大。你会选择哪种团队结构？为什么？你会选择哪（些）种软件过程模型？为什么？
- 21.9 在一个为遗传工程领域服务的公司中，你被指派为项目经理。你的工作是管理一个软件新产品的开发，该产品能够加速基因分类的速度。这项工作是面向研究及开发的，但其目标是在下一年度内生产出产品。你会选择哪种团队结构？为什么？你会选择哪（些）种软件过程模型？为什么？
- 21.10 复习一下Weinberg的书[WEI86]，并写出一份2~3页的总结，说明在使用MOI模型时应该考虑的问题。
- 21.11 在一个信息系统组织中，你被指派为项目经理。你的工作是开发一个应用程序，该程序类似于你的团队以前已经做过的某个项目，只是规模更大而且更复杂一些。需求已经由用户写成文档。你会选择哪种团队结构？为什么？你会选择哪（些）种软件过程模型？为什么？

推荐读物与阅读信息

项目管理研究所（PMI）出版的书（《Guide to the Project Management Body of Knowledge》，PMI，2001）覆盖了项目管理的所有重要方面。Murch（《Project Management: Best Practices for IT Professionals》，Prentice-Hall，2000）讲授了基本技能并提供IT项目所有阶段的详细指导。Lewis（《Project Managers Desk Reference》，McGraw-Hill，1999）提出了一个16步的过程，可用来计划、监控任何类型的项目。 McConnell（《Professional Software Development》，Addison-Wesley，2004）提供了实用的建议，指导如何获得“更短的进度计划、更高质量的产品和更成功的项目”。

由Weinberg所著的一套4册的系列丛书（《Quality Software Management》，Dorset，1992，1993，1994，1996）介绍了基本的系统思想和管理概念，解释了如何有效地使用度量，并且说明了“一致的行为”，即建立管理者需要、技术人员需要以及商业需要之间的协调配合能力。它不仅为新的管理者同时也为有经验的管理者提供了有用的信息。Futrell和他的同事（《Quality Software Project Management》，Prentice-Hall，2002）提出了大量的项目管理的处理方法。

Phillips（《IT Project Management: on Track from Start to Finish》，McGraw-Hill/Osborne，2002）、Charvat（《Project Management Nation》，Wiley，2002）、Schwalbe（《Information Technology Project Management》，second edition，Course Technology，2001）以及Holtsnider和Jaffe（《IT Manager's Handbook》，Morgan Kaufmann Publishers，2000），这些是软件项目管理书籍的代表作。Brown和他的同事（《Antipatterns in Project Management》，Wiley，2000）讨论了软件项目管理中不能做的事情。

Brooks (《The Mythical Man-Month》, Anniversary Edition, Addison-Wesley, 1995) 更新了他的杰作，给出了关于软件项目及管理问题的新见解。McConnell (《Software Project Survival Guide》, Microsoft Press, 1997) 为软件项目经理提供了卓越的有实效的行动指南。Purba和Shah (《How to Manage a Successful Software Project》, Second edition, Wiley, 2000) 提供了很多案例研究，指出为什么一些项目能成功，而另外一些项目会失败。Bennatan (《On Time Within Budget》, third edition, Wiley, 2000) 为软件项目经理提供了实用的提示和指导。

可以证明，软件项目管理中最重要的方面是人员的管理。Cockburn (《Agile Software Development》, Addison-Wesley, 2002) 提出了关于软件人员的论述，是到目前为止关于软件人员的最好的论述之一。DeMarco和Lister[DEM98]撰写了关于软件人员和软件项目的最权威的著作。此外，关于这一主题，近年来出版的如下书籍值得一读：

- Beaudouin-Lafon, M., *Computer Supported Cooperative Work*, Wiley-Liss, 1999.
- Carmel, E., *Global Software Teams: Collaborating Across Borders and Time Zones*, Prentice Hall, 1999.
- Constantine, L., *Peopleware Papers: Notes on the Human Side of Software*, Prentice-Hall, 2001.
- Humphrey, W. S., *Managing Technical People: Innovation, Teamwork, and the Software Process*, Addison-Wesley, 1997.
- Humphrey, W. S., *Introduction to the Team Software Process*, Addison-Wesley, 1999.
- Jones, P. H., *Handbook of Team Design: A Practitioner's Guide to Team Systems Development*, McGraw-Hill, 1997.
- Karolak, D. S., *Global Software Development: Managing Virtual Teams and Environments*, IEEE Computer Society, 1998.

Ensworth (《The Accidental Project Manager》, Wiley, 2000) 对由技术人员转为项目经理的人提供了很多有益的指导。Weinberg的另一本优秀论著[WEI86]是每个项目经理和每个团队负责人必读的书籍，它能指导你更加有效地进行工作。

以下列出的一些畅销“管理”书并不和软件世界特别相关，有时还过于简单，过于概括：Bossidy (《Execution: The Discipline of Getting Things Done》, Crown Publishing, 2002), Drucker (《Management Challenges for the 21st century》, Harper Business, 1999), Buckingham和Coffman (《First, Break All the Rules: What the World's Greatest Managers Do Differently》, Simon and Schuster, 1999)，以及Christensen (《The Innovator's Dilemma》, Harvard Business School Press, 1997)，这些书强调由快速变化的经济定义的“新规则”。比较老的书如《Who Moved My Cheese?》、《The One-Minute Manager》和《In Search of Excellence》仍然很有价值，帮助你更有效地管理人员和项目。

大量的关于软件项目管理的信息源可以在因特网上获得。最新的WWW参考文献目录可在SEPA Web站点<http://www.mhhe.com/pressman>上找到。

第22章 过程和项目度量

要点浏览

概念：软件过程和项目度量是定量的测量，这些测量能使软件工程师更深入地了解软件过程的功效，以及使用该过程作为框架进行开发的项目的功效。度量时，首先收集基本的质量数据和生产率数据，然后分析这些数据、与过去的平均值进行比较，通过评估来确定是否已有质量和生产率的提高。度量也可以用来查明问题区域，以便确定合适的补救方法，并改进软件过程。

人员：软件度量由软件管理者来分析和评估。测量数据通常由软件工程师来收集。

重要性：如果不进行测量，只能根据主

观评价来做判断。通过测量，可以发现趋势（好的或坏的），可以更好地进行估算，随着时间的推移能够获得真正的改进。

步骤：首先确定一组有限的易于收集的过程测量和项目测量。通常使用面向规模或面向功能的度量对这些测量进行规范化。然后，对测量结果进行分析，并与该组织以前完成的类似项目的平均数据进行比较。最后评估趋势，并给出结论。

工作产品：一组软件度量，它们提供了对过程的洞察力和对项目的理解。

质量保证措施：通过采用一致而简单的测量计划来保障，但在对个人表现进行评估、奖励或惩罚时，该计划绝对不适用。

关键概念

DRE
度量
面向功能
面向对象
私有
项目
过程
公用
质量
面向规模
用例
WebApp
度量基线
度量大纲
SSPI

通过提供目标评估的机制，测量使我们能够对项目和过程有更深入的了解。Lord Kelvin曾经说过：

当你能够测量你所说的事物，并能用数字表达它时，你就对它有了一定的了解；当你不能测量它，也不能用数字来表达时，就说明你对它的了解还很贫乏，不能令人满意：这可能是知识的开始，但你在思想上还远远没有进入科学的境地。

软件工程界已经认可了Lord Kelvin的话。但这并不是一帆风顺，也不是只有一点点的争论。

测量可以应用于软件过程中，目的是持续地改进软件过程。测量也可以应用于整个软件项目中，辅助进行估算、质量控制、生产率评估及项目控制。最后，软件工程师还可以使用测量来帮助评估工作产品的质量，并在项目进展过程中辅助进行战术决策（第15章）。

在Park、Goethert和Florac[PAR96]的关于软件测量的指导手册中，他们讨论了进行测量的理由：（1）刻画——通过刻画而获得对过程、产品、资源和环境的了解，并建立同未来评估进行比较的基线；（2）评价——通过评价来确定相对于计划的状况；（3）预测——通过理解过程和产品间的关系，并构造这些关系的模型来进行预测；（4）改进——通过识别障碍、根

本原因、低效率和其他改进产品质量和过程性能的机会来进行改进。

测量是一个管理工具，如果能正确地使用，它将为项目管理者提供洞察力。因此，测量能够帮助项目管理者和软件团队制定出使项目成功的决策。

22.1 过程领域和项目领域中的度量



过度量具有长期的影响，其目的是改进过程本身。项目度量常常对过度量的发展起到促进作用。

过度量的收集涉及所有的项目，而且要经历相当长的时间，目的是提供能够引导长期的软件过程改进的一组过程指标。项目度量使得软件项目管理者能够：(1)评估正在进行中的项目的状态；(2)跟踪潜在的风险；(3)在问题造成不良影响之前发现它们；(4)调整工作流程或任务；(5)评估项目团队控制软件工作产品质量的能力。

测量数据由项目团队收集，然后被转换成度量数据在项目期间使用。测量数据也可以传送给那些负责软件过程改进的人员。因此，很多相同的度量既可用于过程领域，又可用于项目领域。

22.1.1 过度量和软件过程改进

改进任何过程的唯一合理方法就是测量该过程的特定属性，再根据这些属性建立一组有意义的度量，然后使用这组度量提供的指标来导出过程改进策略。但是，在讨论软件度量及其对软件过程改进的影响之前，我们必须注意到：过程仅是众多“改进软件质量和组织性能的控制因素”中的一种[PAU94]。

在图22-1中，过程位于三角形的中央，连接了三个对软件质量和组织绩效有重大影响的因素。其中，人员的技能和动力[BOE81]被认为是对质量和绩效影响最大的因素，产品复杂性对质量和团队绩效也有相当大的影响，过程中采用的技术（即软件工程方法和工具）也有一定的影响。另外，过程三角形位于环境条件圆圈内，环境条件包括：开发环境（如CASE工具）、商业条件（如交付期限、业务规则）、客户特性（例如，交流和协作的容易程度）。

650



软件人员的技能和动力是影响软件质量的最重要的因素。

我们可以间接地测量软件过程的功效。也就是说，我们可以根据从过程中获得的结果来导出一组度量。这些结果包括：在软件发布之前发现的错误数的测度，提交给最终用户并由最终用户报告的缺陷的测度，交付的工作产品（生产率）的测度，花费的工作量的测度，花费时间的测度，与进度计划是否一致的测度，以及其他测度。我们还可以通过测量特定软件工程任务的特性来导出过度量。例如，我们可以测量第2章中所描述的一般软件工程活动所花费的工作量和时间。

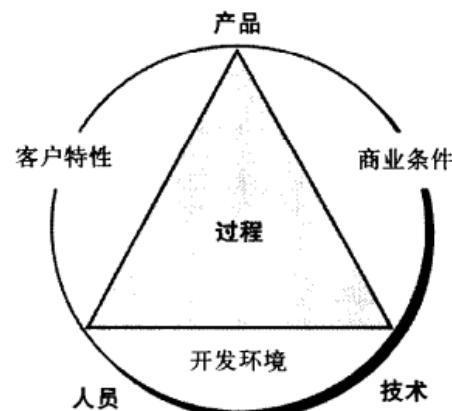
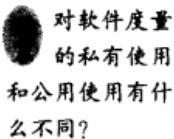


图22-1 软件质量和组织有效性的决定因素（改自[PAU94]）

“软件度量让你知道什么时候笑，什么时候哭。”

—Tom Gilb



对软件度量的私有使用和公用使用有什么不同？

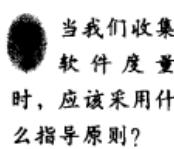
Grady [GRA92] 认为不同类型的过程数据的使用可以分为“私有的和公用的”。软件工程师可能对其个人收集的度量的使用比较敏感，这是很自然的事。这些数据对此人应该是私有的，是仅供此人参考的指标。私有度量的例子有：个人缺陷率、软件构件缺陷率和开发过程中发现的错误数。

“私有过程数据”的观点与Humphrey [HUM95] 所建议的个人软件过程方法（第2章）相一致。Humphrey认为软件过程改进能够、也应该开始于个人级。私有过程数据是软件工程师个人改进其工作的重要驱动力。[651]

有些过程度量对于软件项目团队是私有的，但对所有团队成员是公用的。例如，主要软件功能（由多个开发人员共同完成）的缺陷报告、正式技术评审中发现的错误，以及每个构件或功能的代码行数或功能点数¹。这些数据可由团队进行评审，以便找出能够改善团队性能的指标。

公用的度量一般吸收了原本是个人的或团队的私有信息。收集和评估项目级的缺陷率（肯定不能归咎于某个个人）、工作量、时间以及相关的数据，来找出能够改善组织过程性能的指标。

软件过程度量对于组织提高其整体的过程成熟度能够提供很大的帮助。不过，就像所有其他度量一样，软件过程度量也可能被误用，产生的问题比它们所能解决的问题更多。Grady [GRA92] 提出了一组“软件度量规则”。管理者和开发者在制定过程度量大纲时，这些规则都适用：



当我们收集软件度量时，应该采用什么指导原则？

- 解释度量数据时使用常识，并考虑组织的敏感性。
- 向收集测量和度量的个人及团队定期提供反馈。
- 不要使用度量去评价个人。
- 与开发者和团队一起设定清晰的目标，并确定为达到这些目标需要使用的度量。
- 不要用度量去威胁个人或团队。
- 指出问题区域的度量数据不应该被“消极地”看待，这些数据仅仅是过程改进的指标。
- 不要在某一个别的度量上纠缠，而无暇顾及其他重要的度量。

随着一个组织更加得心应手地收集和使用过程度量，简单的指标获取方式就会逐渐被更加精确的方法所取代，该方法称为统计软件过程改进（statistical software process improvement, SSPI）。本质上，SSPI使用软件失效分析方法收集在应用软件、系统或产品的开发及使用过程中所遇到的所有错误及缺陷信息²。[652]

22.1.2 项目度量

软件过程度量用于战略目的，而软件项目度量则用于战术目的。也就是说，项目管理者和软件项目团队通过使用项目度量及从中导出的指标，可以改进项目工作流程和技术活动。

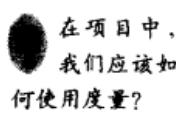
在大多数软件项目中，项目度量的第一次应用是在估算阶段。从过去项目中收集的度量

¹ 代码行和功能点度量将在22.2.1节和22.2.2节讨论。

² 在本书中，错误是指软件工程工作产品中的瑕疵，这些瑕疵在交付给最终用户前已经被发现。而缺陷是指交付给最终用户之后才发现的瑕疵。需要提醒大家的是，其他人并没有进行这样的区分。进一步的讨论将在第26章进行。

可以作为估算当前软件工作工作量及时间的基础。随着项目的进展，可以将花费的工作量及时间的测量与最初的估算值（及项目进度）进行比较。项目管理者可以使用这些数据来监控项目的进展。

随着技术工作的启动，其他项目度量也开始有意义了。生产率可以根据创建的模型、评审的时间、功能点以及交付的源代码行数来测量。此外，对每个软件工程任务中所发现的错误也要进行跟踪。在软件从需求到设计的演化过程中，需要收集技术度量（第15章）来评估设计质量，并提供若干指标，这些指标将会影响代码生成及测试所采用的方法。



项目度量的目的是双重的。首先，利用度量能够对开发进度进行必要的调整，以避免延迟，并减少潜在的问题及风险，从而使得开发时间减到最少。其次，项目度量可在项目进行过程中评估产品质量，必要时可调整技术方法以提高质量。

随着质量的提高，缺陷会减到最少。而随着缺陷数的减少，项目中所需的修改工作量也会降低，这将使整个项目成本降低。

SAFEHOME

建立度量方法

[场景] SafeHome软件项目即将开始，在Doug Miller的办公室。

[人物] Doug Miller, SafeHome软件工程团队经理；Vinod Raman和Jamie Lazar, 产品软件工程团队成员。

[对话]

Doug: 在项目工作开始之前，我想你们应该定义并收集一组简单的度量。首先，必须确定你们的目标。

Vinod（皱着眉头）：以前，我们从来没有做过这些，并且……

Jamie（打断他的话）：基于时间线的管理已经讨论过了，我们根本没有时间。度量到底有什么好处？

Doug（举手示意停止发言）：大家且慢，停一下。正因为我们以前从来没有做过度量，所以现在更要开始做。并且我说的度量工作根本不会占用很多时间……事实上，它只会节省我们的时间。

Vinod：为什么？

Doug：你看，随着我们的产品更加智能化，变成支持Web的，等等，我们将要做更多的内部软件工程工作……我们需要了解开发软件的过程……并改进过程，使得我们能够更好地开发软件。要实现这一点，唯一的方法就是测量。

Jamie：但是我们的时间很紧迫，Doug。我不赞同太多的琐碎的文字工作……我们需要时间来完成工作，而不是收集数据。

Doug：Jamie，一个工程师的工作包括收集数据、评估数据、根据上述结果来改进产品和过程。我错了吗？

Jamie：不，但是……

Doug：如果我们限定要收集的测量数不超过5个或6个，并集中关注质量方面，那会怎么样？

Vinod: 没有人能够反对高质量……

Jamie: 对……但是，我不知道，我仍然认为这是不必要的。

Doug: 在这个问题上，请听我的！关于软件度量你们了解多少？

Jamie (看着Vinod): 不多。

Doug: 这是一些Web参考资料……花几个小时读完它。

Jamie (微笑着): 我还认为你说的这件事不会花费任何时间。

Doug: 花费在学习上的时间绝对不会浪费……去做吧！然后我们要建立一些目标，提几个问题，并且定义我们需要收集的度量。

22.2 软件测量

在第15章中，我们谈到软件测量有两种分类方法：(1) 软件过程（如花费的成本和工作量）和产品（如产生的代码行（LOC）、运行速度以及某段时间内报告的缺陷）的直接测量；(2) 产品的间接测量，包括功能、质量、复杂性、有效性、可靠性、可维护性，以及在第15章中谈到的许多其他的“产品特性”。

“并非能够被计算的每件事物都有价值，也并非有价值的每件事物都能够被计算。”

——阿尔伯特·爱因斯坦

将项目度量联合起来可以得到整个软件组织公用的过程度量。但是，一个组织如何将来自不同个人或项目的度量结合起来呢？



因为有很多因素会影响软件工作，因此不要用度量去比较个人或团队。

为了说明这个问题，我们看一个简单的例子。两个不同项目团队中的人将他们在软件过程中发现的所有错误进行了记录和分类。然后，将这些个人的测量结合起来就产生了团队的测量。在软件发布前，团队A在软件过程中发现了342个错误，团队B发现了184个错误。所有其他情况都相同，那么在整个过程中哪个团队能更有效地发现错误呢？因为不知道项目的规模或复杂性，所以我们不能回答这个问题。不过，如果度量采用规范化的方法，就有可能产生在更大的组织范围内进行比较的软件度量。如此说来，面向规模的度量和面向功能的度量都是规范化的方法。

654

22.2.1 面向规模的度量

面向规模的软件度量是通过规范化质量和（或）生产率的测量值而得到的，这些测量都基于已经开发的软件的规模。如果软件组织一直在做简单的记录，就会产生一个如图22-2所示的面向规模测量的表。该表列出了在过去几年中完成的每一个软件开发项目及其相关的测量数据。查看alpha项目的数据（图22-2）：花费了24人·月的工作量，成本为168 000美元，产生了12 100行代码。需要提醒大家的是，表中记录的工作量和成本涵盖了所有软件工程活动（分析、设计、编码及测试），而不仅仅是编码。有关alpha项目更进一步的信息包括：产生了365页的文档；在软件发布之前，发现了134个错误；在软件发布给客户之后运行的第一年中遇到了29个缺陷；有3个人参加了alpha项目的软件开发工作。

项目	代码行	工作量	成本(千美元)	文档页数	错误	缺陷	人员
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

图22-2 面向规模的度量

为了产生能和其他项目中同类度量进行比较的度量，我们选择代码行作为规范化值。根据表中所包含的基本数据，每个项目都能得到一组简单的面向规模的度量：

- 每千行代码 (KLOC) 的错误数；
- 每千行代码 (KLOC) 的缺陷数；
- 每千行代码 (KLOC) 的成本；
- 每千行代码 (KLOC) 的文档页数。

此外，还能计算出其他有意义的度量：

- 每人·月错误数；
- 每人·月千代码行 (KLOC)；
- 每页文档的成本。

对于面向规模的度量是否为软件过程测量的最好的方法，对此并没有普遍一致的观点[JON86]。大多数争议都围绕着使用代码行 (LOC) 作为关键的测量是否合适。LOC测量的支持者们声称：LOC是所有软件开发项目的“产物”，并且很容易进行计算；许多现有的软件估算模型都是使用LOC或KLOC作为关键的输入；而且已经有大量的文献和数据都涉及LOC。另一方面，反对者们则认为，LOC测量依赖于程序设计语言；它们对设计得很好但较短的程序会产生不利的评判；它们不适用于非过程语言；而且它们在估算时需要一些可能难以得到的信息（如，计划人员必须在分析和设计远未完成之前，就要估算出要产生的LOC）。

KEY POINT

面向规模的度量已经得到了广泛的应用，但对其有效性和适用性的争论一直在进行。

655

22.2.2 面向功能的度量

面向功能的软件度量使用功能（由应用系统提供）测量数据作为规范化值。应用最广泛的面向功能的度量是功能点 (Function Point, FP)。功能点是根据软件信息域的特性及复杂性来计算的。功能点的计算方法已经在第15章中讨论过了³。

与LOC测量一样，功能点测量也是有争议的。支持者们认为FP与程序设计语言无关，对于使用传统语言和非过程语言的应用系统来说，它都是比较理想的；而且它所依据的数据是在项目开发初期就可能得到的数据。因此，作为一种估算方法，FP更有吸引力。反对者们则声称这

³ 对FP计算的详细讨论，参见15.3.1节。

种方法需要某种“熟练手法”，因为计算的依据是主观的而非客观的数据，信息域（及其他方面）的计算可能难以在事后收集。而且，FP没有直接的物理意义，它仅仅是一个数字而已。

22.2.3 调和代码行和功能点的度量方法

代码行和功能点之间的关系依赖于实现软件所采用的程序设计语言及设计的质量。很多研究试图将FP测量和LOC测量联系起来。引用Albrecht和Gaffney [ALB83] 的话：

本研究的论点是：应用（程序）所提供的功能数可以从所使用或提供的数据的主要组成部分的详细记录⁴中估算出来，或是直接从记录中得到。更进一步，功能的估算应该与要开发的LOC数及开发所需的工作量关联起来。

表22-1⁵ [QSM02] 给出了在不同的程序设计语言中实现一个功能点所需的平均代码行数的粗略估算：

656

表22-1 各程序设计语言中实现一个功能点所需的代码行数的粗略估算

程序设计语言	LOC/FP			
	平均值	中值	低值	高值
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP69	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Clipper	38	39	27	70
COBOL	77	77	14	400
Cool: Gen/IEF	38	31	10	180
Culprit	51	—	—	—
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Excel47	46	—	31	63
Focus	43	42	32	56
FORTRAN	—	—	—	—
FoxPro	32	35	25	35
Ideal	66	52	34	203
IEF/Cool: Gen	38	31	10	180
Informix	42	31	24	57
Java	63	53	77	—
JavaScript	58	63	42	75
JCL	91	123	26	150
JSP	59	—	—	—
Lotus Notes	21	22	15	25

⁴ 重要的是要注意，“主要组成部分的详细记录”可用不同的方式来解释。工作在面向对象开发环境中的软件工程师使用类或对象的数量作为主要的规模度量。负责维护的组织可能根据工程变更工单（第27章）的数量来看待项目规模。负责信息系统的组织可能查看受应用影响的业务处理的数量。

⁵ 该表在这里的使用得到了Quantitative Software Management (www.qsm.com) 的许可（2002年版权）。

(续)

程序设计语言	LOC/FP			
	平均值	中值	低值	高值
Mantis	71	27	22	250
Mapper	118	81	16	245
Natural	60	52	22	141
Oracle	30	35	4	217
PeopleSoft	33	32	30	40
Perl	60	—	—	—
PL/I	78	67	22	263
Powerbuilder	32	31	11	105
REXX	67	—	—	—
RPGII/III	61	49	24	155
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
VBScript36	34	27	50	—
Visual Basic	47	42	16	158

查看上表可知，C++的一个LOC所提供的“功能性”大约是C的一个LOC的2.4倍（平均来讲）。而且，Smalltalk的一个LOC所提供的“功能性”至少是传统程序设计语言（如Ada、COBOL或C）的4倍。利用上表中所包含的信息，只要知道了程序设计语言的语句行数，就可以“逆向”[JON98]估算现有软件的功能点数量。

657

人们发现基于功能点和LOC的度量都是对软件开发工作量和成本的比较精确的判定。然而，如果使用LOC和FP进行估算，还必须要建立一个历史信息基线。

在过程度量和项目度量中，我们最关心的是生产率和质量——软件开发“输出量”（作为投入的工作量和时间的函数）的测量和对生产的工作产品的“适用性”的测量。为了进行过程改进和项目策划，我们必须掌握历史的情况。在以往的项目中，软件开发的生产率是多少？生产的软件质量如何？怎样利用以往的生产率数据和质量数据推断现在的生产率和质量？如何利用这些数据帮助我们改进过程，以及更精确地规划新的项目？

22.2.4 面向对象的度量

传统的软件项目度量（LOC或FP）也可以用于估算面向对象的软件项目。但是，这些度量并没有提供对进度和工作量进行调整的足够的粒度，而这却是我们在演化模型或增量模型中进行迭代时所需要的。Lorenz和Kidd[LOR94]提出了下列用于OO项目的度量：



多个场景脚本涉及同一个功能或数据对象，这种情况很常见，因此应该慎重使用脚本数量这个度量。

场景脚本的数量。场景脚本（与贯穿于本书第二、三部分中的用例类似）是一个详细的步骤序列，用来描述用户和应用系统之间的交互。应用系统的规模及测试用例（一旦构造出系统，就必须设计测试用例来测试该系统）的数量都与场景脚本的数量紧密相关。

关键类的数量。关键类是“高度独立的构件”[LOR94]，在面向对象分析的早期进行定义（第8章）⁶。由于关键类是问题域的核心，因此，这些类的数

⁶ 在本书的第二部分中，关键类被称为分析类。



量既是开发软件所需工作量的指标，也是系统开发中潜在的复用数量的指标。

支持类的数量。支持类是实现系统所必需的但又不与问题域直接相关的类。例如，UI类、数据库访问及操作类、计算类。另外，对于每一个关键类，都可以开发其支持类。支持类的数量既是开发软件所需工作量的指标，也是系统开发中潜在的复用数量的指标。

每个关键类的平均支持类数量。通常，关键类在项目的早期就可以确定下来，而支持类的定义则贯穿于项目的始终。对于给定的问题域，如果知道了每个关键类的平均支持类数量，估算（根据类的总数）将变得非常简单。

Lorenz和Kidd指出：在采用GUI的应用中，支持类是关键类的2~3倍；在不采用GUI的应用中，支持类是关键类的1~2倍。

子系统的数量。子系统是实现某个功能（对系统最终用户可见）的类的集合。一旦确定了子系统，人们就更容易制定出合理的进度计划，并将子系统的工作在项目人员之间进行分配。

为了将类似于上述的那些度量有效地应用于面向对象的软件工程环境中，必须将它们随同项目测量（例如花费的工作量、发现的错误和缺陷、建立的模型或文档资料）一起收集。随着数据库规模的增长（在完成大量项目之后），面向对象的测量数据和项目测量数据之间的关系将提供有助于项目估算的度量。

22.2.5 面向用例的度量

与LOC或FP相类似，使用用例⁷作为规范化的测量应该是合理的。同FP一样，用例也是在软件过程早期进行定义的。在重大的建模活动和构造活动开始之前，就允许使用用例进行估算。用例描述了（至少是间接地）用户可见的功能和特性，这些都是系统的基本需求。用例与程序设计语言无关。另外，用例的数量同应用系统的规模（LOC）和测试用例的数量成正比，而测试用例是为了充分测试该应用系统而必须设计的。

由于可以在不同的抽象级别上创建用例，所以用例的大小没有标准。由于对用例本身还没有标准的测量，因此，将用例作为规范化的测量（如，每个用例花费的工作量）是不可信的。尽管许多研究人员试图推导出用例度量（如，[SMI99]），但仍有很多工作要做。

22.2.6 Web工程项目度量

所有Web工程项目（本书第三部分）的目标都是建立一个Web应用（WebApp），交付给最终用户一个内容和功能的结合体。很难将那些用于传统软件工程项目的测量和度量直接转化为Web应用系统中。然而，Web工程组织应该建立一个数据库，随着大量项目的完成，就可以使用该数据库来评估组织内部的生产率和质量。在这些测量中，可以收集的有：

静态Web页的数量。静态内容的Web页（即最终用户不能控制页面显示的内容）是所有Web应用系统最普遍的特征。这些页面复杂性较低，通常构造静态页面所需的工作量少于动态页面。这项测量提供了一个指标，标志着应用系统的整体规模和开发应用系统所需的工作量。

动态Web页的数量。在所有的电子商务应用、搜索引擎、金融应用以及许多其他Web应用中，动态内容的Web页（即根据最终用户的操作，页面上显示出相应的定制内容）是必需的

⁷ 关于用例的讨论，贯穿于本书的第二、三部分。

要素。动态页面复杂性较高，构造动态页面所需的工作量高于静态页面。这项测量提供了一个指标，标志着应用系统的整体规模和开发应用系统所需的工作量。

内部页面链接的数量。内部页面链接就是一个指针，它在Web应用中提供了到达其他某个Web页的超链接。这项测量提供了一个Web应用内部结构互连程度的指标。随着页面链接的增加，花费在导航设计和开发上的工作量也会增加。

永久数据对象的数量。Web应用可以访问一个或多个永久数据对象（如数据库或数据文件）。随着永久数据对象数量的增加，Web应用系统的复杂性也会增加，实现应用系统所需的工作量也会成比例地增加。

通过界面连接的外部系统的数量。Web应用系统必须经常与“后台的”业务应用相连接。随着界面连接需求的增多，系统复杂性和开发工作量也会增加。

静态内容对象的数量。静态内容对象包括静态文本、图像、视频、动画和音频信息，它们在Web应用中被集成在一起。在一个Web页中，可能同时出现多个内容对象。

动态内容对象的数量。动态内容对象是根据最终用户的操作而产生的，包括内部产生的文本、图像、视频、动画和音频信息，它们在Web应用中被集成在一起。在一个Web页中，可能同时出现多个内容对象。

可执行的功能的数量。可执行的功能（如，脚本或小程序）为最终用户提供了某些计算服务。随着可执行功能数量的增加，建模和构造的工作量也会随之增加。

上面提到的每个测量都可以在Web工程过程的初期就确定下来。

例如，我们可以定义一个度量，来反映Web应用所需的最终用户的定制程度，并使它与Web工程项目花费的工作量以及评审和测试中发现的错误关联起来。为此，我们进行以下定义：

$$N_{sp} = \text{静态Web页的数量}$$

$$N_{dp} = \text{动态Web页的数量}$$

那么

$$\text{定制指数 } C = N_{dp}/(N_{dp} + N_{sp})$$

C的取值范围是0到1。随着C值的增大，Web应用的定制水平将成为一个重大的技术问题。

类似的Web应用度量也可以被计算出来，并同项目测量（如花费的工作量、发现的错误和缺陷、已建立的模型或文档）关联起来。随着数据库规模的扩大（在完成了很多项目之后），Web应用测量与项目测量之间的关系将提供有助于项目估算的指标。

SOFTWARE TOOLS

项目度量和过度量

目的：辅助进行软件测量和度量的定义、收集、评估和报告。

机制：每个工具在应用上都有所不同，但是所有的工具都提供了收集和评估数据的方法，这些数据可用来计算软件度量。

代表性工具⁸

Function Point WORKBENCH，由Charismatek (www.Charismatek.com.au) 开发，提供了大量的面向FP的度量。

⁸ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

MetricCenter, 由Distributive Software (www.distributive.com) 开发, 支持自动化的数据收集、分析、图表格式化、报表生成及其他测量任务。

PSM Insight, 由Practical Software and Systems Measurement (www.psmsc.com) 开发; 帮助创建项目测量数据库及随后进行的分析。

SLIM tool set, 由QSM (www.qsm.com) 开发, 提供了一整套的度量和估算工具。

SPR tool set, 由Software Productivity Research (www.spr.com) 开发, 收集了一整套面向FP的工具。

TychoMetrics, 是由Predicate Logic, Inc. (www.predicate.com) 开发的一个工具组, 用来管理度量的收集和报告。

22.3 软件质量度量

软件工程的基本目标是在某个时间框架内开发出满足市场需要的高质量的系统、应用或产品。为了达到这个目标, 软件工程师必须在成熟的软件过程背景下, 使用有效的方法及现代化的工具。此外, 一个优秀的软件工程师(及优秀的软件工程管理者)必须通过测量来判断能否实现高质量。

将软件工程师个人收集的私有度量结合起来, 可以提供项目级的度量。虽然可以收集到很多质量测量数据, 但在项目级上最主要的还是测量错误和缺陷的数量。从这些测量中导出的度量能够提供一个指标, 表明个人及小组在软件质量保证和控制活动上的效力。[661]

度量——比如说工作产品(如需求或设计)——每功能点的错误数、在评审中每小时发现的错误数、及测试中每小时发现的错误数, 使我们能够深入了解度量所涉及的活动的功效。有关错误的数据也能用来计算每个过程框架活动的缺陷排除效率(defect removal efficiency, DRE)。DRE将在22.3.2节中讨论。

22.3.1 测量质量

虽然有很多关于软件质量的测量指标⁹, 但正确性、可维护性、完整性和可用性为项目团队提供了有用的指标。Gilb [GIL88] 分别给出了它们的定义和测量。

WebRef

关于软件质量及相关主题(包括度量)的优秀信息源可在www.qualityworld.com上找到。

正确性: 一个程序必须能够正确地执行, 否则对于用户就没有价值了。正确性是软件完成所要求的功能的程度。最常用的关于正确性的测量是每千行代码(KLOC)的缺陷数, 这里的缺陷是指已被证实不符合需求的地方。当考虑软件产品的整体质量时, 缺陷是指在程序发布后经过了全面使用, 由程序用户报告的问题。为了进行质量评估, 缺陷是按标准时间段来计数的, 典型的是一年。

可维护性: 与任何其他软件工程活动相比, 软件维护需要更多的工作量。可维护性是指遇到错误时程序能够被修改的容易程度, 环境发生变化时程序能够适应的容易程度, 用户希望变更需求时程序能够被增强的容易程度。还没有一种方法可以直接测量可维护性, 因此我们只能采用间接测量。有一种简单的面向时间的度量, 称为平均变更时间(mean-time-to-change, MTTC)。

⁹ 关于影响软件质量的因素和可用于评估软件质量的度量已在第15章中详细讨论。

它包括分析变更请求、设计合适的修改方案、实现变更并进行测试、以及把该变更发布给全部用户所花的时间。一般情况下，与那些不可维护的程序相比，可维护的程序应有较低的MTTC（对于相同类型的变更）。

完整性：在防火墙和黑客的时代，软件完整性变得越来越重要了。这个属性测量的是一个系统对安全性攻击（包括偶然的和蓄意的）的抵抗能力。软件的所有三个成分（即程序、数据及文档）都会遭到攻击。

662 完整性：为了测量完整性，必须定义另外两个属性：危险性和安全性。危险性是指一个特定类型的攻击在给定的时间内发生的概率（能够估算或根据经验数据导出）。安全性是指一个特定类型的攻击将被击退的概率（能够估算或根据经验数据导出）。一个系统的完整性可以定义为：

$$\text{完整性} = \Sigma [1 - (\text{危险性} \times (1 - \text{安全性}))]$$

例如，假设危险性（发生攻击的可能性）是0.25，安全性（击退攻击的可能性）是0.95，则系统的完整性是0.99（很高）；另一方面，假设危险性是0.5，击退攻击的可能性仅是0.25，则系统的完整性只有0.63（低得无法接受）。

可用性：如果一个程序不容易使用，即使它完成的功能很有价值，也常常注定要失败。可用性力图对“使用的容易程度”进行量化，并可以根据第12章中给出的特性来测量。

在被建议作为软件质量测量的众多因素中，上述四个因素仅仅是一个样本。关于这些内容，在第15章中已经给出了更详细的讨论。

22.3.2 缺陷排除效率

缺陷排除效率（DRE）是在项目级和过程级都有意义的质量度量。本质上，DRE是对质量保证及控制活动中滤除缺陷能力的测量，而这些质量保证及质量控制活动贯穿应用于所有过程框架活动中。

当把项目作为一个整体来考虑时，可按如下方式定义DRE：

$$\text{DRE} = E / (E + D)$$

其中E是软件交付给最终用户之前发现的错误数，D是软件交付之后发现的缺陷数。



如果从分析阶段进入到设计阶段时，DRE的值较低，那么就要花些时间去改进正式技术评审的方式。

最理想的DRE的值是1，即在软件中没有发现缺陷。实际上，D的值大于0，但DRE仍可能接近于1。随着E的增加（对于给定的D值），整个DRE的值就开始接近1。事实上，随着E的增加，D的最终值将会降低（错误在变成缺陷之前已经被滤除了）。如果将DRE作为一个度量，提供关于质量控制及质量保证活动的滤除能力的衡量指标，那么DRE就能促进软件项目团队采用先进的技术，力求在软件交付之前发现尽可能多的错误。

在项目内部，也可以使用DRE来评估一个团队在错误传递到下一个框架活动或软件工程任务之前发现错误的能力。例如，在需求分析任务中创建了一个分析模型，而且对该模型进行了评审以发现和改正其中的错误。那些在评审过程中未被发现的错误传递给了设计（在设计中它们可能被发现，也可能没被发现）。在这种情况下，我们将DRE重新定义为：

$$\text{DRE}_i = E_i / (E_i + E_{i+1})$$

其中， E_i 是在软件工程活动*i*中发现的错误数； E_{i+1} 是在软件工程活动*i+1*中发现的错误数，这些错误都是在软件工程活动*i*中没被发现的错误。

软件项目团队（或软件工程师个人）的质量目标是使DRE接近于1，即错误应该在传递到下一个活动之前被过滤掉。

SAFEHOME

建立度量方法

[场景] Doug Miller的办公室，在首次软件度量会议召开两天之后。

[人物] Doug Miller, SafeHome软件工程团队经理；Vinod Raman和Jamie Lazar, 产品软件工程团队成员。

[对话]

Doug: 关于过程度量和项目度量，你们都有所了解了吧。

Vinod和Jamie: [都点头]

Doug: 无论采用何种度量，都要建立目标，这总是正确的。那么，你们的目标是什么？

Vinod: 我们的度量应该关注于质量。实际上，我们的总体目标是使从一个软件工程活动传递给下一个软件工程活动的错误数最少。

Doug: 并且确保随同产品发布的缺陷数尽可能接近于0。

Vinod (点头): 当然。

Jamie: 我喜欢将DRE作为一个度量，我认为我们可以将DRE用于整个项目的度量。同样，在从一个框架活动转到下一个框架活动时，也可以使用它。DRE促使我们在每一步都去发现错误。

Vinod: 我觉得还要收集我们花费在评审上的小时数。

Jamie: 还有我们花费在每个软件工程任务上的总工作量。

Doug: 可以计算出评审和开发的比率……可能很有趣。

Jamie: 我还想跟踪一些用例方面的数据，如：建立一个用例所需的工作量，构造软件来实现一个用例所需的工作量，以及……

Doug (微笑): 我想我们要保持简单。

Vinod: 应该这样，不过你一旦深入到度量中，就可以看到很多有趣的事。

Doug: 同意，但在我们会跑之前要先走，坚持我们的目标。收集的数据限制在5~6项，准备去做吧。

22.4 在软件过程中集成度量

大多数软件开发者还没有进行测量，更可悲的是，他们中的大多数根本没有开始测量的愿望。正如我们在本章开始所提到的，这是文化的问题。试图收集过去从来没有人收集过的测量数据常常会遇到阻力。备受折磨的项目经理会问“为什么我们要做这个”。超负荷工作的开发者会抱怨“我看不出这样做有什么用”。

在本节中，我们要考虑一些有关软件度量的观点，并给出在软件工程组织内部制定度量收集计划的方法。不过，在开始前，我们先来看看Grady和Casewell[GRA87]所说的充满智慧的话：

664

我们在这里描述的一些事情听起来似乎相当容易。但实际上，成功地制定全公司范围内的软件度量计划是很困难的工作。如果我们说，你必须至少等上3年才能在组织内形成显著的趋势，你就可以对这一工作量的规模有个概念了。

作者给出的告诫值得很好地借鉴。但是，测量的作用是如此显著，再艰苦的工作也是值得做的。

22.4.1 支持软件度量的论点

测量软件工程过程及其生产出来的产品（软件）为什么这么重要？答案其实很明显。如果不进行测量，就无法确定我们是否在改进。如果我们没有在改进，就会导致失败。

通过对生产率测量和质量测量提出请求，并进行评估，软件团队（及其管理者）能够建立改进软件工程过程的有意义的目标。在第1章中我们提到：对于许多公司而言，软件都是一个战略性的商业问题。如果软件开发过程能够得到改进，对最终结果（bottom line）将产生直接的影响。而要建立改进目标，就必须了解软件开发的当前状态。因此，可以使用测量来建立过程的基线，并根据此基线来评估改进。

“在我们生活的很多方面，我们都是通过“数字”来管理事物……这些数字使我们对事物有了深入的了解，并有助于指导我们的行动。” ——Michael Mah和Larry Putnam

每天繁重的软件项目工作使得人们几乎没有时间进行战略性的思考。软件项目经理更关心现实的问题（当然这也同样重要）：例如，建立有意义的项目估算、开发高质量的系统、按期交付产品等等。通过使用测量来建立项目基线，将使这些问题变得更容易管理。我们已经知道，基线是估算的基础。此外，质量度量的收集可以使一个组织“调整”其软件工程过程，以消除那些对软件开发有重大影响的缺陷产生的根源¹⁰。

22.4.2 建立基线

 什么是度量基线？它能为软件工程师提供什么益处？

665

通过建立度量基线，在过程级、项目级和产品（技术）级上都能获得收益。而要收集的信息并非完全不同，相同的度量可以用于多个方面。度量基线由从以往开发的软件项目中收集的数据构成，它可能像图22-2所示的表格那样简单；也可能像一个综合数据库那样复杂，其中含有几十个项目测量数据及从中导出的度量。

为了有效地协助进行过程改进和（或）成本及工作量估算，基线数据必须具有下列属性：（1）数据必须相当精确，要避免对过去项目进行“推测”；（2）应该从尽可能多的项目中收集数据；（3）测量数据必须是一致的，例如，在收集数据涉及的所有项目中对代码行的解释必须是一致的；（4）基线数据所在的应用系统应该与将要做估算的工作类似，如果把一个适用于批处理系统工作的基线用于估算实时嵌入式应用系统就没有什么意义。

22.4.3 度量收集、计算和评估

建立度量基线的过程如图22-3所示。理想情况下，建立基线所需的数据已经在项目开发

¹⁰ 这些想法已经被规范化为一种方法，称为统计软件质量保证，将在第26章中详细讨论。

KEY POINT

基线度量数据应该从以往有代表性的大量的软件项目样本中收集。

过程中收集了。但遗憾的是，很少有这样做的。因此，在收集数据时，需要对以往的项目做历史调查，以重建所需的数据。一旦收集好了测量数据（无疑这是最困难的一步），就可以进行度量计算了。依赖于所收集的测量数据的广度，度量可以涵盖众多面向应用的度量（如，LOC，FP，面向对象，WebApp）以及其他面向质量和面向项目的度量。最后，度量还要在估算、技术工作、项目控制及过程改进中加以评估和使用。度量评估主要是分析结果产生的根本原因，并生成一组指导项目或过程的指标。

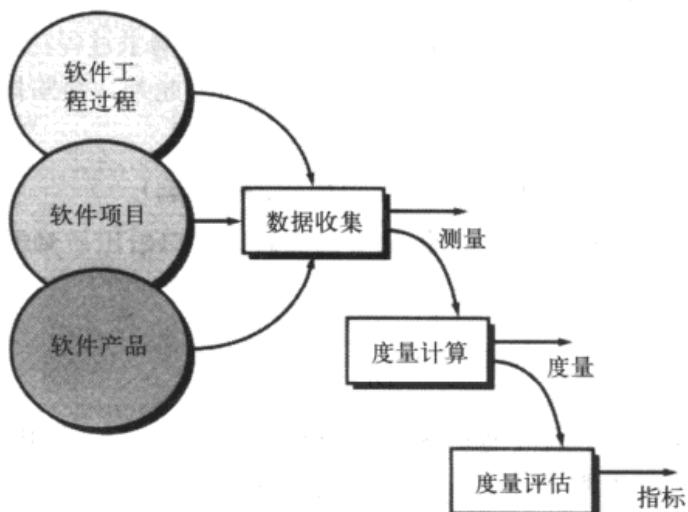


图22-3 软件度量的收集过程

22.5 小型组织的度量

ADVICE

如果你正要开始收集度量数据，记住要保持简单性。如果将自己淹没于数据之中，那么，你的度量工作将注定要失败。

我们应该怎样导出一组“简单的”软件度量？

在绝大多数软件开发组织中，软件人员都不到20人。期望这样的组织能制定出全面的软件度量大纲是不合理的，在多数情况下也是不现实的。然而，建议各种规模的软件组织都进行测量，然后使用从中导出的度量来帮助他们改进其软件过程，提高所开发产品的质量，缩短开发时间，这样的要求是合理的。

要完成任何软件过程相关的活动，一种常识性方法是：保持简单，根据项目需要对它进行定制，确保它带来增值。在下面的段落中，我们将考察如何将这些指导原则应用在小型组织的度量中¹¹。

“保持简单”是一个在很多活动中都相当奏效的原则。但是，我们如何能够得到一组“简单的”又有价值的软件度量？我们如何确定这些简单的度量能否满足一个特殊软件组织的需要？我们不是从测量而是从结果入手，软件小组通过表决来确定一个需要改进的目标，例如，“减少评估和实现变更请求的时间”。根据这个目标，小型组织可以选择下列易于收集的测量：

- 从提出请求到评估完成所用的时间（小时或天）， t_{queue} 。
- 进行评估所用的工作量（人·小时）， W_{eval} 。

¹¹ 对于已经采用了敏捷软件开发过程（第4章）的团队来说，这项讨论具有同样的指导意义。

- 从完成评估到把变更工单派发到员工所用的时间（小时或天）， t_{eval} 。
- 实现变更所需的工作量（人·小时）， W_{change} 。
- 实现变更所需的时间（小时或天）， t_{change} 。
- 在实现变更过程中发现的错误数， E_{change} 。
- 将变更发布给客户后发现的缺陷数， D_{change} 。

一旦从大量变更请求中收集到了这些测量数据，就能计算出从变更请求到变更实现所用的总平均时间，以及初始排队、评估、派发变更和实现变更所占用时间的百分比。类似地，还可以计算出评估和实现变更所需工作量的百分比。这些度量也可以根据质量数据 E_{change} 和 D_{change} 进行评估。从这些百分比数据还可以清楚地看出变更请求过程在什么地方延迟了，从而进行过程改进，以减少 t_{queue} 、 W_{eval} 、 t_{eval} 、 W_{change} 和 E_{change} 。此外，缺陷排除效率又可以用以下公式计算：

$$DRE = E_{change} / (E_{change} + D_{change})$$

将DRE同变更所花费的时间和总工作量进行比较，可以看出质量保证活动对变更所需时间和工作量的影响。
667

22.6 制定软件度量大纲

美国卡内基·梅隆大学软件工程研究所（SEI）已经开发了一套用于制定“目标驱动的”软件度量大纲的综合指导手册[PAR96]。手册中给出了以下步骤：

1. 明确你的业务目标。
2. 弄清你要了解或学习的内容。
3. 确定你的子目标。
4. 确定与子目标相关的实体和属性。
5. 确定你的测量目标。
6. 识别可量化的问题和相关的指标，你将使用它们帮助你达到测量的目标。
7. 明确你要收集的数据元素，从这些数据元素中要得到帮助你回答问题的指标。
8. 定义将要使用的测量，并使这些定义具有可操作性。
9. 弄清楚实现测量需要做的操作。
10. 准备一份实施测量的计划。

关于这些步骤的详细讨论最好参见SEI的手册。不过，有必要概述一下关键问题。



因为软件支持业务功能，它划分为基于计算机的系统或产品，或者本身就是产品，因此针对业务所定义的目标几乎总是可以向下追溯到软件工程层次上的特定目标。例如，有这样一个公司，它生产高级的家庭安全系统，该系统含有重要的软件内容。软件工程师和业务管理者协同工作，制定出了一组按优先级排列的业务目标：

1. 提高客户对产品的满意度。
2. 使产品易于使用。
3. 缩短将新产品推向市场的时间。

4. 使对产品的支持更容易。
5. 提高整体收益率。

软件组织审查每个业务目标并提出问题：“我们管理或执行什么活动？在这些活动中我们要改进什么？”为了回答这些问题，SEI建议创建一个“实体－问题表”。在这个表中，列出所有在软件过程中受软件组织管理或影响的事物（实体）。实体的例子包括开发资源、工作产品、源代码、测试用例、变更请求、软件工程任务及进度安排。对列出的每个实体，软件人员都要提出一组评估其定量特征（如大小、成本、开发时间）的问题。创建实体－问题表引出了这些问题，从而又导出了一组子目标，这些子目标与已创建的实体和已完成的部分软件过程活动紧密相关。

考虑第四个目标：“使对产品的支持更容易。”由这个目标，可以引出下列问题[PAR96]：

- 客户的变更请求中包含及时评估变更并实现变更所需要的信息吗？
- 积压的变更请求有多少？
- 根据客户的需要，我们修正缺陷的响应时间能接受吗？
- 能遵循变更控制过程（第27章）吗？
- 优先级高的变更能及时实现吗？

根据这些问题，软件组织可以导出下面的子目标：改进变更管理过程的效能。然后，确定与该子目标相关的软件过程实体和属性，明确与这些实体和属性相关的测量目标。

SEI[PAR96]对目标驱动测量方法中的步骤6~10，给出了详细的指导。本质上是采用逐步求精的过程，将目标精化为问题，这些问题进一步精化为实体和属性，然后将这些实体和属性精化为度量。

INFO

制定度量大纲

Software Productivity Center (www.spc.ca) 为软件组织提供了一个制定内部度量大纲的方法，称为八步法，除了22.6节中提到的SEI方法，这也是可选用的一种方法。这里对该方法做一个概要介绍。

1. 理解现有的软件过程。
标识框架活动（第2章）。
描述每项活动的输入信息。
定义每项活动的相关任务。
注明质量保证功能。
列出生产的工作产品。
2. 确定通过制定度量大纲要达到什么目标。
例如，提高估算的精确性，改善产品的质量。
3. 确定为实现目标需要哪些度量。
定义要回答的问题，例如，在一个框架活动中发现的错误有多少个可以追踪到前一个框架活动。
提出要进行收集和计算的测量和度量。
4. 标识这些要进行收集和计算的测量和度量。

5. 通过回答这些问题，建立测量收集过程。

测量的来源是什么？

可以使用工具收集数据吗？

由谁负责收集数据？

什么时候收集和记录数据？

如何存储数据？

采用什么确认机制来保证数据的正确性？

6. 选用适当的工具帮助收集和评估这些数据。

7. 建立度量数据库。

建立比较可靠的数据库。

研究相关工具的使用（如，SCM中心存储库，第27章）。

评价现有的数据库产品。

8. 定义适当的反馈机制

谁需要即时的度量信息？

如何导出这些信息？

这些信息采用什么格式？

关于这8个步骤的详细描述可以从网站下载，网址是<http://www.spc.ca/resources/metrics>。

669

22.7 小结

测量能使管理者和开发者改进软件过程，辅助进行软件项目的计划、跟踪及控制，评估生成的产品（软件）的质量。对过程、项目及产品的特定属性的测量可用来计算软件度量。分析这些度量可以获得指导管理及技术行为的指标。

过程度量能够使一个组织从战略角度深入了解一个软件过程的功效。项目度量是战术性的，能使项目管理者实时改进项目的工作流程及技术方法。

面向规模的度量和面向功能的度量在业界都得到了广泛的应用。面向规模的度量以代码行作为其他测量（如人·月，缺陷）的规范化因子。功能点则是从信息域测量及对问题复杂度的主观评估中导出的。

软件质量度量（如生产率度量）关注的是过程、项目和产品。一个组织通过建立并分析质量度量基线，能够纠正那些引起软件缺陷的软件过程区域。

测量会带来企业文化的改变。如果开始进行度量，则数据收集、度量计算和度量分析是必须完成的三个步骤。通常，目标驱动的方法有助于一个组织关注于对其业务的正确度量。通过建立度量基线——一个包含过程和产品测量的数据库，软件工程师及其管理者能够更好地了解他们所做的工作和开发的产品。

参考文献

[ALB83] Albrecht, A. J., and J. E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Engineering*, November 1983, pp. 639-648.

[BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981.

670

- [GRA87] Grady, R. B., and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
- [GRA92] Grady, R. G., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, 1992.
- [GIL88] Gilb, T., *Principles of Software Project Management*, Addison-Wesley, 1988.
- [HET93] Hetzel, W., *Making Software Measurement Work*, QED Publishing Group, 1993.
- [HUM95] Humphrey, W., *A Discipline for Software Engineering*, Addison-Wesley, 1995.
- [IEE93] IEEE Software Engineering Standards, Standard 610.12-1990, pp. 47-48.
- [JON86] Jones, C., *Programming Productivity*, McGraw-Hill, 1986.
- [JON91] Jones, C., *Applied Software Measurement*, McGraw-Hill, 1991.
- [JON98] Jones, C., *Estimating Software Costs*, McGraw-Hill, 1998.
- [LOR94] Lorenz, M., and J. Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, 1994.
- [PAR96] Park, R. E., W. B. Goethert, and W. A. Florac, *Goal Driven Software Measurement—A Guidebook*, CMU/SEI-96-BH-002, Software Engineering Institute, Carnegie Mellon University, August 1996.
- [PAU94] Paulish, D., and A. Carleton, "Case Studies of Software Process Improvement Measurement," *Computer*, vol. 27, no. 9, September 1994, pp. 50-57.
- [QSM02] "QSM Function Point Language Gearing Factors," Version 2.0, Quantitative Software Management, 2002, <http://www.qsm.com/FPGeering.html>.
- [RAG95] Ragland, B., "Measure, Metric or Indicator: What's the Difference?" *Crosstalk*, vol. 8, no. 3, March 1995, p. 29-30.
- [SMI99] Smith, J., "The Estimation of Effort Based on Use-Cases," a white paper by Rational Corporation, 1999, downloaded from <http://www.rational.com/products/rup/whitepapers.jsp>.

习题与思考题

- 22.1 使用22.2.3节中列出的表格，基于每行代码具有的功能性，提出一个反对使用汇编语言的论据。再参考该表，讨论为什么C++比C更好？
- 22.2 为什么有些软件度量是“私有的”？给出三个私有度量的例子，并给出三个公用度量的例子。
- 22.3 用自己的话描述过程度量和项目度量之间的区别。
- 22.4 根据下面的信息域特性，计算项目的功能点值：
- 外部输入数：32
- 外部输出数：60
- 外部查询数：24
- 内部逻辑文件数：8
- 外部接口文件数：2
- 假定所有的复杂度校正值都取“平均”值。使用第15章提到的算法。
- 22.5 给出一个反对将代码行作为软件生产率度量的论据。当考虑几十个或几百个项目时，你说的情况还成立吗？
- 22.6 产品交付之前，团队A在软件工程过程中发现了342个错误，团队B发现了184个错误。对于项目A和B，还需要做什么额外的测量，才能确定哪个团队能够更有效地排除错误？你建议采用什么度量来帮助做决定？哪些历史数据可能有用？
- 22.7 Grady提出了一组软件度量规则，你能在22.1.1节所列的规则中再增加三个规则吗？
- 22.8 什么是间接测量？为什么在软件度量工作中经常用到这类测量？
- 22.9 软件团队将软件增量交付给了最终用户。在使用的第一个月中，用户发现了8个缺陷。在交付之前，软件团队在正式技术评审和所有的测试任务中发现了242个错误。那么，

项目的总的缺陷排除效率（DRE）是多少？

- 22.10 一个Web工程团队已经开发了一个包含145个网页的电子商务WebApp。在这些页面中，65个是动态页面，即根据最终用户的输入而在内部生成的页面。那么，该应用的定制指数是多少？
- 22.11 用于控制影印机的软件需要32 000行C语言代码和4200行Smalltalk语言代码。估算该影印机软件的功能点数。
- 22.12 在一个使用统一过程（第3章）的项目结束时，已经确定在细化阶段发现了30个错误，在构建阶段发现了12个可以追溯到细化阶段的没有发现的错误。那么，这两个阶段的DRE是多少？
- 22.13 一个WebApp及其支持环境没有被充分地加强以抵御攻击。Web工程师估计击退攻击的概率只有30%。系统不包含机密或有争议的信息，因此危险性概率只有25%。那么，该WebApp的完整性是多少？

推荐读物与阅读信息

在过去的20年中，软件过程改进（SPI）得到了极大的关注。因为测量和软件度量是成功改进软件过程的关键，所以在很多SPI方面的书籍中也讨论度量。关于过程度量的有价值的信息源包括：

- Burr, A., and M. Owen, *Statistical Methods for Software Quality*, International Thomson Publishing, 1996.
- El Emam, K., and N. Madhvji (eds.), *Elements of Software Process Assessment and Improvement*, IEEE Computer Society, 1999.
- Florac, W. A., and A. D. Carlton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison-Wesley, 1999.
- Garmus, D., and D. Herron, *Measuring the Software Process: A Practical Guide to Functional Measurements*, Prentice-Hall, 1996.
- Humphrey, W., *Introduction to the Team Software Process*, Addison-Wesley/Longman, 2000.
- Kan, S. H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995.

McGarry及其同事（《Practical Software Measurement》，Addison-Wesley，2001）对评估软件过程提出了深层次的建议。Haug及其同事已经编写了一部值得收藏的著作（《Software Process Improve: Metrics, Measurement, and Process Modeling》，Springer-Verlag，2001）。Florac和Carlton（《Software Metrics: A Rigorous and Practical Approach》，Revised，Brooks/Cole Publishers，1998）探讨了如何利用软件度量取得改进软件过程的必要指标。

Putnam和Myers（《Five Core Metrics》，Dorset House，2003）利用一个包含6000多软件项目的数据库论证了五种核心度量——时间、工作量、规模、可靠性以及过程生产率，可以用于控制软件项目。Maxwell（《Applied Statistics for Software Managers》，Prentice-Hall，2003）给出了分析软件项目数据的技术。Munson（《Software Engineering Measurement》，Auerbach，2003）讨论了大量的软件工程测量问题。Jones《Software Assessments, Benchmarks and Best Practices》，Addison-Wesley，2000）描述了定量的测量以及定性的测量因素，帮助组织评估自己的软件过程和实践。Garmus和Herron（《Function Point Analysis: Measurement Practices for Successful Software Projects》，Addison-Wesley，2000）讨论了侧重

672

于功能点分析的过程度量。

Lorenze、Kidd[LOR94]和DeChampeax (《Object-Oriented Development Process and Metrics》, Prentice-Hall, 1996) 考虑了OO过程，并描述了用于评估OO过程的一组度量。Whitmire (《Object-Oriented Design Measurement》, Wiley, 1995) 和Henderson-Sellers (《Object-Oriented Metrics: Measures of Complexity》, Prentice-Hall, 1995) 关注于用在OO工作上的技术度量，还考虑了能够用于过程级和产品级的测量和度量。

关于Web工程工作的度量，已发表的资料比较少。不过，Stern (《Web Metrics: Proven Methods for Measuring Web Site Success》, Wiley, 2002)、Inan和Kean (《Measuring the Success of Your Website》, Longman, 2002)，以及Nobles和Grady (《Web Site Analysis and Reporting》, Premier Press, 2001) 从企业和市场角度讨论了Web度量。

673

IEEE总结了度量领域最新的研究 (《Symposium on Software Metrics》，每年出版)。大量的关于过程度量和项目度量的信息源可以在因特网上获得。一些最新的WWW参考文献可在SEPA Web站点<http://www.mhhe.com/pressman>上找到。

第23章 估 算

要点浏览

概念：软件的真实需求已经确定；共利益者们都已就绪；软件工程师准备开始；项目将要启动。但是如何进行下去呢？软件项目计划包括五项主要活动——估算、进度安排、风险分析、质量管理计划和变更管理计划。在本章中，我们只考虑估算——尝试确定构造一个特定的基于软件的系统或产品所需要花费的资金、工作量、资源及时间。

人员：软件项目经理——利用从共利益者和软件工程师那里获得的信息以及从以往项目收集的软件度量数据。

重要性：你会在不知道要花多少钱、要完成多少任务以及完成工作需要多少时间的情况下建造房子吗？当然不会。既然大多数基于计算机的系统和产品的成

本大大超出建造一所大房子，那么在开发软件之前进行估算应该是合理的。

步骤：估算首先要描述产品的范围。然后，将问题分解为一组较小的问题，再以历史数据和经验为指南，对每个小问题进行估算。在进行最终的估算之前，要考虑问题的复杂度和风险。

工作产品：生成一个简单的表，描述要完成的任务、要实现的功能，以及完成每一项所需的成本、工作量和时间。

质量保证措施：这很困难。因为要等到项目完成时，你才能真正知道。不过，如果你有经验并遵循系统化的方法，使用可靠的历史数据进行估算，利用至少两种不同的方法创建估算数据点，制定现实的进度表并随着项目的进展不断进行调整，那么你就可以确信你已经为项目做了最好的估算。

关键概念
复杂性
估算
概念
基于FP的
基于LOC的
基于过程的
调和
用例
可行性
项目策划
资源
范围
软件规模

软件项目管理从一组统称为项目策划的活动开始。在项目可以开始前，项目经理和软件团队必须估算将要完成的工作、所需的资源，以及从开始到完成所需要的时间。这些活动一旦完成，软件团队就要制定项目进度计划。在项目进度计划中，要定义软件工程任务及里程碑，确定每一项任务的负责人，详细指明对项目进展影响很大的任务间的相互依赖关系。

在一本关于“软件项目生存”的优秀指南中，Steve McConnell[MCC98]讲述了人们对项目计划的实际看法：

很多技术工作者宁愿从事技术工作，而不愿花费时间制定计划。很多技术管理者没有接受过充分的技术管理方面的培训，对他们的计划能够改善项目成果缺乏信心。既然这两部分人都不想制定计划，因此就经常不制定计划。

但是没有很好地制定计划是一个项目犯的最严重的错误之一……有效的计划是必需的，可以在上游[在项目早期]以较低的成本解决问题，而不是在下游[在项目后期]以较高成本解决问题。一般的项目要将80%的时间花费在返工上——改正在项目早期所犯的错误。

McConnell指出，在每个项目中都可以找出制定计划的时间（并使计划适应于整个项目），只要从因为没制定计划而出现的返工时间中抽出一小部分时间即可。

23.1 对估算的观察

制定计划需要技术管理者和软件团队成员做一个初始约定，即使这个“约定”很可能将被证明是错误的。无论在什么时候进行估算，我们都是在预测未来，自然要接受一定程度的不确定性。下面我们引用Frederick Brooks [BRO75] 的话：

我们的估算技术发展缓慢。更为严重的是，它们隐含了一个很不正确的假设，即“一切都会好的”……因为对自己的估算没有把握，软件管理者常常缺乏让人们得到一个好产品的信心。

估算是一门艺术，更是一门科学，这项重要的活动不能以随意的方式来进行。现在已经有了估算时间和工作量的实用技术。过程度量和项目度量为定量估算从历史角度提供了依据和有效的输入。当建立估算和评审估算时，过去经验（包括所有参与人员的）的辅助作用是不可估量的。由于估算的所有其他项目策划活动的基础，而且项目计划又提供了通往成功的软件工程的路线图。因此，没有估算就着手开发，我们将陷入盲目性。

“好的估算方法和可靠的历史数据提供了最好的希望：现实将战胜不可能的要求。”

——Capers Jones

对软件工程工作的资源、成本及进度进行估算时，需要经验，需要了解有用的历史信息，还要有当只存在定性的信息时进行定量预言的勇气。估算具有与生俱来的风险¹，正是这种风险导致了不确定性。

历史信息的有效性对估算的风险有很大影响。通过回顾过去，我们能够仿效做过的工作，并改进出现问题的地方。如果能取得对以往项目的全面的软件度量（第22章），做估算就会有更大的保证，合理安排进度以避免重走过去的弯路，总体风险也会降低。

“应该满足于事物本性所能容许的精确度，当只可能近似于真理时，不要去寻求绝对的准确。”

——Aristotle

估算的风险取决于对资源、成本及进度的定量估算中存在的不确定性。如果对项目范围不太了解，或者项目需求经常改变，不确定性和估算风险就会非常高。计划人员，尤其是客户，都应该认识到经常改变软件需求意味着在成本和进度上的不稳定性。

不过，项目管理者不应该被估算所困扰。现代软件工程方法（如，增量过程模型）采用迭代方法开发。在这类方法中，当客户改变需求时，应该能够（尽管并不总是易于接受）重新审查估算（在了解更多信息后），并进行修正。

23.2 项目策划过程

软件项目策划的目标是提供一个能使管理人员对资源、成本及进度做出合理估算的框架。

¹ 系统的风险分析技术将在第25章中讨论。



你了解得越多，就估算得越好。因此，要随着项目的进展对你的估算进行更新。

此外，估算应该尝试定义“最好的情况”和“最坏的情况”，使项目的结果能够限制在一定范围内。项目计划是在计划任务中创建的，尽管它具有与生俱来的不确定性，软件团队还是要根据它着手开发。因此，随着项目的进展，必须不断地对计划进行调整和更新。在下面几节中，将讨论与软件项目策划有关的每一项活动。

TASK SET

项目策划任务集

1. 规定项目范围
2. 确定可行性
3. 分析风险（第25章）
4. 确定需要的资源
 - a. 确定需要的人力资源
 - b. 确定可复用的软件资源
 - c. 标识环境资源
5. 估算成本和工作量
 - a. 分解问题
 - b. 使用规模、功能点、过程任务或用例等方法进行两种以上的估算
 - c. 调和不同的估算
6. 制定项目进度计划（第24章）
 - a. 建立一组有意义的任务集合
 - b. 定义任务网络
 - c. 使用进度计划工具制定时间表
 - d. 定义进度跟踪机制

676

23.3 软件范围和可行性



尽管导致不确定性的原因有很多，但问题需求信息的不完整性是最主要的原因。

软件范围描述了将要交付给最终用户的功能和特性、输入和输出的数据、使用软件时要呈现给用户的“内容”，以及界定系统的性能、约束条件、接口和可靠性。定义范围可以使用两种方法：

1. 在与所有共利益者交流之后，写出对软件范围的叙述性描述。
2. 由最终用户开发一组用例²。

在开始估算之前，首先要对范围陈述（或用例）中描述的功能进行评估，在某些情况下，还要进行细化，以提供更多的细节。由于成本和进度的估算都是面向功能的，因此某种程度上的功能分解常常是有用的。性能方面的考虑包括处理时间和响应时间的需求。约束条件则标识了外部硬件、可用存储，或其他现有系统对软件的限制。

² 在本书的第二、三部分中，已经对用例进行了详细的讨论。用例是从用户的角度出发来描述用户与软件的交互场景。



项目可行性很重要，但是考虑商业需要更重要。构造一个没人想要的高科技系统或产品根本没有意义。

一旦确定了软件范围（并取得了用户的同意），人们自然会问：我们能够开发出满足范围要求的软件吗？这个项目可行吗？软件工程师常常匆忙越过这些问题（或由于不耐烦的管理者或客户而被迫越过这些问题），不料竟会一开始就注定要陷入这个项目的泥潭中。Putnam和Myers[PUT97a]是这样阐述这个问题的：

并非每件想象的事情都是可行的，即便是软件也是如此，在外行人看来，它是捉摸不定的。其实，软件可行性有四个固定的因素：**技术**——项目在技术上是可行的吗？它在技术水平范围内吗？能够将缺陷减少到一定程度而满足应用的要求吗？**经济**——它在经济上可行吗？能以开发组织、客户或市场负担得起的成本完成开发吗？**时间**——项目投入市场的时间可以击败竞争者吗？**资源**——组织拥有取得成功所需要的资源吗？

23.4 资源

项目策划的第二个任务是对完成软件开发工作所需的资源进行估算。图23-1描述了三类主要的软件工程资源——人员、可复用的软件构件及开发环境（硬件工具及软件工具）。对每一类资源，都要说明以下四个特征：资源的描述、可用性说明、何时需要资源、使用资源的持续时间。最后两个特性可以看成是时间窗口。对于一个特定的时间窗口，必须在开发初期就建立资源的可用性。

677

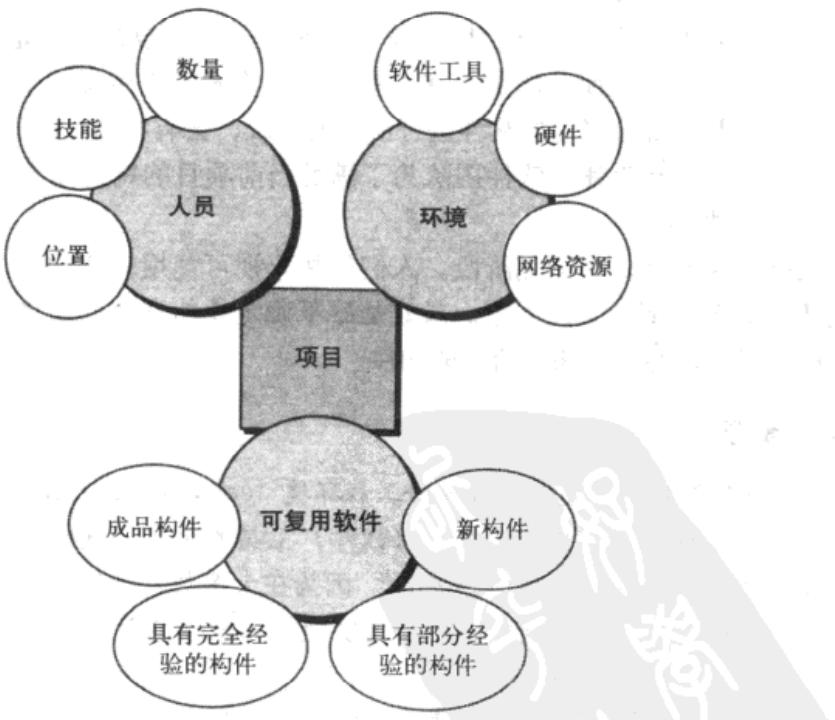


图23-1 项目资源

23.4.1 人力资源

计划人员首先评估软件范围，并选择完成开发所需的技能，还要指定组织中的职位（如

管理人员、高级软件工程师)和专业(如,电信、数据库、客户/服务器系统)。对于一些比较小的项目(几个人·月),只要向专家做些咨询,也许一个人就可以完成所有的软件工程任务。而对于一些较大的项目,软件团队的成员可能分散在很多不同的地方,因此,要详细说明每个人所处的位置。

只有在估算出开发工作量(如,多少人·月)后,才能确定软件项目需要的人员数量。估算工作量的技术将在本章后面讨论。

23.4.2 可复用软件资源

基于构件的软件工程(第30章)强调可复用性,即创建并复用软件构造块[HOO91],这种构造块通常被称为构件。为了容易引用,必须对这些构件进行分类;为了容易应用,必须使这些构件标准化;为了容易集成,必须对这些构件进行确认。
678

Bennatan[BEN92]建议在制定计划时应该考虑以下四种软件资源。



绝对不要忘记:
多种可复用构件
的集成相当具有
挑战性。当更新
多种构件时,集
成问题常常会重
新出现。

成品构件:能够从第三方获得,或在以前的项目中已经进行过内部开发的已有软件。商业成品构件(*commercial off-the-shelf, COTS*)是从第三方购买的,可以直接用于当前的项目,这些构件都已经过完全确认。

具有完全经验的构件:为以前项目开发的,与当前项目要构造的软件相似的已有的规格说明、设计、代码或测试数据。当前软件团队的成员在这些构件所代表的应用领域中具有丰富的经验。因此,对于具有完全经验的构件进行所需的修改,风险相对较小。

具有部分经验的构件:为以前项目开发的,与当前项目要构造的软件有关的已有的规格说明、设计、代码或测试数据,但是需要做实质上的修改。当前软件团队的成员在这些构件所代表的应用领域中经验较少。因此,对于具有部分经验的构件进行所需的修改,会有相当大的风险。

新构件:软件团队为了满足当前项目的特定需要,而必须专门开发的软件构件。

具有讽刺意味的是,在策划阶段,人们往往忽视可复用软件构件,直到软件过程的开发阶段,它才变成最重要的关注对象。最好是尽早确定软件资源的需求,这样才能对各种候选方案进行技术评估,并及时获取所需的构件。

23.4.3 环境资源

支持软件项目的环境,通常称为软件工程环境(*software engineering environment, SEE*),它集成了硬件和软件。硬件提供了支持(软件)工具的平台,这些(软件)工具是采用良好的软件工程实践来获得工作产品所必需的³。因为在大多数软件组织中,有很多人都需要使用SEE,因此,项目计划人员必须详细规定需要硬件和软件的时间窗口,并且验证这些资源是可用的。

当开发一个基于计算机的系统(集成特定的硬件及软件)时,软件团队可能需要用到由其他工程团队开发的硬件元件。例如,在一种机械工具上使用的数控(*numerical control, NC*)软件可能需要特定的机床(如数控机床)作为确认测试步骤的一部分;在开发高级排版软件

³ 其他硬件(目标环境)是指在软件交付给最终用户之后,将要运行该软件的计算机。

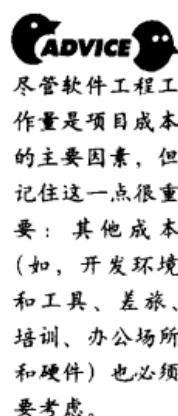
项目的过程中，在某个阶段可能需要一台高质量的印刷机。软件项目计划人员必须指定每一个硬件元素。

23.5 软件项目估算

几乎在所有基于计算机的系统中，软件都是最昂贵的成分。对于复杂的定制的系统，如果成本估算误差很大，就会使赢利变成亏损。对于开发者来说，成本超支可能是灾难性的。

“在外包和竞争更加激烈的时代，更准确地进行估算的能力……已经成为很多IT组织关键的成功因素。”

——Rob Thomsett



软件成本和工作量的估算从来都没有成为一门精确的科学⁴，因为变化的因素太多——人员、技术、环境和行政，都会影响软件的最终成本和开发所用的工作量。不过，软件项目估算还是能够从一种“神秘的”技巧变成一系列系统化的步骤，在可接受的风险范围内提供估算结果。

为得到可靠的成本和工作量估算，有很多选择：

1. 把估算推迟到项目的后期进行（显然，在项目完成之后就能得到100%精确的估算）。
2. 根据已经完成的类似项目进行估算。
3. 使用比较简单的分解技术，生成项目的成本和工作量估算。
4. 使用一个或多个经验模型来进行软件成本和工作量的估算。

遗憾的是，第一种选择，不管它有多吸引人，都是不现实的。成本估算必须“预先”给出。然而，我们应该认识到，我们等的时间越久，了解得就越多，而我们了解得越多，估算中出现的严重误差就越少。

如果当前项目与以前的工作非常相似，并且项目的其他影响因素（如软件团队、客户、商业条件、SEE及交付期限等）也大致相同，第二种选择就能够很好地起作用。遗憾的是，过去的经验并不总是能够指明未来的结果。

余下的两种选择对于软件项目估算也是可行的方法。理想的情况是，同时使用这两种选择所提到的技术，相互进行交叉检查。分解技术采用“分而治之”的方法进行软件项目估算，把项目分解成若干主要的功能和相关的软件工程活动，以逐步求精的方式对成本和工作量进行估算。经验估算模型可以作为分解技术的补充，在它适用的范围内，常常是一种有价值 的估算方法。这些模型将在第23.7节中讨论。

每种可行的软件成本估算方法，其效果的好坏取决于估算使用的历史数据。如果没有历史数据，成本估算也就建立在一个很不稳定的基础上。在第22章中，我们已经考察了一些软件度量的特性，这些度量提供了历史估算数据的基础。

23.6 分解技术

软件项目估算是一种解决问题的形式，在多数情况下，要解决的问题（对于软件项目来说，就是成本和工作量的估算）非常复杂，不能作为一个整体考虑。因此，我们要对问题进

680

⁴ Bennatan[BEN03] 报告说：40%的软件开发者受到估算的困扰，软件规模和开发时间都难以精确估算。

行分解，把它分解成一组较小的（同时有望更容易管理的）问题，再定义它们的特性。

在第21章中，我们从两个不同的角度讨论了分解方法：问题分解和过程分解。估算时可以使用其中一种或两种分解形式。但在进行估算之前，项目计划人员必须理解待开发软件的范围，并估计其“规模”。

23.6.1 软件规模估算

POINT
待开发软件的规模可以使用直接测量 (LOC) 或间接测量 (FP) 来估算。

软件项目估算的准确性取决于许多因素：(1) 计划人员估算待开发产品规模的正确程度；(2) 把规模估算转换成人员工作量、时间及成本的能力（受可靠软件度量的可用性的影响，这些度量数据来自以往的项目）；(3) 项目计划反映软件团队能力的程度；(4) 产品需求的稳定性和支持软件工程工作的环境。

在本节中，我们考虑软件规模的估算问题。由于项目估算的准确程度取决于待完成工作的规模估算，因此，规模估算使项目计划人员面临的主要挑战。在项目计划中，规模是指软件项目的可量化的结果。如果采用直接的方法，规模可以用代码行 (LOC) 来测量。如果选择间接的方法，规模可以用功能点 (FP) 来表示。

Putnam 和 Myers [PUT92] 建议使用四种不同的方法来估算规模问题：

我们如何
估算计划
要开发的软件
的规模。
[681]

“模糊逻辑”法。使用这种方法，计划人员必须先确定应用的类型，定性地确定其量级，然后在初始范围内再细化该量级。

功能点法。计划人员对信息域的特性（在第15章中讨论过）进行估算。

标准构件法。软件是由许多不同的“标准构件”组成的，对于某个特定的应用领域而言，这些构件是通用的。例如，一个信息系统的标准构件是子系统、模块、屏幕、报表、交互程序、批处理程序、文件、LOC 及目标级的指令。项目计划人员估算每个标准构件出现的次数，然后使用历史项目数据来确定每个标准构件交付时的规模。

修改法。当项目要使用已有的软件作为项目的一部分，并且该软件必须做某些方面的修改时，可以使用这种方法。计划人员要估算必须完成的修改的数量和类型（如，复用、增加代码、修改代码、删除代码）。

Putnam 和 Myers 建议。可以将上述每种规模估算方法所产生的结果在统计意义上结合起来，产生一个三点估算或期望值估算结果。实现方法是：先确定规模的乐观值（低）、可能值和悲观值（高），然后使用下一节中描述的式 (23-1) 将它们结合起来。

23.6.2 基于问题的估算

在第22章中，我们已经描述了LOC和FP测量，从中可以计算出生产率度量。在软件项目估算中，LOC和FP数据被用于两个方面：(1) 作为估算变量，用于度量软件中每个元素的规模；(2) 作为基线度量，这些度量数据是从以前的项目中收集起来的，将它们与估算变量联合使用，进行成本和工作量的估算。

LOC估算和FP估算两种不同的估算技术，但两者有很多共同的特性。项目计划人员从

 基于LOC和
基于FP的
估算有什么共
同点？

 当收集项目的生
产率度量时，一
定要划分项目类
型。这样才能计
算出特定领域的
平均值，从而使
估算更精确。

界定的软件范围陈述入手，根据该陈述将软件分解成一些可分别独立进行估算的功能问题。然后，估算每个功能的LOC或FP（即估算变量）。当然，计划人员也可以选择其他元素进行规模估算，如类或对象、变更、受影响的业务过程。

然后，将基线生产率度量（如，LOC/pm或FP/pm⁵）应用于适当的估算变量中，导出每个功能的成本或工作量。将所有功能的估算合并起来，即可产生整个项目的总体估算。

然而，应该注意到：对于一个组织而言，其生产率度量常常是变化的，使用单一的基线生产率度量是不可信的。一般情况下，平均的LOC/pm或FP/pm应该根据项目领域来计算。即，应该根据项目的团队规模、应用领域、复杂性以及其他相关参数对项目进行分类，然后计算出本领域的生产率平均值。当估算一个新项目时，首先应将该项目对应到某个领域中，然后，再使用恰当的领域生产率平均值对其进行估算。[682]

LOC估算技术和FP估算技术在分解应用问题时，所要求的详细程度及划分目标有所不同。当LOC用作估算变量时，分解是绝对必要的，而且常常要达到非常详细的程度。分解的程度越高，就越有可能得到非常精确的LOC估算。

 对于FP估算，
分解时着重于信
息域特性。

 我们如何计
算软件规模
的期望值？

对于FP估算，分解则是不同的。它并不关注于功能，而是要估算五个信息域的特性，以及在第15章已经讨论过的14种复杂度校正值。然后，利用这些估算结果导出FP值，该值可与过去的数据结合，来产生估算。

不管使用哪一种估算变量，项目计划人员都要首先为每个功能或每个信息域值确定一个估算值的范围。利用历史数据或凭直觉（当其他方法都失效时），计划人员为每个功能或每个信息域的计数值都分别估算出一个乐观的、可能的和悲观的规模值。当确定了值的范围后，就得到了一个不确定程度的隐含指标。

接着，计算三点（估算值）或期望值。可以通过乐观值(S_{opt})、可能值(S_m)和悲观值(S_{pess})估算的加权平均值来计算估算变量（规模）的期望值 S ：

$$S = (S_{opt} + 4S_m + S_{pess}) / 6 \quad (23-1)$$

其中，“可能”估算值的权重最大，并遵循 β 概率分布。我们假定实际的规模结果落在乐观值与悲观值范围之外的概率很小。

一旦确定了估算变量的期望值，就可以应用历史的LOC或FP生产率数据。这种估算正确吗？对于这个问题唯一合理的答案就是：“我们不能保证”。任何估算技术，不管它有多先进，都必须与其他方法进行交叉检查。尽管如此，常识和经验一定会占优势。

23.6.3 基于LOC估算的实例

作为LOC和FP基于问题估算技术的实例，我们考察一个为机械零件计算机辅助设计(CAD)应用而开发的软件包。该软件将要运行在一个工程工作站上，且必须与各种外部设备有接口，包括鼠标、数字化仪、高分辨率彩色显示器和激光打印机。可以建立一个初步的软

⁵ 缩写pm代表工作量单位：人·月(person-month)。

件范围陈述：

683 机械CAD软件接受工程师输入的二维或三维几何数据。工程师通过用户界面与CAD系统进行交互并控制它，该用户界面应表现出良好的人机界面设计特征。所有的几何数据和其他支持信息都保存在一个CAD数据库中。要开发一些设计分析模块，以产生所需的输出，这些输出将要显示在各种不同的图形设备上。软件必须设计成能够控制外部设备，并能与外部设备（包括鼠标、数字化仪、激光打印机和绘图机）进行交互。



很多现代应用或者驻留在网络上，或者是客户/服务器体系结构的一部分。因此，要确信你的估算包含了开发“底层软件”所需的工作量。



不要屈服于诱惑而使用这一结果作为你的项目估算结果。应该再使用其他方法导出另一个结果来。

上述关于范围的陈述是初步的——它没有规定边界。必须对每个句子进行补充说明，提供具体的细节及定量的边界。例如，在开始估算之前，计划人员必须要确定“良好的人机界面设计特征”是什么含义，或“CAD数据库”的规模和复杂度是怎样的。

为了进行估算，我们假定已经做了进一步的细化，并确定了该软件包应具有的主要的软件功能，如图23-2中所示。遵照LOC的分解技术，能够得到如图23-2所示的估算表，表中给出了每个功能的LOC估算范围。例如，三维几何分析功能的LOC估算范围是：乐观值——4600，可能值——6900，悲观值——8600。应用式(23-1)，得到三维几何分析功能的期望值是6800LOC。通过类似的方法也可以得到其他估算。对LOC估算这一列求和，就得到了该CAD系统的LOC估算值是33 200。

回顾历史数据可以看出，这类系统的组织平均生产率是620LOC/pm。如果一个劳动力价格是每月8000美元，则每行代码的成本约为13美元。根据LOC估算及历史生产率数据，该项目总成本的估算值是431 000美元，工作量的估算值是54人·月⁶。

功 能	LOC 估 算
用户接口及控制设备 (UICF)	2 300
二维几何分析 (2DGA)	5 300
三维几何分析 (3DGA)	6 800
数据库管理 (DBM)	3 350
计算机图形显示设备 (CGDF)	4 950
外部设备控制功能 (PCF)	2 100
设计分析模块 (DAM)	8 400
总代码行估算	33 200

图23-2 LOC方法的估算表

SAFEHOME

估算

[场景] 项目策划开始时，Doug Miller的办公室。

[人员] Doug Miller，SafeHome软件工程团队经理；Vinod Raman、Jamie Lazar及其他产品软件工程团队成员。

⁶ 假定有估算精确度要求，精确度取整到千美元和人·月。更高的精确度是不必要的，也是不现实的。

[对话]

Doug: 我们需要对这个项目进行工作量估算，然后还要为第一个增量制定一个微观进度计划，为其余的增量制定一个宏观进度计划。

Vinod (点头): 好，但是我们还没有定义任何增量。

Doug: 是这样，但这正是我们需要估算的原因。

Jamie (皱着眉): 你想知道这将花费我们多长时间吗？

Doug: 这正是我需要的。首先，我们要对SafeHome软件进行高层次上的功能分解……接着，我们必须估算每个功能对应的代码行数……然后……

Jamie: 等一下！我们应该怎样来做？

Vinod: 在过去的项目中，我已经做过了。使用用例来确定实现每个用例所需要的功能性，估计每项功能的LOC数。最好的方法是让每个人独立去做，然后比较结果。

Doug: 或者你可以对整个项目做一个功能分解。

Jamie: 但那将花费很长时间，而我们必须马上开始。

Vinod: 不，事实上这可以在几个小时内完成……就今天早上。

Doug: 我同意……我们不能期望有很高的精确性，只是了解一下SafeHome软件的大致规模。

Jamie: 我认为我们应该只估算工作量……仅此而已。

Doug: 这我们也要做。然后用这两种估算进行交叉检查。

Vinod: 我们现在就去做吧……

23.6.4 基于FP估算的实例

基于FP估算时，问题分解关注的不是软件功能，而是信息域的值。项目计划人员分别对CAD软件的外部输入、外部输出、外部查询、内部逻辑文件和外部接口文件进行估算，参看图23-3。然后使用第15章中讨论的技术来计算FP的值。为了估算FP的值，我们假定复杂度加权因子都取平均值。图23-3给出了估算的结果。

信息域值	乐观值	可能值	悲观值	估算值	加权因子	FP值
外部输入数	20	24	30	24	4	97
外部输出数	12	15	22	16	5	78
外部查询数	16	22	28	22	5	88
内部逻辑文件数	4	4	5	4	10	42
外部接口文件数	2	2	3	2	7	15
总计						320

图23-3 估算信息域的值

685

估算出每一个复杂度加权因子，并根据第15章中所描述的方法计算出复杂度校正因子：

因 子	值	因 子	值
1. 备份和恢复	4	3. 分布式处理	0
2. 数据通信	2	4. 关键性能	4

(续)

因 子	值	因 子	值
5. 现有的操作环境	3	11. 设计可复用的代码	4
6. 联机数据输入	4	12. 设计中的转换与安装	3
7. 多屏幕输入切换	5	13. 多次安装	5
8. 主文件联机更新	3	14. 易于变更的应用设计	5
9. 信息域值复杂度	5	复杂度校正因子	1.17
10. 内部处理复杂度	5		

最后，得出FP的估算值：

$$FP_{estimated} = \text{总计} \times [0.65 + 0.01 \times \sum(F_i)]$$

$$FP_{estimated} = 375$$

这类系统的组织平均生产率是6.5FP/pm。如果一个劳动力价格是每月8 000美元，则每个FP的成本约为1230美元。根据FP估算和历史生产率数据，项目总成本的估算值是461 000美元，工作量的估算值是58人·月。

23.6.5 基于过程的估算



如果时间允许，指定图23-4所示的任务时，可以使用更细的粒度，如将分析再分解成若干主要任务并分别估算每一项任务。

686

最通用的项目估算技术是根据将要采用的过程进行估算。即，将过程分解为一组较小的任务，并估算完成每个任务所需的工作量。

同基于问题的估算技术一样，基于过程的估算首先从项目范围中抽取出软件功能。接着给出为实现每个功能所必须执行的一系列的框架活动。这些功能和相关的框架活动⁷可以用表格形式给出，类似于图23-4所示。

一旦将问题功能与过程活动结合起来，计划人员就可以针对每个软件功能，估算完成各个软件过程活动所需的工作量（如人·月），这些数据写在图23-4的中心部分。然后，将平均劳动力价格（即，成本/单位工作量）应用于每个软件过程活动的估算工作量，就可以估算出成本。但各项任务的劳动力价格很可能是不同的。高级技术人员主要投入到早期的框架活动中，而初级技术人员通常参与后期的构造和发布活动，高级技术人员的劳动力价格要高于初级技术人员。

最后一个步骤就是计算每一个功能及框架活动的成本和工作量。如果基于过程的估算不依赖LOC或FP估算而实现的，我们现在就已经有了两组或三组成本与工作量的估算，可以进行比较、调和。如果两组估算非常一致，则有理由相信估算结果是可靠的。反过来，如果这些分解技术得到的结果不一致，则必须做进一步的调查和分析。

“在使用一种估算之前，最好先去了解这种估算的背景。”

——Barry Boehm和Richard Fairley

⁷ 为该项目选择的框架活动与第2章中讨论的一般性活动有所不同。它们分别是客户沟通、策划、风险分析、工程和构造/发布。

活动	客户沟通	策划	风险分析	工程		构造发布		客户评估	合计
				分析	设计	编码	测试		
任务→									
功能									
↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
合计	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
%工作量	1%	1%	1%	8%	45%	10%	36%		

图23-4 基于过程的估算表

23.6.6 基于过程估算的实例

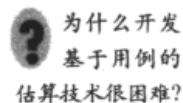
为了说明基于过程估算的使用方法，我们再考虑23.6.3节中所介绍的CAD软件。系统配置和所有软件功能都保持不变，并已在项目范围内说明。

参看图23-4中所示的基于过程的估算表，表中对CAD软件的每个功能（为了简化做了省略），都给出了其各个软件工程活动的工作量估算（人·月）。其中，工程和构造发布活动又被细分成表中所示的主要软件工程任务。对客户沟通、策划和风险分析活动，还给出了总工作量的估算，这些数值都列在图23-4底部的“合计”行中。水平合计和垂直合计为估算分析、设计、编码及测试所需的工作量提供了一个指标。应该注意到：“前期”的工程任务（需求分析和设计）花费了全部工作量的53%，说明了这些工作的相对重要性。687

如果平均一个劳动力价格是每月8 000美元，则项目总成本的估算值是368 000美元，工作量的估算值是46人·月。如果需要的话，每一个框架活动或软件工程任务都可以采用不同的劳动力价格，分别进行计算。

23.6.7 基于用例的估算

正如我们在本书的第二、三部分中提到的，用例能使软件团队深入地了解软件的范围和需求。但由于以下的原因，建立基于用例的估算方法还有困难[SMI99]：



- 描述用例时，可以采用多种格式和风格——没有标准形式。
- 用例表现的是软件的外部视图（用户视图），常常在不同的抽象级别上建立。
- 用例没有标识出它所描述的功能和特性的复杂性。
- 用例没有描述出涉及很多功能和特性的复杂行为（如，交互）。

与LOC或FP不同，一个角色的“用例”可能需要数月的工作量，而另一个角色的“用例”可能在一到两天内就能完成。

尽管有很多研究已经考虑将用例作为估算的输入，但是到目前为止，还没有出现被证实

的估算方法。Smith[SMI99]提出用例可以用于估算，但只有在用例描述的“结构层次”的情况下考虑。

Smith指出：该结构层次中的任一层次都可以由不超过10个用例来描述，而每个用例包括的场景不超过30个。显然，与描述单一子系统的用例相比，描述大型系统的用例要在更高的抽象级别上建立（并代表相当大的开发工作量）。因此，在用例能够用于估算之前，首先要建立结构层次内的层次，确定每个用例的平均长度（页数），定义软件的类型（如，实时、商业、工程/科学、嵌入式），考虑系统大致的体系结构。一旦确立了这些特性，就可以利用经验数据确定每个用例的LOC或FP的估算值（对于层次结构中的每一层）。然后，再根据历史数据计算开发系统所需的工作量。
688

为了说明如何进行计算，考虑以下关系式⁸：

$$\text{LOC估算} = N \times \text{LOC}_{\text{avg}} + [(S_a / S_h - 1) + (P_a / P_h - 1)] \times \text{LOC}_{\text{adjust}} \quad (23-2)$$

其中，

N ：实际的用例数。

LOC_{avg} ：在这种类型的子系统中，每用例的历史平均LOC值。

$\text{LOC}_{\text{adjust}}$ ：校正值，以 LOC_{avg} 的 $n\%$ 来表示，其中 n 根据当前项目的情况来确定，表示该项目与“一般”项目的差异。

S_a ：每个用例包含的实际场景数。

S_h ：在这种类型的子系统中，每个用例包含的平均场景数。

P_a ：每个用例的实际页数。

P_h ：在这种类型的子系统中，每个用例的平均页数。

式(23-2)是根据实际的用例数对LOC值进行大致的估算，其中，实际用例数已根据用例场景数和页长度进行了校正，校正值等于每个用例的历史平均LOC值的 $n\%$ 。

23.6.8 基于用例的估算实例

在23.6.3节中介绍的CAD软件包括三个子系统组：

- 用户界面子系统（包括UICF）。
- 工程子系统组（包括2DGA子系统、3DGA子系统和DAM子系统）。
- 底层子系统组（包括CGDF子系统和PCF子系统）。

用户界面子系统由六个用例来描述，描述每个用例的场景不超过10个，用例的平均长度是6页。工程子系统组由10个用例来描述（这些是在结构层次的较高层次上来考虑的），与每个用例相关的场景不超过20个，用例的平均长度是8页。最后，底层子系统组由5个用例来描述，每个用例平均只有6个场景，平均长度是5页。
689

利用式(23-2)，令 $n=30\%$ ，就产生了如图23-5所示的表格。考虑表的第一行，历史数据显示，当描述用例的场景不超过12个，用例的长度少于5页时，用户界面软件中每个用例平均需要800 LOC。这些数据非常符合CAD系统的实际情况。因此，可以使用式(23-2)来计算用户界面子系统的LOC估算值。使用同样的方法，可以对工程子系统组和底层子系统组做出估算。在图23-5中，还对估算进行了汇总，指出CAD软件总规模的估算值为42 500 LOC。

⁸ 重要的是要注意到，式(23-2)只是用于举例说明。与所有的估算模型一样，必须经过当前项目验证后才能放心使用。

	用例	场景	页	场景	页	LOC	LOC估算
用户界面子系统	6	10	6	12	5	560	3 366
工程子系统组	10	20	8	16	8	3 100	31 233
底层子系统组	5	6	5	10	6	1 650	7 970
LOC估算合计							42 568

图23-5 用例估算

以620LOC/pm作为这类系统的平均生产率，一个劳动力价格是每月8 000美元，则每行代码的成本约为13美元。根据用例估算和历史生产率数据，项目总成本的估算值是552 000美元，工作量的估算值是68人·月。

23.6.9 调和不同的估算方法

在前面章节中讨论的估算技术导出了多种估算方法，必须对这些估算方法进行调和，以得到对工作量、项目持续时间或成本的一致估算。为了说明这个调和过程，我们仍然考虑23.6.3节中介绍的CAD软件。

“复杂的方法并不一定会产生更精确的估算，尤其是当开发者在估算时加进自己直觉的时候。” ——Philip Johnson等

对CAD软件总工作量的估算，最低值是46人·月（由基于过程的估算方法得出），最高值是68人·月（由用例估算方法得出）。平均估算值（使用所有4种方法）是56人·月。与平均估算值相比，最低估算值的偏差约为18%，最高估算值的偏差约为21%。

如果估算方法所得结果的一致性很差，怎么办呢？对这个问题的回答是，需要对估算所使用的信息进行重新评估。如果不同估算之间的差别很大，一般能够追溯到以下两个原因之一：

1. 计划人员没有充分了解或误解了项目范围。

2. 在基于问题的估算技术中所使用的生产率数据不适合本应用系统，过时了（因为这些数据已不能正确地反映软件工程组织的情况），或者是误用了。

计划人员必须确定产生差别的原因，再来调和估算结果。

INFO

软件项目的自动估算技术

自动估算技术使得计划人员能够估算成本和工作量，还可以对重要的项目变量（如交付日期或人员配置）进行假设分析。尽管有很多种自动估算工具，但它们具有相同的本质特性，都能实现以下6项一般性的功能[JON96]：

1. 可交付的项目规模。估算一个或多个软件工作产品的规模。工作产品包括软件的外部表示（如，屏幕、报表）、软件本身（如，KLOC）、交付的功能性（如，功能点）及描述性信息（如，文档）。
2. 选择项目活动。选择适当的过程框架，指定软件工程任务集。
3. 预测人员配置标准。指定可用的工作人员数量。由于可用人员和工作（预测的工作量）之间完全是非线性关系，因此这是一项重要的输入。
4. 预测软件工作量。估算工具使用一个或多个估算模型（23.7节）来预测软件工作量，

这些模型能将交付的项目规模与完成交付项目所需的工作量联系起来。

5. 预测软件成本。如果给出了第4步的结果，再分别指定第2步中确定的项目活动的劳动力价格，就可以计算出成本。

6. 预测软件进度计划。当工作量、人员配置和项目活动已知后，可以根据在24章后面讨论的工作量分配推荐模型，来分配各个软件工程活动的人员，从而制定出进度计划草案。

当把不同的估算工具应用于相同的项目数据时，估算结果会有比较大的变动范围。更重要的是，有时候预测值会与实际值差异很大。这再次证明了应该把估算工具的输出看作是一个“数据点”，再从中导出估算，而不是将其作为估算的唯一来源。

23.7 经验估算模型

计算机软件估算模型使用由经验导出的公式来预测工作量，工作量是LOC或FP⁹的函数。LOC或FP的值是利用23.6.3节和23.6.4节所描述的方法进行估算的，但不使用该节中的表，而是将LOC或FP的结果值代入到估算模型中。

POINT 用以支持大多数估算模型的经验数据都是从有限的项目样本中得出的。因此，还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。所以，从这些模型中得到的结果必须慎重使用。

应该对估算模型进行调整，以反映当前项目的情况。应该使用从已完成项目中收集的数据对该模型进行检验——方法是将数据代入到模型中，然后将实际结果与预测结果进行比较。如果两者一致性很差，则在使用该模型前，必须对其进行调整和再次检验。

23.7.1 估算模型的结构

典型的估算模型是通过回归分析从以往软件项目中收集到的数据而得到的。这种模型的总体结构表现为下面的形式[MAT94]：

$$E = A + B \times (e_v)^C \quad (23-3)$$

其中，A、B、C是经验常数，E是工作量（以人·月为单位）， e_v 是估算变量（LOC或FP）。除了式(23-3)所表示的关系外，大多数估算模型都有某种形式的项目调整成分，使得E能够根据其他的项目特性（如，问题的复杂性、开发人员的经验、开发环境等）加以调整。在文献中提出了很多面向LOC的估算模型：



这些模型中，没有哪个模型能够不经过针对环境的仔细调整就可以使用的。

$$E = 5.2 \times (\text{KLOC})^{0.91}$$

Walston-Felix模型

$$E = 5.5 + 0.73 \times (\text{KLOC})^{1.16}$$

Bailey-Basili模型

$$E = 3.2 \times (\text{KLOC})^{1.05}$$

Boehm简单模型

$$E = 5.288 \times (\text{KLOC})^{1.047}$$

Doty模型，用于KLOC > 9的情况下

同样，也提出了面向FP的估算模型。其中包括：

$$E = -91.4 + 0.355\text{FP}$$

Albrecht和Gaffney模型

⁹ 在23.6.7节中，给出了使用用例作为独立变量的经验模型。但是，到目前为止，文献中关于这方面的介绍非常少。

$$E = -37 + 0.96FP \quad \text{Kemerer模型}$$

$$E = -12.88 + 0.405FP \quad \text{小型项目回归模型}$$

从这些模型可以看出，对于相同的LOC或FP值，不同的模型会产生不同的结果。其含义很清楚，必须根据当前项目的需要对估算模型进行调整。

23.7.2 COCOMO II 模型

WebRef

关于COCOMO II 的详细信息以及可下载的软件，可在 sunset.usc.edu/research/COCOMO_II/cocomo_main.html 上获得。

Barry Boehm[BOE81]在其讨论“软件工程经济学”的经典著作中介绍了一种层次结构的软件估算模型，称为COCOMO (COnstructive COst MOdel，构造性成本模型)。最初的COCOMO模型成为在产业界得到最广泛使用和讨论的软件成本估算模型之一。现在，它已经演化成更全面的估算模型，称为COCOMO II[BOE96, BOE00]。和其前身一样，COCOMO II实际上也是一种层次结构的估算模型，主要应用于以下领域：

- 应用组装模型。在软件工程的早期阶段使用，这时，用户界面的原型开发、对软件和系统交互的考虑、性能的评估以及技术成熟度的评价是最重要的。
- 早期设计阶段模型。在需求已经稳定并且基本的软件体系结构已经建立时使用。
- 体系结构后阶段模型。在软件的构造过程中使用。

692

和所有的软件估算模型一样，COCOMO II模型也需要使用规模估算信息，在模型层次结构中有三种不同的规模估算选项：对象点、功能点和源代码行。

COCOMO II应用组装模型使用的是对象点——一种间接的软件测量，计算对象点时，使用如下的计数值：(1) (用户界面的) 屏幕数，(2) 报表数，(3) 构造应用可能需要的构件数。根据Boehm[BOE96]给出的标准，将每个对象实例（如一个屏幕或一个报表）归类到三个复杂度级别之一（即简单的、中等的或困难的）。本质上，复杂度是以下变量的函数：客户和服务器数据表（产生屏幕显示和报表需要它们）的数量和来源，以及视图或版面（是屏幕或报表的一部分）的数量。

一旦确定了复杂度，就可以根据图23-6对屏幕、报表和构件的数量进行加权。首先将初始的对象实例数与表中的加权因子相乘就确定了对象点数，求和后就得到了总的对象点数。当采用基于构件的开发或一般的软件复用时，

还要估算复用的百分比，并调整对象点数：

$$\text{NOP} = \text{对象点} \times [(100 - \text{复用的百分比}) / 100]$$

其中，NOP是新的对象点。

对象类型	复杂度加权		
	简单的	中等的	困难的
屏幕	1	2	3
报表	2	5	8
3GL构件			10

图23-6 不同对象类型的复杂度加权[BOE96]

根据计算得到的NOP值进行工作量估算时，必须先确定“生产率”的值。图23-7中分别给出了在不同水平的开发者经验和开发环境成熟度下的生产率。

$$\text{PROD} = \text{NOP} / \text{人} \cdot \text{月}$$

一旦确定了生产率，就可以得到项目工作量的估算值：

693

$$\text{估算工作量} = \text{NOP} / \text{PROD}$$

开发者的经验/能力	非常低	低	正常	高	非常高
环境成熟度/能力	非常低	低	正常	高	非常高
PROD	4	7	13	25	50

图23-7 应用于对象点的生产率[BOE96]

在更高级的COCOMO II模型¹⁰中，还需要一系列的比例因子、成本驱动和调整过程。有兴趣的读者可参见[BOE00]或访问COCOMO II的Web网址。

23.7.3 软件方程式

WebRef

关于从软件方程式演化来的软件成本估算工具的信息可在www.qsm.com上获得。

软件方程式[PUT92]是一个多变量模型，它假定在软件开发项目的整个生命周期中有特定的工作量分布。该模型是根据从4000多个当代的软件项目中收集的生产率数据导出的。根据这些数据，估算模型具有以下形式：

$$E = [\text{LOC} \times B^{0.333} / P]^3 \times (1/t^4) \quad (23-4)$$

其中，

E 为工作量，以人·月或人·年为单位。

t 为项目持续时间，以月或年为单位。

B 为“特殊技能因子”¹¹。

P 为“生产率参数”，它反映了：总体的过程成熟度及管理实践；采用良好的软件工程实践的程度；使用的程序设计语言的水平；软件环境的状态；软件团队的技能和经验；应用系统的复杂性。

对于实时嵌入式软件的开发，典型值是 $P = 2000$ ；对于电信及系统软件， $P = 10\,000$ ；而对于商业系统应用， $P = 28\,000$ 。当前情况下的生产率参数可以根据以往开发工作中收集到的历史数据来导出。

694

重要的是要注意到，软件方程式有两个独立的参数：(1) 规模的估算值（以LOC为单位）；(2) 项目持续时间，以月或年为单位。

为了简化估算过程，并将该模型表示成更通用的形式，Putnam和Myers[PUT92]又提出了一组方程式，这些方程式都是从软件方程式中导出来的。最短开发时间定义为：

$$t_{\min} = 8.14 (\text{LOC}/P)^{0.43} \text{，以月为单位，用于 } t_{\min} > 6 \text{ 个月的情况} \quad (23-5a)$$

$$E = 180Bt^3 \text{，以人·月为单位，用于 } E \geq 20 \text{ 人·月的情况} \quad (23-5b)$$

¹⁰ 如前所述，这些模型使用FP或KLOC数作为规模变量。

¹¹ B 随着“对集成、测试、质量保证、文档和管理技能的需求的增长”而缓慢增加[PUT92]。对于较小的程序(KLOC = 5~15)， $B = 0.16$ ；对于超过70KLOC的较大程序， $B = 0.39$ 。

注意方程式 (23-5b) 中的 t 是以年为单位的。

对本章前面所讨论的CAD软件，使用方程式 (23-5)，令 $P = 12\ 000$ (对科学计算软件的推荐值)：

$$t_{\min} = 8.14 (33200/12000)^{0.43}$$

$$t_{\min} = 12.6 \text{ (月)}$$

$$E = 180 \times 0.28 \times (1.05)^3$$

$$E = 58 \text{ (人·月)}$$

由软件方程式得到的结果与23.6节产生的估算值非常一致。同上一节中提到的COCOMO模型一样，软件方程式已经演化了十多年，关于该估算方法扩展版本的进一步讨论请参见[PUT97b]。

23.8 面向对象项目的估算

最好用明确为OO软件设计的估算方法对传统的软件成本估算方法加以补充。Lorenz和Kidd[LOR94]给出了下列方法：

1. 使用工作量分解、FP分析和任何其他适用于传统应用的方法进行估算。
2. 使用面向对象的分析模型（第8章）建立用例并确定用例数。要认识到随着项目的进展，用例数可能会改变。
3. 由分析模型确定关键类（在第8章中称为分析类）的数量。
4. 对应用的界面类型进行归类，确定支持类的乘数：

界面类型	乘 数
没有图形用户界面 (GUI)	2.0
基于文本的用户界面 (GUI)	2.25
图形用户界面 (GUI)	2.5
复杂的图形用户界面 (GUI)	3.0

695

关键类的数量（第3步）乘上乘数就得到了支持类数量的估算值。

5. 将类的总数（关键类 + 支持类）乘以每个类的平均工作单元数。Lorenz和Kidd建议每个类的平均工作单元数是15~20人·日。
6. 将用例数乘以每个用例的平均工作单元数，对基于类的估算做交叉检查。

23.9 特殊的估算技术

在23.6节、23.7节和23.8节中讨论的估算技术可以用于任何软件项目。但是，当软件团队遇到一个持续时间非常短（以周计而不是以月计）的项目，其间又可能出现连续不断的变更时，通常应该对项目计划，特别是估算进行简化¹²。在下面的两小节中，我们来研究两种特殊的估算技术。

¹² “简化”并不意味着消除。再短的项目持续时间都必须要做计划，而估算仍是可靠计划的基础。

23.9.1 敏捷开发的估算

由于敏捷项目（第4章）的需求被定义为一组用户场景（如，极限编程中的“故事”），所以，在项目计划阶段为每个软件增量开发一个非正式的、然而严谨并有意义的估算方法是可能的。

对敏捷项目的估算使用分解的方法，包括下列步骤：

 当采用敏捷
过程时，如
何进行估算？

1. 从估算的目的出发，分别考虑每个用户场景（由最终用户或其他共利益者在项目初期建立，等价于一个大的用例）。
2. 将场景分解成一组功能，确定为实现这些功能需要完成的一组软件工程任务。
- 3a. 分别估算每一项任务。注意，可以根据历史数据、经验模型或“经验”进行估算。
- 3b. 或者，可以利用LOC、FP或某种其他面向规模的测量（如，对象点）来估算场景的“规模”。
- 4a. 对各项任务的估算结果求和，就得到了对整个场景的估算值。
- 4b. 或者，使用历史数据，将场景规模的估算值转换成工作量。
5. 将实现给定软件增量的所有场景的工作量估算值求和，就得到了该增量的工作量估算。

由于软件增量开发所需的项目时间非常短（一般是3~6周），所以该估算方法用于两个目的：（1）确保增量中将要包含的场景数与可用的资源匹配。（2）在开发增量时，为工作量分配提供依据。

23.9.2 Web工程项目的估算

正如第16章中提到的，Web工程项目常常采用敏捷过程模型。利用经过修改的功能点测量，再加上23.9.1节中简述的步骤，就可以进行WebApp的估算。

当把功能点（第15章和第22章）用于WebApp的估算时，Roetzheim[ROE00]给出了下列信息域的值。

- **输入**：每个输入屏幕或表单（如CGI或Java），每个维护屏幕，每一个标签页（无论你在什么地方使用带有标签页的编辑框时）。
- **输出**：每个静态Web页，每个动态Web页脚本（如，ASP、ISAPI或其他DHTML脚本）和每个报表（无论是基于Web的，还是管理的）。
- **表**：数据库中的每一个逻辑表，如果使用XML来存储文件中的数据，则指的是每一个XML对象（或XML属性集）。
- **界面**：定义为逻辑文件（例如，唯一记录格式），作为我们的系统与外部的边界。
- **查询**：每一个查询都是对外发布的界面，或者使用面向消息的界面。典型的例子如DCOM或COM的外部引用。

对于WebApp而言，功能点（使用上述的信息域值进行计算）是一个合理的规模指标。

Mendes和她的同事[MEN01]建议：最好通过收集与应用（如，页数、媒体数、功能数）相关的测量（称为“预测变量”）、其Web页特性（如，页复杂性、链接复杂性、图复杂性）、



696

媒体特性（如，媒体持续时间）以及功能特性（如，代码长度、复用的代码长度）来确定WebApp的规模。可以利用这些测量建立经验估算模型，用以估算项目的总工作量、网页创作的工作量、媒体创作的工作量和编写脚本的工作量。不过，在这些模型能够放心使用之前，还有进一步的工作要做。

SOFTWARE TOOLS

工作量和成本估算

目的：工作量和成本估算工具从项目将要具有的特性和构建项目的环境出发，为项目团队提供对所需工作量、项目持续时间和成本的估算。

机制：通常，成本估算工具要利用从本地项目中导出的历史数据库、整个产业中收集的数据，以及用于导出工作量、项目持续时间和成本估算的经验模型（如，COCOMO II）。这些工具以项目特性和开发环境作为输入，能够给出估算结果的变动范围。

代表性工具¹³

Costar，由Softstar Systems (www.softstarsystems.com) 开发，使用COCOMO II模型进行软件估算。

Cost Xpert，由Cost Xpert Group, Inc. (www.costxpert.com) 开发，其中集成了多种估算模型和一个历史项目数据库。

Estimate Professional，由Software Productivity Centre, Inc. (www.spc.com) 开发，根据COCOMO II模型和SLIM模型进行估算。

Knowledge Plan，由Software Productivity Research (www.spr.com) 开发，是一个完整的估算软件包，使用功能点作为主要的输入数据。

Price S，由Price Systems (www.pricesystems.com) 开发，是最古老的、使用最广泛的估算工具之一，用于大型软件开发项目。

SEER/SEM，由Galarath Inc. (www.galarath.com) 开发，提供了全面的估算能力、敏感分析、风险评估和其他特性。

SLIM-Estimate，由QSM (www.qsm.com) 开发，利用完善的“产业知识库”对使用本地数据导出的估算进行“合理性检查”。

23.10 自行开发或购买的决策

在许多软件应用领域中，直接获取（购买）计算机软件常常比自行开发的成本要低得多。软件工程管理者面临着要做出自行开发还是购买的决策问题，而且由于存在多种可选的获取方案使得决策更加复杂：(1) 购买成品构件（或取得使用许可）；(2) 购买“具有完全经验”或“具有部分经验”的软件构件（见23.4.2节），并进行修改和集成，以满足特定的需求；(3) 由外面的承包商根据买方的规格说明定制开发。

根据要购买软件的迫切程度及最终价格来确定软件获取的步骤。在最后的分析中，自行

¹³ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

开发或购买的决策是根据以下条件决定的：(1) 软件产品的交付日期是否比内部开发要快？(2) 购买的成本加上定制的成本是否比内部开发该软件的成本要低？(3) 外部支持（如，维护合同）的成本是否比内部支持的成本要少？这些条件可以应用于上述的每种可选的获取方案中。

23.10.1 创建决策树

在对自行开发还是购买进行决策时，是否有系统化的方法对与决策有关的选项进行排序？

可以使用统计技术对上述的步骤进行扩充，如决策树分析[BOE89]。图23-8描述了一个基于软件的系统X的决策树。在这个例子中，软件工程组织能够：(1) 从头开始构造系统X；(2) 复用已有的“具有部分经验”的构件来构造系统；(3) 购买现成的软件产品，并进行修改以满足当前项目的需要；(4) 将软件开发承包给外面的开发商。

如果从头开始构造系统，那么这项工作难度较大的概率是70%。项目计划人员使用本章前面所讨论的估算技术预计：一项难度较大的开发工作将需要450 000美元的成本；而一项“简单的”开发工作估计需要380 000美元。沿决策树的任一分支进行计算，得到成本的预期值如下：

$$\text{预期成本} = \sum (\text{路径概率})_i \times (\text{估算的路径成本})_i$$

其中， i 是决策树的某条路径。对于“构造系统”这条路径而言：

$$\text{预期成本}_{\text{build}} = 0.30 (380K) + 0.70 (450K) = 429K \quad (K \text{表示千美元})$$

沿着决策树的其他路径，分别给出了在多种情况下，“复用”、“购买”和“承包”方式的项目成本。这些路径的预期成本分别是：

$$\text{预期成本}_{\text{reuse}} = 0.40 (275K) + 0.60 [0.20 (310K) + 0.80 (490K)] = 382K$$

$$\text{预期成本}_{\text{buy}} = 0.70 (210K) + 0.30 (400K) = 267K$$

$$\text{预期成本}_{\text{contract}} = 0.60 (350K) + 0.40 (500K) = 410K \quad (K \text{表示千美元})$$

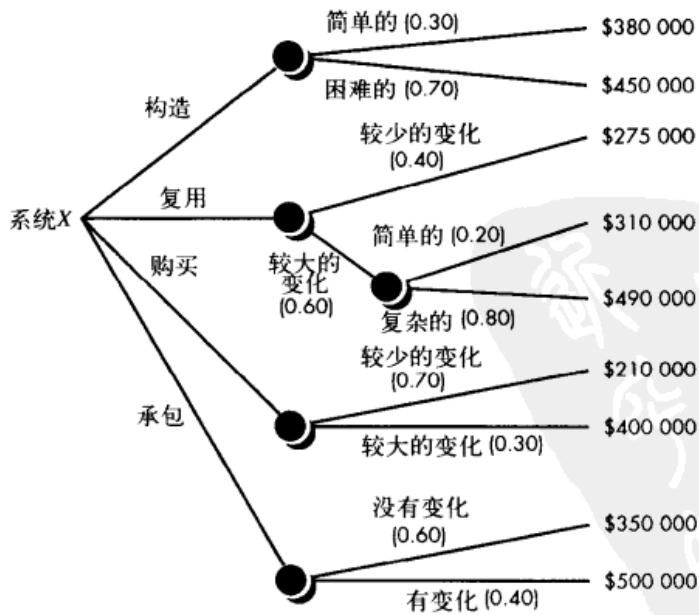


图23-8 一个支持自行开发/购买决策的决策树

根据图23-8给出的路径概率及估算成本，可以看出“购买”方式具有最低的预期成本。

不过，应该注意到，在决策过程中还有许多标准（而不仅仅是成本）必须加以考虑。在最终决定使用构造、复用、购买或承包方式时，可用性、开发者/厂家/承包商的经验、与需求的一致性、当前项目的“策略”及变更的可能性等，这些都可能是影响判断的标准，当然这些也仅仅是其中一部分标准而已。

23.10.2 外包

每一个开发计算机软件的公司迟早都会问到一个基本问题：是否有什么方法能使我们以较低的价格获得所需的软件和系统？这个问题的答案不是唯一的，但对于这个问题的情绪化的回答经常是一句话：外包。

在概念上，外包是非常简单的。软件工程活动被承包给第三方，他们能够以较低的成本并有希望以较高的质量来完成这项工作。公司内部需要做的软件工作已经降至仅仅是合同管理活动¹⁴。

“与内部开发相比，外包需要更高超的管理，这是一条规则。”——Steve McConnell

做外包决策时要从战略上或战术上考虑。在战略层上，业务管理人员要考虑大部分软件工作是否可以承包给其他厂商。在战术层上，项目经理要确定通过外包部分软件工作是否能够最好地完成项目的部分或全部。

不考虑很多其他因素，外包的决策常常是财务的决策。关于外包财务分析的详细探讨超出了本书的范围，其他书（如，[MIN95]）中有很好的讨论。不过，从正反两方面考察一下外包的决策是值得的。

从正面来看，由于减少了软件人员及相应设备（如，计算机、基础设施等），通常能够节约成本。从反面来看，公司失去了对其所需软件的部分控制权。因为软件是一种技术，它不同于公司的系统、服务及产品。这样，公司就会冒着将其竞争命运交到第三方手中的风险。

SAFEHOME

外包

[场景] CPI公司的会议室。

[人物] Mal Golden，产品开发高级经理；Lee Warren，工程经理；Joe Camalleri，业务开发副主管；Doug Miller，软件工程项目经理。

[对话]

Joe： 我们正在考虑外包产品的SafeHome软件工程部分。

Doug（很震惊）：这是什么时候的事？

Lee： 我们得到了一个国外开发者的报价，比你们小组认为要花费的成本低了30%。在这儿[将报价交给Doug看]。

Mal： 你知道，Doug，我们正在设法降低成本，30%，是30%啊。此外，这些人非常受欢迎。

¹⁴ 在广义上，可以将外包看作是任何一项从软件工程组织外获取软件或软件构件的活动。

Doug (深深地吸一口气并努力保持镇静): 你们让我很惊讶。不过在做出最后决定之前, 先听听几条意见吧。

Joe (点点头): 当然, 你说吧。

Doug: 以前, 我们还没有同这家外包公司合作过, 对吗?

Mal: 对, 但是……

Doug: 并且他们提到, 对规格说明的任何变更都要追加另外的费用, 对吗?

Joe (皱着眉): 是真的, 不过我们预期它会相当稳定。

Doug: 一个错误的假定, **Joe**。

Joe: 唔, ……

Doug: 不出几年, 我们就可能发布该产品的版本。我们有理由假定软件将提供很多新的特性, 对吗?

[都点头]

Doug: 我们以前曾经协调过国际项目吗?

Lee (担忧地看着): 没有, 但是我得知……

Doug (设法抑制着愤怒): 你要告诉我的就是: (1) 我们将要同一个不了解的厂商合作; (2) 这项工作的成本并不像他们认为的那样低; (3) 事实上, 无论他们第一次做成什么样, 在多次产品发布中, 我们都必须与他们合作; (4) 我们将要在职学习关于国际项目的相关知识。

[都保持沉默]

Doug: ……我认为这是错误的。我希望你们用一天时间重新考虑。如果在内部完成这项工作, 我们将有更多的控制权。我们拥有专门技术, 并且我能担保不会花费更多……风险更低。我知道, 和我一样, 你们都是反对风险的。

Joe (皱着眉): 你已经指出了几个优点, 但是你具有该项目在内部开发的既得利益。

Doug: 那是真的, 但是它不能改变事实。

Joe (叹气): 好吧, 先把这个问题搁置一两天, 好好考虑一下, 然后再开会做最后的决定。**Doug**, 我可以和你在私下谈谈吗?

Doug: 当然……我实在是想确保我们做的事情一切顺利。

23.11 小结

在项目开始之前, 软件项目计划人员必须先估算三件事: 需要多长时间, 需要多少工作量, 以及需要多少人员。此外, 计划人员还必须预测所需要的资源(硬件和软件)及蕴含的风险。

701 范围陈述能够帮助计划人员使用一种或多种技术进行估算, 这些技术主要分为两大类: 分解和经验建模。分解技术需要划分出主要的软件功能, 接着估算: (1) LOC的数量; (2) 信息域内的选择值; (3) 用例的数量; (4) 实现每个功能所需的人·月数; 或者, (5) 每个软件工程活动所需的人·月数。经验技术使用根据经验导出的关于工作量和时间的公式来预测这些项目数字。可以使用自动工具来实现某一特定的经验模型。

对项目做精确的估算时, 一般至少会用到上述三种技术中的两种。通过对不同技术导出

的估算值进行比较和调和，计划人员更有可能得到精确的估算。软件项目估算永远不会是一门精确的科学，但是，把可靠的历史数据与系统化的技术结合起来能够提高估算的精确度。

参考文献

- [BEN92] Bennatan, E. M., *Software Project Management: A Practitioner's Approach*, McGraw-Hill, 1992.
- [BEN03] Bennatan, E. M., "So What Is the State of Software Estimation?" *The Cutter Edge* (an online newsletter), February 11, 2003, available from <http://www.cutter.com>.
- [BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981.
- [BOE89] Boehm, B., *Risk Management*, IEEE Computer Society Press, 1989.
- [BOE96] Boehm, B., "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, July 1996, pp. 73–82.
- [BOE00] Boehm, B., et al., *Software Cost Estimation in COCOMO II*, Prentice-Hall, 2000.
- [BRO75] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1975.
- [GAU89] Gause, D. C., and G. M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.
- [HOO91] Hooper, J., and R. O. Chester, *Software Reuse: Guidelines and Methods*, Plenum Press, 1991.
- [JON96] Jones, C., "How Software Estimation Tools Work," *American Programmer*, vol. 9, no. 7, July 1996, pp. 19–27.
- [LOR94] Lorenz, M., and J. Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, 1994.
- [MAT94] Matson, J., B. Barrett, and J. Mellichamp, "Software Development Cost Estimation Using Function Points," *IEEE Trans. Software Engineering*, vol. SE-20, no. 4, April 1994, pp. 275–287.
- [MCC98] McConnell, S., *Software Project Survival Guide*, Microsoft Press, 1998.
- [MEN01] Mendes, E., N. Mosley, and S. Counsell, "Web Metrics—Estimating Design and Authoring Effort," *IEEE Multimedia*, January–March 2001, pp. 50–57.
- [MIN95] Minoli, D., *Analyzing Outsourcing*, McGraw-Hill, 1995.
- [PHI98] Phillips, D., *The Software Project Manager's Handbook*, IEEE Computer Society Press, 1998.
- [PUT78] Putnam, L., "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," *IEEE Trans. Software Engineering*, Vol SE-4, No. 4, July 1978, pp. 345–361.
- [PUT92] Putnam, L., and W. Myers, *Measures for Excellence*, Yourdon Press, 1992.
- [PUT97a] Putnam, L., and W. Myers, "How Solved Is the Cost Estimation Problem?" *IEEE Software*, November 1997, pp. 105–107.
- [PUT97b] Putnam, L., and W. Myers, *Industrial Strength Software: Effective Management Using Measurement*, IEEE Computer Society Press, 1997.
- [ROE00] Roetzheim, W., "Estimating Internet Development," *Software Development*, August 2000, available at <http://www.sdmagazine.com/documents/s=741/sdm0008d/0008d.htm>.
- [SMI99] Smith, J., "The Estimation of Effort Based on Use Cases," Rational Software Corp., 1999, download from <http://www.rational.com/media/whitepapers/finalTP171.PDF>.

702

习题与思考题

- 23.1 在计划过程中，性能是一个重要的考虑因素。针对不同的软件应用领域，分别讨论如何以不同的方式来解释性能。
- 23.2 假设你是一家开发家用机器人软件的公司的项目经理，你已经承接了为草坪割草机器人开发软件的项目。写一个范围陈述来描述该软件，确定你的范围陈述是“界定的”。如果你对机器人不熟悉，在你开始写作之前先做一些调研工作。还要说明你对所需硬件的设想。或者，你也可以选择其他你感兴趣的机器人技术问题，而不做草坪割草机器人。
- 23.3 软件项目的复杂性会影响估算的精确性。列出影响项目复杂性的软件特性（如，并发操作、图形输出等），按其对项目的影响程度顺次排列。
- 23.4 对你在问题23.2中所描述的草坪割草机器人软件进行功能分解。估算每个功能的规模（用LOC）。假定你所在的组织平均生产率是450LOC/pm，劳动力价格是每人·月7000

美元，使用本章所讲的基于LOC的估算技术来估算构造该软件所需的工作量及成本。

- 23.5 使用COCOMO II模型来估算构造一个简单的ATM软件所需的工作量，它产生12个屏幕、10个报表、将需要大约80个软件构件。假定该软件具有平均复杂度和平均开发者/环境成熟度。要求使用基于对象点的应用组装模型。
- 23.6 有一点似乎很奇怪：成本和进度估算是在软件项目计划期间完成的——在详细的软件需求分析或设计进行之前。你认为为什么会这样？是否存在不需要这样做的情况？
- 23.7 使用“软件方程式”估算问题23.2中的草坪割草机器人软件。假设采用方程式（23-5），且 $P = 8000$ 。
- 23.8 比较问题23.4和问题23.7中所得到的工作量估算值，求出标准偏差，它如何影响你对估算的确信程度？
- 23.9 使用问题23.8中得到的结果，确定是否可能期望该软件能在6个月内完成，以及完成该工作需要多少人员？
- 23.10 组建一个项目团队，开发一个软件工具来实现本章所介绍的每种估算技术。
- 23.11 建立一个电子表格模型，实现本章所述的一种或多种估算技术。或者从基于Web的资源获取一个或多个在线的软件项目估算模型。
- 23.12 假设每个分支的概率均为50%，重新计算图23-8中决策树上所标注的预期值。这会改变你的最终决定吗？

推荐读物与阅读信息

大多数软件项目管理书籍都包含了对项目估算的讨论。项目管理研究所（《PMBOK Guide》，PMI，2001）、Wysoki和他的同事（《Effective Project Management》，Wiley，2000）、Lewis（《Project Planning Scheduling and Control》，third edition，McGraw-Hill，2000）、Bennatan（《On Time, Within Budget: Software Project Management Practices and Techniques》，third edition，Wiley，2000）和Phillips[PHI98]都提供了有用的估算指南。

703 Jones（《Estimating Software Costs》，McGraw-Hill，1998）撰写的著作是迄今为止发表的对该主题的最全面的论述之一，他的书包含了可应用于各个应用领域的软件估算模型和数据。Coombs（《IT Project Estimation》，Cambridge University Press，2002）、Roetzheim和Beasley（《Software Project Cost and Schedule Estimating: Best Practices》，Prentice-Hall，1997）以及Wellman（《Software Costing》，Prentice-Hall，1992）介绍了很多有用的模型，并一步一步地给出了取得最可能估算的指南。

在Putnam和Myer的关于软件成本估算（[PUT92]和[PUT97b]）的详细论述以及Boehm的关于软件工程经济学（[BOE81]和COCOMO II[BOE00]）的书中都描述了经验估算模型。在这些书中，都对来自数百个软件项目的数据进行了详细分析。在DeMarco所著的一本优秀论著（《Controlling Software Projects》，Yourdon Press，1982）中，提供了关于软件项目的管理、测量及估算的有价值的观点。Lorenz和Kidd（《Object-Oriented Software Metrics》，Prentice-Hall，1994）及Cockburn（《Surviving Object-Oriented Projects》，Addison-Wesley，1998）考虑了面向对象系统的估算。

704 大量的关于软件估算的信息源可以在因特网上获得。一个最新的WWW参考文献列表可在SEPA Web站点<http://www.mhhe.com/pressman>上找到。

第24章 项目进度安排

要点浏览

概念：你已经选择了适合的过程模型，确定了必须完成的软件工程任务，估算了工作量和人员数目，明确了项目结束期限，甚至已经考虑了风险，现在是综合运用它们的时候了。也就是说，你应该创建一个软件工程任务网络，该网络将使你能够按时完成工作。网络创建完成之后，你必须为每一个任务确定责任，还要确保完成这些责任，并在风险到来时调整该网络。简单地说，这就是软件项目进度安排和跟踪。

人员：在项目级，是那些使用从软件工程师处获得的信息的软件项目管理者们。在个体级，是软件工程师自己。

重要性：为了建造复杂的系统，很多软件工程任务会并行地进行，而且在一个任

务中得到的工作结果可能对在另一个任务中将要进行的工作具有深远的影响。没有进度安排，任务之间的这种相互依赖性是非常难以理解的。实际上，没有一个详细的进度安排，要评估中等程度或大型的软件项目的进展情况也是不可能的。

步骤：软件过程模型规定的软件工程任务要根据具体实现的功能进行细化；工作量和工期应分配到每个任务；任务网络（也称为“活动网络”）的创建，使得软件团队能够在最后期限之前完成项目。

工作产品：项目进度安排和相关的信息。

质量保证措施：正确的进度安排需要：(1) 网络中包含所有的任务；(2) 给每个任务合理分配工作量和时间；(3) 明确指出任务间的依赖关系；(4) 资源应分配给具体要完成的工作；(5) 提供短时间间隔的里程碑，以便于过程跟踪。

关键概念
基本原则
延迟
获得值
工作量分配
人员与工作量
PNR曲线
任务网络
任务求精
时间盒
时序图
跟踪
工作分解

20世纪60年代后期，一位热情的青年工程师受命为一个自动制造业应用项目“编写”计算机程序。选择他的原因非常简单，因为在整个技术小组中他是唯一参加过计算机编程培训班的人。这位工程师对汇编语言的IN和OUT指令以及Fortran语言有所了解，但是却根本不懂软件工程，更不用说项目进度安排和跟踪了。

他的老板给了他一大堆相关的手册，口头描述了需要做些什么。年轻人被告知该项目必须在两个月之内完成。

他阅读了这些手册，想好了解决方法，就开始编写代码。两周之后，老板将他叫到办公室询问项目进展情况。

“非常顺利，”工程师以年轻人的热情回答道，“这个项目远比我想象的简单，我差不多已经完成了75%的任务。”

老板笑了，然后鼓励这个青年工程师继续努力工作，准备好一周后再汇报一次工作进度。一周之后老板将年轻人叫到办公室，问道：“现在进度如何？”

“一切顺利，”年轻人回答说，“但是我遇到了一些小麻烦。我会排除这些困难，很快就可以回到正轨上来。”

705

“你觉得在最后期限之前能完成吗？”老板问道。

“没有问题，”工程师答道，“我差不多已经完成90%了。”

如果你在软件领域中工作过几年，你一定可以将这个故事写完。毫不奇怪，青年工程师¹在整个项目工期内始终停留在90%的进度上，实际上直到交付期限之后一个月（在别人的帮助下）才完成。

在过去的40年间，这样的故事在不同的软件开发者中已经重复了成千上万次。我们不禁要问：“这是为什么呢？”

24.1 基本概念

虽然软件延期交付的原因很多，但是大多数都可以追溯到下面列出的一个或多个根本原因上：

- 不切实际的项目结束期限，由软件工程小组以外的某个人所制定，并强加给软件工程小组内的管理者和开发者。
- 客户需求发生变更，而这种变更没有在项目变更进度表上预先安排。
- 对完成该工作所需的工作量和/或资源数量估计不足。
- 在项目开始时，没有考虑可预测的和/或不可预测的风险。
- 出现了事先无法预计的技术难题。
- 出现了事先无法预计的人力问题。
- 由于项目团队成员之间的交流不畅而导致的延期。
- 项目管理者未能发现项目进度拖后，也未能采取措施来解决这一问题。

“在所有的软件项目中，过于理性或缺少理性的进度安排可能最具破坏性影响。”

——Capers Jones

在软件行业中，人们对过于乐观的（即“不切实际的”）项目结束期限已经司空见惯。从

设定项目结束期限的人的角度来看，有时候这样的项目结束期限是合理的。但是常识告诉我们，合理与否还必须由完成工作的人员来判断。

706

拿破仑曾经说过：“任何一个同意执行他本人都认为有缺点的计划的指挥官都应该受到指责；他必须提出自己的反对理由，坚持修改这一计划，最终甚至提出辞职而不是使自己的军队遭受惨败。”这句话非常值得软件项目管理者们深思。

第23章中的估算活动和本章中的进度安排技术，通常都需要在规定的项目结束期限约束下进行。如果最乐观的估算都表明该项目结束期限是不现实的，一个称职的项目管理者就应该“保护其团队免受不适当的（进度安排）压力……（并）将这种压力反映给施加压力的一方”[PAG85]。

举例说明，假定一个软件工程团队受命开发一个医疗诊断仪器的实时控制器，该控制器要在9个月之内推向市场。在进行了仔细的估算和风险分析（参见第25章）之后，软件项目管理者得到的结论是：在现有人员条件下，需要14个月的时间才能完成这一软件。这位项目管理者下一步该怎么办呢？

¹ 你可能觉得惊奇，但这个故事是自己经历过的。

“我热爱结束期限，我喜欢它们飞过时所发出的声音。”

——Douglas Adams

闯进客户的办公室（这里的客户很可能是市场营销人员）并要求修改交付日期似乎不太现实。外部市场压力已经决定了交付日期，届时必须发布产品。而（从事业前途的角度出发）拒绝这一项目同样是鲁莽的。那么应该怎么办呢？在这种情况下，建议按照以下步骤进行处理：

 当管理者要求的项目结束期限我们无法实现时，我们应该怎么办？

1. 按照以往项目的历史数据进行详细的估算，确定项目的估算工作量和工期。
2. 采用增量过程模型（参见第3章）制定一个软件工程策略，以保证能够在规定的交付日期提供主要功能，而将其他功能的实现推到以后。然后将这一计划做成本文档。
3. 与客户交流，并（用详细的估算结果）说明为什么规定的交付日期是不现实的。一定要指出所有这些估算都是基于以往的项目实践，而且为了在目前规定的交付期限完成该项目，与以往相比在工作效率上必须提高的百分比²。可以做如下解释：

“我认为在XYZ控制器软件的交付日期方面存在一个问题，我已经将一份以往项目中生产率的简化明细分类表和以多种不同方式进行的项目估算提交给各位，你们会注意到，在假设与以往的生产率相比有20%的提高的情况下，交付时间仍然需要14个月而不是9个月。”

[707]

4. 将增量开发策略作为可选计划提交给客户：

“我们有几种方案，我希望各位能够在这些方案中做出决策。第一个方案，我们可以增加预算，并引入额外的资源，以使我们能够在9个月内完成这项工作。但是应该知道由于时间限制过于苛刻，这样做将会增加质量变差的风险³；第二个方案，去掉一部分需求中所列出的软件功能和特性，由此得到功能稍弱的产品的最初版本，但是我们可以对外宣布全部功能，并在总共14个月的时间内交付这些功能；第三个方案，是不顾现实条件的约束，而希望项目能够在9个月时间内完成，结果是我们竭尽全力，但是却无法向客户提供任何功能。我希望你们会赞成我的观点，第三个方案是不可行的。过去的历史和我们最乐观的估算都表明这是不现实的，是在制造一场灾难。”

尽管这样做会有人抱怨，但如果你给出了基于准确历史数据的可靠估算，那么最终的谈判结果将可能是选择方案1或者2。不现实的交付期限就不存在了。

24.2 项目进度安排

曾经有人请教著名的《The Mythical Man-Month》（人月神话）[BRO95]一书的作者Fred Brooks，问他软件项目的进度是怎样被延误的？他的回答既简单又深刻：“某天某时”。

技术性项目（不论它是涉及水力发电厂建设，还是操作系统开发）的现实情况是，在实现一个大目标之前必须完成数以百计的小任务。这些任务中有些是处于主流之外的，其进度不会影响到整个项目的完成时间。而有些任务则是位于“关键路径”之上的，如果这些“关

² 如果生产率提高10%~25%，实际上是有可能完成这个项目的。但多数情况是，所需提高的团队生产率要高于50%。所以说这是不切实际的期望。

³ 你还可以补充说：人员数目的增加不会成比例地缩短完成该项目所需要的时间。

键”任务的进度拖后，则整个项目的完成日期就会受到威胁。

项目管理者的职责是定义所有的项目任务，建立相应的网络来描述它们之间的依赖关系，明确网络中的关键任务，然后跟踪关键任务的进展，以确保能够在“某天某时”发现进度延误情况。为了做到这一点，管理者必须建立相当详细的进度表，使得项目管理者能够监督进度，并控制整个项目。

软件项目进度安排是一种活动，它通过将工作量分配给特定的软件工程任务，从而将所估算的工作量分配到计划的项目工期内。但要注意的是，进度是随时间而不断演化的。在项目计划早期，是建立一个宏观进度表，该进度表标识出所有主要的过程框架活动和这些活动所影响的产品功能。随着项目的进展，宏观进度表中的每个条目都会被细化成详细的进度表，这样就标识了特定的（完成一个活动所必须实现的）软件任务，同时也进行了进度安排。

“过于乐观的进度安排并不会缩短实际进度，反而会拖后进度。”——Steve McConnell

可以从两个不同的角度来讨论软件工程项目的进度安排。第一种情况，计算机系统的最终发布日期已经确定（而且不能更改），软件开发组织必须将工作量分布在预先确定的时间框架内。第二种情况，假定已知大致的时间界限，但是最终发布日期由软件工程开发组织自行确定，工作量是以能够最好地利用资源的方式来进行分配，而且在对软件进行仔细分析之后才决定最终发布日期。但不幸的是，第一种情况发生的频率远远高于第二种情况。

24.2.1 基本原则

就像软件工程的所有其他领域一样，软件项目进度安排也有很多的基本指导原则：

划分。必须将项目划分成多个可以管理的活动、动作和任务。为了实现项目的划分，产品和过程都需要进行分解。

相互依赖性。划分后的各个活动、动作或任务之间的相互依赖关系必须是明确的。有些任务必须按顺序出现，而有些任务则可以并发进行。有些活动或动作只有在其他活动产生的工作产品完成后才能够开始，而有些则可以独立进行。

时间分配。每个安排了进度计划的任务必须分配一定数量的工作单位（例如，若干人·日的工作量）。此外，还必须为每个任务指定开始日期和完成日期，任务的开始日期和完成日期取决于任务之间的相互依赖性以及工作方式是全职还是兼职。

工作量确认。每个项目都有预定数量的人员参与。在进行时间分配时，项目管理者必须确保在任意时段中分配的人员数量不会超过项目团队中的总人员数量。例如，某项目分配了3名软件工程师（例如，每天可分配的工作量为3人·日⁴）。在某一天中，需要完成7项并发的任务，每个任务需要0.50人·日的工作量，在这种情况下，所分配的工作量就大于可供分配的工作量。



当你进行进度安排时，要划分工作，描述任务间的相互依赖性，为每个任务分配工作量和时间，确定责任、输出结果和里程碑。

⁴ 实际上，由于与工作无关的会议、病假、休假日及各种其他原因，可供分配的工作量要少于3人·日。但在这里，我们假定员工时间是100%可用的。

确定责任。安排了进度计划的每个任务都应该指定特定的团队成员来负责。

明确结果。安排了进度计划的每个任务都应该有一个明确的输出结果。对于软件项目而言，输出结果通常是一个工作产品（例如一个模块的设计）或某个工作产品的一部分。通常可将多个工作产品组合成“可交付产品”。

确定里程碑。每个任务或任务组都应该与一个项目里程碑相关联。当一个或多个工作产品经过质量评审（参见第26章）并且得到认可时，标志着一个里程碑的完成。

随着项目进度的发展，会应用到以上所述的每一条原则。

24.2.2 人员与工作量之间的关系

对于小型软件开发项目，只需一个人就可以完成需求分析、设计、编码和测试。随着项目规模的增长，必然会有更多的人员参与。（不可能奢望让一个人工作十年来完成10人·年的工作量！）



如果必须给一个已经拖延的项目增加人员，就要确保已将详细划分的任务分配给他们。

许多负责软件开发工作的管理者仍然普遍坚信这样一个神话：“即使进度拖后，我们也总是可以增加更多的程序员，并在后期跟上进度。”不幸的是，在项目后期增加人手通常会对项目产生破坏性的影响，其结果是使进度进一步拖延。后期增加的人员必须学习这一系统，而培训他们的人员正是一直在工作着的那些人，当他们进行教学时，就不能完成任何工作，从而使项目进一步拖延。

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交流的路径数量和整个项目中交流的复杂度。虽然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交流路径就会增加额外的工作量，从而需要更多的时间。

多年以来的经验数据和理论分析都表明项目进度是具有弹性的。即在一定程度上可以缩短项目交付日期（通过增加额外资源），也可以拖延项目交付日期（通过减少资源数量）。

PNR (Putnam-Norden-Rayleigh) 曲线⁵表明了一个软件项目中所投入的工作量与交付时间的关系。项目工作量和交付时间的函数关系曲线如图24-1所示。图中的 t_o 表示项目交付所需的最少时间（即花费工作量最少的项目交付时间），而 t_o 左边（即当我们想提前交付时）的曲线是非线性上升的。

710

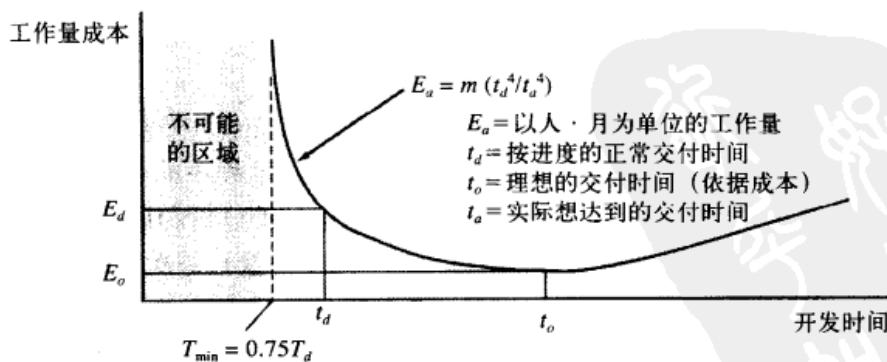


图24-1 工作量和交付时间的关系

⁵ 对它的初始研究参见[NOR70]和[PUT78]。



如果能够拖延项目交付时间，PNR曲线表明可以明显降低项目成本。



离项目的交付日期越来越近时，你意识到，不管参与工作的人数是多少，工作均不能按计划完成。那就面对现实，并确定新的交付日期。

举一个例子，假设一个软件项目团队根据进度和现有的人员配置，估算所需要的工作量应为 E_a ，正常的交付时间应为 t_d 。虽然可以提前交付，但曲线在 t_d 的左侧急剧上升。事实上，PNR曲线不仅说明了项目的交付时间不能少于 $0.75t_d$ ，如果想更少，项目会进入“不可能的区域”，并面临着很高的失败风险；还说明了最低成本的交付时间 t_o 应该满足 $t_o = 2t_d$ ，即拖延项目交付可以明显降低成本，当然，这里的成本必须将与延期相关的营销成本排除在外。

在第23章中介绍的软件方程式[PUT92]就是来源于PNR曲线，它表明了完成一个项目的时间与投入该项目的人员工作量之间是高度非线性的关系。交付的代码（源程序代码）行数 L 与工作量和开发时间的关系可以用下面的方程式表示：

$$L = P \times E^{1/3} t^{4/3}$$

其中， E 是以人·月为单位的开发工作量； P 是生产率参数，它反映了影响高质量软件工程工作的各种因素的综合效果（ P 通常取值在2000到12 000之间）； t 是以月为单位的项目工期。

重新调整这个软件方程式，可以得到开发工作量 E 的计算公式：

711

$$E = L^3 / (P^3 t^4) \quad (24-1)$$

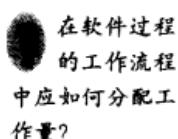
其中， E 是在软件开发和维护的整个生命周期内所需的工作量（按人·年计算）； t 是以年为单位的开发时间；引入平均劳动力价格因素（\$/人·年）之后，开发工作量的计算公式还能够与开发成本相关联。

这一方程式引出了一些有趣的结果。假设有一个复杂的实时软件项目，估计需要33 000 LOC、12人·年的工作量。如果项目团队有8个人，那么项目大约需要1.3年的时间完成。但是如果将交付日期延长到1.75年，则由公式（24-1）所描述的模型所具有的高度非线性特性将得出以下结论：

$$E = L^3 / (P^3 t^4) \sim 3.8 \text{ 人} \cdot \text{年}$$

这意味着通过将交付日期推迟6个月，我们可以将项目团队的人数从8人减少到4人！这一结果的有效性有待考证，但是其含意却十分清楚：通过在略为延长的时间内使用较少的人员，可以实现同样的目标。

24.2.3 工作量分配



在软件过程的工作流程中应如何分配工作量？

在第23章中讨论的各种软件项目估算技术最终都归结为对完成软件开发所需工作单位（如人·月）的估算。软件过程中的工作量分配通常采用40-20-40法则。总体工作量的40%分配给前期的分析和设计，40%用于后期测试。

因此，你可以推断出编码工作（20%的工作量）是次要的。

这种工作量分配方法只能作为指导原则⁶。各个项目的特点决定了其工作量如何分配。用于项目计划的工作量很少超过2%~3%，除非提交给组织的项目计划费用极高而且具有高风险。需求分析大约占用10%~25%的工作量，用于分析或原型开发的工作量应该与项目规模和复杂

⁶ 现在，40-20-40法则不再适用。有些人认为用于分析和设计的工作量应该超过总体工作量的40%。而相反的是，敏捷开发的倡导者（第4章）认为“前期”工作应该越快越好，团队应该快速进入构造阶段。

度成正比地增长。通常有20%~25%的工作量用于软件设计，用于设计评审和随之而来的迭代开发也必须考虑。

因为在软件设计时投入了相当的工作量，随后的编码工作就变得相对简单。总体工作量的15%~20%就可以完成这一工作。测试和随之而来的调试工作将占用30%~40%的软件开发工作量。软件的重要性决定了所需测试工作的分量，如果软件系统是人命相关的（即软件错误可能使人丧命），就应该考虑分配更高的测试工作量比例。[712]

24.3 为软件项目定义任务集

本书第一部分，介绍了很多不同的过程模型。无论一个软件团队选择的是线性顺序模型、增量模型、演化模型，或它们的某种变型，过程模型都是由任务集组成的，这些任务集使得软件团队能够定义、开发和最终维护计算机软件。

没有能普遍适用于所有软件项目的任务集。适用于大型复杂系统的任务集可能对于小型相对简单的软件项目而言就过于复杂。因此一个有效的软件过程应该定义一组任务集来满足不同类型项目的要求。

同第2章介绍的一样，一个任务集包括软件工程工作任务、里程碑，以及为完成某个特定项目所必须完成的工作产品。为了获得高质量的软件产品，所选择的任务集必须提供充分的规程要求，但同时又不能让项目团队负担不必要的工作。

在进行项目进度安排时，必须将任务集分布在项目时序图上。任务集应该根据软件团队所决定的项目类型和严格程度而有所不同。尽管很难建立一个全面详尽的软件项目分类方法，但是大多数软件组织遇到的项目一般属于下述类型：

WebRef

**适应性过程模型
(adaptable process model, APM)**
可以辅助不同的软件项目定义各自的任务集。有关APM的详细信息见www.rspacom/apm。

1. 概念开发项目，目的是为了探索某些新的业务概念或者某种新技术的应用。
2. 新应用开发项目，根据特定的客户需求而承担的项目。
3. 应用增强项目，对现有软件中最终用户可见的功能、性能或界面进行修改。
4. 应用维护项目，以一种最终用户不会立即察觉到的方式对现有软件进行纠错、适应性修改或者扩展。
5. 再工程项目，为了重建一个现有（遗产）系统的全部或部分而承担的项目。

即使在单一的项目类型中，也会有许多因素影响任务集的选择。[PRE99]中描述了很多因素：项目的规模、潜在的用户数量、任务的关键性、应用程序的寿命、需求的稳定性、客户/开发者进行沟通的容易程度、可应用技术的成熟度、性能约束、嵌入式和非嵌入式特性、项目人员配置、再工程因素等。综合考虑这些因素就形成了“严格程度”的指标，它将应用于所采用的软件过程中。[713]

24.3.1 任务集举例

每种项目类型都可以通过线性顺序、迭代（如原型模型或增量模型）或者演化（如螺旋模型）等过程模型来实现。在某些情况下，项目类型可以从一种形式平滑地转换为另一种形式。例如，成功的概念开发项目通常会演化成为新应用开发项目，而新应用开发项目结束之

后，可能又开始了一个应用增强项目。这个进程是自然的和可预测的，不论组织是采用何种过程模型都将会发生。因此，下面几节将要介绍的主要软件工程任务可适用于所有的过程模型流程。作为一个例子，下面我们将介绍概念开发项目的主要软件工程任务。

概念开发项目是在探索某些新技术是否可行时发起的。这种技术是否可行尚不可知，但是某个客户（如营销人员）相信其具有潜在的利益。概念开发项目的完成需要应用以下主要任务：

- 1.1 确定概念范围。确定项目的整体范围。
- 1.2 初步的概念策划。确定组织承担项目范围所涵盖的工作应具有的工作能力。
- 1.3 技术风险评估。评估与项目范围内将要实现的技术相关联的风险。
- 1.4 概念证明。证明新技术在软件环境中的生命力。
- 1.5 概念实现。以一种可以由客户方进行评审的方式实现概念的表示，而且当将概念销售给其他客户或管理者时能够用于“营销”目的。
- 1.6 客户对概念的反应。向客户索取对新技术概念的反馈，并以特定的客户应用作为目标。

快速浏览以上主要任务，你应该不会有任何诧异。实际上概念开发项目的软件工程任务流程（以及其他所有类型的项目）与人们的常识相差无几。

24.3.2 主要任务的求精

在上一小节中所描述的主要任务可以用来制定项目的宏观进度表。但是，必须将宏观进度表进行细化，以创建一个详细的项目进度表。细化工作始于将每个主要任务分解为一组子任务（以及相关的工作产品和里程碑）。

作为任务分解的例子，我们考虑在上一小节中讨论的“任务1.1——确定概念范围”。任务求精可以使用大纲格式（outline format），但是在这里将使用过程设计语言来说明“确定概念范围”这一活动的流程：

- 714 任务定义：任务1.1——确定概念范围
- 1.1.1 确定需求、效益和潜在的客户
 - 1.1.2 确定所希望的输出/控制和驱动应用程序的输入事件
 - 开始任务1.1.2
 - 1.1.2.1 FTR：评审需求的书面描述⁷
 - 1.1.2.2 导出一个客户可见的输出/输入列表
 - 1.1.2.3 FTR：与客户一起评审输出/输入，并在需要时进行修改
 - 结束任务1.1.2
 - 1.1.3 为每个主要功能定义功能/行为
 - 开始任务1.1.3
 - 1.1.3.1 FTR：评审在任务1.1.2中得到的输出和输入数据对象
 - 1.1.3.2 导出功能/行为模型
 - 1.1.3.3 与客户一起评审功能/行为模型，并在需要时进行修改
 - 结束任务1.1.3
 - 1.1.4 把需要在软件中实现的技术要素分离出来

⁷ FTR表示在此需要进行一次正式技术评审（第26章）。

- 1.1.5 研究现有软件的可用性
 - 1.1.6 确定技术可行性
 - 1.1.7 对系统规模进行快速估算
 - 1.1.8 创建“范围定义”
- 结束任务1.1的任务定义

这些任务和用过程设计语言进行细化时所标注的子任务共同构成了制定“确定概念范围”这一活动的详细进度表的基础。

24.4 定义任务网络

单个任务和子任务之间存在顺序上的相互依赖关系。而且，当有多人参与软件工程项目时，多个开发活动和任务并行进行的可能性很大。在这种情况下，必须协调多个并发任务，以保证它们能够在后继任务需要其工作产品之前完成。

KEY POINT
任务网络是描述任务间依赖关系和确定关键路径的有效机制。

任务网络，也称为活动网络，是一个项目任务流程的图形表示。有时将任务网络作为在自动项目进度安排工具中输入任务序列和依赖关系的机制。最简单的任务网络形式（当创建宏观进度表时使用）只描述了主要的软件工程任务。图24-2显示了概念开发项目的任务网络示意图。

软件工程活动的并发本质导致了在进度安排上有很多要求。由于并行任务是异步发生的，所以计划者必须确定任务之间的依赖关系，以保证项目朝着最终完成的方向持续发展。另外，项目管理者应该注意那些位于关键路径之上的任务。也就是说，为了保证整个项目的如期完成，就必须保证这些任务能够如期完成。在本章的后面部分，我们将详细讨论这些问题。

715

值得注意的是，图24-2中所示的任务网络是宏观的。详细的任务网络（详细进度表的前身）中应该对图24-2所示的各个活动加以扩展。例如，应该扩展任务1.1，以表现24.3.2节所述的任务求精中的所有详细任务。

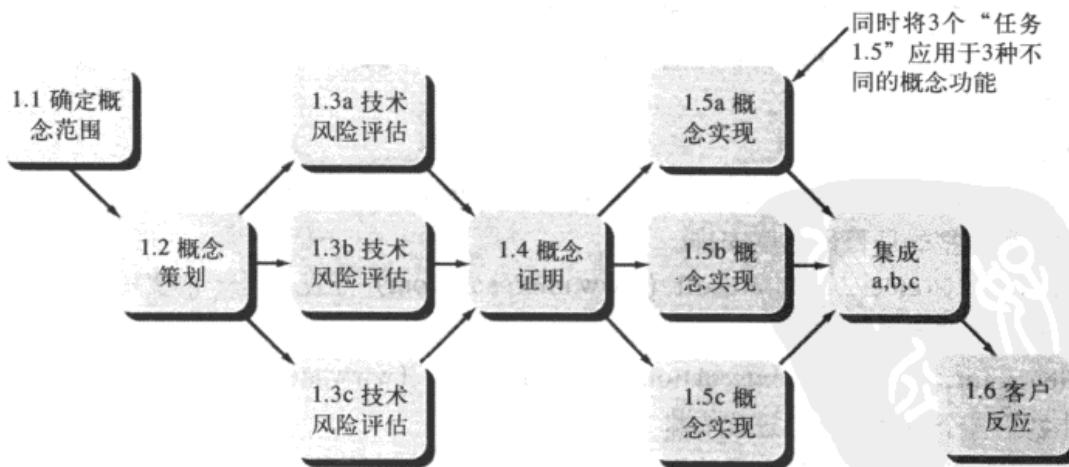


图24-2 概念开发项目的任务网络

24.5 进度安排

软件项目的进度安排与任何其他多任务工程工作的进度安排几乎没有差别。因此，通用

的项目进度安排工具和技术不必做太多修改就可以应用于软件项目。

程序评估及评审技术 (program evaluation and review technique, PERT) 和关键路径方法 (critical path method, CPM) 就是两种可以用于软件开发的项目进度安排方法。这两种技术都是由早期项目计划活动中已经产生的信息来驱动的，这些信息包括：

- 工作量的估算。
- 产品功能的分解。
- 适当过程模型和任务集的选择。
- 任务的分解。

任务之间的依赖关系可以通过任务网络来确定。任务，有时也称为项目的工作分解结构 (work breakdown structure, WBS)，可以是针对整个产品，也可以是针对单个功能来进行定义。

“唯一要我们做出决定的就是怎样分配所给定的时间。”

——Gandalf, 《指环王：护戒使者》

PERT和CPM两种方法都是项目工作定量划分的工具，可以使软件计划者完成：(1) 确定关键路径——决定项目工期的任务链；(2) 通过使用统计模型为单个任务建立“最有可能”的时间估算；(3) 计算为特定任务定义时间“窗口”的“边界时间”。

SOFTWARE TOOLS

项目进度安排

目的：项目进度安排工具的目的是使项目管理者能够确定工作任务，建立工作任务之间的依赖关系，为工作任务分配人员，并能够形成大量图表，以辅助对软件项目进行跟踪和控制。

机制：通常，项目进度安排工具要求完成工作分解结构的规格说明，或者创建任务网络。一旦确定了工作分解结构（大纲形式）或任务网络，就可以为每一个任务指定开始及结束日期、分配人员、确定交付日期以及其他内容。然后，项目进度安排工具可以生成多种时序图及其他表格，使项目管理者能够对项目的任务流进行评估。随着项目的进展，这些信息可以不断地进行更新。

代表性工具⁸

AMS Realtime，由Advanced Management Systems (www.amsusa.com) 开发，可以对各种规模和类型的项目做进度安排。

Microsoft Project，由Microsoft (www.microsoft.com) 开发，是一个使用最广泛的PC项目进度安排工具。

Viewpoint，由Artemis Internation Solutions Corp. (www.atemisp.com) 开发，支持项目计划的各个方面，包括进度安排。

项目管理软件厂商及其产品的详细清单见www.infogal.com/pmc/pmcsrw.htm。

⁸ 这里记录的工具并不代表本书支持这些工具，而只是这类工具的样例。在大多数情况下，工具名字由各自的开发者注册为商标。

24.5.1 时序图

在创建软件项目进度表时，计划者将从一组任务（工作分解结构）入手。如果使用自动工具，就可以采用任务网络或者任务大纲的形式输入工作分解结构，然后再为每一项任务输入工作量、工期和开始日期。此外，还可以将某些任务分配给特定的人员。



时序图使你能够确定在特定时间点将进行什么任务。

输入信息之后，就可以产生时序图（timeline chart），也叫做甘特图（Gantt chart）。可以为整个项目建立一个时序图，或者，也可以为各个项目功能或各个项目参与者分别建立各自的时序图。

图24-3给出了时序图的格式，该图描述了一个字处理软件产品中“确定概念范围”任务的软件项目进度安排。所有的项目任务（针对“确定概念范围”）都在左边的栏中列出。水平条表示各个任务的工期，同一时段中存在多个水平条时，就代表任务之间的并发性，菱形表示里程碑。

717

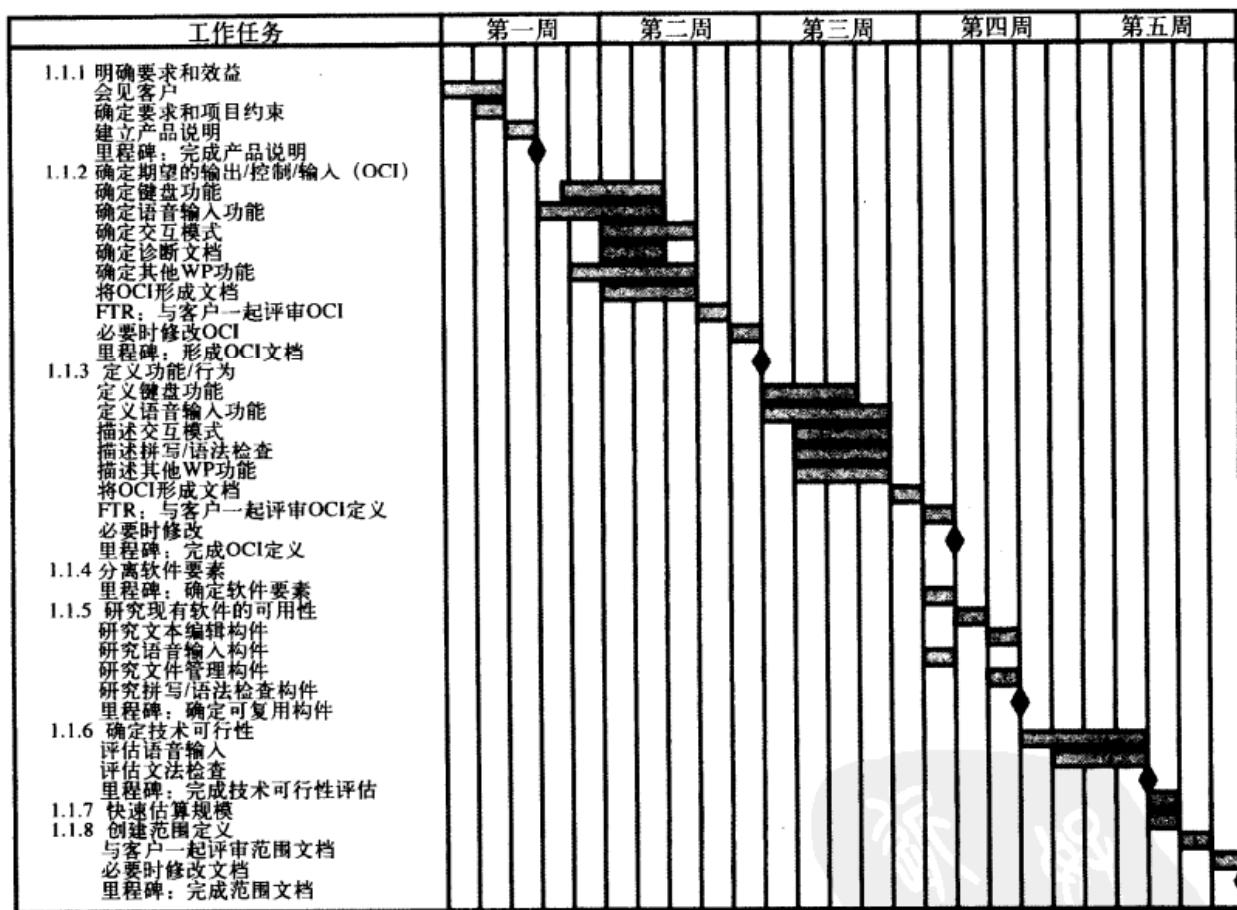


图24-3 一个时序图的例子

输入了生成时序图所需的信息之后，大多数软件项目进度安排工具都能生成项目表——列出所有项目任务的表格，项目表中列出了各个任务计划的开始与结束日期和实际开始与结束日期以及各种相关信息（见图24-4）。通过项目表与时序图，项目管理者就可以跟踪项目的进展情况。

工作任务	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	备注
1.1.1 明确要求和效益 会见客户 确定要求和项目约束 建立产品说明 里程碑：完成产品说明	wk1, d1 wk1, d2 wk1, d3 wk1, d3	wk1, d1 wk1, d2 wk1, d3 wk1, d3	wk1, d2 wk1, d2 wk1, d3 wk1, d3	wk1, d2 wk1, d2 wk1, d3 wk1, d3	BLS JPP BLS/JPP	2 pd 1 pd 1 pd	范围定义需要更多的工作量/时间
1.1.2 确定期望的输出/控制/输入 (OCI) 确定键盘功能 确定语音输入功能 确定交互模式 确定诊断文档 确定其他WP功能 将OCI形成文档 FTR：与客户一起评审OCI 必要时修改OCI 里程碑：形成OCI文档	wk1, d4 wk1, d3 wk2, d1 wk2, d1 wk1, d4 wk2, d1 wk2, d3 wk2, d4 wk2, d5	wk1, d4 wk1, d3 wk2, d2 wk2, d2 wk1, d4 wk2, d2 wk2, d3 wk2, d4 wk2, d5	wk2, d2 wk2, d2 wk2, d3 wk2, d3 wk2, d3 wk2, d3 wk2, d3 wk2, d4 wk2, d5	wk2, d2 wk2, d2 wk2, d3 wk2, d3 wk2, d3 wk2, d3 wk2, d3 wk2, d4 wk2, d5	BLS JPP MLL BLS JPP MLL MLL MLL	1.5 pd 2 pd 1 pd 1.5 pd 2 pd 3 pd 3 pd 3 pd	
1.1.3 定义功能/行为							

图24-4 一个项目表（亦称资源表）的例子

24.5.2 跟踪进度

项目进度表为软件项目管理者提供了一张进度路线图。如果制定正确，项目进度表中应该能够确定在项目进展过程中必须进行跟踪和控制的任务及里程碑。项目跟踪可以通过以下方式实现：

- 718
- 定期举行项目状态会议，由项目团队中的各个成员分别报告进度和存在问题。
 - 评估所有在软件工程过程中所进行的评审的结果。
 - 判断正式的项目里程碑（图24-3中的菱形）是否在预定日期内完成。
 - 比较项目表（图24-4）中列出的各项任务的实际开始日期与计划开始日期。
 - 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。
 - 通过分析获得值（第24.6节）来定量地评估项目进展。
- 实际上，有经验的项目管理者都会使用所有这些跟踪技术。

“软件状态报告的基本规则可以归纳为一句话：一点都不奇怪。” ——Capers Jones



719

软件项目管理者通过控制方法来管理项目资源、处理问题和指导项目参与者。如果一切顺利（即项目在预算范围内按进度进行，评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控制，以便尽快解决问题。当诊断出问题之后，可能需要增加额外的资源来解决问题，如可能需要雇佣新员工，或者需要重新安排项目进度。

在面对交付期限的巨大压力时，有经验的项目管理者有时会使用一种称为时间盒（time-boxing）[ZAH95]的项目进度安排和控制技术。时间盒方法认为完整的产品可能难以在预定时间内交付，因此，应该选择增量软件开发范型（第3章），并为每个增量的交付制定各自的进度表。

跟踪进度

[场景] Doug Miller的办公室，SafeHome软件项目开始之前。

[人物] Doug Miller，SafeHome软件工程团队经理；Vinod Raman、Jamie Lazar以及产品软件工程团队的其他成员。

【对话】

Doug（看着PPT幻灯片）：第一个SafeHome增量的进度表看起来比较合理，但是，我们很难跟踪项目进展情况。

Vinod（看上去非常担心）：为什么？大部分工作产品的任务我们是按天来安排进度的，并且我们保证并没有过度分配资源。

Doug：一切都好。但是，我们怎样才能确定什么时候能完成第一个增量的分析模型呢？

Jamie：任务是迭代的，所以很难。

Doug：我知道，但是……好吧，例如，就拿“确定分析类”来说，你们认为它是一个里程碑。

Vinod：是啊。

Doug：是谁做的决定？

Jamie（很生气）：谁做的有什么关系。

Doug：Jamie，这样不太好。我们必须安排FTR（正式技术评审，参见第26章），可是你还没做呢。例如，成功完成对分析模型的评审就是一个合理的里程碑。清楚了吗？

Jamie（皱着眉）：好吧，回到制图板。

Doug：完成修正不能超过1个小时……现在其他人可以开始了。

721

24.6 获得值分析



在24.5节，我们讨论了一系列项目跟踪的定性方法，为项目管理者提供了项目进展情况的指标。但是，对所提供的信息的评估在某种程度上是主观的。那么当软件小组按项目进度表实施工作任务时，是否存在某种定量的技术来评估项目进展情况呢？事实上，确实存在一种用于项目进展的定量分析技术，称为获得值分析（earned value analysis, EVA）。Humphrey[HUM95]对获得值给出了如下讨论：

不管要完成何种类型的工作，获得值系统为每个（软件项目）任务提供了通用

跟踪进度

[场景] Doug Miller的办公室，SafeHome软件项目开始之前。

[人物] Doug Miller，SafeHome软件工程团队经理；Vinod Raman、Jamie Lazar以及产品软件工程团队的其他成员。

[对话]

Doug（看着PPT幻灯片）：第一个SafeHome增量的进度表看起来比较合理，但是，我们很难跟踪项目进展情况。

Vinod（看上去非常担心）：为什么？大部分工作产品的任务我们是按天来安排进度的，并且我们保证并没有过度分配资源。

Doug：一切都好。但是，我们怎样才能确定什么时候能完成第一个增量的分析模型呢？

Jamie：任务是迭代的，所以很难。

Doug：我知道，但是……好吧，例如，就拿“确定分析类”来说，你们认为它是一个里程碑。

Vinod：是啊。

Doug：是谁做的决定？

Jamie（很生气）：谁做的有什么关系。

Doug：Jamie，这样不太好。我们必须安排FTR（正式技术评审，参见第26章），可是你还没做呢。例如，成功完成对分析模型的评审就是一个合理的里程碑。清楚了吗？

Jamie（皱着眉）：好吧，回到制图板。

Doug：完成修正不能超过1个小时……现在其他人可以开始了。

721

24.6 获得值分析



在24.5节，我们讨论了一系列项目跟踪的定性方法，为项目管理者提供了项目进展情况的指标。但是，对所提供的信息的评估在某种程度上是主观的。那么当软件小组按项目进度表实施工作任务时，是否存在某种定量的技术来评估项目进展情况呢？事实上，确实存在一种用于项目进展的定量分析技术，称为获得值分析（earned value analysis, EVA）。Humphrey[HUM95]对获得值给出了如下讨论：

不管要完成何种类型的工作，获得值系统为每个（软件项目）任务提供了通用的值尺度，可以估算完成整个项目所需要的总小时数，并且可以根据各个任务所估算的小时数占总小时数的百分比来确定该任务的获得值。

更简单地说，获得值是对项目进展的测量。它使得我们能够不依赖于感觉，采用定量分析方法来评估“完成百分比”。事实上，Fleming和Koppleman[FLE98]认为获得值分析“早在项目进展的前15%就提供了精确的、可靠的项目执行状况指示”。

按照以下步骤可以确定获得值：

如何计算获得值以评估项目进展?

1. 为进度表中的每个工作任务确定其预计工作的预算成本 (budgeted cost of work scheduled, BCWS)。在估算过程中, 要计划每个软件工程任务的工作量 (以人·时或人·日为单位), 因此, BCWS_i是指工作任务*i*的计划工作量。为了确定在项目进度表中某特定时间点的项目进展状况, BCWS的值是在项目进度表中该时间点应该完成的所有工作任务的BCWS_i值之和。

2. 所有工作任务的BCWS值加起来, 可计算出完成工作的预算 (budget at completion, BAC), 因此,

$$BAC = \sum (BCWS_k), \text{ 对所有任务 } k$$

3. 接着, 可计算已完成工作的预算成本 (budgeted cost of work performed, BCWP)。BCWP的值是在项目进度表中该时间点已经实际完成的所有工作任务的BCWS值之和。

Wilkens[WIL99]指出: “BCWS和BCWP的不同点是, 前者表示计划将完成的工作的预算, 而后者表示已实际完成的工作的预算。”给定BCWS、BAC和BCWP的值, 就可以得出相关的项目进展指标:

WebRef
大量有关获得值分析的信息源见
www.acq.osd.mil/pm/

进度表执行指标 $SPI = BCWP/BCWS$

进度表偏差 $SV = BCWP - BCWS$

其中, SPI是效率指标, 指出项目使用预定资源的效率, SPI值越接近1.0说明项目的执行效率越高。SV只表示与计划进度的偏差。

预定完成百分比 = $BCWS/BAC$

表示在时间点*t*应该完成工作的百分比值。

完成百分比 = $BCWP/BAC$

表示在特定时间点*t*实际完成工作的百分比值。

也可以计算已完成工作的实际成本 (actual cost of work performed, ACWP)。ACWP的值是在项目进度表中某时间点已经完成的工作任务的实际工作量之和。然后, 再计算:

成本执行指标 $CPI = BCWP/ACWP$

成本偏差 $CV = BCWP - ACWP$

CPI值越接近1.0说明项目与预算越接近。CV表示在项目特定阶段的成本节省 (相对于计划成本) 或短缺。

就像地平线上的雷达一样, 获得值分析在可能出现问题之前就指出了进度安排的难点, 这使得软件项目管理者能够在项目危机出现前采取有效措施。

24.7 小结

计划活动是软件项目管理的重要组成部分, 而进度安排是计划活动的首要任务。进度安排与估算方法和风险分析相结合, 可以为项目管理者画出一张路线图。

进度安排始于过程分解。根据项目特性, 为将要完成的工作选择适当的任务集。任务网络描述了各项工程任务、各项任务与其他任务之间的依赖关系以及各项任务的计划工期。任务网络可以用来计算项目的关键路径、时序图和各种项目信息。以进度表为指导, 项目管理者可以跟踪和控制软件工程过程中的每一个步骤。

722

参考文献

- [723] [BRO95] Brooks, M., *The Mythical Man-Month*, anniversary edition, Addison-Wesley, 1995.
 [FLE98] Fleming, Q. W., and J. M. Kopelman, "Earned Value Project Management," *Crosstalk*, vol. 11, no. 7, July 1998, p. 19.
 [HUM95] Humphrey, W., *A Discipline for Software Engineering*, Addison-Wesley, 1995.
 [NOR70] Norden, P., "Useful Tools for Project Management," in *Management of Production*, M. K. Starr, ed., Penguin Books, 1970.
 [PAG85] Page-Jones, M., *Practical Project Management*, Dorset House, 1985, pp. 90-91.
 [PRE99] Pressman, R. S., *Adaptable Process Model*, R. S. Pressman & Associates, 1999.
 [PUT78] Putnam, L., "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," *IEEE Trans. Software Engineering*, vol SE-4, no. 4, July 1978, pp. 345-361.
 [PUT92] Putnam, L., and W. Myers, *Measures for Excellence*, Yourdon Press, 1992.
 [WIL99] Wilkens, T. T., "Earned Value, Clear and Simple," Primavera Systems, April 1, 1999, p. 2.
 [ZAH95] Zahniser, R., "Time-boxing for Top Team Performance," *Software Development*, March 1995, pp. 34-38.

习题与思考题

- 24.1 假定你要为一所大学开发一个联机课程登记系统 (OLCRS)。首先从客户的角度（如果你是一名学生就很简单了！）指出一个好系统应该具有的特性。（或者你的老师会为你提供一些初步的系统需求。）按照第23章所介绍的估算方法，估算OLCRS系统的开发工作量和工期。建议你如下进行：
 - a.确定OLCRS项目中的并行工作活动。
 - b.将工作量分布到整个项目中。
 - c.建立项目里程碑。
- 24.2 是否存在这种情况：一个软件项目里程碑没有与某个评审相关联？如果有，请至少给出一个例子。
- 24.3 使用进度安排工具（如果有条件）或者纸笔（如果需要）制定OLCRS项目的时序图。
- 24.4 人员和时间的关系是高度非线性的。使用Putnam的软件方程式（见24.2.2节）编制一个表，以反映软件项目中人员数量与项目工期之间的关系——该项目需要50 000 LOC 和15人·年的工作量（生产率参数为5000）。假定该软件必须在24±12个月的时间期限内交付。
- 24.5 尽管为延迟的软件项目增加人手可能会进一步拖延工期，但是否在某些情况下并不如此呢？请举例说明。
- 24.6 为OLCRS项目选择适当的任务集。
- 24.7 “不合理的”项目结束期限是软件行业中存在的现实情况。当你遇到这种情况时应该如何处理？
- 24.8 当多个人员参与软件项目时，就有可能产生“交流费用”。与其他人员进行交流要花费时间，这样就会降低个人生产率 (LOC/人·月)，最终导致整个团队生产率下降。（举几个例子）说明非常精通软件工程实践和运用正式技术评审的软件工程师是如何提高团队生产率的（与个人生产率的总和进行比较）。提示：假设评审减少了返工，而返工可能占一个人20%~40%的时间。
- 24.9 为OLCRS或者你感兴趣的其他软件项目定义任务网络。确信你已给出所有的任务和里

程碑，并为各个任务分配了所估算的工作量和工期。如果可能的话，使用自动进度安排工具来完成这一工作。

- 24.10 如果有自动进度安排工具，请为24.9题中定义的任务网络确定关键路径。
- 24.11 宏观进度表和详细进度表的区别是什么？是否有可能只依据所制定的宏观进度表来管理一个项目？为什么？
- 24.12 假设你是一个软件项目管理者，受命为一个小型软件项目进行获得值统计。这个项目共计划了56个工作任务，估计需要582人·日，目前有12个工作任务已经完成。但是，按照项目进度，现在应该完成15个任务，下面给出相关进度安排数据（单位：人·日），请你做出获得值分析。

724

任 务	计划工作量	实际工作量
1	12.0	12.5
2	15.0	11.0
3	13.0	17.0
4	8.0	9.5
5	9.5	9.0
6	18.0	19.0
7	10.0	10.0
8	4.0	4.5
9	12.0	10.0
10	6.0	6.5
11	5.0	4.0
12	14.0	14.5
13	16.0	—
14	6.0	—
15	8.0	—

计算该项目的进度表执行指标SPI、进度表偏差SV、预定完成百分比、完成百分比、成本执行指标CPI、成本偏差CV。

推荐读物与阅读信息

事实上，每一本有关软件项目管理的书都涉及进度安排，The Project Management Institute (《PMBOK Guide》，PMI，2001)、Wysoki及其同事 (《Effective Project Management》，Wiley，2000)、Lewis (《Project Planning Scheduling and Control, third edition》，McGraw-Hill，2000)、Bennatan (《On Time, Within Budget: Software Project Management Practices and Techniques, third edition》，Wiley，2000)、McConnell (《Software Project Survival Guide》，Microsoft Press，1998,)、Roetzheim和Beasley (《Software Project Cost and Schedule Estimating: Best Practices》，Prentice-Hall，1997) 都详细讨论了这一主题。Boddie写的书 (《Crunch Mode》，Prentice-Hall，1987) 值得所有“要在90天里完成一个6个月的项目”的管理者们阅读。

McConnell (《Rapid Development》，Microsoft Press，1996) 对导致过度乐观的软件项目进度安排问题进行了深入的讨论，并讨论了你能为此做些什么。O'Connell (《How to Run Successful Projects II: The Silver Bullet》，Prentice-Hall，1997) 按步骤给出了项目管理的方

法，该方法能够帮助你为项目建立现实的进度表。

Webb和Wake (《Using Earned Value: A Project Manager's Guide》, Ashgate Publishing, 2003)、Fleming和Koppelman (《Earned Value Project Management》, Project Management Institute Publications, 1996) 较详细地讨论了获得值技术在项目计划、跟踪和控制方面的应用。

因特网上有大量与项目进度安排相关的信息资源。在SEPA的Web站点[http://www.mhhe.](http://www.mhhe.com/pressman)

725 [com/pressman](http://www.mhhe.com/pressman)上有最新的关于项目进度安排的WWW参考列表。

第25章 风险管理

要点浏览

概念：很多问题都会困扰软件项目，风险分析和风险管理就是一系列步骤，用来帮助软件团队理解和管理不确定的事物。风险是潜在的——它可能发生也可能不发生。但是，不管发生还是不发生，我们都应该去识别它，评估它发生的概率，估算它的影响，并制定它实际发生时的应急计划。

人员：软件项目所涉及的每一个人——管理者、软件工程师和共利益者——都要参与风险分析和风险管理。

重要性：想想童子军的格言：时刻准备着。软件项目是一项困难重重的任务，大量的事情可能出错，而且坦率地说，很多事情经常出错。为此，时刻准备着——理解风险、采取主动的措施去回避或管理风险——是一个好的软件项目管

理者应具备的基本条件。

步骤：第一步称为“风险识别”，即辨别出什么情况下可能会出问题。第二步，分析每个风险，确定其可能发生的概率以及发生时将带来的危害。了解这些信息之后，就可以按照可能发生的概率和危害程度对风险进行排序。第三步，制定一个计划来管理那些出现概率高和危害程度大的风险。

工作产品：风险缓解、监测和管理(risk mitigation, monitoring and management, RMMM)计划或一组风险信息表单。

质量保证措施：所要分析和管理的风险，应该通过彻底研究人员、产品、过程和项目来确定。RMMM计划应该随着项目的进展而修订，以保证所考虑的风险是近期可能发生的。风险管理的应急计划应该是符合实际的。

关键概念

评估
识别
预测
求精
原则
主动风险策略
被动风险策略
风险分类
风险显露度
风险表
RMMM
安全和灾难

Robert Charette[CHA89]在他关于风险分析与管理的书中给出了风险概念的定义：

首先，风险涉及的是未来将要发生的事情。今天和昨天的事情已不再关心，如同我们已经在收获由我们过去的行为所播下的种子。问题是：我们是否能够通过改变今天的行为，而为一个不同的、充满希望的、更美好的明天创造机会。其次，风险涉及改变。如思想、观念、行为、地点的改变……第三，风险涉及选择，而选择本身就具有不确定性。因此，就像死亡和税收一样，风险是生活中最不确定的因素之一。

对于软件工程领域中的风险，Charette的三条概念定义是显而易见的。未来是我们所关心的——什么样的风险会导致软件项目彻底失败？改变也是我们所关心的——客户需求、开发技术、目标环境以及所有其他与项目相关因素的改变将会对进度安排和总体成功产生什么影响？最后，我们必须抓住选择机会——我们应该采用什么方法及工具？需要多少人员参与？

对质量的要求要达到什么程度才是“足够的”？

Peter Drucker [DRU75]曾经说过：“当没有办法消除风险，甚至连试图降低该风险也存在疑问时，这个风险就是真正的风险了。”在弄清楚软件项目中的“真正风险”之前，识别出所有对管理者及开发者而言显而易见的风险是很重要的。

726

25.1 被动风险策略和主动风险策略

被动风险策略被戏称为“印地安纳·琼斯学派的风险管理”[THO92]。印地安纳·琼斯在20世纪80年代以其名字为片名的电影中，每当面临无法克服的困难时，总是一成不变地说：“不要担心，我会想出办法来的！”印地安纳·琼斯从不担心任何问题，直到风险发生，再做出英雄式的反应。

“如果你不主动进攻风险，风险将会主动进攻你。”

——Tom Gilb

遗憾的是，一般的软件项目管理者并不是印地安纳·琼斯，软件项目团队的成员也不是他可信赖的伙伴。因此，大多数软件项目团队还是仅仅依赖于被动的风险策略。被动策略最多不过是针对可能发生的风[726]险来监测项目，直到风险发生时，才会拨出资源来处理它们。大多数情况下，软件项目团队对风险不闻不问，直到出现了问题。这时，项目团队才赶紧采取行动，试图迅速地纠正错误，这通常叫做“救火模式”。当这样的努力失败后，“危机管理”[CHA92]接管一切，这时项目已经处于真正的危机中了。

对于风险管理，更好的是主动风险策略。主动风险策略早在技术工作开始之前就已经启动了。识别出潜在的风险，评估它们发生的概率及产生的影响，并按其重要性进行排序。然后，软件项目团队就可以制定一个计划来管理风险。计划的主要目标是回避风险，但不是所有的风险都能够回避，所以，项目团队必须制定一个应急计划，使其在必要时能够以可控和有效的方式做出反应。在本章的后面部分，我们将讨论风险管理的主动策略。

727

25.2 软件风险

虽然对于软件风险的严格定义还存在很多争议，但一般认为软件风险包含两个特性[HIG95]：

- 不确定性——风险可能发生也可能不发生；即，没有100%会发生的风[727]险¹。
- 损失——如果风险发生，就会产生恶性后果或损失。

进行风险分析时，重要的是量化每个风险的不确定程度和损失程度。为了实现这点，必须考虑不同类型的风险。

 在建造软件时，我们可能遇到什么类型的风险？

项目风险威胁到项目计划。也就是说，如果项目风险发生，就有可能会拖延项目的进度和增加项目的成本。项目风险是指预算、进度、人员（聘用职员及组织）、资源、利益相关方、需求等方面的问题以及它们对软件项目的影响。在第23章中，项目复杂度、规模及结构不确定性也属于项目（和估算）风险因素。

技术风险威胁到要开发软件的质量及交付时间。如果技术风险发生，开发

¹ 100%发生的风险是加在软件项目上的约束。

工作就可能变得很困难或根本不可能。技术风险是指设计、实现、接口、验证和维护等方面的潜在问题。此外，规格说明的歧义性、技术的不确定性、技术陈旧以及“前沿”技术也是技术风险因素。技术风险的发生是因为问题比我们所设想的更加难以解决。

商业风险威胁到要开发软件的生存能力。商业风险常常会危害到项目或产品。五个主要的商业风险是：(1) 开发了一个没有人真正需要的优良产品或系统（市场风险）；(2) 开发的产品不再符合公司的整体商业策略（策略风险）；(3) 开发了一个销售部门不知道如何去销售的产品（销售风险）；(4) 由于重点的转移或人员的变动而失去了高级管理层的支持（管理风险）；(5) 没有得到预算或人员上的保证（预算风险）。

应注意简单的分类并不总是行得通，某些风险根本无法事先预测。

另一种常用的分类方式是由Charette [CHA89]提出的。已知风险是通过仔细评估项目计划、开发项目的商业及技术环境以及其他可靠的信息来源（如不现实的交付时间，没有文档化需求或文档化软件范围、恶劣的开发环境）之后可以发现的那些风险。可预测风险能够从过去项目的经验中推断出来（如人员变动、与客户之间欠缺沟通、由于正在进行维护而使开发人员精力分散）。不可预测风险就像纸牌中的大王，它们可能会真的出现，但很难事先加以识别。

INFO

风险管理的七个原则

美国卡内基·梅隆大学软件工程研究所（Software Engineering Institute, SEI, www.sei.cmu.edu）定义了“实施有效风险管理框架”的七条原则：

保持全面观点——在软件所处的系统中考虑软件风险以及该软件所要解决的业务问题。

采用长远观点——考虑将来可能产生的风险（如软件的变更），并制定应急计划使将来发生的事件成为可管理的。

鼓励广泛交流——如果有人提出一个潜在的风险，要重视它；如果以非正式的方式提出一个风险，要考虑它。任何时候都要鼓励共利益者和用户提出风险。

结合——考虑风险时必须与软件过程相结合。

强调持续的过程——在整个软件过程中，团队必须保持警惕。随着信息量的增加，要修改已识别的风险；随着知识的积累，要加入新的风险。

开发共享的产品——如果所有共利益者共享相同版本的软件产品，将更容易进行风险识别和评估。

鼓励协同工作——在风险管理活动中，要汇聚所有共利益者的智慧、技能和知识。

25.3 风险识别

风险识别试图系统化地指出对项目计划（估算、进度、资源分配等）的威胁。通过识别已知的和可预测的风险，项目管理者首先要做的是在可能时回避这些风险，在必要时控制这些风险。

在25.2节中提出的每一类风险又分为两个不同的类型：一般风险和产品特定的风险。一般风险对每一个软件项目而言都是潜在的威胁。产品特定的风险只有那些对当前项目特定的技术、人员及环境非常了解的人才能识别出来。为了识别产品特定的风险，必须检查项目计划及软件

范围说明，然后回答这个问题：“本产品中有什么特殊的特性可能会威胁到我们的项目计划？”

“没有经历实际风险的项目不能认为是成功的。这种项目几乎是无益的，否则这些项目早就有人开发了。”

—Tom DeMarco和Tim Lister

729



虽然考虑一般风险是重要的，但是，通常产品特定的风险会带来更多的问题。一定要花时间尽可能多地识别出产品特定的风险。

识别风险的一种方法是建立风险条目检查表。该检查表可用于风险识别，并且主要用来识别下列几种类型中的一些已知的和可预测的风险：

- 产品规模——与要开发或要修改的软件的总体规模相关的风险。
- 商业影响——与管理者或市场所施加的约束相关的风险。
- 客户特性——与客户的素质以及开发者和客户定期沟通的能力相关的风险。
- 过程定义——与软件过程定义的程度以及该过程被开发组织遵守的程度相关的风险。
- 开发环境——与用来开发产品的工具的可获得性及质量相关的风险。
- 开发技术——与待开发软件的复杂性及系统所包含技术的“新奇性”相关联的风险。
- 人员才干及经验——与软件工程师的总体技术水平及项目经验相关的风险。

风险条目检查表可以采用不同的方式来组织。与上述每个主题相关的问题可以针对每一个软件项目来回答。有了这些问题的答案，计划者就可以估计风险产生的影响。也可以采用另一个不同的风险条目检查表格式，即仅仅列出与每一种类型有关的特性。最终，给出一组“风险因素和驱动因子”[AFC88]以及它们发生的概率。有关性能、支持、成本及进度的驱动因子将在后面进行讨论。

在一些文献（如[SEI93], [KAR96]）中提出了很多种与软件项目风险相关的检查表，为软件项目中的一般风险识别提供了有效的手段，在进行风险分析和风险管理时都可以借鉴。另外，还可使用比较简短的问题列表[KEI98]来初步判断一个软件项目是否“处于风险之中”。

25.3.1 评估整体项目风险

下面的提问来源于对世界各地的有经验的软件项目管理人员的调查而得到的风险资料[KEI98]，根据各个问题对项目成功的相对重要性将问题进行了排序。

730

我们正在进行的软件项目面临严重的风险吗？

WebRef

风险雷达(risk radar)是一种数据库，也是一种工具，它帮助项目管理者识别、排序和交流项目风险，见www.spmn.com。

1. 高层的软件管理者和客户管理者已经正式承诺支持该项目了吗？
2. 最终用户对项目和待开发的系统/产品热心支持吗？
3. 软件工程团队及其客户充分理解需求了吗？
4. 客户已经完全地参与到需求定义中了吗？
5. 最终用户的期望现实吗？
6. 项目范围稳定吗？
7. 软件工程团队的技能搭配合理吗？
8. 项目需求稳定吗？
9. 项目团队对将实现的技术有经验吗？
10. 项目团队的人员数满足项目需要吗？
11. 所有的客户/用户对项目的重要性和待开发的系统/产品的需求有共识吗？

“风险管理是针对成年人的项目管理。”

——Tim Lister

如果对这些问题的任何一个回答是否定的，则应务必启动缓解、监测和管理风险的步骤。项目的风险程度与对这些问题否定回答的数量成正比。

25.3.2 风险因素和驱动因子

美国空军有一本小册子[AFC88]，其中包含了如何很好地识别和消除软件风险的指南。他们所用的方法是要求项目管理者识别影响软件风险因素的风险驱动因子——性能、成本、支持和进度。在这里，风险因素是以如下的方式定义的：

- **性能风险**——产品能够满足需求且符合其使用目的的不确定程度。
- **成本风险**——能够维持项目预算的不确定程度。
- **支持风险**——开发出的软件易于纠错、修改及升级的不确定程度。
- **进度风险**——能够维持项目进度且按时交付产品的不确定程度。

每一个风险驱动因子对风险因素的影响均可分为四个影响类别——可忽略的、轻微的、严重的或灾难的。图25-1[BOE89]指出了由于未识别出的软件失误而产生的潜在影响（标号为1的行），或没有达到预期的结果所产生的潜在影响（标号为2的行）。影响类别的选择是以最符合表中描述的特征为基础的。

731

影响类别		风险因素		性 能	支 持	成 本	进 度		
灾难的	1	无法满足需求而导致任务失败		失误将导致进度延迟和成本增加，预计超支\$500K					
	2	严重退化使得根本无法达到要求的技术性能	无法做出响应或无法支持的软件	资金严重短缺，很可能超出预算		无法在交付日期内完成			
严重的	1	无法满足需求而导致系统性能下降，使得任务能否成功受到质疑		失误将导致系统运行的延迟并使成本增加，预计超支\$100K到\$500K					
	2	技术性能有些降低	在软件修改中有少量的延迟	资金不足，可能会超支	交付日期可能延迟				
轻微的	1	无法满足需求而导致次要任务的降级		成本、影响及可以补救的进度上的小问题，预计超支\$1K到\$100K					
	2	技术性能有些降低	较好的软件支持	有充足的资金来源	现实的、可完成的进度计划				
可忽略的	1	无法满足需求而导致使用不方便或不易操作		失误对进度及成本的影响很小，预计超支少于\$1K					
	2	技术性能没有降低	易于进行软件支持	可能低于预算	交付日期将会提前				

注：(1) 未识别出的软件失误或缺陷所产生的潜在影响。

(2) 如果没有到达预期的结果所产生的潜在影响。

图25-1 影响评估[BOE89]

25.4 风险预测

风险预测，又称风险估计，试图从两个方面评估每一个风险：（1）风险发生的可能性或概率；（2）风险相关问题产生的后果。项目计划人员、其他管理人员及技术人员都要进行以下4步风险预测活动：

1. 建立一个尺度，以反映风险发生的可能性。
2. 描述风险产生的后果。
3. 估计风险对项目及产品的影响。
4. 标明风险预测的整体精确度，以免产生误解。

732

按此步骤进行风险预测，目的是使我们可以按照优先级来考虑风险。任何软件团队都不可能以同样的严格程度来为每个可能的风险分配资源，通过将风险按优先级排序，软件团队可以把资源分配给那些具有最大影响的风险。

25.4.1 建立风险表

风险表给项目管理者提供了一种简单的风险预测方法²。风险表样本如图25-2所示。

风 险	风险类型	概 率	影响值	RMMR
规模估算可能很不正确	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	
最终用户抵制该系统	BU	40%	3	
交付日期将提前	BU	50%	2	
资金将会流失	CU	40%	1	
用户将改变需求	PS	80%	2	
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	
人员变动比较频繁	ST	60%	2	
•				

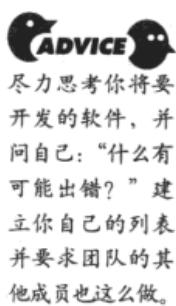
影响值：

- 1—灾难的
- 2—严重的
- 3—轻微的
- 4—可忽略的

风险类型：

- PS—产品规模
- BU—商业影响
- CU—客户特性
- TE—开发技术
- DE—开发环境
- ST—人员

图25-2 排序前的风险表样本



项目团队首先要在表中的第一列列出所有风险（不管多么细微）。这可以利用25.3节所述的风险条目检查表来完成。在第二列中给出每一个风险的类型（按25.3节给出的类型）。在第三列中填入各个风险发生的概率。各个风险的概率值可以由团队成员各自估计，他们可以按循环投票的方式进行，直到大家对风险概率的评估值趋于接近为止。

下一步是评估每个风险所产生的影响。使用图25-1所示的特性评估每个

² 风险表可以采用电子表格模式来实现，使表中的条目易于操作和排序。

KEY POINT

风险表中应按照概率和影响对风险进行排序。

风险因素，并确定其影响类别。将四个风险因素——性能、支持、成本及进度——的影响类别求平均³可得到一个整体的影响值。

733

完成了风险表的前四列内容之后，就可以按照概率和影响来进行排序。高概率、高影响的风险放在表的上方，而低概率风险则移到表的下方。这样就完成了第一次风险排序。

项目管理者研究已排序的表，然后定义一条中截线。该中截线（表中某处之上的一条水平线）表示：只有那些在中截线之上的风险才会得到进一步的关注，而在中截线之下风险则需要重新评估以进行第二次排序。从管理的角度来看，风险的影响和发生的概率是截然不同的，见图25-3。一个具有高影响但发生概率很低的风险因素不应该耗费太多的管理时间，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，应该首先列入随后的风险分析步骤中。

所有在中截线之上的风险都必须进行管理。标有RMMM的列中包含了一个指示器，指向为所有中截线之上的风险所建立的风险缓解、监测及管理计划（Risk Mitigation, Monitoring and Management Plan, RMMM）或一组风险信息表单。RMMM计划和风险信息表单将分别在25.5节和25.6节讨论。

“如今，谁也不会奢望能够很好地了解任务以使其不会出现惊奇，而惊奇就意味着风险。”

— Stephen Grey

734

风险概率可以通过先做个别估计而后求出一个有代表性的值来确定。虽然该方法是可行的，不过还有很多其他更复杂的确定风险概率的技术[AFC88]。风险驱动因子的评估是基于定性的概率尺度，可表示为：不可能、不一定、可能或极可能，然后再将每一个定性值与数学上的概率值相关联（如概率为0.7~0.95表示极可能发生的风险）。

25.4.2 评估风险影响

如果风险真的发生了，有三个因素可能会影响风险所产生的后果，即风险的本质、范围和时间。风险的本质是指当风险发生时可能带来的问题。例如，一个定义很差的与客户硬件的外部接口（技术风险）会妨碍早期的设计和测试，也有可能导致项目后期阶段的系统集成问题。风险的范围包括风险的严重性（即风险有多严重）及风险的整体分布情况（项目中有多少部分受到影响或有多少客户受到损害）。最后，风险的时间是指何时能够感受到风险的影响及风险的影响会持续多长时间。在大多数情况下，项目管理者希望“坏消息”越早出现越好，但在某些情况下，是越迟越好。

让我们再回到美国空军提出的风险分析方法[AFC88]上来。下面的步骤可以用来确定风险

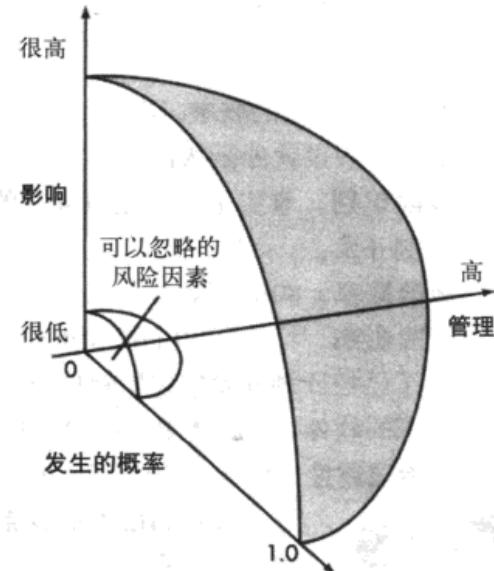


图25-3 风险与管理

³ 如果某个风险因素对项目来说比较重要，可以使用加权平均方法。

的整体影响：

-  我们如何评估风险产生的后果。
1. 确定每个风险因素发生的平均概率。
 2. 使用图25-1中列出的标准来确定每个因素的影响。
 3. 按照前面几节给出的方法填写风险表，并分析其结果。

整体的风险显露度（risk exposure, RE）可由下面的关系确定[HAL98]：

$$RE = P \times C$$

其中 P 是风险发生的概率， C 是风险发生时带来的项目成本。

例如，假设软件团队按如下方式定义项目风险：

风险识别。事实上，计划可复用的软件构件中只有70%将集成到应用系统中，其他功能必须定制开发。

风险概率。80%（大约）。

风险影响。计划了60个可复用的软件构件，如果只能利用70%，则18个构件必须从头开发（除了已经计划开发的定制软件外）。平均每个构件的程序行数是100 LOC，本地数据表明每个LOC的软件工程成本是\$14.00，开发构件的总成本（影响）将是 $18 \times 100 \times 14 = \$25\,200$ 。

风险显露度。 $RE = 0.80 \times \$25\,200 \approx \$20\,200$



将所有风险的RE和项目的成本估算进行比较。如果RE大于项目成本的50%，则必须重新评估项目的生存能力。

风险的成本估算完成之后，就可以为风险表中的每个风险计算其风险显露度。所有风险（风险表中截线之上）的总体风险显露度不仅为调整项目最终的成本估算提供了依据，还可预测在项目进展过程中不同阶段所需人员资源的增长情况。

25.4.1节和25.4.2节所述的风险预测和分析方法可以在软件项目进展过程中反复运用。项目团队应该定期复查风险表，重新评估每一个风险，以确定新环境是否引起其概率和影响发生改变。这样可能需要在风险表中添加一些新的风险，删除一些不再有影响的风险，或改变一些风险的相对位置。

SAFEHOME

风险分析

[场景] Doug Miller的办公室，SafeHome软件项目开始之前。

[人物] Doug Miller，SafeHome软件工程团队经理；Vinod Raman、Jamie Lazar以及产品软件工程团队的其他成员。

[对话]

Doug: 很高兴今天和大家一起讨论SafeHome项目的风险问题。

Jamie: 是讨论什么情况下可能会出问题吗？

Doug: 是的。这儿有几种可能会出问题的类型。[他给每个人展示了25.3节中给出的类型。]

Vinod: 噢……你只是要求我们找出风险，还是……

Doug: 不，我想让每一个人建立一个风险列表……立即动手……

(10分钟过去了，每人都在写着。)

Doug: 好了，停止。

Jamie: 可是我还没有完成！

Doug: 没关系，我们还要对列表进行复查。现在，给列表中的每一个风险指定其发生概率的百分比值，然后按1（较小的）到5（灾难的）的取值范围确定其对项目的影响。

Vinod: 就是说如果我认为风险的发生跟掷硬币差不多，就给50%的概率，如果我认为风险的影响是中等的，就给影响值为3，对吗？

Doug: 非常正确。

(5分钟过去了，每个人都在写着。)

Doug: 好了，停止。现在，我们在白板上建立一组列表，我来写，轮流从你们各自的列表中取出一项。

(15分钟过去了，列表完成。)

Jamie (指着白板并笑着说): **Vinod**，那个风险（指向白板中的一项）很可笑，大家意外选中它的可能性很大，应该删除。

Doug: 不，先留着吧。不管有多么不可思议，我们应考虑所有风险。一会儿我们还要精减这个列表。

Jamie: 已经有40多个风险了……我们究竟怎样才能管理它们呢？

Doug: 管理不了。所以将这些风险排序之后，我们还要定义中截线。明天我们继续开讨论会中截线。现在，回去继续工作……工作之余考虑是否还有遗漏的风险。

736

25.5 风险求精

在项目计划的早期，风险很可能只是一个大概的描述。随着时间的推移，对项目和风险的了解加深，可以将风险精化为一组更详细的风险，在某种程度上，这些风险更易于缓解、监测和管理。



什么是描述风险的好方式？

实现方法之一是按条件—变迁—结果 (condition-transition-consequence, CTC) 格式[GLU94]来表示风险，即采用如下方式来描述风险：

给定<条件>，则有结论：(可能) <结果>

使用CTC格式，在25.4.2节中提到的可复用软件构件的风险可描述为：

给定条件：所有可复用软件构件必须符合特定设计标准，但是某些并不符合；
则有结论：(可能) 仅70%的计划可复用构件将集成到最终的系统中，需定制开发剩余30%的构件。

可按如下方式对这个一般条件进行求精：

子条件1 某些可复用构件是由第三方开发的，没有其内部设计标准相关资料。

子条件2 构件接口的设计标准尚未确定，有可能和某些现有的软件可复用构件不一致。

子条件3 某些可复用构件是采用不支持目标环境的语言开发的。

这些子条件的结论是相同的（即必须定制开发30%的软件构件），但求精可以帮助我们排除重大风险，使我们更易于分析风险和采取措施。

25.6 风险缓解、监测和管理

这里讨论的所有风险分析活动只有一个目的——辅助项目团队制定处理风险的策略。一个有效的策略必须考虑三个问题：

- 风险回避。
- 风险监测。
- 风险管理及应急计划。

如果软件团队采取主动的方法，最好的策略就是风险回避。这可以通过建立一个风险缓解计划来实现。例如，假设频繁的人员变动被标注为项目风险 r_1 。基于以往的历史和管理经验，可以估计频繁人员变动的概率 l_1 为0.70（70%，相当高）。其影响 x_1 预测为严重的，即频繁的人员变动将对项目成本及进度有严重的影响。

“我采取如此多的预防措施，是因为我不想留下任何冒险机会。”

——Napoleon

 如何缓解风险？ 为了缓解这个风险，项目管理者必须制定一个策略来减少人员变动。可能采取的步骤包括：

- 与现有人员一起探讨人员变动的起因（如恶劣的工作条件、报酬低、竞争激烈的劳动力市场）。
- 在项目开始之前采取行动，对那些我们能够控制的起因，想办法减少它们。
- 项目启动之后，假设会发生人员变动，当人员离开时，使用开发技术来保证工作的连续性。
- 组织项目团队，使得每一个开发活动的信息能被广泛传播和交流。
- 制定编制文档的标准，并建立相应机制以确保能够及时创建文档。
- 同等对待所有工作的评审（使得不止一个人能够“跟上进度”）。
- 给每一个关键的技术人员都指定一个后备人员。

随着项目的进展，风险监测活动开始了，项目管理者应该监测那些可以表明风险是否正在变高或变低的因素。在频繁的人员变动的例子中，应该监测下列因素：

- 团队成员对项目压力的普遍态度。
- 团队的凝聚力。
- 团队成员彼此之间的关系。
- 与报酬和利益相关的潜在问题。
- 在公司内及公司外工作的可能性。

除了监测上述因素之外，项目管理者还应该监测风险缓解步骤的效力。例如，前面叙述的风险缓解步骤中要求制定编制文档的标准，并建立相应的机制以确保能够及时建立文档。如果有关键成员离开此项目，这就是一个保证工作连续性的机制。项目管理者应该仔细监测这些文档，以保证每一个文档内容的正确性，在项目进行中有新员工加入时，能为他们提供必要的信息。

 风险管理及应急计划是以缓解工作已经失败、而且风险已经发生为先决条件的。继续前面的例子，假定项目正在进行之中，有一些人宣布将要离开。如果已经按照缓解策略行事，则有后备人员可用，信息已经文档化，有关知识已经在团队中广泛进行了交流。此外，对那

些人员充足的岗位，项目管理者还可以暂时重新调整资源（并重新调整项目进度），从而使得新加入团队的人员能够“赶上进度”。同时，应该要求那些将要离开的人员停止所有的工作，并在最后几星期进入“知识交接模式”。比如：得到视频知识，建立“注释文档”，与仍留在团队中的成员进行交流。



如果某特定风险的RE小于其风险缓解的成本，不要试图缓解该风险，而是继续监测。

值得注意的是，RMMM步骤会导致额外的项目成本。例如，花时间给每个关键技术人员配备“后备人员”得承担费用。因此，风险管理的另一个任务就是评估什么情况下由RMMM步骤所产生的效益高于实现这些步骤所需的成本。通常，项目计划者要进行典型的成本/效益分析。如果频繁的人员变动风险的缓解步骤经评估将会增加15%的项目成本和工期，而主要的成本因素是“配备后备人员”，则管理者可能决定不执行这一步骤。另一方面，如果风险缓解步骤经预测仅增加5%的项目成本和3%的工期，则管理者极有可能将这一步骤付诸实现。

对于大型项目，可以识别出30或40种风险。如果为每一个风险制定3~7个风险缓解步骤，则风险管理本身就可能变成一个“项目”！因此，我们将Pareto的80-20法则用于软件风险上。经验表明，整个项目风险的80%（即可能导致项目失败的80%的潜在因素）能够由仅仅20%已经识别出的风险来说明。早期风险分析步骤中所做的工作能够帮助计划者确定哪些风险在这20%中（如导致高风险显露度的风险）。因此，某些已经识别、评估和预测过的风险可能并不纳入RMMM计划之中——它们不属于那关键的20%（具有最高项目优先级的风险）。

WebRef

来自ACM论坛的有关风险的大量文档见catless.ncl.ac.uk/Risks。

风险并不仅限于软件项目本身。在软件已经成功开发并交付给客户之后，仍有可能发生风险。这些风险一般与该领域中的软件缺陷相关。

软件安全和灾难分析[LEV95]是一种软件质量保证活动（第26章），主要用来识别和评估可能对软件产生负面影响并促使整个系统失效的潜在灾难。如果能够在软件工程过程的早期阶段识别灾难，就可以指定软件设计特性来消除或控制这些潜在的灾难。

739

25.7 RMMM计划

风险管理策略可以包含在软件项目计划中，也可以将风险管理步骤组织成一个独立的风险缓解、监测和管理计划（RMMM计划）。RMMM计划将所有风险分析工作文档化，项目管理者也将其作为整个项目计划的一部分。

某些软件团队并不建立正式的RMMM文档，而是将每个风险分别使用风险信息表单（risk information sheet, RIS）[WIL97]进行文档化。在大多数情况下，RIS采用数据库系统进行维护，这样容易完成创建、信息输入、优先级排序、查找以及其他分析。RIS的格式如图25-4所示。

建立了RMMM计划，而且项目已经启动之后，风险缓解及监测步骤也就开始了。正如我们前面讨论过的，风险缓解是一种问题回避活动，而风险监测则是一种项目跟踪活动，它有三个主要目的：（1）评估所预测的风险是否真正发生了；（2）保证正确地实施了各风险的缓解步骤；（3）收集能够用于今后风险分析的信息。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，风险监测的另一个任务就是试图确定“起源”（在整个项目中是哪些风险引起了哪些问题）。

风险信息表单			
风险标识号: P02-4-32	日期: 5/9/04	概率: 80%	影响: 高
描述: 事实上, 计划可复用的软件构件中只有70%将集成到应用系统中, 其他功能必须定制开发。			
精化/环境: 子条件1: 某些可复用构件是由第三方开发的, 没有其内部设计标准相关资料。 子条件2: 构件接口的设计标准尚未确定, 有可能和某些现有的软件可复用构件不一致。 子条件3: 某些可复用构件是采用不支持目标环境的语言开发的。			
缓解/监测: 1. 与第三方交流以确定其与设计标准的符合程度。 2. 强调接口标准的完整性, 在确定接口协议时应考虑构件的结构。 3. 检查并确定属于子条件3的构件数量, 检查并确定是否能够获得语言支持。			
管理/应急计划/触发: RE的计算结果为\$20 200。在项目应急计划中分配这些费用。修订进度表, 假定必须定制开发18个附加构件, 据此分配人员。 触发: 缓解步骤自7/1/04起没有效果。			
当前状态: 5/12/04: 缓解步骤启动。			
创建者: D.Gagne	受托者: B.Laster		

740

图25-4 风险信息表单[WIL97]

SOFTWARE TOOLS

风险管理

目的: 风险管理工具的目的是辅助项目团队识别风险, 评估风险的影响及发生的概率, 并在整个软件项目过程中跟踪风险。

机制: 通常, 风险管理工具能够提供典型的项目和商业风险列表来辅助识别一般风险; 能够提供检查表或其他“接口”技术来辅助识别项目特定的风险; 能够指定每一个风险发生的概率及影响; 支持风险缓解策略; 能够生成多种不同的风险相关报告。

代表性工具⁴

Riskman, 由亚利桑那州立大学 (www.eas.asu.edu/~sdm/merrill/riskman.html) 开发, 是能够识别项目相关风险的一个风险评估专家系统。

Risk Radar, 由SPMN (www.spmn.com) 开发, 能够辅助项目管理者识别和管理项目风险。

RiskTrak, 由RST (www.risktrac.com) 开发, 在整个软件项目过程中, 能够提供风险识别、分析、报告和管理。

Risk+, 由C/S Solutions (www.CSolutions.com) 开发, 与Microsoft项目集成, 能够对成本和进度不确定性进行量化。

X:PRIMER, 由Grafp Technologies (www.grafp.com) 开发, 是通用的基于Web的工具, 可预测项目中什么可能出错, 并能够识别出潜在错误的根本原因和有效的应对措施。

⁴ 这里记录的工具并不代表本书支持这些工具, 而只是此类工具的例子。在大多数情况下, 工具的名字由各自的开发者注册为商标。

25.8 小结

对软件项目期望很高时，一般都会进行风险分析。不过，即使进行这项工作，大多数软件项目管理者都是非正式和表面上完成它。花在识别、分析和管理风险上的时间可以从多个方面得到回报：更加平稳的项目进展过程；较高的跟踪和控制项目的能力；在问题发生之前已经做了周密计划而产生的信心。

风险分析需要占用大量项目计划的工作量。识别、预测、评估、管理和监测都要花费时间，但这是值得的。引用中国2500多年前的军事家孙子的一句话：“知己知彼，百战不殆”。对于软件项目管理者而言，这个“彼”指的就是风险。

741

参考文献

- [AFC88] *Software Risk Abatement*, AFCS/AFLC Pamphlet 800-45, U.S. Air Force, September 30, 1988.
- [BOE89] Boehm, B. W., *Software Risk Management*, IEEE Computer Society Press, 1989.
- [CHA89] Charette, R. N., *Software Engineering Risk Analysis and Management*, McGraw-Hill/Intertext, 1989.
- [CHA92] Charette, R. N., "Building Bridges over Intelligent Rivers," *American Programmer*, vol. 5, no. 7, September, 1992, pp. 2-9.
- [DRU75] Drucker, P., *Management*, W. H. Heinemann, 1975.
- [GIL88] Gilb, T., *Principles of Software Engineering Management*, Addison-Wesley, 1988.
- [GLU94] Gluch, D. P., "A Construct for Describing Software Development Risks," CMU/SEI-94-TR-14, Software Engineering Institute, 1994.
- [HAL98] Hall, E. M., *Managing Risk: Methods for Software Systems Development*, Addison-Wesley, 1998.
- [HIG95] Higuera, R. P., "Team Risk Management," *CrossTalk*, U.S. Dept. of Defense, January 1995, pp. 2-4.
- [KAR96] Karolak, D. W., *Software Engineering Risk Management*, IEEE Computer Society Press, 1996.
- [KEI98] Keil, M., et al., "A Framework for Identifying Software Project Risks," *CACM*, vol. 41, no. 1, November 1998, pp. 76-83.
- [LEV95] Leveson, N. G., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
- [SEI93] "Taxonomy-Based Risk Identification," Software Engineering Institute, CMU/SEI-93-TR-6, 1993.
- [THO92] Thomsett, R., "The Indiana Jones School of Risk Management," *American Programmer*, vol. 5, no. 7, September 1992, pp. 10-18.
- [WIL97] Williams, R. C., J. A. Walker, and A. J. Dorofee, "Putting Risk Management into Practice," *IEEE Software*, May 1997, pp. 75-81.

习题与思考题

- 25.1 为图25-2中的三个风险建立风险监测策略及特定的风险监测活动。确保识别出了你将监测的风险因素以确定风险正在变大或变小。
- 25.2 你受命开发一个支持低成本的视频编辑系统的软件。该系统输入数字视频信号，将视频信息存在磁盘上，然后允许用户对数字化的视频信息进行各种编辑，最后生成DVD或其他多媒体格式。对这类系统做一些调研，然后列出当你开始启动这类项目时，将面临的技术风险是什么。
- 25.3 为SEPA站点给出的每个风险条目检查表增加三个其他的问题或主题。
- 25.4 为图25-2所描述的风险中的三个建立风险缓解策略及特定的风险缓解活动。

742

- 25.5 举出5个其他领域的例子来说明与被动风险策略相关的问题。
- 25.6 简述“已知风险”和“可预测风险”之间的差别。
- 25.7 简述风险因素和风险驱动因子之间的差别。
- 25.8 假设你是一家大型软件公司的项目管理者，你受命带领团队开发“下一代”字处理软件。请为该项目建立一个风险表。
- 25.9 精化图25-2中的三个风险，并为每个风险建立风险信息表单。
- 25.10 你能否想到一种情况：一个高概率、高影响的风险并未纳入RMMM计划的考虑之中？
- 25.11 用CTC格式表示图25-2中的三个风险。
- 25.12 假定每个LOC成本为\$16，概率为60%，重新计算25.4.2节中讨论的风险显露度。
- 25.13 给出主要关注软件安全和灾难分析的五个软件应用领域。

推荐读物与阅读信息

近几十年来，软件风险管理方面的文献已有很多。DeMarco和Lister写了一本有趣而意义深刻的书（《Dancing with Bears》，Dorset House，2003），该书可以指导软件管理者和开发者进行风险管理。Moynihan（《Coping with IT/IS Risk Management》，Springer-Verlag，2002）介绍了项目风险管理者们所采用的实用方法。Royer（《Project Risk Management》，Management Concepts，2002）及Smith和Merritt（《Proactive Risk Management》，Productivity Press，2002）论述了项目管理的主动过程。Karolak撰写了一本参考书（《Software Engineering Risk Management》，Wiley，2002），书中介绍了一种便于使用的风险分析模型，以及一些有价值的检查表和调查表软件包。

Schuyler（《Risk and Decision Analysis in Projects》，PMI，2001）从统计学的观点讨论了风险分析。Hall（《Managing Risk：Methods for Software Systems Development》，Addison-Wesley，1998）对该主题作了更全面的讨论。Myerson（《Risk Management Processing for Software Engineering Models》，Artech House，1997）讨论了度量、安全、过程模型及其他相关主题。Grey撰写了一本有用的风险评估小册子（《Practical Risk Assessment for Project Management》，Wiley，1995），为该主题提供了一个很好的导论。

Capers Jones（《Assessment and Control of Software Risks》，Prentice-Hall，1994）对软件风险进行了详细讨论，其中包含从数百个软件项目中收集的数据，Jones定义了60个可能影响软件项目结果的风险因素。Boehm[BOE89]给出了很好的调查表和检查表格式，对风险识别具有重大作用。Charette[CHA89]描述了风险分析方法的详细处理，采用了概率论和统计技术来分析风险。Charette的另一本书（《Application Strategies for Risk Analysis》，McGraw-Hill，1990）从系统和软件工程的角度讨论风险，并提出了实用的风险管理策略。Gilb（《Principles of Software Engineering Management》，Addison-Wesley，1988）提出了一组“原则”（通常是有趣的，有时是深刻的），可作为风险管理的有效指南。

Ewusi-Mensah（《Software Development Failures：Anatomy of Abandoned Projects》，MIT Press，2003）及Yourdon（《Death March》，Prentice-Hall，1997）讨论了当软件项目风险发生时会带来什么后果。Bernstein（《Against the Gods》，Wiley，1998）描述了有趣的远古时期的风险历史。

美国卡内基·梅隆大学软件工程研究所（SEI）已经出版了很多关于风险分析和风险管理的详细报告和参考书。美国空军司令部手册AFSCP 800-45[AFC88]描述了风险识别和减少技术。《ACM Software Engineering Notes》每期都有题为“Risk to the Public”（编辑：P. G. Neumann）的讨论，如果你想知道最新和最好的软件恐怖故事，这里就有。

因特网上有大量与软件风险管理相关的信息资源，最新的WWW参考列表可以在SEPA站点<http://www.mhhe.com/pressman>找到。

743



第26章 质量管理

要点浏览

概念：仅仅嘴上说“软件质量是很重要的”是不够的。你必须：(1) 明确给出你所说的“软件质量”的涵义；(2) 建立一组活动，这些活动将有助于保证每一个软件工程工作产品表现出高质量；(3) 对每个软件项目实施质量控制和质量保证活动；(4) 运用度量技术来制定软件过程改进的策略，进而提高最终产品的质量。

人员：软件工程过程涉及的每一个人都要对质量负责。

重要性：你要么一次做好，要么重做。如果一个软件团队在每个软件工程活动中都强调质量，则会减少返工量，进而降低成本，更重要的是可以缩短面市时间。

步骤：在软件质量保证活动启动前，必须按照多种不同的抽象层次来定义“软件质量”。理解了质量的涵义之后，软件团队必须确定一组SQA活动来过滤掉工作产品中的错误，以免这些错误再继续传递下去。

工作产品：为了制定软件团队的SQA策略，需要建立“软件质量保证计划”。在分析、设计、代码生成阶段，主要的SQA工作产品是正式技术评审总结报告；在测试阶段，是制定的测试计划和测试规程。也可能产生其他与过程改进相关的工作产品。

质量保证措施：在错误变成缺陷之前发现它！也就是说，尽量提高缺陷排除效率（第22章），进而减少软件团队不得不付出的返工量。

本书中所讨论的软件工程方法只有一个目标，那就是：生产出高质量的软件。但是很多读者都会遇到这样一个问题：什么是软件质量？

Philip Crosby[CRO79]在他有关质量的著作中间接地回答了这个问题：

质量管理的问题不在于人们不知道什么是质量，而在于人们自认为知道什么是质量……

在这一点上，质量与“性”的相同点颇多。每个人都需要它（当然，是在特定条件下），每个人都觉得自己理解它（尽管人们不愿意解释它），每个人都认为只需遵从自然规律就能得到它（不管怎样，我们都还做得不错）。当然，大多数人都认为这一领域的问题都是由他人引起的。（只要他们肯花时间就能把事情做好。）

有些软件开发者仍然相信软件质量是在编码完成之后才应该开始担心的事情。这是完全错误的！质量管理（也称为软件质量保证）是适用于整个软件过程的一种普适性活动（第2章）。

质量管理包括：(1) 软件质量保证(SQA)过程；(2) 特定的质量保证和质量控制任务（包括正式技术评审和多层次测试策略）；(3) 有效的软件工程实践（方法和工具）；(4) 对所

关键概念
质量成本
缺陷放大
缺陷成本
ISO 9001:2000
可靠性
正式技术评审
质量
质量控制
抽样
六西格玛
软件安全
SQA计划
基于统计的SQA

有软件工作产品及其变更的控制（第27章）；（5）保证符合软件开发标准的规程（在适用时）；（6）测量和报告机制。

在本章中，我们将重点讨论使软件组织能够保证“在恰当的时间、以正确的方式、做正确的事情”的管理问题和特定的过程活动。

26.1 质量概念¹



POINT
控制差异是获得高质量产品的关键。对软件而言，我们力争去控制我们所应用的通用过程中的差异，以及渗透到软件工程工作中的质量重要性的差异。

差异控制是质量控制的核心。制造商希望尽可能减小生产的产品之间的差异，即使像复制DVD盘这样比较简单的工作也不例外。当然，这并不成问题——DVD盘复制只不过是一项微不足道的制造操作，我们可以保证总是能够创建完全相同的软件副本。

但是，我们真的能够做到吗？我们需要保证DVD盘上的磁道都在所规定的容错范围之内，这样才能保证绝大多数的DVD驱动器能够正确读出这些信息。磁盘复制机的确会磨损和超出容错范围，因此，即使像DVD盘复制这样“简单”的过程也会遇到样本差异问题。

但是，如何将这项工作应用于软件工作？软件开发组织应该怎样控制差异呢？对于每个不同的项目，我们希望尽可能减小完成项目预计需要的资源和实际使用的资源之间的差异，包括人员、设备和时间。一般来说，对于软件发布的不同版本，我们希望确保测试程序对软件覆盖达到大家普遍认可的百分比，这不仅是因为我们希望尽可能地减少所发布产品中的缺陷数量，而且还因为我们希望能够保证不同版本之间的缺陷数量差异也保持最小。（如果产品的第三个发布版本中缺陷数量十倍于此前的版本，我们的客户将会感到失望。）我们也希望自己的热线服务在答复不同客户的问题时，速度和准确程度的差异尽可能减小，等等。

745

26.1.1 质量

《American Heritage Dictionary》（美国传统字典）中对质量的定义是：“某一事物的特征或属性”。作为一个事物的属性，质量指的是可测量的特征——与已知标准可以进行比较，如长度、颜色、电气特性、可延展性等等。但是，软件在很大程度上是一种知识实体，其特征的定义远比物理对象要困难得多。

然而，确实存在程序特征的测量，这些特征属性包括循环复杂度、内聚性、功能点数量、代码行数，以及其他很多在第15章中讨论的属性。当根据对象的可测量特征考察一个对象时，可以有两种不同的质量：设计质量和一致性质量。

设计质量是指设计者为一个产品规定的特征。一致性质量是指在制造产品的过程中遵守设计规格说明的程度。

“人们会忘记你做一件工作有多快——但他们总会记得你做得有多好。”

——Howard Newton

¹ 本节由Michael Stovsky编写，根据“Fundamentals of ISO 9000”——这是由R. S. Pressman & Associates, Inc. 为音像教程《Essential Software Engineering》开发的工作手册——改编。翻印得到授权。

在软件开发中，设计质量包括系统的需求、规格说明和设计。而一致性质量主要关注实现问题。如果实现过程符合设计要求，并且开发的系统满足需求和性能目标，则一致性质量较高。

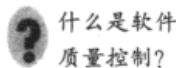
但是，设计质量和一致性质量是软件工程师必须考虑的唯一问题吗？Robert Glass [GLA98]认为它们之间比较“直观的”关系符合下面的公式：

$$\text{用户满意度} = \text{合格的产品} + \text{好的质量} + \text{按预算和进度安排交付}$$

实际上，Glass认为质量是重要的。但是，如果用户不满意，其他任何事情也就都不重要了。DeMarco [DEM99]同意这个观点，他认为：“产品的质量是一个函数，该函数确定了它在多大程度上使这个世界变得更好。”这个质量观点的意思就是：如果一个软件产品能给最终用户带来实质性的益处，他们可能会心甘情愿地忍受偶尔的可靠性或性能问题。

26.1.2 质量控制

746



差异控制可以等同于质量控制。但我们如何实现质量控制呢？质量控制就是为了保证每一件工作产品都能满足对它的需求而在整个软件过程中所运用的一系列审查、评审和测试。质量控制在创建工作产品的过程中还包含一个反馈循环。测量和反馈相结合，使得我们能够在得到的工作产品不能满足其规格说明时调整开发过程。

质量控制中的关键概念之一就是所有工作产品都具有明确的和可测量的规格说明，我们可以将每个过程的产品与这一规格说明进行比较。反馈循环对于减少产生的缺陷是至关重要的。

26.1.3 质量保证

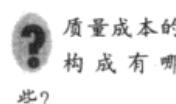
WebRef

大量有关SQA的信息源在
www.qualitytree.com/links/index.htm可找到。

质量保证由评估质量控制活动有效性和完整性的一系列审核和报告功能构成。质量保证的目的是：为管理层提供了解产品质量所必需的数据，从而获得产品质量是否符合预定目标的认识和信心。当然如果质量保证所提供的数据发现了问题，则管理层负责解决这些问题，并为解决这些质量问题分配所需资源。

26.1.4 质量成本

质量成本包括所有由质量工作或者进行与质量有关的活动所引发的成本。进行质量成本研究不仅能够为当前质量成本设定基线，还能确定降低质量成本的时机，以及提供规范化的对照标准。规范化的对照标准几乎全都以“美元”计算。一旦我们将质量成本以“美元”为单位进行了规范化，我们就获得了必要的数据来估算改进现有过程的时机。我们还可以进一步按“美元”估算变更将产生的影响。



质量成本可以细分为预防成本、鉴定成本及失效成本。预防成本包括质量计划、正式技术评审、测试设备及培训。鉴定成本包括为深入了解“首次通过”各个过程时的产品状态而开展的那些活动。鉴定成本的例子如：过程中和过程间的审查，设备校准和维护，测试。

失效成本是指如果在将产品交付给客户之前没有缺陷的话就不会存在的成本。失效成本可以进一步细分为内部失效成本和外部失效成本。内部失效成本是指在产品交付给客户之前发



不要害怕引入大量的预防成本，这样可以保证你的投资会获得很好的回报。

现了错误而引发的成本。内部失效成本包括：返工、修复以及失效模式分析。外部失效成本是指与产品交付给客户之后所发现的缺陷相关的成本。外部失效成本的例子有：解决客户抱怨、退换产品、求助电话支持以及保修工作。

正如我们所预料的，发现和改正一个缺陷的相对成本随着我们从预防到检测、从内部失效到外部失效而急剧增加。图26-1根据 Boehm [BOE81]和其他人所收集的数据说明了这一现象。[747]

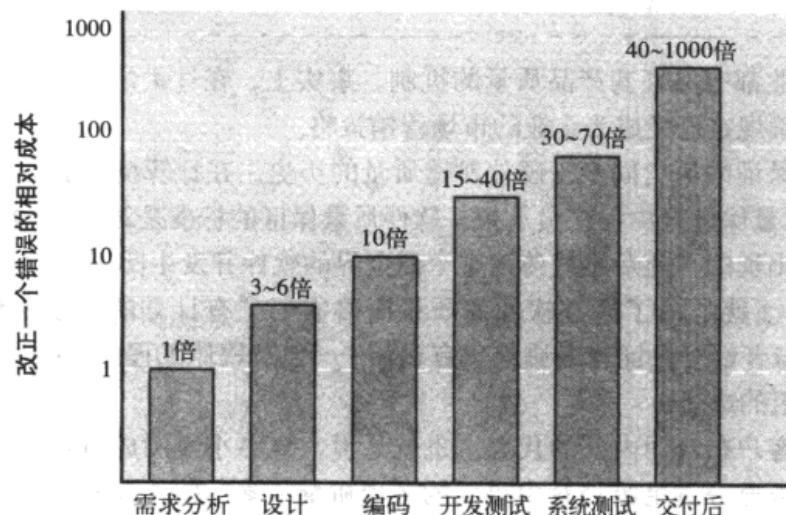


图26-1 改正一个错误的相对成本

“正确地完成一件事情所花的时间比解释你为什么将事情做错所花的时间要少。”

——H. W. Longfellow

26.2 软件质量保证

即使最缺乏热情的软件开发者也会同意：生产高质量的软件是一个十分重要的目标。但我们应该如何定义质量呢？有人曾打趣说：“每个程序都能够做好一些事情，只不过不是我们希望它做到的那些而已。”

文献中对软件质量的定义有很多种。在我们这里对软件质量做如下定义：



我们如何定义
软件质量？

软件要符合明确规定功能和性能需求，符合已清晰文档化的开发标准，并且具有专业人员开发的软件所应有的隐含特征。

显然上述定义还可以修改或扩充。实际上，对软件质量的定义可以无休止地争论下去。在本书中，上述定义强调了以下三个重要方面：

1. 软件需求是进行质量测量的基础，与需求不符就是质量不高。
2. 规定的标准定义了一组指导软件开发的准则。如果不能遵照这些准则，就极有可能导致质量不高。
3. 通常有一些“隐含需求”是不会被明确提出的（如对易用性和易维护性的需求）。如果软件符合了明确提出的需求，却没有满足隐含的需求，软件质量仍然值得怀疑。

26.2.1 背景

对于任何为他人生产产品的企业来说，质量控制和质量保证都是必不可少的活动。在20世纪以前，质量控制只由生产产品的工匠承担。第一个正式的质量保证和质量控制方案于1916年由贝尔实验室提出，此后迅速风靡整个制造行业。在20世纪40年代，出现了更正式的质量控制方法，这些方法都将测量和持续的过程改进[DEM86]作为质量管理的关键成分。

“你已经犯下了太多的错误。”

——Yogi Berra

今天，每个企业都有保证其产品质量的机制。事实上，在过去的几十年中，企业在重视质量方面所做的明确规定已经成为企业的市场营销策略。

软件开发质量保证的历史同步于硬件制造质量的历史。在计算机发展的早期（20世纪50年代和60年代），质量保证只由程序员承担。软件质量保证的标准是20世纪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界的软件开发中[IEE94]。延伸前述的质量定义，软件质量保证就是为了保证软件高质量而必需的“有计划的、系统化的行动模式”[SCH98]。各个参与者都对保证软件质量负有责任——包括软件工程师、项目管理者、客户、销售人员和SQA小组的成员。

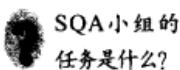
SQA小组充当客户在公司内部的代表。也就是说，SQA小组的成员必须从客户的角度来审查软件。软件是否充分满足第15章中提出的各项质量因素？软件开发是否依照预先制定的标准进行？作为SQA活动一部分的技术规范是否恰当地发挥了作用？SQA小组的工作将回答上述这些问题以及其他问题，以确保软件质量得到维持。

26.2.2 SQA活动

软件质量保证由很多任务组成，这些任务分别与两类不同的参与者相关——一类是做技术工作的软件工程师，另一类是负责质量保证计划、监督、记录保存、分析和报告工作的SQA小组。
749

软件工程师通过运用可靠的技术方法和措施、进行正式的技术评审和执行计划周密的软件测试来解决质量问题（并完成质量保证和质量控制活动）。在本章中只讨论评审问题，其他有关的技术问题在本书中的第一、二、三部分和第五部分进行讨论。

SQA小组的职责是辅助软件团队实现高质量的软件产品。美国卡内基·梅隆大学软件工程研究所（SEI）建议了一组有关质量保证计划、监督、记录保存、分析和报告工作的SQA活动。这些活动都是由一个独立的SQA小组完成（或推动）的，包括以下工作：



为项目准备SQA计划。 SQA计划是在制定项目计划时制定的，并且由所有共利益者进行评审。由软件工程团队和SQA小组执行的质量保证活动将受该计划控制。该计划规定了：需要进行的评估，需要进行的审核和评审，项目可采用的标准，错误报告和跟踪规程，SQA小组需要编写的文档，以及需要为软件项目团队提供的一些反馈信息。

参与开发项目的软件过程描述。 软件团队为需要进行的工作选择一个过程。SQA小组将评审该过程描述，以确保其与组织方针、内部软件标准、外界强制推行的标准（如ISO-9001）以及软件项目计划的其他部分相符。

评审各项软件工程活动，以验证其是否符合定义的软件过程。SQA小组识别、记录和跟踪与该软件过程的偏差，并验证是否已经得到改正。

审核指定的软件工作产品，以验证其是否符合定义的软件过程中的相应部分。SQA小组对选出的工作产品进行评审，识别、记录和跟踪出现的偏差，验证是否已经得到改正，并定期向项目管理者报告工作结果。

确保软件工作及工作产品中出现的偏差已文档化，并且按照文档化的规程进行了处理。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。

记录所有不符合的部分，并报告给高层管理者。应跟踪不符合的部分，直至问题得到解决。

除进行上述活动之外，SQA小组还要协调变更控制和变更管理（第27章），并帮助收集和分析软件度量信息。

750

26.3 软件评审



评审就像软件工作流中的过滤器。太少了，则工作流是“脏的”；太多了，则工作流会变成湍流。通过度量来确定什么评审有效，并重点进行这些评审。

软件评审是软件过程中的“过滤器”。也就是说，在软件工程过程的不同阶段进行软件评审，可以起到发现错误和缺陷，进而消除它们的作用。软件评审还能够“净化”分析、设计和编码等软件工程活动。对于评审的需要，Freedman和Weinberg [FRE90]是这样论述的：

技术工作需要评审的第一个理由就像铅笔需要橡皮：人非圣贤，孰能无过。我们需要技术评审的第二个理由是：尽管人们善于发现自己的某些错误，但是许多情况下犯错误的人发现自己错误的能力远小于其他人。

在软件工程过程中可以进行的评审有很多种，它们各有各的作用。在休息室里讨论技术问题的非正式会见就是一种评审方式。将软件设计正式介绍给客户、管理层和技术人员也是一种评审方式。但是在本书中，我们将重点讨论正式技术评审（FTR），有时也称为走查（walkthrough）或审查（inspection）。从质量保证的角度出发，正式技术评审是最有效的过滤器。由软件工程师（以及其他）对软件工程师进行的正式技术评审是一种发现错误和提高软件质量的有效手段。

INFO

隐错、错误和缺陷

SQA的目标是消除软件中存在的质量问题。有很多术语可用来描述这种质量问题，如“隐错”（bug）、“故障”（fault）、“错误”（error）或“缺陷”（defect）。它们的含义是相同的吗？还是存在微小的差别呢？

在本书中，我们明确指出了“错误”（指在软件交付给最终用户之前发现的质量问题）和“缺陷”（指在软件交付给最终用户之后发现的质量问题²）的差别。这样区别的原因是“错误”和“缺陷”对经济、商业、心理和人员的影响有很大区别。作为软件工程师，我们期望在客户和（或）最终用户遇到问题之前尽可能多地发现并改正“错误”。我们同样

² 如果考虑的是软件过程改进，在过程框架活动之间（如从建模到构造）传递的质量问题同样可以叫做“缺陷”，因为在一个工作产品（如设计模型）“交付”给下一个活动之前就应该发现这个问题。

期望避免“缺陷”——因为“缺陷”（有理由）使软件工作者显得水平低下。

然而要指出的是，本书中描述的“错误”和“缺陷”的差别并不是主流观点。在软件工程领域中，大多数人都认为“缺陷”、“错误”、“故障”和“隐错”是没有差别的。也就是说，遇到问题的时间与采用哪个术语描述毫无关系。这个观点中争论的焦点是：有些时候很难明确地区分是交付之前还是交付之后（如敏捷开发中采用的增量过程——参见第4章）。

不管怎么说都应该认识到：发现问题的时间点是非常关键的。软件工程师应该努力再努力，力图在他们的客户和最终用户遇到问题之前将其发现。有关“隐错”术语的讨论，感兴趣的读者可从www.softwaredevelopment.ca/bugs.shtml找到。

751

26.3.1 软件缺陷对成本的影响



FTR的主要目标
是在错误传递
到下一个软件
工程活动或发
布给最终用户
之前发现它。

正式技术评审的主要目标是在软件过程中发现错误，以使它们不会在软件交付之后变成缺陷。正式技术评审最明显的优点就是可以早些发现错误，以防止将错误传递到软件过程的后续阶段。

产业界(TRW、NEC、Mitre Corp.，以及其他公司)的大量研究表明：设计活动引入的错误占软件过程中出现的所有错误（和最终的所有缺陷）数量的50%~65%。而现有研究[JON86]表明：正式技术评审在发现设计错误方面最高可达到75%的有效性。通过检测和消除大量设计错误，评审过程将极大降低软件过程后续活动的成本。

为了说明尽早进行错误检测对成本产生的影响，我们根据从大型软件项目中收集到的实际成本数据[IBM81]来推导一系列相对成本³。假定在设计阶段发现一个错误的改正成本为1.0个货币单位，则同一个错误在测试开始之前发现的改正成本为6.5个货币单位，在测试过程中发现的改正成本为15个货币单位，而在发布之后发现的改正成本将达到60~100个货币单位。

26.3.2 缺陷放大和消除

可以用“缺陷放大模型”[IBM81]来说明在软件工程过程的初步设计、详细设计和编码阶段中错误的产生和检测过程。该模型如图26-2所示，其中最外层方框代表一个软件开发步骤。在该开发步骤中，错误可能由于疏忽而产生，在评审过程中可能没有发现新产生的错误以及来自前面步骤的错误，从而导致一定数量的错误通过了当前步骤。在某些情况下，从前面步骤传过来的错误在当前步骤中会被放大（放大倍数为x）。将开发步骤方框进一步细分可以说明这些特点及错误检测的有效性百分比，错误检测的有效性百分比是评审完善性的函数。

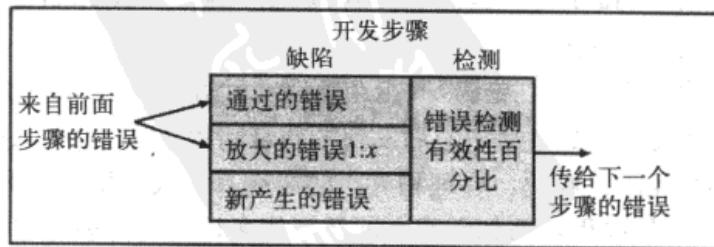


图26-2 缺陷放大模型

³ 虽然这些数据是20年前的，但是在现在的环境中仍然是有效的。

752

“正如医生所说，某些疾病在其初发阶段是易于治愈的，但是难于识别……然而，如果它们没有在早期被发现和治疗，随着时间的推移将变得易于识别而难于治愈。”

—Niccolo Machiavelli

图26-3是一个假设在软件过程中不进行任何评审的缺陷放大的例子。如图中所示，假设每一个测试步骤都能够发现和改正50%的输入错误，而且又不引入任何新的错误（乐观估计）。10个初步设计阶段的错误在测试开始之前就能放大成为94个错误，12个潜伏的缺陷则随软件发布进入客户现场。图26-4的情况与图26-3类似，只是在设计和编码开发步骤中引入了评审。在这种情况下，最初的10个初步设计错误在测试开始之前放大成为24个错误，最后只有3个潜伏的缺陷。回忆一下与发现和改正错误相关的相对成本，将图26-3和图26-4每一个步骤中发现的错误数量乘以消除一个错误所需要的成本（设计是1.5个成本单位，测试前是6.5个成本单位，测试中是15个成本单位，发布后是67个成本单位），由此可以确定总开发成本（对我们假设的例子而言是有或没有评审情况下的成本）。通过这些数据，在进行了评审的情况下，开发和维护的总成本是783个成本单位，而在不进行评审的情况下，总成本是2177个成本单位——几乎是前者的3倍。

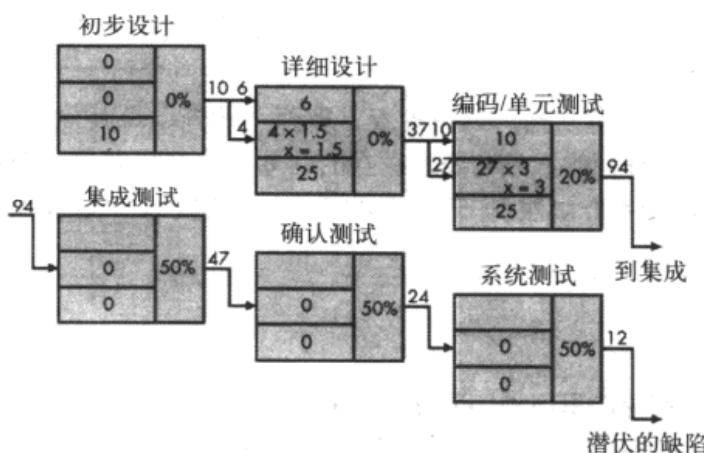


图26-3 缺陷放大——无评审

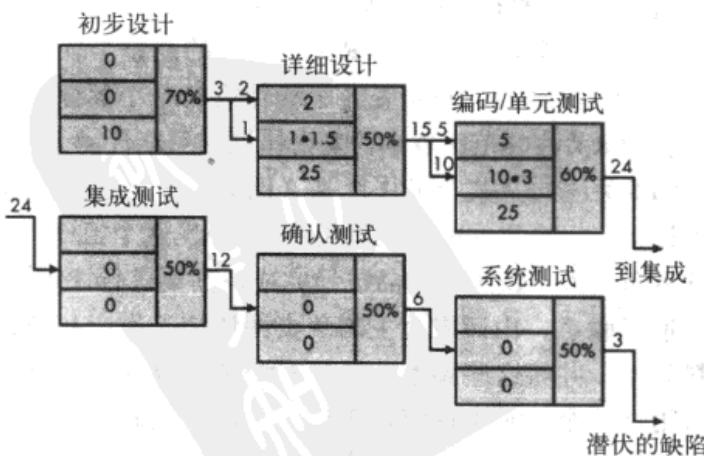


图26-4 缺陷放大——有评审

为了进行评审，软件工程师必须花费时间和精力，开发组织也必须提供相应费用。然而，上述例子的结果已经证明了我们面临的选择是：要么现在付出，否则以后会付出更多。正式技术评审（在设计或其他技术活动中）带来了显而易见的成本效益。因此，应该进行正式技术评审。

26.4 正式技术评审

我们什么时候进行FTR,
我们的目标是什么？

正式技术评审（FTR）是一种由软件工程师（以及其他人）进行的软件质量控制活动。FTR的目标是：（1）发现软件的任何一种表示形式中的功能、逻辑或实现上的错误；（2）验证评审中的软件是否满足其需求；（3）保证软件的表示符合预先定义的标准；（4）得到以统一的方式开发的软件；（5）使项目更易于管理。除此之外，FTR还提供了培训机会，使初级工程师能够了解软件分析、设计和实现的不同方法。由于FTR的进行，使大量人员对软件系统中原本并不熟悉的部分更为了解，因此，FTR还起到了培训后备人员和促进项目连续性的作用。

“没有任何事情能比一个人去校对另一个人的工作更令人振奋了。”——Mark Twain

FTR实际上是一组评审方式，包括走查、审查、轮查以及其他软件小组的技术评估。每次FTR都是以会议形式进行的，只有经过适当的计划、控制和参与，FTR才能获得成功。在后面的几小节中，我们将给出类似于走查的典型正式技术评审指导原则（如[FRE90]，[GIL93]）。

26.4.1 评审会议

WebRef

754

《NASA SATC 正式审查指南》可从satc.gsfc.nasa.gov/fi/fipage.html 下载。

不论选用何种FTR方式，每个评审会议都应该遵守以下条件：

- 评审会议（通常）应该由3~5人参加。
- 应该提前进行准备，但是占用每人的工作时间应该不超过2小时。
- 评审会议的时间应该少于2小时。

按照上述条件，显然FTR应该关注的是整个软件中的某个特定（且较小的）部分。比如说，只走查各个构件或者一小部分构件，不要试图评审整个设计。FTR关注的范围越小，发现错误的可能性越大。

FTR关注的是某个工作产品（例如，一部分需求规格说明、一份详细的构件设计、一个构件的源代码清单）。开发这个工作产品的个人（生产者）通知项目负责人“该工作产品已经完成，需要进行评审”。项目负责人与评审会主席取得联系，由评审会主席负责评估该工作产品是否准备就绪，制作产品材料副本，并将这些副本分发给2到3个评审者以便事先做准备。每个评审者应该用1到2个小时来评审该工作产品，通过做笔记或者其他方法熟悉该工作产品。与此同时，评审会主席也应该评审该工作产品，并制定评审会议的日程表，通常会安排在第二天开会。



FTR关注的是某
个工作产品中比较
小的一部分。



在某些情况下，让其他人而不是生产者本人来走查被评审的产品是一个很好的方法，这样可得到对工作产品的真實描述且更易于发现错误。

评审会议由评审会主席、所有评审者和生产者参加。其中一个评审者还充当记录员的角色，负责记录（书面的）在评审过程中发现的所有重要问题。FTR一般从介绍会议日程并由生产者做简单的介绍开始。然后由生产者“走查”该工作产品，并对材料做出解释，而评审者则根据预先的准备提出问题。当发现了有根据的问题或错误时，记录员逐一加以记录。

在评审结束时，所有FTR与会者必须做出以下决定中的一个：(1) 可以不经修改而接受该工作产品；(2) 由于严重错误而否决该工作产品（错误改正后必须再次进行评审）；(3) 暂时接受该工作产品（发现一些必须改正的小错误，但是不再需要进行评审）。做出决定之后，所有FTR与会者都需要签名，以表示他们参加了此次FTR，并且同意评审小组所做的决定。

26.4.2 评审报告和记录保存

在FTR期间，由一名评审者（记录员）主动记录所有提出的问题。在评审会议结束时要对这些问题进行汇总，并生成一份“评审问题清单”。此外，还要完成一份“正式技术评审总结报告”。评审总结报告中要回答以下3个问题：

1. 评审什么？
2. 由谁评审？
3. 发现和结论是什么？

755

评审总结报告通常是一页纸左右（可能还有附件）。它是项目历史记录的一部分，有可能被分发给项目负责人和其他感兴趣的参与方。

评审问题清单有两个作用：(1) 标识产品中存在问题的区域；(2) 作为行动条目检查表以指导生产者进行改正。通常将评审问题清单附在总结报告的后面。

为了保证评审问题清单中的每一条目都得到适当的改正，建立跟踪规程非常重要。只有做到这一点，才能保证提出的问题真正得到“解决”。方法之一就是将跟踪的责任指派给评审会主席。

“会议经常是几分钟的事，却浪费了几个小时。”

——作者不详

26.4.3 评审指导原则

进行正式技术评审之前必须制定评审的指导原则，并分发给所有评审者以得到大家的认可，然后才能依照它进行评审。不受控制的评审通常糟糕得跟没有评审一个样。下面列出了正式技术评审指导原则的基本要求：



不要严厉地指出错误。可以采用较温和的方式，比如问一个问题，使生产者能够发现自己的错误。

1. 评审工作产品，而不是评审生产者。FTR涉及他人和自己。如果进行得适当，FTR可以使所有参与者体会到温暖的成就感。如果进行得不适当，则可能陷入一种审问的气氛之中。应该温和地指出错误，会议的气氛应该是轻松的和建设性的，不要试图贬低或羞辱他人。
2. 制定并遵守日程表。各种类型会议的主要缺点之一就是：放任自流。必须保证FTR不要离题且按照计划进行。评审会主席负有维持会议程序的责任，在有人转移话题时应该提醒他。

3. 限制争论和辩驳。当评审者提出问题时，未必所有人都认同该问题的严重性。不要花时间去争论这类问题，这类问题应该被记录在案，留到会后进行讨论。

4. 对各个问题都发表见解，但是不要试图解决所有记录的问题。评审不是一个解决问题的会议，问题的解决应该放到评审会议之后进行。

5. 做笔记。有时候让记录员在黑板上做笔记是一种很好的方式，这样在记录员记录信息时，其他评审者可以推敲措辞，并确定问题的优先次序。

6. 限制参与者人数，并坚持事先做准备。**756** 虽然两个人比一个人好，但14个人并不一定就比4个人好。应该根据需要将参与评审的人员数量限制到最低，而且所有参与评审的小组成员都必须事先做好准备。评审会主席应该向评审者要求书面意见（以表明评审者的确对材料进行了评审）。

7. 为每个将要评审的工作产品建立检查表。检查表能够帮助评审会主席组织FTR会议，并帮助每个评审者将注意力集中到重要问题上。

8. 为FTR分配资源和时间。为了提高评审效率，应该将评审作为软件过程中的任务列入进度计划，而且还要为由评审结果所引发的不可避免的修改活动分配时间。

9. 对所有评审者进行有意义的培训。为了提高效率，所有评审参与者都应该接受某种正式培训。培训要强调的不仅有与过程相关的问题，而且还应该涉及评审的心理学方面。

10. 评审以前所做的评审。听取汇报对发现评审过程本身的问题十分有益，最早被评审的工作产品本身就是评审的指导原则。

“生活中最美好的回报之一就是：凡是努力真诚帮助别人的人，同时也是在帮助自己。”

— Ralph Waldo Emerson

由于成功的评审涉及许多变数（如参与者数量、工作产品类型、时间和长度、特定的评审方法等），软件组织应该在实验中确定何种方法最为适用。Porter和他的同事[POR95]为这类实验提供了很好的指南。

26.4.4 样本驱动评审

在理想情况下，每一个软件工程工作产品都要经过正式技术评审。但在实际的软件项目中，由于资源有限和时间不足，即使意识到评审是一种质量控制的机制，评审也常常被省略。Thelin和他的同事[THE01]是这样阐述这个问题的：

757 只有在错误查找阶段发现了多个错误，才能认为审查[FTR]是有效的。如果在制品[工作产品]中发现了多个错误，那么这个审查就是必要的；相反，如果只找到几个错误，则这个审查就是在浪费审查参与者的时间⁴。而且，延误的软件项目一般会缩短审查活动的时间，最终导致质量低下。解决方法之一是将审查活动按资源的优先级次序排序，从而将现有的资源集中到最有错误倾向的制品上。

Thelin和他的同事提出了样本驱动评审过程，在这个过程中，要对所有软件工程工作产品

⁴ 当然，也可以说，即使没有发现任何错误，通过评审也可以激励生产者注重质量。



评审要花费时间，但很值得。然而，如果时间紧迫而你又别无选择的话，也不要省略评审，最好采用样本驱动评审。

的样本进行审查，以决定哪些工作产品是最有错误倾向的，然后集中全部FTR资源，（根据抽样过程中收集的数据）只分配给那些可能具有错误倾向的工作产品。

为了提高效率，样本驱动评审过程必须对作为整个FTR的主要目标的那些工作产品进行量化。量化时一般采用以下步骤[THE01]：

1. 审查每个软件工作产品*i*的一小部分，记做 a_i ，记录在 a_i 中发现的缺陷数量 f_i 。
2. 用 f_i 除以 a_i 可得到在工作产品*i*中缺陷数量的粗略估算值。
3. 按照缺陷数量粗略估算值的递减次序排列这些工作产品。
4. 将现有的评审资源集中到那些具有最高缺陷数量估算值的工作产品上。

从工作产品中抽样的这一小部分必须：(1) 能代表整个工作产品；(2) 对进行抽样的评审者来说足够大，而且有意义。当 a_i 扩大时，样本有效代表工作产品的可能性也随之增长，而进行抽样所需要的资源也随之增长。开发各种类型的工作产品时，软件工程团队必须确定 a_i 的最佳取值⁵。

SAFEHOME

SQA问题

[场景] Doug Miller的办公室，SafeHome软件项目刚开始的时候。

[人物] Doug Miller (SafeHome软件工程团队经理) 以及软件工程团队的其他成员。

[对话]

Doug: 我知道，过去我们没有花时间去为这个项目建立SQA计划。但是现在项目已经启动，我们必须考虑质量……对吗？

Jamie: 是的。我们已经决定了，在建立需求模型[第7章和第8章]的时候，Ed答应负责为每一个需求建立V&V规程。

Doug: 那太好了，可是我们不会等到测试的时候才来评估质量吧，是不是？

Vinod: 不会！当然不会。我们已经将评审的进度安排纳入到这个软件增量的项目计划之中了。我们将通过评审开始质量控制。

Jamie: 我有点担心我们没有足够的时间来进行所有的评审。事实上，我也知道会这样。

Doug: 嗯。那你觉得该怎么办呢？

Jamie: 我认为我们应该选择SafeHome分析和设计模型中最关键的元素并评审它们。

Vinod: 可是如果我们遗漏了模型中某个部分而没有评审，怎么办？

Shakira: 我阅读了有关抽样技术方面的资料[26.4.4节]，也许可以帮助我们命中应评审的元素。(Shakira描述了这种方法。)

Jamie: 也许……但是，我也不可能肯定我们是否有时间对模型中的每个元素进行抽样。

Vinod: Doug，你想让我们怎么做？

Doug: 参照极限编程[第4章]。我们结对儿（两个人）找出每个模型中的元素，并同时进行非正式的评审。之后我们将能够命中“关键”元素，以进行更正式的团队评审，但是应将这样的评审减到最少。这样的话，所有的事情都不止一组人员检查过了，但我们的

758

⁵ Thelin和他的同事已经进行了详细的模拟，这些有助于确定 a_i 的取值，详见[THE01]。

交付日期仍然不会变。

Jamie: 就是说我们必须重新调整进度。

Doug: 就是这样的。在这个项目上质量高于进度。

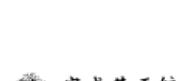
26.5 SQA的形式化方法

在过去的20年中，软件界中有一群人——虽然不多但是很坚决，提出软件质量保证应该采用一种更为形式化的方法。一个计算机程序可以看作是一个数学对象[SOM01]，每一种程序设计语言都有一套定义严格的语法和语义，而且软件需求规格说明也有严格的方法（第28章）。如果需求模型（规格说明）和程序设计语言都以严格的方式表示，就可以采用程序正确性证明来说明程序是否严格符合它的规格说明。

程序正确性证明（第28章和第29章）并不是什么新的思路。Dijkstra [DIJ76]和Linger、Mills、Witt [LIN79]以及其他很多人都倡导程序正确性证明，并将它与结构化程序设计概念联系在一起（第11章）。

26.6 基于统计的软件质量保证

基于统计的质量保证反映了一种在产业界不断增长的趋势：质量的量化。对于软件而言，基于统计的质量保证包含以下步骤：



759

完成基于统

1. 收集和分类软件缺陷信息。
2. 追溯每个缺陷的形成原因（例如，不符合规格说明、设计错误、违背标准、缺乏与客户的交流）。
3. 使用Pareto原则（80%的缺陷可以追溯到所有可能原因中的20%），将这20%（“重要的少数”）原因分离出来。
4. 一旦找出这些重要的少数原因，就可以开始纠正引起缺陷的问题。

“基于统计的质量保证”这个比较简单的概念代表的是创建自适应软件过程的一个重要步骤，要对软件过程进行修改，以改进那些引入错误的过程元素。

“20%的代码中包含了80%的错误，把它们找出来，予以纠正。” ——Lowell Arthur

26.6.1 一个普通的例子

举一个例子来说明统计方法在软件工程工作中的应用。假定软件开发组织收集了为期一年的缺陷信息，其中有些缺陷是在软件开发过程中发现的，有些缺陷则是在软件交付给最终用户之后发现的。尽管发现了数以百计的不同类型的缺陷，但是所有缺陷都可以追溯到下述原因中的一个（或几个）：

- 不完整或错误的规格说明（IES）。
- 与客户交流中所产生的误解（MCC）。
- 故意违背规格说明（IDS）。
- 违反程序设计标准（VPS）。
- 数据表示有错（EDR）。

- 构件接口不一致 (ICI)。
- 设计逻辑的错误 (EDL)。
- 不完整或错误的测试 (IET)。
- 不准确或不完整的文档 (IID)。
- 将设计转换为程序设计语言实现时的错误 (PLT)。
- 不清晰或不一致的人机界面 (HCI)。
- 其他缺陷 (MIS)。

为了使用基于统计的质量保证方法，需要建一张如图26-5所示的表格。表中显示IES、MCC和EDR是造成所有错误的53%的“重要的少数”原因。但是需要注意，在只考虑严重错误时，应该将IES、EDR、PLT和EDL作为“重要的少数”原因。一旦确定了这些重要的少数原因，软件开发组织就可以开始采取改正行动了。例如，为了改正MCC错误，软件开发者可能要采用简便的需求收集技术（第7章），以提高与客户交流和规格说明的质量。为了改正EDR错误，开发者可能要使用工具进行数据建模，并进行更为严格的数据设计评审。[760]

总计		严重		一般		微小		
错误	数量	百分比	数量	百分比	数量	百分比	数量	百分比
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
<u>MIS</u>	<u>56</u>	<u>6%</u>	<u>0</u>	<u>0%</u>	<u>15</u>	<u>4%</u>	<u>41</u>	<u>9%</u>
总计	942	100%	128	100%	379	100%	435	100%

图26-5 基于统计的SQA数据收集

值得注意的是，改正行动主要是针对“重要的少数”。随着这些“重要的少数”原因不断改正，新的“重要的少数”原因将提到改正日程上来。

已经证明基于统计的软件质量保证技术确实使质量得到了提高[ART97]。在某些情况下，应用这些技术后，软件组织已经取得每年减少50%缺陷的好成绩。

基于统计的SQA及Pareto原则的应用可以用一句话概括：将时间用于真正重要的地方，但是首先你必须知道什么是真正重要的！

有关基于统计的SQA的详尽讨论超出了本书的范围，感兴趣的读者请参见文献[GOH02]、[SCH98]或[KAN95]。

26.6.2 软件工程中的六西格玛

六西格玛是当今产业界应用最广泛的基于统计的质量保证策略。20世纪80年代在摩托罗

拉公司最先普及，六西格玛策略“是严格规范的方法学，它运用数据和统计分析，通过识别和消除制造以及服务相关过程中的‘缺陷’来测量和改进企业的运转状况”[ISI03]。“六西格玛”一词来源于6个标准偏差——每百万个操作发生3.4个偏差（缺陷），它意味着非常高的质量标准。六西格玛方法学有三个主要的核心步骤：

761 六西格玛方
法学的核心
步骤是什么？

- 定义：通过与客户交流的方法来定义客户需求、可交付性，以及项目目标。
- 测量：测量现有的过程及其产品，以确定当前的质量状况（收集缺陷度量信息）。
- 分析：分析缺陷度量信息，并挑选出重要的少数原因。

如果某个现有软件过程是适当的，只是需要改进，六西格玛还需要另外两个核心步骤：

- 改进：通过消除缺陷根本原因的方式来改进过程。
- 控制：控制过程以保证以后的工作不会再引入缺陷原因。

以上三个核心步骤和另外两个附加步骤有时叫做DMAIC (Define、Measure、Analyze、Improve和Control)方法。

如果某组织正在开发一个软件过程（而不是改进现有的过程），则需要增加下面两个核心步骤：

- 设计：设计过程以（1）避免缺陷产生的根本原因；（2）满足客户需求。
- 验证：验证过程模型是否确实能够避免缺陷，并且满足客户需求。

上述步骤有时称为DMADV (Define、Measure、Analyze、Design和Verify) 方法。

六西格玛的全面讨论不是本书重点，感兴趣的读者可以参见[ISI03]、[SNE03]和[PAN00]。

26.7 软件可靠性

与其他质量因素不同，软件可靠性可以通过历史数据和开发数据直接测量和估算出来。按统计术语所定义的软件可靠性是：“在特定环境和特定时间内，计算机程序无失效（故障）运行的概率”[MUS87]。举个例子来说，如果程序X在8个处理小时内的可靠性估计为0.96，也就意味着，如果程序X执行100次，每次运行8个小时的处理时间（执行时间），则100次中正确运行（无故障）的次数可能是96 次。

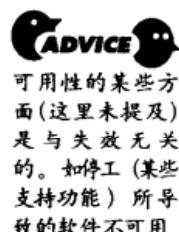
“不可避免的可靠性代价是显而易见的。”

——C.A.R Hoare

无论何时谈到软件可靠性，都会涉及一个关键问题：术语失效（failure）一词是什么涵义？在对软件质量和软件可靠性的任何讨论中，失效意味着与软件需求的不符。但是这一定义是有等级之分的。失效可以仅仅是令人厌烦的，也可以是灾难性的。有的失效可以在几秒钟之内得到纠正，有的则需要几个星期甚至几个月的时间才能纠正。让问题更加复杂的是，纠正一个失效事实上可能会引入其他的错误，而这些错误最终又会导致其他的失效。

26.7.1 可靠性和可用性的测量

早期的软件可靠性测量工作试图将硬件可靠性理论中的数学公式（如[ALV64]）外推来进行软件可靠性的预测。大多数与硬件相关的可靠性模型依据的是由于“磨损”而导致的失效，



而不是由于设计缺陷而导致的失效。在硬件中，由于物理磨损（如温度、腐蚀、震动的影响）导致的失效远比与设计缺陷有关的失效多。但是，软件恰好相反。实际上，所有软件失效都可以追溯到设计或实现问题上，磨损（第1章）在这里根本没有影响。

硬件可靠性理论中的核心概念以及这些核心概念是否适用于软件，对这个问题的争论仍然存在（如，[LIT89]和[ROO90]）。尽管在两种系统之间尚未建立不可辩驳的联系，但是考虑少数几个同时适用于这两种系统的简单概念却很有必要。

当我们考虑一个计算机系统时，可靠性的简单测量是“平均失效间隔时间”（Mean-Time-Between-Failure，MTBF），在这里：

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

其中，MTTF（Mean-Time-To-Failure）和MTTR（Mean-Time-To-Repair）分别是“平均失效时间”和“平均维修时间”⁶。

许多研究人员认为MTBF是一个远比“缺陷数/KLOC”或“缺陷数/FP”更有用的测量指标。简而言之，最终用户关心的是失效，而不是总缺陷数。由于一个程序中包含的每个缺陷所具有的失效率不同，总缺陷数难以表示系统的可靠性。

除可靠性测量之外，我们还必须进行可用性测量。软件可用性是指在某个给定时间点上程序能够按照需求执行的概率。其定义为：

$$\text{可用性} = [\text{MTTF} / (\text{MTTF} + \text{MTTR})] \times 100\%$$

MTBF可靠性测量对MTTF和MTTR同样敏感。而可用性测量在某种程度上对MTTR较为敏感，MTTR是对软件可维护性的间接测量。

26.7.2 软件安全

软件安全[LEV86]是一种软件质量保证活动，它主要用来识别和评估可能对软件产生负面影响并促使整个系统失效的潜在灾难。如果能够在软件过程的早期阶段识别这些灾难，就可以指定软件设计特性来消除或控制这些潜在的灾难。

“我不能想像导致这艘船沉没的任何情况，现代造船技术已经消除了这种可能性。”

——泰坦尼克号船长，E.I.Smith

建模和分析过程可以视为软件安全的一部分。开始时，根据危险程度和风险高低对灾难进行识别和分类。例如，与汽车上的计算机巡航控制系统相关的灾难可能有：

- 产生失去控制的加速，不能停止。
- 踩下刹车踏板后没有反应（关闭）。
- 开关打开后不能启动。
- 加速或减速缓慢。

一旦识别出这些系统级的灾难，就可以运用分析技术来确定这些灾难发生的严重性和概

⁶ 尽管失效可能需要进行调试（和相关的改正），但在大多数情况下，软件不做任何修改，经过重新启动就可以正常工作。

率⁷。为了达到高效，应该将软件置于整个系统中进行分析。例如，一个微小的用户输入错误（人也是系统组成部分）有可能会被软件错误放大，产生将机械设备置于不正确位置的控制数据，此时如果外部环境条件满足（且仅当这些条件满足时），机械设备的不正确位置将引发灾难性的失效。失效树分析[VES81]、实时逻辑[JAN86]或Petri网模型[LEV87]等分析技术可以用于预测可能引起灾难的事件链，以及事件链中的各个事件出现的概率。

WebRef

软件安全方面
值得一看的相关论文可在
[www.safeware-
eng.com/](http://www.safeware-eng.com/)找到。

灾难识别和分析完成之后，就可以进行软件中与安全相关的需求规格说明了。这个规格说明中包括一张不希望发生的事件清单，以及针对这些事件所希望产生的系统响应。这样就指明了软件在管理不希望发生的事件方面应起的作用。

尽管软件可靠性和软件安全彼此紧密相关，但是弄清它们之间的微妙差异非常重要。软件可靠性使用统计分析的方法来确定软件失效发生的可能性，而失效的发生未必导致灾难或灾祸。软件安全则考察失效会导致灾难发生的条件。也就是说，失效不是被置于真空之中，而是被放在整个计算机系统范围内加以考虑的。感兴趣的读者可以参见Leveson的专著[LEV95]。

764

26.8 ISO 9000质量标准⁸

质量保证体系可以定义为：用于实现质量管理的组织结构、责任、规程、过程和资源[ANS87]。创建质量保证体系的目的是帮助组织以符合规格说明的方式，保证组织的产品和服务满足客户的期望。ISO 9000标准描述了质量保证体系的一般要素，能够适用于任何行业——不论提供的是何种产品或服务。



ISO 9000描述的是必须做什么以保证一致，但是并没有描述必须如何做。

某个公司要获得ISO 9000质量保证体系中的一种模式认证，它的质量体系和实施情况应该由第三方的审核人员仔细检查，查看其是否符合标准以及实施是否有效。成功注册之后，这个公司将获得由审核人员所代表的注册登记实体颁发的证书。此后每半年进行一次审核，以此保证该公司的质量体系持续符合标准。

ISO 9001:2000是应用于软件工程的质量保证标准。这一标准在20个方面规定了高效的质量保证体系必须具备的质量要求。由于ISO 9001:2000标准适用于所有的工程行业，因此，为了说明该标准在软件过程中的应用，专门开发了ISO指南的子系列(ISO 9000-3)。(译者注：(1) ISO 9001:2000标准不仅适用于工程领域，它还可用于包括服务业在内的更广泛的领域。(2)本节所讲到的ISO 9001:2000标准的质量要求是前一版本ISO 9001:1994的内容(20个方面)。事实上，ISO 9001:2000涉及了管理职责、资源管理、产品实现以及测量、分析和改进等方面。(3)目前，文中所指的ISO 9000-3也被ISO 90003标准替代。ISO 9001:1994标准已被ISO 9001:2000替代。)

ISO 9001:2000中描述的质量要求涉及管理责任、质量体系、合同评审、设计控制、文档和数据控制、产品标识与跟踪、过程控制、审查和测试、纠

⁷ 这个方法与第25章介绍的风险分析方法类似，主要区别是它注重技术问题，而不是项目相关的问题。

⁸ 本节由Michael Stovsky编写，根据“Fundamentals of ISO 9000”——这是由R. S. Pressman & Associates, Inc.为音像教程“Essential Software Engineering”开发的工作手册——改编。翻印得到授权。

WebRef

大量指向ISO 9000/9001资源的链接见www.tantara.ab.ca/info.htm。

正及预防措施、质量控制记录、内部质量审核、培训、服务以及统计技术等主题。软件组织要通过ISO 9001:2000认证，就必须针对上述每一个方面的质量要求（以及其他方面的质量要求）制定相关的方针和规程，并且有能力证明组织活动的确是按照这些方针和规程实施的。有关ISO 9001的进一步信息，感兴趣的读者可以参见文献[HOY02]、[GAA01]或[CIA01]。

INFO**ISO 9001:2000标准**

下面的大纲定义了ISO 9001:2000标准的基本要素。标准的有关详细信息可从国际标准化组织（www.iso.com）及其他Internet信息源（如www.praxiom.com）找到。

建立质量管理体系的要素

- 建立、实施和改进质量体系；
- 制定质量方针，强调质量体系的重要性。

编制质量体系文档

- 描述过程；
- 编制操作手册；
- 制定控制（更新）文档的方法；
- 制定记录保存的方法。

支持质量控制和质量保证

- 提高所有共利益者对质量重要性的认识；
- 注重客户满意度；
- 制定质量计划来描述目的、职责和权力；
- 制定所有共利益者间的交流机制。

为质量管理体系建立评审机制

- 确定评审方法和反馈机制；
- 制定跟踪规程。

确定质量资源

- 包括：人员、培训、基础设施要素

建立控制机制

- 针对策划；
- 针对客户需求；
- 针对技术活动（如分析、设计、测试）；
- 针对项目监测和管理。

制定补救措施

- 评估质量数据和度量；
- 为持续的过程和质量改进制定措施。

765

26.9 SQA计划

SQA计划为软件质量保证提供了一张路线图。该计划由SQA小组（如果没有SQA小组，

由软件团队) 制定, 作为各个软件项目中SQA 活动的模板。

IEEE已经公布了SQA计划的标准[IEE94]。这个标准建议SQA计划应包括: (1) 计划的目的和范围; (2) SQA覆盖的所有软件工程工作产品的描述(如模型、文档、源代码); (3) 应用于软件过程中的所有适用的标准和习惯做法; (4) SQA活动和任务(包括评审和审核)以及它们在整个软件过程中的位置; (5) 支持SQA活动和任务的工具和方法; (6) 控制变更的软件配置管理的规程(第27章); (7) 评估、保护和维护所有与SQA相关的记录的方法; (8) 与产品质量相关的组织角色和责任。

SOFTWARE TOOLS

软件质量管理

目的: SQA工具的目的是辅助项目团队评估和改进软件工作产品的质量。

机制: 工具的机制各异。一般情况下, 目的是评估特定工作产品的质量。注意, 很多软件测试工具(见第13章和第14章)常被包含在SQA工具类中。

代表性工具⁹

766

ARM, 由NASA (satc.gsfc.nasa.gov/tools/index.html) 开发, 提供了可用于评估软件需求文档质量的方法。

QPR ProcessGuide and Scorecard, 由QPR Software (www.qpronline.com) 开发, 提供对六西格玛及其他质量管理方法的支持。

Quality Tools Cookbook, 由Systma and Manley (www.systsma.com/tqmtools/tqmtoolmenu.html) 开发, 提供了对著名的质量管理体系(例如, 控制表, 分散图、关系图及矩阵图)的有用描述。

Quality Tools and Templates, 由iSixSigma (<http://www.isixsigma.com/tt/>) 开发, 描述了大量有用的质量管理工具和方法。

TQM Tools, 由Bain & Company (www.bain.com) 开发, 提供了大量的TQM管理工具及相关的质量管理方法。

26.10 小结

软件质量管理是在软件过程中的每一步都进行的“普遍性活动”——包括质量控制和质量保证两部分。SQA包括: 对方法和工具有效应用的规程, 正式技术评审, 测试策略和技术, 变更控制规程, 保证符合标准的规程, 以及测量和报告机制。

软件质量, 是计算机程序的一种属性, 其定义是“与明确和隐含定义的需求的符合程度”。软件质量本质上的复杂性也使SQA复杂化。但是一般说来, 软件质量包括了许多不同的产品和过程因素及其相关的度量信息。

软件评审是最为重要的质量控制活动之一。评审作为所有软件工程活动的过滤器, 当发现及改正错误的成本比较小时, 就消除错误。正式技术评审是一种典型的评审会议, 在实践中已证明这种形式对于发现错误极其有效。

为了正确地进行软件质量保证, 必须收集、评估和发布软件工程过程的数据。基于统计

⁹ 这里记录的工具并不代表本书支持这些工具, 而只是这类工具的例子。在大多数情况下, 工具的名字由各自的开发者注册为商标。

的SQA有助于提高产品和软件过程本身的质量。软件可靠性模型将测量加以扩展，能够由所收集的缺陷数据推导出相应的失效率和进行可靠性预测。

总而言之，让我们借用Dunn和Ullman[DUN82]的一句话：“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。当成功实现上述映射时，其结果就是成熟的软件工程。

参考文献

- [ALV64] Alvin, W. H., von (ed.), *Reliability Engineering*, Prentice-Hall, 1964.
- [ANS87] ANSI/ASQC A3-1987, *Quality Systems Terminology*, 1987.
- [ART92] Arthur, L. J., *Improving Software Quality: An Insider's Guide to TQM*, Wiley, 1992.
- [ART97] Arthur, L. J., "Quantum Improvements in Software System Quality," *CACM*, vol. 40, no. 6, June 1997, pp. 47-52.
- [BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981.
- [CIA01] Cianfrani, C. A., et al., *ISO 9001:2000 Explained*, 2nd ed., American Society for Quality, 2001.
- [CRO79] Crosby, P., *Quality Is Free*, McGraw-Hill, 1979.
- [DEM86] Deming, W. E., *Out of the Crisis*, MIT Press, 1986.
- [DEM99] DeMarco, T., "Management Can Make Quality (Im)possible," *Cutter IT Summit*, Boston, April 1999.
- [DIJ76] Dijkstra, E., *A Discipline of Programming*, Prentice-Hall, 1976.
- [DUN82] Dunn, R., and R. Ullman, *Quality Assurance for Computer Software*, McGraw-Hill, 1982.
- [FRE90] Freedman, D. P., and G. M. Weinberg, *Handbook of Walkthroughs, Inspections and Technical Reviews*, 3rd ed., Dorset House, 1990.
- [GAA01] Gaal, A., *ISO 9001:2000 for Small Business*, Saint Lucie Press, 2001.
- [GIL93] Gilb, T., and D. Graham, *Software Inspections*, Addison-Wesley, 1993.
- [GLA98] Glass, R., "Defining Quality Intuitively," *IEEE Software*, May 1998, pp. 103- 104, 107.
- [GOH02] Goh, T., V. Kuralmani, and M. Xie, *Statistical Models and Control Charts for High Quality Processes*, Kluwer Academic Publishers, 2002.
- [HOY02] Hoyle, D., *ISO 9000 Quality Systems Development Handbook: A Systems Engineering Approach*, 4th ed., Butterworth-Heinemann, 2002.
- [IBM81] "Implementing Software Inspections," course notes, IBM Systems Sciences Institute, IBM Corporation, 1981.
- [IEE94] *Software Engineering Standards*, 1994, IEEE Computer Society, 1994.
- [ISI03] iSixSigma, LLC, "New to Six Sigma: A Guide for Both Novice and Experienced Quality Practitioners," 2003, available at <http://www.isixsigma.com/library/content/six-sigma-newbie.asp>.
- [JAN86] Jahanian, F., and A. K. Mok, "Safety Analysis of Timing Properties of Real-Time Systems," *IEEE Trans. Software Engineering*, vol. SE-12, no. 9, September 1986, pp. 890-904.
- [JON86] Jones, T. C., *Programming Productivity*, McGraw-Hill, 1986.
- [KAN95] Kan, S. H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995.
- [LEV86] Leveson, N. G., "Software Safety: Why, What, and How," *ACM Computing Surveys*, vol. 18, no. 2, June 1986, pp. 125-163.
- [LEV87] Leveson, N. G., and J. L. Stolzy, "Safety Analysis Using Petri Nets," *IEEE Trans. Software Engineering*, vol. SE-13, no. 3, March 1987, pp. 386-397.
- [LEV95] Leveson, N. G., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
- [LIN79] Linger, R., H. Mills, and B. Witt, *Structured Programming*, Addison-Wesley, 1979.
- [LIT89] Littlewood, B., "Forecasting Software Reliability," in *Software Reliability: Modeling and Identification*, (S. Bittanti, ed.), Springer-Verlag, 1989, pp. 141-209.
- [MUS87] Musa, J. D., A. Iannino, and K. Okumoto, *Engineering and Managing Software with Reliability Measures*, McGraw-Hill, 1987.
- [PAN00] Pande, P., et al., *The Six Sigma Way*, McGraw-Hill, 2000.
- [POR95] Porter, A., H. Siy, C. A. Toman, and L. G. Votta, "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development," *Proc. Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Washington, D.C., October 1995, ACM Press, pp. 92-103.

768

- [ROO90] Rook, J., *Software Reliability Handbook*, Elsevier, 1990.
- [SCH98] Schulmeyer, G. C., and J. I. McManus (eds.), *Handbook of Software Quality Assurance*, 3rd ed., Prentice-Hall, 1998.
- [SOM01] Somerville, I., *Software Engineering*, 6th ed., Addison-Wesley, 2001.
- [SNE03] Snee, R., and R. Hoerl, *Leading Six Sigma*, Prentice-Hall, 2003.
- [THE01] Thelin, T., H. Petersson, and C. Wohlin, "Sample Driven Inspections," *Proceedings Workshop on Inspection in Software Engineering (WISE'01)*, Paris, France, July 2001, pp. 81–91, can be downloaded from <http://www.cas.mcmaster.ca/~wise/wise01/TheLinPetersson-Wohlin.pdf>.
- [VES81] Veseley, W. E., et al., *Fault Tree Handbook*, U.S. Nuclear Regulatory Commission, NUREG-0492, January 1981.

习题与思考题

- 26.1 有人认为FTR应该主要针对程序设计风格和正确性。这种主意好吗？为什么？
- 26.2 如果客户不断改变想要做什么的想法，是否还有可能评估软件质量？
- 26.3 获取一份ISO 9001:2000和ISO 9000-3的副本。准备一次讨论作业，讨论3个方面的ISO 9001质量要求及其在软件行业中是如何应用的。
- 26.4 在本章前面我们提到“差异控制是质量保证的核心”。既然每个程序都与其他程序互不相同，我们应该寻找什么样的差异？应该如何控制差异？
- 26.5 一个程序能够既正确又表现出高质量吗？请加以解释。
- 26.6 为什么软件工程小组与独立的软件质量保证小组之间的关系经常是紧张的？这种紧张关系是否是正常的？
- 26.7 假设你被赋予改进组织中的软件质量的职责。你要做的第一件事是什么？然后呢？
- 26.8 软件中的MTBF概念仍然不断受到批评。你能够想出几个原因吗？
- 26.9 仅当每个参与者都进行了事先准备时，正式技术评审才是有效的。你怎样能够判断评审参与者是否进行了准备？如果你是评审主席，你该怎么办？
- 26.10 质量和可靠性是两个相关的基本概念，但在许多方面却有根本的不同。请就此进行讨论。
- 26.11 给出两个安全性至关重要的计算机控制系统。并为每个系统列出至少3条与软件失效直接相关的灾难。
- 26.12 研究软件可靠性相关的文献，写一篇文章描述一种软件可靠性模型，一定要给出一个例子。
- 26.13 评审图26-5所示的表，并从中选择导致严重和中等错误的4个“重要的少数”原因。用其他章节中给出的信息，提出相应的改正行动。
- 26.14 除了可以对错误和缺陷计数之外，还有哪些可以计数的软件特征具有质量意义？它们是什么？能否直接测量？
- 26.15 一个程序能够既正确又不可靠吗？请加以解释。

推荐读物与阅读信息

Moriguchi (《Software Excellence: A Total Quality Management Guide》, Productivity Press, 1997) 和Horch (《Practical Guide to Software Quality Management》, Artech Publishing, 1996) 的书籍很好地介绍了管理方面的内容，在制定计算机软件的正式质量保证计划时很有帮助。Deming [DEM86]、Juran (《Juran on Quality by Design》, Free Press, 1992) 和Crosby

([CRO79]和《Quality Is Still Free》, McGraw-Hill, 1995) 的论著并不专门针对软件, 但却是负责软件开发的高层管理者的必读书。Gluckman和Roome (《Everyday Heroes of the Quality Movement》, Dorset House, 1993) 通过讲述质量过程中不同角色的故事将质量问题人性化。Kan (《Metrics and Models in Software Quality Engineering》, Addison-Wesley, 1995) 给出了软件质量的定量观点。

769

Cianfani和他的同事 (《ISO 9001:2000 Explained, second edition》, American Society for Quality, 2001) 及Gaal (《ISO 9001:2000 for Small Business: Implementing Process-Approach Quality Management》, St. Lucie Press, 2001) 讨论了ISO 9001:2000质量标准。Tingley (《Comparing ISO 9000, Malcolm Baldrige, and the SEI CMM for Software》, Prentice-Hall, 1996) 为正在努力改进质量管理过程的组织提供了有用的指南。

George (《Lean Six Sigma》, McGraw-Hill, 2002)、Pande和他的同事 (《The Six Sigma Way Fieldbook》, McGraw-Hill, 2001) 及Pyzdek (《The Six Sigma Handbook》, McGraw-Hill, 2000) 的论著讨论了六西格玛——一个基于统计的质量管理技术, 可以降低产品的缺陷率。

Radice (《High Quality, Low Cost Software Inspections》, Paradoxicon Publishers, 2002)、Wiegers (《Peer Reviews in Software: A Practical Guide》, Addison-Wesley, 2001)、Gilb和Graham (《Software Inspection》, Addison-Wesley, 1993) 以及Freedman和Weinberg (《Handbook of Walkthroughs, Inspections and Technical Reviews》, Dorset House, 1990) 为进行有效的正式技术评审提供了有价值的指南。

Musa (《Software Reliability Engineering: More Reliable Software, Faster Development and Testing》, McGraw-Hill, 1998) 给出了应用软件可靠性技术的实用指南。Kapur等人 (《Contributions to Hardware and Software Reliability Modeling》, World Scientific Publishing Co., 1999)、Gritzalis (《Reliability, Quality and Safety of Software-Intensive Systems》, Kluwer Academic Publishers, 1997) 以及Lyu (《Handbook of Software Reliability Engineering》, McGraw-Hill, 1996) 编辑了关于软件可靠性的重要论文的文集。

Hermann (《Software Safety and Reliability》, Wiley-IEEE Press, 2000)、Storey (《Safety-Critical Computer Systems》, Addison-Wesley, 1996) 和Leveson [LEV95]至今是已出版的软件安全方面最全面的书籍。另外, van der Meulen (《Definitions for Hardware and Software Safety Engineers》, Springer-Verlag, 2000) 提供了完整的有关可靠性和安全的概念和术语。Gartner (《Testing SafetyRelated Software》, Springer-Verlag, 1999) 为测试安全性要求很高的系统提供了专门的指南。Friedman和Voas (《Software Assessment Reliability, Safety and Testability》, Wiley, 1995) 为评估可靠性和安全提供了有用的模型。

因特网上有大量关于软件质量管理的信息资源, 最新的WWW参考列表在SEPA的站点 <http://www.mhhe.com/pressman>可以找到。

770

第27章 变更管理

要点浏览

概念：在开发计算机软件时，会发生变更。而且由于变更的发生，迫使你必须有效地去管理变更。变更管理——也叫软件配置管理（SCM）——是一组活动，它通过以下方式来管理变更：识别可能发生变更的工作产品，建立这些工作产品之间的关系，制定管理这些工作产品的不同版本的机制，控制所施加的变更，审核和报告所发生的变更。

人员：每个参与软件过程的人员都在一定程度上与变更管理有关系。但有时也需要设专人来管理SCM过程。

重要性：如果你不去控制变更，则变更会控制你，这绝不是什么好事。不受控制的一系列变更轻而易举就能使一个运行良好的软件项目变得混乱。因此，变

更管理是好的项目管理和可靠的软件工程实践的基本组成部分。

步骤：由于在开发软件时会产生很多工作产品，所以必须唯一标识每一个工作产品。标识了工作产品之后，就可以制定版本控制和变更控制机制了。为了保证变更发生时维护好质量，就要审核变更过程；为了保证将变更通知到那些需要知道的人员，就要报告变更。

工作产品：软件配置管理计划定义了变更管理的项目策略。此外，当启动正式SCM时，变更控制过程将产生软件变更请求、变更报告和工程变更工单。

质量保证措施：当每一个工作产品都能够被标识、跟踪和控制，当每一个变更都能够被跟踪和分析，当需要知道变更的每一个人都已经被通知到时，你就做对了。

关键概念
审核
基线
变更
变更控制
配置对象
内容管理
CVS
标识
中心存储库
SCI
SCM过程
标准
状态报告
版本控制
WebApp CM

在开发计算机软件时，变更是不可避免的。变更加剧了共同开发某一项目的软件工程师之间的混乱。如果变更之前没有经过分析，变更实现之前没有进行记录，没有向那些需要知道变更的人员报告变更，或者没有以能够提高质量和减少错误的方式控制变更，就会产生混乱。Babich [BAB86]是这样说的：

协调软件开发，使得混乱……减到最小的技术称为配置管理。配置管理是对开发团队正在开发软件的修改进行标识、组织和控制的技术，目的是使错误量降至最少，并使生产率最高。

变更管理，通常叫做软件配置管理（SCM或CM），是贯穿于整个软件过程的普适性活动。因为变更可能随时发生，所以SCM活动的目标就是为了：(1) 标识变更；(2) 控制变更；(3) 保证正确地实现变更；(4) 向那些利害相关的人员报告变更。

搞清楚软件维护和软件配置管理的区别是很重要的。软件维护是一组软件工程活动，它发生在软件已经交付给客户并投入运行之后；而软件配置管理是一组跟踪和控制活动，它开始于软件工程项目开始之时，并且终止于该软件被淘汰之时。

软件配置管理的主要目标是：当变更发生时，能够提高适应变更的容易程度，并且能够减少所需花费的工作量。本章中我们将讨论那些使我们能够管理变更的特定活动。

27.1 软件配置管理

软件过程的输出信息主要分为三类：(1) 计算机程序（源代码和可执行程序）；(2) 描述计算机程序的文档（针对技术开发者和用户）；(3) 数据（包含在程序内部的数据，或程序外部的数据）。在软件过程中产生的所有信息项总称为软件配置。

如果一个软件配置项只是简单地推导出其他一些软件配置项，则几乎不会产生混乱。但不幸的是，在这个过程中还有另一个变量——变更。变更可能毫无理由地随时发生。事实上，正如“系统工程第一定律”[BER80]所述：“不管你处在系统生存周期的什么阶段，系统都可能会发生变更，并且在整个生存周期中将会持续不断提出变更的要求。”

“除了变化，没有什么东西是永恒的。”

—Heraclitus, 公元前500年

变更的起因是什么呢？这个问题的答案就像变更本身那样多变。但有四个基本的原因：

 软件请求变更的起因是什么呢？

- 新的业务或市场条件导致产品需求或业务规则的变更。
- 新的客户需求，要求修改信息系统产生的数据、产品提供的功能或系统提供的服务。
- 企业改组或扩大/缩小规模，导致项目优先级或软件工程团队结构的变更。
- 预算或进度安排的限制，导致系统或产品的重新定义。

772

软件配置管理是一组在计算机软件的整个生存周期内管理变更的活动。可以把SCM看作是应用于整个软件过程的软件质量保证活动。在下面几节中，将介绍帮助我们管理变更的主要SCM任务和重要概念。

27.1.1 SCM场景¹

典型的CM工作场景包括：总体负责软件小组的项目经理、总体负责CM规程和方针的配置管理员、负责开发和维护软件产品的软件工程师以及使用软件产品的客户。在本场景中，我们假定由6个人组成的团队正在开发一个约15 000行代码的小型软件。（注意，也可以组建更小或更大团队的场景，但是，实际上每个这样的项目都面临着一个问题，就是CM。）

在操作级别上，SCM场景包括多种角色和任务。项目经理的职责是保证在确定的时间框架内开发产品。因此，项目经理必须对软件的开发进展情况进行监控，找出问题，并对问题做出反应。这可通过建立和分析软件系统状态报告，并执行对系统的评审来完成。

 每一个参与变更管理的人员的职责和应从事的活动是什么？

配置管理员的职责不仅是要保证代码的创建、变更和测试要遵循相应的规程和方针，还要使项目的相关信息容易得到。为了实现维护代码变更控制的技术，配置管理员可以引入正式的变更请求机制、变更评估机制（通过负责批准软件系统变更的变更控制委员会）和变更批准机制。配置管理员要为工程师们创建和分发任务单，并且还要创建项目的基本环境，而且，还要收集软件系统各个构件的统计信息，比如能够决定系统中哪一个构件有问题的信息。

¹ 本节摘自[DAR01]。已经得到卡内基·梅隆大学软件工程研究所的同意，允许翻印由Susan Dart[DAR01]编写的“Spectrum of Functionality in CM Systems”，© 2001，卡内基·梅隆大学。



773

一定存在某种能够保证适当地跟踪、管理和执行同一个构件中同时发生的多个变更的机制。

软件工程师的职责是高效地工作。也就是说，软件工程师在代码的创建和测试以及编写支持文档时不做不必要的相互交流；但同时，软件工程师们又尽可能地进行有效的沟通和协调。特别是，软件工程师可以使用相应的工具来协助开发一致的软件产品；软件工程师之间可以通过相互通报任务要求和任务完成情况来进行沟通和协调；通过合并文件，可以使变更在彼此的工作中传播。对于同时有多个变更的构件，要用机制来保证具有某种解决冲突和合并变更的方法。历史资料应该记录系统中所有构件的演化过程，以及变更原因日志和实际上进行了哪些变更。软件工程师有他们自己创建、变更、测试和集成代码的工作空间。在特定点，可以将代码转变成基线，并从基线做进一步的开发，或生成针对其他目标机的变体。

客户只是使用产品。由于产品处于CM控制之下，因此，客户要遵守请求变更和指出产品缺陷的正式规程。

理想情况下，在本场景中应用的CM系统应该支持所有的角色和任务。也就是说，角色决定了CM系统所需的功能。项目经理可以把CM看作是一个审核机制；配置管理员可以把CM看作是控制、跟踪和制定方针的机制；软件工程师可以把CM看作是变更、构建以及访问控制的机制；而用户可以把CM看作是质量保证的机制。

27.1.2 配置管理系统元素

Susan Dart[DAR01]在她的软件配置管理白皮书中指出，开发配置管理系统时应具备四个重要元素：

- 构件元素——是一组具有文件管理系统（如，数据库）功能的工具，使我们能够访问和管理每一个软件配置项。
- 过程元素——是一个规程和任务的集合，它为所有参与管理、开发和使用计算机软件的人员定义了变更管理（以及相关活动）的有效方法。
- 构造元素——是一组自动软件构造工具，用以确保装配了正确的有效构件（即，正确的版本）集。
- 人员元素——为了实现高效的SCM，软件团队可利用的一组工具和过程特性（包括其他CM元素）。

以上这些元素（将在后面几节中详细讨论）并不是相互孤立的。例如，随着软件过程的演化，可能会同时用到构件元素和构造元素；过程元素可以指导多种与SCM相关的人员活动，因此也可以将其认为是人员元素。

774

27.1.3 基线



大多数软件变更
是合理的。因此，
不要去抱怨变
更，而是要确定
你是否有合适的
机制来处理变更。

变更是软件开发中必然的事情。客户希望修改需求，开发者希望修改技术方法，而管理者希望修改项目策略。为什么要修改呢？答案其实十分简单，随着时间的流逝，所有的软件参与者也就得到了更多知识（关于他们需要什么，什么方法最好，如何既能完成任务，又能赚钱），这些知识是大多数变更发生的推动力，并且造成了很多软件工程实践者难以接受的事实：大多数变更都是合理的！

基线是一个软件配置管理的概念，它能够帮助我们在不严重阻碍合理变更的条件下控制变更。IEEE (IEEE标准 610.12-1990) 是这样定义基线的：

已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能修改它。



只有经过评审和批准，一个软件工程工作产品才能成为基线。

在软件配置项成为基线之前，可以迅速而随意地进行变更。然而，一旦成为基线，就像是通过了一个单向旋转门，这时虽然可以进行变更，但是必须应用特定的、正式的规程来评估和验证每一个变更。

在软件工程范畴中，基线是软件开发中的里程碑，其标志是在正式技术评审中（第26章）已经获得批准的一个或多个软件配置项 (SCI) 的发布。例如，某设计模型的元素已经形成文档并通过评审，错误已被发现并得到纠正，一旦该模型的所有部分都经过了评审、纠正和批准，该设计模型就成为一个基线。任何对程序体系结构（在设计模型中已形成文档）的变更只能在每一个变更被评估和批准之后方可进行。虽然可以在任意细节层次上定义基线，但最常见的软件基线如图27-1所示。

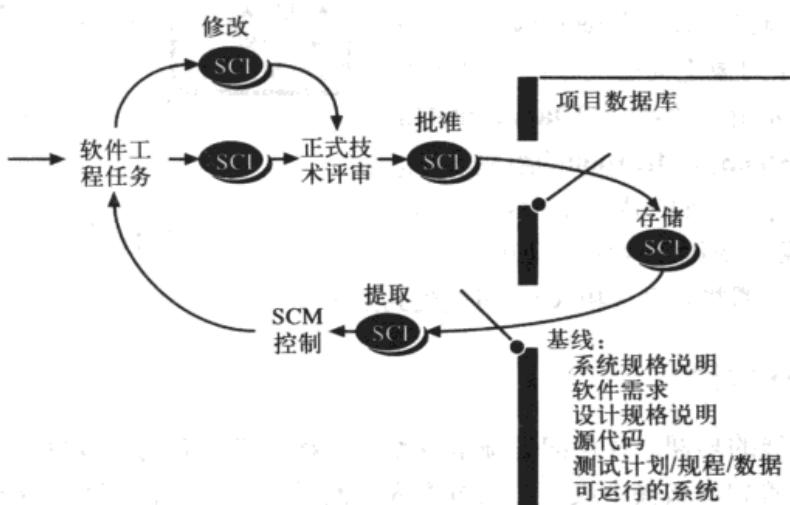


图27-1 基线化的SCI和项目数据库



要确保项目数据库的维护是集中的、可控的。

图27-1也说明了基线的形成过程。软件工程任务可能会产生一个或多个SCI，在这些SCI经过评审并被批准之后，就可以将它们放置到项目数据库（也称为项目库或软件中心存储库，见27.2节）中。当软件团队中的成员想要修改某个基线SCI时，必须将该SCI从项目数据库中拷贝到工程师的私有工作区中。但是，这个被提取出的SCI只有在遵循了SCM控制（在本章后面将讨论）的条件才可以进行修改。图27-1中的箭头说明了某个已成为基线的SCI的修改路径。

27.1.4 软件配置项

软件配置项是在软件工程过程中创建的信息。在极端情况下，大型规格说明中的一节、大型测试用例集中的一个测试用例都可以看作是一个SCI。再实际点，一个SCI可以是一个文

档、一个全套的测试用例，也可以是一个已命名的程序构件（例如，一个C++函数或一个Java程序）。

除了这些来自软件工程工作产品的SCI之外，很多软件工程组织也将软件工具列入配置管理的范畴，即，特定版本的编辑器、编译器、浏览器以及其他自动化工具都被“固定”作为软件配置的一部分。因为要使用这些工具来生成文档、源代码和数据，所以当要对软件配置进行变更时，必须可以得到这些工具。虽然问题并不多见，但一个工具的新版本（如，编译器）有可能产生和原版本不同的结果。因此，就像它们协助开发的软件一样，工具也可以基线化为完整配置管理过程的一部分。

在现实中，是将SCI组织成配置对象，这些配置对象具有自己的名字，并且按类别存储在项目数据库中。一个配置对象具有一个名称和多个属性，并通过关系来表示与其他配置对象的“关联”。在图27-2中，分别定义了配置对象**Design Specification**（设计规格说明）、**DataModel**（数据模型）、**ComponentN**（构件N）、**SourceCode**（源代码）和**TestSpecification**（测试规格说明）。各个对象之间的关系如图中箭头所示，单向箭头表示组成关系，即**DataModel**和**ComponentN**是**Design Specification**的组成部分。双向箭头说明对象之间的内在联系，如果**SourceCode**对象发生变化，软件工程师通过查看内在联系能够确定哪些对象（和SCI）可能受到影响²。

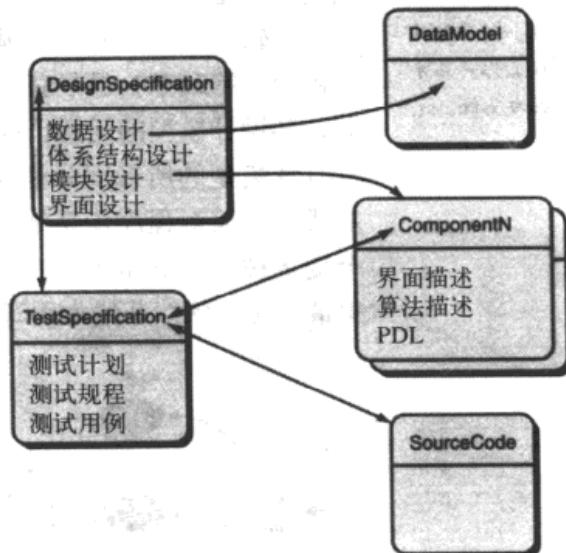


图27-2 配置对象

776

27.2 SCM中心存储库

在软件工程发展的早期，软件配置项是以纸质文档（或打孔的计算机卡片）的形式进行维护的，它们被放置到卷宗夹中或三孔活页夹内，然后保存在金属柜中。这种方法存在许多问题：（1）难以找出所需要的配置项；（2）难以确定更改了哪一个配置项，是什么时候以及由谁更改的；（3）构造现有程序的新版本耗时，并且很可能出错；（4）实际上根本不可能详细描述各配置项或各配置项之间的复杂关系。

如今，我们是在项目数据库或中心存储库（repository）中维护SCI的。《韦伯斯特词典》中将单词“repository”定义为“any thing or person thought of as a center of accumulation or storage（即任何可以认为是积聚或存储中心的事物或人）”。在软件工程历史早期，中心存储库只是一个人（程序员），程序员必须要记住所有与软件项目相关的信息的位置，还必须能够回想起那些从未记录下来的信息，并且能够重构那些已经失去的信息。不幸的是，将人作为“积聚和存储中心”（虽然符合韦伯斯特的定义）并不是很好。今天，中心存储库是一个“事物”——一个作为软件工程信息积聚和存储中心的数据库，而人（软件工程师）的角色是通过与存储库集成在一起的工具与中心存储库进行交互。

² 这些关系是在数据库中定义的。数据库（中心存储库）的结构在27.2节有更详细的讨论。

27.2.1 中心存储库的作用

SCM中心存储库是一组机制和数据结构，它使软件团队可以有效地管理变更。它具有数据库管理系统的一般功能，此外，中心存储库还具有以下功能[FOR89]： 777

 SCM中心
存储库具有
哪些功能？

- **数据完整性。**包括：中心存储库的登录功能；保证相关对象之间一致性的功能；当一个对象的修改导致与其相关的其他对象也要修改时，能够自动完成“级联”修改的功能。
- **信息共享。**提供了在多个开发者和不同工具之间共享信息的机制，能够管理和控制对数据的多用户访问，以及锁定或解锁对象，使得变更不会被不经意间覆盖。
- **工具集成。**建立了可以被多种软件工程工具访问的数据模型，能够控制对数据的访问，并且实现配置管理功能。
- **数据集成。**提供了数据库功能，它允许对一个或多个SCI实施不同的SCM任务。
- **推行方法。**定义了存储在中心存储库中的数据的E-R（实体-关系）模型，E-R模型中可能隐含了特定的软件工程过程模型；至少，关系和对象定义了一系列创建中心存储库的内容所需进行的步骤。
- **文档标准化。**是数据库中对象的定义，它直接给出了创建软件工程文档的标准方法。

为了实现上述功能，我们用术语元模型（meta-model）来定义中心存储库。元模型决定了在中心存储库中信息如何存储、如何通过工具访问数据、软件工程师如何查看数据、维护数据安全性和完整性的能力如何、将现有模型扩展以适应新需求时的容易程度如何等。感兴趣的读者可参考文献[SHA95]和[GRI95]获得更多信息。

27.2.2 一般特征和内容

WebRef

应用于商业上的中心存储库的例子见www.software.hp.com/products/SCMGR或者ott.oracle.com/documentation/repository.html。

中心存储库的特征和内容可以从两个方面来理解：中心存储库中将存储什么；中心存储库提供什么特定服务。中心存储库中存储的表示类型、文档和工作产品的详细分类如图27-3所示。

一个健壮的中心存储库能够提供两种不同类型的服务：(1) 期望从任何一个数据库管理系统得到的服务类型；(2) 特定于软件工程环境的服务类型。

作为软件工程团队的中心存储库，应该：(1) 集成或直接提供过程管理功能；(2) 提供在中心存储库中管理SCM功能的特定规则和维护数据；(3) 提供与其他软件工程工具的接口；(4) 能够存储各种数据对象（如，文本、图形、视频、音频）。 778

27.2.3 SCM 特征

为了支持SCM，中心存储库必须具有支持下列特征的工具集：

版本控制。随着项目进展，每个工作产品都可能有很多版本（27.3.2节）。中心存储库必须能保存所有这些版本，以便有效地管理产品发布，并允许开发者在测试和调试过程中可以返回到早先的版本。

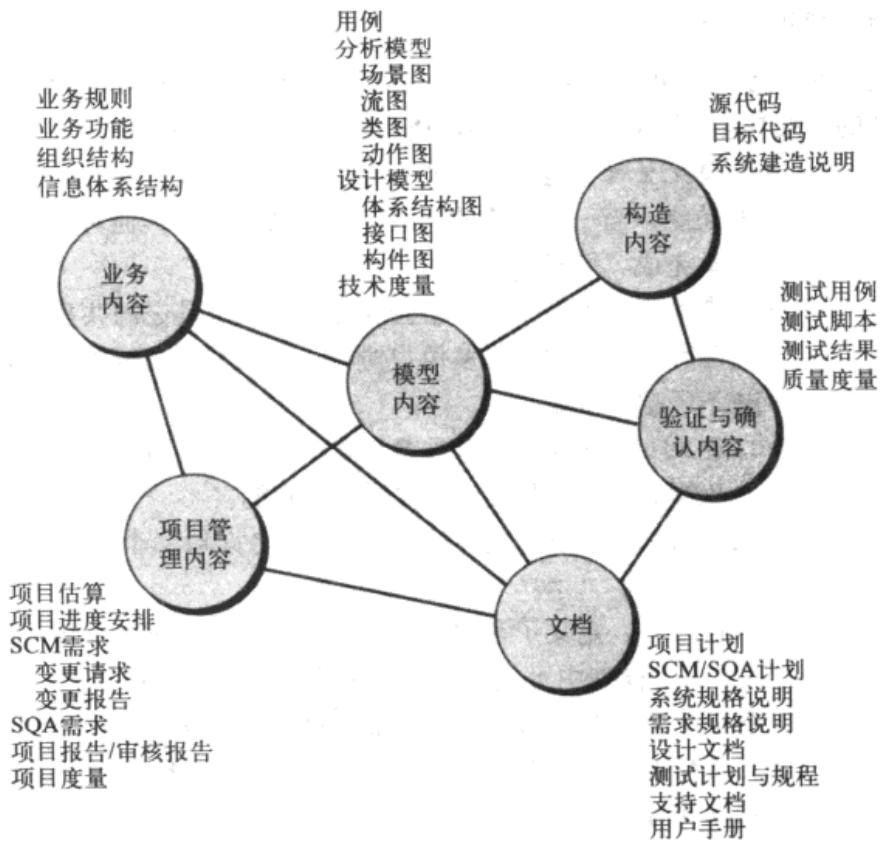


图27-3 中心存储库的内容

KEY POINT
中心存储库必须能够维护与软件多个不同版本相关的SCI。更重要的是，必须提供按特定版本配置来装配这些SCI的机制。

中心存储库必须能控制各种对象，包括文本、图形、位图、复杂文档、屏幕与报告定义、目标文件、测试数据和结果等特殊对象。在一个成熟的中心存储库中可以按任意粒度跟踪对象版本，例如可以跟踪单个数据定义或一组模块。

相关性跟踪和变更管理。中心存储库要管理所存储的配置对象之间的各种关系。这些关系包括：企业实体与过程之间的关系、应用设计各部分之间的关系、设计构件与企业信息体系结构之间的关系、设计元素与其他工作产品之间的关系等等。其中有些仅仅是关联关系，而有些则是依赖关系或强制关系。

记录所有这些关系的能力对中心存储库中存储信息的完整性、基于中心存储库的工作产品的生成是至关重要的，而且这是中心存储库概念对软件开发过程改进最主要的贡献之一。例如，如果修改了某UML类图，则中心存储库能够检测出是否有相关联的类、接口定义和代码构件也需要进行修改，并且能够提醒开发者哪些SCI受到影响。

需求跟踪。能够用来跟踪由特定需求规格说明产生的所有设计构件和架构构件以及可交付产品（正向跟踪）。此外，还能够用来辨别指定的工作产品是由哪个需求产生的（反向跟踪）。

配置管理。配置管理工具能够记录表示特定项目里程碑或产品发布的一系列配置。

审核跟踪。审核跟踪使我们能够了解变更是在什么时候、什么原因以及由谁完成的信息。变更的根源信息可以作为中心存储库中特定对象的属性进行存储。

27.3 SCM过程

软件配置管理过程中定义的一系列任务具有四个主要目标：(1) 统一标识软件配置项；(2) 管理一个或多个软件配置项的变更；(3) 便于构造应用的不同版本；(4) 在配置随时间而演化时，确保能够保持软件质量。

能够取得上述四个目标的过程不应过于原则和抽象，也不要太繁琐，这个过程应该具有使软件团队能够解决一系列复杂问题的特色：

SCM过程
应该回答什
么问题？

- 一个软件团队应该如何标识软件配置的离散元素？
- 一个组织应该如何管理程序（及其文档）的多个已有版本，从而使变更能够高效地进行？
- 一个组织应该如何在软件发布给客户之前和之后控制变更？
- 应该由谁负责批准变更并给变更确定优先级？
- 我们如何保证能够正确地完成变更？
- 应该采用什么机制去评价那些已经发生了的变更？

如图27-4所示，上述问题引导我们定义了五个SCM任务：标识、版本控制、变更控制、配置审核、报告。

如图中所示，SCM任务可以看作是一个同心圆的层次结构。软件配置项（SCI）在它们的有效生存期内都要从内到外经历各个层次，最终成为某应用程序或系统的一个或多个版本软件配置的组成部分。当一个SCI进入某层时，该SCM过程层次所包含的活动可能适用，也可能不适用。例如，在创建一个新SCI时，必须对其进行标识，但是，如果对该SCI没有任何变更请求，那就不必应用变更控制层的活动，而直接将该SCI指定给软件的特定版本（版本控制机制开始起作用了）。为了进行配置审核，就要维护SCI记录（SCI的名称，创建日期，版本设计等），并将这些记录报告给那些需要知道的人员。下面，我们将更详细地介绍每一个SCM过程层次。

27.3.1 软件配置中对象的标识

为了控制和管理软件配置项，必须对每个配置项单独命名，然后用面向对象的方法进行组织。可以进行标识的对象有两种类型[CHO89]：基本对象和聚合对象³。基本对象是软件工程师在分析、设计、编码或测试过程中所创建的“信息单元”。例如，一个基本对象可以是需求规格说明的一节、设计模型的一个部件、一个构件的源代码或用于测试代码的一组测试用

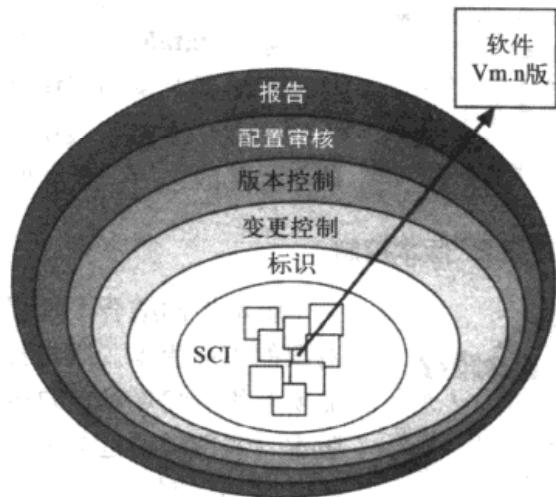


图27-4 SCM过程的层次结构

³ 聚合对象的概念[GUS89]是作为一种表示某软件配置完全版本的机制而提出的。

例。聚合对象是基本对象和其他聚合对象的集合。在图27-2中的**DesignSpecification**（设计规格说明）就是一个聚合对象，在概念上，可视其为一个已命名的（已标识的）指针表，指向基本对象**DataModel**（数据模型）和**ComponentN**（构件N）。

每个对象都具有一组能够唯一地标识它的独特特征：名称、描述、资源表以及“实现”。对象名称是能够清楚地标识对象的一个字符串；对象描述是一个数据项列表，它能够标识出该对象所表示的SCI类型（如，模型元素、程序、数据）、项目标识符以及变更和（或）版本信息。



针对配置对象建立的相互关系使软件工程师可以评估变更的影响。



即使项目数据库提供了建立关系的功能，但是建立起来还是比较耗时，而且很难更新。虽然这些关系对影响分析非常有用，但是对整个变更管理来说，并不是必需的。

标识配置对象时还必须考虑各对象之间的关系。例如，使用下面的简单表示：

```
Class diagram <part-of> analysis model;  
Analysis model <part-of> requirements specification;
```

就可以创建SCI的一个层次关系。

在很多情况下，对象是跨对象层次结构的分支相互关联的。这些交叉的结构关系可以用下面的方式表示：

```
data model <interrelated> data flow model;  
data model <interrelated> test case class m;
```

第一种相互关系存在于复合对象之间，而第二种相互关系则是存在于聚合对象（**DataModel**）和基本对象（**TestCaseClassM**）之间。

在配置对象的标识过程中必须注意到，对象在整个软件过程中是不断演化的。在一个对象被确定为基线之前，它可能会变更很多次，甚至在已经被确定为基线之后，变更也可能会经常发生。

27.3.2 版本控制



“make”工具使软件工程师能够提取所有相关的配置对象和构造软件的特定版本。

782

版本控制结合了规程和工具，可以用来管理在软件过程中所创建的配置对象的不同版本。版本控制系统实现或者直接集成了四个主要功能：（1）存储所有相关配置对象的项目数据库（中心存储库）；（2）存储配置对象所有版本（或能够通过与先前版本间的差异来构造任何一个版本）的版本管理功能；（3）使软件工程师能够收集所有相关配置对象和构造软件特定版本的make工具。此外，版本控制和变更控制系统通常还有问题跟踪（也叫做错误跟踪）功能，使团队能够记录和跟踪与每个配置对象相关的重要问题的状态。

“任何变更，即便是为了追求更好而进行的变更，都总是伴随着缺点和不快。”

——Arnold Bennett

很多版本控制系统都可以建立变更集——构造软件特定版本所需要的所有变更（针对某些基线配置）的集合。Dart [DAR91]写道：变更集“包含了对全部配置文件的所有变更、变更的理由、由谁完成的变更、以及何时进行的变更等详细信息”。

可以为一个应用程序或系统标识很多已命名的变更集。这样就使软件工程师能够通过指定必须应用到基线配置的变更集（按名称）来构造软件的一个版本。为了实现这个功能，就要运用系统建模方法。系统模型包括：（1）一个模板，该模板包含构件层次结构，以及构造

系统时构件的“创建次序”；（2）构造规则；（3）验证规则⁴。

在过去20年中，对于版本控制已经提出了很多不同的自动化方法，这些方法的主要区别在于构造系统特定版本和变体属性时的复杂程度以及构造过程的机制。

SOFTWARE TOOLS

并发版本系统

通过工具来实现版本控制是实现高效变更管理的基础。并发版本系统（concurrent versions system, CVS）是在版本控制中普遍使用的工具。最初是为源代码设计的，但是可运用于任何文本文件。CVS系统：（1）建立了简单的中心存储库；（2）以一个命名文件来维护文件的所有版本，仅仅存储原始文件与渐进版本之间的差异；（3）为每一个开发者建立不同的工作目录以实现相互隔离，可避免同时对某个文件进行变更。当每一个开发者完成各自的工作后，由CVS合并处理所有的变更。

要注意的是，CVS不是一个“构造”系统，即它不能构造软件的特定版本。CVS必须集成其他工具（如，Makefile）才能完成这项工作。CVS也不能实现变更控制过程（如变更请求、变更报告、错误跟踪）。

虽然有上述局限性，CVS仍然“是一个优秀的、源码开放的、网络透明的版本控制系统，从单个开发者到大型、分散的团队都可以使用”[CVS02]。CVS的客户/服务器体系结构使用户可以从任何有Internet连接的地方访问文件，且其源码开放的理念使其适用于大部分流行的平台。

可以免费获得Windows、Macintosh和UNIX环境下的CVS，更详细的信息请查看www.cvshome.org。

783

27.3.3 变更控制

James Bach[BAC98]很好地总结了现代软件工程范畴内变更控制的真实情况：

变更控制是至关重要的。但是，使变更控制至关重要的驱动力也使它令人厌烦。我们为变更所困扰，因为代码中一个极小的混乱就可能引起产品的失效；但是它也能够修复失效或具有奇妙的新能力。我们为变更所困扰，因为某个喜欢恶作剧的开发者可能破坏掉整个项目；但是，奇妙的想法也是源自那些喜欢恶作剧的人，而繁重的变更控制过程可能会严重阻碍他们进行创造性的工作。

Bach认为我们面临的是平衡问题。太多的变更控制会给我们带来这样的问题，太少的变更控制又会给我们带来那样的问题。

“改进的艺术是在变更中保持有序，同时在有序中保持变更。”

——Alfred North Whitehead

对于大型的软件工程项目，不受控制的变更会迅速导致混乱。对于这种大型项目，变更控制应该将人为制定的规程与自动工具结合起来。变更控制过程如图27-5所示，提交一个变

⁴ 也可以通过系统模型来评估一个构件中的变更将对其他构件产生怎样的影响。



应该注意到，多个变更请求可以综合起来只建立一个ECO，而这样的ECO通常引起对多个配置对象的变更。

更请求之后，要对其进行多个方面的评估：技术指标、潜在的副作用、对其他配置对象和系统功能的整体影响，以及预测变更成本。评估的结果形成变更报告，由变更控制授权人（change control authority, CCA，对变更的状态及优先级做出最终决策的人或小组）使用。对每一个被批准的变更，需要建立工程变更工单（engineering change order, ECO），ECO描述了将要进行的变更、必须要考虑的约束以及评审和审核的标准。

可以将要进行变更的对象放到一个目录中，该目录只能由实施变更的软件工程师单独控制。完成变更之后，版本控制系统（见前面CVS补充材料）可以更新原始文件。或者，可以将要进行变更的对象从项目数据库（中心存储库）中“检出”（check out），进行变更，并应用适当的SQA活动，然后，再将对象“检入”（check in）到数据库，并应用适当的版本控制机制（27.3.2节）构建该软件的下一个版本。



混乱会导致错误（其中有些错误是非常严重的）。

784

访问控制和同步控制能够避免混乱，版本控制和变更控制工具可以实现这两个功能。

以上版本控制机制与变更控制过程集成在一起，实现了变更管理的两个主要元素——访问控制和同步控制。访问控制负责管理哪个软件工程师有权限去访问和修改某个特定的配置对象；同步控制协助保证两个不同的人员完成的并行变更不会被相互覆盖[HAR89]。

有些读者可能开始对图27-5描述的变更控制过程所蕴含的繁文缛节感到不适应，这种感觉是很正常的。没有适当的安全措施，变更控制可能会阻碍进展，也可能会产生不必要的繁琐手续。大多数拥有变更控制机制的软件开发者（不幸的是，很多人没有）通过一些控制环节来帮助避免上面提到的这些问题。



选择比你认为需要的变更控制稍微多一点，有可能“太多”就是正好。

785

在SCI成为基线之前，只需要进行非正式的变更控制。还在讨论之中的配置对象（SCI）的开发者可以进行任何变更，只要项目和技术需求证明这些变更适当（只要变更不会影响到开发者工作范围之外的系统需求）。一旦配置对象经过正式技术评审并被批准，它就成为基线⁵。一旦SCI成为基线，就可以实现项目级变更控制了。这时，若要进行变更，开发者必须得到项目管理者的批准（如果变更“局部的”），如果该变更影响到其他SCI，则必须得到CCA的批准。在某些情况下，无需生成正式的变更请求、变更报告和ECO，但是，必须对每一个变更进行评估，并对所有的变更进行跟踪和评审。

当交付软件产品给客户时，正式的变更控制就开始实施了，正式的变更控制规程如图27-5所示。

“变更是不可避免的，自动贩卖机除外。”

—Bumper sticker

CCA在控制的第二层和第三层中起主动作用。CCA可以是一个人——项目经理，也可以是很多人（例如，来自软件、硬件、数据库工程、支持、市场等方面的代表），这取决于软件项目的规模和性质。CCA的作用是从全局的观点来评估变更对SCI之外的事物的影响。变更将

⁵ 也可以按其他方式创建基线。如，当设置为“每日创建”时，所有按指定时间提交的构件就成为第二天工作的基线。

对硬件产生什么影响？变更将对性能产生什么影响？变更将怎样改变客户对产品的感觉？变更将对产品的质量和可靠性产生什么影响？还有很多其他问题都需CCA来处理。

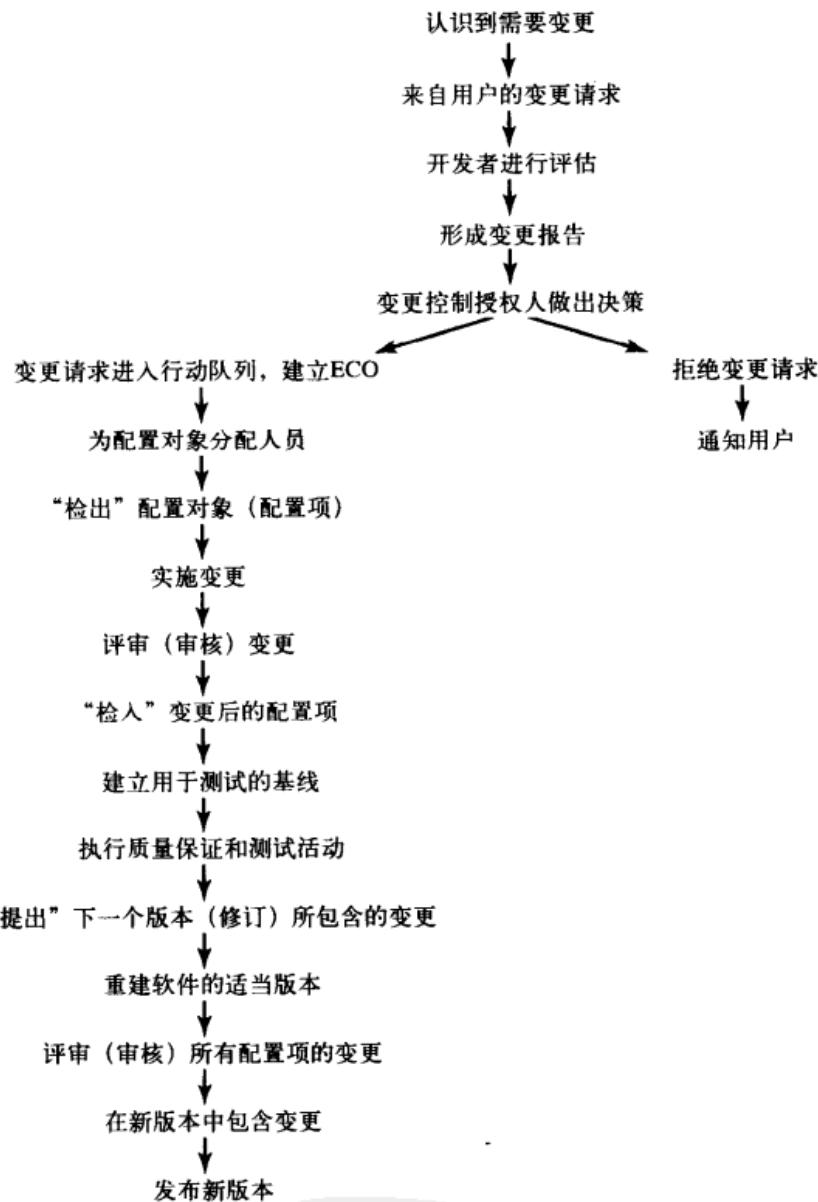


图27-5 变更控制过程

SAFEHOME

SCM问题

[场景] Doug Miller的办公室，SafeHome软件项目开始之初。

[人物] Doug Miller（SafeHome软件工程团队经理）、Vinod Raman、Jamie Lazar以及产品软件工程团队的其他成员。

【对话】

Doug: 我知道时间还早，但是我们应该开始讨论变更管理了。

Vinod（笑着说）：很难。今天早上市场部打电话来，有几个需要“重新考虑”的地方。

都不是主要的，但这只是刚刚开始。

Jamie：在以前的项目中，我们的变更管理就非常不正式。

Doug：我知道，但是这个项目规模更大、更显眼，使我想起……

Vinod（不断点头）：我们已经被“家庭照明控制”项目中不受控制的变更折腾的要命……想起延期就……

Doug（皱着眉）：我不愿意再经历这样的恶梦。

Jamie：那我们该做什么？

Doug：在我看来，三件事。第一件，我们必须建立（或借用）一个变更控制过程。

Jamie：你的意思是如何请求变更吗？

Vinod：是的。但也包括如何评估变更，如何决定何时进行变更（如果由我们决定），以及如何记录变更所产生的影响。

Doug：第二件，我们必须获得一个适用于变更控制和版本控制的SCM工具。

Jamie：我们可以为所有的工作产品创建一个数据库。

Vinod：在这里叫做SCI，绝大部分好的工具都不同程度地支持这个功能。

Doug：这是一个良好的开端，现在我们必须……

Jamie：嗯，**Doug**，你说过是三件事的。

Doug（笑着说）：第三件，我们必须使用工具，而且无论如何大家都必须遵守变更管理过程。行吗？

27.3.4 配置审核

标识、版本控制和变更控制帮助软件开发者维持秩序，否则情况可能将是混乱和不断变化的。然而，即使最优秀的控制机制也只能在ECO建立之后才可以跟踪变更。我们如何能够保证变更的实现是正确的呢？答案分两个方面：（1）正式技术评审；（2）软件配置审核。

正式技术评审（在第26章中已经详细讨论过）关注的是配置对象在修改后的技术正确性。评审者要评估SCI，以确定它与其他SCI是否一致，是否有遗漏，或是否有潜在的副作用。除了那些非常微不足道的变更之外，应该对所有变更进行正式技术评审。

软件配置审核作为正式技术评审的补充，要解决以下问题：

- 1. 在ECO中指定的变更已经完成了吗？引起任何额外的修改了吗？
- 2. 是否已经进行了正式技术评审来评估技术正确性？
- 3. 是否遵循了软件过程，是否正确地应用了软件工程标准？
- 4. 在SCI中“显著标明”所做的变更了吗？是否说明了变更日期和变更者？配置对象的属性反应出该变更了吗？
- 5. 是否遵循了SCM规程中标注变更、记录变更和报告变更的规程？
- 6. 是否已经正确地更新了所有相关的SCI？

在某些情况下，这些问题是在正式技术评审的一部分来询问的。但是，当SCM是一个正式的活动时，SCM审核将由质量保证小组单独进行。这种正式的配置审核还能够保证将正确的SCI（按版本）集成到特定的版本构造中，并且能够保证所有文档都是最新的，而且与所构造的版本是一致的。

在配置审核期间，我们要问的主要问题有哪些？

27.3.5 状态报告



为每一个配置对象建立一个“需知”清单，并且还要实时更新。当变更发生时，要确保通知到清单上的每一个人。

配置状态报告（有时称为状态账目）是一个SCM任务，它解答下列问题：(1)发生了什么事？(2)是谁做的？(3)是什么时候发生的？(4)会影响到别的什么？

配置状态报告（Configuration Status Reporting, CSR）的信息流如图27-5所示，每当赋予一个SCI新的标识或更改其标识时，就会产生一个CSR条目；每当CCA批准一个变更（即创建一个ECO）时，就会产生一个CSR条目；每当进行配置审核时，其结果要作为CSR任务的一部分而提出报告。CSR的结果可以放置到一个联机数据库中或Web站点上，以便软件开发者或维护人员可以按照关键词分类来访问变更信息。此外，定期生成的CSR报告使管理者和开发人员可以评估重要的变更。

SOFTWARE TOOLS

SCM的支持

目的：SCM工具可以支持27.3节所讨论的一个或多个过程活动。

机制：大部分最新的SCM工具都与一个中心存储库（一个数据库系统）相连，能够提供标识、版本控制和变更控制、审核及报告功能。

代表性工具⁶

CCC/Harvest，由Computer Associates (www.cai.com) 发行，是一个支持多种平台的SCM系统。

ClearCase，由Rational (www.rational.com) 开发，提供SCM的全部功能。

Concurrent Versions System (CVS)，是一个源码开放的工具 (www.cvshome.org)，是业界使用最广泛的版本控制系统之一。

PVCS，由Merant (www.merant.com) 发行，提供了适用于传统软件和WebApp的全套SCM工具。

SourceForge，由VA Software (sourceforge.net) 发行，提供了版本控制、构造功能、问题/错误跟踪及其他管理功能。

SurroundSCM，由Seapine Software (www.seapine.com) 开发，提供完整的变更管理功能。

Vesta，由Compac (www.vestasys.org) 发行，是一个能够支持小型(<10 KLOC)和大型(10 000 KLOC)项目的没有版权限制的SCM系统。

很多SCM工具与环境的列表可以在www.cmtoday.com/yp/commercial.html找到。

27.4 Web工程的配置管理

在本书的第三部分，我们讨论了Web应用系统的特殊属性，以及用来开发Web应用系统的Web工程过程。Web应用系统与传统软件的主要区别就是无处不在的变更。

⁶ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

788 Web工程一般采用迭代、增量过程模型（第16章），因为它具有很多敏捷软件开发（第4章）的原则。采用这种方法，通过客户驱动，软件工程团队通常可以在很短时间内开发出WebApp的增量。后续开发的增量只是对内容和功能的扩充，而每个增量都很可能要进行变更，这样可以使内容更丰富、使用更容易、界面更美观、导航栏更好、性能更高、安全性更强。因此，在Web工程的敏捷开发中，变更是有所不同的。

Web工程中一定存在变更，但是一般的敏捷团队都回避那些过程复杂、过于原则和抽象以及形式化的东西，而人们通常认为（虽然不确切）软件配置管理就具有这些特征。解决这个矛盾并不是要否定SCM原则、方法和工具，而是要使它们满足Web工程项目的特定要求。

27.4.1 WebApp的配置管理问题

不受控制的变更对Web-App具有什么影响？ 随着WebApp对企业的生存和发展越来越重要，对配置管理的需求也在增长。原因是什么呢？因为，如果没有有效的控制，对一个Web应用系统实施了不适当的修改（想想看，许多WebApp的主要特征就是即时性和持续演化），那么将导致未经许可就发布新产品信息；错误的或缺乏测试的功能可能会阻碍对Web站点的访问；安全漏洞可能会危害企业内部系统；还可能引起其他经济上的不满或更惨重的后果。

本章中介绍的这些常用软件配置管理（SCM）策略是适用的，但必须对这些策略和工具进行修改，以适应WebApp环境。在制定WebApp配置管理策略时应该考虑四个问题[DAR99]：内容、人员、可伸缩性和法规。

内容：一个典型的WebApp包括很多内容——文本、图形、Java程序、脚本、音频/视频文件、表单、动态页面元素、表格、流数据等等。难点在于如何将这些内容组织为一组合理的配置对象（27.1.4节），然后再为这些对象建立合适的配置控制机制。

人员：因为绝大部分WebApp仍然是以独特的方式开发的，所以任何一个与WebApp相关的人员都可以（且通常可以）创建内容。而多数内容创建者并没有软件工程知识，根本就不知道还需要配置管理，最终导致应用的增长和变更处于不受控状态。

789 可伸缩性：适用于小型WebApp的技术和控制并不能随规模按比例向上伸展。当将现有的信息系统、数据库、数据仓库以及门户站点互相联系起来时，显然一个简单WebApp会显著增长。随着规模和复杂度的增长，小的变更可能具有深远和无意识的影响，这种影响可能是有疑问的。因此，配置控制机制的严格程度应该与应用规模成正比。

如何确定由谁负责Web-App配置管理？ **法规：**谁“拥有”WebApp？不管是大公司还是小公司都在争论这个问题，而这个问题的答案对与WebE相关的管理和控制活动具有重大的影响。在某些场合，Web开发者不属于IT组织，从而形成了潜在的沟通困难。DART [DAR99]建议通过解决以下问题来帮助理解与WebE相关的法规：

- 谁负责保证Web站点上信息的正确性？
- 在信息被发布到站点之前，谁能够保证遵循了质量控制过程？
- 由谁负责完成变更？
- 由谁承担变更的成本？

上述问题的解答有助于确定组织机构中必须采用WebApp配置管理过程的人员。

27.4.2 WebApp的配置对象

WebApp包括很多配置对象：内容对象（如，文本、图形、图像、视频、音频）、功能构件（如，脚本、Java程序）和接口对象（如，COM或CORBA）。可以按任何方式来标识WebApp对象（指定文件名），只要适用于组织就行。但是，为了维护不同平台之间的兼容性，建议采用下面的约定：文件名长度应该不超过32个字符，不要使用大小写混合的或全部大写的名称，也不要使用下划线。另外，配置对象内的URL地址（超级链接）应该使用相对路径（如`../products/alarmsensors.html`）。

所有WebApp的内容都有形式和结构。内部文件的形式是由存储内容的计算机环境所决定的。而表现形式（常称为显示形式）是由WebApp的美学风格和设计规则决定的。内容结构定义了内容的体系结构，即，内容结构确定了装配内容对象的方法，从而能给最终用户呈现出有意义的信息。Boiko [BOI02]将结构定义为“覆盖一组内容块[对象]的图，用来将这些内容组织起来，使需要的人们可以访问到这些内容。”

27.4.3 内容管理

在某种意义上，内容管理和配置管理是相关的，因为内容管理系统（content management system, CMS）确定了如何（从大量WebApp配置对象中）获取现有内容、如何按照能够提交给最终用户的方式构造现有内容、然后在客户端环境下显示这些内容的过程（有适当的工具支持）。 790

“内容管理是解决当今信息风暴的一剂良药。”

——Bob Boiko

内容管理系统在创建动态WebApp的时候最常用。动态WebApp能够“动态地”(on-the-fly) 创建Web页面。即，由用户向WebApp请求特定信息，WebApp查询数据库并形成相应信息，然后提交给用户。例如，某音像公司提供了一个CD销售库，当一个用户想要了解某张CD或其电子产品时，可以查询数据库，下载全部有关该艺术家、CD（如，它的封面图片或图形）、音乐内容及音乐试听片段等信息，并配置到标准的内容模板中。最终的Web页面是在服务器端创建的，然后传送到客户端浏览器由最终用户进行验证。如图27-6所示。

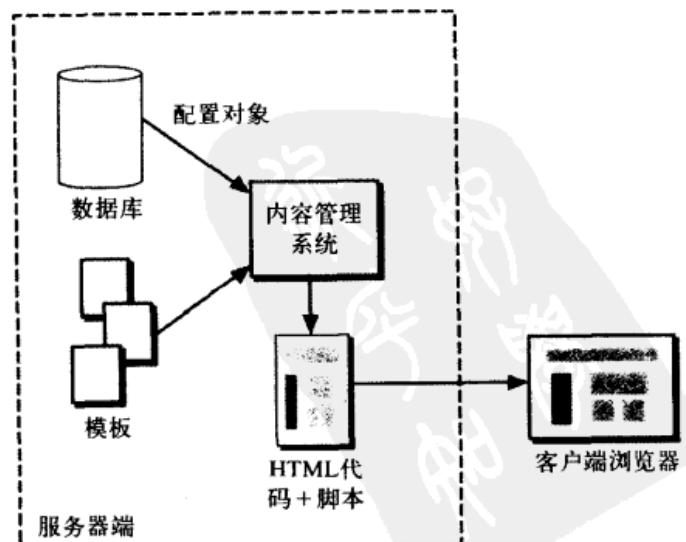


图27-6 内容管理系统 (CMS)

一般来说，CMS通过调用三个子系统来为最终用户“配置”完整的内容：收集子系统、791管理子系统、发布子系统[BOI02]。



收集子系统包含了用来创建内容、获取内容和（或）将内容转换成可以在客户端显示的形式的所有活动。



管理子系统实现了存储所有内容对象的中心存储库，在这个子系统中可以进行配置管理。



发布子系统从中心存储库中提取出内容，然后将内容传送到客户端浏览器。

收集子系统。内容指的是内容开发者必须创建和获取的数据和信息。收集子系统包含了用来创建和（或）获取内容的所有活动以及必要的技术，能够实现：（1）将内容转变成高级语言（如，HTML、XML）可以表示的形式；（2）将内容组织成可以在客户端有效显示的信息包。

管理子系统。收集到内容之后，必须将它分类保存到中心存储库中，以备随后的获取和使用，而且还要对它进行标注，以确定：（1）当前状态（如，内容对象是已经实现还是正在开发）；（2）内容对象的正确版本；（3）相关的内容对象。因此，管理子系统实现了包含下列元素的中心存储库：

- 内容数据库——为存储所有内容对象所创建的信息结构。
- 数据库功能——使CMS能够实现以下功能：查找特定内容对象（或对象的种类），保存和检索内容对象，以及管理为内容所创建的文件结构。
- 配置管理功能——支持内容对象标识、版本控制、变更管理、变更审核和变更报告的功能及相关工作流。

除了上述元素之外，管理子系统还实现了管理功能，包括对元数据的管理，以及对控制整个内容结构及支持方式的规则的管理。

发布子系统。可以从中心存储库中提取出内容，将内容转换为适于发布的形式，然后进行格式化以便传送到客户端浏览器。发布子系统通过一系列模板来完成这些任务。每一个模板对应一个功能，每一个功能都可以使用下面三个元素中的一个来构造一次发布[BOI02]：

- 静态元素——不需要进一步处理的文本、图形、媒体和脚本可以直接传送到客户端。
- 发布服务——调用特殊的检索服务和格式化服务功能来定制所需内容（使用预先制定的规则），完成数据转换，以及创建适当的导航链接。
- 外部服务——提供对外部共同信息（如，企业数据）或“内部”应用的访问。

包含以上三个子系统的[内容管理系统](#)可以适用于大型Web工程项目。但是，CMS的基本

792 理论和功能适用于所有的动态WebApp。

SOFTWARE TOOLS

内容管理

目的：辅助软件工程师和内容开发者管理WebApp的内容。

机制：这类工具使Web工程师和内容提供者能够以受控的方式更新WebApp的内容。大部分工具都建立了文件管理系统，可以按页面进行更新，并对各类WebApp内容进行编辑。一些工具还具有版本控制系统，可以获得实现历史目标的早期内容版本。

代表性工具⁷

Content Management Tools Suite，由interactivetools.com (www.interactivetools.com) 开

⁷ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

发，是一套主要应用于特定应用领域（如新闻文章、分类广告、房地产）的内容管理工具。

ektron-CMS300，由ektron (www.ektron.com) 开发，既提供内容管理功能，也提供Web开发工具。

OmniUpdate，由WebsiteASP, Inc. (www.omniupdate.com) 开发，是一个允许授权的内容提供者在受控的条件下对特定的WebApp内容进行更新的工具。

Tower IDM，由Tower Technologies (www.towertech.com) 开发，是一个文档处理系统和内容中心存储库，适用于管理各种形式的非结构化企业信息——图像、表单、计算机生成的报告、财务报表和发票、办公室文档、电子邮件及Web信息。

有关Web工程SCM和内容管理工具的其他信息见下列Web站点：

Web Developer's Virtual Encyclopedia (www.wdlv.com)，**WebDeveloper** (www.web-developer.com)，**Developer Shed** (www.devshed.com)，**Webknowhow.net** (www.web-knowhow.net) 或**WebReference** (www.webreference.com)。

27.4.4 变更管理

传统软件变更控制的工作过程（27.3.3节）对于Web工程来说通常太冗长了。按照大部分WebApp开发项目所接受的敏捷方式来完成变更请求、变更报告以及工程变更工单是不太可能的。那么我们该如何管理对WebApp内容和功能方面所提出的连续不断的变更请求呢？

WebApp开发中一直坚守的信念是“编码并马上运行”，为了实现此信念下的高效变更管理，就必须修改常规的变更控制过程。可以将每一个变更归为以下四种类型中的一种：

I类——纠正了一个错误或增加了局部内容或功能的内容或功能变更。

II类——对其他内容对象或功能构件具有影响的内容或功能变更。

III类——对整个WebApp具有重大影响的内容或功能变更（如，主要功能的扩充，重要内容的增加或减少，导航中必需的重要变更）。

IV类——立即会使广大用户能够注意到的大的设计变更（如，界面设计或导航方法的变更）。

对请求的变更进行了分类之后，就可以按照图27-7中所示的算法来处理。

在图27-7中，I类和II类变更可以看作是非正式的，并且可以按敏捷方式进行处理。对于I类变更，由Web工程师评估该变更的影响，但不需要任何外部的评审或文档。在实施变更时，配置中心存储库工具只需执行标准的检入（check-in）和检出（check-out）过程。对于II类变更，评审该变更对相关对象的影响是Web工程师的职责（也可以要求负责这些对象的开发者来进行评审）。如果该变更不会引起对其他对象的大量修改，则修改时就不需要其他的评审和文档。如果需要做大量的修改，就必须做进一步评估和编制计划。

III类和IV类变更也可以按敏捷方式进行处理，但是需要一些描述文档和较正式的评审过程。变更描述（描述变更，并对变更所产生的影响进行简单估算）适用于III类变更。将变更描述分发给Web工程团队的所有成员，由这些成员对其进行评审以更好地估算其影响。对于IV类变更也要进行变更描述，但在这种情况下，是由所有的共利益者进行评审的。

793

794

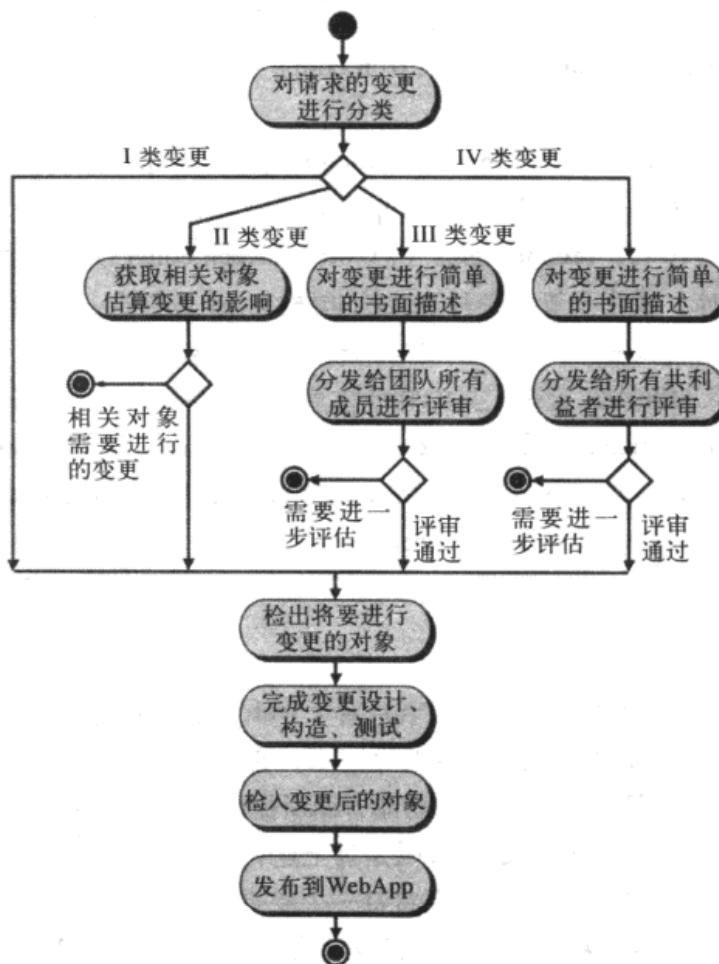


图27-7 WebApp的变更管理

SOFTWARE TOOLS**变更管理**

目的：辅助Web工程师和内容开发者对WebApp配置对象的变更进行管理。

机制：这类工具最初是为传统软件开发的，但是Web工程师可以对它进行修改，用来对WebApp进行受控方式的变更。

代表性工具⁸

ChangeMan WCM，由Serena (www.serena.com) 开发，是在成套变更管理工具中提供SCM功能的一种工具。

ClearCase，由Rational (www.rational.com) 开发，是一套提供全部WebApp配置管理功能的工具。

PVCS，由Merant (www.merant.com) 开发，是一套提供全部WebApp配置管理功能的工具。

Source Integrity，由mks (www.mks.com) 开发，是一个能够集成到指定开发环境的SCM工具。

⁸ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

27.4.5 版本控制

WebApp的发展过程中要经历很多增量，因此可能同时存在多个不同的版本。最终用户通过Internet可以访问某个版本（当前正在运行的WebApp），可能另一个版本（下一个WebApp增量）正处于配置之前测试的最后阶段；而正在开发的第三个版本在内容、界面美观以及功能上都有较大的改变。所以必须清晰地定义配置对象，以便各个配置对象与相应的版本相关联。[795]除此之外，还必须建立控制机制。Dreilinger [DRE99] 认为版本（及变更）控制的重要性如下：

在不受控制的站点中，由于多个创建者具有编辑和上传的权利，就会引起潜在的冲突和问题——如果这些创建者在不同时间、不同的办公室工作，就尤为如此。你可能在白天为客户修改了index.html文件，但在你完成更改后，下班后在家中工作的另一个开发者，或另一个办公室的开发者，可能在晚上上传他们自己最新修订的index.html文件，这样就无法挽回地完全覆盖了你的工作！

这种情形对于每个软件工程师或者Web工程师来说都不陌生。为了避免这种情形，就应该建立版本控制过程。

1. 应该建立WebApp项目的中心存储库。中心存储库中将保存所有WebApp配置对象（内容、功能构件及其他）的当前版本。
2. 每个Web工程师应该创建自己的工作目录。目录中包含了那些在特定时期内正在创建或修改的对象。
3. 所有开发者工作站的时钟应该同步。当两个开发者进行更新的时间互相非常接近时，这样做可以避免覆盖冲突。
4. 当开发新的配置对象或更改现有的对象时，必须将这些对象存入中心存储库。版本控制工具（见本章前面有关CVS的讨论）可以管理来自每一个Web工程师工作目录的所有检入和检出操作。
5. 当向中心存储库存入或取出对象时，自动创建带有时间戳的日志信息。这样可以提供有用的审核信息，这些信息还可以作为报告的一部分。

版本控制工具能够维护WebApp的不同版本，如果需要还能够恢复早先的版本。

27.4.6 审核和报告

在敏捷开发中，Web工程工作中的审核和报告功能是可以忽略的，但不是两个一起去掉。向中心存储库检入或检出的所有对象都被记录在日志中，任何时刻都可以评审这个日志。还可以创建完整的日志报告，这样Web工程团队的所有成员都可以得到指定时间期限内的变更日志。此外，每当向中心存储库检入或检出一个对象时，还可以发送自动电子邮件通知（发给那些感兴趣的开发者和共利益者）。[796]

INFO

SCM标准

下面的SCM标准列表（一部分从www.12207.com摘录）是比较全面的：

IEEE 标准 standards.ieee.org/catalog/oils/

IEEE 828 软件配置管理计划

IEEE 1042 软件配置管理

ISO标准 www.iso.ch/iso/en/ISOOnline.frontpage

ISO 10007-1995 质量管理, CM指南

ISO/IEC 12207 信息技术——软件生命周期过程

ISO/IEC TR 15271 ISO/IEC 12207指南

ISO/IEC TR 15846 软件工程——软件生命周期过程——软件配置管理

EIA标准 www.eia.org/

EIA 649 国家颁布的配置管理标准

EIA CMB4-1A 数字计算机程序的配置管理定义

EIA CMB4-2 数字计算机程序的配置标识

EIA CMB4-3 计算机软件库

EIA CMB4-4 数字计算机程序的配置变更控制

EIA CMB6-1C 配置和数据管理参考书

EIA CMB6-3 配置标识

EIA CMB6-4 配置控制

EIA CMB6-5 配置状态报告教科书

EIA CMB7-1 配置管理数据的电子交换

U.S.军方标准 www-library.itsi.disa.mil

DoD MIL STD-973 配置管理

MIL-HDBK-61 配置管理指南

其他标准

DO-178B 航空软件开发原则

ESA PSS-05-09 软件配置管理指南

AECL CE-1001-STD rev.1 安全至关重要软件的软件工程标准

DOE SCM 清单 cio.doe.gov/ITReform/sqse/download/cmcklst.doc

BS-6488 英国标准, 基于计算机的系统的配置管理

最优方法—UK 政府商业部门: www.ogc.gov.uk

CMII CM最佳实践学会: www.icmhq.com

配置管理资源指南给那些对CM过程和方法感兴趣的人们提供了其他信息, 见www.quality.org/config/cm-guide.html。

27.5 小结

软件配置管理是应用于整个软件过程的普遍性活动。SCM标识、控制、审核和报告修改总是发生在软件开发过程中及交付给客户之后。软件过程中产生的所有信息应该作为软件配置的一部分, 要适当地对配置进行组织, 才能进行有序的变更控制。

软件配置由一组相关联的对象(也称为软件配置项)构成, 这些对象是某些软件工程活动所产生的结果。除了文档、程序和数据外, 用于创建软件的开发环境也应该属于配置管理。应该将所有SCI存放在中心存储库中, 中心存储库具有保证数据完整性的机制和数据结构, 可

以支持其他软件工具，支持软件团队所有成员之间的信息共享，还具有版本控制和变更控制功能。

开发的配置对象经过评审之后，它就成为基线。对基线对象的变更将导致该对象新版本的创建。可以通过分析所有配置对象修订的历史记录来跟踪程序的演化过程。基本对象和复合对象可以形成一个对象池，通过对象池可以构建不同的版本。版本控制就是管理这些对象的一组规程和工具。

变更控制是一个过程活动，它能够在对配置对象进行修改时保证质量和一致性。变更控制过程从变更请求开始，然后决定是否拒绝该变更请求，最后，对将被修改的SCI进行可控制的更新。

配置审核是一个SQA活动，它有助于确保进行修改时仍能维护质量。状态报告为那些需要知道变更的人们提供了每次变更的信息。

Web工程的配置管理和传统软件的SCM在很多方面是相似的。但是，每个核心的SCM任务应该是最简单的，越小越好，而且必须能够达到内容管理的特殊规定。

参考文献

- [BAB86] Babich, W.A., *Software Configuration Management*, Addison-Wesley, 1986.
- [BAC98] Bach, J., "The Highs and Lows of Change Control," *Computer*, vol. 31, no. 8, August 1998, pp. 113–115.
- [BER80] Bersoff, E.H., V.D. Henderson, and S.G. Siegel, *Software Configuration Management*, Prentice-Hall, 1980.
- [BOI02] Boiko, B., *Content Management Bible*, Hungry Minds Publishing, 2002.
- [CHO89] Choi, S.C., and W. Scacchi, "Assuring the Correctness of a Configured Software Description," *Proc. 2nd Intl. Workshop on Software Configuration Management*, ACM, Princeton, NJ, October 1989, pp. 66–75.
- [CVS02] Concurrent Versions System Web site, www.cvshome.org, 2002.
- [DAR91] Dart, S., "Concepts in Configuration Management Systems," *Proc. Third International Workshop on Software Configuration Management*, ACM SIGSOFT, 1991, download from: http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_concepts.html.
- [DAR99] Dart, S., "Change Management: Containing the Web Crisis," *Proc. Software Configuration Management Symposium*, Toulouse, France, 1999, available at <http://www.perforce.com/perforce/conf99/dart.html>.
- [DAR01] Dart, S., *Spectrum of Functionality in Configuration Management Systems*, Software Engineering Institute, 2001, available at http://www.sei.cmu.edu/legacy/scm/tech_rep/TR11_90/TOC_TR11_90.html.
- [DRE99] Dreilinger, S., "CVS Version Control for Web Site Projects," 1999, available at <http://www.durak.org/cvswebsites/howto-cvs/howto-cvs.html>.
- [FOR89] Forte, G., "Rally Round the Repository," *CASE Outlook*, December 1989, pp. 5–27.
- [GRI95] Griffen, J., "Repositories: Data Dictionary Descendant Can Extend Legacy Code Investment," *Application Development Trends*, April 1995, pp. 65–71.
- [GUS89] Gustavsson, A., "Maintaining the Evolution of Software Objects in an Integrated Environment," *Proc. 2nd Intl. Workshop on Software Configuration Management*, ACM, Princeton, NJ, October 1989, pp. 114–117.
- [HAR89] Harter, R., "Configuration Management," *HP Professional*, vol. 3, no. 6, June 1989.
- [IEE94] *Software Engineering Standards*, 1994 edition, IEEE Computer Society, 1994.
- [JAC02] Jacobson, I., "A Resounding 'Yes' to Agile Processes—But Also More," *Cutter IT Journal*, vol. 15, no. 1., January 2002, pp. 18–24.
- [REI89] Reichenberger, C., "Orthogonal Version Management," *Proc. 2nd Intl. Workshop on Software Configuration Management*, ACM, Princeton, NJ, October 1989, pp. 137–140.
- [SHA95] Sharon, D., and R. Bell, "Tools That Bind: Creating Integrated Environments," *IEEE Software*, March 1995, pp. 76–85.

[TAY85] Taylor, B., "A Database Approach to Configuration Management for Large Projects," *Proc. Conf. Software Maintenance—1985*, IEEE, November 1985, pp. 15–23.

习题与思考题

- 27.1 使用UML的聚合或组合关系（第8章）来描述27.1.4节中列出的SCI（配置对象）间的相互关系。
- 27.2 实现高效SCM系统必需的四个元素是什么？简要介绍每一个元素。
- 27.3 用你自己的话谈谈定义基线的理由。
- 27.4 假定你是某个小项目的负责人，你会为项目定义什么基线？如何控制它们？
- 27.5 SCM审核和正式技术评审有什么区别？它们的作用可以放置在一个评审中吗？请说明正反两方面的观点？
- 27.6 研究某现有的SCM工具，然后大概描述它是如何实现版本控制和配置对象控制的。
- 27.7 设计一个项目数据库（中心存储库）系统，使软件工程师能够存储、交叉引用、跟踪、更新和更改所有重要的软件配置项。数据库应该如何处理同一程序的不同版本？源代码的处理会与文档的处理有所不同吗？两个开发者应该如何避免同时对同一个SCI执行不同的修改？
- 27.8 为什么“系统工程第一定律”会成立？变更的主要理由有四个，对每一个理由都举出几个特例。
- 27.9 设计一个用在配置审核中的检查表。
- 27.10 参照图27-5，为变更控制建立一个更详细的工作明细表。描述CCA的作用，并给出变更请求、变更报告和ECO的格式。
- 27.11 研究某现有的SCM工具，并描述它实现版本控制的方法。此外，阅读2~3篇有关SCM的文章，并描述用于版本控制的不同数据结构和引用机制。
- 27.12 关系〈part-of〉和〈interrelated〉表示了配置对象之间的简单关系，描述5种可能在
799 SCM中心存储库中用到的其他关系。
- 27.13 什么是内容管理？通过Web去研究内容管理工具的特性，并写一篇总结。
- 27.14 简要描述传统软件的SCM与WebApp的SCM之间有何不同。

推荐读物与阅读信息

Lyon为CM专业人员写了一本内容全面的指导书（《Practical CM》，Raven Publishing，2003，www.configuration.org），书中包含了实现配置管理系统的全方位的实用指导原则（每年更新）。Hass（《Configuration Management: Principles and Practice》，Addison-Wesley，2002）及Leon（《A Guide to Software Configuration Management》，Artech House，2000）对配置管理问题进行了探讨。White和Clemm（《Software Configuration Management Strategies and Rational ClearCase》，Addison-Wesley，2000）以当今流行的SCM工具为例，介绍了SCM。

Mikkelsen和Pherigo（《Practical Software Configuration Management: The Latenight Developer's Handbook》，Allyn & Bacon，1997）及Compton和Callahan（《Configuration Management for Software》，VanNostrand-Reinhold，1994）提供了有关重要SCM实践的实用教程。Ben-Menachem（《Software Configuration Management Guidebook》，McGraw-Hill，

1994) 及Ayer和Patrinnostro (《Software Configuration Management》, McGraw-Hill, 1992) 为那些想要进一步了解这个主题的人们提供了较详细的叙述。Berlack (《Software Configuration Management》, Wiley, 1992) 对SCM概念进行了探讨, 强调了中心存储库和工具在变更管理中的重要性。Babich [BAB86]提出了软件配置管理中实际问题的简单而有效的处理方法。Arnold和Bohner编写了一本关于在复杂软件系统中如何分析变更影响的文集 (《Software Change Impact Analysis》, IEEE Computer Society Press, 1996)。

Berczuk和Appleton (《Software Configuration Management Patterns》, Addison-Wesley, 2002) 介绍了有助于理解SCM和实现高效SCM系统的很多有用的模式。Brown等人 (《Anti-Patterns and Patterns in Software Configuration Management》, Wiley, 1999) 叙述了在SCM过程中不能做的那些事情 (反模式), 然后找出它们的解决办法。

Buckley (《Implementing Configuration Management》, IEEE Computer Society Press, 1993) 介绍了适用于所有系统元素 (硬件、软件和固件) 的配置管理方法, 并详细讨论了主要的CM活动。Rawlings (《SCM for Network Development Environments》, McGraw-Hill, 1994) 介绍了网络环境下进行软件开发对SCM的影响。Bays (《Software Release Methodology》, Prentice-Hall, 1999) 收集了对应用程序进行修改之后所可能引发的所有活动的最好实践。

由于WebApp的动态性, 使内容管理已经成为Web工程师关注的主题。Addey和他的同事 (《Content Management Systems》, Glasshaus, 2003)、Boiko [BOI02]、Hackos (《Content Management for Dynamic Web Delivery》, Wiley, 2002) 以及Nakano (《Web Content Management》, Addison-Wesley, 2001) 介绍了有价值的内容管理方法。

因特网上有大量关于软件配置管理的信息资源。在SEPA的Web站点<http://www.mhhe.com/pressman>就有最新的WWW参考列表。

800





第五部分

软件工程高级课题

在 本书这一部分中，我们考虑一些高级课题，了解这些能够加深对软件工程的理解。在下面几章中，我们将讨论以下问题：

- 形式化地描述软件需要什么表示法及数学预备知识（“形式化方法”）？
- 什么是净室软件工程过程中需要进行的关键技术活动？
- 如何运用基于构件的软件工程方法由可复用的构件构造系统？
- 软件再工程需要哪些技术活动？
- 软件工程的发展方向是什么？

通过对这些问题的回答，将有助于理解未来十年对软件工程有深远影响的课题。

801

第28章 形式化方法

要点浏览

概念：与传统方法相比，形式化方法能够使软件工程师创建更为完整、一致和无歧义性的规格说明。运用集合论和逻辑符号体系描述“事实（需求）的清晰陈述”。通过对这种数学规格说明的分析，可以提高（或证明）正确性及一致性。由于规格说明是用数学符号体系创建的，相对于非形式化的表示模式，前者从根本上具有较少的歧义性。

人员：由经过特殊训练的软件工程师编写形式化规格说明。

重要性：在安全至关重要或执行重要使命的系统中，故障会带来高昂的代价。当计算机软件出故障时，可能导致失去生命或严重的经济后果。在这种情况下，软件投入运行以前发现错误是很重要的。

形式化方法能在很大程度上减少规格说明错误，只要用户开始使用它，就为开发几乎没有错误的软件奠定了基础。

步骤：集合及构造性规格说明的符号体系及启发规则——集合运算符、逻辑运算符及序列——构成了形式化方法的基础。通过将非形式化的问题需求翻译成更形式化的表示，形式化方法为系统功能定义了数据不变式、状态及操作。

工作产品：应用形式化方法得到的产品是以形式语言（如OCL或Z）表示的规格说明。

质量保证措施：由于形式化方法运用离散数学作为规格说明机制，对于每一个系统功能，可以使用逻辑证明方法证明规格说明的正确性。但是，即使不使用逻辑证明，形式化规格说明的结构和规范也会使软件的质量得到提高。

关键概念
构造性规格说明
数据不变式
形式化规格说明
逻辑运算符
OCL
操作
前置条件和后置条件
模式
集合运算符
状态
Z语言

软件工程方法可以按“形式化”程度进行分类，“形式化”程度与在分析、设计中应用的数学严格程度有关。因此，本书前面讨论的分析与设计方法属于非形式化方法（形式化程度最低的极端情况）。这些方法运用图、文本、表格和简单的符号来创建分析和设计模型，但很少使用数学。

我们现在考虑形式化程度的另一极端情况。这里使用说明系统功能和行为的形式化语法和语义来描述规格说明和设计。规格说明是以数学形式表达的（例如，谓词演算可作为形式化规格说明语言的基础）。

Anthony Hall [HAL90] 在其关于形式化方法的入门讨论中说道：

形式化方法是有争议的。支持者声称形式化方法可以引发软件开发的革命；而批评者认为这是极端困难的。同时，对大多数人来说，他们对形式化方法很不熟悉，因此难于判断这些争论。

在本章，我们探索形式化方法以及它们对未来软件工程的潜在影响。

28.1 基本概念

《The Encyclopedia of Software Engineering》[MAR94]中对形式化方法的定义如下：

如果一个方法有良好的数学基础，特别地，是以形式化说明语言描述的，那么它是形式化的。这种数学基础提供了精确定义一致性和完整性等概念的表示方法，更进一步的是定义规格说明、实现和正确性。

形式化规格说明的期望特性——一致性、完整性及无歧义性——是所有规格说明方法的目标。然而，形式化方法的使用增加了实现这些理想的可能。规格说明语言的形式化语法（28.4节）使得能够唯一地解释需求或设计，从而排除了读者解释自然语言（例如英语）或图形表示时经常产生的歧义性。集合论和逻辑符号的描述工具（28.2节）使得可以清晰地陈述事实（需求）。要达到一致，在规格说明中某地方陈述的事实就不能与其他地方相矛盾。一致性是通过数学证明将初始事实形式化地映射（使用推理规则）到后面的规格说明中的陈述来保证的。

“形式化方法在改善需求规格说明的清晰性和精确性，以及在发现重要的和敏感的错误方面具有极大潜力。”

——Steve Easterbrook等

即使使用形式化方法，完整性也是难于达到的。当创建规格说明时，系统的某些方面可能尚未定义；某些特征可能被有意省略，以允许设计者在选择实现方法时具有一定自由度；最后，在一个大型复杂系统中，不可能考虑每一个操作场景。某些细节可能是由于错误而被遗漏。

虽然数学提供的形式化对某些软件工程师有吸引力，而另外一些人（有人说是大部分）却对软件开发的数学观点怀有疑问。为了了解为什么形式化方法有益于软件开发，我们必须首先考虑非形式化方法的不足之处。

803

28.1.1 非形式化方法的缺陷¹

本书第二、第三部分中讨论的分析和设计方法大量使用自然语言和多种图形表示。虽然认真地应用分析、设计方法，并结合彻底的评审能够得到高质量的软件，但是，这些方法应用的偏差可能产生各种问题。系统规格说明中可能包含矛盾、歧义性、含糊性、不完整陈述，以及不同抽象层次的混杂。



虽然一个好的文档索引不能消除矛盾，但它能够帮助我们发现矛盾。考虑为规格说明及其他文档建立一个索引。

上述的矛盾是指一组相互有分歧的陈述的集合。例如，系统规格说明的某部分可能规定系统必须监控化学反应堆中的所有温度，而规格说明的另一部分——可能因为是由另一个成员撰写，可能规定只监控在一定范围内的温度。

歧义性是指一种陈述可以以很多种方式解释。例如，下面陈述具有歧义性：

操作员标识由操作员姓名和口令组成；口令由6位数字构成。它应该在安全VDU上显示，并且当操作员登录进系统时被存放在注册文件中。

在这段摘录中，“它”到底代表“口令”还是“操作员标识”？

¹ 本节和本章第一部分的其他几节是根据Darrel Ince针对本书第五版的欧洲版本的文稿改编的。

因为系统规格说明是非常庞大的文档，所以含糊性经常发生。要求全部达到很高的精度几乎是不可能的。

“人总是会犯错误的，重复犯错误也是常见的。”

——Malcolm Forbes



有效形式化技术评审能够消除很多这类问题。然而，某些问题可能不会被发现。在设计、编码和测试中要警戒缺陷。

不完整性是系统规格说明中最常发生的问题之一。例如，考虑功能需求：

系统应该维护从位于水库中的深度传感器获取的每小时的水库深度，这些值应保留6个月。

这里描述了系统的主要数据存储部分。假设系统的某一命令是：

AVERAGE 命令的功能是在PC上显示某特定传感器在两个指定时间之间显示的平均水深。

假设在这个命令中没有更多的细节，那么，命令的细节将是严重不完整的。例如，命令的描述中没有包括：如果系统的用户指定的时间是在当前时间的6个月以前，会怎么样？

不同抽象层次的混杂是指非常抽象的陈述中随机地混杂了一些关于细节的低层次的陈述。虽然两种类型的陈述在系统规格说明中都重要，但是，过于混杂将使得非常难看清系统的整体功能结构。

28.1.2 软件开发中的数学

对于大型系统的开发者来说，数学有许多有用的性质。其性质之一是它能够简洁而准确地描述物理现象、对象或动作的结果。可以使用专业数学来描述基于计算机系统的规格说明，就如同电气工程师能够使用数学描述电路一样²。

数学支持抽象，因此它是优秀的建模工具。因为它是准确的，因此几乎没有歧义性。可以使用数学方法对规格说明进行验证，以揭示规格说明中的矛盾性和不完整性，并去除其含糊性。此外，使用数学可按条理化方式表示不同抽象层次的系统规格说明。

最后，数学作为软件开发工具，提供了高层验证的手段。使用数学证明来验证设计是否符合规格说明以及程序代码是否符合设计。

28.1.3 形式化方法概念

本节的目的是给出软件系统的数学规格说明中涉及的主要概念，而不是给读者堆砌太多的数学细节。为此，我们使用一些简单例子进行说明。

例1 符号表。使用一个程序来维护符号表，此符号表在许多不同类型的应用问题中频繁使用。它由一组没有重复的项构成。图28-1给出了一个典型的符号表例子，它表示的是操作系统用来保存系统用户名的表。其他表的例子如：工资管理系统中的职员名字表以及网络通信系统中的计算机名称表。

假设本例中的表包括的职员数量不大于MaxIds。为表设定限制的这一陈述是条件——被

² 有一点要注意，这一章中的数学的系统规格说明不如简单电路的数学规格说明简洁。软件系统极其复杂，期望用一行数学公式来描述是不现实的。

称为数据不变式——的一个构成成分。数据不变式是一个重要的概念，本章将经常用到这个概念。



数据不变式是条件的集合，这些条件在包含一组数据的系统的执行过程中保持恒真。



状态概念的另一种说法是数据决定状态。也就是说，你可以通过检查数据来看系统处于什么状态。

数据不变式是一个条件，它在包含一组数据的系统的执行过程中总保持为真。上面讨论的符号表的数据不变式有两个构成成分：(1) 表中包含的名字数不超过MaxIds；(2) 在表中没有重复的名字。在上面描述的符号表程序例子中，这意味着：在系统执行过程中无论什么时候检查符号表，它包含的职员标识符个数总是不超过MaxIds，并且没有重复。

另一个重要的概念是状态。许多形式化语言，如OCL（28.5节），使用第7章及第8章所讨论的状态概念；也就是说，系统可能处于多种状态之一，每一种状态都表示外部可观察到的行为模式。然而，在Z语言（28.6节）中，对术语状态有不同的定义。在Z语言（及相关的语言）中，系统的状态由系统的存储数据来表示（因此，Z语言给出了太多的状态来表示每种可能的数据配置）。在符号表程序的例子中使用后面的定义，状态就是符号表。

最后一个概念是操作，这是在系统中发生的读写数据的动作。在符号表程序中如果考虑从符号表加入或去除职员名，则它将关联两个操作：将一个指定名增加到符号表，以及从符号表中去除一个现有名³。如果程序提供检查某指定名是否包含在表中的机制，则将有一个返回某种指示值的操作，这个指示值表示名字是否在表中。

有三种类型的条件与操作相关联：不变式、前置条件和后置条件。不变式定义什么保持不变。例如，符号表有一个不变式表示元素的个数总是小于或等于MaxIds。前置条件定义一个特定操作有效的环境。例如，增加一个名字到职员标识符符号表的前置条件是有效的，仅当表中不含有所要加入的名字，而且在表中只有少于MaxIds的职员标识符个数时。操作的后置条件定义当操作完成后保证什么为真，这是通过它对数据的影响来定义的。在增加标识符到职员标识符符号表操作的例子中，后置条件将用数学方法描述表中已经增加了新标识符。



当你需要为一个相当复杂的功能开发一个数据不变式时，集体讨论技术非常有效。让软件团队的每个成员写下功能的范围、约束和界限，然后进行组合及编辑。

例2 块处理器。在操作系统中一个更重要的部分是文件子系统，该子系统维护由用户创建的文件。块处理器是文件子系统中的一部分。文件存储中的文件由存储设备上的存储块构成，在计算机的操作中，文件的创建和删除需要获取和释放存储块。为了处理这些，文件子系统维持一个未用（空闲）块池，并保持对当前使用块的跟踪。当块从被删除文件释放时，它们通常被加入到块队列中以等待进入未用块池。如图28-2所示，图中显示了一些部件：未用块池、被操作系统管理的文件使用的块，以及那些等待加入到未用块池中的块。等待块被组织为队列，队列中每个元素包含来自于被删除文件的一组块。

1. Wilson
2. Simpson
3. Abel
4. Fernandez
5.
6.
7.
8.
9.
10.

图28-1 符号表

³ 需要注意的是，在表满状态时不能增加名字，在表空状态时不能删除名字。

对这个子系统来说，状态是：空闲块的集合，已用块的集合，以及返回块的队列。数据不变式用自然语言表达如下：

807

- 块未同时被标记为未用和已用。
- 所有在队列中的块集合都将是当前已用块集合的子集。
- 队列元素不包含相同的块号。
- 已用块和未用块的集合将是组成文件的块的总和。
- 在未用块集合中没有重复的块号。
- 在已用块集合中没有重复的块号。

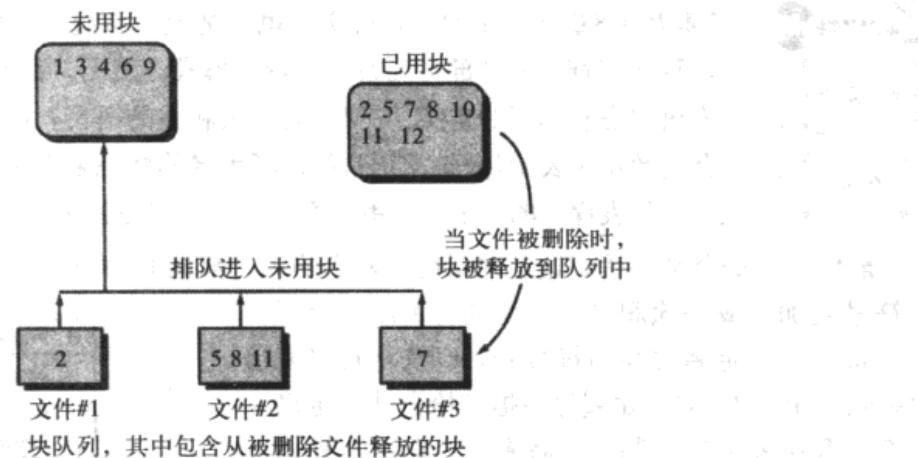


图28-2 块处理器

与数据不变式相关联的某些操作是：将一个块集合加（add()）到队列的末尾，从队列前面去除（remove()）一个已用块集合并将其放到未用块集合中，检查（check()）块队列是否为空。

第一个操作的前置条件是，将被加入的块必须在已用块集合中；后置条件是，这个块集合现在处于队列的末尾。第二个操作的前置条件是，队列中必须至少有一项；后置条件是，块必须加到未用块集合中。最后一个操作——检查返回块的队列是否为空——没有前置条件，这意味着不管状态具有什么值，操作总是有定义的。后置条件是：如果队列为空，返回true；否则，返回false。

在本节提到的每个例子中，我们已介绍了形式化规格说明的关键概念，但没有重点说明进行形式化规格说明时所需的数学知识。在28.2节中，我们将考虑这些数学知识。

28.2 数学预备知识

为了有效地应用形式化方法，软件工程师必须具有与集合和序列相关的数学符号知识，以及谓词演算中的逻辑符号方面的知识。本节的目的是做一简要介绍，更详细的讨论读者可以参考相关书籍（例如，[WIL87]、[GRI93]及[ROS95]）。

28.2.1 集合与构造性规格说明

集合是对象或元素的聚集，它是形式化方法的基础。集合中包含的元素是唯一的（即，不允许重复）。具有少量元素的集合用花括号括起来，元素间用逗号分开。例如，集合

{C++, Smalltalk, Ada, COBOL, Java}

包含五种程序设计语言的名字。

808

集合中元素出现的顺序是不重要的，集合中元素的数量称为集合的基数（cardinality），操作符#返回集合的基数，例如，表达式

{A, B, C, D} = 4

说明基数操作符被用于已知集合，其结果指出集合中项的数量。

 什么是构造性集合规格说明？

有两种定义集合的方式，一是通过枚举出集合的元素来定义（如上面的集合定义），二是创建一个构造性集合规格说明。集合成员的一般形式用布尔表达式来指定。因为构造性集合规格说明可以为大集合提供简洁的定义，所以它优于枚举方式。它也显式地定义了用于构造集合的规则。

考虑下面构造性规格说明的例子：

{n: N | n < 3 · n}

这个规格说明中有三个部分：特征n: N，谓词n < 3 · n，以及项n。特征描述在形成集合时考虑的值的范围，谓词（布尔表达式）定义集合如何被构造，最后，项给出了集合中项的一般形式。在上面例子中，N表示自然数，这样只考虑自然数；谓词指出只有小于3的自然数才包含在集合中；项规定集合中每个元素的形式为n。这样，上面的规格说明定义的集合为：

{0, 1, 2}

当集合元素的形式是明显的时候，项可以省略。例如，上面集合可表示为：

{n: N | n < 3}

 当开发形式化规格说明时，集合运算的知识是不可缺少的。如果你打算应用形式化方法，花时间熟悉每种集合运算。

上述的集合均只含单项元素。集合元素也可以是对、三元组等等。例如，集合规格说明

{x, y: N | x + y = 10 · (x, y²)}

描述了形为(x, y²)的自然数对的集合，这里x 和y 之和是10。下面是此集合：

{(1, 81), (2, 64), (3, 49), ...}

显然，表示某些计算机软件成分所需的构造性集合规格说明会比上面的例子复杂得多，然而，其基本形式和结构是相同的。

809

28.2.2 集合运算符

使用专门的符号集表示集合和逻辑操作，想应用形式化方法的软件工程师必须理解这些符号。

运算符∈用于表示集合中的成员关系，例如，表达式

x ∈ X

的值为真，如果x是集合X中的成员；否则其值为假。例如，谓词

12 ∈ {6, 1, 12, 22}

的值为真，因为12是集合中的成员。

运算符∈的反是运算符∉。表达式

x ∉ X

的值为真，如果x不是集合X的成员；否则为假。例如，谓词

$13 \notin \{13, 1, 124, 22\}$

的值为假。

运算符 \subset 和 \subseteq 将集合作为操作数，谓词

$$A \subset B$$

的值为真，如果集合 A 的成员均包含在集合 B 中；否则值为假。这样，谓词

$$\{1, 2\} \subset \{4, 3, 1, 2\}$$

值为真。然而，谓词

$$\{HD1, LP4, RC5\} \subset \{HD1, RC2, HD3, LP1, LP4, LP6\}$$

值为假，因为元素 $RC5$ 没有包含在运算符右边的集合中。

运算符 \subseteq 类似于 \subset ，然而，如果它的操作数相等，则它的值为真。这样，谓词

$$\{HD1, LP4, RC5\} \subseteq \{HD1, RC2, HD3, LP1, LP4, LP6\}$$

值为假，而谓词

$$\{HD1, LP4, RC5\} \subseteq \{HD1, LP4, RC5\}$$

值为真。

“数学结构是人类智慧的最美丽的发现之一。”

—Douglas Hofstadter

有一个特殊的集合，即空集 \emptyset ，它对应数学中的0。空集具有这样的性质：它是所有其他集合的子集。涉及空集的两个有用的等式是：

$$\emptyset \cup A = A \text{ 和 } \emptyset \cap A = \emptyset \text{ (对任何集合 } A)$$

其中， \cup 被称为“并”运算符，有时称作“cup”； \cap 被称为“交”运算符，有时称作“cap”。

并运算符以两个集合为操作数，结果为一个集合，它包含两个集合中的所有元素(排除重复)。这样，表达式

$$\{File1, File2, Tax, Compiler\} \cup \{NewTax, D2, D3, File2\}$$

的结果为集合 $\{File1, File2, Tax, Compiler, NewTax, D2, D3\}$

交运算符以两个集合为操作数，结果为包含每个集合中的公共元素的一个集合。这样，表达式

$$\{12, 4, 99, 1\} \cap \{1, 13, 12, 77\}$$

的结果为集合 $\{12, 1\}$ 。

集合“差”运算符 \setminus ，顾名思义，结果为从第一个操作数中去掉第二个操作数中的元素而得到的集合。这样，表达式

$$\{New, Old, TaxFile, SysParam\} \setminus \{Old, SysParam\}$$

的结果为集合 $\{New, TaxFile\}$ 。

下面表达式

$$\{a, b, c, d\} \cap \{x, y\}$$

的值为空集 \emptyset 。此运算符总是产生一个集合；然而，在此情形下，两个操作对象间没有公共元素，因此，结果集合中也没有元素。

最后一个运算符是叉积 \times ，有时也称为笛卡儿积。它以两个集合为操作数，其结果是两者元素对的集合，这里每个对由来自第一个操作数的元素和来自第二个操作数的元素构成。

下面是包含叉积表达式的例子，表达式

$$\{1, 2\} \times \{4, 5, 6\}$$

的结果为 $\{(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6)\}$ 。[811]

注意，第一操作数的每个元素均和第二操作数的每个元素组合。

对形式化方法很重要的一个概念是幂集 (powerset)，一个集合的幂集是该集合的子集的集合。本章中用来表示幂集运算符的符号是 \mathbb{P} 。它是一元运算符，当应用于某集合时，得到其操作数的子集的集合，例如：

$$\mathbb{P}\{1, 2, 3\} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

其中所有元素均是集合 $\{1, 2, 3\}$ 的子集。

28.2.3 逻辑运算符

形式化方法的另一个重要成分是逻辑：关于真、假表达式的代数。软件工程师都很了解常见逻辑运算符的意义。然而，和常见的程序设计语言有关的逻辑运算符是用键盘上已有的符号表示的，和这些符号等价的数学运算符为：

\wedge (合取符号)	与 (and)
\vee (析取符号)	或 (or)
\neg (否定符号)	非 (not)
\Rightarrow (推断符号)	蕴含 (implies)

全称量词是对集合中元素的一种陈述方法，该陈述对集合中每个元素都成立。全称量词使用符号 \forall ，例如，

$$\forall i, j: \mathbb{N} \cdot i > j \Rightarrow i^2 > j^2$$

该表达式陈述的是：对在自然数集合中的每一个值对，如果 i 大于 j ，则 i 的平方大于 j 的平方。

28.2.4 序列

序列是一种数学结构，用于对元素是有序的这一事实进行建模。一个序列 s 是“对”的集合，其元素的取值范围从 1 到最大数。例如，

$$\{(1, \text{Jones}), (2, \text{Wilson}), (3, \text{Shapiro}), (4, \text{Estavez})\}$$

是一个序列。形成“对”的第一个元素的项总称为序列的定义域，第二个元素的项总称为序列的值域。本书中序列用尖括号指明。例如，上面的序列通常写作：

$$\langle \text{Jones}, \text{Wilson}, \text{Shapiro}, \text{Estavez} \rangle$$

和集合不同，序列中的元素允许重复，且序列的顺序是重要的。因此，[812]

$$\langle \text{Jones}, \text{Wilson}, \text{Shapiro} \rangle \neq \langle \text{Jones}, \text{Shapiro}, \text{Wilson} \rangle$$

空序列表示为 $\langle \rangle$ 。

在形式化规格说明中使用大量序列运算符。连接 (catenation) 一是一个二元运算符，它通过将第二个操作数加到第一个操作数的尾部而形成新的序列。例如，

$$\langle 2, 3, 34, 1 \rangle - \langle 12, 33, 34, 200 \rangle$$

的结果为序列 $\langle 2, 3, 34, 1, 12, 33, 34, 200 \rangle$

其他的序列运算符有`head`、`tail`、`front`和`last`。运算符`head`抽出序列的第一个元素；`tail`返回长度为 n 的序列中后面的 $n-1$ 个元素所形成的序列；`last`抽出序列的最后一个元素；`front`返回长度为 n 的序列中前面的 $n-1$ 个元素形成的序列。例如，

```
head <2, 3, 34, 1, 99, 101> =2
tail <2, 3, 34, 1, 99, 101> = <3, 34, 1, 99, 101>
last <2, 3, 34, 1, 99, 101> =101
front <2, 3, 34, 1, 99, 101> = <2, 3, 34, 1, 99>
```

因为序列是“对”的集合，所有在28.2.2节中描述的集合运算符均可用于序列。当一个序列被用在状态中时，它应当通过关键字`seq`指明。例如，

`FileList: seq FILES`

`NoUsers: N`

描述了一个具有两个成分的状态：一个文件序列和一个自然数。

28.3 应用数学表示法描述形式化规格说明

为了说明数学表示法在软件构件的形式化规格说明中的使用，我们再一次考虑28.1.3节中提出的块处理器的例子。回顾一下，计算机操作系统中有一个重要构件，它维护用户创建的文件。块处理器维护一个未用块池，并同时保持对当前被使用块的跟踪。当块从被删除文件释放时，它们通常被加入到等待进入未用块池的块队列中。这一点如图28-2所示⁴。

如何使用前面已经介绍的集合和逻辑运算符来表示状态和数据不变式？

813

假定名为`BLOCKS`的集合包含所有块号，集合`AllBlocks`是位于1和`MaxBlocks`间的块的集合。状态将由两个集合和一个序列来模拟，两个集合分别是`used`和`free`。这两个集合均包含块，`used`集合包含当前被文件使用的块，`free`包含对新文件可用的块。序列将包含准备从已删除文件中释放的块的集合。状态可以被描述为：

```
used, free: ⊆ BLOCKS
BlockQueue: seq ⊆ BLOCKS
```

这和程序变量的声明非常相似，它说明`used`和`free`是块的集合，`BlockQueue`将是一个序列，序列中的元素是块的集合。数据不变式可以描述为：

$$\begin{aligned} used \cap free &= \emptyset \wedge \\ used \cup free &= AllBlocks \wedge \\ \forall i: \text{dom } BlockQueue \cdot BlockQueue[i] &\subseteq used \wedge \\ \forall i, j: \text{dom } BlockQueue \cdot i \neq j \Rightarrow BlockQueue[i] \cap BlockQueue[j] &= \emptyset \end{aligned}$$

WebRef

形式化方法的扩展信息可在
www.afm.sbu.ac.uk找到。

这个数据不变式的数学成分和以前描述的自然语言成分中的4条相匹配。数据不变式的第一行说明在块的`used`集合和`free`集合间不存在公共块；第二行说明`used`集合和`free`集合的并集总是等于系统中所有块的集合；第三行指出在块队列中的第 i 个元素总是`used`块的一个子集；最后一行说明对于块队列中任意两个不相同的元素，这两个元素间没有公共块。数据不变式的最后两条自然语言成分是基于这样的事实实现的：`used`和`free`是集合，并且没有重复元素。

⁴ 如果你记不清块处理器的例子，请返回到28.1.3节复习与块处理器相关的数据不变式、操作、前置条件和后置条件。

我们将定义的第一个操作是从块队列头去除一个元素，前置条件是队列中必须至少有一项：

$\#BlockQueue > 0$,

后置条件是队列头从队列中去除，并放入空闲块集合中，调整队列以显示这一操作：

$used' = used \setminus head BlockQueue \wedge$
 $free' = free \cup head BlockQueue \wedge$
 $BlockQueue' = tail BlockQueue$

在许多形式化方法中的一种习惯用法是以操作后的变量值为主，这样，上面给出的表达式的第一行说明新的已用块 ($used'$) 将等于旧的已用块减去已被去除的块；第二行说明新的空闲块 ($free'$) 将等于旧的空闲块加上块队列的头；第三行说明新的块队列将等于旧的块队列的尾，即除了第一个元素外的所有元素的队列。

814

如何表示前置条件和后置条件？ 第二个操作是将一个块集合 $Ablocks$ 加入到块队列中，前置条件是 $Ablocks$ 为当前已用块的集合：

如何表示前置条件？ $Ablocks \subseteq used$

后置条件是块集合被加入到块队列的尾部，同时已用块和空闲块的集合均保持不变：

$BlockQueue' = BlockQueue \cup \langle Ablocks \rangle \wedge$
 $used' = used \wedge$
 $free' = free$

毫无疑问，块队列的数学规格说明比自然语言叙述或图形模型要严格得多，这种严格需要更多的工作量，但从一致性和完整性的提高方面得到的好处可在很多类型的应用中得到证明。

28.4 形式化规格说明语言

形式化规格说明语言通常由三个主要成分构成：(1) 语法，定义用于表示规格说明的特定表示方法；(2) 语义，帮助定义用于描述系统的“对象的全域” [WIN90]；(3) 一组关系，定义确定哪个对象真正满足规格说明的规则。

形式化规格说明语言的语法域通常基于从标准集合论表示法和谓词演算导出的语法。例如，像 x 、 y 、 z 这样的变量描述一组和问题（有时称为论域）相关的对象，并与在 28.2 中描述的运算符联合使用。虽然语法通常是符号的，但如果图符（如方框、箭头和圆等图形符号）没有歧义性，也可以使用。

规格说明语言的语义域指出语言如何表示系统需求。例如，程序设计语言有一组形式语义使得软件开发者可以描述将输入转换成输出的算法。形式文法（如 BNF）可以用于描述程序设计语言的语法。然而，由于程序设计语言只能表示计算功能，所以它并不是好的规格说明语言。规格说明语言必须有更广的语义域，即，规格说明语言的语义域必须能够表达这样的概念：“对于在无限集 A 中的所有 x ，在无限集 B 中存在某个 y ，使得性质 P 对 x 和 y 成立” [WIN90]。其他规格说明语言可应用于描述系统行为的语义。例如，可以设计一种语法和语义用来描述状态和状态转换、事件，以及它们对状态转换的影响、同步及定时。

815

用不同的语义抽象及不同的方式来描述同一个系统是可能的，在第 8 章中我们用非形式化方法这样做了。我们依次描述了类、数据、功能及行为。不同的建模表示法可用于表示同一个系统。每种表示方法的语义提供了互补的系统视图。为了说明使用形式化方法时的情形，

假定使用一种形式化规格说明语言来描述在系统中使特定状态发生的事件的集合，另一种形式关系描述在某给定状态中出现的所有功能，这两种关系的交指明了使特定功能发生的事件。

目前已有许多形式化规格说明语言在使用。OCL[OMG03]、Z ([ISO02]、[SPI88]、[SPI92])、LARCH[GUT93]及VDM[JON91]，这些形式化规格说明语言都具有前面所述特性，并具有代表性。在本章中，我们给出OCL及Z的简要介绍。

28.5 对象约束语言⁵

对象约束语言（Object Constraint Language，OCL）是开发的形式化表示法，以使UML的使用者向他们的规格说明中加入更精确的内容。此语言具有逻辑数学及离散数学的所有优势。然而，OCL的设计者决定在OCL语句中只能使用ASCII字符，而不能使用传统的数学表示方法。这使得这种语言对于那些不太喜欢数学的人更友好，并且更易于被计算机处理。但这也使得OCL在某些地方有点冗长。

28.5.1 OCL语法及语义概述

WebRef

关于OCL的详细信息可以在 www-3.ibm.com/software/awdtools/library/standards/ocl.html 找到。

为了使用OCL，软件工程师需要从一个或多个UML图（通常为类图、状态图、活动图）开始。为此，我们增加说明图中元素的OCL表达式。这些表达式被称为约束（constraint）；从模型导出的任何实现必须保证每一个约束为恒真。

如面向对象程序设计语言一样，OCL表达式包括操作对象的操作符。然而，完整表达式的结果总是一个布尔值，即其值为true或false。对象可以是OCL聚集类的实例，集合和序列是OCL聚集类的两个子类。

在计算OCL表达式的上下文中，对象self是UML图的元素。可以从self对象使用点（dot）符号导航（navigating）获得其他对象。例如：

- 如果self是具有属性a的类C，则self.a计算存储在a中的对象的值。
- 如果C与另一个类D具有称为assoc的一对多的关联关系，则self.assoc计算元素类型为D的集合的值。
- 最后（更巧妙的），如果D有属性b，则表达式self.assoc.b计算属于所有D的所有b组成的集合的值。

OCL提供了内置运算符来实现28.2节中所描述的数学，当然还有更多。在表28-1中列出了部分。

表28-1 关键OCL符号一览

OCL 符 号	含 义
x.y	获得对象x的特性y。一个特性可以是一个属性、一个关联端的对象集合、一个操作的结果，或者依赖于UML图种类的其他事情。如果x是一个集合，则y被应用于x的每一个元素；结果形成一个新的集合
c->f()	将内置的OCL操作作用于聚集c本身（与作用于c中的每一个对象相反）。内置操作的例子在下面列出
and, or, =, <>	逻辑与，逻辑或，等于，不等于

⁵ 这一节由加拿大渥太华大学的Timothy Lethbridge教授提供，本书此处的摘录得到授权许可。

(续)

OCL 符 号	含 义
$p \text{ implies } q$	如果 q 为真, 或 p 为假, 则结果为真
作用于聚集(包括集合与序列)的操作示例	
$c \rightarrow \text{size}()$	聚集 c 中的元素个数
$c \rightarrow \text{isEmpty}()$	如果 c 没有元素, 则为真, 否则为假
$c1 \rightarrow \text{includesAll}(c2)$	如果 $c2$ 中的每一个元素在 $c1$ 中都存在, 则为真
$c1 \rightarrow \text{excludesAll}(c2)$	如果 $c2$ 中的元素在 $c1$ 中都不存在, 则为真
$c \rightarrow \text{forAll}(\text{elem} \mid \text{boolexpr})$	当 boolexpr 应用到 c 中的每一个元素都为真, 则为真。由于是计算元素, 因此这种计算限于可以在 boolexpr 中使用的变量 elem 。这就实现了前面所讨论的全局限制
$c \rightarrow \text{forAll}(\text{elem1}, \text{elem2} \mid \text{boolexpr})$	与上同, 除了对于 c 中的每一个可能的元素对, 包括由相同的元素组成的对, 都要计算 boolexpr 的值
$c \rightarrow \text{isUnique}(\text{elem} \mid \text{expr})$	当应用到 c 的每一个元素时, expr 得到不同的值, 则为真
针对集合操作的示例	
$s1 \rightarrow \text{intersection}(s2)$	在 $s1$ 中出现、也在 $s2$ 中出现的元素的集合
$s1 \rightarrow \text{union}(s2)$	在 $s1$ 中或在 $s2$ 中出现的元素的集合
$s1 \rightarrow \text{excluding}(x)$	将对象 x 去除后的集合 $s1$
针对序列的操作示例	
$\text{seq} \rightarrow \text{first}()$	序列 seq 的第一个元素

817

28.5.2 使用OCL举例

在本节中, 借助28.1.3节中介绍的块处理器实例, 解释使用OCL进行形式化规格说明。第一步是开发一个UML模型。在这个例子中, 我们从类图开始, 如图28-3所示。此图指明了所涉及的对象之间的许多关系; 然而, 我们一定要增加OCL表达式, 以确保系统的实现者更确切地知道当系统运行时什么需要保持为真。

我们将要增加的OCL表达式与在28.1.3节中讨论的不变式的六个部分相对应。对于每一部分, 我们将用自然语言重复不变式, 之后给出对应的OCL表达式。同时提供自然语言文本及形式逻辑是非常好的做法; 这样做可帮助读者理解逻辑, 还有助于评审者发现错误, 即英语与逻辑不对应的情形。

1. 块未同时被标记为未用和已用。

```
context BlockHandler inv:
  (self.used -> intersection(self.free)) -> isEmpty()
```

注意, 每一个表达式都以关键字**context**开头, 以指明此表达式所约束的UML图的元素。换句话说, 软件工程师可以在UML图上直接加约束, 以括号{}括起来。这里的关键字**self**指**BlockHandler**的实例; 下面我们将省略**self**, 如在OCL中允许的那样。

2. 所有在队列中的块集合都将是当前已用块集合的子集。

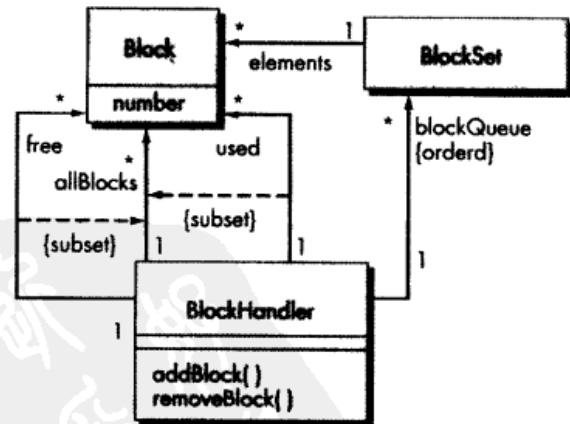


图28-3 块处理器的类图

context BlockHandler inv:
 818 blockQueue->forAll(aBlockSet | used->includesAll(aBlockSet))

3. 队列元素不包含相同的块号。

context BlockHandler inv:
 blockQueue->forAll(blockSet1, blockSet2 |
 blockSet1 <> blockSet2 implies
 blockSet1.elements.number->excludesAll(blockSet2.elements.number))

implies之前的表达式是需要的，以确保我们将两个元素是同一块的对忽略掉。

4. 已用块和未用块的集合将是组成文件的块的总和。

context BlockHandler inv:
 allBlocks = used->union(free)

5. 在未用块集合中没有重复的块号。

context BlockHandler inv:
 free->isUnique(aBlock | aBlock.number)
 6. 在已用块集合中没有重复的块号。
 context BlockHandler inv:
 used->isUnique(aBlock | aBlock.number)

OCL还可以用来指定操作的前置条件和后置条件。例如，考虑从队列中去掉块集合或者向队列中增加块集合。注意符号x@pre指对象x在操作之前；这与前面讨论的数学符号相反，在前面的讨论中，x在操作的后面，此操作被特指为x'。

```
context BlockHandler::removeBlocks()  

  pre: blockQueue->size() > 0  

  post: used = used@pre - blockQueue@pre->first() and  

    free = free@pre->union(blockQueue@pre->first()) and  

    blockQueue = blockQueue@pre->excluding(blockQueue@pre->first)  

context BlockHandler::addBlocks(aBlockSet :BlockSet)  

  pre: used->includesAll(aBlockSet.elements)  

  post: (blockQueue.elements = blockQueue.elements@pre  

    ->append(aBlockSet)) and  

    used = used@pre and  

    free = free@pre
```

OCL是一种建模语言，但它具有形式语言的所有特性。OCL允许各种约束的表达：前置条件，后置条件，警戒哨，以及与各种UML模型中所表示的对象相关的其他特性。

28.6 Z规格说明语言

Z（正确读音为“zed”）是在过去20年里发展起来的规格说明语言，已在形式化方法领域中广泛使用。Z语言在一阶谓词逻辑中应用类型集合、关系及函数来构造schema——一种构造形式化规格说明的工具。

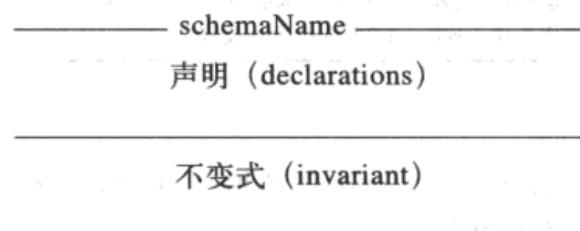
28.6.1 Z语法及语义概述

WebRef

关于Z语言的详细信息可在www-users.cs.york.ac.uk/~susan/abs/z.htm找到。

Z 规格说明由一组schema（盒子式的结构）组成，用于说明变量及这些变量之间的关系。一个schema 实质上类似于程序设计语言构件的形式化规格说明。采用与使用构件构造系统的同样的方式，schema用于构造形式化规格说明。

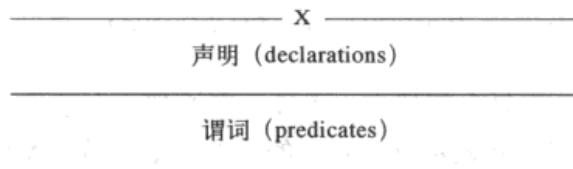
一个schema描述系统存取及修改的存储数据，在Z语言中被称为“状态”。Z语言中术语状态的用法与本书其他部分对这个词的使用有一点不同⁶。而且，schema标识系统内用于改变状态及关系的操作。schema的一般结构如下：



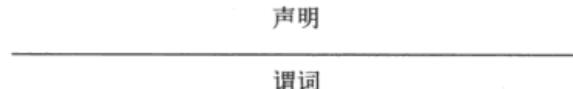
其中，声明表示组成系统状态的变量，不变式对状态演变的方式进行约束。Z 语言表示法的概要在表28-2 中给出。

表28-2 Z表示法概要

Z表示法基于类型集合论和一阶逻辑。Z提供了一个称为schema的结构，用来描述规格说明的状态空间和操作。一个schema将变量声明和限制变量的可能取值的一组谓词组合在一起。在Z 中， schema X 的定义形式如下：



全局函数和常量的定义形式如下：



声明给出函数或常量的类型，而谓词给出它的值。本表中仅列出Z 符号的简化集合。

集合 (set)：

$S : \mathbb{P} X$	S 被声明为 X 的集合
$x \in S$	x 是 S 中的成员
$x \notin S$	x 不是 S 中的成员
$S \subseteq T$	S 是 T 的子集： S 的每个元素均在 T 中
$S \cup T$	S 和 T 的并：包含在 S 或 T 或二者中的每个元素
$S \cap T$	S 和 T 的交：包含同时在 S 和 T 中的元素
$S \setminus T$	S 和 T 的差：包含在 S 中的而不是 T 中的每个元素
\emptyset	空集：不包含任何成员
$\{x\}$	单元素集：只包含 x
\mathbb{N}	自然数集合：0, 1, 2, ...
$S : \mathbb{F} X$	S 被声明为 X 的有限集

⁶ 回想一下，在其他章中，状态已被用来标识从外部观察到的系统的行为模式。

(续)

$\max(S)$	非空的数字集合S中的最大者
函数 (function):	
$f: X \rightarrow Y$	f 被声明为从 X 到 Y 的部分内射
$\text{dom } f$	f 的定义域, 定义 $f(x)$ 的 x 的值集
$\text{ran } f$	f 的值域, 当 x 在 f 的定义域上变化时, $f(x)$ 的值集
$f \oplus \{x \mapsto y\}$	与 f 相同的函数, 除了 x 被映射到 y
$\{X\} \triangleleft f$	一个和 f 相似的函数, 除了 x 被从定义域中去除
逻辑 (logic):	
$P \wedge Q$:	P 与 Q , P 与 Q 都为真时, 其值为真
$P \Rightarrow Q$	P 蕴含 Q : 或者 Q 为真或者 P 为假时, 其值为真
$\theta S' = \theta S$	在操作中, schema S 中没有成分改变

28.6.2 使用Z举例

在本节中, 我们使用Z规格说明语言对本章前面介绍的块处理器的例子建模。下面schema的例子描述了块处理器的状态和数据不变式:

—— BlockHandler ———

used, free : \mathbb{P} BLOCKS
BlockQueue : seq \mathbb{P} BLOCKS

used \cap free = \emptyset \wedge
used \cup free = AllBlocks \wedge
 $\forall i: \text{dom } \text{BlockQueue} \bullet \text{BlockQueue } i \subseteq \text{used} \wedge$
 $\forall i, j: \text{dom } \text{BlockQueue} \bullet i \neq j \Rightarrow \text{BlockQueue } i \cap \text{BlockQueue } j = \emptyset$

820
821

正如前面所述, 该schema包含两部分, 中线上面的部分表示状态变量, 而中线下面的部分描述数据不变式。只要schema表示改变状态的操作, 则以符号 Δ 为其前缀。下面schema的例子描述了从块队列中去除一个元素的操作:

—— RemoveBlocks ———

Δ BlockHandler

#BlockQueue > 0,
 $\text{used}' = \text{used} \setminus \text{head BlockQueue} \wedge$
 $\text{free}' = \text{free} \cup \text{head BlockQueue} \wedge$
 $\text{BlockQueue}' = \text{tail BlockQueue}$

对 Δ BlockHandler的包含使得所有构成状态的变量对RemoveBlock schema 是可用的, 并且保证在操作的执行前及执行后, 数据不变式都成立。

第二个操作, 加一个块集合到队列的尾部, 表示如下:

—— AddBlocks ———

Δ BlockHandler

Ablocks? : BLOCKS

Ablocks? \subseteq used

$\text{BlockQueue}' = \text{BlockQueue} \setminus (\text{Ablocks?}) \wedge$
 $\text{used}' = \text{used} \wedge$
 $\text{free}' = \text{free}$

根据Z中的约定，读入的输入变量如果不形成状态的一部分，则以问号结尾。这样，作为一个输入参数的Ablocks? 以问号结尾。

SOFTWARE TOOLS

形式化方法

目的：形式化方法工具是为了辅助软件团队进行规格说明及正确性验证。

机制：工具的机制各异。一般情况下，工具辅助进行规格说明及自动的正确性证明，通常通过定义一种专业语言进行定理证明。很多工具没有商业化，只是用于研究目的。

代表性工具⁷

ACL2，由德州大学（www.cs.utexas.edu/users/moore/acl2/）开发，它是“一种程序设计语言，可用于计算机系统建模，同时也是一种工具，可用于证明那些模型的性质”。

EVES，由加拿大的ORA（www.ora.on.ca/eves.html）开发，实现了用于形式化规格说明及自动证明产生器的Verdi语言。

在<http://www.afm.sbu.ac.uk/>可找到90多种形式化方法工具。

822

28.7 形式化方法的十条戒律

在现实世界中作出使用形式化方法的决定并不是一件简单的事，Bowen 和Hinchley [BOW95]提出了“形式化方法的十条戒律”，对于想应用这一重要软件工程方法的人们来说，这些可作为指南⁸。

1. 应该选择合适的表示法。为了从众多的形式化规格说明语言中进行有效的选择，软件工程师应该考虑语言词汇、被指定的应用问题类型及语言的使用广度。
2. 应该形式化，但不要过分形式化。通常没有必要对主要系统的各个方面都应用形式化方法。那些安全关键的构件是首选对象，然后是那些不能容忍失效的构件（出于商业理由）。
3. 应该估计成本。形式化方法的启动成本很高，人员的培训、支持工具的获取以及合同顾问的雇用使首次投资很高，在考察与形式化方法相关的投资回报时必须考虑这些成本。
4. 应该有随时可以请教的形式化方法顾问。第一次使用形式化方法时，专家培训和咨询是成功的关键。
5. 不应该放弃传统的开发方法。可以将形式化方法与传统方法或面向对象的方法相集成（本书的第二部分），在很多情况下这也是人们所希望的。每种方法都有各自的优点和缺点，合理地进行组合能够产生很好的结果⁹。
6. 应该建立详尽的文档。形式化方法对建立系统需求文档提供了简洁的、无歧义的和一

823

⁷ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的一些样例。在大多数情况下，工具的名字由各自的开发者注册为商标。

⁸ 这种处理是[BOW95]的一个缩略版本。

⁹ 净室软件工程（第29章）就是一个集成多种方法的实例，其中使用了形式化方法及许多传统开发方法。

致的方法。然而，推荐使用自然语言注释配合形式规格说明，以增强读者对系统的理解。

7. 不应该对质量标准作任何折衷。“形式化方法并不是什么有魔力的东西” [BOW94]，为此，在系统开发过程中，必须坚持应用其他软件质量保证SQA活动（第26章）。

8. 不应该教条化。软件工程师必须认识到形式化方法并不是正确性的保证。即使在开发中使用形式化方法，最终系统仍可能（有人说很可能）有小的遗漏、小的bug以及其他不满足期望的属性。

9. 应该测试、测试、再测试。软件测试的重要性已在第13章和第14章讨论过。形式化方法并不能免除对软件工程师提出的进行良好策划及彻底测试的要求。

10. 应该复用。长期来说，减少软件成本和增加软件质量的唯一合理的方法是复用（第30章）。形式化方法并没有改变这个现实。事实上，形式化方法是一种合适的创建可复用构件库的途径。

28.8 形式化方法——未来之路

虽然基于形式的、数学的规格说明技术还没有在产业界广泛应用，但它确实比非形式化方法有实质性的优点。Liskov 和 Bersins [LIS86] 给出了下面的总结：

形式化方法可以用数学方法来研究，而非形式化方法则不能。例如，一个正确的程序可以被证明满足其规格说明，或两个规格说明的集合可以被证明是相等的……某些形式的不完整性和不一致性可以被自动地检测。

此外，形式化规格说明消除了歧义性，并在软件工程过程的早期阶段鼓励使用更严格的方法。

但是，问题仍然存在。形式化规格说明主要关注于功能和数据，而问题的时序、控制和行为等方面却更难于表示。此外，有些问题元素（如，人机界面）最好用图形技术或原型来描述。最后，使用形式化方法来建立规格说明比结合了UML表示法的其他方法更难于学习，
824 并且对某些软件专业人员来说，它代表了一种重要的“文化冲击”。

28.9 小结

形式化方法提供了规格说明环境的基础，它使得所生成的分析模型比用传统的或面向对象的方法生成的模型更完整、一致和无歧义。集合论和逻辑符号描述工具使得软件工程师能创建事实（需求）的清晰陈述。

支配形式化方法的基本概念是：(1) 数据不变式——一个条件表达式，它在包含一组数据的系统的执行过程中总保持为真；(2) 状态——是从系统的外部能够观察到的行为模式的一种表示，或者系统访问和修改的存储数据（在Z语言或者相关的语言中）；(3) 操作——系统中发生的动作，以及对状态数据的读写。每一个操作都和两个条件相关联，即前置条件和后置条件。

离散数学——与集合和构造性规格说明、集合运算符、逻辑运算符以及序列相关的表示法——形成了形式化方法的基础。离散数学在形式化规格说明语言（如OCL和Z语言）中被实现。这些形式化规格说明语言具有语法域和语义域。语法域使用与集合和谓词演算紧密联系的表示法，语义域使得语言能够以简洁的方式表达需求。

使用形式化方法的决策必须考虑启动成本以及与不同技术相关的文化方面的变更。在大多数情况下，形式化方法对安全关键和业务关键的系统具有最高的回报。

参考文献

- [BOW95] Bowen, J. P., and M. G. Hinchley, "Ten Commandments of Formal Methods," *Computer*, vol. 28, no. 4, April 1995.
- [GRI93] Gries, D., and F. B. Schneider, *A Logical Approach to Discrete Math*, Springer-Verlag, 1993.
- [GUT93] Guttag, J. V., and J. J. Horning, *Larch: Languages and Tools for Formal Specification*, Springer-Verlag, 1993.
- [HAL90] Hall, A., "Seven Myths of Formal Methods," *IEEE Software*, September 1990, pp. 11-20.
- [HOR85] Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall International, 1985.
- [ISO02] *Z Formal Specification Notation—Syntax, Type System and Semantics*, ISO/IEC 13568:2002, Intl. Standards Organization, 2002.
- [JON91] Jones, C. B., *Systematic Software Development Using VDM*, 2nd ed., Prentice-Hall, 1991.
- [LIS86] Liskov, B. H., and V. Berzins, "An Appraisal of Program Specifications," in *Software Specification Techniques*, N. Gehani and A. T. McKittrick (eds.), Addison-Wesley, 1986, p. 3.
- [MAR94] Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering*, Wiley, 1994.
- [OMG03] "Object Constraint Language Specification," in *Unified Modeling Language*, v2.0, Object Management Group, September 2003, download from www.omg.org.
- [ROS95] Rosen, K. H., *Discrete Mathematics and Its Applications*, 3rd ed., McGraw-Hill, 1995.
- [SPI88] Spivey, J. M., *Understanding Z: A Specification Language and Its Formal Semantics*, Cambridge University Press, 1988.
- [SPI92] Spivey, J. M., *The Z Notation: A Reference Manual*, Prentice-Hall, 1992.
- [WIL87] Wiltala, S. A., *Discrete Mathematics: A Unified Approach*, McGraw-Hill, 1987.
- [WIN90] Wing, J. M., "A Specifier's Introduction to Formal Methods," *Computer*, vol. 23, no. 9, September 1990, pp. 8-24.
- [YOU94] Yourdon, E., "Formal Methods," *Guerrilla Programmer*, Cutter Information Corp., October 1994.

825

习题与思考题

- 28.1 使用表28-1或表28-2列出的OCL或Z表示法，选择本书前面描述的SafeHome安全系统的某个部分，并尝试使用OCL或Z进行说明。
- 28.2 开发28.4题的状态及数据不变式的数学描述。用OCL或Z规格说明语言对这种描述进行改进。
- 28.3 你已被分配到一个开发传真调制解调器软件的项目组，你的任务是开发应用系统的电话簿部分。电话簿功能可以存储多达MaxNames个地址名，相关的公司名、传真号码以及其他相关信息。使用自然语言定义：
 - a. 数据不变式。
 - b. 状态。
 - c. 可能的操作。
- 28.4 开发28.3题的状态及数据不变式的数学描述。用OCL或Z规格说明语言对这种描述进行改进。
- 28.5 你已被分配到一个开发内存加倍软件的项目组，该软件为PC机提供比物理内存更大的可见内存。这个功能是通过标识、收集和重分配已经被分配给现有应用但未被使用的内存块而实现的。未用块被重分配给需要追加内存的应用。做适当的假设并用自然语言定义：

- a. 数据不变式。
 - b. 状态。
 - c. 可能的操作。
- 28.6 给出某集合的构造性规格说明，该集合包含形为 (x, y, z^2) 的自然数三元组，其中 x 和 y 之和等于 z 。
- 28.7 用逻辑和集合运算符为下面的陈述写出一个表达式：“对所有 x 和 y ，如果 x 是 y 的父辈且 y 是 z 的父辈，则 x 是 z 的祖父辈。每个人都有父辈。”提示：用函数 $P(x, y)$ 和 $G(x, z)$ 分别表示父辈和祖父辈函数。
- 28.8 给出集合的构造性规格说明，这里每个对的第一个元素是两个非零自然数的和，而第二个元素是这两个数的差。两个数应该在 100 到 200 之间。
- 28.9 基于 PC 应用的安装程序首先确定是否存在可接受的硬件和系统资源集合，它检查硬件配置以确定是否存在各种设备（很多可能设备中的），并且确定是否已安装了系统软件和驱动程序的某特定版本。哪种集合运算符可以完成该功能？给出一个例子。
- 28.10 复习 28.1.1 节中与软件工程的非形式化方法相关的各种不足。根据你的经验对每种不足提供三个实例。
- 826** 28.11 本章用很长的篇幅讨论了将数学作为规格说明工具的好处。是否有不好的地方？
- 28.12 参考本章参考文献或阅读材料中的信息，就 OCL 或 Z 以外的形式化规格说明语言的基础语法及语义做一个演讲，时间大约半小时。

推荐读物与阅读信息

除了本章中作为参考文献使用的书目外，过去几十年中已出版了大量的关于形式化方法的书籍。下面是一些有价值的书籍列表：

- Bowan, J., *Formal Specification and Documentation using Z: A Case Study Approach*, International Thomson Computer Press, 1996.
- Casey, C., *A Programming Approach to Formal Methods*, McGraw-Hill, 2000.
- Clark, T., et al. (eds.), *Object Modeling with OCL*, Springer-Verlag, 2002.
- Cooper, D., and R. Barden, *Z in Practice*, Prentice-Hall, 1995.
- Craigie, D., S. Gerhart, and T. Ralston, *Industrial Application of Formal Methods to Model, Design and Analyze Computer Systems*, Noyes Data Corp., 1995.
- Harry, A., *Formal Methods Fact File: VDM and Z*, Wiley, 1997.
- Hinchley, M., and J. Bowan, *Applications of Formal Methods*, Prentice-Hall, 1995.
- Hinchley, M., and J. Bowan, *Industrial Strength Formal Methods*, Academic Press, 1997.
- Hussmann, H., *Formal Foundations for Software Engineering Methods*, Springer-Verlag, 1997.
- Jacky, J., *The Way of Z: Practical Programming with Formal Methods*, Cambridge University Press, 1997.
- Monin, F., and M. Hinchley, *Understanding Formal Methods*, Springer-Verlag, 2003.
- Rann, D., J. Turner, and J. Whitworth, *Z: A Beginner's Guide*, Chapman and Hall, 1994.
- Ratcliff, B., *Introducing Specification Using Z: A Practical Case Study Approach*, McGraw-Hill, 1994.
- Sheppard, D., *An Introduction to Formal Specification with Z and VDM*, McGraw-Hill, 1995.
- Warmer, J., and A. Kleppe, *Object Constraint Language*, Addison-Wesley, 1998.

Dean (《Essence of Discrete Mathematics》, Prentice-Hall, 1996), Gries和Schneider [GRI93], 及Lipschultz和Lipson (《2000 Solved Problems in Discrete Mathematics》, McGraw-Hill, 1991) 为想更多地学习形式化方法的数学基础的人们展示了有用的信息。

在网上有大量形式化方法的信息。最新的WWW文献列表可在SEPA Web站点<http://www.mhhe.com/pressman>找到。

[827]



第29章 净室软件工程

要点浏览

概念：你曾听人说过多少次“第一次就将事情做正确”？这是软件工程最重要的哲学。净室软件工程是这样一个过程：它强调在程序构造开始前进行正确性的数学验证，并且将软件可靠性认证作为软件测试的一部分。其结果是极低的故障率，这是使用非形式化方法难于或不可能达到的。

人员：经过特殊训练的软件工程师。

重要性：错误导致返工。返工需要时间，并增加成本。如果我们能够大量地减少在软件设计和构造中引入的错误（bug）的数量，不是很好吗？这正是净室软件工程的前提。

步骤：使用盒结构来创建分析和设计模型。“盒”在特定的抽象级别上封装系统

（或系统的某些方面）。一旦盒结构设计完成，即开始正确性验证。一旦对每个盒结构完成了正确性验证，则开始进行“统计使用测试”。通过定义一组使用场景来测试软件，确定对每个场景的使用概率，然后定义符合这些概率的随机测试。分析产生的错误记录，使得能够用数学的方法计算软件构件的预计可靠性。

工作产品：开发黑盒、状态盒、清晰盒规格说明。记录形式化的正确性证明和“统计使用测试”的结果。

质量保证措施：将形式化的正确性证明应用于盒结构规格说明，将“统计使用测试”方法应用于测试使用场景，以保证发现和改正用户功能方面的错误。测试数据可用于提供软件可靠性的指标。

关键概念
黑盒规格说明
盒结构
净室战略
认证
清晰盒规格说明
设计精化
功能规格说明
正确性证明
状态盒规格说明
统计使用测试
子证明
验证

传统的软件工程建模（可能使用形式化方法）、程序验证（正确性证明）以及统计SQA的集成使用组合成了一种可以开发极高质量软件的技术。净室软件工程（cleanroom software engineering）是一种在软件开发过程中强调在软件中建立正确性要求的方法。与传统的分析、设计、编码、测试和调试的周期观点有所不同，净室方法的观点[LIN94]如下：

净室软件工程背后的哲学是：通过在第一次正确地书写代码增量，并在测试前验证它们的正确性来避免成本很高的缺陷消除过程。它的过程模型是在代码增量积聚到系统的过程的同时进行代码增量的统计质量验证。

净室方法在很多方面将软件工程提升到另一个层次。如第28章中讨论的形式化方法一样，净室过程强调规格说明和设计上的严格性，并且使用基于数学的正确性证明来对结果设计模型的每个元素进行形式化验证。作为对形式化方法的扩展，净室方法还强调统计质量控制技术，包括基于客户对软件的预期使用进行测试。

当现实世界中软件失效时，会立即带来危险，甚至是长期的危险。这些危险可能和人的安全、经济损失或业务和社会基础设施的有效运作相关。净室软件工程是一个过程模型，它在缺陷可能产生严重的危险前消除缺陷。

29.1 净室方法

“净室”原理在硬件制造技术方面是相当简单的：建立一种避免引入产品缺陷的制造方法，在成本和时间方面都是合算的。净室方法不是先制造一个产品，然后再去消除缺陷，而是要求在规格说明和设计中消除错误，因而，制造是以“干净”方式进行的。

20世纪80年代，Mills、Dyer及Linger[MIL87]最先建议将净室原理应用于软件工程。虽然使用这种严格方法进行软件开发的早期经验显示了很引人的前景 [HAU94]，但它并没有得到广泛的使用。Henderson[HEN95]为此提出了三个可能的原因：

1. 净室方法学太理论化、太数学化、太激进，难于在实际软件开发中使用。
2. 它提倡开发者不需要进行单元测试，而是进行正确性验证和统计质量控制，这些观念与当前大多数软件开发方式有很大的偏离。
3. 软件开发产业的成熟度。使用净室过程需要在整个生命周期阶段严格按照定义的过程进行。由于大多数软件企业的运作还处于过程成熟度中的较低级别，软件工程师并没有为应用净室技术做好准备。

虽然在上面提到的每个顾虑中都有真实的成分，但是，净室软件工程的潜在益处远远超出要克服文化阻力（这才是这些顾虑的核心）所需要的投资。

“程序中出现错误的唯一方式是作者将错误引入进去的。没有其他方式……正确实践的目标是：设法避免引入错误，如果引入了错误，通过测试或任何其他运行程序的方式来消除错误。”

——Harlan Mills

829

29.1.1 净室策略

净室方法使用增量过程模型（第3章）的专业版。一个“软件增量的流水线”[LIN94]由若干小的、独立的软件团队开发。每当一个软件增量通过认证，它就被集成到整体系统中。因此，系统的功能随时间增加。

每个增量的净室任务序列在图29-1中给出。整个系统或产品需求都使用第6章讨论的系统工程方法开发。一旦将功能分配给系统的软件元素，则可启动净室增量的流水线。需要完成的任务如下：

作为净室软件工程一部分的主要任务是什么？

- **增量策划。**制定一个采用增量策略的项目计划，确定每个增量的功能、预计规模及净室开发进度。必须特别小心，以保证通过认证的增量能够被及时集成。
- **需求收集。**使用类似于第7章介绍的技术，开发更详细的客户级需求描述（为每个增量）。
- **盒结构规格说明。**运用盒结构[HEV93]的规格说明方法描述功能规格说明。遵从在第5章和第7章讨论的操作分析原则，盒结构“在每一个细化级别上使行为、数据及规程的创造性定义独立”。
- **形式化设计。**使用盒结构方法，净室设计是规格说明的自然、无缝扩展。虽然，清楚地区分两个活动是可能的，不过对规格说明（称为黑盒）进行迭代求精（在一个增量内）类似于体系结构设计和构件级设计（分别称为“状态盒”和“清晰盒”）。

830



图29-1 净室过程模型

- **正确性验证。**净室团队对设计及代码进行一系列严格的正确性验证活动。验证（29.3节和29.4节）从最高层次的盒结构（规格说明）开始，然后移向设计细节和代码。正确性验证的第一层次通过应用一组“正确性问题”[LIN88]来进行，如果这些不能证明规格说明是正确的，则使用更形式化的（数学的）验证方法。
 - **代码生成、检查和验证。**将某种专门语言表示的盒结构规格说明翻译为适当的程序设计语言。然后，使用标准的走查或检查技术（第26章）来保证代码和盒结构的语义相符性，以及代码的语法正确性。最后，对源代码进行正确性验证。

“净室软件工程通过在增量开发的软件流水线中严格分离设计过程和测试过程来达到对软件开发的统计质量控制。” ——Harlan Mills

—Harlan Mills



净室方法强调按软件被真正使用的方式进行测试。用例为统计测试策划过程提供了输入。

- **统计测试策划。**分析软件的预计使用情况，策划并设计一组测试用例，以测试使用情况的“概率分布”（29.4节）。如图29-1所示，这个净室活动是和规格说明、验证及代码生成并行进行的。
 - **统计使用测试。**回想一下，对计算机软件进行穷举测试是不可能的（第14章），因此，有必要设计有限数量的测试用例。统计使用技术[POO88]执行由统计样本（上面提到的概率分布）导出的一系列测试，这里的统计样本是从来自目标人群的所有用户对程序的所有可能执行（29.4节）中抽取的。
 - **认证。**一旦完成验证、检查和使用测试（并且所有错误被改正），则对增量进行集成前的认证工作。

与本书中讨论的其他软件过程模型一样，净室过程主要取决于高质量的分析和设计模型。

就如我们将在本章后面看到的那样，盒结构表示法仅仅是软件工程师表示需求和设计的另一

种方式。净室方法的真正特性是将形式化验证应用到工程模型中。

29.1.2 净室方法的特异之处

Dyer [DYE92]在定义过程时，间接提到了净室方法的不同之处：

净室技术结合良好定义的持续过程改进策略，第一次尝试将软件开发过程置于统计质量控制之下。为了达到这个目标，定义了一个独特的净室生命周期，它着重于以正确软件设计为目标的基于数学的软件工程，以及以软件可靠性认证为目标的基于统计的软件测试。



净室方法最重要
的特性是正确性
证明和统计使用
测试。

净室软件工程有别于传统的软件工程方法及面向对象的软件工程方法，因为：

1. 它明确使用统计质量控制。
2. 它使用基于数学的正确性证明来验证设计规格说明。
3. 它实现了一些测试技术，这些测试技术最有可能揭示具有严重影响的错误。

显然，净室方法应用了书中讲述的大多数（即使不是全部）基本的软件工程原理和概念。要产生高质量的结果，好的分析和设计过程是很重要的。但是，净室软件工程和传统软件实践的差别在于：它不再强调（有些人称取消）单元测试和调试的作用，从而大量地减少（或取消）由软件开发者所承担的测试工作量¹。

在传统软件开发中，将错误看成是很自然的事情。因为错误被认为是不可避免的，因此每个程序模块必须进行单元测试（以发现错误），之后进行调试（以消除错误）。当软件最终发布时，现场使用中还会发现更多的缺陷，则又进入另一个测试和调试的周期。与这些活动相关的重复工作既耗钱又耗时，更糟糕的是，这种重复工作可能是负面的——错误修正可能（不经意地）引入更多的错误。

“这是生活中很滑稽的事情：如果你只接受最好的东西，而拒绝接受任何其他的东西，你通常会得到最好的东西。”

——W. Somerset Maugham

在净室软件工程中，单元测试和调试被正确性验证和基于统计的测试所替代。这些活动，配合持续改进所必需的信息记录，使得净室方法具有其独特之处。

832

29.2 功能规格说明

不管选择哪种分析方法，第7章提出的操作分析原理总是适用的。对数据、功能和行为建模，并对结果模型进行分解（求精）以提供更详细的信息。总体目标是从捕获问题实质的规格说明（或模型）移向提供重要实现细节的规格说明。

通过使用盒结构规格说明方法，净室软件工程遵从操作分析的原则。一个“盒”在某个细节层次上封装了系统（或系统的某些方面）。通过逐步求精的过程，盒被精化为层次，其中

¹ 测试由独立的测试团队完成。

每个盒具有引用透明性——“每个盒规格说明的信息内容足以定义其细化信息，不需要依赖任何其他盒的实现”[LIN94]。这使得分析员能够按层次划分系统——从顶层的基本表示到底层实现的特定细节。净室软件工程中的盒有三种类型：

- **黑盒。** 黑盒刻划系统行为或系统部件的行为。通过运用由触发映射到反应的一组转换规则，系统（或部件）对特定的触发（事件）做出反应。
- **状态盒。** 状态盒以类似于对象的方式封装状态数据和服务（操作）。在这种规格说明视图中，表示出状态盒的输入（触发）和输出（反应）。状态盒也表示黑盒的“触发历史”，即封装在状态盒中的、必须在蕴含的转换间保留的数据。
- **清晰盒。** 在清晰盒中定义状态盒所蕴含的转换功能，简单地说，清晰盒包含了对状态盒的过程设计。

作为盒结构
规格说明的
一部分，细化是
如何完成的？

图29-2给出了使用盒结构规格说明的求精方法。黑盒（BB₁）定义了对完整触发集合的反应。可以将BB₁细化成一组黑盒，BB_{1.1} BB_{1.2}, ..., BB_{1.n}，每一个黑盒关系到一类行为。细化过程一直继续下去，直到可以标识出行为的内聚类（例如，BB_{1.1.1}）。然后对黑盒（BB_{1.1.1}）定义状态盒（SB_{1.1.1}）。在这种情况下，SB_{1.1.1}包含了实现BB_{1.1.1}定义的行为所需的所有数据和服务。最后，SB_{1.1.1}被细化为清晰盒（CB_{1.1.1.1} CB_{1.1.1.2}, ..., CB_{1.1.1.n}），并且刻划出过程的设计细节。

KEY POINT
盒结构求精和
正确性验证同
时进行。
833

当进行每一步细化时，也同时进行正确性验证。需要对状态盒规格说明进行验证，以保证每个规格说明均同其父黑盒规格说明定义的行为相一致。类似地，对照父状态盒，对清晰盒规格说明进行验证。

应该注意，基于诸如OCL或Z语言（第28章）的规格说明方法可以与盒结构规格说明方法联合使用，唯一的要求是每一层次的规格说明可以做形式化验证。

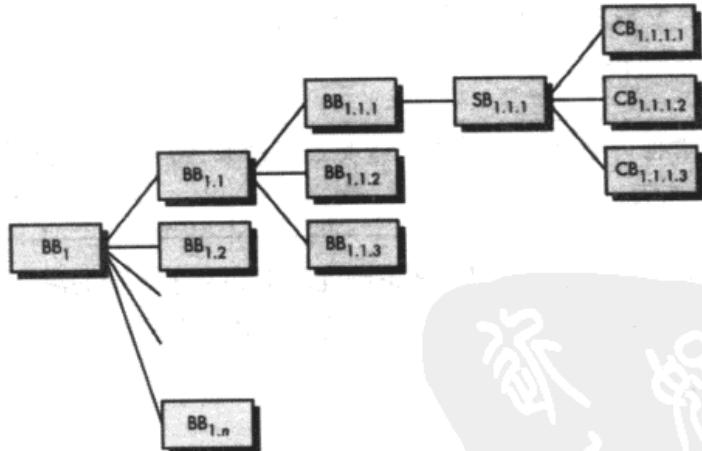


图29-2 盒结构求精

29.2.1 黑盒规格说明

使用图29-3所示的符号[MIL88]，黑盒规格说明描述了一种抽象、触发和反应。函数f被应用到输入（触发）S的序列S*，并将它们变换为输出（反应）R。

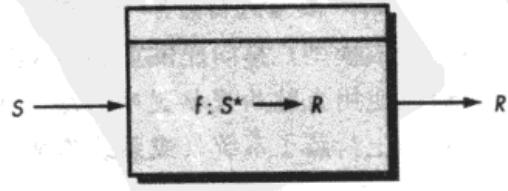


图29-3 黑盒规格说明

对于简单的软件构件， f 可以是一个数学函数，但一般情况下，使用自然语言（或形式化规格说明语言）描述 f 。

为面向对象系统引入的很多概念也适用于黑盒。黑盒封装了数据抽象和操纵抽象数据的操作。和类层次一样，黑盒规格说明可以展示使用的层次，其中，低层盒从树结构中的高层盒继承属性。

29.2.2 状态盒规格说明

状态盒是“状态机的一种简单泛化”[MIL88]。回想第8章对行为建模和状态图的讨论，状态是某个可观察到的系统行为的模型。当进行处理时，系统对事件（触发）做出反应，从当前状态转换到某一新的状态。当进行转换时，可能发生某个动作。状态盒使用数据抽象来确定到下一个状态的转换以及状态转换后将要发生的行为（反应）。

如图29-4所示，状态盒可以与黑盒结合使用。来自某外部源及一组内部系统状态 T 的触发 S 被输入到黑盒中。Mills[MIL88]给出了包含在状态盒内的黑盒的函数 g 的数学描述：

$$g : S^* \times T^* \rightarrow R \times T$$

这里 g 是和特定状态 t 连接的子函数。当整体地考虑时，状态—子函数对—— (t, g) ——定义了黑盒函数 f 。

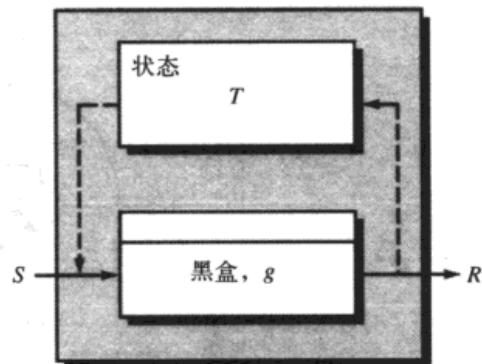


图29-4 状态盒规格说明

29.2.3 清晰盒规格说明

清晰盒规格说明是与过程设计及结构化编程紧密关联的（第11章）。实质上，状态盒中的子函数 g 被实现 g 的结构化编程结构所替代。

例如，考虑图29-5所示的清晰盒。在图29-4中的黑盒 g 被带有条件的顺序结构所替代。随着逐步求精的进行，这些结构又可以依次被细化为更低层次的清晰盒。

值得注意的是：在清晰盒层次中所描述的过程性规格说明可被证明是正确的。这一主题将在下一节中讨论。

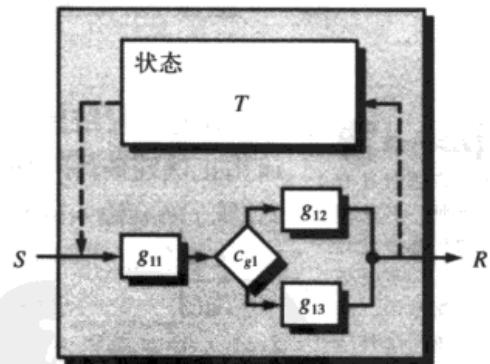


图29-5 清晰盒规格说明

29.3 净室设计

净室软件工程中使用的设计方法主要运用结构化程序设计的原理。但是，在这里结构化程序设计被应用得更严格。

对基本的处理函数（在规格说明的早期求精中描述）进行求精，其方法是“将数学函数逐步扩展为逻辑连接词（如，if-then-else）和子函数构成的结构，这种扩展一直进行下去，直到所有标识出来的子函数可以用程序设计语言直接表达实现”[DYE92]。

使用结构化程序设计方法对函数进行求精很有效，但是，对数据设计如何呢？这里，可

以使用一组基本的设计概念（第5章及第9章），程序数据被封装为由子函数提供服务的一组抽象。使用数据封装、信息隐蔽和数据类型概念进行数据设计。

29.3.1 设计求精与验证

使用什么条件证明结构化结构的正确性？

836

每个清晰盒规格说明代表了一个完成状态盒转换所需的过程（子函数）的设计。对清晰盒规格说明，如图29-6所示，使用结构化程序设计结构和逐步求精。一个程序函数 f 被细化为子函数 g 和 h 的序列，这两个子函数又被进一步细化为条件结构（if-then-else和do-while）。进一步的求精给出了连续的逻辑细化。

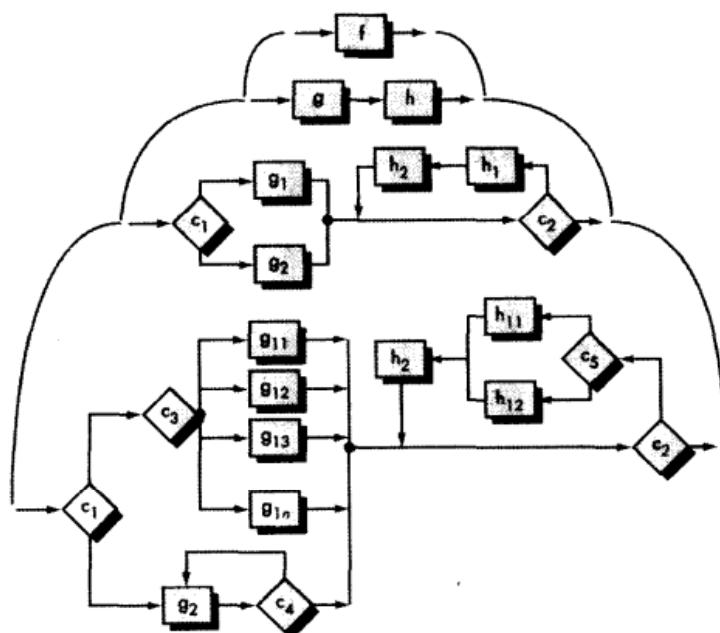


图29-6 逐步求精

ADVICE
如果在进行过程设计时，只使用结构化程序设计结构，正确性证明是简单的。如果你违反这种结构，正确性证明将是困难的，甚至是不可能的。

在每个求精层次，净室团队²执行一次形式化正确性验证。为此，将一组通用正确性条件附加到结构化程序设计结构上。如果函数 f 被扩展为序列 g 和 h ，则 f 所有输入的正确性条件是：

- 执行 g 之后再执行 h 能完成 f 的功能吗？

如果一个函数 p 被细化为形为 $\text{if } <c> \text{ then } q \text{ else } r$ 的条件形式，则对 p 的所有输入的正确性条件是：

- 只要条件 $<c>$ 为真， q 能完成 p 的功能吗？只要条件 $<c>$ 为假， r 能完成 p 的功能吗？

如果一个函数 m 被细化为循环 $\text{do } n \text{ while } <c>$ （原文中无此表达式，译者为方便读者理解特意补充了此表达式。——译者注），则对 m 的所有输入的正确性条件是：

- 能够保证循环终止吗？
- 只要 $<c>$ 为真，循环执行 n 之后能完成 m 的功能吗？只要 $<c>$ 为假，退出循环仍能完成 m 的功能吗？

² 由于整个团队都参与验证过程，实施验证本身产生错误的可能性很小。

每次当一个清晰盒被精化为下一个详细层次时，都应用上面给出的正确性条件。

值得注意的是：结构化程序设计结构的使用限制了必须进行的正确性测试的数量。对顺序结构只检查单个条件，对if-then-else结构只测试两个条件，对循环则只验证三个条件。[837]

为了举例说明过程设计的正确性验证，我们使用由Linger、Mills和Witt[LIN79]给出的一个简单的例子。目的是设计并验证一个小的程序，该程序对某给定的整数 x ，找出其平方根的整数部分 y 。过程设计用图29-7给出的流程图来表示。

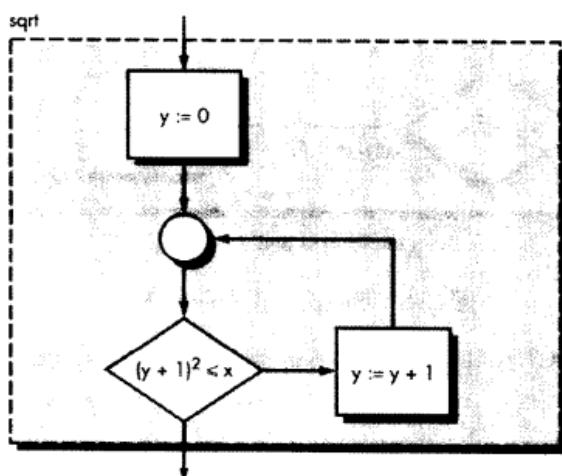


图29-7 计算平方根的整数部分[LIN79]

POINT
为了证明设计正确性，你必须首先标识所有条件，然后证明每个条件取合适的布尔值。将这些证明称为子证明。

为了验证该设计的正确性，我们必须定义如图29-8所示的入口和出口条件。入口条件说明 x 必须大于或等于0；出口条件要求 x 保持不变， y 满足图中描述的表达式。为了证明设计的正确性，需要证明图29-8表示的条件*init*、*loop*、*cont*、*yes*和*exit*在所有情形下都是正确的。有时将这些证明称为子证明。

1. 条件*init*要求 $[x \geq 0 \text{ and } y = 0]$ 。基于问题的需求，假定入口条件是正确的³。因此，*init*条件的第一部分 $x \geq 0$ 是满足的。在流程图中，在*init*条件前的语句设置 $y = 0$ ，因此，*init*条件的第二部分也是满足的，因此，*init*为真。
2. 条件*loop*可能以两种方式之一出现：(1) 直接从*init*（此时*loop*条件被直接满足），或者，(2) 通过穿过条件*cont*的控制流。因为条件*cont*与条件*loop*相同，因此，不管从哪条路径到达它，条件*loop*都为真。

3. 只有在 y 值被递增1后，才能遇到条件*cont*。另外，只有在条件*yes*也为真时，才能调用到达条件*cont*的控制流路径。因此，如果 $(y + 1)^2 \leq x$ ，则 $y^2 \leq x$ ，条件*cont*成立。

4. 条件*yes*在如图所示的条件逻辑中被测试，因此，当控制流沿着所示的路径移动时，条件*yes*一定为真。

5. 条件*exit*首先要求 x 保持不变，对设计进行检查可发现： x 没有出现在赋值操作的左边，没有使用 x 的函数调用，因此， x 保持不变。因为条件测试 $(y + 1)^2 \leq x$ 不成立时，才可能到达条件*exit*，因此 $(y + 1)^2 > x$ 成立。此外，*loop*条件还必须保持为真（即， $y^2 \leq x$ ），因此， $(y + 1)^2 > x$ 和 $y^2 \leq x$ 可以组合在一起满足*exit*条件。

³ 在这里，负数的平方根没有意义。

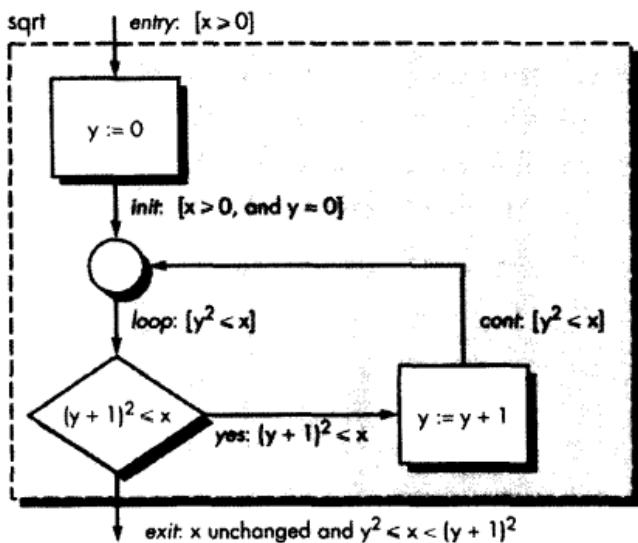


图29-8 证明设计正确性[LIN79]

我们必须进一步保证循环终止。对循环条件的检查显示：因为 y 是递增的，而 $x \geq 0$ ，因此，最后循环一定终止。

上面的5步是图29-7中算法设计的正确性证明，我们现在可以肯定，这个设计将计算出平方根的整数部分。

还有针对设计验证的更严格的数学方法，然而，这个话题的讨论超出了本书范围，有兴趣的读者可参考[LIN79]。

29.3.2 设计验证的优点⁴

对清晰盒设计的每一步求精进行严格正确性验证有许多显著的优点，Linger [LIN94]这样描述这些优点：

通过正确性证明，我们能够获得什么？

- 它将验证简化为一个有限的过程。在清晰盒中，以嵌套的、顺序的方式组织控制结构，这就自然地定义了一个层次，该层次显示了必须被验证的正确性条件。在子证明的层次结构中，“替代公理”[LIN79]允许我们将指定的函数用其控制结构的细化来替换。例如，在图29-9中的指定函数f1的子证明需要证明：操作g1、g2与指定函数f2的组合对数据的操作效果与f1相同。注意，f2代替了在证明中对它求精的所有细节。这一替代的证明参数对当前的控制结构而言是局部的。事实上，软件工程师可以以任意顺序执行证明。
- 再怎么强调将验证简化为有限过程对质量产生的正面效果都不过分。除了那些最微不足道的程序，即使所有程序都具有无限数目的可执行路径，也可以在有限步骤内对它们进行验证。
- 它使得净室团队验证设计和代码的每一行。在正确性定理的基础上，团队可以通过小组分析和讨论来执行验证，并且对于一些生命关键或任务关键的系统，有时需要向用户提供额外的信心，此时可用该技术生成书面的证明。

840

POINT
尽管程序中可执行路径的数量很大，证明程序正确性的步骤却是非常小的。

⁴ 本节及图29-7至图29-9摘自[LIN94]，并获准使用。

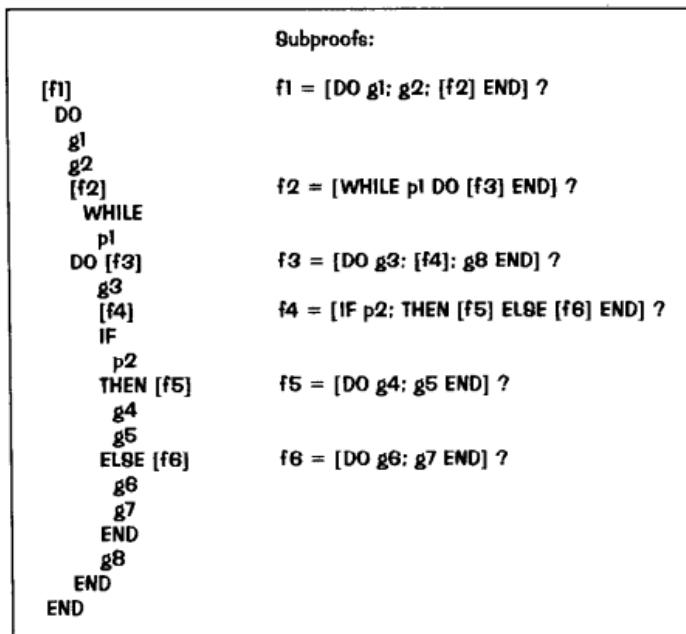


图29-9 带有子证明的设计[LIN94]

- 它达到几乎零缺陷的水平。在团队的评审过程中，每个控制结构的每个正确性条件被依次验证。每个团队成员必须就每个条件都是正确的达成共识，这样只有团队的每个成员均未能正确地验证某条件时，才有可能出现错误。基于个体验证取得无异议的全体认同，这一要求使得生产的软件在其第一次运行前几乎没有或根本没有缺陷。
- 它具有可伸缩性。每个软件系统，不管有多大，均具有顶层的由顺序、选择和循环结构构成的清晰盒过程。通常每个过程均调用一个有数千行代码的大的子系统，并且每个子系统也具有自己的顶层的指定函数和过程。所以，这些高层控制结构的正确性条件的验证方法与低层结构的相同。高层的验证可能需要更多的时间（这是值得的），但它不需要更多的理论。
- 它产生出比采用单元测试更好的代码。单元测试仅仅检查从很多可能的路径中选出的测试路径的执行效果。基于函数验证的理论，净室方法可以验证所有数据的每个可能的结果，因为虽然一个程序可能有很多可执行路径，但它只有一个函数。验证也比单元测试更有效，可以在几分钟之内检查大多数验证条件，但单元测试要花费大量时间去准备、执行和检查。

值得注意的是，设计验证最终必须应用到源代码本身，此时，通常将其称为正确性验证。

29.4 净室测试

净室测试的策略从根本上不同于传统测试方法。传统测试方法导出一组测试用例以发现设计和编码错误；净室测试的目的是通过证明用例（第7章）的统计样本的成功运行来确认软件需求。

“质量不是条例，而是习惯的成果。”¹

—Aristotle

29.4.1 统计使用测试

计算机程序的用户没有必要去了解设计的技术细节。程序用户可见的行为通常是由用户产生的输入和事件所驱动的。但是，在复杂系统中，输入和事件的范围（即用例）是非常广泛的。什么样的用例子集能够充分验证程序的行为？这是统计使用测试关注的第一个问题。

统计使用测试“等同于以用户试图使用软件的方式来测试软件”[LIN94]。为了完成测试工作，净室测试团队（也称为认证团队）必须确定软件的使用概率分布。对软件的每个增量的规格说明（黑盒）进行分析，并定义一组使软件改变其行为的触发（输入或事件）。通过与潜在用户的交流，使用场景的建立，及对应用领域的全面了解，为每个触发分配一个使用概率。

按照使用概率分布为每个触发集合生成测试用例⁵。为了举例说明，考虑本书前面讨论过的SafeHome系统。在此系统中，使用净室软件工程方法开发一个软件增量来管理用户与安全系统键盘的交互。对这个增量，已经标识出5个触发。通过分析，给出每个触发的概率分布百分数。为了更容易地选择测试用例，这些概率被映射到1~99的数字区间[LIN94]，如下表所示。

程序触发	概率	区间
arm/disarm (AD)	50%	1~49
zone set (ZS)	15%	50~63
query (Q)	15%	64~78
test (T)	15%	79~94
panic alarm	5%	95~99

为了生成符合使用概率分布的使用测试用例序列，生成1~99之间的随机数。每个随机数与前面概率分布的区间相对应。因此，使用测试用例序列可以随机定义，但又与触发发生的适当概率相对应。例如，假定生成了下面的随机数序列：

13-94-22-24-45-56

81-19-31-69-45-9

38-21-52-84-86-4

842

根据表中显示的分布区间选择适当的触发，从而导出下面的用例：

AD-T-AD-AD-AD-ZS

T-AD-AD-AD-Q-AD-AD

AD-AD-ZS-T-T-AD

测试团队执行这些测试用例，并对照系统的规格说明来验证软件的行为。记录测试所用的时间，以便可以确定间隔时间。利用间隔时间，认证团队可以计算出平均失效时间MTTF。如果运行很长的测试序列没有出现故障，则MTTF较低，软件可靠性较高。

29.4.2 认证

本章前面讨论的验证和测试技术可用来对软件构件（和完整的增量）进行认证。在净室软件工程方法中，认证意味着可以描述每个构件的可靠性（用平均失效时间MTTF来度量）。

可认证的软件构件的潜在影响远远超出了单个净室项目的范围，可复用的软件构件可以

⁵ 可使用自动化工具完成此项工作。进一步的信息请参见[DYE92]。

和它们的使用场景、程序触发以及概率分布一起存储。在描述的使用场景和测试体系下，每个构件都具有一个经过认证的可靠性。对于那些希望使用这些构件的人来说，这些信息是十分宝贵的。

认证方法包括5个步骤[WOH94]：

如何认证一个软件构件。

1. 必须创建使用场景。
2. 说明使用概貌。
3. 从使用概貌中生成测试用例。
4. 执行测试，记录并分析失效数据。
5. 计算并认证可靠性。

步骤1到步骤4已在前面几节中讨论过。在本节中，我们集中讨论可靠性认证。

净室软件工程的认证需要创建三个模型[POO93]：

- **取样模型。**软件测试执行 m 个随机测试用例，如果没有错误发生或只有少于指定数量的错误发生，则通过认证。用数学方法导出 m 值，以保证能够获得需要的可靠性。
- **构件模型。**对由 n 个构件组成的系统进行认证，构件模型使得分析员能够确定构件 i 在完成前失效的概率。
- **认证模型。**设计并认证系统的整体可靠性。

在统计使用测试完成后，认证团队已得到了交付软件所需要的信息，包括使用上面的每个模型计算出来的经过认证的MTTF。843

关于取样的计算以及构件和认证模型的详细讨论已超出本书的范围，有兴趣的读者参阅文献[MUS87]、[CUR86]和[POO93]可获得更多的详细信息。

29.5 小结

净室软件工程是软件开发的一种形式化方法，它可以生成高质量的软件。它使用盒结构规格说明（或形式化方法）进行分析和设计建模，并且强调将正确性验证而不是测试作为发现和消除错误的主要机制。运用统计使用测试来获取失效率信息，此信息对认证所交付的软件的可靠性是必需的。

净室方法从使用盒结构表示的分析和设计模型入手，一个“盒”在某特定的抽象层次上封装系统（或系统的某些方面）。黑盒用于表示从外部可以观察到的系统行为。状态盒封装状态数据和操作，清晰盒用于对状态盒中的数据和操作所蕴含的过程设计建模。

一旦完成了盒结构设计，就可以应用正确性验证。软件构件的过程设计被划分为一系列子函数，为了证明每个子函数的正确性，要为每个子函数定义出口条件，并应用一组子证明。如果每个出口条件均成立，则设计一定是正确的。

一旦完成了正确性验证，便开始进行统计使用测试。和传统测试不同，净室软件工程并不强调单元或集成测试，而是通过定义一组使用场景、确定对每个场景的使用概率、然后定义符合概率的随机测试来进行软件测试。产生的错误记录与取样模型、构件模型和认证模型相结合，使得可以数学地计算软件构件的预计可靠性。

净室原理是严格的软件工程方法。它是一种软件过程模型，强调正确性的数学验证及软件可靠性的认证。其结果是非常低的失效率，这是使用非形式化方法难于或不可能达到的。

参考文献

- [844] [CUR86] Curritt, P. A., M. Dyer, and H. D. Mills, "Certifying the Reliability of Software," *IEEE Trans. Software Engineering*, vol. SE-12, no. 1, January 1994.
- [DYE92] Dyer, M., *The Cleanroom Approach to Quality Software Development*, Wiley, 1992.
- [HAU94] Hausler, P. A., R. Linger, and C. Trammel, "Adopting Cleanroom Software Engineering with a Phased Approach," *IBM Systems Journal*, vol. 33, no. 1, January 1994, pp. 89-109.
- [HEN95] Henderson, J., "Why Isn't Cleanroom the Universal Software Development Methodology?" *Crosstalk*, vol. 8, No. 5, May 1995, pp. 11-14.
- [HEV93] Hevner, A. R., and H. D. Mills, "Box Structure Methods for System Development with Objects," *IBM Systems Journal*, vol. 31, no. 2, February 1993, pp. 232-251.
- [LIN79] Linger, R. M., H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, 1979.
- [LIN88] Linger, R. M., and H. D. Mills, "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility," *Proc. COMPSAC '88*, Chicago, October 1988.
- [LIN94] Linger, R., "Cleanroom Process Model," *IEEE Software*, vol. 11, no. 2, March 1994, pp. 50-58.
- [MIL87] Mills, H. D., M. Dyer, and R. Linger, "Cleanroom Software Engineering," *IEEE Software*, vol. 4, no. 5, September 1987, pp. 19-24.
- [MIL88] Mills, H. D., "Stepwise Refinement and Verification in Box Structured Systems," *Computer*, vol. 21, no. 6, June 1988, pp. 23-35.
- [MUS87] Musa, J. D., A. Iannino, and K. Okumoto, *Engineering and Managing Software with Reliability Measures*, McGraw-Hill, 1987.
- [POO88] Poore, J. H., and H. D. Mills, "Bringing Software Under Statistical Quality Control," *Quality Progress*, November 1988, pp. 52-55.
- [POO93] Poore, J. H., H. D. Mills, and D. Mutchler, "Planning and Certifying Software System Reliability," *IEEE Software*, vol. 10, no. 1, January 1993, pp. 88-99.
- [WOH94] Wohlin, C., and P. Runeson, "Certification of Software Components," *IEEE Trans. Software Engineering*, vol. SE-20, no. 6, June 1994, pp. 494-499.

习题与思考题

29.1 一个冒泡排序算法定义如下：

```
procedure bubblesort;
var i, t, integer;
begin
repeat until t=a[1]
    t:=a[1];
    for j:= 2 to n do
        if a[j-1] > a[j] then begin
            t:=a[j-1];
            a[j-1]:=a[j];
            a[j]:=t;
        end
    endrep
end
```

将该设计划分为子函数，并定义一组条件，使得你能够证明该算法是正确的。

- 29.2 对思考题8.10中介绍的PHTRS系统的一部分开发盒结构规格说明。
- 29.3 如果请你必须选出净室软件工程和传统的或面向对象的方法根本不同的一个方面，那么这个方面是什么？
- 29.4 增量过程模型和认证工作如何共同作用生产出高质量软件？
- [845] 29.5 使用盒结构规格说明，开发SafeHome系统的“第一遍”分析和设计模型？
- 29.6 用你自己的话，描述净室软件工程中认证的意图。

- 29.7 选择你在其他情形下设计的（或导师布置的）某程序构件，并给出完整的正确性证明。
- 29.8 选择你经常使用的一个程序（例如，e-mail处理器、字处理器、电子表格程序），为此程序创建一组使用场景，定义对每个场景的使用概率，然后开发出类似29.4.1节中的程序触发和概率分布表。
- 29.9 对思考题29.8中开发的程序触发和概率分布表，使用一个随机数生成器开发一组用于统计使用测试的测试用例。
- 29.10 为思考题29.5中讨论的冒泡排序的正确性证明编写文档。

推荐读物与阅读信息

Powell等（《Cleanroom Software Engineering: Technology and Process》，Addison-Wesley，1999）提供了对净室方法所有重要方面的深入探讨。Poore和Trammell（《Cleanroom Software Engineering: A Reader》，Blackwell Publishing，1996）编辑出版的书对净室主题进行了有用讨论。Becker和Whittaker（《Cleanroom Software Engineering Practices》，Idea Group Publishing,1996）为那些不熟悉净室方法的人们提供了非常好的概述。

《Cleanroom Pamphlet》（软件技术支持中心，Hill AF Base, April 1995）包含了许多重要文章的重印。Linger[LIN94]是对该主题的较好的导论之一。“软件数据与分析中心”（The Data and Analysis Center for Software, DACS）（www.dacs.dtic.mil）提供了净室软件工程方面很多有用的论文、指导书及其他信息源。

Linger和Trammell（《Cleanroom Software Engineering Reference Model》，SEI Technical Report CMU/SEI-96-TR-022，1996）定义了14个净室过程和20个工作产品，形成了关于净室软件工程的SEI CMM的基础（CMU/SEI-96-TR-023）。

净室软件工程（www.cleansoft.com）的Michael Deck准备了净室主题方面的参考书目，许多是可下载的。

通过正确性证明进行的设计验证是净室方法的核心。Stavely（《Toward Zero-Defect Software》，Addison-Wesley，1998）、Baber（《Error-Free Software》，Wiley，1991）及Schulmeyer（《Zero Defect Software》，McGraw-Hill，1990）的著作详细地讨论了正确性证明。

净室软件工程方面的多种信息资源可在Internet上获得。最新的净室软件工程的WWW参考文献列表可在SEPA Web站点<http://www.mhhe.com/pressman>找到。

846

第30章 基于构件的开发

要点浏览

概念：你购买了一个娱乐系统回家，此系统中每一个构件的设计都符合特定的音频—视频体系结构——标准化的连接及预先建立的通信协议。组装很容易，因为你不必由数百个离散的零件构造系统。基于构件的软件工程（component-based software engineering, CBSE）试图达到同样的效果。一组预先建造好的、标准化的软件构件可用于满足针对某应用领域的特定体系结构风格。应用系统由这些构件组装而成，而不是使用传统程序设计语言的离散零件进行构造。

人员：软件工程师应用CBSE过程。

重要性：组装一个家庭娱乐系统只需要几分钟，因为构件被设计成非常易于集成的形式。虽然软件的情况比较复杂，但是，基于构件的系统易于组装，因此其成本将低于由离散的零件构造系统。此外，CBSE鼓励可预知的体系结构模式和标准的软件基础设施的应用，因此可以得到高质量的软件。

步骤：CBSE包含两个并行的工程活动：领域工程和基于构件的开发。领域工程探索一个应用领域，试图去发现适合复用的可候选的功能、行为和数据构件。这些构件被放置在复用库中。基于构件的开发从客户处引出需求；选择合适的体系结构风格满足待建系统的目标；然后，（1）选择潜在的可复用构件，（2）认定这些构件确实适合于系统的体系结构，（3）为了更好地集成，对构件进行适应性修改，（4）集成构件以形成子系统及应用系统整体。此外，开发定制的构件以满足系统中不能被现有构件实现的方面。

工作产品：使用现有的软件构件和新开发的软件构件组装而得到的可运行软件是CBSE的结果。

质量保证措施：使用在每个软件工程过程中都相同的SQA实践——正式技术评审——评估分析和设计模型，通过专门评审考虑与所获得的构件相关的问题，通过使用测试发现新开发的软件和被集成到体系结构中的可复用构件中的错误。

关键概念
适应
CBD
CBSE
过程
经济学
分类
构件类型
领域工程
中间件
合格性认证
复用环境
结构点

在软件工程的范畴内，复用既是旧概念，也是新概念。从最早的计算时代开始，程序员就开始复用概念、抽象及过程了，但是早期的复用方法是很特别的。今天，复杂的、高质量的基于计算机的系统必须在非常短的时间内建成，这要求更有组织的复用方法。

基于构件的软件工程（CBSE）强调使用可复用的软件“构件”来设计和构造基于计算机的系统。Clements[CLE95]对CBSE给出了如下描述：

[CBSE]正在改变大型软件系统的开发方式。[CBSE]体现了Frod Brooks和其他人支持的“购买，而非构造”的思想。就如同早期的子程序将程序员从考虑编程细节中解脱出来一样，[CBSE]将考虑的

重点从编码转移到组装软件系统。考虑的焦点是“集成”，而不再是“实现”。这样做的基础是假定在很多大型软件系统中存在足够多的共性，使得开发可复用的构件来满足这些共性是可行的。

但是又出现了很多问题。通过组装一组可复用的软件构件来构造复杂的系统可能吗？这项工作能否以成本及时间合算的方式完成？能否建立适当的激励机制鼓励软件工程师复用而不是重复开发？在管理上需要对开发可复用软件构件增加费用吗？能否以需用者易于访问的方式建造复用所必需的构件库？构件库中存在的构件能否被需用者找到？

847

即使是今天，软件工程师仍在设法解决软件构件复用方面的形形色色的问题。在本章中，我们会看到某些答案。

848

30.1 基于构件系统的工程

WebRef

有关 WebApp 的 CBSE 的有用信息可在 www.cbdhq.com 找到。

从表面上看，CBSE似乎类似于传统软件工程或面向对象的软件工程。当软件团队使用传统的需求导出技术（第7章）建立了待开发软件的系统需求时，该过程开始。体系结构设计（第7章）完成后，并不立即进行详细设计任务，软件团队检查需求以确定系统的哪些子集可以通过直接组装而不是构造完成。也就是说，团队针对每一系统需求提出如下问题：

- 现有商品化构件（COTS）是否能够实现该需求？
- 内部开发的可复用构件是否能够实现该需求？
- 可用构件的接口与待构造系统的体系结构是否相容？

848

团队可以试图修改或去除那些不能用COTS或自有构件实现的系统需求¹。如果不能修改或删除这些需求，则必须应用软件工程方法构造满足这些需求的新构件。但是，对于那些可以由已有构件实现的需求，采用不同的软件工程活动：合格性检验，适应性修改，组装及更新。这些CBSE活动将在30.4节进行更详细的讨论。

在本节的前面部分反复使用术语构件，但并没有给出这个术语的确切描述。Brown和Wallnau[BRO96]给出了如下几种描述：

- 构件——一个系统中有价值的、几乎独立的、并可替换的部分，它在很好定义的体系结构中完成某一确定的功能。
- 运行时软件构件——由一个或多个程序构成的动态绑定包，其中的程序可作为单元进行管理，并可通过在运行时发现的文档化接口来访问。
- 软件构件——仅具有协议规定且明显依赖环境的组装单元。
- 业务构件——某一“自主的”业务概念或业务过程的软件实现。

除了这些描述，还可以基于软件构件在CBSE过程中的使用来描述它们。除了COTS构件，CBSE过程还生成：

- 合格的构件——由软件工程师评估，以确保不仅功能而且性能、可靠性、可用性和其他质量因素（第26章）均符合待构造的系统或产品的需求。
- 适应的构件——对不想要的或不希望的特征进行适应性修改（也称掩盖或包装）

¹ 这意味着软件开发组织调整其业务或产品需求，以便不必定制开发构件即可完成基于构件的实现。这个方法可降低成本，并缩短投入市场的时间，但并非总是可能的。

[BRO96]。

- 组装的构件——被集成到体系结构风格中，并与合适的能够有效地协同和管理构件的基础设施互联。
- 更新的构件——当新版本的构件可用时，替换现有的构件。

849

30.2 CBSE过程

CBSE过程是这样描述的：不仅标识候选的构件，而且对每个构件接口进行合格性检验，适当修改构件以消除体系结构中的不匹配，组装构件到选择的体系结构风格中，以及当系统需求变化时更新构件[BRO96]。基于构件的软件工程的过程模型强调领域工程（30.3节）与基于构件的开发并行进行。

图30-1给出了典型的过程模型，它显然适应CBSE[CHR95]。领域工程创建应用领域的模型，该模型是软件工程中分析用户需求的基础。一般的软件体系结构为应用系统的设计提供了输入。最后，可复用的软件构件可以购买，可从现有库中选择，也可以构造（作为领域工程的一部分），然后，软件工程师就可以在基于构件开发的过程中使用这些构件。

作为基于构件开发（图30-1）组成部分的分析与体系结构设计可在抽象设计范型（abstract design paradigm, ADP）的环境中完成[DOG03]。ADP的意思是软件的总体模型——表示为数据、功能及行为（控制）——可以分解为层次结构。在分解开始时，系统可表示为体系结构框架的集合，每一个框架都由一个或多个设计模式（第10章）组成。进一步的细化标识用于创建每一个设计模式所需要的构件。在理想的情况下，所有这些组件都将从存储库（构件合格性检验、适应性修改及组装活动会应用该库）中获得。当需要特定的构件时，应用构件工程。

850

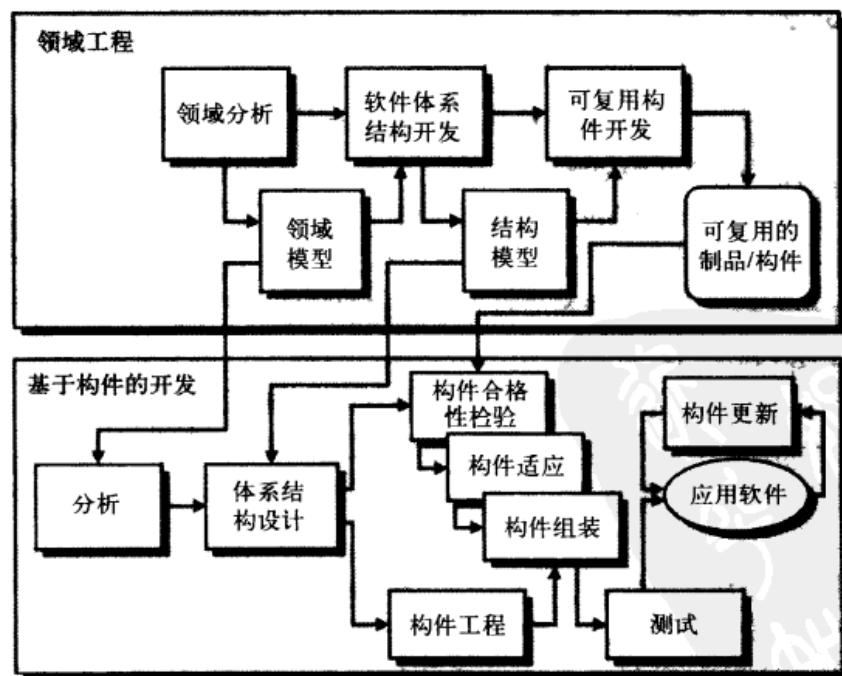


图30-1 支持CBSE的过程模型

30.3 领域工程

领域工程的目的是标识、构造、分类和传播一些软件构件，这些构件将适用于某特定应用领域中现有的和未来的软件。其总体目标是建立相应的机制，使得软件工程师在开发新系统或改造现有系统时可以共享这些构件——复用它们。领域工程包括三个主要活动——分析、构造和传播。

“领域工程是发现系统间的共性以识别可以应用于很多系统的构件，并识别最能充分利用那些构件的程序家族。”
——Paul Clements

有人争论说：“复用将消失，不是被消除而是被集成”到软件工程实践中[TRA95]。随着更多地强调复用，人们相信在未来十年中领域工程将变得和软件工程一样重要。

30.3.1 领域分析过程



我们本节中讨论的分析过程集中
在可复用的构
件。然而，完全
的COTS系统
(如，电子商务
应用，自动售货
机应用)分析也
可以是领域分析
的一部分。

通常在面向对象软件工程的范畴内描述领域分析方法。领域分析过程中的步骤定义如下：

1. 定义待研究的领域。
2. 对从领域中抽取出的项进行分类。
3. 收集领域中有代表性的应用样本。
4. 分析样本中的每个应用，并定义分析类。
5. 开发类的分析模型。

值得注意的是，领域分析适用于任何软件工程范型，因此，可以应用到传统软件开发及面向对象软件开发中。

虽然上面的步骤提供了领域分析的有用模型，但对决定哪些是候选的可复用软件构件并没有指导作用。Hutchinson和Hindley[HUT88]提出了下面一组实际问题来指导识别可复用的软件构件：

在领域分析
过程中标识
的那些构件将成
为候选的可复用
构件？

- 在将来的实现中是否需要构件的功能？
- 构件功能在领域中的共性怎么样？
- 构件是否有硬件依赖性？如果具有依赖性，硬件在不同实现之间保持不变吗？或者，硬件的细节可以移到另一个构件中吗？
- 设计是否为以后的实现进行了足够的优化？
- 能否将一个不可复用的构件参数化使其变成可复用的？
- 是否可以仅对构件进行少量修改，就能在很多实现中复用？
- 通过修改进行复用是可行的吗？
- 不可复用的构件是否能够通过分解产生一组可复用的构件？
- 针对复用的构件分解有效吗？

851

有关领域分析的更多信息请参见[ATK01]、[HEI01]及[PRI93]。

30.3.2 特征化函数

确定一个潜在可复用的构件在特定的情况下是否真正适用，有时是很困难的。为了能够

做这样的决定，有必要定义一组能够被领域中所有软件共享的领域特征。领域特征定义了该领域中所有产品所具有的某些一般属性。例如，一般特征可能包括：安全性或者说可靠性的最重要性、程序设计语言、处理中的并发性及很多其他内容。

一个可复用构件的领域特征集合可以表示为 $\{D_{pi}\}$ ，其中，集合中的每一项 D_{pi} 表示某一特定的领域特征。赋给 D_{pi} 的值表示一个等级，它指明了该特征对构件p的相关性。典型的等级[BAS94]可能是：

1. 与复用是否合适无关。
2. 仅仅在不寻常的情况下相关。
3. 相关——对构件进行修改使其可以被复用，而不管其间的差别。
4. 明显相关，如果新软件不具有这个特征，复用将是无效的，但仍然是可能的。
5. 明显相关，如果新软件不具有这个特征，复用将是无效的，并且不推荐不具有此特征的复用。

WebRef

领域分析方面的有用信息可在www.sei.cmu.edu/str/deda.html找到。

852

当在某应用领域中构造新软件w时，为它导出领域特征集合。然后，在 D_{pi} 及 D_{wi} 之间进行比较，以确定现有的构件p是否能够在应用w中被有效地复用。

即使被开发的软件很明显地属于某一应用领域，也必须对该领域的可复用构件进行分析以确定它们的可用性。在某些情况下（希望是有限的情况），“重新开发”可能仍然是成本最合算的选择。

30.3.3 结构化建模与结构点

当应用领域分析时，分析师在某一领域的应用中寻找重复模式。结构建模是一种基于模式的领域工程方法，应用该方法的前提条件是：每个应用领域都存在具有可复用潜力的重复模式（包括功能的、数据的和行为的）。

每个应用领域可用一个结构模型来描述（例如，飞机的电子设备系统在细节上差别很大，但此领域中的所有现代软件都具有相同的结构模型），因此，结构模型是一种体系结构风格（第10章），它能够也应该在领域的不同应用中被复用。

McMahon[MCM95]将结构点描述为“结构模型中的一个独特的构造”。结构点有三个独特的特征：

什么是结构点？其特征是什么？

1. 一个结构点是一个抽象，它应该具有有限数量的实例。此外，抽象应该在领域的不同应用中重现。否则，用于验证、文档化及传播结构点的成本就不能说是合算的。

2. 支配结构点使用的规则应该容易理解。此外，结构点的接口应该比较简单。

3. 结构点通过封锁所有包含在结构点内部的复杂性而实现信息隐藏，这样减少整个系统感觉上的复杂性。

KEY POINT
结构点类似于设计模式，它可以在某一特定领域的应用中被重复发现。

作为将结构点视为系统体系结构模式的一个例子，考虑报警系统软件领域。该领域可能包含如SafeHome（在前面几章讨论过）这样简单的系统，或如工业过程警报这样复杂的系统。然而，在每种情况下，都会遇到一组可以预见的结构模式：接口，能够使用户与系统交互；范围设置机制，允许用户

设置将测量的参数的范围，传感器管理机制，与所有的监控传感器通信；响应机制，对传感器管理系统提供的输入作出响应；控制机制，使用户能够控制实施监控的方式。这些结构点都被集成到领域体系结构中。

可以定义贯穿许多不同应用领域的一般性结构点[STA94]：

- **应用前端**——包括所有的菜单、面板、输入和命令编辑设施的GUI(图形用户界面)。
- **数据库**——与应用领域相关的所有对象的中心存储库。
- **计算引擎**——操作数据的数值和非数值模型。
- **报告设施**——产生所有种类的输出的功能。
- **应用编辑器**——根据特定用户的需要定制应用的机制。

已建议使用结构点来替代软件成本估算的代码行和功能点[MCM95]，使用结构点进行成本估算的概要讨论在30.6.2节中给出。

30.4 基于构件的开发

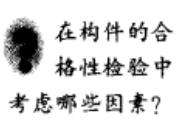
基于构件的开发（CBD）是一个与领域活动并行的CBSE活动。使用在本书前面讨论的分析与体系结构设计方法，软件团队针对为应用系统创建的分析模型的体系结构风格进行细化²。

一旦建立了体系结构，就必须向其中增加构件，这些构件可从复用库中获得，或者根据专门需要而开发。因此，基于构件开发的任务流有两条路径（图30-1）。当可复用构件有可能被集成到体系结构中时，必须对它们进行合格性检验和适应性修改。当需要新的构件时，必须重新开发。产生的新构件被“组装”（集成）到体系结构模板中，并进行全面测试。

30.4.1 构件合格性检验、适应性修改与组装

如我们已经看到的那样，领域工程提供了基于构件的软件工程所需的可复用构件库。某些可复用构件是自行开发的，有些可从现有的系统中抽取得到，还可以从第三方获得。

不幸的是，可复用构件的存在并不能保证这些构件可以很容易或很有效地被集成到为新应用系统所选择的体系结构中去。因此，当计划使用某一构件时，要进行一系列的基于构件的开发活动。



构件合格性检验。构件合格性检验将保证某候选构件执行需要的功能，将完全适合系统的体系结构，并具有该应用所需的质量特性（例如，性能、可靠性、可用性）。

接口描述提供了构件的操作及使用的有用信息，但是，要决定建议的构件是否能够在新的应用中真正有效复用，只有上述信息是不够的。在构件合格性检验中考虑的因素有很多[BRO96]：应用编程接口（API）；构件所需的开发与集成工具；运行时需求，包括资源使用（如内存和存储器）、时间或速度以及网络协议；服务需求，包括操作系统接口及来自其他构件的支持；安全特征，包括访问控制和身份验证协议；嵌入式设计假定，包括特定的数值或非数值算法的使用；以及异常处理。

当计划使用自行开发的可复用构件时，这些因素都是比较容易评估的。然而，确定COTS的内部工作细节或第三方构件是非常困难的，因为唯一可利用的信息可能只是接口规格说明

² 值得注意的是，体系结构风格经常受到在领域工程中创建的一般结构模型的影响（见图30-1）。

本身。

构件适应性修改。在理想的情况下，领域工程创建构件库，构件可以容易地被集成到应用体系结构中。“容易集成”的含义是：(1) 库中的所有构件都已经实现了一致的资源管理方法，(2) 所有构件都存在诸如数据管理等公共活动，(3) 已经以一致的方式实现了体系结构内部及与外部环境的接口。

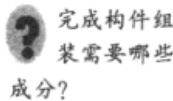


软件团队为复用所做的适应性修改，除了评估成本是否合算，还需要评估取得所需要的功能及性能是否是成本合算的。

855

实际上，即使已经对一个构件在应用体系结构内部的使用进行了合格性检验，也可能在刚才提到的一个或多个地方发生冲突。为了避免这些冲突，经常使用一种称为构件包装[BRO96]的适应性修改技术。当软件团队对某一构件的内部设计和代码具有完全的访问权时（当使用COTS构件时，常常不是如此），应用白盒包装技术。与软件测试中的白盒测试相对应（第14章），白盒包装检查构件的内部处理细节，并进行代码级的修改来消除任何冲突。当构件库提供了能够消除或掩盖冲突的构件扩展语言或API时，应用灰盒包装技术。黑盒包装技术需要在构件接口中引入前处理、后处理以消除或掩盖冲突。软件团队必须确定充分包装构件是否值得，或者考虑是否开发定制构件（对构件进行设计以消除遇到的冲突）。

构件组装。构件组装任务将经过合格性检验的、适应性修改的以及开发的构件组装到为应用建立的体系结构中。为完成这项任务，必须建立一个基础设施以将构件绑定到一个运行系统中。该基础设施（通常是专门的构件库）提供了构件协作的模型和使构件能够相互协作并完成共同任务的特定服务。



完成构件组
装需要哪些
成分？

为了完成构件组装，有很多机制用于创建有效的基础设施，这里介绍其中的四个“体系结构成分”[ADL95]：

- **数据交换模型。**对所有的可复用构件应该定义用户及应用间交互和传递数据的机制（例如，拖和放，剪切和粘贴）。数据交换机制不仅允许人向软件、构件向构件的数据传递，而且也允许在系统资源间进行数据传递（例如，将一个文件拖到打印机图标上输出）。
- **自动化。**应实现多种工具、宏结构和脚本以辅助可复用构件之间的交互。
- **结构化存储。**应将包含在“复合文档”中的异构数据（例如，图形数据、声音/视频、文本和数值数据）组织在一起，并作为一个单独的数据结构进行存取，而不是作为一组分离的文件。“结构化数据维护嵌套结构的一个描述性索引，应用程序可以自由导航以定位、创建或编辑数据内容，就像最终用户直接操作一样”[ADL95]。
- **底层对象模型。**对象模型保证在不同平台上用不同程序设计语言开发的构件可以互操作，也就是说，对象必须具有跨网络进行通信的能力。为了达到这个目标，对象模型定义了构件互操作的标准。

由于复用和CBSE对软件产业的影响非常大，一些大公司及产业联盟已经提出了一些构件软件的标准：

WebRef

有关CORBA
的最新信息可
从www.omg.
org获得。

- **OMG/CORBA。**对象管理组织发布了公共对象请求代理体系结构(OMG/CORBA)，一个对象请求代理(object request broker, ORB)提供了多种服务，使得可复用构件(对象)可以与其他构件通信，而不管这些构件在系统中的位置如何。

WebRef

有关COM的最新信息可从
www.microsoft.com/COM获得。

856

- Microsoft COM。微软开发了构件对象模型 (component object model, COM)，此模型提供了运行于Windows操作系统的单个应用使用不同厂商生产的构件的规格说明。COM包含两个元素：COM接口（实现为COM对象），注册和在COM接口间传递消息的一组机制。
- Sun JavaBean构件。JavaBean构件系统是一个可移植的、平台独立的、使用Java程序设计语言开发的CBSE基础设施。JavaBean构件系统包括一组工具，称为Bean开发工具箱 (Bean Development Kit, BDK)，它允许开发者做以下工作：(1) 分析现有的Bean (构件) 如何工作，(2) 定制它们的行为和外观，(3) 建立协作及通信机制，(4) 开发在特定应用中使用的定制Bean，(5) 测试和评估Bean的行为。

哪种标准将支配产业界？这不是目前容易回答的问题。虽然很多开发者已经采用了某个标准，但大型软件组织根据应用的类型和采用的平台，所有这三个标准都有可能选择。

INFO**对象请求代理体系结构**

客户/服务器 (C/S) 系统是用软件构件 (对象) 实现的，这些软件构件必须能够与位于单个机器 (可以是客户机，也可以是服务器) 或网络上的其他构件进行交互。对象请求代理 (ORB) 是一种“中间件”，它能够使位于客户机上的对象向封装在服务器上的某个对象内的方法发送消息。实质上，ORB拦截消息，并处理所需的所有通信与协作活动，包括发现处理消息的对象、调用其方法、传递合适的数据给该对象并将结果数据传回给最先产生该消息的对象。

CORBA、COM及JavaBean实现了对象请求代理思想，这里用CORBA举例说明ORB中间件。

CORBA体系结构的基本结构如图30-2所示。当在客户/服务器系统中实现CORBA时，使用接口描述语言 (interface description language, IDL) 描述位于客户机及服务器中的对象。接口描述语言是允许软件工程师定义对象、属性、方法及需要调用的消息的说明性语言。为适应客户端对象请求服务端方法的需要，创建客户机及服务器IDL存根 (stub)。存根提供了在C/S系统中请求对象的通路。

由于在运行时会跨越网络请求对象，必须建立对象描述的存储机制，使得在需要的时候可以获得对象及其位置的相关信息。接口库完成此功能。

当客户程序调用位于系统中其他地方的对象中的某一方法时，CORBA使用动态调用完成：(1) 从接口库中获得所期望方法的相关信息，(2) 生成将传递给对象的带有参数的数据结构，(3) 生成一个对象请求，(4) 调用该请求。然后，此请求被传递给ORB内核——管理请求的网络操作系统的特定实现部分，该请求完成。

请求被传递到内核并由服务器处理。在服务器端，对象适配器存储服务器端接口库中的类及对象信息，接收并管理来自客户端的请求，并执行多种其他对象管理功能。在服务器上，IDL存根 (与客户机上定义的存根类似) 用作与服务器上实际对象实现的接口。

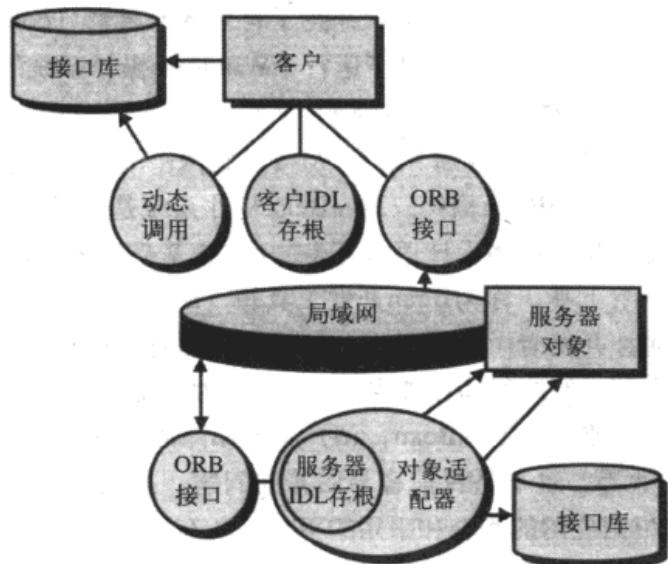


图30-2 基本CORBA体系结构

30.4.2 构件工程

如本章前面我们谈到的那样，CBSE过程鼓励使用现有的软件构件。然而，有时必须开发新构件，也就是说，必须开发新构件并与现有的COTS构件及自有构件集成。因为这些新构件将成为自有的可复用构件库的新成员，应按照复用的要求来开发。

开发可复用的软件构件并不神秘，抽象、隐藏、功能独立性、求精、结构化程序设计、面向对象方法、测试、SQA以及正确性验证方法，所有这些设计概念都可用于创建可复用的软件构件³。本节中，我们不再讨论这些主题，而是考虑特定的复用问题，它们是对可靠的软件工程实践的补充。

30.4.3 复用的分析与设计

对分析模型进行分析以确定模型中指向现有可复用构件的元素。问题是怎样以能够导致“规格说明匹配”的形式从需求模型中抽取信息。



当构件必须与遗产软件、或者与体系结构及接口协议不一致的多个系统连接或集成时，DFR可能是非常困难的。

如果规格说明生成符合当前应用系统所需要的构件，设计者可从可复用构件库中提取这些构件并将它们用于新系统的设计。如果没能找到设计构件，软件工程师必须应用传统的或面向对象的设计方法去创建它们。正是在这点——当设计者开始创建新的构件时，应该考虑为复用而设计（design for reuse，DFR）。

如我们已经提到过的那样，DFR需要软件工程师应用一致的设计概念和原则（第9章），但是，也必须考虑应用领域的特征。Binder[BIN93]提出了构成可复用的设计基础的一系列关键问题⁴：

- **标准数据。**应该研究应用领域，并标识出标准的全局数据结构（例如，文件结构或完整的数据库），然后就可以对所有设计构件如何使用这些标准数据结构进行描述。

³ 要更多地了解这些概念，见本书的第二部分和第五部分。

⁴ 通常，DFR准备应该作为领域工程的一部分来进行（30.3节）。

- **标准接口协议。**应该建立三个层次的接口协议：模块内接口的本质、外部的技术（非人类）接口的设计及人机界面。
- **程序模板。**结构模型（30.3.3节）可以作为新程序体系结构设计的模板。

一旦建立了标准数据、接口和程序模板，设计者就有了一个设计框架。符合这个框架的新构件以后复用的概率更高。

30.5 构件分类与检索

考虑一个大学图书馆，有成千上万的书籍、期刊和其他信息资源。然而为了访问这些资源，必须有合适的分类模式。为了在这大量的信息中导航浏览，图书馆管理员定义了一种分类模式，它包括美国国会图书馆分类码、关键词、作者名及其他索引条目，所有这些使得用户可以快速和方便地找到所需资源。

现在考虑一个大的构件仓库，其中存放了成千上万的可复用构件。但是，软件工程师如何找到他所需要的构件呢？为了回答这个问题，又出现了另一个问题：我们如何以无歧义的、可分类的术语来描述软件构件？这些是困难的问题，至今还没有明确答案。在本节中，我们探讨当前的研究方向，使未来的软件工程师可以导航浏览复用库。

30.5.1 描述可复用构件

可以用很多方式来描述可复用软件构件，但是理想的描述包括Tracz[TRA90]提出的3C模型——概念（concept）、内容（content）和环境（context）。

软件构件的概念是“构件做什么的描述”[WHI95]。对构件的接口进行完整地描述，并且对语义——以带有前置条件及后置条件的上下文来表示——进行标识。概念将传达构件的意图。

构件的内容描述概念如何被实现。在本质上，内容是对一般用户隐蔽的信息，只有那些想要修改或测试该构件的人才需要了解。

环境将可复用软件构件放到其应用领域中。即，通过描述概念的、操作的和实现的特征，环境使得软件工程师能够发现满足应用需求的合适构件。

为了在实际环境中使用，概念、内容和环境必须被转换为具体的规格说明模式。关于可复用构件的分类模式，已有很多文章（例如，[LUC01]及[WHI95]包括广泛的参考书目）。所提出的方法可以分为三大类：图书馆和信息科学方法，人工智能方法，以及超文本系统。到目前为止，已经完成的绝大部分工作建议使用图书馆科学方法进行构件分类。

图30-3给出了源于图书馆科学索引方法的一个分类法，“受控的索引词汇”限制了可用来分类对象（构件）的术语和语法。“不受控的索引词汇”则对描述的性质没有限制。大多数软件构件分类模式可归为如下三类：

- **枚举分类。**通过定义一个层次结构来描述构件，在该层次中定义软件构件的类以及不同层次的子类。枚举分类模式的层次结构使得它易于理解和使用，然而，在建立层次之前，必须进行领域工程以获得足够信息了解层次中的适当项。
- **刻面分类。**对领域进行分析，并标识出一组基本的描述特征。这些特征称为刻面，根据重要性对其进行排序，并与构件相关联。刻面可以描述构件执行的功能、被操作的数据、构件应用的环境或其他特征。描述构件的刻面的集合称为刻面描述符，通常，限定刻面描述不超过7或8个刻面。

859

860

- **属性-值分类。**为领域中的所有构件定义一组属性，然后，与刻面分类方法非常相似，将值赋给这些属性。事实上，属性-值分类方法和刻面分类方法是类似的，只是以下几点不同：(1) 对可使用的属性数量没有限制，(2) 属性没有优先级，(3) 不使用辞典功能。

基于对这些分类技术的实验研究，Frakes和Pole[FRA94]指出：没有明显的“最好”技术，也“没有某种方法比别的方法在查找效果上更适度……”。在复用库的有效分类模式的开发方面，还有更多的工作要做。

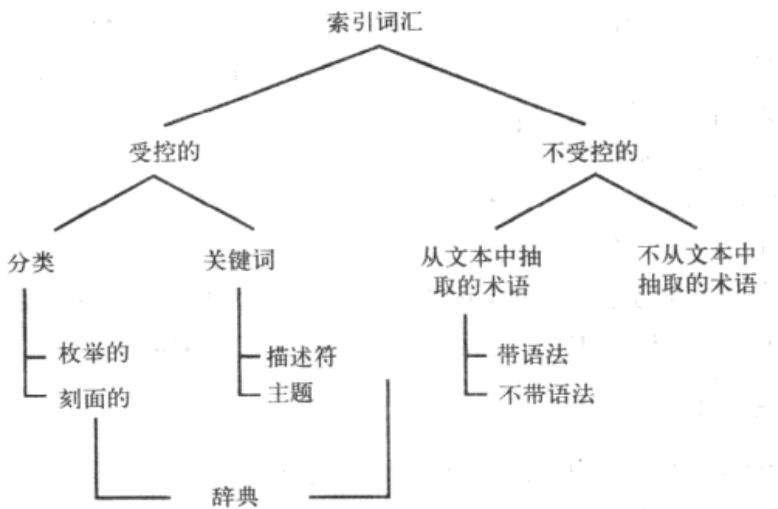


图30-3 一种索引方法的分类法

30.5.2 复用环境

软件构件复用必须有环境的支撑，支撑环境应包含如下元素：

- 能够存储软件构件的构件库及查找这些构件必需的分类信息。
- 提供访问构件库功能的库管理系统。
- 软件构件检索系统（例如，对象请求代理）——以允许客户应用系统从构件库服务器中检索构件和服务。
- 支持将复用的构件集成到新设计或实现中的CBSE 工具。

每一个功能在复用库的范围内交互或者被包含在复用库中。

复用库是更大型软件库（第27章）的一个元素，并且为软件构件及各种可复用的工作产品（例如，规格说明、设计、模式、框架、代码段、测试用例、用户指南）提供存储设施。复用库包含一个数据库以及查询数据库和从数据库中检索构件所必需的工具，构件分类模式（30.5.1节）是构件库查询的基础。

861

WebRef

CBSE方面的
广泛资源可在
<http://www.cbd-hq.com>找到。

通常用前面描述的3C 模型中的环境来描述查询。如果一个初始查询产生大量的候选构件，则对查询进行优化以减少候选对象。然后，抽取概念和内容信息（在找到候选构件之后），以辅助开发者选择合适的构件。

本书不打算详细讨论复用库结构及管理它们的工具，有兴趣的读者可参阅文献[FIS00]和[LIN95]。

SOFTWARE TOOLS**基于构件的开发**

目的：在建模、设计、评审以及集成软件构件作为更大的系统的一部分方面起辅助作用。

机制：工具的机制各异。一般情况下，CBD工具对以下一项或多项工作起辅助作用：软件体系结构的规格说明和建模；可利用的软件构件的浏览及选择；构件集成。

代表性工具⁵

ComponentSource (www.componentsource.com) 提供了大量被许多不同构件标准支持的COTS软件构件（及工具）。

Component Manager，由Flashline (www.flashline.com) 开发，“它是一个应用程序，能支持、促进及测量软件构件复用。”

Select Component Factory，由Select Business Solutions (www.selectbs.com/products) 开发，“它是一个集成的成套产品，用于软件设计、设计评审、服务/构件管理、需求管理及代码生成。”

Software Through Pictures-ACD，由Aonix (www.aonix.com) 发布，它能够运用OMG模型驱动体系结构的UML——一个开放的、卖方中立的CBSE方法，进行广泛的建模。

30.6 CBSE经济学

WebRef

对CBD及基于COTS的系统提供指导的多种文章可在www.sei.cmu.edu找到。

在直觉上，基于构件的软件工程是有吸引力的。在理论上，它将为软件组织提供质量和时间上的优势，而这些将直接导致成本的节省。但是，有实际的数据支持我们的直觉吗？

为了回答这个问题，我们必须首先了解软件工程中什么可以被复用以及和复用相关的成本究竟是多少。因而，可能要开展构件复用的成本—收益分析。

30.6.1 对质量、生产率及成本的影响

来自产业实例研究（例如，[ALL02]、[HEN95]、[MCM95]）的大量证据表明：积极的软件复用可带来可观的商业效益。产品质量、开发生产率以及整体成本都将得到改善。

质量。理想情况下，为了复用而开发的软件构件将被验证是正确的（见第29章）且不含有缺陷。实际上，形式化验证并不能常规性地实施，缺陷可能出现，也确实出现了。然而，随着每一次复用，缺陷被发现并消除，构件的质量也随之改善。随着时间的推移，构件就没有缺陷了。

在惠普公司所做的研究中，Lim[LIM94]报告说：被复用代码的缺陷率是每千行代码（KLOC）中有0.9个错误，而新开发代码的缺陷率是每千行代码（KLOC）中有4.1个缺陷。对一个包含68%的复用代码的应用系统来说，缺陷率是每千行代码（KLOC）中有2.0个缺陷——相对于不使用复用开发的应用系统，对期望的缺陷率有51%的改善。Henry和Faller[HEN95]的研究指出在质量上有35%的改善。虽然不同的报告得到不同的改善率（位于合理的范围内），

862

⁵ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

30.7 小结

基于构件的软件工程为软件质量、开发生产率以及整个系统成本带来了固有的收益，然而，在软件产业界广泛使用CBSE过程模型之前，必须克服很多障碍。

除了软件构件，软件工程师可以获得多种可复用的软件制品，包括软件的技术表示（例如，规格说明、体系结构模型、设计）、文档、模式、框架、测试数据，甚至包括过程相关的任务（例如，检查技术）。

CBSE过程包括两个并发的子过程：领域工程和基于构件的开发。领域工程的目的是在特定应用领域中标识、构造、分类和传播一组软件构件。然后，基于构件的开发对这些构件进行合格性检验、适应性修改，并将它们集成到新系统中。此外，基于构件的开发将基于新系统的特别需求开发新的构件。

可复用构件的分析、设计技术采用与在良好的软件工程实践中使用的相同的原理和概念。可复用构件应该在一个环境中设计，该环境为每个应用领域建立标准数据结构、接口协议和程序体系结构。

基于构件的软件工程使用数据交换模型、工具、结构化存储以及底层对象模型来构造出适用的软件。对象模型通常遵从一个或多个构件标准（例如，OMG/CORBA），这些标准通常定义了应用问题能访问可复用对象的方式。分类模式使得开发者能够发现和检索可复用构件并要依据标识概念、内容和环境的模型。枚举分类、刻面分类和属性—值分类是很多构件分类模式中的典型代表。

软件复用经济学可用一个问题来概括：少构建多复用是成本合算的吗？通常，答案是肯定的，但是，软件项目计划者必须考虑那些和可复用构件的合格性检验、适应性修改及集成相关的可观成本。

参考文献

- [ADL95] Adler, R.M., "Emerging Standards for Component Software," *Computer*, vol. 28, no. 3, March 1995, pp. 68–77.
- [ALL02] Allen, P., "CBD Survey: The State of the Practice," *The Cutter Edge*, March, 2002, available at <http://www.cutter.com/research/2002/edge020305.html>.
- [ATK01] Atkinson, C., et al; *Component-Based Product Line Engineering with UML*, Addison-Wesley, 2001.
- [BAS94] Basili, V. R., L. C. Briand, and W. M. Thomas, "Domain Analysis for the Reuse of Software Development Experiences," *Proc. of the 19th Annual Software Engineering Workshop*, NASA/GSFC, Greenbelt, MD, December 1994.
- [BIN93] Binder, R., "Design for Reuse Is for Real," *American Programmer*, vol. 6, no. 8, August 1993, pp. 30–37.
- [BRO96] Brown, A. W., and K. C. Wallnau, "Engineering of Component-Based Systems," *Component-Based Software Engineering*, IEEE Computer Society Press, 1996, pp. 7–15.
- [CHR95] Christensen, S. R., "Software Reuse Initiatives at Lockheed," *CrossTalk*, vol. 8, no. 5, May 1995, pp. 26–31.
- [CLE95] Clements, P. C., "From Subroutines to Subsystems: Component-Based Software Development," *American Programmer*, vol. 8, No. 11, November 1995.
- [DOG03] Dogru, A., and M. Tanik, "A Process Model for Component-Oriented Software Engineering," *IEEE Software*, vol. 20, no. 2, March/April 2003, pp. 34–41.
- [FIS00] Fischer, B., "Specification-Based Browsing of Software Component Libraries," *J. Automated Software Engineering*, vol. 7, no. 2, 2000, pp. 179–200, available at <http://ase.arc.nasa.gov/people/fischer/papers/ase-00.html>.

30.7 小结

基于构件的软件工程为软件质量、开发生产率以及整个系统成本带来了固有的收益，然而，在软件产业界广泛使用CBSE过程模型之前，必须克服很多障碍。

除了软件构件，软件工程师可以获得多种可复用的软件制品，包括软件的技术表示（例如，规格说明、体系结构模型、设计）、文档、模式、框架、测试数据，甚至包括过程相关的任务（例如，检查技术）。

CBSE过程包括两个并发的子过程：领域工程和基于构件的开发。领域工程的目的是在特定应用领域中标识、构造、分类和传播一组软件构件。然后，基于构件的开发对这些构件进行合格性检验、适应性修改，并将它们集成到新系统中。此外，基于构件的开发将基于新系统的特别需求开发新的构件。

可复用构件的分析、设计技术采用与在良好的软件工程实践中使用的相同的原理和概念。可复用构件应该在一个环境中设计，该环境为每个应用领域建立标准数据结构、接口协议和程序体系结构。

基于构件的软件工程使用数据交换模型、工具、结构化存储以及底层对象模型来构造出适用的软件。对象模型通常遵从一个或多个构件标准（例如，OMG/CORBA），这些标准通常定义了应用问题能访问可复用对象的方式。分类模式使得开发者能够发现和检索可复用构件并要依据标识概念、内容和环境的模型。枚举分类、刻面分类和属性—值分类是很多构件分类模式中的典型代表。

软件复用经济学可用一个问题来概括：少构建多复用是成本合算的吗？通常，答案是肯定的，但是，软件项目计划者必须考虑那些和可复用构件的合格性检验、适应性修改及集成相关的可观成本。

参考文献

- [ADL95] Adler, R.M., "Emerging Standards for Component Software," *Computer*, vol. 28, no. 3, March 1995, pp. 68–77.
- [ALL02] Allen, P., "CBD Survey: The State of the Practice," *The Cutter Edge*, March, 2002, available at <http://www.cutter.com/research/2002/edge020305.html>.
- [ATK01] Atkinson, C., et al; *Component-Based Product Line Engineering with UML*, Addison-Wesley, 2001.
- [BAS94] Basili, V. R., L. C. Briand, and W. M. Thomas, "Domain Analysis for the Reuse of Software Development Experiences," *Proc. of the 19th Annual Software Engineering Workshop*, NASA/GSFC, Greenbelt, MD, December 1994.
- [BIN93] Binder, R., "Design for Reuse Is for Real," *American Programmer*, vol. 6, no. 8, August 1993, pp. 30–37.
- [BRO96] Brown, A. W., and K. C. Wallnau, "Engineering of Component-Based Systems," *Component-Based Software Engineering*, IEEE Computer Society Press, 1996, pp. 7–15.
- [CHR95] Christensen, S. R., "Software Reuse Initiatives at Lockheed," *CrossTalk*, vol. 8, no. 5, May 1995, pp. 26–31.
- [CLE95] Clements, P. C., "From Subroutines to Subsystems: Component-Based Software Development," *American Programmer*, vol. 8, No. 11, November 1995.
- [DOG03] Dogru, A., and M. Tanik, "A Process Model for Component-Oriented Software Engineering," *IEEE Software*, vol. 20, no. 2, March/April 2003, pp. 34–41.
- [FIS00] Fischer, B., "Specification-Based Browsing of Software Component Libraries," *J. Automated Software Engineering*, vol. 7, no. 2, 2000, pp. 179–200, available at <http://ase.arc.nasa.gov/people/fischer/papers/ase-00.html>.

- [FRA94] Frakes, W. B., and T. P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components," *IEEE Trans. Software Engineering*, vol. SE-20, no. 8, August 1994, pp. 617-630.
- [HEI01] Heineman, G., and W. Councill (eds.), *Component-Based Software Engineering*, Addison-Wesley, 2001.
- [HEN95] Henry, E., and B. Faller, "Large Scale Industrial Reuse to Reduce Cost and Cycle Time," *IEEE Software*, September 1995, pp. 47-53.
- [HUT88] Hutchinson, J. W., and P. G. Hindley, "A Preliminary Study of Large Scale Software Reuse," *Software Engineering Journal*, vol. 3, no. 5, 1988, pp. 208-212.
- [LIA93] Liao, H., and Wang, F., "Software Reuse Based on a Large Object-Oriented Library," *ACM Software Engineering Notes*, vol. 18, no. 1, January 1993, pp. 74-80.
- [LIM94] Lim, W. C., "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Software*, September 1994, pp. 23-30.
- [LIN95] Linthicum, D. S., "Component Development (a Special Feature)," *Application Development Trends*, June 1995, pp. 57-78.
- [LUC01] deLucena, Jr., V., "Facet-Based Classification Scheme for Industrial Software Components," 2001, can be downloaded from <http://research.microsoft.com/users/cszypers/events/WCOP2001/Lucena.pdf>.
- [MCM95] McMahon, P.E., "Pattern-Based Architecture: Bridging Software Reuse and Cost Management," *Crosstalk*, vol. 8, no. 3, March 1995, pp. 10-16.
- [ORF96] Orfali, R., D. Harkey, and J. Edwards, *The Essential Distributed Objects Survival Guide*, Wiley, 1996.
- [PRI93] Prieto-Diaz, R., "Issues and Experiences in Software Reuse," *American Programmer*, vol. 6, no. 8, August 1993, pp. 10-18.
- [POL94] Pollak, W., and M. Rissman, "Structural Models and Patterned Architectures," *Computer*, vol. 27, no. 8, August 1994, pp. 67-68.
- [STA94] Starlinger, W., "Constructing Applications from Reusable Components," *IEEE Software*, September 1994, pp. 61-68.
- [TRA90] Tracz, W., "Where Does Reuse Start?" *Proc. Realities of Reuse Workshop*, Syracuse University CASE Center, January 1990.
- [TRA95] Tracz, W., "Third International Conference on Software Reuse—Summary," *ACM Software Engineering Notes*, vol. 20, no. 2, April 1995, pp. 21-22.
- [WHI95] Whittle, B., "Models and Languages for Component Description and Reuse," *ACM Software Engineering Notes*, vol. 20, no. 2, April 1995, pp. 76-89.
- [YOU98] Yourdon, E. (ed.), "Distributed Objects," *Cutter IT Journal*, vol. 11, no. 12, December 1998.

习题与思考题

- 30.1 开发一组和字处理/桌面出版软件相关的领域特征。
- 30.2 应用领域的领域特征和构件分类模式有什么相同点和不同点？
- 30.3 研究一下领域工程并丰富图30-1所示的过程模型。标识领域分析和软件体系结构开发所必需的任务。
- 30.4 虽然软件构件是最明显的可复用“制品”，但作为软件工程一部分的很多其他工作产品也可被复用。考虑项目计划和成本估算，这些如何被复用？这样做带来的收益是什么？
- 30.5 为处理大学学生数据的信息系统开发一组领域特征。
- 30.6 复用的关键障碍之一是使软件开发者考虑复用现有的构件，而不是重新开发新的构件（毕竟，构建东西很有吸引力）。请向软件组织建议3~4种不同的方式来激励软件工程师进行复用。为了支持复用效果，应该采用什么技术？
- 30.7 对导师指定的或你熟悉的某个应用领域设计一种刻面分类模式。
- 30.8 获取最新的CORBA（或COM、JavaBeans）标准方面的资料，准备3~5页的论文讨论其主要特点。获取关于对象请求代理工具的信息，并举例说明工具如何符合标准。

- 30.9 什么是结构点?
- 30.10 对导师指定的或你熟悉的某个应用领域设计枚举分类。
- 30.11 研究文献以获取最近的支持使用CBSE的质量和生产率数据，并在班里展示。
- 30.12 对导师指定的或你熟悉的某个应用领域开发一个简单的结构模型。

推荐读物与阅读信息

近几年出版了很多关于基于构件的开发和构件复用的书籍。Heineman和Council[HEI01]、Brown (《Large Scale Component-Based Development》, Prentice-Hall, 2000)、Allen (《Realizing e-Business with Components》, Addison-Wesley, 2000)、Herzum和Sims (《Business Component Factory》, Wiley, 1999) 以及Allen、Frost和Yourdon (《Component-Based Development for Enterprise Systems: Applying the Select Perspective》, Cambridge University Press, 1998) 覆盖了CBSE过程的所有重要方面。Apperly及其同事 (《Service and Component-Based Development》, Addison-Wesley, 2003)、Atkinson[ATK01]以及Cheesman 和Daniels (《UML Components》, Addison-Wesley, 2000) 重点讨论了使用UML的CBSE。

Leach (《Software Reuse: Methods, Models, and Costs》, McGraw-Hill, 1997) 详细分析了与CBSE和复用相关的成本问题。Poulin (《Measuring Software Reuse: Principles, Practices, and Economic Models》, Addison-Wesley, 1996) 建议了许多评估软件复用收益的定量方法。

近几年已经出版了数十本关于基于构件的工业标准的书籍。这些书籍讨论了标准本身的内容，但也考虑了很多重要的CBSE主题。对于本章讨论的三个标准，有关的书籍有：

CORBA

- Bolton, F., *Pure CORBA*, Sams Publishing, 2001.
- Doss, G. M., *CORBA Networking With Java*, Wordware Publishing, 1999.
- Hoque, R., *CORBA for Real Programmers*, Academic Press/Morgan Kaufmann, 1999.
- Siegel, J., *CORBA Fundamentals and Programming*, Wiley, 1999.
- Slama, D., J. Garbis, and P. Russell, *Enterprise CORBA*, Prentice-Hall, 1999.

COM

- Box, D., K. Brown, T. Ewald, and C. Sells, *Effective COM: 50 Ways to Improve Your COM- and MTS-Based Applications*, Addison-Wesley, 1999.
- Gordon, A., *The COM and COM+ Programming Primer*, Prentice-Hall, 2000.
- Kirtland, M., *Designing Component-Based Applications*, Microsoft Press, 1999.
- Tapadiya, P., *COM+ Programming*, Prentice-Hall, 2000.

很多组织组合使用构件标准。Geraghty及其同事 (《COM-CORBA Interoperability》, Prentice-Hall, 1999)、Pritchard (《COM and CORBA Side by Side: Architectures, Strategies, and Implementations》, Addison-Wesley, 1999) 以及Rosen及其同事 (《Integrating CORBA and COM Applications》, Wiley, 1999) 考虑了同时使用CORBA和COM作为基于构件的开发基础等相关问题。

JavaBeans

- Asbury, S., and S. R. Weiner, *Developing Java Enterprise Applications*, Wiley, 1999.
- Anderson, G., and P. Anderson, *Enterprise Javabeans Component Architecture*, Prentice-Hall,

2002.

Monson-Haefel, R., *Enterprise Javabeans*, third edition, O'Reilly & Associates, 2001.

Roman, E., et al., *Mastering Enterprise Javabeans*, 2nd ed., Wiley, 2001.

大量关于基于构件的软件工程和构件复用的信息源可从因特网上获得。关于CBSE的

868 WWW参考文献最新列表可在SEPA Web站点<http://www.mhhe.com/pressman>找到。



第31章 再 工 程

要点浏览

概念：想想任何曾经服务得很好的技术产品。虽然你定期使用它，但是它正在逐渐老化，经常出故障，对它进行修复所花费的时间已经使你不可忍受，同时，它也不再代表最新的技术。怎么办？如果该产品是硬件，你可能会丢弃，并购买一个新款。但是，如果它是定制构造的软件，丢弃是不行的。你需要对其进行重构，构造一个具有更好的功能、更好的性能以及更好维护的产品。这就是我们所说的再工程。

人员：在组织层次上，再工程由业务专家（通常是咨询公司）完成。在软件层次上，再工程由软件工程师完成。

重要性：我们生活在快速变化的世界中，业务功能的要求和支持它们的信息技术的变化步伐给每一个商业组织带来了巨大的竞争压力，必须对业务和支持业务的（或本身也是业务的）软件进行再工

程，以跟上变化的步伐。

步骤：业务过程再工程（Business process reengineering, BPR）定义业务目标，识别和评估现有的业务过程，并对业务过程进行修订，以更好地满足当前的业务目标。软件再工程过程包括库存目录分析、文档重构、逆向工程、程序和数据重构，以及正向工程。这些活动的意图是创建具有更高质量和更好维护的现有程序的新版本。

工作产品：再工程产生一系列工作产品（例如，分析模型、设计模型、测试规程），最终产品是再工程后的业务过程和（或）支撑该过程的再工程后的软件。

质量保证措施：再工程中使用的SQA（软件质量保证）实践与应用于各个软件工程过程的SQA实践相同——用正式技术评审来评估分析模型和设计模型，用专门的评审考查业务适用性和兼容性，用测试来发现内容、功能和互操作性方面的错误。

关键概念
BPR过程模型
C/S体系结构
数据结构
经济学
正向工程
库存目录分析
维护
OO体系结构
再工程过程
重构
逆向工程

在为《Harvard Business Review》撰写的一篇很有见解性的文章中，Michael Hammer[HAM90]为业务过程和计算方面的管理变革奠定了基础：

现在是该停止铺设牛道的时候了。不要将过时的工艺过程嵌入到硅片和软件中，我们应该删除它们，并从头开始。应该对我们的业务进行“再工程”（reengineer）：使用现代信息技术的优势重新设计业务过程，以获得性能的极大改善。

每个公司按照很多无关联的规则运作……再工程力图使我们摆脱组织和管理业务的旧规则。

和所有革命一样，Hammer的号召产生了正面和负面的影响。上个世纪90年代，某些公司对再工程进行了有益的尝试，并由此增强了公司的竞争能力；而另一些公司只是通过减小规模和外购（代替再工程）来改善他们的基本条件，这些公司将来的发展潜力通常很小[DEM95]。

在21世纪的头十年，对再工程的大肆宣传已经减弱，但是这个过程本身仍在大大小小的公司中继续进行。业务再工程和软件工程间的关系开始进入系统视线中。

869



业务过程再工程经常能够得到新的软件功能，然而，软件再工程工作能够以更好的、可维护性更强的软件替换现有的软件功能。

软件通常是Hammer所讨论的业务规则的实现，当这些规则改变时，软件也必须改变。今天，大多数公司有成千上万的支持旧业务规则的计算机程序，当管理者为了获得更高的效率和竞争力而修改业务规则时，软件也必须保持同步。在某些情况下，这意味着需要构建大量的、新的基于计算机的系统¹，但是，在很多其他情况下，只需要对现有应用系统进行修改或重构。

在本章中，我们以自顶向下的方式考察再工程，从业务过程再工程的概述开始，进而对软件再工程中发生的技术活动进行更详细的讨论。

31.1 业务过程再工程

业务过程再工程（BPR）远远超出了信息技术和软件工程的范畴。有很多关于BPR的定义（大多数有点抽象），刊登于《财富》杂志[STE93]的定义为：“搜寻并实现业务过程中的根本性改变，以取得突破性成果”。但是，如何进行搜寻？如何获得实现？更重要的是，我们如何能够保证所建议的“根本性改变”确实能够取得“突破性成果”，而又不会造成组织的混乱？

870

“用昨天的思考方法去面对明天就是在停顿中想象生活。”

— James Bell

31.1.1 业务过程

一个业务过程是“通过执行一组逻辑相关的任务得出定义的业务结果”[DAV90]。在业务过程中，将人、设备、材料资源以及业务规程综合在一起产生特定的结果。业务过程的例子有：设计新产品，购买服务和支持，雇佣新员工，以及向供应商付费。每种业务过程都需要一组任务，而且要在业务中利用不同的资源。



作为软件工程师，你的再工程工作位于业务层次的底部。但是，要确保已有某人对上层进行过认真考虑。如果不是这样，你的工作则处在危险之中。

每个业务过程有一个指定的客户——即接收过程结果（例如，一个想法、一个报告、一个设计、一个产品）的个人或小组。此外，业务过程一般会跨越组织边界，需要来自不同组织的小组共同参与定义过程的“逻辑相关的任务”。

在第6章中，我们看到每一个系统实际上是由子系统构成的层次结构。业务也不例外，每一个业务系统（也称为业务功能）都是由一个或多个业务过程组成，而每个业务过程又可定义为一组子过程。

BPR可以应用于层次结构的任意级别，但是，随着BPR范围的扩大（即，我们在层次中向上移动），与BPR关联的风险将急剧增长。正因为如此，大多数BPR工作只着重于个体过程或子过程。

“只要能够在一个新事物中展示出往昔的东西，我们就会感到安慰。”

—F.W.Nietzsche

¹ 在本书第三部分中讨论的基于Web的应用及系统的爆炸性发展就预示了这种趋势。

31.1.2 BPR模型

WebRef

关于BPR的更多信息可在www.brint.com/BPR.htm找到。

和大多数工程活动一样，业务过程再工程是迭代的。业务目标及达到目标的过程必须适应不断变化的业务环境。因此，BPR没有开始和结束——它是一个演化的过程。图31-1描述了一个业务过程再工程模型，该模型定义了6项活动：

- **业务定义**。在4个关键驱动因素上下文中识别业务目标，这4个关键驱动因素是：减少成本、减少时间、改善质量、人力开发及授权。可以在业务级或针对某特定业务成分来定义业务目标。
- **过程识别**。对取得业务中所定义的目标起关键作用的过程进行识别，然后可以根据重要性、改变的需要或以任何其他适于再工程活动的方式对这些过程按优先级排序。
- **过程评估**。对现有过程进行透彻分析和测量。确定过程任务，说明过程任务花费的成本和时间，并且明确质量/性能问题。
- **过程规格说明和设计**。根据从上面三个BPR活动中获得的信息，为每个将被重新设计的过程准备用例（第7章）。在BPR的范畴内，用例标识一种场景，这种场景可将某些结果传递给客户。使用用例作为过程的规格说明，为过程设计一组新任务。
- **原型开发**。在一个重新设计的业务过程完全集成到整个业务之前，必须将其原型化。该活动对重新设计的业务过程进行“测试”，测试通过后再进行下一步的求精工作。
- **求精和实例化**。根据来自原型的反馈信息，对业务过程进行求精，然后在业务系统中实例化。

上述的BPR活动有时是和工作流分析工具联合使用的，使用这些工具的目的是建立现有工作流的模型，以便对现有的过程进行更好的分析。此外，通常和业务过程工程活动相关的建模技术（第6章）也可以用于实现上面过程模型中描述的前4项活动。

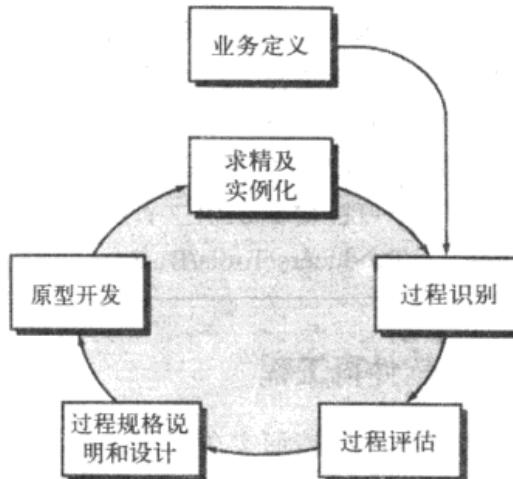


图31-1 BPR模型

871

业务过程再工程

SOFTWARE TOOLS

872

目的：BPR工具支持现有业务过程的分析与估算，以及新业务过程的规格说明与设计。

机制：工具的机制各异。一般情况下，BPR工具允许业务分析员对现有的业务过程建模，主要用于评定工作流的效率缺陷及功能问题。一旦识别出存在的问题，工具允许分析员对修订的业务过程开发原型及/或进行模拟。

代表性工具²

Extend，由ImagineThat, Inc. (www.imagethatinc.com) 开发，是对现有过程建模及探索新过程的一种模拟工具。Extend提供了全面的“如果……，就……”(what if) 的能力，

² 这里记录的工具并不代表本书支持这些工具，而只是此类工具的一些样例。在大多数情况下，工具的名字由各自的开发者注册为商标。

使得业务分析员能够探索不同的过程场景。

e-Work，由Metastorm (www.metastorm.com) 开发，为手工及自动过程提供业务过程管理支持。

IceTools，由Blue Ice (www.blueice.com) 开发，是Microsoft Office及Microsoft Project的BPR模板集合。

SpeeDev，由NimbleStar Group (www.nimberstar.com) 开发，是多种能够使一个组织对过程工作流建模（此处指IT工作流）的工具中的一个。

Workflow tools，由MetaSoftware (www.metasoftware.com) 开发，包括一整套工作流建模、模拟及进度安排工具。

一个可链接到BPR工具的有用列表可以在<http://www.donald-firesmith.com/Components/Producers/Tools/BusinessProcessReengineeringTools.html>找到。

31.2 软件再工程

类似这样的情形实在很普遍：某应用系统已经为公司的业务需要服务了10年或15年，在此期间，已经对它进行了多次改正性、适应性及增强性维护。人们怀着极好的愿望从事这项工作，但是，好的软件工程实践总是被抛在一边（由于其他方面的压力）。现在，应用系统处于不稳定的状态，虽然它仍在工作，但每次维护都会产生预料不到的、严重的副作用。然而，这个应用系统必须继续使用，怎么办？

不可维护的软件并不是什么新问题。事实上，对软件再工程的强调源于40多年来软件维护问题的不断升温。

31.2.1 软件维护

30多年以前，软件维护被描述[CAN72]为“冰山”。我们希望那些一眼可见的就是所有实际存在的，但是我们知道，在表面之下存在大量潜在的问题和成本。在20世纪70年代早期，“维护冰山”大到足以使一艘航空母舰沉没。而今天，它可以轻而易举地使整个海军沉没。

873 对现有软件的维护可能占据开发组织所有工作量的60%以上，而且随着软件产量的增长，这个比例还在继续上升[HAN93]。外行的读者可能会问，为什么需要这么多的维护？为什么要花费这么大的力量来进行维护？Osborne和Chikofsky[OSB90]给出了部分答案：

我们现在使用的很多软件已有10到15年的历史。即使这些程序是采用当时最好的设计和编码技术开发的（大多数并不是这样），当时主要关注的也是程序规模和存储空间。这些软件后来被移植到新的平台上，根据机器和操作系统技术方面的变化对其进行调整，为满足新的用户需要对其进行增强——所有这些并没有对整体体系结构给予足够关注，其结果是，我们现在仍在运行的软件系统的设计结构、编码、逻辑及文档都很差……

软件维护问题的另一个原因是软件人员变动。最初从事开发工作的软件团队（或个人）可能已经不在了。更糟糕的是，后来的软件人员已经修改了系统，最后也离开了。目前，已

经没有人对这个遗产系统直接了解了。

如我们在第27章中讲到的，所有软件工作都是在普遍存在的变更之中进行的。当开发基于计算机的系统时，变更是不可避免的。因此，我们一定要开发评估、控制及进行修改的机制。

“程序可维护性和程序可理解性是相关的概念：程序越难于理解，也就越难于维护。”

—Gerald Berns

读了上面的一段话，读者可能抗议说：“但是我并没有花费60%的时间修改我所开发的程序中的错误”。当然，软件维护并不仅仅是“修改错误”，我们可以通过描述程序在发布使用后发生的4类活动[SWA76]来定义维护。



软件维护可以定义为四类不同的活动：改正性维护，适应性维护，完善性维护或增强，预防性维护或再工程。所有的维护工作中，大约仅有20%是“修改错误”，其余80%花费在：修正现有系统以适应外部环境的改变，根据用户要求进行增强，以及为了未来的使用对应用系统进行再工程。当维护包括所有这些活动时，就不难看出为什么维护工作量这样大了。

31.2.2 软件再工程过程模型

WebRef

有关软件再工程的极好的信息源可以在www.reengineering.net找到。

再工程要花费时间，消耗大量的资金，并占用资源。否则，这些花费本可用于当前关注的事情上。由于所有这些理由，再工程不是在几个月或甚至几年内可以完成的。信息系统的再工程将是消耗信息技术资源长达多年的活动，因此每个组织都需要注重有实效的软件再工程策略。

874

再工程过程模型中包含一个可行的策略，我们将在本节后面讨论该模型，首先讨论某些基本原则。

再工程是一个重构活动，以一个类似的活动——重建一所房子——为例，我们就可以更好地理解信息系统的再工程。考虑如下情况：

假定你在另一个州购买了一所房子。你还没有看到房子，就以令人吃惊的低价格买下了，给你的警告是：它可能需要彻底重建。你将如何进行该项工作？

- 在开始重建前，首先检查一下房子。这种做法似乎是合理的。为了确定它是否确实需要重建，你（或职业检查员）可能列出一组标准，使得检查工作能系统地进行。
- 在拆掉并重建整个房子前，确认其结构是不是牢固的。如果该房子结构良好，则可能是“改造”（remodel），而不是重建（以低得多的价格和少得多的时间）。
- 在开始重建前，确保你已经了解房子最初是如何建造的。看一看墙内部，了解布线、管道以及内部结构。即使你不顾所有这些，详细了解原房屋对你开始建造也一定是有帮助的。
- 如果开始重建，应该只使用最现代的、耐久的材料。现在看来可能会贵一些，但是，这样做会使你避免将来昂贵而耗时的维护。
- 如果决定重建，一定要采用严格的方式，使用现在及将来都将获得高质量的做法。

虽然上面的原则针对的是房子的重建，但它们也同样很好地适用于基于计算机的系统和应用系统的再工程。

为了贯彻这些原则，我们使用图31-2所示的软件再工程过程模型，它定义了6类活动。在

某些情况下，这些活动以线性顺序出现，但并不总是这样。例如，有可能在文档重构开始前，必须先进行逆向工程（理解某程序的内部工作原理）。

在图中显示的再工程范型是一个循环模型。这意味着作为该范型的一部分的每个活动均可能被重复，对任意特定的循环，过程可以在任意一个活动之后终止。

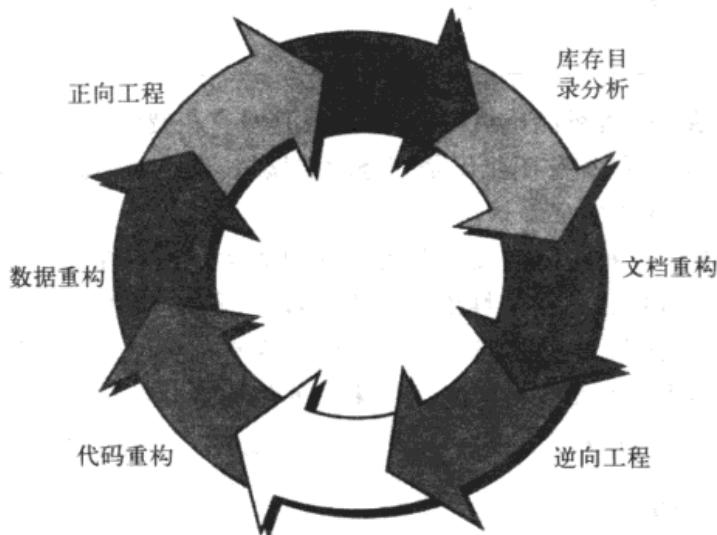


图31-2 软件再工程过程模型



如果时间及资源紧缺，可以考虑将Pareto原理应用到将要构造的软件中。将再工程过程应用到存在80%问题的20%的软件中。

875



只建立你需要的文档以理解软件，不超过一页。

876

库存目录分析 (*inventory analysis*)。每个软件组织应该保存所有应用系统的库存目录。该目录可能仅仅是一个电子表格模型，其中为每一个现行的应用系统提供了详细的描述（例如，规模、年限、业务重要程度）。按照业务重要程度、寿命、当前可维护性、以及其他本地重要性准则对这些信息排序，可以选出再工程的业务，然后为业务应用的再工程工作分配资源。

值得注意的是：应该对库存目录进行定期分析，应用问题的状况（如，业务重要程度）可能随时间发生变化，从而会使再工程的优先级发生变化。

文档重构 (*document restructuring*)。拙劣的文档是很多遗产系统的特点。但是，我们对此可做些什么呢？我们该如何选择？

1. 建立文档是非常耗费时间的。如果系统正常运作，我们将保持现状。在某些情况下，这是一个正确的做法，不可能为数百个计算机程序重新建立文档。如果一个程序是相对静止的，正在走向其使用寿命的终点，并且可能不会再经历什么变化，那么，让它保持现状。

2. 文档必须更新，但是我们只有有限的资源。我们将采取“使用时再建立文档”的方法。可能不需要对某应用问题重构全部文档，而是对系统中当前正在进行改变的那些部分建立完整的文档。随着时间的推移，将得到一组有用的相关文档。

3. 系统是业务关键的，而且必须完全地重构文档。即使是这种情况，也最好是设法将文档精简到最少。

上述每个选项均是可行的，软件组织必须对每种情形选择最适合的方法。

逆向工程 (*reverse engineering*)。术语逆向工程源于硬件领域，一个公司分解某项有竞争

力的硬件产品，以了解竞争者的设计和制造“秘密”。如果得到了竞争者的设计和制造规格说明，则这些秘密会很容易理解。但是，这些文档是别人专有的，对做逆向工程的公司来说是不可得到的。实际上，成功的逆向工程是通过检查产品的实际样本来推导出一个或多个关于产品的设计和制造规格说明。

软件的逆向工程是非常类似的。然而，在大多数情况下，要实施逆向工程的程序不是来自于竞争者，而是公司自己的软件（经常是很多年以前的）。由于未开发过相关的规格说明，需要理解的“秘密”模糊不清。因此，软件的逆向工程是分析程序、在高于源代码的抽象层次上表示程序的过程。逆向工程是一个设计恢复过程。逆向工程工具从现有的程序中抽取数据、体系结构和过程的设计信息。



代码重构 (code restructuring)。最常见的再工程（实际上，在这里使用术语“再工程”是有疑问的）类型是代码重构³。某些遗产系统具有相对可靠的程序体系结构，但是，个别模块的编码方式使得程序难于理解、测试和维护。在这样的情形下，可以对可疑模块内的代码进行重构。

为了完成该活动，用重构工具去分析源代码，将与结构化程序设计概念相违背的部分标注出来，然后对代码进行重构（此工作可以自动进行）；对生成的重构代码进行评审和测试，确保没有引入不规则的代码，并更新内部的代码文档。

数据重构 (data restructuring)。数据体系结构差的程序将难于进行适应性修改和增强。事实上，对很多应用系统来说，数据体系结构比源代码本身对程序的长期生存力的影响更大。

与抽象层次较低的代码重构不同，数据重构是一种全范围的再工程活动。在大多数情况下，数据重构开始于逆向工程活动。对当前的数据体系结构进行分解，并定义必要的数据模型（第9章），标识数据对象和属性，并对现有的数据结构进行质量评审。

当数据结构较差时（例如，普遍采用文件实现，而采用关系型方法将大大地简化处理），则应该对数据进行再工程。

由于数据体系结构对程序体系结构及其上的算法有很强的影响，所以对数据的改变总会导致体系结构或代码层的改变。

正向工程 (forward engineering)。在理想的情况下，可以使用自动的“再工程引擎”来重建应用，旧的程序被输入引擎，分析，重构，然后重新生成以展示软件质量最好的方面。短期内，这样的“引擎”还不可能出现，但是，CASE 厂商已经开发了一些工具，它们提供了针对特定应用领域（例如，用特定数据库系统实现的应用）的这些能力的有限子集。更重要的是，这些再工程工具正不断地变得越来越高级。

正向工程也称为革新或改造[CHI90]，不仅能够从现有软件恢复设计信息，而且还能够使用这些信息去改变或重构现有系统，以改善其整体质量。大多数情况下，实施了再工程的软件可以重新实现现有系统的功能，并且还能够加入新功能和（或）改善整体性能。

31.3 逆向工程

逆向工程幻想有一个“魔术槽”，我们将随便设计的、无文档的源程序放到该槽中，在另

³ 代码重构具有“重构”(refactoring)的某些成分，重构是一种重设计概念，在第4章介绍了这个概念，本书的其他地方也进行了讨论。

一端出来的就是计算机程序的完整设计描述（完整文档）。不幸的是，这样的魔术槽并不存在。逆向工程可以从源程序中抽取设计信息，但是，抽象的层次、文档的完备性、工具与分析人员协同工作的程度、过程的方向性却是高度可变的。

KEY POINT

必须讨论三个逆向工程问题：抽象层次、完备性、方向性。

逆向工程过程和用于实现该过程的工具的抽象层次是指可从源代码中抽取出来的设计信息的精密程度。理想情况，抽象层次应该尽可能高，即，逆向工程过程应该能够导出过程的设计表示（一种低层的抽象）；程序和数据结构信息（稍高层次的抽象）；对象模型、数据和控制流模型（一种相对高层的抽象）；UML类、状态及部署图（一种高层抽象）。随着抽象层次增高，软件工程师能够得到更有助于理解程序的信息。

878 逆向工程过程的完备性是指在某一抽象层次上提供信息的详细程度。在大多数情况下，随着抽象层次增高，完备性就降低。例如，给定源代码列表，得到一个完整的过程设计表示是比较容易的，也可以导出简单的设计表示，但要得到UML图或模型的完整集合却困难得多。

完备性的改善与做逆向工程的人员所完成的分析量成正比。交互性是指为了建立一个有效的逆向工程过程，人与自动工具“结合”的程度。在大多数情况下，随着抽象层次增高，交互性必须增加，而完备性将受到损害。

如果逆向工程过程的方向性是单向的，从源程序中抽取的所有信息都提供给软件工程师，然后他们可以在任何维护活动中使用这些信息。如果方向性是双向的，则信息被输入到再工程工具，以试图重构或重新生成旧程序。

逆向工程过程如图31-3所示，在逆向工程活动可以开始前，无结构的（“脏的”）源代码被重构（31.4.1节），使得它仅包含结构化程序设计结构⁴。这使得源代码容易阅读，并为所有后续的逆向工程活动奠定基础。

879 逆向工程的核心活动是抽取抽象。工程师必须评价旧程序，并从源代码（经常是无文档的）中抽取出被执行的处理、被应用的用户界面以及被使用的程序的数据结构或数据库的有意义的规格说明。

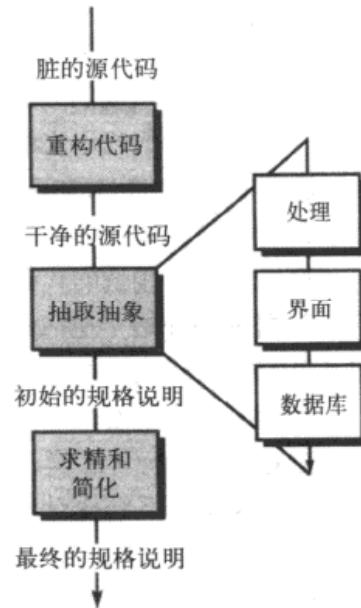


图31-3 逆向工程过程

31.3.1 数据的逆向工程

WebRef

“设计发现及程序理解”的有用资源可在www.sel.iit.nrc.ca/projects/dr/dr.html找到。

数据的逆向工程发生在不同的抽象层次，并且通常是第一项逆向工程任务。在程序层，作为整体再工程工作的一部分，必须对内部的数据结构进行逆向工程。在系统层，全局数据结构（如，文件、数据库）经常被再工程，使其符合新的数据库管理范型（如，从文件移向关系型或面向对象数据库系统）。当前，全局数据结构的逆向工程为引入新的系统范围的数据库奠定了基础。

内部数据结构。针对内部程序数据的逆向工程技术着重于对象的类定义。对程序代码进行检查，组合相关的程序变量以完成类的定义。在很多情况下，在代码中的数据组织标识了

⁴ 可以使用重构引擎对代码进行重构，它是一种重构源代码的工具。



数据结构中的看

似无关紧要的妥
协可能会在将来
导致潜在灾难性
的问题。例如，
想想Y2K问题。

抽象数据类型，例如，记录结构、文件、列表和其他数据结构通常给出了类的最初指示。

数据库结构。不考虑其逻辑组织及物理结构，数据库允许定义数据对象，并支持在对象间建立关系的方法。因此，当要将一个数据库模式再工程为另一个数据库模式时，需要理解现有对象及它们之间的关系。

为了对新数据模型实施再工程，首先要定义现有数据模型，可用下面的步骤[PRESS94]：(1) 构造一个初始的对象模型；(2) 确定候选键；(3) 精化实验性的类；(4) 定义一般化关系；(5) 找出关联关系（使用与CRC方法相似的技术）。一旦知道了在前面步骤中所定义的信息，则可以通过一系列转换[PRESS94]将旧的数据库结构映射到新的数据库结构。

31.3.2 处理的逆向工程

处理的逆向工程开始于试图理解并抽取源代码所表示的过程抽象。为了理解过程抽象，需要在不同的抽象级别（系统级、程序级、构件级、模式级和语句级）分析代码。

在进行更详细的逆向工程前，必须理解整个应用系统的整体功能。这项工作确定了进一步分析的范围，并对系统中应用间的互操作问题有了深入的理解。构成应用系统的每一个程序代表了在更高详细层次上的功能抽象，可以用结构图表示这些功能抽象间的交互作用；每个构件实现某种子功能，并表示某种定义的过程抽象，所以要对每个构件的处理进行描述。在某些情况下，系统、程序和构件的规格说明已经存在，若是这种情况，要对这些规格说明进行评审，确认是否与现有代码相符⁵。880

“有一种理解的激情，就像对音乐的激情一样。那种激情在孩子中间是常见的。但是，后来在大多数人中却消失了。”

——Albert Einstein

当考虑构件中的代码时，事情变得更复杂。工程师需找到表示通用过程模式的代码段。几乎在每个构件中，由一个代码段准备要处理的数据（在模块内），由另一个不同的代码段完成处理工作，然后再由另一个代码段准备从构件中输出的处理结果。在每个代码段中，我们可能遇到更小的模式，例如，数据确认和范围检查经常出现在为处理准备数据的代码段中。

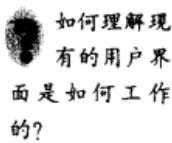
对大型系统，通常用半自动方法完成逆向工程。使用自动化工具帮助软件工程师理解现有代码的语义，然后将该过程的结果传递给重构和正向工程工具以完成再工程过程。

31.3.3 用户界面的逆向工程

高级图形用户界面（GUI）对于基于计算机的产品和各种类型的系统来说都是不可缺少的，因此，用户界面的重新开发已经成为最常见的再工程活动类型之一。但是，在重建用户界面之前，应该先进行逆向工程。

为了能够完全理解现有的用户界面，必须详细说明界面的结构和行为。Merlo 及其同事[MER93]提出了在用户界面的逆向工程开始前必须回答的三个基本问题：

⁵ 通常，在程序生命历史的早期编写的规格说明从未被更新过。随着代码被修改，代码不再与规格说明相符。



如何理解现有的用户界面是如何工作的?

- 界面必须处理的基本动作（例如，击键和点击鼠标）是什么？
- 系统对这些动作的行为反应的简要描述是什么？
- “替代者”意味着什么？或更确切地说，在这里，有哪些界面的等价概念是相关的？

对于头两个问题，行为建模符号（第8章）可以提供求解方法，创建行为模型所必需的信息多数可以通过观察现有界面的外部表现而得到，但是，还有一些信息必须从代码中抽取。

881

值得注意的是，一个替换的GUI可能并不是旧界面的精确镜像（实际上，它可能完全不同）。开发新的交互比喻通常是值得的，例如，一个旧的GUI要求用户提供比例因子（范围从1到10），用来放大或缩小一幅图像，再工程后的GUI可能使用一个滑杆及鼠标完成相同的功能。

SOFTWARE TOOLS

逆向工程

目的：帮助软件工程师理解复杂程序的内部设计结构。

机制：在大多数情况下，逆向工程工具接收源代码作为输入，然后产生多种结构、过程、数据及行为设计表示。

代表性工具⁶

Imagix 4D，由Imagix(www.imagix.com)开发，通过逆向工程及编写源代码的文档，“帮助软件开发者理解复杂的或用C及C++编写的遗产软件”。

Understand，由Scientific Toolworks, Inc.(www.scitools.com)开发，能够分析Ada、Fortran、C、C++及Java程序，“对于逆向工程师，能够自动产生文档，计算代码度量，并帮助你理解、导航及维护源代码”。

逆向工程工具的完整列表可以在<http://scgwiki.iam.unibe.ch:8080/SCG/370>找到。

31.4 重构

软件重构修改源代码和（或）数据，使软件适应未来的变化。通常，重构并不修改整体的程序体系结构，它倾向于关注个别模块的设计细节及定义模块中的局部数据结构。如果重构扩展到模块边界之外，并涉及软件体系结构，则重构变成了正向工程（31.5节）。

当某应用系统的基本体系结构比较好，但技术的内部细节需要修改，则需要对应用系统进行重构。当软件的大部分是有用的，仅仅需要对部分模块和数据进行扩展性修改时，则启动重构活动⁷。

31.4.1 代码重构

进行代码重构是为了生成与源程序具有相同功能、但具有更高质量的设计。通常代码重

⁶ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。大多数情况下，工具的名字由各自的开发者注册为商标。

⁷ 扩展性重构及重开发有时是很难区分的，两者均属于再工程。



虽然代码重构可以缓解与调试或小修改相关的问题，但是，它不是再工程。只有数据和体系结构被重构，才能够取得真正的收益。

构技术（例如，Warnier的逻辑简化技术[WAR74]）用布尔代数对程序逻辑进行建模，然后应用一系列变换规则来重构逻辑。其目标是采用“面条碗”式的代码，并推导出遵从结构化程序设计原理（第11章）的过程设计。882

建议将其他重构技术与再工程工具一同使用，资源交换图能够映射每个程序模块及其与其他模块间交换的资源（数据类型、过程和变量），通过创建资源流的表示，可以对程序体系结构重构，以达到模块间的最小耦合。

31.4.2 数据重构

在数据重构开始前，必须先进行称为源代码分析的逆向工程活动。评估所有包含数据定义、文件描述、I/O以及接口描述的程序设计语言语句，目的是抽取数据项和对象，获取关于数据流的信息，以及理解现有的已实现的数据结构。有时称该项活动为数据分析[RIC89]。

一旦完成了数据分析，就可以开始数据重设计。其最简单的形式为：通过数据记录标准化步骤明确数据定义，从而使现有数据结构或文件格式中的数据项名或物理记录格式取得一致。另一种重设计形式称为数据名合理化，这种重设计形式能够保证所有数据命名约定符合本地标准，并且当数据在系统中流动时可以忽略别名。

当重构超出标准化和合理化的范畴时，要对现有数据结构进行物理修改以使数据设计更为有效。这可能意味着从一种文件格式到另一种文件格式的转换，或在某些情况下，意味着从一种数据库类型到另一种数据库类型的转换。

SOFTWARE TOOLS

软件重构

目的：重构工具的目标是将旧的非结构化的软件转化为现代程序设计语言及设计结构。

机制：通常将源代码输入，然后将其变换为更好的结构化程序。在某些情况下，这种变换发生在同一种程序设计语言中。在有些情况下，能够将一种较老的程序设计语言变换为一种更加现代的语言。

代表性工具⁸

DMS Software Reengineering Toolkit，由Semantic Design(www.semantics.com)开发，提供针对COBOL、C/C++、Java、Fortran 90及VHDL的多种重构能力。

FORSYS，由Simulog (www.simulog.fr) 开发，能够分析和变换用Fortran语言编写的程序。

Function Encapsulation Tool，由Wayne State University (www.cs.wayne.edu/~vip/RefactoringTools/) 开发，能够将旧的C程序重构为C++程序。

plusFORT，由Polyhedron (www.polyhedron.com) 开发，是一套Fortran工具，能够将设计质量低的Fortran程序重构为现代的Fortran或C标准程序。

⁸ 这里记录的工具并不代表本书支持这些工具，而只是此类工具的例子。大多数情况下，工具的名字由各自的开发者注册为商标。

31.5 正向工程

程序的控制流在图形上相当于一碗面条，假设某程序有一个包含2 000行语句的“模块”，在290 000行源代码中几乎没有有意义的注释行，并且没有其他文档，如果该程序必须被修改，以适应用户需求的变化，我们有下列选择：

当我们面临某设计和实现均非常差的程序时，我们有哪些选择？

1. 努力地不断修改，与隐含的设计和源代码搏斗，以实现必要的修改。
2. 试图去理解程序的更多的内部工作，努力使修改更有效。
3. 重设计、重编码并测试该软件需要修改的部分，对所有修改过的片段应用软件工程方法。
4. 完全地重设计、重编码并测试该程序，使用再工程工具辅助我们理解现有的设计。

这里没有单一的“正确”选择。当前的形势可能要求第一种选择，即使更期望其他选择。

不要坐等收到维护请求，开发或维护组织应该使用库存目录分析的结果挑选一个程序，该程序：(1) 将在预先确定的年限内继续使用，(2) 当前的使用是成功的，(3) 很可能会在不远的将来做较大的修改或增强。这样，我们就可以应用上面的选择2、3或4。

这种预防性维护方法是Miller[MIL81]在“结构化革新”这个标题下首先提出的，他把这个概念定义为“将今天的方法学应用到昨天的系统，以支持明天的需求”。

乍一看，重新开发大型程序的已有的可用版本似乎是相当浪费的，在下判断之前，应该考虑如下几点：



再工程在很大程度上类似于清洁你的牙齿。你可以想出上千种理由来拖延它，你可以通过延迟一会儿而逃离，但是，最终你的逃离策略会反过来困扰你。

884

1. 维护一行源代码的成本可能是该行代码初始开发成本的20~40倍。
2. 采用现代设计概念重新设计软件体系结构（程序和/或数据结构）可以大大地便于未来的维护。
3. 因为软件的原型已经存在，开发生产率将远高于平均水平。
4. 现在用户已对该软件有使用经验，因此，可以很容易地确定新的需求和变更的方向。
5. 再工程的自动化工具可以使部分工作自动化。
6. 预防性维护的完成将产生完整的软件配置（文档、程序和数据）。

当软件开发组织将软件作为产品销售时，预防性维护体现在程序的“新版本”中。一个大的内部软件开发者（例如，一个大型消费产品公司的业务系统软件开发小组）可能在其责任范围内有500~2000个产品程序，可以根据其重要程度对这些程序进行优先级排序，然后对所选出的要进行预防性维护的程序进行评审。

正向工程过程应用软件工程的原理、概念和方法来重建现有的应用。在大多数情况下，正向工程并不仅仅是创建某旧程序的一个现代等价物，而是将新的用户和技术需求集成到再工程中，重新开发的程序扩展了原有应用系统的能力。

31.5.1 客户/服务器体系结构的正向工程

在过去的几十年中，为了适应客户/服务器（包括WebApp）体系结构，已对很多大型机应用系统实施了再工程。实际上是将集中式的计算资源（包括软件）分布到很多客户平台上。虽然可以设计出各种不同的分布式环境，但是可以实施再工程使之转换为客户/服务器（C/S）

体系结构的典型的大型机应用系统应具有以下特征：

- 应用功能可以迁移到每个客户计算机。
- 在客户端实现新的GUI界面。
- 数据库功能被分配给服务器。
- 特殊的功能（例如，计算密集型的分析）可以保留在服务器端。
- 必须在客户端和服务器端同时建立新的通信、安全、归档和控制需求。



在某些情况下，向C/S体系结构的迁移不应该作为再工程项目来进行，而应该作为新的开发项目来进行。只有在旧系统的功能被集成到新系统的体系结构中时，才考虑再工程。

值得注意的是，从大型机到C/S计算模式的迁移需要同时进行业务再工程和软件再工程，另外，还应该建立“企业网络基础设施”[JAY94]。

针对C/S应用系统的再工程一般是从对业务环境（包括现有的大型机环境）的彻底分析开始的。可以确定三个抽象层。数据库层是客户/服务器体系结构的基础，并且管理来自客户应用的事务和查询，而这些事务和查询必须被控制在一组业务规则（由现有的或再工程后的业务过程所定义）范围内。客户应用系统提供面向用户的目标功能。

在对数据库层进行重新设计之前，必须首先对现有数据库管理系统的功能和现有数据库的数据体系结构进行逆向工程。在某些情形下，需要创建新的数据模型（第8章）。不管是哪一种情况，都需要对C/S数据库进行再工程，以保证：能够以一致的方式处理事务，所有更新只能被授权的用户完成，能够强制执行核心业务规则（例如，在某厂商记录被删除前，服务器保证没有与该厂商有关的应付款账号、合同或通信存在），能够实现高效查询，以及能够建立完善的归档能力。

885

业务规则层表示同时驻留在客户端和服务器端的软件，该软件执行控制和协调任务，以保证在客户应用和数据库间的事务和查询符合已建立的业务过程。

客户应用层实现特定的最终用户群所需要的业务功能，在很多场合，大型机应用系统可以被分割为一组小的、再工程后的桌面应用系统，桌面应用系统之间的通信（当必要时）由业务规则层控制。

C/S软件设计与再工程的广泛讨论最好留给专门讨论该主题的书籍，感兴趣的读者可参考[VAN02]、[COU00]及[ORF99]。

31.5.2 面向对对象体系结构的正向工程

面向对象软件工程已成为很多软件组织选择的开发范型。但是，对于使用传统方法开发的现有的应用系统该怎么办呢？在某些情况下，应该保持这些应用系统的现状，而在另一些情况下，旧的应用系统必须实施再工程，使得它们能够容易地集成到大型的、面向对象的系统中。

将传统的软件再工程为面向对象的实现采用了很多本书第二部分讨论的相同的技术。首先，要将现有的软件进行逆向工程，以便建立适当的数据、功能和行为模型。如果实施再工程的系统扩展了原应用系统的功能或行为，则还要创建相应的用例（第7、8章）。然后，联合使用在逆向工程中创建的数据模型与CRC建模技术（第8章），以奠定类定义的基础。最后，定义类层次、对象—关系模型、对象—行为模型以及子系统，并开始面向对象的设计。

随着面向对象的正向工程从分析进展到设计，可启用CBSE过程模型（第30章）。如果旧的应用系统所在的领域已经存在很多面向对象的应用，则很可能已存在一个健壮的构件库，

可以在正向工程中使用它们。

对那些必须从头开发的类，有可能复用现有的传统应用系统的算法和数据结构，但是，
886 必须重新设计这些算法和数据结构，以符合面向对象的体系结构。

31.5.3 用户界面的正向工程

随着应用系统从大型机迁移到桌面系统，用户不再愿意忍受不可思议的、基于字符的用户界面。事实上，从大型机到客户/服务器计算模式的变迁中，所有工作量中的很大一部分花费在客户应用系统的用户界面的再工程。

Merlo及其同事[MER95]提出了下面的用户界面再工程的模型：



用户界面再工程
的步骤是什么？

WebRef

一本300多页的再工程模式（作为FAMOOS ESPRIT项目的一部分开发的）手册可以从
[www.i.am.
unibe.ch/~scg/
Archive/famoos/
patterns/index3.html](http://www.i.am.unibe.ch/~scg/Archive/famoos/patterns/index3.html)下载。

1. 理解原界面及其和应用系统的其余部分间交换的数据。目的是理解其他的程序元素如何与实现界面的现有代码进行交互。如果开发新的GUI，则在GUI和其他程序间交换的数据必须与当前基于字符的界面和程序间交换的数据相一致。

2. 将现有界面蕴含的行为重新建模为在GUI环境内的一系列有意义的抽象。虽然交互模式可能有本质的不同，但旧界面和新界面所展示的业务行为（当从使用场景的角度考虑时）必须保持相同。一个重新设计的界面必须仍允许用户展示合适的业务行为，例如，当进行数据库查询时，旧界面可能需要一长串的基于文本的命令来详细描述查询，而再工程后的GUI可能将查询简化为一个简短的鼠标动作序列，但是，查询的意图和内容保持不变。

3. 引入使交互模式更有效的改进。研究现有界面在功效方面的弱点，并在新的GUI设计中进行改正。

4. 构造并集成新的GUI。类库和自动化工具可以大大地减少构造GUI所需的工作量，然而，与现有应用软件的集成可能更费时间。必须小心行事，以保证GUI不会将不利的副作用传播到应用系统的其余部分。

“现在你可能付出很少，而后来你可能付出很多。”

——Sign in an auto dealership suggesting a tune up

31.6 再工程经济学

在一个完美的世界中，应该立即停止使用每一个不可维护的程序，而由运用现代软件工程实践开发的高质量的、再工程后的应用系统所替代。但是，我们生活在一个资源有限的世界，再工程要消耗可能用于其他业务目的的资源，因此，一个组织在试图对现有应用系统实施再工程之前，应该进行成本-效益分析。

Sneed[SNE95]提出了再工程的成本-效益分析模型，其中定义了9个参数：

P_1 = 某应用系统的当前年度维护成本

P_2 = 某应用系统的当前年度运作成本

P_3 = 某应用系统的当前年度业务价值

P_4 = 再工程后的预期年度维护成本

P_5 = 再工程后的预期年度运作成本

P_6 = 再工程后的预期年度业务价值

P_7 = 估计的再工程成本

P_8 = 估计的再工程日程

P_9 = 再工程风险因子 ($P_9 = 1.0$ 为额定值)

L = 期望的系统生命期

与某候选应用系统（即，未执行再工程）的持续维护相关的成本可以定义为：

$$C_{\text{maint}} = [P_3 - (P_1 + P_2)] \times L \quad (31-1)$$

与再工程相关的成本用下面的关系定义：

$$C_{\text{reeng}} = [P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9)] \quad (31-2)$$

使用式 (31-1) 和式 (31-2) 中表示的成本，再工程的整体效益计算如下：

$$\text{成本效益} = C_{\text{reeng}} - C_{\text{maint}} \quad (31-3)$$

可以对所有在库存目录分析 (31.2.2节) 中标识的高优先级应用系统进行上述表示的成本-效益分析，那些显示最高成本-效益的应用系统可以作为再工程对象，而其他应用系统的再工程可以推迟到有足够资源时进行。

31.7 小结

再工程发生在两个不同的抽象层次。在业务层次，再工程着重于业务过程，目的是改变业务过程以改善在某业务领域的竞争力；在软件层次，再工程检查信息系统和应用系统，目的是对它们进行重构以提高质量。

业务过程再工程 (BPR) 定义业务目标，识别并评估现有业务过程（在定义的目标范围内），详细描述并设计修订的过程，并在业务中对它们进行原型化、精化和实例化。BPR的关注点扩展到软件之外，BPR的结果经常能够获得若干方法，这些方法使信息技术能够更好地支持业务。[888]

软件再工程包括一系列的活动：库存目录分析、文档重构、逆向工程、程序和数据重构、正向工程。这些活动的目的是创建现有程序的更高质量和更好维护的版本——将在21世纪具有良好生命力的程序。

库存目录分析使一个组织能够系统地评估每个应用系统，目的是确定再工程的候选者；文档重构能够创建一个文档框架，这对于应用系统的长期支持是必需的；逆向工程是分析程序的过程，试图从中抽取出数据、体系结构及过程的设计信息；最后，正向工程使用现代软件工程方法和在逆向工程中获得的信息对程序进行重构。

可以定量地确定再工程的成本-效益，现状的成本（即，和某现有应用系统的不断的支撑与维护相关的成本）与预期的再工程成本及维护成本的减少进行比较，对于几乎所有生命周期长且当前的可维护性差的程序，再工程均代表了一种成本合算的业务策略。

参考文献

- [CAN72] Canning, R., "The Maintenance 'Iceberg,'" *EDP Analyzer*, vol. 10, no. 10, October 1972.
- [CAS88] "Case Tools for Reverse Engineering," *CASE Outlook*, CASE Consulting Group, vol. 2, no.

- 2, 1988, pp. 1–15.
- [CHI90] Chikofsky, E. J., and J. H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, January 1990, pp. 13–17.
- [COU00] Coulouris, G., J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 3rd ed., Addison-Wesley, 2000.
- [DAV90] Davenport, T. H., and J. E. Young, "The New Industrial Engineering: Information Technology and Business Process Redesign," *Sloan Management Review*, Summer 1990, pp. 11–27.
- [DEM95] DeMarco, T., "Lean and Mean," *IEEE Software*, November 1995, pp. 101–102.
- [HAM90] Hammer, M., "Reengineer Work: Don't Automate, Obliterate," *Harvard Business Review*, July–August 1990, pp. 104–112.
- [HAN93] Manna, M., "Maintenance Burden Begging for a Remedy," *Datamation*, April 1993, pp. 53–63.
- [JAY94] Jaychandra, Y., *Re-engineering the Networked Enterprise*, McGraw-Hill, 1994.
- [MER93] Merlo, E., et al., "Reverse Engineering of User Interfaces," *Proc. Working Conference on Reverse Engineering*, IEEE, Baltimore, May 1993, pp. 171–178.
- [MER95] Merlo, E., et al., "Reengineering User Interfaces," *IEEE Software*, January 1995, pp. 64–73.
- [MIL81] Miller, J., in *Techniques of Program and System Maintenance*, (G. Parikh, ed.) Winthrop Publishers, 1981.
- [ORF99] Orfali, R., D. Harkey, and J. Edwards, *Client/Server Survival Guide*, 3rd ed., Wiley, 1999.
- [OSB90] Osborne, W. M., and E. J. Chikofsky, "Fitting Pieces to the Maintenance Puzzle," *IEEE Software*, January 1990, pp. 10–11.
- [PRE94] Premerlani, W. J., and M. R. Blaha, "An Approach for Reverse Engineering of Relational Databases," *CACM*, vol. 37, no. 5, May 1994, pp. 42–49.
- [RIC89] Ricketts, J. A., J. C. DelMonaco, and M. W. Weeks, "Data Reengineering for Application Systems," *Proc. Conf. Software Maintenance—1989*, IEEE, 1989, pp. 174–179.
- [SNE95] Sneed, H., "Planning the Reengineering of Legacy Systems," *IEEE Software*, January 1995, pp. 24–25.
- [STE93] Stewart, T. A., "Reengineering: The Hot New Managing Tool," *Fortune*, August 23, 1993, pp. 41–48.
- [SWA76] Swanson, E. B., "The Dimensions of Maintenance," *Proc. Second Intl. Conf. Software Engineering*, IEEE, October 1976, pp. 492–497.
- [VAN02] Van Steen, M., and A. Tanenbaum, *Distributed Systems: Principles and Paradigms*, Prentice-Hall, 2002.
- [WAR74] Warnier, J. D., *Logical Construction of Programs*, Van Nostrand-Reinhold, 1974.

889

习题与思考题

- 31.1 研究文献和/或Internet资源，查找一篇或多篇讨论从大型机到客户/服务器结构的再工程案例研究的文章，给出概要介绍。
- 31.2 如何在31.6节给出的成本–效益模型中通过 P_7 确定 P_4 ？
- 31.3 老师从班上每个人在本课程中开发的程序中选择一个，随机地将你的程序和其他人的程序交换，不对该程序进行解释或走查。现在，对你所接收的程序实现某些改进（由老师指定）。
 - a. 完成所有软件工程任务，包括粗略的走查（但不能和程序的作者交流）。
 - b. 对测试中遇到的所有错误仔细跟踪。
 - c. 在班上讨论你的经验。
- 31.4 使用通过Internet获得的信息，向班级介绍三种逆向工程工具的特点。
- 31.5 探讨在SEPA Web站点上列出的库存目录分析检查表，并试图开发一个定量的软件评价系统，可以将其应用到现有程序，并试图从中挑选出再工程的候选程序。你的系统应

该扩展到31.6节讲述的经济分析。

- 31.6 某些人相信人工智能技术将增加逆向工程过程的抽象层次，对此专题（即，人工智能在逆向工程中的使用）进行研究，并撰写一篇支持此论点的简短论文。
- 31.7 提出一种对纸、墨或传统的电子文档的替代物，它可作为文档重构的基础。（提示：考虑新的能用于交流软件目的的描述技术。）
- 31.8 考虑你在过去五年中从事过的任何工作，描述你在其中工作的业务过程。使用在31.1.3节中描述的BPR模型建议修改该过程以提高其效率。
- 31.9 为什么当抽象层次增加时，完备性更难于达到？
- 31.10 在重构和正向工程之间存在细微的不同，这种不同是指什么？
- 31.11 对业务过程再工程的功效做些研究，给出该方法的正面及反面论据。
- 31.12 如果完备性增加，为什么交互性也必须增加？

[890]

推荐读物与阅读信息

和很多商业社会的热点话题一样，围绕业务过程再工程的大肆宣传已经让路给对该主题的更实际的见解。Hammer和Champy的畅销书（《Reengineering the Corporation, revised edition》，HarperBusiness, 2001）促成了早期的兴趣。后来，Hammer（《Beyond Reengineering: How the Process-Centered Organization Is Changing Our Work and Our Lives》，HarperCollins, 1997）通过关注“过程为中心”的问题，精化了他的观点。

由Smith及Fingar（《Business Process Management (BMP): The Third Wave》，MeghanKiffer Press, 2003）、Jacka及Keller（《Business Process Mapping: Improving Customer Satisfaction》，Wiley, 2001）、Sharp及McDermott（《Workflow Modeling》，Artech House, 2001）、Andersen（《Business Process Improvement Toolbox》，American Society for Quality, 1999）及Harrington等（《Business Process Improvement Workbook》，McGraw-Hill, 1997）所著的书籍给出了BPR的案例研究及详细指南。

Feldmann（《The Practical Guide to Business Process Reengineering Using IDEF0》，Dorset House, 1998）讨论了有助于BPR建模的符号体系，Berztiss（《Software Methods for Business Reengineering》，Springer, 1996）及Spurr等（《Software Assistance for Business Reengineering》，Wiley, 1994）讨论了支持BPR的工具和技术。

Secord和他的同事（《Modernizing Legacy Systems》，Addison-Wesley, 2003）、Ulrich（《Legacy Systems: Transformation Strategies》，Prentice-Hall, 2002）、Valenti（《Successful Software Reengineering》，IRM Press, 2002）及Rada（《Reengineering Software: How to Reuse Programming to Build New, State-of-the-Art Software》，Fitzroy Dearborn Publishers, 1999）关注于技术层的再工程策略及实践，Miller（《Reengineering Legacy Software Systems》，Digital Press, 1998）“提供了保持应用系统与业务策略及技术改变同步的框架”，Umar（《Application (Re) engineering: Building Web-Based Applications and Dealing with Legacies》，Prentice-Hall, 1997）为希望将遗产系统转换为基于Web环境的组织提供了有价值的指南，Cook（《Building Enterprise Information Architectures: Reengineering Information Systems》，Prentice-Hall, 1996）讨论了BPR和信息技术之间的桥接，Aiken（《Data Reverse Engineering》，

McGraw-Hill, 1996) 讨论了如何修复、重新组织和复用管理数据, Arnold (《Software Reengineering》, IEEE Computer Society Press, 1993) 出版了一本关注软件再工程技术的早期重要论文的优秀文选。

大量关于软件再工程的信息资源可从Internet获得, 最新的WWW资源列表可在SEPA Web [891] 站点<http://www.mhhe.com/pressman>找到。



第32章 未来之路

要点浏览

概念：未来从来就不容易预测——博学者、领袖人物及行业专家都不能抗拒它。未来的路上散落着令人兴奋的新技术所留下的遗迹，这些新技术从未真正成功（尽管对它们进行了大肆宣传）。未来之路通常由更加现代的技术所形成，这些现代技术从某种程度上改变了未来之路的方向及宽度。因此，我们不去预测未来，而是讨论你需要考虑的一些事项，从而了解软件及软件工程在未来几年中将如何变化。

人员：每一个人。

重要性：为什么古代的帝王雇用占卜者？为什么大型跨国公司雇用咨询公司及智囊团来做预测？为什么有相当比例

的公众阅读星座运势方面的书籍？因为我们想知道正在到来的事情，以使自己做好准备。

步骤：没有预测未来之路的公式。只能通过收集数据、组织数据来提供有用信息，通过考察微妙的联系来抽取知识，并从这些知识中找出预示未来状况的可能发生的事情。

工作产品：可能正确、也可能不正确的一幅近期视图。

质量保证措施：对未来之路的预测是一种艺术，而不是科学。事实上，对未来的严肃预言很少是完全正确或完全错误的（感谢上帝！世界末日的预言除外！）。我们寻找趋势，并尽可能地提前对趋势进行推断，但只有时间过去之后，我们才能够评估推断的正确性。

关键概念
数据
职业道德规范
信息
知识
人
过程
变化范围
软件再论
技术趋势

在前面的31章中，我们探讨了软件工程过程，讨论了管理规程和技术方法、基本原理和专门技术、面向人的活动和适合于自动化的任务、用纸和笔的符号表示，以及软件工具。我们认为，对质量的测量、规定及高度重视将得到满足客户需要的、可靠的、可维护的、更好的软件。但是，我们从未允诺软件工程是万能的。

当我们走向新世纪之时，软件和系统技术对于构造基于计算机的系统的软件专业人员和公司仍是一种挑战。虽然Max Hopper [HOP90]是怀着对21世纪的展望写下了下面这些话的，但却准确地描述了现在的情形：

因为信息技术的变化正变得如此快速和不可遏制，并且落后的后果是如此的不可挽回，因此，公司要么掌握技术，要么倒闭……将它想象为技术“踏车”，公司不得不更加努力地运转它，以站稳位置。

软件工程技术方面的变化确实“快速和不可遏制”，但同时进展经常又非常慢。在决定采用一种新方法（或一种新工具）的时候，为了理解其应用，需要进行培训，然后将技术引入到软件开发文化中，此时会伴随出现某些新东西（以及更好的东西），而且过程也是重新开始。

在本章中，我们谈未来之路。我们的目的不是去探讨每个有希望的研究领域，也不是去凝视“水晶球”预言未来。我们将探讨变化的范围，以及变化本身将如何在未来几年影响软件过程。

32.1 再论软件的重要性

可以从很多方面来叙述计算机软件的重要性。在第1章中，软件被描述为区分器。交付的软件可区分为产品、系统和服务，它提供了市场竞争力。但是，软件并不仅仅是区分器，作为软件的程序、文档和数据可以辅助任何个人、商业或政府获取最重要的日用品——信息。Pressman和Herron [PRE91]是这样描述软件的：

计算机软件是对现代社会的几乎各个方面均有重要影响的为数不多的关键技术之一……它是使商业、产业和政府自动化的机制，是传递新技术的媒介，是捕获有价值的专家意见供其他人使用的方法，是区分一个公司及其竞争对手的产品的手段，是反映企业集体知识的窗口。软件对商业的几乎各个方面均是关键的。但是，在许多方面软件也是一种隐藏技术。当我们去工作、购买商品、进银行、打电话、看医生，或从事任何反映现代生活的上百种日常活动时，都会遇到软件（通常我们没有意识到）。

软件的普及使我们得出一个简单的结论：一旦某项技术具有广泛的影响（可以拯救生命或威胁生命、构造业务或破坏业务、协助政府领导或误导他们），就必须谨慎地对待它。

“预测是很困难的，尤其是对未来的预测。”

——Mark Twain

32.2 变化的范围

在过去50年中，计算领域的变化是由硬科学（物理、化学、材料科学）及工程学的进展所驱动的，这种趋势将持续到21世纪的前25年。新技术的影响是无所不在的——如在通信、能源、保健、交通、娱乐、经济、制造业及战争方面，这里列举的只是少数。
[893]

INFO

技术观察

《PC杂志》[PCM03]的编辑们每年都要撰写关于“未来技术”方面的文章，在这些文章中，“通过对所有谈话（有很多）进行[分类]，挑选出将来最有前途的20项技术。”所选出的技术涉及从保健到战争的所有领域。然而，值得注意的是：软件及软件工程在每一领域中都扮演了重要的角色，它们要么作为使能技术，要么作为领域不可分割的一部分。下面是所记录的有代表性的技术：

纳米碳管——具有微小的石墨状结构，可作为导线将信号从一点传到另一点，也可作为晶体管，通过信号的变化来存储信息。这些设备可用于开发更小、更快、低能耗及低价格的电子产品（例如，微处理器、内存、显示器）。

生物传感器——外部或可植入的微电子传感器已经可以用于探测任何事物：从空气中的化学成分到心脏病人的血液级别。随着这些传感器变得越来越高级，它们可被植入到病

人体内监视与健康相关的各种情形，或者附着在士兵的制服上监视生化武器的存在。

OLED显示器——OLED “使用基于碳的设计分子，当电流流过时分子就会发光。将很多分子合在一起，就能够得到一个超薄的、质量极好的显示器——不需要消耗能量的背景照明。” [PCM03]，这就使得超薄显示器可以被卷起或折叠、被喷射到弯曲的表面，或适合于特定的环境。

网格计算——这种技术（目前已存在）可以从网络上的每一台机器选择上亿空闲的CPU周期来构造一个网络，并且能够在没有专用超级计算机的情况下完成非常复杂的计算任务。一个真实的网格例子中包含了450万台计算机，见<http://setiathome.berkeley.edu/>。

认知机——机器人领域中的“最高荣誉”就是开发一个可以感知自身所处环境的机器，它能够“拾起球杆，对任何时候发生变化的情形都能够做出反应，并且能够很自然地与人进行交互” [PCM03]。认知机还处于发展的早期阶段，但其潜能（如果什么时候能够成功的话）是巨大的。

WebRef

对未来技术的预测及其他资料，请看www.futurefacing.com。

长期以来，计算领域革命性进展的驱动力可能是“软科学”——人的心理学、社会学、哲学、人类学等等。从这些学科所得到的计算技术，在酝酿阶段是非常难于预测的，但已经开始有了早期的影响。例如，用户群体（人类学结构）就是点对点网络的一个分支。

软科学的影响可能有助于形成硬科学中的计算机研究方向，例如，“未来计算机”的设计受大脑生理学的影响比受传统的微电子学的影响可能更大。

在未来十年，软件工程的变化将同时受到来自四个方面的影响：(1) 干工作的人；(2) 他们使用的过程；(3) 信息的性质；(4) 基本的计算技术。在下面几节中，将对每个因素（人、过程、信息和技术）详细论述。

894

32.3 人及其构造系统的方式

高科技系统需要的软件每年都变得越来越复杂，最后形成的程序规模也成比例地增长。如果不是因为“如果程序规模扩大，必须为此程序投入的人力也要增加”这样一个简单事实的话，“平均”程序规模的快速增长不会给我们带来太多问题。

经验表明，如果一个软件项目团队的人数增加，则项目团队的整体生产率可能会下降。针对这个问题的一个解决方法就是增加软件工程项目团队的数量，从而将人员划分为单独的工作小组。然而，随着软件工程项目团队数量的增加，他们之间的交流也会变得困难和费时，就像个体之间的交流一样。更糟糕的是，交流（在个体间或项目团队间）趋向于低效——即用了太多的时间，却只能传递很少的信息内容，而且，通常情况是将重要的信息“分裂为破碎的片断”。

“未来的冲击 [是] 消除我们个体感受到的压力和迷惑，方法是使他们在极短的时间内遭遇很多变化。”

——Alvin Toffler

如果软件工程界有效地解决了交流困难，软件工程师的未来之路必定会涉及个体之间及项目团队之间相互交流方式的根本性改变。电子邮件、网站和网络视频会议是目前常见的能够将大量人员连接到一个信息网络的机制。当然，不能过分强调这些工具在软件工程范畴内

的重要性。通过有效的电子邮件或实时消息系统，在纽约的软件工程师遇到的问题可以通过在东京的同事的帮助而解决。在现实中，集中的聊天会议和专业的新闻组已经形成知识库，它能够将大量技术人员的集体智慧集中起来，共同解决技术问题或管理问题。

视频使交流个性化。它最大的好处是使得在不同地方（或不同大陆）的同事可以定期地“见面”。视频还有另一个好处，可以将它作为软件的知识库，并用于培训项目中的新手。

“对数字技术的恰当的艺术处理是将它放置到一个具有任何事物（常常是人）的新窗口中，并以热情、智慧、勇敢及快乐的心态去使用它。” ——Ralph Lombreglia

895



越来越多的“非程序员”正在构建他们自己的（小的）应用系统，这种趋势会在未来加速。这些“非专业人员”会应用本书所讨论的技术吗？或许不用，但是他们应该采用敏捷软件工程的思想，即使他们没有这种实践经验。

通过最大限度地扩展软件工具的能力，智能代理的进展也将改变软件工程师的工作模式。智能代理通过使用专门的领域知识来交叉检查工程工作产品、充当秘书职责、进行直接调查、协调人与人之间的交流，从而增强软件工程师的能力。

最后，知识获取正以深刻的方式在改变。在Internet上，软件工程师能够订阅新闻组了解自己关心的技术领域。张贴在新闻组上的问题可以得到来自全球范围内的其他感兴趣团体的出其不意的回应。万维网给软件工程师提供了世界上最大的软件工程方面的研究论文和报告、指南、评论及参考资料的图书馆。

如果过去的历史是一面镜子，就可以公正地说“人类本身并没有改变”。但是，他们交流的方式、工作的环境、获取知识的方式、使用的方法和工具、应用的规则以及软件开发的全部文化都将发生重大而深刻的改变。

32.4 新的软件工程过程

将软件工程实践的前20年描述为“线性思维”的时代是合理的。在传统生命周期模型的支持下，软件工程是线性的活动，应用一系列有序的步骤去努力解决复杂的问题。然而，软件开发的线性方法违反了大多数系统的实际构造方式。在现实中，复杂系统迭代地、甚至增量地演化。为此，软件工程界中大多数正转向软件开发的敏捷、增量模型。

敏捷、增量过程模型认识到：不确定性支配了大多数项目，即时间经常是紧张得令人难以忍受。当一个完整的产品不可能在分配的时间内完成时，迭代提供了交付部分产品的能力。演化模型强调对增量式工作产品、风险分析、计划及计划修订以及客户反馈的需要。在许多情况下，软件团队应用“敏捷宣言”（第4章），此宣言中强调“个体及其与过程和工具的交互作用；有全面文档的工作软件；合同协商基础上的用户协作，并且有计划地对变更做出反应”[BEC01]。

“将明天的工作做好的最好前提是今天将工作做好。” ——Elbert Hubbard

896

对象技术与基于构件的软件工程（第30章）相结合是增量及演化过程模型趋势的自然结果，两者都将对软件开发生产率和产品质量产生深远的影响。构件复用提供了直接的且令人信服的利益。当复用与CASE工具（支持应用原型的开发）相结合时，程序增量的构造远比使用传统方法更快。原型方法使客户参与到过程之中，因此，客户和用户有可能更多地参与软

件开发，这本身又将取得更高的最终用户满意度和更好的总体软件质量。

网络及多媒体技术的迅速发展（例如，在过去十年中WebApp以指数级增长）正在改变软件工程过程及其参与者。此外，我们还有敏捷、增量模式，除了传统软件工程所关注的问题，敏捷、增量模式还强调直接、安全及美学。现代软件团队（例如，Web工程团队）通常将技术专家与内容专家（例如，艺术家、音乐家、电视录像制作人）相结合，为数量庞大且不可预知的用户群体构建信息源。由这些技术开发出的软件已经引起了经济及文化的根本改变。虽然，本书中讨论的基本概念及原则是适用的，但软件工程过程也必须与其相适应。

32.5 表示信息的新模式

在计算机的历史上，用于描述商业界软件开发工作的术语发生了一些微妙的变迁。40年前，术语“数据处理”是描述计算机在商业中的应用的习惯用语，今天，数据处理已经让位给另一个短语——“信息技术”，它与“数据处理”所指的事情是相同的，但在关注点上有微妙的偏移，它强调的重点不仅仅是大量数据的处理，而是从这些数据中抽取有意义的信息。显然，这是长远的目标。然而，术语上的改变反应了管理哲学上的更重要的变化。

当我们今天讨论软件应用时，“数据”和“信息”这两个词反复地出现，我们也在某些人工智能应用中遇到词语“知识”，但是它的使用相对较少。事实上，没有人在计算机软件应用的范畴内讨论智慧。

数据是未加工的信息——是事实集合，必须经过处理才具有意义；信息是在给定的环境下通过相互关联的事实而得到的；知识是将某个环境中所获得的信息与在另外不同环境中获得的信息相关联；而智慧是从完全不同的知识中推导出的一般性原理。图32-1以图表形式表示了这四个“信息”的视图。

当前，所构造的绝大多数软件都是用于处理数据或信息，软件工程师还同样关心处理知识的系统¹。知识是二维的。针对一系列相关和不相关的主题而收集的信息被联结在一起，形成一个事实体系我们称之为知识，其中的关键是这样一种能力，即应该如何关联来自一系列不同源（它们之间可能没有明显的联系）的信息，并将它们组合在一起为我们提供某些独特收益的能力。

[897]

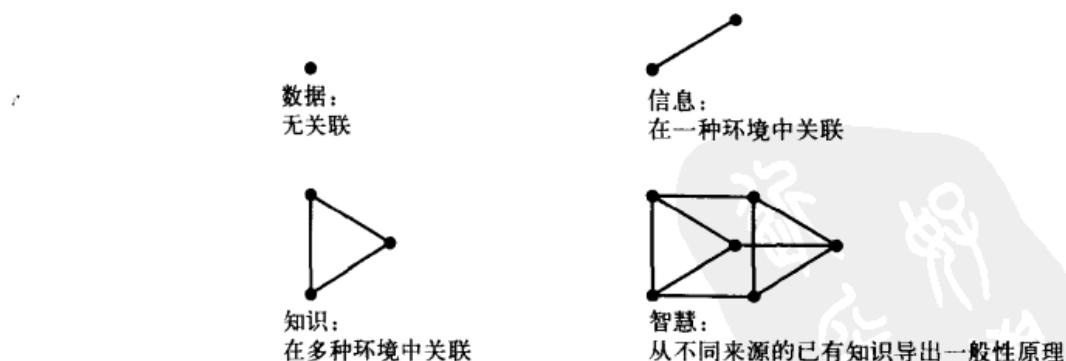


图32-1 “信息”的范围

“智慧是使我们能够运用知识为自己及他人获得利益的力量。” ——Thoms J. Watson

¹ 快速发展的数据挖掘及数据仓库技术反映了这种增长的趋势。

为了说明从数据到知识的发展过程，以人口普查数据为例，它指明在1996年美国的出生率是490万，该数字表示一个数据值。将该数据和前40年的出生率相关联，我们可以导出一种有用的信息——正在变老的20世纪50年代及60年代早期的“生育高峰出生的婴儿”正赶在他们的最佳生育年龄结束前作最后的生育努力，另外，“青年一代”(gen-Xers)也已到了生育的年龄。然后，该信息还可以和其他似乎无关的信息相关联，例如，将在下一个十年退休的小学教师的当前数量；具有初中和中级教育程度毕业生的数量；或者政治家承受的降低税率的压力；以及因此限制教师薪金增长的压力。

[898] 可以组合所有这些信息，然后表示为知识——在21世纪的头10年，美国的教育系统将面临巨大的压力，这种压力将持续十年之久。运用该知识，可能会发现商机，很可能有重要的机会去开发新的比当前的方法更有效、更价廉的学习模式。

软件的未来之路是处理知识的系统。我们使用计算机处理数据已超过50年，抽取信息超过30年。软件工程界面临的最重要的挑战之一是构造迈向下一步的系统——从数据和信息中以实用、有益的方式抽取知识的系统。

32.6 技术作为推动力

构造和使用软件的人员、应用的软件工程过程、生产的信息，这些均受到硬件和软件技术发展的影响。历史上，硬件是计算机领域的技术推动力，新的硬件技术提供了潜能，然后软件构造者根据用户需求力图去开发该潜能。

硬件技术的未来之路可能会沿着两条并行的路径发展。在一条路径上，硬件技术将继续快速地演化，由于传统硬件体系结构提供的更强的能力，对软件工程师的要求将继续增长。

但是，硬件技术方面的真正变化可能发生在另一条路径上，非传统的硬件体系结构（如，纳米碳管、EUL微处理器、认知机、网格计算）的发展可能会导致我们所构造的软件种类上的根本性变化以及软件工程方法基本原则的改变。因为这些非传统的方法才刚刚成熟，很难确定哪一个将有更广的影响，而要预测软件界将如何变化以适应它们则更加困难。

软件工程的未来之路将由软件技术驱动。在系统质量和投入市场的时间方面，复用和基于构件的软件工程提供了数量级改善的最好条件。事实上，随着时间的推移，软件业看起来可能与今天的硬件业比较相似，可能会有构造离散器件（可复用的软件构件）的厂商，也会有其他构造系统构件（如，一组人机交互工具）的厂商，以及为最终用户提供解决方案（产品及定制的系统）的系统集成商。

软件工程将发生变化——对此我们可以肯定。但是不管发生什么根本性的变化，我们可以肯定地说：质量将永远不会失去其重要性，有效的分析、设计及测试将总是在计算机系统的开发中有一席之地。

[899]

INFO

技术趋势

P. Cripwell Associates (www.jpcripwell.com) 是一家从事知识管理及信息工程咨询的公司，它提出了在即将到来的几年中影响技术方向的五个驱动技术。

联合技术：当两项重要的技术被融合后，其影响通常大于每项技术各自的影响之和。例如，GPS卫星技术与装载的计算能力及LCD显示技术相结合会得到高级的、自动的制图

系统。技术通常是沿着独立的路线发展，只有当某人将它们组合起来解决某个问题时，才能产生重要的商业或社会影响。

数据融合：我们获得的数据越多，需要的数据也就越多。更重要的是，我们获得的数据越多，抽取有用的信息就越困难。事实上，我们通常还需要获取更多的数据去理解什么数据是重要的，什么数据与特定的要求或来源相关，以及什么数据将被用于决策。这就是数据融合问题。J.P.Cripwell用一个“高级自动交通监控系统”作为例子。数字速度传感器（在公路上）及数字照相机感应事故，然后必须决定事故的严重程度（通过照相机）；基于事故的严重程度，监控系统一定要与警察、消防或救护车取得联系；重新疏导交通；媒体（收音机）广播警告，并且通知各个轿车（如果安装了数字传感器或无线通信）。为完成这项任务，基于从监控系统获得的数据需要做出各种不同的决定。

技术推动：在过去的几年中，一个问题出现后，一般是开发技术加以解决。因为该问题对很多人来说很清楚，新技术的市场很明确。今天，在寻求问题的解决方案时要发展某项技术，就必须去推动市场，使人们认识到市场需要这项新技术（例如，移动电话，PDA）。当人们认识到这种需要时，技术将加速发展、改进，并经常变种为组合技术。

网络及运气：这里的网络指人之间的连接或人与信息之间的连接，当网络增长时，两个节点（例如，人、信息源）之间协同的可能性也会增长。偶然的连接（运气）能够产生灵感、新技术或应用。

信息过载：通过Internet连接，任何人都可以接近巨大的信息海洋。当然，问题是如何发现正确的信息，确定其有效性，并将其转换为面向商业或个人的实际应用系统。

32.7 软件工程师的责任

软件工程已经发展成为令人敬重的、全球性的职业。作为专业人员，软件工程师应该遵守职业道德规范，以指导他们所做的工作及他们所生产的产品。ACM/IEEE-CS联合工作组已经提出了《Software Engineering Code of Ethics and Professional Practices》（软件工程职业道德规范和职业实践要求）（5.1版），该规范[ACM98]声明如下：

WebRef

ACM/IEEE职业道德规范的完全讨论见
[seeri.etsu.edu/](http://seeri.etsu.edu/Codes/default.shtml)
Codes/default.shtml。

软件工程师应履行其承诺，使软件的需求分析、规格说明、设计、开发、测试和维护成为一项有益和受人尊敬的职业。为实现他们对公众健康、安全和利益的承诺目标，软件工程师应当坚持以下八项原则：

1. 公众——软件工程师的行为应符合公众利益。
2. 客户和雇主——在保持与公众利益一致的原则下，软件工程师的行为应使他们的客户和雇主获得最大利益。
3. 产品——软件工程师应该确保他们的产品和相关的修改符合最高的专业标准。
4. 判断——软件工程师应当维护他们职业判断的完整性和独立性。
5. 管理——软件工程经理和领导应同意和促进对软件开发和维护进行合乎道德规范的管理。
6. 专业——在与公众利益一致的原则下，软件工程师应当推进其专业的完整性和声誉。

900

7. 同事——软件工程师对其同事应持平等和支持的态度。

8. 自我——软件工程师应当参与终生职业实践的学习，并促进合乎道德的职业实践方法。

虽然八项原则中的每一项都同样重要，但最重要的一个主题是：软件工程师应该以公众的利益为目标。从个人的角度上，软件工程师应遵守以下规定：

- 从不将数据据为己有。
- 从不散布或出售在软件项目中工作时所获得的私有信息。
- 从不恶意毁坏或修改另一个人的程序、文件或数据。
- 从不侵犯个人、小组或组织的隐私。
- 从不闯入一个系统胡闹或牟取利益。
- 从不制造或传播计算机病毒。
- 从不使用计算技术去助长偏见或制造麻烦。

在过去的十年中，软件工业界的某些成员已经达成了下面的保护条例[SEE03]：

1. 允许公司在不公开已知缺陷的情况下发布软件；
2. 由这些已知缺陷所引起的任何损害，免除开发者的赔偿责任；
3. 没有得到原始开发者的允许，禁止其他人公开缺陷；
4. 允许将“自助”软件结合到一个产品中——这样能使产品的操作丧失能力（通过远程命令）；
5. 如果第三方使软件丧失能力，免除使用“自助”能力的软件的开发者的赔偿责任。

与所有的立法一样，争论的问题通常集中在政治方面，而不是技术方面。然而，很多人（包括本书的作者）感到：如果没有合理地起草保护条例，通过间接地免除软件工程师生产高质量软件的责任，保护条例会与软件工程道德公约相冲突。

32.8 结束语

自本书第1版的编写，至今25年已经过去了。我仍然能够回忆起作为一位年轻的教授，坐在桌子旁为一本书撰写手稿（手写）的情形，这本书的主题几乎没有人关心，甚至很少有人理解。我还记得出版商的拒绝信，他（礼貌但坚定地）认为“软件工程”方面的书绝对不会有什么市场。幸运的是，McGraw-Hill出版社决定尝试一下²，其余的如他们所说的那样已经成为历史了。

在过去的25年中，本书发生了引人注目的变化——在范围、规模、风格和内容等方面。如软件工程一样，经过这么多年，它已经长大并且（我希望）成熟。

计算机软件开发的工程方法目前只是传统的智慧，虽然在“合适的范型”、敏捷的重要性、自动化程度以及最有效的方法等方面还存在争论，但软件工程的基本原则现在已在产业界得到普遍接受。然而，为什么直到最近我们才看见它们的广泛采用呢？

我认为答案是由于技术转化和伴随而来的文化变化的原因，即使我们大多数人认识到了软件需要工程学科，我们仍需要与过去的习惯作斗争，并且要面对一些容易重复过去所犯错误的新的应用领域（以及在这些领域工作的开发者）。

² 实际上，这应该归功于Peter Freeman和Eric Munson，是他们使McGraw-Hill确信这本书值得出版。

为了使转化更容易，我们需要很多东西——一个灵活的、可适应的、明智的软件过程，更有效的方法，更强大的工具，更好地被实践者接受和获得管理者的支持，以及大量的教育和“广告”。软件工程还没有收到大量广告带来的收益，但是，随着时间的流逝，软件工程概念正在“兜售”它自己。在某种形式上，本书是该技术的一种“广告”。

你不可能同意本书中描述的每一个方法。书中某些技术和观点是相互矛盾的；为了在不同的软件开发环境中更好地发挥作用，必须对书中的某些部分进行调整。然而，我真诚地希望本书已经描述了我们面临的问题，展示了软件工程概念的优势并提供了方法和工具的框架。

当我们走进21世纪之时，软件的影响和重要性已经走过了一段很长的路，软件已经变成了世界上最重要的产品和最重要的产业。然而，新一代的软件开发者必须迎接很多前一代人面临过的同样的挑战。让我们期待迎接挑战的人们（软件工程师）用更多的智慧去开发改善人类条件的系统。[902]

参考文献

- [ACM98] ACM/IEEE-CS Joint Task Force, *Software Engineering Code of Ethics and Professional Practice*, 1998, available at <http://www.acm.org/serving/se/code.htm>.
- [BEC01] Beck, K., et al., "Manifesto for Agile Software Development," <http://www.agilemanifesto.org/>.
- [BOL91] Bollinger, T., and C. McGowen, "A Critical Look at Software Capability Evaluations," *IEEE Software*, July 1991, pp. 25–41.
- [GIL96] Gilb, T., "What Is Level Six?" *IEEE Software*, January 1996, pp. 97–98, 103.
- [HOP90] Hopper, M. D., "Rattling SABRE, New Ways to Compete on Information," *Harvard Business Review*, May–June 1990.
- [PAU93] Paulk, M., et al., *Capability Maturity Model for Software*, Software Engineering Institute, Carnegie Mellon University, 1993.
- [PCM03] "Technologies to Watch," *PC Magazine*, July 2003, available at <http://www.pcmag.com/article2/0,4149,1130591,00.asp>.
- [PRE91] Pressman, R. S., and S. R. Herron, *Software Shock*, Dorset House, 1991.
- [SEE03] The Software Engineering Ethics Research Institute, "UCITA Updates," 2003, available at <http://seeri.etsu.edu/default.htm>.

习题与思考题

- 32.1 复习在第4章中讨论的敏捷、增量过程模型，做一些研究，并收集当前该主题的论文，基于论文中描述的经验总结敏捷范型的优点和缺点。
- 32.2 试着开发一个例子，从原始数据的收集开始，然后是信息获取，再后是知识，最后是智慧。
- 32.3 找一本本周的主要商业和新闻杂志（例如，《新闻周刊》、《时代》、《商业周刊》），列出可用于说明软件的重要性的每一篇文章或每一条新闻。
- 32.4 对大约2010年理想的软件工程师的开发环境给出一个简要的描述，描述环境（硬件、软件及通信技术）中的元素及对质量和投放市场时间方面的影响。
- 32.5 最热的软件应用领域之一是基于Web的系统及应用。讨论人、通信、过程应如何发展才能适应“下一代”WebApp的开发。
- 32.6 提供特定的例子，对在32.7节中描述的软件工程职业道德规范的八项原则之一进行举例说明。

推荐读物与阅读信息

讨论软件和计算机的未来之路的书籍涉及技术、科学、经济、政治和社会等诸多方面的问题。Sterling (《Tomorrow Now》, Random House, 2002) 提醒我们：真正的进步很少是有序和有效的。Teich (《Technology and the Future》, Wadsworth, 2002) 在技术的社会影响以及变化中的文化应如何塑造技术方面进行了思想性的尝试。Naisbitt及Philips (《High Tech/High Touch》, Nicholas Brealey, 2001) 说我们中的许多人“沉醉于”高技术，而“高技术时代的最大讽刺是我们已经受到希望能给我们自由的设备的束缚”。Zey (《The Future Factor》, McGraw-Hill, 2000) 讨论了本世纪塑造人类命运的五种力量。Canton (《Technofutures》, Hay House, 1999) 讨论了在21世纪技术将如何转化为商业。Robertson (《The New Renaissance: Computers and the Next Level of Civilization》, Oxford University Press, 1998) 辩论到：计算机革命可能是文明史上一个最重要的进步。

Broderick (《Spike》, Forge, 2001) 讨论了新兴技术的影响。Dertrouzos及Gates (《What Will Be: How the New World of Information Will Change Our Lives》, Harper-Business, 1998) 对于本世纪头几十年中信息技术的发展方向给出了具有思想性的探讨。Barnatt (《Valueware: Technology, Humanity and Organization》, Praeger Publishing, 1999) 对“概念经济”以及随着cyber (计算机) 商业的发展如何产生经济价值给出了具有诱惑力的讨论。Negroponte (《Being Digital》, Alfred A. Knopf, 1995) 是20世纪90年代中期最畅销的书，提供了计算技术及其总体影响的有趣视图。

Kroker (《Digital Delirium》, New World Perspectives, 1997) 编辑了一部考察数字技术对人们及社会的影响方面的评论、诗歌及幽默的有争议的文集。Brin (《The Transparent Society: Will Technology Force Us to Choose Between Privacy and Freedom?》, Perseus Books, 1999) 讨论了伴随着信息技术的发展，个人隐私的损失不可避免。Shenk (《Data Smog: Surviving the Information Glut》, HarperCollins, 1998) 讨论了被软件产生的大量信息淹没的“受信息侵染的社会”相关的问题。

Brockman (《The Next Fifty Years》, Vintage Books, 2002)、Miller及其同事 (《21st Century Technologies: Promises and Perils of a Dynamic Future》, Brookings Institution Press, 1999) 编辑了一部技术对社会、商业及经济结构有影响的论文及评论集。对技术问题感兴趣的人们，Luryi、Xu及Zaslavsky (《Future Trends in Microelectronics》, Wiley, 1999) 编辑了一部以纳米技术为重点的计算机硬件的可能发展方向的论文集。Hayzelden及Bigham (《Software Agents for Future Communication Systems》, Springer-Verlag, 1999) 编辑了一部讨论智能软件代理发展趋势的文集。

随着软件渗透到我们的生活，“计算机伦理学”已经成为讨论的重要主题。Spinello (《Cyberethics: Morality and Law in Cyberspace》, Jones & Bartlett Publishers, 2002)、Halbert和Ingulli (《Cyberethics》, South-Western College Publishers, 2001)、Baird和他的同事 (《Cyberethics: Social and Moral Issues in the Computer Age》, Prometheus Books, 2000) 撰写的书详细讨论了这方面的主题。美国政府用CD-ROM (《21st Century Guide to Cybercrime》, Progressive Management, 2003) 发表了大量报告，考虑了计算机犯罪、知识所有权问题及国家基础设施保护中心 (NIPC) 的所有方面。

Kurzweil (《The Age of Spiritual Machines, When Computer Exceed Human Intelligence》, Viking/Penguin Books, 1999) 认为在20年内硬件技术将具有完全模仿人类大脑的能力。Borgmann (《Holding on to Reality: The Nature of Information at the Turn of the Millennium》, University of Chicago Press, 1999) 撰写了迷人的信息历史, 描绘了它在文化转换中的作用。Devlin (《InfoSence: Turning Information into Knowledge》, W.H.Freeman & Co., 1999) 试图去搞清楚每天都在轰炸我们的源源不断的信流的含义。Gleick (《Faster: The Acceleration of Just About Everything》, Pantheon Books, 2000) 讨论了技术变化的加速率及对现代生活各个方面的影响。Jonscher (《The Evolution of Wired Life: From the Alphabet to the Soul-Catcher Chip—How Information Technology Change Our World》, Wiley, 2000) 认为人类的思想及交流超越了技术的重要性。

在Internet上有大量关于软件的技术及软件工程的未来方向方面的信息源。WWW参考文献的最新列表可在SEPA网站<http://www.mhhe.com/pressman>找到。

904

索引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

Abstraction (抽象), 265
Acceptance tests (验收测试), 113
Accessibility (可访问性), 380
Action paths (作用路径), 316
Actions (动作, 活动), 55
Activities (活动), 54
Activity diagram (活动图) 168, 197, 223, 348
Activity network (活动网络) 715
Actors (参与者) 191, 221
Adaptive cycle planning (自适应周期设计), 114
Adaptive Software Development (ASD) (自适应软件开发), 114
Aesthetic design (美学设计), 565, 573
 layout issues (布局问题), 573
 WebApps (Web应用), 573
Agile manifesto (敏捷宣言), 103
Agile modeling (敏捷建模), 121
 principles (原则), 122, 143
Agile process (敏捷过程), 106
 politics of (的政治), 107
Agile teams (敏捷团队), 636
Agility (敏捷), 105. *See also* Agile process
 definition of (的定义), 105
 human factors (人员因素), 108
 principles of (的原则), 105
AI software (AI软件), 41
Airlie Council (Airlie Council专家组), 644
Alpha testing (α 测试), 407
Analysis (分析), 208. *See also* Requirements analysis
 Object-oriented (面向对象), 217
 patterns (模式), 200. *See also* Patterns
 rules of thumb (经验法则), 210
Analysis classes (分析类)

 attributes (属性), 235
 characteristics of (的特征), 235
 identification of (的标识), 233
 operations (操作), 238
 types of (的类型), 234, 241
 WebApps (Web应用), 548
Analysis model (分析模型)
 elements of (的元素), 196, 212
 relationship to design (与设计的关系), 260
 WebApps (Web应用), 545
Analysis modeling (分析建模), 178, 207
 approaches (方法), 211
 behavioral (行为), 198, 248
 class-based (基于类的), 198, 233
 content (内容), 545
 flow-oriented (面向信息流的), 199, 226
 interaction (交互), 548
 principles (原则), 140
 scenario-based (基于场景的), 197, 218
 task set (任务集), 141
 Web engineering (Web工程), 540
Analysis packages (分析包), 248
Anchor points (定位点), 86
Application architecture (应用架构), 161
Application software (应用软件), 40
Appraisal costs (鉴定成本), 747
Archetypes (原始模型), 300
Architectural Description Languages(ADLs) (体系结构
 (架构) 描述语言), 307
Architectural design (体系结构 (架构) 设计), 286,
 298
 assessment of (的评估), 304
 complexity of (的复杂性), 306
 mapping data flow (映射数据流), 307
 refinement of (的求精), 301, 320

- WebApps (Web应用), 577
 Architectural patterns (体系结构(架构)模式), 291
 refinement of (的求精), 297
 types of (的类型), 577
 Architectural styles (体系结构(架构)类型), 291
 call and return (调用和返回), 294
 data flow (数据流), 293
 data-centered (数据为中心的), 293
 layered (分层), 295
 object-oriented (面向对象), 294
 taxonomy of (的分类), 292
 Architecture (体系结构, 架构), 265
 data (数据), 161
 description of (的描述), 287
 importance of (的重要性), 288
 MVC (模型-视图-控制器), 580
 sensitivity analysis (敏感性分析), 305
 WebApp (Web应用), 579
 Architecture context diagram (ACD), 298
 Architecture trade-off analysis method(ATAM) (体系结构(架构)权衡分析方法), 304
 Aspect-oriented development (面向方面的开发), 93
 Aspectsual requirements (方面需求) 93
 Associations (关联) 246
 Attributes (属性), 236
 Audits (审核), 780. *See also Configuration audit*
 Authentication (认证), 618
 Authorization (授权), 90
 Availability, WebApps (可用性(可得性), Web应用), 562
- B**
- Backlog (待定项), 119
 Baseline (基线), 775
 Baseline
 definition of (的基线定义), 775
 metrics (度量), 665
 Basis path testing (基本路径测试), 425
 example of (的实例), 429
 Basis set (基本集) 427
 Bathtub curve (浴缸曲线) 37
 Behavioral model (行为模型) 249
 Beta testing (β测试), 407
- Black-box specification (黑盒规格说明), 834
 Black-box testing (黑盒测试), 424, 434
 Boundary classes (边界类), 241
 Boundary value analysis (边界值分析), 438, 612
 Box structure specification (盒结构规格说明), 833
 Bug tracking (缺陷跟踪), 782
 Bugs (缺陷), 751
 Build (构造), 401
 Business process engineering(BPR) (业务过程工程), 161
 hierarchy (层次), 162
 Business process reengineering(BPR) (业务过程再工程), 870
 process model (过程模型), 871
 Business processes (业务过程), 871
- C**
- Cardinality (基数), 215, 246
 CASE, *See Tools* (CASE, 参见工具)
 CBA IPI (基于CMM的内部过程改进评估), 66
 CBSE (基于构件的软件工程), 847
 cost analysis (成本分析), 863
 economics of (的经济学), 862
 process (过程), 850
 Certification (认证), 843
 Change (变更), 38, 138
 impact on software (对软件的影响), 38
 origin of (的原因), 772
 scope of (的范围), 893
 Change control (变更控制), 784
 types of (的类型), 785
 workflow (工作流), 785
 Change control authority(CCA) (变更控制授权人), 784
 Change management (变更管理), 771. *See also SCM*
 WebApps (Web应用), 793
 Change order (变更工单), 784
 Change report (变更报告), 784
 Change request (变更请求), 784
 Change set (变更集), 783
 Chaos (混沌), 77
 Characterization functions (特征化函数), 852
 Checklists
 requirements validation (需求确认检查单), 179, 203

- WebApp design quality (WebApp设计质量), 562
CK metrics suite (CK度量集), 480
Class diagram (类图), 170, 237
Classes (类)
 composite aggregate (聚合), 244
 multiplicity (重数), 246
Classic life cycle model (传统生命周期模型), 79
Class-responsibility-collaborator (类-职责-协作者模型), 240. *See also* CRC modeling
Cleanroom software engineering (净室软件工程), 92, 828
 certification (认证), 843
 design (设计), 836
 differentiating characteristics (特征区别), 832
 formal specification (形式化规格说明), 833
 strategy (策略) 830
 testing (测试), 841
Clear-box specification (清晰盒规格说明), 835
Cluster testing (集群测试), 405
CMMI (能力成熟度模型集成), 59
 adoption of (的采纳), 62
 capability levels (能力水平), 59
 continuous model (连续模型), 59
 goals (目标), 61
 practices (实践), 61
 staged model (分阶段模型), 62
 COCOMO II, 692
Code restructuring (代码重构), 877
Coding, principles (编码, 原则), 145
Cohesion (内聚), 272, 480
 levels of (的层次), 335
 metrics for (的度量), 486
Collaboration (协作), 108, 134, 183
 CRC definition (CRC定义), 242
Common closure principle(CCP) (通用闭合原则), 333
Common reuse principle(CRP) (通用复用原则), 333
Communication (沟通, 交流), 56, 133
 principles of (的原则), 133
task set (任务集), 135
stakeholders (共利益者), 519
Web engineering (Web工程), 507, 519
Compatibility tests (兼容性测试), 610
Completeness (完整性), 480
Complexity (复杂性), 480
 metrics (度量), 488
Component qualification (构件合格性认证), 855
Component wrapping, 855
Component-based construction (基于构件的构造), 39
Component-based development (基于构件的开发), 91
Component-based development(CBD) (基于构件的开发), 851
 activities (活动), 853
Component-based software engineering (基于构件的软件工程), 847. *See also* CBSE
Component-based systems (基于构件的系统), 848
Component-level design (构件级设计), 324
 basic principles (基本原则), 331
 guidelines (指导原则), 334
 steps (步骤), 339
 WebApps (Web应用), 584
Components (构件)
 adaptation (适应), 855
 class-based (基于类的), 330
 classification (分类), 860
 composition (组合), 856
 conventional (传统的), 327, 347
 definition of (的定义), 325, 849
 describing (描述), 859
 design example (设计实例), 329
 engineering (工程), 857
 off-the-shelf (商品化成品), 679
 OO view (OO视图), 326
 reusable (可复用的), 679
Computer-Aided Software Engineering (计算机辅助软件工程), *See also* Tools
Concurrency (并发性), 296
Concurrent development model (并发开发模型), 88
Concurrent Versions System(CVS) (并行版本系统), 783
Condition testing (条件测试), 432
Condition-transition-consequence(CTC) format (条件-变迁-结果 (CTC) 格式), 737
Configuration audit (配置审核), 787
 WebApps (Web应用), 796
Configuration management (配置管理), 801. *See also* SCM
Configuration objects, (配置对象) 776

- WebApps (Web应用), 790
 Configuration review (配置评审), 406
 Configuration testing (配置测试)
 client-side (客户端), 616
 server-side (服务器端), 616
 WebApps (Web应用), 600, 615
 Consequences, unintended (结果, 未预料的), 33
 Construction (构造, 构建), 56
 practice (实践), 144
 task set (任务集), 146
 Web engineering (Web工程), 508
 Constructive specification (构造性规格说明), 809
 Content architecture (内容体系结构), 577
 structures (结构), 577
 Content design (内容设计), 575
 Content hierarchy (内容层次), 547
 Content management (内容管理), 790
 Content model (内容建模), 545
 Content objects (内容对象), 546, 575
 Content relationships (内容关系), 548
 Content testing (内容测试), 601
 Context diagram (语境图), 227
 Control flow modeling (控制流建模), 229
 Control specification(CSPEC) (控制规格说明), 230
 Controller classes (控制类), 241
 Core product (核心产品), 80
 Correctness (正确性), 662
 conditions (条件), 838
 proof of (的证明), 837
 verification (验证), 837
 Cost, variance (成本, 偏差), 723
 Coupling (耦合), 272, 480
 levels of (的层次), 337
 metrics (度量), 485, 487
 CRC model (CRC模型)
 building (构造), 240
 collaborations (协作), 242
 responsibilities (职责), 241
 review guidelines (评审指导原则), 244
 CRC modeling (CRC建模), 240
 in XP (在极限编程中), 111
 Critical path (关键路径), 716
 Crosscutting concerns (横切关注点), 93
 Crystal (Crystal敏捷方法), 119
 Customers (客户), 134
 Cyclomatic complexity (环复杂度), 427
 D
 Data architecture (数据架构), 161
 Data attributes (数据属性), 214
 Data design (数据设计), 275
 architectural level (体系结构级), 289
 component level (构件级), 290
 Data flow diagram(DFD) (数据流图 (DFD)), 227, 309
 Data flow mapping (数据流映射), 308
 Data flow testing (数据流测试), 432
 Data invariant (数据不变式), 805
 Data mining (数据挖掘), 289
 Data modeling (数据建模), 213
 relationships (关系), 214
 Data object (数据对象), 213
 Data restructuring (数据重构), 877
 Data tree (数据树), 547
 Data warehouse (数据仓库), 289
 Database testing (数据库测试), 603
 Debugging (调试), 411
 process (过程), 411
 psychological considerations (心理考虑), 413
 strategies (策略), 414
 tactics (战略), 414
 Decision table (决策表), 349
 Decision tree analysis (决策树分析), 699
 Deep structure (深层结构), 446
 Defect amplification model (缺陷放大模型), 752
 Defect removal efficiency(DRE) (缺陷排除效率), 663
 Defects (缺陷), 751
 Dependencies (依赖), 246
 Dependency inversion principle(DIP) (依赖倒置原则), 332
 Dependency tracking, (依赖跟踪) 779
 Deployment (部署), 56
 principles (原则), 148
 task set (任务集), 149
 Web engineering (Web工程), 508
 Deployment diagram (部署图), 168
 Design (设计), 258, 286, 324, 356

- aesthetic (美学的), 573
architectural (体系结构的), 286, 577
cleanroom software engineering (净室软件工程), 836
component level (构件级), 324, 584
hypermedia (超媒体), 586
metrics (度量), 477
process (过程), 261
proving correct (证明正确性), 837
quality attributes (质量属性), 263
quality guidelines (质量原则), 262
task set (任务集), 144, 264
user interfaces (用户界面), 356
verification (验证), 836
WebApp interfaces (WebApp 界面), 565
WebApps (Web应用), 559
- Design classes (设计类), 271
characteristics of (的特征), 272
types of (的类型), 271
- Design elements (设计元素)
architecture (体系结构), 275
components (构件), 278
data (数据), 275
deployment (部署), 279
interface (界面), 276
- Design engineering (设计工程), 258
- Design model (设计模型), 260
dimensions of (的维度), 274
relationship to analysis (与分析的关系), 260
- Design modeling, principles (设计建模, 原则), 141
- Design notation (设计表示法)
comparison of (的比较), 352
graphical (图形), 348
tabular (表格), 349
text-based (基于文本), 350
- Design patterns (设计模式), 266. *See also* Patterns
description of (的描述), 280
template (模板), 280
use of (的应用), 281
- Design structure quality index(DSQI) (设计结构质量指标), 479
- Deterioration (退化), 38
- Distribution (分布), 297
- Document restructuring (文档重构), 876
- Domain analysis (领域分析), 210
Domain engineering (领域工程), 851
Drivers (驱动), 396
DSDM (动态系统开发方法), 116
- E**
- Earned value analysis(EVA) (获得价值分析), 722
Efficiency (有效性), 465
Effort distribution (工作分配), 712
Elaboration (精化, 细化), 177, 639
 objects (对象), 369
 tasks (任务), 368
Elicitation (引出), 177, 184
 work products (工作产品), 190
- Embedded software (嵌入式软件), 40
- Encryption (加密), 618
- End-users (最终用户), 134
- Engineering/scientific software (工程/科学软件), 40
- Entity classes (实体类), 241
- Entity-relationship diagrams (实体 - 关系图), 216
- Equivalence class (等价类), 437
- Equivalence partitioning (等价划分), 437, 612
- Error handling (错误处理), 378
- Errors (错误), 751
 correcting (更正), 416
 relative cost (相对成本), 748
- Estimates (估算)
 reconciling (调和), 690
 Web engineering (Web工程), 531
- Estimation (估算), 674, 680
 agile development (敏捷开发), 696
 automated techniques (自动化技术), 691
 decomposition (分解), 681
 empirical models (经验模型), 691
 FP-based (基于FP的), 685
 LOC-based (基于LOC的), 683
 observations (观察), 675
 OO projects (OO项目), 695
 Models (模型), 692
 problem-based (基于问题的), 682
 process-based (基于过程的), 686
 use-cases (用例), 688
 WebApps (Web应用), 697

- Ethics (道德规范), 900
 code of (的代码), 900
 personal considerations (个人考虑), 901
- Events (事件), 230, 249
- Evolution. *See Software evolution* (演化, 参见软件演化)
- Evolutionary models (演化模型), 83
- Extreme Programming(XP) (极限编程), 110
 coding (编码), 112
 design (设计), 111
 framework activities (框架活动), 110
 planning (计划/策划), 110
 testing (测试), 113
- F**
- Factoring, first level (分解, 第一级), 312
- Failure costs (失效成本), 747
- Failure curves (失效曲线), 38
- Feasibility (可行性), 677
- Feature, definition of (特征, 的定义), 120
- Feature Driven Development(FDD) (特征驱动开发), 120
- Fiit's Law (Fiit法则), 568
- Filters (过滤器), 293
- Fire-fighting mode (救火模式), 726
- Firewalls (防火墙), 618
- Flow graph (流图), 425, 427
- Flowchart (流程图), 348
- Formal methods (形式化方法), 92, 802
 concepts (概念), 802, 805
 definition of (的定义), 802
 guidelines (指导原则), 823
 mathematical preliminaries (数学预备知识), 808
- Formal specification (形式化规格说明), 813
- Formal specification languages (形式化规格说明语言), 815
- Formal technical reviews(FTR) (正式技术评审), 263, 569, 751. *See also Reviews*
- Formulation (表达), 507, 514
 questions (问题), 515
 requirements gathering (需求收集), 517
- Forward engineering (正向工程), 878, 884
 client server (客户机服务器), 885
 OO systems (OO系统), 886
 user interfaces (用户界面), 887
- 40-20-40 rule (40-20-40法则), 712
- Framework activities (框架活动), 54, 56
 generic (通用的), 56
 PSP (个人软件过程), 68
 TSP (小组软件过程), 70
- Frameworks (框架), 281
- Function points (功能点), 472, 656
 computation of (的计算), 475
 programming languages (编程语言), 657
 reconciling (调和), 656
- Functional decomposition (功能分解), 368
- Functional independence (功能独立性), 268
- Functional model (功能模型), 551
- Functionality (功能), 464
- G**
- GQM paradigm (目标/问题/度量 (GQM) 范例), 468
- Grammatical parse (语法分析), 227
- Granularity (粒度), 137
- Graph matrices (图形度量), 431
- Graphic design. *See Aesthetic design* (美术设计, 见美学设计)
- H**
- Halstead metrics (Halstead度量), 490
- Hatley-Pirbhai modeling (Hatley-Pirbhai建模), 165
- Hazard analysis(灾难分析), 739
- Help facilities (帮助功能), 378
- Human factors (人员因素), 108
- I**
- Idioms (习惯用语), 281
- Immediacy (直接), 503
- Inception (起始), 176
- Increment (增量). *See Software increment*
- Incremental delivery (增量交付), 107
- Incremental models (增量模型), 80
- Independent paths (独立路径), 426
- Independent test group(ITG) (独立测试组), 389
- Indicators (指示), 466
- Information hiding (信息隐蔽), 268
- Information representation of (的信息表示), 897
- Information spectrum (信息领域), 898

Infrastructure, technology (基础设施, 技术), 161

Inspections (审查), 751. *See also* Reviews

Integration testing (集成测试), 391, 397

bottom-up (自底向上), 400

breadth-first (宽度优先), 399

depth-first (深度优先), 398

documentation (文档), 403

top down (自顶向下), 398

Integrity (完整性), 662

Interaction model (交互模型), 548

Interface design (界面设计), 356

analysis (分析), 365

patterns (模式), 376

principles (原则), 357, 566

process (过程), 363

Interface design—*Cont.*

workflow (工作流), 373, 571

WebApps (Web应用), 565

Interface segregation principle(ISP) (界面隔离原则), 332

Internationalization (国际化), 380

Inventory analysis (库存目录分析), 875

ISO 9000 (ISO 9000), 765

ISO 9001:2000 (ISO 9001:2000), 67, 765

outline of (的大纲), 765

Issues list (问题列表), 187

Iterative models (迭代模型), 83

KISS (保持简单性原则), 131

Knowledge (知识), 898

L

Labeling, menus and commands (标签、菜单和命令), 379

Law of Demeter (Demeter法则), 272

Legacy software (遗留软件), 42

quality of (的质量), 43

Line-of-code(LOC) metric (代码行 (LOC) 度量), 655

Link weight (连接权值), 431, 435

Liskov substitution principle(LSP) (Liskov代替原则), 332

Load testing (负载测试), 620

Logic operators (逻辑操作符), 812

Loop testing (循环测试), 433

M

Maintainability (可维护性), 465, 662

Maintenance (维护), 39, 43, 873

metrics (度量), 492

Make facility (make工具), 782

Make-buy decision (自行开发或购买的决策), 698

Manifesto, agile software development (宣言, 敏捷软件开发), 103

McCall's quality factors (McCall质量因素), 463

Mean-time-between-failures(MTBF) (平均失效间隔时间), 763

Mean-time-to-repair(MTTR) (平均维修时间), 763

Measurement (测量), 466

Measures (测度), 466

direct (直接的), 654

indirect (间接的), 654

Metaphor (隐喻/比喻), 569

Methods (方法), 54

Metrics (度量)

analysis (分析), 472

architectural design (体系结构设计), 477

arguments for (争论), 665

attributes (属性), 469

baseline (基线), 665

challenges (挑战), 466

class-oriented (面向类的), 484

cohesion (内聚), 486

complexity (复杂性), 488

component-level (构件级), 486

coupling (耦合), 485

definition (定义), 466

design (设计), 477

establishing (建立), 668

etiquette (规则), 652

function-based (基于功能的), 472

maintenance (维护), 492

object-oriented (面向对象的), 480, 491, 658

operation-oriented (面向操作的), 488

private (私有), 651

process (过程), 649, 666

product (产品), 461, 470, 653

productivity (生产率), 682

public (公用), 652
 quality (质量), 661
 size-oriented (面向规模的), 655
 small organizations (小型组织), 666
 source code (源代码), 490
 specification quality (规格说明质量), 476
 testing (测试), 491
 types of (的类型), 470
 user interface (用户界面), 489
 WebApps (Web应用), 532, 588, 659
 Middleware (中间件), 330
 Milestones (里程碑), 720
 anchor points (定位点), 86
 Mini-specs (小型规格说明), 187
 Modality (形态), 216
 Modeling (建模), 56
 Web engineering (Web工程), 508
 Model-view-controller(MVC) (模型－视图－控制器 (MVC) 模型), 580
 Modularity (模块化), 267
 MOOD metrics suite (MOOD度量集), 484
 Multiplicity (重数), 246
 Myths (神话), 45
 customer (客户), 46
 management (管理), 45
 practitioner (实践者), 46

N

Navigation (导航), 553, 565
 analysis of (的分析), 554, 555
 questions (问题), 554
 semantics (语义), 581, 614
 syntax (语法), 581, 583, 613
 Navigation design (导航设计), 581
 Navigation nodes (导航节点), 582
 Navigation semantic unit(NSU) (导航语义单元), 582, 614
 Navigation testing (导航测试), 613
 Navigational links (导航链接), 582
 Negotiation (谈判), 135, 178, 201
 Netsourcing (网络资源), 41
 New economy (新经济), 42

O

Object constraint language(OCL) (对象约束语言), 345, 816
 example of (的实例), 818
 invariant (不变式), 346
 notation (符号/表示法), 817
 overview (概述), 816
 pre-/post conditions (前置/后置条件), 346
 Object elaboration (对象细化), 369
 Object points (对象点), 693
 Object request broker (对象请求代理)
 architecture (体系结构), 857
 Object-oriented (面向对象)
 concepts (概念), 217
 estimation (估算), 695
 project tracking (项目跟踪), 720
 OMG/CORBA (OMG/公用对象请求代理结构), 856
 OOHDM (面向对象的超媒体设计方法), 586
 abstract interface design (抽象接口设计), 588
 conceptual design (概念设计), 586
 navigational design (导航设计), 587
 Open source (开源), 41
 Open-closed principle(OCP) (开放－封闭原则), 331
 Operations (操作), 237, 805
 identifying (识别), 237
 Orthogonal array testing (正交数组测试), 439
 Outsourcing (外包), 700
 Web engineering (Web工程), 526

P

Packages (包), 247
 Pair programming (结对编程), 112
 Pareto principle (Pareto原则), 147
 Partitioning (划分), 639
 Path testing (路径测试), 612
 Patterns (模式), 132, 142, 146
 analysis (分析), 200
 analysis patterns template (分析模式模板), 200
 architectural (体系结构的), 292, 296
 design (设计), 266
 hypermedia design (超媒体设计), 584
 hypermedia repositories (超媒体库), 585
 process (过程), 66

- template for (的模板), 66
testing (测试), 456
user interface (用户界面), 375
PDL (程序设计语言), 350
People (人员), 599, 895
management issues (管理问题), 629
relationship with effort (与工作量的关系), 710
stakeholders (共利益者), 631
Performance testing (性能测试), 410, 619
Persistence (持久性), 297
Personal software process(PSP) (个人软件过程), 69
PERT/CPM (程序估算和评审技术/关键路径方法), 716
Pipes (管道), 293
Planning (计划/策划), 56, 642
principles (原则), 136, 642
task set (任务集), 139
WebE (Web工程), 508, 522
Portability (可移植性), 465
Postcondition (后置条件), 806
Practice(software engineering) (实践 (软件工程)), 127~153
Precondition (前置条件), 805
Prescriptive models (惯例模型), 78. *See also* Process models
failures of (的失效), 104
Prevention costs (预防的成本), 747
Primitiveness (原始性), 272, 480
Principles (原则)
agile modeling (敏捷建模), 143
analysis (分析), 139
coding (编码), 145
communication (沟通), 133
deployment (部署), 148
design (设计), 142
planning (计划/策划), 136
software engineering (软件工程), 131
testing (测试), 146
Priority points (优先点), 183
Problem decomposition (问题分解), 639
Problem solving (解决问题), 129
Process (过程), 54
assessment (评估), 66
decomposition (分解), 641
framework (框架), 54
future directions (未来方向), 896
management issues (管理问题), 630
metrics (度量), 649
project management issues (项目管理问题), 640
relationship to product (与产品的关系), 72
Process models (过程模型)
agile (敏捷), 106
ASD (适应性软件开发), 114
blocking states (阻塞状态), 80
BPR (业务过程再工程), 871
CBSE (基于构件的软件工程), 850
cleanroom (净室), 829
concurrent development (并发开发), 88
Crystal (Crystal敏捷方法), 119
differences (区别), 58
DSDM (动态系统开发方法), 116
evolutionary (演化型), 83
Extreme Programming (极限编程), 110
FDD (特征驱动的开发), 120
incremental (增量), 80
prescriptive (说明性的), 78
prototyping (原型开发), 83
RAD (快速应用开发), 81
reengineering (再工程), 874
risk driven (风险驱动), 86
Scrum (Scrum敏捷模型), 117
specialized (专用的), 91
Spiral (螺旋型), 86
waterfall (瀑布型), 79
Process patterns (过程模式), 63
example of (的实例), 66
Process specification(PSPEC) (处理规格说明), 232
Process technology (过程技术), 71
Product (产品), 638
management issues (管理问题), 630
relationship to process (与过程的关系), 72
Product engineering (产品工程), 162
Productivity metrics (生产率度量), 682
Product-line software (产品线软件), 40
Program design language. *See* PDL (程序设计语言, 参见PDL)
Project database (项目数据库), 775. *See also*

- Repository
 Project estimation. *See Estimation* (项目估算, 参见估算)
 Project management (项目管理), 628
 critical practices (关键实践), 644
 issues (问题), 630
 Project planning (项目计划), 676
 Project scheduling. *See Scheduling* (项目进度安排, 参见进度安排)
 Project table (项目表), 719
 Project tracking, WebE (项目跟踪, WebE), 531
 Project velocity (项目速度), 111
 Projects (项目)
 differences (区别), 522
 estimation (估算), 674
 metrics for (的度量), 653
 OO metrics (OO度量), 658
 problems (问题), 642
 tracking (跟踪), 718
 types of (的类型), 713
 Proof of correctness (正确性证明), 837
 Prototyping (原型开发), 83
 problems with (的问题), 85
 Putnam-Raleigh-Norden curve (Putnam-Raleigh-Norden 曲线), 711
- Q**
- Quality (质量)
 CBSE context (CBSE环境), 862
 cost of (的成本), 747
 definition of (的定义), 462, 746
 determinants (决定因素), 651
 factors (因素), 463
 guidelines for WebE (WebE指导原则), 511
 ISO 9126 factors (ISO 9126因素), 464
 measurement of (的测量), 662
 metrics (度量), 661
 Quality assurance (质量保证), 747
 activities (活动), 750
 history (历史), 749
 plan (计划), 766
 statistical (统计), 759
 Quality control (质量控制), 746
 Quality function deployment(QFD) (质量功能部署), 188
- Quality management (质量管理), 744. *See also Quality assurance*
 Quality of conformance (一致性质量), 746
 Quality of design (设计质量), 746
 Quality requirements tree, WebApps (质量需求树, Web 应用), 561
 Questions, context free (问题, 上下文无关), 183
- R**
- RAD model (RAD模型), 81
 drawbacks (缺陷), 83
 Recovery testing (恢复测试), 409
 Reengineering (再工程), 873
 economics of (的经济学), 887
 process model (过程模型), 874
 Refactoring (重构), 112, 270
 Refinement (求精, 精化), 269
 Regression testing (回归测试), 401
 Relationship-Navigation analysis(RNA) (关系 - 导航分析), 553
 Release reuse equivalency principle(REP) (发布复用等价原则), 333
 Reliability (可靠性), 464, 762
 measures (测量), 763
 Repository (中心存储库), 775
 characteristics of (的特征), 778
 content (内容), 778
 SCM (软件配置管理), 777
 Requirements analysis (需求分析), 208. *See also Analysis*
 objectives of (的目标), 209
 Web engineering (Web工程), 540
 Requirements elicitation (需求诱导), 184
 Requirements engineering (需求工程), 174
 tasks (任务), 176
 Requirements gathering (需求收集)
 collaborative (协作的), 185 (*See also Elicitation*)
 guidelines (指导原则), 185
 team (团队), 186
 Requirements gathering, WebApps (需求收集, Web应用), 517
 Requirements management (需求管理), 180
 Requirements tracing (需求跟踪), 780

- Requirements validation checklist (需求确认检查单), 179
Requirements, validation of (需求, 确认), 203
Resources, environmental (环境资源), 679
Resources, human (人力资源), 678
Resources, project (项目资源), 677
Response time (响应时间), 378
Responsibilities, CRC definition (职责, CRC定义), 241
Restructuring (重构)
 code (代码), 882
 data (数据), 883
Reusability (可复用性), 91
Reuse (复用, 重用), 39, 132, 280, 333, 861
 analysis and design (分析和设计), 858
 environment (环境), 861
Reverse engineering (逆向工程), 877
 data (数据), 880
 processing (处理), 880
 user interfaces (用户界面), 881
Reviews. *See also* Formal technical reviews(FTR) (评审,
 参见正式技术评审)
 guidelines (指导原则), 756
 meeting format (会议形式), 754
 recordkeeping (记录保持), 755
 reporting (报告), 755
 sample driven (样本驱动), 757
 summary report (总结报告), 755
Risk (风险), 137
 components (构件), 731
 CTC format (条件-变迁-结果 (CTC) 格式), 737
 definition of (的定义), 727
 drivers (驱动), 731
 identification (标识), 729
 impact (影响), 732, 735
 planning (计划/策划), 737
 projection (投影), 732
 refinement of (的求精), 737
 strategies (策略), 726
 types of (的类型), 727
Risk assessment (风险评估), 730, 735
Risk exposure (风险曝光度), 736
 computation of (的计算), 735
Risk information sheets (风险信息表单), 735, 740
Risk item checklist (风险条目检查表), 729
Risk management (风险管理), 726
 principles of (的原则), 729
Risk mitigation (风险缓解), 738
Risk monitoring (风险监测), 738
Risk table (风险表), 732
RMMM Plan (风险缓解、监测和管理计划), 739, 740
- S**
- SafeHome (SafeHome项目)
 ACD (体系结构上下文图表), 300
 activity diagram (活动图), 224, 552
 analysis classes (分析类), 235, 548
 architectural structure (体系结构), 302, 316, 319
 class diagram (类图), 198, 239
 class relationships (类关系), 301
 component design (构件设计), 351
 conceptual schema (概念模型), 587
 content objects (内容对象), 576
 CRC index card (CRC索引卡片), 240
 CRC model (CRC模型), 245
 data flow diagram (数据流图), 227, 228, 310
 data tree (数据树), 547
 deployment diagram (部署图), 279
 design classes (设计类), 273
 NSU (网络支持共用程序), 583
 processing narrative (处理叙述), 227
 PSPEC (处理规格说明), 232
 screen layout (屏幕布局), 376
 security function structure (安全功能结构), 302
 sequence diagram (顺序图), 252, 549
 state diagram (状态图), 231, 251, 550
 swimlane diagram (泳道图), 225
 use-case diagram (用例图), 195, 223
 use-cases (用例), 193, 220, 374, 543
SafeHome (夹在正文中的插入框, 以SafeHome项目为
 例说明了软件工程过程), 48, 85, 90, 113, 135,
 163, 188, 190, 195, 199, 202, 219, 222, 231, 238,
 245, 270, 273, 295, 305, 315, 332, 336, 338, 359,
 368, 392, 407, 413, 423, 428, 447, 471, 483, 518,
 529, 543, 569, 611, 638, 653, 664, 685, 701, 721,
 736, 758, 786
Sample driven reviews (样本驱动评审), 757
Scalability, WebApps (可伸缩性, Web应用), 562

- SCAMPI (过程改进的标准CMMI评估方法), 67
- Scheduling (进度安排), 705, 708, 716
- concepts (概念), 706
 - principles (原则), 709
 - tracking (跟踪), 718
 - variance (偏差), 722
- SCM (软件配置管理), 772
- elements of (的元素), 774
 - functions (功能), 779
 - identification (标识), 781
 - process (过程), 780
 - process layers (过程层次), 781
 - scenario (场景), 773
 - standards (标准), 797
 - Web engineering (Web工程), 788
- Scope (范围), 136
- Scrum (Scrum敏捷模型), 117
- meetings (会议), 119
 - principles (原则), 117
- Security (安全性), 409, 503
- testing (测试), 409, 617
 - WebApps (Web应用), 561
- Self-organization (自组织), 109
- Semantic domain (语义域), 815
- Sequence diagrams (顺序图), 251
- WebApps (Web应用), 549
- Sequences (序列), 812
- Set operators (集合操作符), 810
- Similarity (相似性), 480
- Six Sigma (六西格玛), 761
- Size (规模), 480
- Sizing software projects (定义软件项目规模), 681
- Smoke testing(冒烟测试), 401
- Software (软件)
- characteristics of (的特征), 37
 - definition of (的定义), 36
 - evolving role of (的角色演化), 34
 - fundamental questions (根本问题), 36
 - historical references (历史引用), 35
 - legacy applications (遗产应用系统), 42
 - myths about (关于……的神话), 45
- Software, application categories (软件, 应用系统的分类), 40
- Software capital (软件资产), 52
- Software configuration (软件配置), 772
- Software configuration items(SCIs) (软件配置项), 775
- Software Configuration Management. *See* SCM (软件配置管理, 参见SCM)
- Software engineering (软件工程)
- cleanroom (净室), 828
 - definition of (的定义), 53
 - ethics (道德规范), 900
 - future of (的将来), 892
 - layers (层), 54
 - practice (实践), 128. *See also* Practice principles (原则), 131
- Software equation (软件方程式), 694, 711
- Software evolution (软件演化), 43
- laws of (的规律), 44
- Software increments (软件增量), 80, 107
- Software maintenance (软件维护), 39, 43, 873. *See also* Maintenance
- Software metrics. *See* Metrics (软件度量, 参见度量)
- Software process improvement(SPI) (软件过程改进), 650
- statistical (统计的), 652
- Software project management. *See* Project management (软件项目管理, 参见项目管理)
- Software projects. *See* Projects (软件项目, 参见项目)
- Software Quality. *See* Quality (软件质量, 参见质量)
- Software quality assurance(SQA). *See* Quality assurance (软件质量保证, 参见质量保证)
- Software reviews (软件评审), 751. *See also* Reviews
- Software safety (软件安全性), 739, 763
- Software scope (软件范围), 638, 677
- bounding (边界), 677
- Software teams. *See* Teams (软件团队, 参见团队)
- Software testing. *See* Testing (软件测试, 参见测试)
- Software tools. *See* Tools (软件工具, 参见工具)
- Source code (源代码)
- metrics (度量), 490
 - program level (程序层次), 490
 - volume (信息量), 490
- Specification (规格说明), 179
- SPICE (SPICE), 66
- Spike solution (Spike解决方案), 112
- Spiral model (螺旋模型), 86

- problems with (的问题), 88
Sprint (冲刺), 119
SQA Plan (SQA计划), 766
Stakeholder (共利益者), 56, 631
 identification of (的标识), 182
 multiple viewpoints (多角度), 182
State (状态), 805
State diagram (状态图), 199, 231, 250, 550
 testing using (采用……测试), 451
State-box specification (状态盒规格说明), 835
Statechart (状态图), 344
Statistical SQA (基于统计的SQA), 759
Statistical use testing (统计应用测试), 842
Status reporting (状态报告), 788
Stepwise refinement (逐步求精), 269
Stereotype (构造型), 246
Stress testing (压力测试), 409, 620
Structural modeling (结构建模), 853
Structure chart (结构图), 328
Structure points (结构点), 853
 characteristics of (的特征), 853
 cost analysis (成本分析), 863
Structured constructs (结构化构造), 348
Structured programming (结构化程序设计), 348
Stubs (桩模块), 396
Subproofs (子证明), 838
Sufficiency (充分性), 480
Support (支持), 148
Surface structure (表层结构), 446
System (系统)
 definition of (的定义), 155
 macro element (宏元素), 156
System context diagram (系统环境图), 165
System elements (系统元素), 155
System engineering (系统工程), 154
 hierarchy (层次), 157
 world view (全局视图), 157
System flow diagram(SFD) (系统流图), 166
System model template (系统模型模板), 165
System modeling (系统建模), 158, 164
 restraining factors (限制因素), 159
 UML (统一建模语言), 167
System simulation (系统仿真), 160
System software (系统软件), 40
System testing (系统测试), 391, 408
- T
- Task analysis (任务分析), 367
Task elaboration (任务细化), 368
Task network (任务网络), 715
Task set (任务集)
 analysis modeling (分析建模), 141
 communication (沟通), 135
 construction (构造), 146
 customer communication (客户沟通), 521
 definition of (的定义), 57
 deployment (部署), 149
 design (设计), 144
 planning (计划/策划), 139
 project planning (项目计划), 676
 refinement of (的求精), 714
 selecting (选择), 713
 testing (测试), 147
 WebApp testing (WebApp测试), 601
Team leader (项目组负责人), 632
Team software process(TSP) framework activities (团队
 软件过程框架活动), 70
 objectives (目标), 70
 scripts (脚本), 71
Team toxicity (团队缺陷), 635
Teams (团队)
 agile (敏捷), 108, 636
 chief programmer (主程序员), 634
 coordination (协调), 637
 jelled (有凝聚力的), 109, 115, 635
 self organizing (自组织的), 109
 types of (的类型), 634
 Web engineering (Web工程), 523
Technologies, future (未来的技术), 894, 899
Test cases (测试用例)
 characteristics of (的特征), 422
 deriving (导出), 428
Test planning (测试计划), 386, 598
Testability, characteristics of (可测试性, 的特征), 421
Testing (测试)
 basis path (基本路径), 425

- behavioral models (行为模型), 450
 boundaries (边界), 438
 class hierarchy (类层次), 444
 class-based (基于类的), 447
 cleanroom software engineering (净室软件工程), 841
 client/server (客户/服务器), 452
 completion criteria (完成的标准), 392
 component level (构件级), 611, 600
 content (内容), 599, 601
 control structure (控制结构), 432
 conventional architectures (传统体系结构), 390
 data flow modeling (数据流建模), 437
 database (数据库), 603
 deep structure (深层结构), 446
 documentation (文档), 453
 equivalence partitioning (等价划分), 437
 exhaustive (彻底的), 424
 fault-based (基于故障的), 443
 finite state modeling (有限状态建模), 437
 generic characteristics (通用特征), 387
 graph-based (基于图的), 435
 GUIs (图形用户界面), 452
 help facilities (帮助功能), 453
 interfaces (界面), 599, 605, 606, 608
 metrics for (的度量) 491
 multiple class (多类), 449
 navigation (导航), 600
 OO architectures (OO体系结构), 391
 OO methods (OO方法), 442
 OO strategies (OO策略), 404
 options (选择), 402
 orthogonal array (正交数组), 439
 partition (划分), 448
 patterns (模式), 456
 principles (原则), 146
 process for WebApps (Web应用的过程), 598
 random (随机), 447
 real-time systems (实时系统), 454
 scenario-based (基于场景的), 444
 specialized (特定的), 452
 statistical use (统计使用), 842
 strategies (策略), 387, 393
 surface structure (表层结构), 446
 task set (任务集), 147
 techniques (技术), 420
 thread-based (基于线程的), 405
 transaction flow modeling (事务流建模), 436
 usability (可用性), 608
 use-based (基于使用的), 405
 WebApps (Web应用), 594
 Time-boxing (时间盒), 720
 Timeline chart (时间表), 717
 Time-to-market, WebApps (WebApps投入市场时间), 562
 Tools (工具), 54
 ADLs (体系结构描述语言), 307
 agile development (敏捷开发), 123
 analysis modeling in UML (UML分析建模), 253
 BPR (业务过程再工程), 872
 CBD (基于构件的开发), 862
 change management (变更管理), 795
 content management (内容管理), 793
 data mining (数据挖掘), 290
 data modeling (数据建模), 216
 data warehousing (数据仓库), 290
 debugging (调试), 415
 estimation (估算), 698
 formal methods (形式化方法), 822
 middleware (中间件), 330
 PDL (程序设计语言), 352
 process management (过程管理), 94
 process modeling (过程建模), 72
 product metrics (过程度量), 493
 project and process metrics (项目及过程度量), 661
 project management (项目管理), 645
 quality management (质量管理), 766
 requirements engineering (需求工程), 181
 reverse engineering (逆向工程), 881
 restructuring (重构), 883
 risk management (风险管理), 741
 scheduling (进度安排), 717
 SCM support (SCM支持), 788
 structured analysis (结构化分析), 233
 system modeling (系统建模), 171
 system simulation (系统仿真), 160
 test case design (测试用例设计), 441
 test planning (测试计划), 410

UML/OCL (UML/OCL), 347
use-cases (用例), 196
user interface development (用户界面开发), 380
Web engineering project management (Web工程项目管理), 532
WebApp metrics (WebApp度量), 589
WebApp testing (WebApp测试), 621
Traceability tables (跟踪表), 180
Transaction (事务), 308
Transaction center (事务中心), 316
Transaction mapping (事务映射), 316
Transform flow (变换流), 308
Transform mapping (变换映射), 309

U

Ubiquitous computing (遍在计算), 41
Umbrella activities (普适性活动), 54, 57
UML (UML)
activity diagram (活动图), 168, 197, 224, 343
class diagram (类图), 170, 198
collaboration diagram (协作图), 340
component elaboration (构件细化), 327
deployment diagram (部署图), 168, 279
interface representations (界面表示), 277, 341
OCL (OCL), 345, 816
packages (包), 248
sequence diagrams (顺序图), 252
state diagram (状态图), 199, 231, 251
statechart (状态图), 344
stereotype (构造型), 246
swimlane diagram (泳道图), 224, 371
system modeling diagram (系统建模图), 168
use-case diagram (用例图), 170, 195, 542
Unified Process(UP) (统一过程), 94
history of (的历史), 95
phases (阶段), 96
work products (工作产品), 98
Unit test (单元测试), 113
Unit testing (单元测试), 391, 394
considerations (考虑), 395
environment for (的环境), 397
procedures (规程), 396
Usability (可用性), 361, 464, 663

testing (测试), 608
Use-case diagram (用例图), 170
Use-cases (用例), 96, 367
development of (的开发), 191
identifying events (识别事件), 249
metrics (度量), 659
questions about (关于……的问题), 192
task analysis (任务分析), 335 WebApps (Web应用), 521, 542, 549
writing, 218
User analysis, methods for (用户分析, 的方法), 367
User categories, definition of (用户分类, 的定义), 517
User hierarchy (用户层次), 541
User interface (用户界面), 356. *See also under interface analysis* (分析), 365
analysis model (分析模型), 362
consistency (一致性), 360
control mechanisms (控制机制), 571
display content (显示内容), 372
memory load (记忆负担), 359
patterns (模式), 376
prototype (原型), 551, 572
reverse engineering (逆向工程), 881
testing (测试), 605
user control (用户控制), 357
User interface design (用户界面设计), 356
evaluation of (的评估), 381
golden rules (黄金规则), 357
issues (问题), 377
metrics (度量), 489
principles of (的原则), 358
process (过程), 363
steps (步骤), 373
task analysis (任务分析), 367
workflow analysis (工作流分析), 370
User scenarios (用户场景), 189
User stories (用户故事), 110

V

Validation (确认), 179, 388
Validation testing (确认测试), 391, 406
criteria (标准), 406
principles (原则), 145

Variation control (差异控制), 745

Verification (验证), 388

Version control (版本控制), 779, 782

 WebApps (Web应用), 795

Vital few (重要的少数), 760

Volatility (易变性), 480

W

W⁵HH Principle (W⁵HH原则), 138, 644

Walkthrough (走查), 751. *See also* Reviews

Waterfall model (瀑布模型), 79

 problems with (的问题), 79

Ways of navigation(WoN) (导航路径), 582

Wear (磨损), 37

Web applications. *See* WebApps (Web应用, 参见
 WebApps)

Web engineering(WebE) (Web工程), 500

 analysis modeling (分析建模), 539

 architectural design (体系结构设计), 577

 basic questions (基本问题), 509

 best practices (最佳实践), 510

 component-level design (构件级设计), 584

 content design (内容设计), 575

 design metrics (设计度量), 588

 estimation (估算), 697

 formulation (表达), 514 (*See also* Formulation)

 framework activities (框架活动), 507

 in-house development (内部开发), 530

 interface design (界面设计), 565

 methods (方法), 505

 metrics (度量), 532

 navigation design (导航设计), 581

 outsourcing (外包), 526

 planning (计划/策划), 522

 process (过程), 504, 506, 508

 quality guidelines (质量指导原则), 511

 SCM (软件配置管理), 788

 team building (建立团队), 524

 team characteristics (团队特点), 523

 testing (测试), 594, 597, 601

 tools (工具), 506

use-cases (用例), 542

worst practices (最差实践), 534

Web Engineering, design pyramid (Web工程, 设计金字塔), 564

Web sites, well designed (网站, 设计充分的), 575

WebApps (Web应用), 41, 501

 aesthetics (美学), 503

 analysis model (分析模型), 545

 attributes of (的属性), 502

 business value metrics (商业价值度量), 534

 change management (变更管理), 793

 configuration objects (配置对象), 790

 content management (内容管理), 790

 database layers (数据库层), 604

 design of (的设计), 559, 663

 errors (错误), 596

 metrics (度量), 588

 navigation testing (导航测试), 613

 performance testing (性能测试), 619

 quality attributes (质量属性), 560

 quality dimensions (质量维度), 595

 requirements analysis (需求分析), 539

 SCM (软件配置管理), 788

 security testing (安全测试), 617

 types of (的类型), 504

 user hierarchy (用户层次), 541

 version control (版本控制), 795

White-box testing (白盒测试), 424

Wisdom (智慧), 898

Work breakdown structure(WBS) (工作分解结构), 716

Work environment (工作环境), 373

Workflow analysis (工作流分析), 370

Wrapping (包装), 855

X

XP (极限编程). *See* Extreme Programming

Z

Z Specification language (Z规格说明语言), 820

 example (实例), 820

 notation (符号/表示法), 821