HZ BOOKS 华章教育

计 算 机 科 学 丛 书

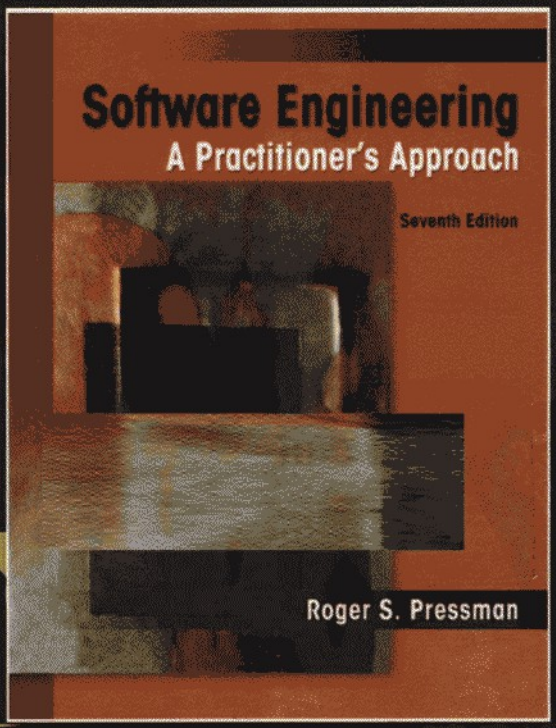
Mc Graw Hill Education

原书第7版

软件工程
实践者的研究方法

（美）Roger S. Pressman 著　郑人杰 马素霞 等译

Software Engineering
A Practitioner's Approach Seventh 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7版）

Software Engineering A Practitioner's Approach Seventh Edition

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，叙述清晰而又具有权威性。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

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程的重要教学参考书。近30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置疑的权威地位。它在全面而系统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。

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7版在结构和内容上均有调整、更新和充实，论述了很多人们称之为"21世纪工程学科"的重要主题。第7版更加突出软件过程，强调普遍使用的软件工程方法。

对第7版的内容做了如下划分，这样更便于课堂教学及自学使用：

- 第一部分 软件过程，介绍了说明性模型和敏捷过程模型。
- 第二部分 建模，介绍了现代分析与设计方法，新的重点放在基于UML的建模方面。
- 第三部分 质量管理，是第7版中新增加的内容，描述软件测试、质量保证、形式化验证技术和变更管理的各个方面。
- 第四部分 软件项目管理，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，用专门的章节讲述软件过程改进及将来的软件工程趋势。

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

封面设计：锡 彬

Mc Graw Hill Education
www.mheducation.com

上架指导 计算机 软件工程

ISBN 978-7-111-33581-8

9 787111 335818

定价：79.00元

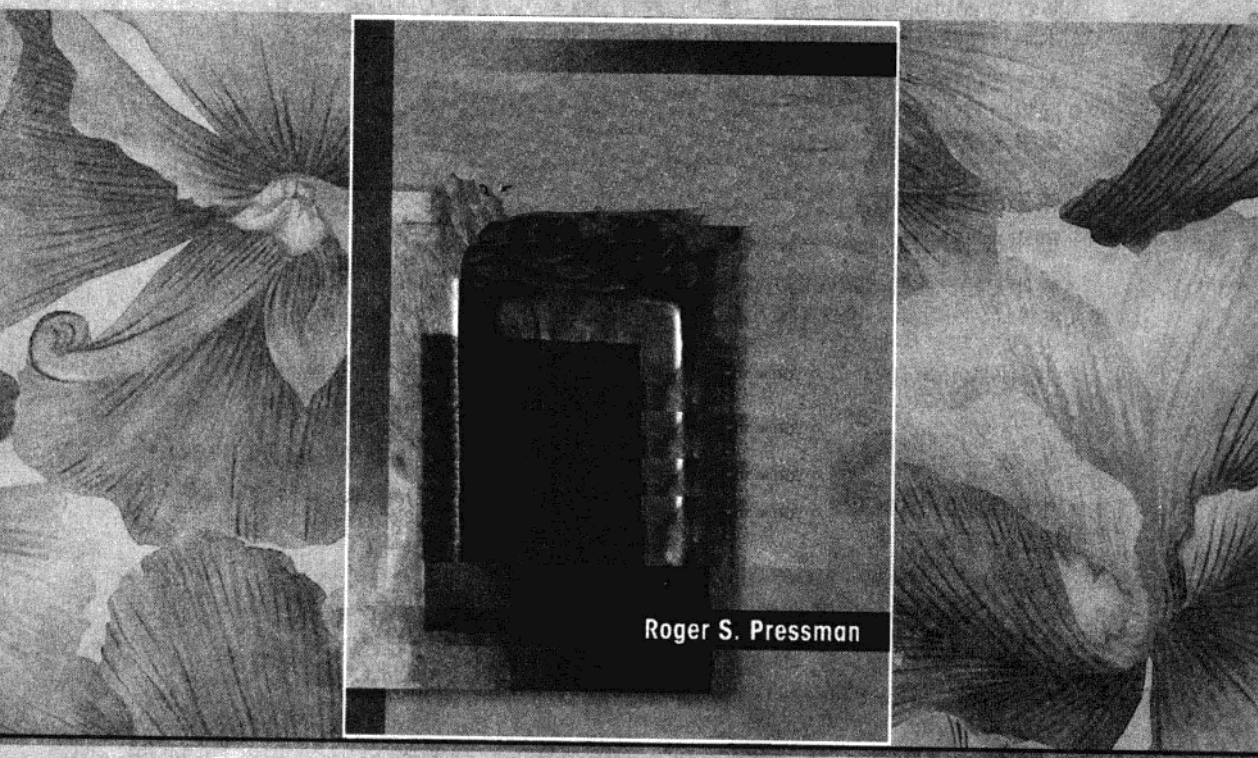
计 算 机 科 学 丛 书

原书第7版

软件工程
实践者的研究方法

（美）Roger S. Pressman 著　郑人杰 马素霞 等译

Software Engineering
A Practitioner's Approach Seventh Edition



机械工业出版社
China Machine Press

本书自近30年前第1版问世以来，在软件工程界始终发挥着巨大而深远的影响，其权威性是公认的、无可置疑的。第7版绝不是前一版的简单更新，它包含了很多新的内容，而且调整了全书的结构，以改进教学顺序，同时更加强调一些新的、重要的软件工程过程和软件工程实践知识。全书分软件过程、建模、质量管理、软件项目管理和软件工程高级课题五个部分，系统地论述了软件工程领域最新的基础知识，包括新的概念、原则、技术、方法和工具，同时提供了大量供读者进一步研究探索的参考信息。

本书适合作为本科生和研究生的软件工程及相关课程的教材，新版中五个部分的划分有利于教师根据学时和教学要求安排教学，同时也适合作为软件专业人员的工作指南，即使是资深专业人员，阅读本书也能获益匪浅。

本书版权登记号：图字：01-2009-4561

图书在版编目（CIP）数据

软件工程：实践者的研究方法（原书第7版）/（美）普雷斯曼（Pressman, R. S.）著；郑人杰等译．—北京：机械工业出版社，2011.4

（计算机科学丛书）

书名原文：Software Engineering: A Practitioner's Approach, Seventh Edition

ISBN 978-7-111-33581-8

Ⅰ.软… Ⅱ.① 普… ② 郑… Ⅲ.软件工程—高等学校—教学参考资料 Ⅳ.TP311.5

中国版本图书馆CIP数据核字（2011）第031569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

北京京北印刷有限公司 印刷

2011年5月第1版第1次印刷

185mm×260mm · 41.5印张

标准书号：ISBN 978-7-111-33581-8

定价：79.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991，88361066

购书热线：（010）68326294，88379649，68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

出版者的话

Software Engineering：A Practitioner's Approach (Seventh Edition)

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，还揭示了学术的源变，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和发展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到积极的推动作用，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公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson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、研究及珍藏。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”已经出版了近两百个品种，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。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

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。随着计算机科学与技术专业学科建设的不断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将步入一个新的阶段，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。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：

华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

联系电话：（010）88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037

华章教育

华章科技图书出版中心

译 者 序

Software Engineering：A Practitioner's Approach (Seventh Edition)

本书是国际知名软件工程专家Roger S. Pressman最近编著、并由美国McGraw-Hill出版社出版的《Software Engineering：A Practitioner's Approach》第7版的译本。该书自近30年前第1版问世以来，在软件工程界始终发挥着巨大而深远的影响，其权威性是公认的、无可置疑的，它在培养软件工程专业人才方面所起的作用也是显而易见的。

本人自上世纪80年代中期开始从事高校软件工程课的教学工作，多年来一直是该书各个版本的忠实读者。这本书已成为我的重要教学参考，给了我许多启发和帮助。

如今基于计算机的系统已经广泛而深入地渗透到国民经济、国防和人们日常生活的各个领域。计算机软件已成为许多关键系统的核心，甚至是灵魂，其作用越来越突出。如何提供合格、优良软件的问题必须受到重视。本书系统地论述了软件工程领域最新的基本知识，包括新的概念、原则、技术、方法和工具。同时书中还提供了许多供读者进一步研究的线索。

与五年前的第6版相比，本书继承了一些优点，同时也做了不少改动、扩充和更新。

第7版特点

1. 全书内容分为五个部分，共32章，还有两个附录。五个部分涉及：软件过程、建模、质量管理、软件项目管理以及软件工程高级课题。

2. 与前一版本相比，本书在内容上更加突出了质量管理，将其作为全书的五个部分之一。并且将软件过程改进作为软件工程的高级课题之一，专门在一章里作了深入的论述。此外，有关Web工程的内容也从原来版本中单列一部分共五章的内容，改为分散到各个相关章节之中。

3. 在各章的最后仍然给出了小结、习题与思考题以及推荐读物与阅读信息，这些都非常适合有兴趣、有需要的读者沿着这些线索开展进一步的学习和研究。

4. 本书仍然保留了历次版本的版面格式传统，即除了各章节文中插入的图表外，还穿插了许多方框，框中内容丰富多彩、形式多样，它们非常有利于理解相关的内容。这些方框包括：

- 各章开头的“要点浏览”和“关键概念”。
- 全书各章贯穿了一个统一的实例：住宅安全系统（SafeHome）的开发人员对话。
- 各章文中夹有专题性注释框，为读者提供了专题信息，如：

  “任务集”——应开展工作的说明；

  “信息”——专门术语的解释；

  “软件工具”——作者推荐的最新市售软件工具。
- 页边注，给出了“关键概念”、“引述”、“网上参考”、“建议”和“问题”等解释或信息。

读者对象

本书仍然面向三类读者，即高校学生（特别是研究生）、教师和软件专业人员。总体上，本书适合

于高校计算机相关专业教学，为软件工程课的教学服务。

教师若以本书作为教材，有以下几点建议：

1. 由于学时有限，不可能将全部内容纳入教学，从中抽取适合的部分是必然的。或许有关管理的部分要作压缩，但译者以为即使如此，也不应把管理的内容完全删除。

2. 目前敏捷开发方法在国内开始流行，但对于初学者或本科生而言，尚需更为重视和注意掌握传统的、严谨的开发方法。

3. 软件工程课的实践环节不可缺少，教师如何把书中的内容，特别是基本的概念、原则和方法结合到练习或其他形式的实践中，值得认真思考。

参加本书翻译工作的以华北电力大学和清华大学的教师为主，包括马素霞（第17～21章，附录）、宋兰（第5～7章）、韩新启（第14～16章）、王素琴（第24～26章）、谢萍（第27～29章）、胡海涛（第30～32章）、白晓颖（第1、2章）、董渊（第3、4章）、石敏（第8、9章）、周长玉（第10、11章）、金花（第12、13章）、马应龙（第22、23章）。在翻译过程中，得到了清华大学计算机系宋克清、毛苗同学及华北电力大学控制与计算机工程学院董哲、王琰洁、孙胜晶、赵东旭、秦贞远同学的帮助，内蒙古大学的郇失宇老师专门抽出时间对第3、4两章进行了审阅，在此对他们的辛勤劳动表示感谢。本人将全部译稿、马素霞教授将大部分译稿作了仔细审核与修改。尽管已经尽了最大的努力，但限于水平，对内容的理解和中文表达难免有不当之处，敬请读者批评指正。另外，原书中个别的问题（包括错误及不妥之处）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，本人十分高兴地向国内读者推荐。我们相信，认真阅读它，会使你获益匪浅。

郑人杰

2011年1月

前　言

Software Engineering：A Practitioner's Approach (Seventh Edition)

成功的计算机软件能够很好地满足使用者的要求，能在相当长时间内无故障地运行，容易修改还很好用，这样的软件能够也确实会把事情办好。但是，如果软件做得不好，用户就会不满意，它经常出错，难于修改，甚至难于使用，就可能（也的确）会把事情办糟。我们当然希望开发出好的软件，把事情办好，避免那些潜在的糟糕事情发生。要获得成功，在设计和构建软件时需要有规范，需要采用工程化方法。

在本书第1版问世以来的近30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，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都已在全行业的所有环节成功采用。

尽管管理人员和工作在第一线的专业人员都承认，需要有更为规范的软件方法，但他们却始终在争论着应该采用什么样的规范。有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要服务于当今最为先进的技术，也仍然如此。许多专业人员和学生并不了解现代方法。于是所开发的软件质量很差，造成了严重的后果。此外，有关软件工程方法真实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。人们对软件工程的态度已经有所改变，研究工作已取得了进展，不过要成为一门完全成熟的学科还有大量的工作要做。

作者希望本书第7版成为引导读者进入正在成熟的工程学科的入门读物。和以前的六个版本一样，第7版对学生和专业人员同样具有很强的吸引力，它既是软件专业人员的工作指南，同时也是大学高年级学生和一年级研究生的综合性参考书。

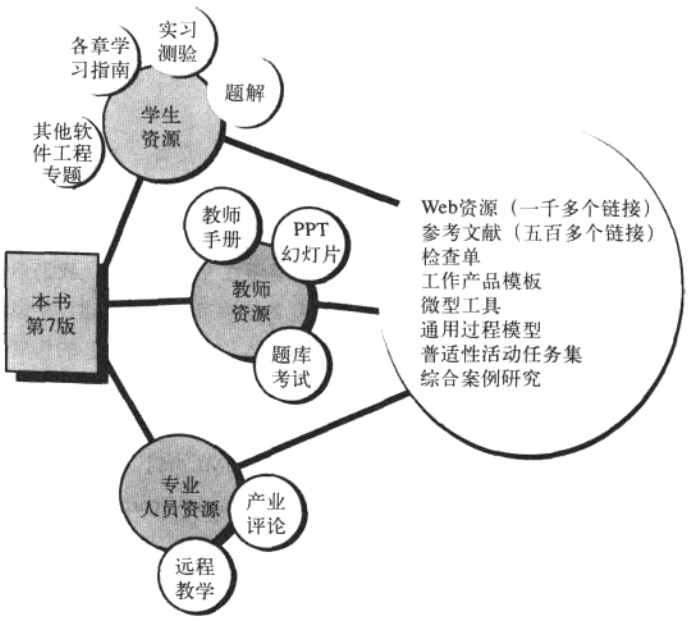
第7版中包含了很多新的内容，它绝不是前一版的简单更新。这一版不仅作了适当修改，而且调整了全书的结构，以改进教学顺序；同时更加强调一些新的和重要的软件工程过程和软件工程实践知识。此外，下面的图示表明了已作了修订与更新的“支持系统”，它为学生、教师和专业人员提供了大量的专业知识资源，从而丰富和充实了本书的内容。读者可查阅专为本书建立的网站（www.mhhe. com/pressman）获得这些资源。

第7版内容

第7版共有32章，分为5个部分。这种分法与第6版有很大不同，其目的在于帮助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，给出了软件过程的各种不同观点，考虑到所有重要的过程模型，还涉及惯用过程和敏捷过程在指导思想上的分歧。
- 第二部分　建模，给出了分析方法和设计方法，其中重点介绍了面向对象方法和UML建模。同时也考虑到基于模式的设计及Web应用系统的设计。
- 第三部分　质量管理，提供了有关质量管理的概念、规程、技术和方法，使得软件团队能够很好地

评估软件质量，评审软件工程工作产品，实施软件质量保证规程，并正确地运用有效的测试策略和战术。此外，这一部分还讨论了形式化建模和验证方法。



第7版支持系统

- 第四部分　软件项目管理，介绍了与计划、管理和控制软件开发项目的人员有关的问题。
- 第五部分　软件工程高级课题，考虑了软件过程改进和软件工程的发展趋势。
- 沿用前面几版的做法，全书各章中仍然使用了对话框（名为SafeHome），框中针对某个虚构的软件团队工作中遇到的困难展开对话，其目的是对相关各章的内容给出方法与工具的补充材料。
- 附录中为不熟悉UML和面向对象思想这两个重要主题的读者提供了简明的辅导。

第7版中五个部分的划分有利于教师根据学时和教学要求安排讲课内容。在一个学期内可以安排一个部分，也可以安排多个部分的内容。软件工程概论课可从五个部分中选择若干章作为教材。软件工程课侧重于分析和设计，其教学内容可从第一部分和第二部分中选取。面向测试的软件工程课则可从第一、第三部分选取，还应加上第二部分中的一些内容。管理课应突出第一部分和第四部分的内容。用上述方式组织第7版的内容，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7版的内容，都可获得“支持系统”的补充支持。

为学生提供的资源

为学生学习提供的各种材料有：在线学习中心提供的各章学习指南、实习测验、题解以及多种网上资源（包括软件工程检查单、一套正在演化的微型工具、综合案例研究、工作产品模板及其他）。此外，还有一千多种网上参考文献可供学生更为深入地探究软件工程问题，包含五百多篇可下载文章链接的参考文献库为读者提供高级软件工程课题的更为详尽的信息。

为教师提供的资源

本书第7版为教师提供了广泛的资源，包括一个在线教师指南（也可下载）及含有700个讲课用PPT幻灯片的教学辅助材料和试题库。当然所有这些资源（如微型工具、Web资源及参考文献）也可供学生和专业人员使用。

在本书的教师指南中，为各种类型的软件工程课提出了建议，介绍了与课程配合开展的软件项目、部分问题的题解和许多有用的教学辅助工具。

为专业人员提供的资源

有许多资源可供产业界专业人员（也包括在校师生）使用，包括软件工程文档和其他工作产品的大纲和模板、一套有用的软件工程检查单、软件工程（CASE）工具目录、综合性Web资源以及给出软件工程过程具体任务划分的“通用过程模型”。

有了在线支持系统的配合，使本书第7版既有内容上的深度，又有一定的灵活性，这些优势是单本教科书无法比拟的。

致谢

本书第7版的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，准备加以吸收和组织，并且对来自世界各地读者的意见和建议加以评估和分类。正是这个原因，我十分感谢这些书籍和文章（包括发表在纸制媒体和电子媒体上）的作者，在过去近30年中是他们给了我不少见解和想法。

我要特别感谢渥太华（Ottawa）大学的Tim Lethbridge，他帮助我制作了UML和OCL的案例及配合本书内容的案例研究。Colby学院的Dale Skrien制作了附录1的UML教材。他们的帮助和意见都是十分珍贵的。还要特别感谢密歇根大学迪尔本分校（University of Michigan-Dearborn）的Bruce Maxim，他帮助我开发了与本书配套的网站。最后我想感谢第7版的审校人员，他们提出的有深度的评审意见和批评都非常有价值。

Osman Balci,
Virginia Tech University

Max Fomitchev,
Penn State University

Jerry (Zeyu) Gao,
San Jose State University

Guillermo Garcia,
Universidad Alfonso X Madrid

Pablo Gervas,
Universidad Complutense de Madrid

SK Jain,
National Institute of Technology Hamirpur

Saeed Monemi,
Cal Poly Pomona

Ahmed Salem,
California State University

Vasudeva Varma,
IIIT Hyderabad

使用本书早期版本的产业界专业人员、大学教授和学生塑造了本书第7版的内容，他们花了很多时间提出建议、批评和想法，在此向他们致意。另外，还要感谢世界各地许多产业界客户，他们教给我的要比我能够教给他们的还要多。

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了。最后，感谢我的妻子Barbara，她的宽容允许我花如此多的时间在办公室，她还鼓励我继续开展下一个版本的工作。

Roger S. Pressman

Roger S. Pressman是软件过程改进和软件工程技术领域的国际知名权威。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Edistalearning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，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》，并打算再写一部。

目 录

Software Engineering：A Practitioner's Approach (Seventh Edition)

第二部分 建模

第三部分 质量管理

第四部分 软件项目管理

软件和软件工程

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：可以在各种不同规模及体系结构的计算机上运行的程序，程序运行过程中产生的各种结果，以及各种描述信息，这些信息可以以硬拷贝或是各种电子媒介形式存在。软件工程包括过程、各种方法（实践）以及各类工具，以协助专业人员构建高质量的计算机软件。

人员及责任：软件工程师开发软件并提供技术支持，产业界中几乎每个人都直接或间接地使用软件。

重要性：软件之所以重要是因为它在我们的生活中无所不在，并且日渐深入到商业、文化和日常生活各个方面。软件工程之所以重要是因为软件工程使我们可以高效、高质量地构建复杂系统。

步骤：开发计算机软件就像开发任何成功的产品一样，需采用灵活、可适应的软件开发过程，完成可满足使用者需求的高质量的软件产品。这就是软件工程化方法。

工作产品：从软件工程师的角度来看，最终产品是计算机软件，包括程序、内容(数据)和各种工作产品；而从用户的角度来看，最终产品是可以改善生活和工作质量的最终信息。

质量保证措施：阅读本书的后面部分，选择切合你所构建的软件特点的思想，并在实际工作中加以应用。

关键概念
应用领域
软件特点
框架活动
遗留软件
实践
原则
软件工程
软件神话
软件过程
普适性活动
Web应用

我谈话的对象具有大型软件公司高级主管的典型特征——45岁左右、鬓角微白、身材匀称而健壮，说话时眼睛似乎能够洞穿倾听者的内心。但是，他说的话却让我感到震惊："软件已经死了。"

我惊奇地眨眨眼，随后笑道："你在开玩笑，是吗?软件已成为世界进步的驱动力，也给你们的公司带来了巨大的利益。软件没有死！它还活着，并且还在茁壮成长。"

他用力地摇了摇头说："不，软件已经死了……至少就像我们曾经知道的那样。"

我倾身向前："你继续讲！"

他敲着桌子以示强调，一边说道："根据学院派观点——购买软件、拥有软件，并将使用和管理软件作为一项工作，这种模式已经走到了尽头。今天，随着Web 2.0和普适计算的发展，我们即将看到全然不同的新一代软件产品。软件通过网络交付使用，看起来就像驻留在每个用户的计算设备上运行一样……但实际上，软件可能运行在遥远的服务器上。"

我不得不同意："因此，你的生活将变得更加简单。你们不必再为成千上万个用户交叉访问同一个应用系统的5个不同版本而担心。"

他笑道："千真万确。我们仅在服务器上运行当前最新版本的软件。每当对软件做了更新或改正，我们为每个用户提供更新的软件功能和内容，每个用户都可以立刻拥有最新的版本！"

我做了个鬼脸，说道："但如果你们犯了错误，每个用户也立刻受到影响。"

他轻声笑道："是的，这就是我们加倍努力做好软件工程的原因。问题是我们必须'快速'，因为在每个应用领域，市场都在加速发展变化。"

我坐直身体，把双手放在脑后说："你知道人们怎么说……快速、正确或者廉价，你只能选择其中两个！"

他边起身边说道："我会选择快速和正确。"

我也站起身来："那么你确实需要软件工程。"

"我知道，"他准备离开，边走边说："问题是我们必须意识到还需要新一代的高科技专家，并且确实如此。"

软件真的死了吗？如果是那样，你就不必阅读这本书了！

"创新观念和科技发现是经济增长的推进器。"——《华尔街日报》

计算机软件仍然是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。50年前，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；软件促进了新科技的创新（例如基因工程和纳米技术）、现代科技的发展（例如通信），以及传统技术的根本转变（例如印刷业）；软件技术已经成为个人电脑革命的推动力量；消费者可以很容易地在附近的商店购买到包装好的软件产品；软件还将由产品逐渐演化为服务，软件公司随需应变，通过Web浏览器发布即时更新功能；软件公司可以比几乎任何传统工业时代的公司更大、更有影响力；在大量应用软件的驱动下，互联网将迅速发展，并将对人们生活的诸多方面——从图书馆搜索、消费购物、政治演说到年轻人（或者不那么年轻的人）的约会习惯——引起革命性的变化。

没有人曾想到软件可嵌入到各种系统中：交通运输、医疗、通信、军事、工业、娱乐以及办公设备等不胜枚举。如果笃信"意外效应法则"的话，那么还有很多结果和影响是我们尚未预料到的。

没有人曾想到，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。

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。某些技术主要针对特定应用领域（例如，网站设计和实现）；有些着眼于技术领域（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，我们尚未开发出一种软件技术可以实现上述所有需求，而且未来能够产生这种技术的可能性很小。人们也尚未将其工作、享受、安全、娱乐、决策以及全部生活都完全依赖于计算机软件。这或许是正确的选择。

本书为需要构建正确软件的计算机软件工程师提供了一个框架。该框架包括过程、一系列方法以及我们称为软件工程的各种工具。

1.1 软件的本质

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交付载体的双重作用。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。无论是驻留在移动电话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、获取、修改、显示或者传输各种不同的信息，简单如几个比特的传递或复杂如从多个独立的数据源获取的多媒体演示。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。

"软件是播撒梦想和收获噩梦的地方，是一片恶魔和神仙相竞争的抽象而神秘的沼泽，是一个狼人和银弹共存的矛盾世界。"——Brad J.Cox

软件提供了我们这个时代最重要的产品——信息。它会转换个人数据（例如个人财务交易），使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。

在最近半个世纪里，计算机软件的作用发生了很大的变化。硬件性能的极大提高、计算机结构的巨大变化、内存和存储容量的大幅度增加、还有种类繁多的输入和输出方法都促使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。

现在，一个庞大的软件产业已经成为了工业经济中的主导因素。早期的独立程序员也已经被专业的软件开发团队所代替，团队中的不同专业技术人员可分别关注复杂的应用系统中某一个技术部分。然而同过去独立程序员一样，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[⊖]：

- 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仍旧难以度量？

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方法的采纳。

1.1.1 定义软件

今天，绝大多数专业人员和许多普通人认为他们已经大概了解软件。真的是这样吗？

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，使得程序可以合理利用信息；（3）软件描述信息，它以硬拷贝和虚拟形式存在，用来描述程序操作和使用。

如何定义软件？

当然，还有更完整的解释。

但是一个非常形式化的定义可能并不能显著改善其可理解性，为了更好地理解"软件"的含义，有必要将软件和其他人工产品的特点加以区分。软件是逻辑的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：

KEY POINT
软件是设计开发的，而非制造加工。

1. 软件是设计开发的，而不是传统意义上生产制造的

虽然软件开发和硬件制造存在某些相似点，但二者根本不同：两者均可通过优秀的设计获得高品质产品，然而硬件在制造阶段可能会引入质量问题，这在软件中并不存在（或者易于纠正）；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。

KEY POINT
软件不会磨损，但会退化。

⊖ 在一本优秀的关于软件商业的论文集中，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。"

2. 软件不会“磨损”

图1-1描述了硬件的失效率，该失效率是时间的函数。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，失效率随之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造成的硬件组件损耗累积的效果，使得失效率再次提高。简而言之，硬件开始“磨损”了。

ADVICE
若希望降低软件退化，需要改进软件的设计（第8～13章）。

而软件不会受引起硬件磨损的环境问题的影响。因此，从理论上来说，软件的失效率曲线应该呈现为图1-2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，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。曲线的含义很明显——软件不会磨损，但是软件退化的确存在。

KEY POINT
软件工程方法的目的是降低图1-2中向上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[⊖]，软件将会面临变更，每次变更都可能引入新的错误，使得失效率像“实际曲线”（图1-2）那样陡然上升。在曲线回到最初的稳定失效率状态前，新的变更会引起曲线又一次上升。就这样，最小的失效率点沿类似于斜线逐渐上升，可以说，不断的变更是软件退化的根本原因。

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部件替换，而软件却不存在备用部件。每个软件的缺陷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，软件维护要应对变更请求，比硬件维护更为复杂。

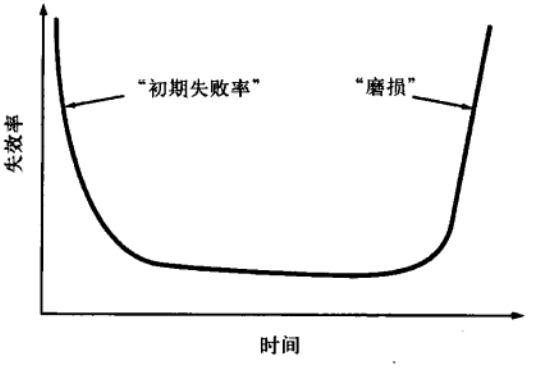


图1-1 硬件失效曲线图

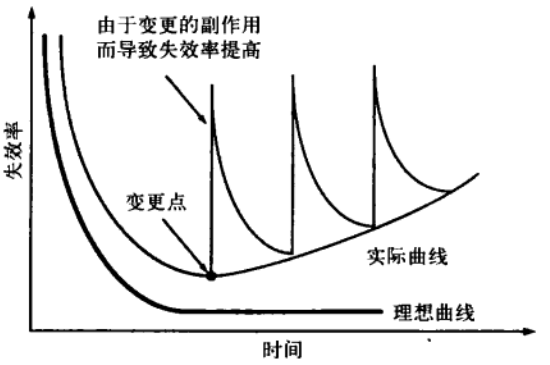


图1-2 软件失效曲线图

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

工程学科的发展将产生一系列标准的设计器件。标准螺丝钉及可订购的集成电路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器件中的两种。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，也就是，设计中真正新的内容。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。

⊖ 事实上，从软件开发开始，在第一个版本发布之前的很长时间，会有许多利益相关者提出变更要求。

软件的构件应该设计并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造新的应用程序[⊖]。例如，现在的交互式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，构造用户界面所需要的数据结构和处理细节被封装在用于用户界面设计的可重用构件库中。

1.1.2 软件应用领域

WebRef

目前最大的共享软件/免费软件库之一：shareware.cnet.com。

今天，计算机软件可分为七个大类，软件工程师正面临持续的挑战。

系统软件——系统软件是一整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、文件管理软件）处理复杂但确定的[⊜]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。无论何种情况，系统软件多具有以下特点：和计算机硬件大量交互；多用户大量使用；需要调度、资源共享和复杂进程管理的同步操作；复杂的数据结构以及多种外部接口。

应用软件——解决特定业务需要的独立应用程序。这类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功能的实时控制（例如：销售点的交易处理，实时制造过程控制）。

工程／科学软件——这类软件通常带着“数值计算”算法的特征，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，从自动应力分析到航天飞机轨道动力学，从分子生物学到自动制造业。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性甚至具有系统软件的特性。

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的特性和功能。嵌入式软件可以执行有限但难于实现的功能（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、刹车系统等汽车电子功能）。

产品线软件——产品为多个不同用户的使用提供特定功能。产品线软件关注有限的特定的专业市场（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——叫做“Web应用”（WebApp），是一类以网络为中心的软件，其概念涵盖了宽泛的应用程序产品。最简单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Web 2.0的出现，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独立的特性、计算功能和内容信息，还和企业数据库和商务应用程序相结合。

“没有任何一台计算机具备基本常识。”——Marvin Minsky

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。

全世界成百万的软件工程师在为以上各类软件项目努力地工作着。有时是建立一个新的系统；而有时只是对现有应用程序的纠错、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软件，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

⊖ 第10章讨论了基于构件的软件工程。

⊜ 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。

（参见第31章）也已经逐渐显现出来：

开放计算——无线网络的快速发展也许将很快促成真正的普适计算、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，以使得移动设备、个人电脑和企业应用可以通过大量的网络设施进行通信。

"你通常无法预测未来，但是可以时刻为未来做好准备。"
——无名氏

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。

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）代码开放，使得很多人能够为软件开发做贡献,这种方式正在逐步成为一种趋势。软件工程师面对的挑战是开发可以自我描述的代码，而更重要的是，开发某种技术，以便于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。

所有这些新的挑战毫无疑问将遵守"意外效应法则"，会产生现在无法预测的结果（对商务人员，软件工程师和最终用户）。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，以适应在未来十年中必将发生的技术和业务的种种巨大变化。

1.1.3 遗留软件

成千上万的计算机程序都可以归于在前一小节中讨论的7大类应用领域。其中一些是当今最先进的软件——最近才对个人、产业界和政府发布。但是另外一些软件则年代较久，甚至过于久远了。

这些旧的程序——通常称为遗留软件（legacy software）——从20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：

遗留软件系统……在几十年前开发，它们不断被修改以满足商业需要和计算平台的变化。这类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]进一步扩展了这个描述，指出："许多遗留软件系统仍然支持核心的商业功能，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。

如果遇到质量低下的遗留软件项怎么办？

对遗留软件进行哪些类型的改变？

然而不幸的是，遗留软件常常存在另一个特点——质量差[⊖]。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，变更历史管理混乱等，有着数不清的问题。然而，这些系统仍然支撑"核心的应用，并且是业务必不可少的支撑"。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，那么它就没有失效，不需要修改。然而，随着时间的推移，遗留系统经常会由于下述原因发生演化：

所有软件工程师都必须认识到，变化是不可避免的。不要反对变化。

- 软件需要进行适应性调整，从而可以满足新的计算环境或者技术的需求。
- 软件必须升级以实现新的商业需求。
- 软件必须扩展使之具有与更多新的系统和数据库的互操作能力。
- 软件架构必须进行改建使之能适应多样化的网络环境。

当这些变化发生时，遗留系统需要经过再工程（参见第29章）使之适应未来的多样性。当代软件工程的目标是"修改基于进化论理论建立的方法论"，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

⊖ 所谓"质量差"是基于现代软件工程思想的，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。

有系统都必须具有互操作性和协作性” [Day99]。

1.2 WebApp的特性

“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

WebApp与其他软件有哪些不同特性?

WWW的早期（大约从1990年到1995年），Web站点仅包含链接在一起的一些超文本文件，这些文件使用文本和有限的图形来表示信息。随着时间的推移，一些开发工具（例如，XML、Java）扩展了HTML的能力，使得Web工程师在向客户提供信息的同时也能提供计算能力。基于Web的系统和应用[⊖]（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。

如1.1.2节所述，WebApp是独特的软件类型中的一种，尽管可能存在质疑WebApp是否是不同的软件。Powell[Pow98]提出基于Web的系统和应用“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”。绝大多数WebApp具备下列属性：

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。网络提供开放的访问和通信（如因特网）或者受限的访问和通信（如企业内联网）。

并发性（concurrency）：大量用户可能同时访问WebApp。很多情况下最终用户的使用模式存在很大差异。

无法预知的负载量（unpredictable load）：WebApp的用户数量每天都可能有数量级的变化。周一显示有100个用户使用系统，周四就可能会有10000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、客户端格式化显示），该用户就可能转向其他地方。

可用性（availability）：尽管期望百分之百的可用性是不切实际的，但是对于热门的WebApp，用户通常要求能够24/7/365（全天候）访问。澳大利亚或亚洲的用户可能在北美传统的应用软件脱机维护时间要求访问。

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。除此之外，WebApp还常被用做访问那些存储在Web应用环境之外的数据库中的信息（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（content 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而言（特别是WebApp的内容），按分钟发布更新，或者对每个请求动态更新页面内容，这些都是司空见惯的事。

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[⊖]。

安全性（security）：由于WebApp是通过网络访问来使用的，因此要限制访问的最终用户的数量，即使可能也非常困难。为了保护敏感的内容，并提供保密的数据传输模式，在支持

⊖ 在本书中，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，到为商务旅行和度假提供全套旅游服务的复杂的Web站点。其中包括完整的Web站点、Web站点的专门功能以及Internet、Intranet或Extranet上的信息处理应用。

⊖ 应用先进的工具，我们只需几小时就能完成复杂网页的开发。

WebApp的整个基础设施上和应用本身内部都必须实施较强的安全措施。

美观性（aesthetics）：不可否认，WebApp的用户界面外观很有吸引力。是否能将产品或是思想成功地推向市场，界面设计的美观和技术设计同样重要。

可以这样说，1.1.2节中提到的其他应用类型可能具备上述某些特性，但WebApp几乎具备了所有属性。

1.3 软件工程

要构建能够适应21世纪挑战的软件产品，就必须认识到以下几个简单的事实：

KEY POINT 在制定解决方案之前要理解问题。

- 软件已经深入到我们生活的各个方面，其后果是，对软件应用[⊖]所提供的特性和功能感兴趣的人们显著增多。当要开发一个新的应用领域或嵌入式系统时，一定会听到很多不同的声音。很多时候，每个人对发布的软件应该具备什么样的软件特性和功能似乎都有着些许不同的想法。因此，在制定软件解决方案前，必须尽力理解问题。

KEY POINT 设计是一项关键的软件工程活动。

- 年复一年，个人、企业和政府的信息技术需求日臻复杂。过去一个人可以构建的计算机程序，现在需要由一个庞大的团队来共同实现。曾经运行在一个可预测、自包含的、特定计算环境下的复杂软件，现在可以嵌入到消费类电子产品、医疗设备、武器系统等各种环境上执行。这些基于计算机的系统或产品的复杂性，要求对所有系统元素之间的交互非常谨慎。因此，设计已经成为关键活动。

KEY POINT 质量和可维护性都来自于好的设计。

- 个人、企业、政府在进行日常运作管理以及战略战术决策时越来越依靠软件。软件失效会给个人和企业带来诸多不便，甚至是灾难性的失败。因此，软件必须保证高质量。
- 随着特定应用感知价值的提升，其用户群和软件寿命也会增加。随着用户群和使用时间的增加，其适应性和可扩展性需求也会同时增加。因此，软件需具备可维护性。

由这些简单事实可以得出一个结论：各种形式、各个应用领域的软件都需要工程化。这也是本书的主题——软件工程。

"工程不仅仅是一个学科或知识体，它是一个动词，一个行为动词，一个解决问题的方法。" —— Scott Whitmire

尽管有很多作者都给出了各自软件工程的定义，但Fritz Bauer[Nau69]在该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，以便经济地获得可靠的、可以在实际机器上高效运行的软件。

如何定义软件工程？

你也许试图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"获得"可靠的"软件？如何构建程序使其能够不是在一个而是在多个不同的"实际机器"上都能"高效运行"？这些都是对软件工程师提出进一步挑战的问题。

IEEE[IEE93a]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

⊖ 在本书的后续部分，将这些人统称为"利益相关者"。

⊜ 有关软件工程的其他定义，参见 www.answers.com/topic/software-engineering#wp-_note-13。

工程化方法应用于软件。(2) 在 (1) 中所述方法的研究。

KEY POINT
软件工程包括过程、方法和工具。

然而，对于某个软件开发队伍来说可能是“系统化的、规范的、可量化的”方法，对于另外一个团队却可能是负担。因此,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1-3所示）。任何工程方法（包括软件工程）必须构建在质量承诺的基础之上。全面质量管理、六西格玛和类似的理念[⊖]促进了不断的过程改进文化，正是这种文化，最终引导人们开发更有效的软件工程方法。支持软件工程的根基在于质量关注点（quality focus）。

WebRef
《Cross Talk》杂志提供了关于过程、方法和工具的许多具体的信息，网站地址是：www.s tsc. hill. af.mil。

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，使得合理、及时地开发计算机软件成为可能。过程定义了一个框架，构建该框架是有效实施软件工程技术必不可少的。软件过程构成了软件项目管理控制的基础，建立了工作环境以便于应用技术方法、提交工作产品（模型、文档、数据、报告、表格等）、建立里程碑、保证质量及正确管理变更。

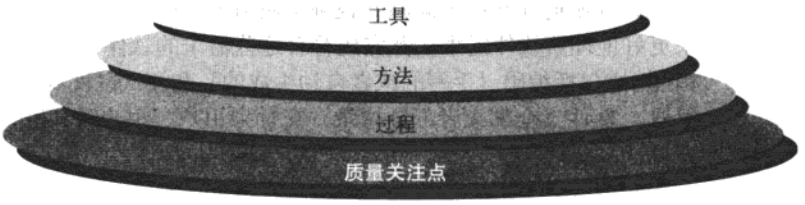


图1-3　软件工程层次图

软件工程方法（method）为构建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、测试和技术支持。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）。

1.4　软件过程

软件过程包含哪些要素？

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, Grandy Booch和James Rumbaugh

软件过程是工作产品构建时所执行的一系列活动、动作和任务的集合。活动（activity）主要实现宽泛的目标（如与利益相关者进行沟通），与应用领域、项目大小、结果复杂性或者实施软件工程的重要程度没有直接关系。动作（action）（如体系结构设计）包含了主要工作产品（如体系结构设计模型）生产过程中的一系列任务。任务（task）关注小而明确的目标，能够产生实际产品（如构建一个单元测试）。

在软件工程领域，过程不是对如何构建计算机软件的严格的规定，而是一种可适应性调整的方法，以便于工作人员（软件团队）可以挑选适合的工作动作和任务集合。其目标通常是及时、高质量地交付软件，以满足软件项目资助方和最终用户的需求。

⊖　质量管理和相关方法在本书第14章及第3部分进行讨论。

过程框架（process framework）定义了若干个框架活动（framework activity），为实现完整的软件工程过程建立了基础。这些活动可广泛应用于所有软件开发项目，无论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整个软件过程的普适性活动（umbrella activity）。一个通用的软件工程过程框架通常包含以下5个活动：

五个最基本的过程框架活动是什么？

沟通 在技术工作开始之前，和客户（及其他利益相关者[⊖]）的沟通与协作是极其重要的；其目的是理解利益相关者的项目目标，并收集需求以定义软件特性和功能。

策划 如果有地图，任何复杂的旅程都可以变得简单。软件项目好比是一个复杂的旅程，策划活动就是创建一个“地图”，以指导团队的项目旅程，这个地图称为软件项目计划，它定义和描述了软件工程工作，包括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。

“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

建模 无论你是庭园设计家、桥梁建造者、航空工程师、木匠还是建筑师，你每天的工作都离不开模型。你会画一张草图来辅助理解整个项目大的构想——体系结构、不同的构件如何结合，以及其他一些特征。如果需要，可以把草图不断细化，以便更好地理解问题并找到解决方案。软件工程师也是如此，利用模型来更好地理解软件需求，并完成符合这些需求的软件设计。

构建 它包括编码（手写的或者自动生成的）和测试以发现编码中的错误。

部署 软件（全部或者部分增量）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既适用于简单小程序的开发，也可用于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。不同的应用案例中，软件过程的细节可能差别很大，但是框架活动都是一致的。

对许多项目来说，随着项目的开展，框架活动可以迭代应用。也就是说，在项目的多次迭代过程中，**沟通**、**策划**、**建模**、**构建**、**部署**等活动不断重复。每次项目迭代都会产生一个软件增量（software increment），每个软件增量实现了部分的软件特性和功能。随着每一次增量的产生，软件逐渐完善。

软件工程过程框架活动由很多普适性活动来补充实现。通常，这些普适性活动贯穿软件项目始终，以帮助软件团队管理和控制项目进度、质量、变更和风险。典型的普适性活动包括：

软件项目跟踪和控制——项目组根据计划评估项目进度，并且采取必要的措施保证项目按进度计划进行。

KEY POINT
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。

风险管理——对可能影响项目成果或者产品质量的风险进行评估。

软件质量保证——确定和执行软件质量保证的活动。

技术评审——评估软件工程产品，尽量在错误传播到下一个活动之前，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利益相关者要求。同时，测量还可与其他框架活动和普适性活动配合使用。

软件配置管理——在整个软件过程中，管理变更所带来的影响。

⊖ 利益相关者（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就是掌握巨额投资（stake）的人…… 如不照看好你的Stakeholder，将失去投资”。

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——包括了生成产品（诸如建模、文档、日志、表格和列表等）所必需的活动。

上述每一种普适性活动都将在本书后面详细讨论。

KEY POINT 对软件过程的普适性调整是项目成功的关键。

在本节前面部分曾提到，软件工程过程并不是教条的法则，要求软件团队机械地执行；而应该是灵活可适应的（根据软件所需解决的问题、项目特点、开发团队和组织文化等进行适应性调整）。因此，不同项目所采用的项目过程可能有很大不同。这些不同主要体现在以下几个方面：

- 活动、动作和任务的总体流程，以及相互依赖关系。
- 在每一个框架活动中，动作和任务细化的程度。
- 工作产品的定义和要求的程度。
- 质量保证活动应用的方式。
- 项目跟踪和控制活动应用的方式。
- 过程描述的详细程度和严谨程度。
- 客户和利益相关者对项目参与的程度。
- 软件团队所赋予的自主权。
- 队伍组织和角色明确程度。

? 过程模型之间有哪些不同之处？

"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

本书第一部分将详细介绍软件过程。过程模型说明（prescriptive process model）（参见第2章）强调对过程活动和任务的详细定义、识别和应用。其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，这些目的往往并没有达到。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，并给开发人员和利益相关者制造麻烦。

? 敏捷过程的特点是什么？

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。

敏捷过程模型（agile process model）（参见第3章）强调项目的灵活性，并在一些基本原则的指导下，采用非正式的方式（支持者认为这并不会降低过程的有效性）执行软件过程。由于强调可操作性和可适应性，这些过程模型普遍具有敏捷的特征，对某些类型的项目很适用，尤其是Web应用开发。

1.5 软件工程实践

在1.4节中，介绍一种由一组活动组成的通用软件过程模型，建立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**——和普适性活动构成了软件工程工作的体系结构的骨架。但是软件工程的实践如何融入该框架呢？在以下几节里，读者将会对应用于这些框架活动的基本概念和原则有一个基本了解[⊖]。

ADVICE 你可能会觉得Polya的方法只是简单的常识，的确如此。但是令人惊奇的是，在软件世界里，很多常识常常不为人知。

1.5.1 实践的精髓

在现代计算机发明之前，有一本经典著作《How to Solve It》，在本书中，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（沟通和分析）。

⊖ 在本书后面对于特定软件工程方法和普适性活动的讨论中，你应该重读本章中的相关章节。

2. 计划解决方案（建模和软件设计）。

3. 实施计划（代码生成）。

4. 检查结果的正确性（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[改自Pol45]：

理解问题。虽然难于承认，但我们遇到的问题很多都源于我们的傲慢。我们只听了几秒钟就断言“好，我懂了，让我们开始解决这个问题吧”。不幸的是，理解一个问题不总是那么容易，需要花一点时间回答几个简单问题：

- 谁将从问题的解决中获益？也就是说，谁是利益相关者？
- 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？是否可以描述为更小、更容易理解的问题？
- 问题可以图形化描述吗？可以建立分析模型吗？

“在任何问题的解决方案中都会有所发现。”——George Polya

计划解决方案。现在你理解了要解决的问题（或者你这样认为），并迫不及待地开始编码。在编码之前，稍稍慢下来做一点点设计：

- 以前曾经见过类似问题吗？在潜在的解决方案中，是否可以识别一些模式？是否已经有软件实现了所需要的数据、功能、特征和行为？
- 类似问题是否解决过？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，子问题是否已有解决方案？
- 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。前面所创建的设计勾画了所要构建的系统的路线图。也许存在没有想到的路径，也可能在实施过程中会发现更好的解决路径，但是这个计划可以保证在实施过程中不至于迷失方向。需要考虑的问题是：

- 解决方案和计划一致吗？源码是否可追溯到设计模型？
- 解决方案的每个组成部分是否可以证明正确？设计和代码是否经过评审？或者更好的算法是否经过正确性证明？

检查结果。你不能保证你的解决方案是最完美的，但是你可以保证设计足够的测试来发现尽可能多的错误。为此，需回答：

- 能否测试解决方案的每个部分？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？是否按照项目共同利益者的需求进行了确认？

不足为奇，上述方法大多是常识。但实际上，有充足的理由说明，在软件工程中采用常识，将让你永远不会迷失方向。

1.5.2 一般原则

“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**沟通**），还有一些关注软件工程的动作（比如架构设计）或者技术任务（比如编制用例场景）。无论关注哪个层次，原则都可以帮助我们建立一种思维方式，进行扎实的软件工程实践。因此，原则非常重要。

David Hooker[Hoo96]提出了7个关注软件工程整体实践的原则，这里复述如下[⊖]：

⊖ 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。

在开始一个软件项目之前，应首先确保软件具有商业目标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意的过程，在软件设计中需要考虑很多因素。所有的设计都应该尽可能简洁，但不是过于简化。这有助于构建更易于理解和易于维护的系统。这并不是说那些特征甚至是内部特征应该以“简练”为借口而取消。的确，优雅的设计通常也是简洁的设计，简练也不意味着“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，这样做的回报是所得到的软件更易于维护且存在更少错误。

“简洁比所有巧妙的措词更加美妙。”——Alexander Pope (1688-1744)

第3原则：保持愿景

清晰的愿景是软件项目成功的基础。没有愿景，项目将会由于它有“两种或者更多种思想”而永远不能结束；如果缺乏概念的一致性，系统就好像是由许多不协调的设计补丁、错误的集成方式强行拼凑在一起……如果不能保持软件系统体系架构的愿景，将削弱甚至彻底破坏设计良好的系统。授权体系架构师，使其能够保持愿景，并保证系统实现始终与愿景保持一致，这对项目开发成功至关重要。

如果软件有价值，其价值在其生命周期中将发生变化。因此，软件必须构建成可维护的。

第4原则：关注使用者

有产业实力的软件系统不是在真空中开发和使用的。通常软件系统必定是由开发者以外的人员使用、维护和编制文档等，这就必须要让别人理解你的系统。因此，在需求说明、设计和实现时，经常要想到要让别人理解你所做的事情。对于任何一个软件产品，其工作产品都可能有很多读者。需求说明时应时刻想到用户；设计中始终想到实现；编码时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使用者，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

生命期持久的系统具有更高的价值。在现今的计算环境中，需求规格说明随时会改变，硬件平台几个月后就会淘汰，软件生命周期都是以月而不是以年来衡量。然而，真正具有“产业实力”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以这种路线设计的系统。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”，构建可以解决通用问题的系统，为各种可能的方案做好准备[⊖]，这很可能会提高整个系统的可复用性。

第6原则：计划复用

复用既省时又省力[⊜]。软件系统开发过程中，高水平的复用是很难实现的一个目标。代码和设计复用曾宣称是面向对象技术带来的主要好处，然而，这种投入回报不会自动实现。为达到面向对象（或是传统）程序设计技术所能够提供的复用性，需要有前瞻性的设计和计划。系

⊖　把这个建议发挥到极致可能很危险，设计通用方案会带来性能损失，并降低特定解决方案的效率。

⊜　尽管对于准备在未来的项目中复用软件的人而言，这种说法是正确的，但对于设计和实现可复用构件的人来说，复用的代价会很昂贵。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，在有些情况下，这些费用差别很难核实。

统开发过程中各种层面，都有多种技术实现复用。提前做好复用计划，将降低开发费用，并增加可复用构件以及构件化系统的价值。

第7原则：认真思考

这最后一条规则可能是最容易被忽略的。在行动之前清晰定位、完整思考通常能产生更好的结果。仔细思考，可以提高做好事情的可能性，而且也能获得更多的知识，明确如何把事情做好。如果仔细思考过后，还是把事情做错了，那么，这就变成了很有价值的经验。思考的一个副作用是学习和了解本来一无所知的事情，成为研究答案的起点。当明确的思想应用在系统中，就产生了价值。使用前六条原则需要认真思考，这将带来巨大的潜在回报。

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。

1.6 软件神话

"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

软件神话，即关于软件及其开发过程被人盲目相信的一些说法，可以追溯到计算技术发展的初期。神话具有一些特点，让人们觉得不可捉摸。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、有经验的从业人员拿来宣传。

今天，大多数有见地的软件工程师已经意识到软件神话的本质——它实际上误导了管理者和从业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，这一切难以改变，软件神话遗风犹在。

WebRef

软件项目经理网有助于人们澄清这些神话：www.spmn.com。

管理神话。承担软件职责的项目经理，像所有领域的经理一样，肩负着维持预算、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）。

神话：我们已经有了一本写满软件开发标准和规程的宝典。难道不能提供我们所需要了解的所有信息吗？

事实：这本宝典也许的确已经存在，但它是否已在实际中采用？从业人员是否知道这本书的存在呢？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？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，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）。

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。

事实：软件开发并不是像机器制造那样的机械过程。Brooks曾说过[Bro95]："在软件工程中，为赶进度而增加人手，只能使进度更加延误。"初看起来，这种说法似乎与直觉不符。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，因此减少了本应用于高效开发的时间。只有有计划、有序的进行，增加人员对项目进度才有意义。

神话：如果决定将软件外包给第三方公司，就可以放手不管，完全交给第三方公司开发。

事实：如果开发团队不了解如何在内部管理和控制软件项目，那无一例外地将在外包项目中遇到困难。

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？"

客户神话。软件产品的客户也许是隔壁的某个人，楼下的一个技术团队，市场／销售部门或者签订软件合同的某个外部公司。多数情况下，客户之所以相信所谓的软件神话，是因为项目经理和从业人员没有及时纠正他们的错误信息。软件神话导致客户错误的期望，最终导致对开发者的不满。

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

事实：虽然通常很难得到综合全面且稳定不变的需求描述，但是对项目目标模糊不清的描述将为项目实施带来灾难。要得到清晰的需求描述（经常是逐步变得清晰的），只能通过客户和开发人员之间保持持续有效的沟通。

神话：虽然软件需求不断变更，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），则费用的影响较小[⊖]；但是，随着时间的推移，变更的代价也迅速增加—因为资源已经分配，设计框架已经建立，而变更可能会引起剧变，需要添加额外的资源或者修改主要设计架构。

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：曾经有人说过，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——技术评审，可以从项目启动就开始实行。软件评审（参见第15章）作为"质量过滤器"，已经证明可以比软件测试更为有效地发现多种类型的软件缺陷。

神话：对于一个成功的软件项目，可执行程序是唯一可交付的工作成果。

事实：软件配置包括很多内容，可执行程序只是其中之一。各样工作产品（如模型、文档、计划）是成功实施软件工程的基础，更重要的是，为软件技术支持提供了指导。

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，而是为了保证软件产品的开发质量。好的质量可以减少返工，从而加快交付时间。

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依然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。

1.7　这一切是如何开始的

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；改变遗留系统以适应新的业务环境；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。以下的一段简短谈话就是一个典型的例子。

⊖ 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3章讨论敏捷方法。

SAFEHOME

如何开始一个软件项目

[场景] CPI公司的会议室里。CPI是一个虚构的生产家用和商用消费产品的公司。

[人物] Mal Golden，产品开发部高级经理；Lisa Perez,营销经理；Lee Warren，工程经理；Joe Camallen，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传感器上，比如数码相机里，总之任何东西里。采用802.11g无线网络协议，可以通过无线连接获得它的输出。我们认为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和我已经做了一些市场调查，我们都认为该系列产品具有很大的市场潜力。

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”。产品采用新型无线接口，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、监视、应用和设备控制。例如，你可以在回家的路上关闭家里的空调，或者诸如此类的应用。

Lee（插话）：Joe，工程部已经作了相关的技术可行性研究。它可行且制造成本不高。大多数硬件可以在市场购买产品，不过软件方面是个问题，但并非做不到。

Joe：有意思!我想知道底线。

Mal：在美国，70%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技术。我们将在这个方向上保持两年的领先地位。收入嘛，在第二年大约可达到3千万到4千万。

Joe（微笑）：我很感兴趣，让我们继续讨论一下。

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。

1.8 小结

软件以是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，软件已经从解决特定问题和信息分析的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对的难题。

软件——程序、数据和描述信息——覆盖了科技和应用的很多领域。遗留软件仍旧给维护人员带来了特殊的挑战。

基于Web的系统和应用已经从简单的信息内容集合演化为能够展示复杂功能和多媒体信息

⊖ 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。

的复杂系统。尽管Web应用具有独特的特性和需求，他们仍然属于软件范畴。

软件工程包含过程、方法和工具，这些工具使得快速构建高质量的复杂的计算机系统成为可能。软件过程包括五个框架活动：沟通、策划、建模、构建和部署，这些活动适用于所有软件项目。软件工程实践遵照一组核心原则，是一个解决问题的活动。

尽管我们关于构建软件所需的软件知识和技能增长了，但仍有大量的软件神话将管理者和从业人员诱入歧途。随着对软件工程理解的深化，你就会逐渐明白，为什么无论何时遇到这些神话，都要不遗余力地揭露。

习题与思考题

1.1　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。

1.2　举例说明软件对社会的影响（包括正面影响和负面影响）。

1.3　针对1.1节提出的5个问题，请给出你的答案，并与同学讨论。

1.4　在交付最终用户之前，或者首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退化，请提出一些有效的解决措施。

1.5　思考1.1.2节中提到的7个软件分类。请问能否采用一个软件工程方法，应用于所有的软件分类？并就你的答案加以解释。

1.6　图1-3中，将软件工程三个层次放在了“质量关注点”这层之上。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。

1.7　软件工程对构建Web应用是否适用？如果适用，如何改进，以适应Web应用的独特特点？

1.8　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。设想一个真实场景，由于软件错误而引起“世界末日”般重大危害（危害社会经济或是人类生命财产安全）。

1.9　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。

1.10　普适性活动存在于整个软件过程中，你认为它们均匀分布于软件过程中，还是会集中在某个或者某些框架活动中？

1.11　在1.6节所列举的神话中，增加两种软件神话，同时指出与其相对应的真实情况。

推荐读物与阅读信息[⊖]

在数千本关于软件工程的书中，大多数讨论的是程序设计语言和软件应用，很少有涉及软件本身的。Pressman 和Herron（《Software Shock》，Dorset House，1991）最早讨论了软件和专业开发方法的问题（针对门外汉）。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（《Why does Software Cost So Much？》，Dorset House，1995）就软件和开发过程发表了一系列惊人且见解独到的论文。

Minasi在其著作中（《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，在本世纪的第一个十年里，是否能够访问到信息资源（如Web）的差别将越来越小。在Greenfield的著作（《Everyware:The Dawning Age of Ubiquitous Computing》，New Riders

⊖　在每章小结之后，“进一步阅读资料”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信息以作为本书的补充，并给出了每一本书在Amanzon.com的链接。

Publishing, 2006）和Loke的著作（《Context-Aware Pervasive Systems:Architectures for a New Breed of Applications》, Auerbach, 2006）中介绍了“普适性”（open-world）软件的概念，并指出在无线网络环境中软件必须能够进行适应性调整，以满足实时涌现的需求。

软件工程及软件过程的当前发展状况可以参阅期刊，如《IEEE Software》、《 IEEE Computer》、《CrossTalk》和《IEEE Transactions on Software Engineering》。行业期刊例如《Application Development Trends》和《Cutter IT Journal》通常包含一些关于软件工程的文章。每年，由IEEE 和ACM资助的研讨会论文集《Proceeding of the International Conference on Software Engineering》，都是对当年学术成果的总结，并且在《ACM Transactions on Software Engineering and Methodology》、《ACM Software Engineering Notes》和《Annals of Software Engineering》等期刊上有进一步的深入讨论。当然在互联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过程和软件工程的书籍，有些是关于整个过程的概要介绍，有些则深入讨论过程中的一些重要专题。下面是一些畅销书：

Abran, A.,和J. Moore,《SWEBOK: Guide to the Software Engineering Body of Knowledge》, IEEE, 2002.

Andersson,E.,等《Software Engineering for Internet Applications》, The MIT Press,2006.

Christensen, M.和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.

Jacobson, I.,《Object-Oriented Software Engineering:A Use Case Driven Approach, 2nd ed.》, Addison-Wesley, 2008.

Jalote, P.,《An Integrated Approach to Software Engineering》, Springer, 2006.

Pfleeger, S.,《Software Engineering: Theory and Practice, 3rd ed.》, Prentice-Hall, 2005.

Schach, S.,《Object-Oriented and Classical Software Engineering, 7th ed.》, McGraw-Hill,2006.

Sommerville, I.,《Software Engineering, 8th ed.》, Addison-Wesley, 2006.

Tsui, F., and O.karam, Essentials of Software Engineering, Jones&Bartlett Publishers, 2006.

在过去的几十年里，IEEE、ISO以及附属其下的标准化组织发布了大量软件工程标准。Moore（《The Road Map to Software Engineering: A Standards-Based Guide》, Wiley-IEEE Computer Society Press, 2006）对相关标准进行了调查并指出了这些标准如何应用到实际工程中。

因特网上有很多有关软件工程和软件过程相关问题的信息资源。许多最新的软件过程相关资源可以参阅本书网站：http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm。

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？
- 什么是惯用过程模型？有哪些优缺点？
- 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？

在解决了上述问题之后，就可以对软件工程实践的应用背景有更清楚的认识。

第2章

Software Engineering：A Practitioner's Approach (Seventh Edition)

过 程 模 型

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”。

人员及责任：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、控制以及工作产品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需要采用两种截然不同的软件过程。

工作产品：从软件工程师的角度来看，工作产品就体现为在执行过程所定义的任务和活动的过程中，所产生的程序、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，软件过程有效性最好的指标还是所构建产品的质量、及时性和长期生存能力。

关键概念

基于构件的开发
并发模型
演化过程模型
形式化方法模型
通用过程模型
增量过程模型
个体软件过程
惯用过程模型
过程模式
任务集
团队软件过程
统一过程

Howard Baetjer, Jr.[Bae98]曾著书从经济学家的角度分析软件和软件工程，该书引人入胜，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的过程，在对话中，获取需要转化为软件的知识，并在软件中实现这些知识。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）之间的互动。软件开发是一个迭代的过程，在其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。

构建计算机软件确实是一个迭代的社会学习的过程，其输出——即Baetjer所称的“软件资本”——是知识的载体，这些知识在过程执行中进行收集、提炼和组织。

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活动、动作和任务的框架。过程与软件工程同义吗？答案是“是，也不是”。软件过程定义了软件工程化中采用的方法，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 通用过程模型

在第1章中，过程定义为在工作产品构建过程中，所需完成的工作活动、动作和任务的集合。这些活动、动作、任务中的每一个都隶属于某一框架或者模型，框架或模型定义了它们同过程之间或者相互之间的关系。

KEY POINT
在软件过程中，技术工作的层次包括活动，活动由动作构成，动作由任务组成。

软件过程示意图如图2-1所示。由图可以看出，每个框架活动由一系列软件工程动作构成；每个软件工程动作由任务集合来定义，这个任务集合明确了将要完成的工作任务、将要产生的工作产品、所需要的质量保证点，以及用于表明过程状态的里程碑。

正如在第1章中讨论的，软件工程的通用过程框架定义了五种框架活动——**沟通、策划、建模、构建**以及**部署**。此外，一系列普适性活动——项目跟踪控制、风险管理、质量保证、配置管理、技术评审以及其他活动——贯穿软件过程始终。

"我们认为软件开发人员遗漏了一个事实：多数机构并不清楚他们在做什么，他们以为清楚了，但实际上没清楚"。——Tom DeMarco

你也许注意到了，软件过程的一个很重要的方面还没有讨论，即过程流（process flow）。过程流描述了在执行顺序和执行时间上，如何组织框架中的活动、动作和任务，如图2-2所示。

线性过程流（linear process flow）从沟通到部署顺序执行五个框架活动（参见图2-2a）。迭代过程流（iterative process flow）在执行下一个活动前重复执行之前的一个或多个活动（参见图2-2b）。演化过程流（evolutionary process flow）采用循环的方式执行各个活动，每次循环都能产生更为完善的软件版本（参见图2-2c）。并行过程流（parallel process flow，参见图2-2d）将一个或是多个活动与其他活动并行执行（如，软件一个方面的建模可以同软件另一个方面的构建活动并行执行）。

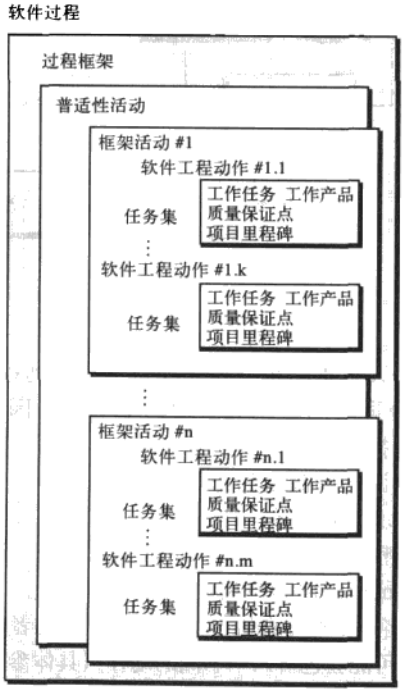


图2-1 软件过程框架

2.1.1 定义框架活动

尽管在第1章中，我们描述了5种框架活动，并给出了每种活动的基本定义，但是一个软件团队要在软件过程中具体执行这些活动中的任何一个，还需要更多信息。因此，我们面临一个关键问题：针对给定的问题、开发人员和利益相关者，哪些动作适合于框架活动？

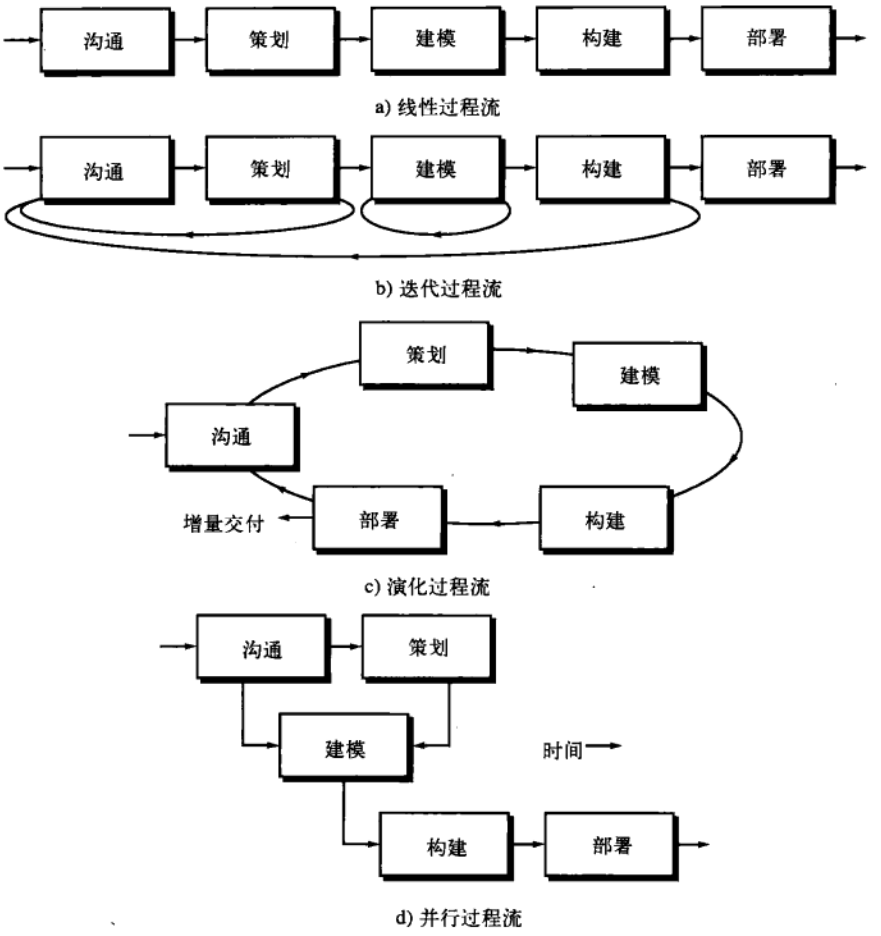


图2-2 过程流

框架活动如何随着项目性质的变化而变化？

对于由个人负责的小型软件项目（可能远程），其需求简单明确，沟通也许仅仅是与合适的利益相关者的一个电话。因此，主要的动作是电话交流，这个动作所包括的主要工作任务集有：

1. 通过电话与利益相关者取得联系。
2. 讨论需求并做记录。
3. 将笔记整理成一份简单的书面需求。
4. 通过E-mail，请利益相关者审阅并批准。

如果项目有多个利益相关者，则要复杂得多，每个利益相关者都有着不同需求（有时这些需求甚至是相互冲突的），沟通活动可能会包含六个不同的动作（具体参见第5章）：启动、需求获取、需求系统、谈判、规格说明和确认。每个软件工程动作都可能有很多工作任务和一些不同的工作成果。

KEY POINT
不同的项目需要不同的任务集。软件团队根据问题和项目的特点，选择任务集。

2.1.2 明确任务集

让我们再来看图2-1，每一个软件工程动作[或称行动（action），如需求获取，这是与沟通活动相关的动作]都由若干个任务集（task set）构成，而每一个任务集都由软件工程工作任务、相关工作产品、质量保证点和项目里程碑等部分组成，需要选择最满足项目需要并适合开发团队特点的任务集。这就意味着软件工程动作可以根据软件项目的特定需要和开发队伍的特点作适当的调整。

INFO

任务集

任务集定义了为达到一个软件工程动作的目标所需要完成的工作。例如，需求获取elicitation（通常称为“需求收集requirement gathering”）就是发生在沟通活动中一个重要的软件工程动作。需求获取的目的是理解利益相关者对将构建的软件的需求。

对于一个小型、相对简单的项目而言，获取需求的任务集可能包括：

1. 制定项目的利益相关者列表。
2. 邀请所有的利益相关者参加一个非正式会议。
3. 征询每一个人对于软件特征和功能的需求。
4. 讨论需求，并确定最终的需求列表。
5. 划定需求优先级。
6. 标出不确定领域。

对于大型、复杂的软件工程项目而言，可能需要如下不同的任务集：

1. 制定项目的利益相关者列表。
2. 和利益相关者的每一个成员分别单独讨论，获取所有的要求。
3. 基于任务集2中的调查，建立初步的功能和特征列表。
4. 安排一系列促进需求获取的会议。
5. 组织会议。
6. 在每次会议上建立非正式的用户场景。
7. 根据利益相关者的反馈，进一步细化用户场景。
8. 建立一个修正的需求列表。
9. 使用质量功能部署技术，划分需求优先级。
10. 将需求打包以便于软件可以增量交付。
11. 标注系统的约束和限制。
12. 讨论系统验证方法。

上面两种任务集都可以完成需求获取，但是无论从深度还是形式化的程度上来说，二者都有很大区别。软件团队应采取适当的任务集，以达到每个动作的目的，并且保持软件质量和开发的灵活性。

2.1.3 过程模式

什么是过程模式？

每个软件团队在软件过程里都会遇到很多问题。针对这些问题，如果软件团队能够得到已有的经过验证的解决方案，将有助于他们快速地分析和解决问题。过程模式（process pattern）[⊖]描述了软件工程工作中遇到的过程相关的

⊖ 关于模式的详细讨论参见第12章。

"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，而模式确是相同的。"
——Christopher Alexander

问题、明确了问题环境并给出了针对该问题的一种或几种可证明的解决方案。通俗地讲，过程模式提供了一个模板[Amb98]——一种在软件过程的背景下，统一描述问题解决方案的方法。通过模式组合，软件团队可以解决问题并定义最符合项目需求的开发过程。

我们可以在不同抽象层次上定义模式[⊖]。在某些情况下，模式可以描述一个与完整过程模型（例如原型开发）相关的问题（及其解决方案），在其他的情况下，模式可以描述一个与框架活动（譬如**策划**）或者框架活动中的一项具体任务（譬如项目估算）相关的问题（及其解决方案）。

模式模板提供了描述模式的一般性方法。

Ambler[Amb98]提出了下面的过程模式的描述模板：

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含义（例如，**技术评审**）。

驱动力。模式使用环境及主要问题，以明确主要难点并可能影响解决方案。

类型。定义模式类型。Ambler[Amb98]提出了三种类型：

1. 步骤模式（stage pattern），定义了与过程的框架活动相关的问题。由于框架活动包括很多动作和工作任务，步骤模式包括与步骤（框架活动）有关的许多任务模式（见以下描述）。例如，**建立沟通**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。

2. 任务模式（task pattern），定义了与软件工程动作或是工作任务相关、关系软件工程实践成败的问题（例如，**需求获取**是一个任务模式）。

3. 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。例如，**螺旋模型**和**原型开发**[⊖]就是两种阶段模式。

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）已经成功完成一些客户**沟通**模式中特定的任务模式；（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具体问题。

解决办法。描述如何成功实现模式。这部分主要讨论随着模式的启动，过程的初始状态（模式应用之前就已经存在）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变换的。

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）必须完成哪些开发组织或是开发团队相关的活动？（2）过程的结束状态是什么？（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直接相关的其他过程模式。例如步骤模式**沟通**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**场景模式的创建**等。

已知应用实例。说明该模式可应用的具体实例。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，并规定在部署活动中必须进行。

⊖ 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。

⊖ 第2.3.3节将讨论这些阶段模式。

WebRef

可在下面的网站查看过程模式的全面的资源：www.ambysoft.com/processPattensPage.html。

过程模式提供了一种有效的机制，用以解决任何与软件过程相关的问题。模式使得软件工程组织能够从高层抽象开始（阶段模式），建立层次化的过程描述。高层抽象描述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为更详细的任务模式。过程模式一旦建立起来，就可以复用来定义各种过程变体——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。

INFO

一个过程模式的例子

当利益相关者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。需求不清。

目的。该模式描述了一种构建模型（或是原型系统）的方法，使得利益相关者可以反复评估，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）确定利益相关者；（2）已经建立起利益相关者和软件开发队伍之间的沟通方式；（3）利益相关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，且该问题需要通过软件解决。利益相关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2.3.3节。

结束条件。开发了软件原型，识别了基本的需求（例如，交互模式、计算特征、处理功能等），并获得了利益相关者的认可。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，演化成为软件产品；或是（2）原型系统被抛弃，采用其他过程模式建立产品软件。

相关的模式。以下模式与该模式相关：**客户沟通、迭代设计、迭代开发、客户评价、需求抽取。**

已知应用和实例。当需求不确定时，推荐原型开发方法。

2.2 过程评估与改进

KEY POINT

以改进为目标，评估力求理解软件过程的当前状态。

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，或是软件具备了长期质量保证的技术特点（参见第14、16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，以确保满足了成功软件工程所必需的基本过程标准要求[⊖]。

? 软件过程评估有哪些形式化的技术？

在过去的几十年中，提出了很多种不同的软件过程评估和改进方法：

用于过程改进的CMMI标准评估方法（Standard CMMI Assessment Method for Process Improvement，SCAMPI）——提供了五步的过程评估模型：启动（initiating）、诊断（diagnosing）、建立（establishing）、执行（acting）和学习（learning）。SCAMPI方法采用SEI的CMMI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作为评估的依据[Dun01]，提供了一种诊断方法，

⊖ SEI CMMI[CMM07]丰富详实地介绍了软件过程的基本特征以及过程成功的标准。

"软件组织还存在很大的能力缺陷，难以将其从整个项目中所获得的经验转化成资产。"——NASA

用以分析软件开发机构相对成熟度。

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[ISO08]。

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，都可以采用这个标准。因此，该标准可直接应用于软件组织和公司[Ant06]。

有关软件评估和过程改进方法的详细讨论参见第30章。

2.3 惯用过程模型

最早提出惯用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，软件工程工作及其产品仍然停留在"混乱的边缘"。

"正确的过程产生正确的结果。"——Takashi Osada

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和他的同事指出[Nog00]:

混乱的边缘可定义为"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。

传统过程模型定义了规定的过程元素集合及可预测的过程工作流。

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、自我组织的过程和更高的回报[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。变化通常发生在某些结构中，这些结构使得变更可以被有效组织，但还不是死板得使变更无法发生。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。

上述这段话的哲学思想对于软件工程有着重要的意义。如果传统过程模型[⊖]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模型，是否会使软件工作无法达到协调和一致？

这些问题无法简单回答，但是软件工程师有很大的选择余地。在接下来的章节中，我们将探讨以秩序和一致性作为主要问题的传统过程方法。我们称之为"传统"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过程流（也称为工作流）——也就是说，过程元素相互之间关联的方式。

所有的软件过程模型都支持第1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，并且定义了不同的过程流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）。

2.3.1 瀑布模型

有时候，可以清楚地了解问题的需求，当从**沟通**到**部署**采用线性工作流方式的时候。这种情况通常发生在需要对一个已经存在的系统进行明确定义的适应性调整或是增强的时候（比如政府修改了规则，财务软件必须进行相应修改）；也可能发生在很少数新的开发工作上，但是

⊖ 惯用过程模型通常称为"传统"过程模型。

需求必须是准确定义和相对稳定的。

瀑布模型（waterfall model），又被称为经典生命周期（classic life cycle），它提出了一个系统的、顺序的软件开发方法[⊖]，从用户需求规格说明开始，通过计划、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（参见图2-3）。

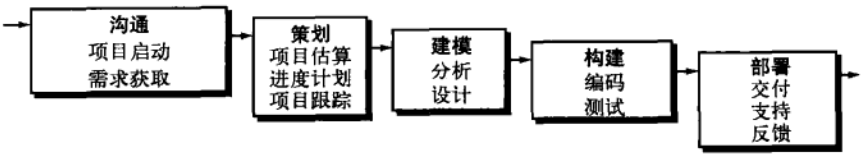


图2-3 瀑布模型

KEY POINT
V模型阐明了验证确认动作如何与早期工程动作相互关联。

瀑布模型的一个变体，称为V模型（V-model）。如图2-4，V模型[Buc99]描述了质量保证动作同沟通、建模相关动作以及早期构建相关的动作之间的关系。随着软件团队工作沿着V模型左侧步骤向下推进，基本问题需求逐步细化，形成问题及解决方案的技术描述。一旦编码结束，团队沿着V模型右侧的步骤向上推进工作，其本质上是执行了一系列测试（质量保证动作），这些测试验证了团队沿着V模型左侧步骤向下推进过程中所生成的每个模型[⊜]。实际上，经典生命周期模型和V模型没有本质区别，V模型提供了一种将验证确认动作应用于早期软件工程工作中的方法。

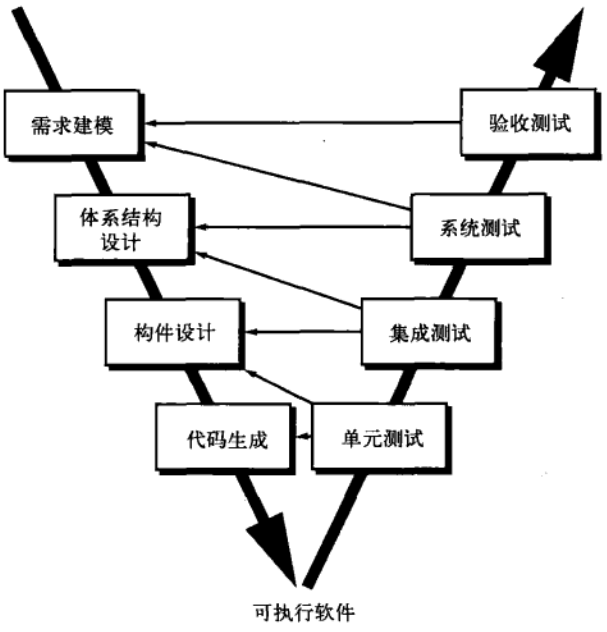


图2-4 V模型

⊖ 尽管对Winston Royce [Roy70]提出的最早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。

⊜ 质量保证动作的详细讨论参见本书第三部分。

为什么瀑布模型有时候会失效?

瀑布模型是软件工程最早的范例。尽管如此，在过去的30多年中，对这一过程模型的批评使它最热情的支持者都开始质疑其有效性 [Han95]。在运用瀑布模型的过程中，人们遇到的问题包括：

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，变更可能造成混乱。

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。

"大多数情况下，软件工作遵从骑自行车第一定律：不论你去哪，你都会顶风骑上坡路。"——作者不详

在分析一个实际项目时，Bradac[Bra94]发现，经典生命周期模型的线性特性在某些项目中会导致"阻塞状态"，由于任务之间的依赖性，开发团队的一些成员要等待另一些成员工作完成。事实上，花在等待上的时间可能超过花在生产性工作上的时间。在线性过程的开始和结束，这种阻塞状态更容易发生。

目前，软件工作快速进展，经常面临永不停止的变化流，特性、功能和信息内容都会变化，瀑布模型往往并不合适这类工作。尽管如此，当需求确定、工作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。

2.3.2 增量过程模型

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。

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进行细化和扩展功能。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。

增量模型综合了在2.1节中讨论的线性过程流和并行过程流的特征。如图2-5所示，随着时间的推移，增量模型在每个阶段运用线性序列。每个线性序列以一种演化过程流（见2.3.3节）生产增量类似的方式生产出一个软件的可交付增量[McD93]。

ADVICE 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。

例如，采用增量模型开发的文字处理软件，在第一个增量中提供基本的文件管理、编辑和文档生成功能；在第二个增量中提供复杂的编辑和文档生成功能；在第三个增量中提供拼写和语法检查功能；在第四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原型模型。

运用增量模型的时候，第一个增量往往是核心产品（core product）。也就是，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。

增量模型侧重于每个增量都提交一个可以运行的产品。早期的增量可以看做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了一个平台[⊖]。

⊖ 需要注意的是，增量原理也运用在第3章介绍的所有敏捷过程模型中。

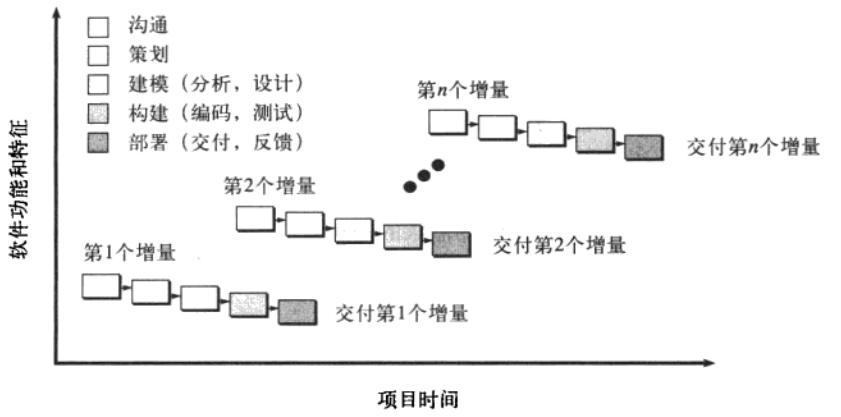


图2-5 增量模型

如果在项目既定的商业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，一个系统需要利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。

2.3.3 演化过程模型

KEY POINT
演化过程模型中，每个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。在开发过程中，商业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，软件开发人员需要一种专门应对不断演变的软件产品的过程模型。

"你可以采用任何方式生产一种产品，你唯一的选择是如何将它卖给客户。"
——Frederick P. Brooks

演化模型是迭代的过程模型，使得软件开发人员能够逐步开发出更完整的软件版本。在接下来的段落，将介绍两种常用的演化过程模型。

原型开发。很多时候，客户提出了软件的一些基本功能，但是没有详细定义功能和特性需求。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并不确定。在这些情况和类似情况下，采用原型开发范型（prototyping paradigm）是最好的解决办法。

ADVICE
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，原型开发模型可帮助软件开发人员和利益相关者更好地理解究竟需要做什么。

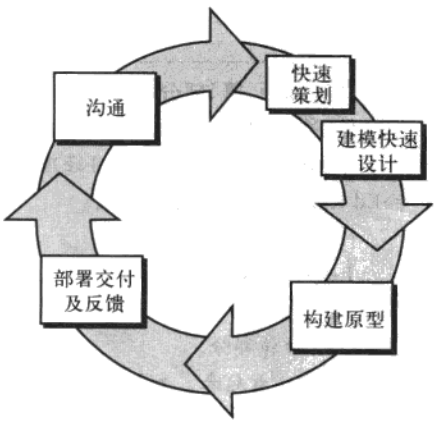


图2-6 原型开发模型

原型开发模型（图2-6）开始于沟通。软件开发

人员和利益相关者进行会晤，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。快速设计产生了一个原型。对原型进行部署，然后由利益相关者进行评价。根据利益相关者的反馈信息，进一步细化软件的需求。在原型系统不断调整以满足各种利益相关者需求的过程中，采用迭代技术，同时也使开发者逐步清楚用户的需求。

理想状况下，原型系统提供了定义软件需求的一种机制。当需要构建可执行的原型系统时，软件开发人员可以利用已有的程序片段或应用工具（如报告生成器和窗口管理器），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[Bro95]给出了答案：

在大多数项目中，构建的第一个系统很少是好用的，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。采用更巧妙的方法是构建一个重新设计的版本，解决上述问题。

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。尽管许多原型系统是临时系统，会被废弃，而其他一些原型系统将会慢慢演化为实际系统。

利益相关者和软件工程师确实都喜欢原型开发模型。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：

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到损害。

1. 利益相关者看到了软件的工作版本，却未察觉到整个软件是随意搭成的，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，利益相关者会不愿意，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. 作为一名软件工程师，软件开发人员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，软件开发人员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的情况。

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，所有利益相关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开发的。

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。

[成员] 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产品的产品线。毫无疑问，我们做了很多工作定义这个东西，我想请各位谈谈你们打算如何做这个产品的软件部分。

Doug：看起来，我们过去在软件开发方面相当混乱。

Ed：Doug，我不明白，我们总是能成功开发出产品来。

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

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

Doug：（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。线性模型太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开发模型。

螺旋模型。最早由Boehm[Boe88]提出，螺旋模型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[Boe01a]这样描述螺旋模型：

螺旋模型是一种风险驱动型的过程模型生成器，对于软件集中的系统，它可以指导多个利益相关者的协同工作。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，确保利益相关者都支持可行的和令人满意的系统解决方案。

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。

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是原型。在后来的迭代中，会产生一系列逐渐完整的系统版本。

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[⊖]。如图2-7，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28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。

WebRef
可以在如下网址获得关于螺旋模型的有用信息：www.sei.cmu.edu/publications/documents/00.reports/00sr008.html。

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，项目经理还会调整完成软件开发需要迭代的次数。

⊖ 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的螺旋模型的更多更新的讨论可参见[Boe98]。

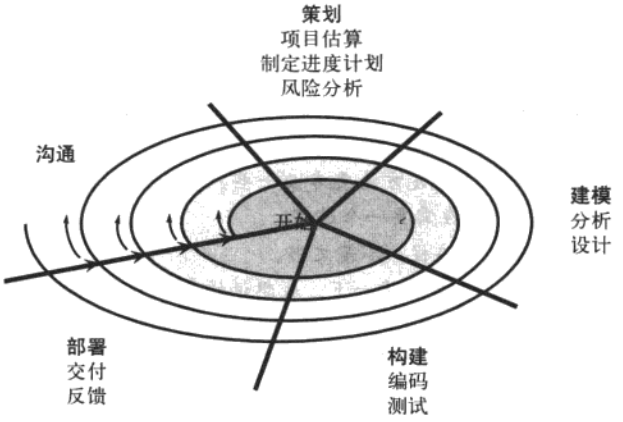


图2-7 典型的螺旋模型

ADVICE 如果你的项目要求固定预算开发（通常不是一个好主意），螺旋模型会带来问题：每一圈完成的时候，都将重新计划和修改项目开销。

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[⊖]，直到概念开发的结束。如果这个概念将被开发成为实际的产品，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入口点启动（如产品提高）。

“我今天只走这么远，只有明天才能为我指明方向。”——Dave Matthews乐队

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。螺旋模型把原型作为降低风险的机制，更重要的是，开发人员可以在产品演进的任何阶段使用原型方法。它保留了经典生命周期模型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。如果存在较大的风险没有被发现和管理，肯定会发生问题。

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；Vinod 和 Jamie，软件工程团队成员。

[会话]（Drog 介绍了一些可选的演化模型）

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

⊖ 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存在潜在的局部迭代。

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在每个增量中添加功能。

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

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

Lee（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。

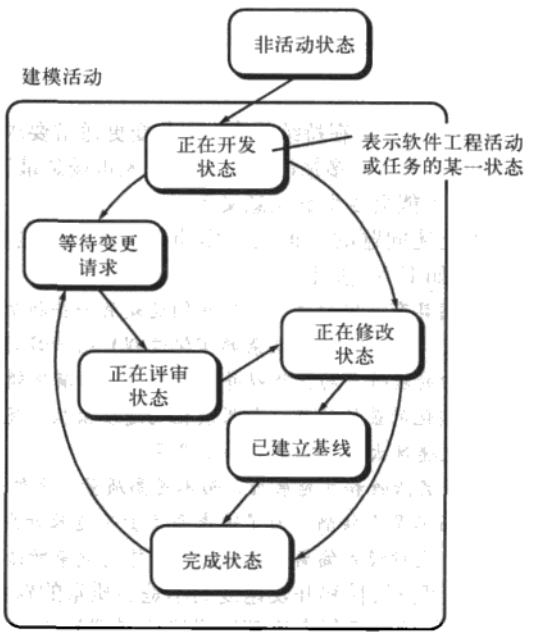


图2-8 协同过程模型的一个元素

2.3.4 协同模型

协同模型更适合不同的工程团队共同开发的系统工程项目。

协同开发模型（concurrent development model），有时候叫做协同工程，它允许软件团队表述本章所描述的任何模型中的迭代和并发元素。例如，螺旋模型定义的建模活动由以下一种或几种软件工程动作完成：原型开发、分析以及设计[⊖]。

对于协同模型方法中建模活动的某一软件工程活动，图2-8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中[⊜]。其他

⊖ 需要注意的是，分析和设计都是很复杂的任务，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。

⊜ 状态是外部可见的某种行为模式。

确保你所在组织的每一个过程都有客户，如果没有客户过程就会变成空中楼阁，失去目标。——V. Daniel Hunt

活动、动作或任务（如**沟通**或**构建**）可以用类似的方式表示。所有的软件工程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**非活动**状态，现在转换到**正在开发**状态。如果客户要求必须完成需求变更，建模活动从**正在开发**状态转换到**等待变更**状态。

协同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转换。例如，设计的早期阶段（建模活动期间发生的主要软件工程动作），发现了需求模型中的不一致性，于是产生了分析模型修正事件，该事件将触发需求分析动作从**完成**状态到**等待变更**状态。

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的序列，而是定义了一个过程网络。网络上每个活动、行为和任务与其他活动、行为和任务同时存在。过程网络中某一点产生的事件可以触发状态的转换。

2.3.5 演化模型的最终评述

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要求。如果市场时间错过了，软件项目自身可能会变得毫无意义[⊖]。

演化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和他的同事对此概括如下[Nog00]：

> 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，由于构建产品需要的周期数目不确定，原型开发（和其他更加复杂的演化过程）给项目计划带来了困难。大多数的项目管理和估算技术是基于活动的线性布局，所以并不完全适用于演化软件过程。
>
> 其次，演化模型没有确定演化的最快速度。如果演化的速度太快，完全没有间歇时间，项目肯定会陷入混乱；反之，如果演化速度太慢，则会影响生产率……
>
> 再次，软件过程应该侧重于灵活性和可延展性，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。这种模式偏离是处在混乱边缘的竞争所造成的。

确实，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[⊜]。可是，用演化模型也可以做到强调灵活性、可延展性和开发速度。软件团队及其经理所面临的挑战就是在这些严格的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。

2.4 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而专一，只适用于某些特定的软件工程方法[⊝]。

⊖ 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成功。事实上，许多非常成功的软件是第二个或是第三个进入市场的，它们的成功在于吸取了前人的教训。

⊜ 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14、16章将要讲的各种技术指标。

⊝ 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。

2.4.1 基于构件的开发

WebRef
基于构件开发的相关信息可以参见www.cbd-hq.com。

商品化成品（Commercial Off-The-Shelf，COTS）软件构件由厂家作为产品供应，通过良好定义的接口提供特定的功能，这些构件能够集成到正在构建的软件中。基于构件开发模型（component-based development model）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应用系统。

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通用的软件模块，有些设计成面向对象的类或软件包[⊖]。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（采用演化方法）：

1. 对于该问题领域研究和评估可用的基于构件的产品。
2. 考虑构件集成的问题。
3. 设计软件架构以容纳这些构件。
4. 将构件集成到架构中。
5. 进行充分的测试以保证功能正常。

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。如果构件复用已经成为你所在的软件工程团队文化的一部分，可以减少项目开发费用，同时还会缩短开发周期。基于构件的软件开发将在第10章进行详细讨论。

2.4.2 形式化方法模型

形式化方法模型（formal methods model）的主要活动是生成计算机软件形式化的数学规格说明。形式化方法使软件开发人员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（cleanroom software engineering）[Mil87, Dye92]，这一软件工程方法目前已应用于一些软件开发机构。

形式化方法（第21章）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模型时难以解决。使用形式化方法诸如歧义性问题、不完整问题、不一致问题都能够更容易地被发现和改正——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，使软件开发人员能够发现和改正一些往往被忽略的问题。

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，比如有人用其开发高度关注安全性的软件（如飞行器和医疗设施），或者开发出错将导致重大经济损失的软件。

2.4.3 面向方面的软件开发

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——已经体现在整个

⊖ 附录2讨论了面向对象技术，该技术的使用贯穿了本书第2部分。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。

WebRef
关于AOP的资源和信息可以参见aosd.net。

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，商业规则的应用），另外有一些强调系统性（例如，任务同步或内存管理）。

KEY POINT
AOSD定义“方面”，表示用户跨越多个系统功能、特性和信息的关注点。

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）。方面性需求（Aspectual Requirement）定义那些对整个软件体系结构产生影响的横切关注点。面向方面的软件开发（Aspect-Oriented Software Development，AOSD）通常称为面向方面编程（Aspect-Oriented Programming，AOP），是相对较新的一种软件工程模型，为定义、说明、设计和构建方面（aspect）提供过程和方法——是对横切关注点局部表示的一种机制，超越了子程序和继承的方法 [Elr01]。

Grundy[Gru02]在其称为面向方面的构件工程（Aspect-Oriented Component Engineering，AOCE）中进一步讨论了方面：

> AOCE对纵向分解的软件构件进行横向切片，称为“方面”（aspect），以表示构件功能及非功能的横切属性。通常，系统的方面包括用户接口、协同工作、发布、持续性、存储器管理、事务处理、安全、完整性等。构件也许提供或是需要某一方面一种或多种“方面的细节信息”，如视图机制、可扩展性和接口类型（用户接口方面）；事件生成、传输和接收（分布式方面）；数据存取/查询和索引（持久性方面）；认证、编码和访问权限（安全方面）；原子事务、协同控制和登录策略（事务方面）等。每个方面细节有大量的属性，这些属性都与方面细节的功能或非功能特性有关。

具有自身特点的面向方面的过程还不成熟。尽管如此，这种过程模型看似有了演化模型和协同过程模型的共同特点。演化模型适合定义和构建方面；而协同开发的并行特点很重要，因为方面是独立于局部的软件构件开发的，并且对这些构件的开发有直接影响。因此，在构建方面和构件的过程活动之间建立起异步的通信非常重要。

关于面向方面的软件开发最好查阅相关专著中的详细讨论。感兴趣的读者可以参看[Saf08]、[Cla05]、[Jac04]和[Gra03]。

SOFTWARE TOOLS

过程管理

目标：辅助定义、执行和管理传统过程模型。

机制：过程管理工具帮助软件组织或团队定义完整的软件过程模型（框架活动、动作、任务、质量保证检查点、里程碑和工作产品）。而且，该工具为软件工程师的技术工作提供路线图，为经理们跟踪和控制软件过程提供模板。

代表性工具：[⊖]

GDPA，一个研究性的过程定义工具包，该工具包由德国的Bremen大学开发(www.informatik.uni-bremen.de/uniform/gdpa/home.htm)，它提供了大量的过程模型和管理功能。

SpeeDev，由Speedev公司开发的一个工具包（www.speedev.com），用于过程定义、需求管理、问题决议、项目策划和跟踪。

ProVision BPMx，Proforma公司开发（www.proformacorp.com），它提供了很多过程定义和工作流自动化方面的工具。

以下网站提供了很多与软件过程相关的各种很有价值的工具：www.processwave.net/Links/tool_links.htm。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

2.5 统一过程

Ivar Jacobson、Grady Booch和James Rumbaugh[Jac99]在他们关于统一过程（Unified Process）的影响深远的著作中，讨论了关于有必要建立一种“用例驱动，以架构为核心，迭代并且增量”的软件过程的问题，并阐述如下：

当前，软件朝着更大、更复杂的系统发展。部分原因在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，还是受Internet应用膨胀的影响，促使各类信息交换……我们从软件版本的提升中学会了如何改进产品，使我们对更复杂软件的胃口越来越大。同时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，统一过程尝试着从传统的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3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[⊖]）并保持该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]。它建立了迭代的、增量的过程流，提供了演进的特性，这对现代软件开发非常重要。

2.5.1 简史

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面向对象分析和设计方法中最好的特点，并吸收其他面向对象模型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（unified modeling language），这种语言包含了大量用于面向对象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实际标准。

UML作为需求模型和设计模型的表示方式，其应用贯穿本书第二部分。附录1简要介绍了UML的基本概念和建模规则，有关UML的全面介绍可以参考有关UML的材料，附录1中列出了相关书籍。

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，这是用UML进行面向对象软件工程的框架。目前，统一过程和UML广泛应用在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。

2.5.2 统一过程的阶段[⊖]

在本章前面，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2-9描述了统一过程（UP，Unified Process）的阶段，并将它们与第1章及本章前面部分讨论的通用活动进行了对照。

KEY POINT
UP阶段的目的与本书中定义的通用框架活动的目的类似。

UP的起始阶段（inception phase）包括客户沟通和策划活动。通过与利益相关者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（第5章）描述每一类用户所需要的主要特征和功能。此时的体系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。策划识别各种资源，评估主要风险，制定进度计划，并为其在软件增量开发的各个阶段中的应用建立基础。

⊖ 用例（use case）（第5章）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。

⊖ 统一过程有时也用Rational公司（后来被IBM收购）的命名，称为Rational统一过程（Rational Unified Process，RUP）。Rational公司是早期开发和细化该统一过程的主要投资方，并建立了支持该过程的完整环境（工具及技术）。

细化阶段（elaboration phase）包括沟通和通用过程模型的建模活动（参见图2.9）。细化阶段扩展了初始阶段定义的用例，并扩展了体系结构以包括软件的五种视图——用例模型、需求模型、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]，这是建立可执行系统的“第一步”（first cut）[⊖]。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、风险和交付日期的合理性。该阶段对项目计划进行修订。

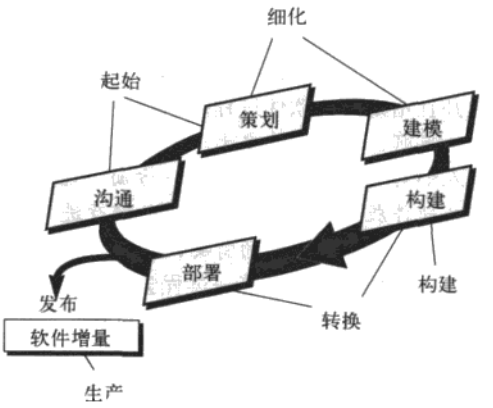


图2-9 统一过程

WebRef

敏捷开发中统一过程的有趣讨论参见：www.ambysoft.com/unifiedprocess/agileUP.html。

UP的构建阶段（construction phase）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，开发或是获取软件构件，使得最终用户能够操作用例。为达到上述目的，要对在细化阶段开始的需求模型和设计模型加以完善，以反映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源代码中实现。随着构件的实现，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。

UP的转换阶段（transition phase）包括通用构建活动的后期阶段以及通用部署（交付和反馈）活动的第一部分。软件被提交给最终用户进行Beta测试，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的生产阶段（production phase）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。

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（参见本章前面部分的描述）。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。软件开发团队应根据各自的需要适当调整过程（动作、任务、子任务及工作产品）。

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，只有对其进行充分改进，并使其真正满足实施软件工程项目的要求，该模型才能有效。理想化的情况下，软件开发人员会建立最适合他们的过程，并同时能够与开发队伍及

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

⊖ 有关软件测试（包括单元测试）的深入讨论参见第17～20章。

"成功者不过是养成了成功人士做事的习惯。"——Dexter Yager

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]和[Hum00]认为，有可能建立"个人软件过程"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[⊖]。

2.6.1 个人软件过程

每个开发人员都采用某种过程来开发计算机软件。这种过程也许是随机的，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过程模型定义了五个框架工作活动：

WebRef

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

PSP中用到了哪些框架活动？

策划。它将需求活动分离出来，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是模板记录。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。所有问题都要记录和跟踪。

高层设计评审。使用正式的验证方法（参见第21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。

KEY POINT

PSP强调对所犯的错误类型进行记录和分析，以便制定消除错误的策略。

PSP强调尽早发现错误，并且分析易犯的错误的种类，后者和前者同样重要。这是通过对软件开发人员提交的所有产品进行严格评估实现的。

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量将大幅度提高[Fer97]。然而PSP没有被工业界广泛地加以采纳。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其经理），这种支持通常很难得到。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。

PSP能否用作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

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）。TSP的目标是建立一个能够"自

⊖ 需要指出的是，敏捷方法的支持者（参见第3章）同样认为过程应贴近团队。他们只不过提出了达到同样目的的另外一种解决方法。

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

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。

为了组建有自我管理能力的团队，必须能够做到内部相互配合，并与外部建立良好沟通。

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；跟踪量化的项目数据（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。

TSP定义了以下的框架活动：**项目启动**、**高层设计**、**实现**、**集成**、**测试**以及**后验**。正如在PSP中与其对应的活动（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，同时量化地评测软件过程和产品。后验确定过程改进的步骤。

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），以及作为团队过程的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）。

TSP脚本定义了团队过程的组成部分，以及过程中的活动。

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

前面章节中讨论的一些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）来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。

过程技术工具帮助软件开发组织对2.1节中讨论的过程框架、任务集和普适性活动构建自动化的模型。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候选过程结构。

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、甚至控制过程模型中定义的所有软件工程任务、动作和任务。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。

⊖ 能力成熟度模型（CMM）是一种衡量软件过程效率的技术，将在第30章进行讨论。

⊖ 第3章讨论了敏捷模型中的关键因素“自管理”团队的重要性。

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过程描述的链接，以协助过程的参与人员了解并掌握所需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：[⊖]

Lgrafx 过程工具集，是对软件过程进行映射、度量和建模的一组工具(www.micrografx.com)。

Adeptia BPM Server，是用来管理、增强自动化、优化业务过程的工具(www.adeptia.com)。

SpeedDev Suite，是一个重点关注沟通和建模活动管理的工具集，由六个工具组成(www.speedev.com)。

2.8 产品与过程

如果过程很薄弱，最终产品必将受到影响。但是对过程的过于依赖也是很危险的。Margaret Davis[Dav95a] 在多年前写的一篇简短的文章里如下评述产品和过程的双重性：

大约每十年或五年，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和敏捷软件开发]。

钟摆的自然趋势是停留在两个极端的中点，与之类似，软件界的关注点也在不断地摆动，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，给软件工程实践人员造成混乱。而且这些摆动并没有解决问题，只是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相互矛盾的结果做出合理解释时，就会出现二象性理论。由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。如果仅仅将软件看做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许多人在不同的情况下反复使用。也就是说，我们是从我们自己或他人对我们产品的复用中得到满足。

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

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

感，也增加了接受“产品和过程二象性”这一观点的紧迫性。对于一个可复用的部件，如果仅仅从产品或是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。

正如从产品获得满足一样，人们在创造性的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；作家不仅欣赏已出版的书籍，更为每一个经过苦思冥想得到的比喻而欣喜。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。产品和过程的二象性已经成为保留推动软件工程不断进步的创造性人才的一个重要因素。

2.9 小结

一个软件工程通用过程模型包含了一系列的框架和普适性活动、动作以及工作任务。每一种不同的过程模型都可以用不同的过程流来描述，工作流描述了框架活动、动作和任务如何按顺序组织。过程模式用来解决软件过程中遇到的共性问题。

传统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。每一个模型都建议了一种不同的过程流，但所有模型都实现同样的一组通用框架活动：沟通、策划、建模、构建和部署。

类似瀑布模型和V模型的顺序过程模型是最经典的软件工程模型，顺序过程模型建议采用线性过程流，这在软件世界里通常与当代的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）。但线性过程模型确实适用于需求定义清楚且稳定的软件开发。

增量过程模型采用迭代的方式工作，能够快速地生成一个软件版本。演化过程模型认识到大多数软件工程项目的迭代、递增特性，其设计的目的是为了适应变更。演化模型，例如原型开发及螺旋模型，会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。

并发过程模型为软件团队提供了过程模型中的重叠和并发元素的描述方法。专用模型主要包括基于构件的模型，强调软件构件的重用和组装；形式化方法模型提倡采用数学方法进行软件开发与验证；面向方面的模型的目的是解决跨越整个软件体系架构的横切关注问题。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。

软件过程的个人模型和团队模型，都强调了成功软件过程的关键因素：测量、策划和自我管理。

习题与思考题

2.1 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]”。对于要构建的软件产品，在以下方面设计五个问题：(a) 设计者应该问用户的问题；(b) 用户应该问设计者的问题；(c) 用户对将要构建的软件自问的问题；(d) 设计者对于软件产品和建造该产品采取的软件过程自问的问题。

2.2 为沟通活动设计一系列动作，选定一个动作为其设计一个任务集。

2.3 在沟通过程中，遇到两位对软件如何做有着不同想法的利益相关者是很常见的问题。也就是说你得到了相互冲突的需求。设计一种过程模式（可以是步骤模式），利用2.1.3节中针对此类问题的模板，

给出一种行之有效的解决方法。

2.4 进一步研究PSP，并简要介绍针对软件工程师个人的度量类型以及这些度量方法如何改善个人的工作效率。

2.5 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。

2.6 阅读[Nog00]，然后写一篇2～3页的论文，讨论混乱对软件工程的影响。

2.7 详细描述三个适于采用瀑布模型的软件项目。

2.8 详细描述三个适于采用原型模型的软件项目。

2.9 如果将原型变成一个可发布的系统或者产品，应该如何调整过程?

2.10 详细描述三个适于采用增量模型的软件项目。

2.11 当沿着螺旋过程流发展的时候，你对正在开发或者维护的软件的看法是什么?

2.12 可以合用几种过程模型吗?如果可以，举例说明。

2.13 协同过程模型定义了一套“状态”，用你自己的话描述一下这些状态表示什么，并指出它们在协同过程模型中的作用。

2.14 开发质量“足够好”的软件，其优点和缺点是什么?也就是说，当我们追求开发速度胜过产品质量的时候，会产生什么后果?

2.15 详细描述三个适于采用基于构件模型的软件项目。

2.16 我们可以证明一个软件构件甚至整个程序的正确性，可是为什么并不是每个人都这样做?

2.17 统一过程和UML是同一概念吗?解释你的答案。

推荐读物与阅读信息

大多数软件工程课本都会详细介绍传统过程模型。Sommerville（《Software Engineering》，8th ed., Addison-Wesley, 2006），Pfleeger、Atlee（《Software Engineering》，3rd ed., Prentice-Hall, 2005）和Schach（《Object-Oriented and Classical Software Engineering》，7th ed., McGraw-Hill, 2006）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。Glass（《Facts and Fallacies of Software Engineering》，Prentice-Hall, 2002）提出了一种保证软件工程过程不加修饰的真实性观点。Brooks（《The Mythical Man-Month》，2nd ed., Addison-Wesley, 1995）在他的书中虽然没有直接讲过程，但是他一生的学识涉及了和过程相关的每一个方面。

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。Madachy（《Software Process Dynamics》，Wiley-IEEE, 2008）讨论了一种对软件过程中的相关技术和社会因素进行分析的建模技术。Sharpe 和McDermott（《Workflow Modeling: Tools for Process Improvement and Application Development》，Artech House, 2001）介绍了为软件和商业过程建模的工具。

Lim（《Managing Software Reuse》，Prentice Hall, 2004）从管理者的角度讨论了软件复用技术。Ezran、Morisio和Tully（《Practical Software Reuse》，Springer, 2002）以及Jacobson、Griss和Jonsson（《Software Reuse》，Addison-Wesley, 1997）介绍了很多基于构件开发技术的有用信息。Heineman和Council（《Component-Based Software Engineering》，Addison-Wesley, 2001）描述了实现基于构件系统的过程需求。Kenett和Baker（《Software Process Quality：Management and Control》，Marcel Dekker, 1999）考虑了高质量的管理和过程设计是如何相互影响的。

Nygard（《Release It！：Design and Deploy Production-Ready Software》，Pragmatic Bookshelf, 2007）和Richardson、Gwaltney（《Ship it！A Practical Guide to Successful Software Projects》，Pragmatic

Bookshelf，2005）介绍了适用于开发活动的一些指导方针。

除Jacobson、Rumbaugh和Booch的有关统一过程的书籍[Jac99]，Arlow和Neustadt（《UML 2 and the Unified Process》，Addison-Wesley，2005）、Kroll和Kruchten（《The Rational Unified Process Made Easy》，Addison-Wesley，2003）、Farve（《UML and the Unified Process》，IRM Press，2003）等人的书籍提供了很好的补充信息。Gibbs（《Project Management with the IBM Rational Unified Process》，IBM Press，2006）讨论了统一过程中的项目管理问题。

互联网上有大量关于软件工程和软件过程的信息。和软件过程有关的互联网链接可以参考SEPA网站http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm。

敏捷开发

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的早期增量发布，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。开发的指导方针强调超越分析和设计（尽管并不排斥这类活动）的发布，以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利益相关者（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。敏捷团队鼓励所有参与人员之间的交流与合作。

重要性：孕育着基于计算机的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针对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”，它保留了基本的框架活动：客户沟通、策划、建模、构建和部署，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（有人认为这种方法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着共同的观点：唯一真正重要的工作产品是在合适的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，并且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
自适应软件开发
敏捷过程
敏捷统一过程
敏捷性
Crystal
DSDM
极限编程
FDD
工业极限编程
精益软件开发
结对编程
项目速度
重构
Scrum
故事
极限编程过程

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]（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，我们认识到：

个人和这些个人之间的交流胜过了开发过程和工具

可运行的软件胜过了宽泛的文档

客户合作胜过了合同谈判

对变更的良好响应胜过了按部就班地遵循计划

也就是说，虽然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，敏捷开发确实是这样一场运动。

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，但真正将它们凝聚到一场“运动”中还不到二十年。从本质上讲，敏捷方法[⊖]是为了克服传

⊖ 敏捷方法有时也被称为轻量级方法或精简方法。

统软件工程中认识和实践的弱点而形成的。敏捷开发可以带来多方面的好处，但它并不适用于所有的项目、所有的产品、所有的人和所有的情况。它也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。

在现代经济生活中，通常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。市场环境飞快变化，最终用户需求不断变更，新的竞争威胁毫无征兆地出现。在很多情况下，项目实施之前，我们无法充分定义需求。因此，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。

不确定性意味着变更，而变更意味着付出昂贵的成本，特别是在其失去控制或疏于管理的情况下。而敏捷方法最具强制性的特点之一就是它能够通过软件过程来降低由变更所引起的代价。

难道说认识到现实的挑战，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。和其他所有工程学科一样，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[Coc02]在他那本发人深省的敏捷软件开发著作中，论证了本书第2章介绍的惯用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，他们在工作方式上有很大差别，在技能水平、创造性、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共同弱点"，因而大多数惯用过程模型选择了纪律。他还指出："不能始终一致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。"

要想让过程模型可用，要么必须提供实际可行的机制来维持必要的纪律，要么必须"宽容"地对待软件工程师。显而易见，宽容实践更易于被接受和保持，但是（正如Cockburn所认同的）可能效率低下。正像人生中的大多数事情一样，必须权衡利弊。

3.1 什么是敏捷

就软件工程工作而言，什么是敏捷？Ivar Jacobson[Jac02a]给出了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的每一件事中，因为它是软件的心脏与灵魂。敏捷团队意识到软件是由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看来，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。

不要误解，以为敏捷会赋予你随意做出拙劣产品的权利。过程还是需要的，而且纪律是必不可少的。

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。它鼓励能够使沟通（组员之间、技术和商务人员之间、软件工程师和经理之间）更便利的团队结构和协作态度。它强调可运行软件的快速交付而不那么看重中间产品（这并不总是好事情）；它将客户作为开发团队的成员以消除一直普遍存在于多数软件项目中的"区分你我"的态度；它意识到在不确定的世界里计划是有局限性的，项目计划必须是可以灵活调整的。

敏捷可以应用于任何一个软件过程。但是，为了实现这一目标，非常重要的一点是：过程的设计应使项目团队适应于任务，并且使任务流水线化，在了解敏捷开发方法的

流动性的前提下进行计划的制定，消除所有最基本软件产品并精简软件开发过程，强调这样一个增量交付策略：根据具体的产品类型和运行环境，尽可能快地将切实可行的软件交付给用户。

3.2 敏捷及变更的成本费用

"敏捷是动态的、针对特定内容的、主动应对变更以及适应增长特点的。"
—— Steven Goldman等

软件开发的传统方法中（有几十年的开发经验作支持）变化的成本费用随着计划的进展成非线性增长（图3-1，实黑曲线）。这种方法在软件开发团队收集需求时（在项目的早期）相对容易适应变化。应用场景需要修改，功能表应该扩充，或者书面说明书需要编辑。这项工作的费用是最小的，所需的时间不会严重影响项目的结果。但是，如果我们在经过数月的开发时间之后将会怎么样？团队在进行确认测试的过程中（也许是在项目后期的某个活动中），一个重要的利益相关者要求变更一个主要的功能。这一变更需要对软件的体系结构设计进行修改，包括设计和构建三个新组件，修改另外五个组件，设计新的测试等。费用会迅速升级，所需的时间和费用完全是为了保证变化不会引起非预期的副作用，而这方面的开销则是可观的。

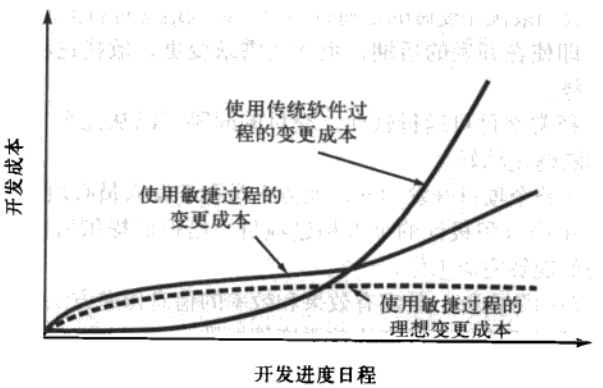


图3-1 变更成本是开发时间的一个函数

KEY POINT
敏捷过程能够降低变更的成本是因为软件产品以增量方式发布，而且在增量内部变更能得到较好的控制。

敏捷的拥护者（例如，[Bec00]，[Amb04]）认为，一个设计良好的敏捷过程"拉平"了变更成本曲线（图3-1，虚线），使软件开发团队在没有超常规的时间和费用影响的情况下，在软件项目后期能够适应各种变化。大家已经学习过，敏捷过程包括增量交付。当增量交付与其他敏捷实践耦合时，例如连续单元测试及结对编程（在本章后面讨论），引起变更的费用会衰减。虽然关于拉平曲线的程度的讨论仍然在进行，但是证据表明[Coc01a]，变更费用显著降低。

3.3 敏捷过程是什么

WebRef
敏捷过程的有关综合文集可在以下网站找到：www.aanpo.org/artides/index。

任何一个敏捷过程都可以由所强调的三个关键假设来识别[Fow02]，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而哪些需求会变更非常困难。同样，预测项目进行中客户优先级的变更也很困难。

2. 对很多软件来说，设计和构建是交错进行的。也就是，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。

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和技术条件）的可适应性。因此，敏捷过程必须具有可适应性。

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）变更的步伐。这种迭代方法使客户能够：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更。

3.3.1 敏捷原则

尽管敏捷过程能够包容和妥当地处理变更，但我们仍然要认真考察变更的理由。

让软件能够运行是很重要的，但不可忘记还必须使软件具有各种质量属性，其中包括可靠性、可用性以及可维护性。

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、持续地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. 经常交付可运行软件，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. 围绕有积极性的个人构建项目。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. 可运行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长期保持稳定的开发速度。

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（使不必做的工作最大化的艺术）是必要的。

11. 最好的架构、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。

并不是每一个敏捷模型都同等使用这12项原则，一些模型可以选择忽略（或至少淡化）一个或多个原则的重要性。然而，上述原则定义了一种敏捷精神，这种精神在本章中提出的每一个过程模型中得以维护。

3.3.2 敏捷开发的战略

与传统的软件工程过程相反，敏捷软件开发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）对传统软件工程阵营的立场时，他则给出完全相反的观点：“轻量级方法或者说敏捷方法学家是一群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。”

在敏捷与软件工程之间做选择不是必须的。自定义一个敏捷软件工程方式是最好的选择。

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？

对于这两个问题，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3.4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简单地修正而来的。归根结底，兼顾两派的优点则双方都能得到很多好处，而相互诽谤则两败俱伤。

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。

3.3.3 人的因素

“敏捷方法源于敏捷性，这种敏捷性取决于体现在团队内部那些不言而喻的知识，而不是写在计划里的知识。”
——Barry Boehm

敏捷软件开发的拥护者们不厌其烦地强调“人的因素”在成功敏捷开发中的重要性。正如Cockburn和Highsmith[Coc01a]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。”这一描述的关键点在于“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[⊖]。

有效的软件团队中，其成员必须具有哪些显著特点？

如果敏捷开发团队成员希望努力维护所使用的过程的特性，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。关于过程的技能和知识可以而且应该教给敏捷团队的每一位成员。

共同目标。虽然敏捷团队成员能完成不同的任务，为项目提供不同的技能，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的适应性变化，以使过程更适合于团队的需要。

“对一个团队刚刚够用的东西对别的团队来说要么不够用，要么太多。”
—— Alistair Cockburn

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；产生能够帮助所有利益相关者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，项目组与所有其他利益相关者之间必须精诚合作。

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。这意味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，被迫不断和变更作斗争。有时，项目组不得不接受今天正在解决的问题明天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）中学习到经验。

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24章），这样的团队展现出的相互信任和尊重使其形成“一个强有力的组织，确保整体的实力大于各部分实力之和”[DeM98]。

⊖ 一些成功的软件工程组织也认识到这一事实，因而忽略他们所选择的过程模型。

KEY POINT
自组织的团队对所承担的工作自行管理。团队做出承诺后制定完成工作的计划。

自组织。自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。本质上，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的了。"

3.4 极限编程

为了更详尽地说明敏捷过程，在此提供一个称为极限编程（eXtreme Programming，XP）的论述，它是敏捷软件开发使用最广泛的一个方法。虽然极限编程相关的思想和方法最早出现于20世纪80年代后期，但具有开创意义的著作由Kent Beck撰写[Bec04a]。近年来，XP的变种，称为工业XP(IXP)被提了出来[Ker05]。IXP细化了XP，目标是在庞大的组织内部使用敏捷过程。

3.4.1 极限编程的权值

Beck[Bec04a]为实施XP的全部工作定义了五个有重要意义的要素，即沟通、简明、反馈、鼓励和尊重。这五个要素中的每一个都是完成特定的XP活动、动作和任务的驱动力。

为了在软件工程师和其他利益相关者之间获得有效的沟通（例如，为软件建立所需的特性和功能），XP强调在用户和开发者之间进行紧密的、非正规的（口头的）合作，建立交流重要理念的有效隐喻[⊖]，连续的反馈，避免以大量的文档作为交流媒介。

ADVICE
只要有可能就应作到简明，不过必须认识到总是做重构会消耗大量的时间和资源。

为了做到简明，XP限制开发者只对即时需求做设计，而不考虑长远需求。这样做的目的是为了使代码设计简单化。如果设计需要改进，那么以后能够实现重构[⊖]。

反馈来自于以下三项：已实现的软件本身、客户和其他软件团队成员。通过设计和完成一个有效的测试策略（第17～20章），软件（通过测试结果）给敏捷团队提供反馈信息。XP使用单元测试作为主要的测试策略。每进行一级开发，开发团队就设计一个单元测试来测试每个操作是否按照规定功能完成。当一个增量提交给客户时，经由增量完成的用户故事或用例（第5章）就作为用于验收测试的一个基础。软件完成输出、功能和用例行为的程度构成了一种反馈。最后，当新需求作为迭代计划的一部分而提出时，团队就把费用和进度影响的反馈信息提供给客户。

"我们能开发多么小的软件，又能开发多么大的软件呢？XP可以给出回答。"
——匿名

Beck[Bec04a]指出与某个XP实践密切相关的就是需要鼓励。更贴切一些的词或许可以称之为纪律。例如，为将来的需求做设计有着显而易见的压力。多数软件团队慑服于此，辨解说"为明天做设计"可以在长跑中节省时间和精力。一个敏捷XP团队必须有为今天做设计的纪律（鼓励），认识到将来的需求可能会有显著的变化，从而需要对设计和已完成代码进行返工。

通过遵循以上这些权值，敏捷团队还应在团队成员之间，在其他利益相关者和团队成员之间，间接地，包括软件本身，灌输相互尊重的思想。当团队成功交付了软件增量时，他们对XP过程的尊重也会增加。

⊖ 在XP中，隐喻就是"每个人（客户、设计人员以及管理者）可以讲述的系统如何工作的故事"[Bec04a]。

⊖ 重构可以让软件工程师在不改变外部功能和行为的情况下改进设计的内部结构（或是源代码）。实际上，重构可以用来提高设计的效能、可读性或性能，或是改进实现设计的代码。

3.4.2 极限编程过程

WebRef

XP规则的很好综述可见于：www.extremeprogramming.org/rules.html。

XP使用面向对象方法作为推荐的开发范型（见本书附录2），它包含了策划、设计、编码和测试4个框架活动的规则和实践。图3-2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。下面将概括XP关键的活动。

策划。策划活动（也称为策划比赛）开始于倾听，这是一个需求获取活动，该活动要使XP团队技术成员理解软件的商业背景以及充分感受要求的输出和主要特征及主要功能。倾听产生一系列“故事”（也称为“用户故事”），描述即将建立的软件的需要的输出、特征以及功能。每个故事（类似于第5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综合业务价值标明故事的权值（即优先级）[⊖]。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，则将请客户把该故事进一步细分，重新赋予权值并计算成本。重要的是应注意到新故事可以在任何时刻书写。

什么是XP故事？

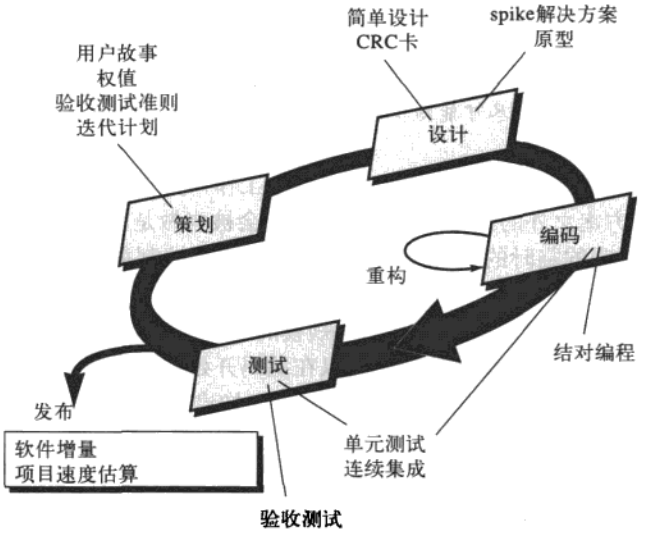


图3-2 极限编程过程

WebRef

很好的“策划比赛”可以参见c2.com/cgi/wiki?planningGame。

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）。一旦认可对下一个发布版本的基本承诺（就包括的故事、交付日期和其他项目事项），XP团队将以下述三种方式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）具有最高权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。

KEY POINT

项目速度是团队生产力的精妙度量。

项目的第一个发行版本（也称为一个软件增量）交付之后，XP团队计算项目的速度。简而言之，项目速度是第一个发行版本中实现的客户故事个数。项目速度将用于：（1）帮助估计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。

⊖ 一个故事的权值也可能取决于其他故事的存在。

XP并不强调设计的重要性。这一点不是所有人都会同意的。事实上，有时设计还是应该强调的。

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。

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[⊖]。

XP鼓励使用CRC卡（第7章）作为在面向对象环境中考虑软件的有效机制。CRC（类-责任-协作者）卡确定和组织与当前软件增量相关的面向对象的类[⊜]。XP团队使用类似于第8章描述的过程来管理设计工作。CRC卡也是作为XP过程一部分的唯一的设计工作产品。

WebRef

重构技术及其工具可在以下站点找到：www.refactoring.com。

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。

前一小节提到，XP鼓励既是构建技术又是设计优化方法的“重构”，Fowler[Fow00]描述“重构”如下：

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[并修改或简化内部设计]以尽可能减少引入错误的严格方法。实质上，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，如果有的话也只不过是生成除CRC卡和Spike解决方案之外的工作产品，所以设计会被当做是可以并且应当在构建过程中连续修改的临时的人工产品。重构的目的是控制那些由于提出“可以根本改进设计”的小设计修改而造成的（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。

重构能够改进设计（或源代码）的内部结构，但并未改变其外部功能或行为。

XP的中心观念是设计可以在编码开始前后同时进行，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推荐在故事开发和初步设计完成之后，团队不是直接开始编码，而是开发一系列用于检测本次（软件增量）[⊜]发布的包括所有故事的单元测试，一旦建立了单元测试[⊗]，开发者就更能够集中精力于必须实现的内容以通过单元测试。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，从而向开发者提供即时的反馈。

WebRef

有关XP的有用信息可以在以下网址找到：www.xprogramming.com。

XP编码活动中的关键概念（也是讨论最多的方面）之一是结对编程。XP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（在代码写出后及时得到复审），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，一名成员考虑设计特定部分的编码细节，而另一名成员确保编码遵循特定的标准或者（XP所要求的那些），确保故事相关的代码能够通过相对于故事而开发的单元测试。

何谓结对编程？

⊖ 虽然复杂的设计表示和术语可以加以简化，但每一个软件工程方法都应遵循这些设计准则。

⊜ 面向对象类将在本书附录2、第8章及整个第二部分讨论。

⊜ 这一方法类似于在学习开始之前得到考试题目，这样会使学生很容易地将注意力集中于要提的问题。

⊗ 单元测试将于第17章中详细讨论，集中于单个软件构件，检查构件接口、数据结构及功能，其目的在于发现构件内的错误

许多软件团队中聚集了个人主义者。如果要使结对编程工作有效，就必须设法改变团队的文化。

当结对的两人完成其工作，他们所开发的代码将与其他人的工作集成起来。有些情况下，这种集成作为集成团队的日常工作实施。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17章）。

测试。正如已经指出的，在编码开始之前建立单元测试是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（因此易于执行并可重复），这种方式支持代码修改之后即时的回归测试策略（第17章）（会经常发生，为XP提供重构支持）。

在XP中怎样应用单元测试?

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：“每几个小时修改一些小问题，比仅仅在最后截止期之前修正大问题要节省时间。”

XP验收测试是由用户故事驱动的。

XP验收测试，也称为客户测试，由客户规定技术条件，并且着眼于客户可见的、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。

3.4.3 工业极限编程

Joshua Kerievsky[Ker05]这样描述工业极限编程（IXP）：“IXP是XP的一种有机进化。它由XP的最低限要求、以客户为中心、测试驱动精神组成。IXP与原来的XP的主要差别在于其管理具有更大的包容性，它扩大了用户角色，升级了技术实践。”IXP合并了六个新实践，这些新实践的设计是为了有助于确保在一个庞大的组织内一些重要项目中XP工作成功地实施。

为了生成IXP，在XP上附加了哪些新的实践?

准备评估。在IXP项目开始执行前，组织机构应进行准备评估（readiness assessment）。评估应确定是否（1）存在支持IXP的适合的开发环境，（2）开发团队由合适的利益相关者组成，（3）组织机构具有清晰的质量大纲并且支持连续的改进，（4）组织文化会支持一个敏捷团队的新的权值，（5）组成较为广泛的项目社区。

能力是你能够做什么，激励决定了你做什么，而态度决定了你做得怎样。——Lou Holtz

项目社区。经典XP建议选择适合的人员组成敏捷团队可以确保成功。就是说团队成员必须经过良好的训练，具有良好的适应性和技能，以及适宜的性格为自组织团队做出贡献。当在一个大型组织内将XP应用于重要的项目，团队的概念就变成社区。一个社区可能拥有一个技术专家和处于项目成功核心地位的客户们，以及其他利益相关者（例如，法律人员、质量检验员、生产或销售人员），“他们通常位于IXP计划的周边，但在项目中他们扮演着重要的角色”[Ker05]。在IXP内，应明确定义社区成员和他们的角色，应建立社区成员之间交流和合作机制。

项目承租。IXP团队通过对项目本身进行评估来确定对于项目的合适的商业调整是否存在，以及是否可以进一步深化组织机构的全部目标和目的。承租也要检查项目环境来决定项目如何完成，如何扩展，或者如何替代现在的系统或过程。

测试驱动管理。一个IXP项目需要可测量的标准来评估项目的状态和迄今为止的进展情况。测试驱动管理建立一系列可测量的“目标”[Ker05]，然后定义一些机制来确定目标是否可以实现。

回顾。IXP团队在一个软件增量交付之后要实施特定的技术评审（第15章）。这种评审称为回顾，它在软件增量过程中以及/或者全部软件的发布过程中复查“问题、事件以及教训”[Ker05]。这样做的目的是为了改善IXP过程。

持续学习。由于学习是持续过程改进中至关重要的组成部分，因此，鼓励（可能激励）

XP团队的成员去学习新的方法和技术来提高软件产品质量。

除了以上讨论的六个新实践，IXP还修改了大量已有的XP实践。故事驱动开发（SDD）主张验收测试的故事写在所有代码生成之前。领域驱动设计（DDD）是XP中“系统隐喻”概念的改进。DDD[Eva03]建议渐进建立域模型，“域模型可以精确表示领域专家如何考虑课题”[Ker05]。结对（pairing）扩展了XP结对编程的概念，包括了管理者和其他利益相关者，目的是提高那些可能不直接参与技术开发的XP团队成员间的知识共享程度。迭代可用性（interative usability）并不鼓励前载接口（front-loaded interface）部件设计，其用意在于支持可用性设计，从而有利于软件增量交付以及用户与在研软件的交互。

IXP对其他XP实践进行了少量的修改，并重新确定某些角色和责任，使他们担负起大型组织重要项目的责任。关于IXP的进一步讨论，可以访问http://industrialxp.org。

3.4.4 关于XP的争论

所有的新模型和方法都会刺激有价值的讨论和某些情况下的热烈争论。这些现象对于极限编程来说都发生了。在一本有趣的书中研究了XP的功效，Stephens和Rosenberg[Ste03]认为许多XP实践是有价值的，但是其他的则属炒作过度，少部分是有疑问的。作者提出XP实践的依存性既具有优点也存在缺点。由于许多组织机构采用的只是XP实践的一部分，这就减弱了整个过程的效力。支持者称XP是持续发展的，许多受到批评的问题的产生可以看做是XP实践的成熟。在一直困扰的问题中对XP的批评意见有[⊖]：

是什么问题导致了XP的争论？

- *需求易变。*因为客户是XP团队的成员，对需求的改变不是正式地提出。结果是，项目的范围会发生变化，早期的工作或许得进行修改来适应当前的要求。拥护者认为这种情况的发生是由于不顾所应用的过程以及XP为控制范围渐变的机制所导致的。
- *矛盾的客户需求。*许多项目都有众多客户，每个客户都有自己的一套需求。在XP中，团队自身需要吸纳不同客户的要求，这项工作可能超出了自己的职权范围。
- *需求的非正规表示。*在XP中，用户故事和验收测试是对需求的唯一明确的表现形式。批评者指出需要更为正规的模型或规格说明来保证遗漏、不一致以及错误在系统建立前就被发现。拥护者则认为，需求的变化特性在其发展时使这些模型和规格说明变得过时了。
- *正规设计的缺乏。*XP削弱了对体系结构的设计的要求，在许多案例中，都建议所有类型的设计都不必那么正规。批评者主张当开发复杂的系统时，必须强调设计要保证软件的体系结构能够展示其质量和可维护性。XP的拥护者指出XP过程的增量特性限制了复杂性（简单性是其核心价值），因此降低了扩展设计的必要性。

应该注意到，每一个软件过程都有缺陷，而且许多软件机构已经成功地使用了XP。关键是认识到一个过程的弱点在哪里，然后使其适应于组织机构的特定要求。

考虑敏捷软件开发 SAFEHOME

[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和Vinod来到Doug的办公室）

Jamie：Doug，有时间吗？

⊖ 针对XP的评价意见可在以下网站获得：www.softwarereality.com/ExtremeProgramming.jsp。

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，有好处也有坏处。

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，还有利益相关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）：他们也是利益相关者，不是吗？

Jamie：哇，他们会每隔5分钟就提出一些变更。

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，实际上我坚信这样的做法没问题。

3.5 其他敏捷过程模型

“我们这个专业实施方法论就如同14岁的少年穿衣服一样，还不成熟。”
—— Stephen Hawrysh和Jim Ruprecht

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[⊖]。

就像前一节提到的，在所有敏捷过程模型中使用最广泛的就是XP。但是也提出许多其他敏捷过程模型，并且也在行业中使用。最普遍的有：

- 自适应软件开发（ASD）
- Scrum
- 动态系统开发方法（DSDM）

⊖ 这并不是坏事。在某个模型或方法被当做事实上的标准之前，都在尽力争取软件工程师群体的人心。最终胜利者将发展成为最佳实践，而失败者将销声匿迹或是被融入取胜的模型。

- Crystal
- 特征驱动开发（FDD）
- 精益软件开发（LSD）
- 敏捷建模（AM）
- 敏捷统一过程（AUP）

在以下的几节中，我们将简要介绍这几个敏捷过程模型。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3.3.1节中提到的那些原则。更为详尽的细节可参考在每个小节或评述中的参考资料，进入Wikipedia参看“敏捷软件开发”[⊖]。

3.5.1 自适应软件开发

WebRef

ASD方面的材料可见于：www.adaptivesd.com。

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3-3）的定义包含思考、协作和学习三个阶段。

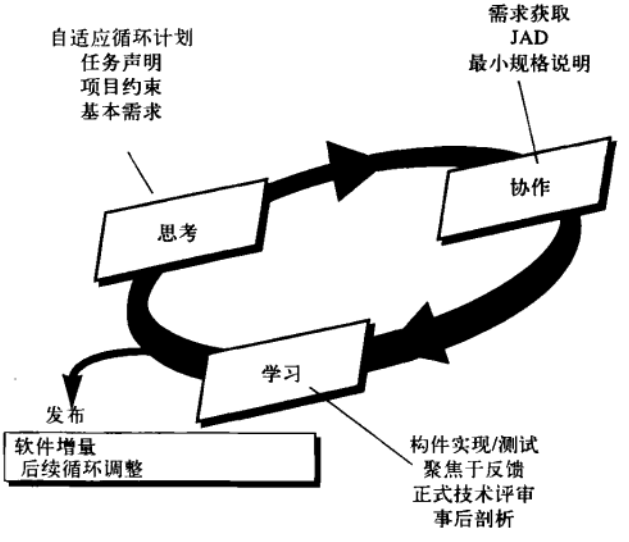


图3-3 自适应软件开发

只是在抛弃了“我们和他们”的观念后，与客户的有效协作关系才会建立起来。

思考阶段中，启动项目并完成自适应周期策划。自适应周期策划通过使用项目启动信息——客户任务描述、项目约束（如交付日期或用户描述）和基本需求——来确定项目所要求的一套软件增量发布周期。

无论是多么有远见和完整的周期计划，总是要发生变化的。基于第一个周期完成后所获得的信息，对计划进行评审和调整，使计划可以更好地适应一个ASD团队正在工作的现状。

有积极性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

⊖ 请参考http://en.wikipedia.org/wiki/Agile_software_development#Agile_methods。

有敏捷方法中不断重现的主旋律。但协作并不容易。它不仅强调沟通和团队合作，而且也不排斥个人的作用，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）用有效的方式沟通问题和事务。

KEY POINT
ASD强调把学习当做达到"自我组织"团队的关键元素。

当ASD团队成员开始开发构成自适应周期的构件时，其重点是朝着完成周期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：客户反馈意见（第5章），技术评审（第15章），事后剖析。

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，从而使团队更有可能取得成功。

3.5.2 Scrum

WebRef
关于Scrum的有用信息和资源可见于：www.controlchaos.com。

Scrum（得名于橄榄球比赛[⊖]）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[Sch01a]对其做了进一步的发展。

Scrum原则与敏捷宣言是一致的，应用Scrum原则指导过程中的开发活动，过程由"需求、分析、设计、演化和交付"等框架性活动组成。每一个框架活动中，发生于一个过程模式（在下文中讨论）中的工作任务称为一个冲刺（sprint）。冲刺中进行的工作（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）适应于当前的问题，由Scrum团队规定并常常作实时修改。Scrum过程的全局流程如图3-4所示。

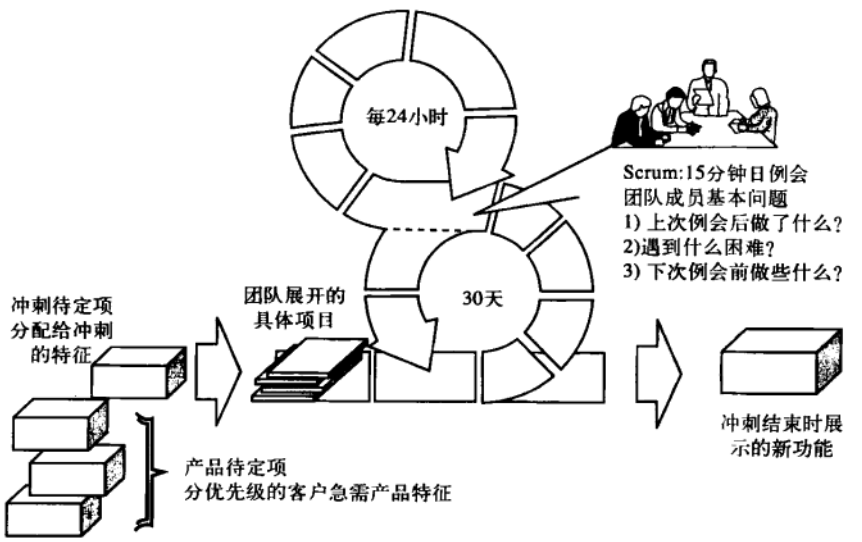


图3-4 Scrum过程流

Scrum强调使用一组"软件过程模式"[Noy02]，这些过程模式被证实在时间紧张的需求

⊖ 一组球员围着球共同努力（有时具有暴力性），带球扑地。

KEY POINT
Scrum是由含有以下内容的一组过程模式构成：强调项目优先次序、分离的工作单元、沟通以及频繁的客户反馈。

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）——一个能为用户提供商业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新项（这就是变更的引入）。产品经理根据需要评估待定项并修改优先级。

冲刺（sprint）——由一些工作单元组成，这些工作单元是达到待定项中定义的需求所必需的，并且必须能在预定的时间段（time-box[⊖]）内（一般情况下为30天）完成。冲刺过程中不允许有变更（例如，积压工作项）。因此，冲刺给开发团队成员的工作提供了短期但稳定的环境。

Scrum例会——Scrum团队每天召开的短会（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？
- 遇到了什么困难？
- 下次例会前计划做些什么？

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“知识社会化交流”[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，这可以试用并评价向用户演示所实现的功能并由客户对其评价。需提醒的很重要的一点是，演示不需要包含计划内的所有功能，但是规定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提到：“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。

3.5.3 动态系统开发方法

WebRef
有关DSDM的有用信息见于：www.dsdm.org。

动态系统开发方法（Dynamic System Development Method，DSDM）[Sta97]是一种敏捷软件开发方法，该方法提供一种框架，使其“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]。DSDM建议借用修改版Pareto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。

DSDM使用迭代软件过程，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）是一个由一些共同充当DSDM方法管理者的成员公司所组成的全球性组织。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个不同的迭代周期，前面还加了两个生命周期活动：

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

业务研究——建立能为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原型都倾向于逐渐发展成可交付的应用系统）。这一迭代的意图是在用户使用原型系统时诱导出反馈信

⊖ Time-box是一个项目管理术语（见本书第4部分），指的是分配给完成某一任务的时间段。

息以获取其他的需求。

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）置于操作环境中。应当注意：（1）增量不见得100%完成；（2）增量置于操作环境以后可能需要改变。在这两种情况下，DSDM开发转向功能模型迭代后继续进行。

KEY POINT
DSDM是一个过程框架，它可采用其他敏捷方法（如XP）的策略。

DSDM和XP(3.4节)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(DSDM生命周期)，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。

3.5.4 Crystal

KEY POINT
Crystal是一个具有相同“遗传密码”的过程系列，只是针对项目的特点采用了不同的方法。

Alistair Cockburn[Coc05]和Jim Highsmith[Hig02b]建立了Crystal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的、可工作的软件，其第二目标是为下一次行动做准备”[Coc02]。

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，以及独一无二的角色、过程模式、工作产品和实践。Crystal系列实际上是一组经过证明对不同类型项目都非常有效的敏捷过程。它的目的是使得敏捷团队可以根据其项目和环境选择最合适的Crystal系列成员。

3.5.5 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的实用过程模型而构思的。Stephen Palmer和John Felsing[Pal02]扩展并改进了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。

WebRef
有关FDD的许多文章和文献可见于：www.featuredriven-development.com/。

像其他敏捷方法一样，FDD采用这样一个基本原理：（1）重点强调在FDD团队成员间的合作；（2）使用基于特征分解随后集成软件增量的方法管理问题和计划复杂性，并且（3）使用口头的、图解的以及基于文本的方法交流技术细节。FDD重点强调通过鼓励增量开发策略、使用设计和代码检查、应用软件质量保证审查（第16章）、收集度量、使用模板等活动（用于分析，设计和构建）来确保软件质量。

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]。强调特征的定义是为了如下好处：

- 特征是小块可交付功能，用户可以更容易地对其进行描述、轻松地理解他们之间的相互关系，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的分组。
- 由于特征是FDD可以交付的软件增量，团队每两周便可开发出可供使用的特征。
- 由于特征很小，其设计和代码表示都可以很容易、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>**

⊖ Crystal（水晶）这一名称是取自地质学水晶的特征，每一种水晶都有独特的颜色、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目录项的描述）”。例如，一个电子商务应用项目的特征可能如下所示：

将产品加入购物卡

显示产品详细技术说明

为顾客存储购物信息

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，出售一件商品（Making a product sale）是一个特征集，它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3-5所示。

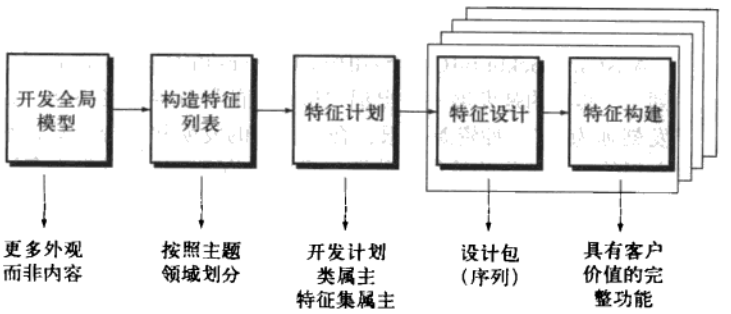


图3-5 特征驱动开发[Coa99]（经过授权）

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、管理者和其他的利益相关者而言，非常有必要理解项目状态（已经完成了什么，遇到了什么问题）。如果最后期限压力很大，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]。

3.5.6 精益软件开发

精益软件开发（Lean Software Development，LSD）在软件工程领域适应于精益制造的原则。精益原则鼓励LSD过程（[Pop03]，[Pop06a]）消除耗损、把质量体现于产品、创造知识、遵从承诺、快速交付、尊重成员以及整体优化。

每一个原则都适应于软件过程。例如，在一个敏捷软件计划中消除耗损可以解释为[Das05]：（1）不加入无关的特性和功能，（2）评估任何一个新的需求对费用和进度的影响，（3）去掉任何多余的过程步骤，（4）建立能改进团队成员获取信息渠道的机制，（5）保证测试能够发现尽可能多的错误，（6）缩短获取需求所需的时间，以及缩短获得影响软件或影响创建软件的步骤的结论的时间，（7）将信息传送给过程中所有利益相关者的方式流线化。

对LSD细节以及完成过程的实际的指导原则的讨论，可以参看[Pop06a]和[Pop06b]。

3.5.7 敏捷建模

很多情况下，软件工程师必须构建大型的、业务关键型的系统，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）对正在设计和构建的系统质量进行评估。

WebRef
有关敏捷建模更为全面的信息可见于：www.agilemodeling.com。

过去的30多年来，人们提出了许多用于分析和设计（架构和构件级）的软件工程建模方法和表示方法，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更发生时维护的难度。就算没有其他原因，仅仅考虑项目智能化的可管理性，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解决这些问题提供可选方案呢？

几天前我去药店打算买些感冒药，可是不容易啊。看到有太多的药可选，往前走走，是一种速效的，还有一种是长效的。究竟选哪个好呢？是图眼前还是图长远呢？——Jerry Seinfeld

在敏捷建模官方网站上，Scott Ambler[Amb02a]用以下方式描述敏捷建模（Agile Modeling，AM）：

> AM是一种用于对基于软件的系统实施有效建模和文档编制的基于实践的方法学。AM是可以有效并以轻量级方式用于软件开发项目中软件建模的标准、原则和实践。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更有效。

AM采纳了与敏捷宣言一致的全部标准。敏捷建模的指导思想认为，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提出一系列的“核心”和“补充”建模原则，但其中独具特色的是[Amb02a]：

ADVICE
对所有软件工程工作而言，“轻装”都是适合的指导思想。我们只需要那些有价值的模型，不要多，也不要少。

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，或有助于更好地理解软件的某些方面），一旦确定模型的目标，该用哪种类型的表达方式以及所需要的具体细节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，并且应使用能够为那些预期的读者提供有价值的模型。

轻装上阵。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更而进行维护，这些描述工作将使整个团队进度变慢。Ambler[Amb02a]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、团队与利益相关者增强沟通）。

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

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。

当前软件工程领域界大都已采用了统一建模语言（UML）[⊖]作为首选的方法来表示分析和设计模型。统一过程（第2章）已经为UML提供了一个框架。Scott Ambler[Amb06]已经开发出集成了其敏捷模型原理的UP的简单版本。

3.5.8 敏捷统一过程

敏捷统一过程（Agile Unified Process，AUP）采用了一个“全局串行”以及“局部迭代”[Amb06]的原理来构建基于计算机的系统。采用经典UP阶跃性活动——开始、加工、构建以

⊖ 在本书附录1中提供了UML的简明教程。

及变迁，UP提供一系列覆盖（例如，软件工程活动的一个线性序列），能够使团队为软件项目构想出一个全面的过程流。然而，在每一个活动里，一个团队迭代使用敏捷，并且将有意义的软件增量尽可能快地交付给最终用户。每个AUP迭代执行以下活动[Amb06]：

建模。UML建立了对商业和问题域的表述。然而，为了保持敏捷，这些模型应当“足够好”，[Amb06]使团队继续前进。

- 实现。将模型翻译成源代码。
- 测试。像XP，团队设计和执行一系列的测试来发现错误以保证源代码满足需求。
- 部署。就像第一章和第二章中讨论过的一般过程活动，这部分的部署重点仍然是对软件增量的交付以及获取最终用户的反馈信息。
- 配置及项目管理。在AUP中，配置管理（第22章）着眼于变更管理、风险管理以及对开发团队的任一常效产品[⊖]的控制。项目管理追踪和控制开发团队的活动情况和工作进展。
- 环境管理。环境管理协调过程基础设施，包括标准、工具以及适用于开发团队的支持技术。

虽然AUP与统一建模语言有历史上和技术上的关联，但是很重要的一点必须注意，UML模型可以与任一敏捷过程模型相结合（3.5节）。

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，强调便利地快速构建可执行软件。这些工具也可以用于惯用（传统）过程模型（见第2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：[⊜]

注：由于敏捷开发是一个热门话题，大多数软件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。

OnTime，由Axosoft开发（www.axosoft.com），提供对各种技术活动的敏捷过程管理支持。

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销售（www.borland.com），提供支持XP和其他敏捷过程中许多技术活动的工具包。

3.6 敏捷过程工具集

KEY POINT

这里所提到的敏捷过程“工具集”更多地是从人员的方面，而不是从技术方面支持敏捷过程。

敏捷哲学的拥护者指出，自动软件工具（例如，设计工具）应当被看做是对开发团队活动小小的补充，而不是团队成功的关键。然而，Alistair Cockburn[Coc04]建议，工具是有益处的，“敏捷团队强调使用工具可以达到快速理解的目的。有些工具是社会性的，甚至开始于租赁阶段。有些工具是技术性的，可以帮助使用者团队模拟物理现状。很多工具是物理性的，允许人们在工作场所操作这些工具。”

由于找到工具需要的人（租用者）、团队内部的合作、利益相关者间的相

⊖ 常效工作产品指的是被管理的团队在不同阶段开发出的模型、文档或测试用例，这些产品在软件增量交付后并不废弃。

⊜ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

互沟通以及间接管理，事实上都是敏捷过程模型的关键元素，因此Cockburn指出，用于解决这些问题的“工具”是敏捷的至关重要的成功因素。例如，一个租用的“工具”或许正是寻找一个期望的团队成员花极少的时间与一个现有团队成员结对编程所需要的。这种“适合”能够立即判定。

协作和沟通“工具”通常技术含量较低并且与可以提供信息以及协调敏捷开发人员的任何机制相结合（“这些机制可以是物理上的近距离性、白色书写板、海报、索引卡以及可粘贴留言条”）[Coc04]。积极的沟通是通过团队能动性获得的（例如，结对编程），而被动的沟通是通过“信息辐射体”实现的（例如，一台平面显示器可以显示一个增量不同组件的全部状态）。项目管理工具降低了Gantt（甘特）图的重要性，使用挣值图（earned value chart）来取代Gantt图或“对过去增量测试图……还有其他工具可用来优化敏捷团队工作的环境（例如，更有效的会议区），通过培养社交互动（例如，配置团队）、物理设备（例如，电子白色书写板）、过程增强（例如，结对编程或时间盒）”[Coc04]等提高团队文化。

所有这些都是工具吗？如果它们能够促进敏捷团队成员的工作以及提高最终产品的质量，它们就是工具。

3.7 小结

在现代经济中，市场条件变化十分迅速，客户和最终用户的要求在演变，新一轮竞争威胁会没有任何征兆地出现。从业者必须使软件工程工作保持敏捷——要限定过程应是灵活机动的、有适应能力的和精益的以适应现代商务的需求。

软件工程的敏捷理念强调4个关键问题：自我组织团队对所开展工作具有控制力的重要性；团队成员之间以及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。

其他敏捷过程模型也强调人员合作和团队自组织，只是定义自己的框架活动，选择不同的侧重点。例如，ASD使用迭代过程，该过程由自适应周期计划、相对严格的需求收集方法和一个迭代开发周期构成，其中迭代开发周期包括客户关注点组和正式技术评审作为实时反馈机制。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、需求变更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团队以适应其项目要求的方式构建过程。动态系统开发方法（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。Crystal是一系列敏捷过程模型，可用于具有特定特征的项目。

特征驱动开发（FDD）在某种程度上比其他敏捷方法更“形式化”，但是通过使项目开发团队专注于特征的开发来维持敏捷性——可在2周或更短时间内实现对客户有价值的功能。精益软件开发（LSD）是在软件工程界采用精益制造的原则。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、类型和规模必须根据所构建的软件来调节。敏捷统一过程（AUP）采用“全局串行”以及“局部迭代”的原则来构建软件。

习题与思考题

3.1 重新阅读一遍本章开头的“敏捷软件开发宣言”，你能否想出一种情况，此时4个权值中的一个或多

个将使软件开发团队陷入麻烦？

3.2 用自己的语言描述（用于软件项目的）敏捷性？

3.3 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。

3.4 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。

3.5 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。

3.6 选择3.3.1节提到的一条敏捷性原则，讨论本章所描述的各过程模型是否符合该原则。[注意：我只是给出了这些过程模型的一个概述，因此，确定一个或多个模型是否符合某个原则或许不可能，除非你做过额外的研究（对这个问题并不需要）。]

3.7 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？

3.8 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。

3.9 写出一个描述多数网页浏览器所具有的“我的最爱”或“书签”特征的XP用户故事？

3.10 什么是XP的“Spike解决方案”？

3.11 用自己的语言描述XP的重构和结对编程的概念。

3.12 增加阅读，然后描述什么是时间盒？它是如何帮助ASD团队在短时期内传递软件增量的？

3.13 80%的DSDM原则和ASD定义的时间盒可以得到相同的结论吗？

3.14 使用第2章描述的过程模式模板，为3.5.2节所描述的任一Scrum模式开发一种过程模式。

3.15 为什么称Crystal是一个敏捷方法系列？

3.16 使用3.5.5节所描述的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。

3.17 访问敏捷建模官方站点（www.agilemodeling.com），给出所有核心和补充AM原则的列表。

3.18 3.6节中给出的工具集为敏捷方法的“软件”方面提供了很多支持。由于交流非常重要，推荐一种实际工具集可以用来增强敏捷团队中客户间的交流。

推荐读物与阅读信息

敏捷软件开发的全部理论和基本原则在本章提供的很多参考书中都有更深层的论述。另外，在Shaw和Warden（《The Art of Agile Development》，O'Reilly Media,Inc.,2008），Hunt（《Agile Software Construction》，Springer，2005），以及Carmichael和Haywood（《Better Software Faster》，Prentice-Hall，2002）的书中给出了关于主题的有用的讨论。在Aguanno（《Managing Agile Project》，Mult-media Publications，2005），Highsmith（《Agile Project Management: Creating Innovation Products》，Addison-Wesley,2004),以及Larman（《Agile and Iterative Development: A Manager's Guide》，Addison-Wesley,2003)的书中,给出了关于管理的概述及项目管理问题的思考。在 Highsmith （《Agile Software Development Ecosystem,》 Addison-Wesley,2002) 提出了一份关于敏捷原则、过程和实践的调查。由Booch及其同事的书中（《Balancing Agile and Discipline》，Addison-Wesley,2004),给出了关于敏捷和规则间平衡的颇有价值的讨论。

Martin（《Clean Code：A Handbook of Agile Software Craftmanship》，Prentice-Hall, 2009）指出了在敏捷软件工程环境下开发“简洁程序代码”所需敏捷原则、模式和实践。Leffingwell（《Scaling Software Agility：Best Practices for Large Enterprises》，Addison-Wesley, 2007）讨论了在大型计划中按比例提高敏捷实践的策略。Lippert 和Rook（《Refactoring in Large Software Projects：Performing Complex Restructurings Successfully》，Wiley，2006）讨论了在大型、复杂系统中重构的使用。Stamelos

和Sfetsos（《Agile Software Development Quality Assurance》，IGI Global，2007）讨论了符合敏捷理论的SQA技术。

在过去十年间，很多书中描述了极限编程。Beck（《Extreme Programming Explained：Embrace Change》，2d ed. Addison-Wesley, 2004）的书是论述得比较全面的。另外，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）。

Highsmith[HIG00]深入强调了ASD。Schwaber（《The Enterprise and Scrum》，Microsoft Press。2007）讨论了对主营业务产生影响的计划中使用Scrum。Schwaber及Beedle（《Agile Software Development with SCRUM》，Prentice-Hall, 2001）发表了对Scrum方法的深入探讨。关于DSDM方法的有价值的论述可以在DSDM Consortium（《DSDM: The Method in Practice》，2nd ed., Person Education,2003）及Stapleton（《DSDM: The Method in Practice》，Addison-Wesley, 1997）的书中找到。Cockburn（《Crystal Clear》，Addison-Wesley, 2005）对过程中的Crystal系列进行了精彩的论述。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。Poppendieck和Poppendieck（《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。在Ambler和Jeffries（《Agile Modeling》，Wiley, 2002）深入探讨了AM。

Internet网上可以获得关于敏捷软件开发的更为广泛的信息资源。在SEPA网站http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm上可以找到最新的敏捷过程参考文献。

建　模

在本书这一部分，读者将学到构建高质量需求模型和设计模型的基原则、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概念和原则？
- 什么是需求工程？什么是能导致良好需求分析的基本概念？
- 如何创建需求模型？它包含哪些元素？
- 好的设计包含哪些元素？
- 体系结构设计如何为其他的设计活动建立一个架构，使用什么模型？
- 如何设计高质量软件部件？
- 在设计用户接口时使用什么概念、模型及方法？
- 什么是基于模式的设计？
- 使用什么特别策略和方法设计WebApp？

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

第4章

Software Engineering：A Practitioner's Approach (Seventh Edition)

指导实践的原则

要点浏览

概念：软件工程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。指导实践的原则成为软件工程实施的基础。

人员：软件工程实践由实践者（软件工程师）和软件项目经理共同完成多种软件工程任务。

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、哪里有岔路，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；确信你已经理解了如何运用这种方法以及使用适合此任务的自动工具，并且坚信需要一些技术来保证工作产品的质量。

关键概念
核心原则
指导原则：
　编码
　沟通
　部署
　设计
　建模
　计划
　需求
　测试

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（发光二极管）微波显示器在闪烁，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；在其他时间——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、为了谁以及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[⊖]，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每天使用的概念、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

⊖ 一些作者主张使用这些词语中的一个，而排除其他词语。事实上，软件工程包括所有这三个含义。

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。

软件工程实践的各个不同方面将在本书的剩余部分介绍。本章的重点是指导软件工程实践的原则和概念。

4.1　软件工程知识

十年前编辑出版的《IEEE Software》期刊中，Steve McConnel[McC99]发表了以下评论：

许多软件实践者认为软件工程知识无疑与特定技术知识：Java、Perl、html、C++、Linux、Windows NT等相同。这些特定技术知识是用来进行计算机程序设计的。如果某人派你去编写一个C++程序，你就得掌握关于C++的知识来完成你的编程工作。

经常听到人们说，软件开发知识的半衰期为3年：现在你需要知道的那些知识，在三年内会有一半将过时。在技术相关的知识领域内，这种说法可能是正确的。但还有另一种软件开发知识——一种我认为是“软件工程原则”——并没有3年半衰期的说法。这些软件工程原则可以为专业程序设计人员在其整个职业生涯内提供服务。

McConnel 继续论述了软件工程知识体（大约在2000年）已经演变为“稳定的核心”，他估计该知识体提供了大约“开发一个复杂系统所需的75%知识”。但是这个“稳定的核心”里面都有些什么呢？

如McConnel 所述，核心原则——指导软件工程师工作的基本概念——现在提供从软件工程模型、方法及工具中得来的，可以应用和评价的基本原理。

4.2　核心原则

“理论上理论与实践没有区别，而事实上，差别是存在的。”
——Jan van de Snepscheut

软件工程是以一系列核心原则作指导的，这些核心原则为应用具有重大意义的软件过程以及执行有效的软件工程方法提供了帮助。在过程级上，核心原则建立了哲学基础从而指导软件开发团队执行框架活动和普适性活动、引导过程流以及生产一系列软件工程产品。在实践级，核心原则建立了一系列价值和规则，为分析问题、设计解决方案、实现和测试解决方案以及最终在用户社区部署软件提供指导。

在第1章，我曾提出一系列跨越软件工程过程和实践的通用原则：（1）为最终用户提供价值，（2）保持简洁，（3）维护可见的东西（产品和计划），（4）认识（必须理解）到别人消费你生产的产品，（5）面向未来，（6）计划复用，以及（7）认真思考！虽然这些通用原则非常重要，但是以这样高度抽象的语言来表述，使它们有时很难转化为日常的软件工程实践。在以下的部分，我将更为具体地着眼于指导过程和实践的核心原则。

4.2.1　指导过程的原则

在本书的第一部分，我讨论了软件过程的重要性，描述了提供给软件工程的许多不同的过程模型。没有论及模型是线性的或是迭代的、传统的或敏捷的，这可以用适用于所有过程模型的普通过程框架来描述。以下一组核心原则能够适用于框架，并延伸至每一个软件过程。

每一个计划和每一个团队都是独立的。这就意味着，必须使过程尽可能地适应于需求。

原则1：敏捷。你所选择的过程模型是否是传统的或敏捷的，敏捷开发的基本原则会提供给你判断的方法。你所做的工作的每一方面都应着重于活动的经济性——保持你的技术方法尽可能简单，保持你的工作产品尽可能简洁，

"事实是你总是知道该做的正确事情。困难的是去做。"
—— General H.Norman Schwarzkopf

无论何时尽可能根据具体情况做出决定。

原则2：每一步都关注质量。每个过程活动、动作及任务的出口条件应关注所生产的工作产品质量。

原则3：做好适应的准备。过程不是信仰体验，其中没有信条。当需要的时候，就让你的方法适应于由问题、人员以及项目本身施加的限制。

原则4：建立一个有效的团队。软件工程过程和实践是重要的，但最根本的还是人。必须建立一个彼此信任和尊重的自组织团队。

原则5：建立沟通和协调机制。项目失败是由于遗漏重要信息，以及（或者）利益相关者未能尽力去创造一个成功的最终产品。这些属于管理的问题，必须设法解决。

原则6：管理变更。管理变更的方法可以是正式的或非正式的，但是必须建立一种机制，来管理变更要求的提出、变更的评估、变更的批准以及变更实施的方式。

原则7：评估风险。在进行软件开发时，会做错很多事。建立应变计划是非常重要的。

原则8：创造能给别人带来价值的工作产品。仅仅创建那些能为其他过程活动、动作或任务提供价值的工作产品。每一个工作产品都作为软件工程实践的一部分传递给别人。需求功能和特征表会传递给开发设计的人员，设计会传递给编码的人，等等。一定要确保工作产品所传达的是必要信息不会模棱两可或遗失。

本书的第4部分重点是在项目管理和过程管理问题，以及这些原则各个方面的细节。

4.2.2 指导实践的原则

软件工程实践有一个最重要的目标——按时交付包含了满足所有利益相关者要求的功能和特性的高质量、可运行软件。为了实现这一目标，必须采用一系列的核心原则来指导技术工作。这些原则的优点是不用考虑你所使用的分析方法和设计方法、你所使用的构建技术（例如，程序设计语言、自动工具），或者你所选用的验证和确认方法。以下列举的一组核心原则是软件工程实践的基础。

当问题被分解为多个独立的部分，问题会比较容易求解，每个部分都是明确的、独立求解和可验证的。

原则1：分治策略（分割和攻克）。更具技术性的表达方式是：分析和设计中应经常强调关切问题分解（SoC，Separation of Concerns）。一个大的问题分解为一组小的元素（或关切问题）之后就比较容易求解。从概念上讲，每个关切问题都传达了可以开发、在某些情况下可被确认的特定功能，这与其他关切问题是无关的。

原则2：理解抽象的使用。在这一核心原则中，抽象就是对一个系统中一些复杂元素的简单化，用以传达词组的含义。我使用抽象报表，是假设你可以理解什么是报表，报表表示的是目录的普通结构，可以将经典的功能展示其中。在软件工程实践中，可以使用许多不同层次的抽象，每个通告或暗示都意味着必需交流信息。在分析和设计中，软件团队通常从高度抽象的模型开始（例如，报表），然后逐渐将这些模型提炼成较低层次的抽象（例如，专栏或SUM功能）。

Joel Spolsky[Spo02]提出"在某种程度上，即使是正规的抽象也都是有漏洞的。"抽象的意图是减弱交流细节的需求。但有时，由于细节的"渗漏"，有问题的影响会不期而至。由于对细节不了解，所以对问题产生的原因并不容易分析。

原则3：力求一致性。是否创建一个需求模型、开发一个软件设计、开发源代码、或者创建测试用例，一致性原则建议熟悉的上下文使软件易于使用。例如，为WebApp设计一个用户界面，一致的菜单选择、一致的彩色设计的使用，以及可识别的图标的一致性使用都有助于使界面在人体工程学方面更为合理。

KEY POINT
未来的软件工程师使用范例（第12章）来获取知识和经验。

原则4：关注信息传送。软件所涉及的信息传送是多方面的——从数据库到最终用户、从现有的应用系统系统到WebApp、从最终用户到图形用户界面（GUI）、从操作系统到应用问题、从一个软件构件到另一个构件。每一种情况，信息都会流经界面，因而，就有可能出现错误、或者遗漏、或者歧义的情况。这一原则的含义是必须特别注意界面的分析、设计、构建以及测试。

原则5：构建能展示有效模块化的软件。对重要事务的分割（原则1）建立了软件的哲学。模块化提供了认知这一哲学的机制。任何一个复杂的系统都可以被分割许多模块（构件），但是好的软件工程实践不仅如此，它还要求模块必须是有效的。也就是说，每个模块都应该专门地集中表示系统的一个良好约束的方面——其功能具有内聚性和（或）局限于所表示的内容范围。另外，模块应当以相对简单的方式关联起来——每个模块应同其他模块、数据源以及环境方面是低耦合的。

原则6：寻找模式。Brad Appleton[App00]指出：软件界内模式的目标是建立一个典集，帮助软件开发者解决整个软件开发过程中反复出现的问题。模式还有助于创建一种共同语言，交流有关这些问题及其解决办法的见解和经验。对这些解决办法以及它们之间关系的正式地编纂成典可以使我们成功地捕获知识体系，这一知识体系中明确了对满足用户需求的良好体系结构的认识。

原则7：在可能的时候，用大量不同的观点描述问题及其解决方法。当对一个问题及其求解方法从大量不同的观点进行描述时，就很有可能获得更深刻的认识，发现错误和遗漏。例如，需求模型可以用面向数据的观点、面向功能的观点、或者行为的观点（第6章和第7章）来陈述，每个观点都提供了一个对问题及其需求的不同看法。

原则8：记住：有人将要对软件进行维护。从长期看，缺陷暴露出来时，软件需要修正；环境发生变化时，软件需要适应；利益相关者需要更多功能时，软件需要增强。如果可靠的软件工程实践能够贯穿于整个软件过程，就会便于这些维护活动的实施。

虽然这些原则不能包含构建高质量软件所需要的全部内容，但是它们为本书讨论的每个软件工程方法奠定了基础。

4.3 指导每个框架活动的原则


“理想的软件工程师是复合型的……他不是科学家，不是数学家，不是社会学家或者作家；但是他或许会使用到任一或全部这些学科规范来解决软件工程问题。”——N.W.Dougherty

以下几节中，我关注的是对作为软件过程一部分的每一个通用框架活动的成功产生重要影响的原则。在很多情况下，所讨论的每个框架活动的原则都是对4.2节中提出的原则的提炼。它们只是处于抽象的低层次的核心原则。

4.3.1 沟通原则

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来收集，客户有一些问题适合于使用计算机求解，软件人员就要对客户请求作出响应。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、与客户和其他利益相关者的沟通、与项目经理的沟通）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，许多原则同样适用于软件项目内部的沟通。

原则1：倾听。一定要仔细倾听讲话者的每一句话，而不是急于叙述你对这些话的看法。如果有什么事情不清楚时可以要求他澄清，但是要避免经常打断别人的讲述。当别人正在陈述

在沟通之前确定你理解了其他人的观点，知道他或她需要什么，然后倾听。

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：沟通活动需要有人推动。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。

"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4：最好当面沟通。但是，如果能把一些相关信息写出来，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。

原则5：记笔记并且记录所有决定。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。每次小型的协作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：把讨论集中在限定的范围内。在任何交流中，参与的人越多，话题转移到其他地方的可能性就越大。最简便的方法就是保持谈话模块化，只有当某个话题完全解决之后再开始别的话题（不过还应该注意到原则9）。

原则8：如果某些东西很难表述清楚，采用图形表示。语言沟通的效果很有限，当语言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：(a) 一旦认可某件事情，转换话题；(b) 如果不认可某件事情，转换话题；(c) 如果某项特性或者功能不清晰，当时无法澄清，转换话题。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），"转换话题"有时是达到敏捷交流的最好方式。

如果无法与客户就项目相关问题达成一致，将会发生什么？

原则10：协商不是一场竞赛或者一场游戏，协商双赢时才发挥了协商的最大价值。很多时候软件工程师和利益相关者必须商讨一些问题，如功能和特征、优先级和交付日期等。若要团队合作得很好，那么各方要有一个共同的目标，并且，谈判还需要各方的协调。

客户与最终用户之间的差别

软件工程师与许多利益相关者交流，但是客户与最终用户对将采用的技术影响最大。有的时候客户与最终用户是相同的人，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）为项目协调资金的人。在产品业务或者系统业务中，客户通常是销售部门；在IT环境下，客户或许是一个业务单位或部门。

最终用户是这样的人（或组织）：（1）为了达到某种商业目的而将真正使用所编写软件的人；（2）为达到其商业目的将定义软件可操作细节的人。

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？

Jamie：是这样，我给Lisa Perez 打了一个电话，她是销售部经理。

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：那个销售部门需要我们扮演顾问的角色，我们最好在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。

4.3.2 策划原则

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而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。

不管我们做多少努力，都不可能准确预测软件项目会如何进展。也不存在简单的方法来确定可能遇到的不可预见的技术问题，在项目后期还有什么重要信息没有掌握，以及会出现什么误解，或者会有什么商务问题发生变化。然而，软件团队还必须制定计划。

“在为战役做准备中，我经常发现，计划是无用的，但做计划是必不可少的。”——General Dwight D.Eisenhower

有很多不同的制定计划的哲学[⊖]。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，并且计划得越详细，团队损失的可能性就越小；也有一些人是“敏捷主义者”，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”是一种时间的浪费并且是在做无用功（因为事物有太多的变化），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的指导——

⊖ 关于软件项目制定计划和管理的详细讨论在本书第4部分给出。

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。

WebRef
一个关于计划和项目管理信息的优秀资料库可以在以下网址找到：www.4pm.com/repository.html。

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。

原则2：吸收利益相关者参与策划。利益相关者能够限定一些优先次序，确定项目的约束。为了适应这种情况，软件工程师必须经常商谈交付的顺序、时间表以及其他与项目相关的问题。

原则3：要认识到计划的制定应按照迭代方式进行。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要根据用户反馈的信息（在每个软件增量交付之后）重新制定计划。

"成功更多的是一种协调能力而不是天赋。"
——An Wang

原则4：基于已知的估计。估计的目的是基于项目组对将要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。

原则5：计划时考虑风险。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（包括进度计划）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。噪音总能侵入人们的交流之中。生活的现实常常会有疏忽与含糊。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制定计划的时候考虑。

KEY POINT
粒度表明项目计划中表述和控制的元素的详细程度。

原则7：调整计划粒度。粒度主要指项目计划细节的精细程度。一个"细粒度"的计划可以提供重要的工作任务细节，这些细节是在相对短的时间段内计划完成的（这样就常常会有跟踪和控制的问题）。一个"粗粒度"的计划提供了更宽泛的长时间工作任务。通常，粒度随项目的进行而从细到粗。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则不需要细化（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[⊖]的话，应该将其列入进度；如果在构建过程中用到了结对编程（第3章），那么在计划中要明确描述。

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一些变化需要怎样去适应，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：经常跟踪并根据需要调整计划。项目一次会落后进度一天的时间。因此，需要每天都追踪计划的进展，找出计划与实际执行不一致的问题所在，当任务进行出现延误时，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，项目组成员才能很好地认可所指定的计划。

4.3.3 建模原则

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

⊖ 技术评审在第15章讨论。

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。

需求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：需求模型和设计模型。需求模型（也称为分析模型）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、功能域和行为域。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、用户界面以及构件细节。

Scott Ambler和Ron Jeffries[Amb02b]在他们关于敏捷建模的书中定义了一系列建模原则[⊖]，提供给使用敏捷模型（第3章）的人，但也适用于执行建模活动和任务的软件工程师。

任一模型的目标都是传递信息。为了实现这一目标，使用一致的格式。假设你不能够解释此模型，那么它始终都只是个模型而已。

原则1：软件团队的主要目标是构建软件而不是创建模型。敏捷的意义是尽可能最快地将软件提供给用户。可以达到这个目标的模型是值得软件团队去构建的，但是，我们需要避免那些降低了开发过程的速度以及不能提供新的见解的模型。

原则2：轻装前进——不要创建任何你不需要的模型。变化发生时，创建的模型必须是最新的。更重要的是，每创建一个新模型所花费的时间，还不如花费在构建软件上（编码或测试）。因此，只创建那些可以使软件的构建更加简便和快速的模型。

原则3：尽量创建能描述问题和软件的最简单模型。不要建造过于复杂的软件[Amb02b]。保持模型简单，产生的软件必然也会简单。最终的结果是，软件易于集成、易于测试以及易于维护（对于变更）。另外，简单的模型易于开发团队成员理解和评判，从而使得反馈正在进行的形式可以对最终结果进行优化。

原则4：用能适应模型改变的方式构建模型。假设模型发生了变化，但做这种假设并不草率。例如，由于需求发生变化，就需要迅速改变需求模型。为什么？因为不管怎样它们都会发生变化。所带来的问题是如果没有相当完整的需求模型，你所创建的设计（设计模型）会常常丢失重要功能和特性。

原则5：明确描述创建每一个模型的目的。每次创建模型，都问一下自己为什么这么做。如果不能为模型的存在提供可靠正当的理由，就不要再在这个模型上花费时间。

原则6：调整所开发模型来适应待开发系统。或许需要使模型的表达方式或规则适用于应用问题；例如，一个视频游戏应用或许需要的建模技术与实时的、嵌入式控制汽车引擎的软件所需的建模技术完全不同。

原则7：尽量构建有用的模型而不是完美的模型。当构建需求模型和设计模型时，软件工程师要达到减少返工的目的。也就是说，努力使模型绝对完美和内部一致并不值得。我建议过模型应当草率或低质量吗？答案是“没有”。但是，对当前建模工作的管理要始终考虑到软件工程的下一步骤的实施。无休止地使模型“完美”并不能满足敏捷的要求。

原则8：对于模型的构造方法不要过于死板。如果模型能够成功地传递信息，那么表述形式是次要的。虽然软件团队的每个人在建模期间都应使用一致的表达方式，但模型最重要的特性是交流信息，以便软件工程执行下一个任务。如果一个模型可以成功地做到这一点，不正确的表达方式可以忽略。

⊖　本节给出的原则因本书目标已经做了删节和改写。

原则9：如果直觉告诉你模型不很妥当，尽管看上去很正确，那么你要仔细注意了。如果你是个有经验的软件工程师，就应相信你的直觉。软件工作中有许多教训——其中有些是潜意识的。如果有些事情告诉你设计的模型注定会失败（尽管你不能明确地证明），你有理由再花一些时间来检查模型或开发另一个模型。

原则10：尽可能快地获得反馈。每个模型都应经过软件团队的评审。评审的目的是为了提供反馈，用于纠正模型中的错误、改变误解，并增加不经意遗漏的功能和特性。

需求建模原则。在过去的30年里，已经开发出了大量的需求建模方法。研究人员已经弄清了需求分析中的问题及其出现原因，也开发出了各式各样的建模表达方法以及相关启发性解决方法。每一种分析方法都有其独立的观点。不过，所有的分析方法都具有共同的操作原则：

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存储（例如：永久存储的数据）。

KEY POINT

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服务并且为用户可见的特征提供内部支持。一些功能对流入系统的数据进行转换。在其他情况下，有些功能在某种程度上影响着对软件内部过程或外部系统元素的控制。功能可以以不同的抽象层次描述，这些抽象层次从对目标的笼统陈述到对那些必不可少的过程细节的描述。

原则3：必须描述软件的行为（作为外部事件的结果）。软件的行为受到与外部环境交互的驱动。最终用户提供的输入，由外部系统提供的控制数据，或者通过网络收集的监控数据都会引起软件的特定行为。

"工程师在设计时的首要问题是发现真正的问题"。——无名氏

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。需求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。

原则5：分析任务应该从本质信息转向实现细节。需求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宫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实现，对于视频游戏来说，可能需要用到语音输入。或者采用另一种办法，键入键盘命令，或者朝某个特定方向移动操纵杆（或者鼠标），或者在空中挥舞动作感应设备。

"首先要理解，设计时考虑周全而且合理的：一经证明是无误的、就要坚定地追求它；不要因为有人排斥已经确定的意图，使你动摇取得最终结果的决心。"——William Shakespeare

通过应用这些原则，软件工程师可以系统地解决问题。但这些原则如何应用到实践中呢？对这个问题的回答在第5章到第7章可以找到。

设计建模原则。软件设计模型类似于建筑师的房屋设计方案。首先表达所有需要建造的东西（例如，房屋的三维透视图），然后逐渐进行细化，为构建各个细节（例如，管线分布）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。

现在已经有许多方法可以导出各种软件设计的要素。一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最终的处理构件；另外一部分方法是模式驱动的，也就是使用问题域（需求模型）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，使用问题域对象来驱动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：设计可追溯到需求模型。需求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需求类，需求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需求类的构件。设计模型的各个元素都应该能追溯到需求模型。

WebRef

带有设计美学讨论、关于设计过程的深入评论可以在以下地址找到：cs.wwc.edu/~aabyan/Design/。

原则2：要始终关注待建系统的架构。软件架构（参看第9章）是待建系统的骨架，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。在设计中数据对象的实现方法绝不能忽略，一个结构良好的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，使得整个处理过程更为高效。

"差异并不小，就像Salieri和Mozart之间的差异。一项又一项的研究表明最优秀的设计师构建的结构更快、更小、更简单、更清晰、而且不甚费力。"——Frederick P.Brooks

原则4：必须精心设计接口（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构件集成变得更为容易并能辅助测试人员确认构件的功能。

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令人感到整个软件很糟糕。

原则6：构件级设计应是功能独立的。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，它应该关注于一个且仅仅一个功能或子功能[⊖]。

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，如构件接口、消息传递及全局数据。随着耦合程度的提高，错误传播的几率也会随之提高，整个软件的可维护性也会降低。因此，应该合理地保持较低的构件耦合度。

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、测试人员以及未来的维护人员传递信息，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，设计者都应该力求简洁。就像大多数创造性活动一样，设计是迭代完成的，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[Mye78]。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非常重要，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8章）。

4.3.4 构造原则

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（例如Java）；（2）使用待

⊖ 内聚的附加讨论在第8章。

开发构件的类似设计的中间表示来自动生成源代码；或者（3）使用第四代编程语言（例如Visual C++）自动生成可执行代码。

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——由客户检验系统所有要求的功能和特性。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。

“在生活中，我有软件窥探癖，常常偷看到别人很糟糕的代码。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

编码原则 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：

准备原则。在写下每行代码之前，要确保：

- 理解所要解决的问题。
- 理解基本的设计原则和概念。
- 选择一种能够满足构建软件以及运行环境要求的编程语言。
- 选择一种能提供工具以简化工作的编程环境。
- 构件级编码完成后进行单元测试。

编程原则。在开始编码时，要确保：

- 遵循结构化编程方法来约束算法[Boh00]。
- 考虑使用结对编程。
- 选择能满足设计要求的数据结构。
- 理解软件架构并开发出与其相符的接口。
- 尽可能保持条件逻辑简单。
- 开发的嵌套循环应使其易于测试。
- 选择有意义的变量名并符合相关编码标准。
- 编写注释，使代码具有自说明性。
- 增强代码的可读性（例如：缩进和空行）。

确认原则。在完成第一阶段的编码之后，要确保：

- 适当进行代码走查。
- 进行单元测试并改正所发现的错误。
- 重构代码。

ADVICE 避免开发一个解决错误问题的漂亮程序。特别要注意第一个准备原则。

讲述程序设计（编码）以及编码原则和概念的书比软件过程其他论题的书要多很多。如[Ker78]的编程风格入门，[McC04]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。关于这些原则和概念的广泛讨论不是这本书的涉及范围。如果你有更多的兴趣，可参看一个或更多上述参考书籍。

WebRef
各种关于编码标准的链接参看：www.literateprogramming.com/fpstyle.html。

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？

- 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。

在软件设计中，我们通常由着眼于软件架构的“宏观”层面开始，到着眼于构件模块的“微观”层面结束。在测试中，则正好相反。

如果测试成功（按照前面论述的目标），就会发现软件中的错误。测试的第二个好处就是，它能表明软件功能的执行似乎是按照规格说明来进行的，行为需求和性能需求似乎也可以得到满足。此外，测试时收集的数据为软件的可靠性提供了一个很好的说明，并且从整体看来也为软件质量提供了一些说明。但是测试并不能说明某些错误和缺陷不存在；它只能显示出现存的错误和缺陷。在测试时保持这样一个观念是非常重要的，其实这并不是悲观。

Davis[Dav95b]提出了一套测试原则[⊖]，本书对这些原则做了一些改动：

原则1：所有的测试都应该可以追溯到用户需求[⊜]。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之前就开始着手。测试计划（第17章）在分析模型一完成就应该开始。测试用例的详细定义可以在设计模型确定以后开始。因此，所有的测试在编码前都应该计划和设计好了。

原则3：将Pareto原则应用于软件测试。简单地说，Pareto原则认为在软件测试过程中80%的错误都可以在大概20%的程序构件中找到根源。接下来的问题当然就是要分离那些可疑的构件，然后对其进行彻底的测试。

原则4：测试应该从“微观”开始，逐步转向“宏观”。最初计划并执行的测试通常着眼于单个程序模块，随着测试的进行，着眼点要慢慢转向在集成的构件簇中找错误，最后在整个系统中寻找错误。

原则5：穷举测试是不可能的。即便是一个中等大小的程序，其路径排列组合的数目都非常庞大。因此，在测试中对每个路径组合进行测试是不可能的。然而，充分覆盖程序逻辑并确保构件级设计中的所有条件都通过测试是有可能的。

4.3.5　部署原则

正如我们在本书第一部分中提到的，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实质上是演化式或是增量式的，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。每个交付周期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。每个支持周期都会为部署周期中所提到的所有功能和特征提供一些文档和人员帮助。每个反馈周期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。

在提交软件增量前，要确保客户知道所期待的是什么。否则，可以保证，客户期待的一定会超出了你提交的。

软件增量交付对于任何一个软件项目来说都是一个重要的里程碑。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重要原则：

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更多的作出承诺后又很快令其失望。这将导致客户反馈变得没有积极意义并且还会挫伤软件开发团队的士气。Naomi Karten[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承诺；

⊖ 这里给出的仅仅是Davis 测试原则的一小部分。更多的信息参见[Dav95b]。

⊜ 这个原则是针对功能测试，也就是，测试重点在需求。结构测试（测试重点在体系结构的和逻辑的细节）可能没有直接涉及详细的需求。

在某次软件增量交付时交付了多于当初承诺要交付的工作，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。光盘或其他包含可执行软件的介质（包括Web下载包）以及一些支持数据文件、文档和一些相关的信息都必须组装起来，并经过实际用户完整的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配置（例如：硬件、操作系统、外围设备、网络）环境中实施充分的检验。

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准确的信息。如果技术支持跟不上或者根本就没有技术支持，那么客户会立即表示不满。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。

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[⊖]。

原则5：有缺陷的软件应该先改正再交付。迫于时间的压力，某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这样一条谚语：“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。

4.4 小结

软件工程实践包括概念、原则、方法和在整个软件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实践任务。

核心原则集有助于有意义的软件过程的应用以及有效的软件工程方法的执行。在过程级，核心原则建立了一个哲学基础，指导软件开发团队导引软件过程。在实践级，核心原则建立了一套价值准则，可以在分析问题、设计解决方案、实施方案以及测试解决方案以及最终向用户部署软件时提供指导。

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：减少争吵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。计划可以只是为某个软件增量而设计，或者也可以为整个项目而制定。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。建模原则就作为建立对软件进行表述的方法和注释的基础。

构造包括了编码和测试循环，在这个循环中为每个构件生成源码并对其进行测试。编码原则定义了一些通用动作，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。

部署发生在向客户展示每个软件增量的时候，它包括了交付、支持和反馈。交付的关键原

⊖ 在交流活动中，软件团队应该确定用户希望的帮助材料是什么类型。

则是管理客户期望并且能为客户提供合适的软件支持信息。支持需要预先准备。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迭代周期提供输入。

习题与思考题

4.1　由于关注质量需要资源和时间，那么既要敏捷又要维持对质量的关注可能吗？

4.2　在指导过程的8个核心原则中（在4.2.1节中讨论），你认为哪一个是最重要的？

4.3　用你自己的语言描述“关切问题分解”这个概念。

4.4　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？

4.5　在指导实践的8个核心原则中（4.2.2节），你认为哪一个是最为重要的？

4.6　敏捷沟通与传统的软件工程沟通有什么不同？又有哪些相似之处？

4.7　“转换话题”为什么是必要的？

4.8　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。

4.9　说明为什么项目策划需按迭代方式进行？

4.10　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？

4.11　在设计建模的过程中有哪3个软件“特征”需要考虑？

4.12　试着为4.3.4节中的编码添加一条附加的原则。

4.13　说明为什么穷举测试不能用以证明软件不含有错误？

4.14　你是否同意下面的说法：“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。

4.15　为什么对于软件开发团队来说反馈是至关重要的？

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。Withall的书（《Software Requirements, Patterns》，Microsoft Press，2007）中提出了大量解决沟通问题的有用模式。Sutliff的书（《User-Centred Requirement Engineering》，Springer，2002）中重点聚焦于与沟通相关的挑战。Weigers的书（《Software Requirements》，2nd ed. Microsoft Press, 2003），Pardee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。虽然并没有聚焦于软件，但Hooks和 Farry的书（《Customer Centered Products》，American Management Association，2000）中对与客户沟通提供了有用的通用指导方针。Young（《Effective Requirements Practice》，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。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。

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：Bechtold（《Essentials of Software Project Management》，2nd ed. Management Concept，2007），Wysocki（《Effective Project Management：Traditional，Adaptive，Extreme》，4th ed. Wiley，2006），Leach（《Lean Project Management：Eight Principles for Success》，BookSurge Publishing，2006），Hughes（《Software Project Management》，McGraw-Hill, 2005），以及Stellman 和Green（《Applied Software Project Management》，O'Reilly Media，Inc.，2005）。

软件工程原则在Davis[Dav95]的书中有非常好的整理和评述。此外，每一本软件工程书中实际上都

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（除了本书以外！）：

Abran，A.和J.Moore，《SWEBOK：Guide to the Software Engineering Body of Knowledge》，IEEE，2002.

Christensen，M.和R.Thayer，《A Project Manager's Guide to Software Engineering Best Practices》，IEEE-CS Press（Wiley），2002.

Jalote，P.，《An Integrated Approach to Software Engineering》，Springer，2006.

Pfleeger，S.，《Software Engineering: Theory and Practice》，3rd ed.，Prentice-Hall，2005.

Schach，S.，《Object-Oriented and Classical Software Engineering》，McGraw-Hill，7th ed.，2006.

Sommerville I.，《Software Engineering》，8th ed.，Addison- Wesley，2006.

这些书也对建模和创建原则作了详细的讨论。

建模原则可以参考很多着眼于需求工程和（或）软件设计的书。Lieberman的书（《The Art of Software Modeling》，Auerbach，2007），Rosenberg和Stephen的书（《Use Case Driven Object Modeling with UML：Theory and Practice》，Apress，2007），Roques的书（《UML in Practice》，Wiley，2004），Penker 和Eriksson的书（《Business Modeling with UML：Business Patterns at Work》，Wiley，2001）都讨论了建模原则和方法。

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优秀文章。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以及团队的能力对向客户提供巨大价值软件的影响。Stahl及其同事（《Model-Driven Software Development：Technology，Engineering》，Wiley，2006）讨论了模型驱动开发的原则。

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[Knu99]写过一套关于编码艺术的三卷的系列丛书，Hunt[Hun99]提出了实用的编程原则。

Myers及其同事（《The Art of Software Testing》，2nd ed.，Wiley，2004），主要修订了他的经典著作版本，并且讨论了许多重要的测试原则。Perry（《Effective Methods for Software Testing》，3rd ed.，Wiley，2006）、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/engcs/compsci/pressman/professional/olc/ser.htm上找到。

理解需求

要点浏览

概念：在开始任何技术工作之前，关注于一系列需求工程任务是个好主意。这些任务有助于你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（在IT界有时指系统工程师或分析员）和与项目利益相关的其他人员（项目经理、客户、最终用户）都将参与需求工程。

重要性：在设计和开发某个优秀的计算机软件时，如果软件解决的问题是错误的，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：需求工程首先从“起始”开始定义将要解决的问题的范围和性质；然后是导出，帮助利益相关者定义需要什么；接下来是精化，精确定义和修改基本需求。当利益相关者提出问题后，就要进行谈判：如何定义优先次序？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利益相关者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。不过依然可以得到工作产品：用户场景、功能和特征列表、需求模型或规格说明。

质量保证措施：利益相关者评审需求工程的工作产品，以确保你所理解的正是他们真正想要的。需要提醒大家的是：即使参与各方均认可，事情也会有变化，而且变化可能贯穿整个项目实施过程。

关键概念
分析模型
分析模式
协作
精化
导出
起始
谈判
质量功能部署
需求工程
需求收集
需求管理
规格说明
利益相关者
用例
确认需求
确认
视点
工作产品

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。第一次考虑开发一个清晰易于理解的需求的时候，可能看起来并不困难。试问，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性能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。

在Ralph Young[You01]的一本关于有效需求实践的书的前言中我写道：

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：已经作出交付期限的承诺，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推导出需求，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自己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人员和实际工作人员也会感到头痛。但是，确实存在解决方法。

把需求工程称做以上所述挑战的“解决方案”可能并不合理，但需求工程确实为我们提供了解决这些挑战的可靠途径。

5.1 需求工程

“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，并且以后修补竟会如此的困难。”——Fred Brooks

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利益相关者才能够更好地理解要求；事情变化太快，以至于理解需求细节是在浪费时间；最终要做的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[⊖]，但是这中间的每个论点都存在一些小问题，汇集在一起就可能导致软件项目的失败。

需求工程（Requirement Engineering，RE）是指致力于不断理解需求的大量任务和技术。从软件过程的角度来看，需求工程是一个软件工程动作，开始于沟通活动并持续到建模活动。它必须适应于过程、项目、产品和人员工作的需要。

KEY POINT 需求工程为设计和构造奠定了坚实的基础。如果没有需求工程，那么实现的软件很有可能无法满足客户的需求。

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利益相关者（如项目经理、客户、最终用户），也就是在他们那里定义业务需求，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。但是不管起始点在哪里，横跨这座桥梁将把我们带到项目之上更高的层次：允许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；以及将深切影响随后设计的信息、功能和行为。

ADVICE 可以在需求阶段作一些设计工作，在设计阶段作一些需求工作。

需求工程为以下工作提供了良好的机制：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转化为可运行系统[Tha97]。需求工程过程通过执行七个不同的活动来实现：起始、导出、精化、协商、规格说明，确认和管理。重要的是注意到，某些需求工程任务并行发生且要全部适应项目的要求。

起始

“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

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？或是要求随时间流逝而发展吗？这些问题没有确定的答案。某些情况下，一个偶然的交谈就可能必须投入大量的软件工程工作。但是多数项目都是当确定了商业要求或是发现了潜在的新市场、新服务时才开始。业务领域的利益相关者（如业务管理人员、市场人员、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，但是应该充分地与软件工程组织[⊖]及时讨论。

⊖ 对小项目（不超过一个月）和只涉及简单的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了。

⊖ 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。请访问本书所列的公司网站获取更多系统工程的详细讨论。

在项目起始阶段中[⊖]，要建立基本的理解，包括对问题、谁需要解决方案、所期望解决方案的性质、与项目利益相关者和开发人员之间达成初步交流合作的效果。

导出

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，这些问题是非常简单的。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，可以帮助理解为什么导出需求这么困难。

为什么对客户的需要获得清晰的理解会非常困难？

- **范围问题**：系统的边界不清楚，或是客户或用户的说明带有不必要的技术细节，这些细节可能会导致混淆而不是澄清系统的整体目标。
- **理解问题**：客户或用户并不能完全确定需要什么，他们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，与系统工程师在沟通上存在问题，省略那些他们认为是“明显的”信息，确定的需求和其他客户或用户的需求相冲突，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组织的方式开展需求收集活动。

精化

精化是件好事，但你必须知道何时可以停止精化。关键是能采用为设计建立一个坚实基础的方式说明问题。如果超出这个点就是在做设计。

在起始和导出阶段获得的信息将在精化阶段进行扩展和提炼。该任务集中于开发一个精确的需求模型（第6、7章），用以说明软件的功能、特征和信息的各个方面。

精化是由一系列的用户场景建模和求精任务驱动的。这些用户场景描述了如何让最终用户和其他参与者与系统进行交互。解析每个用户场景以便提取分析类——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[⊖]，确定类之间的关联和协作关系，并完成各种补充图。

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“交易”的坚实基础。

只给定了有限的业务资源，而客户和用户提出了过高的要求，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”。

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、用户和其他利益相关者对各自的需求排序，然后按优先级讨论冲突。使用迭代的方法给需求排序，评估每项需求对项目产生的成本和风险，表述内部冲突，删除、组合和（或）修改需求，以便参与各方均能达到一定的满意度。

规格说明

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。

在基于计算机的系统（和软件）的环境下，术语规格说明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、一个形式化的数学模型、一组使用场景、一个原型或上述各项的任意组合。

有人建议应该开发一个“标准模板”[Som97]并用于规格说明，他们认为这样将促使以一致的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，文档最好采用自然语言描述和图形化

⊖ 应该记得第2章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、导出和精化工作。

⊖ 服务通过对类的封装操作数据，也可使用术语“操作”和“方法”。如果你不熟悉面向对象的概念，附录2有基本的入门指导。

模型来编写。而对于技术环节明确的较小产品或系统，使用场景可能就足够了。

INFO

软件需求规格说明模板

软件需求规格说明（SRS）是在项目商业化之前必须建立详细描述软件各个方面的文档。值得注意的是常常并没有写正规的SRS。事实上很多实例表明花在软件需求规格说明的工作量还不如投入到其他软件工程活动。然而，当软件由第三方开发时，当缺少规格说明导致严重业务问题时，或是当系统非常复杂或涉及十分重要的业务时，才能表明需求规格说明是非常必要的。

Process Impact公司的Karl Wiegers[Wie03]开发了一套完整的模板(参考www.processimpact.com/process_assets/srs_template.doc)能为那些必须建立完整需求规格说明书的人提供指导。大纲如下：

目录
版本历史
1 导言
1.1 目的
1.2 文档约定
1.3 适用人群和阅读建议
1.4 项目范围
1.5 参考文献
2 总体描述
2.1 产品愿景
2.2 产品特性
2.3 用户类型和特征
2.4 操作环境
2.5 设计和实现约束
2.6 用户文档
2.7 假设和依赖
3 系统特性
3.1 系统特性1
3.2 系统特性2（等等）
4 外部接口需求
4.1 用户接口
4.2 硬件接口
4.3 软件接口
4.4 通信接口
5 其他非功能需求
5.1 性能需求
5.2 安全需求
5.3 保密需求
5.4 软件质量属性
6 其他需求
附录A：术语表
附录B：分析模型
附录C：问题列表

每个需求规格说明主题的详细描述可以从前面所提的URL下载SRS模板来得到。

ADVICE

需求确认时的一个重要问题是一致性。使用分析模型可以保证需求说明保持一致性。

确认

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：已无歧义地说明了所有的系统需求；已检测出不一致性、疏忽和错误并得到纠正；工作产品符合为过程、项目和产品建立的标准。

正式的技术评审（第15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、用户和其他利益相关者，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）、冲突的需求或是不可实现的（不能达到的）需求。

⊖ 应该记得每个项目有不同的规格说明特性。在某些情况下，规格说明是指收集到的用户场景和其他一些事物。在另一些情况下，规格说明可以包括用户场景、模型和写成说明书的文档。

INFO

需求确认检查表：

通常，按照检查表上的一系列问题检查每项需求是非常有用的。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？
- 其他哪些需求和此需求相关？是否已经使用交叉索引矩阵或其他机制清楚地加以说明？
- 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，能否说明检验需求的测试（有时称为确认准则）？
- 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、轻松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？
- 与系统性能、行为及运行特征相关需求的说明是否清楚？哪些需求是隐含出现的？

需求管理

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需求变更的一组活动[⊖]。这类活动中的大部分内容和第22章中讨论的软件配置管理（SCM）技术是相同的。

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。

工作机制：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：[⊜]

Volere需求资源网站（www.volere.co.uk/tools.htm）提供了一个相当全面（也是最新的）列表。我们在第6、7章将讨论需求建模工具。下面提到的工具主要侧重于需求管理。

EasyRM：由Cybernetic Intelligence GmbH（www.easy-rm.com）开发，构建项目专用的字典或术语，包含详细的需求说明和属性。

Rational RequisitePro：由Rational Software（www-306.ibm.com/software/awdtools/reqpro/）开发，允许用户构建需求数据库，描述需求之间的关联，组织、优先级排序以及跟踪需求。

从前面所提的Volere网站和WWW.jiludwig.com/requirements_management_tools.html，还有另外一些的需求管理工具。

⊖ 正规的需求管理只适用于具有数百个可确认需求的大型项目。对于小项目，该需求工程工作可以适当的裁减，一定程度的不正规也是可以接受的。

⊜ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

5.2 建立根基

在理想情况下，利益相关者和软件工程师在同一个小组中工作[⊖]。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。

KEY POINT
利益相关者是那些对将要开发的系统有直接的兴趣或直接从中获益的人。

客户或最终用户可能位于不同的城市或国家，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有冲突的意见，他们的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，以便理解软件需求。这是项目自始至终走向成功解决方案的方向。

5.2.1 确认利益相关者

Sommerville和Sawyer[Som97]把利益相关者定义为“直接或间接地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利益相关者：业务运行管理人员、产品管理人员、市场销售人员、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。每个利益相关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是不同的。

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(5.3节)。最初的人员列表将随着接触的利益相关者人数增多而增加，因为每个利益相关者都将被询问“你认为我还应该和谁交谈”。

5.2.2 识别多重观点

把三个利益相关者请进一个房间，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

因为存在很多不同的利益相关者，所以系统需求调研也将从很多不同的视角开展。例如，市场销售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哪些非技术背景的利益相关者看不到的软件基础设施能够支持更多的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。

这些参与者（以及其他人）中的每一个人都将为需求工程过程贡献信息。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是相互矛盾。需求工程师的工作就是把所有的利益相关者提供的信息（包括不一致或是矛盾的需求）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。

5.2.3 协同合作

假设一个项目中有五个利益相关者，那么对一套需求就会有五种或更多的正确观点。我们提醒客户（和其他利益相关者）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（即所有利益相关者都同意的需求）和矛盾区域或不一致区域（即某个利益相关者提出的需求和其他利益相关者的需求相矛盾）。很明显，后一种分类更有挑战性。

⊖ 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。

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。所有的利益相关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。每个利益相关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（从他或她的个人观点）该需求的相对重要性。优先点用过之后就不能再次使用，一旦某个利益相关者的优先点用完，他就不能再对需求实施进一步的操作。所有利益相关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，利益相关者的协作是提供他们各自关于需求的观点，而一个强有力的“项目领导者”（例如业务经理或高级技术员）可能要对删减哪些需求做出最终决策。

5.2.4 首次提问

在项目需求导出时的提问应该是“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利益相关者、整体目标、收益。例如，需求工程师可能会问：

“知道问题比知道所有的答案更好。”——James Thurber

- 谁是这项工作的最初请求者？
- 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？
- 对于这个解决方案你还需要其他资源吗？

什么问题有助于你获得对问题的初步认识？

这些问题有助于识别所有对构建软件感兴趣的利益相关者。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。

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：

“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

- 如何描述由某成功的解决方案产生的“良好”输出的特征？
- 该解决方案强调解决了什么问题？
- 能向我们展示（或描述）解决方案使用的商业环境吗？
- 存在将影响解决方案中特殊的性能问题或约束吗？

最后一组问题关注于沟通活动本身的效率。Gause 和Weinberg[Gau89]称之为“元问题”。下面并给出了元问题的简单列表：

- 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？

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）并非是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，然后应该用需求诱导方式包括问题求解、协商和规格说明取代。在5.3节中将介绍这类方法。

5.3 导出需求

导出需求（又称为需求收集）是与问题求解、精化、谈判和规格说明等方面的元素结合在

一起的。为了鼓励合作，一个包括利益相关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，商讨不同的方法并描述初步的需求解决方案 [Zah90]⊖。

5.3.1 协作收集需求

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，各种方法适用于稍有不同的场景，而且所有这些均是在下面的基本原则之上作了某些改动：

主持一个协作的需求收集会议的基本原则是什么？

- 会议由软件工程师和其他的利益相关者共同举办和参与。
- 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，使其涵盖所有的重点；但也不能太正式，以鼓励思想的自由交流。
- 由一个“调解人”（可以是客户、开发人员或其他人）控制会议。
- 采用“方案论证手段”（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）。

“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

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确定一套解决需求问题的初步方案。为了更好地理解在会议召开时不断发生的事件流，我们提出了一个概略的场景，用其表述在需求收集会议中及会议后将发生的一些事件。

在需求的起始阶段（5.2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。除了这些启动会议之外，开发人员和客户要写下一个1～2页的“产品要求”。

WebRef

联合应用程序开发（JAD）是需求收集普遍采用的技术。可以从以下地址找到详细的说明：www.carolla.com/wp-jad.htm。

此外还要选择会议地点、时间和日期，并选择调解人。来自开发组和其他利益相关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。

举一个例子⊜，考虑SafeHome项目中的一个市场营销人员撰写的产品要求。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。

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，那么接收风险将会非常高。

住宅安全功能应该为防止和（或）识别各种不希望出现的“情况”提供保护，如非法入侵、火灾、漏水、一氧化碳浓度超标等。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，也有可能发生错误。但对此时而论，上面的“功能描述”是足够了。

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，

⊖ 这种方法有时称为协助应用规格说明技术（Facilitated Application Specification Technique, FAST）

⊜ SafeHome例子（有一定的扩展和改动）在下面很多章节中用于说明软件工程的重要方法，作为一个练习，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是值得的。

"事实不会因忽视它们而不存在。" —— Aldous Huxley

ADVICE 一定要避免攻击客户意见"太昂贵"或"不实际"这样的严厉谴责。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，必须保持一个开放的思想。

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。

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）。

这些对象列表可以用大的纸张钉在房间的墙上或用便签纸贴在墙上或写在墙板上。或者，列表也可以贴在内部网站的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，以便于合并列表、删除项以及加入新项。在本阶段，严禁批评和争论。

当某一专题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，由调解人主持开始讨论协调。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或系统。目标是为所开发系统的对象、服务、约束和性能提交一个意见一致的列表。

在很多情况下，列表所描述的对象或服务需要更多的解释。为了完成这一任务，利益相关者为列表中的条目编写小规格说明（mini-specification）[⊖]。每个小规格说明是对包含在列表中的对象和服务的精练，例如，对SafeHome对象**控制面板**的小规格说明如下：

控制面板是一个安装在墙上的装置，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3×3的LCD彩色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。

将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。

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是软件团队成员；Doug Miller是软件工程经理；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。

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？

⊖ 除了创建小规格说明，很多软件团队选择创建用户场景，即所谓的用例（use-case）。这些在5.4节和第6章中有详细说明。

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、使系统失效、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……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，然后继续讨论（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20分钟提炼并扩展住宅安全功能的细节。）

5.3.2 质量功能部署

QFD以最大限度地满足客户的方式来定义需求。

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质量管理技术。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。

每个人都希望实现大量的“令人兴奋的需求”，但是必须小心，那是“需求蔓延”开始的原因。然而另一方面，经常是令人兴奋的需求才最终导致突破性的产品！

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非常不满。这方面的例子如：人机交互的容易性、整体的运行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，新移动电话的软件来自标准特性，但关联了一些超出期望的能力（例如多重触控技术的触摸屏，可视语音邮箱），这些能力让产品的用户很欣喜。

虽然QFD概念可应用于整个软件过程[Par96]，但是特定的QFD技术可应用于需求提取活动。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，并由客户和利益相关者评审。接下来使用各种图表、矩阵和评估方法抽取期望的需求并尽可能导出令人兴奋的需求[Aka04]。

WebRef

有关QFD的更多信息可从以下地址获得：www.qfdi.org。

5.3.3 用户场景

当收集需求时，系统功能和特性的整体愿景开始具体化。但是在软件团队弄清楚不同类型的最终用户如何使用这些功能和特性之前，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系统的使用线索。场景通常称为用例[Jac92]，它提供了将如何使用系统的描述。在5.4节中将更详细地讨论用例。

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是软件团队成员；Doug Miller是软件工程经理；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。）

5.3.4 导出工作产品

什么样的信息是需求收集产生的？

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：

- 要求和可行性陈述。
- 系统或产品范围的界限说明。
- 参与需求导出的客户、用户和其他利益相关者的名单。
- 系统技术环境的说明。
- 需求列表（最好按照功能加以组织）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。

所有参与需求导出的人员需要评审以上的每一个工作产品。

5.4 开发用例

在一本讨论如何编写有效用例的书中，Alistair Cockburn[Coc01b]写道："一个用例捕获一个合同……即说明当对某个利益相关者的请求响应时，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形表示。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。

KEY POINT 从参与者的角度定义用例。参与者是人员（用户）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确定故事中所包含的"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。

WebRef 可以从以下地址下载一篇非常好的关于用例的论文：www.ibm.com/developerworks/webservices/library/codesign7.html

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（其中装有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4种不同的交互模式（角色）：编程模式、测试模式、监测模式和故障检查模式。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。

为了开发有效的用例我需要知道什么？

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。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是要获取所需的系统功能并从系统得到预期收益。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[⊖]：

• 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有关外部环境的改变吗？
• 参与者希望从系统获取什么信息？

⊖ Jacobson的问题已经被扩展到为用例场景提供更复杂的视图。

• 参与者希望得知会有意料之外的变更吗？

回顾基本的SafeHome需求，我们定义了4个参与者：**房主**（用户）、**配置管理人员**（很可能就是**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操作。
• 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。
• 激活或关闭安全系统。

考虑房主使用控制面板的情况，系统激活的基本用例如下[⊖]：

1. 房主观察SafeHome控制面板（图5-1），以确定系统是否已准备好接收输入。如果系统尚未就绪，“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或门以使得“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5-1）启动系统。“stay”只激活外部传感器（内部的运行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。

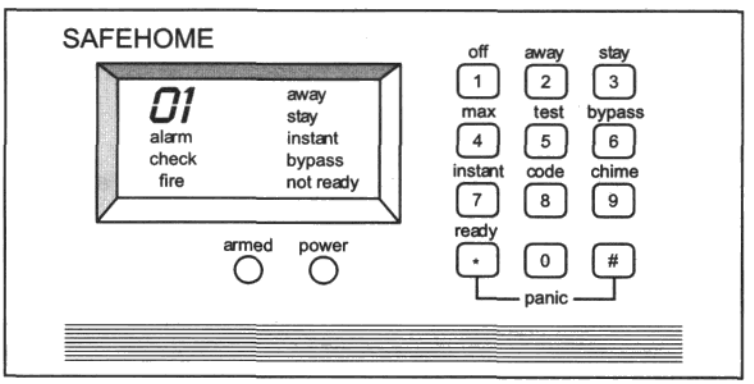


图5-1 SafeHome控制面板

用例通常写得不规范，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。

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，进一步细化用例以便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：

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。

⊖ 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图形用户接口GUI。

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. 控制面板没有准备就绪：房主检查所有的传感器，确定哪些是开着的（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；激活边界传感器。
5. 选择away：控制面板鸣叫三声并且away灯点亮；激活所有传感器。

优先级：必须实现。

何时可用：第一个增量。

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。

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房主的交互用例。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是软件团队成员；Doug Miller是软件工程经理；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5-2。）

Jamie：我恰好刚开始学习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？

⊖ 对于不熟悉UML符号的读者请参考附录1介绍UML的基本指南。

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，我使用作了标记的矩形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。）

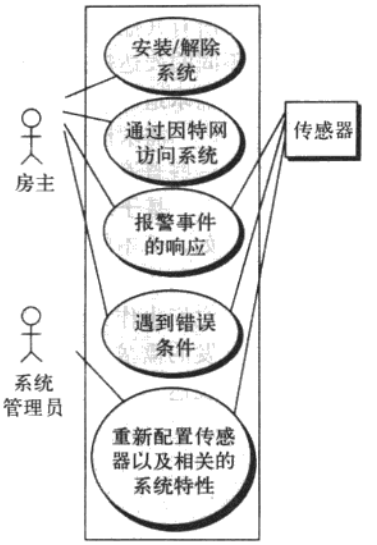


图5-2 住宅安全系统功能的UML用例图

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：[⊖]

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：（www.objectsbydesign.com/tools/umltools_ byCompany.html）提供对该类工具的全面链接。

5.5 构建需求模型[⊜]

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。随着软件工程师更多地了解将要实现的系统以及其他利益相关者更多地了解他们到底需要什么，模型将动态的变更。因此，分析模型是任意给定时刻的需求快照，我们对这种变更更应有思想准备。

当需求模型演化时，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，这表明利益相关者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6、7章详细说明，下面小节仅提供简要的概述。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

⊜ 本书通篇把分析模型和需求模型作为同义词使用。它们都是用于描述信息、功能和行为领域问题的需求。

5.5.1 需求模型的元素

有很多不同的方法考察计算机系统的需求。某些软件人员坚持最好选择一个表达模式（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需求模型是值得的，不同的表达模式促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。

ADVICE

把利益相关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利益相关者写下描述将如何使用软件的用例。

需求模型的特定元素取决于将要使用的分析建模方法（第6、7章）。但是，一些普遍的元素对大多数分析模型来说都是通用的。

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（5.4节）及其相应的用例图（图5-2）演化成更精细的基于模板的用例。需求模型基于场景的元素通常是正在开发模型的第一部分。同样，它们也作为创建其他建模元素时的输入。例如，图5-3描绘了一张使用用例引发的需求并表达它们的UML活动图[⊖]。给出了最终基于场景表示的三层详细表达。

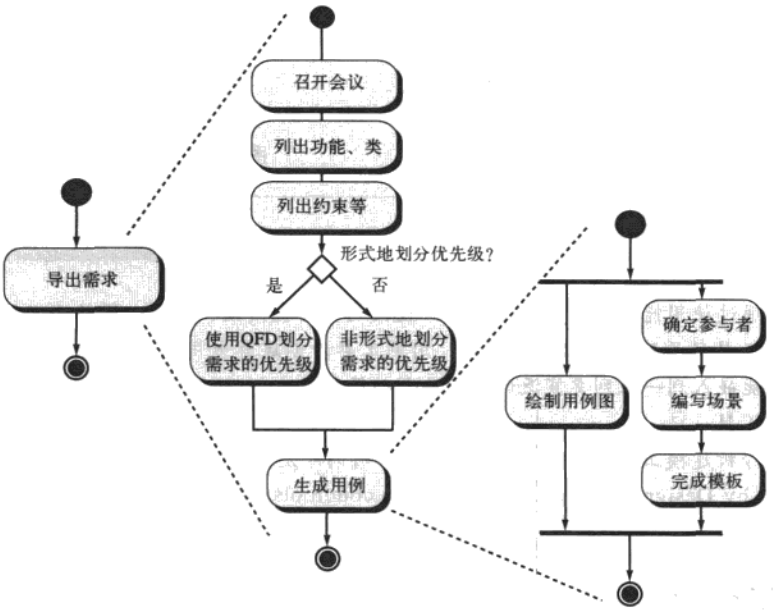


图5-3 导出需求的UML活动图

ADVICE

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章中将有更多内容说明这一点。

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，可以用UML类图描绘SafeHome安全功能的传感器Sensor类如图5-4所示。注意，UML类图列出了传感器的属性（如name，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

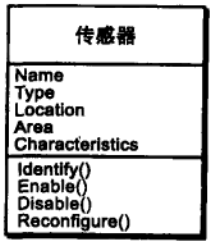


图5-4 传感器Sensor的类图

⊖ 对于不熟悉UML符号的读者请参考附录1介绍UML的简要指南。

相互之间的协作以及类之间的关联和交互。在第7章中将有更详细的讨论。

行为元素：基于计算机系统的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，需求分析模型必须提供描述行为的建模元素。

KEY POINT
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。

状态图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）。

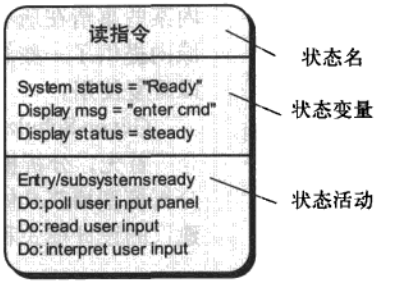


图5-5 UML状态图表示

为了更好地说明状态图的使用，考虑将软件嵌入到SafeHome的控制面板负责读取用户的输入信息。简化的UML状态图如图5-5。

另外作为一个整体的系统行为表达也能够建模于各个类的行为之上。第7章将有更多关于行为建模的讨论。

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是软件团队成员；Doug Miller是软件工程经理；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：你这家伙开始转过弯儿了，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。

面向数据流的元素：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；使用函数将其转换；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文件。转换可能由单一的逻辑比较、复杂的数值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）创建数据流模型。第7章将更详细地讨论流的建模。

5.5.2 分析模式

"析衷是这样一种艺术：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

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[⊖]。这些分析模式[Fow97]在特定应用领域内提供了一些解决方案（如类、功能、行为），在为许多应用项目建模时可以重复使用。

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转变到设计模型。

通过参照模式名把分析模式整合到分析模型中。同时还将存储在仓库中以便需求工程师能通过搜索工具发现并应用。在标准模板[Gey01][⊜]中会提供关于分析模式（和其他类型模式）的信息，更多细节在第12章讨论。分析模式的样例和有关这一论题更多的讨论在第7章中。

5.6 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。但遗憾的是，这几乎不可能发生。实际中，一个或多个利益相关者恐怕得进入到协商的过程中，在多数情况下要让利益相关者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，在满足利益相关者要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）。

WebRef

可以从以下地址下载一篇关于软件需求协商的简短文章：www.alexanderegyed.com/publications/Software_Requirements_Negotiation Some_ Lessons_ Learned.html。

最好的协商是争取"双赢"的结果[⊜]，即利益相关者的"赢"在于获得满足客户大多数需要的系统或产品，而作为软件团队一员的"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。

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利益相关者。
2. 确认利益相关者"赢"的条件。
3. 就利益相关者"赢"的条件进行协商，以便使其与所有涉及人（包括软件团队）的一些双赢条件一致。

这些初期步骤的成功实施可以达到双赢的结果，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。

⊖ 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。

⊜ 文献中已经提出了各种各样的模式模板，感兴趣的读者可以参阅[Fow97]、[Bus07]、[Yac03]和[Gam95]。

⊜ 有许多关于谈判技巧的书籍（如[Lew06]、[Rai06]、[Fis06]）。这是一个应该学习的许多重要技巧之一。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……他们确实有很多贡献了。

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（皱眉）：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次发布的产品中加以考虑。

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。我们还必须在首次发布的产品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次发布的产品再予考虑……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？

5.7 确认需求

当需求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利益相关者划分优先级并组合成一个整体，该需求整体将以软件逐步加以实现。需求模型的评审将提出如下问题：

- 每项需求都和系统或产品的整体目标一致吗？
- 所有的需求都已经在相应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。

当我评审需求时，我将提出什么问题？

- 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？
- 每项需求都有界定且无歧义吗？
- 每项需求都有归属吗？换句话说，是否每个需求都标记了来源（通常是一个明确的个人）？
- 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，每项需求是可测试的吗？
- 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吗？
- 已经使用了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？

应当提出以上这些问题和其他一些问题，并回答问题，以确保需求模型精确地反映利益相关者的要求并为设计奠定坚实的基础。

5.8 小结

需求工程的任务是为设计和构建活动建立一个可靠坚固的基础。需求工程发生在与客户沟通活动和为一般的软件过程定义的建模活动过程中。软件团队成员要实施7个不同的需求工程职能：起始、导出、精化、协商、规格说明、确认和管理。

在项目起始阶段，项目利益相关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，必须让系统表现出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和使用场景的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、基于类、行为和面向数据流的模型元素的集合。模型可能对分析模式和在不同的应用系统中重复出现分析问题的解决方案加以注解。

当确定需求并且创建分析模型时，软件团队和其他项目利益相关者协商优先级、可用性和每条需求的相对成本。协商的目标是开发一个现实可行的项目计划。此外，将按照客户需求确认每个需求和整个需求模型，以确认将要构建的系统对于客户的要求是正确的。

习题与思考题

5.1 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？

5.2 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？

5.3 讨论一下当需求必须从3、4个不同的客户中提取时会发生什么问题。

5.4 为什么我们说需求模型表现了系统的时间快照？

5.5 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，这能够让你成为一个高明的协商人员吗？为什么？

5.6 想出三个以上在需求起始阶段可能要问利益相关者的“与情景无关的问题”。

5.7 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它任何可能有助于定义需求的条款。

5.8 你的指导老师将把班级分成4或6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。

5.9 为如下活动之一开发一个完整的用例：

a. 在ATM提款。

b. 在餐厅使用信用卡付费。

c. 使用一个在线经纪人账户购买股票。

d. 使用在线书店搜索书（某个指定主题）。

e. 你的指导老师指定的一个活动。

5.10 用例“异常”代表什么？

5.11 用你自己的话描述一个分析模式。

5.12 使用5.5.2节描述的模板，为下列应用领域建议1~2个分析模式：

a. 会计软件系统

b. E-Mail电子邮件系统

c. 互联网浏览器

d. 字符处理系统

e. 网站生成系统

f. 由指导老师特别指定的应用领域

5.13 在需求工程活动的谈判情境中，“双赢”意味着什么？

5.14 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？

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系统的关键，所以大量的书籍都在讨论需求工程。Hood和他的同事（《Requirements Management》，Springer，2007）讨论了大量需求工程关于横跨系统硬件和软件工程的问题。Young（《The Requirements Engineering Handbook》和Artech House publishers，2007）对需求工程任务进行了更深层次的讨论。Wiegers（《More About Software Requirements》和Microsoft Press，2006）提供了很多需求收集和需求管理的技术实践。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，这些书都讨论了该主题。Gottesdiener（《Requirements by Collaboration:Workshops for Defining Needs》，Addison-Wesley，2002）为项目利益相关人员协同收集需求环境提供非常有用的指导。

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》，2nd ed.，Addison-Wesley，2003）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。

Withall（《Software Requirement Patters》，Microsoft Press，2007）描述了基于模式视图的需求工程。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，讨论该主题的资料有Rosenberg和Stephens（《Use Case Driven Object Modeling with UML:Theory and Practice》，Apress, 2007）、Denny（《Succeed with Use Cases:Working Smart to Deliver Quality》，Addison-Wesley, 2005）、Alexander 和Maiden（《Scenarios,Stories,Use Cases:Through the Systems Development Life-Cycle》，Wiley, 2004）、Leffingwell和他的同事（《Managing Software Requirements: A Use Case Approach》，2nd ed., Addison-Wesley，2003）表述了非常有用的需求收集的最佳实践。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）讨论中强调使用用例模型进行需求收集。

在Internet上有大量的、丰富的关于需求工程和分析的信息资源。与需求工程和需求分析相关的最新的Web引用列表可以在SEPA的Web站点http://www.mhhe.com/ engcs/ compsci/ pressman/ professional/ olc/ser.htm上找到。

需求建模：场景、信息与类分析

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，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。本章将从三个不同的视角考虑需求建模：基于场景的模型、数据（信息）模型和面向类的模型。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：基于场景的建模从用户的角度表现系统；数据建模提出了信息空间同时描述了软件要加工的数据对象以及数据对象间的关系；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。在创建了模型的雏形以后，就要对其不断改进，并分析评估其清晰性、完整性和一致性。在第7章扩展了模型维度，增加了表示方法，提供了更完善的需求视图。

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：必须评审需求建模工作产品的正确性、完整性和一致性，必须反映所有利益相关者的要求并建立一个可以从中导出设计的基础。

关键概念
活动图
分析类
分析包
关联
基于类建模
CRC建模
数据建模
域分析
语法解析
需求建模
基于场景建模
泳道图
UML模型
用例

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设计表示。需求模型实际上是一组模型[⊖]，是系统的第一个技术表示。

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：

> 回顾已确认分析阶段的问题和过失，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析建模方法和表示方法。

⊖ 在本书过去的版本中，我使用分析模型而不是需求模型。在这一版本中我决定使用这两个用语，以便表达在解决问题的多个方面时定义的建模活动。当导出需求时分析其中的动作。

6.1 需求分析

需求分析产生软件工作特征的规格说明，指明软件和其他系统元素的接口，规定软件必须满足的约束。需求分析让软件工程师（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的起始、导出、谈判任务中建立的基础需求（第5章）。

"任何一个需求'视图'都不足以理解和描述一个复杂系统所需的行为。"——Alan M. Davis

需求建模动作产生为以下一种或多种模型类型：

- 场景模型：出自各种系统"参与者"观点的需求。
- 数据模型：描述问题信息域的模型。
- 面向类的模型：表示面向对象类（属性和操作）的模型，其方式为通过类的协作获得系统需求。
- 面向流程的模型：表示系统的功能元素并且描述当功能元素在系统中运行时怎样进行数据变换的模型。
- 行为模型：描述如何将软件行为看做是外部"事件"后续的模型。

KEY POINT
一旦软件完成后，分析模型和需求规格说明书将成为提供评估软件质量的手段。

这些模型为软件设计者提供信息，这些信息可以转化为结构、接口和构件级的设计。最终，在软件开发完成后，需求模型（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。

本章我关注基于场景的建模，这项技术在整个软件工程界迅猛发展；数据建模是较为特殊的技术，适用于当一个应用问题必须生成或处理一个复杂的信息空间时；面向类建模表示面向对象类和允许的系统功能间的有效协作。面向流程的模型、行为模型、基于模式的建模和Web应用模型将在第7章讨论。

6.1.1 总体目标和原理

需求不是架构。需求既不是设计，也不是用户接口。需求就是指需要的。——Andrew Hunt 和 David Thomas

在整个需求建模过程中，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：在特定环境下发生哪些用户交互？系统处理什么对象？系统必须执行什么功能？系统展示什么行为？定义什么接口？有什么约束[⊖]？

KEY POINT
分析模型应该描述什么是客户所需，应该建立设计的基础，建立有效的目标。

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，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[⊜]。

需求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和软件设计（第8章至第13章）之间建立了桥梁。这里的系统级描述给出了在软件、硬件、数据、人员和其他系统元素共同作用下的整个系统或商业功能，而软件设计给出了软件的应用程序结构、用户接口以及构件级的结构。这个关系如图6-1所示。

⊖ 应该注意当客户是更为老练的技术人员时，规格说明书应侧重How同What一样重要。但是，基本观点应保留在What上。

⊜ 或者软件团队可以花些功夫选择生成一个原型(第2章)，以便更好地理解系统的需求。

重要的是要注意需求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。

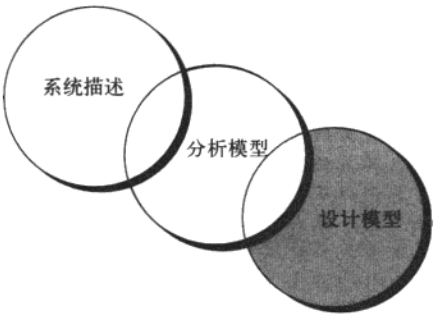


图6-1　分析模型在系统描述和设计模型之间建立桥梁

6.1.2　分析的经验原则

我们做需求分析时有没有可以帮助我们的指南？

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。

“值得进攻的问题总是通过反击证明其价值。”——Piet Hein

- 需求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、访问数据库所需的功能，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。
- 确认需求模型为所有利益相关者都带来价值。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，就不要使用复杂的表示方法。

6.1.3　域分析

在需求工程讨论中（第5章），我们注意到分析模式通常在特定业务领域内的很多应用系统中重复发生。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。

WebRef

可以从以下地址找到很多关于域分析的有用信息：www.iturls.com/english/SoftwareEngineering/SE_mod5.asp。

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：

> 软件域分析是识别、分析和详细说明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需求……[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。

“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：查找或创建那些广泛应用的分析类和（或）分析模式，使其能够复用[⊖]。

⊖　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利益相关者可以更好的学习……不是所有域的类都必然导致可复用的类”[Let03]。

KEY POINT
域分析不关注特定的应用系统，相反关注应用所属的领域。其目的在于识别那些解决可用于域内所有应用系统的共同问题。

使用在本书前面介绍的术语，域分析可以被看做是软件过程的一个普适性活动。意思是域分析是正在进行的软件工程活动，而不是与任何一个软件项目相关的。域分析师的角色有些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行类似的而不一定是同样的工作。域分析师[⊖]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。

图6-2 [Ara89]图示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整个领域内复用的对象。

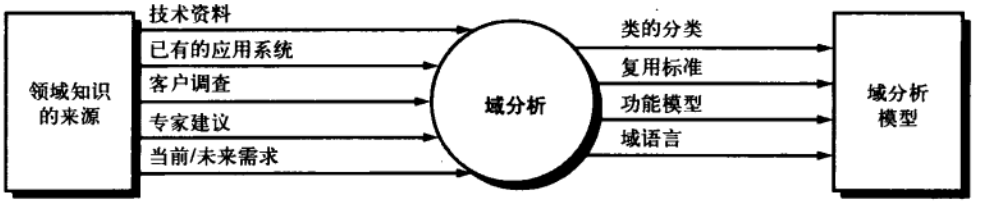


图6-2 域分析的输入和输出

SAFEHOME

域分析

[场景] Doug Miller的办公室，在销售业务会议之后

[人物] Doug Miller软件工程经理，Vinod Raman 软件工程团队成员

[对话]

Doug：我需要你做一个特殊的项目，Vinod。我将会把你从需求收集会议中抽出来。

Vinod（皱眉不悦）：太糟了。这可行吗……我正在从中获取了一些需求。出什么事啦？

Doug：Jamie和Ed将接管你的工作。不管怎样，市场部坚持要我们在第一次发布的SafeHome中交付具有互联网能力的家庭安全功能。我们一直紧张地为此工作着……没有足够的时间和人力，所以我们马上要解决PC机接口和Web接口两个问题。

Vinod（看上去很疑惑）：我不知道原先设定的计划……我们甚至还没有完成需求收集。

Doug（无精打采地微笑）：我知道，但时间太紧了我决定马上和市场部开始战略合作……无论怎样，一旦从所有需求收集会议上获得信息，我们将重新审视任何不确定的计划。

Vinod：好的，会发生什么事？你要我做什么事吗？

Doug：你知道“域分析”吗？

Vinod：略知一些。在建立应用系统时为做相同工作的应用系统寻找相似的模式。如果可能，可以在工作中剽窃这些模式并复用它们。

Doug：我觉得用剽窃这个词不太恰当，但你的意思基本是正确的。我想让你做的事是开始研究为控制像SafeHome这类系统的现存用户接口。我想你要组织一套模式和分析类，它们通常既能坐在房间里处理基于PC机的接口，也能处理通过互联网进入基于浏览器的接口。

Vinod：把它们做成相同的可以节省时间……为什么不这么做呢？

⊖ 不要认为因为有域分析员在工作，软件工程师就不需要理解应用问题的领域。软件团队的每个成员都应该在一定程度上了解软件将要工作的领域。

Doug：哦……有你这种想法的人真是非常好。整个核心要点是如果能识别出两种接口，采用相同的接口等，我们就节省了时间和人力。这些正是市场部坚持的。

Vinod：那你想要什么？类、分析模式还是设计模式？

Doug：所有的。非正式地说这就是关键所在。我就是想稍早一些开始我们内部分析和设计工作。

Vinod：我将在类库中看看我们得到了哪些。我也会使用几个月前我读的一本书中的模式模板。

Doug：太好了，继续工作吧。

……分析容易使人灰心丧气，全都是非常复杂的人际关系，不确定的和困难的东西。总而言之，分析让人着迷。一旦沉迷，原来轻松构建系统的快乐将难以令你满足。——Tom DeMarco

6.1.4 需求建模的方法

一种考虑数据和处理的需求建模方法被称作结构化分析，其中处理将数据作为独立实体加以转换。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。

需求建模的第二种方法称做面向对象的分析，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2章）主要是面向对象的分析方法。

尽管本书中我们提出的需求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利益相关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。

在描述需求模型时常用哪些不同的观点？

如图6-3所示需求模型的每个元素表示源自不同观点的问题。基于场景的元素表述用户如何与系统和使用软件时出现的特定活动序列进行交互。基于类的元素建模于系统操作的对象，应用在这些对象间影响操作和对象间关系（某层级）的操作，以及定义的类间发生的协作。行为元素描述了外部事件如何改变系统或驻留在系统里的类的状态。最后，面向流的元素表示信息转换的系统，描述了数据对象在流过各种系统功能时是如何转换的。

"我们为什么要构建模型？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

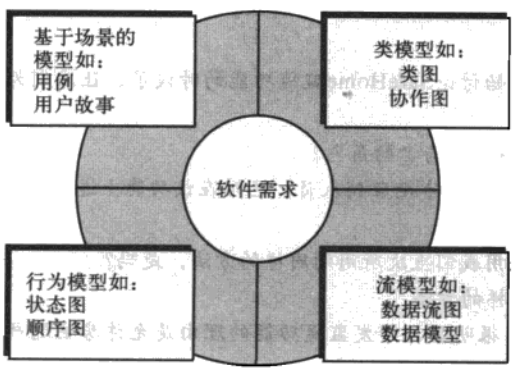


图6-3　需求模型的元素

需求建模导出每个建模元素的派生类。然而，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。

6.2 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的成果，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、更准确地刻画需求特征，完成更有针对性的分析和设计模型。因此，使用UML[⊖]需求建模，将从开发用例、活动图和泳道图形式的场景开始。

6.2.1 新建初始用例

"[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

Alistair Cockburn刻画了一个名为"合同行为"的用例[Coc01b]。我们在第5章讨论的"合同"定义了一个活动者[⊜]使用基于计算机的系统完成某个目标的方法。本质上用例捕获了信息的产生者、使用者和系统本身之间发生的交互。在本节，我们研究如何开发用例，这是需求建模活动的一部分[⊜]。

第5章中我们已经提到，用例从某个特定参与者的角度出发，采用简明的语言描述一个特定的使用场景。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需求建模工具那样提供价值，那么必须回答这些问题。

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适用的其他的建模方法。

编写什么？两个首要的需求工程工作是起始和导出，它们提供了开始编写用例所需要的信息。运用需求收集会议、QFD和其他需求工程机制确定利益相关者，定义问题的范围，说明整体的运行目标，建立优先级顺序，概述所有已知的功能需求，描述系统将处理的信息（对象）。

开始开发用例时，应列出特定参与者执行的功能或活动。这些可以从所需系统功能的列表通过与利益相关者交流，或通过评估活动图（作为需求建模中的一部分而开发）获得（见6.3.1节）。

SAFEHOME

开发另一个初始的用户场景

[场景] 会议室，第二次需求收集会议中。

[人物] Jamie Lazar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：现在是开始讨论SafeHome监视功能的时候了，让我们为访问监视功能开发一个用户场景。

Jamie：谁在其中扮演参与者的角色？

主持人：我想Meredith（市场营销人员）已经在该功能上进行了一些工作，你来试试这个角色吧。

Meredith：你想采用我们上次所用的同样的方法，是吗？

主持人：是的，同样的方法。

Meredith：好的，很明显，开发监视功能的理由是允许房主远距离检查宅舍、记录并回放捕获的录像……就是这样。

⊖ UML是本书通篇使用的建模符号。附录1为那些不熟悉UML基本符号的读者提供了简要指南。

⊜ 参与者不是一个确定的人员，而是人员（或设备）在特定的环境内所扮演的一个角色，参与者"呼叫系统并由系统提供一种服务"[Coc01b]。

⊜ 用例是用户接口的分析建模中特别重要的一部分，将在第11章详细讨论接口分析。

Ed：我们采用压缩的方法存储图像吗？

主持人：好问题，但是我们现在先不考虑实现的问题，Meredith，你说呢？

Meredith：好的，这样对于监视功能基本上就有两部分……第一部分是配置系统，包括布置建筑平面图—我们需要工具帮助房主做这件事，第二部分是实际的监视功能本身。因为布局是配置活动的一部分，我将重点集中在监视功能。

主持人（微笑）：你说出了我想说的话。

Meredith：哦……我希望通过电脑或通过Internet访问监视功能。我的感觉是Internet访问可能使用的频率更高一些。不管怎样，我希望能够在计算机上和控制面板上显示摄像机图像并移动某个摄像机镜头。在房屋平面设计图上可以选择指定摄像机，我希望可以有选择地记录摄像机输出和回放摄像机输出，我还希望能够使用特殊的密码阻止对某个或多个摄像机的访问。希望有支持小窗口显示形式的选项，即从所有的摄像机显示图像，并能够选择某一个放大。

Jamie：那些称做缩略视图。

Meredith：对，然后我希望从所有摄像机获得缩略视图。我也希望监视功能的接口和所有其他的SafeHome接口有相同的外观和感觉。

主持人：干得好，现在，让我们更详细地讨论这个功能……

上面SafeHome框中讨论的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）：

- 选择将要查看的摄像机。
- 提供所有摄像机的缩略视图。
- 在计算机的窗口中显示摄像机视图。
- 控制某个特定摄像机的镜头转动和缩放。
- 可选择地记录摄像机的输出。
- 回放摄像机的输出。
- 通过Internet访问摄像机监视功能。

随着和利益相关者（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5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。

为了举例说明，考虑“通过互联网访问摄像机监视设备—显示摄像机视图（access camera surveillance-display camera views，ACS-DCV）”功能，扮演参与者**房主**的利益相关者可能会编写如下说明：

用例：通过互联网访问摄像机监视设备—显示摄像机视图（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合适的浏览器软件登录SafeHome产品网站。输入我的用户身份证号和两级密码，一旦被确认，我可以访问已安装的SafeHome系统的所有功能。为取得某个摄像机视图，从显示的主功能按钮中选择“监视”，然后选择“选取摄像机”，将会显示房屋的平面设计图，再选择感兴趣的摄像机。另一种可选方法是，通过选择“所有摄像机”，可以同时从所有的摄像机查看缩略视图快照。当选择了某个摄像机时，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。如果希望切换摄像机，选择“选取摄像机”，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机，以便显示新的查看窗口。

描述性用例的一种表达形式是通过用户活动的顺序序列表现交互，每个行动由声明性的语句表示。再以ACS-DCV功能为例，我们可以写成：

用例：通过互联网访问摄像机监视设备-显示摄像机视图（ACS-DCV）

用例可以用在许多“软件”过程中，我们感兴趣的是迭代和风险驱动过程。——Geri Schneider and Jason Winters

参与者：房主

1. 房主登录SafeHome产品网站。
2. 房主输入用户身份证号。
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“选取摄像机”。
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机图标。
9. 房主选择“查看”按钮。
10. 系统显示一个由摄像机编号确定的查看窗口。
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，这个连续步骤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]。

6.2.2 细化初始用例

为全面理解用例描述功能，对交互操作给出另外的描述是非常有必要的。

因此，主场景中的每个步骤将通过如下提问得到评估[SCH98]：

在开发场景用例时，如何检查动作的可选过程？

- 在这一状态点，参与者能进行一些其他动作吗？
- 在这一状态点，参与者有没有可能遇到一些错误的条件？如果有可能，这些错误会是什么？
- 在这一状态点，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，这些行为是什么？

这些问题的答案导致创建一组次场景（Secondary Scenarios），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“选取摄像机”。
7. 系统显示房屋的平面设计图。

在这一状态点上，参与者能进行一些其他动作吗？答案是肯定的。参考无障碍解说方式，参与者可以选择同时查看所有摄像机的缩略视图。因此，一个次场景可能是“查看所有摄像机的缩略视图”。

在这一状态点，参与者有没有可能遇到一些错误的条件？作为基于计算机的系统操作，任何数量的错误条件都可能发生。在该上下文内，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。带有摄像机图标的房屋平面图可能还没有配置，这样选择“选取摄像机”就导致错误的条件：“没有为该房屋配置平面设计图”[⊖]。该错误条件就成为一个次场景。

在这一状态点，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

⊖ 在该例子中，另一个参与者，系统管理员将必须配置平面设计图、安装并初始化（如分配设备编号）所有的摄像头，并且通过系统平面设计图访问和测试每个摄像头能达到的特定作用。

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”——这个用例可以被其他用例引用。

前面段落描述的每种情景都是以客户用例的异常处理为特征的。一个异常处理描述了这样一种情景（既可能是失败条件或参与者选择了替代方案），它导致系统展示出某些不同的行为。

Cockburn[Coc01b]推荐使用“头脑风暴”推动合理完成每个用例一系列的异常处理。除了本节前面的部分提到了三个常规问题外，还应该研究下面的问题：

- 在这个用例中是否有某些具有“有效功能”的用例出现？包括引用确认功能，可能出现的出错条件。
- 在这些用例中是否有支持功能（或参与者）的应答失败？例如，一个用户动作是等待一个应答，但该功能已经应答超时了。
- 性能差的系统是否会导致无法预期或不正确的用户活动？例如，一个基于Web的接口应答太慢，导致用户在处理按钮上已经做了多重选择。这些选择队列最终不恰当地生成一个出错条件。

这里列出的扩展部分是由具有因果关系的提问和回答问题开发的，应用下面的标准使这些问题合理化[Co01b]：用例应该注明异常处理即如果软件能检测出异常所发生的条件就应该在检测出后马上处理这个条件。在某些情况下，异常处理可能拖累影响其他用例处理条件的开发。

6.2.3 编写正规的用例

在6.2.1节表述的非正规用例对于需求建模常常是够用的。但是，当一个用例包括关键活动或描述一套具有大量异常处理的复杂步骤时，就会希望采用更为正规的方法。

在SafeHome框中显示的ACS-DCV用例依照了正规用例的典型描述要点。在以下的SafeHome框中：“情境目标”确定了用例的全部范围。“前提条件”描述在用例初始化前应该知道哪些信息。“起动”确定“用例开始”的事件或条件[Coc01b]。“场景”列出由参与者和恰当的系统应答所需要的特定活动。“异常处理”用于细化初始用例时没有涉及的情境（6.2.2节）。此外还可能包含其他的标题，并给出合理的自我解释。

SAFEHOME

监视的用例模板

用例：通过互联网访问摄像头监视——显示摄像头视图（ACS-DCV）。

迭代：2，最新更改记录：V.Raman 1月14日。

主要参与者：房主。

情境目标：从任何远程地点通过互联网查看遍布房间的摄像头输出。

前提条件：必须完整配置系统；必须获得正确的用户身份证号和密码。

起动：房主在远离家的时候决定查看房屋内部。

场景：

1. 房主登录SafeHome产品网站。
2. 房主输入他或她的用户身份证号。
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编号确定的视图窗口。

11. 系统在视图窗口中以每秒一帧显示视频输出。

异常：

1. 身份证号或密码不正确或无法确认——参看用例：**“确认身份证号和密码”**。

2. 没有为该系统配置监视功能——系统显示恰当的错误消息；参看用例：**“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”——参看用例：**“查看所有摄像头的缩略视图快照”**。

4. 平面设计图不可用或未配置——显示恰当的错误消息，参看用例：**“配置平面设计图”**。

5. 遇到报警条件——参看用例：**“遇到报警条件”**。

优先级：必须在基础功能之后实现中等优先级。

何时可用：第三个增量。

使用频率：中等频度。

使用方式：通过基于个人计算机的浏览器和互联网连接到SafeHome网站。

次要参与者：系统管理员，摄像头。

次要参与者的使用方式：

1. 系统管理员：基于个人计算机的系统。

2. 摄像机：无线连接。

未解决的问题：

1. 有什么机制保护SafeHome产品的雇员在未授权的情况下能使用该功能？

2. 足够安全吗？黑客入侵该功能将使最主要的个人隐私受侵。

3. 在给定摄像机视图所要求的带宽下，可以接受通过互联网的系统响应吗？

4. 当可以使用高带宽的连接时，能开发出比每秒一帧更快的视频速度吗？

WebRef

什么时候结束编写用例？关于该话题有价值的论述，可以参阅ootips.org/use-cases-done.html。

在很多情况下，不需要创建图形化表示的用户场景。然而，尤其当场景比较复杂时图形化的表示更有助于理解。就如我们在本书前面所说提到的，UML的确提供了图形化表现用例的能力。图6-4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。仅在本节中详细讨论了ACS-DCV。

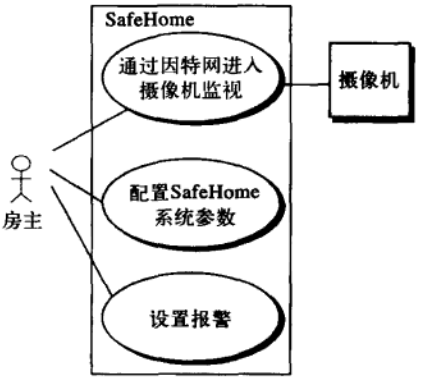


图6-4 SafeHome系统的初步用例图

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

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

6.3 补充用例的UML模型

很多基于文本的需求建模情景，甚至和用例一样简单，不能简明扼要地传递信息。在这种情况下，你应能从大量的UML图形模型中进行选择。

6.3.1 开发活动图

KEY POINT
一个UML活动图表现了实施某功能时发生的动作和判定。

UML活动图在特定场景内通过提供迭代流的图形化表示来补充用例。类似于流程图，活动图使用两端为半圆形的矩形表示一个特定的系统功能，箭头表示通过系统的流，判定菱形表示判定分支（标记从菱形发出的每个箭头），实心水平线意味着并行发生的活动。ACS-DCV用例的活动图如图6-5所示。应注意到活动图增加了额外的细节，而这些细节是用例不能直接描述的（是隐含的）。例如，用户尽可以尝试有限次数地输入**用户身份证号**（ID）和**密码**。这可以通过“**提示重新输入**”的判定菱形来体现。

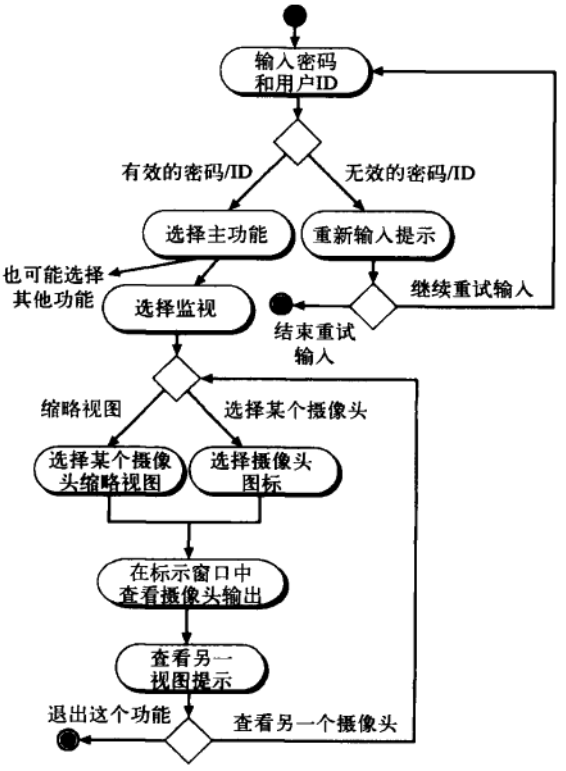


图6-5　通过互联网进入摄像机监视并显示摄像机视图功能的活动图

KEY 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。

6.3.2 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（如果在某个特定用例中涉及了多个参与者）或分析类（本章后面章节会讨论）是由活动矩形所描述的活动来负责。职责由纵向分割图的并行条表示，就像游泳池中的泳道。

三种分析类（**房主**、**摄像机**和**接口**）对于在图6-5所表示的活动图中的情景具有直接或间接的责任。参看图6-6，重新排列活动图，与某个特殊分析类相关的活动按类落入相应的泳道中。例如，**接口**类表示房主可见的用户接口。活动图标记出对接口负责的两个提示—“提示重新输入”和“提示另一视图”。这些提示以及与此相关的判定都落入了**接口**泳道。但是，从该泳道发出的箭头返回到**房主**泳道，这是因为房主的活动在房主泳道中发生。

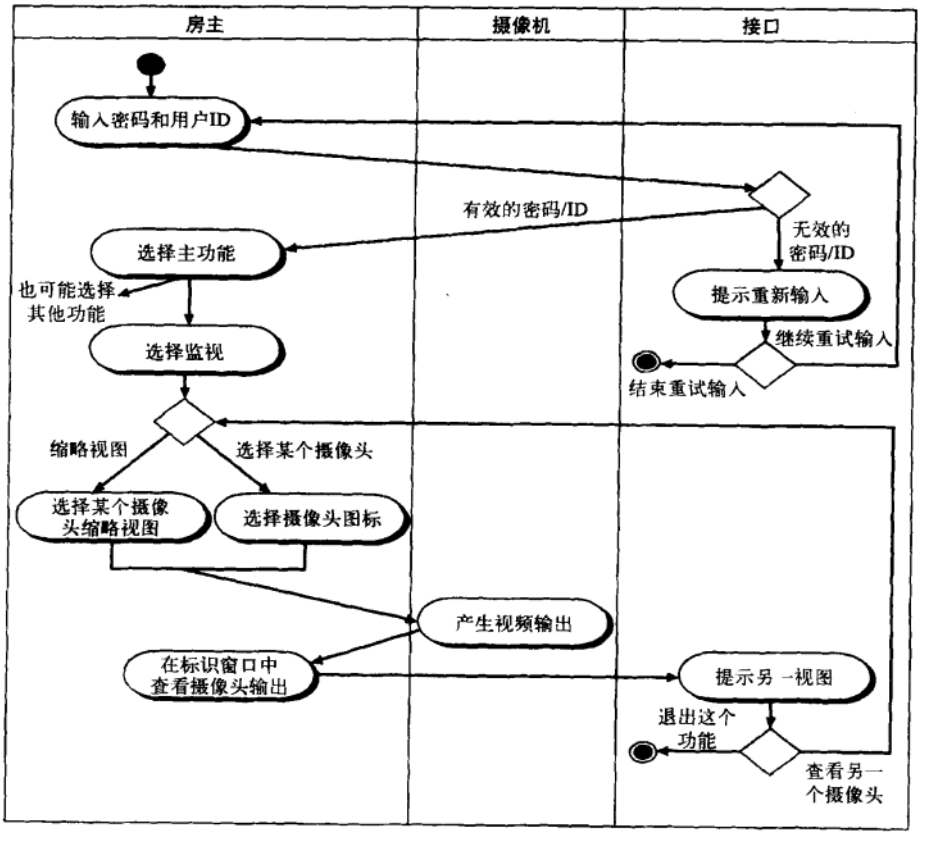


图6-6 通过互联网进入摄像机监视并显示摄像机视图功能的泳道图

“好的模型引导你的思考，而坏的模型会误导思考。”——Brian Marick

伴随着活动图和泳道图，面向过程的用例表示各种参与者行使一些特定功能（或其他处理步骤），以便满足系统需求。但是需求的过程视图仅表示系统的单一维度，在6.4节，我们将考察信息空间，并提供如何表达数据需求。

6.4 数据建模概念

WebRef

在www.data-model.org上可以找到数据建模的有用信息。

如果软件需求包括建立、扩展需求，或者具有数据库的接口，或者必须构建和操作复杂的数据结构，软件团队可以选择建立一个数据模型作为全部需求建模的一部分。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，数据对象之间的关联以及其他与此相关的信息。实体关系图（ERD）描述了这些问题并提供了在一个应用项目中输入、存储、转换和产生的所有数据对象。

6.4.1 数据对象

数据对象如何在一个应用系统的环境内表现自己？

数据对象是必须由软件理解的复合信息表示。复合信息是指具有若干不同的特征或属性的事物。因此，“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。

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

KEY POINT
数据对象是由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——在数据对象中没有操作数据的引用[⊖]。因此，数据可以表示为如图6-7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**生产商**、**车型**、**标识号**、**车体类型**、**颜色**和**车主**定义的。该表的主体表示了数据对象的特定实例。例如，Chevy Corvette牌的车是数据对象**car**的一个实例。

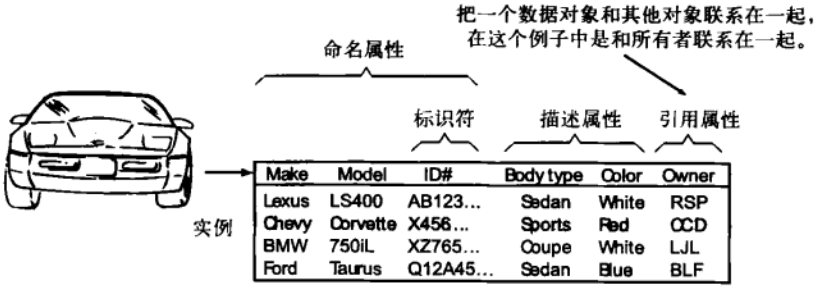


图6-7　数据对象的表格表示

6.4.2 数据属性

KEY POINT
属性命名某数据对象，描述其特征，并在某些情况下引用另一个对象。

数据属性定义了数据对象的性质，可以具有三种不同的特性之一。它们可以用来：(1) 为数据对象的实例命名；(2) 描述这个实例；(3) 建立对另一个表中的另一个实例的引用。另外，必须把一个或多个属性定义为标识符——也就是说，当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。在数据对象汽车的例子中，标识号可以作为一个合理的标识符。

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。汽车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，汽车公司需要制造中的控制软件。在后一种情况下，汽车的属性可能也包括标识号、车体类型和颜色，但为了使汽车在制造中的控制环境下成为一个有用的对象，必须增加许多其他的属性（如内部代码、驱动系统类型、车内包装设计师、传动类型）。

WebRef
一种称为“规范化”的概念对于偏好全部使用数据建模的人非常重要，可以在www.datamodel.org中找到有用的信息。

⊖　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（附录2）。

INFO

数据对象和面向对象类——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。

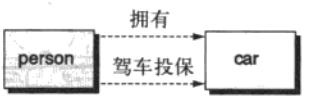
数据对象定义了一个复合的数据项，也就是说合并一组独立的数据项（属性）并为数据项集合命名（数据对象名）。

一个面向对象类封装了数据属性，但对这些属性所定义数据的操作（方法）进行合并。另外，类的定义暗示了一个全面的基础设施，该基础设施是面向对象软件工程方法的一部分。类之间通过消息通信，它可以按层次关系组织，并为某个类的一个实例这样的对象提供继承特性。

6.4.3 关系

数据对象可以以多种不同的方式与另一个数据对象连接。考虑一下两个数据对象：person和car。这些对象可以使用图6-8a所示的简单标记表示。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”来定义相互的关系，例如：

a) 数据对象之间的基本连接



b) 数据对象之间的关系

图6-8 数据对象间的关联关系

KEY POINT 关系指明数据对象相互“链接”的方式。

- 拥有车的人。
- 汽车驾车投保人。

关系“拥有”和“驾车投保”定义了person和car之间的相关连接。图6-8b以图形方式说明了这些对象/关系对，图6-8b标注的箭头提供了关联方向的重要信息，这一方向信息通常可以减少歧义或误解。

INFO

实体-关系图

对象/关系对是数据模型的基石。可以使用实体-关系图（ERD）[⊜]图形化地表示这些对象/关系对。ERD最初是由Peter Chen[Che77]为关系数据库系统设计提出的，并由其他人进行了扩展。ERD标识了一组基本元素：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，用带标记的矩形表示数据对象，用带标记的线连接对象表示关系。在ERD的某些变形中，连接线包含一个带有关系标记的菱形。使用各种指示基数和模态的特殊符号来建立[⊟]数据对象和关系的连接。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[Hob06]或[Sim05]。

⊖ 如果读者不熟悉面向对象的内容和专有名词，应该参考附录2中介绍的简要指南。

⊜ 尽管ERD仍然在某些数据库设计应用中使用，UML符号（附录1）在现在的数据设计中也很常用。

⊟ 对象关系对的基数（Cardinality）特指“一个[对象]出现的次数与另一个[对象]出现次数相关联”[Til93]。对于关系的模态（Modality）是指：如果没有明确发生关系的需要或者关系是可选的，那么关系的模态是0；如果关系必须出现一次，那么模态是1。

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。该类工具往往用来作数据库设计，还可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：[⊖]

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。

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。

6.5 基于类的建模

基于类建模表示了系统操作的对象、应用于对象间能有效控制的操作(也称为方法或服务)、这些对象间（某种层级）的关系以及已定义类之间的协作。基于类的分析模型的元素包括类和对象、属性、操作、类的职责协作者(CRC)模型、协作图和包。下面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。

6.5.1 识别分析类

"真正困难的问题是首先发现什么才是正确的对象（类）。"
——Carl Argila

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）就没有那么容易了。

通过检查需求模型开发的使用场景，对系统开发的用例进行"语法解析"[Abb83]，我们可以开始进行类的识别。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，标注出同义词。如果要求某个类（名词）实现一个解决方案，那么这个类就是解决方案空间的一部分；否则，如果只要求某个类描述一个解决方案，那么这个类就是问题空间的一部分。

分析类如何把自己表现为解决方案空间的元素？

不过一旦分离出所有的名词，我们该寻找什么？分析类表现为如下方式之一：

- 外部实体（例如，其他系统、设备、人员），产生或使用基于计算机系统的信息。
- 事物（例如，报告、显示、字母、信号），问题信息域的一部分。
- 偶发事件或事件（例如，所有权转移或完成机器人的一组移动动作），在系统操作环境内发生。

⊖ 这里给出的工具只是此类工具的例子，并不代表本书支持采用这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

- 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），和某个应用系统相关。
- 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[⊖]。例如，Budd[Bud96]建议了一种类的分类法，包括数据产生者（源点）和数据使用者（汇点）、数据管理者、查看或观察者类以及帮助类。

还需要特别注意的是：什么不能是类或对象。通常，决不应该使用“命令过程式的名称”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对象的一个操作，很可能将翻转定义为对于对象Image的一个操作，但是不可能定义单独的类来暗示“图像翻转”。如Cashman[Cas89]所言：“面向对象的目的是封装，但仍保持独立的数据以及对数据的操作。”

为了说明在建模的早期阶段如何定义分析类，考虑对SafeHome安全功能的“处理叙述”[⊜]进行语法分析（对第一次出现的名词加下划线，第一次出现的动词采用斜体）。

虽然语法解析不能保证万无一失，但如果你正在定义数据对象及其操作的转变，语法解析会让你飞跃上一个非常出色的起始点。

SafeHome安全功能可以帮助房主在安装时配置安全系统，监控所有链接到安全系统的传感器，通过互联网、计算机或控制面板和房主交互信息。

在安装过程中，用SafeHome个人计算机设计和配置系统。为每个传感器分配一个编号和类型，用主密码控制启动和关闭系统，而且当传感器事件发生时会拨打输入的电话号码。

当识别出一个传感器事件时，软件激活装在系统上可发声的警报，由房东在系统配置活动中指定的延迟时间后，软件拨打监测服务的电话号码并提供位置信息，报告检测到的事件性质。电话号码将每隔20秒重拨一次，直至达到电话接通。

房主通过控制面板、个人计算机或浏览器这些统称为接口来接收安全信息。接口在控制面板、计算机或浏览器窗口中显示提示信息和系统状态信息。房主采用如下形式进行交互活动……

抽取这些名词，可以获得如下表所示的一些潜在类：

潜在类	一般分类
房主	角色或外部实体
传感器	外部实体
控制面板	外部实体
安装	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

⊖ 另一个重要的分类是指定义实体、边界和控制类，在6.5.4节中讨论。

⊜ “处理叙述”类似于用例的风格，但目标稍有不同。“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的场景。但是要注意很重要的一点，在需求收集（导出）部分也会使用语法解析开发每个用例。

这个表应不断完善，直到已经考虑到了处理叙述中所有的名词。注意，我们称列表中的每一输入项为潜在的对象，在进行最终决定之前还必须对它们每一项深思熟虑。

Coad和Yourdon[Coa91]建议了6个选择特征，在分析模型中分析师考虑每个潜在类是否应该使用如下这些特征：

如何确定某个潜在类是否应该真的成为一个分析类？

1. 保留信息。只有记录潜在类的信息才能保证系统正常工作，在这种分析过程中的潜在类是有用的。

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

3. 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，最好把它作为另一个类的某个属性。

“阶级斗争，一些阶级胜利了，一些阶级消灭了……”——毛泽东

4. 公共属性。可以为潜在类定义一组属性，这些属性适用于类的所有实例。

5. 公共操作。可以为潜在类定义一组操作，这些操作适用于类的所有实例。

6. 必要需求。在问题空间中出现的外部实体，和任何系统解决方案运行时所必需的生产或消费信息，几乎都被定义为需求模型中的类。

考虑包含在需求模型中的合法类，潜在类应全部（或几乎全部）满足这些特征。判定潜在类是否包含在分析模型中多少有点主观，而且后面的评估可能会舍弃或恢复某个类。然而，基于类建模的首要步骤就是定义类，因此必须进行决策（即使是主观的）。以此为指导，根据上述选择特征进行了筛选，分析师列出SafeHome潜在类，如下表所示：

潜在类	适用的特征编号
房主	拒绝：6适用但是1、2不符合
传感器	接受：所有都适用
控制面板	接受：所有都适用
安装	拒绝
系统（别名安全系统）	接受：所有都适用
编号，类型	拒绝：3不符合，这是传感器的属性
主密码	拒绝：3不符合
电话号码	拒绝：3不符合
传感器事件	接受：所有都适用
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）上表并不全面，必须添加其他类以使模型更完整；（2）某些被拒绝的潜在类将成为被接受类的属性（例如，*编号*和*类型*是Sensor的属性，*主密码*和*电话号码*可能成为System的属性）；（3）对问题的不同陈述可能导致作出“接受或拒绝”不同的决定。（例如，如果每个房主都有个人密码或通过声音确认，Homeowner类有可能接受并满足特征1和2）。

6.5.2 描述属性

属性描述了已经选择包含在需求模型中的类。实质上，属性是定义类以澄清类在问题空间的环境下意味着什么。例如，如果我们建立一个系统跟踪职业棒球手的统计信息，类Player的属性与用于职业棒球手的养老系统中的属性是截然不同的。在前者，属性是与*名字、位置、平均击球次数、担任防守百分比，从业年限、比赛次数*等相关的。后者，某些属性具有相同的含义，另外一些属性将被替换（或引起争议），例如，*平均工资、充分享受优惠权后的信用、所*

选的养老计划、邮件地址等。

KEY POINT 属性是在问题的环境下完整定义类的数据对象集合。

为了给分析类开发一个有意义的属性集合，软件工程师应该研究用例并选择那些合理“属于”类的“事物”。此外，每个类都应回答如下问题：什么数据项（组合项和/或者基本项）能够在当前问题环境内完整地定义这个类？

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

识别信息=系统编号+确认电话号码+系统状态

报警应答信息=延迟时间+电话号码

激活或者关闭信息=主密码+允许重试次数+临时密码

等式右边的每一个数据项可以进一步地精化到基础级，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6-9中的阴影部分）。

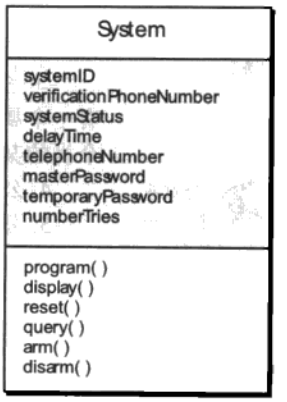


图6-9 System类的类图

传感器是整个SafeHome系统的一部分，但是并没有列出如图6-9中的数据项或属性。已经定义**Sensor**为类，多个**Sensor**对象将和**System**类关联。通常，如果有超过一个项和某个类相关联，就应避免把这个项定义为属性。

ADVICE 当分析类定义操作时，集中于面向问题的行为而不是实现所要求的行为。

6.5.3 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4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；（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（6.5.5节）上的操作实现。因此，操作必须“理解”为类的属性和相关属性的性质。

在第一次迭代要导出一组分析类的操作时，可以再次研究处理叙述（或用例）并合理地选择属于该类的操作。为了实现这个目标，可以再次研究语法解析并分离动词。这些动词中的一部分将是合法的操作并能够很容易地连接到某个特定类。例如，从本章前面提出的SafeHome处理叙述中可以看到，“为传感器分配编号和类型”、“主密码用于激活和解除系统”，这些短语表明了一些事物：

- assign()操作和**Sensor**类相关联。
- program()操作应用于**System**类。
- arm()和disarm()应用于**System**类。

再进一步的研究，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()隐含着电话号码、配置系统特性（如创建传感器表、输入报警特征值）和输入密码。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，开始进行需求建模。

[人物] Jamie、Vinod和Ed，SafeHome软件工程团队的成员。

[对话]

[Ed已经从ACS-DCV（本章前面SafeHome框中已有介绍）的用例模板中做了提取类方面

的工作，并向他的同事展示了已经提取的类。]

Ed：那么当房主希望选择一个摄像机的时候，他或她可能从一个平面设计图中进行选择。我已经定义了一个FloorPlan类，这个图在这里。

（他们查看图6-10。）

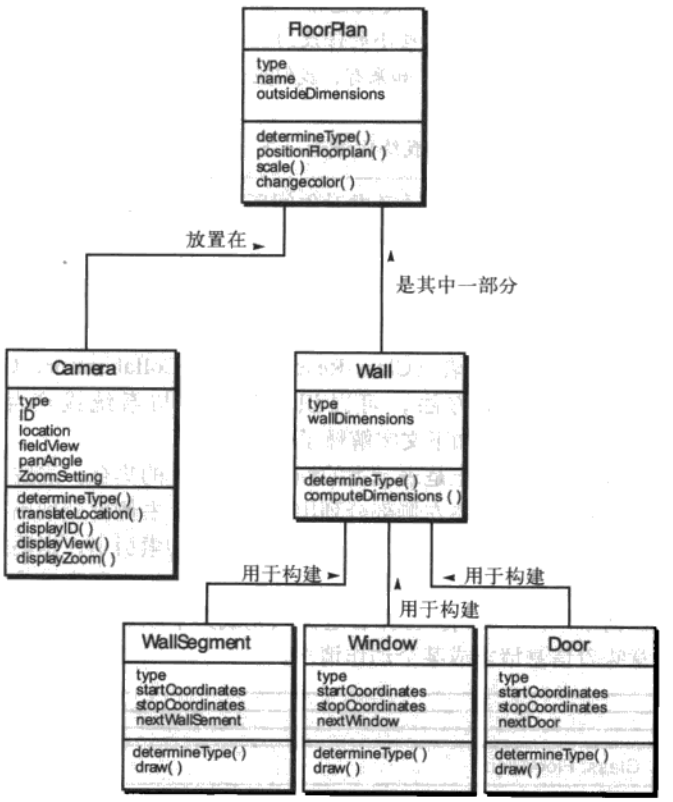


图6-10　FloorPlan类的类图

Jamie：那么FloorPlan这个类把墙、门、窗和摄像机都组织在一起。这就是那些标记线的意义，是吗？

Ed：是的，它们被称作“关联”，一个类根据我在图中所表示的关联关系和另一个类相关联（在6.5.5节中讨论关联）。

Vinod：那么实际的平面设计图是由墙构成的，并包含摄像机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：平面设计图不知道，但是其他类知道。例如察看属性WallSegment，该属性用于构建墙，墙段（Wallsegment）具有起点坐标和终点坐标，其他由draw()操作完成。

Jamie：这些也适用于门窗。看起来摄像机有一些额外的属性。

Ed：是的，我要求它们提供转动信息和缩放信息。

Vinod：我有个问题，为什么摄像机有ID编号而其他的没有呢？我注意到有个属性叫nextWall。WallSegment如何知道什么是下一堵墙？

Ed：好问题，但正如他们所说，那是由设计决定的，所以我将推迟这个问题直到……

Jamie：让我休息一下……我打赌你已经想出办法了。

Ed（羞怯地笑了笑）：确实，当我们得到设计时我要采用列表结构来建模。如果你坚信分析和设计是分离的，那么我安排的详细程度等级就有疑问了。

Jamie：对我而言看上去非常好。只是我还有一些问题。

（Jamie问了一些问题，因此做了一些小的修改。）

Vinod：你有每个对象的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。

另外，对于语法分析，分析师能通过考虑对象间所发生的通信获得对其他的操作更为深入的了解。对象通过传递信息与另一个对象通信。在继续对操作进行说明之前，探测到了更详实的信息。

6.5.4 类－职责－协作者建模

使用CRC卡的一个目的是早些舍弃、频繁舍弃，并且低成本舍弃。事实上抽出一叠卡片要比改编大量源代码要容易得多。"
——C. Horstman

类-职责-协作者（Class-Responsibility-Collaborator，CRC）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。这些卡片分三部分，顶部写类名，卡片主体左侧部分列出类的职责，右侧部分列出类的协作者。

事实上，CRC模型可以使用真的或虚拟的索引卡，意图是开发有组织表示的类。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]。协作者是提供完成某个职责所需要信息的类。通常，协作意味着信息请求或某个动作请求。

图6-11 CRC模型索引卡

FloorPlan类的一个简单CRC索引卡如图6-11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。

类。在本章前面已经说明识别类和对象的基本原则。6.5.1节所说的类的分类可以通过如下

WebRef
关于这些类的类型的精彩讨论，可以参阅www.theumlcafe.com/a0079.htm。

分类方式进行扩展：

- 实体类，也称作模型或业务类，是从问题说明中直接提取出来的（例如FloorPlan和Sensor）。这些类一般代表保存在数据库中和贯穿应用程序（除非被明确删除）的事物。
- 边界类用于创建用户可见的和在使用软件时交互的接口（如交互屏幕或打印的报表）。实体类包含对用户来说很重要的信息，但是并不显示这些信息。设计边界类的职责是管理实体对象对用户的表示方式。例如，一个称做Camera Window的边界类负责显示SafeHome系统监视摄像机的输出。
- 控制类自始至终管理“工作单元”[UML03]。也就是说，设计控制类可以管理（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）对象集合间的复杂通信；（4）对象间或用户和应用系统间交换数据的确认。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、损坏的和丢失的。”
——Russsell Baker

为类分配职责时可以采用什么指导原则？

职责。在6.5.2节和6.5.3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5个指导原则：

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

如果智能系统更平均地分布在应用系统的所有类中，每个对象只了解和执行一些事情（通常是适度集中），并提高系统的内聚性[⊖]，这将提高软件的可维护性并减少变更的副作用影响。

为了确定是否恰当地分布智能系统，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。如果有这种情况就表明智能太集中[⊜]。此外，每个类的职责应表现在同一抽象层上。例如在聚合类checkingAccount操作列表中评审人员注意到两项职责：账户余额和已结算的支票。第一个操作的职责意味着复杂的算术和逻辑过程；第二个操作的职责是指简单的办事员活动。既然这两个操作不是相同的抽象级别，已结算的支票应该被放在CheckEntry的职责中，这是由聚合类CheckingAccount压缩得到的一个类。

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

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

4. **某个事物的信息应局限于一个类中而不要分布在多个类中**。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。通常这个职责不应由多个类分担。如果信息是分布的，软件将变得更加难以维护，测试也会面临更多挑战。

5. **适合时，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间展示同样的行为。例如，考虑一个视频游戏，必须显示如下类：**Player**、**PlayerBody**、**PlayerArms**、**PlayerLegs**和**PlayerHead**。每个类都有各自的属性（例如，**position**、**orientation**、**color**和

⊖ 内聚性将在第8章设计概念中讨论。

⊜ 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。

speed）并且所有这些属性都必须在用户操纵游戏杆时更新和显示。因此，每个对象必须共享职责update()和display()。Player知道在什么时候发生了某些变化并且需要update()操作。它和其他对象协作获得新的位置或方向，但是每个对象控制各自的显示。

协作。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；或者（2）一个类可以和其他类协作。Wirfs-Brock和她的同事[Wir90]如下定义协作：

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

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

例如，考虑SafeHome的安全功能。作为活动流程的一部分，**ControlPanel**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。如果传感器是开启的，**ControlPanel**必须设置属性**status**为“未准备好”。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。

为帮助识别协作者，分析师可以检查类之间三种不同的通用关系[Wir90]：（1）is-part-of（是……一部分）关系；（2）has-knowledge-of（有……的知识）关系；（3）depends-upon（依赖……）关系。在下面的段落中将简单地分别说明这三种通用关系。

属于某个聚合类一部分的所有类可通过is-part-of关系和聚合类连接。考虑前面提到的视频游戏中所定义的类，**PlayerBody**是**Player**的一部分，**PlayerArms**、**PlayerLegs**和**PlayerHead**也类似。在UML中，使用如图6-12所示的聚合方式表示这些关系。

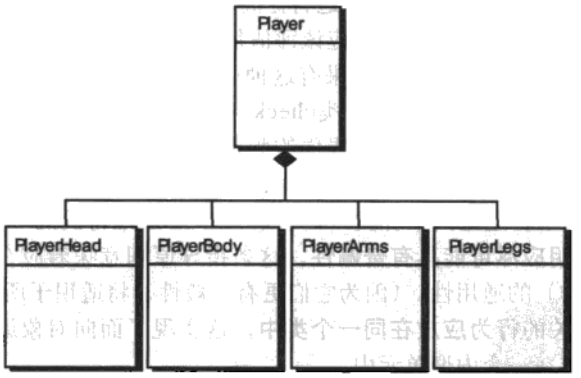


图6-12 复合聚合类

当一个类必须从另一个类中获取信息时，就建立了has-knowledge-of关系。前面所说的determine-sensor-status()职责就是has-knowledge-of关系的一个例子。

depends-upon关系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**（除非视频游戏特别异常），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，即**PlayerBody**需要**Player**。因此，**PlayerHead**依赖**PlayerBody**。

所有情况下，把协作类的名称记录在CRC模型索引卡上，紧靠在协作的职责旁边。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6-11）。

当开发出一个完整的CRC模型时，利益相关者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。拆分协作卡片（也就是说每个评审员不得有两张存在协作关系的卡片）。

2. 分类管理所有的用例场景（以及相关的用例图）。

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

房主观察SafeHome控制面板以确定系统是否已经准备接收输入。如果系统没有准备好，房主必须手工关闭窗户（门）以便指示器呈现就绪状态。（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。）

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

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

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

该过程持续进行直到用例编写结束。当评审完所有的用例，将继续进行需求建模。

SAFEHOME

CRC模型

[场景] Ed的办公室，刚开始需求建模。

[人物] Vinod和Ed，SafeHome软件工程团队成员。

[对话]

（Vinod已经决定通过一个例子向Ed展示如何开发CRC卡。）

Vinod：*在你着手于监视功能而Jamie忙着安全功能的时候，我在准备住宅管理功能。*

Ed：*情况怎样？市场营销人员的想法总是在变化。*

Vinod：*这是整个功能的第一版用例……我们已经改进了一点，它应该能提供一个整体视图。*

用例：SafeHome住宅管理功能。

说明：*我们希望通过个人计算机上的住宅管理接口或互联网连接，来控制有无线接口控制器的电子设备。系统应该让我能够打开或关闭指定的灯，控制连接到无线接口的设备，设置取暖和空调系统达到预定的温度。为此，我希望从房屋平面图上选择设备。每个设备必须在平面图上标识出来。作为可选的特性，我希望控制所有的视听设备—音响设备、电视、DVD、数字录音机等。*

通过一个选择就能够针对各种情况设置整个房屋，第一个选择项是“在家”，第二个是“不在家”，第三个是“彻夜不归”，第四个是“长期外出”。所有这些情况都适用于所有设备的设置。在彻夜不归和长期外出时，系统将以随机的间隔时间开灯和关灯（以造成有人在家的错觉），并控制取暖和空调系统。我应能够通过有适当密码保护的互联网撤销这些设置……

Ed：*那些负责硬件的伙计已经设计出所有的无线接口了吗？*

Vinod（微笑）：*他们正在忙这个，据说没有问题。不管怎样，我从住宅管理中提取了一批类，我们可以用一个做例子。就以**HomeManagementInterface**类为例吧！*

Ed：好……那么职责是什么……类及协作者们的属性和操作是那些职责所指向的类。

Vinod：我想你还不了解CRC。

Ed：可能有点，但还是继续吧。

Vinod：这就是我给出的**HomeManagementInterface**的类定义。

属性：

optionsPanel——在按钮上提供信息，用户可以使用这些信息选择功能。

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、家用电器、HVAC等。

devicePanels——模拟家用电器或设备控制面板，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()操作时，将和**FloorPlan**对象协作，类似我们为监视开发的对象。等一下，我这里有它的说明（他们查看图6-10）。

Vinod：确实如此。如果我们希望评审整个类模型，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。

6.5.5 关联和依赖

KEY POINT
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联系着，就如同彼此相互联系的两个数据对象（6.4.3节）。在UML中，这些联系被称作关联（accociations）。参考图6-10，通过识别**FloorPlan**和另外两个类**Camera**和**Wall**之间的一组关联确定**FloorPlan**类。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。参考图6-10，一个**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6-13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[⊖]。

⊖ 其他的多样性关联（一对一、一对多、多对多、一对某指定上下限的范围，以及其他）可以标识为关联的一部分。

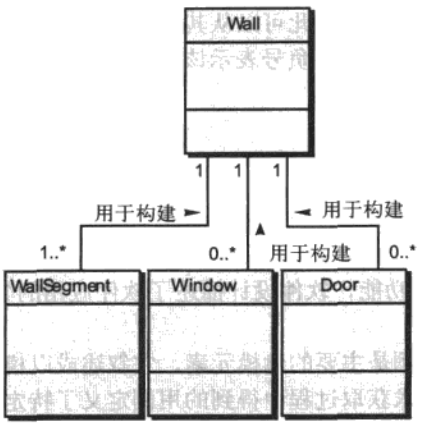


图6-13　多样性

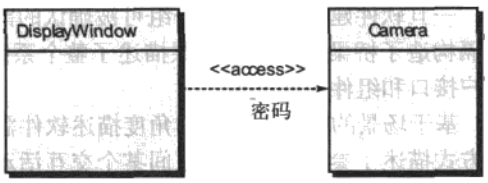


图6-14　依赖

什么是构造型？

在很多事例中，两个分析类之间存在客户-服务器关系。这种情况下，客户类以某种方式依赖于服务器类并且建立了依赖关系。依赖是由一个构造型（stereotype）定义的。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》）。

下面举例说明SafeHome监视系统中的简单依赖关系。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。这两个对象之间的关系不是简单的关联，而是存在着依赖关系。在监视用例中（没有列出来），建模者知道必须提供特定的密码才能查看指定摄像机的位置。其实现的一种方法是让**Camera**请求密码，然后在获得**DisplayWindow**的允许后显示视频。这可以由图6-14表示，其中《access》意味着通过特定的密码控制使用摄像机的输出。

6.5.6　分析包

KEY POINT
分析包用来集合一组相关的类。

分析建模的一部分重要工作是分类，也就是将分析模型的各种元素（如用例、分析类）以一种方式分类，分组打包后称其为分析包，并取一个有代表性的名。

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。有一些类关注于游戏环境，即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人物，说明他们的肢体特点、动作和局限。也可能定义了（前面提到的）**Player**、**Protagonist**、**Antagonist**、**SupportingRoles**类。还需要一些类用来说明游戏的规则即游戏者如何穿越环境。例如这里可以选**RulesOfMovement**类和**ConstraintsOnAction**类。还可能存在很多其他类，可以由图6-15中的分析包表示这些类。

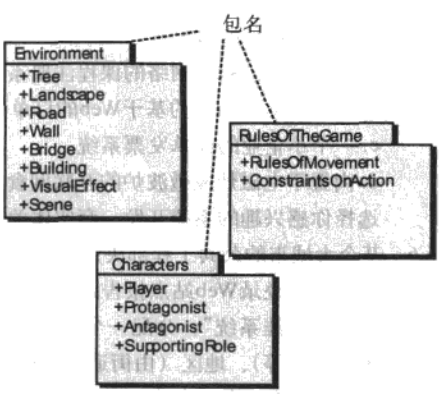


图6-15　包

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。尽管目前的图中没有显示，但包中的任何元素之前可以添加其他符号。负号表示该元素对其他包是隐藏的，#号只能由指定包中的类访问表示该元素。

6.6 小结

需求建模的目标是创建各种表现形式，用其描述什么是客户需求，建立生成软件设计的基础，一旦软件建立就能定义一组可被确认的需求。需求模型为系统级表示层和软件设计之间的间隔构造了桥梁。系统表示层描述了整个系统和业务功能，软件设计描述了软件应用的架构、用户接口和组件级的结构。

基于场景的模型从用户的角度描述软件需求。用例是主要的建模元素，它叙述或以模板驱动方式描述了参与者和软件之间某个交互活动。在需求获取过程中得到的用例定义了特定功能或交互活动的关键步骤。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，即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。泳道图显示了如何给不同的参与者或类分配处理流。

数据建模常用于描述了软件构建或操作的信息空间。数据建模由所代表的数据对象开始，这些数据对象必须由软件所理解的信息组成。每个数据对象的属性得到识别，数据对象间的关系得到描述。

为了识别分析类，基于类的建模使用从基于场景和面向流的建模元素中导出信息。可以使用语法分析从文本叙述中提取候选类、属性和操作，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建模方法定义类之间的层次、关系、关联、聚合和依赖。使用一种分析包方式进行分类和分组，从而在某种意义上为大型系统提供了更好的管理。

习题与思考题

6.1 有没有可能在分析模创建后立即开始编码？解释你的答案，然后说服反方。

6.2 一个单凭经验的分析原则是：模型“应该关注在问题域或业务域中可见的需求”。在这些域中哪些类型的需求是不可见的？提供一些例子。

6.3 域分析的目的是什么？如何将域分析与需求模式概念相联系？

6.4 有没有可能不完成如图6-3所示的4种元素就开发出一个有效的分析模型？解释一下。

6.5 构建如下系统中的一个：

a. 你所在大学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. 内置于电磁灶或微波炉的互联网食谱

选择你感兴趣的系统开发一套实体关系图并说明数据对象、关系和属性。

6.6 某个大城市的公共工程部决定开发基于Web的跟踪和修补路面坑洼系统（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。当上报路面坑洼时，它被记入“跟踪和修补路面坑洼系统”，分配一个标识号，保存如下信息：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联，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、修复耗时、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

（从修复耗时、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报告信息，并包含公民的姓名、地址、电话号码、损失类型、损失金额。PHTRS是基于在线系统，可交互地进行所有的查询。

a. 为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。

b.为PHTRS系统开发一个类模型。

6.7　为6.5.4节中非正式描述的SafeHome住宅管理系统编写一个基于模板的用例。

6.8　为6.5题所选的产品或系统开发一组完整的CRC模型索引卡。

6.9　指导你的同事一起评审CRC索引卡，评审结果中增加了多少类、职责和协作者？

6.10　什么是分析包？如何使用分析包？

推荐读物与阅读信息

用例是为所有需求建模方法服务的基础。有大量详细讨论这一主题的书，包括Rosenberga 和 Stephens（《Use Case Driven Object Modeling with UML:Theory and Practice》，Apress，2007）、Denny（《Succesding with Use Cases:Working Smart to Deliver Quality》、Addison-Wesley, 2005), Alexsander 和 Maiden（《Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle》，Wiley, 2004）、Bittner和Spence （《Use Case Modeling》，Addison-Wesley，2002）、Cockburn[Coc01b]，以及其他参考资料请参考第5章和第6章。

数据建模为检验信息空间描述了非常有用的方法，相关书目有：Hoberman [Hob06]和Simsion 和 Witt[Sim05]提出了更合理全面的处理。另外，Allen和Terry（《Beginning Relational Data Modeling》，2nd ed., Apress,2005）、Allen（《Data Modeling for Everyone》，Wrox Press，2002）、Teorey和他的同事（《Database Modeling and Design: Logical Design》，4th ed., Morgan Kaufmann, 2005）、Carlis 和 Maguire（《Mastering Data Modeling》，Addison-Wesley，2000) 给出了生成行业质量数据模型的详细指南。另一本有趣的书是Hay（《Data Modeling Patterns》，Dorset House, 1995)。描述了在许多不同业务领域使用典型数据模型的模式。

UML建模技术可以应用于分析和设计，讨论这方面的书目有O'Docherty（《Object-Oriented Analysis and Design: Understanding System Development with UML 2.0》,Wiley, 2005）、Arlow 和Neustadt（《UML 2 and the Unified Process》，2nd ed., Addison-Wesley,2005）、Roques（《UML in Practice》，Wiley，2004）、Dennis 和他的同事（《Systems Analysis and Design with UML 2.0》，Wiley，2004）、Larman（《Applying UML and Patterns》，2nd ed., Prentice-Hall, 2001）、Rosenberg 和Scott（《Use Case Driven Object Modeling with UML》，Addison-Wesley, 1999)。

互联网上有很多关于需求建模的信息。可以参考SEPA网站上有关分析建模的更新列表：www.mhhe.com/engcs/compsci/pressman/professional /olc/ser.htm。

第7章

Software Engineering：A Practitioner's Approach (Seventh Edition)

需求建模：流程、行为、模式和Web应用

要点浏览

概念：需求建模有很多不同的维度。在这一章将学习面向流程建模、行为建模，以及当开发Web应用时要考虑的特定需求分析问题。这里的每种模型都是补充第6章讨论的用例、数据模型和基于类的模型。

人员：一位软件工程师（有时称为分析师）从各个利益相关者中提取需求进行建模。

重要性：软件工程师洞悉到软件需求的增长与来自不同需求模型维度的增长成正比。尽管可能没有时间，没有资源，没有趋于采用本章和第6章所建议的任何一种表示方法进行开发，但是软件工程师可以认知每种不同的建模方式，这会给他提供一种审视问题的不同方法。继而他（和其他利益相关者）将能够更好地进入状态，才有可能更好地界定是否已把必须完成的需求真正描述清楚。

步骤：面向流程的建模提供了一种如何利用处理功能转换数据对象的表示方法。行为建模描述了系统及其类的状态，以及事件对这些类的影响。基于模式的建模利用现有领域的知识使得需求分析更为容易。Web应用的需求模型特别适用于表述内容、交互操作、功能和配置相关的需求。

工作产品：为需求建模选择了大量基于文本和图形的格式。每种表示方法都提供了一种或多种模型元素。

质量保证措施：必须评审需求建模产品的正确性、完备性和一致性。必须反映所有利益相关者的要求，建立得以导出设计的根基。

关键概念
分析模式
行为模型
配置模型
内容模型
控制流模型
数据流模型
功能模型
交互模型
导航建模
处理规格说明
顺序图
Web应用

在第6章讨论了用例、数据建模和基于类模型后，很自然会问“这些需求建模表示方法就足够了吗？”

唯一合理的回答是：“要看情况而定”。

对于某种类型软件，用例可能是唯一可行的需求建模表示方法。而其他类型的软件，则需要选择面向对象的方法开发基于类的模型。但在另外一些情形下，复杂应用需求可能必须做个检测，查看当数据对象在系统中移动时是如何转换的；查看作为外部事件的后果一个应用系统是如何工作的；查看现存知识领域能否解决当前问题；或者在基于Web系统和应用中，如何将内容和功能融合在一起，并提供给最终用户成功导向一个Web应用的能力，以便达到适用目标。

7.1 需求建模策略

一种考虑数据和处理的分析建模方法被称作结构化分析，其中数据可作为独立实体转换。数据对象模型的方式定义了它们的属性和关系。操作数据对象的建模过程表明当数据对象通过系统时如何转换数据。分析建模的第二种方法称做面向对象分析，这种方法关注于定义类以及影响客户需求的类之间的协作方式。

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

并排斥所有其他的表示方法。问题不是哪一种方法最好，而是怎么组合这些表示方法才能够为利益相关者提供最好的软件需求模型和通往软件设计最为有效的桥梁。

7.2 面向数建模

ADVICE
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达模式。但如果有帮助，就应该使用DFD。

对于一些软件工程师而言，虽然认为面向流程建模的数据是过时的技术，但它将继续成为当前使用最广泛的需求分析表达方式之一[⊖]。尽管数据流图（Data Flow 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程的补充认识。

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，经由处理元素变换，最后以结果数据对象的形式流出软件。带标记的箭头表示数据对象，圆圈（也称作泡泡）表示转换。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）表示整个系统，随后的数据流图改进环境图，在每个后续层提供更多的细节。

"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

7.2.1 创建数据流模型

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。当把DFD逐步细化时，分析师同时也就完成了系统功能分解。与此同时，当数据在应用系统中的多个处理间流动时，DFD的精炼结果导致了相应的数据精化。

KEY POINT
当精化每一层DFD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该层的输入和输出相同。

导出数据流图时有一些有用而简单的指导原则：（1）第0层的数据流图应将软件或系统描述为一个泡泡；（2）应仔细标记主要的输入和输出；（3）通过把选定的处理、数据对象和数据存储分离为下一层表示而开始精化过程；（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[⊜]；（6）一次精化一个泡泡。可能会有一种趋势使数据流图过于复杂。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。图7-1显示了安全功能的0层DFD，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、所有的激活或解除命令、所有各式各样的交互活动以及所有限定或扩展某命令的输入数据。

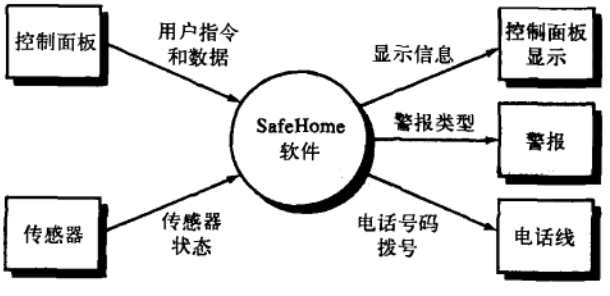


图7-1　SafeHome安全功能的环境层DFD

⊖ 在结构化分析中，数据建模是核心建模活动。

⊜ 也就是说，流入系统或流入某一层变换的数据对象必须与流入更细化层的变换具有相同的数据对象（或其组成部分）。

现在必须把第0层的DFD扩展到第1层数据流模型。但我们应如何进行呢？根据第6章建议的方法，应该对描述环境层泡泡的用例叙述采用“语法解析”[Abb83]，即将第一次需求收集会议获得的SafeHome处理叙述中的所有名词（和名词短语）与动词（和动词短语）分离开来。回想一下在6.5.1节表示已经被解析的处理描述。

SafeHome安全功能能够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，通过互联网、个人计算机或控制面板和房主进行交互活动。

ADVICE
语法解析并非绝对可靠，但是如果你正在努力的定义数据对象和转换，它将提供极好的启始准备。

在安装中，SafeHome使用个人计算机设计和配置系统，为每个传感器分配一个编号和类型，用主密码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，报告所检测到的事件性质，电话号码将每20秒重拨一次，直至取得电话接通。

房主通过控制面板、个人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、个人计算机或浏览器窗口中显示提示信息和系统状态信息。房主的交互活动采用如下形式……

ADVICE
要确定对解析的处理叙述是在完全相同的抽象层面上编写的。

根据语法解析，动词是SafeHome处理，在后续的DFD中用泡泡表示，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）。回想在第6章讨论过名词和动词之间可以互相连接起来。（例如，为每个传感器分配一个编号和类型，这样编号和类型就是数据对象Sensor的属性。）因此，在任何DFD层次中对某个泡泡的处理叙述文字进行语法解析，可以产生许多关于如何精化到下一个层次的有用信息。使用这些信息生成第1层的DFD如图7-2所示。在图7-1中显示环境层的处理，已经被扩展为6个处理，这些处理来自于语法解析检查。类似地，也通过解析获得第1层处理之间的信息流。此外，在第0层和第1层之间要保持信息流的连续性。

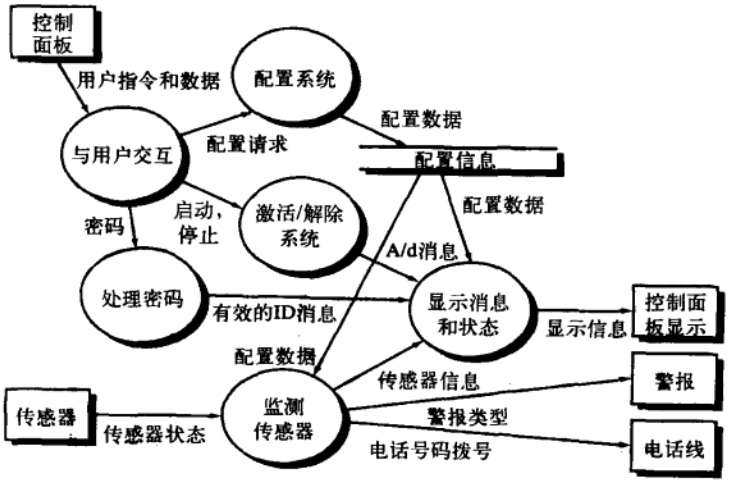


图7-2 SafeHome安全功能的第1层DFD

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，可以精化监测传感器

处理如图7-3所示的第2层DFD。再次提醒注意的是，在这两层之间保持了信息流的连续性。

持续进行DFD的求精，直到每个泡泡都执行了某个单一的功能，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成为一个程序构件。在第8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处理关注点。现在，我们只是努力精化DFD，直到每个泡泡都是“功能单一的”。

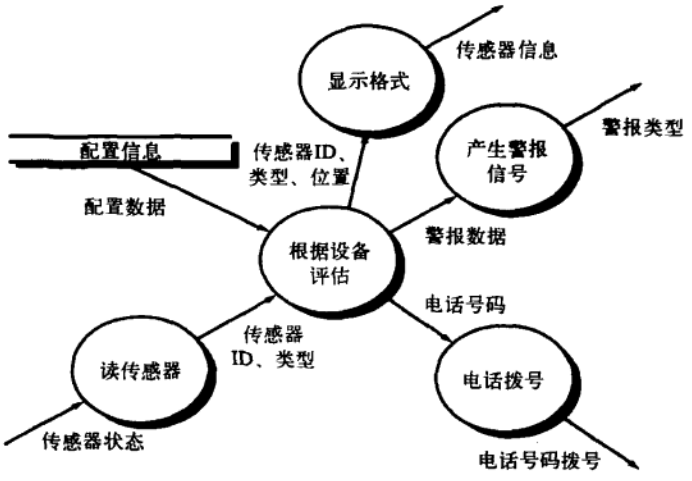


图7-3　精化监测传感器处理的第2层DFD

7.2.2　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。这样的应用系统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，真或假、开或关、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：

如何为控制流图、状态图或控制规格说明CSPEC选择潜在的事件？

- 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或动词的语法解析，考察所有可能作为控制规格说明输入/输出的“控制项”。
- 通过标识其状态来描述系统的行为，标识如何达到这些状态，并定义状态间的迁移。
- 关注可能的疏忽，即关注那些描述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？”

在很多事件和控制项中部分SafeHome软件工作是传感器事件（例如，传感器断开）、闪烁标志（显示闪动信号）以及启动或停止开关（开关系统的信号）。

7.2.3　控制规格说明

控制规格说明（Control Specification, CSPEC）（在被引用的层次上）使用两种不同的方式

表现系统的行为[⊖]。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即行为的组合说明。

图7-4为SafeHome的第1层控制流模型描述了一个初步的状态图[⊜]，图中显示了当系统在这个层次上定义的4个状态之间来回移动的时候，系统如何对事件作出响应。通过考察状态图，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所描述的行为中是否存在“空洞”(holes)。

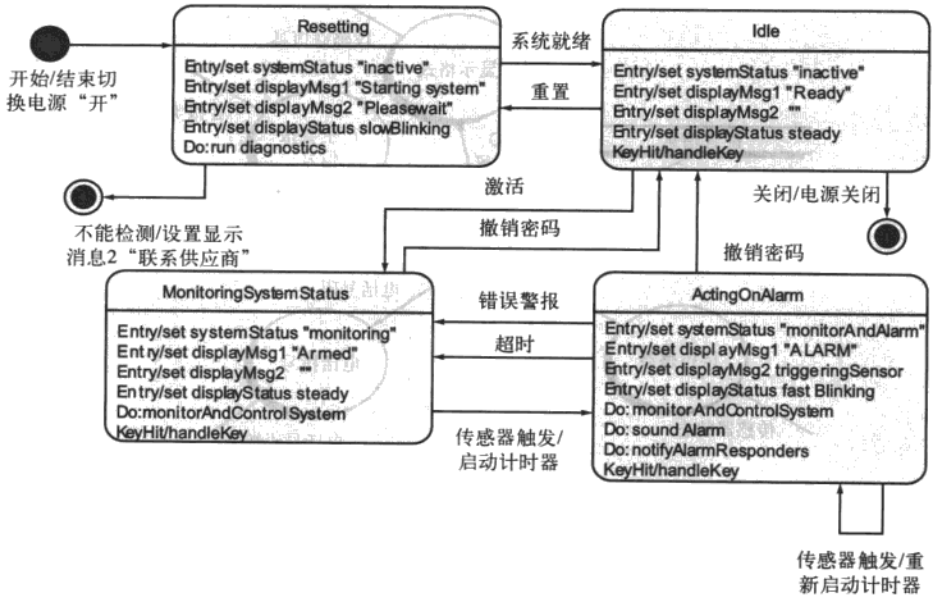


图7-4 SafeHome安全功能的状态图

例如，状态图（图7-4）指明：当系统重置、激活或电源关闭时，可能会发生Idle（空闲）状态的转换；如果激活系统（也就是打开报警系统），将会转换到MonitoringSystemStatus状态（监测系统状态），显示信息也将发生变化，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，转换回到Idle空闲状态；(2) 当触发传感器时，转换到ActingOnAlarm状态。在评审中将考虑所有的转换和所有状态的内容。

行为表达的一种非常不同的模式是在处理激活表（Process Activation Table, PAT），它表示了在状态图中处理环境所包含的信息，不包括状态。因此这张表指出了当有事件发生时会引入流程模型中哪个处理（泡泡）。处理激活表可以作为设计人员的指南，必须建立在这一级上可以执行的处理控制。

SafeHome第1层流程模型的处理激活表如图7-5所示。

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，这些处理激活了行为的结果。在7.2.4节中将讨论提供这种信息的建模表示方法。

⊖ 更多行为建模表示方法将在7.3节讨论。

⊜ 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为优秀。

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7-1到图7-5所示的模型草图，并正在展示给Ed和Vinod看。）

Jamie：我在学校里上过软件工程的课并学到这些东西，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。但是我在这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？

输入事件						
传感器事件	0	0	0	0	1	0
闪烁标志	0	0	1	1	0	0
启动停止开关	0	1	0	0	0	0
显示动作状态完成	0	0	0	1	0	0
正在进行中	0	0	1	0	0	0
超时	0	0	0	0	0	1
输出						
警告信号	0	0	0	0	1	0
处理激活						
监视控制系统	0	1	0	0	1	1
激活/撤销系统	0	1	0	0	0	0
显示消息和状态	1	0	1	1	1	1
与用户交互活动	1	0	0	1	0	1

图7-5　SafeHome安全功能的处理激活表

Vinod：就是Input-Process-Output（输入-处理-输出），DFD确实非常直观……如果你观察一会儿，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？

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，尽管状态图和PAT也能完成同样的工作，必须开始讨论基于类的元素，还有行为方面的工作。

Ed：我们有那么多的活要做，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，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。）

7.2.4 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程模型的处理。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[⊖]、数学方程、表或UML活动图。通过为流模型中的每个泡泡提供PSPEC，软件工程师创建了“小型规格说明”（mini-spec），小型规格说明可以

⊖ PDL把编程语言的语法和叙述性正文混在一起，目的是提供处理流程的设计细节。PDL将在第10章讨论。

KEY POINT
PSPEC是指在DFD的最低求精层的每个转换都是一个“小型规格说明”。

作为软件构件实现处理的设计指南。

为了说明如何使用PSPEC，考虑在SafeHome的流程模型中表示“处理密码”变换（图7-2），该功能的PSPEC可能采用如下形式：

PSPEC：处理密码（控制面板上）。在SafeHome安全功能中“处理密码”变换完成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，如果与主密码匹配，则向“显示消息和状态”功能传送有效信息<valid id message = true>。如果与主密码不匹配，则把4位密码与次密码表比较（这些密码可能会授予房子的客人和（或）在房主不在家时需要进入房子的工人）。如果密码和表中的某项匹配，将向“显示消息和状态”功能传送有效信息<valid id message = true>；如果不匹配，则向“显示消息和状态”功能传送无效信息<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才确定。

SOFTWARE TOOLS

结构化分析

目的：结构化分析工具有助于软件工程师进行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、流程模型和行为模型。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，并在错误扩散到设计甚至更糟的是扩散到实现之前就帮助他们消除错误。

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。一旦定义完字典里的条目，就可以创建实体－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提供一些特性。分析工具还帮助软件工程师使用状态图作为有效的表示方法来创建行为模型。

代表性工具：⊖

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），该工具用于定义分析或设计方法（包括结构化分析），还有其概念、规则、表示方法、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。

7.3 生成行为模型

如何用模型表明软件对某些外部事件的影响？

在此之前我们曾讨论的建模表示方法仅是表达了需求模型中的静态元素。现在则是要把它转换成系统或产品的动态行为的时候了。要实现这一任务，可以把系统行为表示成一个特定的事件和时间的函数。

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：

1. 评估所有的用例，以保证完全理解系统内的交互顺序。
2. 识别驱动交互顺序的事件，并理解这些事件如何与特定的对象相互关联。

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

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。

7.3.1 识别用例事件

在第6章获知用例表现了涉及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。在7.2.3节中所指的事件，应该不是被交换的信息，而是已交换信息的事实。

为了说明如何从信息交换的角度检查用例，让我们再来考察SafeHome安全功能中的一部分用例。

房主使用键盘键入4位密码。该密码和保存在系统中的有效密码相比较。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，而且应列出任何条件或限制。

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”。在需求模型的环境下，对象Homeowner房主[⊖]向对象ControlPanel发送一个事件。这个事件可以称做“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”不会明显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有效密码相比较”的交互中得到）的结果将明显地影响到SafeHome软件的信息流和控制流。

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房主生成“输入密码”事件）或识别已经在其他地方发生的事件（例如，ControlPanel控制面板识别“比较密码”事件的二元结果）。

7.3.2 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[⊜]。

类状态具有被动和主动两种特征[Cha93]。被动状态只是某个对象所有属性的当前状态。例如，类Player的被动状态（在第6章讨论的视频游戏应用程序中）将包含**Player**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或处理时的当前状态。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。必然发生事件（有时被称为触发器）才能迫使对象做出从一个主动状态到另一个主动状态的转移。

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前系统执行功能时的行为。

下面将讨论两种不同的行为表现形式。第一种显示一个类如何改变基于外部事件的状态，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。UML状态图就是一种行为模型[⊜]，该图为每个类呈现了主动状态和导致这些主动状态变化的事件（触发器）。图7-6举例说明了SafeHome安全功能

⊖ 在这个例子中，假设SafeHome交互的每个用户（房主）都有正确的密码，因此就是合法的对象。
⊜ 在第6章和7.3.2节介绍的状态图中描述了系统的状态。本节我们的讨论将集中在分析模型中每个类的状态。
⊜ 如果读者对UML不熟悉，在附录1中有这些重要建模符号的简单介绍。

中ControlPanel类的状态图。

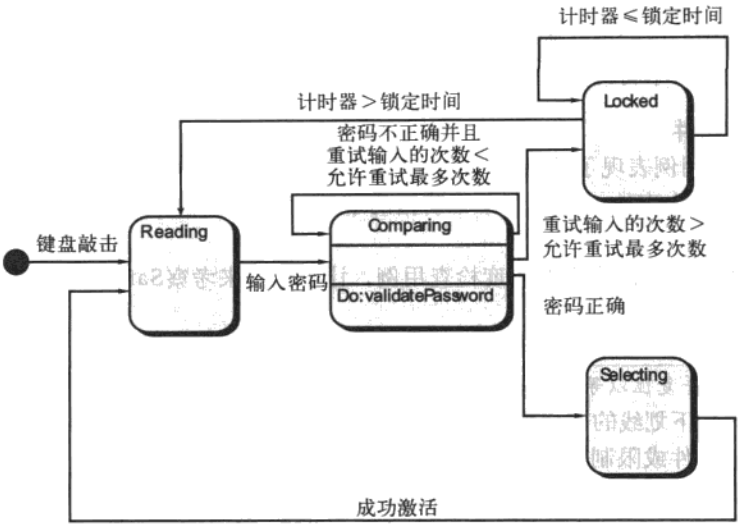


图7-6 ControlPanel类的状态图

图7-6中显示的每个箭头表示某个对象从一个主动状态转移到另一个主动状态。每个箭头上的标注都体现了触发状态转移的事件。尽管主动状态模型在提供对象的“生命历史”信息方面非常有用，但也能提供另外一些信息以便加深理解对象的行为。除了说明导致转移发生的事件外，分析师还可以说明守卫Guard和动作[Cha93]。 守卫是为了保证转移发生而必须满足的一个布尔条件。例如，图7-6中从“读取”状态转移到“比较”状态的守卫可以由检查用例来确定：

if (password input = 4 digits) then compare to stored password

一般而言，转移的守卫通常依赖于某个对象的一个或多个属性值。换句话说，守卫依赖于对象的被动状态。

动作是与状态转移同时发生的或者它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（见图7-6）相关联的动作是由名为validatePassword( )操作的，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。

KEY POINT
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状态来表现行为。

顺序图。第二种表现行为的方式在UML中称做顺序图，它表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，即用时间函数表现如何引发事件从一个对象流到另一个对象。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7-7给出了SafeHome安全功能的部分顺序图。每个箭头代表了一个事件（源自一个用例）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。

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”事件发送到System，它在一个简单的数据库中查找密码并向ControlPanel（现在处在比较状态）返回（找到或没有找到）结果。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，通过控制把“成功激活”事件返回到房主。

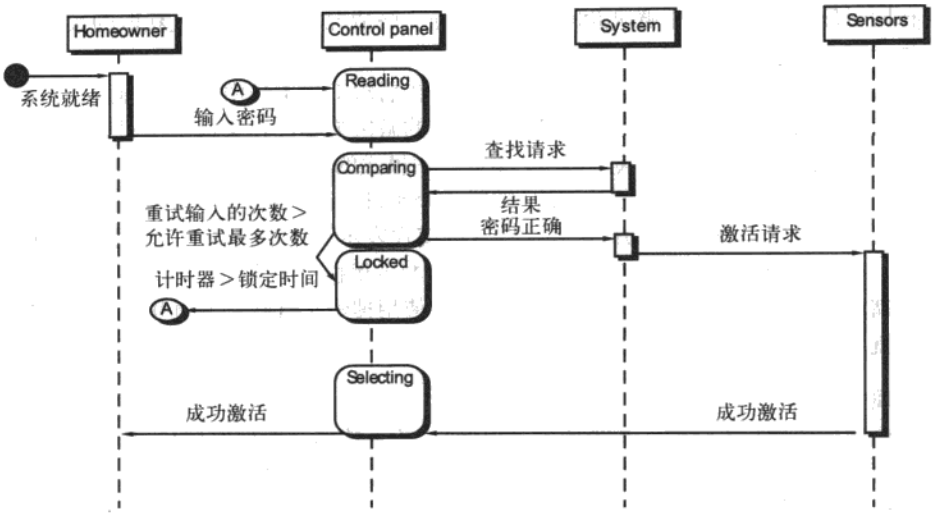


图7-7　SafeHome安全功能的顺序图（部分）

一旦完成了完整的顺序图，把所有导致系统对象之间转移的事件整理为输入事件集合和输出事件集合（来自一个对象）。对于将要构建的系统而言，这些信息对于创建系统的有效设计非常有用。

SOFTWARE TOOLS

UML中的通用分析建模

目的：分析建模工具可以使用UML语言开发基于场景的模型、基于类的模型和行为模型。

机制：这类工具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，这类工具：（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，以便实现管理和评估复杂系统所需的大型UML模型。

代表性工具：⊖

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）。

Enterprise Architect，由Sparx Systems开发（www.sparxsystems.com.au）。

PowerDesigner，由Sybase开发（www.sybase.com）。

Rational Rose，由IBM (Rational) 开发（www. ibm.com/software/rational）。

System Architect，由Popkin Software开发（www.popkin.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）。

7.4　需求建模的模式

软件模式是获取领域知识的一种机制从而遇到新问题时可以反复使用。在某些情况下，领

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

域知识在同一应用领域中用于解决新问题。在另外一些情况下，通过模式获取的领域知识可借助模拟用于完全不同的应用领域。

分析模式的最初创作者没有“创建”模式，但在需求工程工作中发现了模式。一旦发现模式则记载“明确的常见问题：哪种模式适用、规定的解决方案、在实践中使用模式的假设和约束，以及关于模式的某些常见的其他消息，诸如使用模式的动机和驱动力，讨论模式的优缺点，参考在实践应用中某些已知的使用模式的样例”[Dev01]。

在第5章，引入了分析模式的概念并指明这些模式表示了在某些应用领域中常合并一个类、一个功能、或一个行为的解决方案。当为某个领域的应用执行需求建模时会重用模式[⊖]。分析模式都存储于一个仓库中以便软件团队的成员能够使用搜索工具找到并复用。一旦选择到合适的模式，就可以通过参考模式名称整合到需求模型中。

7.4.1 发现分析模式

需求模型由各种元素组成：基于场景（用例）、基于数据（数据模型）、基于类、基于流和行为。其中每个元素都是从不同的视角检查问题，并且每一个都提供一种发现模式的机会，可能发生在整个应用领域，或者发生在类似但横跨不同的应用领域。

在需求模型的描述中最基本的元素是用例。在这个讨论环境下，一套连贯用例可以成为服务于发现一个或多个分析模式的基础。语义分析模式（semantic analysis pattern, SAP）“是描述了一小套连贯用例，这些用例一起描述了通用应用的基础”[Fer00]。

以下我们考虑要求控制和监控“实时查看摄像机”和汽车临近传感器的软件用例。

用例：监控反向运动

描述：当车辆安装了反向齿轮，控制软件就能从后向视频摄像机将一段视频输入到仪表板显示器上。控制软件在仪表板显示器上叠加各种各样距离和方向的线，以便车辆向后运动时驾驶员能保持方向。控制软件还能监控临近传感器，以判定在车后方10英尺内是否有物体存在。如果临近传感器检测到某个物体在车后方x英尺内就会让车自动停止，这个x值由车辆的速度决定。

在需求收集和建模阶段，本用例包含（在一套连贯用例中）各种将要精炼和详细阐述的功能。无论完成得如何精炼，建议用例要简单，但还要广泛地适用于SAP，即具有基于软件的监控和在一个物理系统中对传感器和执行器的控制。在本例中，“传感器”提供临近信息和视频信息。“执行器”用于车辆的停止系统（如果一个物体离车辆很近就会调用它）。但是更常见的情况是发现大量的应用模式。

许多不同应用领域的软件需要监控传感器和控制物理执行器。所依照的分析模式描述了能广泛应用的通用需求。这个模式叫做**Actuator-Sensor**（执行器-传感器），将用作SafeHome的部分需求模型，将在随后的7.4.2节讨论。

7.4.2 需求模式举例：执行器-传感器[⊜]

SafeHome安全功能需求之一是能监控安全传感器（例如，入侵传感器，火、烟或一氧化碳传感器，水传感器）。基于扩展至互联网的SafeHome将要求控制住所内安全摄像机的活动能力（例如摇摄、聚焦）。暗指SafeHome软件必须管理各种传感器和“执行器”（例如摄像机控制机构）。

Konrad和Cheng[Kon02]建议需求模式命名为执行器-传感器，它为SafeHome软件这项需求的建模提供有用的指导。执行器-传感器模式的简约版本最初是为如下的汽车应用开发的。

⊖ 在软件设计中使用模式的深入讨论将在第12章表述。

⊜ 本节内容来自[Kon02]（得到作者允许）。

模式名：执行器-传感器

目的：详细说明在嵌入系统中的各种传感器和执行器。

动机：嵌入系统常有各种传感器和执行器。这些传感器和执行器都直接或间接连接到一个控制单元。虽然许多传感器和执行器看上去十分不同，但它们的行为是相似的，足以让它们构成一个模式。这个模式显示了如何为一个系统指定传感器和执行器，包括属性和操作。执行器-传感器模式为被动式传感器使用“拉机制”（对消息的显示要求），为主动传感器使用“推机制”（消息广播）。

约束：

- 每个被动传感器必须有某种方法读取传感器的输入和表示传感器值的属性。
- 每个主动传感器必须能在其值发生变更时广播更新消息。
- 每个主动传感器应该能发送一个生命刻度，即在特定时间帧中发布状态信息，以便检测出错误动作。
- 每个执行者必须有某种方法调用由ComputingComponent计算构件决定的适当应答。
- 每个传感器和执行器应有实施检测其自身操作状态的功能。
- 每个传感器和执行器能测试接受值或发送值的有效性，并且当值超出指定边界时能设定其操作状态。

适用性：对有多个传感器和执行器的任何系统都是非常有用的。

结构体：图7-8显示了执行器-传感器模式的UML类图，执行器、被动传感器和主动传感器是抽象类。这个模式中有四种不同的传感器和执行器。

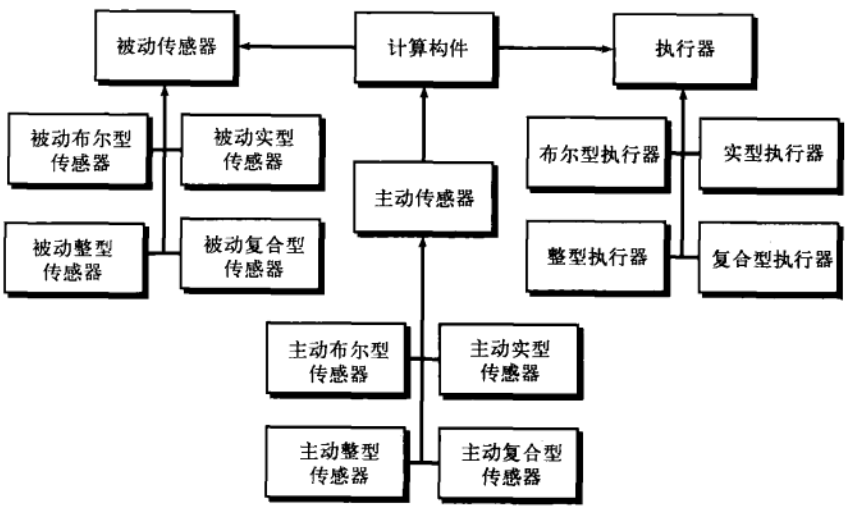


图7-8　传感器和执行器的UML顺序图

资料来源：来自[Kon02]（得到许可）。

传感器和执行器的常见类型为Boolean、Integer和Real。就基本数据类型而言，复杂类是不能用值简单表示的，例如雷达设备。虽然如此但是这些设备还应该继承来自抽象类的接口，因为它们应该具备基本的功能如查询操作状态。

行为：图7-9描述了执行器-传感器例子的UML顺序图。这个例子可以应用于SafeHome功能，用以控制安全摄像机的调整（例如摇摄、聚焦）。这里在读取或设置一个值时，

ControlPanel[⊖]需要一个传感器（被动传感器）和一个执行器（摇摄控制器）进行以诊断为目的核查操作状态。名为Set Physical Value 和Get Physical Value的消息不是对象间的信息，相反它们描述了系统物理设备和相关软件对应项之间的交互活动。在图的下部即在水平线以下，PositionSensor报告操作状态为零。接着ComputingComponent（表示为ControlPanel）发送位置传感器失败的错误代码给FaultHandler错误处理程序，它将决定这些错误如何影响系统，并需要采取什么措施。从传感器获得的数据经过计算得到执行器所需的应答。

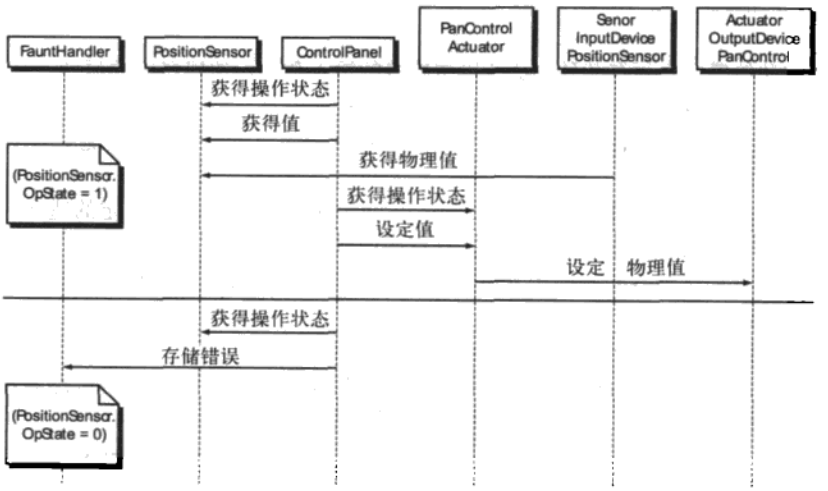


图7-9 执行器—传感器模式的UML类图。

注：重印源自[Kon02]并经作者允许

参与者。模式描述的这节内容“详细列举了包括在需求模式中的类或对象”[Kon02]，并描述了每个类或对象（图7-8）的职责。简单列表如下：

- PassiveSensor abstract：定义被动传感器接口
- PassiveBooleanSensor：定义被动布尔型传感器
- PassiveIntegerSensor：定义被动整型传感器
- PassiveRealSensor：定义被动实型传感器
- ActiveSensor abstract：定义主动传感器接口
- ActiveBooleanSensor：定义主动布尔型传感器
- ActiveIntegerSensor：定义主动整型传感器
- ActiveRealSensor：定义主动实型传感器
- Actuator abstract：定义执行器接口
- BooleanActuator：定义布尔型执行器
- IntegerActuator：定义整型执行器
- RealActuator：定义实型执行器
- ComputingComponent：控制器的中心部分，它从传感器得到数据并为执行器计算所需的应答。
- ActiveComplexSensor：主动复合型传感器具备抽象类ActiveSensor的基本功能，但还需

⊖ 原本模式使用通用词组ComputingComponent。

要指定额外更详细的方法和属性。

- PassiveComplexSensor：被动复合型传感器具备抽象类PassiveSensor的基本功能，但还需要指定额外更详细的方法和属性。
- ComplexActuator：复合型执行器具备抽象类Actuator的基本功能，但还需要指定额外更详细的方法和属性。

协作。本节描述的是对象和类之间如何进行交互活动以及让它们各自如何能实现自身的责任。

- 当ComputingComponent需要更新PassiveSensor的值时，它询问传感器，通过发送适当的消息请求传值。
- ActiveSensor无需查询。这些对象和类启动向计算机单元中传送传感器的值，使用适当方法设定在ComputingComponent中的值。对象和类在指定的时间帧发送至少一次生命刻度，以便用系统的时钟时间更新它们的时间戳。
- 当ComputingComponent需要设定执行器的值时，给执行器发送值。
- ComputingComponent能使用适当的方法查询和设定传感器和执行器的操作状态。如果发现操作状态为零，就发送错误给FaultHandler错误处理程序，这个类包含处理错误消息的方法，比如启动更详细的恢复机制或者一个备份设备。如果不可恢复，系统只能使用传感器最后的已知值或者默认值。
- ActiveSensor提供增加或消除地址的方法或者组件的地址范围的方法，一旦值发生变化组件就能获得消息。

结果

1. 传感器和执行器类有一个通用接口。

2. 只能通过消息访问类的属性，并且类决定是否接受这个消息。例如，如果设定执行器的值超过其最大值，执行器类就不可能接受消息，或者使用默认的最大值。

3. 降低系统潜在的复杂度是因为传感器和执行器有统一的接口。

需求模式的描述也能提供参考给其他相关的需求模式和设计模式。

7.5 Web应用系统的需求建模[⊖]

Web开发者时常怀疑为Web应用建议的需求分析观念。他们争辩道："Web开发过程终归必须使用敏捷方法，并且分析是非常耗时的。当我们需要设计和建立Web应用系统时就会减慢我们的工作进度"。

需求分析确实很花时间，但是解决错误的问题甚至更花时间。每个Web应用系统开发者的问题是简单的，但是能否确保理解了问题的需求？如果答案是毫不含糊的"是"，那么可以跳过需求建模，但是如果答案是"否"就应该实施需求建模。

7.5.1 如何分析

Web应用需求建模的重视程度依据以下因素：

- Web应用的规模和复杂度的增长。
- 相关利益者的人数（所做的分析能帮助识别不同来源的需求冲突）。
- Web应用团队的人数。
- 一起工作以前该Web应用团队成员的级别（所做的分析有助于达成对项目的共同理解）。
- 组织成功的程度直接依赖于Web应用的成功。

⊖ 本节采用Pressman 和Lowe的内容[Pre08]，并得到作者许可。

与前面相反的观点是当这个项目规模更小，利益相关者更少，开发团队更有内聚力，应用系统并非十分重要，采用更轻量级的分析方法是合理的。

虽然在开始设计前作问题分析是个好主意，但并不见得所有的分析一定在所有的设计之前。事实上，Web应用系统中仅有特定部分的设计会要求分析对Web应用系统有影响的需求。例如SafeHome，对所有网站做符合要求的美学设计（版面布局、色彩框架等）不需要分析电子商务能力的功能需求。为提高交付成果软件分析师只需要分析与递增式交付相关的设计工作部分。

7.5.2 需求建模的输入

当Web应用系统已经工程化时，可以采用第2章讨论的常规软件过程的敏捷版本。这个过程包含的沟通活动可以识别利益相关者和用户类别、业务环境、界定信息和可用的目标、普通的Web应用系统需求和使用场景，这些都是能成为需求建模输入的信息。这些信息以自然语言的描述、概要大纲、草图以及其他非正式形式表达。

分析这类信息，并使用（所适用的）正规定义的表达模式构造它，接着生成更缜密的模型作为输出。需求模型提供了揭示问题真实结构的详细指导，并提供洞察其特性的解决方案。

在第6章介绍SafeHome的ACS-DCV（摄像机监视）功能时这个功能看上去相当清晰，在（6.2.1节）描述了用例的部分细节。然而，当再次审查用例时就可能发现丢失的信息、模棱两可的信息或不清晰的信息。

某方面的丢失信息自然会在设计阶段显露出来。例如功能键的特定布局、美学外观、快照视图尺寸、摄像机观察点位置、房间地板平面图、或者如密码的最大和最小长度这些细节。这些方面的某些内容是由设计决定的（如按键布局），另一些内容是根本不能影响早期设计工作的需求（如密码长度）。

但是有些丢失的信息可能实际上影响总体设计，更多的则与需求的实际理解相关。例如：

问题1：通过SafeHome摄像机输出的视频分辨率是多少？

问题2：如果正在被监控的摄像机遇到警报条件，会发生什么？

问题3：系统如何操作能摇摄和聚焦的摄像机？

问题4：摄像机的视图上还应该提供哪些信息？（例如位置在何处？时间或日期如何？上一次访问如何？）

在用例的初始开发阶段都没有考虑到这些问题，但是这些问题的答案却对设计的不同方面有着很大影响。

虽然沟通活动提供了很好的理解基础，但是需求分析通过提供额外的解释会使理解更精确，因此这个结论是合理的。当把问题的结构描述成需求模型的部分内容时，新问题又出现了。这就是弥补理解差距的问题，（或者在某些情况下）实际上这些问题也帮助我们在第一时间发现了这些差距。

总之，在沟通活动中收集到的信息作为需求模型的输入，即从非正规的电子邮件到包括综合使用场景和产品规格说明书中详细项目概述的任何内容。

7.5.3 需求建模的输出

需求分析提供了严格规定的机制来描述和评估Web应用系统的内容和功能、用户将遇到的交互模式，以及Web应用系统所处的环境和基础设施。

每种特性都能表示成一套模型，这套模型允许以结构化方式分析Web应用系统的需求。当特定模型很大程度上依赖于Web应用系统的性质时，会有5种主要的模型类型：

- **内容模型**给出由Web应用系统提供的全部系列内容。内容包括文本、图表和图像、视频和音频数据。
- **交互模型**描述了用户与Web应用系统采用了哪种交互方式。

- **功能模型**定义了将用于Web应用系统内容并描述其他处理功能的操作，这些处理功能不依赖于内容却是最终用户所必需的。
- **导航模型**为Web应用系统定义所有导航策略。
- **配置模型**描述Web应用系统所在的环境和基础设施。

分析师可以使用描述模式开发每一种模型，这种常称为“语言”的描述模式考虑到其意图和结构，使Web工程团队的成员和相关利益者之间更容易进行沟通和评估。结果是识别出关键问题列表（例如错误、遗漏、不一致性、供增强和更改的建议、澄清观点）并照此行动。

7.5.4　Web应用系统内容建模

内容模型包含结构元素，它们为Web应用系统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和所有分析类，在用户与Web应用系统交互时生成并操作用户可见的实体[⊖]。

内容的开发可能发生在Web应用系统实现之前、Web应用系统构建之中，或者Web应用系统投入运行以后的很长一段时间里。在每种情况下，它都通过导航链接合并到Web应用系统的总体结构中。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、在体育运动事件中拍摄的一张动作照片、在一次论坛讨论中某个用户的回答、公司徽标（logo）的一种生动演示、一个演讲的简短视频，或者是一套演示幻灯片中的音频插曲。这些对象内容可能存储于分离的文件中，直接嵌入Web页中，或从数据库中动态获得。换句话说，内容对象是呈现给最终用户具有汇聚信息的所有条目。

通过检查直接或间接引用内容的场景描述，可以根据用例直接确定出内容对象。例如，建立在SafeHomeAssured.com中支持SafeHome的Web应用系统。用例“选择SafeHome构件” 描述了需要购买SafeHome构件的场景并包含如下句子：

“我将能得到每种产品构件的说明和价格信息。”

内容模型必须具备描述内容对象构件的能力。在许多实例中，用一个简单的内容对象列表，并给出每个对象的简短描述就足以定义设计和实现所必须的内容需求。但是在某些情况下，内容模型可以从更丰富的分析中获得好处，即在内容对象之间的关系和（或者）Web应用系统维护的内容层次中采用图形化表示其关系。

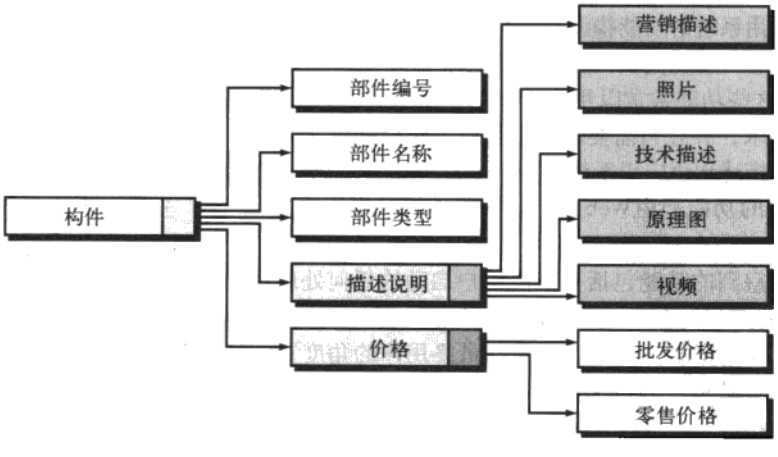


图7-10　SafeHomeAssured.com构件的数据树

⊖　分析类已在第6章讨论。

例如考虑图7-10中为SafeHomeAssured.com构件建立的数据树[Sri01]。该树表述了常用于描述某个构件的一种信息层次。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（描述说明）是由5个内容对象（有阴影的长方形）定义的。在某些情况下，随着数据树的扩展，其中的一个或多个对象将会得到进一步细化。

由多项内容对象和数据项组成的任何内容都可以生成数据树。开发数据树尽力在内容对象中定义层级关系，并提供一种审核内容的方法，以使在开始设计前得到揭露遗漏和不一致的内容。另外，数据树可以作为内容设计的基础。

7.5.5 Web应用系统的交互模型

绝大多数Web应用系统都能使最终用户与应用系统的功能、内容及行为之间进行“会话”。可以使用交互模型描述会话，这种交互模型由一种或多种元素组成：(1) 用例；(2) 顺序图；(3) 状态图[⊖]；和（或）(4) 用户界面原型。

在许多实例中，一套用例足以描述在分析阶段的所有交互活动（在设计阶段引入更精化的细节）。然而当遇到复杂的交互顺序并包含多个分析类或多任务时，有时更值得采用严格图解方式描述它们。

用户界面的布局、介绍的内容、实现的交互机制以及用户与Web应用系统连接上的总体审美观，都与用户的满意度和Web应用系统的整体成功密切相关。虽然有理由认为用户界面原型的创建是一种设计活动，但在创建分析模型期间执行用户界面原型的创建仍是一个好办法。对用户界面的物理表示评估得越早，就越有可能满足最终用户的需求。在第11章将详细讨论用户界面的设计。

因为Web应用系统的构造工具种类繁多，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，并采用同样的构造方式表现总体的屏幕布局。例如，如果为用户提供5种主要系统功能，当用户看到这些原型第一次输入到Web应用系统时，这些原型就表示了这些系统功能。如果要问将会提供图形链接吗？导航菜单会显示在哪里？用户会看到哪些其他信息？可以由原型回答以上类似的问题。

7.5.6 Web应用系统的功能模型

许多Web应用系统提供大量的计算和操作功能，这些功能与内容直接相关（既能使用又能生成内容）。这些功能常常以用户-Web应用系统的交互活动为主要目标。正是因为这个原因必须分析功能需求，并且当需要时也可以用于建模过程。

功能模型描述Web应用系统的两个处理元素，每个处理元素代表抽象过程的不同层次：(1) 用户可观察到的功能是由Web应用系统传递给最终用户的；(2) 分析类中的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Web应用系统可以执行多种财务功能（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使用分析类中的操作完成，但是从最终用户的角度看，这些功能（更正确地说，这些功能所提供的数据）是可见的结果。

在抽象过程的更低层次，分析模型描述了由分析类操作执行的处理。这些操作可以操纵类的属性，并参与类之间的协作来完成所需要的行为。

不管抽象过程的层次如何，UML的活动图可用来表示处理细节。在分析阶段，仅在功能

⊖ 使用UML符号为顺序图或状态图建模。7.3节描述了状态图。更多的细节请参看附录1。

相对复杂的地方才会使用活动图。许多Web应用系统的复杂性不是出现在提供的功能中，而是与可访问信息的性质以及操作的方式相关。

在SafeHomeAssured.com中相对复杂功能的举例可以由一个用例来描述，这个用例名为“为我的空间中传感器的布局提供建议”。用户已经开发了一个可以监控空间的布局，在该用例中选择该布局以及要求传感器在该布局中放于建议的位置。SafeHomeAssured.com用该布局的图形化表示方法回应，为推荐位置的传感器提供额外的信息。交互活动是非常简单的，某些内容有些复杂，然而它的底层功能非常复杂。系统必须承担地面布局的复杂分析，以便决定传感器的优化位置。必须检查房间面积、门窗位置，并协调这些传感器的功能和技术要求。这绝不是个小任务！使用一套活动图来描述这个用例所做的处理。

第二个例子是名为“控制摄像机”的用例。在这个用例中，交互活动相对简单，但存在潜在的复杂功能，所给的“简单”操作需要通过访问互联网和远程设备进行复杂通信。当有多个已经授权的人同时想监视并且（或者）控制某个传感器时，可能有更深一步复杂情况与控制权谈判有关。

图7-11为takeControlCamera( )操作描绘了一幅活动图，它是“控制摄像机”用例中所用摄像机分析类的一部分。应该注意到程序流程中包括两项额外操作：requestCameraLock( )是想为这个用户锁定摄像机；getCurrentCameraUser( )是获取当前控制这台摄像机的用户名字。暂不考虑描述如何调用这些操作的结构细节和每项操作的接口细节，直到Web应用系统设计阶段才会给出建议。

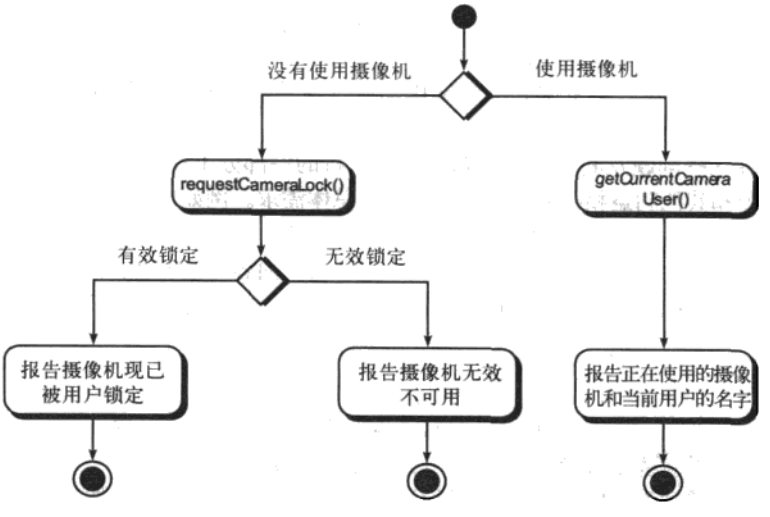


图7-11　takeControlOfCamera()操作的活动图

7.5.7　Web应用系统的配置模型

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Web应用系统来说，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。

应该指定通过所有主要的Web客户端访问SafeHomeAssured.com的公共内容和功能（例如，

那些多于1%市场占有率的Web客户端[⊖]）。相反地，对于少量客户端也可限于接受复杂控制和监控功能（只允许HomeOwner房主用户访问）。SafeHomeAssured.com的配置模型也将与指定的现存产品数据库和监控应用进行交互操作。

7.5.8 导航建模

导航建模考虑了每一类用户如何从一个Web应用系统元素（例如，内容对象）导航到另一个元素。把导航机制定义为设计的一部分。在这个阶段，分析人员应该关注于总体的导航需求，应该考虑以下问题：

- 某些元素比其他元素更容易到达吗（即需要更少的导航步骤）？表示的优先级别是什么？
- 为了促使用户以他们自己的方向导航，应该强调某些元素吗？
- 应该怎样处理导航错误？
- 导航到相关元素组的优先权应该高于导航到某个特定元素的优先权吗？
- 应该通过链接方式、基于搜索的访问方式还是其他方式来实现导航？
- 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？
- 在用户交互的每一点处，（与单个“返回”链接或有方向的指针相比）一个完整的导航地图或菜单都可用吗？
- 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Web应用系统元素可感知的重要性来驱动？
- 为了促进将来使用快捷，一个用户能否“存储”他以前对Web应用系统的导航？
- 应该为哪类用户设计最佳导航？
- 应该如何处理Web应用系统外部的链接？应该覆盖现有的浏览器窗口吗？能否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工作。

软件工程师和其他利益相关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Web应用系统的结构吗？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（包括内容对象和功能）吗？用户能够访问内容对象或者由其元素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）？

7.6 小结

在数据流对象作处理功能转换时，面向流程建模关注于数据流对象。从结构化分析进行推导，面向流建模使用了数据流图，建模符号描述了当数据对象移动通过一个系统时如何将输入转化为输出。由一个处理的规格说明或者叙述来描述转换数据的每个软件功能。除了数据流以外，这些建模元素也描述了控制流，控制流是说明事件如何影响系统行为的一种图示。

行为建模描述了动态行为。行为模型采用从基于场景、面向流程、基于类的输入，从而把分析类和系统的状态作为一个整体来表达。为达到这一目的，要识别状态，定义引起类（或系统）由一个状态转换到另一状态的事件，还要识别当完成转换后发生的活动。状态图和顺序图是行为建模的常用表达方式。

分析模式能让软件工程师使用现有的知识领域，便于建立需求模型。一个分析模式描述了一个特定的软件特性或功能，该特性或功能由一套相关用例描述。本章详细说明了模式的目的、

⊖ 判定浏览器的市场份额是众所周知的问题，其变化是由所采用的调查方式决定的。然而在写作本书时，据报道只有Internet Explorer 和Firefox的市场份额超过了30%，而Mozilla、Opera和Safari等仅各占1%。

使用的动机、使用的限制约束、各种问题域中的可应用性、模式的完整结构、其行为和协作，以及其他辅助信息。

Web应用系统的需求建模能使用本书讨论的大多数的建模元素。但是只在一套指定的模型中使用这些元素，它涉及驻留于Web应用系统中的内容、交互操作、功能、导航和服务客户端的配置。

习题与思考题

7.1　对于需求分析，结构分析与面向对象策略有何本质区别？
7.2　在数据流图中，一个箭头表示控制流还是其他？
7.3　什么是“信息流连续性”？当重新定义一个数据流时，如何应用它？
7.4　在生成DFD图时，如何使用图形化解析？
7.5　什么是控制规格说明？
7.6　PSPEC和用例是同一事物吗？如果不是，请解释区别。
7.7　表示行为建模时有两种不同“状态”类型，它们是什么？
7.8　如何从状态图区分顺序图？它们有何相似之处？
7.9　为现代移动手机建议3种需求模式，并简约描述每种模式。这些模式能否被其他设备使用？举例说明。
7.10　在问题7.9中选择一个你要开发的模式，模拟7.4.2节中表达的某个内容和模型，开发出一个合理并完整的模式描述。
7.11　你认为SafeHomeAssured.com需要多少种分析模型？这些是在7.5.3节中描述的每种模型类型所需要的吗？
7.12　Web应用系统交互建模的目的是什么？
7.13　引起争议的问题是一个Web应用系统的功能模型应延期开发直至设计阶段。表述关于这个议题的支持和反对观点。
7.14　配置模型的目的是什么？
7.15　如何从交互模型中区分导航模型？

推荐读物与阅读信息

已经发表了大量关于结构分析的书，所有这些书都充分涉及这一主题，但只有少量真正出色的作品。DeMarco和Plauger（《Structured Analysis and System Spcification》，Pearson，1985）仍然是一部介绍基本表达方式的很好的经典之作。Kendall （《Systems Analysis and Design》，5th ed.，Prentice-Hall，2002）、Hoffer et al.（《Modern Systems Analysis and Design》，Addison-Wesley，3rd ed.，2001）、Davis 和Yen（《The Information System Consultant's Handbook:Systems Analysis and Design》，CRC Press，1998）以及Modell（《A Professional's Guide to Systems Analysis》，2nd ed.，McGraw-Hill，1996）的书都很值得参考。Yourdon关于这一主题的书（《Modern Structured Analysis》，Yourdon-Press，1989）仍然是至今发表的所有出版物中最全面的一本书。

行为建模表述了一个系统行为的动态视图。Wagner和他同事的书（《Modeling Software with Finite State Machines: A Practical Approach》，Auerbach，2006）以及Boerger和Staerk（《Abstract State Machines》，Springer，2003）深入讨论了状态图和其他行为的表示方法。

为软件模式所写的大部分书关注于软件设计。然而Evans（《Domain-Driven Design》，Addison-Wesley，2003）及Fowler（[Fow03]和[Fow97]）的书特别阐述了分析模式。

由Pressman 和Lowe[Pre08] 表述了需要深层次对待Web应用系统的分析模型。Murugesan和Desphande编写的论文集中包含了一篇（《Web Engineering:Managing Diversity and Complexity of Web Application Development》，Springer，2001），处理了Web应用系统需求的各个方面。另外International Conference on Web Engineering论文集每年发表解决需求建模的文章。

在互联网上开放了大量需求建模的资源。关于分析建模的最新WWW参考列表可以在SEPA网站上找到：www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm。

第8章

Software Engineering：A Practitioner's Approach (Seventh Edition)

设 计 概 念

要点浏览

概念：几乎每位工程师都希望做设计工作。设计体现了创造性——利益相关者的需求、业务要求和技术考虑都集中地体现在产品或系统的形成中。设计创建了软件的表示或模型，但与需求模型（注重描述所需的数据、功能和行为）不同，设计模型提供了软件体系结构、数据结构、接口和构件的细节，而这些都是实现系统所必需的。

人员：软件工程师负责处理每一项设计任务。

重要性：设计是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、进行测试以及大量最终用户使用之前，要对模型的质量进行评估，并进行改进。软件质量是在设计中建立的。

步骤：设计可以采用很多不同的方式描述软件。首先，必须表示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、软件和其他系统与设备以及软件和自身组成的构件之间起到连接作用；最后，设计构成系统的软件构件。每个视图表现了不同的设计活动，但所有的视图都要遵循一组指导软件设计工作的基本设计概念。

工作产品：在软件设计过程中，包含体系结构、接口、构件级和部署表示的设计模型是主要的工作产品。

质量保证措施：软件团队从以下各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；是否存在更好的可选方案；设计模型是否可以在已经设定的约束、时间进度和成本内实现。

关键概念

抽象
体系结构
方面
内聚
数据设计
设计过程
功能独立
良好的设计
信息隐藏
模块化
面向对象设计
模式
质量属性
质量指导原则
重构
关注点分离
软件设计
逐步精化

软件设计包括一系列原理、概念和实践，可以指导高质量的系统或产品开发。设计原理建立了指导设计工作的最重要原则。在运用设计实践的技术和方法之前，必须先理解设计概念，设计实践本身会产生软件的各种表示，以指导随后的构建活动。

设计是软件工程是否成功的关键。在20世纪90年代早期，Lotus 1-2-3的创始人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、适用和令人愉悦的特点。对好的软件来说也是如此。坚固：程序应该不含任何妨碍其功能的缺陷。适用：程序应该符合开发的目标。愉悦：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。

设计的目标是创作出坚固、适用和令人愉悦的模型或表示。为此，设计师必须先实现多样化，然后再进行聚合。Belady[Bel81]指出“多样化是指要获取所有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，应从满足需求工程和分析模型（第5到第7章）所定义的需求中挑选适合的元素。此时，考虑并取舍候选方案，并进行

聚合，使之成为“构件的某种特定的配置，从而创建最终的产品”[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。

软件设计随着新的方法、更好的分析和更广泛理解的进展而不断变更[⊖]。即使是今天，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9～13章中，我们将介绍应用于体系结构、接口和构件级设计的多种软件设计方法，也会介绍基于模式和面向Web的设计方法。

8.1 软件工程中的设计

“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转换。”——Richard Due'

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。一旦对软件需求进行分析和建模，软件设计就开始了。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（代码生成和测试）。

需求模型的每个元素（第6、7章）都提供了创建4种设计模型所必需的信息，这4种设计模型是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8-1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示的需求模型是设计任务的输入。使用后续章节所讨论的设计表示法和设计方法，将得到数据或类的设计、体系结构设计、接口设计和构件设计。

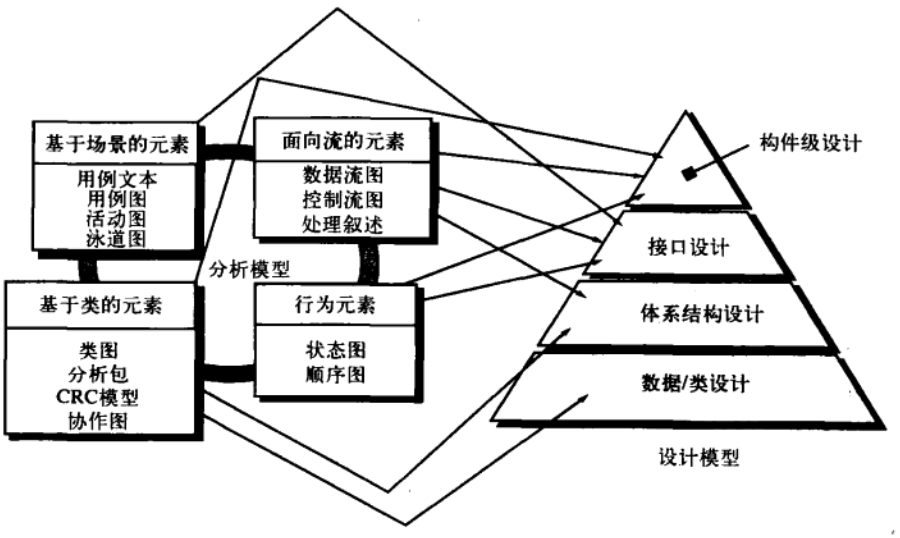


图8-1 需求模型到设计模型的转换

数据设计或类设计将类模型（第6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图中定义的对象和关系、类属性和其他表示法刻画的详细数据内容为数据设计活动提供了基础。在与软件体系结构设计的连接中可能会进行部分的类设计，更详细的类设计在设计

⊖ 对软件设计原理感兴趣的读者可能会对Philippe Kruchen's关于“后现代”设计的有趣讨论感兴趣[Kru05a]。

每个软件构件时进行。

体系结构设计定义了软件的主要构造元素之间的关系、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——可以从需求模型导出。

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。接口意味着信息流（如数据和（或）控制）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构造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作为构件设计的基础。

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。在软件工程中，设计是质量形成的地方，设计提供了可以用于质量评估的软件表示，设计是将相关方需求准确地转化为最终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，将会存在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。

ADVICE

软件设计总应该先考虑数据，数据是所有其他设计元素的基础。基础奠定之后，才能进行体系结构设计。之后，才能进行其他设计任务。

"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没有明显的不足。第一种方式更为困难。"——C.A.R Hoare

SAFEHOME

设计与编码

[场景] Jamie的房间，团队成员准备将需求转化为设计。

[人物] Jamie、Vinod和Ed，SafeHome软件工程团队所有成员。

[对话]

Jamie：大家都知道，Doug[团队管理者]沉迷于设计。老实说，我真正喜欢的是编码。如果给我C++或者Java，我会非常高兴。

Ed：不，你喜欢设计。

Jamie：你没听我说吗？编码才是我喜欢的。

Vinod：我想Ed的意思是你并不是真的喜欢编码，而是喜欢设计，并喜欢用代码表达设计。代码是你用来表示设计的语言。

Jamie：那有问题吗？

Vinod：抽象层。

Jamie：嗯？

Ed：程序设计语言有利于表示诸如数据结构和算法的细节，但不利于表示体系结构或者构件之间的协作……就是这个意思。

Vinod：一个糟糕的体系结构甚至能够摧毁最好的代码。

Jamie(思考片刻)：那么，你们的意思是我不能用代码表示体系结构……这不是事实。

Vinod：你肯定能在代码中隐含体系结构，但在大部分程序设计语言中，通过检查代码而快速看到体系结构的全貌是相当困难的。

Ed：那正是我们在开始编码之前需要的。

Jamie：我同意，也许设计和编码不同，但我仍然更喜欢编码。

8.2 设计过程

软件设计是一个迭代的过程，通过这个过程，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。

8.2.1 软件质量指导原则和属性

“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

在整个设计过程中，使用第15章中讨论的一系列技术评审来评估设计演化的质量。McGlaughlin[McG91]提出了可以指导良好设计演化的3个特征：

- 设计必须实现所有包含在需求模型中的明确需求，而且必须满足利益相关者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？

质量指导原则。为了评估某个设计表示的质量，软件团队中的成员必须建立良好设计的技术标准。在8.3节中，我们将讨论设计概念，这些概念也可以作为软件质量的标准。现在，考虑下面的指导原则：

良好设计的特征是什么？

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构成（将在本章后面讨论）；(c) 能够以演化的方式实现[⊖]，从而便于实现和测试。
2. 设计应该模块化；也就是说，应将软件逻辑地划分为元素或子系统。
3. 设计应该包含数据、体系结构、接口和构件的清晰表示。
4. 设计应导出数据结构，这些数据结构适用于要实现的类，并从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。

INFO

评估设计质量——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[⊜]——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呢？此时，不可能去测试软件，因为还没有可执行的软件，那怎么办？

在设计阶段，通过开展一系列的技术评审（Technical Review，TR）来评估质量。TR将在第15章[⊜]中进行详细的讨论，这里先概述一下该技术。技术评审是由软件团队成员召开的

⊖ 对于更小的系统，设计有时可以线性地开发。

⊜ 第23章讨论的质量因素可以帮助评审小组评估质量。

⊜ 在设计时你可以回顾一下第15章。技术评审是设计过程中的关键部分，也是达到设计质量的重要方法。

会议。通常，根据将要评审的设计信息的范围，选择2人、3人或4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，评审小组的每个成员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。目的是在会议开始时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。TR通常持续90分钟到两个小时。在TR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。

"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]制订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：

- 功能性通过评估程序的特征集和能力、所提交功能的通用性以及整个系统的安全性来评估。
- 易用性通过考虑人员因素（第11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、平均故障时间（Mean-Time-To-Failure，MTTF）、故障恢复能力和程序的可预见性来评估。
- 性能通过考虑处理速度、响应时间、资源消耗、吞吐量和效率来度量。
- 可支持性综合了扩展程序的能力（可扩展性）、适应性和耐用性这3方面的能力（这3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22章）、系统安装的简易性和问题定位的容易性。

ADVICE
软件设计师往往注重于问题的解决。不要忘记的是：FURPS属性总是问题的一部分，因此必须要考虑到这些属性。

在进行软件设计时，并不是每个软件质量属性都具有相同的权重。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构造已经开始时才考虑。

8.2.2 软件设计的演化

"设计师知道当设计中不再需要减去任何东西，而并不是不再需要增加任何东西的时候，他的设计就很完美了。"
——Antoine de St-Expurey

软件设计的演化是一个持续的过程，它已经经历了近60年的发展。早期的设计工作注重于模块化程序开发的标准[Den73]和以自顶向下的方式对软件结构进行求精的方法[Wir71]。设计定义的过程方面演化为一种称之为结构化程序设计的方法[Dah72]，[Mil72]。后来的工作提出了将数据流[Ste74]或数据结构(如，[Jac75]，[War74])转化为设计定义的方法。较新的设计方法（如[Jac92],[Gam95]）提出了进行设计导出的面向对象方法。近年来，软件体系结构[Kru06]和可用于实施软件体系结构及较低级别的设计抽象(如[Hol06][Sha05])的设计模式已经成为软件设计的重点。面向方面的方法(如，[Cla95]，[Jac04])、模型驱动开发[Sch06]以及测试驱动开发[Ast04]日益受到重视，这些方法强调在设计中获得更有效的模块化和体系结构的技术。

? 所有设计方法的共同设计特征是什么？

许多设计方法刚刚被提出，它们从工作中产生，并正被运用于整个行业。正如第6章和第7章提出的分析方法，每一种软件设计方法介绍了独特的启发式和表示法，同时也引入了某种标定软件质量特征的狭隘观点。不过，所有

的这些方法都有一些共同的特征：(1) 将需求模型转化为设计表示的方法；(2) 表示功能性构件及他们之间接口的表示法；(3) 精化和分割的启发式方法；(4) 质量评估的指导原则。

不管使用哪种设计方法，都应该将一套基本概念运用到数据设计、体系结构设计、接口设计和构件级设计，这些基本概念将在后面几节中介绍。

TASK SET

通用设计任务集

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，选择一种适用于软件的体系结构风格。
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：
   确保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：
   将每个分析类描述转化为设计类。
   根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，并根据需要进行修改。
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。

8.3 设计概念

在软件工程的历史上，产生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。每一种方法可以回答下面的问题：

- 使用什么标准将软件分割为独立的构件？
- 功能和数据结构细节如何从软件的概念表示中分离出来？
- 定义软件设计技术质量的统一标准是什么？

M. A. Jackson[Jac75]曾经说过："软件工程师的智慧开始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。

在后面几节中，简要介绍传统软件开发和面向对象软件开发中重要的软件设计概念。

8.3.1 抽象

“抽象是人类处理复杂问题的基本方法之一。”——Grady Boach

当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。当力图陈述一种解决方案时，面向问题的术语和面向实现的术语会同时使用。最后，在最低的抽象层次上，解决方案将以一种能直接实现的方式得到陈述。

ADVICE

作为设计师，致力于得到解决现有问题的过程抽象和数据抽象，但如果他们能在整个问题域中起作用，就更好了。

当开发不同层次的抽象时，软件设计师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，从移动门走开等）[⊖]。

数据抽象是描述数据对象的冠名数据集合。在过程抽象“开”的场景下，我们可以定义一个名为door的数据抽象。同任何数据对象一样，door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、开门方法、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。

8.3.2 体系结构

WebRef

关于软件体系结构深入的讨论可以在www.sei.cmu.edu/ata/ata_init.html找到。

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，构件可以概括为表示主要的系统元素及其交互。

软件设计的目标之一是导出系统的体系结构示意图，该示意图作为一个框架，将指导更详细的设计活动。一系列的体系结构模式使软件工程师能够解决常见的设计问题。

“软件体系结构是在质量、进度和成本方面具有最高投资回报的工作产品。”——Len Bass等

Shaw和Garlan[Sha95a]描述了一组属性，这组属性应该指定为体系结构设计的一部分。

结构特性。体系结构设计表示定义了系统的构件（如模块、对象、过滤器）、构件被封装的方式以及构件之间相互作用的方式。例如，对象封装了数据和过程，过程操纵数据并通过方法调用进行交互。

ADVICE

如果要做，就不仅要设计体系结构，还要尽力使项目其他部分的开销与设计开销相适宜，应该明确地设计体系结构。

外部功能特性。体系结构设计描述应当指出设计体系结构如何满足需求，这些需求包括：性能需求、能力需求、可靠性需求、安全性需求、可适应性需求以及其他系统特征需求。

相关系统族。体系结构应当能抽取出在一类相似系统开发中经常遇到的重复性模式。本质上，设计应当能够重用体系结构构件。

一旦给出了这些特性的规格说明，体系结构设计就可以用一种或多种不同的模型来表示[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

⊖ 然而，需要注意的是：只要过程抽象隐含的功能相同，一组操作可以被另一组操作代替。因此，如果门是自动的、并连接到传感器上，那么实现“开”所需的步骤将会完全不同。

"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，而根本不需要用同样的方式重复工作两次。"
——Christopher Alexander

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。

已经开发了许多不同的体系结构描述语言（architectural description languages，ADL）来表示上述描述的模型［Sha95b］。尽管已经提出了许多不同的ADL，但大多数ADL都提供描述系统构件和构件之间相互联系方式的机制。

需要注意的是，关于体系结构在设计中的地位还存在一些争论。一些研究者主张将软件体系结构设计从设计中分离出来，并在需求工程活动和更传统的设计活动之间进行。另外一些研究者认为体系结构设计是设计过程不可分割的一部分。在第9章中将讨论确定软件体系结构特征的方式及软件体系结构在设计中的作用。

8.3.3 模式

Brad Appleton以如下方式定义设计模式："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。"［App00］。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）模式是否适用于当前的工作；（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的、但是功能或结构不同的模式。设计模式将在第12章进行详细讨论。

8.3.4 关注点分离

关注点分离是一个设计概念[Dij82]，它表明任何复杂问题如果被分解为可以独立解决和（或）优化的若干块，该复杂问题能够更容易地被处理。一个关注点是一个特征或行为，被指定为软件需求模型的一部分。通过将关注点分割为更小的关注点（由此产生更多可管理的块），使得解决一个问题需要付出更少的工作量和时间。

关注点分离可以进行非常深入的探讨。如果你把一个问题分解为过多非常小的问题，求解每一个问题很容易，但将它们集成到一起会非常困难。

考虑两个问题，$p1$和$p2$，如果$p1$的认知复杂度高于$p2$的认知复杂度，结果是：求解$p1$所需的工作量就会大于求解$p2$所需的工作量。一般情况下，这种结果直观上是显而易见的，因为求解复杂问题确实需要更多的时间。

另一个结果是：两个问题被结合到一起的认知复杂度经常会高于每个问题各自的认知复杂度之和，这就引出了"分而治之"的策略——把一个复杂问题分解为若干可管理的块来求解时将会更容易。这对于软件模块化具有重要的意义。

关注点分离在其他相关设计概念中也有体现：模块化、方面、功能独立、求精。每个概念都会在后面的小节中讨论。

8.3.5 模块化

模块化是关注点分离最常见的表现。软件被划分为独立命名的、可处理的构件，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"模块化是软件的单一属性，它使程序能被智能化地管理"［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）。对于单块大型程序，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。几乎所有的情况下，为了理解更容易，都应当将设计划分成许多模块，这样做的结果，构建软件所需

的成本将会随之降低。

回顾我们关于关注点分离的讨论，可以得出结论：如果我们无限制地划分软件，那么开发软件所需的工作量将会小到忽略不计！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8-2所示，开发单个软件模块的工作量（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。这些特性形成了图示中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，我们缺乏成熟的技术来精确地预测M。

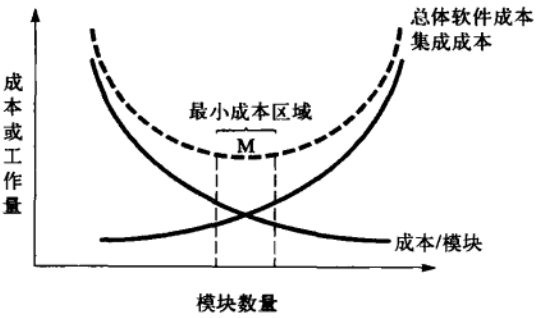


图8-2 模块化和软件成本

对于一个给定的系统，合适的模块数量是多少？

当考虑模块化的时候，图8-2所示的曲线确实提供了有益的指导。应该进行模块化，但是应注意保持在M附近。应该避免不足的模块化或过度的模块化问题。但是如何知道M的附近在哪里呢？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。

模块化设计（以及由其产生的程序）使开发工作更易于规划；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；可以进行长期维护而没有严重的副作用。

8.3.6 信息隐蔽

模块化概念让你面对一个基本问题："我们将如何分解一个软件解决方案以获得最好的模块集合？"信息隐蔽原则[Par72]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，模块应该规定并设计成为在模块中包含的信息（算法和数据）不被不需要这些信息的其他模块访问。

KEY POINT
信息隐蔽的目的是将数据结构和处理过程的细节隐蔽在模块接口之后。用户不需要了解模块内部的具体细节。

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。隐蔽定义并加强了对模块内过程细节的访问约束和对模块所使用的任何局部数据结构的访问约束[Ros75]。

将信息隐蔽作为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需要进行修改时，可提供最大的益处。由于大多数数据和过程对软件的其他部分是隐蔽的，因此，在修改过程中不小心引入的错误不太可能传播到软件的其他地方。

8.3.7 功能独立

功能独立的概念是关注点分离、模块化、抽象和信息隐蔽概念的直接产物。在关于软件设计的一篇里程碑性的文章中，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]等人在其后又巩固了这一概念。

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。换句话说，软件设计时应使每个模块仅涉及需求的某个特定子集，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。人们会提出一个很合理的问题是：独立性为什么如此重要。

为什么我们总是努力构造独立模块?

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，功能独立是良好设计的关键，而设计又是软件质量的关键。

KEY POINT
内聚性是一个模块侧重于一件事情的程度的定性指标。

独立性可以通过两条定性的标准进行评估：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8.3.6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。即使我们总是争取高内聚性（即专一性），一个软件构件执行多项功能也经常是必要的和可取的。然而，为了实现良好的设计，应该避免“分裂型”构件（执行多个无关功能的构件）。

KEY POINT
耦合性是一个模块和其他模块及外部世界连接程度的定性指标。

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进行传递。在软件设计中，应当尽力得到最低可能的耦合。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]。当在某个地方发生错误并传播到整个系统时，就会引起“涟漪效果”。

8.3.8 求精

ADVICE
有一种趋势是直接进行全面详细的设计，而跳过精化步骤，这将导致错误和遗漏，并使得设计更难于评审。因此，一定要实施逐步求精。

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。通过连续精化过程细节层次来实现程序的开发，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来进行层次开发。

求精实际上是一个细化的过程。该过程从高抽象级上定义的功能陈述（或信息描述）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的工作或信息内部的结构。可以在原始陈述上进行细化，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。抽象能够明确说明内部过程和数据，但对“外部使用者”隐藏了低层细节；精化有助于在设计过程中揭示低层细节。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。

8.3.9 方面

“如果不定期浏览一下封面，确保它不是一本关于软件设计的书的话，那么阅读一本关于魔术原理的书来了解书的思想将会非常困难。”——Bruce Tognazzini

当我们开始进行需求分析时，一组“关注点”就出现了。这些关注点“包括需求、用例、特征、数据结构、服务质量问题、变量、知识产权边界、合作、模式以及合同”[Aos07]。理想情况下，可以按某种方式组织需求模型，该方式允许分离每个关注点（需求），使得能够独立考虑每个关注点（即需求）。然而实际上，某些关注点跨越了整个系统，从而很难进行分割。

当开始进行设计时，需求被精化为模块设计表示。考虑两个需求，*A*和*B*。“如果已经选择了一种软件分解[精化]，在这种分解中，如果不考虑需求*A*的话，需求*B*就不能得到满足”[Ros04]，那么需求*A*横切需求*B*。

例如，考虑SafeHomeAssured.com网站应用中的两个需求。用第6章中讨论的用例ACS-DCV描述需求*A*，设计求精将集中于那些能够使注册用户通过放置在空间中的相机访问视频的模块。需求*B*是一个通用的安全需求，要求注册

KEY POINT
横切关注点是系统的某个特征，它适用于许多不同的需求。

用户在使用SafeHomeAssured.com之前必须先进行验证，该需求用于SafeHome注册用户可使用的所有功能中。当设计求精开始的时候，*A**是需求*A*的一个设计表示，*B**是需求*B*的一个设计表示。因此，*A**和*B**是关注点的表示，且*B**横切*A**。

方面是一个横切关注点的表示，因此，需求“注册用户在使用SafeHomeAssured.com之前必须先进行验证”的设计表示*B**是SafeHome 网站应用的一个方面。标识方面很重要，以便于在开始求精和模块化的时候，设计能够很好地适应这些方面。在理想情况下，一个方面作为一个独立的模块(构件)进行实施，而不是作为“分散的”或者和许多构件“纠缠的”软件片断进行实施[Ban06]。为了做到这一点，设计体系结构应当支持定义一个方面，该方面即一个模块，该模块能够使该关注点经过它横切的所有其他关注点而得到实施。

8.3.10 重构

WebRef
重构的优秀资源可以在网站www.refactoring.com找到。

很多敏捷方法（第3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[Fow00]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。”

WebRef
大量重构模式可以在网站http://c2.com/cgi/wiki?RefactoringPatterns找到。

当重构软件时，检查现有设计的冗余性、没有使用的设计元素、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）。在仔细思考之后，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。其结果是软件更容易集成、测试和维护。

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；还有Shakira，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，从没有真正理解信息隐藏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、隐藏、模式……明白。

Jamie：我至今还记得我上的第一节编程课……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。

Vinod：我以前从未听说过的概念是“方面”和“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，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。

8.3.11 面向对象的设计概念

面向对象（object-oriented, OO）范型广泛应用于现代软件工程。附录2是为那些不熟悉面向对象概念（如，类、对象、继承、消息和多态等）的读者提供的。

8.3.12 设计类

需求模型定义了一组分析类（第6章），每一个分析类都描述问题域中的某些元素，这些元素关注用户可见的问题方面。分析类的抽象级相对较高。

在设计模式演化时，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）实现支持业务解决方案的软件基础设施。[Amb01]建议了五种不同类型的设计类，每一种都表示了设计体系结构的一个不同层次：

设计者创建哪些类型的类？

- 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可视化表示。
- 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：实现完整的管理业务域类所必需的低层业务抽象。
- 持久类：表示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。

随着体系结构的形成，每个分析类转化为设计表示，抽象级就降低了。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，用于指导实现。

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4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可以合理预见到的（根据对类名的理解）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。和一个设计类相关的方法应该关注于实现类的某一个服务。一旦服务已经被某个方法实现，类就不应该再提供完成同一事情的另外一种方法。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（每一个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（Law of Demeter）[Lie03]，该定律建议一个方法应该只向周边类中的方法发送消息[⊖]。

SAFEHOME

将分析类精化为设计类

[场景] Ed的房间，开始设计建模。

[人物] Vinod和Ed，SafeHome软件工程团队成员。

[对话]

（Ed正进行FloorPlan类的设计工作（参考6.5.3节的讨论以及图6-10），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，我们要对设计进行精化，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（一种特定的数据结构）。像这样……我必须精化分析类FloorPlan（图6-10），实际上，是它的一种简化。

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是段（即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8-3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（即聚集）中添加或删除，而不会有任何问题。

Ed（点头）：是的，我认为这样是可以的。

Vinod：我也赞同。

⊖ Demeter定律的一种非正式表述是“每个单元应该只和它的朋友谈话，不要和陌生人谈话。”

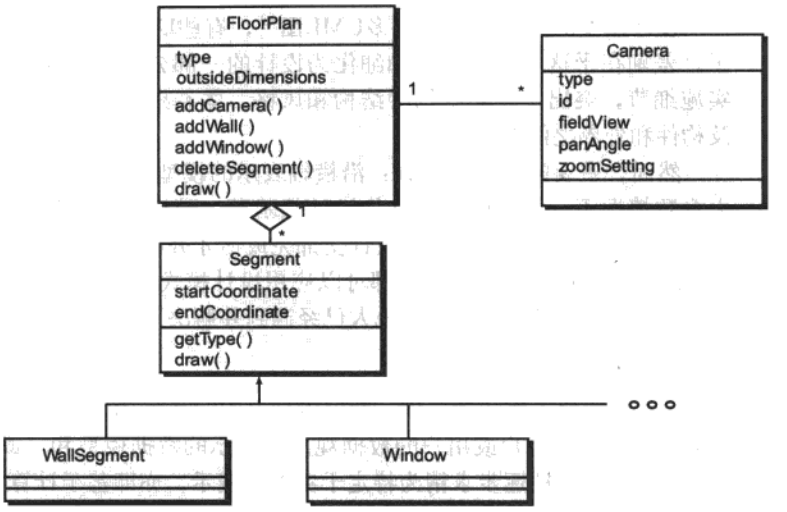


图8-3 FloorPlan的设计类和类的复合聚集（见补充讨论）

8.4 设计模型

可以从两个不同的维度观察设计模型，如图8-4所示。过程维度表示设计模型的演化，设计任务作为软件过程的一部分被执行。抽象维度表示详细级别，分析模型的每个元素转化为一个等价的设计，然后迭代求精。参考图8-4，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明显的差别；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。

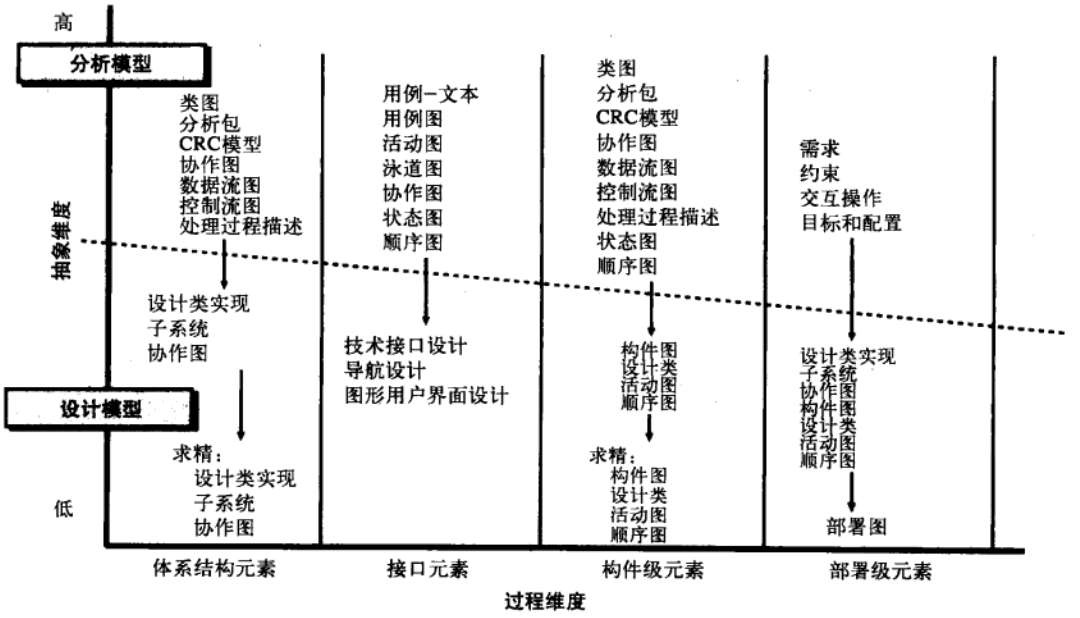


图8-4 设计模型的维度

KEY POINT
设计模型有4个主要元素：数据、体系结构、构件和接口。

设计模型的元素使用了很多UML图[⊖]，有些UML图在分析模型中也使用了。差别在于这些图被求精和细化为设计的一部分，并且提供了更多具体的实施细节，突出了体系结构的结构和风格、体系结构中的构件、构件之间以及构件和外界之间的接口。

然而，重要的是要注意到：沿横轴表示的模型元素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。

在设计过程中的任何地方都可以应用设计模式（第12章）。这些模式能够使设计人员将设计知识应用到他人已经遇到并解决了的特定领域问题。

"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

8.4.1 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）创建在高抽象级上（以客户或用户的数据观点）表示的数据模型和（或）信息模型。之后，数据模型被逐步求精为特定于实现的表示，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，数据结构设计以及处理这些数据的相关算法对于创建高质量的应用程序是至关重要的。在应用级，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。在每一种情况下，数据设计都发挥了重要作用。在第9章，将更详细地讨论数据设计。

KEY POINT
在体系结构（应用）级，数据设计关注于文件或数据库；在构件级，数据设计考虑实现局部数据对象所需的数据结构。

8.4.2 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[Sha96]从以下3个来源获得：（1）关于将要构建的软件的应用域信息；（2）特定的需求模型元素，如数据流图或分析类、现有问题中它们的关系和协作；（3）体系结构风格（第9章）和模式（第12章）的可获得性。

体系结构设计元素通常描述为一组相互联系的子系统，且常常从需求模型中的分析包中派生出来。每个子系统有其自己的体系结构（如，图形用户接口可能根据之前存在的用户接口体系结构进行了结构化）。体系结构模型特定元素的导出技术将在第9章中介绍。

8.4.3 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

⊖ 附录I提供了基本UML概念和符号的使用手册

"与良好的设计相比，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。"——aul Rand

的方式和平面图上描绘的在室内的分布方式。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）作为平面图的一部分，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。软件接口设计元素描述了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。

接口设计有3个重要的元素：（1）用户界面（user interface，UI）；（2）和其他系统、设备、网络或其他信息生成者或使用者的外部接口；（3）各种设计构件之间的内部接口。这些接口设计元素能够使软件进行外部通信，还能使软件体系结构中的构件之间进行内部通信和协作。

UI设计（越来越多地被称作可用性设计）是软件工程中的主要活动，这会在第11章中详细地考虑。可用性设计包含美学元素（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是整个应用体系结构内独一无二的子系统。

KEY POINT
接口设计元素有三部分：用户接口、系统和外部应用的接口、应用系统内部构件之间的接口。

外部接口设计需要关于发送和接收信息的实体的确定信息。在所有情况下，这些信息都要在需求工程（第5章）中收集，并且一旦开始进行接口设计，还要检验这些信息[⊖]。外部接口设计应包括错误检查和（在必要时）适当的安全特征检查。

"现在的每一样东西以后都会消失。稍微放松一下，再返回到你的工作中，你的判断将更可靠。因为离开一定距离，工作看起来更小，更容易远瞰，更容易发现和谐和比例的缺失。"——Leonardo DaVinci

内部接口设计和构件级设计（第10章）紧密相关。分析类的设计实现表示了所有操作和消息传递模式，使得不同类的操作之间能够进行通信和协作。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操作的特定功能需求。如果选择经典的"输入-处理-输出"设计方法，每个软件构件接口就基于数据流表示和处理过程说明中描述的功能进行设计。

在有些情况下，接口建模的方式和类所用的方式几乎一样。在UML中，接口如下定义[OMG03a]："接口是类、构件或其他分类符（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，控制面板允许户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8-5）提供了和键盘相关的行为，因此必须实现操作readKeyStroke()和decodeKey()。如果这些操作提供给其他类（在此例中是WirelessPDA和MobilePhone），定义如图8-5所示的接口是非常有用的。名为KeyPad的接口表示为<<interface>>构造型（stereotype），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。

WebRef
有关UI设计非常有用的信息可以在www.useit.com找到。

在控制面板的右边带有三角箭头的虚线（图8-5）表示ControlPanel类提供了KeyPad操作作为其行为的一部分。在UML中，这被称为实现。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

⊖ 接口特征可能随时间变化。因此，设计者应当确保接口的规格说明是准确且完整的。

给那些访问该接口的其他类。

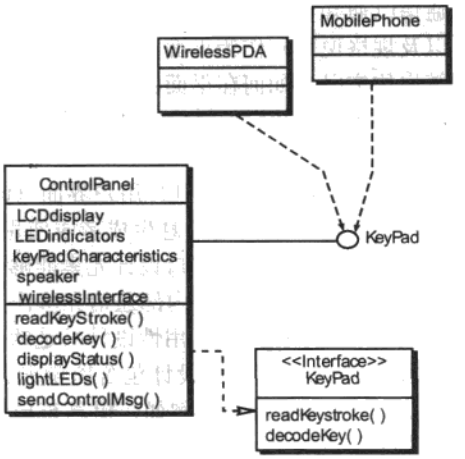


图8-5 ControlPanel的接口表示

"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

8.4.4 构件级设计元素

软件的构件级设计相当于一个房屋中每个房间的一组详图（以及规格说明）。这些图描绘了每个房间内的布线和管道、电器插座和墙上开关、水龙头、水池、淋浴、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，构件级设计为所有局部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。

"细节并不仅仅是细节，细节构成设计。" —— Charles Eames

在面向对象的软件工程中，使用UML图表现的一个构件如图8-6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。

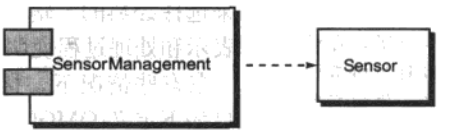


图8-6 UML构件图

SensorManagement构件完成所有和SafeHome传感器相关的功能，包括监测和配置传感器。第10章将进一步讨论构件图。

构件的设计细节可以在很多不同的抽象级别建模。UML活动图可用来表示处理逻辑，构件详细的过程流可以使用伪代码表示（类编程语言的表示方法，在第10章讨论），也可以使用一些图形（例如流程图或盒图）来表示。算法结构遵守结构化编程的规则（即一套约束程序构造）。对于基于待处理的数据对象的特性所选择的数据结构，在实现中通常使用伪代码或程序语言建模。

8.4.5 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产品元素被配置在3种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8-7所示。图中显示了3种计算环境（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

KEY POINT
部署图刚开始使用描述符形式，粗略描述部署环境。后来使用实例形式，明确描述配置的元素。

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。每个子系统需要进行细化，用以说明该子系统所实现的构件。

图8-7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Mac PC还是一台“基于windows”的PC、Sun工作站或Linux系统。在后面的阶段或构建开始时，需要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）。

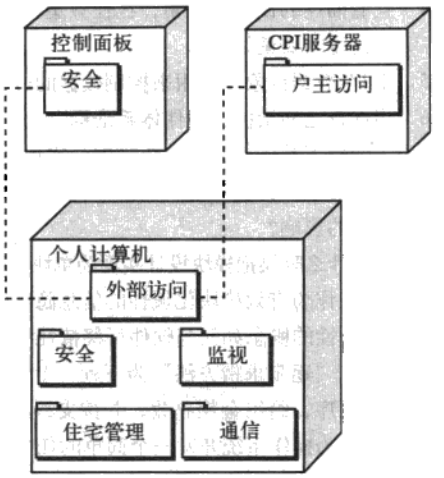


图8-7 UML 部署图

8.5 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开发的原则、概念和实践。设计的目标是创建软件模型，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。软件设计人员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利益相关者需要的解决方案。

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。

设计概念在软件工程开始的60年不断发展。这些概念描述了计算机软件的属性（描述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所使用的编程语言。实质上，设计概念强调了：(1) 抽象的必要性，它提供了一种创造可重用软件构件的方法；(2) 体系结构的重要性，它使得能够更好地理解系统整体结构；(3) 基于模式的工程的有益性，它是一项用于已证明能力的软件的设计技术；(4) 关注点分离和有效的模块化的价值，它们使得软件更容易理解、更容易测试以及更容易维护；(5) 信息隐藏的直接作用，当错误发生时，它能够减少负面影响的传播；(6) 功能独立的影响，它是构造有效模块的标准；(7) 求精作为一种设计方法的作用；(8) 横切系统需求方面的考虑；(9) 重构的应用，它是为了优化已导出的设计；(10) 面向对象的类和与类相关特征的重要性。

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、需求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。

习题与思考题

8.1 当你“编写”程序时设计软件吗？软件设计和编码有什么不同吗？

8.2 如果软件设计不是程序（它肯定不是），那么它是什么？

8.3 我们如何评估软件设计的质量?

8.4 查看描述的设计任务集，在任务集中的哪里对质量进行评估？评估是如何完成的？如何达到8.2.1节中讨论的质量属性?

8.5 举3个数据抽象和能用来控制数据的过程抽象的例子。

8.6 用你自己的话描述软件体系结构。

8.7 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，简要地描述该模式。

8.8 用你自己的语言描述关注点分离。分而治之的方法有时候是否不合适？这种情况对模块化的观点有多大的影响?

8.9 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?

8.10 讨论作为有效模块化属性的信息隐蔽概念和模块独立性概念之间的联系。

8.11 耦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。

8.12 应用“逐步求精方法”为下列一个或多个程序开发3种不同级别的过程抽象：(a) 开发一个支票打印程序，给出金额总数，并按支票的常规要求给出大写金额数；(b) 为某个超越方程迭代求解；(c) 为操作系统开发一个简单的任务调度算法。

8.13 考虑需要实现汽车导航功能（GPS）、手持通信设备的软件。描述两三个要表示的横切关注点。讨论你如何将其中一个关注点作为方面来表示。

8.14 “重构”意味着迭代地修改整个设计吗？如果不是，它意味着什么?

8.15 简要描述设计模型的四个元素。

推荐读物与阅读信息

Donald Norman编写了两本书（《The Design of Everyday Things》, Doubleday, 1990；《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，任何人想设计人类使用的任何东西，都“必须”阅读这些著作。Adams的著作（《Conceptual Blockbusting》，第3版, Addison-Wesley, 1986）对那些希望拓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》, 2nd ed., 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）讨论了成功与不成功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）理解最终用户是如何工作的，然后设计满足用户需要的软件。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。Bain（《Emergent Design》, Addison-Wesley, 2008）将模式、重构和测试驱动的开发方法结合到有效的设计方法中。

Fox（《Introduction to software Engineering Design》, Addison-Wesley, 2006）和Zhu（《Software Design Methodology》, Butterworth-Heinemann, 2005）介绍了软件工程中设计的综合性处理。McConnell（《Code Complete》,第2版, Microsoft Press, 2004）对高质量软件设计的实践方面进行了精彩的论述。Robertson（《Simple Program Design》, 3rd ed. Boyd and Fraser Publishing, 1999）介绍性论述了软件设计，这对初学者很有帮助。Budgen（《Software Design》, 第2版, Addison-Wesley, 2004）介绍了大量流行的软件设计方法，并对它们进行了对比和对照。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。Rosenberg和Stevens（《Use Case Driven Object Modeling with UML》, Apress, 2007）讨论了以用例为基础的面向对象的设计开发。

从Free-man和Wasserman（《Software Design Techniques》，第4版，IEEE，1983）编辑的文集中，可以一览软件设计精彩的发展历史。这本教程中转载了许多经典的论文，这些论文已经奠定了软件设计当前发展趋势的基础。Card和Glass（《Measuring Software Design Quality》，Prentice-Hall，1990）从技术和管理两个角度介绍并思考了软件质量的度量。

关于软件设计的信息在Internet上有大量的各种资源。在SEPA网站www.mhhe.com/ engcs/compsci/pressman/professional/olc/ser.htm上可以找到和软件设计及设计工程相关的最新WWW参考信息。

第9章

Software Engineering：A Practitioner's Approach (Seventh Edition)

体系结构设计

要点浏览

概念：体系结构设计表示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和性质，以及系统中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，这项工作往往由专家来完成。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"根据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：没有图纸就不要企图盖房子，不是吗？同样，也不能通过勾画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"宏观"视图，即房子本身。这就是体系结构设计需要做的事情——它为你提供"宏观"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，得出最适合于客户需求和质量属性的结构。方案一旦选定，就需要使用体系结构设计方法对体系结构进行细化。

工作产品：在体系结构设计过程中，要创建一个包括数据体系结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关系（交互作用）。

质量保证措施：在每个阶段，都要对软件设计的工作产品进行评审，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

原型
体系结构描述语言
体系结构
候选方案
构件
复杂度
以数据为中心
数据流
设计
类型
分层
面向对象
模式
求精
风格
模板
ATAM
分解
实例化
映射

设计通常被描述为一个多步过程，该过程从信息需求中综合出数据和程序结构的表示、接口特征和过程细节。Freeman[Fre80]扩展了该描述：

> 设计是一项活动，它关注于如何做出重要决策，往往是结构性的。设计和编程都关注抽象信息表示和处理顺序，但是在详细程度上两者俨然不同。设计就是构建内聚的、良好规划的程序表示，它关注高层各部分之间的相互关系和低层所包括的逻辑操作。

正如第8章提到的，设计是由信息驱动的。软件设计方法是通过仔细考虑分析模型的3个域而得到的。数据、功能和行为3个域是创建软件设计的指南。

本章将介绍建立设计模型数据和体系结构层的"内聚的、良好规划的表示"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。

9.1 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，体系结构隐含了不同的内容——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。

今天，有效的软件体系结构及其明确的表示和设计已经成为软件工程领域的主导主题。

9.1.1 什么是体系结构

"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合。"——Jerrold Grochow

当你考虑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，会考虑物理结构的整体形状。但在实际中，体系结构还包含更多的方面。它是各种不同建筑构件集成为一个有机整体的方式；是建筑融入所在环境并与相邻的其他建筑相互吻合的方式；是建筑满足既定目标和满足主人要求的程度；是对结构的一种美学观感（建筑的视觉效果），以及纹理、颜色和材料结合在一起创建外观和内部"居住环境"的方式；是很多微小的细节——灯具、地板类型、壁挂布置等的设计。总而言之，它是艺术。

但体系结构也可以是其他的东西。它是"数以千计的或大或小的决定"[Tyr05]。其中一些决定是设计初期做出的，并可能会对所有的其他设计行为产生深刻的影响。另外一些决定一直推迟到后来才做出，因此消除了过分限制性的制约因素，而这些制约因素可能会导致拙劣的体系结构风格。

软件体系结构必须对系统结构以及数据和过程构件相互协作的方式进行建模。

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：

程序或计算系统的软件体系结构是指系统的一个或者多个结构，它包括软件构件、构件的外部可见属性以及它们之间的相互关系。

体系结构并非可运行的软件。确切地说，它是一种表达，使能够：（1）对设计在满足既定需求方面的有效性进行分析；（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的风险。

"体系结构忙中建，闲时悔。"——Barry Boehm

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

软件工程界的一些成员（如[Kaz03]）对导出软件体系结构（称为"体系结构设计"）和导出软件设计这两种行为作了区分。一位评论者指出：

WebRef

可以从以下地址在获得许多软件体系结构站点的可用链接：www2.umassd.edu/SECenter/SAResources.html。

"体系结构"和"设计"这两个术语之间有明显的不同。设计是体系结构的一个实例，类似于对象是类的实例。例如，考虑"客户机-服务器"体系结构，我们可以使用Java平台（Java EE）或者Microsoft平台（.NET 框架），选择基于该体系结构的多种不同的实现方式设计一个以网络为中心的软件系统。可见，对于同一个体系结构，可能会产生多种基于该体系结构的设计。因此，不能把"体系结构"和"设计"混为一谈。

尽管本书同意软件设计是特定软件体系结构的实例，但元素和结构作为体系结构的一部分，仍是每个设计的根本，这些设计是从特定的体系结构演化产生出来的。设计开始于体系结构的深思熟虑。

在本书中，软件体系结构的设计考虑了设计金字塔（图8-1）中的两个层次——数据设计和体系结构设计。在前面的讨论中，数据设计能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、属性和交互作用的表示。

9.1.2 体系结构为什么重要

"体系结构是那么重要，以至于无论这些人有多么聪明，都不得不把它留在手中。"——Scott Ambler

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3个关键原因：

- 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（利益相关者）开展交流。
- 体系结构突出了早期的设计决策，这些决策对随后所有的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]。

体系结构模型提供了系统的Gestalt视图，允许软件工程师在总体上审查它。

体系结构设计模型和包含在其中的体系结构模式都是可以传递的，也就是说，体系结构的类型、风格和模式（9.2～9.4节）可以被应用于其他系统的设计，并且表示了一组抽象，使得软件工程师能以可预见的方式描述体系结构。

9.1.3 体系结构描述

体系结构这个词的意义是什么，我们每个人都会有一种理解。实际上，对于不同的人群，它的意义也不同。因为，不同的利益相关者会从不同的角度理解体系结构，这个角度是由不同的关注点驱动的。这就意味着体系结构描述实际上是一组体现系统不同视图的工作产品。

你需要努力的是关注体系结构的表示，它将指导设计过程的其他方面。需要花些时间仔细地评审体系结构，这里的一个错误将会产生长期的负面影响。

例如，办公大楼的建筑师必须和许多不同的利益相关者一起工作。办公楼业主（一个利益相关者）关心的主要问题是确保大楼美观，以及提供充足的办公空间和基础设施，以保证它的盈利能力。因此，建筑师必须使用建筑视图进行描述，以解决业主关心的问题。建筑师使用的是建筑物的三维图纸（为了说明外观美感）和一组二维平面图（为了解决利益相关者关心的办公空间和基础设施问题）。

但是，办公楼还有许多其他利益相关者，包括结构钢制造商，他们将提供建筑骨架所需的钢材。结构钢制造商需要有关支持大楼的结构钢的详细建筑信息，包括工字钢梁的尺寸、连接、材料和许多其他详细信息。这些问题通过不同的工作产品来处理，这些工作产品显示了建筑物的不同视图。建筑的结构钢骨架的一张特定图纸（另一个视点）集中于结构钢制造商关注的许多问题中的一个。

软件系统的体系结构描述也必须展示出类似于上述指出的办公大楼的特征。Tyree和Akerman[Tyr05]注意到了这一点，他们写道："开发人员想要对设计进行明确、果断的指导；客户想要对必然发生的环境变化进行清晰的理解，以及确保体系结构将满足他们的业务需要；而体系结构设计师想要对体系结构的关键方面进行清晰而深入的理解。"这里每一个"想要"的东西都反映了不同视点上的不同视角。

IEEE计算机学会提出了IEEE-Std-1471-2000，密集型软件系统体系结构描述的推荐实践做法（《Recommended Practice for Architectural Description of Software-Intensive System》），[IEE00]，目标如下：（1）建立软件体系结构设计过程中使用的概念性框架和词汇表，（2）提

供表示体系结构描述的详细准则，(3) 鼓励良好的体系结构设计实践。

IEEE标准将体系结构描述（architectural description，AD）定义为“记录体系结构的产品集合”。该描述本身使用多视图来表述，这里的每个视图是“从一组[利益相关者]关注点的角度观察的整个系统的一种表示”。视图根据在视点中定义的规则和约定来创建——“构造和使用视图约定的规格说明”[IEE00]。有很多种不同的工作产品可用来开发软件体系结构的不同视图，本章后面部分将对这些工作产品进行讨论。

9.1.4 体系结构决策

开发的每个视图都作为体系结构描述的一部分，它涉及某个特定的利益相关者的关注点。为了开发每个视图（和作为整体的体系结构描述），系统体系结构设计师考虑多种可选方案，并最终就最能满足关注点的特定的体系结构特征做出决策。因此，可以将体系结构决策本身看做是体系结构的一种视图。通过理解做出体系结构决策的缘由可以深刻洞悉系统的结构以及系统与利益相关者关注点的一致性。

作为一名系统体系结构设计师，可以使用下面推荐的模板记录每个主要的决策。通过记录，设计师为他的工作提供了逻辑依据，并且还可以建立历史记录，该记录在需要进行设计修改时可能有用。

INFO

体系结构决策描述模板

每个主要的体系结构决策可以被记录在案，以便以后评审，评审由想要理解已提出的体系结构描述的利益相关者进行。这里给出的是Tyree和Ackerman[Tyr05]所提出模板的修改和缩略版本。

设计问题：描述将要解决的体系结构设计问题。

解决方案：陈述所选择的解决设计问题的方法。

分类：指定问题和解决方案陈述的分类（例如，数据设计、内容结构、构件结构、集成、介绍）。

假设：指出任何有助于制定决策的假设。

约束：指定任何有助于制定决策的环境约束（例如，技术标准、可用的模式、项目相关问题）。

候选方案：简要描述所考虑的体系结构设计候选方案，并描述为什么要摒弃这些方案。

争论：陈述你为什么选择了这种解决方案而不是其他的候选方案。

意义：指出制定决策对设计的影响。选择方案如何影响其他的体系结构设计问题？解决方案会在某种程度上约束设计吗？

相关决策：其他记录的决策和该决策有什么相关性？

相关关注点：其他需求和该决策有什么相关性？

工作产品：指出在体系结构描述中，决策会在哪里体现出来。

注释：参考可用来制定决策的其他团队的备忘录或文档。

9.2 体系结构类型

尽管体系结构设计的基本原则适用于所有类型的体系结构，但对于需要构建的结构，体系结构类型经常规定特定的体系结构方法。在体系结构设计环境中，类型（genre）隐含了在整个软件领域中的一个特定类别。在每种类别中，会有很多的子类别。例如，在建筑物类型中，

KEY POINT
许多不同的体系结构风格可以用于一种特定的类型（也称为应用领域）。

大致会有以下几种一般风格：房子、单元楼、公寓、办公楼、工厂厂房、仓库等。在每一种一般风格中，也会运用更多的具体风格（9.3节）。每种风格有一个结构，可以用一组可预测模式进行描述。

Grady Booch在他的《Handbook of Software Architecture》（软件体系结构手册）[Boo08]的改进版本中，提出了以下几种软件系统的体系结构类型：

- **人工智能**——模拟或扩大人类认知、运动或其他有机体过程的系统。
- **商业和非盈利的**——工商企业营运必要的系统。
- **通信**——提供用于数据传输和数据管理、数据的用户连接或者数据展示的基础设施的系统。
- **内容创作**——用来创建或管理文字或多媒体人造物品的系统。
- **设备**——与物理世界交互的系统，可以为个人提供某种有意义的服务。
- **娱乐与运动**——管理公众事件或者提供大众娱乐体验的系统。
- **金融**——为转账和理财以及其他安全事务提供基础设施的系统。
- **游戏**——为个人或群体提供娱乐体验的系统。
- **行政管理**——支持地方、州、联邦、全球或者其他政治实体的管理和运作方式的系统。
- **工业**——模拟或控制物理过程的系统。
- **法律**——支持法律的系统。
- **医疗**——诊断或治疗，或者有助于医学研究的系统。
- **军事**——用于商议、通信、指挥、控制和信息（C4I）的系统，也有用于进攻和防卫武器的系统。
- **操作系统**——位于硬件之上提供基本软件服务的系统。
- **平台**——位于操作系统之上提供高级服务的系统。
- **科学**——用于科学研究和应用的系统。
- **工具**——用来开发其他系统的系统。
- **运输**——控制水上、地面、空中或者太空交通工具的系统。
- **实用程序**——与其他软件交互作用的系统，可以提供某些有意义的服务。

"如果没有一个总的体系结构或者设计考虑就进行编程，就如同只带了一个手电筒去探索洞穴：你不知道你已经去过哪里，也不知道将要去哪里，甚至都不知道现在在哪里。"——Danny Thorpe

从体系结构设计的立场看，每一种类型表述了一个特有问题。例如，考虑一个游戏系统的软件体系结构。游戏系统有时称做沉浸式交互应用（immersive interactive application），它需要密集型算法的计算方法、成熟的计算机图形图像技术、流媒体数据源、通过常规或非常规输入进行的实时的交互操作以及许多其他专业知识。

Alexandre Francois[Fra03]提出了可应用于游戏环境的immersipresence[⊖]的软件体系结构，描述方式如下：

immersipresence的软件体系结构（Software Architecture for Immersipresence，SAI)是一种新的软件体系结构模型，用于设计、分析和实现执行一般数据流的分布式、异步并行处理的应用系统。SAI的目标是为算法的分布式实施和容易地将其集成到复杂系统提供通用框架……底层可扩展数据模型和混合（共享存储和信息传递）分布式异步并行处理模型允许自然和有效地处理一般数据流，并可以使用已有的库或本地代码。类型模块化使得分布式代码开发、测试和重用以及快速系统设计与集成、维护和演化更加便利。

SAI的详细讨论超出了本书的范围。然而，认识到这一点很重要：游戏系统类型可以用专

⊖ Francois使用术语immersipresence表示沉浸式交互应用。

门设计的强调游戏系统关注点的体系结构风格（9.3节）来陈述。如果有兴趣，请参阅[Fra03]。

9.3 体系结构风格

"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. Cheserton

当建筑师用短语"殖民式中厅"（center hall colonial）来描述某座房子时，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作为描述手段，将该房子和其他风格（例如，A框架、砖房、科特角式等）的房子区分开来。但更重要的是，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。

什么是体系结构风格？

为基于计算机的系统构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）完成系统需要的某种功能的一组构件（例如，数据库、计算模块）；（2）能使构件间实现"通信、合作和协调"的一组连接件；（3）定义构件如何集成为系统的约束；（4）语义模型，能使设计者通过分析系统组成成分的已知属性来理解系统的整体性质[Bas03]。

WebRef

基于属性的体系结构风格（ABAS）可用作软件体系结构的构造块。可从www.sei.cmu.edu/architecture/abas.html上获得相关信息。

一种体系结构风格就是施加在整个系统设计上的一种变换，目的是为系统的所有构件建立一个结构。在对已有体系结构再工程（第29章）时，一种体系结构风格的强制采用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）模式涉及的范围要小一些，它更多集中在体系结构的某一方面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（9.4节）倾向于在体系结构环境中处理特定的行为问题（例如，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建立整个系统结构的外形。在9.3.1节中，我们将讨论经常使用的软件体系结构风格和模式。

INFO

典型的体系结构

本质上，软件体系结构表示了一种结构。在该结构中，某个实体集（经常称做构件）通过一组已定义的关系（经常称做连接件）进行连接。无论是构件还是连接件，它们都与一组特性相关，这组特性使设计者能够区别使用的构件和连接件的类型。但是，哪种结构（构件、连接件和特性）可用来描述体系结构？Bass和Kazman[Bas03]给出了5种典型的基本体系结构：

功能结构。构件表示功能或处理实体，连接件表示接口，它提供"使用"构件或"传递数据到"构件的功能。特性描述构件的特征和接口的组织。

实现结构。"构件可以是包、类、对象、过程、函数、方法等所有在不同抽象层上打包的功能"[Bas03]。连接件包括传递数据和控制、共享数据、"使用"以及"是一个实例"等能力。特性关注于结构实现时的质量特征（例如，可维护性、可重用性）。

并发结构。构件表示"并发单元"，这些"并发单元"被组织为并行任务或线程。"关系[连接件]包括同步于、优先级高于、发送数据到、运行必须有、运行不能有。与结构相关的特性包括优先级、抢先占有以及执行时间"[Bas03]。

物理结构。物理结构类似于设计开发中的部署模型，构件是物理硬件，软件驻留在硬件上。连接件是硬件构件之间的接口，特性用来描述容量、带宽、性能和其他属性。

开发结构。该结构定义构件、工作产品以及软件工程过程中所需的其他信息源。连接件表示工作产品之间的关系，特性标识每项的特征。

以上每一种结构表示软件体系结构的不同视图，显示出进行建模和构建时对软件团队的有用信息。

9.3.1 体系结构风格的简单分类

"设计模式和风格的使用在工程学科中是非常普遍的。"
—— Mary Shaw 和 David Garlan

在过去的60年中，尽管已经创建了数百万的计算机系统，但是，绝大多数都可以归为少数几种体系结构风格之一：

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9-1描述了一种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，数据存储库是被动的，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"，当客户感兴趣的数据发生变化时，它将通知客户软件。

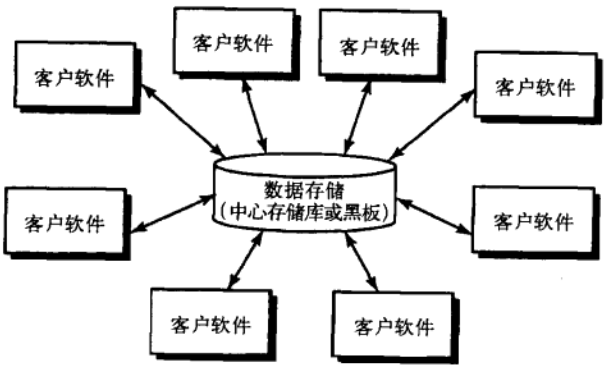


图9-1 以数据为中心的体系结构

以数据为中心的体系结构促进了可集成性（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体系结构之中，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。

数据流体系结构。当输入数据经过一系列的计算构件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-过滤器模式（图9-2）拥有一组称为过滤器的构件，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。

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）完成变换。

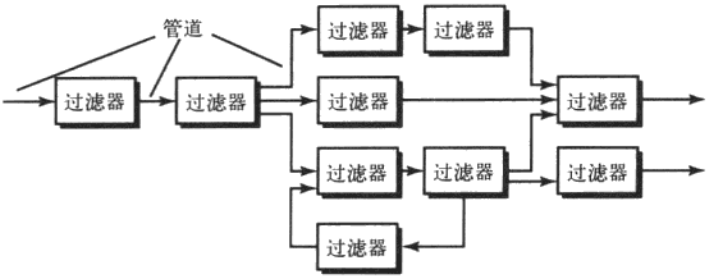


图9-2　数据流体系结构

调用和返回体系结构。该体系结构风格能够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，存在几种子风格[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其他构件。图9-3描述了该类型的体系结构。

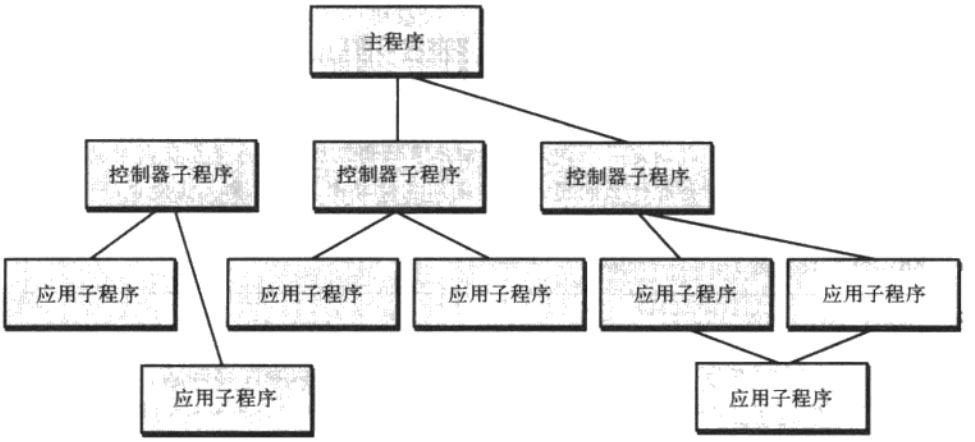


图9-3　主程序/子程序体系结构

- 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中的多台计算机上。

面向对象体系结构。系统的构件封装了数据和必须用于控制该数据的操作，构件间通过信息传递进行通信与合作。

层次体系结构。层次体系结构的基本结构如图9-4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，这些操作逐渐接近机器的指令集。在外层，构件完成建立用户界面的操作；在内层，构件完成建立操作系统接口的操作；中间层提供各种实用工具服务和应用软件功能。

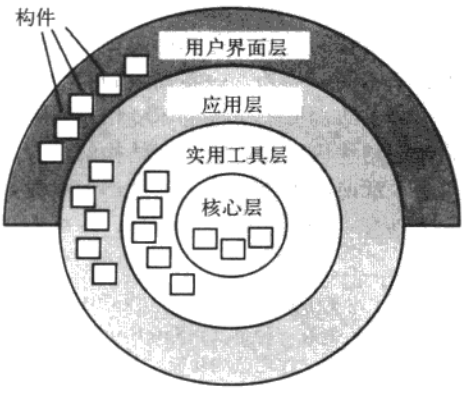


图9-4　层次体系结构

以上描述的体系结构风格仅仅是可用风格中的一小部分[⊖]。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和（或）风格的组合。在很多情况下，会有多种模式是适合的，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。

SAFEHOME

选择体系结构风格

[场景] Jamie的房间，设计建模还在继续进行。

[人物] Jamie和Ed，SafeHome软件工程团队的成员。

[对话]

Ed（皱着眉）：我们已经使用UML对安全功能进行了建模……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[⊖]应该是合适的方案。

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。

Jamie（微笑）：无组织？

Ed：是的……我的意思是说我不能想象出实际的结构，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对象设计的层次（聚集）[图8-3]。面向对象的体系结构是结构与类之间相互连接的组合，你知道，类之间的相互连接即相互协作。我们可以通过描述详细的类结构、属性、操作和类之间的消息来体现它。

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对此不会有什么问题，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：好，我知道了。

9.3.2 体系结构模式

"也许这是在地下室，让我们上楼去检查。"
——M.C.Escher

当开发需求模型时，你会注意到软件必须解决许多问题，这些问题很广泛，跨越了整个应用领域。例如，对于几乎每一个电子商务应用系统，其需求模型都会遇到下述问题：我们如何给各方面的客户提供不同类型的产品，并且允许那些客户在线购买我们的产品？

需求模型也定义了必须对上述问题进行回答的环境。例如，面向顾客销售高尔夫设备的电子商务和面向媒体和大中型公司销售高价工业设备的电子商务，它们的营运环境完全不同。另外，一系列限制和约束可能会影响到需要解决问题的处理方式。

体系结构模式在特定环境和一系列限制与约束下处理特定的应用问题。模式提出了能够作为体系结构设计基础的体系结构解决方案。

本章前面的部分曾提到过，大多数应用系统都符合特定领域或特定类型，适合于这种类型的风格有一种或者多种。例如，一个应用系统的整体体系结构风格可能是"调用和返回"或者

⊖ 参见[Bus07]、[Gor06]、[Roz05]、[Bas03]、[Bos00]或[Hof00]，其中有体系结构风格与模式的更详细讨论。

⊖ 可能有一种争议：SafeHome体系结构所处的层次应比记录的层次更高。SafeHome有许多不同的子系统——住宅监测功能、公司的监测网站以及运行在业主PC机上的子系统。在子系统中，并行过程（即那些监测传感器）和事件处理非常普遍。在产品工程过程中，要做出该层次上的某些体系结构决策，但在软件工程中，体系结构设计可能要考虑这些问题。

“面向对象”型，但在那种风格中，你会遇到一系列常见问题，这些问题最好是用具体的体系结构模式来处理。第12章将对这些问题中的一部分以及体系结构模式进行详细论述。

9.3.3 组织和求精

由于设计过程经常会留下许多种可供选择的体系结构方案，因此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[Bas03]有助于更深入地了解体系结构风格：

如何评估导出的体系结构风格？

- **控制**。在体系结构中如何管理控制？是否存在清楚的控制层次？如果存在，构件在控制层次中有什么作用？构件如何在系统中传递控制？构件间如何共享控制？控制的拓扑结构（即控制呈现的几何形状）如何？控制是否同步或者构件操作是否异步？
- **数据**。构件间如何进行数据通信？数据流是否是连续地传递给系统，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（即，数据是从一个构件传递到另一个构件，还是数据被系统中的构件全局共享）？是否存在数据构件（如黑板或中心存储库）？如果存在，它们的作用是什么？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的其他构件交互）？数据和控制如何在系统中交互？

这些问题有助于设计者对设计质量进行早期评估，也为更详细地分析体系结构奠定了基础。

9.4 体系结构设计

“医生可以掩盖他的错误，但是建筑师只能建议他的客户牵萝补屋。”
—— Frank Lloyd Wright

在体系结构设计开始的时候，待开发的软件必须放在所处的环境中，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。这些信息一般可以从需求模型中获得，而所有其他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，就可以确定体系结构原型集。原型（类似于类）是表示系统行为元素的一种抽象。这个原型集提供了一个抽象集，如果要对系统结构化，就必须要对这些原型进行结构化建模，但原型本身并不提供足够的实施细节。因此，设计人员通过定义和细化实施每个原型的软件构件来指定系统的结构。这个过程不停地迭代，直到获得一个完善的体系结构。下面几节中我们更详细地探讨体系结构设计中的每一项任务。

KEY POINT
体系结构环境表示软件如何与其外围实体进行交互。

9.4.1 系统环境的表示

系统相互之间如何进行交互操作？

在体系结构设计层，软件体系结构设计师用体系结构环境图（Architectural Context Diagram，ACD）对软件与其外围实体的交互方式进行建模。图9-5给出了体系结构环境图的一般结构。

根据图中所示，与目标系统（为该系统所开发的体系结构设计）交互的系统可以表示为：

- 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（即信息或者由同级系统和目标系统产生，或者被目标系统和同级系统使用）。
- 参与者——通过产生和消耗必要处理所需的信息，实现与目标系统交互的实体（人、设备）。

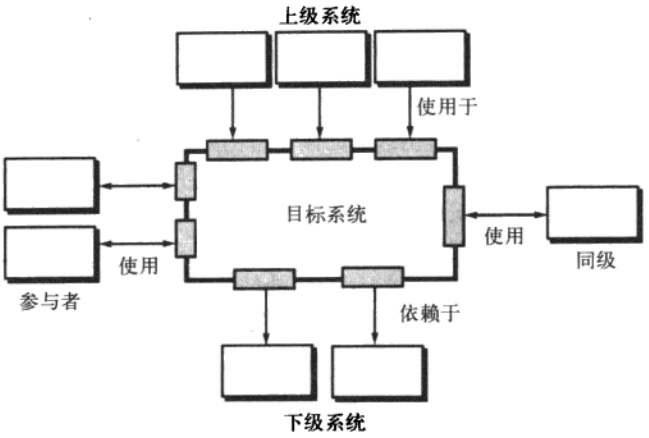


图9-5 体系结构环境图（摘自[Bos00]）

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。整个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9-6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使用者。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9-6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。

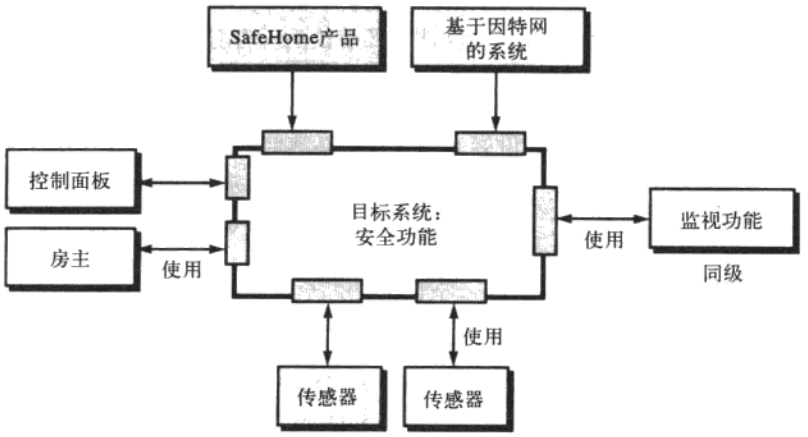


图9-6 SafeHome安全功能的体系结构环境图

9.4.2 定义原型

KEY POINT 原型是体系结构设计的抽象构造块。

原型（archetype）是表示核心抽象的类或模式，该抽象对于目标系统体系结构的设计非常关键。一般来讲，即使设计相对复杂的系统，也只需要相对较小的原型集合。目标系统的体系结构由这些原型组成，这些原型表示体系结构中稳定的元素，但可以基于系统行为以多种不同的方式对这些元素实例化。

很多情况下，可以通过检验作为需求模型一部分的分析类来获得原型。继续关于SafeHome住宅安全功能的讨论，可能会定义下面的原型：

- **结点**。表示住宅安全功能的输入和输出元素的内聚集合，例如，结点可能由如下元素构成：(1) 各种传感器；(2) 多种警报（输出）指示器。
- **探测器**。对所有为目标系统提供信息的传感设备的抽象。
- **指示器**。表示所有指示警报条件发生的报警机械装置（例如，警报汽笛、闪灯、响铃）的抽象。
- **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9-7显示了使用UML符号对每一个原型的描述结果。回想那些构成体系结构基础的原型，它们是抽象的，随着体系结构设计的进行，这些抽象必须被进一步求精。例如，**探测器**可以被精化为传感器的类层次。

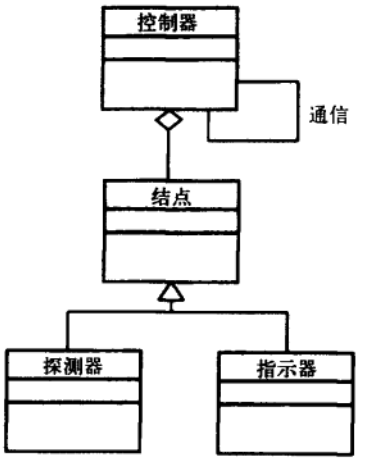


图9-7　SafeHome安全功能原型的UML关系

资料来源：摘自[Bos00]

9.4.3　将体系结构精化为构件

"软件系统的结构提供一种'生态环境'，代码在这个环境里诞生、成长和消亡。一个设计良好的'生态环境'允许软件系统所需要的所有构件成功进化。"
——R. Pattis

在将软件体系结构精化为构件时，系统的结构就开始显现了。但是，如何选择这些构件呢？为了回答这个问题，先从需求模型[⊖]所描述的类开始。这些分析类表示软件体系结构中必需处理的应用（业务）领域的实体。因此，应用领域是构件导出和精化的一个源泉。另一个源泉是基础设施域。体系结构必须提供很多基础设施构件，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经常集成到软件体系结构中。

体系结构环境图（9.4.1节）描述的接口隐含着一个或者多个特定的构件，这些构件处理经过接口的数据。在某些情况下（如图形用户界面），需要设计具有许多构件的、完整的子系统体系结构。

继续SafeHome住宅安全功能的例子，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（例如，基于Internet的系统与外部报警通知）的通信。
- 控制面板处理——管理所有的控制面板功能。
- 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迭代精化，然后在总的SafeHome体系结构中进行定位。每个构件都需要定义设计类（包含相应的属性和操作）。然而，重要的是需要注意到，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10章）。

图9-8描述了整体体系结构（由UML构件图表示）。当事务从处理SafeHome 的GUI（图形用户界面）和Internet接口的构件进入时，它们就被外部通信管理获取。该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。控制面板处理构件通过与户主交互

⊖　如果选择了常规的（非面向对象）方法，构件从数据流模型中导出。我们将在9.6节简要讨论这个方法。

作用来实现安全功能的安装或解除（arm/disarm）。探测器管理构件定时查询传感器以检测报警条件，一旦检测到警报，警报处理构件将产生相应的输出。

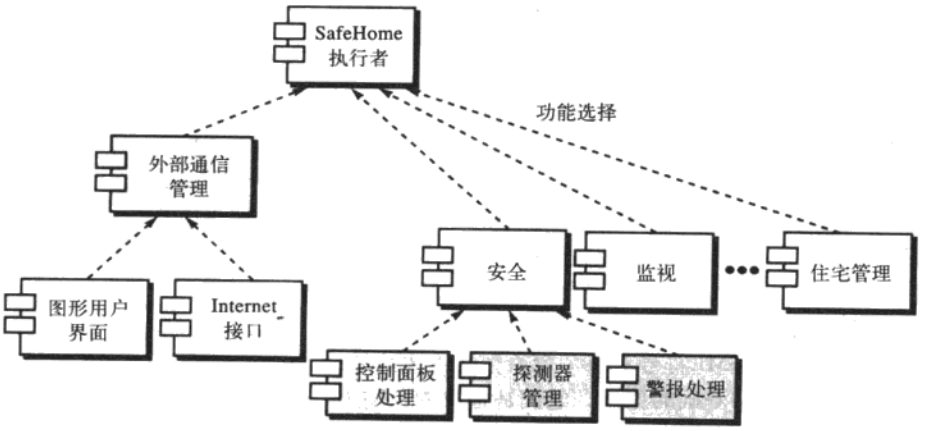


图9-8 带有顶层构件的SafeHome整体体系结构

9.4.4 描述系统实例

到目前为止，所建模的体系结构设计依然处于比较高的层次。系统环境已经表明，表示问题域中重要抽象的原型已经定义，系统的整体结构已经显现，并且主要的软件构件也都确定了。然而，更进一步的精化（回想一下所有的设计都是迭代的）仍然是必要的。

为了进行体系结构的求精，需要开发体系结构的实际用例。这样做的用意是将体系结构应用到特定问题上，目的是证明结构和构件都是合理的。

图9-9描述的是安全系统SafeHome体系结构的一个实例。图9-8中显示的构件被进一步细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安全系统中使用的每个传感器对象的定时查询。图9-8中显示的每个构件都作了类似的细化。

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：⊖

Adalon，由Synthis公司（www.synthis.com）开发，是一种专用设计工具，用于设计和构建特定的基于Web构件的体系结构。

ObjectiF，由microTOOL GmbH（www.microtool.de/objectiF/en/）开发，是一个基于UML的设计工具，它产生的体系结构（例如，Coldfusion、J2EE和Fusebox等）符合基于构件的软件工程（第29章）。

Rational Rose，由Rational开发（www-306.ibm.com/software/rational/），是基于UML的设计工具，它支持体系结构设计中的所有方面。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

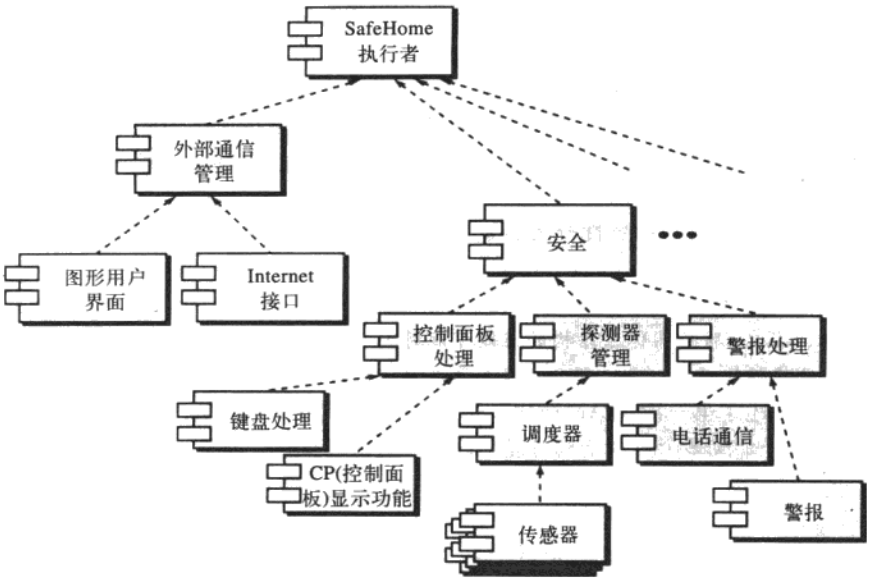


图9-9　构件细化的安全功能实例

9.5　评估可选的体系结构设计

Clements和他的同事[Cle03]在他们关于软件体系结构进化的著作中声称：

说穿了，体系结构就是一个赌注，将注下在系统的成功上。如果你已经提前把赌注放在赢家，而不是等到系统快完成的时候才知道它是否满足需求，这样不是很好吗？如果你准备购买一个系统或者为系统开发付费，难道你不愿意有某种保证，使得系统不偏离正确的道路？如果你自己就是系统架构师，难道你不愿意有一种好方法来验证你的直觉和经验，以致知道所依赖的设计是有基础的时候，在夜里睡得踏实？

回答这些问题确实有价值。设计会导致多种可选的体系结构，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。在下面几节中，我们提出两种不同的候选体系结构设计的评估方法。第一种方法使用迭代方法评估设计折中，第二种方法运用伪定量技术来评估设计质量。

9.5.1　体系结构权衡分析方法

WebRef

关于ATAM的更深入的信息可以从www.sei.cmu.edu/activities/architecture/ata_method.html获得。

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：

1. 收集场景。开发一组用例（第5章和第6章），从用户角度描述系统。

2. 引出需求、约束和环境描述。这些信息作为需求工程的一部分得到确定，并用来确保所有利益相关者的关注点都会被处理。

3. 描述那些已经被选择用于解决场景和需求的体系结构风格/模式。体系结构风格应该使用下面任意一种体系结构视图来描述：

- 模块视图：用于分析带有构件的工作任务以及信息隐蔽获得的程度。
- 过程视图：用于分析系统性能。
- 数据流视图：用于分析体系结构满足功能需求的程度。

4. 通过单独地考虑每个属性来评估质量属性。用于分析的质量属性个数是评审可用时间和质量属性与现存系统相关程度的函数。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，确定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）对该变更的敏感性来完成。受体系结构变更影响很大的属性称为敏感点。

6. 使用在第5步进行的敏感性分析鉴定候选体系结构（在第3步开发的）。SEI用如下方式描述该方法[Kaz98]：

一旦确定了体系结构敏感点，寻找这种权衡点就简单了，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）……那么，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，再次应用ATAM步骤[⊖]。

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：我知道大家为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，我们该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？

Jamie：我在高年级时学习了一门计算机科学课程，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，并使用用例和场景来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，我们将每个用例都应用到这两个体系结构中，查看系统的反应，即在用例环境中构件和连接件是如何工作的。

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，系统是不是必须转换到它的分支上来完成工作。

Jamie：场景不就是用例的别名吗？

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更场景，是吗？

Vinod：是的，我们要做的就是回到利益相关者那里，问问他们SafeHome在未来三年可能会有什么变更，如，新版本、特征等这类变更。我们创建了一套变更场景，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：我们把它们运用到体系结构中。

Vinod：是的，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。

⊖ 软件体系结构分析方法（SAAM）是ATAM的替代方案，对体系结构分析感兴趣的那些读者值得研究。可以从www.sei.cum.edu/publications/articles/saam-metho-propert-sas.html网站上下载关于SAAM的论文。

9.5.2 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流或控制流驱动的。Zhao[Zha98]建议了3种类型的依赖：

共享依赖表示使用相同资源的消费者之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*必须在控制流进入*v*（先决条件）之前完成，或*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*之间存在约束依赖关系。

Zhao提到的共享依赖和流依赖类似于第8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23章将讨论用来评估耦合的简单度量。

9.5.3 体系结构描述语言

房屋建筑师拥有一套标准的工具和符号，能够以一种明确的、可理解的方式描述设计。尽管软件体系结构设计师也能够使用UML符号、其他图表和一些相关工具进行绘图，但还是需要更加形式化的方法对体系结构设计进行规格说明。

体系结构描述语言（architectural description language, ADL）提供了一种描述软件体系结构的语义和语法。Hofmann和他的同事[Hof01]建议ADL应该使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接口（连接机制）的能力。一旦进行了描述，就建立了基于语言技术的体系结构设计，当设计演化时，则更可能建立起有效的体系结构评估方法。

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（http://poset.stanford.edu/rapide/）建立在偏序集的概念之上，因此介绍了相当新（但看起来功能很强）的编程结构。

UniCon（www.cs.cmu.edu/~UniCon）是“一种体系结构描述语言，旨在帮助设计者用他们发现的很有用的抽象形式定义软件体系结构。”

Aesop（www.cs.cmu.edu/~able/aesop/）解决风格的复用问题。使用Aesop，可以定义风格并在构建实际系统的时候使用所定义的风格。

Wright（www.cs.cmu.edu./~able/wright/）是一种包含如下元素的形式化语言：带有端口的构件、带有角色的连接件，以及将角色连接到端口上的粘合剂。可以使用谓词语言规范体系结构风格，从而允许静态检查，以确定体系结构的一致性和完整性。

Acme（www.cs.cmu.edu/~acme/）是第二代ADL，换句话说，它的目的是确定一种最少共同点ADL。

UML（www.uml.org/）包括很多体系结构描述需要的人工制品——过程、结点、视图等。但是没有其他ADL那样完整。对于非正式的描述，UML是相当适用的，因为它有广泛的理解标准。然而，UML没有足够的能力进行体系结构的充分描述。

9.6 使用数据流进行体系结构映射

9.3.1节讨论的体系结构风格描述了本质上不同的体系结构，因此，不存在一种能够实现从需求模型到各种体系风格转换的全面映射，这也并不奇怪。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9.4节讨论的技术将需求转换到这些风格的设计上。

为了描述一种体系结构映射方法，考虑“调用和返回”体系结构，这种体系结构是非常普通的结构，很多类型的系统都采用这种结构。“调用和返回”体系结构可以驻留在本章前面讨论的其他更复杂的体系结构中。例如，客户机-服务器体系结构的一个或多个构件的体系结构可能是“调用和返回”结构。

结构设计[You79]是一种映射技术，常被作为面向数据流的设计方法，因为它提供了从数据流图（第7章）到软件体系结构[⊖]的便捷的转化。从信息流（由DFD表示）到程序结构的转换由下述6个步骤的一部分完成：（1）建立信息流的类型；（2）标注流的边界；（3）将DFD映射到程序结构；（4）定义控制层级；（5）使用设计度量和启发式精化产生的结构；（6）求精并细化体系结构描述。

作为数据流映射的简单实例，我们提出SafeHome安全功能一小部分的逐步“变换”映射[⊜]。为了完成映射，必须确定信息流的类型。变换流是信息流的一种类型，显示了线性特性。数据流沿着输入流路径进入系统，在该路径上，数据流从外部世界表示变换为内在形式。一旦内在形式化完成，数据流在变换中心进行处理。最后，数据流沿着输出流路径从系统流出，该路径将数据变换成外部世界的形式[⊜]。

9.6.1 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格。为了说明这个方法，我们再次考虑SafeHome安全功能[⊛]。分析模型有一个元素是描述安全功能中信息流的一组数据流图。为了把这些数据流图映射到软件体系结构，可以从下面的设计步骤开始：

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，描述了流入和流出该功能的数据的外部生产者和消费者。图9-10描绘了一个0层环境模型，图9-11显示了求精后的安全功能数据流。

图9-10 SafeHome安全功能的环境级DFD

⊖ 需要注意的是，在映射方法中，需求模型的其他元素也会用到。

⊜ 在该书站点中，会给出结构设计更详细的讨论。

⊜ 信息流的另一个重要类型是事务流，本例中没有考虑，但在本书站点给出的结构设计例子中进行了陈述。

⊛ 我们只考虑了SafeHome使用控制面板的安全功能的一部分。这里并没有考虑本书讨论的其他特征。

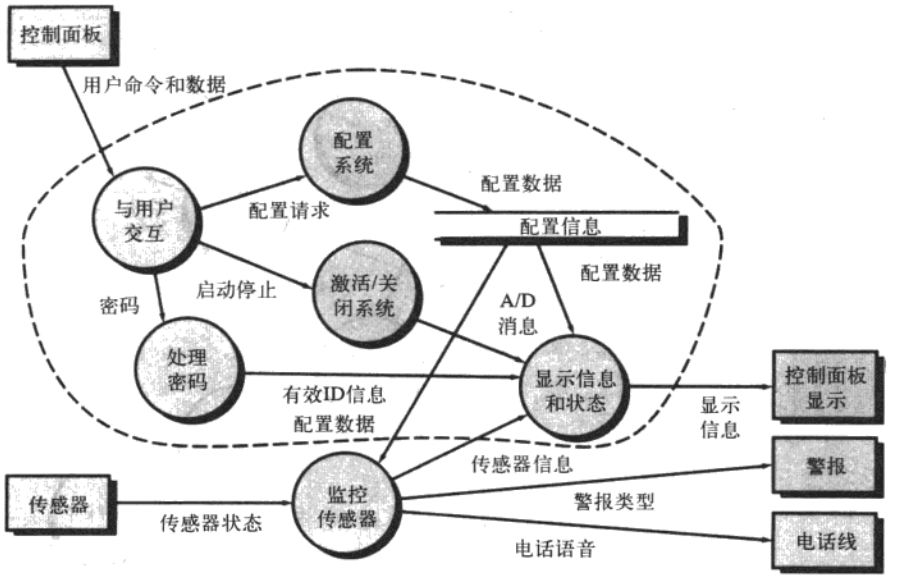


图9-11　SafeHome安全功能的第一级DFD

ADVICE
如果此时对DFD进一步求精，将争取导出显示高内聚性的“泡泡”。

步骤2：评审和精化软件的数据流图。对从需求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9-12），并导出图9-13显示的第3层数据流图。在第3层，数据流图中的每个变换都展示了相对高的内聚性（第8章），也就是说，变换所隐含的过程完成单一的、清楚的功能，该功能可作为SafeHome软件中的一个构件来实现。因此，图9-13中的DFD包含了设计监控传感器子系统体系结构所需的充分的细节信息，不需要再进一步精化。

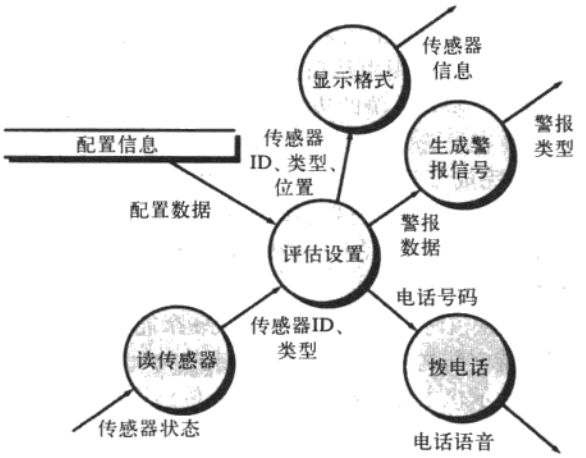


图9-12　精化“监控传感器”变换的第2层DFD

步骤3：确定DFD是否含有变换流或事务流[⊖]特征。通过评估DFD（图9-13），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。因此，信息流将呈现出一个从头到尾的总体变换特征。

⊖　在事务流中，有一个数据项被称作事务，它使数据流沿着根据事务特性定义的许多流路径中的一个进行分岔。

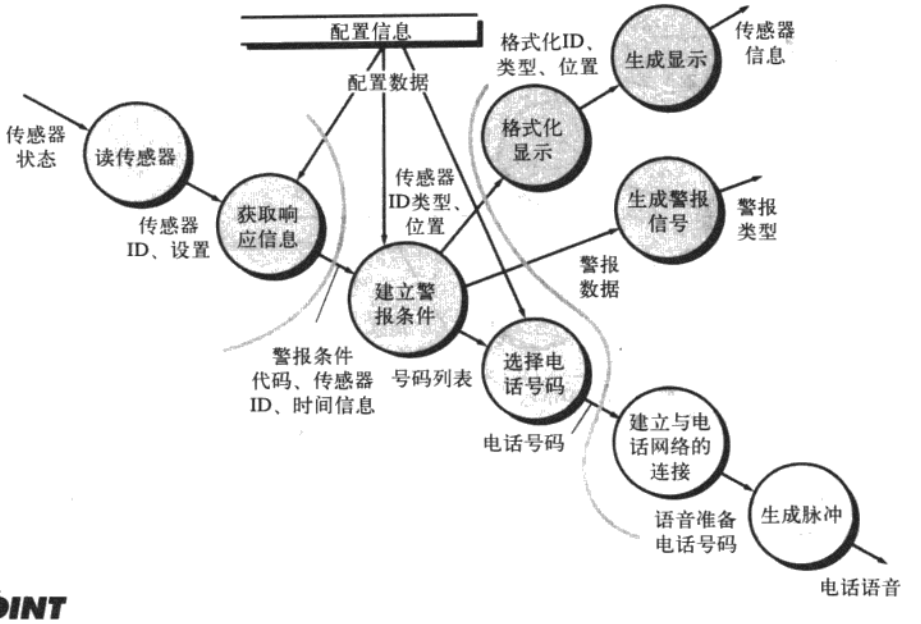


图9-13 具有流边界监控传感器的第3层DFD

KEY POINT
在同一种面向流的模型中，你经常会遇到两种数据流类型。对这两类数据流加以区分，并使用合适的映射导出程序结构。

步骤4：通过确定输入和输出流的边界，分离出变换中心。输入数据流沿着一条路径流动，在该路径上，信息从外部形式转换为内部形式；输出流将内部数据转化为外部形式。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9-13中纵向穿过流的阴影曲线给出。组成变换中心的变换（泡泡）位于图中从顶到底的两条加阴影的边界曲线之间。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。

ADVICE
尽可能探索候选程序结构，改变流边界位置。这项工作花费的时间不多，却提供了重要的启示。

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，其中顶层构件完成决策制定；底层构件完成大多数输入、计算和输出工作；中间层构件完成一部分控制和适度的任务。

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）。监控传感器子系统的第一级分解如图9-14所示，主控制器（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：

ADVICE
这一步不能太教条。根据要构建系统的复杂度，也可能有必要为输入处理或计算建立两个或多个控制器。常识指导这样做的话，那就这样做！

- 输入信息处理控制器（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（称为警报条件控制器）负责管理内部形式数据上的所有操作（例如，调用各种数据变换过程的模块）。
- 输出流信息处理控制器（称为警报输出控制器）负责协调输出信息的产生。

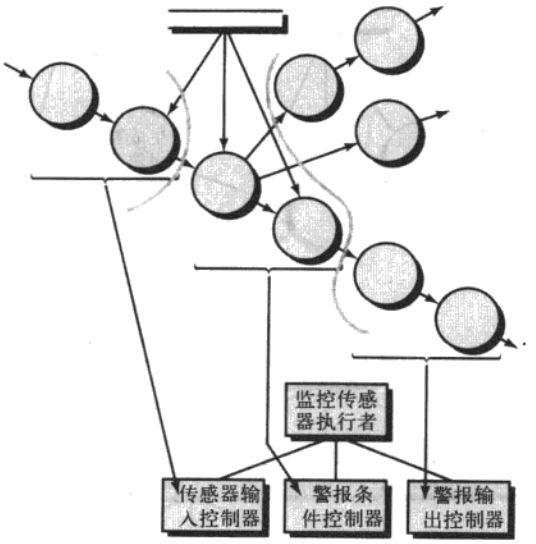


图9-14　监控传感器的第一级分解

ADVICE
消除冗余控制模块。换句话说，如果一个控制模块只是控制另外一个模块，并无其他功能，就应该将其控制功能结合到更高级别的模块中。

虽然图9-14隐含了三叉结构，但是，大型系统的复杂数据流可能会要求为前面描述的每个通用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保持良好的功能独立特征所要求的最少模块数。

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）映射到体系结构中的相应模块。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9-15描述了第二级分解的一般方法。

虽然图9-15描述了DFD变换和软件模块间的一对一映射，但不同的映射方式也经常发生。两个甚至三个泡泡可以合并在一起表示为一个构件，一个单独的泡泡也可以扩展成两个或者多个构件。实际考虑和设计质量度量决定着第二级分解的输出结果，评审和精化也可能导致这个结构的改变，但这个结构仍然应该作为第一次迭代设计。

ADVICE
在程序结构中，设置低层“工作”模块，这将使体系结构更容易维护。

对于输入流的第二级分解遵循同样的方式，通过从输入流一侧的变换中心边界开始向外移动来完成分解。监控传感器子系统软件的变换中心映射略有不同，DFD变换部分的每个数据转换或计算变换都被映射为变换控制器的下级模块。图9-16给出了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9-16所示）描述了软件体系结构的初始设计。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说明（根据处理的规格说明得到的。该规格说明是为需求建模过程中建立的数据变换所开发的）。该说明描述构件接口、内部数据结构、功能说明以及限制和特殊功能的简要描述（例如，文件输入输出、硬件依赖特征、特殊的定时需求）。

“尽可能简单，但没有更简单。”——Albert Einstein

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8章），能够精化第一次迭代得到的体系结构。对构件进行“分解”（explode）或“结合”（implode），可以产生合理的分解、关注点分离、好的内聚性、低的耦合性，最重要的是获得易于实现、测试和易于维护的程序结构。

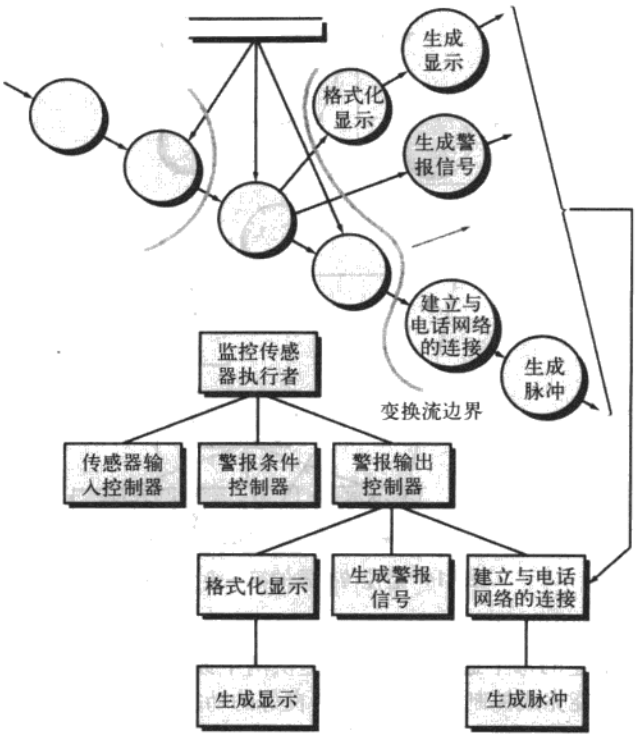


图9-15 监控传感器的第二级分解

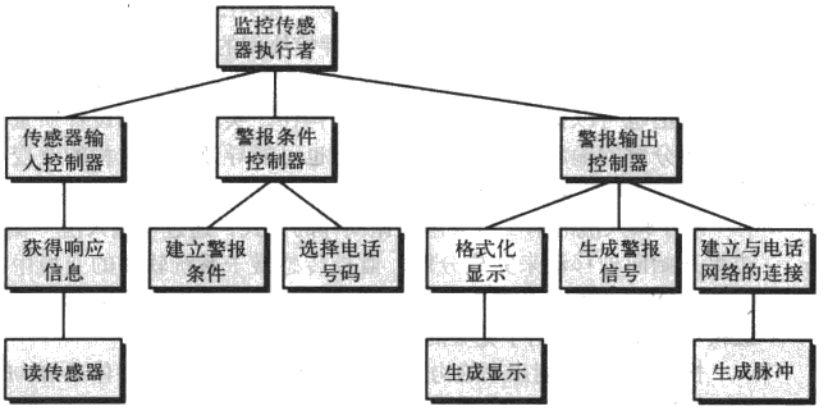


图9-16 监控传感器第一次迭代结构

求精往往是在9.5节简要描述的分析和评估方法以及实际考虑和常识等指导之下进行的。例如，有时输入数据流控制器完全没有必要，有时需要在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裁决。

以上7个步骤的目的是开发一个软件的体系结构表示，也就是说，一旦定义了结构，我们就可以将其视为一个整体，并对软件体系结构进行评估和精化。这时，所作的修改需要很少额外的工作，它对软件的质量具有深远的影响。

你可以停下来考虑一下以上介绍的设计方法与“编程”过程之间的区别。如果软件只用代码来表示，你和你的同事将很难在全局和整体的层次上对软件进行评估或精化，事实上，也将容易导致“只见树木不见森林”。

SAFEHOME

精化初级体系结构

[场景] Jamie的房间，正开始设计建模。

[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9-16，她看了一会儿。]

Jamie：不错，但是我认为可以再简化一些，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：因为需要一个映射控制器。

Jamie：不是这样的，由于我们管理的是输入数据的单一流路径，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！我们还可以将建立警报条件和选择电话号码两个构件结合在一起。你在图中给出的转换控制器没有必要，稍微降低内聚性是允许的。

Ed：简单，哈哈。

Jamie：是的，而且当你进行精化的时候，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9-17。]

Jamie：这只是个好的开始罢了。

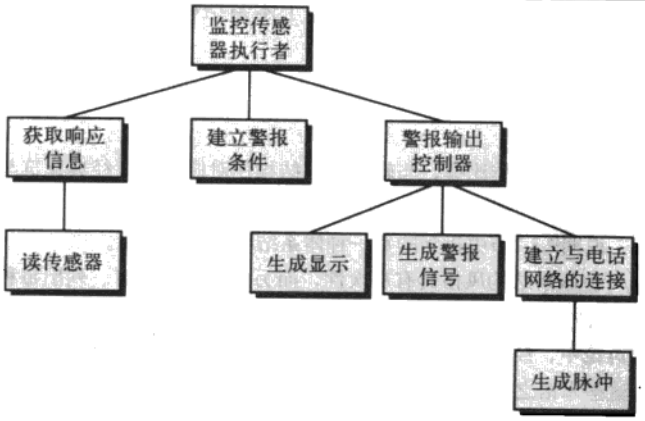


图9-17　监控传感器精化后的程序结构

9.6.2　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，你应该始终考虑开发一个满足所有功能和性能需求以及

在体系结构创建之后，会发生什么？

值得接受的软件表示。

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，为了获得“最好”的方法，候选体系结构风格可以被导出、精化以及评估。这种优化方法也是开发软件体系结构表示的真正好处之一。

重要的是要注意到，结构上的简单往往反映出程序的精致和高效。设计求精应在满足有效模块要求的前提下尽量减少构件的数量，在充分满足信息需求的前提下尽量减少复杂的数据结构。

9.7 小结

软件体系结构提供了待构造系统的整体视图，它描述软件构件的结构和组织、构件的特点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组成部分。体系结构注重于早期的设计决策，并提供了考虑多个可选系统结构优点的机制。

软件工程师可以使用许多种不同的体系结构风格和模式，在给定的体系结构类型中可以使用这些风格和模式。每个风格描述了一个系统类别，它包含：一组完成系统所需功能的构件；一组使构件间通信、协调及合作的连接件；定义如何集成构件以构成系统的约束条件；使设计者能够理解系统整体特性的语义模型。

一般来说，体系结构设计需要4个不同步骤来完成。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，称之为原型，该原型可以表示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识别和描述这些构件。最后，开发体系结构的特定实例，在真实世界中验证设计。

作为体系结构设计的一个简单例子，本章介绍的映射方法使用数据流特征来导出常用的体系结构风格。使用变换映射方法将数据流图映射为程序结构。变换映射应用到信息流中，展示了输入和输出数据之间清楚的边界。DFD被映射到一种结构，该结构沿着3个单独分解的模块层次将控制分配给输入、处理与输出。一旦导出了体系结构，就对其进行细化，然后使用质量标准对其进行分析。

习题与思考题

9.1 用一个房屋或建筑物的结构作比喻，与软件体系结构作对照分析。经典建筑与软件体系结构的原则有什么相似之处？又有何区别？

9.2 举出2到3个例子，说明9.3.1节中提到的每一种体系结构风格的应用。

9.3 9.3.1节中提到的一些体系结构风格具有层次性，而另外一些则没有。列出每种类型。没有层次的体系结构风格如何实现？

9.4 在软件体系结构讨论中，经常会遇到体系结构风格、体系结构模式及框架（本书中没有讨论）等术语。研究并描述这些术语之间的不同。

9.5 选择一个你熟悉的应用，回答9.3.3节中对于控制与数据提出的每一个问题。

9.6 研究ATAM（[Kaz98]）并对9.5.1节提出的6个步骤进行详细讨论。

9.7 如果还没有完成问题6.6，请先完成它。使用本章描述的设计方法开发PHTRS的软件体系结构。

9.8 使用数据流图和过程说明，描述一个有清楚变换流特征的计算机系统。定义流边界并使用9.6.1节描述的技术将DFD映射到软件体系结构中。

推荐读物与阅读信息

在过去的10年中，已经有了很多关于软件体系结构的文献。Gorton（《Essential Software Architecture》，Springer, 2006）、Reekie和McAdam（《A Software Architecture Primer》，Angophora Press, 2006），Albin（《The Art of Software Architecture》，Wiley, 2003），以及Bass和他的同事（《Software Architecture in Practice》，2nd ed., Addison-Wesley, 2002)的书中，对于智能领域挑战性的话题给出了有意义的介绍。

Buschman和他的同事（《Patten-Oriented Software Architecture》，Wiley, 2007)及Kuchana（《Software Architecture Design Patterns in Java》，Auerbach, 2004)讨论了体系结构设计的面向模式的方面。Rozanski和Woods（《Software Systems Architecture》，Addison-Wesley, 2005），Fowler(《Pattern of Enterprise Application Architecture》，Addison-Wesley, 2003），Clements和他的同事（《Documenting Software Architecture: View and Beyond》，Addison-Wesley, 2002），Bosch[BOS00]，以及Hofmeister和他的同事[HOF00]提供了软件体系结构的深入方法。

Hennesey和Patterson（《Computer Architecture》，4th ed., Morgan-Kaufmann, 2007)对软件体系结构的设计问题作了清楚的定量描述。Clements和他的同事（《Evaluating Software Architecture》，Addison-Wesley, 2002)对软件体系结构候选方案评估以及给定问题域的最好软件体系结构选取的相关问题进行了研究。

关于体系结构具体实现的书目陈述了具体开发环境和技术中的体系结构设计。Marks和Bell（《Service-Oriented Architecture》，Wiley, 2006)讨论了联系业务和计算资源与客户定义需求的设计方法。Stahl和他的同事（《Model-Driven Software Development》，Wiley, 2006)讨论了在具体领域建模方法环境中的体系结构。Radaideh和Al-ameed（《Architecture of Reliable Web Applications Software》，GI Global, 2007)研究了适用于Web应用的体系结构。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。Shanley（《Protected Mode Software Architecture》，Addison-Wesley, 1996)给出了任何人设计基于PC的实时操作系统、多任务操作系统或者设备驱动程序的设计指导原则。

当前的软件体系结构研究每年都会记录在由ACM和其他计算机组织发起的软件体系结构国际研讨会（Proceedings of the International Workshop on Software Architecture）和软件工程国际会议(Proceedings of the International Conference on Software Engineering)论文集中。

关于体系结构设计的大量的信息资源可以在Internet上获得。在SEPA网站（www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm）上可以找到和体系结构设计相关的最新WWW参考信息。

第10章

Software Engineering：A Practitioner's Approach (Seventh Edition)

构件级设计

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，该设计采用图表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过程设计。通常的做法是采用现有的可复用软件构件，而不是开发新的构件。

工作产品：每个构件的设计都以基于图形的表格或文本方式表示，这是构件级设计阶段产生的主要工作产品。

质量保证措施：采用设计评审机制。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
分类
适应
组合
面向对象的
合格性
传统的
Web应用
基于构件的开发
内容设计
耦合
设计准则
领域工程
表格设计表示

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。Edsgar Dijkstra（帮助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，程序设计过程的成本更低……高效率的程序员……不应该将他们的时间浪费在调试上——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，而且还能够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，这样做有助于避免在过程设计演化中犯错误。本章将讨论这些设计指导准则和完成设计的方法。

10.1 什么是构件

"细节不仅是细节，它们构成了设计。"——Charles Eames

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 统一建模语言规范[OMG03a]是这样定义构件的："系统中模块化的、可部署的和可替换的部件，该部件封装了实现并暴露一组接口。"

正如第9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着重要作用。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。

对于术语构件的实际意义，软件工程师可能有不同的见解。在接下来的几节中，我们将了解关于"什么是构件以及在设计建模中如何使用构件"的3种重要观点。

10.1.1 面向对象的观点

KEY POINT 以面向对象的观点来看，构件是协作类的集合

在面向对象的软件工程环境中，构件包括一组协作的类[⊖]。构件中的每个类都得到详细阐述，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必须定义。为此，设计师需要从需求模型开始，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）。

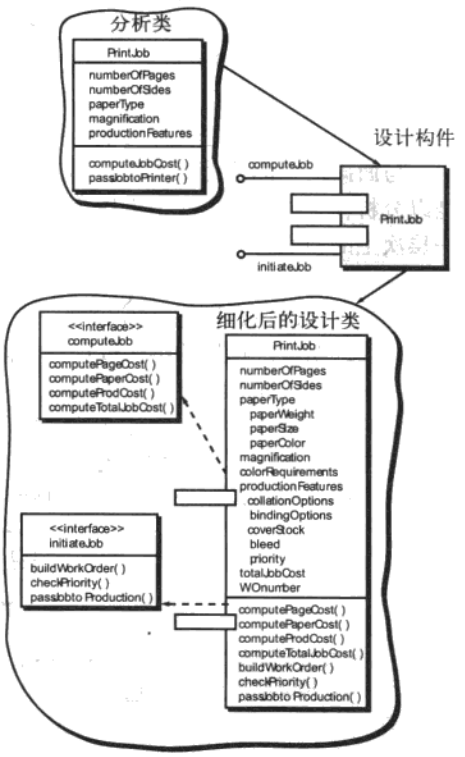


图10-1 设计构件的细化

ADVICE 回想一下，分析建模和设计建模都是迭代动作。细化原分析类可能需要额外的分析步骤，表示细化设计类的设计建模步骤紧随其后（构件的细节）。

为了说明设计细化过程，考虑为一个高级印刷车间构造软件。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称为PrintJob的分析类。分析过程中定义的属性和操作在图10-1的上方给出了注释。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10-1中部靠右的位置[⊜]。需要注意的是，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，以提供指导实现的足够信息。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10-1右下部分的描述，细化后的设计类PrintJob包含更多的属性信息和构件实现所需要的更

⊖ 在某些情况下，构件可以包含一个单独的类。

⊜ 不熟悉UML表示法的读者可以参考附录1。

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）操作可能与包含任务定价信息的PricingTable构件进行协作。checkPriority()操作（initiateJob接口的组成部分）可能与JobQueue构件进行协作，用来判断当前等待生产的任务类型和优先级。

对于体系结构设计组成部分的每个构件都要实施细化。细化一旦完成，要对每个属性、每个操作和每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。

10.1.2 传统观点

"可工作的复杂系统都是由可工作的简单系统演化来的。"
——John Gall

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。传统构件也称为模块，作为软件体系结构的一部分，它承担如下3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，实现客户需要的全部功能或部分功能；（3）基础设施构件，负责完成问题域中所需支持处理的功能。

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。不同的是在这种情况下，是以分析模型中的数据流要素作为导出构件的基础。数据流图最低层的每个变换都被映射为某一层次上的模块（9.6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8章）。

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造的软件。在需求建模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10-2中所显示的体系结构中。图中每个方框都表示一个软件构件。带阴影的方框在功能上相当于10.1.1节讨论的为PrintJob类定义的操作。然而，在这种情况下，每个操作都被表示为如图10-2所示的能够被调用的单独模块。其他模块用来控制处理过程，也就是前面提到的控制构件。

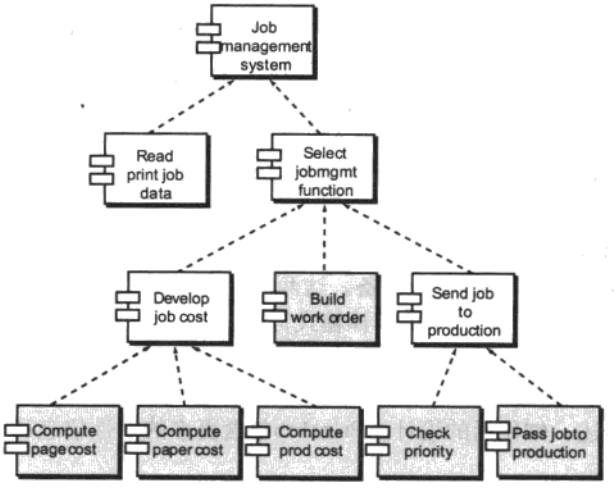


图10-2 一个传统系统的结构图

随着每个软件构件设计的逐步细化，设计的焦点转向特定数据结构的设计和操作数据结构的过程设计上。但是，不要忘了体系结构，它必须容纳构件或服务于多数构件的全局数据结构。

在构件级设计中，图10-2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8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成本。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成本——这是一个通过接口将所有数据传递给模块的功能。每页的成本与任务量成反比，与任务的复杂度成正比。

图10-3给出了使用改进的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成本的数据库）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10-3）。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。设计细化的过程一直进行下去，直到能够提供指导构件构造的足够细节为止。

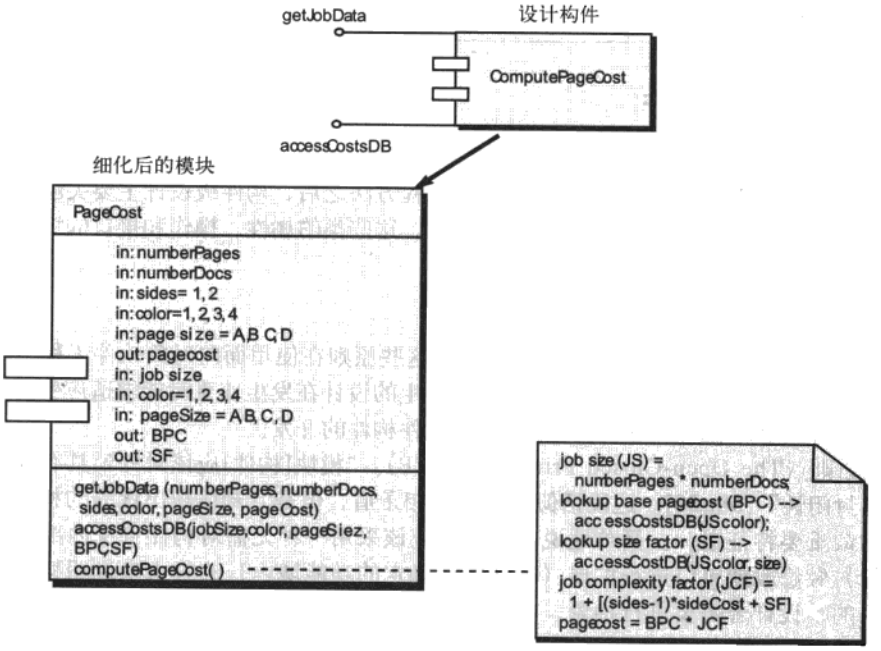


图10-3 ComputePageCost的构件级设计

10.1.3 过程相关的观点

10.1.1节和10.1.2节提到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，设计者必须根据从需求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20年间，软件工程界已经开始强调使用已有构件或设计模式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目录。当软件体系结构设计完成后，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10.6节将讨论基于构件的软件工程（Component-Based Software Engineering，CBSE）的某些重要方面。

INFO

基于构件的标准和框架

基于构件的软件工程（CBSE）成功或者失败的重要因素之一就是基于构件标准（有时称为中间件）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，他们可以使用如下的标准：

OMG CORBA（http://www.corba.org/）。

Microsoft COM（http://www.microsoft.com/com/tech/complus.asp）。

Microsoft .NET（http://msdn2.microsoft.com/en-us/netframework/default.aspx）。

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大量资源。

10.2 设计基于类的构件

正如前面提到的，构件级设计利用了需求模型（第6、7章）开发的信息和体系结构模型（第9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需求模型中问题域特定类的细化和基础设施类的定义和精化。这些类的属性、操作和接口的详细描述是开始构造活动之前所需的设计细节。

10.2.1 基本设计原则

有4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。

开闭原则（The Open-Closed Principle，OCP）。“模块[构件]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，但是它却体现出优秀的构件级设计应该具有的最重要特征之一。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，设计者需要进行抽象，在那些可能需要扩展的功能与设计类本身之间起到缓冲区的作用。

例如，假设SafeHome的安全功能使用了对各种类型的安全传感器进行状态检查的Detector类。随着时间的推移，安全传感器的数量和类型将会不断增长。如果内部处理逻辑采用一系列if-then-else结构来实现，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（依然是另外的if-then-else结构），这显然违背OCP原则。

图10-4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，sensor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，对Detector类（构件）无需进行任何改变。这个设计遵守了OCP原则。

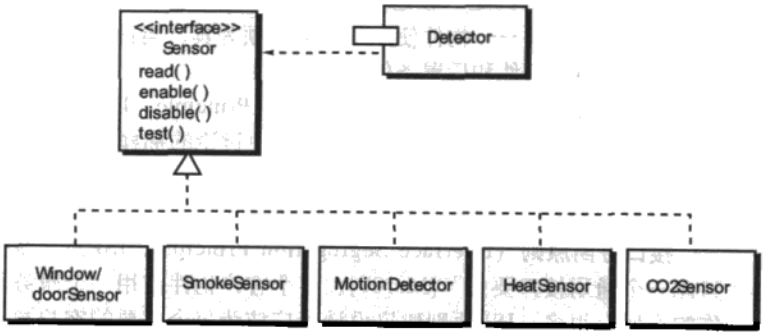


图10-4 遵循OCP原则

SAFEHOME

OCP的应用

[场景] Vinod的工作间。

[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，别再加了。

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10-4）。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合适的构件……哦，类。根本不用改变Detector构件。

Vinod：好的，我会告诉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，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。

Vinod（想了想）：你听说过开-闭原则吗？

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，将从基类导出的类传递给构件时，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类，则要确保这些导出类必须遵守前置条件和后置条件。

如果你省去设计并且破解出代码，记住代码只是最终的“具体”，你违背了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。构件依赖的其他具体构件（不是依赖抽象类，如接口）越多，扩展起来就越困难。

接口分离原则（Interface Segregation Principle，ISP）。“多个客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个特定的接口中都加以说明。

例如，假设FloorPlan类用在SafeHome的安全和监视功能中（第6章）。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remove-Device()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监视功能除了需要这4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placeDevice()、showDevice()、groupDevice()和removeDevice()4种操作。监视接口应该包括placeDevice()、showDevice()、groupDevice()、removeDevice()、showFOV()和showDeviceID()6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到子系统或包中。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：

设计可复用的构件不仅需要优秀的技术设计，而且需要高效的配置控制机制（第22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]。当设计类或构件用以复用时，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，同时用户逐渐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并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。当某个域的一些特征必须变更时，只有相应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。

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，包的发布版本号也会发生变更。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。

10.2.2 构件级设计指导方针

除了10.2.1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[Amb02b]给出了如下的指导方针：

命名构件时需要考虑些什么？

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，并对其做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，并且对于考虑体系结构模型的所有利益相关者来说是意义明确的。例如，无论技术背景如何，FloorPlan这个类的名称对于任何读到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以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，因为即使是非技术人员也不会误解[⊖]。

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[Amb02c]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[⊜]；（2）为了保持一致，接口都放在构件框的左边；（3）即使其他接口也适用，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。

依赖与继承。为了提高可读性，依赖关系是自左向右，继承关系是自底（导出类）向上（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。

10.2.3 内聚性

在第8章中，我们将内聚性描述为构件的专一性。在为面向对象系统进行构件级设计时，内聚性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Laganiére[Let01]定义了许多不同类型的内聚性（按照内聚性的级别排序[⊝]）：

尽管理解各种级别的内聚性很有益，但是更为重要的是在设计构件时对内聚性有全面的理解，尽可能保持高内聚性。

功能内聚。主要通过操作来体现，当一个模块只完成某一组特定操作并返回结果时，就称此模块是功能内聚的。

分层内聚。由包、构件和类来体现。高层能够访问低层的服务，但低层不能访问高层的服务。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10-5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包向下进行的。

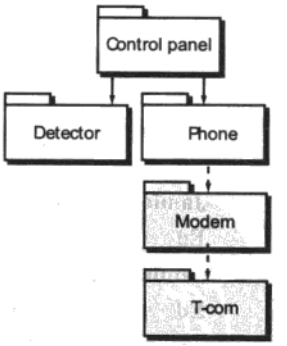


图10-5 分层内聚

⊖ 销售部和顾客部（非技术类型）详细检查设计信息是不太可能的。

⊜ 这里指的是类似图10-1中的接口表示。——译者注

⊝ 一般来说，内聚性级别越高，构件的实现、测试和维护就越容易。

通信内聚。访问相同数据的所有操作被定义在一个类中。一般来说，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，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。

SAFEHOME

内聚性的应用

[场景] Jamie的工作间。

[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5种操作。看……

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，将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() 操作不是专一的。

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，并进行这种改变。

Jamie：那么引起的副作用怎么办？

Ed：什么意思？

Jamie：也就是说你做了修改，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做这种改变怎么办。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：好主意。

10.2.4 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。

有多种方法表示类之间的耦合。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，就会发生这种类型的耦合。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这种耦合。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），但是当这种耦合进行变更时，会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*传递控制标记时，就会发生这种耦合。接着，控制标记将会指引*B*中的逻辑流程。这种耦合形式的主要问题在于*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。

标记耦合。当类*B*被声明为类*A*某一操作中的一个参数类型时，就会发生这种耦合。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。当操作需要传递长串的数据参数时，就会发生这种耦合。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，就会发生这种耦合。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，就会发生这种耦合（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]，就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，就会发生这种耦合。

外部耦合。当一个构件和基础设施构件（例如，操作系统功能、数据库容量、无线通信功能等）进行通信和协作时会发生这种耦合。尽管这种类型的耦合是必要的，但是在一个系统中应该尽量将这种耦合限制在少量的构件或者类范围内。

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。

耦合的应用 SAFEHOME

[场景] Shakira的工作间。

[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：是的，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（外呼）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，并且让其启动外呼，这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……

10.3 实施构件级设计

"如果我有更多的时间，我将写一封更短的信。" —— Blaise Pascal

在本章的前半部分，我们已经知道构件级设计本质上是精细化的。设计者必须将需求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件（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典型的任务集。

步骤1：标识出所有与问题域相对应的设计类。使用需求模型和架构模型，正如10.1.1节所描述的那样，每个分析类和体系结构构件都要细化。

ADVICE

如果在非面向对象的工作环境下工作，那么前3步的重点在于提炼数据对象和处理功能（转换），这些功能被视为需求模型的一部分。

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（通用用户界面）构件（通常为可复用构件）、操作系统构件以及对象和数据管理构件等。

步骤3：细化所有不需要作为可复用构件的设计类。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，必须考虑采用设计的启发式规则（如构件的内聚和耦合）。

步骤3a：在类或构件协作时说明消息的细节。需求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，但是它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。

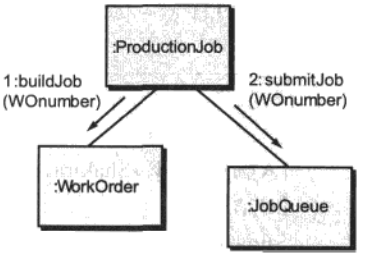


图10-6 带消息的协作图

图10-6给出了前面提到的印刷系统的一个简单协作

图。ProductionJob、WorkOrder和JobQueue这3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。在需求建模时，消息说明如图10-6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）[⊖]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序号的整数（或其他样式的表明发送顺序的指示符，如3.1.2）；（return value）是由消息唤醒的操作返回的信息名；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.....”[Ben02]。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10-1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多个抽象类。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。

参照图10-1，由于initiateJob接口没有展现出足够的内聚性而受到争议。实际上，它完成了3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。操作buildWorkOrder()成为该类的一部分。类似地，我们可能需要定义包括操作checkPriority()在内的JobQueue类。ProductionJob类包括给生产线传递生产任务的所有相关信息。initiateJob接口将采用图10-7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相关的接口近乎都是专一的。

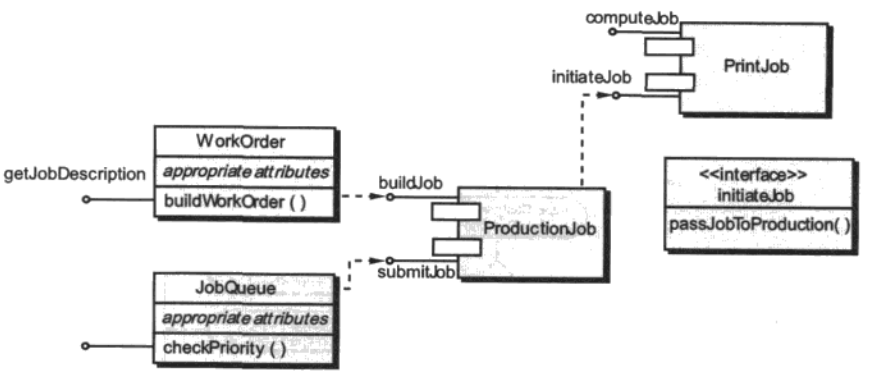


图10-7 为PrintJob重构接口和类定义

步骤3c：细化属性，并且定义实现属性所需要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取值；

⊖ OCL在附录1中有简明的介绍。

property string用于定义属性的特性或特征。

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10-1，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，可以取值为集合{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8章）通过很多次迭代进行细化。

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，操作应该确保具有高内聚性的特性；也就是说，一个操作应该完成单一的目标功能或者子功能。接下来的一轮迭代，只是完成对操作名的详细扩展。例如，图10-1中的操作computePaperCost()可以采用如下方式进行扩展：

在精化构件设计时使用逐步细化的方法。经常提出这样的问题“是否存在一种方法可以简化问题并仍能达到相同的结果？”

computePaperCost(weight, size, color): numeric

这种方式说明computPageCost()要求属性weight、size和color作为输入，并返回一个数值（实际上为金额）作为输出。

如果实现computePaperCost()的算法简单而且易于理解，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10-8给出了操作computePaperCost()的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，这部分内容将在10.5.3节进行讨论。

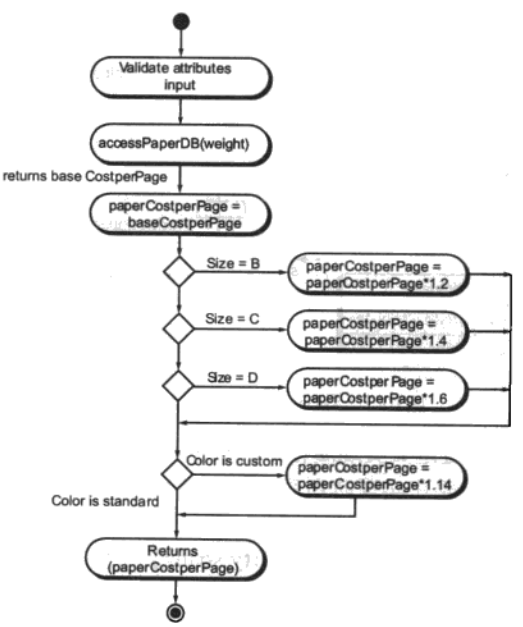


图10-8 computePaperCost()操作的UML活动图

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，这些持久数据存储最初都作为体系结构设计的一部分进行说明，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的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状态图被用作需求模型的一部分，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述影响对象的事件，以及随着时间流逝和事件的发生对象所处的状态。图10-9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）。

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OCL书写，并描述了事件发生前必须满足的条件，action expression定义了状态转换时发生的动作。

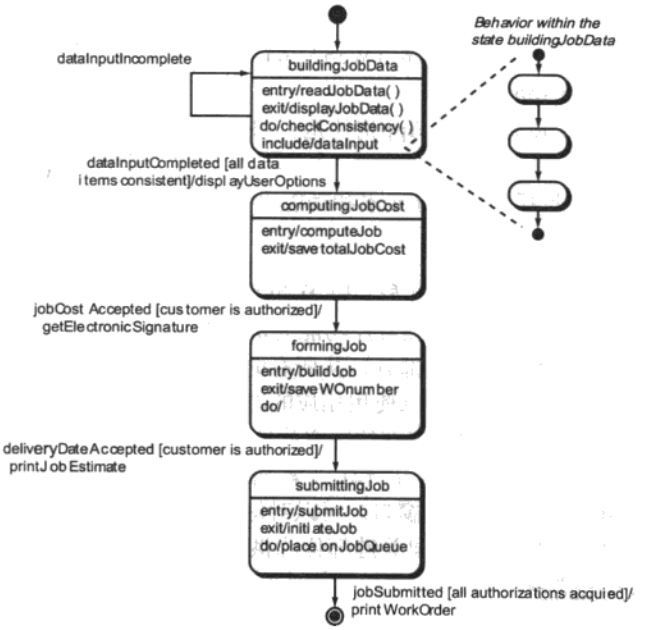


图10-9 PrintJob类的状态图

参照图10-9，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。在大多数情况下，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进行细化的手段。

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10-9中的状态图可以知道，当得出印刷任务的成本和进度数据时，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8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，应该对部署图进行细化，以表示主要构件包的位置。然而，一般在构件图中不单独表示构件，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。这意味着要对指定的硬件和使用的操作系统环境加以说明，而构件包在这个环境中的位置等也需要确定。

步骤7：考虑每个构件级设计表示，并且时刻考虑其他可选方案。综观全书，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8章和本章介绍的设计原则和概念开发其他可选方案，并且仔细考虑和分析这些方案。

10.4 WebApp的构件级设计

考虑到基于Web的系统和应用程序（WebApp）时，内容和功能的界限是模糊的。因此有必要去考虑：什么是WebApp构件。

根据本章的内容可以知道，WebApp构件是：（1）定义良好的聚合功能，为最终用户处理内容，或提供计算或数据处理；（2）内容和功能的聚合包，提供最终用户所需的功能。因此WebApp构件级设计通常包括内容设计元素和功能设计元素。

10.4.1 构件级内容设计

构件级内容设计关注于内容对象，以及包装展示给WebApp最终用户的方式。以SafeHomeAssured.com的基于网络视频监控功能为例，在众多功能中，用户可以选择和控制任意一个摄像头绘制建筑平面图的一个部分，从所有的摄像头获得视频捕获的缩略图，以及从任意摄像头显示视频流。另外，用户可以使用相应的图标控制镜头进行摇摄和变焦。

可以为视频监控器功能定义很多潜在的内容构件：(1) 内容对象：表示具有传感器与摄像头等特殊图标位置的空间布局（平面图）；(2) 采集到的极小的视频缩略图（每个都是一个独立的数据对象）；(3) 专用摄像头的视频流窗口。可以对每种构件单独命名并作为一个包进行操作。

以那个具有4个不同的分布在房间关键部位的摄像头的平面图为例。根据用户的请求，每个摄像头所拍摄的每帧图像都会作为动态内容对象VideoCaptureN保存下来，其中N代表摄像头1～摄像头4。内容构件Thumbnail-Images将所有的（4个）内容对象VideoCaptureN结合起来，然后将它们显示在一个视频监控页面上。

构件级内容设计应该适合创建的WebApp的特性。在很多情况下，内容对象不需要被组织成为构件，它们可以分别实现。但是，随着WebApp、内容对象以及它们相互关系的规模和复杂度的增长，在更好的参考和设计方法下组织内容是十分必要的[⊖]。此外，如果内容显示出高度的动态性（如一个在线拍卖网站的内容），那么建立一个包含有内容构件的、清晰的结构模型是非常重要的。

⊖ 内容构件可以在其他的WebApp中复用。

10.4.2 构件级功能设计

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) 执行本地化处理，从而动态地产生内容和导航功能；(2) 提供适合于WebApp业务领域的计算或数据处理；(3) 提供高级的数据库查询和访问；(4) 建立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WebApp程序构件，这些构件在形式上类似于传统的软件构件。

WebApp是作为一系列构件交付的，这些构件与信息体系结构并行开发，以确保它们的一致性。最重要的是，在一开始就要考虑需求模型和初始信息体系结构，然后才去考查：功能如何影响用户与系统的交互、要展示的信息以及要管理的用户任务。

在体系结构设计中，往往将WebApp的内容和功能结合在一起来设计应用系统的功能体系结构。在这里，功能体系结构代表的是WebApp的功能域，并且描述了关键的功能构件和这些构件是如何进行交互的。

例如，SafeHomeAssured.com视频监控的缩放功能是作为CameraControl构件的一部分来实现的。或者，该缩放功能也可以用Camera类的pan()和zoom()操作来实现。对于以上任何一种方式，与缩小或是放大功能相关的所有功能都应该在SafeHomeAssured.com中作为模块来实现。

10.5 设计传统构件

KEY POINT
结构化程序设计是一种设计技术，该技术将程序逻辑流程限制为以下3种结构：顺序型、条件型和重复型。

传统软件构件[⊖]的构件级设计基础在20世纪60年代早期已经形成，在Edsgar Dijkstra及其同事的著作（[Boh66]，[Dij65]，[Dij76b]）中又得到了进一步完善。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结构上。这组逻辑结构强调“对功能域的支持”，其中每个逻辑结构都是可预测的，过程流从顶端进入，从底端退出，读者很容易理解。

这些结构包括顺序型、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3种结构是结构化程序设计的基础，而结构化程序设计是一种重要的构件级设计技术。

结构化的构造使得软件的过程设计只采用少数可预知的逻辑结构。复杂性度量（见第23章）表明，使用结构化的构造降低了程序复杂性，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结构也符合心理学家所谓的人类“成块”的理解过程。要理解这一过程，可以考虑阅读一页的方式。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是字母块。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元素，而不必逐行阅读设计或是代码。当遇到了容易辨认的逻辑模式时，理解力就得到了提高。

无论是对于应用领域还是对于技术复杂度，任何程序都可以只用这3种结构化的构造来设计和实现。然而，需要注意的是，教条地使用这3种结构在实践中会遇到困难。10.5.1节将深入讨论这些问题。

10.5.1 图形化设计表示

“一张图片的信息量相当于千字的信息量”。但是更加重要的是要明确什么样的图片相当于

⊖ 传统的软件构件实现处理元素，这些处理元素涉及问题域中的功能或子功能，或者涉及基础设施域中的某种性能。通常将传统构件称为模块、程序或子程序，传统构件不像面向对象构件那样封装数据。

什么样的千字说明。毫无疑问，图形化工具（如UML活动图或是流程图）提供了有用的、可读性很高的程序细节。但是，如果错误地使用图形化工具，那么错误的图例会导致错误的软件。

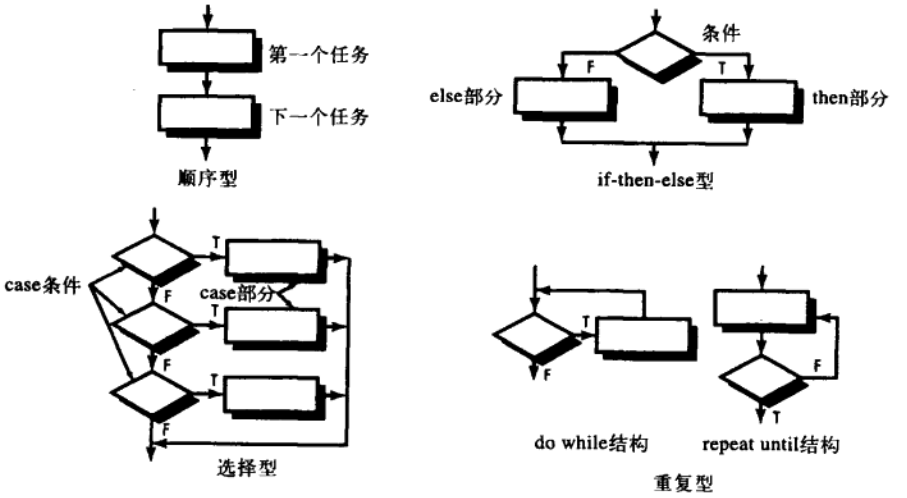


图10-10 流程图结构

活动图可以让设计者表示顺序、条件和重复结构（结构化程序设计的所有元素），它是由早期的形象化设计表示（至今依旧广泛使用）派生而来的，称为流程图。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10-10表示了3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；条件型也称作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，只要检验条件为真就重复执行循环任务。repeat until结构首先执行循环任务，然后再检验条件，并重复执行循环任务直到检验条件为假。图中的选择型（也称为select-case）构造实际上是if-then-else的扩展，参数被连续检验，直到有一次条件为真，其对应的case部分处理路径才能执行。

一般来说，如果需要从一组嵌套的循环或者条件中退出，完全依赖结构化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，降低可读性和维护性。如何解决这一问题呢？

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）在控制方式上违反结构化的构造，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程序设计的精神。

10.5.2 表格式设计表示

在一个构件中遇到一系列复杂的条件和动作集合时，应当使用决策表（来解决）。

在许多软件应用系统中，需要对模块计算复杂的条件组合，并根据这些条件选择合适的动作。决策表[Hur83]提供了一种表示方法，可以将（在处理叙述或用例中描述的）动作和条件翻译成表格，该表格很难被误解，并且能够作为表驱动算法的机器可识别的输入。

图10-11说明了决策表的组织结构。决策表分为4个部分，左上部列出了所有条件；左下部列出了所有基于组合条件的可能的动作；右半部分构成一个矩阵，该矩阵表示在特定的组合下，满足条件组合将会触发相应的动作。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：

如何创建决策表?

1. 列出特定过程（或构件）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，考虑下面这段摘自印刷车间系统中非正式用例的文字。

这里规定了3种会员：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%的优惠价格，并且在作业队列中排在普通会员的前面；黄金会员享有15%的优惠价，并且在作业队列中排在普通会员和白银会员的前面。根据管理制度，除了必要的打折优惠外，管理人员还可以让每位用户享受不同的优惠价格。

	规则					
条件	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	✓					
8%折扣			✓	✓		
15%的优惠					✓	✓
额外的x%的优惠		✓		✓		✓

图10-11 决策表

图10-11是处理上面非正式用例的决策表。共有6条规则，每条规则代表6个可执行的条件之一。一般情况下，可以有效地使用决策表来补充其他方式的过程设计表示。

10.5.3 程序设计语言

程序设计语言（Program Design Language，PDL）也称为结构化语言或伪代码。它结合了编程语言的逻辑结构和自然语言（如英语）的灵活表达形式。编程语言风格的语法中嵌入叙述性文字（如英语）。自动化工具[例如，Cai03]可以用来增强PDL的应用。

基本的PDL语法应该包括多种构造：构件定义、接口描述、数据声明、块结构、条件结构、重复结构和I/O结构。值得注意的是，PDL包括多任务处理和（或）并发处理、中断处理、交互同步以及其他特征的关键字。用于应用系统设计的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、水和温度（比如冬天房主外出时炉子熄灭）等方面的警报：产生报警信号，调用监控服务，发出合成语音信息。

考虑到PDL不是编程语言，设计人员可以根据需要进行修改，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前进行评审（你发现问题了吗）并精化。下面的PDL[⊖]对警报管理构件中过程设计的早期版本进行了精化。

```
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
```

⊖ 可以局部地定义由PDL所表示的详细程度的级别。有些人喜欢更接近自然语言的描述，而有些人则更喜欢接近程序设计语言的形式。

```
    check controlPanelSwitches (cps)
        if cps ="test" then invoke alarm set to "on"
        if cps = "alarmOff" then invoke alarm set to "off"
        if cps = "newBoundingValue" then invoke keyboardInput
        if cps = "burglarAlarmOff" invoke deactivateAlarm;
        •
        •
        •
        default for cps = none
    reset all signalValues and switches
    do for all sensors
        invoke checkSensor procedure returning signalValue
        if signalValue > bound [alarmType]
        then phoneMessage = 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
            endpar
        else skip
        endif
    enddofor
end alarmManagement
```

注意alarmManagement构件的设计人员使用了parbegin...endpar结构，该结构定义了一个并行块，在parbegin块内定义的所有任务都可以并行执行。但是，在上面这个例子中，并没有考虑实现的方法。

10.6 基于构件的开发

在软件工程领域，复用既是老概念，也是新概念。在计算机发展的早期，程序员就已经复用概念、抽象和过程，但是早期的复用更像是一种临时的方法。今天，复杂的、高质量的计算机系统往往必须在短时间内开发完成，所以就需要一种更系统的、更有组织性的复用方法来帮助这样的快速开发。

基于构件的软件工程（Component-Based Software Engineering，CBSE）是一种强调使用可复用的软件构件来设计与构造计算机系统的过程。Clements [Cle95]对CBSE给出了如下描述：

> [CBSE]体现了Fred Brooks和一些人推崇的“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把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”。

“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……”
—— Paul Clements

但是很多问题出现了，仅仅将多组可复用的软件构件组合起来，就能够构造出复杂的系统吗？这种工作能够以一种高效和节省成本的方式完成吗？能否建立恰当的激励机制鼓励软件工程师复用而不是重复开发？管理团队是否也愿意为在构造可复用的软件构件的过程中的额外开销买单？能否以使用者易于访问的方式构造复用所必需的构件库？那些具备复用性的构件可以被大家很容易地找到并使用吗？

大家渐渐地发现了这些问题的答案，而且这些问题的答案都是“可以”。在以下章节中，将讨论在软件工程组织中使CBSE成功的一些关键因素。

10.6.1 领域工程

领域工程的目的是识别、构造、分类和传播一组软件构件，这些构件在某一特定的应用领域中可以适用于现有的和未来的软件[⊖]。总体目标是为软件工程师建立一种机制来分享这些构件，从而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3种主要活动：分析、构造和传播。

这一节所讨论的分析过程主要集中于可复用构件。但是，完整的COTS系统（例如，电子商务应用、自动销售应用）分析也可以是领域分析的一部分。

领域分析的总体方法通常在面向对象软件工程的环境中被赋予特色。领域分析过程中的步骤定义如下：

1. 定义待研究的领域。
2. 把从领域中提取的项进行分类。
3. 收集领域中有代表性的应用系统样本。
4. 分析样本中的每个应用系统，并且定义分析类。
5. 为这些类开发需求模型。

值得注意的是，领域分析适用于任何软件工程范型，因此，领域分析可以应用到传统的软件开发和面向对象的软件开发中。

10.6.2 构件合格性检验、适应性修改与组合

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。正因为如此，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（第9章），并具有该应用系统所需的质量特性（例如，性能、可靠性、可用性）。

接口描述提供了有关软件构件的操作和使用的有用信息，但是，对于确定该构件是否能在新的应用系统中高效复用，它并未提供需要的所有信息。这里列出了构件合格性检验的一些重要因素：

在构件合格性检验期间要考虑哪些重要因素？

- 应用系统的编程接口（Application Programming Interface，API）。
- 构件所需的开发工具与集成工具。
- 运行时需求，包括资源使用（如内存或外存储器）、时间或速度以及网络协议。
- 服务需求，包括操作系统接口和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。
- 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。如果构件开发过程应用了良好的软件工程实践，之前列出的问题就容易回答。但是，要确定商业成品构件（Commercial Off-The-Shelf，COTS）或第三方构件的内部工作细节就比较困难了，因为能够得到的信息可能只有接口规格说明。

⊖ 第9章中曾经提到识别特定应用领域的体系结构类型。

构件适应性修改。在理想情况下，领域工程建立构件库，构件可以很容易地被集成到应用系统体系结构中。“容易集成”的含义是：（1）对于库中的所有构件，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的内部接口及与外部环境的接口。

软件团队除了需要评估为复用所做的适应性修改是否是成本合算的，还需要评估取得所需要的功能和性能是否也是成本合算的。

实际上，即使已经对某个构件在应用系统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（通常不是这样，除非使用开源的COTS构件），则可应用白盒包装技术。与软件测试中白盒测试（第18章）相对应，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预处理和后处理来消除或掩盖冲突。软件团队需要决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系统建立的体系结构中。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。

由于复用和CBSE对软件业的影响是巨大的，大量的主流公司和产业协会已经提出了软件构件标准[⊖]。

WebRef

有关CORBA的最新信息可以从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（Object Management Group/Common Object Request Broker Architecture，OMG/CORBA），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和.NET的最新信息可以从www.microsoft.com/COM和msdn2.microsoft.com/en-us/netframeworkdefault.aspx获得。

Microsoft COM和.NET。微软开发了构件对象模型（component object model，COM），此模型提供了在Windows操作系统上运行的单个应用系统内使用构件的规格说明，这些构件可以是不同厂商生产的。从应用的角度来说，“关键不在于COM对象是如何实现的，而只在于对象拥有在系统中注册的接口，并使构件系统与其他COM对象通信”[Har98a]。Microsoft .NET框架涵盖了COM，并提供了覆盖大量应用领域的可复用类库。

WebRef

有关Java Bean的最新信息可从Java.sun.com/products/javabeans/docs获得。

Sun JavaBean构件。JavaBean构件系统是一个可移植的、平台独立的CBSE基础设施，是使用Java程序设计语言开发的。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。

这些标准中没有哪一种在产业界占优势。虽然很多开发者已经采用了其中的某个标准，但大型软件组织仍可能根据应用系统的类型和采用的平台来选择使用某种标准。

⊖ 为了实现CBSE，过去和现在工业界都做了很多工作，Greg Olesn[Ols06]对此给出了精彩的讨论。

10.6.3 复用的分析与设计

虽然基于构件的软件工程（CBSE）鼓励使用现有的软件构件，但是很多时候都需要开发新的软件构件，并与现有的COTS及公司内部开发的构件集成。因为这些新的构件将会成为公司内部可复用构件库的成员，就应该为了将来的复用来开发它们。

设计概念，如抽象、隐藏、功能独立、精化和结构化程序设计，以及面向对象方法、测试、软件质量保证（SQA）和正确性验证方法（第21章），所有这些都有助于创建可复用的软件构件。本节考虑一些特定的复用问题，可作为良好软件开发实践的补充。

对需求模型进行分析以确定哪些指向现有可复用构件的元素。将这些需求模型中的元素与可复用构件描述进行比较，有时称这个过程为“规格说明匹配”[Bel95]。如果规格说明匹配指出一个现有的构件和现有应用系统需求相符合，那么就可以从可复用构件库中提取这些构件，并将它们应用于新系统的设计中。如果没有发现这样的构件（即没有匹配），就需要创建新构件。在这一点上，当需要创建新构件时，应该考虑可复用性设计（Design For Reuse，DFR）。

正如我们已经谈到的，DFR需要软件工程师采用良好的软件设计概念和规则（第8章）。但是，也必须考虑应用领域的特点。Binder [Bin93]提出了构成可复用设计基础的一系列关键问题[⊖]。

当构件必须与遗留系统及多个系统（它们的体系结构和接口协议不一致）进行接口或集成时，DFR可能会非常困难。

标准数据。应该仔细研究应用领域，并定义标准的全局数据结构（例如，文件结构或完整的数据库）。然后，所有的设计构件会被赋予使用这些标准数据结构的特性。

标准接口协议。应该建立3层接口协议：模块内部接口的基本属性，外部技术（非人）接口设计和人机接口设计。

程序模板。选取一种体系结构风格（第9章），可以作为新软件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，就有了进行设计所依托的框架。符合此框架的新构件将来复用的可能性就比较高。

10.6.4 构件分类与检索

考虑一座大型的大学图书馆，有成千上万的书籍、期刊和其他信息资源可供使用。然而为了访问这些资源，就必须有合适的分类模式。为了浏览这么庞大的信息，图书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、方便地查到所需的资源。

现在，考虑一个大型构件库，其中存放了成千上万的可复用构件。但是，软件工程师如何才能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？这些问题太难，至今还没有明确答案。在本节中，我们讨论当前的研究方向，使未来的软件工程师可以方便地浏览复用库。

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的描述，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。

⊖ 一般来说，DFR准备工作应当是领域工程的一部分。

为了在实际环境中使用，概念、内容和环境必须转换为具体的规格说明模式。关于可复用软件构件的分类模式，已有很多文章（例如，有关当前趋势的概述，参见[Cec06]）。

分类能够使软件工程师发现和检索到候选的可复用构件，但是必须具有能有效集成这些构件的可复用环境。可复用环境具备以下几方面的特点。

构件复用环境的关键特性是什么？

- 能够存储软件构件和检索构件所需分类信息的构件数据库。
- 提供访问数据库的库管理系统。
- 软件构件检索系统（例如，对象请求代理）：允许客户应用系统从构件库服务器中检索构件和服务。
- CBSE工具：支持将复用构件集成到新的设计或实现中。

每种功能都与复用库交互，或是嵌入在复用库中。

WebRef

CBSE方面的广泛资源可以在www.cbd-hq.com/找到。

复用库是更大型软件库（第22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，构件分类模式是构件库查询的基础。

通常用本节前面描述的3C 模型中的环境元素来描述查询。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。

SOFTWARE TOOLS

CBSE

目的：在软件构件的建模、设计、评审以及集成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：[⊖]

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 Solution（www.selectbs.com）开发，“它是一个集成的成套产品，用于软件设计、设计审查、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，对于由OMG模型驱动的体系结构（一个开放的、供应商中立的CBSE方法），它能够运用UML进行广泛的建模。

10.7 小结

构件级的设计过程包含一系列活动，这些活动逐渐降低描述软件的抽象层次。构件级设计最终在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3个不同的角度来进行构件设计。面向对象的视角注重细化来自于问题域和基础设施域的设计类。传统的视角细化3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。在这两种视角中，都需要应用那些能够得到高质量软件的基本设

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

计原则和概念。当从过程视角考虑时，构件设计采用了可复用的软件构件和设计模式，这些都是基于构件级的软件工程的关键要素。

在进行类的细化时，有许多重要的原则和概念可以指导设计者，包括开闭原则、依赖倒置原则，以及耦合性和内聚性概念等，这些都可以指导工程师构造可测试、可实现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：详细描述消息细节，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程，在类或构件层次上表示行为等。在任何情况下，设计迭代（重构）都是重要活动。

传统的构件级设计需要足够详细地表示出程序模块的数据结构、接口和算法，以指导生成用编程语言书写的源代码。为此，设计者采用某种设计表示方法来表示构件级详细信息，可以使用图形、表格，也可以使用文本格式。

WebApp的构件级设计既要考虑内容，也要考虑功能性，因为它是由基于Web的系统发布的。构件级的内容设计关注展示给WebApp最终用户的内容对象及这些内容对象的包装方式。WebApp的功能性设计关注处理功能，这些处理功能可以操作内容、执行计算、查询和访问数据库，并建立与其他系统的接口。所有的构件级设计原则和指导方针都适用。

结构化程序设计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的数量和类型。结构化程序设计的目的是帮助设计师定义简单的算法，使算法易于阅读、测试和维护。

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

习题与思考题

10.1 术语“构件”有时很难定义。请首先给出一个一般的定义，然后针对面向对象软件和传统软件给出更明确的定义，最后选择3种你熟悉的编程语言来说明如何定义构件。

10.2 为什么传统软件当中必要的控制构件在面向对象的软件中一般是不需要的？

10.3 用自己的话描述OCP。为什么创建构件之间的抽象接口很重要？

10.4 用自己的话描述DIP。如果设计人员过于依赖具体构件，会出现什么情况？

10.5 选择3个你最近开发的构件，并评估每个构件的内聚类型。如果要求定义高内聚的主要优点，那么主要优点会是什么？

10.6 选择3个你最近开发的构件，并评估每个构件的耦合类型。如果要求定义低耦合的主要优点，那么主要优点会是什么？

10.7 问题领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？

10.8 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）前几章讨论过的某个SafeHome类的详细状态图。

10.9 逐步求精和重构是一回事吗？如果不是，它们有什么区别？

10.10 什么是WebApp构件？

10.11 选择已有程序的某一小部分(大概50～75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程序设计原则的构造？如果存在，重新设计代码使其遵守结构化编程的构造，如果不违反，对于画出的框你注意到了什么？

10.12 所有现代编程语言都实现了结构化的程序设计构造。用3种程序设计语言举例说明。

10.13 选择有少量代码的构件，并使用下面的工具来描述：（1）活动图；（2）流程图；（3）决策表；（4）PDL。

10.14 在构件级设计的评审过程中，为什么“分块”很重要？

推荐读物与阅读信息

最近几年，已经出版了许多关于构件级开发和构件级复用的书籍，包括：Apperly和他的同事（《Service- and Component-Based Development》，Addison-Wesley，2003）、Heineman 和Councill（《Component-Based Software Engineering》，Addison-Wesley，2001）、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）、Allen、Frost和Yourdon（《Component-Based Development for Enterprise Systems: Applying the Select Perspective》，Cambridge University Press，1998）。这些书覆盖了CBSE过程中的所有重要方面。Cheesman 和 Daniels（《UML Components》，Addison-Wesley，2000）则侧重于用UML讨论CBSE。

Gao和他的同事（《Testing and Quality Assurance for Component-Based Software》，ArtechHouse，2006）和Gross（《Component-Based Software Testing with UML》，Springer，2005）论述了构件级系统的测试和SQA问题。

近些年出版了大量书籍来描述产业界基于构件的标准。这些著作既强调了关于制定标准本身的工作，也讨论了许多重要的CBSE话题。

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，以及关于结构化程序设计分支的细节讨论。其他书籍则集中在传统系统的过程设计问题方面，如Robertson（《Simple Program Design》，3rd ed.，Course Technology，2000）、Farrell（《A Guide to Programming Logic and Design》，Course Technology，1999）、Bentley（《Programming Pearls》，2nd ed.，Addison-Wesley，1999）、Dahl（《Structured Programming》，Academic Press，1997）等。

最近出版的书籍中，专门讨论构件级设计的书很少。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，这方面有成百上千本书。

在网上有大量关于构件级设计的信息源，有关构件级设计的最新WWW参考文献列表可在SEPA Web站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

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：不管软件展示了什么样的计算能力、发布了什么样的内容及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。屏幕布局描述了图标的图形设计和位置，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
命令标签
控制
设计评估
错误处理
黄金规则
帮助设施
界面
分析
一致性
设计
模型
国际化
记忆负担
原则和指导方针
过程
响应时间
任务分析
任务细化
可用性
用户分析
WepApp界面设计

我们生活在充满高科技产品的世界里。几乎所有这些产品，诸如消费电子产品、工业设备、社团系统、军事系统、个人电脑软件及WebApp，都需要人参与交互。如果要使一个产品取得成功，它就必须展示出良好的可用性。可用性是指用户在使用高科技产品所提供的功能和特性时，对使用的容易程度和有效程度的定量测量。

不管界面是为数字音乐播放器设计，还是为战斗机的武器控制系统而设计，可用性都至关重要。如果对界面机制进行了良好的设计，用户可以流畅、顺利地进行交互，使工作变得不费吹灰之力。但如果界面设计得很糟糕，用户使用时断时续、不流畅，最终的结果是，用户会感到很沮丧，且工作效率很差。

在计算时代的前30年里，可用性并不是软件开发者主要关心的。Donald Norman[Nor88]在其关于设计的经典书籍中曾经主张（对待可用性的）态度改变的时机已经到来：

> 为了使技术适应人类，必须要研究人类。但我们现在倾向于只研究技术。结果，人们不得不顺从技术。而现在是扭转这个趋势的时候了，是让技术适应人类的时候了。

随着技术专家对人类交互的研究，出现了两个主要的问题。第一，定义一组黄金规则（11.1节）。这些规则可以应用于所有人类交互的技术产品。第二，定义交互机制使软件设计人员建立起可以恰当实现黄金规则的系统。这些交互

机制消除了与人类界面结合的一些极坏的问题，统一称之为图形用户界面（graphical user interface，GUI）。但即便是在“窗口的世界”，我们还是遇到过这样的用户界面：难学、难用、令人迷惑、不直观、不可原谅，在很多情况下，让人感到十分沮丧。然而，仍然有人在花费时间和精力去创建这样的界面，看起来，创建者并不是有意制造麻烦。

11.1 黄金规则

Theo Mandel在其关于界面设计的著作[Man97]中提出了3条“黄金规则”：

1. 用户操纵控制。
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一系列用户界面设计原则的基础，这些原则可以指导软件设计的重要方面。

“依照用户的习惯来设计，比矫正用户的习惯要好。”——Jon Meads

11.1.1 用户操纵控制

在重要的、新的信息系统的需求收集阶段，曾经征求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，并使我可以非常容易地完成。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做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。Mandel[Man97]定义了一组设计原则，允许用户操纵控制：

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。由于不同的用户有不同的交互偏好，因此应该提供选择机会。例如，软件可能允许用户通过键盘命令、鼠标移动、数字笔、多触摸屏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。例如，考虑使用键盘命令（或语音输入）来画一幅复杂形状的图形是有一定难度的。

“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++发明人)

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。

11.1.2 减轻用户记忆负担

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的动作、输入和结果。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的动作，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联系到相关动作（例如，使用要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的工作间，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：我已经在考虑监控功能的界面了。

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，但是会带来很多开发工作量。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。

Vinod：房主如何记住有多少个摄像头以及它们都安装在什么地方呢？

Jamie（有点不高兴）：他是房主，应该知道。

Vinod：但是如果不知道呢？

Jamie：应该知道。

Vinod：这不是问题的关键……如果忘记了呢？

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，但是为什么要有一份清单呢？
Jamie：好的，无论用户是否有这方面的要求，我们都提供一份清单。
Vinod：这样更好。至少用户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）好吧，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。

"看起来不同的事物产生的效果应该不同，而看起来相同的事物产生的效果应该相同。" —— Larry Marine

11.1.3 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规则来组织可视信息；（2）将输入机制约束到有限的集合，在整个应用系统中得到一致的使用；（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：

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（例如，窗口标题、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系统中均会如此期望。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？"他给出了下面的回答：

"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软件，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[Don99]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？
- 随着用户的知识不断增多，交互机制是否能变得更灵活？
- 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、图标和过程是否在整个界面中一致？
- 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？
- 交互是否简单？

如果上述每个问题的回答都是肯定的，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价、获得良好的口碑、降低支持成本、提升最终用户生产力、降低培训费用、减少文档开销、减少来自不满意用户的投诉。

WebRef

可以在www.useit.com找到用户界面设计信息的优秀资源。

11.2 用户界面的分析与设计

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并最终实现设计模型；最后由最终用户从质量的角度对结果进行评估。

"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

11.2.1 用户界面分析和设计模型

分析和设计用户界面时要考虑4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4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。

ADVICE

即使用户是新手也会有使用快捷键的需求；即使是经常使用系统的用户有时候也需要指导。他们的要求都要满足。

用户模型确立了系统最终用户的轮廓（profile）。JeffPatton[Pat07] 在《用户为中心的设计》的前言中写道：

"事实是，设计者和开发者（包括我自己）都经常考虑到用户。然而，在缺少特定用户有力的心理模型情况下，开发设计人员自我替代。自我替代并不是用户为中心，而是自我为中心。"

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"[Shn04]。此外，可以对用户进行如下分类：

KEY POINT
用户心里模型显示用户对界面的感知以及用户界面是否满足用户的需求。

新手。对系统没有任何语法知识的了解[⊖]，并且对应用系统或计算机的一般用法几乎没有掌握什么语义知识[⊜]。

对系统有了解的间歇用户。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。对应用系统有很好的语义知识和语法知识的了解（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。

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，请某个特定文字处理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验（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。一个对文字处理有深刻了解，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。

实现模型组合了计算机系统的外在表现（界面的观感），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）。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。

“注意用户的行为而不是他们的言语。”——Jakob Nielsen

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]。从本质上看，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：“了解用户，了解任务。”

11.2.2 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2章讨论过的螺旋模型表示。如图11-1所示，用户界面分析和设计过程开始于螺旋模型的内部，且包括4个不同的框架活动[Man97]：（1）界面分析及建模。（2）界面设计。（3）界面构造。（4）界面确认。图11-1中的螺旋意味着每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，构造活动涉及原型开发——这是唯一实用的确认设计结果的方式。

“依照用户的习惯来设计，比矫正用户的习惯要好。”——Jon Meads

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。对每个用户类别，进行需求引导。本质上，软件工程师试图去理解每类用户的系统感觉（11.1.2节）。

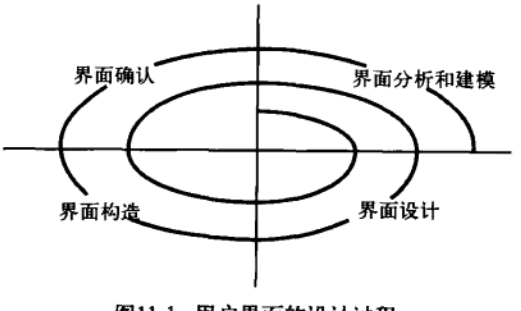


图11-1 用户界面的设计过程

⊖ 在这里，语法知识是指更有效地使用界面的交互机制。

⊜ 语义知识是指对应用程序的基本感知，对执行的功能、输入输出含义及系统目标的理解。

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。11.3节将对任务分析进行更详细的讨论。最后，用户环境的分析着重于物理工作环境。需要问的问题有：

在开始用户界面设计时，关于环境我们应当知道些什么？

- 界面的物理位置如何？
- 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），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11.4节详细讨论。

界面构造通常开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（11.4节）可用来完成界面的构造。

界面确认着重于：(1) 界面正确地实现每个用户任务的能力，适应所有任务变化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) 用户将界面作为其工作中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的细节（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。

11.3 界面分析[⊖]

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执行的任务；（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的基础，我们来检查界面分析的每个成分。

11.3.1 用户分析

在担忧技术上的问题之前，“用户界面”这个词完全有理由需要我们花时间去理解用户。之前，我们提到每个用户对于软件都存在心理想象，而这可能与其他用户开发的心理想象存在着差别。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，这些途径包括：

我们如何知道用户从用户界面上想要什么？

用户访谈：这是获取信息的最直接的办法。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、企业文化和其他问题。可以是一对一的会议方式，也可以是群体讨论的形式。

销售输入：销售人员与用户定期见面，能够收集到有助于软件团队对用户进行分类和更好地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

⊖ 因为需求分析问题在第5、6、7章已经讨论过，所以有理由把这一节放到这几章中去。本节之所以放在这里，是因为界面的分析和设计两者紧紧相连，两者的界限常常模糊不清。

总的来说，花时间与实际的用户进行交谈，但要小心。一个强烈的建议并不代表大多数用户都会赞同。

场每个部分使用软件的细微差别的理解。

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（改编自[Hac98]）将有助于界面设计师更好地理解系统的用户：

我们如何知道最终用户的人数和特征？

- 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集中培训？
- 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？
- 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出错，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，用户界面必须具有哪些特性才能满足用户的需求。

11.3.2 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（工作流）如何？
- 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本书前面所讨论的分析技术，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，参与者通常是某个人）和系统的交互方式。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（一段简单的文字）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：

我从勾画房间的平面图、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，关心房间内活动空间的通道大小。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，由于需要经常移动这些项，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，让客户感受到房间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，系统中能够使得室内设计师感到愉悦的其他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。

SAFEHOME

用户界面设计的用例

[场景] Vinod的工作间，用户界面设计正在进行。

[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：站在谁的角度来写？

Jamie：当然是房主，还会有谁？

Vinod：还有系统管理员这个角色。即使是房主担任这个角色，这也是一个不同的视角。“管理员”启动系统，配置零件，布置平面图，安置摄像头……

Jamie：当房主想看视频时，我只是让她扮演房主的角色。

Vinod：好的，这只是监视功能界面主要行为之一。但是，我们也应该调查一下系统管理员的行为。

Jamie（有些不悦）：你是对的。

（Jamie离开去找销售人员。几个小时以后她回来了。）

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看这个非正式的用例。）

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。当我启动系统时，我选择某个管理功能。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（我可以选择度量系统）。我能够从传感器和摄像头库中进行选择，并且将它们放置在平面图中。我标记每个传感器和摄像头，或者系统自动进行标记。我可以通过合适的菜单对传感器和摄像头进行设置。如果选择编辑，就可以移动传感器和摄像头，添加新的或删除已有的传感器和摄像头，编辑平面图并编辑摄像头和传感器的设置。在每种情形下，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）：好的，对于绘画程序，可能有一些有用的设计模式[第12章]或可复用的图形用户界面构件。我打赌：通过使用可复用构件，我们可以实现某些或大部分管理员界面。

Jamie：同意！我马上进行查看。

任务细化。在第8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

任务细化非常有用，但也很危险。仅仅因为你已经细化了一个任务，没有假定是否有其他方法，当实现用户界面的时候，可能会尝试其他方法。

析采用了一种详细阐述的办法来辅助理解用户界面必须采纳的用户活动。

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活动。为了理解实现活动目标而必须完成的任务，工程师必须明白人们当前所执行的任务（在使用手工方式的时候），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，考虑前面讨论的为室内设计师开发的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）、结构和材料的选择、墙壁和窗户装饰物的选择、展示（对于客户而言）、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) 标出尺寸以显示其位置；(7) 为用户勾画透视图。也可以应用类似的方法对其他主任务进行细化。

上面1～7的每个子任务都可以进一步精化。其中子任务1～6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[⊖]。界面的设计模型应该以一种与用户模型（典型室内设计师的轮廓图）和系统感觉（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。

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。这些对象可以分为不同的类。需要定义每个类的属性，并且通过对每个对象动作的评估为设计师提供一个操作列表。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。室内设计师会从Furniture类中选择对象，将其移到平面图（在此处，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，对每个操作的细节都会进行定义。

尽管对象细化十分有用，但它应当作为独立的方法去使用。在任务分析的过程中，应当考虑用户的声音。

工作流分析。当大量扮演着不同角色的用户使用某个用户界面时，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涉及多个成员（角色）时，工作过程是如何完成的。假如某个公司打算将处方药的开方和给药过程全部自动化。全部过程[⊖]将围绕着一个基于Web的应用系统进行考虑，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。

⊖ 然而，事实可能不是这样。室内设计师可能想要指定所画的透视图、缩放比例、色彩的运用和其他信息。与透视渲染相关的用例将提供解决这些问题的信息。

⊖ 这个例子选自[Hac98]。

下面只考虑工作过程中的一小部分：当病人请求重填处方时发生的情形。图11-2给出了一个泳道图，该图表明了前面提及的3个角色的任务和决定。这些信息可以通过访谈或每个角色书写的用例获取。不管怎样，事件流（图中显示的）使得界面设计师认识到3个关键的界面特征：

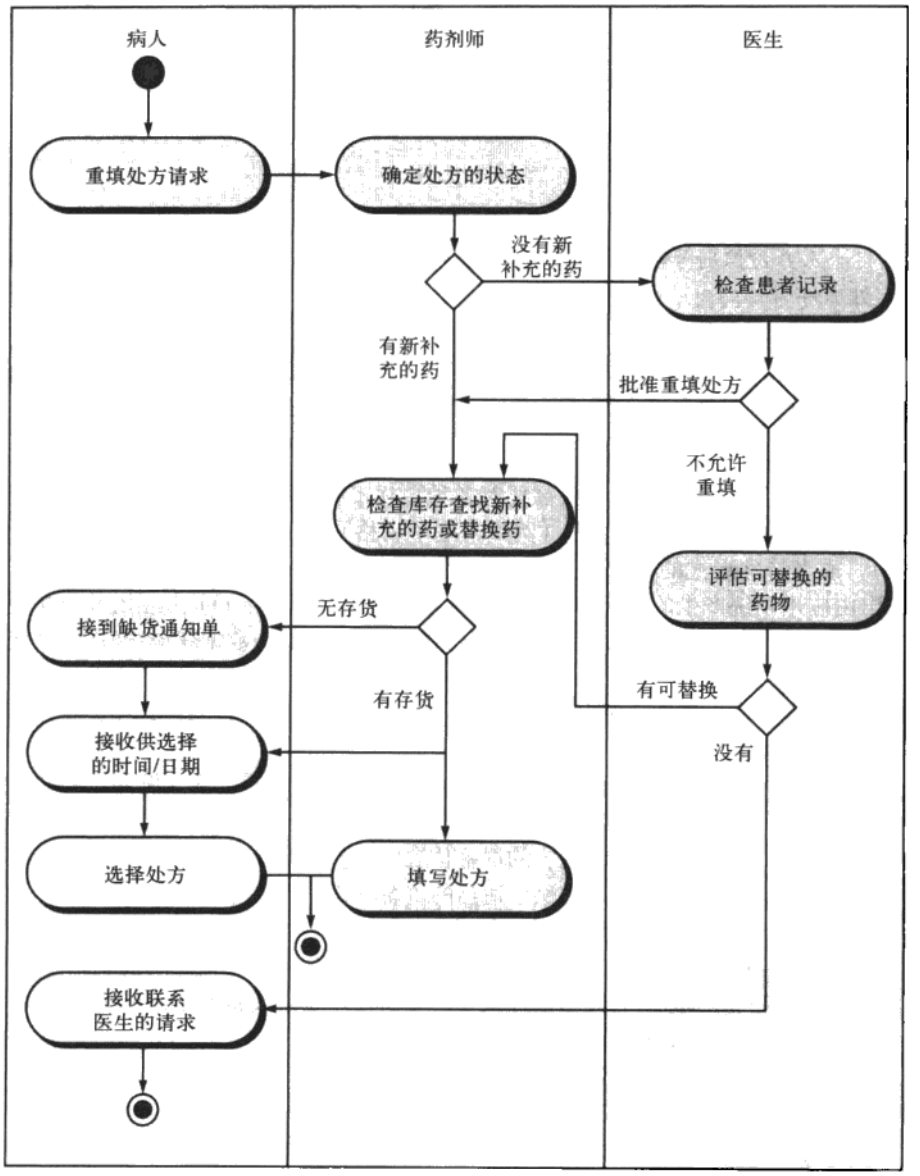


图11-2 处方重填功能的泳道图

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感观上将与为药剂师或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辅助信息源的信息（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药物信息）。

使技术适应用户要比用户适应技术好。——Larry Marine

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，考虑如下用户任务和子任务层次。

用户任务：请求重填处方

- 提供辨识信息
  - 指定姓名
  - 指定用户ID
  - 指定个人身份识别号码（PIN）和密码
- 指定处方序号
- 指定重填处方所需的日期

为了完成填写处方的任务，定义了3个子任务。可以将其中的第一个子任务“提供辨识信息”进一步细化成3个另外的子子任务。

11.3.3 显示内容分析

11.3.2节中标识出的用户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，电子表格）、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6章和第7章中讨论过的分析建模技术标识出由应用系统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的构件（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。其中需要提问和回答的问题包括：

作为界面设计的一部分，我们如何决定所显示内容的格式和美感？

- 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？

对这些（和其他）问题的回答有助于软件工程师建立起内容表示的需求。

11.3.4 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能适应环境，那么它们会难于使用或者使用起来不方便。”

在某些应用系统中，计算机系统的用户界面被放在“用户友好”的位置（例如，合适的亮度、良好的显示高度、简单方便的键盘操作），但有些地方（例如，工厂的地板和飞机座舱）

亮度可能不是很适合，噪音也可能是个问题，也许不能选择使用键盘或鼠标，显示方位也不甚理想。界面设计师可能会受到某些因素的限制，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。可否采用某种方式（例如，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？在界面设计开始之前，应该对上述问题和更多的相关问题给予回答。

11.4 界面设计步骤

"交互设计是图形艺术、技术和心理学的无缝结合。"——Brad Wieners

一旦完成了界面分析，最终用户要求的所有任务（对象和动作）都已经被详细确定下来，界面设计活动就开始了。与所有的软件工程设计一样，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（例如，[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（11.3节），定义界面对象和动作（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），并对行为建模。
3. 描述每个界面状态，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（例如，在各种环境下用户界面看起来是什么样子），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11.1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（例如，显示技术、操作系统、开发工具）。

11.4.1 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对象上的动作。为了完成这个目标，需要使用类似于第6章介绍的方法来分析用户场景，也就是说，撰写用例的描述。名词（对象）和动词（动作）被分离出来形成对象和动作列表。

一旦完成了对象和动作的定义及迭代细化，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。将源对象（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系统特有的数据，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被拖动和删除。

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。与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（由房主写的）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），我可以决定报警系统的状态、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，我需要提供标识符和密码，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。一旦确认了身份，我就可以检查系统状态，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以提供房子内部的不同视角。

基于这个用例，确定房主的任务、对象和数据项如下：

- *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*平面图和传感器位置。
- *显示*平面图上的区域。
- *改变*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（每秒4帧）。
- *摇动*或*变焦*摄像头。

尽管自动化的工具在开发布局原型中十分有用，但有时候铅笔和纸也是需要的。

从房主的这个任务清单中抽取出对象（黑体）和动作（斜体）。所提到的大部分对象是应用系统对象。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（视频显示的窗口）。

为视频监控设计的屏幕布局初步草图如图11-3所示[⊖]。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

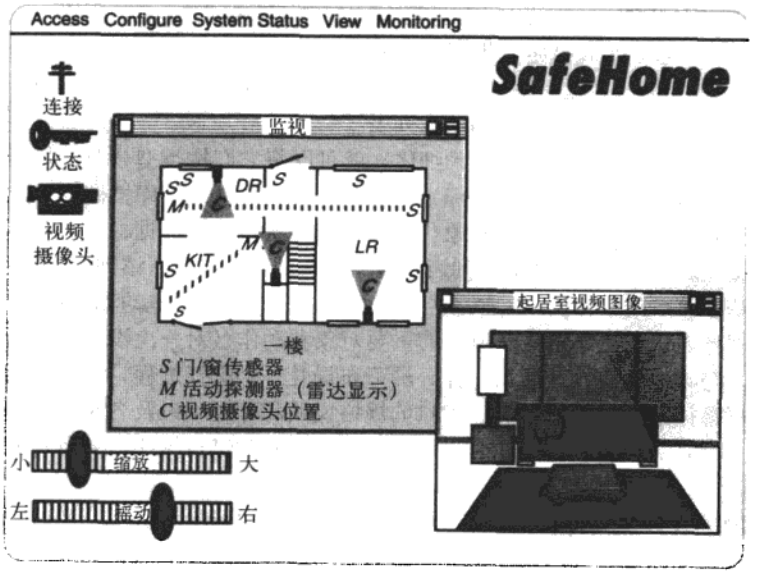


图11-3 基本的屏幕布局

⊖ 注意这里与前几章讲到的这些特性的实现有所不同。这里应该是第一次设计的草图，可以考虑提供备选的设计草图。

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。变焦和摇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。

11.4.2 用户界面设计模式

图形用户界面已经变得如此普遍，以至于涌现出各式各样的用户界面设计模式。正如本书前面提到的，设计模式是一种抽象，描述了特定的、界限明确的设计问题的设计解决方案。

WebRef

已经提出了大量用户界面设计模式，访问www.hcipatterns.org可以找到大量模式的站点链接。

作为通常碰到的界面设计问题的一个例子，考虑用户必须一次或多次输入日历日期这种情况，有时候需要提前输入月份。对于这个简单的问题，有很多可能的解决方案，为此也提出了很多种不同的模式。Laakso[Laa00]提出了一种称为CalendarStrip的模式，此模式生成一个连续、滚动的日历，在这个日历上，当前日期被高亮度显示，未来的日期可以在日历上选择。这个日历隐喻在用户中具有很高的知名度，并提供了一种有效的机制，可以在上下文环境中设置未来的日期。

在过去的十年间，人们已经提出了很多用户界面设计模式。在第12章中有关于用户界面设计模型的更为详尽的论述。此外，Erickson[Eri08] 提供了许多基于Web的文献资料。

11.4.3 设计问题

在进行用户界面设计时，几乎总会遇到以下4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、项目拖延及用户的挫折感，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。

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这段时间。

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。

"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智慧。" —— Douglas Adams

帮助设施。几乎所有计算机交互式系统的用户都时常需要帮助。有时，简单的问题问一下同事就可以解决，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。

考虑帮助设施时需要在设计中解决如下问题[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3种选择：帮助菜单、特殊功能键或HELP命令。

- 如何表达帮助？有3种选择：提供单独的帮助窗口、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的返回按钮、功能键或控制序列。
- 如何构造帮助信息？有3种选择：平面结构（所有信息均通过关键词来访问）、分层结构（用户可以进入分层结构得到更详细的信息）和超文本的使用。

"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

错误处理。出错信息和警告是指出现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。几乎所有的用户都会遇到以下错误信息："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。

"好的"错误提示信息应当具备哪些特性？

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（或者在已经出现的情况下进行改正）。
- 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，无论出错消息设计得多么好，都很少有用户喜欢。但是，当问题真的出现时，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）和选取（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。在提供命令或菜单标签交互方式时，必须考虑以下问题：

- 每个菜单选项是否都有对应的命令？
- 以何种方式提供命令？有3种选择：控制序列（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的？
- 子菜单是否与主菜单项所指功能相一致？

WebRef

开发可访问软件的指导原则可以在www3.ibm.com/able/guidelines/software/accesss of tware.html找到。

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表示删除一个图形对象，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。

应用系统的可访问性。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。对于那些身体上面临挑战的用户（和软件工程师）来说，由于道义、法律和业务等方面的原因，可访问性是必须的。多种可访问性指导方针（如[W3C03]）——很多都是为Web应用系统设计的，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。其他指南（如[App08]，[Mic08]）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应付出的努力和技能。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（例如，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（例如，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软件工程师只需做有限的定制开发就可以建立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用户界面。

机制：现代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（例如，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：[⊖]

LegaSuite GUI，由Seagull Software（www.seagullsoftware.com）开发，能够创建基于浏览器的图形用户界面（GUI）并且提供了对过时界面的再造功能。

Motif Common Desktop Environment，由Open Group（www.osf.org/tech/desktop/cde/）开发，是一个集成的图形用户界面，用于开放系统桌面计算。它对数据、文件（图形化桌面）和应用系统的管理提供了单一的、标准的图形化界面。

Alita Design 8.0，由Altia（www.altia.com）开发，是一种可以在多种平台（例如，自动的、手持的、工业的）上创建图形用户界面（GUI）的工具。

11.5 WebApp界面设计

用户若可以从内容层次级别的任何地方进入WebApp，确保在每个页面都设计导航特性，这样用户就可以找到他们感兴趣的内容。

无论是为WebApp、传统的软件应用系统、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本章之前所讲的特性。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[⊜]；（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

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。

⊜ 我们每个人在网页上加入书签，以便今后访问，但它并不代表网站或本页的上下文内容（也不能移动到网站的其他位置）。

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。

11.5.1 界面设计原则与指导方针

“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

WebApp的用户界面是它的“第一印象”。不管它的内容、处理能力和服务以及WebApp的整体效益如何，一个设计糟糕的用户界面将会使潜在的用户失去信心。事实上，用户甚至可能转向使用别的WebApp，因为几乎在每个主题领域内，WebApp的竞争都是十分激烈的，用户界面应当迅速“抓住”潜在的用户。

Bruce Tognozzi [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。

KEY POINT

一个良好的WebApp界面是易于理解的、宽容的，给用户可操控感。

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。

在设计图形用户界面时是否要遵循一些基本的原则？

为了设计具有这些特性的界面，Tognozzi[Tog01]确定了一组非常重要的设计准则[⊖]：

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。再比如，一个带有黄色三角形的对象表明是一条警告信息，这种警告信息发生在用户调用特殊的功能或动作之前，则在WebApp的其他任何地方不能以其他的目的使用。最后，界面每个特征的反应方式都应该与用户的期望相一致[⊖]。

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

⊖ 本书对Tognozzi的最初准则进行了修改与扩展。这些原则的进一步讨论参见[Tog01]。

⊖ Tognozzi[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）。

WebApp的Web工程师的效率，也不是优化运行WebApp的客户/服务器环境的效率。Tognozzi[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。"

"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

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。

关注点——WebApp界面（及界面表示的内容）应该关注在用户正在完成的任务上。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，并且会错过当初最希望看到的内容所在的站点。

WebRef

在Web上进行查找会发现很多有用的库，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20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。

隐喻是一种出色的想法，因为隐喻能够反映现实世界的经验。只是要确保你选择的隐喻是最终用户所熟悉的。

每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"，就以下拉菜单的形式显示产品类别（例如，音响设备、电视机、DVD播放器），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。

人机界面对象——对于WebApp，已经开发了大量可复用的人机界面对象库。使用这些对象库，能被最终用户"看到的、听到的、接触到的以及别的方式感知到的"[Tog01]任何界面对象都能从大量对象库中的任何一个获得。

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；（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）。

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。

隐喻——只要隐喻适合应用系统和用户，使用交互隐喻的界面就更容易学习和使用。隐喻应该采用用户熟悉的图片和概念，但是并不要求是现实生活的精确再现。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。虽然隐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。用户界面应该支持此项功能，并给用户提供简单的方法来恢复“丢失”的信息。

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了，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。

Doug：它们是主要功能，对吗？

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。我认为这样更合理。

Doug：我同意。关于这个问题你和Sharon谈过了吗？

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：

“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

- 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。
- 避免“正在构建中”的标志——一个不必要的链接，这样的标志让人感到失望。
- 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。

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。

11.5.2 WebApp的界面设计工作流

本章前面曾经提到用户界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。

需求模型包含的信息构成了创建屏幕布局的基础，屏幕布局描述图标的图形设计和放置、描述性屏幕文本的定义、窗口标题定义及规格说明、主菜单和子菜单项目的规格说明。接着使用工具创建原型，并最终实现用户界面模型。下面的任务代表了WebApp界面设计的基本工作流程：

1. 对需求模型中的信息进行评审，并根据需要进行优化。

2. 开发WebApp界面布局的草图。界面原型（包含布局）可能已经作为分析建模活动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工程团队应该与利益相关者合作完成开发。图11-4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。应该将这些目标映射到特定的界面行为，如图11-4所示。实际上，界面设计人员必须回答下面的问题："界面是如何让用户完成每个目标的？"

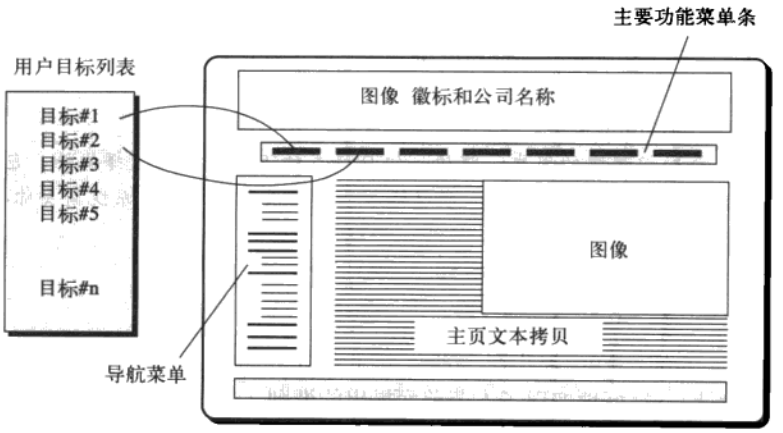


图11-4 将用户目标映射到特定的界面行为

4. 定义与每个行为相关的一组用户任务。每个界面行为（例如，购买商品）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建立对应关系，这些交互包括导航事件、内容对象和WebApp功能。

5. 为每个界面行为设计情节故事板屏像（storyboard screen images）。当考虑每种行为时，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（即使它们还没有设计和开发），展示WebApp功能并确定导航链接。

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。在大多数情况下，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而不是技术专家完成的。美学设计（第13章）集成于界面设计工作当中。

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（附录1）描述用户与WebApp交互时的活动（和决定）流。

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（附录1）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，把确定的布局和屏幕图像与任务8中描述的每个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。

11.6 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）。

用户界面评估的循环如图11-5所示。完成设计模型后就开始建立第一级原型；用户对该原型进行评估[⊖]，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）；针对用户的意见对设计进行修改，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。

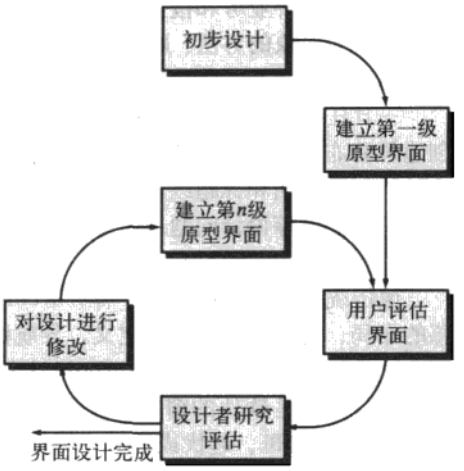


图11-5 界面设计评估循环

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，就可以减少评估循环执行的次数，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. 系统及其界面的需求模型或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. 指定用户任务的个数以及每个任务动作平均数在一定程度上表示了系统的交互时间和系统的总体效率。

3. 设计模型中动作、任务和系统状态的数量反映了用户学习系统时所要记忆内容的多少。

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，可以把问卷分发给原型用户，每项提问的答案可以是（1）简单的“是”或“否”；（2）数字；（3）（主观的）等级；（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期研究分析。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。

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[Hac98]和[Sto05]等文献。

11.7 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）用户操纵控制；（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。

⊖ 有必要注意由人类工程学和界面设计方面的专家们也对界面进行审查。这些审查叫做启发评估或认知走查。

用户界面的开发首先从一系列的分析任务开始。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。环境分析可弄清界面必须操作的物理结构和社会结构。

一旦任务被确定下来，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。

像传统软件的界面设计一样，WebApp界面的设计表示了用户界面的组织结构，表现了屏幕的布局，是对交互模式的定义，导航机制的描述。当设计布局和界面控制机制时，界面设计原则和界面设计工作流为WebApp设计者提供了向导。

用户界面是软件的窗口。在很多情况下，界面塑造了用户对系统质量的感知。如果这个“窗口”污点斑斑、起伏或损坏，用户也许会选择其他更有效的计算机系统。

习题与思考题

11.1 描述一下你操作过的最好和最不好的系统界面，采用本章介绍的相关概念对其进行评价。

11.2 在11.1.1节的基础上，再给出两条“用户操纵控制”的设计原则。

11.3 在11.1.2节的基础上，再给出两条“减轻用户记忆负担”的设计原则。

11.4 在11.1.3节的基础上，再给出两条“保持界面一致”的设计原则。

11.5 考虑下面几个交互应用程序（或者导师布置的应用程序）：

a. 桌面发布系统。

b. 计算机辅助设计系统。

c. 室内设计系统（如11.3.2节所描述的）。

d. 大学课程自动注册系统。

e. 图书管理系统。

f. 基于网络的公共选举投票系统。

g. 家庭银行系统。

h. 导师布置的交互应用系统。

对上面给出的每个系统，开发用户模型、设计模型、心理模型和实现模型。

11.6 选择习题11.5中所列的任何一个系统，使用细化或面向对象的方法进行详细任务分析。

11.7 在11.3.3节提供的内容分析列表中至少再添加5个问题。

11.8 继续做习题11.6，对于你所选择的应用系统，定义界面对象和动作。确定每个对象类型。

11.9 对于在习题11.5中所选的系统，开发一组带有主菜单和子菜单项定义的屏幕布局。

11.10 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11-3所示的方法。

11.11 对于在习题11.5到习题11.8中所完成的任务分析设计模型和分析任务，描述你采用的用户帮助设施方法。

11.12 举例说明为什么反应时间变动是一个问题。

11.13 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。

11.14 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，并在班上做介绍。

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，但其中涉及了进行有效设计的哲学，可以应用于用户界面的设计。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、汽车的电子仪表板、Web站点，或者是在商业应用系统中，用户界面都为软件提供了窗口。正因如此，关于界面设计的书籍有很多，Butow（《User Interface Design for Mere Mortals》，Addison-Wesley，2007）、Galitz（《The Essential Guide to User Interface Design》 3rd ed.，Wiley, 2007）、Lehikonen 和他的同事（《Personal Content Experience：Managing Digital Life in the Mobile Age》，Wiley-Interscience, 2007），Cooper和他的同事（《About Face 3: The Essentials of Interaction Design》，3rd ed., Wiley，2007）、Ballard（《Designing the Mobile User Experience》，Wiley，2007）、Nielsen（《Coordinating User Interfaces for Consistency》，Morgan-Kaufmann，2006）、Lauesen（《User Interface Design：A Software Engineering Perspective》，Addison-Wesley，2005）、Barfield（《The User Interface：Concepts and Design》，Bosko Books，2004）编写的这些书都讨论了可用性、用户界面概念、原则和设计技巧，并且包含了很多有用的例子。

早一些的书籍如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）还提供了设计准则和原则，以及用户需求引导、设计建模、实现和测试的建议。

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提供了有用的指导。Cooper编写的受欢迎的书（《The Inmates Are Running the Asylum》，Sams Publishing，1999）讨论了为什么高技术产品常令人感到疯狂，以及如何设计不让人疯狂的产品。

在网上有大量关于用户界面设计的信息源，有关用户界面设计的最新WWW参考文献列表可在SEPA Web站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第12章

Software Engineering：A Practitioner's Approach (Seventh Edition)

基于模式的设计

要点浏览

概念：对于清晰描述的一组问题，基于模式的设计通过查找一组已被证明有效的解决方案来创建新的应用系统。每个问题及其解决方案都用一个设计模式来描述，事实上，在此之前曾有其他软件工程师在设计其他应用系统时，已经遇到了这个问题并实现了解决方案，因此将这个设计模式编入目录并做了检验。每个设计模式都提供了一种已证明的方法，用于解决问题的一部分。

人员：软件工程师在遇到一个新的应用问题时，通过搜索一个或多个模式存储库来寻找相关的解决方案。

重要性：你是否听说过“重复发明车轮子”这样的话？这种情况在软件开发领域一直都存在，既浪费时间也浪费精力。通过使用已经存在的设计模式，可以获得特定问题的已被证明的解决方案。随着每种模式的应用，解决方案被集成到完整的设计方案中，所构建的应用系统就向着完整的设计又迈进了一步。

步骤：为了划分要解决的一系列问题的层次，需要检查需求模型。将问题空间分割，可以确定与软件功能和特性相关的问题子集。还可以对这些问题进行如下分类：体系结构、构件级、算法、用户界面等。一旦定义了问题子集，就查找一个或多个模式库，以确定在适当的抽象层次上是否有设计模式存在。可以针对待构建软件的特定要求，对选择使用的设计模式进行修改。如果找不到合适的模式，则可以自定义问题的解决方案。

工作产品：开发的设计模型用来描述体系结构、用户界面以及构件级的细节。

质量保证措施：由于每个设计模式都要被转化成设计模型的一些要素，因此要对工作产品进行评审，以检查清晰性、正确性、完整性、与需求的一致性以及工作产品之间的一致性。

关键概念

设计错误
影响因素
框架
粒度
模式语言
模式
体系结构
行为
构件级
有生产力的
创建型
结构型
用户界面
Web应用

我们每个人都遇到过设计问题，并且会想：是否已经有人研究出了这个问题的解决方案？对这个问题的回答几乎总是肯定的！问题是要找到解决方案，确保此方案适合解决所遇到的问题，理解限制此解决方案应用方式的约束，最后，将建议的解决方案应用到软件开发的设计环境中。

但是，如果以某种方式规范解决方案会怎么样呢？如果有某种描述问题的标准方式（使人们可以查找），并用有组织的方法来描述问题的解决方案结果会如何呢？结果是可以用标准化的模板来规范和描述软件问题，并提出了问题（连同约束）的解决方案。所谓设计模式，是用来描述问题以及解决方案的规范化方法，在某种程度上允许软件工程界获取设计知识，使解决方案能够得到重用。

软件模式的前身（模式）不是由计算机科学家首先提出来的，是由建筑师Christopher Alexander提出来的，他发现在设计建筑物时总会遇到一系列重复的问题，于是他把这些重复出现的问题以及解决方案定义为模式，可以用下面的方式来描述模式[Ale77]：

每个模式都描述了在我们所处环境内反复出现的问题，然后描述该问题的核心解决方案，这样你就可以几百万次地重复使用该解决方案，而不必用同样的方式重复工作两次。

Gamma[Gam95]、Buschmann[Bus96]以及他们的许多同事首次将Alexander的思想引入到软件领域的书籍中[⊖]，现在已经有很多模式存储库，基于模式的设计可以用于很多不同的应用领域。

12.1 设计模式

KEY POINT
影响因素是限制以某种方式进行设计的那些问题的特征及解决方案的属性。

设计模式可以描述为“表示特定上下文、问题和解决方案三者之间关系的三部规则”[Ale79]。对于软件设计，上下文使读者理解问题所发生的环境，以及在此环境中什么样的解决方案才适合。一组需求（包括限制和约束）起到影响因素的作用，会影响如何在此问题的上下文环境中对问题进行解释以及如何有效地应用解决方案。

“我们的责任是尽力去做、去学习，改进解决方案，并传递下去。”——Richard P. Feynman

为了更好地理解这些概念，可以设想这样的情形[⊜]：一名旅客在纽约和洛杉矶之间旅行。在这样的背景下，旅行是在一个工业化的国家（美国）发生，旅客使用已有的交通设施（例如，公路、航空、铁路）。影响这个旅行问题解决方式的系统影响因素包括：这名旅客想在多短的时间内从纽约到达洛杉矶，此行是否包括观光或中途停留，他准备了多少旅费，此行是否有特殊目的，以及这名旅客是否想使用自己的交通工具等。给定了这些影响因素，就可以很好地定义这个旅行问题（从纽约到洛杉矶）。例如，调查（需求收集）说明这名旅客的钱很少，只有一辆自行车（是一个喜欢骑自行车的人），希望此行可以为她热衷的慈善事业募集资金，这名旅客有很多空闲时间。给定了上下文环境和系统影响因素，这个问题的解决方案可能是越野自行车旅行。如果影响因素不同（例如，旅行时间要最少，旅行的目的是商业会议），那么可能会有其他更合适的解决方案。

有理由认为大多数问题都有多种解决方案，但是只有适合问题所处环境的解决方案才是有效的。是影响因素促使设计者去选择某一特定的解决方案。目的是提供一个最能满足影响因素的解决方案，即使这些影响因素是矛盾的。最后要说明的是，每种解决方案都有各自的结果，这些结果会影响软件的其他方面，并且在较大的系统中，对于要解决的其他问题，这些结果本身又会成为影响因素的一部分。

Coplien [Cop05]用下面的方法描述了有效的设计模式的特点：

- **设计模式可以解决问题。**模式可以捕捉解决方案，不只是抽象的原则或策略。
- **设计模式是已经得到验证的概念。**模式借助于踪迹记录捕捉解决方案，而不是根据理论或猜测。
- **解决方案并不明显。**很多问题解决技术（例如，软件设计范型或方法）都试图从基本原理导出解决方案。最好的模式会间接产生问题的解决方案——对于大多数困难的设计问题而言，这是必要的方法。
- **设计模式描述关系。**模式不只是描述模块，而且描述更深层的系统结构和机制。
- **模式具有显著的人性元素**（将人工干预降到最少）。所有软件都是为改善人类的生活舒适程度或生活质量而服务的；最好的模式明确要求具有美学和实用性。

⊖ 早期关于软件模式的讨论确实存在，但是这两本经典的书首次对这个主题进行了集中论述。

⊜ 这个例子根据[Cor98]改写。

更实际地说，一个良好的设计模式以某种方式捕获来之不易并且实用的设计知识，使别人重用这些知识“百万次，而不必用同样的方式重复工作两次。”设计模式可以使你免于“重复发明车轮子”，或更糟的是发明的“新车轮子”不是很圆，对于其使用目的来说又太小，对于行驶的路面来说太窄。如果能有效地使用设计模式，你一定会成为更好的软件设计师。

12.1.1 模式的种类

软件工程师对设计模式感兴趣甚至着迷的原因之一是由于人类天生善于模式识别。如果不是这样，我们就会停滞在空间和时间中——不能吸取过去的经验，不愿冒险前进，因为我们无法识别那些可能引起高风险的情况，而且让我们发疯的是，世界似乎没有规律或逻辑一致性。幸运的是，不会有这些情况发生，因为在现实生活的各个方面，实际上，我们都能识别模式。

在现实中，我们识别的模式来自人生经验。我们可以立即识别并理解模式的含义以及如何使用它。有些模式可以使我们深入了解重复出现的现象。例如，你正在下班回家的州际公路上，导航系统或是收音机通知你，在相反方向的路上发生了一起严重事故。你距离事故发生地点4公里，但是已经看到交通放缓，这种模式称为RubberNecking。旅游车道的人慢慢向你的方向移动过来，以便看清楚高速路的对面究竟发生了什么事情。RubberNecking模式很明显会产生可以预料的结果（交通堵塞），但它只是描述现象而已。用模式的术语来说，称为无生产力的模式，因为它只是描述问题和背景，但不提供明确的解决方案。

KEY POINT
“有生产力”的模式描述了问题、环境和影响因素，但它也描述了问题的实际解决方案。

考虑软件设计模式时，我们总是力图识别和记载有生产力（generative）的模式。也就是说，我们识别能描述系统中重要的和可重复方面的模式，这样的模式在影响因素（对于给定的上下文环境是独一无二的）内为我们提供了一种方式来构造重要的和可重复的方面。在理想的环境下，一组有生产力的设计模式能够用于“产生”一个应用系统或基于计算机的系统，其体系结构能够使系统适应变更。有时我们称之为生产性，“就是连续应用几种模式，每种模式都封装自己的问题和影响因素，更大的解决方案是以一些较小的解决方案的形式间接地表现出来的”[App00]。

设计模式所涉及的抽象和应用的范围很广。体系结构模式描述了很多可以用结构化方法解决的设计问题。数据模式描述了重现的面向数据的问题以及用来解决这些问题的数据建模解决方案。构件模式（也认为是设计模式）涉及与开发子系统和构件相关的问题、它们之间相互通信的方式以及它们在一个较大的体系结构中的位置。界面设计模式描述公共用户界面问题及具有影响因素（包括最终用户的具体特征）的解决方案。Web应用模式解决构建Web应用时遇到的问题，而且往往包括很多前面提到的一些其他模式。在较低的抽象层，习惯用语（idiom）描述如何在特定的编程语言环境下实现软件构件的全部或部分特定算法或数据结构。

? 有办法划分模式的类型吗?

Gamma和他的同事[⊖][Gam95]在关于设计模式的具有重大影响的书中，着眼于与面向对象设计相关的3种模式：创建型模式、结构型模式及行为型模式。

创建型模式着眼于对象的“创建、组合及表示”。Gamma和他的同事[Gam95]指出，创建型模式“封装了有关系统使用的具体类的知识”，但同时“隐藏了如何创建和组合这些类的实例”。创建型模式提供了一种机制，使对象实例在一个系统内更容易生成，并坚持“在一个系统内创建的对象类型及数量方面的约束”[Maa07]。

⊖ 在模式文献中，Gamma和他的同事[Gam95]常被称为“四人帮”（GoF）。

结构型模式着眼于有关如何将类和对象组织和集成起来，以创建更大结构的问题和解决方案。本质上，结构型模式是帮助建立系统内实体之间的关系。例如，侧重于面向类问题的结构型模式可以提供继承机制，以产生更有效的程序接口。侧重于对象的结构型模式提出了在其他对象内组合对象的技术以及将对象集成为更大结构的技术。

行为型模式解决与对象间任务分配以及影响对象间通信方式的有关问题。

INFO

创建型模式、结构型模式及行为型模式

已经提出了适合于创建型、结构型和行为型类别的大量设计模式，并可以在网上找到。Wikipedia（www.wikipedia.org）记录了下面的模式：

创建型模式

- **抽象工厂模式**：集中决定实例化什么工厂。
- **工厂方法模式**：集中创建某一特定类型的对象，并从几种实现中选择其中的一种。
- **生成器模式**：将一个复杂对象的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。
- **原型模式**：对给定的应用程序，如果用标准方法（例如，使用“新的”关键字）创建新对象的固有成本过于昂贵时，就使用原型模式。
- **单例模式**：限制类实例只有一个对象。

结构型模式

- **适配器模式**：将一个类的接口转换成客户希望的另外一个接口。
- **聚集模式**：是组合模式的一个版本，采用将孩子聚集在一起的方法。
- **桥接模式**：将抽象部分与它的实现部分分离，使两者可以独立地变化。
- **复合模式**：复合模式就是一个具有同样接口的处理对象的树结构的模式。
- **容器模式**：创建对象的唯一目的就是装载其他对象并管理它们。
- **代理模式**：一个类的作用是作为另一个类的接口。
- **管道和过滤器**：是一个过程链，每个过程的输出是下一个过程的输入。

行为型模式

- **责任链模式**：对命令对象进行处理，或者通过逻辑包含的处理对象传递给其他对象进行处理。
- **命令模式**：命令对象把行动和参数封装起来。
- **事件监听器**：把数据分配给对象，这些对象已经注册使之接收数据。
- **解释器模式**：实现一种特殊的计算机语言，以迅速解决一组特定的问题。
- **迭代器模式**：迭代器用于按顺序访问一个聚集对象中的元素，而不必暴露其底层表示。
- **中介者模式**：对于子系统中的一组接口，提供一个统一的接口。
- **访问者模式**：将算法从一个对象中分离出来的方法。
- **单次访问者模式**：对于分配的访问者，优化其实现。优化实现只使用一次，然后删掉。
- **层次访问者模式**：提供一种方式，访问层次数据结构（例如，树）上的每个节点。

可以通过链接www.wikipedia.org获得每个模式的全面描述。

12.1.2 框架

模式本身可能不足以开发一个完整的设计。在某些情况下，可能需要为设计工作提供与实现相关的架构基础设施，称为框架。也就是说，可以选择“一个可重用的微型体系结构，此体

KEY POINT
框架是可以重复使用的“微型体系结构”，可以作为应用其他设计模式的基础。

系结构在某种上下文环境下为一簇软件抽象提供了通用的结构和行为……并在给定的领域内规定了这些结构和行为之间的协作和使用”[Amb98]。

框架不是体系结构模式，而是一个具有“插入点”（也称为钩子和插槽）集合的架构，可以适应特定的问题域。插入点使得软件工程师能够在架构内集成特定问题的类或功能。在面向对象的环境下，框架是相互协作的类的集合。

Gamma和他的同事[Gam95]是这样描述设计模式和框架之间的区别的：

1. 设计模式比框架更抽象。框架可以用代码表示出来，但只有模式的举例才可以用代码表示出来。框架的长处是可以用编程语言编写，不仅可以用来研究，也可以直接执行和重用……

2. 设计模式是比框架更小的体系结构元素。一个典型的框架包括一些设计模式，而设计模式却不包括框架。

3. 对设计模式的研究要比框架少。框架总是有特定的应用领域。与此相反，设计模式几乎可以用在任何类型的应用问题中。当然很可能有更多专用的设计模式，即使这些设计模式不能确定一个应用系统的体系结构。

实际上，框架设计师认为，在限定的应用领域内，可重用的微型体系结构适用于所有软件的开发。为了最有效，框架可以不进行任何修改而直接使用。可能还需要增加更多的设计元素，但设计师只能通过插入点来充实框架。

12.1.3 描述模式

基于模式的设计开始于识别要构建的应用系统中的模式，然后进行查找，确定别人是否已经论述过这个模式，最后采用适合于自己当前问题的模式。其中第二个任务常常是最困难的。那么怎样才能找到所需要的模式呢？

对这个问题的回答取决于以下4个方面的有效交流，即模式要解决的问题，模式所在的环境，影响环境的影响因素及所提出的解决方案。为了清楚无误地交流这些信息，需要描述模式的标准格式或模板。虽然已经提出了很多不同的模式模板，几乎所有的模式模板都包含Gamma和他的同事[Gam95]所建议的主要内容。简化的模式模板如下面框内所示。

INFO

设计模式模板

模式名称——以简短但富于表现力的名字描述模式的本质

问题——描述模式涉及的问题

动机——提供问题的实例

环境——描述问题所在的环境，包括应用领域

影响因素——列出影响问题解决方式的全部影响因素，包括必须要考虑的限制和约束的讨论

解决方案——提供问题解决方案的详细描述

目的——描述模式以及模式所做的工作

协作——描述其他模式对解决方案的贡献

效果——描述实现模式时必须考虑的可能要做的折中以及使用模式的效果

实现——当实现模式时，确定应该考虑的特殊问题

已知应用——提供了设计模式在实际应用系统中的使用实例

相关模式——相关设计模式的交叉索引

"模式是不成熟的——意味着需要自己去完成，并使之适应你自己的环境。" ——Martin Fowler

应该小心选择设计模式的名称。基于模式的设计的关键技术问题之一是在成千上万在候选模式中无法找到现有的模式。有意义的模式名称非常有助于搜索"正确"的模式。

模式模板为描述设计模式提供了标准化的方法。每个模板项都表现了要查找的设计模式的特征（例如，通过数据库查找），使得能够找到合适的模式。

12.1.4 模式语言和存储库

当提到语言这个词时，我们首先会想到的是自然语言（例如，英语、西班牙语、汉语），还是程序设计语言（例如，C++、Java）。这两种语言都有语法和语义，用于以有效的方式交流思想或过程指令。

语言这个术语用于设计模式环境时意义稍有不同。模式语言包括模式集合，每个模式都用标准化的模板（12.1.3节）来描述并相互关联，以显示这些模式如何相互合作来解决应用领域中的问题[⊖]。

ADVICE 如果找不到模式语言解决你的问题域，可以在另外的模式集中查找类似的模式语言。

在自然语言中，将单词组织成句子来表达意思。句子的结构通过语言的语法来描述。在模式语言中，以某种方式来组织设计模式，这种方式提供了"在特定的领域内描述良好设计实践的结构化方法"[⊜]。

某种程度上，模式语言类似特定应用领域内问题解决的超文本指令手册。首先按层次来描述所考虑的问题域，从领域相关的宏观设计问题开始，然后将宏观问题细化成较低的抽象层次。实质上，在软件环境中，宏观设计问题倾向于体系结构问题，它涉及应用系统的总体结构以及服务于系统的数据或内容。将体系结构问题细化为更低的抽象层次，从而导出解决子问题的设计模式，以及在构件（或类）级的相互协作。模式语言不是模式的顺序列表，而是一个内部相互连接的集合，在这个集合中，用户从宏观设计问题开始，并"向下挖掘"以发现特定问题及其解决方案。

WebRef 有用的模式语言列表参见c2.com/ppr/titles.html。补充信息可以从hillside.net/patterns/获得。

对于软件设计，已经提出了很多模式语言[Hil08]。在大多数情况下，设计模式是模式语言的一部分，并存储在从网上可以访问的模式存储库中（例如，[Boo08]，[Cha03]，[HPR02]）。模式库提供了所有设计模式的索引，以及使用户理解模式间协作的超链接。

12.2 基于模式的软件设计

任何领域的顶级设计人员都具有一种神奇的能力，他们能看出表明问题特征的模式，并将相应的多个模式组合起来形成解决方案。关于这一点，微软[Mic04]的软件开发人员这样写道：

虽然基于模式的设计在软件开发领域中还是较新的事物，但在工业技术领域，基于模式的设计已经用了几十年甚至几百年。机械目录和标准配置提供了设计元素，这些设计元素可以用来设计汽车、飞机、机床以及机器人等。在软件开发中，应用基于模式的设计给软件带来的好处同给工业技术带来的好处一样：可预测性、减少风险以及提高生产率。

在整个设计过程中，（当模式符合设计要求时）应该利用一切机会去寻找现成的设计模式，而不是去创建新模式。

⊖ 模式语言最初是Christopher Alexander为建筑结构和城市规划提出来的。现在，模式语言已经发展到从社会科学到软件工程等各个方面。

⊜ Wikipedia关于模式语言的描述，请参见http://en.wikipedia.org/wiki/Pattern_language。

12.2.1 不同环境下基于模式的设计

基于模式的设计不是脱离现实的。前面所讨论的体系结构设计、构件级设计及用户界面设计（第9章到第11章）的概念和技术都是与基于模式的方法联合使用的。

第8章所提到的一系列质量指导方针和属性可以作为所有软件设计决策的基础。决策本身是受许多基本设计概念（例如，关注点分离、逐步求精、功能独立）和最佳实践（例如，技术、建模表示方法）影响的。这些基本设计概念是利用经过了几十年演变的启发法获得的；作为构造的基础，所提出的最佳实践使设计更容易实施，也更为有效。

图12-1说明了基于模式的设计的步骤。软件设计师从描述系统抽象表示的需求模型（明确的或隐含的）开始工作。需求模型描述问题集合、建立上下文环境并明确主要影响因素。需求模型只是用抽象的方式暗示了设计，但并不能明确表示设计。

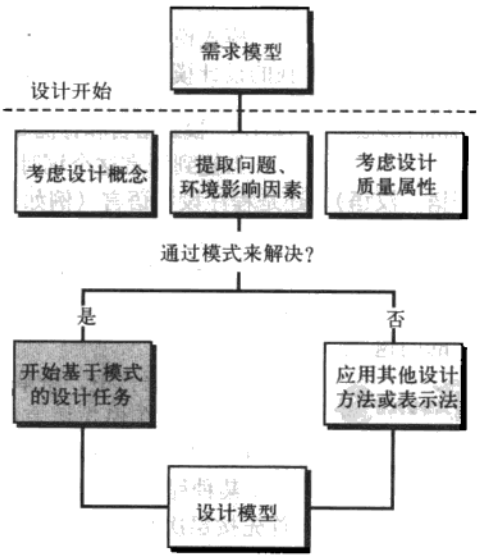


图12-1 不同环境下的基于模式的设计

当软件设计师开始工作时，牢记质量属性总是很重要的。这些属性（例如，设计必须实现在需求模型中所有明确的需求）建立了评估软件质量的方法，但对实际获得软件质量的帮助很小。所创建的软件设计应该展示出在第8章中讨论的基本设计概念。因此，要应用成熟的技术把需求模型中的抽象表示转化成更具体的形式，这就是软件设计。为了完成这项任务，应利用可以得到的体系结构设计、构件级设计及接口设计方法和建模工具，但只有当以前还没有解决面临的问题、环境及系统影响因素时才这样做。如果解决方案已经存在，就直接使用！这就意味着应用了基于模式的设计方法。

12.2.2 在模式中思考

Shalloway和Trott[Sha05]在其关于基于模式设计的优秀著作中，当模式作为设计活动的一部分时，他们提出了“一种新的思考方式”：

我要采取一种新的思考方式。当我这样做时，我听见Christopher Alexander说：“简单地将执行部分累加起来，并不能获得好的软件设计”。

好的软件设计是从考虑环境（全局）开始的。当评价环境时，需要提炼出要解决问题的层次结构。其中的一些问题是全局性质的，其他问题要解决的是软件的特定性质和功能。影响因素会影响所有这些问题以及要提出的解决方案的性质。

基于模式的设计关心要解决的问题。我怎样开始工作？

为了使设计者用模式思考，Shalloway和Trott[Sha05]提出了下面的方法[⊖]：

1. 保证理解全局——将要建立的软件所处的环境。需求模型表达了这一点。
2. 检查全局，提取在此抽象层上表示的模式。
3. 从“全局”模式开始设计，为将来的设计工作建立环境或架构。
4. “在环境的内部工作”[Sha05]在更低的抽象层上寻找有助于设计方案

⊖ 此方法基于Christopher Alexander[Ale79]的工作。

的模式。

5. 重复步骤1到步骤4，直到完成完整的设计。

6. 通过使每个模式适应将要建立的软件细节对设计进行优化。

注意到模式不是独立的实体是很重要的。表示在高抽象层的设计模式常常影响在较低抽象层的其他模式的应用方式。另外，模式间也经常相互协作。含义是——当选择一个体系结构模式时，它对所选择的构件级设计模式有很大影响。同样的，当选择一个特定的接口设计模式时，有时不得不使用与之协作的其他模式。

举例说明，考虑Web应用SafeHomeAssured.com。如果想要考虑全局，这个Web应用必须解决许多基本问题，例如：

- 如何提供关于SafeHome产品和服务的信息？
- 如何将SafeHome产品和服务卖给顾客？
- 如何建立基于网络的监控，并对已安装的安全系统进行控制？

可以将每个基本问题进一步细化为一系列更小的子问题。例如，如何通过因特网进行销售？这个问题就隐含了使用**电子商务**（E-commerce）模式，电子商务模式本身又意味着许多较低抽象层上的模式。**电子商务**模式（可能是一个体系结构模式）意味着建立客户账户、展示要销售的产品、选择购买的产品等。因此，如果用模式思考，重要的是，要确定是否存在建立账户的模式。在问题所处的环境下，如果建立账户（SetUpAccount）是一个可以使用的可行模式，这个模式可能要与其他模式进行协作，例如BuildInputForm、ManageFormsInput及ValidateFormsEntry。每个模式都描述了要解决的问题以及可能应用的解决方案。

12.2.3 设计任务

当使用基于模式的设计思想时，实施的设计任务如下：

创建基于模式的设计需要哪些任务？

1. **检查需求模型，并开发问题的层次结构。**通过分离问题、环境及相关的影响因素，描述每个问题及子问题。然后，从宏观问题（在较高的抽象层）到更小的子问题（在较低的抽象层）进行处理。

2. **针对问题域，确定是否已经开发了可靠的模式语言。**如在12.1.4节谈到的那样，模式语言涉及与特定应用领域相关的问题。SafeHome软件组应该去寻找专门为家庭安全产品开发的模式语言。如果不能找到那个级别的模式语言，软件组应该将SafeHome软件问题划分成一系列通用问题域（例如，数字设备监控问题，用户界面问题，数字视频管理问题），并查找合适的模式语言。

3. **从宏观问题开始设计，确定是否存在一个或多个体系结构模式是适用的。**如果一个体系结构模式是可用的，那么一定要检查所有相互协作的模式。如果这个模式合适，就对所提出的设计方案进行修改，并构建一个充分代表此模式的设计模型元素。正如在12.2.2节提到的，SafeHomeAssured.com Web应用的宏观问题是用**电子商务**（E-commerce）模式解决的。这个模式为解决电子商务的需求提出了一个特定的体系结构。

4. **使用为体系结构模式所提供的协作，检查子系统或构件级问题，并查找合适的模式解决这些问题。**有必要在其他模式库及与体系结构解决方案相对应的模式列表中进行查找。如果找到了合适的模式，就对所提出的设计方案进行修改，并构建一个充分代表此模式的设计模型元素。一定要应用步骤7。

5. **重复步骤2到步骤5，直到所有的宏观问题都得到了解决。**即从全局开始，并逐渐在更详细的层次上解决问题。

6. **如果已经分离出了用户界面设计问题（几乎总是这种情况），那么为了找到合适的模式，**

查找多个用户界面设计模式库。用类似于步骤3、步骤4和步骤5的方式继续进行。

7. 不考虑抽象层的情况下，如果模式语言和（或）模式库或单个的模式是存在的，那么将要解决的问题和现存的模式进行比较。检查环境和影响因素，以确保模式提供的解决方案与问题相吻合。

8. 当从模式导出设计后，使用设计质量标准作为指南，确保对设计进行优化。

虽然上面列举的设计方法是自顶向下的，但是现实的设计方案有时要复杂得多。关于这一点，Gillis[Gil06]是这样论述的：

软件工程中的设计模式是以演绎和推理的方式使用的。设一个通用问题或需求是X，Y模式可以解决X，因此就使用模式Y。从整个过程来考虑，我相信我在这里并不孤单——我发现更有组织，更多归纳而不是演绎，更多是自底向上而不是自顶向下。

显然，需要获得一种平衡。当项目处于初始引导阶段时，我尝试着从抽象的需求跳到具体的设计方案，经常执行一种广度优先搜索……我发现设计模式有助于用具体的术语快速地构思设计问题。

另外，基于模式的设计方法必须与其他软件设计概念和技术配合使用。

ADVICE

可以用有关模式的适应性说明来补充表中的条目。

12.2.4 建立模式组织表

随着基于模式的设计的进展，对来自许多模式语言和模式库中的候选模式进行组织和分类是很麻烦的。为了帮助对候选模式的评估进行组织，Microsoft[Mic04]建议创建一张模式组织表，其一般格式如图12-2所示。

	数据库	应用	实现	基础设施
数据/内容				
问题陈述	模式名称		模式名称	
问题陈述		模式名称		模式名称
问题陈述	模式名称			模式名称
体系结构				
问题陈述		模式名称		
问题陈述		模式名称		模式名称
问题陈述				
构件级				
问题陈述		模式名称	模式名称	
问题陈述				模式名称
问题陈述		模式名称	模式名称	
用户界面				
问题陈述		模式名称	模式名称	
问题陈述		模式名称	模式名称	
问题陈述		模式名称	模式名称	

图12-2 模式组织表

资料来源：改自[Mic04]

通过采用图中所示的形式，模式组织表可以作为电子表格模型实现。问题陈述的简单列表被组织为数据/内容、体系结构、构件级以及用户界面问题，显示在左（阴影）列。上面一行

列出了数据库、应用、实现和基础设施这4种模式类型。表中的格里注明了候选模式的名称。

为了提供组织表的条目，需要在模式语言和模式库中查找解决特定问题陈述的模式。当找到一个或多个候选模式时，找到问题陈述所对应的行，以及模式类型对应的列，填入模式名称。模式的名称作为超链接填入，链接到包含此模式完整描述的Web地址URL。

12.2.5 常见设计错误

基于模式的设计使你成为更好的软件设计师，但它不是灵丹妙药。像所有的设计方法一样，需要从基本原理开始，强调软件质量基础，并确保设计真正满足由需求模型所表达的要求。

即使模式能解决手头的问题，也不要急于使用。如果环境和影响因素是错的，就要去寻找另一个模式。

在使用基于模式的设计时，会发生很多常见的错误。在某些情况下，由于没有足够的时间去理解根本问题、问题所在环境及影响因素，所以，可能会选择看似正确但并不适合解决方案所要求的模式。一旦选择了错误的模式，就会拒绝正视自己的错误，并强迫去适合模式。在另外的情况下，你所选择的模式没有考虑到问题的某些影响因素，导致了所选择的模式不理想或者是错误的。有时，只照着字面的意思去应用模式，而没有针对具体问题对模式进行修改。

这些错误能够避免吗？在大多数情况下，答案是“肯定的”。好的设计师会寻找其他意见，并欢迎对其工作进行评审。评审技术在第15章讨论，对于要解决的软件问题，评审技术有助于确保应用基于模式的设计能够得到高质量的解决方案。

12.3 体系结构模式

一个软件体系结构可能有很多体系结构模式涉及诸如并发性、持久性及分布性等问题。

如果房屋建造者决定建造一个具有旧式建筑风格的中央门厅式建筑，那么应用的建筑风格是唯一的。建筑风格的细节（例如，壁炉的数目，房子的外观以及门窗的位置等）可以有变化，但是一旦房屋的整体结构确定下来了，那么设计的风格就随之确定了。[⊖]

体系结构模式与房屋建筑结构有点区别。例如，每户都要使用厨房（Kitchen）模式。厨房模式以及与它合作的其他模式能涉及的问题有：食物的存储与准备，完成这些任务所需要的工具，以及这些与工作流程相关的工具在房间中的放置规则等。另外，该模式还能涉及的有关问题有：厨房台面、照明、墙壁开关、中心岛、室内地面等问题。显然，对于一个厨房，会存在多种设计，通常取决于环境和影响因素。但是，可以在厨房模式所建议的“解决方案”的环境内构思每个设计。

典型的体系结构模式领域是什么？

如前所述，软件的体系结构模式定义了特定的方法处理系统的某些特性。Bosch[Bos00]和Booch[Boo08]定义了许多体系结构模式领域。有代表性的实例如下：

访问控制。在很多情况下对于由某应用系统所提交的数据、特征及功能的访问要受限于一些特定的最终用户。从体系结构的观点看，必须严格控制访问软件体系结构的某些部分。

并发性。很多应用程序必须以模拟并行的方式来处理多任务（例如，当由一个处理器管理多个“并行”任务或部件时，会发生这种情况）。一个应用程序有很多不同的方法来解决并发问题，每种方法由不同的体系结构模式表示。例如，一种方法使用了**操作系统进程管理**

⊖ 这意味着将有中央大厅和走廊，房间在大厅的左边和右边，房屋由两（或更多）层，卧室在楼上，等等。一旦决定采用旧时建筑风格的中央大厅，这些“规则”就确定了。

(OperationgSystemProcessManagement) 模式，这个模式提供了内置的操作系统特性，允许部件并发执行。该模式包含的操作系统的功能还有：管理进程间的通信、调度以及为实现并发性所需要的其他能力。另一种方法是在应用程序级别定义任务调度。**任务调度** (TaskScheduler) 模式包含一组活动对象，每个对象都包含tick()操作[Bos00]。调度程序周期性地调用每个对象的tick ( ) 操作，然后执行相应的功能，必须在控制权回到调度程序前执行该操作，然后调度程序调用下一个并发对象的tick ( ) 操作。

分布性。分布性问题关系到在分布式环境中系统或系统内部件相互通信的方式。这里要考虑两个子问题：(1) 实体间联系的方式，(2) 发生通信的性质。最常见的解决分布式问题的体系结构模式是**代理模式** (Broker pattern)。代理的作用是在客户端构件和服务器端构件间担当“中间人”。客户端把信息发送给代理 (包括完成通信时所需的全部信息)，然后代理完成连接。

持久性。如果数据在创建它的进程运行结束之后仍然要存在，则数据是持久的。持久性数据存储在数据库或文件中，以后可能会被其他进程读取或修改。在面向对象的环境中，持久性对象的思想对持久性概念做了进一步扩展。为了以后检索或使用，所有对象属性的值、对象总的状况，以及其他补充信息都要保存起来。通常，要获得持久性，可以使用两种体系结构模式——**数据库管理系统** (DatabaseManagementSystem) 模式或**应用级持久性** (ApplicationLevelPersistence) 模式。数据库管理系统模式将DBMS的存储功能和检索功能应用到应用系统的体系结构中，应用级持久性模式则在应用系统体系结构的内部构造持久性特征 (例如，字处理软件管理它本身的文档结构)。

对于上面段落中列出的有代表性的体系结构模式，在选择其中任何一个之前，必须评估它在以下方面的适合程度：应用系统、总体的体系结构风格、模式指定的环境及影响因素。

INFO

设计模式存储库

在网上有很多设计模式的原始资料。一些模式可以从独立发布的模式语言中得到，另一些模式可以从模式门户或模式库中得到。下面的网络原始资料值得一看：

Hillside.net (http://hillside.net/patterns/)。提供最全面的模式和模式语言集的网站之一。

Portland Pattern Repository (Portland模式库，http://c2.com/ppr/index.html)。包含各种模式资源和模式集的链接。

Pattern Index (模式索引，http://c2.com/cgi/wiki?PatternIndex)。一个“选择的模式集”。

Booth's Architecture Patterns Handbook (Booch的体系结构模式手册，www.booch. com/architecture/index.jsp)。数百个体系结构和构件设计模式的参考书目。

用户界面模式集

UI/HCI Patterns (UI/HCI模式，www.hcipatterns.org/patterns.html)。

Jennifer Tidwell's UI Patterns (Jennifer Tidwell的用户界面模式，www.time-tripper.com/uipatterns/)。

Mobile UI Design Patterns (移动用户界面设计模式，http://patterns.littlespringsdesign.com/wikka.php?wakka=Mobile)。

模式

Pattern Language for UI Design（用户界面设计的模式语言，www.maplefish.com/todd/papers/Experiences.html）。

Interaction Design Library for Games（游戏的交互设计库，www.eelke.com/research/usability.html）。

UI Design Patterns（用户界面设计模式，www.cs.helsinki.fi/u/salaakso/patterns/）。

专业的设计模式

Aircraft Avionics（机载航空电子设备，http://g.oswego.edu/dl/acs/acs/acs.html）。

Business Information Systems（商业信息系统，www.objectarchitects.de/arcus/cookbook/）。

Distributed Processing（分布式处理，www.cs.wustl.edu/~schmidt/）。

IBM Patterns for e-Business（电子商务的IBM模式，www128.ibm.com/ developer works/patterns/）。

Yahoo! Design Pattern Library（雅虎！设计模式库，http://developer. yahoo.com/ypatterns/）。

WebPatterns.org（http://webpatterns.org/）。

12.4 构件级设计模式

构件级设计模式可以提供验证了的解决方案，这些解决方案可以解决从需求模型中提取的一个或多个子问题。在很多情况下，这种类型的设计模式关注系统的某些功能元素。例如，SafeHomeAssured.com应用系统必须解决下面的设计子问题：我们如何得到SafeHome设备的产品规格说明及相关信息？

在阐明了必须要解决的子问题后，现在应该考虑影响解决方案的环境和影响因素。通过研究合适的需求模型用例，会发现消费者通过使用SafeHome设备（例如，安全传感器或照相机）的规格说明来获取信息，但是，当选择了电子商务功能后才能使用与规格说明有关的其他信息（例如，定价）。

子问题的解决方案包括搜索。既然搜索是一个很常见的问题，所以有很多与搜索有关的模式并不奇怪。通过检查很多模式库，会找到如下的模式，以及每个模式解决的问题：

AdvanceSearch——用户必须在大量的项里找到特定项。

HelpWizard——用户在查找与网站有关的某一主题或者当用户想要在网站找到某一特定网页时需要帮助。

SearchArea——用户必须找到一个网页。

SearchTips——用户需要知道如何控制搜索引擎。

SearchResults——用户必须处理一个搜索结果的列表。

SearchBox——用户必须找到一项或特定的信息。

对于SafeHomeAssured.com而言，产品的数目不是特别大，且每个产品有一个相对简单的分类，所以AdvanceSearch和HelpWizared也许不必要。同样，搜索很简单，不需要SearchTips。下面给出了关于SearchBox的部分描述：

SearchBox

(根据www.welie.com/patterns/showPattern.php?patternID=search改写。)

问题：用户需要找到一项信息或特定信息。

动机：在任何情况下，关键词搜索都作用于组织成网页的内容对象集合。

环境：用户不会用导航去获取信息或内容，而是在包含很多网页的内容里直接搜索。所有网站都有主导航系统。用户可能想在一个类里搜索一项，也可能想进一步指定一个查询。

影响因素：网站有主导航系统。用户可能想在一个类里搜索一项，也可能通过用简单的布尔运算符进一步指定一个查询。

解决方案：搜索功能是由搜索标签，关键词字段、筛选程序（如果适用）以及“go”（搜索）按钮构成的。按回车键和选择搜索按钮的功能是一样的。同时在一个单独的网页上提供了搜索提示和示例。进入这个网页的链接就放在搜索功能的旁边。编辑搜索词的编辑框足够容纳3个典型的用户查询（典型的约20个字符）。如果筛选条件多于2个，可以用组合框选择，否则用单选按钮选择。

搜索结果显示在新的网页上，这个网页上有清除标签，而且至少含有“搜索结果”或类似的信息。位于网页顶部、带有输入关键词的搜索功能可以重复执行，使用户知道关键词是什么。

在12.1.3节中继续描述模式的其他条目。

接下来，模式要描述如何访问、表示、匹配搜索结果等。在此基础上，SafeHome Assured.com团队需要设计实现搜索的构件，或（更可能）去获取现有的可重用构件。

SAFEHOME

应用模式

[场景] SafeHomeAssured.com通过因特网实现传感器控制的软件增量设计的非正式讨论。

[人物] Jamie(负责设计)和Vinod (SafeHomeAssured.com首席系统架构师)。

[对话]

Vinod：照相机控制界面的设计进展得怎样？

Jamie：还好。没有太多问题。我已经完成了大部分连接实际传感器的功能，开始考虑用户界面，从远程网页实际移动、转动、变焦相机，但是我不能肯定我没弄错。

Vinod：你有没有想出什么？

Jamie：嗯，要求是这样的，照相机控制器需要高交互性——当用户移动控制器时，照相机也要尽快地移动。所以，我在想用一组按钮排列成普通的照相机的样子，但当用户点击按钮时，就控制了照相机。

Vinod：嗯。是的，它会工作，但我不确定会正确地工作——每次点击一个控制按钮时，你需要等待整个客户-服务器通信程序实现，所以你无法得到快速反馈的良好感觉。

Jamie：我也是这么想的——正是我不喜欢这种方法的原因，但我不确定是否还有别的做法。

Vinod：为什么不用InteractiveDeviceControl模式！

Jamie：嗯，那是什么？我没听说过。

Vinod：你所描述的问题基本上就是一个模式。它提出的解决方案基本上是建立服务器与设备的一个控制连接，通过发送控制命令完成，而不用发送通常的HTTP请求。模式还可以指导你如何用一些简单的AJAX技术实现连接。一些简单的客户端Java脚本就可以直接和服务器通信并发送命令，让用户尽快完成任务。

Jamie：太棒了！那正是我需要用来解决这件事的。在哪能找到它？

Vinod：可以在联机库中得到。这里有网址。

Jamie：我会去查查看。

Vinod：是的——但是要记得去检查模式的结果字段。我似乎记得有关安全问题的某些事情需要特别小心。因为你建立了一个单独的控制通道，所以绕过了通常的网络安全机制。

Jamie：有道理！我可能还没有想到那一点！谢谢。

12.5 用户界面设计模式

近年来，已经提出了数百个用户界面(UI)模式。Tidwell[Tid02]和vanWelie[Wel01]描述了10类模式（用典型实例讨论[⊖]），大部分模式属于下列10类模式之一：

Whole UI（整个用户界面）。为最高级结构及整个用户界面的导航提供设计指导。

模式：TopLevelNavigaion

概述：用于网站或应用程序完成一些主要功能。提供了一个高级菜单，商标或图形标识常常与菜单一起出现，通过菜单可以直接导航到系统的主要功能。

详细资料：主要功能（一般仅限于4到7个功能名）用一条水平的文字横列在显示器的上部（也可能是垂直纵列格式）。每个功能名提供链接到相应的功能或信息源。通常使用后面讨论的BreadCrumbs模式。

导航元素：每个功能或内容的名称代表连接相应功能或内容的链接。

Page layout（页面布局）。处理页面（网站的）或清晰的屏幕显示（交互应用程序的）一般组织结构。

模式：CardStack

概述：当必须按任意顺序选择与特征或功能相关的一些特定子功能或内容分类时，使用该模式。该模式提供了大量附有翼片的选项卡，用鼠标点击可以选择选项卡，每个选项卡代表特定的子功能或内容分类。

详细资料：选项卡是容易理解的隐喻，且易于用户操作。每个选项卡的格式可能有些不同。一些选项卡可能需要输入信息，会有按钮或其他导航机制；另一些选项卡可能是提供信息的；该模式可结合DropDownList，Fill-in-the-Blanks等其他模式一起使用。

导航元素：用鼠标点击标签，就会显示相应的选项卡。选项卡的导航功能也可能同时出现，但是通常情况下，这些只是初始化与选项卡数据相关的功能，并不连接到其他显示。

Forms and inout（表单和输入）。考虑完成表单级输入的各种设计技术。

模式：Fill-in-the-Blanks

概述：将字母数字数据输入到“文本框”中。

详细资料：可以将数据输入到文本框中。一般地，通过选取某些文本或图形指示器（例如，包括“搜索”、“提交”、“下一个”等按钮）可以验证并处理数据。很多情况下，该模式与下拉列表（drop-down list）或其他模式（例如，SEARCH<drop down list>FOR<fill-in-balnks text box>）结合起来使用。

导航元素：文本或图形指示器启动验证并处理数据。

Tables（表）。为创建和处理各种表格数据提供设计指导。

⊖ 这里用的是一个扼要的模式模板。完整的模式描述（连同数十个其他模式）可以在[Tid02]和[Wel01]中找到。

模式：SortableTable

概述：显示一长串记录，这些记录可以按选择的任意列的标签排序。

详细资料：表中的每行表示一条完整的记录。每列表示记录的一个字段。每列的标题实际上是一个可选按钮，通过选择这些按钮，所有的记录可以按该列升序或降序排列显示。表的大小通常是可以改变的，而且如果记录的数目超过了可显示的窗口空间，就会出现滚动条。

导航元素：每列的标题都可以对全部记录启动某种排序。虽然在一些情况下，每条记录本身可能就包含了连接到其他内容或功能的导航链接，但没有提供其他导航。

Direct data manipulation（直接数据操作）。处理数据编辑、修改及变换。

模式：BreadCrumbs

概述：当用户使用复杂层次的页面或显示屏时，会提供一个完整的导航路径。

详细资料：每个页面或显示屏都有唯一的标识符。对每个显示，导航到当前位置的路径是预先定义好的。路径的形式是：主页>主要主题页>次主题页>特定页>当前页面。

导航元素：BreadCrumbs显示内的任意输入项都可用作链接回更高层次的一个指针。

Navigation（导航）。辅助用户在层次菜单、网页及交互式显示屏中导航。

模式：EditInPlace

概述：对于显示出的某些类型的内容，该模式提供了简单的文本编辑功能，而不需要用户明确指定文本编辑功能或状态。

详细资料：用户在显示屏上看到需要修改的内容。用鼠标在显示的内容上双击，就是向系统表示希望编辑该内容。该内容则突出显示，表示处于可编辑状态，用户就可以进行适当的修改了。

导航元素：无。

Searching（搜索）。可以在网站或包含持久性数据的数据库中搜索特定内容，可通过交互式应用程序访问持久性数据。

模式：SimpleSearch

概述：该模式可在网站或持久性数据源中搜索简单数据项，该数据项是由字符数字串描述的。

详细资料：该模式在局部（一个网页或一个文件）或全局（整个网站或全部数据库）搜索字符串。按照满足用户要求的可能性大小的顺序，会产生一个“命中”列表。该模式不提供多个数据项的搜索或特殊的布尔运算（见高级搜索模式）。

导航元素：命中列表的每个条目都代表一个链接，可链接到条目所引用的数据。

Page elements（页面元素）。实现网页或显示屏的具体元素。

模式：Wizard

概述：带领用户逐步完成复杂的任务，通过一系列简单的窗口显示，为完成任务提供指导。

详细资料：一个典型的实例是包括4个步骤的注册过程。Wizard模式为每步产生一个窗口，每步同时向用户请求特定信息。

导航元素：在wizard过程中，用户可通过上一步和下一步导航重新访问每一步。

E-commerce（电子商务）。针对网站，这些模式能实现电子商务应用系统的重复元素。

模式：ShoppingCart

概述：列出了要购买的选项清单。

详细资料：列出项目、数量、产品代码、可供性（有现货、无现货）、价格、交付信息、运费及其他相关的购买信息。同时也提供了编辑功能（例如，删除、修改数量）。

导航元素：包括继续购买或前往结账。

Miscellaneous（杂项）。有些模式并不容易归入上述分类之一。在某些情况下，这些模式依

赖于领域或者只出现在特定的用户类中。

模式：ProgressIndicator

概述：当操作所花费的时间比n秒长时，该模式可以表示操作的进度。

详细资料：可用动画图标或信息框表示进度，其中包括表示处理正在进行的一些视觉指示器（例如，一个旋转的“理发店招牌”，一个指示进度百分比的滑块）。也可能包含文本内容指示器，用来表示处理状态。

导航元素。通常是允许用户中止或取消处理的一个按钮。

前面提到的每个模式的例子（每一类的所有模式）都有一个完整的构件级设计，包括设计类、属性、操作以及界面等。

关于用户界面模式的全面讨论超出了这本书的范围。如果想进一步了解，请参看[Duy02]、[Bor01]、[Tid02]及[Wel01]。

12.6 WebApp设计模式

通过这一章的学习，我们了解到模式具有不同的类型及不同的分类方法。在建立WebApp的过程中，当考虑必须要解决的设计问题时，侧重于二维的模式分类值得考虑，即以模式为焦点的设计以及粒度的级别。设计焦点标识着设计模型的哪个方面是相关的（例如，信息体系结构、导航、交互作用）。粒度则标识着要考虑的抽象级别（例如，该模式是否适用于整个WebApp、单个网页、子系统或单独的WebApp构件）。

12.6.1 设计焦点

焦点变得“越窄”、设计就更深入。

在前面几章中，强调了设计过程首先要从考虑体系结构、构件级问题及用户界面表示开始。在每一步，都是在高层抽象级上开始考虑问题并提出解决方案的，然后慢慢地变得更详细和具体。另一种说法是，随着设计的深入，设计的焦点就会变得更“窄”。在为WebApp设计信息体系结构时要遇到的问题（解决方案）与在执行界面设计时遇到的问题（解决方案）是不同的。因此，针对不同级别的设计焦点来开发WebApp设计模式，并不奇怪，这样可以处理在每个抽象级别上遇到的独特问题（及相关的解决方案）决。可按以下设计焦点的级别对WebApp模式进行分类：

- **信息体系结构模式**涉及信息空间的总体结构及用户与信息进行交互的方式。
- **导航模式**定义导航链接结构，例如层次、环、导航路径等。
- **交互模式**有助于用户界面的设计。这类模式解决界面是如何将特定动作的结果通知给用户，用户如何基于使用环境和用户期望扩展内容，如何更好地描述一个链接所暗示的目的地，如何通知用户正在进行的交互的状态，以及与界面有关的问题等。
- **表示模式**通过展示给用户的界面辅助内容的表示。这类模式处理如何组织用户界面控制功能以获得更好的可用性，如何表现界面动作与所影响的内容对象间的关系，以及如何建立有效的内容层次。
- **功能模式**定义了工作流、行为、过程、通信以及WebApp中的其他算法元素。

大多数情况下，在交互设计中遇到问题时，去研究信息体系结构模式的集合是白费力气的。这时应该去研究交互模式，因为交互模式才是与目前工作有关的设计焦点。

12.6.2 设计粒度

当我们要解决的问题涉及“全局”时，可以试着开发关注全局的解决方案（并使用相关模

式)。相反，当焦点很窄时（例如，从包含5项或更少元素的集合中选择一项），解决方案（及相应模式）的目标也很窄。根据粒度的级别，模式可以按下面的级别描述：

- **体系结构模式**。这个抽象级别通常与定义WebApp总体结构的模式有关，表示不同构件或增量间的关系，为指定体系结构元素（网页、包、构件、子系统）间的关系定义规则。
- **设计模式**。设计模式处理特定的设计元素（例如构件的聚合）来解决一些设计问题，网页上元素间的关系，或影响构件间通信的机制。例如，Broadsheet模式可用于WebApp主页的布局。
- **构件模式**。这个抽象级别与WebApp的个别小规模元素有关。这方面的例子有：个别交互元素（例如，单选按钮），导航元素（例如，如何格式化链接）或功能性元素（例如，特定算法）。

对不同类别的应用问题或领域定义不同模式的相关性也是可能的。例如，（处于不同级别的设计焦点和粒度的）模式集合可能与电子商务特别相关。

INFO

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（www.designpattern.lu.unisi.ch/）
- Tom Erickson提出的交互模式（www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html）。
- Martijn vanWelie提出的Web设计模式（www.welie.com/patterns/）。
- 用于UI设计的Web模式（http://harbinger.sims.berkeley.edu/ui_designpatterns/webpatterns2/webpatterns/home.php）。
- 用于个人网站的模式（www.rdrop.com/%7Ehalf/Creations/Writings/Web.patterns/index.html）。
- 用导航模式改善Web信息系统（http://www8.org/w8-papers/5b-hypertext-media/improving/improving.html）。
- HTML2.0模式语言（www.anamorph.com/docs/patterns/default.html）。
- Common Ground——用于HCI设计的模式语言（www.mit.edu/~jtidwell/interaction_patterns.html）。
- 个人网站的模式（www.rdrop.com/~half/Creations/Writings/Web.patterns/index.html）。
- 索引模式语言（www.cs.brown.edu/~rms/InformationStructures/Indexing/Overview.html）。

12.7 小结

设计模式为描述问题及其解决方案提供了一种机制，允许软件工程组织获取可重用的设计知识。一个模式描述了一个问题，使用户能够理解问题所处的环境，并列出了影响因素，用来表明在环境中是如何解释这个问题的，以及是如何应用解决方案的。在软件工程的工作中，我们识别有生产力的模式，并将其制成文档，该模式用来描述一个系统的重要且可重复的方面，然后在给定环境是唯一的影响因素内，提供一种方式构建这个可重复的方面。

体系结构模式描述了广泛的设计问题，这些问题是用结构化方法来解决的。数据模式描述

了面向递归数据的问题以及解决这些问题的数据模型解决方案。构件模式（也称设计模式）解决与子系统和构件开发相关的问题，还有相互之间通信的方式，以及它们在一个较大的体系结构中的位置。界面设计模式描述常见的用户界面问题以及在影响因素(包括最终用户的具体特征)内的解决方案。WebApp模式解决在建立Web应用系统时遇到的问题集，通常与刚才提到的很多其他模式类别结合使用。框架提供了模式所在的基础结构，而习惯用语描述了特定程序设计语言对于全部或部分指定算法和数据结构的实现细节。标准化的表格或模板用来描述模式。一个模式语言包含了模式集，每个模式都使用一个标准化的模板来描述，这些模式之间相互关联来显示它们如何协作来解决应用领域中的问题。

基于模式的设计与体系结构、构件级以及用户界面设计方法联合使用。这种设计方法从检查需求模型开始，去分解问题、定义环境并描述影响因素。接下来，查找问题领域的模式语言，以确定已被分解问题的模式是否存在。一旦找到了适合的模式，就可用作设计指南。

习题与思考题

12.1 讨论设计模式的三个“部分”，对于每个部分从软件以外的其他领域提供具体的例子。

12.2 非生产模式和生产模式之间的区别是什么？

12.3 体系结构模式与构件模式有什么区别？

12.4 什么是框架？框架与模式的区别在哪里？什么是习惯用语？习惯用语与模式的区别是什么？

12.5 用12.1.3节描述的设计模式模板,为老师提出的一个模式开发完整的模式描述。

12.6 为你熟悉的一项运动开发一个简要模式语言。你可以从处理环境、系统影响因素、教练和球队必须解决的主要问题开始。你只需要指定模式名称，并用一句话来描述每个模式。

12.7 寻找5个模式资源库，并且简要描述包含在其中的模式类型。

12.8 Christopher Alexander说：“简单地通过把可执行部分加在一起，不能实现好的设计”，你认为他说的意思是什么？

12.9 用12.2.3节所提到的基于模式的设计任务为11.3.2节中的“室内设计系统”开发一个简要设计。

12.10 为在习题12.9中用到的模式建立一个基于模式的组织表。

12.11 用12.1.3节介绍的设计模式模板为12.3节提到的厨房模式开发一个完整的模式描述。

12.12 四人帮[Gam95]提出了很多构件模式，适用于面向对象系统。从中选择一个（可以在网上找到）进行讨论。

12.13 对于用户界面模式，寻找3个模式资源库。从每个资源库中选择一个模式进行简要描述。

12.14 对于WebApp模式，查找3个模式资源库。从每个资源库中选择一个模式进行简要描述。

推荐读物与阅读信息

在过去的10年中，在基于模式的设计方面，已经编写了很多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。最近出版的书籍包括：Laser （《Design Patterns》，Wordware Publishing, Inc., 2007）、Holzner （《Design Patterns for Dummies》，For Dummies, 2006）、Freeman和她的同事们 （《Head First Design Patterns》，O'Reilly Media, Inc., 2005），以及Shalloway和Trott （《Design Patterns Explained》，2nd.ed., Addison-Wesley, 2004）。IEEE Software的专刊（July/August, 2007）讨论了广泛的软件模式主题。Kent Beck （《Implementation Patterns》，Addison-Wesley, 2008）阐述了在构造活动中遇到的编码及实现方面的模式。

其他书籍关注特定应用开发和语言环境下的设计模式。这方面的书籍包括：Bowers （《Pro CSS and HTML Design Patterns》，Apress, 2007）、Tropashko和Burleson （《SQL Design Patterns: Expert Guide to

SQL Programming》, Rampant Techpress, 2007)、Mahemoff (《Ajax Design Patterns》, O'Reilly Media, Inc., 2006)、Metsker和Wake (《Design Patterns in Java》, Addison-Wesley, 2006)、Nilsson (《Applying Domain-Driven Design and Patterns: With Examples in C# and .NET》, Addison-Wesley, 2006)、Sweat (《PHP Architect's Guide to PHP Design Patterns》, Marco Tabini & Associates, Inc., 2005)、Metsker (《Design Patterns C#》, Addison-Wesley, 2004)、Grand和Merrill (《Visual Basic .NET Design Patterns》, Wiley, 2003)、Crawford和Kaplan (《J2EE Design Patterns》, O'Reilly Media, Inc., 2003)、Juric 等人 (《J2EE Design Patterns Applied》, Wrox Press, 2002), 以及Marinescu和Roman (《EJB Design Patterns》, Wiley, 2002)。

还有其他书籍涉及了特殊的应用领域。这些书包括：Kuchana (《Software Architecture Design Patterns in Java》, Auerbach, 2004)、Joshi (《C++ Design Patterns and Derivatives Pricing》, Cambridge University Press, 2004)、Douglass (《Real-Time Design Patterns》, Addison-Wesley, 2002), 以及Schmidt和Rising (《Design Patterns in Communication Software》, Cambridge University Press, 2001)。

对于每个希望全面了解设计模式的软件设计师，架构师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) 很值得一读。

在Internet上可以获得大量基于模式的设计方面的信息资源。与基于模式的设计有关的WWW引用的最新列表可在SEPA站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

WebApp设计

要点浏览

概念：WebApp设计包括的技术性活动和非技术性活动有：建立WebApp的外观和印象，创建用户界面的美学布局，定义总体结构，开发体系结构中的内容和功能，以及设计WebApp的导航等。

人员：Web工程师、美术设计师、内容开发者及其他共同利益者都参加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。内容设计利用内容模型（是在分析期间开发的）作为建立内容对象设计的基础。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：要对设计模型的每个元素进行评估，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
内容
体系结构
对象
设计
美学
体系结构
构件级
内容
目标
图形
导航
金字塔
质量
MVC体系结构
OOHDM
WebApp体系结构

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。设计从艺术想象力发展而来，而艺术想象力本身又是随着WebApp结构的出现而发展的。

即使在今天，很多Web开发者使用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。"Web工程[⊖]采纳了这一思想，一种更严格的WebApp设计方法使开发人员能够实现这种设想。

⊖ Web工程 [Pre08]是本书介绍的软件工程方法的改编版。为了建立达到工业质量要求的基于Web的系统及应用，提出了一种敏捷但严格的框架。

13.1 WebApp 设计质量

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示，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 Weins chenk

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？

实际上，在第8章讨论的软件质量的所有技术特征，以及第14章介绍的通用质量属性都适用于WebApp。然而，其中一些最相关的通用特性——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。

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[⊖]。图13-1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。

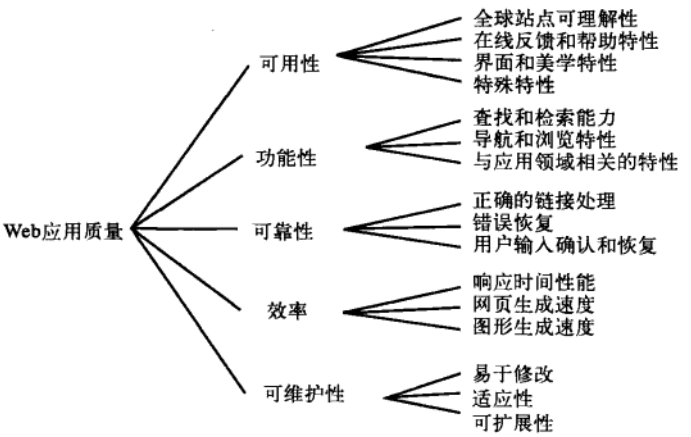


图13-1 质量需求树

资料来源：[Ols99]

Offutt[Off02]又补充了下面的属性，对图13-1描述的5个质量属性进行了扩展。

什么是WebApp的主要质量属性？

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）阻挡恶意攻击的能力。

⊖ 这些质量属性与在第8章和第14章介绍的属性十分相似。这意味着，质量特征对所有的软件都是通用的。

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[Vac06]、[Kiz05]或[Kal03]。

可用性：如果不可用的话，即使是最好的WebApp也不能满足用户的要求。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天都是可用的（常简写为24/7/365），对可用性的任何减少都是不可接受的[⊖]。但是，“正常运行时间”并不是可用性的唯一指标。Offutt[Off02]认为：“使用仅限于在一种浏览器或平台上可用的特性”会使WebApp在那些具有不同浏览器或平台的配置中变得不可用，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。

投放市场时间：虽然投放市场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设计的质量检查单

下面的清单是根据Webreference.com上所提供的信息修改而来的，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？
- 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。即使是很好地命中目标的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：

- 能否很容易地判断内容的范围和深度，确保满足用户的要求？
- 是否容易识别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）？

当评估内容质量时，我们应该考虑什么？

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？

⊖ 当然，这种期望是不现实的。大多数的WebApp必须安排“停机时间”以进行修复和升级。

- 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？是否容易选取？

本节提及的清单只是设计WebApp时应该考虑的问题中的一小部分。

13.2 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，但是最好还是尽量做到适度和简单。

内容应该充实而简洁，使用适合于信息的提交方式（例如，文本，图形，视频，音频）传送信息。美学应当令人愉快，但不能过度（例如，颜色过多使用户分心，不能加强互动）。体系结构应该用可能的最简单的方式实现WebApp的目标。导航应该简单明了，导航机制对终端用户应该直观明显。功能应该容易使用和理解。

一致性：这一设计目标几乎适用于设计模型的每个元素。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。如Kaiser[Kai02]所提到的：“记住，对于访问者来说，一个网站是一个物理位置。如果网站的网页在设计上不一致，会使人感到困惑。”

“仅仅只是因为你能做，并不意味着你应该这么做。”——Jean Kaiser

“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，还有些人可能将Web设计看成是一种技术……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

符合性：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，嬉皮士组织（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的相符性。

健壮性：在已经建立的符合性的基础上，WebApp通常会给用户以不言而喻的“承诺”。用户期待与他们的要求相关的健壮的内容和功能，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。例如，选作导航机制的图标或图像，必须能够直观地识别。在很多图形图像中寻找工作着的链接是很令人厌烦的。

在每个网页的可预测的位置，设置指向主要WebApp内容和功能的链接也是很重要的。如果需要页面滚动条（这是很常见的），在网页上方和下方的链接可以使用户的导航任务更容易。

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、正文布局是否匀称、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），并且必须设计为互相兼容。

13.3 WebApp设计金字塔

"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

什么是WebApp设计？这个简单的问题比想象的更难于回答。在David Lowe和我所编写的关于Web工程的书 [Pre08] 中，是这样讨论这个问题的：

> 创建有效的设计通常需要各种技术。有时，对小项目，开发者可能需要成为多面手。对比较大的项目，借鉴专家的专门知识是明智的而且（或）可行的：例如，Web工程师、图形设计师、内容开发者、程序员、数据库专家、信息架构师、网络工程师、安全专家及测试人员。借鉴这些技术，可以评估创建的模型质量，在产生内容和代码、进行测试以及大量的终端用户参与之前，对模型进行改进。如果分析是建立WebApp质量，那么设计就是真正地嵌入质量。

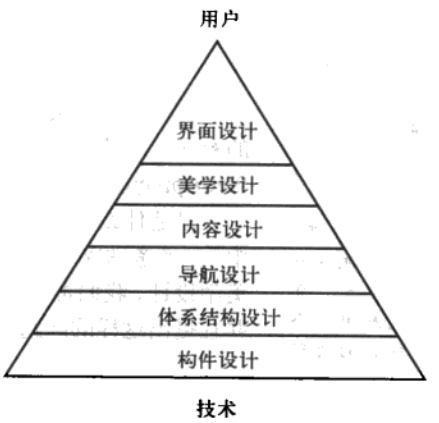


图13-2 WebApp设计金字塔

要根据WebApp性质的不同，而适当混合各种技术。图13-2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动作，这些设计动作在后面的章节中介绍。

13.4 WebApp界面设计

当用户与基于计算机的系统交互时，要应用一套基本原则和重要的设计准则。这些在第11章讨论[⊖]。虽然WebApp表现出一些特殊的用户界面设计上的挑战，但是基本原则和方针仍然是适用的。

WebApp界面设计的挑战之一是，用户的进入点不明确。即，用户可能在主页进入WebApp，或者可能链接到WebApp体系结构的一些较低层。在某些情况下，可用更改用户到主页的路线方法来设计WebApp，但是如果不想这样做，那么WebApp设计必须提供界面导航特征以及全部内容对象，这样就可以不考虑用户是如何进入系统的。

WebApp界面的目标是：(1) 建立一致性的窗口进入界面提供的内容和功能；(2) 通过一系列的与WebApp的交互指导用户；(3) 组织用户可用的导航选项和内容。为了获得一致的界面，首先要用美学设计（13.5节）去建立一致的外观。这包括很多特点，但必须强调布局和导航机制的形式。为了指导用户交互，可以适当借鉴隐喻（metaphor）[⊜]，使用户能直观地理解界面。为了实现导航选项，可以从下面的很多交互机制中选择一个：

⊖ 11.5 节专门介绍了WebApp界面设计。如果还没有读过，现在可以去读。

⊜ 在本书中，隐喻是一种表示方式（来自用户工作经验），可以在界面环境内建模。一个简单的例子是控制.mpg文件音量的滚动条开关。

对于WebApp设计者来说，可以得到哪些交互机制？

- 导航菜单——关键字菜单（纵向或横向组织）列出了关键内容和（或）功能。实现这些菜单，使得当选择了当主菜单选项时，用户可以从显示的次标题层次中进行选择。
- 图形图标——按钮、开关以及类似的图形图像，可以让用户选择某些属性或指明一种决定。
- 图形图像——用户可选择的一些图形表示，并执行链接到一个内容主题或WebApp功能。

值得注意的是，在内容层次的每个级别上都应提供一个或多个控制机制。

13.5 美学设计

不是每位Web工程师（或软件工程师）都有艺术（美学）才能。如果你也没有这种才能，那么就雇用一个有经验的图形设计师进行美学设计工作。

美学设计，又称美术设计，是一种艺术工作。它是对Web设计在技术方面的补充。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5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"？

13.5.1 布局问题

"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

每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：

不要担心留下空白。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网页的80%应该是内容，剩余的资源为导航和其他要素。

比起水平滚动条，人们更容易容忍垂直滚动条。要避免宽大的网页格式。

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[⊖]。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。

在页面内按导航、内容和功能安排布局。在几乎所有的事情中，人们都会寻找事实上已有的模式。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。

在设计布局时，考虑分辨率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，而不是在布局中规定固定的尺寸[Nie00]。

⊖ 基于文化和语言的不同，也有例外，但这条规则对多数用户来说是有效的。

13.5.2 美术设计问题

美术设计需要考虑到WebApp外观的每个方面。美术设计过程从布局（13.5.1节）开始，并在设计过程中考虑全局颜色配置、字体、字号、风格、补充媒体（例如，音频、视频、动画）的使用，以及应用系统的所有其他美学元素。

关于WebApp的美术设计问题的全面讨论超出了本书的范围。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，[Roc06]以及[Gor02]）中获得设计忠告和指导原则。

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.creativepro.com/designresource/home/787.html——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.creativehotlist.com/index.html——介绍了由广告代理商、美术设计公司以及其他通信专家开发的设计良好的网站。

www.btdnyc.com——Beth Toudreau领导的设计公司。

13.6 内容设计

"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

内容设计关注两个不同的设计任务，每个人都可以使用不同的技巧描述每个任务。首先，为内容对象开发一种设计表示，并且开发一种机制建立内容对象之间的关系。其次，也要生成特定的内容对象内的信息。后者可由广告撰稿人员、美术设计人员以及产生WebApp内容的其他人员完成。

13.6.1 内容对象

WebApp需求模型所定义的内容对象和代表内容的设计对象之间的关系类似于前面几章描述的分析类和设计构件之间的关系。在WebApp设计环境中，内容对象与传统软件中的数据对象关系更加紧密。内容对象具有的属性，包括特定的内容信息（通常在WebApp需求建模期间定义）的属性和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。分析类的属性description在这里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13-3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格式的图示）包含属性horizontal dimension、vertical dimension和border style。

UML的关联和聚合符号[⊖]可以用来表示内容对象之间的关系。例如，图13-3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

⊖ UML聚合和关联的表示在附录1中讨论。

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。

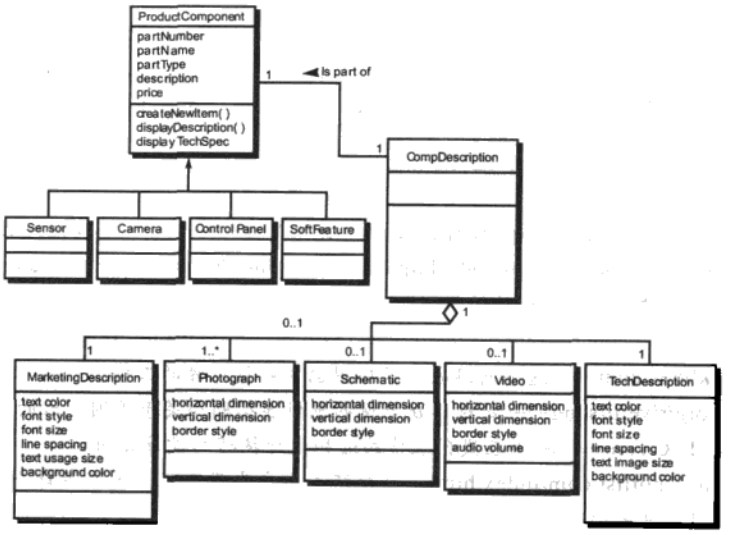


图13-3 内容对象的设计表示

13.6.2 内容设计问题

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。内容编辑是相关领域专家的工作，他们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（13.5节）为内容设计合适的外观。

设计内容对象时，将内容对象“分块”[Pow02]，然后形成WebApp页面。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以及用户能够忍受的滚动次数有关。

13.7 体系结构设计

“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

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（Content architecture）[⊖]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。

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。

⊖ 术语信息体系结构也用于表示结构，使得能够更好地组织、标识、导航及搜索内容对象。

13.7.1 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。虽然有时可以创建定制的体系结构，但通常会在下面4种不同的内容结构中进行选择[Pow00]：

? 一般会遇到哪些类型的内容体系结构？

线性结构。当内部交互的可预测顺序（带有一些变化或转向）很常见时，会遇到线性结构（图13-4）。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示、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13-4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向来获取补充的内容（图13-4右边所示的结构）。

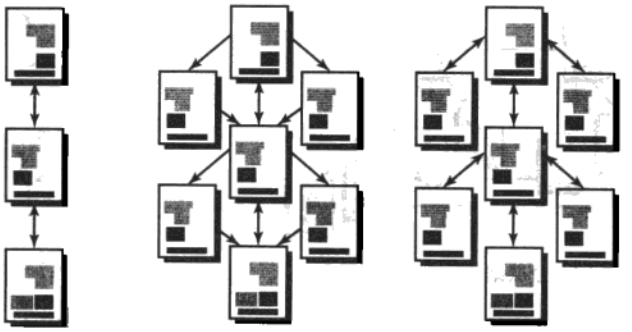


图13-4 线性结构

网格结构。网格结构（图13-5）是另一种体系结构，当WebApp的内容按类别组织成两维（或更多维）时，可以采用这种结构。例如，考虑这样的情况：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]。

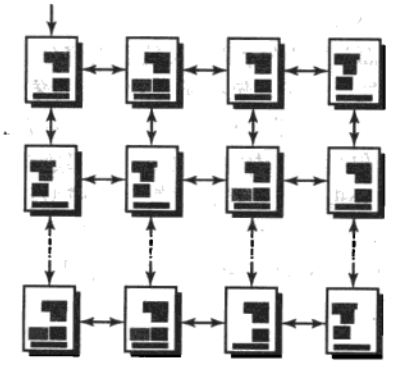


图13-5 网格结构

分层结构。分层结构（图13-6）毫无疑问是最常见的WebApp体系结构。与第9章讨论的分区的软件层次有所不同，在第9章，只有在层次的垂直分支上支持控制流程，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。

网络结构或“纯Web”结构。这种结构（图13-7）在很多方面类似于为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）到系统中的几乎所有其他构件。这种方法使导航相当灵活，但同时可能使用户感到困惑。

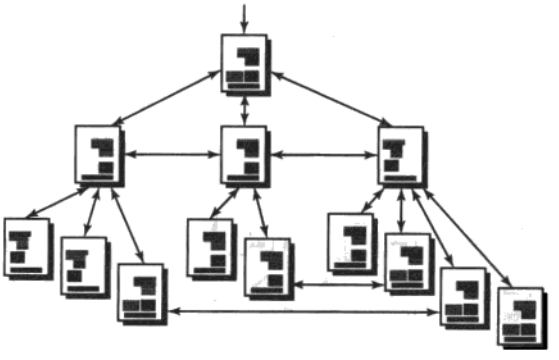


图13-6 分层结构

图13-7 网络结构

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。

13.7.2 WebApp体系结构

WebApp体系结构描述了使基于Web的系统或应用系统达到其业务目标的基础结构。Jacyntho和他的同事[Jac02b]对这一结构的基本特性做了如下描述：

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，应用系统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系统行为相分离，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。

模型－视图－控制器（Model-View-Controller，MVC）结构[Kra88]⊖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处理逻辑，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13-8所示。

KEY POINT
MVC体系结构将用户界面与WebApp功能和信息内容分离。

⊖ 值得注意的是，MVC实际上是为Smalltalk环境开发的体系结构设计模式（见www.cetus-links.org/oo_smalltalk.html），并且可用于任何交互式应用系统。

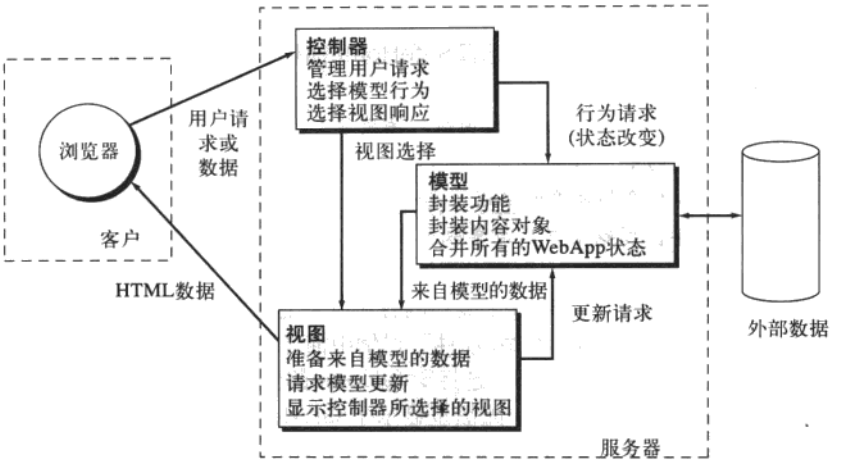


图13-8 MVC结构（改自[Jac02]）

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。感兴趣的读者请查阅[Fow03]，其中深入讨论了开发环境及其在Web应用系统体系结构设计中的作用。

13.8 导航设计

"Gretel，等到月亮升起来的时候，我们就能看到我撒落的面包屑。这些面包屑会指引我们回家的路。" ——Hansel和Gretel

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）。

13.8.1 导航语义

像很多WebApp设计活动一样，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5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"[Cac02]。

KEY POINT
NSU描述了每个用例的导航需求。本质上，NSU表现了角色是如何在内容对象之间或WebApp功能之间移动的。

NSU由一组导航元素组成，其中，导航元素也称作导航方式（Way of Navigating，WoN）[Gna99]。对于特定类型的用户来说，为了达到导航目的，WoN展示了最佳的导航路径。每个WoN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，在某些情况下，导航链接可能就是另一个NSU。因此，可以将WebApp的总体导航结构组织为

NSU的层次结构。

为了举例说明NSU的开发，考虑“选择SafeHome部件”用例：

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。

“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

图13-9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[⊖]，并且采用用户触发行为进行标注，这些行为使链接发生。

WebApp设计者可以为每个与用户角色相关的用例都创建一个NSU。例如，SafeHomeAssured.com的新客户可能有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。

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。

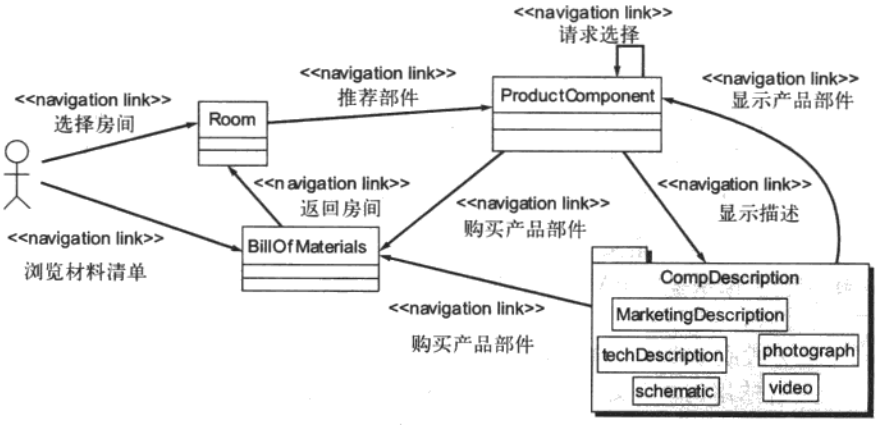


图13-9 创建NSU

13.8.2 导航语法

在进行设计时，下一项任务就是定义导航机制。在找到实现每个NSU的途径时，有多种可能的选择：

⊖ 有时将这些导航链接称为导航语义链接（Navigation Semantic Link，NSL）[Cac02]。

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。

网站地图应该从任何一页都能访问。地图本身应被组织成WebApp信息显而易见的结构。

- 单独的导航链接——包括基于文本的链接、图标、按钮、开关及图形隐喻。设计者必须选择适合内容的导航链接，并与能产生高质量界面设计的启发式方法一致。
- 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；或（2）列出WebApp中几乎所有的重要内容对象。如果选择了第2个选项，这种导航列可以“展开”，将内容对象描述为层次的一部分（即，在初始列中选择一项，就会引发展开，从而列出相关内容对象的第二层）。
- 标签——作为一种隐喻，其实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。
- 网站地图——向WebApp中的所有内容对象和功能提供完整的导航内容表格。

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。

13.9 构件级设计

现代WebApp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能力；（3）提供了高级的数据库查询和访问；（4）建立了与外部协作系统的数据接口。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。

第10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、设计模式、框架和软件可能会有些不同，但是，总的设计方法是一样的。

13.10 面向对象的超媒体设计方法

在过去的十年中，提出了很多Web应用系统的设计方法。至今，还没有哪一种方法能够取得主导地位[⊖]。在这一节，简要概述其中讨论最广泛的WebApp设计方法之一——OOHDM（面向对象的超媒体设计方法）。

Daniel Schwabe和他的同事[Sch95, Sch98b]最早提出了面向对象的超媒体设计方法（Object-Oriented Hypermedia Design Method, OOHDM），这种方法由4个不同的设计活动组成：概念设计、导航设计、抽象界面设计及实现。这些设计活动的概况如图13-10所示，并在下面的小节中简要讨论这些设计活动。

⊖ 实际上，设计WebApp时，很少有Web开发者使用某种特定的方法。随着时间的推移，这种特殊的设计方法很可能有所改变。

	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、导航环境、导航变换	抽象的界面对象、响应外部事件、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之间的映射	建立导航对象和可感知对象之间的映射	目标环境提供的资源
设计问题	应用领域的建模语义	考虑用户描述和任务，侧重于可感知的方面	可感知对象建模，实现选择的隐喻，描述导航对象的界面	正确性；应用程序的性能；完整性

图13-10 OOHDM方法小结（根据[Sch95]改写）

13.10.1 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UML[⊖]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。

作为OOHDM概念设计的一个简单例子，我们再次考虑SafeHomeAssured.com电子商务应用系统。SafeHomeAssured.com的部分“概念模式”如图13-11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。

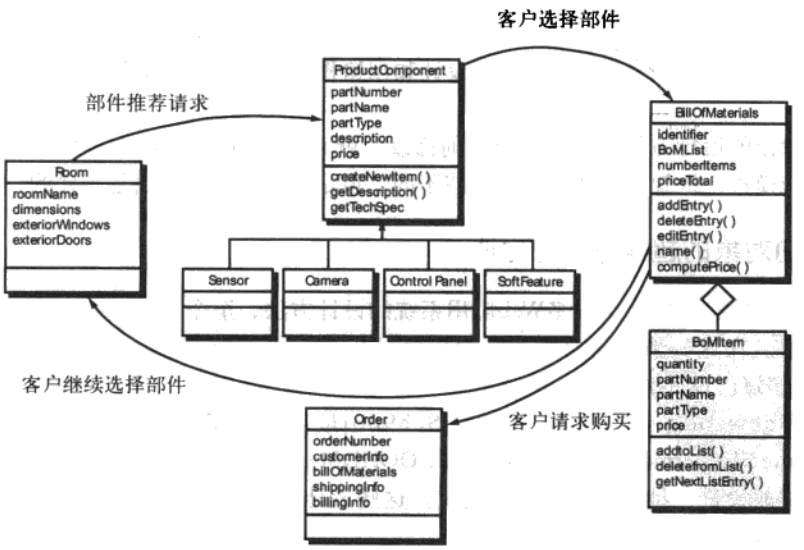


图13-11 SafeHomeAssured.com的部分概念模式

⊖ OOHDM没有规定专门的符号，但是，当应用这种方法时，通常使用UML。

13.10.2 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[Sch98b]。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”[Sch98b]。上下文包括局部导航结构的详细描述、内容对象的访问限制以及完成访问内容对象所需的方法（操作）。创建上下文模板（类似于第6章讨论的CRC卡片），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13.8.1节称之为WoN）。

13.10.3 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（abstract data view，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”[Sch98b]，这种布局描述界面隐喻，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，哪个界面会发生转换”[Sch01a]。

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。

13.11 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。为了达到这些质量属性，好的WebApp设计应该展现出简单性、一致性、符合性、健壮性、导航性和视觉吸引力。为了实现这些特征，WebApp设计活动集中在6个不同的设计元素。

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。当设计布局和界面控制机制时，有一套界面设计原则和界面设计工作流来引导设计者。

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。一套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的系统或应用达到其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。通过描述一组导航语义单元来定义导航语义。每个单元由导航路径、导航链接和节点组成。导航语法描

述用于实现导航的机制，导航也是语义描述的一部分。

构件设计开发了实现WebApp功能构件所需要的详细处理逻辑。第10章所描述的设计技术可应用于WebApp的构件工程。

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、抽象界面设计和实现。

习题与思考题

13.1 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，在这种情况中艺术理念就是要遵循的准则？

13.2 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3个质量特性，给出论据说明为什么每个质量特性都应该在Web工程设计工作中受到重视。

13.3 在13.1节描述的“WebApp设计质量检查单”中，至少再增加5个问题。

13.4 你为一个远程学习公司FutureLearning Corporation设计WebApp。你想实现一个基于Internet的能够将课程内容发布给学生的“学习引擎”。此学习引擎为发布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。

13.5 你曾访问过的最美观的网站是什么？它为什么美观？

13.6 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制订UML描述。

13.7 内容体系结构与WebApp体系结构之间的区别是什么？

13.8 再考虑13.4题描述的FutureLearning“学习引擎”，选择一种适合于WebApp的内容结构，讨论一下为什么你要这样选择？

13.9 在设计13.4题描述的FutureLearning“学习引擎”时，使用UML为所遇到的内容对象开发3个或4个设计表示。

13.10 对MVC体系结构做一些研究，针对13.4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。

13.11 导航语法与导航语义之间的区别是什么？

13.12 为SafeHomeAssured.com WebApp定义2个或3个NSU，对于每个NSU，给出详细的描述。

13.13 写一篇除了OOHDM的关于超媒体设计方法的简要论文。

推荐读物与阅读信息

Van Duyne和他的同事（《The Design of Sites》，2nd ed., Prentice Hall, 2007）编写了一本内容全面的书籍，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。

虽然“Web设计”方面的书籍已经有成百上千种，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。在这方面的很多书籍中，Sklar（《Principles of Web Design》，4th ed., Course Technology, 2008）、McIntire（《Visual Design for the Modern Web》，New Riders Press, 2007）、Niederst（《Web Design in a Nutshell》，3rd ed., O-Reilly, 2006）、Eccher（《Advanced Professional Web Design, Charles River Media, 2006》、Cederholm（《Bulletproof Web Design》，New Riders Press, 2005），

以及Shelly和他的同事（《Web Design》，2nd ed., Course Technology, 2005），Powell的[Pow02] 百科全书式的讨论以及Nielsen的[Nie00]关于设计的深入讨论都值得收入到任何图书馆中。

强调美学设计的书有：Beaird（《The Principles of Beautiful Web Design》，SitePoint, 2007）、Clarke和Holzschlag（《Transcending CSS: The Fine Art of Web Design》，New Riders Press，2006）以及Golbeck（《Art Theory for Web Design》，Addison Wesley，2005)等，这些书籍值得在这个主题上经验不多的从业者阅读。

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》，2nd ed., 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。

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HTML、CSS、J2EE、Java、.NET、XML、Perl、Ruby on Rails、Ajax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，有关WebApp设计的最新WWW参考文献列表可以在SEPA站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

质 量 管 理

在本书的这一部分将学习用来管理和控制软件质量的原理、概念和技术。在后面几章中会涉及下列问题：

- 高质量软件的一般特性是什么？
- 如何评审质量，如何有效地进行质量评审？
- 什么是软件质量保证？
- 软件测试需要应用什么策略？
- 使用什么方法才能设计出有效的测试用例？
- 有没有确保软件正确性的可行方法？
- 如何管理和控制软件开发过程中经常发生的变更？
- 使用什么标准和尺度评估需求模型和设计模型、源代码以及测试用例的质量？

回答了这些问题，就为保证生产出高质量软件做好了准备。

第14章

Software Engineering：A Practitioner's Approach (Seventh Edition)

质 量 概 念

要点浏览

概念：究竟什么是软件质量？答案不是想象中的那样容易，当你看到它时你知道质量是什么，可是，它却不可捉摸，难以定义。但是对于计算机软件，质量是必须定义的，这正是本章要讲的。

人员：软件过程所涉及的每个人（软件工程师、经理和所有利益相关者）都对质量负有责任。

重要性：你可以把事情做好，或者你再做一遍。如果软件团队在所有软件工程活动中强调质量，就可以减少很多必需的返工，结果是降低了成本，更为重要的是缩短了上市时间。

步骤：为实现高质量软件，必须做4项活动：已验证的软件工程过程和实践、扎实的项目管理、全面的质量控制和具有质量保证基础设施。

工作产品：满足客户需要的、准确而可靠地运行的、为所有使用者提供价值的软件。

质量保证措施：通过检查所有质量控制活动的结果跟踪质量，通过在交付前检查错误，在发布到现场后检查缺陷来衡量质量。

关键概念
质量成本
足够好
责任
管理活动
质量
质量窘境
质量维度
质量因素
量化观点
风险
安全

随着软件日益融入人们生活的方方面面，提高软件质量的鼓声真的要敲响了。截至20世纪90年代，大公司认识到由于软件达不到承诺的特性和功能，每年浪费掉的钱财多达数十亿美元。更严重的是，政府和产业界开始日益担心严重的软件缺陷有可能使重要的基础设施陷入瘫痪，从而花费数百亿还要多。世纪之交，CIO杂志[Lev01]一篇标题为“停止每年浪费780亿美元”的文章，对“美国企业在不能如预期的那样工作的软件上花费数十亿美元”这样的事实表示遗憾。InformationWeek[Ric01]也表达了同样的担忧：

> 市场研究公司的Standish Group 谈到：尽管意愿良好，有缺陷的代码仍然是软件工业的幽灵，计算机系统的故障时间高达45%，美国公司去年花费了大约一千亿美元，用在了丧失的生产率和修补上”，这还不包括使客户生气而失去了这些客户的代价。因为IT企业依赖基础软件包来开发应用系统，所以不好的代码也能损毁定制的应用系统。

怎样差的软件才是劣质软件呢？定义是不同的，但是专家认为，只要每1000行代码有3或4处缺陷就能使程序执行得很差，大多数程序员每写10行代码大约注入一个错误，许多商业产品有数百万行代码，软件经销商至少将开发预算的一半花费在了测试时修改错误上。

2005年，ComputerWorld[Hil05]遗憾地表示“劣质软件惹烦了几乎所有使用计算机的组织机构，在计算机发生故障期间造成工作时间损失、数据丢失或毁坏、销售时机丧失、IT支持与维护费用高昂，以及客户满意度低等后果。一年后，InfoWorld[Fos06] 以“软件质量的可悲状况”作为主题书写了报告，报告称质量问题依然没有任何改观。

现如今，软件质量仍然是个问题，但是应该责备谁？客户责备开发人员，认为粗心的实践导致低质量的软件。开发人员责备客户（和其他项目利益相关者），认为不合理的交工日期以及连续不断的变更使开发人员在还没有完全验证时就交付了软件。谁说的对？都对，这正是问

题所在。本章把软件质量作为一个概念，考查为什么软件质量值得认真考虑，何时应用软件工程实践。

14.1 什么是质量

Robert Persig[Per74]在他的神秘的书《zen and the Art of MotorCycle Maintenance》中就我们称为“质量”的东西发表了看法：

质量…… 你知道它是什么，也不知道它是什么。这样说是自相矛盾的，但没有比这更好的说法了；也即：这样说更有质量。但是当要试图说明质量是什么时，除了我们知道的这些之外，总是所知了了！没有什么好谈论的，但是如果不能说明质量是什么，又怎能知道质量是什么，或者又怎能知道质量甚至是否存在呢？如果没有人知道质量是什么，那么所有实际用途根本不存在，但是所有实际用途确实是存在的。质量等级依据别的什么来划分？为什么人们将机会给某些东西而把其他东西扔到垃圾堆？显而易见某些东西好于其他东西……但是，好在什么地方呢？……所以就任凭金属的轮子一圈又一圈地转动，而找不到摩擦力在哪里。质量到底是什么？什么是质量？

的确——什么是质量？

可以用哪些不同的方式来看待质量？

在更为实用的层面上，哈佛商学院的David Garvin[Gar84]给出 建议：“质量是一个复杂多面的概念”，可以从5个不同的观点来描述。玄妙观点（如Persig）认为质量是马上就能识别的东西，却不能清楚地定义。用户观点是从最终用户的具体目标来说的。如果产品达到这些目标，就显示出质量。制造商观点是从产品的原始规格说明的角度来定义质量，如果产品符合规格说明，就显示出质量。产品观点认为质量是产品的固有属性（比如，功能和特性）。最后，基于价值的观点根据客户愿意为产品支付多少钱来评测质量。实际上，质量涵盖所有这些观点，或者更多。

“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

设计质量是指设计师赋予产品的特性。原料等级、公差和性能等规格说明决定了设计质量。如果产品是按照规格说明书制造的，那么使用了较高等级的原料，规定了更严格的公差和更高级别的性能，产品的设计质量就能提高。

在软件开发中，设计质量包括设计满足需求模型规定的功能和特性的程度。符合质量关注的是实现遵从设计的程度以及所得到的系统满足需求和性能目标的程度。

但是，设计质量和符合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：

$$\text{用户满意度} = \text{合格的产品} + \text{好的质量} + \text{按预算和进度安排交付}$$

总之，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 [DeM98]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。

14.2 软件质量

高质量的软件是一个重要目标，即使最疲倦的软件开发人员也会同意这一点。但是，如何定义软件质量呢？在最一般的意义上，软件质量可以这样定义[⊖]：在一定程度上应用有效的软

⊖ 该定义节选自[Bes04]，取代该书前几版中出现的更面向生产的观点。

件过程，创造有用的产品，为生产者和使用者提供明显的价值。

毫无疑问，上述定义可以修改、扩展以及无休止的讨论。针对本书的目的，该定义强调了以下3个重要的方面：

1. 有效的软件过程为生产高质量的软件产品奠定了基础。过程的管理方面进行检验和平衡，以避免项目混乱——低质量的关键因素。软件工程实践允许开发人员分析问题、设计可靠的解决方案——二者皆为生产高质量软件的关键所在。最后，诸如变更管理和技术评审等普适性活动与其他部分的软件工程活动密切相关。

2. 有用的产品是指交付最终用户要求的内容、功能和特征，但最重要的是，以可靠、无误的方式交付这些东西。有用的产品总是满足利益相关者明确提出的那些需求，另外，也要满足一些高质量软件应有的隐性需求（例如，易用性）。

3. 通过为软件产品的生产者和使用者增值，高质量软件为软件组织和最终用户群体带来了收益。软件组织获益是因为高质量的软件在维护、改错及客户支持方面的工作量都降低了，从而使软件工程师减少了返工，将更多的时间花费在开发新的应用系统上，软件组织因此而获得增值。用户群体也得到增值，因为应用系统提供有用的能力，在某种程度上加快了一些业务流程。最后的结果是，（1）软件产品的收入增加；（2）当应用系统支持业务流程时，收益更好；（3）提高了信息可获得性，这对商业来讲是至关重要的。

14.2.1 Garvin的质量维度

David Garvin[Gar87]建议采取多维的观点考虑质量，包括从符合性评估到抽象的（美学）观点。尽管Gravin的8个质量维度没有专门为软件制定，但考虑软件质量时依然可以使用：

性能质量。软件是否交付了所有的内容、功能和特性，这些内容、功能和特性在某种程度上是需求模型所规定的一部分，可以为最终用户提供价值。

特性质量。软件是否首次提供了使最终用户惊喜的特性？

可靠性。软件是否无误地提供了所有的特性和能力，当需要（使用该软件）时，它是否是可用的，是否无错地提供了功能？

符合性。软件是否遵从本地的和外部的与应用领域相关的软件标准，是否遵循了事实存在的设计惯例和编码惯例？例如，对于菜单选择和数据输入等用户界面的设计是否符合已接受的设计规则？

耐久性。是否能够对软件进行维护（变更）或改正（改错），而不会粗心大意地产生料想不到的副作用？随着时间的推移，变更会使错误率或可靠性变得更糟吗？

适用性。软件能在可接受的短时期内完成维护（变更）和改正（改错）吗？技术支持人员能得到所需的所有信息以进行变更和修正缺陷吗？Douglas Adams[Ada93]挖苦地评论道："可能发生故障的东西与不可能发生故障的东西之间的差别是，当不可能发生故障的东西发生了故障，通常结果是不可能发现和补救的"。

审美。毫无疑问，关于什么是美的，我们每个人有着不同的、非常主观的看法。可是，我们中的大多数都同意美的东西具有某种优雅、特有的流畅和醒目的外在，这些都是很难量化的，但显然是不可缺少的。美的软件具有这些特征。

感知。在某些情况下，一些偏见将影响人们对质量的感知。例如，有人给你介绍了一款软件产品，该软件产品是由过去曾经生产过低质产品的厂家生产的，你的自我保护意识将会增加，你对于当前软件产品质量的感知力可能受到负面影响。类似地，如果厂家有极好的声誉，你将能感觉到好的质量，甚至在质量实际并不存在的时候。

Garvin的质量维度提供了对软件质量的"软"评判。这些维度的多数（不是所有）只能主

观地考虑。正因如此，也需要一套“硬”的质量因素，这些因素可以宽泛地分成两组：（1）可以直接测量的因素（例如，测试时发现的缺陷数）；（2）只能间接测量的因素（例如，可用性或可维护性）。在任何情况，必须进行测量，应把软件和一些基准数据进行比较来确定质量。

14.2.2 McCall的质量因素

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3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14-1所示。

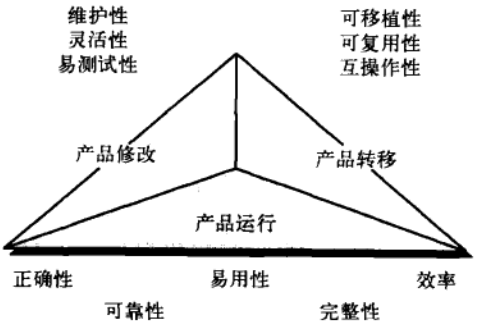


图14-1 McCall的软件质量因素

针对图14-1中所提到的因素，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25章）。

效率。程序完成其功能所需的计算资源和代码的数量。

满足进度计划的喜悦已经被遗忘很久之后，低质量所带来的痛苦依然挥之不去。——Karl Weigers

完整性。对未授权的人员访问软件或数据的可控程度。

易用性。对程序进行学习、操作、准备输入和解释输出所需要的工作量。

维护性。查出和修复程序中的一个错误所需要的工作量。[这是一个非常受限的定义。]

灵活性。修改一个运行的程序所需的工作量。

易测试性。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环境所需要的工作量。

可复用性。程序（或程序的一部分）可以在另一个应用系统中使用的程度。这与程序所执行功能的包装和范围有关。

互操作性。将一个系统连接到另一系统所需要的工作量。

要想求得这些质量因素的直接测度[⊖]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间接地测量。不过，使用这些因素评估应用系统的质量可以真实地反映软件的质量。

14.2.3 ISO 9126质量因素

尽管为这里提到的质量因素制定量化测量很诱人，你也可以创建一个简单的属性清单来提供显示质量因素的可靠指标。

ISO 9126国际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6个关键的质量属性：

功能性。软件满足已确定要求的程度，由以下子属性表征：适合性、准确性、互操作性、依从性和安全保密性。

可靠性。软件可用的时间长度，由以下子属性表征：成熟性、容错性和易恢复性。

易用性。软件容易使用的程度，由以下子属性表征：易理解性、易学习性

⊖ 直接测度意味着存在一个简单可计算的值，该值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。

和易操作性。

效率。软件优化使用系统资源的程度，由以下子属性表征：时间特性和资源利用特性。

维护性。软件易于修复的程度，由以下子属性表征：易分析性、易改变性、稳定性和易测试性。

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，由以下子属性表征：适应性、易安装性、符合性和易替换性。

与前面几小节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，它们确实为间接测量提供了有价值的基础，并为评估系统质量提供了一个优秀的检查单。

14.2.4 定向质量因素

"当从业人员想将事情做正确或做得更好时，任何活动都将变得富有创造性。"——John Updike

14.2.1节、14.2.2节提出的质量维度和因素关注于软件整体，可以用其作为应用系统质量的一般性指标。软件团队可以提出一套质量特征和相关的问题以调查满足每个质量因素的程度[⊖]。例如：McCall把易用性看做重要的质量因素。当要求评审用户界面和评估易用性时，该如何进行？可能要从McCall提出的子属性——易理解性、易学习性和易操作性开始，但是在实用的意义上，这些子属性表示什么意思呢？

为了进行评价，需要说明白界面的具体的、可测量的（或至少是可识别的）属性。例如[Bro03]：

直觉。界面遵照预期使用模式的程度，使得即使是新手，不经过专门培训也能使用。

- 界面布局易于理解吗？
- 界面操作容易找到和上手吗？
- 界面使用了可识别的隐喻吗？
- 输入安排得节约敲击键盘和点击鼠标吗？
- 界面符合3个重要原则吗？（第11章）
- 美学的运用有助于理解和使用吗？

效率。定位或初步了解操作和信息的程度。

- 界面的布局和风格可以使用户有效地找到操作和信息吗？
- 一连串的操作（或数据输入）可以用简单动作达到吗？
- 输出的数据和显示的内容是否能立即被理解？
- 分层操作是否组织得能使用户完成某项工作所需导航的深度最小？

健壮性。软件处理有错的输入数据或不恰当的用户交互的程度。

- 如果输入了规定边界上的数据或恰好在规定边界外的数据，软件能识别出错误吗？更为重要的是，软件还能继续运行而不出错或性能不下降吗？
- 界面能识别出常见的可识别错误或操作错误，并能清晰地指导用户回到正确的轨道上来吗？
- 当发现了错误的情况（与软件功能有关），界面是否提供有用的诊断和指导？

丰富性。界面提供丰富特征集的程度。

- 界面是否能按照用户的特定要求进行客户化？
- 界面是否提供宏操作以使用户将单个的行为或命令当做一连串的常用操作？

当界面设计展开后，软件团队将评审设计原型，询问关注的问题。如果对这些问题的大多数回答是"肯定的"，用户界面就显示出了高质量。应该为每个待评估的质量因素开发出类似

⊖ 这些特征和问题将作为软件评审的一部分来对待（第15章）。

的一组问题。

14.2.5 过渡到量化观点

前面几节讨论了一组测量软件质量的定性因素。软件界也力图开发软件质量的精确的测度，但有时又会为活动的主观性而受挫。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］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并列对比。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，最终的结果必须由专家给出。

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测量方法……既然没有绝对知识这种事物，就不要期望精确测量软件质量，因为每一种测量都是部分地不完美的。Jacob Bronkowski是这样描述知识（认识）的自相矛盾现象："年复一年，我们设计更精准的仪器用以更精细地观察自然界，可是当我们留心这些观察数据，却很不愉快地发现这些数据依然是模糊的，我们感到它们还和过去一样不确定。"

在第23章，将提出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。

14.3 软件质量困境

在网上发布的一篇访谈[Ven03]中，Bertrand Meyer 这样论述我所称谓的质量困境：

当面临质量困境时（每个人都会在某个时期面对它），尽力取得平衡——用足够的工作量去开发质量可接受的产品、而不会将项目葬送掉。

如果生产了一个存在严重质量问题的软件系统，你将受到损失，因为没有人想去购买。另一方面，如果你花费无限的时间、极大的工作量和高额的资金来开发一个绝对完美的软件，那么完成该软件将花费很长的时间，生产成本是极其高昂的，以至于破产。要么错过了市场机会，要么几乎耗尽所有的资源。所以企业界的人们努力达到奇妙的中间状态：一方面，产品要足够好，不会立即被抛弃（比如在评估期）；另一方面，又不是那么完美，不需花费太长时间和太多成本。

软件工程师应该努力生产高质量的系统，在试图这样做时采用好的做法就更好了。但是，Meyer所讨论的情况是现实的，甚至对于最好的软件工程组织，这种情况也表明了一种两难的困境。

14.3.1 "足够好"的软件

坦率地说，如果我们准备接受Meyer的观点，那么生产"足够好"软件是可接受的吗？对于这个问题的答案只能是"肯定的"，因为大的软件公司每天都这么做。这些大公司生产带有已知缺陷的软件，并发布给大量的最终用户。他们认识到，1.0版提供的一些功能和特性达不到最高质量，计划在2.0版改进。他们这样做，知道有些客户会抱怨，但他们认识到上市时间胜过更好的质量，只要交付的产品"足够好"。

到底什么是"足够好"？足够好的软件提供用户期望的高质量功能和特性，但同时也提供了其他更多的包含已知错误的难解的或特殊的功能和特性。软件供应商希望广大的最终用户忽视错误，因为他们对其他的应用功能是如此满意。

这种想法可能引起许多读者的共鸣。如果你是其中之一，我只能请你考虑一些论据来反对"足够好"。

诚然，“足够好”可能在某些应用领域和几个主要的软件公司起作用。毕竟，如果一家公司有庞大的营销预算，并能够说服足够多的人购买1.0版本，那么该公司已经成功地锁定了这些用户。正如我前面所指出的，可以认为，公司将在以后的版本提高产品质量。通过提供足够好的1.0版，公司垄断了市场。

如果你工作的是一个小公司，就要警惕这一观念，当你交付一个足够好（有缺陷的）产品时，你是冒着永久损害公司声誉的风险。你可能再也没有机会提供2.0版本了，因为不良言论可能会导致你的销售暴跌和贵公司关门。

如果你工作在某个应用领域（如实时嵌入式软件）或者你建造的是与硬件集成的应用软件（如汽车软件、电信软件），可能是疏忽，你交付了带有已知错误的软件，可能使公司处于代价昂贵的诉讼之中。在某些情况下，甚至可以是刑事犯罪，没有人想要足够好的飞机航空电子系统软件！

因此，如果你认为“足够好”是一个可以解决您的软件质量问题的捷径，要谨慎行事。“足够好”可以起作用，但只是对于少数几个公司，而且只是在有限的几个应用领域[⊖]。

14.3.2 质量成本

关于软件成本的争论是这样的：我们知道，质量是重要的，但是花费时间和金钱——花费太多的时间和金钱才能达到我们实际需要的软件质量水平。 表面上看，这种说法似乎是合理的（见本节前面Meyer的评论）。毫无疑问，质量是有成本的，但缺乏质量也有成本——不仅是对必须忍受缺陷软件的最终用户，而且是对已经开发且必须维护该软件的软件组织。真正的问题是：我们应该担心哪些成本？要回答这个问题，你必须既要了解实现质量的成本，又要了解低质量软件的成本。

质量成本包括追求质量过程中或在履行质量有关的活动中引起的费用以及质量不佳引起的下游费用等所有费用。为了解这些费用，一个组织必须收集度量数据，为目前的质量成本提供一个基准，找到降低这些成本的机会，并提供一个规范化的比对依据。质量成本可分为预防成本、评估成本和失效成本。

预防成本包括：（1）计划和协调所有质量控制和质量保证所需管理活动的成本；（2）为开发完整的需求模型和设计模型所增加的技术活动的成本；（3）测试计划的成本；（4）与这些活动有关的所有培训成本。

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。

评估成本包括为深入了解产品“第一次通过”每个过程的条件而进行的活动。评估成本的例子包括：

- 对软件工程工作产品进行技术审查（第15章）的成本。
- 数据收集和度量估算（第23章）的成本。
- 测试和调试（第18章到第21章）的成本。

失效成本是那些如果在将产品发给客户之前或之后没有错误就不会存在的费用。失效成本可分为内部失效成本和外部失效成本。内部失效成本发生在当你在发货之前发现错误时，内部失效成本包括：

- 为纠正错误进行返工（修复） 所需的成本。
- 返工时无意中产生副作用，必须对副作用加以缓解而发生的成本。
- 组织为评估失效的模型而收集质量数据，由此发生的相关成本。

外部失效成本是在产品已经发运给客户之后发现了缺陷时的相关成本。外部成本的例子是

⊖ 关于“足够好”软件的优缺点的有价值的探讨请参阅[Bre02]。

"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

解决投诉，产品退货和更换，帮助作业支持，与保修工作相关的人力成本。不良的声誉和由此产生的业务损失是另一个外部失效成本，是很难量化但非常现实的。生产了低质量的软件产品时，不好的事情就要发生。

在对拒绝考虑外部失效成本的软件开发者的控告书中，Cem Kaner [Kan95]是这样说的：

> 许多外部失效成本，如商誉的损失难以量化，许多企业因此在计算其成本效益的权衡时忽视了这些成本。其他外部失效成本在不增加客户的满意度时可降低（例如，通过提供更便宜、质量较低、售后支持，或向客户收取支持费用）。质量工程师靠忽视使用不良产品的客户成本，鼓励作出与质量相关的决策，这种决策只会欺骗客户，而不会使客户高兴。

正如预料的那样，当我们从预防到检查内部失效和外部失效成本，找到并修复错误或缺陷的相关成本急剧增加。根据Boehm和Basili收集的数据[Boe01b]和Cigital Inc[Cig07]的阐述，图14-2说明了这一现象。

在代码生成期纠正缺陷的行业平均成本是每个错误大约977美元，如果发现在系统测试期，纠正同样的错误，行业平均成本是每个错误7136美元。[Cig07]认为，一个大型应用程序在编码期会引入200个错误。

根据行业平均水平的数据，在编码阶段发现和纠正缺陷的成本是每个缺陷977美元，因此，在本阶段纠正这200 个“关键”缺陷的总费用大约是195 400美元（200 × 977美元）。

行业平均水平数据显示，在系统测试阶段发现和纠正缺陷的代价是每个缺陷7136美元。在这种情况下，假定该系统测试阶段发现大约50个关键缺陷（或者Cigital在编码阶段只发现了这些缺陷的25%），发现和解决这些缺陷的代价将是大约356 800美元（50 × 7136美元）。这将导致150处关键错误会未被发现和矫正。在维护阶段发现和解决这些遗留的150个缺陷的代价将是2 115 300美元（150 × 14 102美元）。因此，在编码阶段之后发现和解决这200个缺陷的代价将是2 472 100美元（2 115 300美元+356 800美元）。

即使您的软件组织所花费的是行业平均水平的一半（大多数企业尚不知道他们的成本！），与早期的质量控制和保证活动（进行需求分析和设计）有关的成本节约也是不得不做的。

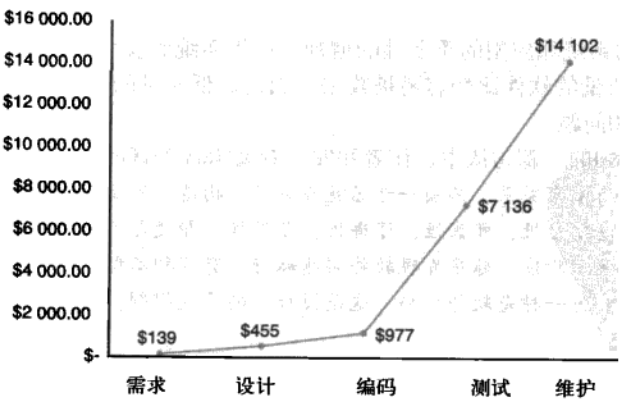


图14-2 改正错误和缺陷的相对成本（来源：摘自[Boe01b]）

14.3.3 风险

在这本书的第1章写到“人们拿自己的工作、自己的舒适、自己的安全、自己的娱乐、自

己的决定以及自己的生命在计算机软件上下赌注。最好这是正确的。”这意味着，低质量的软件为开发商和最终用户都增加了风险。在前面小节中讨论了这些风险（成本）中的一个。但设计和实现低劣的应用系统所带来的损失并不总是限于美元和时间。一个极端的例子[Gag04]可能有助于说明这一点。

整个2000年11月份在巴拿马的一家医院，28位病人在治疗多种癌症过程中接收了过量的伽玛射线照射。此后数月内，其中5例死于辐射病，15人发展成严重的并发症。是什么造成了这一悲剧？为了计算每位病人接受的辐射剂量，医院的技术人员对一家美国公司开发的软件包进行了修改。

为了提供额外的能力，这3个巴拿马医疗物理学家“调整”了软件，他们被指控犯有二级谋杀罪，这家美国公司正在两个国家面临着严重的诉讼。Gage和McCormick评论道：

这不是一个对医疗技术人员的警示故事，即使他们可以找到使自己不进监狱的一线机会，如果他们误解或误用了技术。尽管有很多例子说明这一点，这也不是一个关于人类如何被伤害的故事，或者更糟的是被设计不良或说明不清的软件伤害。这是对任何一个计算机程序创作者的教训：软件质量问题很重要，不管是嵌入在汽车引擎中的，工厂里的机械手臂中的，还是嵌入在医院的治疗设备中的，这些应用必须做到万无一失，低劣部署的代码可以杀人。

质量低劣导致风险，其中一些非常严重。

14.3.4 疏忽和责任

这种情况太常见了。政府或企业雇用一个较大的软件开发商或咨询公司来分析需求，然后设计和建造一个基于软件的“系统”，用以支撑某个重大的活动。系统可能支持主要的企业功能（例如，养老金管理），或某项政府职能（例如，卫生保健管理或国土安全）。

工作始于双方良好的意愿，但是到了系统交付时，情况已变得糟糕。系统延期，未能提供预期的特性和功能，易出错，不能得到客户的认可。接下来就要打官司。

在大多数情况下，顾客称开发商马虎大意（已带到了软件实践中），因此拒绝付款。而开发商则常常声称，顾客一再改变其要求，并在其他方面破坏了开发伙伴关系。无论是哪一种情况，交付系统的质量都会有问题。

14.3.5 质量和安全

随着基于Web的系统和应用的重要性的增加，应用系统的安全性已变得日益重要。简而言之，没有表现出高质量的软件比较容易被攻击。因此，低质量的软件会间接地增加安全风险，随之而来的是费用和问题。

在ComputerWorld的一篇访谈中，作者和安全专家Gary McGraw这样评论[Wil05]：

软件安全完全与质量有关系。必须一开始就在设计、构造、测试、编码阶段以及在整个软件生命周期（过程）中考虑安全性、可靠性、可得性、可信性。即使是已认识到软件安全问题的人们也主要关注生命周期的晚些阶段。越早发现软件问题越好。有两种类型的软件问题，一种是隐藏的错误，这是实现的问题。另一种是软件缺陷，这是设计中的构造问题。人们对错误关注太多，却对缺陷关注不够。

要构造安全的系统，就必须注重质量，并在设计时必须开始关注。本书第二部分讨论的概念和方法可以导出减少“缺陷”的软件架构。通过消除架构缺陷（从而提高软件质量），将会使攻击软件更加困难。

14.3.6 管理活动的影响

软件质量受管理决定的影响往往和受技术决定的影响是一样的。即使最好的软件工程实践

也能被糟糕的商业决策和有问题的项目管理活动破坏。

本书第四部分讨论软件过程环境下的项目管理。由于每个项目任务开始时，项目领导人都要作决策，这些决策可能对产品质量有重大影响。

估算决策。正如在第26章指出的，在确定交付日期和制定总预算之前，给软件团队提供项目估算数据是很少见的。反而是团队进行了“健康检查”，以确保交付日期和里程碑是合理的。在许多情况下，存在着巨大的上市时间压力，迫使软件团队接受不现实的交付日期。结果，抄了近路，可以获得更高质量软件的活动被忽略掉了，产品质量受到损害。如果交付日期是不合理的，坚持立场是重要的。这就是解释为什么你需要更多的时间，或者也可以建议在指定的时间交付一个（高质量的）功能子集。

进度安排决策。当建立了软件项目时间表时（第27章），按照依赖性安排任务的先后顺序。例如，由于A构件依赖于B、C和D构件中的处理，直到B、C和D构件完全测试后，才能安排A构件进行测试。项目计划将反映这一点。但是，如果时间很紧，为了做进一步的关键测试，A必须是可用的。在这种情况下，可能会决定在没有其附属构件（这些附属构件的运行要稍落后于时间表）的情况下测试A，这样对于交付前必须完成的其他测试，就可以使用A了。毕竟，期限正在逼近。因此，A可能有隐藏的缺陷，只有晚些时候被发现。质量会受到影响。

面向风险的决策。风险管理（第28章）是成功软件项目的关键特性之一。您真的需要知道哪儿可能会出问题，并建立一项如果确实出问题时的应急计划。太多的软件团队喜欢盲目乐观，在什么都不会出问题的假设下建立开发计划。更糟的是，他们没有办法处理真的出了差错的事情。结果，当风险变成现实，一片混乱，并且随着疯狂程度的上升，质量水平必然下降。

用Meskimen定律能最好地概括软件质量面临的困境——从来没有时间做好，但总是有时间再做一遍。我的建议是，花点时间把事情做好，这几乎从来都不是错误的决定。

14.4 实现软件质量

良好的软件质量不会自己出现，它是良好的项目管理和扎实的软件工程实践的结果。帮助软件团队实现高质量软件的四大管理和实践活动是：软件工程方法、项目管理技术、质量控制活动以及软件质量保证。

14.4.1 软件工程方法

? 怎样才能以积极的方式影响质量?

如果您希望建立高质量的软件，必须理解要解决的问题。还须能够创造一个符合问题的设计，该设计同时还要有一些性质，这些性质可以导致具有14.2节讨论过的质量维度和因素的软件。

本书第二部分提出了一大串概念和方法，可能导致对问题合理完整的理解和综合性设计，从而为构造活动建立了坚实的基础。如果您应用这些概念，并采取适当的分析和设计方法，那么创造高质量软件的可能性将大大提高。

14.4.2 项目管理技术

在14.3.6节已经讨论了不良管理决策对软件质量的影响。其中的含义是明确的：如果（1）项目经理使用估算以确认交付日期是可以达到的；（2）进度依赖关系是清楚的，团队能够抵抗走捷径的诱惑；（3）进行了风险规划，这样出了问题就不会引起混乱，软件质量将受到积极的影响。

此外，项目计划应该包括明确的质量管理和变更管理技术。导致良好项目管理实践的技术将在本书第四部分讨论。

14.4.3 质量控制

什么是软件质量控制?

质量控制包括一套软件工程活动，以帮助确保每个工作产品符合其质量目标。评审模型以确保它们是完整的和一致的。检查代码，以便在测试开始前发现和纠正错误。应用一系列的测试步骤以发现处理逻辑、数据处理以及接口通信中的错误。当这些工作成果中的任何一个不符合质量目标时，测量和反馈的结合使用使软件团队调整软件过程。本书第三部分的其余章节对质量控制活动进行了详细的讨论。

WebRef

软件质量保证(SQA)资源的有用链接可以在www.niwotridge.com/Resources/PMSWEResources/SoftwareQualityAssurance.htm找到。

14.4.4 质量保证

质量保证建立基础设施，以支持坚实的软件工程方法，合理的项目管理和质量控制活动——如果你打算建立高品质软件，所有这些都是关键活动。此外，质量保证还包含一组审核和报告功能，用以评估质量控制活动的有效性和完整性。质量保证的目标是为管理人员和技术人员提供所需的数据，以了解产品的质量状况，从而理解和确信实现产品质量的活动在起作用。当然，如果质量保证中提供的数据发现了问题，处理问题和使用必要的资源来解决质量问题是管理人员的职责。软件质量保证将在第16章详细论述。

14.5 小结

随着软件融入我们日常生活的每一方面，对于基于软件系统质量的关注逐渐多起来。但是很难给出软件质量的一个全面的描述。在这一章，质量被定义为一个有效的软件过程，用来在一定程度上创造有用的产品，为那些生产者和使用者提供适度的价值。

这些年来，提出了各种各样的软件质量度量和因素，都试图定义一组属性，如果实现，将达到高的软件质量。McCall的质量因素和ISO 9126的质量因素建立了很多特性（如可靠性、易用性、维护性、功能性和可移植性）作为质量存在的指标。

每个软件组织都面临软件质量困境。从本质上说，每个人都希望建立高质量的系统，但生产“完美”软件所需的时间和工作量在市场主导的世界里根本无法达到。这个问题就转化成为，我们是否应该生产“足够好”的软件呢？虽然许多公司是这样做的，但是这样做有很大的负面影响，必须加以考虑。

不管选择什么方法，质量是有成本的，质量成本可以从预防、评估和失效方面来说。预防成本包括所有被设计成将预防缺陷放在首位的软件工程活动。评估成本是与评估软件工作产品以确定其质量的活动有关的成本。失效成本包括失效的内部代价和低劣质量造成的外部影响。

软件质量是通过软件工程方法、扎实的管理措施和全面质量控制的应用达到的——所有这些都是靠软件质量保证基础设施支持的。后续章节将详细讨论质量控制和质量保证。

习题与思考题

14.1 描述在申报之前你将怎样评估一所大学的质量。哪些因素重要？哪些因素关键？

14.2 Garvin[Gar84]描述了质量的5种不同的观点，用一个或多个您熟悉的知名电子产品为每个观点举一个例子。

14.3 使用14.2节提出的软件质量定义，不使用有效的过程来创建能提供明显价值的有用产品，你认为可能吗？解释你的答案。

14.4 为14.2.1节介绍的Garvin的每个质量维度添加两个额外的问题。

14.5 McCall质量因素是在20世纪70年代提出的，自提出以来，几乎计算的每个方面都发生了翻天覆地的变化，然而，McCall质量因素继续适用于现代软件。基于这一事实你是否能得出一些结论？

14.6 使用14.2.3节所述ISO9126质量因素“维护性”的子属性，提出一组问题，探讨是否存在这些属性，仿效14.2.4节所示的例子。

14.7 用你自己的话描述软件质量困境。

14.8 什么是“足够好”的软件？说出具体公司的名字，以及您认为运用足够好思想开发的具体产品。

14.9 考虑质量成本4个方面的每个方面，你认为哪个方面是最昂贵的，为什么？

14.10 进行网络搜索，寻找直接由低劣的软件质量所致“风险”的其他3个例子。考虑从http://catless.ncl.ac.uk/risks开始搜索。

14.11 质量和安全是一回事吗？请加以解释。

14.12 解释为什么我们许多人仍在沿用Meskimen定律，造成软件业这样的原因是什么？

推荐读物与阅读信息

Henry和Hanlon（《Software Quality Assurance》，Prentice-Hall，2008）、Khan和他的同事（《Software Quality: Concepts and Practice》，Alpha Science International，Ltd.，2006）、O'Regan（《A Practical Approach to Software Quality》，Springer，2002）以及Daughtrey（《Fundamental Concepts for the Software Quality Engineer》，ASQ Quality Press，2001）的书讨论了基本的软件质量概念。

Duvall和他的同事（《Continuous Integration: Improving Software Quality and Reducing Risk》，Addison-Wesley，2007）、Tian（《Software Quality Engineering》，Wiley-IEEE Computer Society Press，2005）、Kandt（《Software Engineering Quality Practices》，Auerbach，2005）、Godbole（《Software Quality Assurance: Principles and Practice》，Alpha Science International，Ltd.，2004）以及Galin（《Software Quality Assurance: From Theory to Implementation》，Addison-Wesley，2003）介绍了软件质量保证的具体做法。Stamelos和Sfetsos（《Agile Software Development Quality Assurance》，IGI Global，2007)讨论了在敏捷过程环境中的质量保证问题。

可靠的设计可以得到软件的高质量。Jayasawal和Patton（《Design for Trustworthy Software》，Prentice-Hall，2006）以及Ploesch（《Contracts，Scenarios and Prototypes》，Springer，2004）讨论了开发“健壮”软件的工具和技术。

测量是软件质量工程的一个重要部分。Ejiogu（《Software Metrics: The Discipline of Software Quality》，BookSurge Publishing，2005）、Kan（《Metrics and Models in Software Quality Engineering》，Addison-Wesley，2002）以及Nance和Arthur（《Managing Software Quality》，Springer，2002）讨论了重要的质量相关的测度和模型。Evans（《Achieving Software Quality through Teamwork》，Artech House Publishers，2004）考虑了软件质量面向团队方面的问题。

软件质量方面的大量信息源可以在Internet上获得，关于软件质量方面最新的WWW参考文献列表可以在SEPA站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

评 审 技 术

要点浏览

概念：在开发软件工程工作产品时可能会犯错误，这并不是羞耻的事——只要在产品交付最终用户之前，努力、很努力地发现并纠正错误即可。技术评审是在软件过程早期查错最有效的机制。

人员：软件工程师和同事一起进行技术评审，也叫同行评审。

重要性：如果在软件过程的早期发现错误，修改的成本就较少。另外，随着软件过程的推进，错误会随之放大，因此，过程早期留下的没有处理的小错误，可能在项目后期放大成一组严重的错误。最后，通过减少项目后期所需的返工，评审节省了时间。

步骤：评审方法因所选评审的正式程度而异，尽管对每一种类型的评审，不是所有步骤都用到，一般分6个步骤：计划、准备、组织会议、记录错误、进行修改（评审之后做）、验证是否恰当地进行了修改。

工作产品：评审的输出是发现问题和（或）错误的清单。另外，还标示出工作产品的技术状态。

质量保证措施：首先，选择适合你的开发文化的评审类型，遵守导致成功评审的指导原则，如果你进行的评审导致了高质量的软件，说明你做对了。

关键概念
缺陷放大
缺陷
错误密度
错误
记录
评审
　度量
　报告
评审
　成本
　有效性
　非正式
　样本驱动
　技术的

评审就像软件过程工作流程中的过滤器一样，评审太少、工作流是"脏"的。评审太多，工作流很慢。使用度量以确定哪些评审起作用，并进行加强。从工作流中去除无效的评审以加速软件过程。

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"需求模型、设计模型、源代码和测试数据等软件工程工作产品。对于评审的需要，Freedman和Weinberg [Fre90]是这样论述的：

> 技术工作需要评审正像铅笔需要橡皮：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。因此，评审过程是对Robert Burns的祈祷的解答：
>
> 哦，聚集才能给我们力量
> 像别人看待我们那样，看待自己
>
> 评审（任何评审）是使用人群之间的差异达到以下目的：
>
> 1. 指出个人或团队的产品中需要改进的地方；
> 2. 确认产品中不期望或不需要改进的部分；
> 3. 与没有评审相比，得到质量更统一或至少更可预测的技术工作，以使技术工作更加可管理。

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谈就是一种评审方式。将软件架构正式介绍给客户、管理层和技术人员也是一种评审方式。但是，本书将重点讨论技术评审，即同行评审，通过举例说明非正式评审、走查和审查。从质量控制的角度出发，技术评审（Technical Review，TR）是最有效的过滤器。由软件工程师（以及其

他人）对软件工程师进行的技术评审是一种发现错误和提高软件质量的有效手段。

15.1 软件缺陷对成本的影响

在软件过程的环境中，术语缺陷和故障是同义词，两者都是指在软件发布给最终用户（或软件过程内其他框架活动）后发现的质量问题。在前面几章，我们使用了术语错误来描绘在软件发布给最终用户（或软件过程内其他框架活动）之前软件工程师（或其他人）发现的质量问题。

INFO

隐错、错误和缺陷

软件质量控制的目标是消除软件中存在的质量问题，从广义上讲，这也是一般质量管理的目标。有很多术语可用来描述这些质量问题，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[⊖]）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）。

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。

ADVICE

正式技术评审的主要目标是在错误传递到另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%。然而，已经证明，评审技术在发现设计缺陷方面高达75%有效[Jon86]。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。

15.2 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设计和编码活动中错误的产生和检测。该模型如图15-1所示，其中方框表示软件工程活动。在该活动中，可能由于疏忽产生错误，评审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

⊖ 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。

"正如医生所说，某些疾病在其初发阶段易于治愈，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

效性百分比是评审完善性的函数。

图15-2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为94个错误，12个潜伏的错误（缺陷）则随软件发布进入客户现场。图15-3的情况与图15-2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为24个错误，最后只剩3个潜伏的错误。回忆一下与发现和改正错误相关的相对成本，将图15-2和图15-3每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。

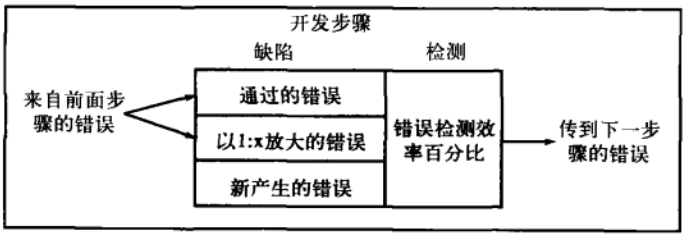


图15-1 缺陷放大模型

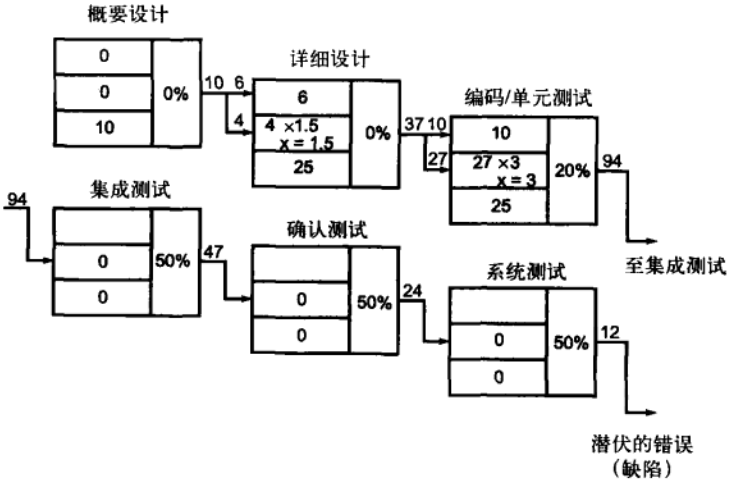


图15-2 缺陷放大——无评审

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。

⊖ 这些乘数与图14-2给出的数据稍有不同，图14-2更新。但是，它们都很好地例证了缺陷放大成本。

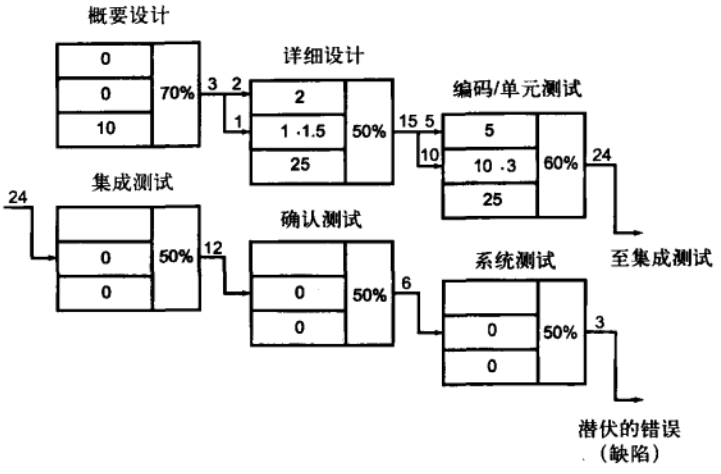


图15-3 缺陷放大——有评审

15.3 评审度量及其应用

技术评审是好的软件工程实践所需要的很多活动之一。每项活动都需要付出人力，由于可用的项目工作量是有限的，重要的是，软件工程组织要定义一套可以用来评估其工作效率的度量（第23章）来理解每项活动的有效性。

尽管可以为技术评审定义很多度量，一个相对较小的子集就可以提供有益的见解。可以为所进行的每项评审收集以下评审度量数据：

- 准备工作量E_p——在实际评审会议之前评审一个工作产品所需的工作量（单位：人时）。
- 评估工作量E_a——实际评审工作中所花费的工作量(单位：人时)。
- 返工工作量E_r——修改评审期间发现的错误所用的工作量（单位：人时）。
- 工作产品规模WPS——被评审的工作产品规模的衡量（例如，UML模型的数量、文档的页数或代码行数）。
- 发现的次要错误Err_{minor}——发现的可以归为次要错误的数量（要求少于预定的改错工作量）。
- 发现的主要错误Err_{major}——发现的可以归为主要错误的数量（要求多于预定的改错工作量）。

通过将所评审的工作产品类型与所收集的度量数据相关联，这些度量数据可以进一步细化。

15.3.1 分析度量数据

在开始分析之前，必须进行一些简单的计算。总评审工作量E_{review}和发现的错误总数Err_{tot}定义为：

$$E_{\text{review}} = E_p + E_a + E_r$$

$$Err_{\text{tot}} = Err_{\text{minor}} + Err_{\text{major}}$$

错误密度表示评审的每单位工作产品发现的错误数。

$$\text{错误密度} = \frac{Err_{\text{tot}}}{WPS}$$

例如，如果评审需求模型以发现错误、不一致之处和遗漏，将有可能以一些不同的方式计算错误密度。需求模型的描述性材料共有32页，其中包含18个UML图。评审发现18处次要错误和4处主要错误。因此，$Err_{\text{tot}} = 22$。错误密度为每个UML图1.2个错误，或者说，每页需求

模型有0.68个错误。

如果是对一些不同类型的工作产品（例如，需求模型、设计模型、代码、测试用例）进行评审，可以通过所有评审所发现的错误总数来计算每次评审发现的错误百分比。此外，也可以计算每个工作产品的错误密度。

在为多个项目收集到许多评审数据后便可利用其错误密度的平均值估计一个新的项目中将发现的错误数。例如，如果需求模型的平均错误密度是每页0.6个错误，一个新的需求模型为32页，粗略估计，你的软件团队在评审该文档时将能发现大约19或20个错误。如果你只发现了6个错误，说明你在开发需求模型方面的工作非常出色或者说明评审工作做得不够彻底。

在进行了测试（第17~20章）之后，有可能收集到另外一些错误数据，包括在测试期间发现和纠正错误所需要的工作量，以及软件的错误密度。可以将测试期间发现和纠正错误的相关成本与评审期间的相比较，这部分内容在15.3.2节讨论。

15.3.2 评审的成本效益

若要实时地测量任何技术评审的成本效益都是困难的。只有在评审工作已经完成，已收集了评审数据，计算了平均数据，并测量了软件的下游质量（通过测试）之后，软件工程组织才能够对评审的有效性和成本效益进行评估。

返回15.3.1节所介绍的例子，需求模型的平均错误密度确定为每页0.6个错误。修改一个次要模型错误需要4人时（评审后立即修改），修改一个主要需求错误需要18人时。对所收集的评审数据进行分析，发现次要错误出现的频度比主要错误出现的频度高6倍。因此，可以估计，评审期间查找和纠正需求错误的平均工作量大约为6人时。

对于测试过程中发现的与需求有关的错误，查找和纠正的平均工作量为45人时（对于错误的相对严重性没有任何数据可用）。使用提到的平均数，我们得到：

$$\begin{aligned}\text{每个错误节省的工作量} &= E_{\text{testing}} - E_{\text{reviews}} \\ &= 45-6 = 39 \text{ 人时/错误}^{\ominus}\end{aligned}$$

由于在需求模型评审时发现了22个错误，节省了约858人时的测试工作量[⊖]。而这只是与需求有关的错误，与设计和代码相关的错误将加入到整体效益中。无疑，节省的工作量使交付周期缩短了，上市时间提前了。

Karl Wiegers [Wie02]在他的有关同行评审的书中讨论了从大公司得到的传闻数据，这些大公司已经使用审查（一种比较正式的技术评审）作为软件质量控制活动的一部分。Hewlett Packard称审查有10：1的投资回报率，并指出实际产品交付时间平均提前了1.8月。AT&T公司表示，审查使软件错误总成本降低到原来的十分之一，质量提高了一个数量级，而且生产率提高了14%。其他还有类似的效益报告。技术评审（为设计和其他技术活动）提供了明显的成本效益，并且确实节省了时间。

但对于许多软件开发人员，这种说法违反直觉。软件开发人员认为“评审需要时间，我们没有多余的时间！”他们认为，在每个软件项目中时间是一种宝贵的商品，评审“每个工作产品细节”占用了太多时间。

本节先前提出的例子显示不是这样。更重要的是，对软件评审行业数据的收集已有20多年，可以用图形的方式定性地加以概括，如图15-4所示。

从图15-4可以看到，使用评审时花费的工作量在软件开发的早期确实是增加的，但是评审的早期投入产生了效益，因为测试和修改的工作量减少了。重要的是，有评审开发的发布日期比没有评审时要快。评审不费时间，而是节省时间。

⊖、⊖ 此处原书数据有误。——译者注

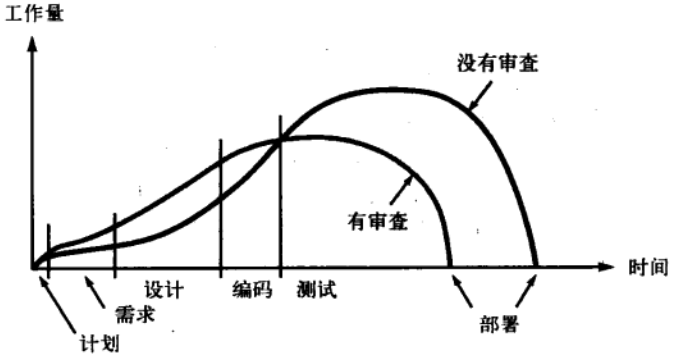


图15-4 有评审和没有评审时花费的工作量(来源：摘自[Fag86])

15.4 评审：正式程度

应该以某种正式程度应用技术评审，该正式程度应该适合所生产的产品、项目的时间线和做评审工作的人。图15-5描述了技术评审的参考模型[Lai02]，该模型中的4个特性有助于决定进行评审的形式。

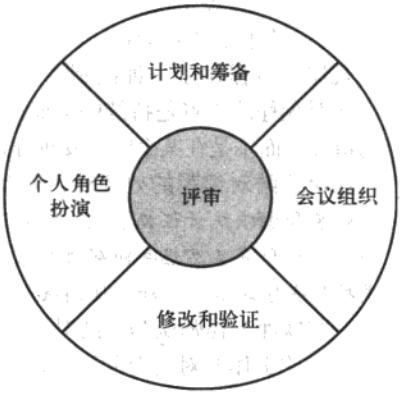


图15-5 技术评审参考模型

参考模型的每个特性有助于确定评审的正式程度。评审的正式度提高需当：（1）明确界定每位评审人员的不同职责（2）为评审进行充分的计划和准备，（3）为评审定义了清晰的结构（包括任务和内部工作产品），以及（4）评审人员对所做修改的后续跟踪。

为理解参考模型，让我们假设你已决定评审SafeHomeAssured.com的界面设计。你可以以各种不同的方式进行评审，从相对非正式的到极其严格的。如果你觉得非正式的方法更适合，你可以要求一些同事（同行）检查界面原型，努力发现潜在的问题。大家认为将不进行事先准备，但也可以以合理的结构化方式来评估原型——首先查看布局，接下来看是否符合美学，再下来是导航选项，等等。作为设计者，你可以记些笔记，但不用那么正式。

但是如果界面是整个项目成败的关键呢？如果人的生命依赖于完全符合人体工程学的界面呢？您可能会认为更严格的做法是必要的。于是成立一个评审小组，小组中的每个人承担特定的职责——领导小组工作、记录发现的问题、提供材料，等等。评审之前每个评审人员将有机会接触工作产品（该例中，即为界面原型），花时间查找错误、不一致和遗漏。按照评审之前制定的议程执行一组任务。正式记录评审的结果，评审小组根据评审结果对工作产品的状态作出决议。评审小组的成员可能还核实是否适当地进行了修改。

本书关注两类广为采用的技术评审：非正式评审和更为正式的技术评审。在每个类中都有一些不同的方法可以选择。这些都在接下来的几节中介绍。

15.5 非正式评审

非正式评审包括与同事就软件工程产品进行的简单桌面检查，以评审一个工作产品为目的

的临时会议（涉及两人以上），或结对编程评审（第3章）。

与同事进行的简单桌面检查或临时会议是一种评审。但是，因为没有事先规划或筹备工作，没有会议的议程或结构，没有对发现的错误进行后续的跟踪处理，这种评审的有效性大大低于更为正式的方法。但是，简单桌面检查可以也的确能发现错误，否则这些错误可能传播到后续的软件过程。

提高桌面检查评审效能的一种方法是为软件团队的每个主要的工作产品制定一组简单评审检查单。检查单中提出的问题都是常见问题，但有助于指导评审人员检查工作产品。例如，让我们重新就SafeHomeAssured.com的界面原型进行桌面检查。设计师和同事使用界面的检查清单来检查原型，而不是在设计者的工作站上简单地操作原型：

- 布局设计是否使用了标准惯例？是从左到右？还是从上到下？
- 显示是否需要滚动？
- 是否有效使用了不同的颜色和位置，字体和大小？
- 所有导航选项或表示的功能是否在同一抽象级别？
- 所有导航选择是否清楚地标明了？

等等。评审人员指出的任何错误和问题由设计人员记录下来以在稍后的时间进行解决。桌面检查可以以一个特别的方式安排，或者也可以授权作为良好的软件工程实践的一部分。一般来说，桌面检查评审材料的数量相对较少，总体上花费时间大致在一两个小时。

第3章描述了结对编程，方式如下："极限编程建议，两个人一起在一台计算机工作站编写一段代码。这提供了一种机制，可以实时解决问题（两人智慧胜过一人）和实时的质量保证。"

结对编程的特点是持续的桌面检查。结对编程鼓励在创建工作产品（设计或代码）时进行持续的审查，而不是在某个时候安排评审。好处是即时发现错误，结果是得到更好的工作产品质量。

在讨论结对编程的效能时，Williams和Kessler[Wil00]这样描述：

轶事和初步统计证据表明，结对编程是一种强大的技术，可以有效创造高品质的软件产品。两人一起工作和分享思想共同解决复杂的软件开发。他们不断进行检查彼此的产品，从而以最有效的形式尽早去除缺陷。此外，他们彼此一心一意专注于手头的任务。

一些软件工程师认为，结对编程固有的冗余是浪费资源。毕竟，为什么为两个人指派一个人可以完成的工作？对这个问题的回答可以在第15.3.2节找到。如果作为结对编程的结果生产出的工作产品的质量明显优于单个人的工作，在质量方面的节约，足以弥补结对编程带来的"冗余"。

INFO

评审检查清单

即使当评审已经很好地组织并严格地进行，为评审人员提供一张"参考清单"也是一个不错的主意。也就是说，需要有这样一个检查清单，它为每位评审人员对正在进行评审的具体的工作产品列出要问的问题。

*最全面的评审清单集之一是美国航天局在戈达德空间飞行中心（Goddard Space Flight Center）开发的，可以在*http://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php*得到。*

其他有用的技术评审清单有：

Process Impact(www.processimpact.com/pr_goodies.shtml)

Software Dioxide(www.softwaredioxide.com/Channels/ConView.asp?id=6309)

Macadamian(www.macadamian.com)

The Open Group Architecture Review Checklist (www.opengroup.org/architecture/togaf7-doc/arch/p4/comp/clists/syseng.htm)

DFAS（可下载）(www.dfas.mil/technology/pal/ssps/docstds/spm036.doc)

15.6 正式技术评审

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指定的标准；（4）获得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实际上是一类评审方式，包括走查（walkthrough）和审查（inspection）。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。如果你对审查以及走查的其他信息感兴趣，可参见[Rad02]、[Wie02]或[Fre90]。

15.6.1 评审会议

不论选用何种FTR方式，每个评审会议都应该遵守以下约束：

- 评审会（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。
- 评审会的时间应该少于2小时。

WebRef

《NASA SATC正式评审指南》可从satc.gsfc.nasa.gov/Documents/fi/gdb/fi.pdf下载。

考虑到这些限制，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。

KEY POINT

FTR关注的是某工作产品中比较小的一部分。

FTR关注的是某个工作产品（例如，一部分需求模型、一份详细的构件设计、一个构件的源代码）。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"。项目负责人与评审主席取得联系，由评审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位评审员以便事先做准备。每位评审员应该用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。

ADVICE

在某些情况下，让其他人而不是开发人员本人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实解释且更易于发现错误。

评审会议由评审会主席、所有评审员和开发人员参加。其中一位评审员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开发人员做简单的介绍开始。然后由开发人员"走查"该工作产品，并对材料做出解释，而评审员则根据预先的准备提出问题。当发现了明确的问题或错误时，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（发现了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。

15.6.2 评审报告和记录保存

在FTR期间，由一名评审员（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. 评审的产品是什么？
2. 谁参与了评审？
3. 发现的问题和结论是什么？

评审总结报告通常只是一页纸的形式（可能还有附件）。它是项目历史记录的一部分，有可能将其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）作为行动条目检查单以指导开发人员进行改正。通常将评审问题清单附在总结报告的后面。

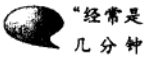
为了保证评审问题清单中的每一条都得到适当的改正，建立跟踪规程非常重要。只有做到这一点，才能保证提出的问题真正得到“解决”。方法之一就是将跟踪的责任指派给评审会主席。

15.6.3 评审指导原则

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，使开发人员能够自己发现错误。

进行正式技术评审之前必须制定评审的指导原则，并分发给所有评审员以得到大家的认可，然后才能依照它进行评审。不受控制的评审通常比没有评审还要糟糕。下面列出了最低限度的一组正式技术评审指导原则：

1. 评审工作产品，而不是评审开发人员。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。评审会主席应该引导评审会议，确保维护适当的气氛和态度，应立即终止已变得失控的评审。



“经常是几分钟的会议，却浪费了几个小时。”——作者不详

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。当评审员提出问题时，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。

“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。”——Palph Waldo Emerson

4. 要阐明问题，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，问题的解决应该由开发人员自己或是在个别人帮助下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，其他评审员可以推敲措辞，并确定问题的优先次序。或者，笔记可直接输入到笔记本电脑。

6. 限制参与者人数，并坚持事先做准备。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员要求书面意见（以表明评审员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清单。检查清单能够帮助评审会主席组织FTR会议，并帮助每位评审员将注意力集中到重要问题上。应该为分析、设计、代码、甚至测试等工作产品建立检查清单。

8. 为FTR分配资源和时间。为了进行有效的评审，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. 对所有评审员进行有意义的培训。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。Freedman 和 Weinberg [Fre90]估计每20人进行为期一个月的学习，将能够有效地参与评审。

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。

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对自己的特定环境最为适用。

15.6.4 样本驱动评审

在理想情况下，每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]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。

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，也不要省略评审，最好采用样本驱动评审。

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的若干分之一，记做$1/a_i$，记录在其中发现的缺陷数量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能代表整个工作产品，并且要足够大，对进行抽样的评审者来说有意义。当a_i增大时，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[⊖]。

SAFEHOME

质量问题

[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质量计划。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6章和第7章]的时候，Ed答应负责为每个需求建立测试规程。

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。

⊖ 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]。

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分析和设计模型中对SafeHome最关键的元素进行评审。

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[15.6.4节]，可以帮助我们明确应评审的元素。(Shakira描述了这种方法。)

Jamie：也许……但是，我也不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3章]。我们结对（两个人）找出每个模型中的元素，并同时进行非正式的评审。之后我们将能够明确“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，所有的事情都不只一组人员检查过了，但我们仍然可以维持原来的交付日期。

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。

15.7 小结

每次技术评审的目的都是找出错误和发现可能对将要部署的软件产生负面影响的问题。越早发现并纠正错误，错误传播到其他软件工程产品并扩大，导致需要更多的工作来纠正的可能就越小。

为了确定质量控制活动是否在起作用，应该收集一组度量。评审度量的重点放在进行评审需要的工作量、评审中发现的错误的类型和严重程度方面。一旦收集了度量数据，就可以用来评估你所进行的评审的效率。行业数据显示评审提供可观的投资回报。

评审正式程度的参考模型以评审人员承担的职责、计划和筹备、会议结构、纠正的办法和验证为特性，这些特性表明了进行评审的正式程度。非正式评审在性质上是临时的，但仍然可以有效地用于发现错误。正式评审更有条理，最有可能产生高质量软件。

非正式评审的特点是最低限度的规划和准备，并少有记录。桌面检查和结对编程属于非正式评审类。

正式技术评审是一个程式化的会议，已证明在发现错误方面是非常有效的。走查和审查为每个评审人员建立了确定的职责，鼓励计划和事先准备，需要应用规定的评审原则、授权记录和状态报告。当不可能对所有工作产品进行正式技术评审时，可以使用模型驱动评审。

习题与思考题

15.1 解释错误和缺陷的不同。

15.2 为什么我们不能只是等到测试的时候才去发现和纠正所有的软件错误？

15.3 假设需求模型引入了10个错误，每个错误按2：1的比例在设计阶段放大，设计阶段引入了另外的20个错误，并且这20个错误按1.5：1的比例在编码阶段放大，在编码阶段又引入了另外30个错误。进一步假设，所有单元测试会发现所有错误的30%，集成测试将找到剩余错误的30%，验证测试会发现剩余错误的50%。没有进行评审，有多少错误将被发布到现场？

15.4 重新考虑习题15.3所述情形，但现在假设进行了需求评审、设计评审和代码评审，并且这些步骤能有效地发现所有错误的60%。有多少错误将被发布到现场？

15.5 重新考虑习题15.3和习题15.4所述情形，对于每个发布到现场的错误，发现和改正的成本是4800美元，在评审时发现并改正每个错误花费240美元，通过进行评审节约多少钱？

15.6 用你自己的话描述图15-4的含义。

15.7 你认为参考模型的哪一个特点对评审的正式程度影响最大？解释原因。

15.8 桌面检查可能造成问题而没有带来好处，你能想到几个实例吗？

15.9 仅当每个参与者都事先进行了准备时，正式技术评审才是有效的。你如何识别没有准备好的评审参与者？如果你是评审主席，你该如何做？

15.10 考虑15.6.3节提出的所有评审准则，你认为哪一条是最重要的，为什么？

推荐读物与阅读信息

软件评审方面的书籍相对较少。最近出版的有指导价值的书有Wong（《Modern Software Review》，IRM Press, 2006）、Radice（《High Quality》，Low Cost Software Inspections, Paradoxicon Publishers, 2002）、Wiegers（《Peer Reveiws in Software: A Practical Guide》，Addison-Wesley, 2001）以及Gilb 和Graham （《Software Inspection》，Addison-Wesley, 1993）。Freedman 和Weinberg （《Handbook of Walkthroughs, Inspections and Technical Reviews》，Dorset House, 1990）仍是一部经典的教材，提供了有关软件评审这个重要课题的有用信息。

软件评审方面多种多样的信息资源可在互联网上得到。软件评审相关的WWW最新参考列表可以在SEPA网站www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第16章

Software Engineering：A Practitioner's Approach (Seventh Edition)

软件质量保证

要点浏览

概念：仅仅嘴上说“软件质量重要”是不够的。你必须：（1）明确给出你所说的“软件质量”的涵义；（2）提出一组活动，这些活动将有助于保证每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。

人员及责任：软件工程过程涉及的每个人都要对质量负责。

重要性：要么一次做好，要么重做。如果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播下去。

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”。在建模、编码阶段，主要的SQA工作产品是技术评审的输出（第15章）；在测试阶段（第17~20章），主要的SQA工作产品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23章），进而减少软件团队不得不付出的返工量。

关键概念
正式方法
目标
ISO 9001:2000
标准
六西格玛
软件可靠性
软件安全
SQA
要素
计划
统计
任务

本书所讨论的软件工程方法只有一个目标：在计划时间内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]在他有关质量的且颇具影响力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……
>
> 每个人都需要质量（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），每个人都认为只要顺其自然就可以（毕竟，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。）

确实，质量是一个具有挑战性的概念，已经在第14章进行了详细论述[⊖]。

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！软件质量保证（通常称为质量管理）是适用于整个软件过程的一种普适性活动（第2章）。

软件质量保证（SQA）包括：（1）SQA过程，（2）具体的质量保证和质量控制任务（包括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）对所有软件

⊖ 如果你还没有阅读第14章，那么现在应该阅读了。

工作产品及其变更的控制（第22章），（5）保证符合软件开发标准的规程（当适用时），（6）测量和报告机制。

本章侧重于管理问题和特定的过程活动，使软件组织确保“在恰当的时间以正确的方式做正确的事情”的具体活动。

16.1 背景问题

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。随着时间推移，大量生产技术逐渐普及，质量控制由生产者之外的其他人承担。

“你犯了太多错了。”——Yagi Berra

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，出现了更多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。

如今，每个公司都有保证其产品质量的机制。事实上，公司重视质量的明确声明已经成为过去几十年中市场营销的策略。

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[IEE93]。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”[Sch98c]。质量保证责任的范围最好可以用曾经流行的一个汽车业口号来概括：“质量是头等重要的工作”。对软件来说含义就是，各个参与者都对软件质量负有责任——包括软件工程师、项目管理者、客户、销售人员和SQA小组成员。

SQA小组充当客户在公司内部的代表。也就是说，SQA小组成员必须从客户的角度来审查软件。软件是否充分满足第14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。

16.2 软件质量保证的要素

WebRef

SQA的深度探讨（包含一系列定义）可在www.swqual.com/newsletter/vol2/no1/vol2no1.html得到。

软件质量保证涵盖了广泛的内容和活动，这些内容和活动侧重于软件质量管理，可以归纳如下[Hor03]：

标准：IEEE、ISO及其他标准化组织制定了一系列广泛的软件工程标准和相关文件。标准可能是软件工程组织自愿采用的，或者是客户或其他利益相关者责成采用的。软件质量保证的任务是要确保遵循所采用的标准，并保证所有的工作产品符合标准。

评审和审核：技术评审是由软件工程师执行的质量控制活动（第15章），目的是发现错误。审核是一种由SQA人员执行的评审，意图是确保软件工程工作遵循质量准则。例如，要对评审过程进行审核，确保以最有可能发现错误的方式进行评审。

测试：软件测试（第17~20章）是一种质量控制功能，它有一个基本目标——发现错误。SQA的任务是要确保测试计划适当和实施有效，以便最有可能实现软件测试的基本目标。

错误/缺陷的收集和分析：改进的唯一途径是衡量如何做。软件质量保证人员收集和分析错误和缺陷数据，以便更好地了解错误是如何引入的，以及什么样的软件工程活动最适合消除它们。

"卓越在于提高所供产品质量的能力是无限的"——Rick Petin

变更管理：变更是对所有软件项目最具破坏性的一个方面。如果没有适当的管理，变更可能会导致混乱，而混乱几乎总是导致低质量。软件质量保证确保进行足够的变更管理实践（第22章）。

教育：每个软件组织都想改善其软件工程实践。改善的关键因素是对软件工程师、项目经理和其他利益相关者的教育。软件质量保证组织牵头软件过程改进（第30章），并是教育计划的关键支持者和发起者。

供应商管理：可以从外部软件供应商获得三种类型的软件：(1) 简易包装软件包（shrink-wrapped package）（例如，微软Office）；(2) 定制外壳（tailored shell）[Hor03]——提供可以根据购买者需要进行定制的基本框架结构；(3) 合同软件（contracted software）——按客户公司提供的规格说明书定制设计和构造。软件质量保证组的任务是，通过建议供应商应遵循的具体的质量做法（在可能的情况下），并将质量要求作为与任何外部供应商签订合同的一部分，确保高质量的软件成果。

安全管理：随着网络犯罪和新的关于隐私的政府法规的增加，每个软件组织应制定政策，在各个层面保护数据，建立防火墙保护Web应用系统，并确保软件在内部没有被篡改。软件质量保证确保应用适当的过程和技术来实现软件安全。

安全：因为软件几乎总是人设计系统（例如，汽车应用或飞机应用）的关键组成部分，潜在缺陷的影响可能是灾难性的。软件质量保证可能负责评估软件失效的影响，并负责启动那些减少风险所必需的步骤。

风险管理：尽管分析和减轻风险（第28章）是软件工程师考虑的事情，但是软件质量保证组应确保风险管理活动适当进行，且已经建立风险相关的应急计划。

除了所有这些关注的问题和活动，软件质量保证还确保将质量作为主要关注对象的软件支持活动（如维修、求助热线、文件和手册）高质量地进行和开展。

INFO

质量管理资源

网上有很多质量管理资源，包括专业团体、标准组织和一般的信息资源。以下网址提供了不错的起点：

美国质量协会（ASQ）软件分会 www.asq.org/software
美国计算机协会 www.acm.org
软件数据和分析中心（DACS）www.dacs.dtic.mil/
国际标准化组织（ISO） www.iso.ch
ISO SPICE www.isospice.com
美国波多里奇国家质量奖 www.quality.nist.gov
软件工程研究所 www.sei.cmu.edu
软件测试和质量工程 www.stickyminds.com
六西格玛资源
www.isizsigma.com
www.asq.org/sixsigma/
TickIT国际：质量认证主题 www.tickit.org/international.htm
全面质量管理（TQM）
概要信息：www.gslis.utexas.edu/~rpollock/tqm.html
文章：www.work911.com/tqmarticles.htm
词汇表：www.quality.org/TQM-MSI/TQM-glossary.html

16.3 软件质量保证的任务、目标和度量

软件质量保证是由与两个不同人群相联系的多种任务组成——这两个人群分别是做技术工作的软件工程师和负有质量策划、监督、记录、分析和报告责任的软件质量保证组。

软件工程师通过采用可靠的技术方法和措施，进行技术评审，并进行周密计划的软件测试来获得质量（和执行质量控制活动）。

16.3.1 软件质量保证任务

软件质量保证组的行动纲领是帮助软件团队实现高品质的目标产品。软件工程研究所推荐一套质量保证活动，即从事质量保证策划、监督、记录、分析和报告。这些活动由独立的软件质量保证组执行（和完成）：

SQA小组的作用是什么？

编制项目质量保证计划。 该计划作为项目计划的一部分，并经所有利益相关者评审。软件工程组和软件质量保证组进行的质量保证活动都受该计划支配。该计划确定要进行的评估、要进行的审核和评审、适用于项目的标准、错误报告和跟踪的规程、软件质量保证组产出的工作产品以及将提供给软件团队的反馈意见。

参与项目的软件过程描述的编写。软件团队选择完成工作的过程。软件质量保证组审查该过程描述是否符合组织方针、内部软件标准、外部要求的标准（例如，ISO-9001），以及是否和软件项目计划的其他部分一致。

评审软件工程活动，以验证是否符合规定的软件过程。软件质量保证组识别、记录和跟踪偏离过程的活动，并验证是否已作出更正。

审核指定的软件工作产品以验证是否遵守作为软件过程一部分的那些规定。质量保证工作小组审查选定的产品；识别、记录并跟踪偏差；验证已经作出更正；并定期向项目经理报告其工作成果。

确保根据文档化的规程记录和处理软件工作和工作产品中的偏差。在项目计划、过程描述、适用的标准或软件工程工作产品中可能会遇到偏差。

记录各种不符合项并报告给高层管理人员。跟踪不符合项，直到解决。

除了这些活动，软件质量保证组还协调变更的控制和管理（第22章），并帮助收集和分析软件度量。

16.3.2 目标、属性和度量

执行上节所述软件质量保证活动，以实现一套务实的目标：

质量从来没有意外，它始终是崇高的意图，真诚的努力，聪明的管理和熟练执行的结果；它表示从多个选项中所做的明智选择。——William A.Faster

需求质量。需求模型的正确性、完整性和一致性将对所有后续工作产品的质量有很大的影响。软件质量保证必须确保软件团队严格评审需求模型，以达到高水平的质量。

设计质量。软件团队应该评估设计模型的每个元素，以确保设计模型显示出高质量，并且设计本身符合需求。SQA寻找能反映设计质量的属性。

代码质量。源代码和相关的工作产品（例如，其他说明资料）必须符合本地的编码标准，并显示出易于维护的特点。SQA应该找出那些能合理分析代码质量的属性。

质量控制有效性。软件团队应使用有限的资源，在某种程度上最有可能得到高品质的结果。SQA分析在评审和测试上的资源分配，评估是否以最有效的方式进行分配的。

对于所讨论的每个目标，图16-1（摘自[Hya96]）标出了现有的质量属性。可以使用度量数据来标明所示属性的相对强度。

目标	属性	度量
需求质量	歧义	引起歧义地方的修改数量（例如：许多、大量、与人友好）
	完备性	TBA, TBD的数量
	可理解性	节/小节的数量
	易变性	每项需求变更的数量
		变更所需要的时间（通过活动）
	可追溯性	不能追溯到设计/代码的需求数
	模型清晰性	UML模型数
		每个模型中描述文字的页数·
		UML错误数
设计质量	体系结构完整性	是否存在现成的体系结构模型
	构件完备性	追溯到结构模型的构件数
		过程设计的复杂性
	接口复杂性	挑选一个典型功能或内容的平均数
		布局合理性
	模式	使用的模式数量
代码质量	复杂性	环路复杂性
	可维护性	设计要素（第8章）
	可理解性	内部注释的百分比
		变量命名约定
	可重用性	可重用构件的百分比
	文档	可读性指数
质量控制效率	资源分配	每个活动花费的人员时间百分比
	完成率	实际完成时间与预算完成时间之比
	评审效率	参见评审度量（第14章）
	测试效率	发现的错误及关键性问题数
		改正一个错误所需的工作量
		错误的根源

图16-1 软件质量目标、属性和度量（来源：摘自[Hya96]）

16.4 软件质量保证的形式化方法

前面几节讨论了软件质量是每个人的工作，可以通过出色的软件工程实践以及通过应用技术评审、多层次的测试策略、更好地控制软件工作产品和所做的变更、应用可接受的软件工程标准而实现。此外，质量可定义成一组质量属性，并使用各种指标和度量进行测量（间接）。

WebRef

SQA和形式化质量方法方面的有用的信息可以在www.gslis.utexas.edu/~rpollock/tqm.html找到。

在过去的30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做是一个数学对象，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21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。

程序正确性证明并不是什么新的思路。Dijkstra [Dij76a]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的使用联系在一起（第10章）。

16.5 统计软件质量保证

统计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，统计质量保证包含以下步骤：

进行统计SQA需要哪些步骤？

1. 收集软件的错误和缺陷信息，并进行分类。
2. 追溯每个错误和缺陷形成的根本原因（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错误和缺陷的问题。

“统计质量保证”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，在这个过程中要进行修改，以改进那些引入错误的过程元素。

16.5.1 一个普通的例子

适当进行的统计分析，是对不确定事物的微妙解剖，是对推测的调查。——M. J. Moroney

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错误和缺陷信息，其中有些错误是在软件开发过程中发现的，另外一些（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问题，所有问题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（MIS）。

20%的代码中有80%的错误。找到错误并修改！——Lowell Arthur

为了使用统计质量保证方法，需要建一张如图16-2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需求收集技术（第5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。

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。

已经证明统计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。

	总 计		严 重		中 等		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%
MIS	56	6%	0	0%	15	4%	41	9%
总计	942	100%	128	100%	379	100%	435	100%

图16-2 统计SQA数据收集

统计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！

16.5.2 软件工程中的六西格玛

六西格玛是目前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“是严格且规范的方法学，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”[ISI08]。“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3个主要的核心步骤：

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、可交付的产品及项目目标。
- 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3个核心步骤和另外两个附加步骤有时叫做DMAIC（定义、测量、分析、改进和控制）方法。

如果某组织正在开发软件过程（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，以达到（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（定义、测量、分析、设计和验证）方法。

六西格玛的全面讨论不是本书重点，有兴趣的读者可请参考[ISI08]、[Pyz03]和[Sne03]。

16.6 软件可靠性

毫无疑问，计算机程序的可靠性是其整个质量的重要组成部分。如果某个程序经常反复不

可靠性不得以的代价是简明性。——C.A.R.Hoare

能执行，那么其他软件质量因素是不是可接受的就无所谓了。

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，计算机程序正常运行的概率"[Mus87]。举个例子来说，如果程序*X*在8小时处理时间内的可靠性估计为0.999，也就意味着，如果程序*X*执行1000次，每次运行8小时处理时间（执行时间），则1000次中正确运行（无失效）的次数可能是999次。

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）一词是什么含义？在有关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。失效可能仅仅是令人厌烦的，也可能是灾难性的。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。

16.6.1 可靠性和可用性的测量

KEY POINT 软件可靠性问题几乎总能追溯到设计或实现中的缺陷。

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。不幸的是，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（见第1章）在这里根本没有影响。

KEY POINT 重要的是要注意，平均无故障时间和相关测量是基于CPU时间，而不是挂钟时间。

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。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。

当我们考虑基于计算机的系统时，可靠性的简单测量是"平均失效间隔时间"（Mean-Time-Between-Failure，MTBF）：

$$MTBF = MTTF + MTTR$$

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"[⊖]。

许多研究人员认为MTBF是一个远比第23章讨论的其他与质量有关的软件度量更有用的测量指标。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。例如，考虑一个程序，已运行3000处理器小时没有发生故障。该程序中的许多缺陷在被发现之前可能有数万小时都不会被发现。隐藏如此深的错误的MTBF可能是30000甚至60000处理器小时。其他尚未发现的缺陷，可能有4000或5000小时的失效率。即使第一类错误（那些具有长时间的MTBF）都去除了，对软件可靠性的影响也是微不足道的。

ADVICE 可用性的某些方面（这里不讨论）与失效没有关系。例如，计划停机（为技术支持功能）也使软件不可用。

然而，MTBF可能会产生问题的原因有两个：（1）它突出了失效之间的时间跨度，但不会为我们提供一个凸显的失效率；（2）MTBF可能被误解为平均寿命，即使这不是它的含义。

⊖ 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。

可靠性另一个可选的衡量是失效率（Failures-In-Time，FIT）——一个部件每10亿机时发生多少次失效的统计测量。因此，1FIT相当于每10亿机时发生一次失效。

除可靠性测量之外，还应该进行可用性测量。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：

$$\text{可用性} = \frac{\text{MTTF}}{\text{MTTF+MTTR}} \times 100\%$$

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。

16.6.2 软件安全

民众的安全应是最崇高的法律。——Cicero

我无法想象是什么条件可能使该船沉没。现代造船术已经不存在这种可能了。——E.I.Smith（泰坦尼克船长）

WebRef

关于软件安全的有价值的论文集可以在www.safeware-eng.com找到。

软件安全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出这些灾难，就可以指定软件设计特性来消除或控制这些潜在的灾难。

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：

(1) 产生失去控制的加速，不能停止。

(2) 踩下刹车踏板后没有反应（关闭）。

(3) 开关打开后不能启动。

(4) 减速或加速缓慢。

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率[⊖]。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，此时当且仅当外部环境条件满足时，机械设备的不正确位置将引发灾难性的失效。失效树分析、实时逻辑及Petri 网模型等分析技术[Eri05]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。

一旦完成了灾难识别和分析，就可以进行软件中与安全相关的需求规格说明了。在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导致灾难发生的条件。也就是说，不能在真空中考虑失效，而是要在整个计算机系统及其所处环境的范围内加以考虑。

软件安全的全面讨论超出了本书的范围，对软件安全和相关系统问题有兴趣的读者可参考[Smi05]、[Dun02]和[Lev95]。

16.7 ISO 9000质量标准[⊜]

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

⊖ 这个方法与第28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。

⊜ 本节由Michael Stovsky编写，根据“Fundamentals of ISO 9000”改编，这是由R.S.Pressman & Associates, Inc. 为音像教程“Essential Software Engineering”开发的工作手册。翻印得到授权。

[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。这些体系覆盖了产品整个生命周期的多种活动，包括计划、控制、测量、测试和报告，并在产品的开发和制造过程中改进质量等级。ISO 9000标准以通用的术语描述了质量保证体系要素，能够适用于任何行业——不论提供的是何种产品或服务。

某个公司要登记成为ISO 9000质量保证体系中的一种模式，该公司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监督审核，以此保证该公司的质量体系持续符合标准。

WebRef

大量有关ISO 9000/9001资源的链接可在www.tantara.ab.ca/info.htm找到。

ISO 9001:2000中描述的质量要求涉及管理者责任、质量体系、合同评审、设计控制、文件和资料控制、产品标识与可追溯性、过程控制、检验和试验、纠正及预防措施、质量记录的控制、内部质量审核、培训、服务以及统计技术等主题[⊖]。软件组织要登记为ISO 9001:2000认证，就必须针对上述每个方面的质量要求（以及其他方面的质量要求）制定相关的政策和规程，并且有能力证明组织活动的确是按照这些政策和规程实施的。希望进一步了解有关ISO 9001:2000信息的读者可参考[Ant06]、[Mut03]或[Dob04]。

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。

编制质量体系文件。
描述过程。
编制操作手册。
制定控制（更新）文件的方法。
制定记录保持的方法。

支持质量控制和质量保证。
提高所有利益相关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。
制定所有利益相关者间的交流机制。

为质量管理体系建立评审机制。
确定评审方法和反馈机制。
制定跟踪程序。

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、试验）。
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。

16.8 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[IEE93]。该标准建议SQA计划应包括：(1) 计划的目的

⊖ 这里给出的实际上是国际标准ISO 9001：1994的内容。而ISO 9001：2000的主要内容包括：质量管理体系、管理者职责、资源管理、产品实现、测量、分析和改进。该国际标准已转换为我国国家标准的标准号为GB/T 19001-2000。——译者注

和范围；（2）SQA覆盖的所有软件工程工作产品的描述（例如，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）软件配置管理的规程（第22章）；（7）收集、保护和维护所有SQA相关记录的方法；（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：使用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，SQA工具类通常包含很多软件测试工具（见第17章到第20章）。

代表性工具：[⊖]

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 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。

NASA Quality Resources，由Goddard Space Flight Center (sw-assurance.gsfc.nasa.gov/index.php)开发，提供了有用的表格、模板、检查清单和SQA工具。

16.9 小结

软件质量保证是在软件过程中的每一步都进行的“普适性活动”。SQA包括：对方法和工具有效应用的规程，对诸如技术评审和软件测试等质量控制活动的监督，变更管理规程，保证符合标准的规程，以及测量和报告机制。

为了正确地进行软件质量保证，必须收集、评估和发布有关软件工程过程的数据。基于统计的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。

总之，应该注意Dunn和Ullman所说的话[Dun82]：“软件质量保证就是将质量保证的管理规则和设计规范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尺度。当成功实现上述映射时，其结果就是成熟的软件工程。

习题与思考题

16.1 有人说“差异控制是质量控制的核心”，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？应该如何控制这些差异？

16.2 如果客户不断改变他想做的事情，是否还有可能评估软件质量？

16.3 质量和可靠性是两个相关的概念，但在许多方面却有根本的不同，请就此进行讨论。

16.4 一个正确的程序还能不可靠吗？请加以解释。

⊖ 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。

16.5 一个正确的程序能不表现出高质量吗？请加以解释。

16.6 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？

16.7 假设赋予你改进组织中软件质量的职责，你要做的第一件事是什么？然后呢？

16.8 除了可以统计对错误和缺陷之外，还有哪些可以统计的软件特征具有质量意义？它们是什么？能否直接测量？

16.9 软件中的MTBF概念仍然不断受到批评，请解释为什么？

16.10 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。

16.11 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。

推荐读物与阅读信息

Hoyle（《Quality Management Fundamentals》，Butterworth-Heinemann, 2007）、Tian（《Software Quality Engineering》，Wiley-IEEE Computer Society Press, 2005）、El Emam（《The ROI from Sofware Quality》，Auerbach, 2005）、Horch（《Practical Guide to Software Quality Management》，Artech House，2003）、Nance和Arthur（《Managing Software Quality》，Springer，2002）的书是关于规范的计算机软件质量保证规程方面的优秀管理水平的展示。Deming[Dem86]、Juran（《Juran on Quality by Design》，Free Press, 1992）和Crosby（[Cro79]和《Quality Is Still Free》，McGraw-Hill, 1995）的书虽然不是软件方面的书，但是对软件开发负责的高级经理们来说是必读的。Gluckman和Roome（《Everyday Heroes of the Quality Movement》，Dorset House, 1993）通过讲述质量过程中参与者的一个故事，把质量问题人性化了。Kan（《Metrics and Models in Software Quality Engineering》，Addison-Wesley, 1995）提出了软件质量的量化观点。

Evans（《Total Quality: Management, Organization and strategy》，4th ed., South-Western College Publishing, 2004）、Bru（《Six Sigma for Managers》，McGraw-Hill, 2005）和Dobb（《ISO9001:2000 Quality Registration Step-by-Step》，3rd ed., Butterworth-Heinemann, 2004）分别是许多关于TQM，Six Sigma和ISO 9001:2000书籍的代表。

Pham（《System Software Reliability》，Springer, 2006）、Musa（《Software Reliability Engineering: More Reliable Software, Faster Development and Testing》，2nd ed., McGraw-Hill, 2004）和Peled（《Software Reliability Methods》，Springer, 2001）撰写了介绍测量和分析软件可靠性方法的实用指南。

Vincoli（《Basic Guide to System Safety》，Wiley, 2006）、Dhillon（《Engineering Safety》，World Scientific Publishing Co., Inc., 2003）、Hermann（《Software Safety and Reliability》，Wiley-IEEE Computer Society Press, 2000）、Storey（《Safety-Critical Computer Systems》，Addison-Wesley, 1996）及Leveson [Lev95]是目前出版的最全的讨论软件安全和系统安全的书籍。此外，van der Meulen（《Definitions for Hardware and Software Safety Engineers》，Springer-Verlag, 2000）提供了一个完整的可靠性和安全性方面重要概念和术语的汇编；Gartner（《Testing Safety-Related Software》，Springer-Verlag, 1999）提供了测试安全关键系统的专业指导。Friedman和Voas（《Software Assessment: Reliability Safety and Testability》，Wiley，1995）提供了评估可靠性和安全性的有用模型。Ericson（《Hazard Analysis Techniques for System Safety》，Wiley，2005）讲述了日益重要领域的风险分析。

网上有软件质量保证和相关主题的大量信息资源。SQA相关的最新WWW资源可在SEPA网站www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

⊖ 目前ISO 9001: 2000在计算机软件的使用指南已从ISO 9000-3改版为ISO 90003。——译者注

第17章

Software Engineering：A Practitioner's Approach (Seventh Edition)

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？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小部分？当向一个大型系统加入新的构件时，对于已经做过的测试，是否还要重新测试？什么时候需要客户参与测试工作？当制定测试策略时，就需要回答上述及一些其他问题。

人员：软件测试策略由项目经理、软件工程师及测试专家来制定。

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，会遗漏错误。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“小范围”开始，并逐步过渡到“软件整体”。这意味着，早期的测试关注单个构件或相关的一小组构件，利用测试发现封装在构件中的数据错误和处理逻辑错误。当完成单个构件的测试后，需要将构件集成起来，直到建成整个系统。这时，执行一系列的高阶测试（high-order test）以发现不满足顾客需求的错误。随着错误的发现，必须利用调试过程对错误进行诊断和纠正。

工作产品：测试规格说明是将软件测试团队的具体测试方法文档化。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及将要进行测试的类型。

质量保证措施：在测试进行之前通过对测试规格说明进行评审，可以评估测试用例及测试任务的完整性。有效的测试计划和规程可以引导有秩序地构造软件，并且在构造过程中能够发现各个阶段引入的错误。

关键概念

α测试
β测试
类测试
配置评审
调试
部署测试
独立测试组
集成测试
回归测试
系统测试
单元测试
确认测试
V&V

软件测试策略提供了一张路线图：描述将要进行的测试步骤，这些步骤计划和执行的时机以及需要的工作量、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，以促进测试方法的定制；同时又必须足够严格，以支持在项目进行过程中对项目进行合理策划和追踪管理。Shooman[SHO83]对这些问题进行了讨论：

> 从许多方面看，测试都是一个独立的过程，并且同软件开发方法一样，测试类型也有很多种。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内在的初始错误。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”。本章讨论软件测试策略，本书的第18～20章介绍实施测试策略的测试方法和测试技术。

17.1 软件测试的策略性方法

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到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，且具备下述一般特征：

- 为完成有效的测试，应该进行有效的、正式的技术评审（第15章）。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软件工程方法和不同的时间点。
- 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，但任何测试策略都必须包括调试。

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/。

软件测试策略必须提供必要的低级测试，可以验证小段源代码是否正确实现，也要提供高级测试，用来确认系统的主要功能是否满足用户需求。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往往是在软件完成的最后期限的压力开始呈现时才开始进行的，因此，测试的进度必须是可测量的，并且应该让问题尽可能早地暴露。

17.1.1 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，而确认指的是确保开发的软件可追溯到客户需求的另外一系列活动。Boehm[BOE81]用另一种方式说明了这两者的区别：

“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

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？

验证与确认的定义包含很多软件质量保证活动（第16章）[⊖]。

验证与确认包含广泛的SQA活动：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、验收测试和安装测试。虽然测试在验证与确认中起到了非常重要的作用，很多其他活动也是必不可少的。

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引发的所有错误。应该在整个软件过程中注重质量和错误检测。

测试确实为软件质量的评估（更实际地说是错误的发现）提供最后的堡垒。但是，测试不应当被看做安全网。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佳。”在软件工程的整个过程中，质量已经被包含在软件之中了。方法和工具的正确运用、有效的正式技术评审、坚持不懈的管理与测量，这些都形成了在测试过程中所确认的质量。

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有效的方法来确认软件质量”。

⊖ 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，而确认是在对需求进行评审和认可时进行的，也许更晚一些，当系统投入运行时由用户进行的。另外一些人将单元测试和集成测试（17.3.1节和17.3.2节）看成验证，而将高阶测试（17.6节和17.7节）看做确认。

17.1.2 软件测试的组织

"乐观主义是编程的职业障碍；测试是治疗良方。"——Kent Beck

对每个软件项目而言，在测试开始时就会存在固有的利害关系冲突。要求开发软件的人员对该软件进行测试，这本身似乎是没有恶意的！毕竟，谁能比开发者本人更了解程序呢？遗憾的是，这些开发人员感兴趣的是急于显示他们所开发的程序是无错误的，是按照客户的需求开发的，而且能按照预定的进度和预算完成。这些利害关系会影响软件的充分测试。

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企图，试图摧毁软件工程师所建造的大厦。以开发者的观点来看，可以认为（心理学上）测试是破坏性的。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，客户也会发现。

上述的讨论通常会使人们产生下面的误解：（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。

软件开发人员总是要负责程序各个单元（构件）的测试，确保每个单元完成其功能或展示所设计的行为。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，独立测试组才开始介入。

KEY POINT 独立测试组没有软件开发者可能经历的"利益冲突"。

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。独立测试可以消除利益冲突。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，开发人员和测试组要密切配合，以确保进行充分的测试。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负责保证质量。"——Brian Marick

从分析与设计到策划和制定测试规程，ITG参与整个项目过程。从这种意义上讲，ITG是软件开发项目团队的一部分。然而，在很多情况下，ITG直接向软件质量保证组织报告，由此获得一定程度的独立性。如果ITG是软件工程组织的一部分，这种独立性是不可能获得的。

17.1.3 软件测试策略——宏观

什么是软件测试的总体策略？

可以将软件过程看做图17-1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，在需求分析中建立了软件的信息域、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（顺时针），每走一圈都会降低软件的抽象层次。

WebRef 有关软件测试人员的有用资源可在站点：www.SQAtester.com中找到。

软件测试策略也可以放在螺旋模型中来考虑（图17-1）。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（例如，构件、类或Web应用内容对象）。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（作为软件需求建模的一部分而建立）

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，以顺时针方向沿着流线向外螺旋前进，每转一圈都拓宽了测试范围。

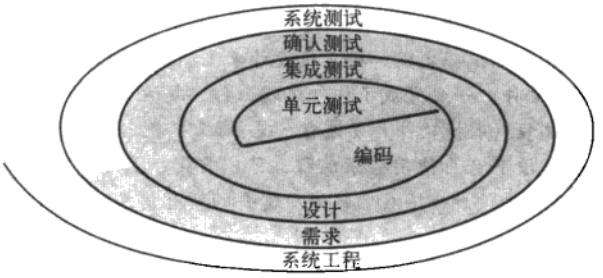


图17-1 测试策略

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4个步骤，如图17-2所示。最初，测试侧重于单个构件，确保它起到了单元的作用，因此称之为单元测试。单元测试充分利用测试技术，运行构件中每个控制结构的特定路径，以确保路径的完全覆盖，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终保证。

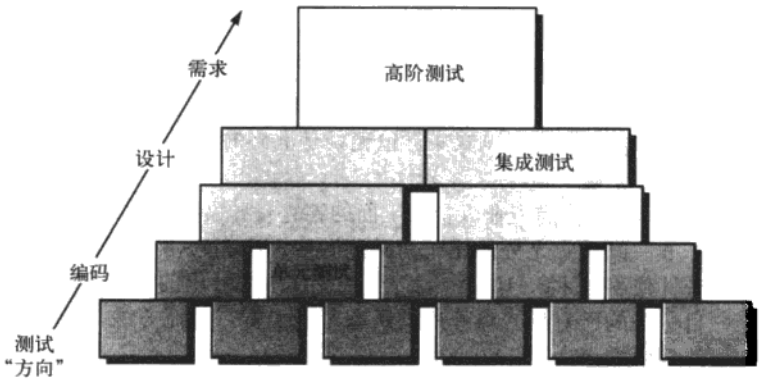


图17-2 软件测试步骤

最后的高阶测试步骤已经超出软件工程的边界，属于更为广泛的计算机系统工程范围。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。系统测试验证所有成分都能很好地结合在一起，且能满足整个系统的功能或性能需求。

SAFEHOME

准备测试

[场景] Doug Miller的办公室，继续构件级设计，并开始特定构件的构造。

[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，似乎是没有花费太多的时间讨论测试。

Vinod：对，但我们都有点忙。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：在对构件开始编码之前，我喜欢设计单元测试，因此，那是我一直尽力去做的。我有一个相当大的测试文件，一旦完成了构件编码工作，就运行这个测试文件。

Doug：那是极限编程（敏捷软件开发过程，见第3章）概念，不是吗？

Ed：是的。尽管我没有亲自使用极限编程，可以肯定，在建立构件之前设计单元测试是个好主意，这种单元测试的设计会给我们提供所需要的所有信息。

Jamie：我一直在做这件事情。

Vinod：我负责集成，因此，每当别人将构件传给我，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。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。

17.1.4 测试完成的标准

每当讨论软件测试时，就会引出一个典型的问题："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，这个问题没有确定的答案，只是有一些实用的答复和早期的尝试可作为经验指导。

? 什么时候我们完成测试？

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）身上转移到最终用户身上。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：当你的时间或资金耗尽时，测试就完成了。

WebRef

测试术语的综合词汇可以在站点www.testingstandards.co.uk/living_glossary.htm找到。

尽管很少有专业人员对上面的答复有异议，软件工程师还是需要更严格的标准，以确定充分的测试何时能做完。净室软件工程方法（第21章）提出了统计使用技术[ke100]：运行从统计样本中导出的一系列测试，统计样本来自目标群的所有用户对程序的所有可能执行。其他方法（如[Sin99]）提倡使用统计建模和软件可靠性理论来预测测试的完成。

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，提出有意义的指导性原则是可能的。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纯粹的直觉好得多。

17.2 策略问题

本章的后面几节介绍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，要使软件测试策略获得成功，必须解决下述问题：

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14章）。这些都应该以可测量的方式加以规定，从而保证测试结果无歧义性。

什么样的指导原则使软件测试策略获得成功？

明确地陈述测试目标。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、平均故障时间、发现和修正缺陷的成本、剩余缺陷的密度或出现频率、测试的工作时间，这些都应当在测试计划中陈述。

了解软件的用户并为每类用户建立用户描述。描述每类用户交互场景的用例，侧重于测试产品的实际使用，可以减少整个测试的工作量。

制定强调“快速周期测试”的测试计划。Gilb[Gil95]建议软件工程团队“对客户有用的至少是可作检查的功能增量和（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。

WebRef

相当好的测试资源列表可在www.io.com/~wazmo/qa/找到。

建立能够测试自身的“健壮”软件。软件应该利用防错技术（17.3.1节）进行设计。也就是说，软件应该能够诊断某些类型的错误。另外，软件设计应该包括自动化测试和回归测试。

“仅对最终用户需求的测试就像内部装饰设计师检查一座建筑物，会以牺牲地基、大梁和管道设备为代价。”——Boris Beizer

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15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。

17.3 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。虽然这种方法很有吸引力，但效果不好，可能得到的是有许多缺陷的软件，致使所有的利益相关者感到失望。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。尽管这种方法对很多人都缺少吸引力，但确实很有效。遗憾的是，许多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。下面几节将对这几种不同的测试进行描述。

17.3.1 单元测试

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。

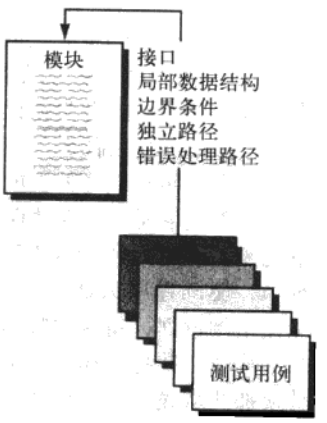


图17-3 单元测试

单元测试问题。图17-3对单元测试进行了概要描述。测试

⊖ 本书使用术语常规软件和传统软件来表示在多种应用领域中经常碰到的普通分层软件体系结构或调用-返回软件体系结构。传统的软件体系结构不是面向对象的，也不包括Web应用。

在为构件开发代码之前就设计单元测试用例是个不错的想法，有助于确保开发的代码能够通过测试。

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持其完整性；执行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。设计测试用例是为了发现因错误计算、不正确的比较或不适当的控制流而引起的错误。

有关“敏捷测试”的各种论文和资源的有用信息见tesing.com/agile/。

边界测试是最重要的单元测试任务之一。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并设置异常处理路径，以便当错误确实出现时重新确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，存在的一种趋势是在软件中引入异常处理，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。

确信已经设计了执行每个异常处理路径的测试。若没有，当执行这样的路径时，可能失败，从而加重了危险的形势。

按照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，加入了这样的异常处理信息：“ERROR! THERE IS NO WAY YOU CAN GET HERE”。但这个“出错信息”确实由一位客户在“用户培训”期间发现。

当评估异常处理时，应能测试下述的潜在错误：(1) 错误描述难以理解；(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) 异常条件处理不正确；(5) 错误描述没有提供足够的信息，对确定错误产生原因没有帮助。

单元测试过程。单元测试通常被认为是编码阶段的附属工作。可以在编码开始之前或源代码生成之后进行单元测试的设计。设计信息的评审可以指导建立测试用例，发现前面所讨论的各类错误，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程序和（或）桩程序。单元测试环境如图17-4所示。在大多数应用系统中，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被测构件（或被其调用）的模块。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。

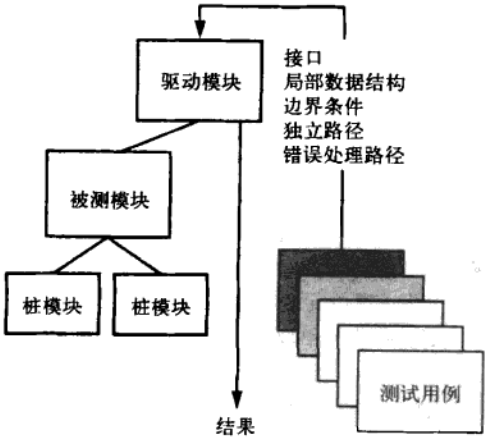


图17-4 单元测试环境

在没有资源做全面测试的情况下，只选择关键模块和复杂模块做单元测试。

驱动程序和桩程序都意味着测试开销。也就是说，两者都是必须编写的软件（通常并没有使用正式的设计），但并不与最终的软件产品一起交付。若驱动程序和桩程序保持简单，实际开销就会比较低。遗憾的是，使用“简单”的驱动程序和桩程序，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）。

当设计高内聚的构件时，就可以简化单元测试。当构件只强调一个功能时，测试用例数就会降低，且比较容易预见错误和发现错误。

17.3.2 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”的接口相连。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；子功能联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；全局数据结构可能产生问题。遗憾的是，问题还远不止这些。

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种懒惰的策略，注定会失败。当进行测试时，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集成策略。

当制定项目进度时，必须考虑将要采取的集成方式，使得构件在需要时是可用的。

自顶向下集成。自顶向下集成测试是一种构造软件体系结构的增量方法。模块的集成顺序为从主控模块（主程序）开始，沿着控制层次逐步向下，以深度优先或广度优先的方式将从属于（和间接从属于）主控模块的模块集成到结构中去。

参见图17-5，深度优先集成是首先集成位于程序结构中主控路径上的所有构件。主控路径的选择有一点武断，也可以根据特定应用系统的特征进行选择。例如，选择最左边的路径，首先集成构件M_1、M_2和M_5。其次，集成M_8或M_6（若M_2的正常运行是必需的），然后集成中间和右边控制路径上的构件。广度优先集成首先沿着水平方向，将属于同一层的构件集成起来。如图17-5中，首先将构件M_2、M_3和M_4集成起来，其次是下一个控制层M_5、M_6，依此类推。集成过程可以通过下列5个步骤完成：

自顶向下集成的步骤是什么？

1. 主控模块用作测试驱动模块，用直接从属于主控模块的所有模块代替桩模块；
2. 依靠所选择的集成方法（即，深度优先或广度优先），每次用实际模块替换一个从属桩模块；
3. 集成每个模块后都进行测试；
4. 在完成每个测试集之后，用实际模块替换另一个桩模块；
5. 可以执行回归测试（在本节的后面讨论）以确保没有引入新的错误。

回到第2步继续执行此过程，直到完成了整个程序结构的构造。

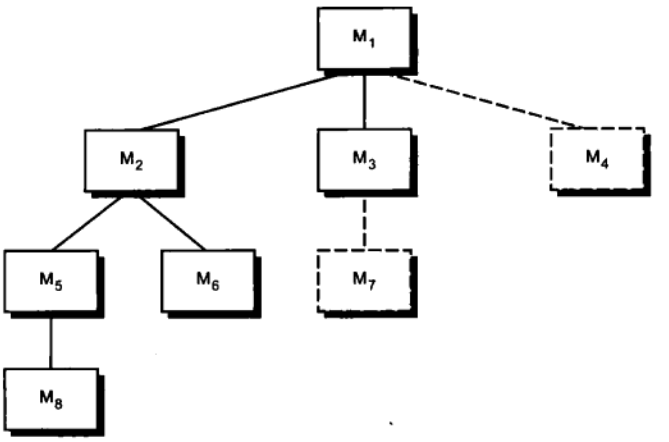


图17-5 自顶向下集成

当选择自顶向下集成方法时，可能会遇到什么问题？

自顶向下集成策略是在测试过程的早期验证主要控制点或决策点。在能够很好分解的程序结构中，决策发生在层次结构的较高层，因此首先要遇到。如果主控问题确实存在，尽早地发现是有必要的。若选择了深度优先集成方法，可以实现和展示软件的某个完整功能。较早的功能展示可以增强开发者、投资者及用户的信心。

自顶向下的集成策略相对来说似乎并不复杂，而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）使我们在特定测试与特定模块集成之间的相关性方法失去某些控制。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——自底向上集成——将在下面讨论。

自底向上集成测试。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. 编写驱动模块（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。

KEY POINT
自底向上集成排除了对复杂桩的需要。

遵循这种模式的集成如图17-6所示。连接相应的构件形成簇1、簇2和簇3，利用驱动模块（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，去掉驱动模块D_1和D_2，将这两个簇直接与M_a相连。与之相类似，在簇3与M_b连接之前去掉驱动模块D_3。最后将M_a和M_b与构件M_c连接在一起，依此类推。

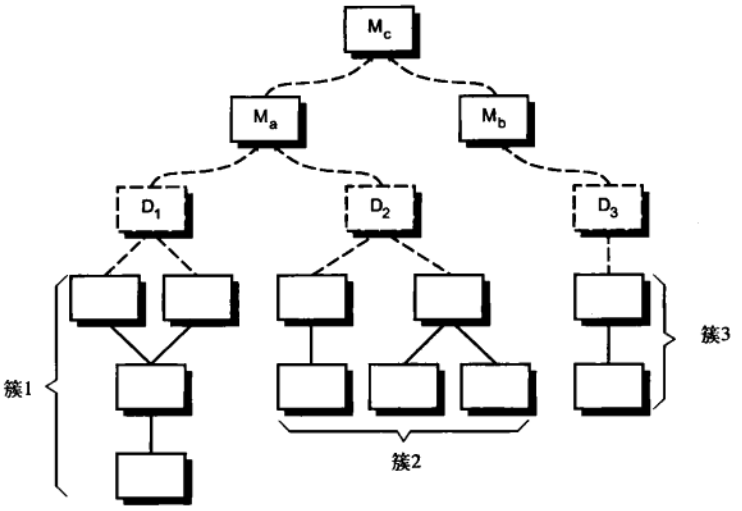


图17-6 自底向上集成

随着集成向上进行，对单独的测试驱动模块的需求减少。事实上，若程序结构的最上两层是自顶向下集成的，驱动模块的数量可以大大减少，而且簇的集成得到明显简化。

ADVICE
回归测试是减少“副效应”的重要方法。每次对软件做重要变更时（包括新构件的集成），都要进行回归测试。

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，建立了新的数据流路径，可能出现新的I/O，以及调用新的控制逻辑。这些变更可能会使原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测试过的某些子集，以确保变更没有传播不期望的副作用。

在较广的环境中，（任何种类的）成功测试都能发现错误并改正错误。无论什么时候修正软件，软件配置的某些方面（程序、文档或支持数据）也发生变更。回归测试有助于保证变更（由于测试或其他原因）不引入无意识行为或额外的错误。

回归测试可以手工进行，方法是重新执行所有测试用例的子集，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具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；
- 侧重于已发生变更的软件构件测试。

KEY POINT
冒烟测试被称为是一种滚动的（rolling）集成测试方法。每天对软件进行重构（加入新的构件），并进行冒烟测试。

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，应将回归测试套件设计成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，而且效率很低。

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，是时间关键项目的决定性机制，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到构建（build）中。一个构建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构建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. 每天将该构建与其他构建及整个软件产品（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。

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都能够对集成测试的进展做出实际的评估。McConnell[MCO96]是这样描述冒烟测试的：

> "将每日构建当成项目的心跳。如果没有了心跳，项目就死了。"——Jim McCarthy

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，以使得若构造通过测试，则可以假定它足够稳定以致能经受更彻底的测试。

> 从冒烟测试中可以得到什么好处？

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建（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，产品的质量就会更好。
- 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，新发现的错误可能来自刚加入到构建中的软件。
- 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状况。这就提高了团队的士气，并使管理者对项目进展有较好的把握。

WebRef

有关测试策略的评论可以在www.qalinks.com找到。

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所带来的测试难题。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，程序才作为一个实体存在[Mye79]。这个缺点则因测试用例设计比较容易和无需桩模块而得到补偿。

> 什么是"关键"模块，为什么应该标识关键模块？

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的较高层，用自底向上方法测试其从属层，这可能是最好的折衷。

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）涉及几项软件需求；（2）含有高层控制（位于程序结构的较高层次）；（3）复杂或容易出错；（4）具有明确的性能需求。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。

集成测试工作产品。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。这项工作产品包含测试计划和测试规程，并成为软件配置的一部分。测试可以分为若干个阶段和处理软件特定功能和行为特征的若干个构造来实施。例如，SafeHome安全系统的集成测试可以划分为以下测试阶段：

- 用户交互（命令输入与输出、显示表示、出错处理与表示）；
- 传感器处理（获取传感器输出、确定传感器的状态、作为状态的结果所需要的动作）；

- 通信功能（与中央监测站通信的能力）；
- 警报处理（测试遇到警报发生时的软件动作）。

每个集成测试阶段都刻画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构造(模块集)。下列准则和相应的测试应用于所有的测试阶段：

应该使用什么标准来设计集成测试?

接口完整性。当每个模块（或簇）引入到程序结构中时，要对其内部和外部接口进行测试；

功能有效性。执行旨在发现功能错误的测试；

信息内容。执行旨在发现与局部或全局数据结构相关错误的测试；

性能。执行旨在验证软件设计期间建立的性能边界的测试。

集成的进度、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）的简要描述侧重于可能需要特殊工作的特征。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步骤中对应的测试，其中也包括所有的测试用例（带注释以便后续工作参考）和期望的结果列表。

实际测试结果、问题或特例的历史要记录在测试报告中，若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具体要求进行剪裁。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。

17.4 面向对象软件的测试策略[⊖]

简单地说，测试的目标就是在现实的时间范围内利用可控的工作量找到尽可能多的错误。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19章）。

17.4.1 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。封装导出了类和对象的定义。这意味着每个类和类的实例包装有属性（数据）和处理这些数据的操作。封装的类常是单元测试的重点，然而，类中包含的操作（方法）是最小的可测试单元。由于类中可以包含很多不同的操作，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。

KEY POINT
面向对象软件的类测试与传统软件的模块测试相似。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，有必要在每个子类的环境中测试操作X。这意味着在面向对象环境中，以独立的方式测试操作X（传统的单元测试方法）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

⊖ 基本的面向对象概念在附录2中介绍。

测试是由封装在该类中的操作和类的状态行为驱动。

17.4.2 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（17.3.2节）已没有太大意义。另外，由于类的成分间直接或间接的相互作用，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]。

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做出反应的类集合。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。

面向对象系统的集成测试有两种不同的策略[Bin94b]。一种策略是基于线程的测试（thread-based testing），对响应系统的一个输入或事件所需的一组类进行集成。每个线程单独地集成和测试。应用回归测试以确保没有产生副作用。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，驱动模块和桩模块的使用也发生了变化。驱动模块可用于低层操作的测试和整组类的测试。驱动模块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。桩模块可用于类间需要协作但其中的一个或多个协作类还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个步骤。这里，借助设计试图发现协作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。

17.5 WebApp的测试策略

WebApp测试策略采用所有软件测试的基本原理，并使用面向对象系统所使用的策略和战术。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。

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. 对每个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。

WebRef
WebApp测试方面的优秀文章可以在www.stickyminds.com/testing.asp找到。

因为很多WebApp在不断进化，所以WebApp测试是Web支持人员所从事的一项持续活动，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。WebApp测试方法在第20章考虑。

17.6 确认测试

与所有其他测试步骤类似，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。在进行确认测试或系统级测试时，传统软件、面向对象软件及WebApp之间的差别已经消失，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客户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？"如果已经开发了软件需求规格说明文档，此文档就描述了所有用户可见的软件属性，并包含确认准则部分，确认准则部分就形成了确认测试方法的基础。

17.6.1 确认测试准则

软件确认是通过一系列表明与软件需求相符合的测试而获得的。测试计划列出将要执行的测试类，测试规程定义了特定的测试用例，设计的特定测试用例用于确保满足所有的功能需求，具有所有的行为特征，所有内容都准确无误且正确显示，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，可以接受；（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改正。此时往往必须与客户进行协商，确定解决缺陷的方法。

17.6.2 配置评审

确认过程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，且具有改善支持活动的必要细节。有时将配置评审称为审核（audit），将在第22章详细讨论。

"给予足够的关注，所有的bug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

17.6.3 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（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。

当为客户开发定制软件时，执行一系列验收测试能使客户确认所有的需求。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。

α测试和β测试之间的区别是什么？

若将软件开发为产品，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为α测试与β测试的过程，以期查找到似乎只有最终用户才能发现的错误。

α测试是由有代表性的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测试在一个或多个最终用户场所进行。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（现实存在的或想象的），并定期地报告给开发者。接到β测试的问题报告之后，开发人员对软件进行修改，然后准备向最终用户发布软件产品。

β测试的一种变体称为客户验收测试，有时是按照合同交付给客户时进行的。客户执行一系列的特定测试，试图在从开发者那里接收软件之前发现错误。在某些情况下（例如，大公司或政府系统），验收测试可能是非常正式的，可能会测试很多天，甚至好几个星期。

SAFEHOME

准备确认

[场景] Doug Miller的办公室，构件级设计及确定构件的构造正继续进行。

[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：将在三个星期内准备好第一个增量的确认，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，但还没有我们处理不了的事情。到目前为止，一切都很好。

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测试一起考虑了，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：我认为确认测试已经在我们的控制之中了。

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：我们的时间很紧了，Doug，我没有时间培训新人来做这项工作。

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：我仍然认为确认测试已经在我们的控制之中了。

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。

17.7 系统测试

“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

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。

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谓的争论之中，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。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。

17.7.1 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。

恢复测试是一种系统测试，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。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，则应计算平均恢复时间（Mean-Time-To-Repair，MTTR）以确定其值是否在可接受的范围之内。

17.7.2 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。做什么都可以。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。

17.7.3 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：“在系统失效之前，能将系统运行提高到什么程度？”

“若你正在尽力查找实际系统的bug，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

压力测试是一种要求以非正常的数量、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）设计可能在实际的运行系统中产生惨败的测试用例；（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。

压力测试的一个变体称为敏感性测试。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。敏感性测试试图在有效输入类中发现可能

会引发系统不稳定或者错误处理的数据组合。

17.7.4 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。在测试过程的任何步骤都可以进行性能测试。即使是在单元级，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完全集成之后，才能确定系统的真实性能。

性能测试经常与压力测试一起进行，且常需要硬件和软件工具。也就是说，以严格的方式测量资源（例如，处理器周期）的利用往往是必要的。当有运行间歇或事件（例如，中断）发生时，外部工具可以监测到，并可定期监测采样机的状态。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。

17.7.5 部署测试

在很多情况下，软件必须在多种平台及操作系统环境中运行。有时也将部署测试称为配置测试，是在软件将要在其中运行的每一种环境中测试软件。另外，部署测试检查客户将要使用的所有安装程序及专业安装软件（例如，“安装程序”），并检查用于向最终用户介绍软件的所有文档。

作为一个例子，考虑SafeHome软件的Internet访问版，此版本允许顾客远程监测安全系统。这就需要使用可能碰到的所有Web浏览器对SafeHome Web应用系统进行测试。更彻底的部署测试应该包括Web浏览器与不同操作系统（例如，Linux、Mac OS、Windows）的组合。由于安全是主要问题，一组完整的安全测试应该与部署测试结合起来进行。

SOFTWARE TOOLS

测试计划与管理

目的：这些工具辅助软件团队根据所选择的测试策略制订计划，并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、管理与控制、需求追踪、集成、错误追踪和报告生成。项目经理用这些工具作为项目策划工具的补充；测试人员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。

代表性工具：[⊖]

QaTraq测试用例管理工具，由Traq Software（www.testmanagement.com）开发，“鼓励以结构化方法进行测试管理”。

QADirector，由Compuware Corp.（www. compuware.com/qacenter）开发，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，包括测试管理和报告工具。

OpensourceTesting.org（www.opensourcetesting.org/testmgt.php）列出了各种开源测试管理和策划工具。

NI TestStand，由National Instruments Corp.（www.ni.com）开发，允许“开发、管理和运行以任何编程语言编写的测试序列。”

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

17.8 调试技巧

"我们惊奇地发现，并不是像我们想象的那样容易让程序正确。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

软件测试是一种能够系统地加以计划和说明的过程，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，调试是使错误消除的过程。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。

17.8.1 调试过程

调试并不是测试，但总是发生在测试之后[⊖]。参看图17-7，执行测试用例，对测试结果进行评估，而且期望的表现与实际表现不一致时，调试过程就开始了。在很多情况下，这种不一致的数据是隐藏在背后的某种原因所表现出来的症状。调试试图找到隐藏在症状背后的原因，从而使错误得到修正。

ADVICE 确保避免第3种结果：错误原因找到了，但所做的"改正"并没有解决问题，或者还引入了另外的错误。

为什么调试如此困难？

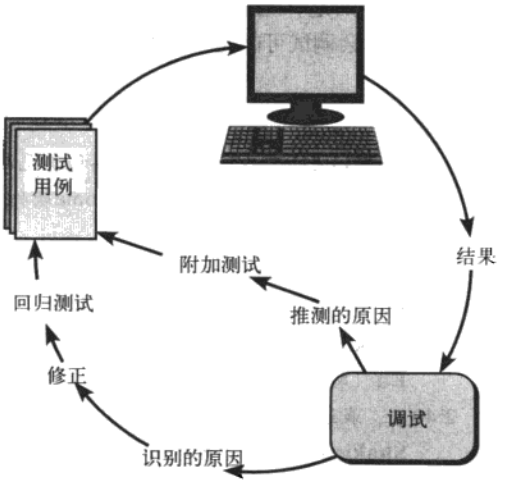


图17-7 调试过程

调试过程通常得到以下两种结果之一：（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。

"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

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，软件bug的以下特征为我们提供了一些线索：

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。高度耦合的构件（第8章）加剧了这种情况的发生；
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（例如，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. 症状可能是由计时问题而不是处理问题引起的；
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系统）；

⊖ 需要说明的是，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。

7. 症状可能时有时无，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的很多任务引起的。

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（如系统失效，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。

17.8.2 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。焦虑和不情愿接受错误存在的可能性等原因加剧了调试任务的难度。幸运的是，当bug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讨论。

SAFEHOME

调试

[场景] Ed的工作间，进行编码和单元测试。

[人物] Ed和Shakira——SafeHome软件工程团队的成员。

[对话]

Shakira（经过工作间门口时向里张望）：嘿……午饭时你在哪儿？

Ed：就在这里……工作。

Shakira：你看上去很沮丧……怎么回事？

Ed（轻声地叹息）：我一直忙于解决这个bug，从今天早晨9:30发现它之后，现在已下午2:40了，我还没有线索。

Shakira：我想大家都同意在调试我们自己的东西时花费的时间不应该超过一小时，我们请求帮助，怎么样？

Ed：好，但是……

Shakira（走进工作间）：什么问题？

Ed：很复杂。而且，我查看这个问题已有5个小时，你怎么能在5分钟内找到原因。

Shakira：真让我兴奋……什么问题？

（Ed向Shakira解释问题，Shakira看了大约30秒没有说什么，然后……）

Shakira（笑了）：哦，就是那个地方，在循环开始之前，变量setAlarmCondition是不是不应该设置为“false”？

（Ed不相信地盯着屏幕，向前躬着腰，开始对着监视器轻轻地敲自己的头。Shakira开怀大笑，起身走出去。）

ADVICE

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。

17.8.3 调试策略

不论使用什么方法，调试有一个基本目标：查找造成软件错误或缺陷的原因并改正。通过系统评估、直觉和运气相结合可以实现这个目标。Bradley[Bra85]是这样描述调试方法的：

调试是对过去2500年间发展起来的科学方法的直接应用。调试的基础是利用二分法，通过有效假设——该假设预测被检查的新值——定位问题的来源。

以一个简单的非软件问题为例：我房间的一盏台灯不亮了。若整个房间其他电器也不能工作，则一定是总闸或外边的线路坏了。我出去看邻居家是否也是黑的，若不是，我就把台灯插到另一个好的插座里试试，或把别的电器插到台灯的插座里检查一下。假设与测试就这样交替进行。

总的来说，有三种调试方法[Mye79]：（1）蛮干法；（2）回溯法；（3）原因排除法。这三种调试方法都可以手工执行，但现代的调试工具可以使调试过程更有效。

"修改一个已坏程序的第一步是让它重复失败（尽可能是在最简单的例子上）。"——T. Duff

调试策略。蛮干法可能是分离软件错误原因最常用但最低效的方法。当所有其他方法都失败的情况下，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。尽管产生的大量信息可能最终获得成功，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，潜在的回溯路径的数量可能会变得难以控制。

第3种调试方法——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。以上调试方法都可以使用辅助调试工具。当尝试调试策略时，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，而且许多商业调试环境也已经具备。集成开发环境（IDE）提供了一种方法，无需编译就可以捕捉特定语言的预置错误（例如，语句结束符的丢失、变量未定义，等等）。"可用的工具包括各种调试编译器、动态调试辅助工具（"跟踪工具"）、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。

SAFEHOME

调试

目的：这些工具为那些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：[⊖]

Borland Gauntlet，由Borland（www.borland.com）开发，辅助测试和调试。

Coverty Prevent SQS，由Coverty（www.coverty.com）开发，提供C++和Java的辅助调试。

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。

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。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

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[⊖]。因此，调试的最终箴言应该是："若所有的方法都失败了，就该寻求帮助！"。

17.8.4 纠正错误

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，不当修改造成的危害会超过带来的益处。Van Vleck[Van89]提出，在进行消除错误原因的"修改"之前，每个软件工程师应该问以下三个问题：

"最好的测试人员不是发现错误最多的人，而是纠正错误最多的人。"——Cem Kaner等人

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16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。

17.9 小结

软件测试在软件过程中所占的技术工作量比例最大。不考虑所构建软件的类型，系统测试计划、运行和控制策略从考虑软件的小元素开始，逐渐面向整个软件。

软件测试的目标是发现错误。对于传统软件，这个目标是通过一系列测试步骤达到的。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；确认测试验证软件需求的可追踪性；系统测试在软件集成为较大的系统时对软件进行确认。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，用于考虑软件的抽象层次都得到了扩展。

测试面向对象软件的策略开始于类中操作的执行，然后转到以集成为目的的基于线程的测试。线程是响应输入或事件的一组类。基于使用的测试关注那些不与其他类过多协作的类。

对Web应用系统的测试方法与面向对象系统是一样的。然而，所设计的测试用于检查内容、功能性、界面、导航以及Web应用系统的性能和安全性方面。

与测试（测试是一种系统的、有计划的活动）不同的是，调试必须被看做一种技术。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。

习题与思考题

17.1 用自己的话描述验证与确认的区别。两者都要使用测试用例设计方法和测试策略吗？

17.2 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？

17.3 使用17.1.3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，会出现哪些可能的复杂情况？

⊖ 当对软件进行设计和编码时，结对编程（建议将结对编程作为第3章所讨论的极限编程模型的一部分）的概念提供了一种调试机制。

17.4 为什么具有较高耦合度的模块难以进行单元测试?

17.5 “防错法”(17.2.1节)的概念是一个非常有效的方法。当发现错误时,它提供了内置调试帮助:

a. 为防错法开发一组指导原则。

b. 讨论利用这种技术的优点。

c. 讨论利用这种技术的缺点。

17.6 项目的进度安排是如何影响集成测试的?

17.7 在所有的情况下,单元测试都是可能的或是值得做的吗?提供实例来说明你的理由。

17.8 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。

17.9 为本书讨论的SafeHome系统开发一个完整的测试策略,并以测试规格说明的方式形成文档。

17.10 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,并在你的局部范围内将这个指南发布给其他人。

推荐读物与阅读信息

实际上,每本软件测试的书都讨论测试策略和测试用例设计方法。Everett和Raymond(《Software Testing》, Wiley-IEEE Computer Society Press, 2007)、Black(《Pragmatic Software Testing》, Wiley, 2007)、Spiller与他的同事(《Software Testing Process: Test Management》, Rocky Nook, 2007)、Perry(《Effective methods for Software Testing》, 3rd ed., Wiley, 2005)、Lewis(《Software Testing and Continuous Quality Improvement》, 2nd ed., Auerbach, 2004)、Loveland与他的同事(《Software Testing Techniques》, Charles River Media, 2004)、Burnstein(《Practical Software Testing Techniques》, Springer, 2003)、Dustin(《Effective Software Testing》, Addison-Wesley, 2002)、Kaner和他的同事(《Lessons learned in Software Testing》, Wiley, 2001)所写的书只是讨论测试原理、概念、策略和方法的众多书籍中的一小部分。

对于敏捷软件开发方法有兴趣的读者,Crispin与House(《Testing Extreme Programming》, Addison-Wesley, 2002)及Beck(《Test Driven Development: By Example》, Addison-Wesley, 2002)针对极限编程技术描述了测试策略与战术。Kamer与他的同事(《Lessons Learned in Software Testing》, Wiley, 2001)描述了每个测试人员应该学习的300多条实用的“教训”(指导原则)。Watkins(《Testing IT: An Off-the Shelf Testing Process》, Cambridge University Press, 2001)为所有类型的软件(开发的和获取的)建立了有效的测试框架。

Sykes与McGregor(《Practical Guide to Testing Object-Oriented Software》, Addison-Wesley, 2001)、Bashir与Goel(《Testing Object-Oriented Software》, Springer-Verlag, 2000)、Binder(《Testing Object-Oriented Systems》, Addison-Wesley, 1999)、Kung与他的同事(《Testing Object-Oriented Software》, IEEE Computer Society Press, 1998)以及Marick(《The Craft of Software Testing》, Prentice-Hall, 1997)描述了测试面向对象系统的策略与方法。

Grotker和他的同事(《The Developer's Guide to Debugging》, Springer, 2008)、Agans(《Debugging》, Amacon, 2006)、Zeller(《Why Programs Fail: A Guide to Systematic Debugging》, Morgan Kaufmann, 2005)、Tells与Heieh(《The Science of Debugging》, The Coreolis Group, 2001)、Robbins(《Debugging Applications》, Microsoft Press, 2000)所编写的书中包括调试指南。Kaspersky(《Hacker Debugging Uncovered》, A-list Publishing, 2005)讲述了调试工具的技术。Younessi(《Object-Oriented Defect Management of Software》, Prentice-Hall, 2002)描述了面向对象系统的缺陷管理技术。Beizer[BEI84]描述了有趣的“bug 分类”,这种分类引领了很多制定测试计划的有效方法。

Madisetti和Akgul（《Debugging Embeded Systems》，Spring, 2007）、Robbins（《Debugging Microsoft .NET 2.0 Applications》，Microsoft Press, 2005）、Best（《Linux Debugging and Performance Tuning》，Prentice-Hall, 2005）、Ford和Teorey（《Practical Debugging in C++》，Prentice-Hall, 2002）、Brown（《Debugging Perl》，McGraw-Hill, 2000）、Mitchell（《Debugging Java》，McGraw-Hill, 2000）都针对标题所指的环境，讲述了调试的特殊性质。

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www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

测试传统的应用系统

要点浏览

概念：一旦生成了源代码，必须对软件进行测试，以便在交付给客户之前尽可能多地发现（和改正）错误。目标是设计一系列极有可能发现错误的测试用例。但是，如何做呢？这就是软件测试技术发挥作用的地方。这些技术为设计测试提供系统化的指导：(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，随着测试过程的进行，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，但只有这些做法是远远不够的。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：(1) 利用“白盒”测试用例设计技术执行程序内部逻辑；(2) 利用“黑盒”测试用例设计技术确认软件需求。用例可辅助测试的设计，在软件确认的层面发现错误。在每种情况下，其基本意图都是以最少的工作量和最少的时间来发现最大数量的错误。

工作产品：设计一组测试用例使其不仅测试内部逻辑、接口、构件协作，还测试外部需求，并形成文档。定义期望结果，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
基本路径测试
黑盒测试
边界值分析
控制结构测试
环路复杂性
等价划分
流图
基于图的测试方法
图矩阵
基于模型的测试
正交数组测试
模式
专门的环境
白盒测试

对于本质上具有建设性的软件工程师来说，测试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]有效地描述了这种情况：

> 有这样一个神话：若我们确实擅长编程，就应当不会有错误。只要我们确实很专注，只要每个人都使用结构化编程，采用自顶向下的设计方法……那么就不应该有错误。所以才有了这样的神话。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，也是失败的一剂良药。测试的枯燥是对我们错误的处罚。为什么被罚？由于我们是人类？为什么内疚？由于没能达到非人的完美境界？由于没能区分另一个程序员所想的和所说的之间存在的差异？由于没有心灵感应？由于没有解决交流问题？……由于人类四千年历史的缘故？

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”！

本章针对传统的应用系统讨论软件测试用例设计技术。测试用例设计关注创建测试用例的一系列技术，这些测试用例的设计符合总体测试目标及第17章所述的测试策略。

18.1 软件测试基础

"每个程序都做对某件事。可是那恰恰不一定是我们想让它做的事情！"
——作者不详

测试的目标是发现错误，并且好的测试发现错误的可能性较大。因此，软件工程师在设计与实现基于计算机的系统或产品时，应该想着可测试性。同时，测试本身必须展示一系列特征，达到以最少工作量发现最多错误的目标。

可测试性。James Bach[⊖]为可测试性提供了下述定义："软件可测试性就是（计算机程序）能够被测试的容易程度。"可测试的软件应具有下述特征。

可测试性的特征是什么？

可操作性。"运行得越好，越能有效地测试。"若设计和实现系统时具有质量意识，那么妨碍测试执行的错误将很少，从而使测试顺利进行。

可观察性。"你所看见的就是你所测试的。"作为测试的一部分所提供的输入会产生清楚的输出。测试执行期间系统状态和变量是可见的或可查询的，不正确的输出易于识别，内部错误会被自动检测和报告，源代码是可访问的。

可控制性。"对软件控制得越好，测试越能被自动执行和优化。"通过输入的某些组合可以产生所有可能的输出，并且输入/输出格式是一致的和结构化的。通过输入的组合，所有代码都可以执行到。测试工程师能够控制软硬件的状态和变量，能够方便地对测试进行说明、自动化执行和再现。

可分解性。"通过控制测试范围，能够更快地孤立问题，完成更灵巧的再测试。"软件由能够进行单独测试的独立模块组成。

简单性。"需要测试的内容越少，测试的速度越快。"程序应该展示功能简单性（例如，程序特性集是满足需求的最低要求）、结构简单性（例如，将体系结构模块化以限制错误的传播）以及代码简单性（例如，采用编码标准以使代码易于审查和维护）。

稳定性。"变更越少，对测试的破坏越小。"软件的变更不经常发生，当发生时是可以控制的，且不影响已有的测试，软件失效后得到良好恢复。

易理解性。"得到的信息越多，进行的测试越灵巧。"体系结构设计以及内部构件、外部构件和共享构件之间的依赖关系能被较好地理解。技术文档可随时获取、组织合理、具体而详细、并且准确。设计的变更要通知测试人员。

"软件中的错误比其他技术中的错误更普通、普遍且更烦人。" —— David Parnas

可以使用Batch所建议的属性来开发易于测试的软件配置（即程序、数据和文档）。

测试特征。关于测试本身有哪些特征呢？Kaner、Falk和Nguyen[Kan93]提出"好"的测试具有以下属性：

什么是"好"的测试？

好的测试具有较高的发现错误的可能性。为达到这个目标，测试人员必须理解软件并尝试设想软件怎样才能失败。理想情况下，应探测失败的类别。例如，GUI（图形用户界面）中的一类潜在错误是不能识别正确的鼠标位置。应该设计一个测试集来测试鼠标，以试图展示鼠标位置识别中的错误。

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。每个测试都应该有不同的目标（即使是细微的差别）。

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别错误的测试。

⊖ 后面几段取得了James Bach（copyright 1994）的使用许可，并对最初出现在新闻组comp.software-eng的资料进行了改编。

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软件工程团队的成员。

[对话]

Vinod：这些是你打算用于测试操作passwordValidation的测试用例吗？

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……你提到正确的密码是8080，对吗？

Ed：嗯。

Vinod：你指定密码1234和6789是要测试在识别无效密码方面的错误？

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。但是我并不认为运行1234和6789两个输入有多大意义。这两个输入是冗余的……测试同样的事情，不是吗？

Vinod：确实是这样。倘若输入1234不能发现错误……换句话说……操作passwordValidation指出它是无效密码，那么输入6789也不可能显示任何新的东西。

Ed：我明白你的意思。

Vinod：我不是吹毛求疵……只是我们做测试的时间有限，因此，好的方法是运行最有可能发现新错误的测试。

Ed：没问题……我再想想。

18.2 测试的内部视角和外部视角

"在设计测试用例中只有一条规则，那就是覆盖所有特征，但并不创建太多的测试用例。"——Tsuneo Yamaura

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）了解已设计的产品要完成的指定功能，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）了解产品的内部工作情况，可以执行测试以确保"所有的齿轮吻合"——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。第一种测试方法采用外部观察，也称为黑盒测试，第二种方法需要内部观察，也称为白盒测试[⊖]。

KEY POINT

只有在构件设计（或源代码）存在之后，才设计白盒测试。此时，一定要获得程序的逻辑细节。

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功能方面[⊜]，而不考虑软件的内部结构。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）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。

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"。我们需要做的只是识别所有的逻辑路径，开发相应的测试用例，执行测试用例，并评估结果；即，生成测试用例，彻底地测试程序逻辑。遗憾的是，穷举测试存在某种逻辑问题，即使对于小程序，可能的逻辑路径的数量也可能非常大。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

⊖ 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。

⊜ 原文为"some fundamental aspect"，应为"some functional aspects"。——译者注

数量的重要逻辑路径，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，我们假设已经开发了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）做穷举测试。在1毫秒内，处理器可以开发一个测试用例，执行测试用例，并评估测试结果。若处理器每天工作24小时，每年工作365天，要对这个程序做穷举测试，则要工作3170年。不可否认，这将对大多数的开发进度造成巨大障碍。

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。

18.3 白盒测试

bug潜伏在角落及在边界处聚集。——Boris Beizer

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所描述的控制结构来生成测试用例。利用白盒测试方法导出的测试用例可以：(1) 保证一个模块中的所有独立路径至少被执行一次；(2) 对所有的逻辑判定均需测试取真（true）和取假（false）两个方面；(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。

18.4 基本路径测试

基本路径测试是由Tom McCabe[McC76]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计算出过程设计（procedural design）的逻辑复杂性测量（logical complexity measure），并以这种测量来指导定义执行路径的基本集。执行该基本集导出的测试用例保证程序中的每一条语句至少执行一次。

18.4.1 流图表示

ADVICE

仅当构件的逻辑结构复杂的情况下，才应该画流图。使用流图可以更轻易地追踪程序路径。

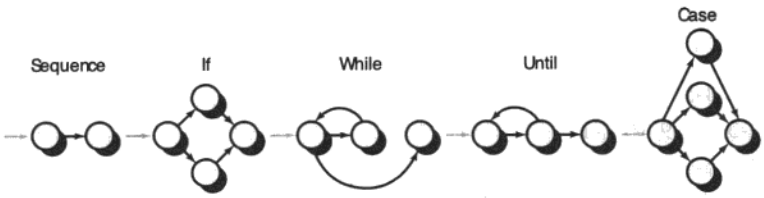
在考虑基本路径方法之前，必须介绍一种简单的控制流表示方法[⊖]，称为流图（或程序图）。流图利用图18-1所示的表示描述逻辑控制流。每种结构化构造（第10章）都有相应的流图符号。

为了说明流图的使用，考虑图18-2a所示的过程设计表示。这里，流程图用于描述程序的控制结构。图18-2b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18-2b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（例如，见表示 if-else-then结构的流图符号）。由边和结点限定的区域称为域。当计算域时，将图的外部作为一个域[⊖]。

⊖ 事实上，不使用流图也可以执行基本路径测试方法，但是，流图是用于理解控制流和解释方法的一种有用表示。

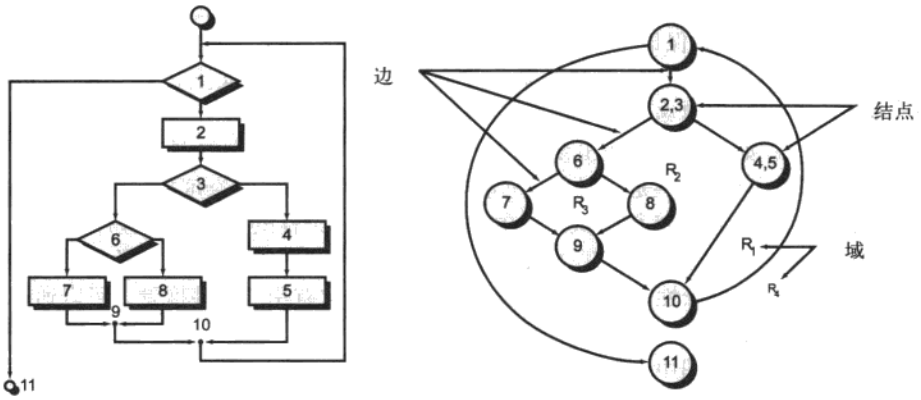
⊖ 18.6.1节更详细地讨论图及其使用。

流图中的结构化构造



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18-1　流图表示



a) 流程图　　b) 流图

图18-2　流程图与流图

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、NOR）时，复合条件就出现了。图18-3给出了一段程序设计语言（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，其特征是由它发射出两条或多条边。

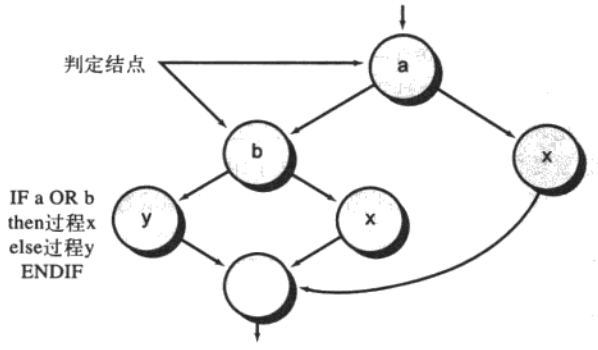


图18-3　复合逻辑

18.4.2　独立程序路径

独立路径是任何贯穿程序的、至少引入一组新处理语句或一个新条件的路径。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18-2b所示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。

路径1、2、3和4构成图18-2b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（基本集合），则可以保证程序中的每条语句至少执行一次，且每个条件的取真和取假都被执行。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，可以导出很多不同的基本集合。

ADVICE

在预见易于出错的模块方面，环复杂性是一种有用的度量，可以用于做测试计划及测试用例设计。

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，环复杂性的值定义了程序基本集合中的独立路径数，并提供了保证所有语句至少执行一次所需测试数量的上限。

如何计算环复杂性？

环复杂性以图论为基础，并提供了非常有用的软件度量。可以通过以下三种方法之一来计算环复杂性：

1. 流图中域的数量与环复杂性相对应。

2. 对于流图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. 对于流图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。

KEY POINT

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界。

再回到图18-2b中的流图，环复杂性可以通过上述3种算法来计算：

1. 该流图有4个域。

2. $V(G)$ =11(边数)－9(结点数)＋2＝4。

3. $V(G)$ =3(判定结点数)＋1＝4。

因此，图18-2b中流图的环复杂性是4。

更重要的是，$V(G)$的值提供了组成基本集合的独立路径的上界，并由此得出覆盖所有程序语句所需设计和运行的测试数量的上界。

SAFEHOME

使用环复杂性

[场景] Shakira的工作间。

[人物] Vinod和Shakira，SafeHome软件工程团队的成员，他们正在为安全功能准备测试计划。

[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：嗯？

Vinod：环复杂性。只要计算每个构件中每个操作的环复杂性。看看哪些操作的$V(G)$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）：好的，这计算看上去并不难。我试一试。具有最高$V(G)$值的操作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。但这至少降低了必须进行白盒测试的构件数。

18.4.3 导出测试用例

"只是由于在将64位浮点值转换为16位整数的操作中包含了一个软件缺陷（代码错误），Ariane 5型火箭在升空时发生爆炸。这枚火箭和它的4颗卫星都没有投保，价值5亿美元。如果进行路径测试是可以发现这个错误的，但由于预算原因被否决了。"——一篇新闻报道

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一系列步骤。以图18-4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18.4.1节给出的符号和构造规则创建流图。参见图18-4中过程average的PDL描述，将那些PDL语句进行编号，并映射到相应的流图结点，以此来创建流图。图18-5给出了相应的流图。

2. **确定所得流图的环复杂性。**通过运用18.4.2节中描述的算法来确定环复杂性$V(G)$的值。应该注意到，不建立流图也可以确定$V(G)$，方法是通过计算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。在图18-5中：

$$V(G)=6(\text{域数})$$
$$V(G)=17(\text{边数})-13(\text{结点数})+2=6$$
$$V(G)=5(\text{判定结点数})+1=6$$

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数量的上界。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。

```
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
      5      THEN increment total.valid by 1;
      7           sum = s 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18-4 已标识结点的PDL

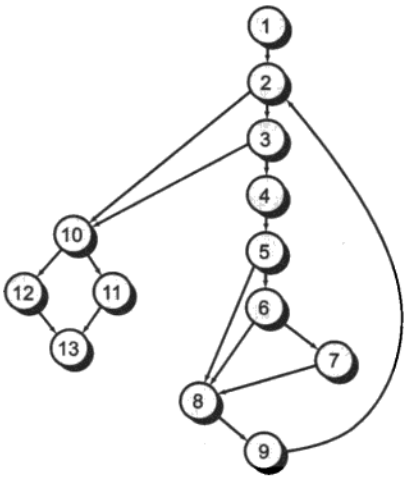


图18-5 过程average的流图

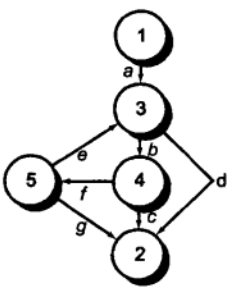
4. 准备测试用例，强制执行基本集合中的每条路径。测试人员应该选择测试数据，以便在测试每条路径时适当地设置判定结点的条件。执行每个测试用例并将结果与期望值进行比较。一旦完成了所有的测试用例，测试人员可以确信程序中所有的语句至少已被执行一次。

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。

18.4.4 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。有一种称为图矩阵（graph matrix）的数据结构，对于开发辅助基本路径测试的软件工具相当有用。

图矩阵是一种方阵，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的连接（边）。图18-6给出了一个简单流图及相应的图矩阵[Bei90]。



流图

结点 \ 连接到的结点	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

图矩阵

图18-6 图矩阵

什么是图矩阵，如何对其进行扩展以用于测试？

如图18-6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。

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（连接存在）或0（连接不存在），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。

18.5 控制结构测试

"将执行测试看得比设计测试更重要是一个典型的错误。"——Brian Marick

18.4节所描述的基本路径测试是控制结构测试技术之一。虽然基本路径测试简单、高效，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。

18.5.1 条件测试

KEY POINT 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（¬）算符的关系表达式。关系表达式的形式如下：

E1＜关系算符＞E2

其中，E1和E2是算术表达式，而＜关系算符＞是下列算符之一："＜"、"≤"、"＝"、"≠"（不等于）、"＞"或"≥"。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（|）、AND（&）和NOT（¬）。不含关系表达式的条件称为布尔表达式。

"好的测试人员是注意到"奇怪的事情"就会对它采取行动的大师。"——Brian Marick

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。

18.5.2 数据流测试

数据流测试方法[Fra93]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，

DEF(*S*)＝{*X* | 语句*S*包含*X*的定义}

USE(*S*)＝{*X* | 语句*S*包含*X*的使用}

若语句*S*是一个if或loop语句，它的DEF集合为空，而USE集合取决于语句*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'*中仍有效。

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DEF(*S*)和USE(*S'*)中，且在语句*S*中定义的*X*在语句*S'*中有效。

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。

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，只在很少的情况下，分支才不被DU测试覆盖，例如，if-then-else中的then部分没有定义变量且不存在else部分。在这种情况下，if语句的else分支并不需要由DU测试覆盖。

18.5.3 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却往往很少关注它。

循环测试是一种白盒测试技术，完全侧重于循环构成的有效性。可以定义4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（图18-7）。

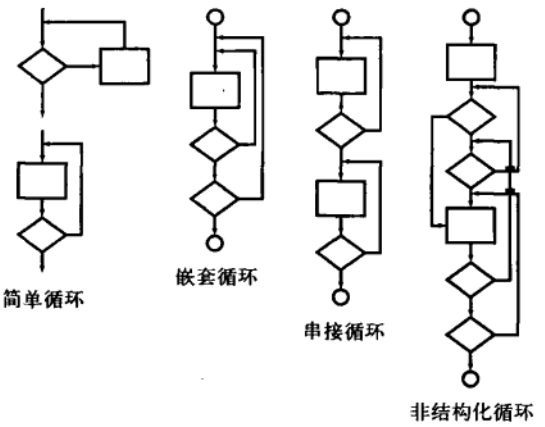


图18-7 循环的类别

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（例如，循环计数）值最小，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。

ADVICE 不能对非结构化循环进行有效测试，需要对它们进行重新设计。

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。

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10章）。

18.6 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）不正确或遗漏的功能；（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。

"犯错误的是人，找到错误的是神。" —— Bobert Dunn

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（见第17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：

什么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速率和数据量？
- 特定类型的数据组合会对系统运行产生什么样的影响？

KEY POINT

图表示数据对象与程序对象间的关系，它使我们能够设计测试用例，查找与这些关系有关的错误。

通过运用黑盒测试技术，可以生成满足[illegible]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。

18.6.1 基于图的测试方法

黑盒测试的第一步是理解软件中建模的对象[⊖]及这些对象间的关系。这一步一旦完成，下一步就是定义一系列验证"所有对象之间具有预期关系"的测试[Bei95]。换言之，软件测试首先是创建重要对象及其关系图，然后设计覆盖图的一系列测试用例，使得图中的每个对象和关系都测试到，并发现错误。

为完成这些步骤，软件工程师首先要创建图，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18-8a所示。结点用圆表示。而连接有几种形式，有向连接（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。

考虑一个简单的例子，字处理应用中图的一部分，如图18-8b所示，其中

对象#1＝新建文件（菜单选择）

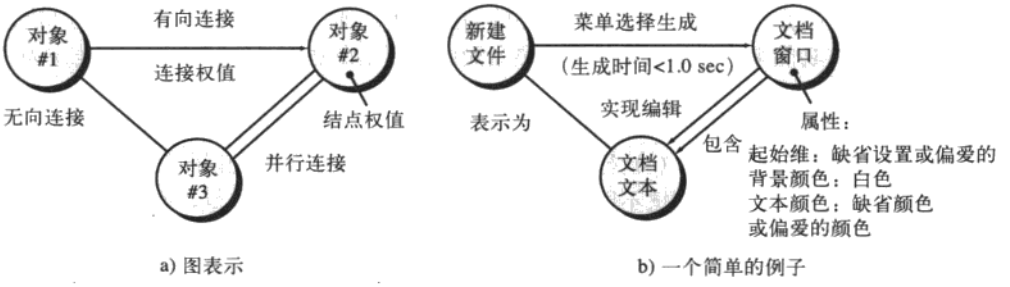
对象#2＝文档窗口

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"的结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

⊖ 这里，我们在最广泛的环境中考虑术语"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。

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：



a) 图表示　　b) 一个简单的例子

图18-8 图符号表示及示例

事务流建模。结点表示事务的步骤（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（例如，“航班信息输入”的后面跟着“确认有效性的处理”）。数据流图（第7章）可用于辅助创建这种类型的图。

有限状态建模。结点表示用户可见的不同软件状态（例如，订票人员处理电话订票时的各个屏幕），连接表示状态间的转换（例如，在“库存有效性检查”期间，“订单信息”会得到验证，之后会输入“客户账单信息”）。状态图（第7章）可用于辅助创建这种图。

数据流建模。结点表示数据对象，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，结点FICA扣缴税款(FTW，FICA tax withheld) 由总工资(GW，gross wages) 利用关系$FTW = 0.62 \times GW$计算出来。

时间建模。结点为程序对象，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。感兴趣的读者参看[Bei95]，可以对其有全面的了解。

18.6.2 等价类划分

在软件过程的早期阶段，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？

等价类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。理想的测试用例可以单独发现一类错误（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。

等价类划分的测试用例设计是基于对输入条件的等价类进行评估。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等价类和两个无效等价类；

2. 若输入条件需要特定的值，则可以定义一个有效等价类和两个无效等价类；

3. 若输入条件指定集合的某个元素，则可以定义一个有效等价类和一个无

效等价类；

4. 若输入条件为布尔值，则可以定义一个有效等价类和一个无效等价类。

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。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。

18.6.3 边界值分析

"测试代码的一种有效方式是在其自然边界处运行它。" —— Brian Kernighan

大量错误发生在输入域的边界处，而不是发生在输入域的"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。

边界值分析是一种测试用例设计技术，是对"等价划分"的补充。BVA不是选择等价类的任何元素，而是在等价类"边缘"上选择测试用例。BVA不是仅仅侧重于输入条件，也从输出域中导出测试用例[Mye79]。

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。

BVA的指导原则在很多方面类似等价划分的原则：

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和b，略大于和略小于a和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，以及略大于和略小于最大值和最小值的值；

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，应该设计测试用例创建输出报告，输出报告可生成所允许的最大（和最小）数目的表项；

4. 若内部程序数据结构有预定义的边界值（例如，表具有100项的定义限制），一定要设计测试用例，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。

18.6.4 正交数组测试

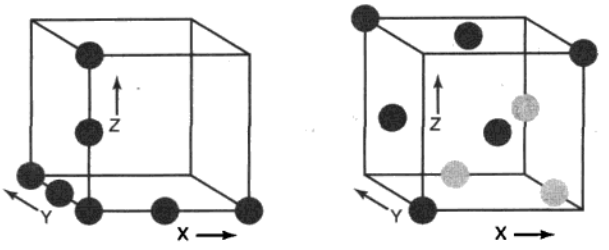
许多应用程序的输入域是相对有限的。也就是说，输入参数的数量不多，且每个参数可取的值有明确的界定。当这些数量非常小时（例如，3个输入参数，取值分别为3个离散值），则有可能考虑每个输入排列，并对所有的输入域进行测试。然而，随着输入值数量的增加及每个数据项的离散值数量的增加，穷举测试是不切实际或不可能的。

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。

正交数组测试使得软件工程师能够设计测试用例，以合理数量的测试用例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3个输入项X、Y和Z的系统。每个输入项有3个不同的离散值。这样可能有$3^3=27$个测试用例。Phadke[Pha97]提出了一种几何观点，来组织与X、Y和Z相关的测试用例。如图18-9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18-9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18-9中右图立方体所示。整个输入域的测试覆盖会更完全。



一次一个输入项　　L9正交数组

图18-9 测试用例的几何视图

为了说明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4个参数P1、P2、P3和P4。其中每个参数取3个不同的值。例如，P1的取值如下：

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]是这样评估这些测试用例的：

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，但还是能够做到的。可以发现所有与数据项排列相关的错误，但所需的工作量较大。

正交数组测试方法使我们可以提供好的测试覆盖，而测试用例比穷举测试少得多。send函数的L9正交数组如图18-10所示。

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18-10 L9正交数组

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，它就是一个单模式错误。在这个例子中，测试1、2和3（图18-10）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。在这个例子中，注意到测试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。这样的错误分离对于修改错误是很重要的。

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，双模式错误表示成对不相容问题或两个测试参数间的有害干扰问题。

多模式错误。（所显示类型的）正交数组仅可以保证单模式和双模式错误的检测。然而，

有些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3种不同类型的静态测试工具——基于代码的测试工具、基于专用测试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。专用测试语言（例如，ATLAS）使软件工程师能够书写详细的测试规格说明，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：[⊖]

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。

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，有助于开发C、C++和Java软件的测试用例设计，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。

e-Test Suite，由Empirix, Inc.（www.empirix.com）开发，拥有测试Web应用系统的完整工具集，包括辅助测试用例设计工具和测试计划工具。

18.7 基于模型的测试

"当你在代码中寻找错误时，很难发现它；当你认为自己的代码没有错误时，就更难发现它。"——Steve McConnell

基于模型的测试（Model-based testing，MBT）是一种黑盒测试技术，它使用需求模型中的信息作为生成测试用例的基础。在很多情况下，基于模型的测试技术使用UML状态图——一种行为模型（第7章）作为测试用例设计的基础[⊖]。MBT技术需要5个步骤：

1. 分析软件的已有行为模型或创建一个行为模型。回忆一下，行为模型指明软件是如何响应外部事件或刺激的。为了创建行为模型，我们需要执行第7章所讨论的步骤：（1）评价所有的用例，使得完全理解系统内的交互顺序；（2）标识驱动交互顺序的事件，并理解这些事件如何与特定的对象相关；（3）为每个用例创建交互顺序；（4）构造系统的UML状态图（如图7-6）；（5）评审行为模型，验证其精确性和一致性。

2. 遍历行为模型，并标明促使软件在状态之间进行转换的输入。输入将触发事件，使转换发生。

3. 评估行为模型，并标注当软件在状态之间转换时所期望的输出。回想一下，每个转换

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

⊖ 当软件需求是用决策表、语法或Markov链表示时，也可以使用基于模型的测试[DAC03]。

都由一个事件触发，作为转换的结果，某些方法会被调用并产生输出。对于步骤2所指定的每个输入（用例）集合，指定所期望的输出，以说明它们在行为模型中的特点。“这种测试的一个基本假设是具有某种机制——测试预言，能够确定测试运行的结果是否正确”[DAC03]。实际上，测试预言建立了确定输出正确性的基础。在大多数情况下，预言就是需求模型，但它也可能是其他文档或应用系统、其他地方记录的数据、或者甚至是某个专家。

4. **运行测试用例。**可以手工执行测试，也可以创建测试脚本并使用测试工具执行测试。

5. **比较实际结果和期望结果，并根据需要进行调整。**

MBT帮助发现软件行为中的错误，因此，它在测试事件驱动的应用系统时也非常有用。

18.8 针对特定环境、体系结构和应用系统的测试

当考虑环境、体系结构和应用系统时，有时还需要专门的测试指导原则和方法。虽然本章前面及19、20章所讨论的测试技术通常可以适用于专门的情形，对于特定环境、体系结构和应用系统，还是需要考虑其他各自独特的需要。

18.8.1 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11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。

由于许多现代的GUI具有相同的观感，因此可以设计一系列标准测试。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。这种基于模型的测试技术已在18.7节讨论过了。

由于存在大量与GUI操作相关的排列，应该使用自动化工具进行GUI测试。在过去几年里，市场上已经出现了大量的GUI测试工具[⊖]。

18.8.2 客户/服务器体系结构测试

“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

WebRef

有用的C/S测试信息和资源可在以下地址找到：www.csst-technologies.com。

客户/服务器（C/S）环境的分布特征、与事务处理相关的性能问题、很多不同硬件平台存在的可能性、网络通信的复杂性、中心（或分布式）数据库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系统的测试要困难得多。事实上，最近的行业研究表明，当开发C/S环境时，测试时间和成本显著增加了。

一般来说，C/S软件测试发生在3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底层网络；（2）以协同方式测试客户软件和有关服务器应用，但不明确检查网络操作；（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应用系统，通常会用到下面的测试方法：

应用功能测试。利用本章前面及19、20章所讨论的方法测试客户应用系统的功能。实际上，以独立的形式测试应用系统，以发现它的操作错误。

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（总体响应时间和数

⊖ 在Web上可以评估的GUI测试工具资源没有几千，也有几百。了解开源工具可以从www. opensourcetesting. org/functional.php开始。

测试实时系统的有效策略是什么？

据吞吐量）。

数据库测试。测试服务器所存储数据的精确性和完整性。检查客户应用系统提交的事务以保证数据正确地存储、更新和检索，同时也对归档进行测试。

事务测试。创建一系列测试，以保证按照需求处理每类事务。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[Mus93]建议开发由C/S使用场景生成的运行剖面（operational profiles）[⊖]。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”。例如，对于特定类型的用户，多少比例的事务会是查询事务、更新事务、命令事务？

为了开发运行剖面，有必要导出与用例类似的场景集合（第5、6章）。每个场景说明人物（who）、位置（where）、事务（what）、原因（why）。即说明用户是谁，系统交互发生在哪里，事务是什么，以及为什么发生。可以使用需求引导技术（第5章）或通过与最终用户进行的不太正式的讨论来导出场景。无论使用哪种方法，其结果应该是一样的。每个场景都应该提供服务于特定用户所需要的系统功能、所需功能的顺序、期望的响应时间以及使用每个功能的频率。然后将（所有用户的）这些数据组合起来生成运行剖面。一般来说，根据执行功能的使用频率和关键程度，将测试工作量和运行的测试用例的数量分配给每个使用场景。

18.8.3 文档测试和帮助设施测试

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，测试应该扩展到软件配置的第3个元素—文档。

文档中的错误与数据或源代码中的错误一样，会影响程序的验收。完全按照用户指南或在线帮助进行操作，但得到的结果或行为却与文档的描述不符，没有什么比这种情况更让人沮丧了。因此，文档测试应该是所有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。第一阶段为技术评审（第15章），检查文档编辑的清晰性；第二阶段是现场测试（live test），结合实际程序使用文档。

令人感到意外的是，对文档的现场测试竟可以采用与前面讨论的许多黑盒测试方法相似的技术，包括：基于图的测试可用于描述程序的使用；等价类划分和边界值分析方法可用于定义各种输入类和相关的交互操作；MBT则可用以确保文档规定的行为和实际行为的吻合。因而程序的用法可以贯穿全部文档而得到追踪。

INFO

文档测试

在测试文档和（或）帮助设施时，应该回答下列问题：

- *该文档准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否准确？*
- *实例准确吗？*
- *术语、菜单描述以及系统响应与实际程序一致吗？*
- *在文档中能够比较容易地找到指导吗？*

⊖ 应该注意到，运行剖面可用于所有类型的系统体系结构的测试中，而不仅仅是C/S体系结构。

- 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否健壮、准确和完整?
- 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?
- 如果提供超文本链接，链接是否准确和完整?
- 如果提供超文本链接，导航设计是否适合信息获取?

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。

18.8.4 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（即中断处理）、数据的定时以及处理数据的任务（进程）的并行性。在许多情况下，实时系统在一种状态下提供的测试数据可以正常处理，而在另一种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（处于copying状态）时，接收操作员的中断（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“卡纸”状态时，则会显示诊断代码，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。这种故障很难实时模拟。

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4个步骤的策略：

? 测试实时系统的有效策略是什么?

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。使用类似等价划分的技术（18.6.2节），对事件（如，中断、控制信号）进行分类测试。例如，复印机的事件可能是用户中断（如，重置计数器）、机械中断（如，卡纸）、系统中断（如，碳粉低）及失效模式（如，滚筒过热）。这些事件的每一个都要单独测试，并检查可执行系统的行为，以检测与这些事件有关的处理错误。可以对（在分析活动期间开发的）系统模型的行为与可执行软件的行为是否相符进行比较。一旦每类事件都已经测试，事件就以随机的顺序和频率呈现给系统，对软件的行为进行检查，以检测行为错误。

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。用不同的数据速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以发现软件/硬件接口处的错误。多数实时系统都能处理中断，因此，测试布尔事件的处理尤其重要。利用状态图（第7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，作为中断处理的一部分、用来传输信息的全局数据区域也应该测试，以评估产生副作用的可能性。

18.9 软件测试模式

WebRef

软件测试模式目录可在www.rbsc.com/pages/TestPatternList.htm找到。

模式作为描述对特定设计问题解决方案的一种机制，其使用已经在第12章讨论过了。但模式也可以用于提出其他软件工程状况——此处为软件测试。测试模式描述常见的测试问题和解决方案，可以辅助软件工程师处理这些问题。

测试模式不仅在测试活动开始时为软件工程师提供有用的指导，而且也提供另外3个益处，Marick[Mar02]描述这3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。”

KEY POINT

测试模式有助于软件团队对测试进行更有效的交流，并对采用特定测试方法的影响力有更好的理解。

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作为一种进化活动，每次迭代都产生更完整的测试用例，测试模式就达到了预期的目的。

WebRef

描述测试组织、效率、策略以及问题求解的模式可在www.testing.com/test-patterns/ patterns/找到。

测试模式可以采用与设计模式（第12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3种测试模式（仅以摘要的形式给出）提供了有代表性的例子：

模式名称：**结对测试**

摘要：一种面向过程的模式，**结对测试**描述了一种与结对编程（第3章）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测试，可以应用于单元测试、集成测试或确认测试活动中。

模式名称：**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构件暴露任何接口的类）。**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构件可见）的特定测试”[Lan01]。

模式名称：**场景测试**

摘要：一旦已经执行了单元测试与集成测试，就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测试软件的技术。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。对于这个重要主题的其他信息，有兴趣的读者可以参看[Bin99]和[Mar02]。

18.10 小结

测试用例设计的主要目标是设计最有可能发现软件错误的测试用例集。为达到这个目标，可采用两种不同的测试用例设计技术：白盒测试和黑盒测试。

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，循环测试补充其他白盒测试技术，检查不同复杂度的循环。

Hetzel[Het84]将白盒测试描述为“小型测试”。他的意思是，本章所考虑的白盒测试一般应用于小的程序构件（例如，模块或一小组模块）。而黑盒测试放宽了测试的焦点，可以将其称为“大型测试”。

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在可接受的限度内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。基于模型的测试使用需求模型的元素测试应用系统的行为。

特定的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。

有经验的软件开发人员经常说：“测试永无止境，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，因此可以在“客户的测试”开始之前，发现和改正尽可能多的错误。

习题与思考题

18.1 Myers[Mye79]用以下程序作为对测试能力的自我评估：某程序读入3个整数值，这3个整数值表示三角形的3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。

18.2 设计并实现习题18.1描述的程序（适当时使用错误处理）。从该程序中导出流图并用基本路径测试方法设计测试，以保证程序中的所有语句都被测试到。执行测试用例并显示结果。

18.3 你能够想出18.1.1节中没有讨论的其他测试目标吗？

18.4 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执行所有的语句。

18.5 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。

18.6 阅读Beizer[Bei95]或相关的基于Web的资源（例如，www.laynetworks.com/ Discrete%20 Mathematics_1g.htm），并确定如何扩展习题18.5所开发的程序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。

18.7 设计一个自动化测试工具，使其能够识别循环并按照18.5.3节中的方法分类。

18.8 扩展习题18.7中描述的工具，为曾经遇到的每个循环类生成测试用例。与测试人员交互地完成这个功能是有必要的。

18.9 至少给出3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。再至少给出3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。

18.10 穷举测试（即便对非常小的程序）是否能够保证程序100%正确？

18.11 测试你经常使用的某个应用系统的用户手册（或帮助设施）。在文档中至少找到一个错误。

推荐读物与阅读信息

实际上，所有软件测试方面的书籍都同时考虑测试策略和测试技术。因此，第17章的推荐读物同样适用于本章。有许多讨论测试原理、概念、策略和方法的书籍，下面的书籍只是其中的一小部分：Everett和Raymond（《Software Testing》，Wiley-IEEE Computer Society Press，2007）、Black（《Pragmatic Software Testing》，Wiley，2007）、Spiller和他的同事（《Software Testing Process：Test Management》，Rocky Nook，2007）、Perry（《Effective Methods for Software Testing》，3rd ed.，Wiley，2005）、Lewis（《Software Testing and Continuous Quality Improvement》，2nd ed.，Auerbach，2004）、Loveland和他的同事（《Software Testing Techniques》，Charles River Media，2004）、Burnstein（《Practical Software Testing》，Springer，2003）、Dustin（《Effective Software Testing》，Addison-Wesley，2002）、Craig和Kaskiel（《Systematic Software Testing》，Artech House，2002）、Tamres（《Introducing Software Testing》，Addison-Wesley，2002）以及Whittaker（《How to Break Software》，Addison-Wesley，2002）。

Myers[Mye79]经典文稿的第2版（《The Art of Software Testing，2nd ed.，Wiley，2004》）由Myers及其同事编写，非常详细地覆盖了测试用例的设计技术。Pezze和Young（《Software Testing and Analysis》，Wiley，2007）、Perry（《Effective Methods for Software Testing》，3rd ed.，Wiley，2006）、Copeland（《A Practitioner's Guide to Software Test Design》，Artech，2003）、Hutcheson（《Software Testing Fundamentals》，Wiley，2003）、Jorgensen（《Software Testing：A Craftsman's Approach》，2nd ed.，CRC Press，2002）都提供了测试用例设计方法和技术的有用介绍。Beizer[Bei90]经典文章全面覆盖了白盒测试技术，引入了数学级别上的严格性，这在其他测试技术中一般是不具备的。他后来的书籍[Bei95]对重要方法作了简明介绍。

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。Li和Wu（《Effective Software Test Automation》，Sybex，2004）、Mosely和Posey（《Just Enough Software Test Automation》，Prentice-Hall，2002）、Dustin、Rashka和Poston（《Automated Software Testing：Introduction，Management，and Performance》，Addison-Wesley，1999）、Graham和她的同事（《Software Test Automation》，Addison-Wesley，1999）以及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。Nquyen和他的同事（《Global Software Test Automation》，Happy About Press，2006）介绍了测试自动化的执行视图。

Thomas和他的同事（《Java Testing Patterns》，Wiley，2004）以及Binder[Bin99]描述了测试模式，覆盖了方法测试、类/簇测试、子系统测试、可复用构件测试、框架测试、系统测试、测试自动化及特定数据库测试。

从网上可以获得大量的有关测试用例设计方法的信息源。有关测试技术的最新WWW资源列表可在SEPA Web站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第19章

Software Engineering：A Practitioner's Approach (Seventh Edition)

测试面向对象的应用系统

要点浏览

概念：面向对象（OO）软件的体系结构是包含协作类的一系列分层的子系统。这些系统的每个元素（子系统和类）都执行帮助完成系统需求的功能。有必要在各种不同的层次上测试面向对象系统，尽力发现当类之间协作以及子系统穿越体系结构层通信时可能发生的错误。

人员：面向对象测试由软件工程师和测试专家执行。

重要性：在将程序交付给客户之前，必须运行程序，试图去除所有的错误，使得客户不会经历由质量糟糕的产品所带来的挫败。为了发现尽可能多的错误，必须进行系统的测试，并且必须使用严格的技术设计测试用例。

步骤：面向对象测试在策略上类似传统系统的测试，但在战术上是不同的。由于面向对象分析和设计模型在结构和内容上类似于最终的面向对象程序。"测试"开始于对这些模型的评审。一旦代码已经生成，面向对象测试开始进行"小规模"的类测试。设计一系列测试检查类操作及一个类与其他类协作时是否存在错误。当将类集成起来构成子系统时，应用基于线程的测试、基于使用的测试、簇测试以及基于故障的测试方法彻底检查协作类。最后，运用用例（作为需求模型的一部分开发）发现软件确认级的错误。

工作产品：设计并文档化一组测试用例来检查类、协作和行为。定义期望的结果，并记录实际结果。

质量保证措施：测试时改变观点，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。

关键概念
类测试
簇测试
基于故障的测试
多类测试
划分测试
随机测试
基于场景的测试
基于线程的测试
基于使用的测试

在第18章已经谈到：简单地说，测试的目标就是在可行的时间期限内，以可行的工作量发现最大可能数量的错误。虽然这个基本目标对于面向对象软件没有改变，但面向对象程序的本质改变了测试策略和测试战术。

可以断定，由于可复用类库规模的增大，更多的复用会缓解对面向对象系统进行繁重测试的需求。确切地说，相反的情况的确是存在的。Binder[Bin94b]在讨论这种情况时说道：

> 每次复用都是一种新的使用环境，重新测试需要谨慎。为了在面向对象系统中获得高可靠性，似乎需要更多的测试，而不是更少的测试。

为了充分测试面向对象的系统，必须做3件事情：（1）对测试的定义进行扩展，使其包括应用于面向对象分析和设计模型的错误发现技术；（2）单元测试和集成测试策略必须彻底改变；（3）测试用例设计必须考虑面向对象软件的独特性质。

19.1 扩展测试的视野

面向对象软件的构造开始于需求（分析）和设计模型的创建[⊖]。由于面向对象软件工程模

⊖ 分析建模和设计建模技术在本书第二部分介绍。基本的面向对象概念在附录2中介绍。

式的进化特性，这些模型开始于系统需求的不太正式的表示，并进化到更详细的类模型、类关系、系统设计和分配以及对象设计（通过消息传递合并对象连接模型）。在每一个阶段，都要对模型进行“测试”，尽量在错误传播到下一轮迭代之前发现错误。

可以肯定，面向对象分析和设计模型的评审非常有用，因为相同的语义结构（例如，类、属性、操作、消息）出现在分析、设计和代码层次。因此，在分析期间所发现的类属性的定义问题会防止副作用的发生。如果问题直到设计或编码阶段（或者是分析的下一轮迭代）还没有发现，副作用就会发生。

尽管面向对象分析和设计模型的评审是测试面向对象应用系统不可分割的一部分，但要认识到这是不够的，还要实施可运行的测试。

例如，在分析的第一轮迭代中，考虑定义了很多属性的一个类。一个无关的属性被扩展到类中（由于对问题域的错误理解），然后指定了两个操作来处理此属性。分析模型进行了评审，领域专家指出了这个问题。在这个阶段去除无关的属性，可以在分析阶段避免下面的问题和不必要的工作量。

1. 可能会生成特殊的子类，以适应不必要的属性或例外。去除无关的属性后，与创建不必要的子类相关的工作可以避免。

2. 类定义的错误解释可能导致不正确或多余的类关系。

3. 为了适应无关的属性，系统的行为或类可能被赋予不适当的特性。

如果问题没有在分析期间被发现并进一步传播，在设计期间会发生以下问题（早期的评审可以避免这些问题的发生）。

1. 在系统设计期间，可能会发生将类错误地分配给子系统和（或）任务的情况。

2. 可能会扩展不必要的设计工作，为涉及无关属性的操作创建过程设计。

3. 消息模型可能不正确（因为会为无关的操作设计消息）。

“我们使用的工具对我们的思考习惯具有深远的影响，因此，也对我们的思考能力具有深远的影响。”——Edsger Dijkstra

如果问题没有在设计期间检测出来，并传递到编码活动中，将增加可观的工作量生成代码，实现不必要的属性、两个不必要的操作、驱动对象间通信的消息以及很多其他相关的问题。另外，类的测试会消耗更多不必要的时间。一旦最终发现了这个问题，一定对系统执行修改，由变更所引起的潜在副作用将永远存在。

在开发的后期，面向对象分析（OOA）和面向对象设计（OOD）模型提供了有关系统结构和行为的实质性信息。因此，在代码生成之前，需要对这些模型进行严格的评审。

应该在模型的语法、语义和语用方面对所有的面向对象模型进行正确性、完整性和一致性测试（在这里，术语测试包括技术评审）[Lin94a]。

19.2 测试OOA和OOD模型

不能在传统意义上对分析和设计模型进行测试，因为这些模型是不能运行的。然而，可以使用技术评审（第15章）检查模型的正确性和一致性。

19.2.1 OOA和OOD模型的正确性

用于表示分析和设计模型的符号和语法是与为项目所选择的特定分析和设计方法连接在一起的。由于语法的正确性是基于符号表示的正确使用来判断的，必须对每个模型进行评审以确保维持了正确的建模习惯。

在分析和设计期间，可以根据模型是否符合真实世界的问题域来评估模型的语义正确性。如果模型准确地反映了真实世界（详细程度与模型被评审的开发阶段相适应），则在语义上是

正确的。实际上，为了确定模型是否反映了真实世界的需求，应该将其介绍给问题领域的专家，由专家检查类定义和层次中遗漏和不清楚的地方。要对类关系（实例连接）进行评估，确定这些关系是否准确地反映了真实世界的对象连接[⊖]。

19.2.2 面向对象模型的一致性

面向对象模型的一致性可以通过这样的方法来判断："考虑模型中实体之间的关系。不一致的分析模型或设计模型在一部分中的表示没有正确地反映到模型的其他部分"[McG94]。

为了评估一致性，应该检查每个类及与其他类的连接。可以使用类-责任-协作（class-responsibility-collaboration，CRC）模型或对象-关系图来辅助此活动。如在第6章所学到的，CRC模型由CRC索引卡片组成。每张CRC卡片都列出了类的名称、责任（操作）和协作者（接收其消息的其他类及完成其责任所依赖的其他类）。协作意味着面向对象系统的类之间的一系列关系（即连接）。对象关系模型提供了类之间连接的图形表示。这些信息都可以从分析模型（第6章和第7章）中获得。

推荐使用下面的步骤对类模型进行评估[McG94]：

1. **检查CRC模型和对象-关系模型。**对这两个模型做交叉检查，确保需求模型所蕴含的所有协作都正确地反映在了这两个模型中。

2. **检查每一张CRC索引卡片的描述以确定委托责任是定义协作者的一部分。**例如，考虑为销售积分结账系统定义的类，称为CreditSale，这个类的CRC索引卡片如图19-1所示。

对于这组类和协作，如果将责任（例如，读信用卡）委托给已命名的协作者（CreditCard），看看此协作者是否完成了这项责任。也就是说，类CreditCard是否具有读卡操作？在此实例中，回答是肯定的。遍历对象-关系模型，确保所有此类连接都是有效的。

类的名称：CreditSale	
类的类型：交易事件	
类的特性	nontangible, atomic, sequential, permanent, guarded
责任：	协作者：
读信用卡	信用卡
取得授权	信用权利
显示购物金额	产品票
	销售总账
	审计文件
生成账单	账单

图19-1 用于评审的CRC索引卡片实例

3. **反转连接，确保每个提供服务的协作者都从合理的地方收到请求。**例如，如果CreditCard类收到了来自CreditSale类的请求purchase amount，就有问题了。CreditCard不知道购物金额是多少。

4. **使用步骤3中反转后的连接，确定是否真正需要其他类，或者责任在类之间的组织是否合适。**

⊖ 对于面向对象系统，在对照真实世界的使用场景追踪分析和设计模型方面，用例是非常有价值的。

5. 确定是否可以将广泛请求的多个责任组合为一个责任。例如，读信用卡和取得授权在每一种情形下都会发生，可以将这两个责任组合为验证信用请求（validate credit request）责任，此责任包括取得信用卡号和取得授权。

可以将步骤1到步骤5反复应用到每个类及需求模型的每一次评估中。

一旦创建了设计模型（第9章到第11章），就可以进行系统设计和对象设计的评审了。系统设计描述总体的产品体系结构、组成产品的子系统、将子系统分配给处理器的方式、将类分配给子系统的方式以及用户界面的设计。对象模型描述每个类的细节以及实现类之间的协作所必需的消息传送活动。

系统设计评审是这样进行的：检查面向对象分析期间所开发的对象-行为模型，并将所需要的系统行为映射到为完成此行为而设计的子系统上。在系统行为的范畴内也要对并发和任务分配进行评审。对系统的行为状态进行评估以确定并发行为。使用用例进行用户界面设计。

对照对象-关系网检查对象模型，确保所有的设计对象包括必要的属性和操作，以实现为每个CRC索引卡片所定义的协作。另外，要对操作细节的详细规格说明（即实现操作的算法）进行评审。

19.3 面向对象测试策略

如在第18章讲到的[⊖]，经典的软件测试策略从“小范围”开始，并逐步过渡到“软件整体”。用软件测试的行话来说（第18章），就是先从单元测试开始，然后过渡到集成测试，并以确认测试和系统测试结束。在传统的应用系统中，单元测试关注最小的可编译程序单元——子程序（例如，构件、模块、子程序、程序）。一旦完成了一个单元的单独测试，就将其集成到程序结构中，并进行一系列的回归测试，以发现模块的接口错误及由于加入新模块所引发的副作用。最后，将系统作为一个整体进行测试，确保发现需求方面的错误。

19.3.1 面向对象环境中的单元测试

KEY POINT
在OO软件中，最小的可测试“单元”是类，类测试是由封装在类中的操作和类的状态行为驱动的。

当考虑面向对象软件时，单元的概念发生了变化。封装是类和对象定义的驱动力，也就是说，每个类和类的每个实例（对象）包装了属性（数据）和操纵这些数据的操作（也称为方法或服务）。最小的可测试单元是封装了的类，而不是单独的模块。由于一个类可以包括很多不同的操作，并且一个特定的操作又可以是很多不同类的一部分，因此，单元测试的含义发生了巨大的变化。

我们已经不可能再独立地测试单一的操作了（独立地测试单一的操作是单元测试的传统观点），而是要作为类的一部分进行测试。例如，考虑在一个类层次中，为超类定义了操作*X*()，并且很多子类继承了此操作。每个子类都使用操作*X*()，但是此操作是在为每个子类所定义的私有属性和操作的环境中应用的。由于使用操作*X*()的环境具有微妙的差异，因此，有必要在每个子类的环境中测试操作*X*()。这就意味着在真空中测试操作*X*()（传统的单元测试方法）在面向对象的环境中是无效的。

面向对象软件的类测试等同于传统软件的单元测试[⊖]。面向对象软件的类测试与传统软件的单元测试是不同的，传统软件的单元测试倾向于关注模块的算法细节和流经模块接口的数据，而面向对象软件的类测试由封装在类中的操作和类的状态行为驱动。

⊖ 应为第17章。——译者注

⊖ 面向对象类的测试用例设计方法在19.4节到19.6节讨论。

19.3.2 面向对象环境中的集成测试

由于面向对象软件不具有层次控制结构，因此传统的自顶向下和自底向上的集成策略没有意义。另外，由于“组成类的构件之间的直接和非直接的交互”[Ber93]，每次将一个操作集成到类中通常是不可能的。

KEY POINT
OO软件的集成测试是对响应一个给定事件所需要的一组类进行测试。

面向对象系统的集成测试有两种不同的策略[Bin94a]。第一种集成策略是基于线程的测试，将响应系统的一个输入或一个事件所需要的一组类集成到一起。每个线程单独集成和测试，并应用回归测试确保不产生副作用。第二种集成策略是基于使用的测试，通过测试那些很少使用服务器类的类（称为独立类）开始系统的构造。测试完独立类之后，测试使用独立类的下一层类（称为依赖类）。按照这样的顺序逐层测试依赖类，直到整个系统构造完成。与传统集成不同，在可能的情况下，这种策略避免了作为替换操作的驱动模块和桩模块的使用（第18章）。

簇测试[McG94]是面向对象软件集成测试中的一个步骤。通过设计试图发现协作错误的测试用例，对一簇协作类（通过检查CRC和对象-关系模型来确定）进行测试。

19.3.3 面向对象环境中的确认测试

在确认级或系统级，类连接的细节消失了。如传统的确认方法一样，面向对象软件的确认关注用户可见的动作和用户可以辨别的来自系统的输出。为了辅助确认测试的导出，测试人员应该拟定出用例（第5章和第6章），用例是需求模型的一部分，提供了最有可能发现用户交互需求方面错误的场景。

传统的黑盒测试方法（第18章）可用于驱动确认测试。另外，测试人员可以选择从对象-行为模型导出测试用例，也可以从创建的事件流图（OOA的一部分）导出测试用例。

19.4 面向对象测试方法

“我将测试人员看成是项目的护卫。我们保护开发者的侧面，使其免于失败，而测试人员则关注于使项目成功。”
——James Bach

面向对象体系结构导致封装了协作类的一系列分层子系统的产生。每个系统成分（子系统和类）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穿越体系结构层通信时可能出现这些错误。

面向对象软件的测试用例设计方法还在不断改进，然而，对于面向对象测试用例的设计，Berard已经提出了总体方法[Ber93]：

1. 每个测试用例都应该唯一地标识，并明确地与被测试的类相关联。
2. 应该叙述测试的目的。
3. 应该为每个测试开发测试步骤，并包括以下内容：

a. 将要测试的类的指定状态列表
b. 作为测试结果要进行检查的消息和操作列表
c. 对类进行测试时可能发生的异常列表
d. 外部条件列表（即软件外部环境的变更，为了正确地进行测试，这种环境必须存在）
e. 有助于理解或实现测试的补充信息

面向对象测试与传统的测试用例设计是不同的，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，而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。

19.4.1 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重要设计概念，但它可能成为测试的一个小障碍。如Binder [Bin94a]所述："测试需要报告对象的具体状态和抽象状态"。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，否则，可能很难获得一个对象的状态快照。

继承也为测试用例设计提出了额外的挑战。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[⊖]使测试进一步复杂化[Bin94a]。若将从超类派生的子类实例用于相同的问题域，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集。

19.4.2 传统测试用例设计方法的可应用性

第18章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一个操作中的每条语句都测试到。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投入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。如我们在第18章提到的，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。

19.4.3 基于故障的测试[⊖]

KEY POINT
基于故障的测试策略是假设一组似乎可能出现的故障，然后导出测试去证明每个假设。

在面向对象系统中，基于故障测试的目标是设计测试，所设计的测试最有可能发现似乎可能出现的故障（以下称为似然故障）。由于产品或系统必须符合客户需求，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。测试人员查找似然故障（即系统的实现有可能产生错误的方面）。为了确定这些故障是否存在，需要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理解似然故障。若在面向对象系统中真正的故障被理解为"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，若分析模型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。

? 在操作调用和消息连接中会遇到哪些类型的故障？

集成测试寻找的是操作调用或信息连接中的似然错误。在这种环境下，可以发现3种错误：非预期的结果、使用了错误的操作/消息，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，同样也适用于操作。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。

⊖ 应非常小心使用的一个面向对象概念。

⊖ 19.4.3节到19.4.6节是从Brain Marick贴在因特网新闻组comp.testing上的文章摘录的，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，19.4.3节到19.4.6节讨论的技术也适用于传统软件。

19.4.4 测试用例与类层次

尽管已经对基类进行了彻底的测试，仍然要对由它派生的所有类进行测试。

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重定义了redefined()以用于某个局部环境中。毫无疑问，必须对Derived::redefined()进行测试，因为它表示的是新设计和新代码。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()可能会误用这个新行为，因此，尽管设计与代码没有发生变化，还是需要对它进行新的测试。然而，重要的是要注意，只需要执行Derived::inherited()所有测试的一个子集。若inherited()的部分设计和代码不依赖于redefined()（即，不调用它，也不间接调用它），则不需要重新测试派生类中的代码。

Base::redefined()和Derived::redefined()是具有不同规格说明和实现的两个不同操作。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。但操作有可能是类似的，它们的测试需求集将重叠。面向对象设计得越好，重叠就越多。仅需要针对Base::redefined()测试不能满足的那些Derived::redefined()需求生成新的测试。

总而言之，Base::redefined()测试可应用于类Derived的对象。测试输入可能同时适用于基类和派生类，但是，预期的结果在派生类中可能有所不同。

19.4.5 基于场景的测试设计

基于故障的测试忽略了两种主要类型的错误：（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。当一个子系统的行为创建的环境（例如，事件、数据流）使另一个子系统失效时，则出现了与子系统交互相关的错误。

基于场景的测试将发现任何角色与软件进行交互时出现的错误。

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）。

作为一个例子，通过审查下面的用例，考虑设计文本编辑器的基于场景的测试：

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错误。该用例描述了当这种情况发生时出现的事件序列：

“如果你盼望一个程序能正常运行，你就更倾向于看到正常运行的程序，这时你会忽视可能出现的故障。”——Cem Kaner等

1. 打印整个文档；
2. 翻动文档，修改某些页；
3. 打印修改的每一页；
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）打印多页的方法。至于测试，需要在打印后测试编辑（以及编辑后测试打印）。因此，测试人员需要设计测试，以发现由打印功能引起的编辑功能的错误。也就是说，这样的错误说明两个软件功能

不是完全独立的。

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。

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，就会使上次改正过的页面再打印一次。这样，根据编辑器，正确的场景应该如下：

尽管基于场景的测试具有优点，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会获得较高的回报。

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，单击以打印整个文档；
4. 单击“打印”按钮；
5. 关闭文档。

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。生气的用户会报告规格说明错误。

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。测试人员则必须应对可能的反应：“这就是它工作的本来方式！”

19.4.6 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，无论是什么界面，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，在使用面向命令界面的传统系统中，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）。在基于对象的界面中，测试人员可以使用所有的对象列表作为测试检查表。

表层结构的测试类似于黑盒测试，深层结构的测试类似于白盒测试。

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界面，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。

“不要为错误感到羞愧，从而使错误变成遗憾。”——Confucius

深层结构是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的依赖关系、行为和通信机制。

需求模型和设计模型可用来作为深层结构测试的基础。例如，UML协作图或部署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：我们是否已经捕获了（作为测试的）某些任务来测试协作图中记录的协作？若没有，为什么没有？

19.5 类级可应用的测试方法

随机测试可能的排列数可以变得相当大。与正交数组测试相类似的策略可以用于提高测试的有效性。

“小范围”测试侧重于单个类及该类封装的方法。面向对象测试期间，随机测试和分割是用于检查类的测试方法。

19.5.1 面向对象类的随机测试

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，可以在这个序列中发生大量其他行为：

Open●setup●deposit●[deposit|withdraw|balance|summarize|creditLimit]^n●withdraw●close

可以随机产生一些不同的操作序列，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，以检查不同类实例的生命历史。

SAFEHOME

类测试

[场景] Shakira的工作间。

[人物] Jamie与Shakira，SafeHome软件工程团队的成员，负责安全功能的测试用例设计。

[对话]

Shakira：我已经为Detector类（图10-4）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4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，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。

（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#2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4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。

19.5.2 类级的划分测试

在类层次上，什么测试选项可供使用？

与传统软件的等价划分（第18章）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分类。但划分类别是如何得到的呢？

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对类操作进行分类。对于类Account，属性balance和creditLimit可用于定义划分。操作可分为3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序列。

基于类别划分就是根据每个操作所完成的一般功能对类操作进行分类。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）及终止操作（close）。

19.6 类间测试用例设计

"对于面向对象的开发，定义单元测试和集成测试范围的边界是不同的。在这个过程中，可以在很多点上设计和检查测试。因此，'设计一点儿，编码一点儿'变成了'设计一点儿，编码一点儿，测试一点儿'。"
——Robert Binder

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19.5节中讨论的银行例子，让它包括图19-2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、基于场景测试及行为测试来完成。

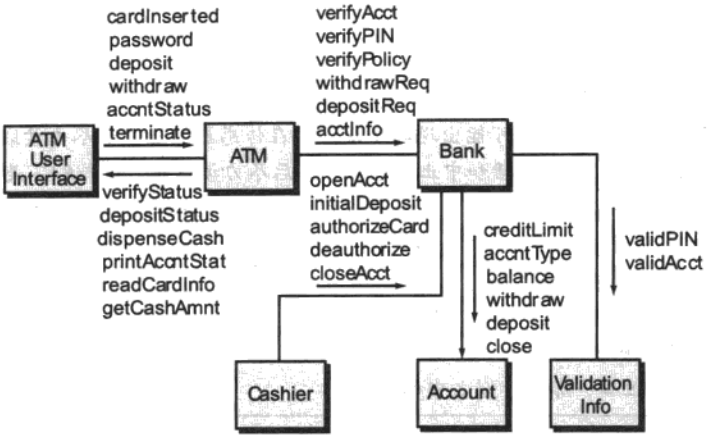


图19-2 银行应用的类协作图（改自[Kir94]）

19.6.1 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：

1. 对每个客户类，使用类操作列表来生成一系列随机测试序列。这些操作将向其他服务类发送消息；

2. 对生成的每个消息，确定协作类和服务对象中的相应操作；

3. 对服务对象中的每个操作（已被来自客户对象的消息调用），确定它传送的消息；

4. 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。

为便于说明[Kir94]，考虑Bank类相对于ATM类的操作序列（图19-2）：

verifyAcct●VerifyPIN●[[verifyPolicy●withdrawReq]|depositReq|acctInfoREQ]^n

Bank类的一个随机测试用例可以是：

测试用例r_3: verifyAcct●VerifyPIN●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。为了执行verifyAcct()与verifyPIN()，Bank类必须与ValidationInfo类协作。为了执行depositReq()，Bank类必须与Account类协作。因此，检查这些协作的新测试用例为：

测试用例r_4：verifyAcct[Bank:validAcctValidationInfo] ●verifyPIN

[Bank:validPinValidationInfo] ● depositReq[Bank:depositaccount]

多个类的划分测试方法与单个类的划分测试方法类似，单个类的划分测试方法如在19.5.2节讨论的那样。然而，可以对测试序列进行扩展，以包括那些通过发送给协作类的消息而激活的操作。另一种划分测试方法基于特殊类的接口。参看图19-2，Bank类从ATM类和Cashier类接收消息，因此，可以通过将Bank类中的操作划分为服务于ATM类的操作和服务于Cashier类的操作对其进行测试。基于状态的划分（19.5.2节）可用于进一步细化上述划分。

19.6.2 从行为模型导出的测试

在第7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19-3[Kir94]给出了前面讨论的Account类的状态图。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Account类分别向“Nonworking acct”状态和“Dead acct”状态发生转换。

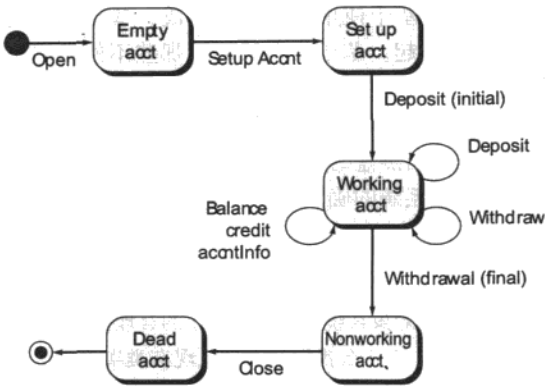


图19-3 Account类的状态转换图（改自[Kir94]）

将要设计的测试应该覆盖所有的状态，也就是说，操作序列应该使Account类能够向所有可允许的状态转换：

测试用例 s_1: open●setupAccnt●deposit(initial) ●withdraw(final) ●close

应该注意到，这个序列与19.5.2节所讨论的最小测试序列相同。下面将其他测试序列加入最小测试序列中：

测试用例 s_2: open●setupAccnt●deposit(initial) ●deposit●balance●credit●withdraw(final) ●close

测试用例 s_3: open●setupAccnt●deposit(initial) ●deposit●withdraw●acctInfo●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。在该类的行为与一个或多个类产生协作的情况下，可以用多个状态图来追踪系统的行为流。

可以通过“广度优先”[McG94]的方式来遍历状态模型。在这里，广度优先意味着一个测试用例检查单个转换，当测试新的转换时，仅使用前面已经测试过的转换。

考虑银行系统中的一个CreditCard对象。CreditCard对象的初始状态为undefined（即未提供信用卡号）。在销售过程中一旦读取信用卡，对象就进入了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，它处于submitted状态，当接收到授权时，它处于approved状态。可以通过设计使转换发生的测试用例来测试CreditCard对象从一个状态到另一个状态的转换。对这种测试类型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转换，从而违反了广度优先准则。

19.7 小结

面向对象测试的总体目标是以最少的工作量发现最多的错误，这与传统软件的测试目标是一样的。但是，面向对象测试的策略和战术与传统软件有很大差异。测试的视角扩展到了需求模型和设计模型，另外，测试的重点从过程构件（模块）移向了类。

由于面向对象需求模型、设计模型和最终的源代码在语义上是有联系的，在建模活动期间，测试（以技术评审的形式）就开始了。因此，可以将CRC、对象关系模型、对象行为模型的评审看成是第一阶段的测试。

一旦有了代码，就对每个类进行单元测试。可以使用多种方法对类测试进行设计：基于故障测试、随机测试和划分测试。每种方法都检查类中封装的操作。设计测试序列以保证对相关的操作进行了检查。通过检查类的状态（由属性值表示）以确定是否存在错误。

集成测试可以通过基于线程或基于使用的策略来完成。基于线程的测试将响应一个输入或事件而相互协作的一组类集成在一起。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。

面向对象系统的确认测试是面向黑盒的测试，可以应用传统软件中所讨论的黑盒方法来完成。然而，基于场景的测试在面向对象系统的确认中占优势，使用例成为确认测试的主要驱动者。

习题与思考题

19.1 用自己的话，描述为什么在面向对象系统中，类是最小的合理测试单元。

19.2 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？

19.3 为什么“测试”应该从面向对象分析和设计开始？

19.4 为SafeHome导出一组CRC索引卡片，按照19.2.2节讲述的步骤确定是否存在不一致性。

19.5 基于线程和基于使用的集成测试策略有什么不同？簇测试如何适应？

19.6 将随机测试和划分方法运用到设计SafeHome系统时定义的3个类。产生展示操作调用序列的测试用例。

19.7 运用多类测试及从SafeHome设计的行为模型中生成的测试。

19.8 运用随机测试、划分方法、多类测试及19.5节和19.6节所描述的银行应用的行为模型导出的测试，再另外生成4个测试。

推荐读物与阅读信息

第17章和第18章的推荐读物与阅读信息部分所记录的很多测试方面的书都在一定程度上讨论了面向对象系统的测试。Schach（《Object-Oriented and Classical Software Engineering》，McGraw-Hill, 6th ed., 2004）在更广的软件工程实践环境中考虑面向对象测试。Sykes和McGregor（《Practical Guide for Testing Object-Oriented Software》，Addison-Wesley, 2001）、Bashir和Goel（《Testing Object-Oriented Software》，Springer, 2000）、Binder（《Testing Object-Oriented Systems》，Addison-Wesley, 1999）以及Kung和他的同事（《Testing Object-Oriented Software》，Wiley-IEEE Computer Society Press, 1998）都非常详细地描述了面向对象测试。

从网上可以获得大量面向对象测试方法的信息源。最新的有关测试技术的WWW资源列表可在SEPA站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

测试Web应用系统

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，要同时应用包括评审及运行测试的测试策略。

人员：所有参加WebApp测试的Web工程师和其他项目利益相关者（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上线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，之后进行技术及内部结构方面的测试。要进行7个步骤的测试：内容测试、界面测试、导航测试、构件测试、配置测试、性能测试及安全测试。

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：虽然你从来都不能保证你已经完成了需要的每一项测试，但是，你能够肯定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
兼容性测试
构件级测试
配置测试
内容测试
数据库测试
质量维度
界面测试
负载测试
导航测试
性能测试
策划
安全性测试
策略
压力测试
可用性测试

WebApp项目普遍都很紧迫。利益相关者由于担心来自其他WebApp的竞争，迫于客户的要求，并担心他们将失去市场，因而迫使WebApp仓促上线。其结果是，在Web开发过程中，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需求模型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15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

测试是为了发现（并最终改正）错误而运行软件的过程。这个首次在第17章介绍的基本原理对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（位于很多不同的设备上）、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[⊖]。在此讨论中，我们考虑在讨论Web开发的测试工作中都特别关注的质量维度，同时，我们也考虑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[Mil00a]：

我们如何在WebApp及其所处的环境中评定质量？

- 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、不稳定性及与相应的实现标准(例如，Java或AJAX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，并支持新内容、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需要的导航语法及语义。
- 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，而且没有出现操作上不可接受的性能降低。
- 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。

“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的技术时，却出现了一项新技术[WebApp]，以前的好办法都不灵了。”——James Bach

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。

20.1.2 WebApp环境中的错误

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？

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]：

1. 由于WebApp测试发现的很多类型的错误都首先表现在客户端的问题（即通过在特定浏览器或个人通信设备上实现的接口），Web工程师看到了错误的征兆，而不是错误本身。

2. 由于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不合适的HTML（或其他程序设计语言）编码的结果，

⊖ 第14章讨论的通用软件质量维度同样可以应用于WebApp。

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），而另外一些错误可归于动态的操作环境（即瞬间的资源负载或时间相关的错误）。

上述5个错误特点说明：在WebApp测试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17章），并建议使用面向对象系统所使用的策略和战术（第19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。

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。

WebRef
WebApp测试方面的优秀文章可以在www.stickyminds.com/testing.asp找到。

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以下方面的评估，包括内容和导航错误、可用性、兼容性、WebApp的安全性、可靠性及性能等方面的评估。

由于很多WebApp在不断进化，WebApp测试是Web支持人员所从事的一项持续活动，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，使用策划一词（在任何环境中）是不受欢迎的。这些开发人员不做策划，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。更严格的方法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。Splain和Jaskiel[Spl01]在他们关于WebApp测试的书中谈道：

除了最简单的网站，对某种测试计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应用系统人员的负担。他们不仅要幻想虚构的新测试，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。

KEY POINT
测试计划确定了测试任务集、将被开发的工作产品，以及结果被评估、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) 当测试开始时，所应用的任务集合[⊖]；(2) 执行每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单独的文档。

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。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。

图20-1将WebApp的测试过程与WebApp的设计金字塔（第13章）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。

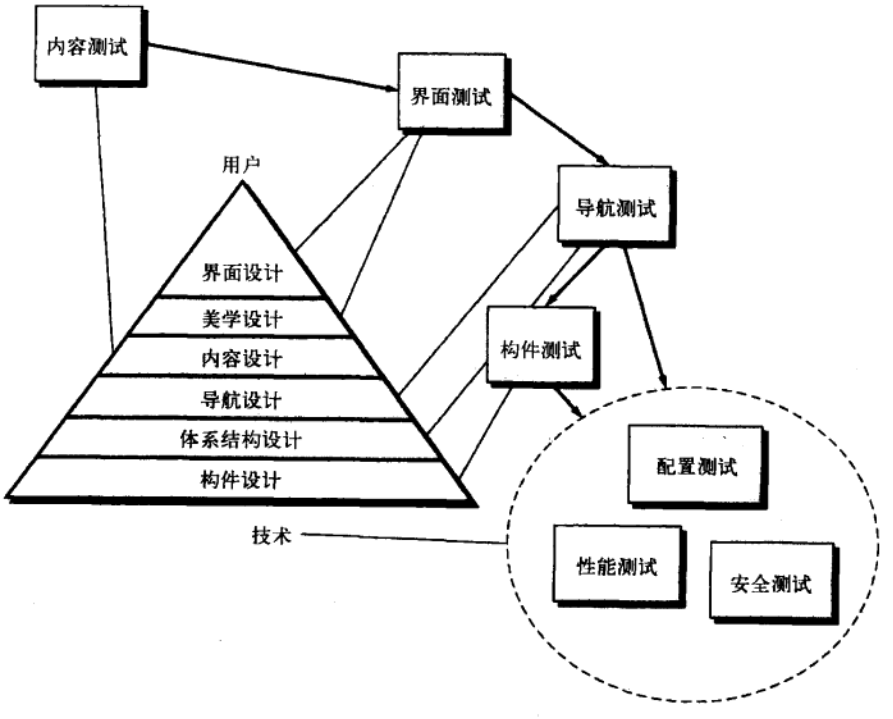


图20-1 测试过程

20.3 内容测试

Web应用内容中的错误可以小到较小的印刷错误，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

⊖ 第2章对任务集合进行了讨论。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。

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）。

ADVICE
虽然正式的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。

20.3.1 内容测试的目标

内容测试具有3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（即信息的准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。

KEY POINT
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。

为了达到第一个目标，可以使用自动的拼写和语法检查。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。实际上，大型网站会得到专业稿件编辑器的帮助，以发现印刷错误、语法错误、内容一致性错误、图形表示错误和交叉引用错误。

要发现内容中的语义错误，应该询问和回答哪些问题？

语义测试关注于每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：

- 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？或者是否会引起诉讼？
- 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。

“一般情况下，在其他应用中使用的软件测试技术与在基于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

内容对象存在于具有特定风格的体系结构之中（第13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 WebApp显示了关于传感器的多种信息，其中传感器是安全和监视产品的一部分。内容对象提供描述信息、技术规格说明、照片和相关的信息。SafeHomeAssured.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接口，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。数据库测试的目标就是发现这些错误，然而，数据库测试会由于以下多种原因而变得复杂：

哪些问题使WebApp的数据库测试变得复杂了？

1. 客户端请求的原始信息很少能够以能够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[⊖]。

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4种因素，对图20-2中记录的每一“交互层”[Ngu01]，都应该使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，此功能格式化为合适的查询（例如，SQL）；(4) 查询被传递到数据管理层[⊖]，此层与数据库访问程序（很可能位于另一台机器）通信。

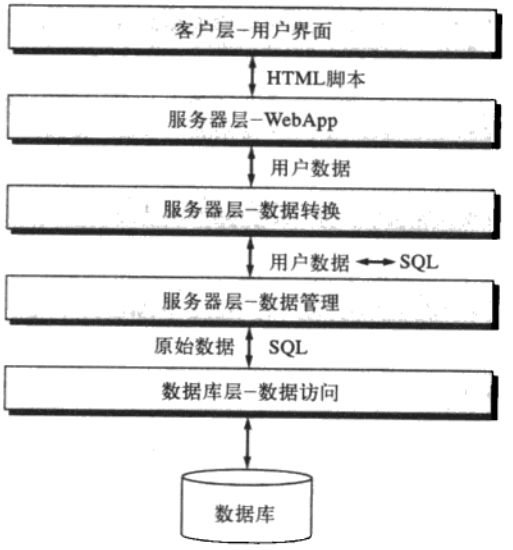


图20-2 交互层

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合格性确认，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。

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

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

⊖ 当遇到分布式数据库，或者需要访问数据仓库（第1章）时，这些测试可能变得非常复杂。

⊖ 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）。

数据管理构件。

合理地设计这些数据库测试需要了解的主要技术的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]。

20.4 用户界面测试

除了面向WebApp的详细设计说明以外，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。

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。在需求分析阶段，对界面模型进行评审，确保与利益相关者的需求及分析模型的其他元素相一致；在设计阶段，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（第11章），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

界面测试检查用户界面的交互机制，并从美学角度对用户界面进行确认。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，必须启动下面的一些战术步骤：

- 对界面要素进行测试，确保设计规则、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、脚本、流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，在用例或导航语义单元NSU（第13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。
- 与选择用例及NSU有所不同，此方法要对所有界面进行测试，发现界面的语义错误。这种测试方法类似于确认测试，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。

外部链接测试应该贯穿WebApp的整个生命期，支持策略的一部分应该是有规律的、定期的链接测试。

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]：

链接。对每个导航链接进行测试，确保获得了正确的内容对象或功能[⊖]。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，

⊖ 这些测试可以作为界面测试或导航测试的一部分。

然后分别运行每个链接。另外，一定要执行每个内容对象内的链接，发现错误的统一资源定位符URL或者链接到不正确的内容对象或功能。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) 执行对输入数据进行错误检查的脚本工作正常，并且提供了有意义的错误信息。

在更具有目标性的层次上，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。

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。

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所提供的数据，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“你应该保证你公司的[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的每一个网页，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。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]。

流内容。测试应该证明流数据是最新的、显示正确、能够无错误地暂停，并且很容易重新启动。

Cookies：服务器端的测试和客户端的测试都是需要的。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是否具有合适的持久性，确保有效日期正确。在客户端，用测试来确定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）。

特定应用系统的界面机制。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]为电子商务应用系统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18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久性（这一点应该作为客户需求的一部分详细说明）。
- 测试WebApp在将来某个日期是否能够记起购物车的内容（假设没有购买活动发生）。

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关注点转移到界面的语义测试。界面的语义测试是要“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”[Ngu00]。

彻底复习一下界面设计模型，我们就能够得到前面段落中所蕴涵问题的部分答案。然而，一旦实现了WebApp，就应该对每个用例场景（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，确保每个菜单项都至少被运行一次，并且内容对象中的每个嵌入的链接都已经使用。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[⊖]。

可用性测试可以由测试人员设计，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。

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。

? 可用性的哪些特性变成了测试的焦点？描述了哪些特殊的目标？

可用性测试的第一步是确定一组可用性类别，并对每一类可用性建立测试目标。下面的测试类别和目标（以问题的形式书写）举例说明了这种方法[⊜]：

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[⊝]？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？

⊖ 这种问题常用术语用户友好性来表示。当然，问题在于一个用户对于“友好”界面的感觉可能根本不同于另外一个用户。

⊜ 对于可用性的其他信息，参见第11章。

⊝ 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。

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？

个性化——WebApp是否能够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：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。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。

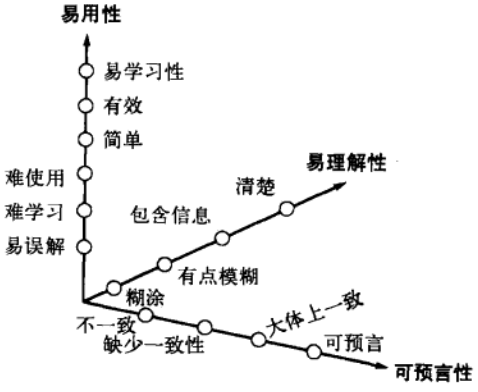


图20-3 可用性的定性评估

20.4.5 兼容性测试

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的标准化程度如何，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。

KEY POINT
WebApp运行于不同的客户端环境中。兼容性测试的目标是发现与特定的环境（例如，浏览器）有关的错误。

在某些情况下，小的兼容性问题显得不是很严重，而在有些情况下，就可能遇到严重的错误。例如，下载速度可能变得不能接受，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，导出一系列兼容性确认测试，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产品软件工程团队的成员。

[对话]

Doug：对于SafeHomeAssured.com电子商务WebApp 0.0版，你是怎样看的？

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。我们仍然在致力于推出产品软件。

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，此系统将让你

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。每个WebApp功能都是一个软件构件（用多种编程语言或脚本语言中的一种实现的），并且可以用第18章讨论的黑盒（及在某些情况下的白盒）技术对其进行测试。

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18章）：

- *等价类划分*——将功能的输入域划分为输入种类或输入类，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、不存在的邮政编码和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[⊖]。
- *路径测试*——如果功能的逻辑复杂性较高[⊜]，可以使用路径测试（白盒测试用例设计方法）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失败，由错误的输入而导致的错误输出，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输出。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领域中不可分割的一部分。

20.6 导航测试

用户在WebApp中旅行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，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件事的影响，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。

20.6.1 测试导航语法

"我们没有迷路，我们面临定位挑战。"——John M. Ford

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，以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，选择的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

⊖ 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，从而排除用户指定超出范围的输入。

⊜ 逻辑复杂性可以通过计算算法的环复杂性来确定。详细内容见第18章。

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、下载性能和浏览器兼容性方面的测试。

- *站点地图*——站点地图提供了所有网页内容的完整列表，应该对每个站点地图的入口进行测试，确保链接引导用户到达合适的内容或功能。
- *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准确性和完备性、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），而另外一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

在测试NSU时，必须提问和回答哪些问题？

在第13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。

在测试每个NSU时，Web工程团队一定要回答下面的问题：

- 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结点上运行，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结点上运行，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？

ADVICE

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利益相关者、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

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以预料的。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、新的界面分离原则ISP（见10.2节）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，以配置测试用例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。

当进行服务器端配置测试时，必须询问和回答哪些问题？

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：

- 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？
- 是否已经利用所选择的分布式服务器配置[⊖]（如果存在一种配置）对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得到检查？
- 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*——Firefox、Safari、Internet Explorer、Opera、Chrome及其他浏览器。
- *用户界面构件*——Active X、Java applets及其他构件。
- *插件*——QuickTime、RealPlayer及很多其他插件。

⊖ 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。

• 连通——电缆、DSL、常规的调制解调器、T1、WiFi。

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[⊖]。为了实现这一点，要对每类用户进行评估，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可能的构件组合，然后，在这些环境中测试WebApp。

20.8 安全性测试

"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

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，必须要对该主题有充分的了解[⊖]。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。

应该设计安全性测试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。

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），或者在最坏的情况下破坏用户数据。

ADVICE

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。

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。

KEY POINT

应该将安全性测试设计为检验防火墙、鉴定、加密和授权。

在服务器端，薄弱环节包括拒绝服务攻击和恶意脚本，这些恶意脚本可以传送到客户端，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]：

• 防火墙（firewall）——是硬件和软件相结合的过滤机制，它检查每一个

⊖ 在每一种可能的配置构件的组合中运行测试是非常耗费时间的。

⊖ 由Cross和Fisher[Cro07]、Andrew和Whittaker[And06]及Trivedi[Tre03]编写的书籍提供了关于此主题的有用信息。

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。

- 鉴别（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 certificates），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，只有对那些具有合适的授权码（例如，用户身份证号和密码）的人，才允许访问客户或服务器环境。

应该设计安全性测试，探查每种安全性技术来发现安全漏洞。

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。

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感到灰心的了；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感到烦恼的了；WebApp对某些情形能够立即做出反应，而对某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、不完善或不强的操作系统能力、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）增加做出反应；（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

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。

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：

- 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的可靠性和准确性是否会受影响？
- 当负载大于服务器容量的最大值时，会发生什么情况？
- 性能下降是否对公司的收益有影响？

为了得到这些问题的答案，要进行两种不同的性能测试：

（1）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。

（2）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；

T，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WebApp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。

每种情况下，在系统正常的操作范围内定义这些变量。当每种测试条件运行时，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：

$$P = [20\,000 \times 0.5 \times 3\text{KB}]/60 = 500\text{KB/s}$$
$$= 4\ \text{Mb/s}$$

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？

压力测试是负载测试的继续，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？
- 当容量超过时，事务会丢失吗？
- 当容量超过时，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型称为脉冲/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，可以直接适用于Web工程环境中的测试。对每类工具我们都提供了代表性工具[⊖]。

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。他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）；

Apple Debugging Tools（developer.apple.com/tools/performance/）；

IBM VisualAge Environment（www.ibm.com）；

Microsoft Debugging Tools（www. microsoft.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）；

ForeSoft BugTrack（www.bugtrack.net）；

McCabe TRUETrack（www.mccabe.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）；

开源测试工具（www.opensourcetesting.org/performance.php）；

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

Web Performance Load Tester（www.webperformanceinc.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.softwareqatest.com/qatweb1.html上有全面的列表。

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WebApp设计完成后进行评审，一旦实现了WebApp，就对其进行测试。

WebApp测试策略检查每个质量维度，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WebApp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准确性，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。

导航测试使用从建模活动中得出的用例，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。

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。

习题与思考题

20.1 是否存在应该完全忽视WebApp测试的任何情况?

20.2 用你自己的话，讨论在Web应用环境下测试的目标。

20.3 兼容性是一项重要的质量维度，必须对哪方面进行测试才能确保WebApp具有兼容性?

20.4 哪些错误趋向于更加严重，是客户端错误还是服务器端错误？为什么?

20.5 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?

20.6 开发正式书写的测试计划是否总是必要的？给出解释。

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 内容测试是传统意义上的真正测试吗？给出解释。

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 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?

20.11 假设你正在开发满足老年人需要的在线药房（YourCornerPharmacy.com）。药房提供了典型的功能，但也为每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。

20.12 假设你已经为YourCornerPharmacy.com（习题20.11）实现了一个药品互作用检查功能。讨论必须进行的构件级测试的类型，以确保这项功能工作正常。[注意：实现这项功能将必须使用数据库。]

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 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?

20.15 安全性测试的目标是什么？谁来进行这种测试活动?

20.16 YourCornerPharmacy.com（习题20.11）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。

20.17 为使其成功，CornerPharmacy.com（习题20.11）已经实现了一个特殊的服务，单独处理处方的重新填写。平均情况下，1000个并发用户每两分钟提交一次重填请求，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Andrews和Whittaker（《How to Break Web Software》，Addison-Wesley，2006）、Ash（《The Web Testing Companion》，Wiley，2003）、Nguyen和他的同事（《Testing Applications for the Web》，second edition，Wiley，2003）、Dustin和他的同事（《Quality Web Systems》，Addison-Wesley，2002）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。

微软（《Performance Testing Guidance for Web Applications》，Microsoft Press，2008）和Subraya（《Integrated Approach to Web Performance Testing》，IRM Press，2006）对WebApp性能测试进行了详细

论述。Chirillo（《Hack Attacks Revealed》，second edition，Wiley，2003）、Splaine（《Testing Web Security》，Wiley，2002）、Klevinsky和他的同事（《Hack I.T.：Security through Penetration Testing》，Addison-Wesley，2002）、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。另外，讲述软件安全测试的书籍通常能够为那些必须测试WebApp的人提供重要的指导。有代表性的书籍包括：Basta和Halton（《Computer Security and Penetration Testing，Thomson Delmar Learning，2007》、Wysopal和他的同事（《The Art of Software Security Testing》，Addison-Wesley，2006）以及Gallagher和他的同事（《Hunting Security Bugs》，Microsoft Press，2006）。

在Internet上可以获得大量WebApp测试的信息资源。与WebApp测试有关的WWW最新引用列表可以在SEPA站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第21章

Software Engineering：A Practitioner's Approach (Seventh Edition)

形式化建模与验证

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”？如果我们在软件开发中做到了这一点，就可以大大降低不必要的软件返工的成本。两种高级的软件工程方法——净室软件工程方法和形式化方法——可以帮助软件工程团队“第一次就将事情做正确”，这两种方法提供了基于数学的方法进行建模，并具有验证模式正确性的能力。净室软件工程强调在程序构造开始之前进行正确性的数学验证，并且将软件可靠性认证作为测试活动的一部分。形式化方法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（甚至证明）正确性和一致性。两种方法的底线是创建具有极低故障率的软件。

人员：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？这正是形式化建模和验证的前提。

步骤：使用数学验证易于处理的特殊符号创建需求和设计模型。净室软件工程使用盒结构表示方法，盒结构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。形式化方法将软件需求翻译成更形式化的表示，方法是使用符号和启发式规则为系统功能定义数据不变式、状态及操作。

工作产品：开发特殊的、形式化的需求模型。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：形式化的正确性证明应用于需求模型。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。

关键概念

盒结构规格说明
认证
净室设计
净室过程模型
正确性验证
设计求精
形式化规格说明语言
功能规格说明
对象约束语言(OCL)
Z规格说明语言

一旦完成了软件模型和代码的开发就要进行评审和测试。形式化建模和验证与评审和测试有所不同，形式化建模和验证是融合到特殊的建模方法中的，这些建模方法往往与规定的验证方法集成在一起。没有合适的建模方法，就不可能完成验证。

本章讨论两种形式化建模与验证方法——净室软件工程和形式化方法。两种方法都要求特殊的规格说明方法，并且每种方法都适合于一种独特的验证方法。两种方法都非常严格，都没有被软件工程团体广泛使用。但是，如果要构造出非常健壮（子弹打不穿）的软件，这些方法给你的帮助会是巨大的，因此，值得学习。

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同时进行代码增量的统计质量验证。

通过强调证明正确性的要求，净室方法在很多方面将软件工程提升到另一个层次。

在形式化方法中，使用说明系统功能和行为的形式化语法和语义来描述所开发的模型。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] 在介绍形式化方法时给出了下面的论述（同样适用于净室方法）：

形式化方法[和净室软件工程]是有争议的。支持者声称该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[和净室软件工程]很不熟悉，因此难于判断这些争论。

在本章，我们探索形式化建模和验证方法以及它们对未来软件工程的潜在影响。

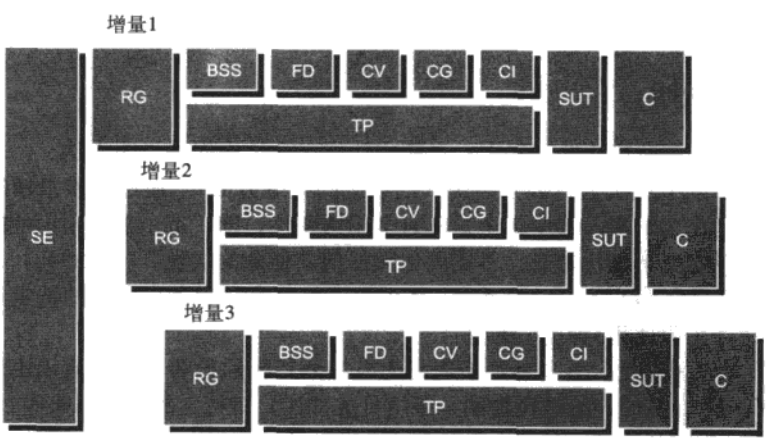
21.1 净室策略

"程序中出现错误的唯一方式是：作者将错误引入进去。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

净室方法使用第2章所介绍的增量过程模型的专业版。一个"软件增量的流水线" [Lin94b] 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21-1中给出。在净室增量的流水线中，需要完成以下任务：

- **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及时集成。
- **需求收集。**使用类似于在第5章介绍的技术，（为每个增量）开发更详细的客户级需求描述。
- **盒结构规格说明。**运用盒结构的规格说明方法描述功能规格说明。盒结构"在每个细化级别上使行为、数据及规程的构造性定义独立" [Hev93]。



SE—系统工程
RG—需求收集
BSS—盒结构规格说明
FD—形式化设计
CV—正确性验证
CG—代码生成
CI—代码检查
SUT—统计使用测试
C—认证
TP—测试策划

图21-1 净室过程模型

WebRef
有关净室软件工程的最佳信息源可在www.cleansoft.com找到。

"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

ADVICE
净室方法强调按软件真正使用的方式进行测试。用例为测试策划过程提供了输入。

- **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）。
- **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（21.3.2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，使用技术评审（第15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。
- **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"（21.4节）。如图21-1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18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（21.4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。

净室过程的前4项活动确定了后面的形式化验证活动阶段。因此，下面开始讨论具有建模活动的净室方法，这些建模活动对于将要应用的形式化验证非常重要。

21.2 功能规格说明

"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

作为盒结构规格说明的一部分，细化是如何完成的？

在净室软件工程中，建模方法使用盒结构规格说明方法。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，其中每个盒具有引用透明性，即"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"[Lin94b]。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。净室软件工程使用3种类型的盒：

- **黑盒**。黑盒刻画系统行为或部分系统的行为。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。

图21-2给出了使用盒结构规格说明的求精方法。黑盒（BB_1）定义了对完整触发集合的反应。可以将BB_1细化成一组黑盒，$BB_{1.1}$到$BB_{1.n}$，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（$CB_{1.1.1.n}$），并且刻画出过程的设计细节。

KEY POINT
盒结构求精和正确性验证同时进行。

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。

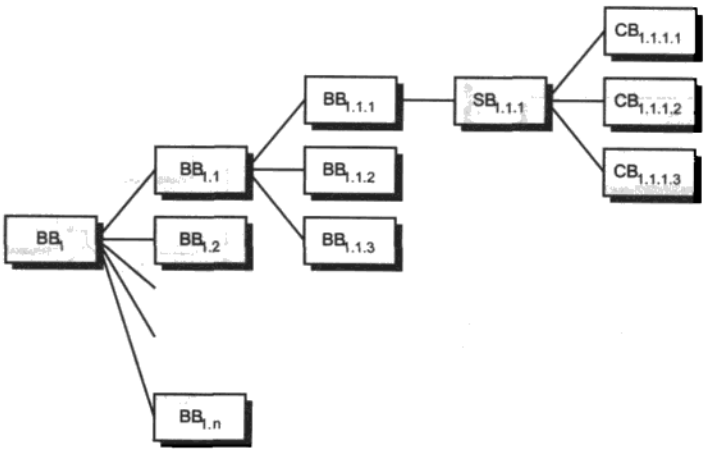


图21-2 盒结构求精

21.2.1 黑盒规格说明

使用图21-3所示的符号［Mil88］，黑盒规格说明描述一种抽象、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*。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*。

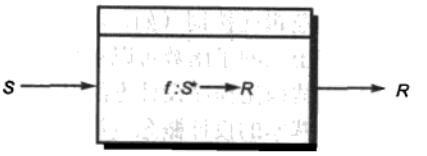


图21-3 黑盒规格说明

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。

21.2.2 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7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动作（反应）。

如图21-4所示，状态盒可以与黑盒*g*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函数*f* 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。当整体考虑时，状态-子函数对（t, g）定义了黑盒函数f。

21.2.3 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。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21-5所示的清晰盒。在图21-4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证明是正确的，这一主题在21.3节讨论。

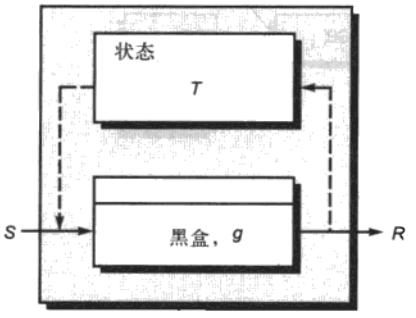


图21-4 状态盒规格说明

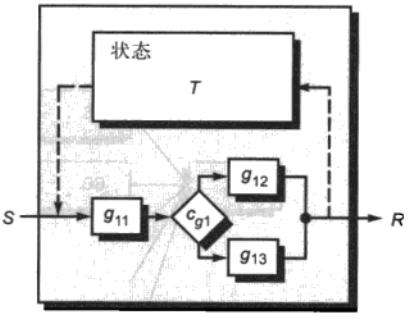


图21-5 清晰盒规格说明

21.3 净室设计

净室软件工程主要运用结构化程序设计原理（第10章）。但是，在这里结构化程序设计应用得更严格。

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8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、信息隐藏和数据类型概念进行数据设计。

21.3.1 设计求精

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。在清晰盒中，使用结构化程序设计结构和逐步求精表示过程细节。例如，一个程序函数f被细化为子函数g和h的序列，这两个子函数又进一步细化为条件结构（例如，if-then-else 和do-while）。进一步求精直到具有足够的过程细节来创建所需要的构件。

在每个求精层次，净室团队[⊖]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：

⊖ 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。

使用什么条件证明结构化构造的正确性？

• 执行g之后再执行h能完成f的功能吗？

如果一个函数p被细化为形如if＜c＞then q 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do n while ＜c＞[⊖]，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？

每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。

21.3.2 设计验证

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。

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。如果违反这种结构，正确性证明将是困难或不可能的。

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]给出的一个简单例子。目的是设计并验证一个小程序，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21-6给出的流程图来表示[⊜]。

为了验证该设计的正确性，我们必须定义如图21-6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21-6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(subproof)。

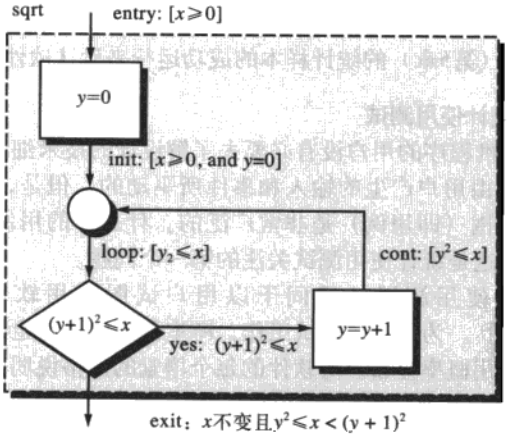


图21-6 计算平方根的整数部分[Lin79]

1. 条件init 要求［x≥0，and y=0］。基于问题的需求，假定入口条件是正确的[⊛]。因此，

⊖ 原书中无此表达式，译者为方便读者理解特意补充了此表达式。——译者注

⊜ 图21-6采自[Lin94]，已得到使用许可。

⊛ 在这里，负数的平方根没有意义。

为了证明设计正确性，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。

init 条件的第一部分$x \geqslant 0$ 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（此时直接满足loop条件），或（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$(y+1)^2 \leqslant x$，则$y^2 \leqslant x$，条件cont成立。

4. 在如图所示的条件逻辑中测试条件yes，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。因为当条件测试$(y+1)^2 \leqslant x$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，$y^2 \leqslant x$），因此，$(y+1)^2 > x$和$y^2 \leqslant x$ 可以组合在一起满足exit条件。

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$x \geqslant 0$，因此，最后循环一定终止。

上面的5 步是图21-6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。

针对设计验证的更严格的数学方法是可能的，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] 。

21.4 净室测试

“质量不是条例，而是习惯的成果。” —— Aristotle

净室测试的策略在根本上不同于传统测试方法（第17章～第20章）。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5章）的统计样本的成功运行来确认软件需求。

21.4.1 统计使用测试

即使你决定使用净室方法，将统计使用测试作为测试策略不可分割的一部分也值得考虑。

计算机程序的用户没有必要去了解设计的技术细节。程序的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”[Lin94b] 。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、使用场景的建立以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[⊖]。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。为了更容易选择测试用例，这些概率被映射为1 ~99 的数字区间 [Lin94] ，如下表所示。

⊖ 可使用自动化工具完成此项工作。进一步的信息请参见[Dye92]。

程序触发（Program stimulus）	概率（Probability）	区间（Interval）
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一个区间相对应。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

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使得可以确定间隔时间。利用间隔时间，认证团队可以计算出平均失效时间MTTF。如果进行一个很长的测试序列，没有出现故障，则MTTF较低，软件可靠性较高。

21.4.2 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。

如何认证一个软件构件？

认证方法包括5个步骤［Woh94］：

1. 必须创建使用场景。
2. 指定使用概貌。
3. 从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。净室软件工程的认证需要创建3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。有兴趣的读者参阅文献[Cur86]、［Mus87］和[Poo93]可获得更多的详细信息。

21.5 形式化方法的概念

《The Encyclopedia of Software Engineering》[Mar01]中对形式化方法的定义如下：

在开发计算机系统中使用的形式化方法是描述系统特性的基于数学的技术。这种形式化方法提供了框架，在这些框架内，人们能够以系统的方式而不是随意的方式来说明、开发和验证系统。

"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

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，用于形式化方法的基于数学的规格说明语言增加了实现这些理想的可能性。规格说明语言的形式化语法（21.7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（如英语）或图形表示（如UML）时经常产生的歧义性。集合论和逻辑符号的描述工具使得可以清晰地陈述需求。要达到一致，在规格说明中某地方陈述的需求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[⊖]。

为了介绍形式化方法的基本概念，我们考虑几个简单的例子来说明数学规格说明的使用，而不是给读者堆砌太多的数学细节。

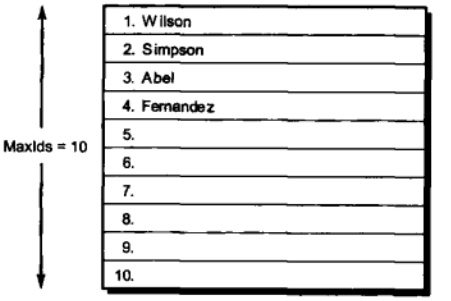


图21-7 符号表

KEY POINT 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。

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21-7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、网络通信系统中的计算机名称表、产品运输时间表系统中的目的地表。

ADVICE 状态概念的另一种说法是数据决定状态。也就是说，可以通过检查数据来看系统处于什么状态。

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——被称为数据不变式——的一个构成成分。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。

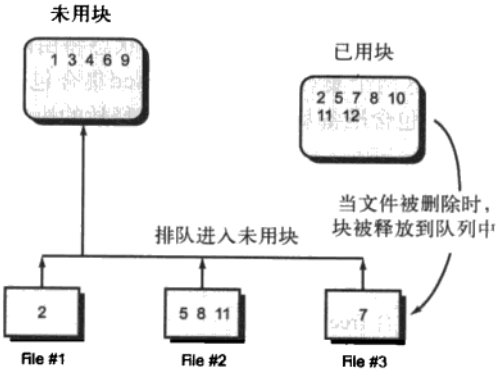
另一个重要的概念是状态。许多形式化语言，如OCL（21.7.1节），使用第7章所讨论的状态概念；也就是说，系统可能处于多种状态之一，每一种状态都表示外部可观察到的行为模式。然而，在Z语言（21.7.2节）中，对术语状态有不同的定义。在Z语言（及相关的语言）中，系统的状态由系统的存储数据来表示（因此，Z语言给出了太多的状态来表示每种可能的数据配置）。在符号表程序的例子中使用后面的定义，状态就是符号表。

⊖ 实际上，即使使用形式化方法，完整性也是难于保证的。在创建规格说明时，系统的某些方面可能还没有定义；某些特性可能被故意遗漏，以允许设计人员具有某些自由度来选择实现方法；最后，在大型、复杂的系统中，不可能考虑每个操作场景。某些事情可能由于疏忽而被遗漏。

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：操作add()将一个指定名增加到符号表，操作remove()从符号表中去除一个现有名[⊖]。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。对于add()操作，后置条件将用数学方法描述表中已经增加了新标识符。

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21-8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。



块队列，其中包含从被删除文件释放的块

图21-8 块处理器

当你需要为一个相当复杂的功能开发数据不变式时，让大家各自发表独创性意见（也称头脑风暴）的方法非常有效。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。

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：

- 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。

与数据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。

add()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。remove()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。check()操作没有前置条件，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。

⊖ 需要注意的是：在表满状态时不能增加名字，在表空状态时不可能删除名字。

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21.6节中，我们将考虑如何使用这些数学表示法对系统的某些元素进行形式化说明。

21.6 应用数学表示法[⊖]描述形式化规格说明

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？

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21.5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，并同时保持对当前已使用块的跟踪。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21-8所示。

假定名为BLOCKS的集合包含所有块号，集合AllBlocks是位于1和MaxBlocks间的块组成的集合。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集合。状态可以描述为：

used，free：P BLOCKS

BlockQueue：seq P BLOCKS

这和程序变量的声明非常相似，它说明used和free是块集合，BlockQueue是一个序列，序列中的元素是块集合。数据不变式可以描述为：

used ∩ free =∅ ∧

used ∪ free = AllBlocks ∧

∀i: dom BlockQueue · BlockQueus i ⊆ used ∧

∀i, j: dom BlockQueue · i≠j (BlockQueue i ∩ BlockQueue j = ∅

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。

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，即

#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 \ head BlockQueue ∧

free' = free ∪ head BlockQueue ∧

BlockQueue' = tail BlockQueue

如何表示前置条件和后置条件？

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

⊖ 我在编写这一节时假设读者已经熟悉与集合和序列有关的数学表示法，以及在谓词演算中使用的逻辑符号。如果你需要复习，在第7版的Web站点上有简要介绍，可作为补充资料。更详细的信息请参见[Jec06]或[Pot04]。

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。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：

Ablocks ⊆ 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 ⌢ <Ablocks> ∧

used' = used ∧

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。

21.7 形式化规格说明语言

形式化规格说明语言通常由3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。规格说明语言的语义域指出语言如何表示系统需求。

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6章和第7章中我们用非形式化方法这样做了。我们依次描述了信息、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，这两种关系的交集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[OMG03b]、Z[ISO02]、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。

21.7.1 对象约束语言[⊖]

对象约束语言（Object Constraint Language，OCL）是成熟的形式化表示法，UML的使用者可以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。

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（附录1）——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 Collection类的实例，Set和Sequence是OCL Collection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：

⊖ 这一节由加拿大渥太华（Ottawa）大学的Timothy Lethbridge 教授提供，并允许在此使用。

- 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有名为assoc的一对多关联关系，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提供了内置运算实现集合和逻辑操作符、构造规格说明及相关的数学。在表21-1中列出了一部分。

表21-1 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、一个关联末端的对象集合、一个操作的结果，或者依赖于UML图类型的其他事情。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
pimplies 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→ size()	聚集c中的元素个数
c → isEmpty()	如果c没有元素，则为真，否则为假
c1→includesAll(c2)	如果c2中的每个元素在c1中都存在，则为真
c1 →excludesAll(c2)	如果c2中的元素在c1中都不存在，则为真
c →forAll(elem | boolexpr)	当boolexpr应用到c中的每个元素都为真，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→ forAll(elem1，elem2 | 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→ isUnique(elem | expr)	当应用到c的每个元素时，expr得到不同的值，则为真
针对集合操作的示例	
s1→intersection(s2)	在s1中出现，也在s2中出现的元素组成的集合
s1 →union(s2)	在s1中出现，或在s2中出现的元素组成的集合
s1→ excluding(x)	将对象x去除后的集合s1
针对序列的操作示例	
seq→ first()	序列seq的第一个元素

在本节中，借助21.5节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（图21-9）。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21.5节中讨论的不变式的6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

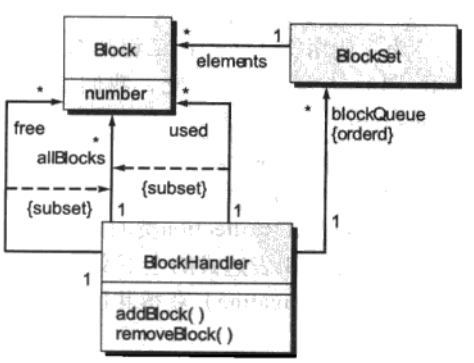


图21-9 块处理器的类图

辑，还有助于评审者发现错误，例如，英语与逻辑不对应的情形。

1. 块不会同时被标记为未用和已用。

```
context BlockHandler inv:
  (self.used -> intersection(self.free)) -> isEmpty()
```

注意，每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，用括号{}括起来。这里的关键字self指BlockHandler的实例；下面我们将省略self，这在OCL中是允许的。

2. 所有在队列中的块集合都将是当前已用块集合的子集。

```
context BlockHandler inv:
   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. 未用块集合中没有重复的块号。

```
context BlockHandler inv:
   free -> isUnique(aBlock | aBlock.number)
```

6. 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.elements) 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。

21.7.2 Z规格说明语言

Z（正确读音为“zed”）是一种规格说明语言，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。

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规格说明被组织为一组schema——用于说明变量及这些变量之间关系的语言结构。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相同的方式，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稍有不同[⊖]。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：

—— schemaName ——
声明（declarations）

不变式（invariant）

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 语言表示法的概要说明在表21-2中给出。

表21-2 Z表示法概要说明

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：

——x——
声明（declarations）

谓词（predicates）

全局函数和常量的定义形式如下：

声明

谓词

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中的而不在T中的每个元素。
$\varnothing$	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。
max(S)	非空的数字集合S中的最大者。
函数（Function）：	
$f : X \rightarrowtail Y$	f被声明为从X到Y的部分内射。

⊖ 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）

dom f	f的定义域：定义$f(x)$的x的值集。
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。
$\{X\} \lhd f$	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 and 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。

下面schema 的例子描述了块处理器的状态和数据不变式：

BlockHandler

$used, free$：P $BLOCKS$

$BlockQueue$：seq P $BLOCKS$

$used \cap free = \varnothing \wedge$

$used \cup free = AllBlocks \wedge$

$\forall i$：dom $BlockQueue \bullet BlockQueue\ i \subseteq used \wedge$

$\forall i, j$：dom $BlockQueue \bullet i \neq j \Rightarrow BlockQueue\ i \cap BlockQueue\ j = \varnothing$

如我们注意到的那样，该schema 包含两个部分，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△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：

RemoveBlock

△ $BlockHandler$

#BlockQueue > 0,

$used' = used \setminus head\ BlockQueue \wedge$

$free' = free \cup head\ BlockQueue \wedge$

$BlockQueue'$ = tail $BlockQueue$

对△ 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

△$BlockHandler$

$Ablocks?$ ： $BLOCKS$

$Ablocks? \subseteq used$

$BlockQueue' = BlockQueue \cap \langle Ablocks? \rangle \wedge$

$used' = used \wedge$

$free' = free$

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：[⊖]

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://vl.fmnet.info/可找到90多种形式化方法工具。

21.8 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，此信息对认证被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，清晰盒用于对某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。产生的错误记录和取样、构件和认证模型相结合，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。其底线是非常低的失效率，这是使用非形式化方法难于或不可能达到的。

形式化方法使用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。每个操作都与两个条件相关联，即前置条件和后置条件。

净室软件工程或形式化方法是否会广泛使用？答案是“可能不会”。与传统的软件工程方法相比，这两种方法更难于学习，并且对于很多软件工作者是重要的“文化冲击”。但是，下一次你听到某人悲叹“为什么我们不能第一次就将软件做正确？”时，你应该知道确实有技术可以做到。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

习题与思考题

21.1 如果必须选出净室软件工程和传统的或面向对象的软件工程方法根本不同的一个方面，那么这个方面是什么？

21.2 增量过程模型和认证如何一同工作生产出高质量软件？

21.3 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。

21.4 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，从而能够证明该算法是正确的。

21.5 为思考题21.4中讨论的冒泡排序的正确性证明编写文档。

21.6 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，定义每个场景的使用概率，然后开发出类似21.4.1节中的程序触发和概率分布表。

21.7 对思考题21.6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。

21.8 用你自己的话描述净室软件工程中认证的意图。

21.9 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。

21.11 使用表21-1或表21-2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。

21.12 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

近年来出版的高级规格说明和验证技术方面的书籍很少。然而，一些新的文献值得考虑。Gabbar编写的书（《Modern Formal Methods and Applications》，Springer, 2006）不仅介绍了基础、新的开发，还

介绍了高级应用。Jackson（《Software Abstractions》，The MIT Press，2006）介绍了所有基础支持及他称为的“轻量级形式方法。”Monin和Hinchey（《Understanding Formal Methods》，Springer，2003）提供了该主题的精彩介绍。Butler和其他作者（《Integrated Formal Methods》，Springer，2002）介绍了大量形式化方法方面的论文。

除了本章中作为参考文献使用的书目外，Prowell和他的同事（《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版。“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。

通过正确性证明进行的设计验证是净室方法的核心。Cupillari（《The Nuts and Bolts of Proofs》，3rd ed.，Academic Press，2005）、Solow（《How to Read and Do Proofs》，4th ed.，Wiley，2004）、Eccles（《An Introduction to Mathematical Reasoning》，Cambridge University Press，1998）的书提供了对数学基础的精彩介绍。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。

在形式化方法领域，Casey（《A Programming Approach to Formal Methods》，McGraw-Hill，2000）、Hinchey和Bowan（《Industrial Strength Formal Methods》，Springer-Verlag，1997）以及Sheppard（《An Introduction to Formal Specification with Z and VDM》，McGraw-Hill，1995）的书提供了有用的指导。另外，针对特定语言的书，如Warmer和Kleppe（《Object Constraint Language》，Addison-Wesley，1998）、Jacky（《The Way of Z: Practical Programming with Formal Methods》，Cambridge University Press，1997）、Harry（《Formal Methods Fact File: VDM and Z》，Wiley，1997）以及Cooper和Barden（《Z in Practice》，Prentice-Hall，1995）的书提供了形式化方法及多种建模语言的有用介绍。

在网上有净室软件工程及形式化方法的大量信息。有关形式化建模和验证的最新WWW文献列表可在SEPA Web站点www. mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

软件配置管理

要点浏览

概念：当开发计算机软件时，变更就会发生。而且，因为变更的发生，需要有效地管理变更。软件配置管理（Software Configuration Management, SCM）也称为变更管理，是一组管理变更的活动。它通过下面的方式来管理变更：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：参与软件过程的每个人在某种程度上都参与变更管理，但是有时候也设专人来管理SCM过程。

重要性：如果你不控制变更，那么变更将控制你。这绝不是一件好事情。一个未受控制的变更流可以很容易地将一个运行良好的软件项目带入混乱。结果会影响软件质量并且会推迟软件交付。为此，变更管理是质量管理的重要部分。

步骤：因为在构建软件时会创建很多工作产品，因此每个工作产品都需要唯一标识。一旦成功完成标识，则可以建立版本和变更控制机制。为保证变更发生时维护质量，变更过程需要审核；为了通知那些需要知道变更的人员，需要进行变更报告。

工作产品：软件配置管理计划（Software Configuration Management Plan）定义变更管理的项目策略。另外，当启动正式的SCM时，变更控制过程将产生软件变更请求、报告和工程变更工单。

质量保证措施：当每个工作产品都可以标识、跟踪和控制时，当每个变更可以跟踪和分析时，当每个需要知道变更的人员都通知到时，你就做对了。

关键概念
基线
变更控制
配置审核
配置对象
内容管理
标识
中心存储库
SCM过程
软件配置项
状态报告
版本控制
WebApp配置对象
WebApp

在开发计算机软件时，变更（change）是不可避免的。并且，对于共同研发项目的软件开发团队来说，变更增加了混乱的程度。如果变更之前没有经过分析，变更实现时没有做相应的记录，没有向需要了解变更的人员报告变更，或没有以一种能够改进质量并减少错误的方式控制变更时，都会产生混乱。关于该问题，Babich[Bab86]是这样说的：

> 协调软件开发以最大限度地减少混乱的技术称为配置管理。配置管理是对软件开发团队正在构建的软件的修改进行标识、组织和控制的技术，其目标是使错误量减少到最小，并使生产率最高。

软件配置管理（SCM）是在整个软件过程中应用的一种普适性活动。因为变更可能随时出现，SCM活动用于：（1）标识变更；（2）控制变更；（3）保证恰当地实施变更；（4）向其他可能的相关人员报告变更。

明确地区分软件支持和软件配置管理是很重要的。软件支持是一组发生在软件已经交付给客户并投入运行后的软件工程活动。而软件配置管理则是在软件项目开始时就启动，并且只有当软件被淘汰时才终止的一组跟踪和控制活动。

软件工程的主要目标是当发生变更时，使变更更容易地被接受，并减少变更发生时所花费的工作量。本章我们将探讨使得我们能够管理变更的具体活动。

22.1 软件配置管理概述

软件过程的输出信息可以分成3个主要类别：(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（针对不同的软件开发人员和客户）；(3) 数据或内容（包含在程序内部和在程序外部）。在软件过程中产生的所有信息项总称为软件配置（Software Configuration）。

"除变更以外没有什么事情是永恒的。"——Heraclitus，公元前500年

随着软件工程工作的进行，软件配置项（Software Configuration Items，SCI）的层次结构就产生了。每一项可以是单个UML图一样小或者和完整的设计文档一样大的命名信息元素。如果每个SCI只是简单地产生了其他的SCI，则几乎不会产生混乱。不幸的是，另一个变量进入到过程中，即变更。变更可以因为任意理由随时发生。事实上，正如系统工程第一定律（First Law of System Engineering）[Ber80]所述：不管你处在系统生存周期的什么阶段，系统都可能发生变更，并且在整个生存周期中将会持续不断地提出变更的要求。

这些变更的起因是什么？这个问题的答案就像变更本身一样那么多变。然而，有4种基本的变更源：

请求软件变更的起因是什么？

- 新的业务或市场条件，引起产品需求或者业务规则的变更。
- 新的客户需要，要求修改信息系统产生的数据、产品提供的功能或基于计算机的系统提供的服务。
- 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。
- 预算或进度的限制，导致系统或产品的重定义。

软件配置项管理是一组用于在计算机软件的整个生命周期内管理变更的活动。SCM可被视为应用于整个软件过程的软件质量保证活动。在下面的几节中，我们将描述能够帮助我们管理变更的主要SCM任务和重要概念。

22.1.1 SCM场景[⊖]

典型的CM（配置管理）工作场景包括：负责软件小组的项目经理、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。)

每个参与变更管理的人员的职责和应从事的活动是什么？

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出产品。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。

一定存在某种机制，能够保证适当的跟踪、管理和执行同一个构件中同时发生的那些变更。

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些构件有问题的信息。

⊖ 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"，(c)2001，卡内基·梅隆大学。

软件工程师的目标是高效地工作。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。依据系统变更原因日志和究竟如何变更的记录，历史资料应该保持对系统中所有构件的演化过程的记录。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看做是一个审核机制；配置管理员可以把CM看做是控制、跟踪和制定方针的机制；软件工程师可以把CM看做是变更、构建以及访问控制的机制；而用户可以把CM看做是质量保证的机制。

22.1.2 配置管理系统元素

在Susan Dart关于软件配置管理的内容全面的白皮书[Dar01]中，她指明了开发软件配置管理系统时应该具备4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个软件配置项。
- *过程元素*——是一个动作和任务的集合，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*——由实施有效SCM的软件团队使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。

22.1.3 基线

大多数软件变更是合理的，因此没有什么可抱怨的。更确切地说，要确信你已经有了恰当地控制变更的机制。

变更是软件开发中必然会出现的事情。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），这些额外的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：大多数变更是合理的！

基线是一个软件配置管理概念，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。

在软件配置项成为基线之前，可以较快地且非正规地进行变更。然而，一旦成为基线，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次变更。

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15章）已经获得批准的一个或多个软件配置项的交付。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，该设计模型就成为了基线。任何对程序体系结构（在设计模型中已形成文档）的进一步变更只能在每次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22-1所示。

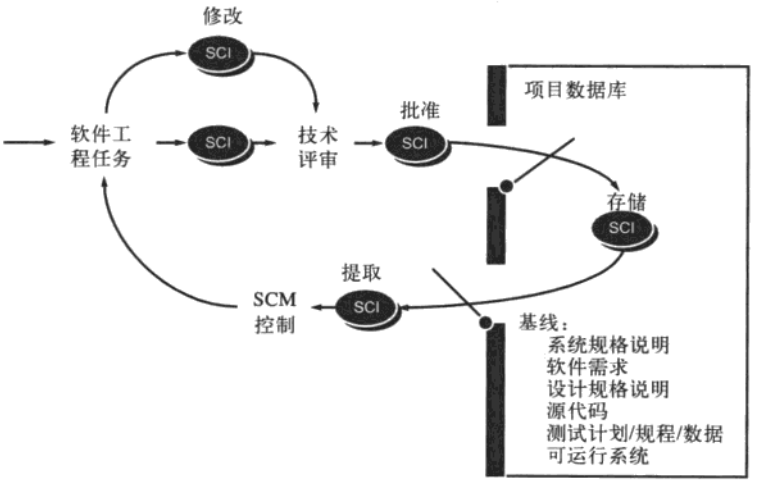


图22-1 基线化的SCI和项目数据库

确保在一个集中的、可控的地点维护项目数据库。

图22-1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22.2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22-1中的箭头说明了某个已成为基线的SCI的修改路径。

22.1.4 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做是一个SCI。再实际点，一个SCI可以是工作产品的全部或部分（例如，一份文档、一整套测试用例，或者是一个已命名的程序构件）。

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“固化”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22-2中，分别定义了配置对象DesignSpecification、DataModel、ComponentN、SourceCode和TestSpecification。各个对象之间的关系如图中箭头所示，单向箭头表示组合关系，即DataModel和ComponentN是DesignSpecification的组成部分。双向箭头说明对象之间的内在联系，如果SourceCode对象发生变更，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[⊖]。

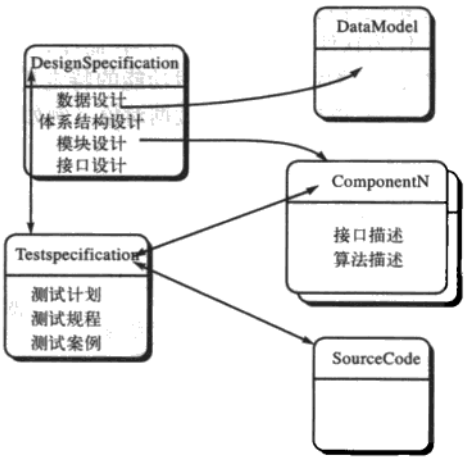


图22-2 配置对象

22.2 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）难以确定更改了哪些配置项，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“中心存储库”定义为“聚集中心或存储中心的任何事物或人”。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“聚集中心和存储中心”（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“物件”——一个作为软件工程信息聚集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。

22.2.1 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。通过保证数据完整性，信息共享和数据集成，它具有数据库管理系统的一般功能，此外，SCM中心存储库为软件工具的集成提供了中枢，它是软件过程流的核心。它能够使软件工程工作产品强制实施统一的结构和格式。

为了实现这些功能，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，以及将现有模型扩展以适应新需求时的容易程度如何等。

WebRef

可以通过网址www.oracle.com/technology/products/repository/index.html获得商业化的中心存储库。

22.2.2 一般特征和内容

中心存储库的特征和内容可以从两个方面来理解：中心存储库存储什么，以及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22-3所示。

⊖ 这些关系可以在数据库内部定义。数据库（中心存储库）的结构将在22.2节详细讨论。

一个健壮的中心存储库能够提供两种不同类型的服务：（1）期望从任何一个复杂的数据库管理系统得到相同的服务类型；（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）集成或直接支持过程管理功能；（2）支持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（例如，文本、图形、视频、音频）。

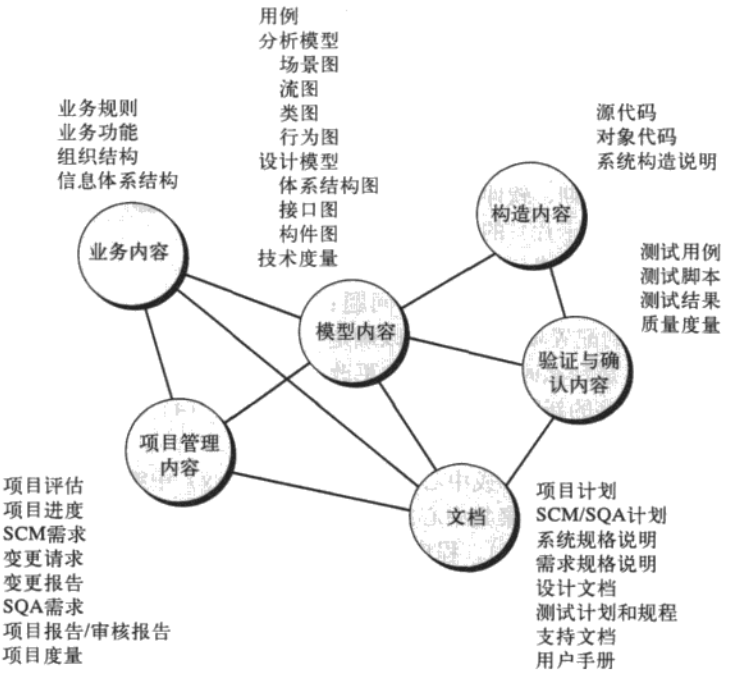


图22-3 中心存储库的内容

22.2.3 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：

KEY POINT

中心储存库必须能维护与软件的许多不同版本相关的SCI。更重要的是，它必须提供保证将这些SCI组装成一个具体版本配置的机制。

版本控制。随着项目进展，每个工作产品都可能有很多版本（22.3.2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。

中心存储库必须能控制各种类型的对象，包括文本、图形、位图、复杂文档及一些特殊对象，如屏幕、报告定义、目标文件、测试数据和结果等。在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。

依赖性跟踪和变更管理。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、应用系统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、设计元素与其他可交付工作产品之间的关系等。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。

保持追踪这些关系的能力对中心存储库中存储信息的完整性、基于中心存储库的可交付工

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、接口描述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。这种特殊的功能依赖于相关管理，并可以跟踪由特定需求规格说明产生的所有设计构件、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。配置管理设施能够跟踪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等信息。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。每当设计元素进行了修改，中心存储库触发机制有助于提示开发人员或创建审核信息条目（如变更理由）的工具。

22.3 SCM过程

"任何变更，甚至是为了更好的变更，也伴随着缺陷和不适。" —— Arnold Bennett

软件配置管理过程中定义的一系列任务具有4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）便于构造应用系统的不同版本；（4）在配置随时间演化时，确保能够保持软件质量。

能够取得上述4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？

- 软件团队应该如何标识软件配置的离散元素？
- 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？
- 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？

上述问题引导我们定义了5个SCM任务：标识、版本控制、变更控制、配置审核和报告，如图22-4所示。

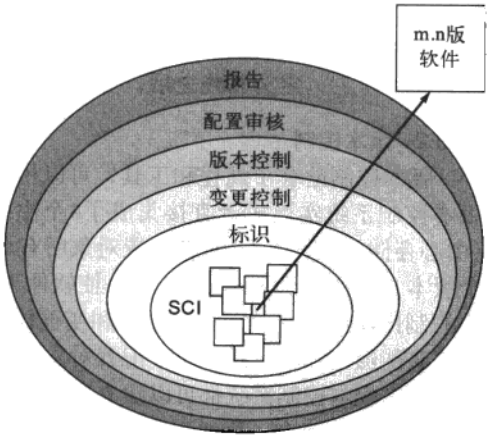


图22-4 SCM过程的层次

如图22-4所示，SCM任务可以看做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，在创建一个新的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，版本标识等），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个SCM过程层次。

22.3.1 软件配置中的对象标识

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[⊖]。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例。聚合对象是基本对象和其他聚合对象的集合。例如，在图27-2中的DesignSpecification就是一个聚合对象。在概念上，可将其视为一个已命名的（已标识的）指针列表，指向诸如ArchitectureModel和DataModel的聚合对象，以及诸如ComponentN和UMLClassDiagramN的基本对象。

针对配置对象建立的互联关系可用于评估变更的影响。

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（例如，模型元素、程序、数据）、项目标识符以及变更和（或）版本信息；资源是“对象提供的、处理的、参考的实体，或者是对象所需要的实体”[Cho89]。例如，数据类型、特定功能或者甚至是变量名都可以认为是对象资源。“实现”是一个指针，对于基本对象，该指针指向其文本单元；对于聚合对象，该指针为空。

标识配置对象时也可以考虑各个标识对象之间的关系。例如，使用下面的简单表示：

即使项目数据库提供了建立这些关系的功能，但建立起来比较耗时，而且很难保持最新。虽然这些关系对影响分析是很有用的，但是对于整个变更管理不是必需的。

类图<part-of> 需求模型

需求模型<part-of> 需求规格说明

可以为其创建SCI层次结构。

在很多情况下，在对象层次结构中，对象是跨层次分支而互相关联的。这些交叉的结构关系可以用下面的方式表示：

DataModel <interrelated> DataFlowModel

DataModel <interrelated> TestCaseClassM

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。

22.3.2 版本控制

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4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制作功能。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。

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

⊖ 提出聚合对象的概念是为了将其作为描述软件配置的完整版本而提出的一种机制[Gus89]。

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，该模板包含构件的层次结构，以及构造系统时构件的“创建次序”；（2）构造规则；（3）验证规则[⊖]。

在过去几十年中，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（CVS）

通过工具来实现版本控制对于实现高效变更管理是必须的。并发版本系统（Concurrent Version 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系统的功能有：（1）建立简单的中心存储库；（2）以一个命名文件来维护文件的所有版本，仅仅存储原始文件的各个渐进版本之间的差异；（3）为每位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修改。当开发者完成各自的工作后，由CVS合并处理所有的修改。

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（例如，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、变更报告和错误跟踪）。

虽然有上述局限性，CVS仍然“是一个优秀的、开源的、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”[CVS07]。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，且其开源理念使其适用于大部分流行的平台。

可以免费获得Windows、Mac OS、LINUX和UNIX环境下的CVS，更详细的信息请查看[CVS07]。

22.3.3 变更控制

“改进的艺术是在变更中保持有序，并且在有序中保持变更。”
——Alfred North Whitehead

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问题，太少的变更控制又会给我们带来其他问题。

KEY POINT

应该注意到，许多变更请求可能联合起来形成一个ECO，而ECO通常引起多个配置对象的变更。

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22-5所示，提交一个变更请求之后，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，以及变更的预计成本。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。

⊖ 为了评估构件的变更将怎样影响其他构件，可能需要查询该系统模型。

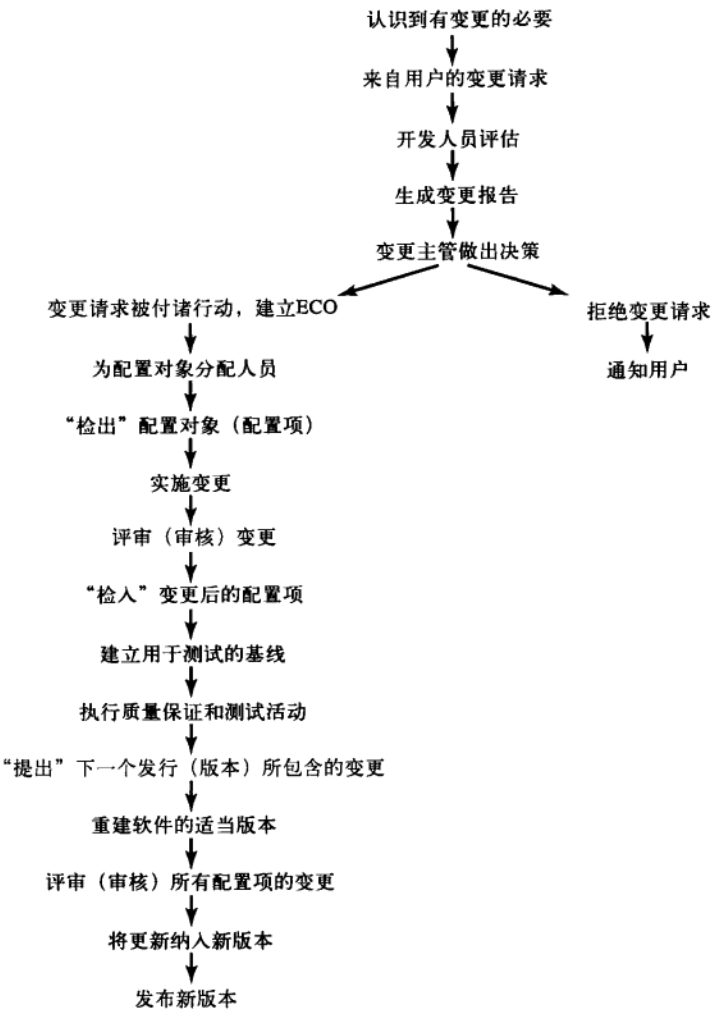


图22-5 变更控制过程

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(22.3.2节）构建该软件的下一个版本。

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。

选择比你认为需要的变更控制稍微多一些。很可能“较多”就是正好的。

读者可能开始对图22-5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。没有适当的防护措施，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）通过许多控制环节来帮助避免上面提到的这些问题。

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是适当的（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[⊖]。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（如果变更是“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22-5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扮演着主动的角色。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之外事物的影响。变更将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要CCA来处理。

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得要命……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，应该做三件事。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要遵守变更管理过程。行吗？

⊖ 也可能因为其他理由创建基线。例如，当“每日构建”基线创建后，在给定时间提交的所有构件都变成下一日工作的基线。

22.3.4 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）技术评审；（2）软件配置审核。

技术评审（在第15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，或是否具有潜在的副作用。除了那些非常微不足道的变更之外，应该对所有变更进行技术评审。

作为技术评审的补充，软件配置审核针对在评审期间通常不被考虑的特征对配置对象进行评估。软件配置审核要解决以下问题：

配置审核期间需要关注的主要问题是什么？

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？配置对象的属性反映出该变更了吗？
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，这些审核问题是作为技术评审的一部分来询问的。但是，当SCM是正式活动时，配置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，且与所构造的版本是一致的。

22.3.5 状态报告

ADVICE

为每个配置对象创建一个“须知”列表，并要实时更新。当变更发生时，保证列表上的每个人都被通知到。

配置状态报告（有时称为状态账目）是一项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）会影响到其他哪些事情？

配置状态报告（Configuration Status Reporting，CSR）的信息流如图22-5所示，每当赋予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22.3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：[⊖]

CCC/Harvest，由Computer Associates（www.cai.com）发行，是一个支持多种平台的SCM系统。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

ClearCase，由Rational（www-306.ibm.com/software/awdtools/clearcase/index.html）开发，提供SCM的一系列功能。

Serena ChangeMan ZMF,由Serena（www.serena.com/US/products/zmf/index.aspx）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。

商用SCM工具与环境的综合列表可以在www.cmtoday.com/yp/commercial.html找到。

22.4 WebApp配置管理

在本书的前面部分，我们讨论了Web应用系统的特殊属性，以及用来开发Web应用系统的特殊化方法（称为Web工程方法[⊖]）。WebApp与传统软件的主要区别就是无处不在的变更。

不受控制的变更对一个Web应用系统有什么影响？

WebApp的开发者通常采用迭代、增量过程模型，用到了很多敏捷软件开发（第3章）的原理。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的时间内开发出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WebApp的敏捷开发中，变更是有所不同的。

如果你是WebApp团队中的一员，你一定会与变更打交道，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WebApp项目的特定要求。

22.4.1 WebApp配置管理的主要问题

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WebApp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），那么将导致以下问题：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。

本章已介绍的软件配置管理（SCM）一般策略是适用的，但必须对这些策略和工具进行修改，以适应WebApp的独特环境。在制定WebApp配置管理策略时应该考虑4个问题[Dar99]：内容、人员、可伸缩性和策略。

内容。典型的WebApp包括很多内容——文本、图形、小型程序（applet）、脚本、音频/视频文件、表单、动态页面元素、表格、流数据等。难点在于如何将这些内容组织为一组合理的配置对象（22.1.4节），然后再为这些对象建立合适的配置控制机制。一种方法就是使用传统的数据建模技术（第6章）对WebApp内容进行建模，并为每个对象附上特殊的属性集。为了建立有效的SCM方法，就需要每个对象的动态/静态特性和预期寿命（如临时的、特定时长或

⊖ 文献[Pre08]有关于Web工程方法的综合介绍。

永久对象）等属性。例如，如果内容项每小时变更一次，则是临时的。与应用到表单构件这种永久对象的控制机制相比，这种项的控制机制是不同的（不太正式）。

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，最终导致应用系统的增长和变更处于不受控状态。

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，显然一个简单的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。

策略。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Web工程相关的策略：

如何确定负责WebApp配置管理的人员？

- 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。

WebApp配置管理会持续演化（例如，[Ngu06]）。传统的SCM过程可能过于繁琐，但是，在过去几年中，为Web工程专业化设计的新一代内容管理工具已经出现了。这些工具建立了获取现有信息（来自于各种WebApp对象）和管理对象变更的过程，该过程是以一种使最终用户可见的方式构建，然后在客户端环境中显示。

22.4.2 WebApp的配置对象

WebApp包括很多配置对象：内容对象（例如，文本、图形、图像、视频、音频）、功能构件（例如，脚本、小型应用程序）和接口对象（例如，COM或CORBA）。可以按任何方式来标识WebApp对象（指定文件名），只要适用于组织就可以。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，避免使用大小写混合的或全部大写的名称，也应避免使用下划线。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）。

WebApp的所有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）是由美学风格和为WebApp所建立的设计规则决定的。内容结构定义内容体系结构，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 [Bio04]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”。

22.4.3 内容管理

内容管理是解决当今信息风暴的一剂良药。
——Bob Boiko

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）。

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”创建Web页面。也就是说，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（例如，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。比较有代表性的CMS如图22-6所示。

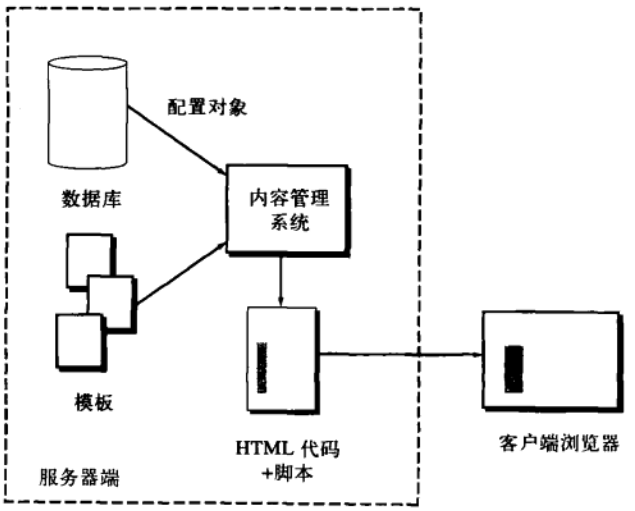


图22-6 内容管理系统

KEY POINT
收集子系统包含以下活动：内容创建、获取，并（或）将其转换为能在客户端表示的形式。

一般来说，CMS通过调用3个集成的子系统为最终用户“配置”内容，这3个子系统是：收集子系统、管理子系统和发布子系统[Boi04]。

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）将内容转变成标记语言（例如，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。

内容创建和获取（经常称为创作）通常是和其他Web应用系统的开发活动并行出现的，并且经常是由非技术开发人员处理。该活动结合了创新和研究的元素，并且通过工具支持使得内容作者可以在Web应用内部以标准化使用的方式刻画内容。

KEY POINT
管理子系统实现了一个针对所有内容的中心存储库。配置管理在该子系统内执行。

一旦有了内容，就要对其进行转换以满足CMS的需求。这就意味着从原始内容中剥离不必要的信息（例如，冗余的图形表示）、改变内容格式以满足该CMS的需求、并将结果映射到一个能够管理和发布的信息结构中。

管理子系统。收集到内容之后，就必须将其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（例如，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：查找特定的内容对象（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、变更审核和报告的功能元素和相关工作流。

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。

KEY POINT
发布子系统从中心存储库抽取内容，并发送其到客户端浏览器。

发布子系统。可以从中心存储库中提取内容，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。每个模板对应一个功能，每个功能都可以使用下面3个元素（构件）之一来构造一次发布[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——访问外部的企业信息基础设施，如企业数据或“内部”应用系统。

包含以上3个子系统的内容管理系统可以适用于大多数的WebApp项目。但是，CMS的基本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简单的文件管理系统，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功能，可以获得历史上的早期内容版本。

代表性工具：[⊖]

Vignette Content Management，由Vignette（www.vignette.com/us/Products）开发，是一套企业内容管理工具。

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。

有关SCM和针对Web工程的内容管理工具的其他信息见下列Web站点：

Web Developers'Virtual Encyclopedia（www.wdlv.com）。

WebDeveloper（www.webdeveloper.com）。

Developer Shed（www.devshed.com）。

Webknowhow.net（www. webknowhow.net）。

WebReference（www.webreference.com）。

22.4.4 变更管理

传统软件变更控制的工作过程（22.3.3节）对于WebApp开发来说通常太冗长了。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的顺序是不

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

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。可以将每个变更归为以下4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（例如，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——使一类或多类用户能够立即注意到的重要设计变更（例如，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22-7中所示的算法来处理。

在图27-7中，I类和II类变更可以看做是非正式的，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评审和文档。如果需要做大量修改，就必须做进一步评估和计划编制工作。

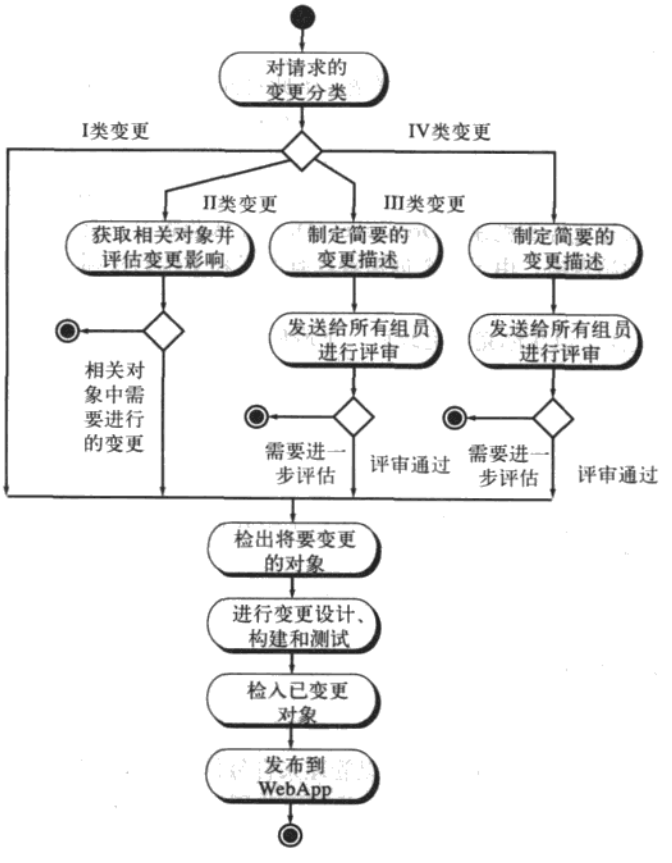


图22-7 对WebApp的变更管理

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要估算）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，是由所有的利益相关者进行评审的。

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工程师和内容开发者也可以使用，以控制WebApp变更。它们支持自动的检入检出、版本控制和恢复、变更报告和其他SCM功能。

代表性工具：[⊖]

ChangeMan WCM，由Serena（www.serena.com）开发，是一套变更管理工具，提供所有SCM功能。

ClearCase，由Rational（www-306.ibm.com/softwear/rational/sw-atoz/indexC.html）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，是一种SCM工具，能够与所选择的开发环境集成在一起。

22.4.5 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）正处于部署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。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. 每位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）可以管理来自每位Web开发者工作目录的所有检入和检出操作。当中心存储库发生变更时，该工具也可以为所有相关部分提供自动的电子邮件更新。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

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有效报告的一部分。

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。

22.4.6 审核和报告

在敏捷开发中，不再强调Web工程工作中的审核和报告功能[⊖]，但不能两者都被忽视。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对象时，还可以自动发送电子邮件通知（发给那些感兴趣的开发者和利益相关者）。

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标准实施指南

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

美国军用标准 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 航空软件开发准则

⊖ 这是变更的开始。逐渐强调将SCM作为WebApp安全的一个元素[Sar06]。通过提供跟踪和报告每个WebApp对象所做的每个变更的机制，变更管理工具能提供有价值的保护，以防止恶意的变更。

ESA PSS-05-09 软件配置管理指南
AECL CE-1001-STD rev.1 安全关键软件的软件工程标准
DOE SCM 检查清单 http:// cio.doe.gov/ITReform/sqse/download/cmcklst.doc
BS-6488 英国标准，基于计算机系统的配置管理
最佳实践——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补充信息，见www.quality.org/config/cm-guide.html。

22.5 小结

软件配置管理是应用于整个软件过程的普适性活动。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。应该将所有的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。

一旦开发的配置对象通过了评审，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。

变更控制是一种过程活动，它能够在对配置对象进行变更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，对将要变更的SCI进行可控制的更新。

配置审核是一种SQA活动，它有助于确保进行变更时仍能维护质量。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任务应该尽可能简化，而且必须能够达到内容管理的特殊规定。

习题与思考题

22.1 为什么“系统工程第一定律”会成立？变更的主要理由有4个，对每个理由都举出几个特例。
22.2 实现高效SCM系统必需的4个元素是什么？简要介绍每个元素。
22.3 用你自己的话谈谈定义基线的理由。
22.4 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？
22.5 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、更新和变更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？
22.6 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。
22.7 关系〈part-of〉和〈interrelated〉表示配置对象之间的简单关系，描述5种可能在SCM中心存储库中用到的其他关系。

22.8 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。

22.9 设计一个用在配置审核中的检查表。

22.10 SCM审核和技术评审有什么区别？它们的作用可以归纳为一种评审吗？请说明正反两方面的观点。

22.11 简要描述传统软件的SCM与WebApp的SCM之间有何不同。

22.12 什么是内容管理？通过Web去研究内容管理工具的特性，并给出简要的总结。

推荐读物与阅读信息

关于SCM的最新资料有Leon（《Software Configuration Management Handbook》，second edition, Artech House Publishers, 2005）、Maraia（《The Build Master: Microsoft Software Configuration Management Best Practice》，Addison Wiley, 2005）、Keyes（《Software Configuration Management》，Auerbach, 2004）、以及Hass（《Configuration Management Principles and Practice》，Addison-Wesley, 2002）。每本书都非常详细地描述了整个SCM过程。Maraia（《Software Configuration Management Implementation Roadmap》，Wiley, 2004）提出了独特的指南，用以帮助那些在组织内必须实施SCM的工程人员。Lyon（《Practical CM》，Raven Publishing, 2003, www.configuration.org）为CM专业人员写了一本内容全面的指导书，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）。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。

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。Bays（《Software Release Methodology》，Prentice-Hall, 1999）重点讲述成功产品的发布机制，这也是对有效SCM的重要补充。

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。White的书（《The Content Management Handbook》，Curtin University Books, 2005）、Jenkins和他的同事（《Enterprise Content Management Methods》，Open Text Cooperation, 2005）、Bioko[Bio04]、Mauthe和Thomas（《Professional Content Management Systems》，Wiley, 2004）、Addey和他的同事（《Content Management Systems》，Glasshaus, 2003）、Rockly（《Managing Enterprise Content》，New Riders, 2002）、Hackos（《Content Management for Dynamic Web Delivery》，Wiley, 2002）以及Nakano（《Web Content Management》，Addison-Wesley, 2001）介绍了这方面的有价值的内容管理方法。

除了软件配置管理的一般主题外，Lim和他的同事（《Enhance Microsoft Content Management Server with ASP.Net 2.0》，Packt Publisher, 2006）、Ferguson（《Creating Content Management Systems in Java》，Charles River Media, 2006）、IBM Redbook（《IBM Workplace Web Content Management for Portal 5.1》和《IBM Workplace Web Content Management 2.5》，Vivante, 2006）、Fritz和他的同事（《Typo3: Enterprise Content Management》，Packt Publishing, 2005）、Forta（《Reality ColdFusion: Intranets and Content Management》，Pearson Education, 2002）也描述了在使用具体工具和语言环境下的内容管理。

因特网上有大量关于软件配置管理和内容管理的信息资源。有关软件配置管理的最新WWW资源可以在SEPA网站www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第23章

Software Engineering：A Practitioner's Approach (Seventh Edition)

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。产品度量关注软件工程工作产品具体的、可测量的属性，帮助软件工程师认识他们所开发的软件的设计和构造。

人员：软件工程师利用产品度量创建高质量的软件。

重要性：对计算机软件开发而言，定性要素总是存在的。但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需求模型、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：使用从需求模型和设计模型、源代码及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。只定义几个度量，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
目标/问题/度量
指标
测度
测量
测量原则
度量
　度量属性
　体系结构设计
　面向类
　面向对象设计
　需求模型
　源代码
　测试
　用户接口设计
　WebApp设计

测量是任何工程过程的一个关键环节。使用测量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。直接测量（例如，电压、质量、速度或温度）在软件世界是不常见的。由于软件测量与度量经常是间接得到的，因此存在公开争论。Fenton[Fen91]在阐述这个问题时说道：

> 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密……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。

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是不完善的，但产品度量为我们提供了一种基于一组明确规定的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，我们提出了产品开发时，能用于评估产品质量的测度（measures）。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：需求模型、设计模型及代码模型的功效，测试用例的有效性，以及待开发软件的总体质量。

23.1 产品度量框架

"一门科学与其测量工具一样成熟。"——Louis Pasteur

测度和测量有什么不同？

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，[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。

23.1.1 测度、度量和指标

尽管测量（measurement）、测度（measure）和度量（metrics）这3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范围、数量、维度、容量或大小提供量化的指标。而"测量"是确定测度的动作。度量在《软件工程术语的IEEE标准词汇表》[IEE93b]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。

一个指标就是提供了对过程、产品或项目深入理解的一个或一些度量。

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。

23.1.2 产品度量的挑战

"就像温度测量开始于一个指数……逐渐增长到复杂的级别、工具和技术。软件测量成熟也是如此。"——Shari Pfleeger

在过去40年中，许多研究人员试图开发出单一的度量值，能为软件复杂性提供全面的测量。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。

然而，仍有必要去测量和控制软件的复杂度。如果一个质量度量的单一值难以获取，那么可能应该开发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8章的其他属性）的测度。这些测量和由此产生的度量可用作需求模型和设计模型的独立指标。但是新问题再次出现，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

可避免地必须满足有冲突的目标。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了一致和客观的机制。

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。

WebRef

Horst Zuse汇编了关于产品度量的大量信息，网址是：irb.cs.tu-berlin.de/~zuse/。

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[⊖]。若要获得质量，则测量是必需的。

23.1.3 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]提出了能用以下5个活动描述的测量过程：

? 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。为获得对所表示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。

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。已经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[Let03b]：

ADVICE

实际上，目前使用的许多产品度量并不完全符合这些原则。但这并不意味着它们没有价值——只要小心使用，明白产品度量的目的是提供对产品的深入理解而不是严格的科学证明。

- 度量应该具有符合要求的数学特性。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。
- 如果度量代表一个软件特征，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。度量值也应该以同样的方式提高或降低。
- 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建议。

23.1.4 面向目标的软件测量

目标/问题/度量（Goal/Question/Metric，GQM）范型是由Basili和Weiss[Bas84] 开发的，这

⊖ 虽然文献中对具体度量的评论是常见的，但许多评论关注深奥的问题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种系统、客观的方法，来获得对其工作的深入理解，并最终提高产品的质量。

WebRef

GQM的有益讨论可在www.thedacs.com/GoldPractices/practic-es/gqma.html找到。

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）确定一些良好定义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}…的角度，关于…{所考虑的活动或属性}…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：

在未来三年要对产品进行改进的环境下，从软件工程师完成工作的角度，关于SafeHome具有较强可扩展性能力方面，为了评估体系结构构件，分析SafeHome软件体系结构。

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（或其他利益相关者）确定是否已达到测量目标。可能会问到的问题如下：

问题1：体系结构构件是否将以功能与相关数据分开的方式描述？

问题2：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8章）指标可能对回答问题1有用。本章后面讨论的度量有助于理解问题2。在每种情况下，所选择（或派生）的度量应该与23.1.3节所讨论的测量原则和23.1.5节所讨论的测量属性相符。

23.1.5 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有专业人员能够理解，另一些则违背了高质量软件的直观概念。

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：

? 怎样评估已提出的软件度量的质量？

- 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。
- 在经验上和直观上有说服力。度量应该满足工程师直觉上对所考虑的产品属性的概念（例如，测度模型内聚的度量在价值方面和内聚等级方面的增长应保持一致）。
- 一致的和客观的。度量产生的结果应该总是无歧义的。独立的第三方应该能够利用软件中相同的信息导出相同的度量。
- 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。例如，在项目组的多个人使用多样的编程语言就会导致可疑的单位组合，这样就没有直观的说服力。
- 编程语言的独立性。度量应该基于需求模型、设计模型或程序结构本身，而不应该依赖于变化多样的编程语言的语法或语义。
- 有效的、高质量反馈机制。也就是，度量应该提供信息以产生高质量的最终产品。

ADVICE

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。

⊖ van Solingen和Berghout [Sol99] 建议：目标几乎总是“理解、控制或改善”过程活动或产品属性。

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23.2.1节讨论）——一种软件交付功能的测量。有人提出[⊖]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某个属性。

SAFEHOME

关于产品度量的辩论

[场景] Vinod的工作间。

[人物] Vinod、Jamie、Ed, SafeHome软件工程团队的成员，他们正在进行构件级设计和测试用例设计。

[对话]

Vinod：Doug[Doug Miller，软件工程经理]告诉我，我们都应该使用产品度量，但他有点含糊，又说他不强迫这件事情……是否采用由我们自己决定。

Jamie：那好。我们没有时间做测量工作，我们都在为维持进度而奋战。

Ed：我同意Jamie的观点。我们的确面临这个问题，这里……没时间。

Vinod：是的，我知道，但是使用产品度量可能会有一些好处。

Jamie：我并不怀疑这个问题，Vinod，它是时间问题，我没有任何剩余时间来做产品度量。

Vinod：但是，如果测量能节省你的时间，那又怎么样呢？

Ed：你错了，就像Jamie所说的那样，需要时间……

Vinod：不，等等……如果它能节省我们的时间，那又怎么样呢？

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和Jamie淡漠地点点头）

23.2 需求模型的度量

软件工程的技术工作开始于需求模型的创建阶段。在这个阶段，产生需求并建立设计的基础。因此，非常希望产品度量能提供对分析模型质量的深入理解。

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来研究需求模型。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。

⊖ 可以提出同样有力的辩论。这是软件度量的本质。

23.2.1 基于功能的度量

WebRef

关于功能点的更多有用信息可在www.ifpug.org和www.functionpoints.com找到。

功能点（FP）度量可用做测量系统交付功能的有效手段[⊖]。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）预告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[⊖]：

外部输入数（EI）。每个外部输入源于一个用户，或从另一个应用系统中传送过来；它提供了面向不同应用系统的数据或控制信息。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区分开来。

外部输出数（EO）。每个外部输出从应用系统中导出，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（EQ）。一个外部查询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（ILF）。每个内部逻辑文件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（EIF）。每个外部接口文件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信息。

一旦收集了这些数据，就完成了图23-1所示的表，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、中等还是复杂。不过，复杂度的确定毕竟有一定的主观性。

利用下面的关系式计算功能点（FP）：

$$FP = 总计 \times [0.65 + 0.01 \times \Sigma(F_i)] \tag{23-1}$$

其中，“总计”是从图23-1中得到的所有FP项的总数。

			加权因子			
信息域值	计数		简单	中等	复杂	
外部输入(EI)		×	3	4	6	=
外部输出(EO)		×	4	5	7	=
外部查询(EQ)		×	3	4	6	=
内部逻辑文件(ILF)		×	7	10	15	=
外部接口文件(EIF)		×	5	7	10	=
总计						

图23-1 计算功能点

F_i (i =1～14) 是值调整因子（VAF），它基于对下列问题的回答来确定[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？

⊖ 关于FP度量的书、论文、和文章数以百计。一些有价值的文献可以在[IFP05]找到。

⊖ 实际上，信息域值的定义及其计算的方式有点复杂。有关的详细内容，有兴趣的读者可以参看[IFP01]。

KEY POINT
值调整因子用于提供问题复杂度的指标。

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在线更新吗？
9. 输入、输出、文件或查询复杂吗？
10. 内部处理复杂吗？
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？

WebRef
一个在线的FP计算器可在irb.cs.unimagdeburg.de/ sw-eng/us/java/fp/找到。

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。公式（23-1）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23-2所示。图中描述了SafeHome软件的一个功能的数据流图（第7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提示信息且发送合适的控制信号到安全系统的不同构件。

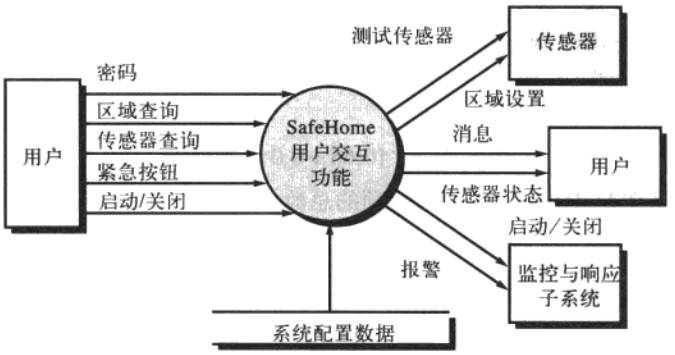


图23-2 SafeHome软件的数据流模型

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3个外部输入——密码、紧急按钮和启动/关闭；以及两个外部查询——区域查询与传感器查询。同时，给出了一个内部逻辑文件——系统配置文件；两个外部输出——消息与传感器状态；4个外部接口文件——测试传感器、区域设置、启动/关闭和报警。这些数据及相应的复杂度在图23-3中给出。

不仅要沉思“什么是可以应用的新度量……我们也应该问自己较基本的问题：我们将如何处理度量？”——Michael Mah和Larry Putnam

图23-3中显示的“总计”必须用公式（23-1）调整。对于这个例子，假设$\Sigma(F_i)$为46（一个中等复杂的产品）。因此，

$$FP = 50 \times [0.65 + (0.01 \times 46)] = 56$$

基于由需求模型中导出的FP值，项目组可以估算SafeHome软件用户交互功能的整体实现规模。假设过去的数据表明一个FP转换为60行代码（使用面向对象语言）和每人月生产12个功能点。这些历史数据为项目经理提供重要的

计划信息，且这些信息是基于需求模型而不是基于初步估算。进一步假设过去的项目中需求和设计评审期间平均每个功能点有3个错误，单元测试和集成测试期间平均每个功能点有4个错误。这些数据最终将有助于软件工程师评估他们的评审和测试活动的完整性。

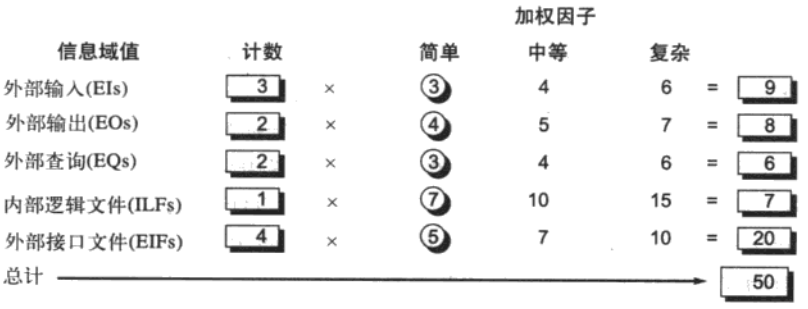


图23-3 计算功能点

Uemura与他的同事[Uem99]提出，功能点也可以从UML类图和时序图中计算出来。进一步的细节参见[Uem99]。

23.2.2 规格说明质量的度量

Davis与其同事[Dav93]提出了用于评估需求模型和相应的需求规格说明质量的一系列特征：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、可达性、简洁性、可跟踪性、可修改性、精确性和可复用性。另外，他们注意到，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，另外还应是稳定的、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为功能需求的数量，n_{nf}为非功能（如性能）需求的数量。

KEY POINT 通过测量规格说明的特征，就可能获得量化的确定性和完整性。

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项需求解释的一致性：

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

其中，n_{ui}是所有评审者都有相同解释的需求数目。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计算下面的比率来确定：

$$Q_2 = n_u / (n_i \times n_s)$$

其中，n_u为独特功能需求的数目，n_i是由规格说明明确定义或隐含的输入（激励）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个数，n_{nv}是尚未确认的需求个数。

测量可测量的东西，且使不可测量的东西可测量。——Galileo

23.3 设计模型的度量

KEY POINT
度量能提供对结构数据和与体系结构设计相关的系统复杂度的深入理解。

很难想象一架新飞机、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，基于软件的复杂系统设计通常是在没有测量的情况下进行的。更具有讽刺性的事实是，软件的设计度量是可以得到的，但绝大多数软件工程师却一直不知道它们的存在。

与所有其他软件度量一样，计算机软件的设计度量并不完美。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，没有测量的设计是难以接受的选择。

本节将介绍一些非常通用的计算机软件的设计度量。每个度量都提供了如何改进设计的理解，而且所有的度量都有助于改进设计使其具有更高的质量。

23.3.1 体系结构设计度量

体系结构设计度量侧重于程序体系结构（第9章）的特征，它强调体系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来讲是“黑盒的”，它并不需要特定软件构件的内部运作知识。

Card与Glass[Car90] 定义了3种软件设计复杂度测量：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-返回体系结构），模块i的结构复杂度的定义方式如下：

$$S(i) = f^2_{out}(i)$$

其中，$f_{out}(i)$ 是模块i的扇出数[⊖]。

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：

$$D(i) = v(i) / (f_{out}(i) + 1)$$

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：

$$C(i) = S(i) + D(i)$$

系统的总体体系结构复杂度会随着每个复杂度值的增加而增加。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23-4中调用-返回体系结构，可以定义下述度量：

规模 = $n + a$

其中，n为结点数，a为弧数。对于图23-4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；

弧与结点的比例定义为$r = a / n$，它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23-4中的体系结构，$r = 18/17 = 1.06$。

⊖ 扇出数定义为直接隶属于模块i的模块数量。也就是，直接被模块i调用的模块数量。

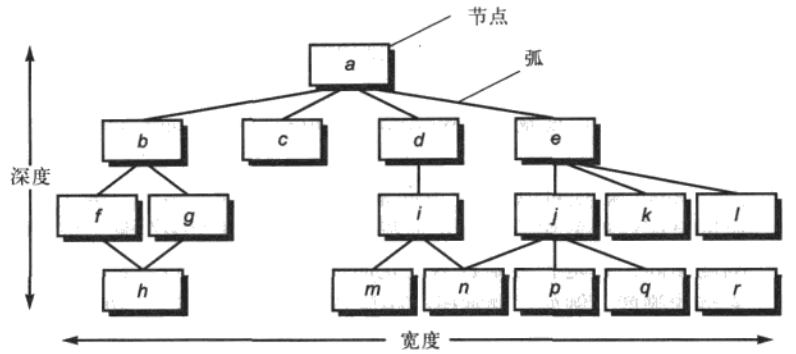


图23-4 形态度量

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。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＝模块数，其正确功能依赖于数据源输入或产生的数据用于其他地方（通常控制模块不计入S_2之内）

S_3＝模块数，其正确功能依赖于前面的处理

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＝具有单个入口和单出口（异常处理不看做多重出口）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：$D_6=1-(S_7/S_1)$

“测量可以看做是一条绕行的路，它是必不可少的，因为[没有量化的支持]多数人不能做出明确和客观的决策。”——Horst Zuse

当这些中间值确定后，DSQI用下列方式来计算：

$$\mathrm{DSQI}=\sum w_i D_i$$

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。

23.3.2 面向对象设计度量

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客户需求。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：

? 当评估面向对象设计时都测量哪些特征？

规模。规模可以从4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通过某个面向对象应用系统交付给客户的价值。

"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

复杂性。与规模一样，有许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。面向对象设计成分间的物理连接（例如，类间的协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。

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]。充分性是从当前应用的角度来与抽象比较。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、明确定义的目标。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多次提到，当修改需求或应用系统的其他部分时，可能会发生设计变更，从而导致设计构件的适应性修改。面向对象设计构件的易变性是要测量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。另外，还要探讨适用于项目管理和测试的度量。

23.3.3 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。类封装数据和操作数据的功能。通常的情况是子类（有时称为子女）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[⊖]。

⊖ 应该注意到本章讨论的一些度量的有效性问题当前在一些技术文献中还在争论。那些支持测量理论的人们要求某种程度的形式化，而某些OO度量无法提供。然而，所关注的度量能为软件工程师提供有用的理解也是可接受的。

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6个面向对象系统的基于类的设计度量，通常称为是CK度量套件[⊖]。

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$c_1, c_2, \cdots, c_n$，所选择的特定复杂度度量（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。

$$\mathrm{WMC} = \sum C_i$$

其中，$i=1, \cdots, n$。

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，它可能越来越特定于应用，由此限制了潜在的复用。因此，WMC应保持合理的低值。

虽然对类中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]。

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23-5，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。

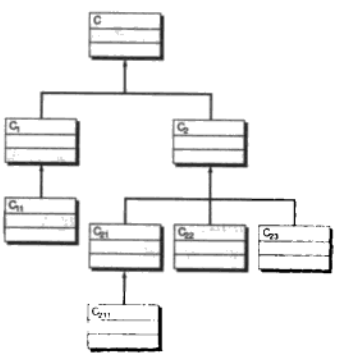


图23-5 类层次

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23-5所示，类C_2有3个子女——子类C_{21}、C_{22}和C_{23}。随着子女数量的增长，复用也增加。但是，当NOC增大时，如果有些子女不是父类的合适成员，父类所表示的抽象可能削弱。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。

ADVICE
耦合和内聚的概念可以应用到传统软件和面向对象软件。保持类之间的低耦合、以及类和操作的高内聚。

对象类之间的耦合（Coupling Between Object classes, CBO）。CRC模型（第6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数量[⊜]。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性的一般性指导原则是一致的。

对类的响应（Response For a Class, RFC）。类的响应集是“该类的某对象接收到消息，做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19章）增加，测试的工作量也随之增加。同样，当RFC增大时，类的整体设计复杂性也随之增加。

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数量[⊝]。若没有方法访问相同的属性，则LCOM=0。为说明LCOM≠

⊖ Chidamber、Darcy和Kemerer使用术语方法（method）而不是操作（operation）。本节反映了这些术语的使用。

⊜ CRC索引卡是手工开发的。在可靠地确定CBO之前，必须评估其完整性和一致性。

⊝ 该形式化定义有点复杂。详见[Chi94]。

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，也就是使LCOM保持低值[⊖]。

SAFEHOME

CK度量的应用

[场景] Vinod的工作间。

[人物] Vinod、Jamie、Shakira和Ed，SafeHome软件工程团队的成员，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度量集的描述了吗？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。

"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

23.3.4 面向类的度量——MOOD度量集

Harrison、Counsell和Nithi[Har98b]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：

$$\mathrm{MIF}=\sum M_i(C_i)\,/\,\sum M_a(C_i)$$

这里是对i从1到TC求和。TC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，

⊖ 在一些情况下，LCOM测度提供了有用理解，但是在其他一些情形下可能具有误导性。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。

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

其中，$M_a(C_i)$ 为可在C_i关联中被调用的方法数量；

$M_d(C_i)$ 为类C_i中声明的方法数量；

$M_i(C_i)$ 为类C_i中继承的（未被覆写的）方法数量。

MIF的值[属性继承因子（Attribute Inheritance Factor，AIF）以类似的方式定义]指示了继承对面向对象软件的影响。

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：

$$CF = \sum_i \sum_j is\_client(C_i, C_j)/(T_c^2 - T_c)$$

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$C_c \neq C_s$；否则，$is\_client = 0$。

尽管许多因素影响软件复杂性、可理解性和可维护性，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，可理解性和可维护性和潜在的可复用性将受到损害。

Harrison与他的同事[Har98b]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。

23.3.5 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4类：规模、继承、内部和外部。对于面向对象设计类，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（23.3.3节）与面向代码的问题。外部度量检查耦合性与复用。Lorenz与Kidd提出的一个度量的例子如下：

在分析模型的评审期间，CRC索引卡将提供CS期望值的合理指标。如果遇到一个类包含大量的职责，则考虑将其分解。

类的规模（Class Size，CS）。类的整体规模可以用下面的测度来确定：

- 封装在类中的操作总数（包括继承来的和私有的实例操作）；
- 封装在类中的属性总数（包括继承来的和私有的实例属性）。

Chidamber和Kemerer提出的WMC度量（23.3.3节）也是类规模的加权测量。正如前面提到的，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，继承的或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。规模的平均值越低，类在本系统中能被广泛复用的可能性越高。

23.3.6 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，并包括“3C”的测度——内聚性（cohesion）、耦合性（coupling）和复杂度（complexity）。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），就可以应用构件级设计度量。另外，它们也可以延迟到有源代码时才用。

内聚度量。Bieman和Ott[Bie94]定义了一组度量，为模块内聚提供了指示（第8章）。该度量用5个概念和测度来定义：

KEY POINT
可以计算构件的功能独立性（耦合和内聚）的测度，并用其评估设计的质量。

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定位的数据值。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。一个胶合记号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。这些度量的详细讨论见文献[Bie94]。

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3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_0 =输出数据参数的个数

c_0 =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

$$M = d_i + (a \times c_i) + d_0 + (b \times c_0) + g_d + (c \times g_c) + w + r$$

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，耦合度随着M值的增加而增加。

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18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。

环复杂度只是大量复杂性度量中的一个。

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]提出的，在第18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的研究工作是迄今为止最为全面的。

23.3.7 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，在语句数量和逻辑复杂性[Wil93]这两个方面，方法都倾向于小，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。Lorenz与Kidd[Lor94]提出的3个简单度量比较适当：

平均操作规模（average operation size，OS_{avg}）。可以通过对代码行计数或操作发送的消息数来确定规模。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。

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，对象间的协作越复杂。一般来讲，应该保持NP_{avg}的值尽可能的低。

23.3.8 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11章），但是，有关界面质量和易用性度量的信息却比较少。

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

Sears[Sea93]提出：布局恰当性（layout appropriateness，LA）是有价值的人机界面设计度量。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完成给定的任务，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变迁到另一个的"开销"，这些都影响着界面的恰当性。

Web页面度量研究[Ivo01]显示，布局元素的简单特征也对GUI设计的感知质量有重要的影响。在Web页面中包含的单词、链接、图形、颜色和字体（连同其他特征）的数量都会影响该页面的感知复杂度和质量。

值得指出的是，GUI设计的选择可以用度量（如LA）作指导，但终裁者应该是基于GUI原型的用户输入。Nielsen和Levy[Nie94]谈道："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。"

23.4 WebApp的设计度量

WebApps的有用的测度和度量集为如下问题提供了定量答案：

- 用户接口是否提高了可用性？
- WebApps的美学设计对于应用领域是否合适？是否能够得到客户的认可？
- 是否能够以最小的工作量提供最多的信息的方式来设计内容？
- 导航是否有效和直接？

其中许多度量都可应用于所有的用户界面，应该和23.3.8节中的度量结合考虑。

- WebApp体系结构设计是否适应WebApp用户的特定目标和目的、内容结构和功能结构以及有效地使用系统所需的导航流程？
- 构件设计是否减少了流程复杂度且提高了正确性、可靠性和性能？

因为现在还没有能提供量化答案的有效度量集，所有这些问题只能定性地处理。

接下来，我们提供在文献中已经提出的关于WebApp度量的代表性例子。值得注意的是，这些度量还没有被验证，因此应该审慎地使用。

界面度量。对于WebApp，可以考虑下面的界面度量：

建议的度量	描　述
布局恰当性	见23.3.8节
布局复杂度	界面定义的独特区域的数量⊖
布局区域复杂度	每个区域不同链接的平均数
识别复杂度	在进行导航或数据输入之前用户必须查看的不同项的平均数
识别时间	用户为给定任务选择恰当活动的平均时间（单位：秒）
输入工作量	具体功能所需的敲键平均数
鼠标选中工作量	每个功能鼠标选中的平均数
选择复杂度	每个页面能选择链接的平均数
内容获取时间	每个Web页面文本单词的平均数
记忆负担	为实现特定目的用户所必须记住的不同数据项的平均数

美学（图形设计）度量。本质上，美学设计依赖于定性的判断，通常不遵守测量和度量的规则。然而，Ivory及其同事[Ivo01]提出的一组测度可能在评估美学设计的影响时有用。该度量集如下：

建议的度量	描　述
单词个数	一个页面中出现的单词总数
主体（Body）文本百分比	主体文本与显示文本（即Header部分）的单词百分比
强调主体文本%	强调（如粗体、大写）的主体文本的比例
文本定位数量	左对齐文本位置变动次数
文本群数量	用颜色、边框、破折号或列表等强调的文本域
链接数量	页中的总链接数
页面大小	页面总字节数（包括元素、图形、样式表）
图形百分比	页面图形的字节数量的百分比
图形数	页中图形的个数（不包含脚本、applet和对象中的图形）
颜色数	采用的颜色总数
字体数	采用的字体总数（即字体类型+大小+粗体+斜体）

内容度量。该类度量强调内容复杂度和内容对象的聚集。相关内容可以参考文献[Men01]。这些度量包括：

建议的度量	描　述
页面等待	以不同的连接速度下载页面所需的平均时间
页面复杂度	页面使用的不同类型的媒体的平均数量，不包括文本
图形复杂度	每页图形的平均数

⊖ 一个不同区域指的是一个在布局显示范围内的区域，能够完成一些具体的相关功能（例如，菜单栏、静态图形显示、内容域、动画显示）。

（续）

建议的度量	描 述
音频复杂度	每页音频的平均数
视频复杂度	每页视频的平均数
动画复杂度	每页动画的平均数
扫描图形复杂度	每页扫描图形的平均数

导航度量。该类度量处理导航流程的复杂度[Men01]。通常来说，它们只可应用于静态的Web应用，不包含动态产生的链接和页面。

建议的度量	描 述
页面链接复杂度	每页链接的数量
连通性	内部链接的总数，不包括动态产生的链接
连通密度	连通性除以页面个数

利用建议度量的子集可能会得到一些经验关系。根据这些经验关系，WebApp开发团队可以基于复杂度的预测估计，评估技术质量和预计工作量。该领域的工作还有待将来完成。

SOFTWARE TOOLS

WebApp的技术度量

目的：帮助Web工程人员开发有意义的WebApp度量，这些度量能够衡量应用系统的总体质量。

机制：工具机制多样。

代表性工具：[⊖]

Netmechnic Tools，由Netmechnic（www.netmechnic.com）开发，是一组工具的集合。有助于提高网站的性能，关注具体的实现问题。

NIST Web Metrics Testbed，由国家标准技术研究所（zing.ncsl.nist.gov/WebTools/）开发，包含下面一些可下载的有用工具。

Web Static Analyzer Tool (WebSAT)——根据典型的可用性原则检查HTML Web页面。

Web Category Analysis Tool (WebCAT)——使工程师构造和管理Web分类分析。

Web Varuable 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——通过网站实现用户导航路径的3D可视化。

TreeDec——将导航辅助加到网站的页面中。

23.5 源代码度量

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”[⊜]。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，给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：

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。

⊜ 应该注意到Healstead的“定律”已经引起了很多争议。许多人认为其基本理论有缺点。然而，已经进行了对已选编程语言的实验验证（例如，[Fel89]）。

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

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

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（例如，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，它表示说明一个程序所需要的信息量（以“位”计数）。

ADVICE

操作符包含所有的控制流结构、条件和数学操作。操作数包含所有的程序变量和常量。

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：

$$L = \frac{2}{n_1} \times \frac{n_2}{N_2}$$

Halstead的工作有必要通过实验验证，而且大量的研究已针对软件科学进行。此项工作的讨论超出了本书的范围，进一步信息参见[Zus90]、[Fen91]和[Zus97]。

23.6 测试的度量

KEY POINT

测试度量分成两大类：（1）预计在不同的测试级别可能需要的测试的数量；（2）强调针对给定构件的测试覆盖。

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。

体系结构设计度量提供了与集成测试相关的难易信息（23.3节）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18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。

23.6.1 用于测试的Halstead度量

从Halstead测度（23.5节）导出的度量也可用来估算测试工作量。利用程序信息量V的定义和程序层次PL，可以计算Halstead工作量e：

$$PL = 1/[(n_1/2) \times (N_2/n_2)] \quad (23\text{-}2a)$$

$$e = V/PL \quad (23\text{-}2b)$$

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：

$$测试工作量百分比(k) = e(k) / \sum e(i) \qquad (23\text{-}3)$$

其中，$e(k)$ 是利用公式（23-2）计算的模块k的测试工作量，公式（23-3）的分母之和是系统所有模块的Halstead工作量的总和。

23.6.2 面向对象测试的度量

在23.3节提到的面向对象设计度量为设计质量提供了一种指标，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标。Binder[Bin94b]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。

OO测试相当复杂。基于测量特征，度量可以帮助工程人员将测试资源锁定在值得"怀疑"的线程、场景和类的包。可以加以使用。

方法缺少内聚（lack of cohesion in methods, LCOM）[⊖]。LCOM的值越高，为保证方法不会产生副作用，需要测试的状态越多。

公有与保护属性的百分比（percent public and protected, PAP）。公有属性是从其他类继承的，因此对这些类是可见的。保护属性对子类的方法是可访问的。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[⊜]。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。

扇入（fan-in, FIN）。当用于面向对象环境时，在继承层次中的扇入是多继承的指示。FIN＞1指示类从多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）[⊜]。如第19章所讨论的，对每个子类，必须重新测试超类的方法。

23.7 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。

IEEE Std.982.1-1988[IEE93]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（基于产品每次发布所发生的变更）。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。

⊖ LCOM的描述参见23.3.3节。

⊜ 一些人支持这样的观点：设计中没有属性是共有的或私有的，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

⊜ NOC和DIT的描述参见23.3.3节。

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1.0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：[⊖]

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IBM（www-304.ibm.com/jct03001c/software/awdtools/developer/rose/）经销。它是一种综合的UML建模工具集，并结合了很多度量分析特性。

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。

23.8 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。度量为创建有效的需求模型、设计模型、可靠的代码和严格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和可计算的、有说服力的、一致和客观的。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。

需求模型的度量侧重于需求模型的3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间接的测度，提供了模块质量指示；用户界面设计度量为GUI的易用性提供指标。WebApp度量考虑了用户界面方面，也考虑到了WebApp的美学、内容和导航。

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。CK度量集定义了6个以类和类层次为重点的面向类的软件度量。该度量集也开发了度量，用于评估类间协作以及类中方法的聚合度。在面向

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

类级别，Lorenz和Kidd提出的度量以及MOOD度量套件可以作为CK度量集的扩展。

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，软件科学提供了多种度量，用于评估程序质量。

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。

习题与思考题

23.1 测量理论是与软件度量密切相关的高级主题。利用[Zus97]、[Fen91]、[Zus90]或基于Web的资源，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。

23.2 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？试着从日常生活中提出一些不符合23.1.5节定义的有效软件度量属性的测度或度量。

23.3 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个不同的遗留系统相连接（6 EIF）。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。

23.4 软件系统*X*有24个功能需求和14个非功能需求。需求的确定性及完备性是什么？

23.5 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个数据对象，每个对象平均有3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。

23.6 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度量。

23.7 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。

23.8 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。

23.9 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。

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。Lanza和他的同事（《Object-Oriented Metrics in Practice》，Springer, 2006）讨论了面向对象度量以及如何使用它们评估设计质量。Genero（《Metrics for Software Conceptual Models》，Imperial College Press, 2005）和Ejiogu（《Software Metrics》，BookSurge Publishing, 2005）提出了很多关于用例、UML模型和其他建模表示的技术度量。Hutcheson（《Software Testing Fundamentals: Methods and Metrics》，Wiley, 2003）提出了一组测试的度量。Kan（《Metrics and Models in Software Quality Engineering》，Addison-Wesley, 2nd ed., 2002）、Fenton和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和Glass[Car90]、Zuse[Zus90]、Fenton[Fen91]、Ejiogu[Eji91]、Moeller和Paulish（《Software Metrics》，Chapman and Hall, 1993）以及Hetzel[Het93]都比较详细地讨论了产品度量。Oman和Pfleeger（《Applying Software Metrics》，IEEE Computer Society Press, 1997）编辑出版了关于软件度量的重要论文集。

Ebert和他的同事（《Best Practices in Software Measurement》，Springer, 2004）讨论了建立度量程序

的方法和软件测量的基本原则。Shepperd（《Foundations of Software Measurement》，Prentice-Hall，1996）也详细地讨论了测量理论，当前的研究情况在《Proceedings of the Symposium on Software Metrics》（IEEE，年刊）中有介绍。

[IEE93]全面总结了很多有用的软件度量。一般来讲，每个度量的讨论，都对计算该度量所需的关键“基本方面”（测度）加以精炼，并总结了影响计算的适当关系，其附录中提供了讨论和许多参考文献。

Whitmire[Whi97]关于面向对象度量的讨论是迄今为止最全面和数学上最完善的。Lorenz和Kidd[Lor94]以及Hendersen-Sellers（《Object-Oriented Metrics: Measures of Complexity》，Prentice-Hall，1996）专门讨论了面向对象度量。

有关软件度量的大量信息资源可在因特网上获得。有关软件度量的最新WWW参考文献可以在SEPA网站www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

软件项目管理

在本书的这一部分，将学习计划、组织和监控软件项目所需要的管理技术。在下面的章节中，我们将讨论以下问题：

- 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？
- 软件团队如何可靠地估算软件项目的工作量、成本和工期？
- 采用什么技术来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？
- 为什么维护和再工程对于软件工程管理人员和实践人员都如此重要？

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24章

Software Engineering：A Practitioner's Approach (Seventh Edition)

项目管理概念

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。在软件从初始的概念演化为可运行的实现的过程中，项目管理涉及对人员、过程和所发生事件的策划和监控。

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；必须和客户及其他利益相关者很好地沟通，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来监控计划所规定的工作。

工作产品：在管理活动开始时，首先是制定项目计划。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
协调与沟通
关键实践
人员
问题分解
产品
项目
软件范围
软件团队
利益相关者
团队负责人
W^{5}HH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。

在本章以及第25章到29章中，将给出进行有效的软件项目管理的关键概念。本章考虑软件项目管理的基本概念和原则。第25章介绍过程和项目度量，这是做出有效管理决策的基础。第26章讨论用于估算成本的技术。第27章将帮助你编制实际的项目进度表。第28章阐述了进行有效的风险监测、风险缓解和风险控制的管理活动。最后，第29章将考虑维护和再工程，并讨论了处理遗留系统时将遇到的管理问题。

24.1 管理涉及的范围

有效的软件项目管理集中于4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利益相关者之间的广泛交流，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。

24.1.1 人员

从20世纪60年代起人们就一直在讨论要培养有创造力、高技术水平的软件人员。实际上，“人的因素”的确非常重要，美国卡内基·梅隆大学的软件工程研究所（SEI）认识到这一事实——“每个组织都需要不断地提高他们的能力来吸引、发展、激励、组织和留住那些为实现其战略业务目标所需的劳动力”[Cur01]，并开发了一个人员能力成熟度模型（People-CMM）。

人员能力成熟度模型中针对软件人员定义了以下关键实践域：人员配备、沟通与协调、工作环境、业绩管理、培训、报酬、能力素质分析与开发、个人事业发展、工作组发展、团队精神或企业文化培养等。People-CMM成熟度达到较高水平的组织，更有可能实现有效的软件项目管理实践。

People-CMM与软件能力成熟度集成模型（第30章）相伴而生，后者可指导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考虑。

24.1.2 产品

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（精确的）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。

作为软件开发者，必须与其他利益相关者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，并一直持续到作为软件需求工程的第一步（第5章）。确定产品的目标只是识别出产品的总体目标（从利益相关者的角度），而不用考虑如何实现这些目标。而确定产品的范围，是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。

了解产品的目标和范围之后，就要开始考虑备选的解决方案。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，约束条件包括产品交付期限、预算限制、可用人员、技术接口以及其他各种因素。

坚持敏捷过程方法（第3章）的人指出敏捷过程比其他过程更简单，这可能是真的。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。

24.1.3 过程

软件过程（第2章和第3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、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。

24.1.4 项目

我们实施有计划的、可控制的软件项目的主要理由是：这是我们知道的管理复杂事物的唯一方法。然而，软件团队仍需要努力。在对1998年到2004年之间的250个大型软件项目的一份研究中，Capers Jones[Jon04]发现“大约有25个项目被认为是成功的，达到了他们的计划、成本和质量目标；大约有50个项目延迟或超期在35%以下；而大约有175个项目经历了严重的延迟和超期，或者没有完成就中途夭折了。”虽然现在软件项目的成功率可能已经有所提高，但项目的失败率仍然大大高于它的应有值[⊖]。

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避免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24.5节及以后的章节中讨论。

24.2 人员

在IEEE[Cur88]发表的一项研究中提到，当向3个大型技术公司中主管工程的3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，同时我也培养优秀的人员，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，我们所有人，从高级工程副总裁到最基层的开发人员，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。本节将分析参与软件过程的利益相关者，并研究组织人员的方式，以实现有效的软件工程。

24.2.1 利益相关者

参与软件过程（及每一个软件项目）的利益相关者可以分为以下5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——阐明待开发软件需求的人员以及关心项目成败的其他利益相关者。
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。

每个软件项目都有上述人员的参与[⊜]。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。

24.2.2 团队负责人

项目管理是人员密集型活动，因此，胜任开发的人却常常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

⊖ 看到这些统计数据，人们自然会问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象的那样重要)，进而取消了这些项目。

⊜ 当开发Web应用软件时，在内容创作方面需要其他非技术人员参与。

在项目经理的位置上，也就意外地成为项目经理”[Edg95]。

在一本关于技术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）、Ideas or Innovation（思想或创新））。

当我们选择软件项目的负责人时，我们在寻找什么？

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系统的约束下工作，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个人都知道（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，负责人知道他想去哪里，并起身朝那里走。”——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4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；能够将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，一位称职的项目经理必须奖励那些工作积极主动并且做出成绩的人。必须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。

24.2.3 软件团队

“并非每个小组都是团队，并非每个团队都是有效的。”——Glenn Parker

选择软件团队的结构时，应该考虑什么因素？

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的组织结构。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，对新的软件项目中所直接涉及的人员进行组织，则是项目经理的职责。

“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7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。
- 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。

Constantine[Con93]提出了软件工程团队的4种“组织范型”：

确定软件团队的结构时，我们有哪些选择？

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。

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。

“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

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，但必须将创新能力引入高效团队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。

无论是什么类型的团队，每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。

什么是“有凝聚力”的团队？

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。

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。他们拥有统一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。

为什么团队没有凝聚力？

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”。

“只有做或不做，没有尝试。”——Yoda（《星球大战》）

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多的决策权，就能使团队避免挫败。通过理解将要开发的产品和完成工作的人员，以及允许团队选择过程模型，可以避免选择不适当的软件过程（如不必要的或繁重的工作任务，或没有很好地选择工作产品）。团队本身应该建立自己的责任机制（技术评审[⊖]是实现此目标的极好方式），并规定一系列当团队成员未能完成任务时的纠正方法。最后，避免失败的关键是建立基于团队的信息反馈方法和解决问题的技术。

除了Jackman描述的5个毒素以外，团队成员的个性不同也给软件团队带来了一系列的问题。有些团队成员性格外向，有些团队成员性格内向。有些人依靠直觉收集信息，从分离的事实中提炼主要概念；有些人则是线性地处理信息，从提供的数据中收集和组织微小的细节。有些团队成员，只有当给出逻辑的、有序的论据时，才能做出决策；有些团队成员则依靠直觉，喜欢根据“感觉”做出决策。有些人希望有详细的任务进度计划，使得他们能够按部就班地完成项目的各个部分；有些人则喜欢更自发的环境，在这种环境中，允许开放地讨论问题。有些人工作刻苦，在最后期限前很长时间就完成了任务，从而避免了时间逼近所带来的压力；而有些人则经常为了时限的逼近而在最后一分钟加班、冲刺。熟练的团队负责人一般都掌握了如何帮助不同个性的人协同工作的方法。如何对待个性不同的人是心理学问题，这超出了本书研究的范围[⊜]。不过，重要的是要注意到，识别人员差异是建立有凝聚力的团队的第一步。

24.2.4 敏捷团队

在过去的十年中，敏捷软件开发（第3章）已被当成解决软件项目开发诸多困扰的一剂良方。回顾一下，敏捷方法学倡导的是：通过尽早地逐步交付软件来使客户满意；组织小型的充满活力的项目团队；采用非正式的方法；交付最小的软件工程工作产品；以及总体开发简易性。

小型的充满活力的项目团队，也称为敏捷团队，这种团队采纳了很多成功的软件项目团队的特性（在上一节内容中谈到的），避免了很多产生问题的毒素。同时，敏捷方法学强调团队成员的个人能力与团队协作精神相结合，这是团队成功的关键因素。对此，Cockburn和Highsmith [Coc01a]这样写道：

敏捷团队是自组织的团队，拥有制定计划和做技术决定的自主权。

如果项目成员足够优秀，那么他们几乎可以采用任何一种过程来完成任务。如果项目成员不够优秀，那么没有任何一种过程可以弥补这个不足。“人员胜过过程”阐明的正是这样的含义。然而，如果缺乏用户和主管人员的支持，也可以毁掉一个项目，即“政策胜过人员”。缺乏支持可以阻止最好的人员完成任务。

在软件项目中，为了充分发挥每个团队成员的能力，并培养有效的合作，敏捷团队是自组织的。自组织团队不必保持单一的团队结构，而是采用在24.2.3节中讨论的由Constantine提出的随机、开放、同步式的范型。

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（例如，过程、方法和工具），只受业务需求和组织标准的限制。在项目进展过程中，自组织团队

⊖ 技术评审将在第15章详细讨论。

⊜ 关于这些问题（当它们和软件项目团队相关时）的一个很好的介绍可在[Fer98]中找到。

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，敏捷团队召开日常团队例会，对当天必须完成的工作进行协调和同步控制。

基于在团队例会中获取的信息，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接近的完工的目标前进。

24.2.5 协调与沟通问题

"集体所有权只不过是如下观念的产物：产品属于（敏捷）团队，而不属于团队中的个人。"——Jim Highsmith

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）确实都存在。为了有效地处理这些问题，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。

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及其他产品软件工程团队成员。

[对话]

Doug：你们看过市场销售部准备的有关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，好像在背诵什么）：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。

24.3 产品

从软件项目一开始，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。然而，计划总是“眼前”就需要的！

不管喜欢与否，从项目一开始，就要研究应该开发哪些产品以及要解决哪些问题。至少，我们要建立和界定产品的范围。

24.3.1 软件范围

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）。

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：

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变换成输出数据？软件需要满足什么特殊的性能要求？

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（例如，并发用户数、目标环境、允许的最大响应时间）；说明约束和（或）限制（例如，产品的成本要求会限制内存的大小），并描述其他的调节因素（例如，期望的算法能被很好地理解，并采用Java实现）。

为了制定合理的项目计划，必须对问题进行分解。这可以使用一系列功能或用例来完成。

24.3.2 问题分解

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6章和第7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）必须交付的功能和内容（信息）；（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更易处理的小问题。这是项目计划开始时所采用的策略。在开始估算（第26章）前，必须对软件范围中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很有益处。同样，为了对软件生成的信息提供合理的解释，要将主要的内容对象或数据对象分解为各自的组成部分。

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：连续的语音输入以及通过多点触摸屏进行虚拟键盘输入；高级的“自动复制编辑”特性；页面布局功能；自动建立索引和目录等。项目经理首先必须写出软件范围陈述来界定这些特性（以及其他通用功能，如编辑、文件管理、文档生成）。例如，连续的语音输入功能是否需要用户对产品进行“训练”？复制编辑特性提供什么功能？页面布局功能要高级到什么程度，其中包括多点触摸屏所具有的功能吗？

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）使整个文档保持一致的样式表的实现；（5）大型文档中章节引用的确认。其中每一项都是软

件要实现的子功能。同时，如果分解能使制定计划更容易，则每一项又可以进一步细化。

24.4 过程

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。

团队必须决定哪种过程模型最适合于：（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）软件团队所处的项目工作环境。当选定过程模型后，项目团队可以基于这组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完成的工作任务。在下面的几小节中，我们将对这些活动进行简单的研究，更详细的信息将在第26章中给出。

24.4.1 合并产品和过程

项目计划开始于产品和过程的合并。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。

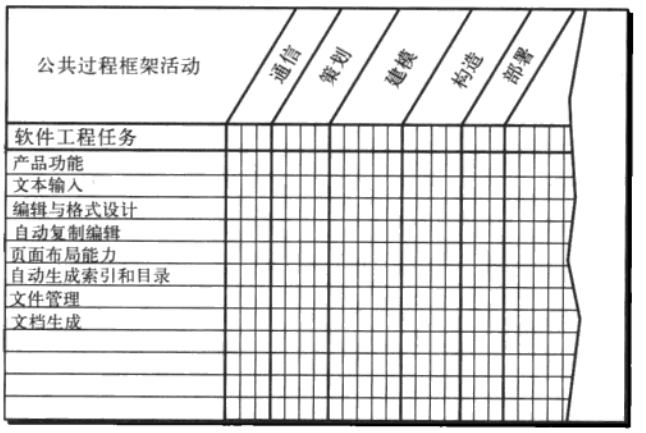


图24-1 合并问题和过程

假定软件组织采用了在第2章中讨论的通用框架活动——沟通、策划、建模、构造和部署。团队成员完成任何一项功能时，都要应用各个框架活动。实质上，产生了一个类似于图24-1所示的矩阵。产品的每个主要功能（图中列出了前面讨论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）列在后续的行中[⊖]。项目经理（及其他团队成员）的工作是估算每个矩阵单元的资源需求，与每个单元相关的任务的起止日期，以及每项任务所产生的工作产品。这些活动将在第26章中考虑。

KEY POINT
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。

24.4.2 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开发过的项目相似，可以采用线性

⊖ 应该注意，根据很多适应性标准，工作任务必须适合于项目的特殊需要。

顺序方法。如果时间要求太紧，不可能完成所有功能时，增量策略可能是最好的。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），可能就要选择其他过程模型[⊖]。

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都可以使用。它既适用于线性模型，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，是软件组织进行的所有工作的基础。

但实际的工作任务是不同的。当项目经理问道："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. 与利益相关者会面商讨需澄清的问题。
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小时的时间内发生。这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，考虑一个更复杂的项目，它的范围更广，具有更重要的商业影响。这样一个项目在沟通中可能需要完成下列工作任务：

1. 评审客户需求。
2. 计划并安排与全体利益相关者召开正式的、有人主持的会议。
3. 研究如何说明推荐的解决方案和现有的方法。
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、功能和行为特性的微型规格说明。或者，从用户的角度出发建立描述软件的用例。
7. 评审每一份微型规格说明或用例，确认其正确性、一致性和无歧义性。
8. 将这些微型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是第二个项目团队的一半。

24.5 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，以便避免这些问题。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：

什么信号表示软件项目正处于危险状态？

1. 软件人员不了解客户的需要。
2. 产品范围定义得很糟糕。
3. 没有很好地管理变更。

⊖ 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24.2.3节。

4. 所选的技术发生了变化。
5. 业务需求发生了变化（或未能很好地定义）。
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（和开发人员）没有很好地利用已学到的最佳实践和经验。

"我们没有时间停下来空谈，我们已经来不及了。"——M.Cleron

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90-90规则：系统前面90%的任务会花费所分配总工作量和时间的90%，系统最后10%的任务也会花费所分配总工作量和时间的90%[Zah94]。导致该90-90规则的根源就在上面列出的"信号"中。

这太消极了！管理者如何避免上面提到的这些问题呢？Reel[Ree99]针对软件项目提出了以下易于理解的方法，包含5部分：

1. 在正确的基础上开始工作。通过以下两点来实现：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（24.2.3节），并给予团队工作时所需的自由、权力和技术而得到加强。

"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

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，团队应该重视它完成的每项任务的质量，而高层管理应该尽可能不干涉团队的工作[⊖]。

3. 跟踪进展。对于软件项目而言，当工作产品（如模型、源代码、测试用例集）正在产生或被认可（通过技术评审）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25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. 做出英明的决策。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或模式，可以采用标准方法时避免定制接口，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，从团队成员和客户处获取反馈，并记录所有的发现。

24.6 W⁵HH原则

我们如何定义关键的项目特性？

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，对它进行缩减来为简单的项目提供简单的（项目）计划。"Boehm给出了一种方法，该方法描述项目的目标、里程碑与进度、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：

⊖ 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，不再强调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。

为什么（Why）要开发这个系统？所有利益相关者都应该了解软件工作的商业理由是否有效。该系统的商业目的是否值得花费这些人力、时间和金钱。

将要做什么（What）？定义项目所需的任务集。

什么时候（When）做？团队制定项目进度，标识出何时开展项目任务以及何时到达里程碑。

某功能由谁（Who）负责？规定软件团队每个成员的角色和责任。

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、用户和其他利益相关者也有责任。

如何（How）完成技术工作和管理工作？一旦确定了产品范围，就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26章）而得到的。

Boehm的W^5HH原则可适用于任何规模和复杂度的软件项目。给出的问题为你和你的团队提供了很好的计划大纲。

24.7 关键实践

Airlie Council[⊖]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）一直普遍采用，并被认为的确是关键的”[Air99]。

关键实践[⊜]包括：基于度量的项目管理（第25章）、成本及进度的经验估算（第26和27章）、获得价值跟踪（第27章）、根据质量目标跟踪缺陷（第14到16章）和人员计划管理（第24.2节）。每一项关键实践都贯穿于本书的第三和第四部分。

SOFTWARE TOOLS

项目管理的软件工具

这里列出的都是通用工具，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：[⊟]

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www.spmn.com/products_software.html。

Ganthead.com（www.gantthead.com/）已经为项目经理开发了一套实用的检查表。

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。

⊖ Airlie Council由美国国防部组织的软件工程专家组组成，其目的是开发在软件项目管理和软件工程中最佳的实践指南。关于最佳实践更多的信息，参见www.swqual.com/newsletter/vol1/no3/vol1no3.html。

⊜ 这里只讨论与“项目完整性”有关的关键实践。

⊟ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

24.8 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建模、构造和部署之中。

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产品。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。可以按照多种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团队的工作。一般而言，技术评审和非正式的人与人交流对开发者最有价值。

项目管理活动包括测量与度量、估算与进度安排、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。

习题与思考题

24.1 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。

24.2 SEI的人员能力成熟度模型（People-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，请你对它进行分析和总结。

24.3 描述3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3种他们是不同人的情况。

24.4 高级管理者所做的决策会对软件工程团队的效率产生重大影响。请列举5个实例来说明这一点。

24.5 温习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。

24.6 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已经做过的项目，只是规模更大而且更复杂。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？

24.7 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？你会选择哪些软件过程模型？为什么？

24.8 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。由于竞争激烈，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？你会选择哪些软件过程模型？为什么？

24.9 在一个为基因工程领域服务的公司中，你被指派为软件项目经理。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？你会选择哪些软件过程模型？为什么？

24.10 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。

24.11 给出24.3.2节中讨论的页面布局功能的第一级功能分解。

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。Bechtold（《Essentials of Software Project Management》，2nd ed.，Management Concepts，2007），Wysocki（《Effective Software Project Management》，Wiley，2006），Stellman与Greene（《Applied Software Project Management》，O'Reilly，2005），以及Berkun（《The Art of Project Management》，O'Reilly，2005）讲授了基本技能，并提供软件项目管理所有任务的详细指导。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”。Henry（《Software Project Management》，Addison-Wesley，2003）为所有项目经理提供了现实的建议。

Tom DeMarco和他的同事（《Adrenaline Junkies and Template Zombies》，Dorset House，2008）对软件项目中会遇到的每种人员模式都给出了有深入见解的处理方法。由Weinberg所著的一套4册的系列丛书（《Quality Software Management》，Dorset House，1992，1993，1994，1996）介绍了基本的系统思想和管理概念，讲解了如何有效地使用测量，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关系。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处理方法。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）为那些必须管理软件项目的人提供了卓越的有实效的行动指南。Purba和Shah（《How to Manage a Successful Software Project》，2nd ed.，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》，3rd ed.，Wiley，2000）为软件项目经理提供了实用的提示和指导。Weigers（《Practical Project Initiation》，Microsoft Press，2007）为软件项目的成功实施提供了实用的指南。

可以证明，软件项目管理中最重要的方面是人员管理。Cockburn（《Agile Software Development》，Addison-Wesley，2002）给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著作。此外，关于这一主题，近年来出版的如下书籍值得一读：

Cantor，M.，《Software Leadership: A Guide to Successful Software Development》，Addison-Wesley，2001。

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。

Garton，C.，and K. Wegryn，《Managing Without Walls》，McPress，2006。

Humphrey，W.S.，《Managing Technical People:Innovation,Teamwork,and the Software Process》，Addison-Wesley，1997。

Humphrey，W.S.，《TSP-Coaching Development Teams》，Addison-Wesley，2006。

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。

Peters，L.，《Getting Results from Software Development Teams》，Microsoft Press，2008。

Whitehead，R.，《Leading a Software Development Team》，Addison-Wesley，2001。

以下列出的一些畅销“管理”书籍并不和软件世界特别相关，有时还过于简单，过于概括：Kanter（《Confidence》，Three Rivers Press，2006），Covy（《The 8th Habit》，Free Press，2004），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。最新的与软件项目管理相关的WWW参考文献目录可在SEPA Web站点www.mhhe.com/engcs/compsci/pressman/professnal/olc/ser.htm找到。

过程度量和项目度量

要点浏览

概念：软件过程度量和项目度量是定量的测量，这些测量能使你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、项目测量和产品测量。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，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，给出结论。
工作产品：一组软件度量，它们提供了对过程深入透彻的认识和对项目的理解。
质量保证措施：通过采用一致而简单的测量计划来保障，但该计划绝对不能用于对个人表现进行的评估、奖励或惩罚。

关键概念
缺陷排除效率
功能点
测量
度量
争论
基线
制定大纲
面向功能
基于LOC
面向对象
过程
生产率
项目
公有和私有
面向规模
软件质量
面向用例
Web应用

通过提供目标评估的机制，测量能使我们对过程和项目有更深入的了解。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；如果不能测量它，也不能用数字来表达，就说明你对它的了解还很贫乏，不能令人满意：这可能只是知识的起点，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，也不是只有一点点争论。

测量可以应用于软件过程，目的是持续改进软件过程。测量也可以应用于整个软件项目，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23章）。

从软件过程以及使用该过程进行开发的项目出发，软件团队主要关注生产率度量和质量度量——前者是对软件开发“输出”的测量，它是投入的工作量和时间的函数，后者是对所生产的工作产品“适用性”的测量。为了进行计划和估算，需要了解历史数据。以往项目的软件开发生产率是多少？开发出来的软件质量怎么样？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？这些数据如何帮助你更精确地计划和估算？

在Park、Goethert和Florac[Par96b]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、低效率和其他提高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。

25.1 过程领域和项目领域中的度量

KEY POINT
过程度量具有长远的影响，其目的是改进过程本身。项目度量常常会促进过程度量的发展。

过程度量的收集涉及所有的项目，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风险；（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（第30章）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。

25.1.1 过程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（第30章）。但是在讨论软件度量及其对软件过程改进的影响之前，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]。

KEY POINT
软件人员的技能和动力是影响软件质量的最重要因素。

在图25-1中，过程位于三角形的中央，连接了3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。

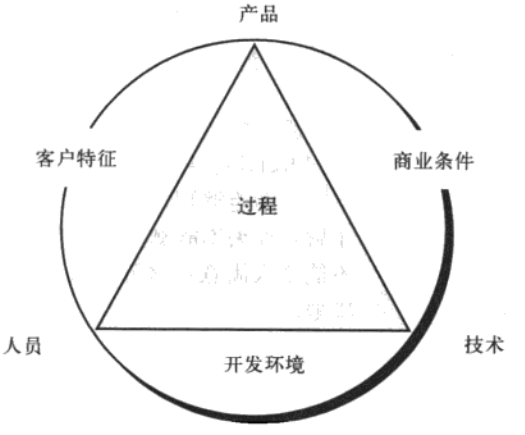


图25-1 软件质量和组织有效性的决定因素（改自[Pau94]）

“软件度量让你知道什么时候笑，什么时候哭。”——Tom Gilb

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（如集成的软件工具）、商业条件（如交付期限、业务规则）、客户特征（例如，交流和协作的容易程度）。

软件过程的功效只能间接地测量。也就是说，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。也可以通过测量特定软件工程任务的特性来导出过程度量。例如，测量第2章中所描述的普适性活动和一般软件工程活动所花费的工作量和时间。

Grady [Gra92] 认为不同类型过程数据的使用可以分为“私有的和公有的”。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。这些数据对本人应该是私有的，是仅供本人参考的指标。私有度量的例子有：缺陷率(个人)、缺陷率(软件构件)和开发过程中发现的错误数。

对软件度量的私有使用和公有使用有什么不同？

“私有过程数据”的观点与Humphrey [Hum97]所提出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。私有过程数据是改进自身软件工程方法的重要驱动力。

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有的。例如，主要软件功能（由多个开发人员共同完成）的缺陷报告、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[⊖]。团队评审这些数据，以找出能够提高团队效能的指标。

公有度量一般吸收了原本是个人或团队的私有信息。收集和评估项目级的缺陷率（绝对不能归咎于某个人）、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过程成熟度的整体水平能够提供很大的帮助。不过，与其他所有度量一样，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会逐渐被更精确的称为统计软件过程改进（statistical software process improvement，SSPI）的方法所取代。本质上，SSPI使用软件失效分析方法来收集在应用软件、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[⊜]。

25.1.2 项目度量

软件过程度量用于战略目的，而软件项目测量则用于战术目的。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。从以往项目中收集的度量可以作为当前软件工作的工作量及时间估算的基础。随着项目的进展，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23章）来评估设计质量，

⊖ 代码行和功能点度量将在第25.2.1和25.2.2节中讨论。

⊜ 在本书中，错误（error）是指软件工程工作产品中的瑕疵（flaw），这些瑕疵在交付给最终用户之前已经被发现。而缺陷（defect）是指交付给最终用户之后才发现的瑕疵。应该注意到，其他人并没有进行这样的区分。

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，并减少潜在的问题和风险，从而使得开发时间减到最少。其次，项目度量可用于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。

在项目中，我们应该如何使用度量？

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，这将使项目总体成本降低。

SAFEHOME

建立度量方法

[场景] SafeHome软件项目即将启动，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，变得支持Web，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，工程师的工作包括收集数据、评估数据、使用评估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。

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。

25.2 软件测量

在第23章中，谈到物理世界中的测量可分为两种方法：直接测量（如螺栓的长度）和间接

“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计算。”——阿尔伯特·爱因斯坦

测量（如生产的螺栓的“质量”，通过统计废品数量来测量）。软件度量也可以这样来划分。

软件过程的直接测量包括花费的成本和工作量。产品的直接测量包括产生的代码行（LOC）、运行速度、存储容量以及某段时间内报告的缺陷；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，以及许多在第14章中谈到的其他“产品特性”。

构造软件所需的成本和工作量，产生的代码行数以及其他直接测量，相对容易收集，只要事先建立特定的测量协议即可。但是，软件的质量和功能、有效性或可维护性则很难获得，只能间接地测量。

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。

我已将软件度量范围分为过程度量、项目度量和产品度量。注意，产品度量对个人来讲是私有的，常常将它们合并起来生成项目度量，而项目度量对软件团队来说是公有的。再将项目度量联合起来可以得到整个软件组织公有的过程度量。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？

为了说明这个问题，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？由于不了解项目的规模或复杂性，所以不能回答这个问题。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。

25.2.1 面向规模的度量

面向规模的软件度量是通过对质量和（或）生产率的测量进行规范化而得到的，而这些测量都是根据开发过的软件的规模得到的。如果软件组织一直在做简单记录，就会产生一个如图25-2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25-2）：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25-2 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，你可以选择代码行作为规范化值。根据表中包含的基本数据，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；
- 每人月千行代码数；
- 每页文档的成本。

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持续。

面向规模的度量是否是软件过程测量的最好方法，对此并没有普遍一致的观点。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；当考虑生产率时，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；而且在估算时需要一些可能难以得到的信息（例如，计划人员必须在分析和设计远未完成之前，就要估算出将产生的LOC）。

25.2.2 面向功能的度量

面向功能的软件度量以功能（由应用程序提供）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23章中讨论过了[⊖]。

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，FP是一种更有吸引力的估算方法。反对者们则声称这种方法需要某种“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）的数据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。

25.2.3 调和代码行度量和功能点度量

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。表25-1[⊜] [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：

查看这些数据可知，C++的一个LOC所提供的“功能”大约是C的一个LOC所提供功能的2.4倍（平均来讲）。而且，Smalltalk的一个LOC所提供的“功能”至少是传统程序设计语言（如Ada、COBOL或C）的4倍。利用上表包含的信息，只要知道了程序设计语言的语句行数，就可以“逆向”[Jon98] 估算出现有软件的功能点数量。

⊖ 对FP计算的详细讨论，参见23.2.1节。

⊜ 已得到Quantitative Software Management（www.qsm.com）的使用许可，copyright 2002。

表25-1 各种程序设计语言实现一个功能点所需的代码行数的粗略估算

	LOC/FP			
程序设计语言	平均值	中值	低值	高值
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP 69	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
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/1	78	67	22	263
Powerbuilder	32	31	11	105
REXX	67	—	—	—
RPG II/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

LOC和FP测量经常用来导出生产率度量，这总会引起关于这些数据使用的争论。一个小组的LOC/人月（或功能点/人月）应该与另一个小组的类似数据进行比较吗？管理者应该使用这些度量来评价个人绩效吗？对这些问题都斩钉截铁地回答“不！”，其原因是生产率受很多

因素的影响，进行“苹果和橘子”式的比较很容易曲解。

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，为了使用LOC和FP进行估算（第26章），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，应该主要关注生产率和质量——软件开发“输出量”（作为投入的工作量和时间的函数）的测量以及对产生的工作产品“适用性”的测量。为了进行过程改进和项目策划，必须掌握历史情况。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？

25.2.4 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。有时可以将多个脚本简化为一个类或一组代码。

场景脚本的数量。场景脚本（与贯穿于本书第二部分中的用例类似）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。每个脚本采用如下3个一组的形式来组织：

{发起者，动作，参与者}

其中，发起者是指请求某个服务（首先传递一个消息）的对象，动作是该请求的结果，参与者是满足该请求的服务对象。场景脚本的数量与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6章）[⊖]。由于关键类是问题域的核心，因此，这些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。

每个类的规模和复杂性不同，因此可以考虑将类按照规模和复杂性的不同分别统计其数量。

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，用户界面（GUI）类、数据库访问及操作类以及计算类。另外，对于每一个关键类，都可以开发其支持类。在演化过程中，支持类是迭代定义的。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）将变得极其简单。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。

为了将上述这些度量有效地应用于面向对象的软件工程环境中，应该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），面向对象测量和项目测量之间的关系将提供有助于项目估算的度量。

⊖ 在本书的第二部分中，关键类被称为分析类。

25.2.5 面向用例的度量

用例[⊖]被广泛地用于描述客户层或业务领域的需求，而这些需求中隐含着软件的特性和功能。类似于LOC或FP，使用用例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。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要设计的。

由于可以在不同的抽象级别上创建用例，所以用例的“大小”没有统一标准。由于对用例本身都没有标准的测量，因此，将用例作为规范化的测量（例如，每个用例花费的工作量）是不可信的。

研究人员提议将用例点（UCP）作为一种估算项目工作量及其他特性的机制。UCP是用例模型所包含的参与者数量及业务数量的函数，在某些方面与FP相似。如果有兴趣进一步了解，可参见[Cle06]。

25.2.6 Web应用项目度量

所有Web应用项目的目标都是向最终用户交付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应用于Web应用系统中。然而，建立一个数据库，随着大量项目的完成，允许访问该数据库获得内部的生产率和质量测量，这是可能的。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，构造静态页面所需的工作量通常少于动态页面。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或其他外部因素，页面上显示出相应的定制内容）是必需的要素。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，系统的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（例如，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。

⊖ 关于用例的介绍，已在第5章和第6章中给出。

上述的每个测量在初期就可以确定下来。例如，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，进行以下定义：

N_{sp} = 静态Web页的数量

N_{dp} = 动态Web页的数量

那么，定制指数，$C = \dfrac{N_{dp}}{N_{dp} + N_{sp}}$

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（例如，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

项目和过程度量

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：⊖

Function Point WORKBENCH，由Charismatek（www. charismatek. com. au）开发，提供了大量的面向功能点的度量。

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、报表生成以及其他测量任务。

PSM Insight，由Practical Software和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，收集了一整套面向功能点的工具。

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。

25.3 软件质量度量

ADVICE

软件是一个复杂的实体。因此随着工作产品的开发，错误就会产生。过程度量就是要改进软件过程，以便更有效地发现错误。

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，你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。

系统、应用或产品的质量取决于描述问题的需求、建模解决方案的设计、导出可执行程序的编码以及执行软件来发现错误的测试。可以使用测量来获知需求与设计模型的质量、源代码的质量以及构造软件时就要创建的测试用

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

例的质量。为了做到这种实时的评价，必须应用产品度量（第23章）来客观地、而不是主观地评估软件工程工作产品的质量。

随着项目的进展，项目经理也必须评估质量。将软件工程师个人收集的私有度量结合起来，可以提供项目级的结果。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。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。

度量，如每功能点的工作产品错误数、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隐含的每项活动的功效。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25.3.3节讨论。

25.3.1 测量质量

虽然有很多关于软件质量的测量指标[⊖]，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，否则对于用户就毫无价值。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，软件维护和支持需要更多的工作量。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。还没有直接测量可维护性的方法，因此只能采用间接测量。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）。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费的时间。一般情况下，与那些不可维护的程序相比，可维护的程序应该有较低的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3个成分(即程序、数据及文档)都会遭到攻击。

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：

$$完整性=\Sigma[1-(危险性\times(1-安全性))]$$

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11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4个因素仅仅是一个样本。关于这些内容，

⊖ 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23章详细讨论。

在第23章中已经给出了更详细的讨论。

25.3.2 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动作中滤除缺陷能力的测量，而这些质量保证及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：

$$\text{DRE} = \frac{E}{E + D}$$

如果从分析阶段进入设计阶段时，DRE的值较低，你就要花些时间去改进技术评审方式了。

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。

DRE最理想的值是1，即在软件中没有发现缺陷，实际上，D的值大于0。但对于给定的D值，随着E的增加，DRE仍可能接近于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动作之前发现错误的能力。例如，需求分析创建了一个需求模型，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：

$$\text{DRE}_i = \frac{E_i}{E_i + E_{i+1}}$$

其中，E_i是在软件工程动作i中发现的错误数；E_{i+1}是在软件工程动作$i+1$中发现的错误数，这些错误都是在软件工程动作i中没被发现的错误。

软件团队（或软件工程师个人）的质量目标是使DRE_i接近于1，即错误应该在传递到下一个活动或动作之前被过滤掉。

SAFEHOME

建立度量方法

[场景] Doug Miller的办公室，在首次召开软件度量会议两天之后。

[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，

当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：可以计算出评审与开发的比率……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：我同意，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到6项，准备去做吧。

25.4 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。正如本章开头所提到的，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“为什么我们要做这些”。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，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Caswell[Gra87]所说的充满智慧的话（距现在已有20多年）：

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大纲是很困难的工作。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，你就能对这一工作量的规模有概念了。

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。

25.4.1 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，就无法确定你是否在改进。如果你没有在改进，就会导致失败。

“在生活的很多方面，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

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。在本书的开头部分提到，对于许多公司而言，软件都是一个战略性的商业产物。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，就必须了解当前的软件开发状态。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。

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。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[⊖]。

⊖ 这些想法已经规范化成为一种方法，称为统计软件质量保证。

25.4.2 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？

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。度量基线由以往开发的软件项目中收集的数据构成，它可能像图25-2所示的表格那样简单；也可能像一个综合数据库那样复杂，其中包含了几十个项目的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，如果将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，就没有什么意义了。

KEY POINT

基线度量数据应该从以往有代表性的大量软件项目样本中收集

25.4.3 度量收集、计算和评估

建立度量基线的过程如图25-3所示。理想情况下，建立基线所需的数据已经在项目开发过程中收集了。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（例如，LOC、FP、面向对象、Web应用）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。

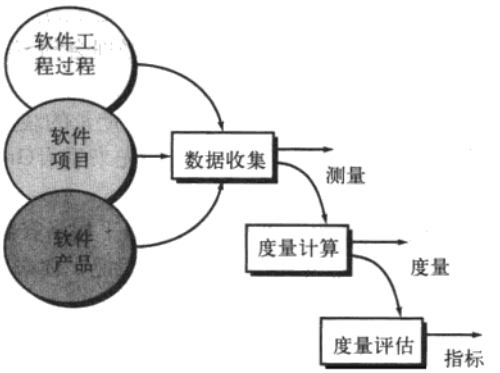


图25-3 软件度量的收集过程

25.5 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。

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，将考察如何将这些指导原则应用于小型组织的度量中[⊖]。

“保持简单”是一个在很多活动中都相当奏效的原则。但是，应该怎样导出一组“简单的”又有价值的软件度量？如何确定这些简单的度量能满足软

⊖ 对于已经采用了敏捷软件开发过程（第3章）的团队来说，这项讨论同样具有指导意义。

我们应该怎样导出一组“简单的”软件度量？

件组织的特定需要？一开始，先不要关注测量，而是要从结果入手。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}。
- 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，还可以计算出评估变更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：

$$\text{DRE} = \frac{E_{change}}{E_{change} + D_{change}}$$

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。

对于小型团体来说，收集测量及计算度量的成本，在学习阶段占项目预算的3%到8%，而当软件工程师和项目经理已经熟悉度量大纲后，会降至项目预算的1%以下[Gra99]。如果由度量数据得到的洞察力导致了软件组织的有意义的过程改进，那么这些成本就带来了实实在在的投资回报。

25.6 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[Par96b]。手册中给出了以下步骤：

WebRef

目标驱动的软件测量指导手册可以从www.sei.cmu.edu下载。

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，有必要简要阐述一下关键问题。

因为软件支持业务功能，将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，考虑

KEY POINT
你选择的软件度量应该由你所希望达到的业务和技术目标来驱动。

SafeHome产品。软件工程师和业务管理者协同工作，制定出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. 使产品支持更容易。
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、执行或支持什么活动？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（例如，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4个目标："使产品支持更容易。"由这个目标，可以引出下列问题[Par96b]：

- 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？
- 遵循变更控制过程（第22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：提高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[Par96b]对目标驱动测量方法中的步骤6～10，给出了详细指导。本质上是将测量目标精化为问题，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25.6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，提高产品的质量。

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中，有多少个可以追踪到前一个框架活动。

   创建帮助回答这些问题的测量和度量。

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（例如，SCM中心存储库，第26章）的使用。
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www.spc.ca/resources/metrics。

25.7 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，评估所生成的产品（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。

过程度量能使一个组织从战略角度深入了解软件过程的功效。项目度量是战术性的，能使项目管理者实时地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应用。面向规模的度量以代码行作为其他测量（如人月或缺陷）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。此外，还能使用面向对象的度量和Web应用度量。

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，那么数据收集、度量计算和度量分析是必须完成的3个步骤。通常，目标驱动方法有助于一个组织关注自身业务的正确度量。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。

习题与思考题

25.1 用自己的话描述过程度量和项目度量之间的区别。

25.2 为什么有些软件度量是“私有的”？给出3个私有度量的例子，并给出3个公有度量的例子。

25.3 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？

25.4 Grady提出了一组软件度量规则，你能在25.1.1节所列的规则中再增加3个规则吗？

25.5 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能有助于做出判定？哪些历史数据可能有用？

25.6 给出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？

25.7 根据下面的信息域特性，计算项目的功能点值：

用户输入数：32

用户输出数：60

用户查询数：24

文件数：8

外部接口数：2

假定所有的复杂度校正值都取“中等”值。使用第23章描述的算法。

25.8 利用第25.2.3节中给出的表格，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？

25.9 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。

25.10 某Web工程团队已经开发了一个包含145个网页的电子商务Web应用系统。在这些页面中，有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，该应用系统的定制指数是多少？

25.11 一个Web应用系统及其支持环境没有被充分地加强来抵御攻击。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Web应用系统的完整性是多少？

25.12 在一个项目结束时，确定在建模活动中发现了30个错误，在构造活动中发现了12个可以追溯到建模活动中没有发现的错误。那么，建模活动的DRE是多少？

25.13 软件团队将软件增量交付给最终用户。在第一个月的使用中，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测试任务中发现了242个错误。那么在使用一个月之后，项目总的缺陷排除效率（DRE）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）得到了极大关注。由于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。Rico（《ROI of Software Process Improvement》，J. Ross Publishing，2004）深入地讨论了SPI以及能够帮助组织进行过程改进的度量。Ebert及其同事（《Best Practices in Software Measurement》，Springer，2004）在ISO和CMMI标准的范畴内讨论了测量的使用。Kan（《Metrics and Models in Software Quality Engineering》，2d ed.，Addison-Wesley,2002）介绍了相关度量的收集。

Ebert和Dumke（《Software Measurement》，Springer，2007）提供了当将测量和度量应用于IT项目时的有用的处理措施。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及其同事已经撰写了一部值得收藏的著作（《Software Process Improve：Metrics，Measurement，and Process Modeling》，Springer-Verlag，2001）。Florac与Carlton（《Measuring the Software Process》，Addison-Wesley，1999）以及Fenton与Pfleeger（《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。

Laird与Brennan（《Software Measurement and Estimation》，Wiley-IEEE Computer Society Press，2006）以及Goodman（《Software Metrics：Best Practices for Successful IT Management》，Rothstein Associates，Inc.，2004）讨论了将软件度量用于项目管理和估算。Putnam和Myers（《Five Core Metrics》，Dorset House，2003）利用一个包含6000多个软件项目的数据库，论证了如何使用5种核心度量——时间、工作量、规模、可靠性以及过程生产率，来控制软件项目。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，帮助组织评估自身的软件过程和实践。

功能点测量已经成为一种广泛应用于软件工程工作很多领域的技术。Parthasarathy（《Practical Software Estimation：Function Point Methods for Insourced and Outsourced Projects》，Addison-Wesley，2007）提供了综合性的指南。Garmus和Herron（《Function Point Analysis：Measurement Practices for Successful Software Projects》，Addison-Wesley，2000）讨论了侧重于功能点分析的过程度量。

关于Web工程工作的度量，已发表的资料比较少。不过，Kaushik（《Web Analytics：An Hour a Day》，Sybex，2007）、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）从商业和市场角度讨论了Web度量。

IEEE总结了度量领域的最新研究（《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。一些最新的与过程度量和项目度量相关的WWW参考文献可在SEPA Web站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第26章

Software Engineering：A Practitioner's Approach (Seventh Edition)

软件项目估算

要点浏览

概念：软件的真实需求已经确定；利益相关者们都已就绪；软件工程师准备开始工作；项目将要启动。但是如何进行下去呢？软件项目策划包括5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投入的资金、工作量、资源及时间。

人员：软件项目经理——利用从项目利益相关者那里获得的信息以及从以往项目中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过建造一所大房子，那么在开发软件之前进行估算应该是合理的。

步骤：估算首先要描述问题的范围。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。因为一直要等到项目完成时，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
估算
敏捷
经验模型
基于FP
面向对象的项目
基于问题
基于过程
调和
用例
Web应用
可行性
项目策划
软件方程
软件范围

软件项目管理从一组统称为项目策划的活动开始。在项目启动之前，软件团队应该估算将要做的工作、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，软件团队就应该制定项目进度计划。在项目进度计划中，要定义软件工程任务及里程碑，指定每一项任务的负责人，详细说明对项目进展有较大影响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，也不愿花费时间制定计划。很多技术管理者没有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]以较高的成本解决问题。一般的项目要将80%的时间花费在返工上——改正在项目前期所犯的错误。

McConnell指出，每个团队都能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。

26.1 对估算的观察

制定计划需要你做一个初始约定，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，都是在预测未来，自然要接受一定程度的不确定性。下面引用Frederick Brooks［Bro95］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。

“良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。”——Caper Jones

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进行定量估算提供了历史依据和有效输入。当建立估算和评审估算时，以往的经验（包括所有参与人员的）具有不可估量的辅助作用。由于估算是所有项目策划活动的基础，而项目策划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。

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（度量）。当只存在定性的信息时，还要有进行定量预言的勇气。估算具有与生俱来的风险[⊖]，正是这种风险导致了不确定性。

KEY POINT 项目复杂性、项目规模以及结构的不确定程度都影响着估算的可靠性。

项目的复杂性对计划固有的不确定性有很大影响。但是，复杂性是一个相对的数量，受人员在以往工作中对它的熟悉程度的影响。一个高级的电子商务应用对于首次开发者来说可能极其复杂。但是，一个Web工程团队当它第十次开发电子商务Web应用时，会认为这样的工作很普通。现在已经提出了很多软件复杂性的定量测量方法[Zus97]。这些测量方法都应用于设计层或代码层，因此在软件策划（先于设计和代码的存在）期间难以使用。但是，其他更主观的复杂性评估方法（例如，第23章描述的功能点复杂度校正系数）可以在早期的策划过程中建立。

“应该满足于事物本性所能容许的精确度，当只可能近似于准确时，不要去寻求绝对的准确。”——Aristotle

项目规模是另一个能影响估算精确度和功效的重要因素。随着规模的扩大，软件各个元素之间的相互依赖迅速上升[⊖]。问题分解，作为一种重要的估算方法，会由于问题元素的精化仍然难以完成而变得更加困难。用Murphy法则来解释：“凡事只要有可能出错，那就一定会出错。”——如果有更多事情可能失败，则这些事情一定失败。

结构的不确定程度也影响着估算的风险。这里，结构是指需求已经被固化的程度、功能被划分的容易程度以及必须处理的信息的层次特性。

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，你能仿效做过的工作，并改进出现问题的地方。如果能够取得以往项目的全面的软件度量（第25章），估算就会有更大的保证，能够合理安排进度以避免重走过去的弯路，总体风险就会降低。

估算的风险取决于资源、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。作为计划人员，你和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，你不应该被估算所困扰。现代软件工程方法（例如，演化过程模型）采用迭代开发

⊖ 系统的风险分析技术将在第28章讨论。

⊖ 当问题的需求改变时，由于“范围的蔓延”使得问题的规模常常会扩大。而项目规模的扩大对项目的成本和进度有几何级数的影响（Michael Mah，personal communication）。

方法。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。

26.2 项目策划过程

你了解的越多，就估算得越好。因此，要随着项目的进展更新你的估算。

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。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个动作。

项目计划任务集 TASK SET

1. 规定项目范围。
2. 确定可行性。
3. 分析风险（第28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. 识别环境资源。
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27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。

26.3 软件范围和可行性

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。

软件范围描述了将要交付给最终用户的功能和特性、输入和输出数据、作为使用软件的结果呈现给用户的“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技术：

1. 在与所有利益相关者交流之后，写出软件范围的叙述性描述。
2. 由最终用户开发的一组用例[⊖]。

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，因此一定程度的功能分解常常是有益的。性能方面的考虑

⊖ 在本书的第2部分中，已经对用例进行了详细的讨论。用例是从用户的观点出发来描述用户与软件的交互场景。

包括处理时间和响应时间的需求。约束条件表示外部硬件、可用存储，或其他现有系统对软件的限制。

一旦确定了软件范围（并征得用户的同意），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（或是被不耐烦的管理者或其他利益相关者催促着越过这些问题），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4个固定因素：技术——项目在技术上可行吗？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？

Putnam与Myers正确地指出，确定范围并不够。一旦理解了范围，就必须进一步确定标识的范围是否能够完成。这是估算过程至关重要的部分，但常常被忽略。

26.4 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26-1描述了3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（硬件和软件工具）。对每类资源，都要说明以下4个特征：资源描述、可用性说明、何时需要资源以及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，必须在最早的使用时间就建立资源的可用性。

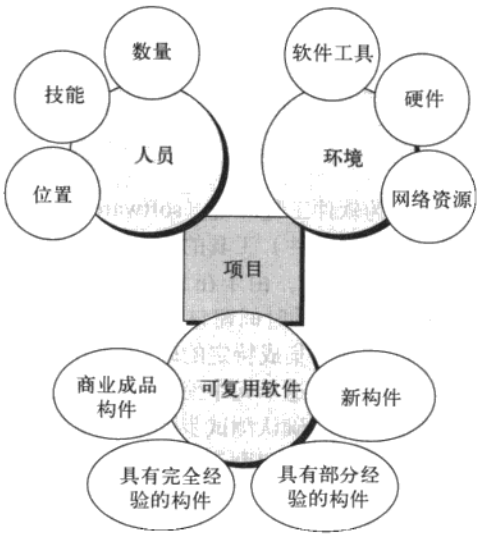


图26-1 项目资源

26.4.1 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（例如，管理人员、高级软件工程师）和专业（例如，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。

26.4.2 可复用软件资源

基于构件的软件工程（CBSE）[⊖]强调可复用性，即创建并复用软件构造块，这种构造块通常称为构件。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。Bennatan[Ben00] 建议在制定计划时应该考虑以下4种软件资源。

绝对不要忘记：多种可复用构件的集成具有相当大的挑战。更为严重的是，当升级多种构件时，集成问题常常会重新出现。

成品构件：能够从第三方或者从以往项目中获得的现成软件。商业成品构件（commercial off-the-shelf，COTS）是从第三方购买的，准备用于当前项目，并已经过完全确认的构件。

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相关的已有的规格说明、设计、代码或测试数据，但是需要做很多修改。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，直到软件过程的后期，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。

26.4.3 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。硬件提供支持（软件）工具的平台，而这些（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[⊖]。由于在大多数软件组织中，有很多人都需要使用SEE，因此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。

当软件团队构造基于计算机的系统（集成特定的硬件和软件）时，可能需要使用由其他工程团队所开发的硬件元素。例如，在为制造单元中使用的机器人装置开发软件时，可能需要特定的机器人（例如，机器人焊接工）作为确认测试步骤的一部分；在开发高级排版软件项目的过程中，在某个阶段可能需要一套高速数字印刷系统。作为计划的一部分，必须指定每一个硬件元素。

26.5 软件项目估算

软件的成本及工作量估算从来都没有成为一门精确的科学。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

⊖ 在第10章考虑CBSE。

⊖ 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。

"在外包和竞争愈加激烈的时代，更准确地进行估算的能力……已经成为很多IT组织成功的关键因素。"——Rob Thomsett

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，第一种选择不论多吸引人，都是不现实的。成本估算必须"预先"给出。不过，应该认识到，你等待的时间越久，了解得就越多，而了解得越多，在估算中出现严重错误的可能性就越小。

"不采用定量的方法、几乎没有数据支持、主要由管理人员的直觉来保证，这样就很难制定有效、可靠、防御工作风险的估算。"——Fred Brooks

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（例如，客户、商业条件、软件工程环境、交付期限）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，常常是一种有潜在价值的估算方法。一个基于经验（历史数据）的模型形式如下：

$$d = f(v_i)$$

其中，d是很多估算值（例如，工作量、成本、项目持续时间）中的一种，v_i是所选的独立参数（例如，被估算的LOC或FP）。

自动估算工具实现一种或多种分解技术或经验模型，提供了有吸引力的估算选择。使用这样的系统进行估算时，要描述开发组织的特性（如经验、环境）和待开发的软件，再由这些数据导出成本和工作量估算。

每种可行的软件成本估算方法，其效果的好坏取决于估算所使用的历史数据。如果没有历史数据，成本估算就建立在了不稳定的基础上。在第25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。

26.6 分解技术

软件项目估算是解决问题的一种形式，在多数情况下，要解决的问题（对于软件项目来说，就是成本和工作量估算）非常复杂，不能作为一个整体考虑。因此，要对问题进行分解，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24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，你必须理解待开发软件的范围，并估计其"规模"。

KEY 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。

26.6.1 软件规模估算

软件项目估算的准确性取决于许多因素：(1) 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，考虑软件规模估算的问题。由于项目估算的准确程度取决于待完成工作的规模估算，因此规模估算是计划人员面临的第一个主要挑战。在项目计划中，规模是指软件项目的可量化结果。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。

我们如何估算计划要开发的软件的规模。

Putnam和Myers[Put92]建议使用4种不同的方法来估算问题规模：

- “模糊逻辑”法。该方法使用近似推理技术，而近似推理技术是模糊逻辑的基础。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23章讨论过）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，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和目标级指令。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估算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（例如，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26.6.2中描述的式（26-1）将它们结合起来。

26.6.2 基于问题的估算

在第25章中，已经描述了代码行和功能点测量，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，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，将它们与估算变量结合使用，进行成本和工作量的估算。

基于LOC和基于FP的估算有什么共同点？

LOC估算和FP估算是两种不同的估算技术，但两者有很多共同特性。首先从界定的软件范围陈述入手，尝试将范围陈述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，也可以选择其他元素进行规模估算，例如，类或对象、变更、受影响的业务过程。

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。

然后，将基线生产率度量（例如，LOC/pm或FP/pm[⊖]）应用于适当的估算变量，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，使用单一基线的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，首先应将项目对应到某个领域中，然后，再使用适当的领域生产率平均值对其进行估算。

⊖ 缩写pm代表工作量的单位——人月（person-month）。

在分解应用问题时，LOC估算技术和FP估算技术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，分解则是不同的。它关注的不是功能，而是5个信息域特性——输入、输出、数据文件、查询和外部接口，以及在第23章讨论过的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，你都应该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），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。

我们如何计算软件规模的“期望值”？

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）S的期望值，例如：

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6} \tag{26-1}$$

其中，“可能”估算值的权重最大，并遵循b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。这个估算正确吗？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。即便如此，常识和经验还是会占优势。

26.6.3 基于LOC估算的实例

作为LOC和FP基于问题估算技术的实例，考察为机械零件计算机辅助设计（CAD）应用系统开发的软件包。该软件将在工程工作站上运行，且必须与各种计算机外部绘图设备有接口，包括鼠标、数字化仪、高分辨率彩色显示器和激光打印机。可以给出初步的软件范围陈述：

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。开发一些设计分析模块，以产生所需的输出，这些输出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（包括鼠标、数字化仪、激光打印机和绘图机），并能与外部设备进行交互。

很多现代应用或者驻留在网络上，或者是客户机/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“基础设施软件”所需的工作量。

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，以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，假定已经做了进一步的细化，确定该软件包应具有的主要软件功能，如图26-2中所示。遵照LOC的分解技术，得到如图26-2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(26-1)，得到三维几何分析功能的期望值是6800 LOC。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。

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。

回顾历史数据可以看出，这类系统的组织平均生产率是620 LOC/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[⊖]。

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26-2 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。

[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微观进度计划，为其余的增量制定宏观进度计划。

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。你从用例入手，确定实现每个用例所需要的功能，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进行功能分解。

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……

⊖ 估算单位是千美元和人月。如果给定了估算的精确度要求，更高的精确度是不必要的，也是不现实的。

26.6.4 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。分别对CAD软件的输入、输出、查询、文件和外部接口进行估算，参看图26-3给出的表。然后使用第23章讨论的技术来计算FP的值。为了进行估算，假定复杂度加权因子都取平均值。图26-3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26-3 估算信息域的值

估算出每一个复杂度加权因子，根据第23章所描述的方法计算出复杂度校正因子的值：

因子	值
备份和恢复	4
数据通信	2
分布式处理	0
关键性能	4
现有的操作环境	3
联机数据输入	4
多屏幕输入切换	5
主文件联机更新	3
信息域值复杂度	5
内部处理复杂度	5
设计可复用的代码	4
设计中的转换与安装	3
多次安装	5
易于变更的应用设计	5
复杂度校正因子	1.17

最后，得出FP的估算值：

$$FP_{estimated} = 总计 \times [0.65 + 0.01 \times \Sigma(F_i)] = 375$$

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。

26.6.5 基于过程的估算

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，并估算完成每项任务所需的工作量。

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框架活动。这些功能及其相关的框架活动[⊖]可以用表格形式给出，类似于图26-4所示。

活动	客户沟通	策划	风险分析	工程		构造发布		客户评估	合计
任务→				分析	设计	编码	测试		
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%		

CC = customer communication CE = customerevaluation

图26-4 基于过程的估算表

如果时间允许，可以使用更细的粒度来指定图26-4中所示的任务，例如，将分析分解为若干主要任务，并分别估算每一项任务。

一旦将问题功能与过程活动结合起来，就可以针对每个软件功能，估算出完成各个软件过程活动所需的工作量（如人月），这些数据构成了图26-4中表格的中心部分。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是不依赖LOC或FP估算而实现的，我们现在就已经有了两种或三种成本与工作量估算，可以进行比较和调和。如果两组估算非常一致，则有理由相信估算是可靠的。相反，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。

26.6.6 基于过程估算的实例

为了说明基于过程估算的使用方法，考虑26.6.3节介绍的CAD软件。系统配置和所有软件功能都保持不变，并已在项目范围中说明。

"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

参看图26-4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为主要的软件工程任务。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，这些数值都列在表格底部的"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%，说明这些工作相对更重要。

⊖ 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。这些框架活动是客户沟通（CC）、策划、风险分析、工程和构造/发布。

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。

26.6.7 基于用例的估算

正如本书第二部分中提到的，用例能使软件团队深入地了解软件的范围和需求。但由于以下原因，建立基于用例的估算方法还有困难[Smi99]：

为什么开发基于用例的估算技术很困难？

- 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），因此可以在很多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。
- 用例不能描述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”可能一到两天就能完成。

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，但只有在用例描述所在的“结构层次”下来考虑。

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（并代表相对更多的开发工作量）。因此，在用例能被用于估算之前，首先要建立结构层次中的层次，确定每个用例的平均长度（页数），定义软件的类型（例如，实时、商业、工程/科学、Web应用、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（层次结构中每一层的）每个用例的LOC或FP的估算值。然后，再根据历史数据计算开发系统所需的工作量。

为了说明如何进行计算，考虑以下关系式[⊜]：

$$\text{LOC估算} = N \times \text{LOC}_{\text{avg}} + [(S_a / S_h - 1) + (P_a / P_h - 1)] \times \text{LOC}_{\text{adjust}} \qquad (26\text{-}2)$$

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$\text{LOC}_{\text{avg}}$的$n$%来表示。$n$的值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（26-2）是根据实际用例数对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%。

⊖ 原文为“用例能够描述涉及很多功能和特性的复杂行为（如交互）”。——译者注

⊜ 近期在用例点[Cle06]推导方面的工作最终可能导出可操作的使用用例的估算方法。

⊜ 重要的是要注意到，式（26-2）只是用于举例说明的目的。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。

26.6.8 基于用例的估算实例

在26.6.3节介绍的CAD软件包括3个子系统组：用户界面子系统（包括UICF）、工程子系统组（包括2DGA、3DGA和DAM子系统）、基础设施子系统组（包括CGDF子系统和PCF子系统）。用户界面子系统由6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，基础设施子系统组由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。

利用式（26-2），令$n=30\%$，就产生了如图26-5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（26-2）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基础设施子系统组做出估算。在图26-5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。

	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	3100	31 233
基础设施子系统组	5	6	5	10	6	1650	7 970
LOC估算合计							42 568

图26-5 用例估算

以620 LOC/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。

26.6.9 协调不同的估算方法

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，再来考虑26.6.3节介绍的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）计划人员没有充分理解或是误解了项目范围。（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反映软件工程组织的情况），或者是误用了。应该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

自动估算工具允许计划人员估算成本和工作量，还可以对重要的项目变量（例如，交付日期或人员配置）进行假设分析。尽管目前已有很多自动估算工具（参看本章后面的补充材料），但它们具有相同的本质特性，都能实现以下6项一般性功能[Jon96]：

1. 估算项目可交付产品的规模。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（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（26.7节）来预测软件工作量，这些模型能将交付的项目规模与开发所需的工作量联系起来。

5. 预测软件成本。如果给出第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，可以根据本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做是一个“数据点”，再从中导出估算，而不是将其作为估算的唯一来源。

26.7 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FP的函数[⊖]。LOC或FP的值采用26.6.3节和26.6.4节所描述的方法进行估算，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。

估算模型反映的是它被导出所基于的项目集，因此模型具有领域敏感性。

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软件类型和开发环境。所以，从这些模型中得到的结果应该慎重使用。

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，则使用该模型前，必须对其进行调整和再次检验。

26.7.1 估算模型的结构

典型的估算模型是通过对以往软件项目中收集的数据进行回归分析而导出的。这种模型的总体结构表现为下面的形式[Mat94]：

$$E = A + B \times (e_v)^C \tag{26-3}$$

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。

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（26-3）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（例如，问题的复杂性、开发人员的经验、开发环境）加以调整。在文献中提出了很多面向LOC的估算模型：

$E = 5.2 \times (\text{KLOC})^{0.91}$ Walston-Felix模型

$E = 5.5+0.73 \times (\text{KLOC})^{1.16}$ Bailey-Basili模型

⊖ 26.6.7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，在文献中出现的非常少。

$E = 3.2 \times (\text{KLOC})^{1.05}$ Boehm简单模型

$E = 5.288 \times (\text{KLOC})^{1.047}$ Doty模型，用于KLOC＞9的情况

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

$E = -37+0.96\ \text{FP}$ Kemerer模型

$E = -12.88+0.405\ \text{FP}$ 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，必须根据项目特定环境的要求对估算模型进行调整。

26.7.2 COCOMO II模型

WebRef

关于COCOMO II的详细信息，包括可下载的软件，能够在sunset.usc.edu/research/COCOMOII/cocomo_main.html获得。

Barry Boehm[Boe81]在其关于“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模型是得到产业界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COCOMOII[Boe00]。和其前身一样，COCOMOII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。在软件工程的前期阶段使用，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。

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3种不同的规模估算选择：对象点、功能点和源代码行。

什么是“对象点”？

COCOMO II应用组装模型使用的是对象点，这将在下段中说明。应该提到，其他更复杂的估算模型（使用FP和KLOC）也会作为COCOMO II的一部分出现。

与功能点一样，对象点也是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）构造应用系统可能需要的构件数。根据Boehm[Boe96]给出的标准，复杂度分为3个等级（即简单、中等或困难），将每个对象实例（例如，一个屏幕或一个报表）分别归类到一个等级上。本质上，复杂度是以下变量的函数：客户和服务器数据表（产生屏幕显示或报表时需要它们）的数量和来源，以及视图或版面（作为屏幕或报表的一部分显示）的数量。

一旦确定了复杂度，就可以根据图26-6中的表格，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：

对象类型	复杂度加权		
	简单	中等	困难
屏幕	1	2	3
报表	2	5	8
3GL构件			10

图26-6 不同类型对象的复杂度加权（来源于[Boe96]）

$$\text{NOP} = \text{对象点} \times [(100-\text{复用的百分比}) / 100]$$

其中，NOP是新的对象点。

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26-7给出了在不同水平的开发者经验和不同开发环境成熟度下的生产率。

$$\text{PROD} = \frac{\text{NOP}}{\text{人月}}$$

一旦确定了生产率，就可以计算项目工作量的估算值：

$$\text{估算工作量} = \frac{\text{NOP}}{\text{PROD}}$$

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26-7 基于对象点的生产率（来源于[Boe96]）

在更高级的COCOMO II模型[⊖]中，还需要一系列的比例因子、成本驱动和调整过程。对这些内容的完整讨论超出了本书范围，有兴趣的读者可参见[Boe00]或访问COCOMO II的Web网站。

26.7.3 软件方程

软件方程[Put92]是一个动态的多变量模型，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多个当代软件项目中收集的生产率数据导出的。根据这些数据，我们导出以下形式的估算模型：

$$E = \frac{\text{LOC} \times B^{0.333}}{P^3} \times \frac{1}{t^4} \tag{26-4}$$

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”[⊜]。

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。

WebRef

关于从软件方程式演化而来的软件成本估算工具的信息可在www.qsm.com获得。

应该注意到，软件方程有两个独立的参数：（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。

Putnam和Myers[Put92]为了简化估算过程，并将估算模型表示成更通用的形式，他们给出了一组从软件方程式中导出来的方程式。最短开发时间定义为：

⊖ 如前所述，这些模型使用FP或KLOC数作为规模变量。

⊜ 随着“对集成、测试、质量保证、文档和管理技能的需求的增长”，B的值缓慢增加[Put92]。对于较小的程序(KLOC=5～15)，B=0.16。对于超过70 KLOC的较大程序，B=0.39。

$$t_{\min} = 8.14 \frac{\text{LOC}}{P^{0.43}}，以月为单位，用于t_{\min} > 6个月的情况 \quad (26\text{-}5a)$$

$$E = 180Bt^3，以人月为单位，用于E \geqslant 20人月的情况 \quad (26\text{-}5b)$$

注意方程式（26-5b）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（26-5），令$P = 12\,000$（对科学计算软件的推荐值）：

$$t_{\min} = 8.14 \times \frac{33\,200}{12\,000^{0.43}} = 12.6（月）$$

$$E = 180 \times 0.28 \times (1.05)^3 = 58（人月）$$

由软件方程式得到的结果与26.6节产生的估算值非常一致。同26.7.2节中提到的COCOMO模型一样，软件方程式将继续演化下去，关于该估算方法扩展版本的进一步讨论请参见[Put97b]。

26.8 面向对象项目的估算

使用明确为OO软件设计的估算技术来对软件成本估算的传统方法进行补充是值得的。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. 使用需求模型（第6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. 由需求模型确定关键类（在第6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	2.0
基于文本的用户界面	2.25
图形用户界面	2.5
复杂的图形用户界面	3.0

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。

26.9 特殊的估算技术

26.6节到26.8节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，中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[⊖]。在下面的两小节中，研究两种特殊的估算技术。

26.9.1 敏捷开发的估算

由于敏捷项目（第3章）的需求是通过一组用户场景（如极限编程中的“故事”）来定义的，所以在项目计划阶段为每个软件增量开发一个非正式的、比较严谨并有意义的估算方法是可能的。敏捷项目的估算采用分解法，包括下列步骤：

⊖ “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，而估算是可靠计划的基础。

当采用敏捷过程时，如何进行估算？

1. 从估算目的出发，分别考虑每个用户场景（由最终用户或其他利益相关者在项目初期建立，等价于一个微型用例）。

2. 将场景分解成一组开发它所需要完成的软件工程任务。

3a. 分别估算每一项任务所需的工作量。注意，可以根据历史数据、经验模型或“经验”进行估算。

KEY POINT
在敏捷项目的估算中，“规模”是指使用LOC或FP对用户场景总规模的估算。

3b. 或者，可以利用LOC、FP或其他某种面向规模的测量（如用例点）来估算场景的“规模”。

4a. 对每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）确保增量中包含的场景数与可用资源相匹配。（2）在开发增量时，为工作量的分配提供依据。

26.9.2 Web应用项目的估算

Web应用项目常常采用敏捷过程模型。利用修改过的功能点测量，配合26.9.1节中简述的步骤，就可以进行Web应用的估算了。当把功能点用于Web应用的估算时，Roetzheim[Roe00]给出了下列方法。

- 输入：每个输入屏幕或表单（例如，CGI或Java），每个维护屏幕，每个标签页（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（例如，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Web应用而言，功能点（按照上述的说明）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（例如，页数、媒体数、功能数）相关的测量（称为“预测变量”）、其Web页特性（例如，页复杂性、链接复杂性、图复杂性）、媒体特性（例如，媒体持续时间）以及功能特性（例如，代码长度、复用的代码长度）来确定Web应用的规模。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（例如，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：[⊖]

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Galorath Inc.(www.galorath.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”。

26.10 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26.4.2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成本来确定软件获取的步骤。在有些情况下（例如，低成本的PC软件），购买并试用的方式比购买软件包并对其做冗长评估成本要低。在最后的分析中，自行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低？（3）外部支持（例如，维护合同）的成本是否比内部支持的成本低？这些条件可以应用于上述每种可选的获取方案中。

26.10.1 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步骤进行扩充，如决策树分析[⊖]。图26-8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）复用现有的“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讨论的估算技术进行估算：一项难度较大的开发工作将

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

⊖ 在http://en.wikipedia.org/wiki/Decision_tree上能找到关于决策树分析的有价值的介绍。

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，得到成本的预期值是：

$$\text{预期成本} = \Sigma(\text{路径概率})_i \times (\text{估算的路径成本})_i$$

其中，i是决策树的某条路径。对于“构建系统”这条路径而言：

$$\text{预期成本}_{\text{构建}} = 0.30\ (380\text{K}) + 0.70\ (450\text{K}) = 429\text{K}\ (\text{K表示千美元})$$

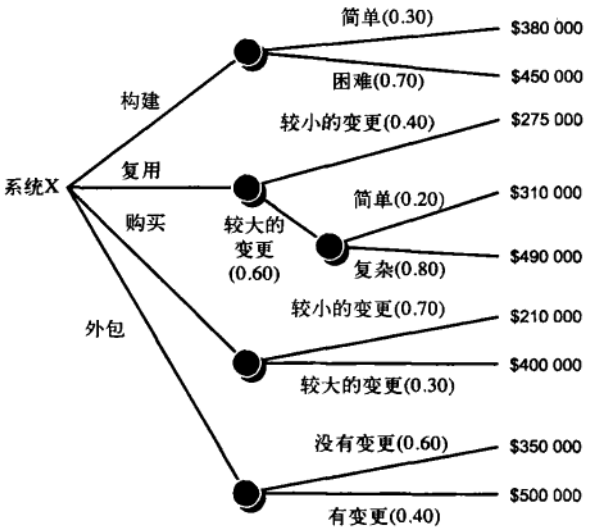


图26-8 一个支持自行开发/购买决策的决策树

沿着决策树的其他路径，分别给出了多种情况下，“复用”、“购买”和“外包”方式的项目成本。这些路径的预期成本分别是：

$$\text{预期成本}_{\text{复用}} = 0.40\ (275\text{K}) + 0.60\ [0.20\ (310\text{K}) + 0.80\ (490\text{K})] = 382\text{K}$$

$$\text{预期成本}_{\text{购买}} = 0.70\ (210\text{K}) + 0.30\ (400\text{K}) = 267\text{K}$$

$$\text{预期成本}_{\text{外包}} = 0.60\ (350\text{K}) + 0.40\ (500\text{K}) = 410\text{K}\ (\text{K表示千美元})$$

根据图26-8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准则（而不仅仅是成本）必须加以考虑。在最终决定使用构建、复用、购买或外包方式时，可用性、开发者/厂家/承包商的经验、与需求的一致性、本地的政策环境及变更的可能性等，这些都可能是影响判断的准则，当然这些也仅仅是其中一部分准则而已。

26.10.2 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，但对于这个问题的直接回答经常是一句话：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[⊖]。

⊖ 在广义上，可以将外包看做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。

"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。

不考虑太多其他因素，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（例如，计算机、基础设施），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、服务和产品。这样，公司就会冒着将其竞争命运交到第三方手中的风险。

毫无疑问，向外包发展的趋势将会继续，减缓这种趋势的唯一方法是让人们认识到所有层次上的软件工作都具有强烈的竞争。生存的唯一办法就是具有与外包厂商同等的竞争力。

SAFEHOME

外包

[场景] 项目初期，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：我们正在考虑将产品的SafeHome软件工程部分外包出去。

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，这些人非常受推崇。

Doug（深深地吸一口气并努力保持镇静）：你们让我很惊讶。不过在做最后决定之前，先听听几条意见吧。

Joe（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新版本。我们有理由假定软件将提供很多新特性，对吗？

[都点头]

Doug：我们以前曾经协调过一个国际项目吗？

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。我们拥有专门的技术，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，然后再开会做最后决定。Doug，我可以和你私下谈谈吗？

Doug：当然……我实在是想确保我们做的事情一切顺利。

26.11 小结

在项目开始之前，软件项目计划人员必须先估算3件事：花费多长时间，需要多少工作量，涉及多少人员。此外，计划人员还必须预测所需要的资源（硬件和软件）及蕴涵的风险。

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软件的主要功能，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；（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值。可以使用自动工具来实现特定的经验模型。

对项目做精确估算时，一般至少要用到上述3种技术中的两种。通过对不同技术产生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。

习题与思考题

26.1 假设你是一家开发家用机器人软件公司的项目经理，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，你也可以选择其他感兴趣的问题，而不做草坪割草机器人。

26.2 在26.1节简要讨论了软件项目的复杂性。列出影响项目复杂性的软件特性（例如，并发操作、图形输出），按其对项目的影响程度顺次排列。

26.3 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。

26.4 对你在问题26.1中描述的机器人软件进行功能分解。估算每个功能的规模（用LOC）。假定你所在组织的平均生产率是450 LOC/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。

26.5 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。

26.6 使用“软件方程”来估算草坪割草机器人软件。假设采用方程（26-4），且$P=8000$。

26.7 比较问题26.4和问题26.6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？

26.8 使用问题26.7中得到的结果，确定能否期望该软件在6个月内完成，以及完成该工作需要多少人员？

26.9 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估算模型。

26.10 组建一个项目团队，开发软件工具来实现本章所介绍的每种估算技术。

26.11 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之前进行。你认为为什么会这样？是否存在不需要这样做的情况？

26.12 假设每个分支的概率均为50%，重新计算图26-8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》，3rd ed.，McGraw-Hill，2000）、Bennatan（《On Time，Within Budget：Software Project Management Practices and Techniques》，3rd ed.，Wiley，2000）和Phillips[Phi98]都提供了有用的估算指南。

McConnell（《Software Estimation：Demystifying the Black Art》，Microsoft Press，2006）撰写了实用指南，为那些必须估算软件成本的人提供了有价值的指导。Parthasarathy（《Practical Software Estimation》，Addison-Wesley，2007）强调功能点作为估算度量。Laird与Brennan（《Software Measurement and Estimation：A Practical Approach》，Wiley-IEEE Computer Society Press，2006）论述了测量及其在软件估算中的应用。Pfleeger（《Software Cost Estimation and Sizing Methods,Issues and Guidelines》，RAND Corporation，2005）给出了一个浓缩的指南，描述了估算的很多基本原理。Jones（《Estimating Software Costs》，2nd ed.，McGraw-Hill，2007）撰写的著作是对模型和数据最全面的论述之一，这些模型和数据可用于各个应用领域的软件估算。Coombs（《IT Project Estimation》，Cambridge University Press，2002）以及Roetzheim与Beasley（《Software Project Cost and Schedule Estimating：Best Practices》，Prentice-Hall，1997）介绍了很多有用的模型，并逐步给出了生成最可能估算的指南。

大量的关于软件估算的信息源可以在因特网上获得。一个最新的与软件估算相关的WWW参考文献列表可在SEPA Web站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

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量，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。任务网络创建完成之后，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。

人员及责任：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：软件过程模型中规定的软件工程任务要根据具体实现的功能进行细化；为每一个任务分配工作量和工期；创建任务网络（也称为"活动网络"），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：正确的进度安排要求：(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
关键路径
挣值
工作量分配
人员与工作量
WebApp进度安排原则
任务网络
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。"

"你觉得在最后期限之前能完成吗？"老板问道。

“没问题，”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[⊖]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50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，这是为什么呢？

27.1 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，由软件团队以外的某个人制定，并强加给软件团队的管理者和开发者。
- 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，没有考虑到可预测的和（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。

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26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]。

“我热爱最后期限，我喜欢它们飞过时所发出的声音。”——Douglas Adams

举例说明，假定你负责的软件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。在你进行了仔细的估算和风险分析（第28章）之后得到的结论是：在现有人员条件下，需要14个月的时间才能完成这一软件。你下一步该怎么办呢？

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：

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（第2章）制定软件工程策略，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文档。

⊖ 你可能觉得惊奇，但这个故事是我自己经历过的事。

当管理者要求的项目最后期限无法实现时，应该怎么办？

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[⊖]。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。”

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[⊖]；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。

27.2 项目进度的安排

为达到项目管理者的目标所必须完成的任务不应该手工完成，有很多优秀的项目进度安排工具可供使用。

曾经有人向Fred Brooks请教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日期。而有些任务则是位于“关键路径”之上的，如果这些“关键”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确定所有的项目任务，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。

“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

软件项目进度安排（software project scheduling）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，建立的是一张宏观进度表，该进度表标识了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）软件活动和

⊖ 如果生产率提高10%～25%，实际上是有可能完成这个项目的。但是大多数情况是，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。

⊖ 你还可以补充说：人员数量的增加不会成比例地缩短完成该项目所需要的时间。

任务，同时也进行了进度安排。

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。

27.2.1 基本原则

KEY POINT
在进度安排时，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。

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（compartmentalization）。必须将项目划分成多个可以管理的活动和任务。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（Interdependency）。划分后的各个活动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（time allocation）。每个要进行进度安排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（effort validation）。每个项目都有预定人员数量的软件团队参与。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[⊖]）。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。

确定责任（defined responsibilities）。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。

明确输出结果（defined outcomes）。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（defined milestones）。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（第15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。

27.2.2 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！）

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，我们也总是可以在项目后期增加更多的程序员来跟上进度。"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

⊖ 实际上，由于与工作无关的会议、病假、休假以及各种其他原因，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。

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。

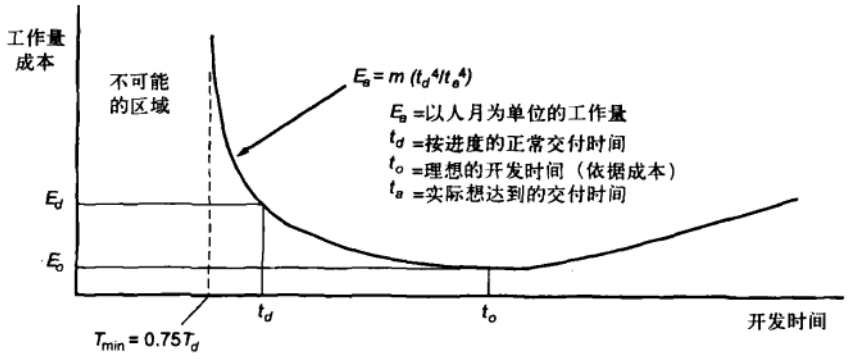


图27-1 工作量和交付时间的关系

ADVICE
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。

除去学习系统所需的时间之外，整个项目中，新加入人员将会增加人员之间交流的路径数量和交流的复杂度。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）。

KEY POINT
PNR曲线表明，拖延项目交付时间可以显著地降低项目成本。

PNR（Putnam-Norden-Rayleigh）曲线[⊖]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27-1所示。图中的t_o表示项目最低交付成本所需的最少时间（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。

举一个例子，假设一个软件项目团队根据进度安排和现有的人员配置，估算所需要的工作量应为E_d，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26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公式表示：

$$L = P \times E^{1/3}\ t^{4/3}$$

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，可以得到开发工作量E的计算公式：

$$E = L^3/(P^3\ t^4) \qquad (27\text{-}1)$$

⊖ 有关的初始研究参见[NOR70]和[PUT78]。

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，确定新的交付日期。

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 源代码行和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，则由式（27-1）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。

27.2.3 工作量分配

在第26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。

在软件过程的工作流程中应如何分配工作量？

这种工作量分配方法只能作为指导原则[⊖]。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。客户交流与需求分析大约占用10%～25%的项目工作量，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。

27.3 为软件项目定义任务集

无论选择哪一种过程模型，一个软件团队要完成的工作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，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，任务集中包含了为完成某个特定项目所必须完成的所有软件工程工作任务、里程碑、工作产品以及质量保证过滤器。为了获得高质量的软件产品，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：

1. 概念开发项目（concept development projects），目的是为了探索某些新的业务概念或者

⊖ 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超过总工作量的40%。而相反的是，敏捷开发的倡导者（第3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。

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.rspa.com/apm。

某种新技术的应用。

2. 新应用系统开发（new application development）项目，根据特定的客户需求而承担的项目。

3. 应用系统增强（application enhancement）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. 应用系统维护项目（application maintenance projects），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、修改或者扩展。

5. 再工程项目（reengineering projects），为了全部或部分重建一个现有（遗留）系统而承担的项目。

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。[PRE05]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、项目人员配置以及再工程因素等。综合考虑这些因素就形成了严格程度（degree of rigor）的指标，它将应用于所采用的软件过程中。

27.3.1 任务集举例

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活动：

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。**确定为承担项目范围所涵盖的工作组织应具有的工作能力。

1.3 **技术风险评估。**评估与项目范围中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，而且当将概念推荐给其他客户或管理者时能够用于“营销”目的。

1.6 **客户反应。**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。

快速浏览以上活动，你应该不会有任何诧异。实际上概念开发项目的软件工程流程（以及其他所有类型的项目）与人们的常识相差无几。

27.3.2 软件工程活动求精

在上一小节中所描述的软件工程活动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，以创建详细的项目进度表。细化工作始于将每个行动分解为一组任务（以及相关的工作产品和里程碑）。

这里以“行动1.1—确定概念范围”任务分解为例。任务求精可以使用大纲格式，但是在这里将使用过程设计语言方法来说明“确定概念范围”这一行动的流程：

```
任务定义：行动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 TR：评审需求的书面描述[⊖]；
    1.1.2.2 导出客户可见的输出/输入列表；
```

⊖ TR表示在此需要进行一次技术评审（第15章）。

1.1.2.3 TR：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 TR：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 TR：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；

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”；

结束行动1.1的任务定义

在过程设计语言中标注的这些任务和子任务共同构成了“确定概念范围”这个行动的详细进度表的基础。

27.4 定义任务网络

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。

单个任务和子任务之间顺序上存在相互依赖的关系。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。

任务网络（task network），也称为活动网络（activity network），是项目任务流程的图形表示。有时将任务网络作为向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活动。概念开发项目的任务网络示意图如图27-2所示。

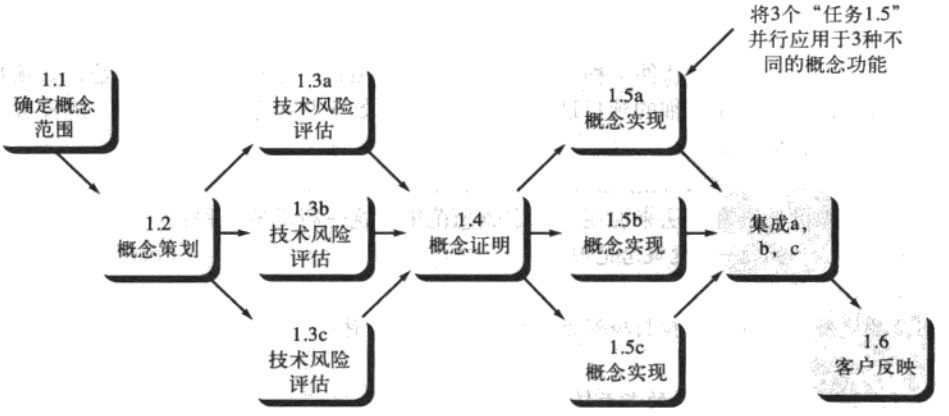


图27-2 概念开发项目的任务网络

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

以项目管理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（critical path）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。在本章的后面部分将详细讨论这些问题。

值得注意的是，图27-2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27-2所示的各个活动加以扩展。例如，应该扩展任务1.1，以表现27.3.2节所述的行动1.1求精中的所有任务。

27.5 进度安排

"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》

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。

进度计划评估及评审技术（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，早期的项目计划活动包括：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，以及所选择的任务的分解。

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。

PERT和CPM两种方法都是定量划分的工具，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）基于统计模型为单个任务进行"最有可能"的时间估算；（3）为特定任务的时间"窗口"计算"边界时间"。

SOFTWARE TOOLS

项目进度安排

目标：项目进度安排工具的目标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，并能够形成各种图表，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各任务的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和结束日期、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程进行评估。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：⊖

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。

4C，由4C Systems（www.4csys.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.infogoal.com/pmc/pmcswr.htm。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

27.5.1 时序图

在创建软件项目进度表时，计划者可以从一组任务（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。

输入信息之后，就可以生成时序图（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。

通过时序图可以确定在特定时间点将进行什么任务。

图27-3给出了时序图的格式，该图描述了字处理（word-processing, WP）软件产品中“确定概念范围”这一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”）都在左边栏中列出。水平条表示各个任务的工期，当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。

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（project tables）——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27-4）。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。

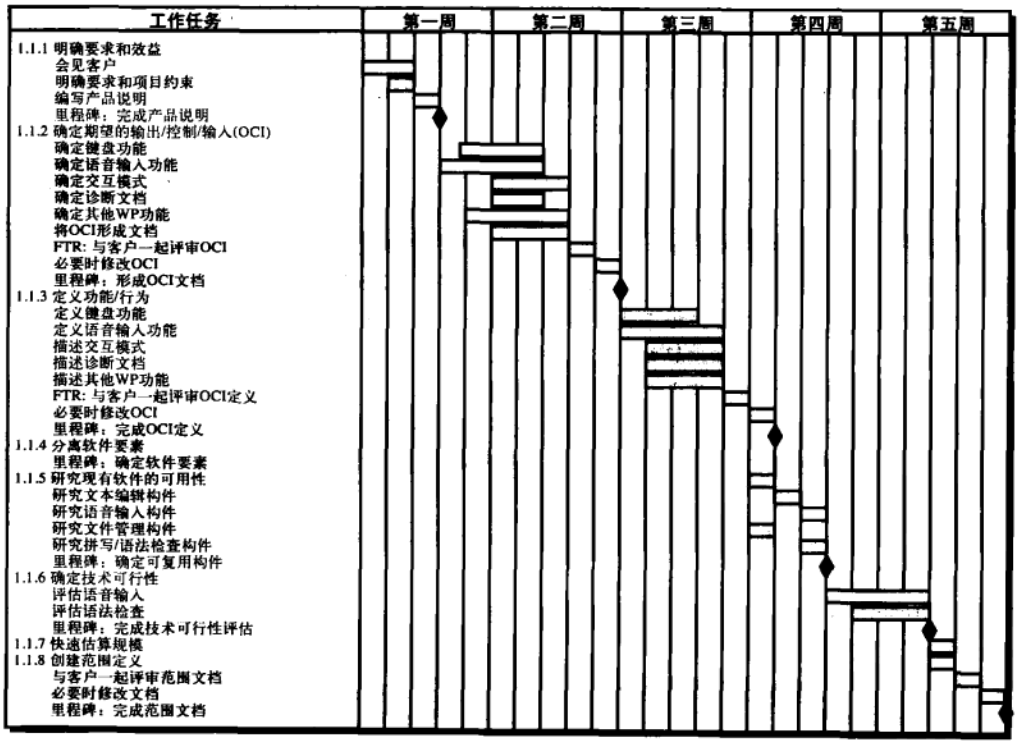


图27-3 一个时序图的例子

工作任务	计划开始	实际开始	计划完成	实际完成	人员分配	工作量分配	备注
1.1.1 I明确要求和效益							范围定义需要更多的工作量/时间
会见客户	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
明确要求和项目约束	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
编写产品说明	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
里程碑：完成产品说明	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
1.1.2 确定期望的输出/控制/输入(OCI)							
确定键盘功能	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
确定语音输入功能	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
确定交互模式	wk2, d1		wk2, d3		MLL	1 p-d	
确定诊断文档	wk2, d1		wk2, d2		BLS	1.5 p-d	
确定其他WP功能	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
将OCI形成文档	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: 与客户一起评审OCI	wk2, d3		wk2, d3		all	3 p-d	
必要时修改OCI	wk2, d4		wk2, d4		all	3 p-d	
里程碑：形成OCI文档	wk2, d5		wk2, d5				
1.1.3 定义功能/行为							

图27-4 一个项目表的例子

27.5.2 跟踪进度

"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

如果制定正确，项目进度表应该成为一个能够确定在项目进展过程中跟踪和控制任务及里程碑的线路图。项目跟踪可以通过以下方式实现：

- 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的问题。
- 评估在软件工程过程中所进行的所有评审的结果。
- 判断正式的项目里程碑（图27-3中的菱形）是否在预定日期内完成。
- 比较项目表（图27-4）中列出的各项任务的实际开始日期与计划开始日期。
- 与开发人员进行非正式会谈，获取他们对项目进展及可能出现的问题的客观评估。
- 通过挣值分析（第27.6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。

项目进展的最佳指标就是所定义的软件工作产品的实现和成功评审。

软件项目管理者通过施加控制来管理项目资源、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：雇用新员工或者重新安排项目进度。

当时间盒上的任务到达了预定的完成日期时，该项任务停止，下一任务开始。

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）[Jal04]的项目进度安排与控制技术。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2章），并为每个增量的交付制定各自的进度表。

接着，对与每个增量相关的任务实行时间盒技术。也就是按该增量的交付日期向后进行推算来调整各个任务的进度。将各个任务放入相应的"盒子"中，当一个任务触及其时间盒边界时（±10%的范围内），则该项任务停止，下一任务开始。

对时间盒方法的最初反应通常是消极的："如果工作尚未完成，我们该如何继续？"这个

问题的答案在于完成工作的方式。当遇到时间盒的边界时，很可能已经完成了任务的90%[⊖]，余下10%的工作尽管重要，但是可以（1）推迟到下一个增量中；或（2）在以后需要时再完成。项目朝着交付日期推进，而不是“卡”在某项任务上。

27.5.3 跟踪OO项目的进展

虽然迭代模型是最好的OO项目框架，但是，任务的并行性使得OO项目很难跟踪。困难在于项目管理者很难为OO项目建立有意义的里程碑，因为很多不同事物都是同时发生的。通常，有相应的准则来衡量下列主要的里程碑是否已经“完成”。

技术里程碑：OO分析完成

- 已经定义和评审了所有的类和类层次。
- 已经定义和评审了与每一个类相关的属性和操作。
- 已经建立和评审了各类之间的关系（第6章）。
- 已经建立和评审了行为模型（第7章）。
- 已经确定可复用的类。

技术里程碑：OO设计完成

- 已经确定和评审了子系统集合。
- 各类已经分配给相应的子系统，并且已经通过评审。
- 已经建立和评审了任务分配。
- 已经明确责任和协作。
- 已经设计和评审了属性和操作。
- 已经创建和评审了通信模型。

技术里程碑：OO程序设计完成

- 按照设计模型，每一个新类都已经编码实现。
- （从可复用库中）提取的类已经实现。
- 已经构建了原型或增量。

技术里程碑：OO测试

调试和测试是相辅相成的，通常可以通过考察“公开的”错误（缺陷）类型和数量来判断调试的状态。

- 已经评审了OO分析和设计模型的正确性和完整性。
- 已经建立和评审了类－责任－协作网络（第6章）。
- 已经设计了测试用例，并且已经对每个类进行了类级测试（第19章）。
- 已经设计了测试用例，并且已经完成簇测试（第19章），已经完成类的集成。
- 已经完成系统级测试。

就像前面介绍的，建立OO过程模型是迭代方式进行的，在交付不同的增量给用户时，上述的每一个里程碑都可以进行修订。

27.5.4 WebApp项目进度安排

WebApp项目进度安排（WebApp project scheduling）就是将所估算的工作量分配到构建各个WebApp增量的计划时序（项目工期）内，这可以通过将工作量分配给那些特定的任务来完成。但要注意的是，整个WebApp进度是随时间而不断演化的。刚开始的时候，建立的是一张宏观进度表，该进度表标识了所有的WebApp增量以及每一个增量的计划部署日期。随着增量

⊖ 爱冷嘲热讽的人也许会想起一句谚语：“完成系统的前90%需要90%的时间，完成剩下的10%也要用90%的时间”。

开发的进展，宏观进度表中与各个增量对应的条目会被细化成详细的进度表，这样就标识了（完成一个活动所必须实现的）特定开发任务，同时也进行了进度安排。

这里以SafeHomeAssured.com WebApp的宏观进度安排为例。就像前面介绍的，基于Web的项目构件可以定义为7个增量：

增量1：企业基本情况与产品基本信息

增量2：详细的产品信息与下载

增量3：产品报价与产品订单处理

增量4：空间布局与安全系统设计

增量5：监控服务信息与监控服务排序

增量6：监控设备的联机控制

增量7：账户信息访问

软件团队与利益相关者共同磋商后，制定了针对这7个增量的初步的（preliminary）部署进度表。与该进度表对应的时序图如图27-5所示。

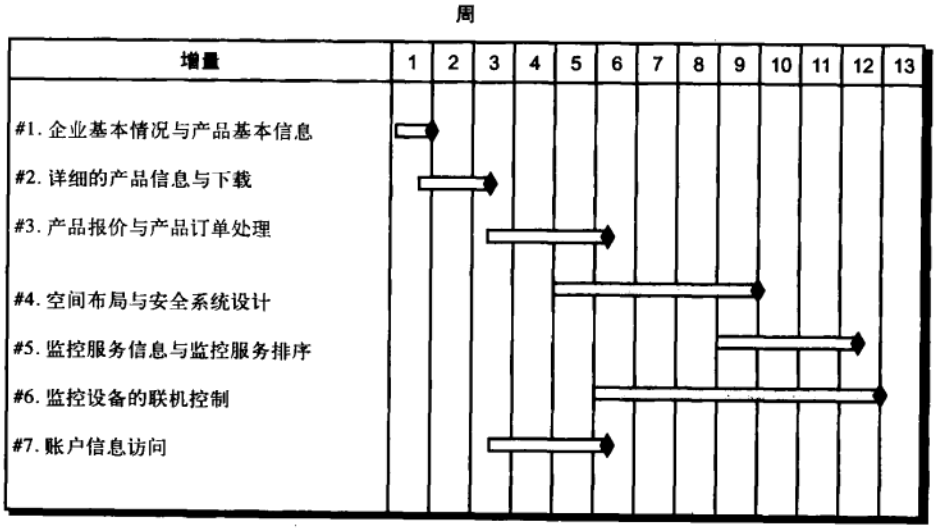


图27-5 宏观项目进度表时序图

要注意的是，这里的部署日期（时序图中的菱形）只是大概的日期，在对增量做更详细的进度安排时可以进行修改。根据这个宏观进度表，管理者可以了解到：什么时候可以得到内容和功能以及整个项目什么时候可以完成。作为初步估算，软件团队将在12周的时间内部署所有增量。从图中还可以看出有些增量是并行开发的（例如，增量3、4、6和7），这里假定软件团队有足够的人员来完成并发的开发工作。

完成宏观进度表后，软件团队就可以开始为特定的增量安排工作任务。计划者可以采用适用于任何一个WebApp增量的通用过程框架来完成这项工作。首先，将框架中的通用任务提取出来组成任务列表（task list），然后，根据特定WebApp增量要提取的内容和功能来处理这些任务。

每一个框架行动（及其相关的任务）可以按下面4种方式之一进行处理：（1）应用任务；（2）删除任务，该任务对这个增量来说不重要；（3）增加新的（自定义）任务；（4）将任务求精（详细描述）为多个命名的子任务，并将这些子任务写入进度表。

例如，一般的WebApp设计建模（design modeling）行动可以通过下列部分或全部任务来实现：

- WebApp美学设计
- 界面设计
- 导航流程设计
- WebApp体系结构设计
- 能够支持WebApp的内容与结构设计
- 功能构件设计
- 适当的安全与保密机制设计
- 设计评审

这里以SafeHomeAssured.com第4个增量的“界面设计”任务为例。就像前面介绍的，第4个增量要实现能够描述居住空间或营业空间的内容及功能，这些居住空间或营业空间将受到SafeHome安全系统的保护。根据图27-5，第4个增量从第5周初开始，第9周末结束。

毫无疑问，必须完成“界面设计”任务。软件团队意识到界面设计是该增量能否成功的关键，因此决定对这个任务进行求精（详细描述）。下列子任务全部来源于第4个增量的“界面设计”任务：

- 绘制空间设计页面的页面布局草图
- 与利益相关者一起评审布局
- 空间布局导航机制设计
- “绘图板”布局[⊖]设计
- 给出墙壁绘图功能的程序细节；
- 给出墙壁的长度计算与显示功能的程序细节；
- 给出窗户绘图功能的程序细节；
- 给出门绘图功能的程序细节；
- 选择安全系统构件（传感器、摄像机、麦克风等）的机制设计
- 给出安全系统构件绘图功能的程序细节；
- 必要时两人一组进行走查。

将这些子任务写入WebApp第4个增量的进度表，并分配到整个增量开发进度表中，这样就可以将这些子任务输入到进度安排软件中以在跟踪和控制时使用。

SAFEHOME

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。

⊖ 在这里，软件团队可以想象一下通过绘图功能真实地绘出空间的墙壁、窗户和门。墙壁线通常与网格点“对齐”，墙壁的大小会自动显示，窗户和门可以通过图形定位。空间创建好之后，最终用户还可以选择特定的传感器、摄像机等，并确定这些设备在空间内的位置。

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TR（技术评审，第15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。

27.6 挣值分析

KEY POINT

挣值提供了定量的项目进展指标。

在27.5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，对所提供信息的评估在某种程度上是主观的。那么当软件团队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，称为挣值分析（earned value analysis，EVA）。Humphrey[Hum95]对挣值给出了如下讨论：

不管要完成何种类型的工作，挣值系统为每个（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挣值。

更简单地说，挣值是对项目进展的测量。它使得计划者能够不依赖于感觉，采用定量的分析方法来评估一个项目的“完成百分比”。事实上，Fleming和Koppleman[Fle98]就认为挣值分析“早在项目进展的前15%就提供了精确而可靠的性能数据”。按照以下步骤可以确定挣值：

? 如何计算挣值以评估项目进展？

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，

$$\text{对所有任务}k,\ BAC = \sum(BCWS_k)$$

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：

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。

WebRef

大量有关挣值分析的信息源见www.acq.osd.mil/pm/。

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。

也可以计算出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。

就像超地平线的雷达一样，挣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。

27.7 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及风险分析相结合，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、每一项任务与其他任务之间的依赖关系以及计划工期。任务网络可以用来确定项目的关键路径、时序图以及各种项目信息。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。

习题与思考题

27.1 “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？

27.2 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？

27.3 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。

27.4 当多个人员参与软件项目时，就有可能产生“交流开销”。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/月），最终导致整个团队生产率下降。举几个例子（定量）说明非常精通软件工程实践和运用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。

27.5 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？请说明。

27.6 人员和时间的关系是高度非线性的。使用Putnam的软件方程（见27.2.2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，B=0.37）。假定该软件必须在24±12个月的时间期限内交付。

27.7 假定你要为一所大学开发一个联机课程登记系统（online course registration system, 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26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。

27.8 为OLCRS项目选择适当的任务集。

27.9 为27.7题中的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，并为每一项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。

27.10 如果有自动进度安排工具，请为27.9题中定义的任务网络确定关键路径。

27.11 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。

27.12 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挣值统计。这个项目共计划了56个工作任务，估计需要582人日才能完成。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），请你做出挣值分析。

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、进度偏差SV、预定完成百分比、完成百分比、成本执行指标CPI和成本偏差CV。

推荐读物与阅读信息

事实上，几乎每一本有关软件项目管理的书都涉及进度安排，Wysoki（《Effective Project Management》，Wiley，2006）、Lewis（《Project Planning Scheduling and Control，4th edition》，McGraw-Hill，2006）、Lucky和Philips（《Software Project Management for Dummies》，For Dummies，2006）、Kerzner（《Project Management：A Systems Approach to Planning，Scheduling，and Controlling，9th edition》，Wiley，2005）、Hughes（《Software Project Management》，McGraw-Hill，2005）、Project Management Institute（《PMBOK Guide，3rd edition》，PMI，2004）、Lewin（《Better Software Project Management》，Wiley，2001）以及Bennatan（《On Time，Within Budget：Software Project Management Practices and Techniques，3rd edition》，Wiley，2000）都详细讨论了这一主题。虽然Harris（《Planning

and Scheduling Using Microsoft Office Project 2007》, Eastwood Harris Pty Ltd., 2007）只是一个应用特例，但是深入讨论了如何有效利用进度安排工具对软件项目进行跟踪和控制。

Fleming和Koppelman（《Earned Value Project Management, 3rd edition》, Project Management Institute Publications, 2006）、Budd（《A Practical Guide to Earned Value Project Management》, Management Concepts, 2005）以及Webb和Wake（《Using Earned Value: A Project Manager's Guide》, Ashgate Publishing, 2003）较详细地讨论了挣值技术在项目计划、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。最新的关于项目进度安排的WWW参考列表见SEPA的Web站点www.mhhe. com/engcs/compsci/pressman/professional/olc/ser.htm。

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辅助软件团队理解和管理不确定事物的活动。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，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。

人员及责任：软件过程中涉及的每一个人——管理者、软件工程师和利益相关者——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发生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，应该经过对人员、产品、过程和项目的彻底研究后再确定。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
风险分类
风险显露度
风险条目检查表
风险表
RMMM
安全和灾难
策略
　主动
　被动

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。未来是项目管理者所关心的——什么样的风险会导致软件项目彻底失败？改变也是项目管理者所关心的——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，项目管理者必须抓住选择机会——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。

28.1 被动风险策略和主动风险策略

"如果你不主动进攻风险，风险将会主动进攻你。" —— Tom Gilb

被动风险策略（reactive risk strategies）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。

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"（fire-fighting mode）。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（proactive）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，将讨论风险管理的主动策略。

28.2 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（uncertainty）——风险可能发生也可能不发生；即，没有100%会发生的风险[⊖]。损失（loss）——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。

在建造软件时可能会遇到什么类型的风险？

项目风险（project risks）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、利益相关者、需求等方面的潜在问题以及它们对软件项目的影响。在第26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（technical risks）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。

"不经历实际风险的项目是不可能成功的。这种项目几乎是无益的，否则早就有人开发了。" —— Tom DeMarco和Tim Lister

商业风险（business risks）威胁到要开发软件的生存能力，且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的保证（预算风险）。

⊖ 100%发生的风险是强加在软件项目上的约束。

应该注意的是，单一的风险分类并不总是行得通，有些风险根本无法事先预测。

另一种常用的风险分类方式是由Charette [Cha89]提出的。已知风险（known risks）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（predictable risks）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（unpredictable risks）就像纸牌中的大王，它们可能会真的出现，但很难事先识别。

INFO

风险管理的7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——考虑将来可能发生的风险（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。任何时候都要鼓励利益相关者和用户提出风险。

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——如果所有利益相关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，要汇聚所有利益相关者的智慧、技能和知识。

28.3 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。识别出已知风险和可预测风险后，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。

ADVICE

虽然考虑一般风险很重要，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。

28.2节中提出的每一类风险又可以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（generic risks）对每一个软件项目而言都是潜在的威胁。产品特定的风险（product-specific risks）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？”

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风险和可预测风险：

- 产品规模（product size）——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（business impact）——与管理者或市场所施加的约束相关的风险。
- 客户特性（stakeholder characteristic）——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（process definition）——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（development environment）——与用来开发产品的工具的可得性及质量相关的风险。
- 开发技术（technology to be built）——与待开发软件的复杂性及系统所包含技术的“新奇性”相关的风险。
- 人员才干及经验（staff size and experience）——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，项目管理者就可以估计风险产生的影响。也可以采用另一种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。

Web上有很多针对软件项目风险的检查表（例如，[Baa07]，[Nas07]，[Wor04]），项目管理者可以利用这些检查表来提高识别软件项目一般风险的洞察力。

28.3.1 评估整体项目风险

下面的提问来源于对世界各地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。

我们正在进行的软件项目面临严重的风险吗？

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或产品热心支持吗？
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. 项目团队的人员数量满足项目需要吗？
11. 所有的客户或用户对项目的重要性和待开发的系统或产品的需求有共识吗？

WebRef

风险雷达（risk radar）是一种数据库，也是一种工具，它可以帮助项目管理人员识别、排序和交流项目风险，见www.spmn.com。

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。

28.3.2 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，风险因素包括：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：

“风险管理是针对成年人的项目管理”。——Tim Lister

- 性能风险（performance risks）——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（cost risks）——能够维持项目预算的不确定程度。
- 支持风险（support risks）——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（schedule risks）——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28-1[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。

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、影响和（或）可以补救的进度延误上的小问题，预计超支$1K到$100K	
	2	技术性能有些降低	较敏感的软件支持	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		失误对进度和（或）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28-1 影响评估

资料来源：[Boe89]

28.4 风险预测

风险预测（risk projection），又称风险估计（risk estimation），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）如果风险发生，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. 估算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。

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

尽力思考你将要开发的软件，并问自己："估计会出什么问题？"建立你自己的列表，并要求团队的其他成员也这么做。

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。

28.4.1 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[⊖]。风险表样本如图28-2所示。

项目管理者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28.3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（例如，PS指产品规模风险，BU指商业影响风险）。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首先由团队成员各自估算，然后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。

风险	风险类型	发生概率	影响值	RMMM
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	
交付期限太紧	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	
Σ				
Σ				
Σ				

影响值：
1—灾难的
2—严重的
3—轻微的
4—可忽略的

图28-2 排序前的风险表样本

下一步是评估每个风险所产生的影响。使用图28-1所示的特性评估每个风险因素，并确定其影响类别。将4个风险因素——性能、支持、成本及进度——的影响类别求平均[⊜]可得到一个整体的影响值。

风险表中应该按照概率和影响值对风险进行排序。

完成了风险表的前4列内容之后，就可以按照概率和影响值来进行排序。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。

项目管理者研究排序后的表，然后定义一条中截线。该中截线（cutoff line，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，而在中截线之下的风险则需要重新评估以进行第二次排

⊖ 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。

⊜ 如果某个风险因素对项目来说比较重要，可以使用加权平均法。

序。参照图28-3，从管理关注的角度来看，风险的影响和发生的概率是截然不同的。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。

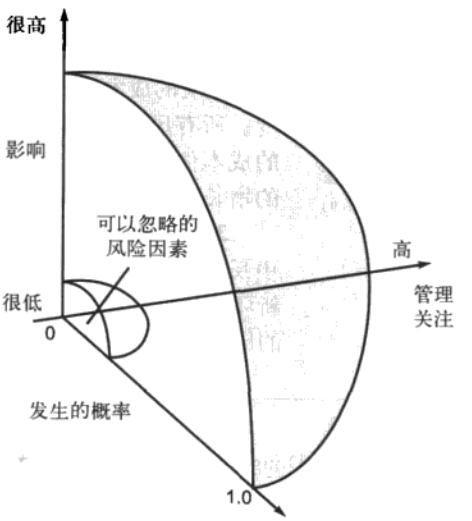


图28-3 风险与管理

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计划）或一组风险信息表单。RMMM计划和风险信息表单将分别在28.5节和28.6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

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。虽然这个方法是可行的，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等级，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概率等级与数学上的概率值相关联（如概率为0.7～0.99表示极可能发生的风险）。

28.4.2 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，但在某些情况下则是越迟越好。

如何评估风险产生的后果？

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的整体影响：（1）确定每个风险因素发生的平均概率。（2）使用图28-1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了项目风险：

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

将所有风险的RE与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。

LOC的软件工程成本是$14.00，开发构件的总成本（影响）将是18 × 100 × 14 = $25 200。

风险显露度。RE = 0.80 × $25 200 ~$20 200

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。

28.4.1节和28.4.2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28.3节中给出的类型。]

Vinod：嗯……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，每个人都在写着。）

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。一会儿我们还要精简这个列表。

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。明天我们继续开会讨论中截线。现在，回去继续工作……工作之余考虑是否还有遗漏的风险。

28.5 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。

用什么好方式来描述风险?

实现方法之一是按条件-转化-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（可能）将导致：<结果>

使用CTC格式，在28.4.2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。

28.6 风险缓解、监测和管理

"我采取如此多的预防措施，是因为我不想有任何冒险。"——Napoleon

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（risk mitigation）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），并预测影响x_1为严重的。也就是说，频繁的人员变动将对项目成本及进度有严重的影响。

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：

如何缓解风险?

- 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，设法缓解那些我们能够控制的起因。
- 项目启动之后，假设会发生人员变动，当人员离开时，找到能够保证工作连续性的方法。
- 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。
- 制定工作产品标准，并建立相应机制以确保能够及时创建所有的模型和文档。
- 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（risk-monitoring）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：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工作产品标准，并建立相应机制以确保能够适时开发出工作产品。万一有关键成员离开此项目，有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工作产品，以保证每一个工作产品的正确性，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（risk management and contingency planning）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“注释文档或Wiki”和（或）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，不用试图缓解该风险，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，项目管理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，项目管理者可以将Pareto的80-20法则用于软件风险上。经验表明，整个项目的80%风险（即可能导致项目失败的80%的潜在因素）可能是由只占20%的已经识别出的风险所引发。早期风险分析步骤中所做的工作能够帮助项目管理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被纳入RMMM计划之中——这些风险不属于那关键的20%（具有最高项目优先级的风险）。

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（software safety and hazard analysis）（例如，[Dun02]，[Her00]，[Lev95]）是一种软件质量保证活动（第16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，就可以安排某些软件设计特性令其消除或控制这些潜在的灾难。

28.7 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，项目管理者还可将其作为整个项目计划的一部分。

有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28-4所示。

风险信息表单			
风险标识号：P02-4-32	日期：5/9/09	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7/1/09起没有效果。			
当前状态 5/12/09：缓解步骤启动。			
创建者：D.Gagne		受托者：B.Laster	

图28-4 风险信息表单

资料来源：[Wil97]

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。正如前面讨论过的，风险缓解是一种问题规避活动，而风险监测则是一种项目跟踪活动，这种监测活动有三个主要目的：（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，所以风险监测的另一个任务就是试图找到“起源”（在整个项目中是哪些风险引起了哪些问题）。

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：[⊖]

@Risk，由Palisade Corporation（www.palisade.com）开发，是一个利用蒙特卡罗（Monte Carlo）模拟法来驱动分析机的一般风险分析工具。

Riskman，由ABS Consulting（www.absconsulting.com/riskmansoftware/ index.html）发布，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。

Risk+，由Deltek（www.deltek.com）开发，与Microsoft Project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，是一个通用的Web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并制定有效的应对措施。

28.8 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。

习题与思考题

28.1 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。

28.2 简述"已知风险"和"可预测风险"之间的差别。

28.3 为SEPA站点上给出的每个风险条目检查表增加三个问题或主题。

28.4 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。

28.5 请为某Web应用项目制定风险表，该项目可让家庭成员上传并共赏照片。

28.6 针对你近期完成的一个软件项目，列出其相关的风险因素及一些可能导致项目失误的风险驱动因子。

28.7 为图28-2所描述的风险中的3个制定风险缓解策略及特定的风险缓解活动。

28.8 为图28-2所描述的风险中的3个制定风险监测策略及特定的风险监测活动。确保你所监测的风险因素可以确定风险正在变大或变小。

28.9 精化图28-2中的3个风险，并为每个风险建立风险信息表单。

28.10 用CTC格式表示图28-2中的3个风险。

28.11 假定每个LOC的成本为$15，概率为50%，重新计算28.4.2节中讨论的风险显露度。

28.12 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？

28.13 给出主要关注软件安全和灾难分析的五个软件应用领域。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

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。Vun（《Modeling Risk》，Wiley，2006）详细介绍了适用于软件项目的风险分析数学处理方法。Crohy及其同事（《The Essentials of Risk Management》，McGraw-Hill，2006）、Mulcahy（《Risk Management, Tricks of the Trade for Project Managers》，RMC Publications，Inc.，2003）、Kendrick（《Identifying and Managing Project Risk》，American Management Association，2003）以及Marrison（《The Fundamentals of Risk Measurement》，McGraw-Hill，2002）介绍了项目管理者们所采用的各种方法及工具。

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。

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]描述了风险识别及降低技术。《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，最新的风险管理WWW参考列表见SEPA的Web站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm。

第29章

Software Engineering：A Practitioner's Approach (Seventh Edition)

维护与再工程

要点浏览

概念：想想任何一个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越来越长，已经使你不可忍受，同时，它也不再代表最新的技术。怎么办？有一段时间，你试图修复它、给它打补丁、甚至扩展它的功能，这称为维护。但是，随着时间的推移，维护变得越来越困难，到了重新构造它的时候了。构造一个具有更好的功能、更好的性能、更好的可靠性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，维护由支持人员（软件工程组织的一部分）完成，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撑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，这就是为什么必须对软件进行持续的维护，并在适当的时候对其实施再工程，以跟上变化的步伐。

步骤：维护改正缺陷，对软件进行适应性修改以满足变化的环境，增强功能以满足客户不断进化的需求。在策略级，业务过程再工程（business process reengineering，BPR）制订业务目标，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易于维护的现有程序的新版本。

工作产品：产生一系列可维护的和再工程工作产品（例如，用例、分析模型和设计模型、测试规程），最终产品是升级的软件。

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、功能性和互操作性方面的错误。

关键概念

业务过程再工程
文档重构
正向工程
库存目录分析
可维护性
重构
　代码
　数据
逆向工程
　数据
　处理
　用户界面
软件维护
软件再工程
可支持性

不管计算机软件的应用领域、规模或复杂性如何，计算机软件都将随时间而不断演化，因为变更驱动着这个过程。对于计算机软件，在很多情况下都会发生变更：当进行纠错时会发生变更；当修改某个软件以适应新环境时会发生变更；当客户要求新的特性或新功能时会发生变更；将应用系统再工程以适应现代环境时也会发生变更。在过去的30年中，Manny Lehman [如Leh97a]及其同事已经对行业级的软件和系统进行了详细分析，目的是为了得到软件演化的统一理论（unified theory for software evolution）。虽然本书没有详细介绍这项工作，但是所推导出的基本规律值得关注[Leh97b]：

持续变更法则（1974）：由于软件是在真实世界的计算环境中实现的，因此将随着时间不断演化（称为E型系统），所以必须持续修改软件，否则这些软件将变得越来越不令人满意。

复杂性增长法则（1974）：随着E型系统的不断演化，系统的复杂性也随之

增长，除非采取措施使系统保持或降低复杂性。

自调节法则（1974）：E型系统的演化过程可以自动调节产品分布和过程测量以接近正常状态。

组织稳定性守恒法则（1980）：演化中的E型系统的平均有效全局活动率在整个产品的生命期内是恒定的。

熟悉度守恒法则（1980）：随着E型系统的不断演化，与该系统相关的所有人员，如：开发人员、销售人员、用户，都必须始终掌握系统的内容和行为，以使得演化过程令人满意。过度的增长会削弱对其内容和行为的掌握程度。因此，在系统的演化过程中，平均增长值是恒定的。

持续增长法则（1980）：在E型系统的整个生命期内，其功能内容必须持续增长以满足用户需求。

随着时间的推移，遗留系统是如何进行演化的？

质量下降法则（1996）：如果E型系统没有严格地进行维护，也没有随着操作环境的改变做适应性修改，那么E型系统的质量将有所下降。

反馈系统法则（1996）：E型演化过程由多层次、多循环、多代理的反馈系统组成，而且，要想在任何合理的基础上达到有意义的改进就必须这样进行处理。

Lehman及其同事定义的这些法则是软件工程现实的固有部分。本章将讨论软件维护所面临的挑战以及延长遗留系统的有效生命周期所需要的再工程活动。

29.1 软件维护

软件维护几乎是在软件交付给最终用户后就立即开始。软件交付给最终用户后，没几天功夫，缺陷报告就有可能送到软件工程组织；没几周功夫，某类用户就可能会提出必须修改软件以适应他们所处环境的特殊要求；没几个月功夫，另一个公司集团在软件发布时认为他们与这个软件毫不相干，但现在意识到该软件可能会给他们带来意想不到的好处，因此他们需要做些改进，使软件可以用于他们的环境。

软件维护所面临的挑战已经开始。软件工程组织面临着不断增长的代码错误修改任务、适应性修改请求及彻底的增强，这些都必须进行策划、安排进度并最终完成。用不了多久，维护队列就已经很长，这意味着修正工作可能会用尽现有的资源。随着时间的推移，软件工程组织会发现自己花在维护现有程序上的资金和时间远比建造新的应用系统多得多。事实上，软件组织将全部资源的60%～70%花费在软件维护上是很常见的。

有人可能会问："为什么需要这么多的维护？为什么要花费这样多的工作量来进行维护？"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了调整，为满足新的用户要求对其进行了改进——所有这些并没有对整个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，已经没有人直接了解这个遗留系统了。

就像在第22章中讲到的，所有软件工作都普遍在变更之中进行。开发计算机系统时，变更是不可避免的。因此，软件组织一定要建立评估、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

本书从始至终，都一直强调弄清楚问题（分析）和给出结构明确的解决方案（设计）的重要性。实际上，本书第二部分就主要讨论了这两个软件工程活动的机制，第三部分重点介绍了能够保证软件组织正确完成这两个软件工程活动所需要的技术。分析和设计都可以达到一个重要的软件特性，即可维护性。本质上，可维护性（maintainability）是一个定性指标[⊖]，它表明对现有软件进行改正、适应或增强的容易程度。软件工程所涉及的大部分内容都是建造能够表现出高可维护性的系统。

但什么是可维护性？可维护的软件表现为有效的模块性（第8章），它采用易于理解的设计模式（第12章），建造时采用了明确定义的编码标准及约定，编写的源代码能够自身文档化且易于理解。它应用了大量质量保证技术（本书第三部分），在软件交付之前就已经找出了潜在的维护问题。创建它的软件工程师们已经意识到：实施变更时他们可能已经离开了，因此，软件的设计与实现必须对实施变更的人员"有帮助"。

29.2 软件可支持性

为了有效支持行业级软件，一个组织（或其指定的人员）必须具有完成修正、改写及改进的能力，而修正、改写以及改进是维护活动的一部分。除此之外，在软件的整个生命周期内，该组织还必须提供运行支持、最终用户支持以及再工程活动等重要的支持活动。软件可支持性（software supportability）的合理定义如下：

……在软件系统的整个产品生命周期内支持软件系统的能力。这意味着不仅要满足所有必要的要求或需求，而且还要能够供应装置、支持基础设施、附加软件、工具、人力、或所需要的任何资源以维护软件的运行并满足软件功能[SSO08]。

本质上，在软件过程中进行分析和设计时就应该考虑作为质量因素之一的可支持性。可支持性应该在需求模型（或规格说明）中进行处理，并且要在设计的演化过程中以及构造开始时认真考虑。

WebRef

有关软件可支持性的大量可下载的文档见www.softwaresupportability.org/Downloads.html。

例如，本书前面已经讨论过构件级和代码级的"防错（antibug）"软件需求。当在运行环境中出现缺陷时（没有错误，但会有缺陷），该软件应该具有协助支持人员进行处理的工具。此外，支持人员应该能够访问到记录已经出现的所有缺陷的数据库，获得各个缺陷的特征、起因以及补救措施。这就使得支持人员能够考察"类似的"缺陷，从而提供更快的诊断及纠错手段。

虽然在应用系统中出现缺陷属于关键性的支持问题，但是，可支持性还需要提供相应的资源以支持日复一日的最终用户问题。最终用户支持人员的工作就是回答用户对应用系统的安装、运行以及使用等方面的疑问。

29.3 再工程

Michael Hammer为《Harvard Business Review》撰写的研讨论文 [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，我们应该将它们删除，并从头开始。应该对我们的业务进行"再工程"：采用现代信息技术的优势重新

⊖ 有些定量测量可以提供可维护性的间接指标（例如，[Sch99]、[SEI02]）。

"用昨天的思考方法面对明天就是在停顿中想像生活。"——James Bell

设计业务过程，以获得性能的极大改善。

各个公司都是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。

和所有变革一样，Hammer的号召产生了正面和负面的影响。上个世纪90年代，有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力；而有些公司只是通过减小规模和外购（而不是再工程）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[DeM95a]。

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。

软件通常是Hammer 所讨论的业务规则的实现。今天，大多数公司都有成千上万支持"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新的计算机系统[⊖]，但是，在多数情况下，只需要对现有的应用系统进行修改或重构。

在下面的章节中将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。

29.4 业务过程再工程

虽然软件再工程是以更好的、更易于维护的软件替换现有的软件功能，但是BPR经常能够得到新的软件功能。

业务过程再工程（business process reengineering，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？

29.4.1 业务过程

作为软件工程师，你的工作位于业务层次的底部。但是，要确保有人已对上层的工作进行过认真考虑。如果不是这样，你的工作则处在危险之中。

业务过程是指"执行一组逻辑相关的任务以获得定义明确的业务结果"[Dav90]。业务过程将人员、设备、材料资源以及业务规程综合在一起以产生特定的结果。业务过程的例子有：设计新产品、购买服务和支持、雇用新员工，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。

每个业务过程都有一个指定的客户——接收业务过程结果（例如，想法、报告、设计、服务、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"。

各个系统实际上都是由多个子系统构成的层次结构。业务也不例外，总业务可以按照下面的方式进行分解：

业务→业务系统→业务过程→业务子过程

每一个业务系统（也称为业务功能）可以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工作只着重于单个过程或子过程。

⊖ 基于Web的应用及系统的爆炸性发展就预示了这种趋势。

29.4.2 BPR模型

和大多数工程活动一样，业务过程再工程也是迭代的。所有的业务目标及为达到这些目标所采用的过程都必须能够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29-1描述了一个业务过程再工程模型，该模型定义了6项活动：

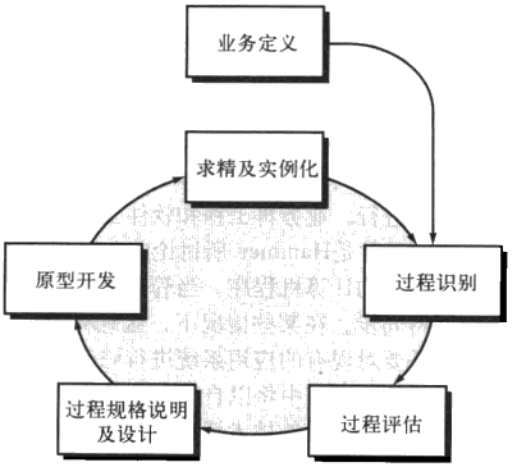


图29-1 BPR模型

业务定义。在4个关键的驱动因子范畴内制定业务目标，这4个关键的驱动因子是：降低成本、缩短时间、提高质量、人力开发以及授权。可以在业务级或针对业务的某个特定部分来制定业务目标。

过程识别。识别对业务定义中所制定的业务目标起关键作用的所有过程，然后可以按照重要性、变更的要求、或以任何适用于再工程活动的方式将这些过程排序。

过程评估。对现有过程进行透彻的分析和测量。确定所有的过程任务，说明这些过程任务花费的成本和时间，并且明确质量/性能问题。

过程规格说明及设计。根据从上面3个BPR活动中获得的信息，为每个即将被重新设计的过程准备用例（第5章和第6章）。在BPR范畴内，这些用例描述了将某些结果传递给客户的场景。将这些用例作为过程的规格说明，为过程设计一组新任务。

"只要能够在新事物中展示出往昔的东西，我们就会感到欣慰。"——F.W.Nietzsche

原型开发。在将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。这个活动可以"测试"重新设计的业务过程，有利于实现求精。

求精及实例化。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。

SOFTWARE TOOLS

业务过程再工程

目的：BPR工具支持对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或功能问题。一旦识别出存在的问题，这些工具还允许分析员开发原型和（或）模拟修订的业务过程。

代表性工具：[⊖]

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）能力，使得业务分析能够探索不同的过程场景。

e-Work，由Metastorm（www.metastorm.com）开发，为手工过程以及自动过程提供了业务过程管理支持。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

IceTools，由Blue Ice（www.blueice.com）开发，集合了Microsoft Office及Microsoft Project的BPR模板。

SpeeDev，由SpeeDev Inc.（www.speedev.com）开发，是能够使一个组织对过程工作流建模（此处指IT工作流）的多种工具之一。

Workflow tool suite，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。

可链接到BPR工具的有用列表见www.opfro.org/index.html?Components/Producers/Tools/BusinessProcessReengineeringTools.html~Contents。

29.5 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纠错、适应性修改及增强。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，应用系统已经处于不稳定的状态，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。

29.5.1 软件再工程过程模型

再工程要花费时间，消耗大量的资金，并占用资源，而这些花费本可用于当前关注的事情上。因此，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。

再工程过程模型提供了一个可行的策略，我们将在本节后面讨论该模型，首先讨论一些基本原则。

WebRef

有关软件再工程的信息源见reengineer.org。

再工程是一项重构活动。为了更好地理解再工程，考虑与再工程相似的“重建一所房子”的活动。考虑如下情况：假定你在另一个州购买了一所房子。你还没有亲眼看到房子，就以令人吃惊的低价格买下了。在可能需要彻底重建的情况下，你将如何进行重建？

- 在开始重建前，先检查一下房子似乎是合理的。为了确定它是否确实需要重建，你（或职业检查员）可以列出一组标准，使得检查工作能系统地进行。
- 在拆掉并重建整座房子前，确认其结构是不是牢固的。如果该房子结构良好，则可能只需要“改造”（remodel），而不是重建（花更低的成本、更少的时间）。
- 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，弄清楚布线、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、最耐久的材料。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，但它们也同样适用于计算机系统和计算机应用系统的再工程。

为了贯彻这些原则，可以使用图29-2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（弄清楚程序的内部工作原理）。

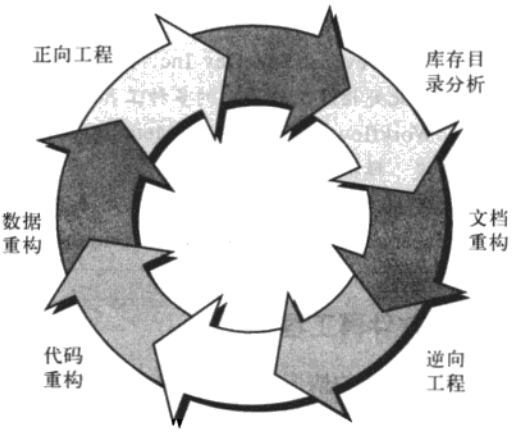


图29-2 软件再工程过程模型

29.5.2 软件再工程活动

如果时间及资源紧缺，可以考虑将Pareto原理应用到将要实施再工程的软件中。将再工程过程应用到存在80%问题的20%的软件中。

图29-2所示的再工程范型是一个循环模型。这意味着该范型中的每项活动均可能重复出现。对任意一个特定的循环，其过程可以在任意一项活动之后终止。

库存目录分析。各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，其中为每个常用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、当前可维护性和可支持性以及其他本地重要性准则对这些信息排序，可以选出再工程的应用系统，然后为这些应用系统的再工程工作分配资源。

只建立理解软件所需要的文档，一页都不要多写。

值得注意的是：应该对库存目录进行定期分析。因为应用系统的状态（例如，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。拙劣的文档是很多遗留系统的特点。但是，对此能做些什么呢？如何进行选择？

1. 建立文档是非常耗费时间的。如果系统正常运作，可以选择保持现状。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固定的，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，就让它保持现状。

2. 文档必须更新，但是资源有限。可以采取“使用时再建立文档”的方法。不需要重构应用系统的全部文档，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。即使出现这种情况，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，软件组织必须针对不同的情形选择最适合的方法。

逆向工程。术语逆向工程（reverse engineering）源于硬件领域，一个公司分解某个有竞争力的硬件产品，以弄清楚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弄明白。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。

再工程领域的大量资料见www.comp.lancs.ac.uk/projects/RenaissanceWeb/。

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（通常是很多年以前的产品）。

由于未开发过相关的规格说明，要弄清楚的这些“秘密”变得模糊不清。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(design recovery)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。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）类型就是代码重构（code restructuring）[⊖]。有些遗留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。

要实现代码重构，可以使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）或者用更现代的程序设计语言重新编写。对生成的重构代码进行评审和测试，以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。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，对大部分应用系统来说，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6章和第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，通常采用平面文件实现，而关系型方法可以大大地简化处理），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算法有很强的影响，所以对数据的变更总会导致体系结构或代码层的变更。

正向工程。在理想情况下，可以使用自动的“再工程引擎”来重建应用系统。将旧程序输入引擎，经过分析、重构，然后重新生成能够表现出最好的软件质量的程序。短期内，这样的“引擎”还不可能出现，但是，有的厂商已经开发了针对特定应用领域（例如，用特定数据库系统实现的应用）的一些工具，能够实现一部分功能。更重要的是，这些再工程工具正在变得越来越成熟。

正向工程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，以提高其整体质量。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）提高整体性能。

29.6 逆向工程

逆向工程假想有一个“魔术通道”，在通道的一端输入随意设计的、无文档的源程序文件，另一端出来的就是计算机程序的完整设计描述（以及完整文档）。不幸的是，这样的魔术通道并不存在。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。

逆向工程过程以及用于实现逆向工程过程的工具的抽象层次（abstraction level）是指可以从源代码中抽取出来的设计信息的精密程度。理想情况下，抽象层次应该尽可能高，即，逆向工程过程应该能够推导出过程的设计表示（低层抽象），程序和数据结构信息（稍高层次的抽象），对象模型、数据和（或）控制流模型（相对高层的抽象）以及实体关系模型（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。

⊖ 代码重构具有“重构”(refactoring)的某些成分，重构是一个设计概念，在第8章中介绍了这个概念，本书的其他地方也进行了讨论。

逆向工程过程的完备性（completeness）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，要得到完整的过程设计表示是比较容易的。虽然也可以推导出简单的体系结构设计表示，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（interactivity）是指为了建立一个有效的逆向工程过程，人员与自动工具“结合”的程度。在大多数情况下，随着抽象层次增高，交互性必定增加，否则完备性将受到损害。

如果逆向工程过程的方向性（directionality）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，则需要将这些信息输入到再工程工具，以试图重构或重新生成旧程序。

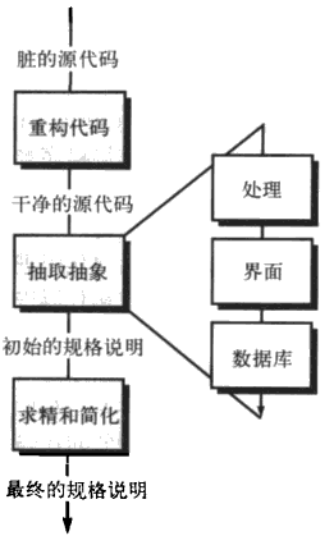


图29-3 逆向工程过程

WebRef

“设计恢复及程序理解”的有用资源见www.sel.iit.nrc.ca/projects/dr/dr.html。

逆向工程过程如图29-3所示，在逆向工程活动可以开始前，要重构无结构的（“脏的”）源代码（29.5.1节），使得它仅包含结构化程序设计结构[⊖]。这样使得源代码更容易理解，并为所有后续的逆向工程活动奠定基础。

逆向工程的核心活动是抽取抽象（extract abstractions）。工程师必须评估旧程序，并从（通常是无文档的）源代码中抽取出有意义的规格说明，包括：要执行的处理、要使用的用户界面以及所采用的程序数据结构或数据库。

ADVICE

在某些情况下，第一个逆向工程活动是构造UML类图。

29.6.1 理解数据的逆向工程

数据的逆向工程可以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，必须对内部程序的数据结构进行逆向工程。在系统层，通常要将全局数据结构（例如，文件、数据库）实施再工程以符合新的数据库管理范型（例如，从平面文件转移到关系数据库系统或面向对象数据库系统）。对现有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。

ADVICE

针对传统软件的数据，逆向工程方法遵循类似的步骤：（1）建造数据模型；（2）标识数据对象的属性；（3）确定关系。

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，代码中的数据组织标识了抽象数据类型，例如，记录结构、文件、列表和其他数据结构通常都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，当要将一种数据库模式实施再工程使之成为另一种数据库模式时，需要弄清楚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[Pre94]：（1）构造初始的对象模型；（2）确定候选关键字（考察每一个属性，以确定其是否被用来指向另外的记录或另一张表，充当指针的那些属性就

⊖ 可使用重构引擎（restructuring engine）对代码进行重构，重构引擎是一种可以重构源代码的工具。

是候选关键字）；（3）精化实验性的类；（4）定义一般化关系；（5）使用与CRC方法相似的技术找出关联关系。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[Pre94]将旧的数据库结构映射到新的数据库结构。

29.6.2 理解处理的逆向工程

"对理解的激情，就像对音乐的激情一样，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

处理的逆向工程开始于试图弄清楚并抽取出源代码所表示的过程抽象。为了弄清楚过程抽象，需要在不同的抽象级别分析代码：系统级、程序级、构件级、模式级和语句级。

在进行更详细的逆向工程前，必须弄清楚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。每一个构件实现某个子功能，同时也表示某个定义的过程抽象，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，以确认是否与现有代码相符[⊖]。

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。几乎在每个构件中都是由一个代码段准备（在模块内）要处理的数据，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构件要输出的处理结果。在每个代码段中，工程师都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。

对于大型系统，通常采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弄清楚现有代码的语义，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。

29.6.3 用户界面的逆向工程

高级图形用户界面（GUI）对于计算机产品和各类计算机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。

为了能够完全弄清楚现有的用户界面，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（UI）的逆向工程开始前必须回答的3个基本问题：

如何弄清楚现有用户界面是怎样工作的？

- 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7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。

值得注意的是，替代的GUI可能并不是旧界面的精确镜像（实际上，它可能完全不同）。开发新的交互隐喻通常是值得的，例如，旧的UI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可能使用滚动条及鼠标来完成相同的功能。

⊖ 通常，在程序生命历史早期编写的规格说明从未更新过。随着代码被修改，代码不再与规格说明相符。

SOFTWARE TOOLS

逆向工程

目的：帮助软件工程师弄清楚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、数据以及行为的设计表示。

代表性工具：[⊖]

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。

29.7 重构

虽然代码重构可以缓解调试与小的修改相关的问题，但是，它不是再工程。只有重构数据和体系结构，才能够取得真正的收益。

软件重构工作是要修改源代码和（或）数据，使软件适应未来的变化。通常，重构并不修改总体程序结构，它倾向于关注单个模块的设计细节及模块中所定义的局部数据结构。如果重构扩展到模块边界之外，而且涉及软件体系结构，则重构变成了正向工程（29.8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[⊖]。

29.7.1 代码重构

进行代码重构（code restructuring）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，代码重构技术（例如，Warnier的逻辑简化技术[War74]）都是利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。其目的是采用“面条碗”式的代码，并推导出遵从结构化程序设计原理（第10章）的过程设计。

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。

29.7.2 数据重构

在数据重构开始前，必须先进行称为源代码分析（analysis of source code）的逆向工程活动。评估所有包含了数据定义、文件描述、I/O以及接口描述的程序设计语言语句，目的是抽取数据项及对象，获取关于数据流的信息，以及弄清楚现有的已实现的数据结构。有时也称该项活动为数据分析（data analysis）。

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

⊖ 扩展性重构和重开发有时很难区分，两者均属于再工程。

一旦完成了数据分析，就可以开始数据重设计（data redesign）。其最简单的形式为：通过数据记录标准化（data record standardization）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（data name rationalization），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，并且当数据在系统内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计算机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成一种更加现代的语言。

代表性工具：⊖

DMS Software Reengineering Toolkit，由Semantic Design(www.semdesigns.com)开发，提供了针对COBOL、C/C++、Java、Fortran 90及VHDL的多种重构能力。

Clone Doctor,由Semantic Designs,Inc.(www.semdesigns.com)开发，能够分析和变换用C、C++、Java、COBOL或其他任何基于文本的计算机语言编写的程序。

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。

大量再工程和逆向工程的工具的链接可在www.csse.monash.edu/～ipeake/reeng/free-swre-tools.html及www.cs.ualberta.ca/～kenw/toolsdir/all.html找到。

29.8 正向工程

当面对设计和实现均非常差的程序时，我们有哪些选择？

程序的控制流在图形上相当于一碗意大利面，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，如果该程序必须修改，以适应用户需求的变化，软件工程师有下列选择：

1. 努力地不断修改，与特定的设计和复杂的源代码搏斗，以实现必要的修改。

2. 试图去理解程序更多的内部工作，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，使用再工程工具来辅助理解现有的设计。

这里没有唯一的“正确”选择。当前遇到的情况可能要求作第一种选择，即使更期望的是

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

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。

其他选择。

不要坐等维护请求，开发或维护组织应该从库存目录分析的结果中挑选出一个程序，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，我们就可以应用到上面的第2、3或4选项。

乍一看，重新开发大型程序的现有可用版本似乎是相当浪费的，不过在做出判断之前，应该考虑如下几点：

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. 由于软件的原型已经存在，开发生产率将远高于平均水平。
4. 用户现在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简化。
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）。

大规模的内部软件开发者（例如，一个大型消费品公司的业务系统软件开发小组）可能在其职责范围内有500~2000个产品程序，可以根据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正向工程的程序进行评审。

正向工程过程能够应用软件工程的原理、概念和方法来重建现有的应用系统。在大多数情况下，正向工程并不仅仅是创建某旧程序的现代等价物，而是将新的用户和技术需求集成到再工程中，重新开发的程序可以扩展原有应用系统的能力。

29.8.1 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/服务器体系结构，已对很多大型机应用系统实施了再工程（包括WebApp）。实际上是将集中式的计算资源（包括软件）分布到多个客户机平台上。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）体系结构的典型大型机应用系统应具有以下特征：

- 应用系统的功能可以迁移到每个客户端计算机。
- 在客户端可以实现新的GUI界面。
- 数据库功能由服务器完成。
- 特殊功能（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，而应该作为新的开发项目。只有在旧系统的功能被集成到新系统的体系结构中时，才考虑作为再工程项目。

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从对现有大型机的业务环境的彻底分析开始。可以确定3个抽象层次。数据库层（database sits）是客户/服务器体系结构的基础，负责管理来自服务器端应用的事务和查询，而这些事务和查询必须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。客户端应用系统提供面向用户的目标功能。

在对数据库层进行重设计之前，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6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；只能由授权用户完成更新；能够强制执行核心业务规则（例如，在删除某厂商记录前，服务器能够保证不存在与该厂商有关的应付款账号、合同或沟通）；能够实现高效查询；以及能够建立完善的归档能力。

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，以保证处于客户端应用和数据库之间的事务和查询符合已建立的业务过程。

客户端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，可以将大型机应用系统分割为一组更小的、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]或[Orf99]。

29.8.2 面向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，必须对旧的应用系统实施再工程，使得它们能够容易地集成到大型的、面向对象的系统中。

将传统软件进行再工程使其成为面向对象实现需要用到本书第二部分讨论的很多技术。首先，要将现有的软件进行逆向工程，以建立适当的数据、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5章和第6章）。然后，联合使用在逆向工程中创建的数据模型以及CRC建模技术（第6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10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。

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，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。

29.9 再工程经济学

在理想世界中，应该立即淘汰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。

“现在你可能付出很少，但后来你可能付出很多。”——作者不详

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 = 应用系统当前的年度维护成本
P_2 = 应用系统当前的年度运作成本
P_3 = 应用系统当前的年度业务价值
P_4 = 再工程后预期的年度维护成本
P_5 = 再工程后预期的年度运作成本
P_6 = 再工程后预期的年度业务价值
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，未执行再工程的应用系统）的持续维护相关的成本可以定义为：

$$C_{维护} = [P_3-(P_1 + P_2)] \times L \tag{29-1}$$

与再工程相关的成本用下面的关系定义：

$$C_{再工程} = P_6-(P_4 + P_5) \times (L-P_8)-(P_7 \times P_9) \tag{29-2}$$

利用式（29-1）和式（29-2）中计算出的成本，可以计算出再工程的整体效益：

$$成本效益 = C_{再工程}-C_{维护} \tag{29-3}$$

可以对所有在库存目录分析（29.4.2节）中标识为高优先级的应用系统进行上述的成本-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。

29.10 小结

软件维护与软件支持是整个应用系统生命周期中的持续活动。在软件维护与软件支持活动中，改正了缺陷，做了适应性修改以适应不断变化的运行环境或业务环境，在利益相关方的要求下，完成了增强，而且能够支持用户将应用系统集成到他们的个人工作流或业务工作流中。

再工程发生在两个不同的抽象层次。在业务层，再工程着重于业务过程，目的是改变业务过程以提高在某业务领域的竞争力；在软件层，再工程考察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）能够制订业务目标；识别并评估现有业务过程（在确定的目标范围内）；详细描述并设计修订过程；并在业务中对它们进行原型化、精化和实例化。BPR的关注点也可以扩展到软件之外，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。

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、程序和数据重构以及正向工程。这些活动的目的是创建现有程序的更高质量和更易于维护的版本——将在21世纪具有良好生命力的程序。

可以定量地确定再工程的成本-效益，现状的成本（即，与某现有应用系统不断发生的支持和维护相关的成本）与预期的再工程成本及维护成本和支持成本的减少进行比较，对于几乎所有生命期长且当前的可维护性或支持性较差的程序，再工程代表了成本合算的业务策略。

习题与思考题

29.1 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。建议使用在29.4.2 节中描述的BPR模型修改该过程以提高其效率。

29.2 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。

29.3 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. 在班上介绍你的经验。

29.4 探讨在SEPA网站上列出的库存目录分析检查表，然后尝试开发一个适用于现有程序的定量软件评价系统，目的是从中挑选出再工程的候选程序。你的系统应该扩展到29.9节所述的经济分析。

29.5 提出一种对纸、墨或传统的电子文档的替代物，可将它作为文档重构的基础。(提示：考虑能够用于软件交流的新的描述技术。)

29.6 有人相信人工智能技术将提高逆向工程过程的抽象层次，对此专题（也就是说，人工智能在逆向工程中的应用）进行研究，并撰写一篇支持此论点的简短论文。

29.7 为什么当抽象层次增加时，完备性更难于达到?

29.8 如果完备性增加，为什么交互性也必须增加?

29.9 使用通过Web获得的信息，向班级介绍3种逆向工程工具的特点。

29.10 重构和正向工程之间存在差别，这种不同是什么?

29.11 对文献和（或）Internet资源进行研究，找出至少一篇讨论从大型机到客户/服务器模式的再工程案例研究的文章，给出概要介绍。

29.12 如何在29.9节给出的成本-效益模型中确定P_4到P_7?

推荐读物与阅读信息

具有讽刺意味的是，软件维护与软件支持是应用系统生命期中花费成本最高的活动。但是，描写维护和支持的书要比描写任何其他软件工程主题的书少得多。Jarzabek（《Effective Software Maintenance and Evolution》，Auerbach，2007）、Grubb和Takang（《Software Maintenance：Concepts and Practice，2nd edition》，World Scientific Publishing Co.，2003）以及Pigoski（《Practical Software Maintenance》，Wiley，1996）都是最近加入到文献中的。这些书涵盖了基本的维护和支持实践，而且介绍了实用的管理指南。Schneberger（《Client/Server Software Maintenance》，McGraw-Hill，1997）论述了C/S环境下的维护技术。Mens和Demeyer编辑的文集（《Software Evolution》，Springer，2008）收集了有关“软件演化”的最新研究。

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，2003）促进了早期的兴趣。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 Sociaty 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。

Fong（《Information System Reengineering and Integration》，Springer，2006）论述了适用于大部分信息系统的数据库转换技术、逆向工程及正向工程。Demeyer及其同事（《Object Oriented Reengineering Patterns》，Morgan Kaufmann，2002）针对OO系统的重构和（或）再工程提出了基于模式的观点。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）、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”。

Cameron（《Reengineering Busuness for Success in the Internet Age》，Computer Technology Research，2000）和Umar（《Application（Re）engineering：Building Web-Based Applications and Dealing with Legacies》，Prentice-Hall，1997）为希望将遗留系统转换为基于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，最新的与软件维护和再工程相关的WWW资源列表见SEPA 的Web站点 www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm。

软件工程高级课题

在本书这一部分中，我们考虑一些能够对软件工程加深理解的高级研究课题。在下面的章节中，我们将讨论以下问题：

- 什么是软件过程改进（Software Process Improvement, SPI），以及怎样利用软件过程改进来提高软件工程实践的现状？
- 对未来十年软件工程实践有重大影响的趋势是什么？
- 软件工程师未来的发展方向是什么？

通过对这些问题的回答，将有助于理解在不远的将来对软件工程有深远影响的那些课题。

软件过程改进

要点浏览

概念：软件过程改进包含了一系列活动，这些活动可以产生更好的软件过程，因而，更高质量的软件就可以及时地交付给客户。

人员：主持SPI的人员可分为3组：技术管理者、软件工程师和承担质量保证责任的个人。

重要性：一些软件组织更关注特别的软件过程。当他们努力改进软件工程实践的时候，他们不得不关注已有过程中的缺陷，并且尽量改进他们在软件工作中的方法。

步骤：SPI方法是迭代的和连续的，它包括5个步骤：（1）当前软件过程的评估；（2）对业务人员和管理者的教育和培训；（3）过程要素、软件工程方法以及工具的选取和合理性判定；（4）SPI计划的实现；（5）基于计划结果的评价和调整。

工作产品：尽管有很多中间的SPI产品，但最终的结果还是改进软件过程，并产生更高质量的软件。

质量保证措施：组织所生产的软件将以更少的缺陷提交给客户，软件过程中每一阶段的返工将会减少，及时交付产品的可能性将会得到提高。

关键概念
评估
CMMI
关键成功要素
教育和培训
评价
设置/迁移
合理性判定
成熟度模型
人员CMM
投资收益率
风险管理
选择
软件过程改进
　适应性
　框架
　过程

很久以前，“软件过程改进”这个术语就被广泛使用了，作者本人工作过的大多数公司都试图改进他们的软件工程实践的状况。因此根据我的经验，编写了《making software engineering happen》这本书[Pre88]。在这本书的序言中，我写了下面这段话：

> 过去的十年，我有一些机会帮助很多大型公司实现他们的软件工程实践。这个工作是很困难的，并且很少能如人们希望的那样顺利——但是，一旦成功，其意义就是深远的。软件项目更可能按时完成，软件开发中涉及的所有人员之间的交流会得到改善。在大型软件项目中的混淆和混乱程度往往会大幅减少。客户遇到的错误数大幅度下降。软件组织的信誉在提高。管理上也少了一个需要担心的问题。
>
> 但是，所有这些并不都是令人愉快的和充满光明的。许多公司试图实施软件工程实践，但在受挫后不得不放弃。其他一些公司也半途而废，从来没有看到如上所述的那些好处。还有一些公司以一种严格的方式做了尝试，其结果是技术人员和管理人员公开抵触，随后导致士气低落。

尽管这些话是20多年前写的，但今天依然适用。

当我们进入21世纪的第二个10年的时候，绝大多数的软件工程组织都试图“使软件工程成为现实”。一些组织已经实现了一些个别的实践用来帮助改进其产品的质量，并且提高了交付的及时性。另外一些组织建立了“成熟的”软件过程，用来指导他们的技术和项目管理活动。但是，还有一些组织仍然在努力摸索。他们的实践是碰巧的，过程也是特别的。偶尔，他们的工作也是非常杰出的，但是，更主要的是他们的每项项目都在冒险，没有人知道结局是好还是坏。

所以，上面提到的后两种组织都需要软件过程改进吗？答案（可能会令你大吃一惊）是肯

定的。那些已经成功地使软件工程成为现实的组织并不能自鸣得意。他们必须继续工作来改进软件工程方法。那些还在努力摸索的组织更要朝着改进的道路前进。

30.1 什么是SPI

KEY POINT
SPI意味着一个已定义的软件过程、一种组织方法和一种改进策略。

术语软件过程改进（software process improvement, SPI）包含很多方面。首先，它包含以有效方式定义的有效过程的一些要素；其次，组织内已存在的关于软件开发的方法要依据这些要素进行评估；第三，它定义了有价值的改进策略。SPI策略将已有软件开发的方法转换成一些更集中、更可重复、更可靠的事物（就所生产产品的质量和交付给客户的时间而言）。

由于SPI需要有投入，它必然会产生相应的投资回报。实施SPI策略所付出的工作量和时间必须要有某种度量方法。这样做，改进过程和实践的结果必然会减少解决软件“问题”的费用和时间。必须减少交付给最终用户软件的缺陷，减少由于质量问题导致的返工次数，减少软件维护和支持（第29章）的相关费用，并减少软件延期交付导致的间接成本。

“大多数软件危机是自己造成的，正如一位CIO所说：‘我宁愿做错也不愿迟做，以后总会有机会修正它’”。——Mark Paulk

30.1.1 SPI的方法

尽管一个组织可能会选择不太正式的SPI方法，但大多数组织还是会从众多的SPI框架中选择一个框架。SPI框架定义了以下内容：（1）如果要获得有效的软件过程，就要给定一组特性；（2）用来评估是否具有这些特性的一种方法；（3）一种总结这些评估结果的机制；（4）用来帮助软件组织弥补在实施过程中发现弱点和缺失的一种策略。

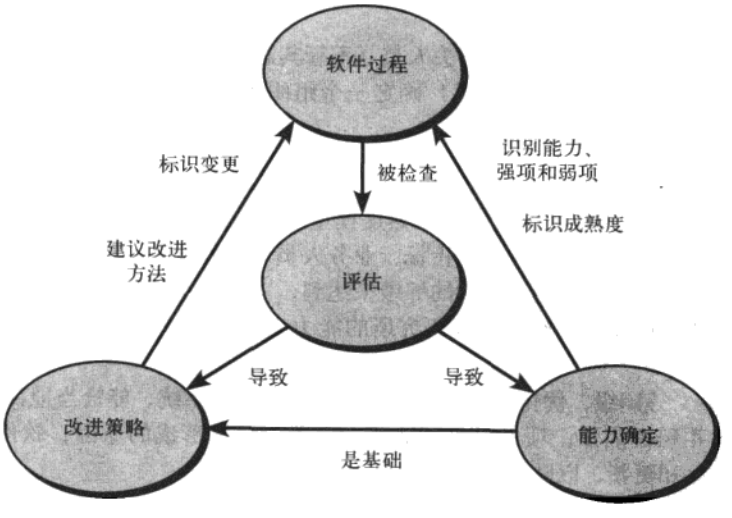


图30-1 SPI框架的要素 摘自[Rou02]

SPI框架评估一个组织软件过程的“成熟度”，并提供成熟度等级定性的表示。实际上，术语“成熟度模型”（30.1.2节）经常会用到。从本质上讲，SPI框架包括一个成熟度模型，此模型又包含了一组过程质量指标，这些指标提供了对过程质量的整体测度，而过程质量决定了产品质量。

拥护SPI尝试的分组是什么？

图30-1提供了一个典型的SPI框架视图。该框架的关键元素和它们彼此之间的关系如图所示。

应该注意到，不存在通用的SPI框架。实际上，一个组织选择的SPI框架关系到主持SPI 工作的相关人员。Conradi[Con96]定义了6种不同的支持SPI的相关人员：

质量认证人员。这类人员拥护的过程改进工作关注以下关系：

质量（过程）⇒质量（产品）

他们的方法是强调评估方法，并检查一组明确定义的特性，这些特性允许他们确定过程是否展示了质量。他们最可能采用的过程框架有CMM、SPICE、TickIT或Bootstrap[⊖]。

形式主义者。这部分人想理解（并且如果可能就尽可能地优化）过程工作流。要完成这些，他们使用过程建模语言（process modeling language, PML）对已存在的过程创建模型，然后进行设计扩展或修改，使过程更加高效。

有哪些类人员支持SPI?

工具倡导者。这部分人坚持有工具辅助的SPI方法，该方法用工作流或其他过程特性建模，这种方式可以对改进进行分析。

业务人员。这些人员使用务实的方法，“强调主流项目管理、质量管理和产品管理，应用项目级的策划和度量，但是很少有形式化的过程建模或规则的支持”[Con96]。

改革者。这些人的目标是组织变革，这种变革可能会导致更好的软件过程。他们往往把重点放在人的问题上（30.4节），更强调人的能力和结构的测度。

理论家。针对特定应用领域或组织结构，这部分人关注于特定过程模型的适合性。与典型的软件过程模型（如，迭代模型）相比，它们对支持复用或再工程的过程更有兴趣。

随着SPI框架的应用，发起SPI的相关人员（不管其总体重点如何）必须建立一些机制以达到以下目的：（1）支持技术变迁；（2）确定一个组织吸收所提出的过程变革的准备程度；（3）衡量已经采取变革的程度。

30.1.2 成熟度模型

成熟度模型在SPI框架环境中应用。成熟度模型的目的是提供软件组织所具有的“过程成熟度”的总体指标，即软件过程质量的指标、业务人员对过程理解和应用的程度、软件工程实践的总体状况。这可以使用一些有顺序的等级来达到。

例如，卡内基·梅隆大学软件工程研究所的能力成熟度模型（30.3节）给出了5个成熟度等级[Sch96]：

KEY POINT
成熟度模型定义了软件过程能力和执行的水平。

第5级，优化级——组织具有量化的反馈系统，能恰当地识别过程的弱点，并积极加强。项目组分析缺陷，以确定缺陷形成的原因；软件过程要进行评估和更新，以防止已知类型的缺陷再次发生。

第4级，已管理级——详细的软件过程和产品质量度量建立了定量评估的基础。可以将过程执行中有意义的变更与偶然现象区分开来，也可以预测过程质量和产品质量的发展趋势。

第3级，已定义级——管理过程和工程过程已经文档化、标准化，并且已经集成到了组织的标准软件过程中。所有项目都使用组织内认可的、经过裁剪

⊖ 这些SPI框架会在本章后面讨论。

的标准软件过程版本来开发软件。

第2级，可重复级——建立了基本的项目管理过程，用以跟踪成本、进度及功能。可以根据类似项目所取得的经验来策划和管理新产品。

第1级，初始级——几乎没有定义过程，成功更多地依赖于个人英雄的能力，而不是使用成熟的过程及团队合作。

CMM 成熟度等级到此为止，但经验表明，许多组织展示的“过程不成熟”的等级[Sch96] 破坏了改进软件工程实践的合理尝试。Schorsch [Sch96] 建议了4个不成熟级别，常常在现实世界的软件开发组织中遇到，这4个不成熟级别如下：

怎样识别那些抵制SPI尝试的组织?

第0级，粗心大意——无法让成功的开发过程取得成功。认为所有的问题都是技术问题。管理和质量保证活动被认为是软件开发过程中多余的开销。依靠银弹。

第1级，故意阻碍——强加了一些适得其反的过程。过程是严格定义的并且强调对形式的固守。华而不实的仪式比比皆是。集体管理阻碍责任的分配。所有的事情都维持现状。

第2级，蔑视——漠视良好的软件工程制度化。完全分裂软件开发活动和软件过程改进活动。缺乏完整的培训计划。

第3级，暗中破坏——完全忽视自己的章程，蓄意诋毁同行组织软件过程改进的努力。鼓励失败和不良行为。

Schorsch的不成熟级别对软件组织是有危害的。如果你遇到它们中的某一个，SPI尝试注定要失败。

最主要的问题是成熟度级别，例如作为CMM一部分提出来的，是否带来了实质的好处。我认为是的。成熟度级别提供了易于理解的过程质量快照，业务人员和管理者可以将其作为参考基准使用，从中可以规划改进的策略。

30.1.3 SPI适合每个人吗

ADVICE

如果一个特定的过程模型或SPI方法过度地伤害了你的组织，很可能就是这样。

很多年来，SPI一直被视为“大型企业的”活动——仅仅在大公司起作用的一种委婉说法。但是今天，职员少于一百人的所有软件开发公司中有相当大的比例用到了SPI。一个小的软件公司能真正用到SPI活动并保证它成功吗？

在大型软件开发组织和小型软件开发组织之间存在重大的文化差异。小型组织是更非正式的，很少应用标准的实践，倾向于自我管理，这不足为奇。他们对软件组织个别成员的“创造力”还往往感到自豪，并且最初以为SPI框架过于官僚和笨重。然而，过程改进对于小型软件开发组织和大型软件开发组织一样，都是非常重要的。

在小型组织内，实施SPI框架所需要的资源可能是组织内缺少的。管理者必须分配相应的人力和财力，以使软件工程成为现实。因此，不管软件组织的规模是大是小，考虑实施SPI的商业动机都是合理的。

只有当SPI的支持者证明了它的财务杠杆作用，SPI才会被核准并实施[Bir98]。财务杠杆通过检查技术益处（例如，交付时更少的缺陷、减少返工、更低的维护成本或更快的上市时间）并将它们转化成金钱来证明。本质上，你必须对SPI成本提供现实的投资回报。

30.2 SPI过程

使用SPI困难的部分不是一组特性的定义，这些特性定义一个高质量的软件过程或一个过

程的成熟度模型的创建。这些东西是比较容易的。相反，最困难的部分是就如何在整个软件组织中对启动SPI以及制定SPI不断前进的策略达成共识。

卡内基·梅隆大学软件工程研究所已经开发了IDEAL——“一个组织改进模型，作为路线图服务于启动、策划和实施改进活动”[SEI08]。IDEAL 是很多SPI过程模型的代表，它定义了5个不同的活动——启动、诊断、建立、行动和学习，通过这些SPI活动指导组织。

在本书中，基于原来在[Pre88]中提出的SPI过程模型，作者提出了一个有些不同的SPI路线图。该模型应用常识哲学，要求组织做到：（1）自我检查；（2）使过程更敏捷以便于做出智能的选择；（3）选择能最好地满足其需求的过程模型（以及相关的技术要素）；（4）在组织的运行环境和组织文化内将该模型实例化；（5）评价完成的工作。这5项活动（在随后的小节[⊖]中讨论）以一种迭代（循环）的方式应用，以便于促进持续的过程改进。

30.2.1 评估和差距分析

若先不评估当前框架活动和相关的软件工程实践的有效性，而是试图改善当前的软件过程，这样做的任何尝试都像是到一个新地方的漫长旅途的开始，而你却又不知道从哪里开始。你会充满兴致地出发，四处徘徊，试图弄清自己的处境，花费了大量的精力，承受挫折带来的痛苦。很可能，你决定真的不想再走了。简单地说，在你开始任何旅行之前，最好先准确地知道你在哪里。

确切地知道你的优势和弱点。如果你是聪明的，你就会取长补短。

路线图上的第一项活动是评估，要让你弄清自己的处境。评估的目的是揭示组织以某种方式应用现有软件过程及构成此过程的软件工程实践的优势和弱点。

评估在很宽的范围内检查活动和任务，这将导致高质量的过程。例如，不管是选择什么样的过程模型，软件组织必须建立通用的机制，如：定义客户沟通的方法；建立表示用户需求的方法；定义包括范围、估计、进度要求以及项目跟踪的项目管理框架；风险分析方法；变更管理规程；质量保证以及包括评审的控制活动等。在已经建立的框架活动和普适性活动（第2章）中，每项活动都经过了深思熟虑，并且要进行评估，以确定以下问题是否得到了解决：

- 是否清楚地定义了每项活动的目标？
- 是否标识和描述了需要作为输入的工作产品及作为输出产生的工作产品？
- 是否清晰地描述了要执行的工作任务？
- 是否通过角色标识了执行这些活动的人员？
- 入口和出口标准已经建立了吗？
- 活动的度量是否已经建立？
- 支持这些活动的工具是否是可用的？
- 针对这些活动有明确的培训大纲吗？
- 对所有的项目，活动是否一致地执行？

尽管上述这些问题的答案不是“是”就是“否”，对评估的作用是看后面的答案，以确定行动问题的执行方式是否符合最佳实践。

评估活动期间你所寻找的一般属性是什么？

随着过程评估的实施，你（或那些已经被雇佣执行评估的人）应关注以下属性：

一致。所有的软件团队是否在所有的软件项目中一致地应用了重要的活动、

⊖ 经过允许，[Pre88]中的一些内容已经重新改写。

行动和任务？

成熟。执行了一定成熟级别的管理和技术行动是否意味着对最佳实践有了透彻的理解？

认可。软件过程和软件工程实践是否得到了管理部门和技术人员的广泛认可？

承诺。管理者已经承诺为达到一致、成熟和认可所需要的资源了吗？

本单位应用和最佳实践之间的差异表示存在改进机会的“空间”。在何种程度上能够获得一致、成熟、认可和承诺表明：需要做多少文化上的变更才能获得意义深远的改进。

30.2.2 教育和培训

尽管很少有人怀疑敏捷的、有组织的软件过程或者实体的（相对于敏捷的而言——译者注）软件工程实践的好处，但很多从业人员和管理者并不都充分了解这些课题[⊖]。因此，在引入SPI框架时，对过程和实践不正确的认识会导致不恰当的决定。由此得出结论，任何SPI策略的关键要素是对从业人员、技术管理人员和直接接触软件组织的高级管理人员的教育和培训。三种类型的教育和培训应该进行：

针对软件团队的真正需要，尽量提供“及时”的培训。

一般的概念和方法：直接面向管理者和从业人员，这类教育和培训既强调过程又强调实践。目的是给从业者提供智能的工具，利用这些工具可以更有效地提高软件过程，并且对改进过程做出合理的决策。

特定的技术和工具：主要面向从业人员，这类教育和培训突出那些本单位已经采用的技术和工具。例如，如果已经选择UML进行分析和设计建模，就应该设置使用UML的软件工程培训课程。

商业交流和质量相关的课题：直接面向所有利益相关者，这类教育和培训关注能帮助所有利益相关者更好地交流和促进获得更好质量的“软”课题。

在现代化条件下，教育和培训可以包括各种不同的方式。一切从播客到以互联网为基础的培训（例如[QAI08]）、到DVD、再到教室的课程都可以提供SPI策略的一部分。

当你在做选择时，一定要考虑你的组织文化以及对每个选择的接受程度。

30.2.3 选择和合理性判定

一旦已经完成了最初的评估活动[⊖]，并且教育已经开始，软件组织就应该开始做出选择。这些选择在选择和合理性判定活动期间作出，其中，选择过程特性及特定的软件工程方法和工具在软件过程中占有重要位置。

首先，应该选择最适合你的组织、利益相关者和所开发软件的过程模型（第2章和第3章）。应该决定应用一组框架活动中的哪一个、要生产的主要工作产品以及使团队能够评估进展的质量保证检查点。如果SPI评估活动表明了一些特定的弱项（例如，不正规的SQA功能），则应该关注那些与弱项直接相关的过程特性。

其次，对每个框架活动（例如，建模）进行工作分解，定义应用于典型项目中的任务集。还应该考虑能完成这些任务的软件工程方法。当选择一旦确定下来，就应该协调教育和培训，以确保加强理解。

理想的情况是每个人都参与选择各种过程和技术要素的工作，并且向着设置和迁移活动（30.2.4节）顺利地过渡。实际上，选择活动可能是一项艰难的活动。在不同的支持者之间达成共识经常是很难的。如果委员会确立了选择的标准，人们可能会喋喋不休地争论标准是否是

⊖ 如果你花了很多时间读本书，你就不会像他们一样。

⊖ 实际上，评估是一项持续的活动。它需要定期进行，以确定SPI策略是否实现了其近期目标，并设定今后改进的阶段任务。

适当的以及做出的选择是否真正符合已确立的标准。

诚然，不好的选择会比好的选择造成更大的危害。但是“分析麻痹（指注意力集中在一点上而导致动作不连贯）”意味着几乎没有取得任何进展，过程中出现的问题依然存在。只要过程特性或技术要素有满足组织需要的很好时机，有时候，行动起来并且做出选择更好，而不是等待最优的解决方案。

一旦做出了选择，时间和金钱必须花在组织内的实例化中，这些资源支出应该是合理的。SPI成本合理性判定以及投资收益率将在30.7节进行讨论。

30.2.4 设置/迁移

设置是由于实施了SPI路线图软件组织感受到变更效果的第一项工作。在某些情况下，将一个全新的过程推荐给组织，必须定义框架活动、软件工程行动以及人员的工作任务，并作为新的软件工程文化的一部分进行设置。这样的变化表示重要组织和技术的变迁，需要精心地管理。

在其他情况下，与SPI相关联的变化相对较少，但对已有的过程模型做了有意义的修改。通常将这样的变化称为过程迁移。今天，很多软件组织都有恰当的“过程”。问题是，现有过程的工作效率可能不高。因此，从一个过程（不像期望的那样工作）到另一个过程的增量式迁移是更有效的策略。

设置和迁移实际上是软件过程再设计（software process redesign, SPR）活动。Scacchi[Sca00]认为“SPR是与识别、应用和求精相关的一种新方式，并能极大地提高和改变软件过程”。当对SPR启动形式化方法时，需要考虑3个不同的过程模型：（1）已存在的（“现有”）过程；（2）过渡的（“这里到那里”）过程；（3）目标（“将要成为的”）过程。如果目标过程和已存在过程有很大的差别，设置的唯一合理方法是采用增量策略，分步执行过渡过程。过渡过程提供了一系列导航点，能保证软件组织文化经过一段时间可以适应一些小的变化。

30.2.5 评价

评价活动在整个SPI中都存在的，尽管将其列为SPI路线图的最后一项活动。评价活动评估设置及采纳变更的程度、这些变更在多大程度上提高了软件质量或其他可见的过程收益以及随着SPI活动的进行过程和企业文化的总体状况如何。

在评价活动中，定性因素和定量度量都要考虑。从定性的角度看，过去的管理者和业务人员对软件过程的态度可以和设置过程后接受调查的态度进行对比。定量度量（第25章）是从已经使用过渡过程或目标过程的项目中收集的，并与为使用现有过程的项目所收集的类似度量进行比较。

30.2.6 SPI的风险管理

SPI经常失败，因为没有恰当地考虑风险，也没有制定应急计划。

SPI是有风险的。实际上，在所有的SPI尝试中，一半以上都以失败而告终。失败的原因各不相同，且与特定的组织有关。最普遍的风险是：缺少管理者的支持，技术人员文化上的抗拒，规划糟糕的SPI策略，SPI过度形式化的方法，选择了不恰当的过程，缺少主要利益相关者的投资，不合适的预算，工作人员缺少培训，组织的不稳定以及很多其他因素。列出这些与SPI相关风险的作用是分析可能的风险，并制定减轻这些风险的内部策略。

软件组织应该就SPI过程从以下3个关键点进行风险管理[Sta97b]：在启动SPI路线图之前，在执行SPI活动（评估、教育、选择、设置）期间，以及一些过程特性实例化之后的评估活动期间。一般地，可以对SPI风险因素进行下面的分类[Sta97b]：预算和成本、内容和交付、文化、SPI交付物的维护、任务和

目标、组织的管理者、组织的稳定性、过程的利益相关者、SPI工作开展的时间安排、SPI工作开展环境、SPI工作开展过程、SPI项目管理以及SPI工作人员。

在每一类中，定义了很多通用的风险因素。例如，组织文化对风险产生重大的影响。从组织文化方面，可以定义如下的通用风险因素[⊖][Sta97b]：

- 对变革的态度，这与对变革的前期工作量投入相关
- 质量大纲的经验，成功的程度
- 解决问题的行动定向与对方针不同意见的争论
- 使用事实管理组织和业务
- 改变的耐心，花时间参与交往的能力
- 提倡采用工具——期望工具能够解决问题
- “计划充实性”的等级——对计划的组织能力
- 组织成员在各级组织会议上公开的参与能力
- 组织成员有效地掌控会议的能力
- 在组织明确的过程中经验的等级

使用风险因素和通用属性作为指导，风险显露度（exposure）可以通过下面的方式计算：

$$\text{风险显露度} = (\text{风险可能性}) \times (\text{估计的损失})$$

为了避开这些风险，开发了一个风险表（第28章）以保证管理者的进一步关注。

30.2.7 关键的成功因素

在30.2.6节曾提到：SPI是一种有风险的尝试，对于那些努力改进过程的公司来说失败率是相当高的。组织风险、人员风险以及项目管理风险对任何实施SPI尝试的组织都提出了严峻的挑战。尽管风险管理很重要，但认识到那些导致成功的关键因素同样重要。

在考察了56个软件组织及其在SPI方面投入的工作以后，Stelzer和Mellis[Ste99]认为：如果要使SPI取得成功，必须给出一组关键的成功因素（critical success factor，CSF）。本节给出其中5个最重要的CSF。

? 对于成功的SPI，哪些关键的成功因素是决定性的？

管理者的承诺和支持：像大多数使组织和文化发生变更的活动一样，SPI要想获得成功必须有管理者的积极参与。高级业务管理者应该意识到软件对于公司的重要性并且积极支持SPI尝试。技术经理应该深入地参与到本单位SPI策略的开发中。如上述作者的研究中表明的那样：“没有时间、金钱和工作量的投入，软件过程改进是不可行的”[Ste99]。管理者的承诺和支持对维持投资是至关重要的。

工作人员的参与：SPI不能自顶向下地强硬执行，也不能从外界强加。SPI尝试要取得成功，改进必须是根本性的——由技术管理者和高级技术人员发起，并由本单位业务人员接受。

过程的集成和理解：软件过程并不是在一个组织内孤立存在的，它必须以某种方式和业务过程或需求集成起来。为了做到这一点，那些负责SPI实施的人必须精通其他业务过程，并有很好的理解。此外，他们必须理解软件过程的“现状”，并且知道在本单位文化的范围内做多大变迁是可以容忍的。

定制的SPI策略：没有任何千篇一律的SPI策略。如本章前面部分提到的，SPI路线图必须能够适应本单位的环境——团队文化、产品组合以及本单位的长项和弱项都应该考虑到。

⊖ 本节提到的每类风险因素都可以在[Sta97b]中找到。

对SPI项目的可靠管理：如其他项目一样，SPI项目包括协调、进度计划、并行任务、可交付产品、适应性（当风险成为现实时）、政策、预算控制甚至更多。没有积极、高效的管理，SPI项目注定会失败。

30.3 CMMI

WebRef

关于CMMI的完整信息可以从www.sei.cmu.edu/cmmi/获得。

作为完整的SPI框架，最初的CMM是由卡内基·梅隆大学软件工程研究所（Software Engineering Institute，SEI）在20世纪90年代开发并升级的。今天，它已经演变为能力成熟度模型集成（Capability Maturity Model Integration，CMMI）[CMM07]，它是一个综合的过程元模型，以一组系统和软件工程能力为基础，能够表示组织可以达到的过程能力及成熟度的不同等级。

CMMI以两种不同的方式表示过程元模型：（1）作为一个“连续式”模型；（2）作为一个“分级式”模型。“连续式”CMMI元模型以两个维度描述过程，如图30-2所示。每个过程域（例如，项目策划或需求管理）根据特定目标和特定实践进行正式评估，并且与下面的能力等级相关联：

能力等级0：不完全级（Incomplete）——过程域（例如，需求管理）或者没有执行，或者已经执行，但没有达到该过程域CMMI 1级成熟度所规定的所有目标。

能力等级1：已执行级（Performed）——（由CMMI所定义的）过程域的所有特定目标都已经满足。生产已规定工作产品所需的工作任务都已执行。

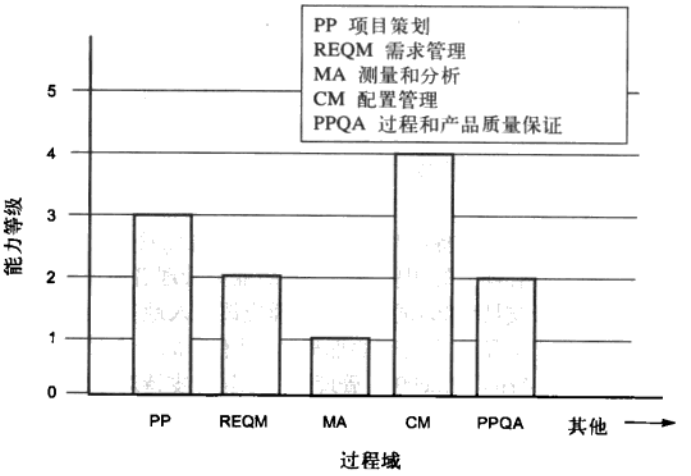


图30-2 CMMI过程域能力分布图（来源：[Phi02]）

ADVICE

每个组织都应该努力获得CMMI的真正意图。然而，模型每个方面的实现都有可能对你的现状具有矫枉过正的影响。

能力等级2：已管理级（Managed）——能力等级1中所有的标准都已经满足。此外，所有与过程域相关的工作都符合组织规定的方针，所有的工作人员都可以得到完成工作所需的足够资源，利益相关者都按照需要积极地投入到过程域中，所有的工作任务和工作产品都可以被“监督、控制和评审，并评估是否与过程描述相一致”[CMM07]。

能力等级3：已定义级（Defined）——能力等级2中所有的标准都已经满足。另外，这个过程是“根据组织剪裁准则，对其标准过程进行了剪裁，剪裁过的过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”[CMM07]。

能力等级4：定量管理级（Quantitatively managed）——能力等级3中所有的标准都已经满足。此外，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”[CMM07]。

能力等级5：优化级（Optimized）——能力等级4中所有的标准都已经满足。此外，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。

CMMI定义了每个过程域的“特定目标”，以及为达到这些目标所需的“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，软件过程应具备的特点。特定实践将目标细化成一组过程相关的活动。

例如，**项目策划**是CMMI为“项目管理”类定义的8个过程域之一[⊖]。为项目策划定义的特定目标（Specific Goals，SG）和相关的特定实践（Specific Practices，SP）如下[CMM07]：

WebRef

关于CMMI的完整信息和可下载版本可以从www.sei.cmu.edu/cmmi/获得。

SG 1 确立估计值

SP 1.1-1 确立项目范围
SP 1.2-1 建立工作产品和任务属性的估计值
SP 1.3-1 定义项目的生命周期
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 编制预算和进度计划表
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 制定利益相关者的参与计划
SP 2.7-1 制定项目计划

SG 3 获取对计划的承诺

SP 3.1-1 评审影响项目的计划
SP 3.2-1 使工作和资源投入相匹配
SP 3.3-1 获得对计划的承诺

除了特定目标和特定实践，CMMI还为每个过程域定义了一组5个通用目标和相关的实践。其中，每个通用目标都对应着5个能力等级之一。因此，要想达到特定的能力等级，就必须满足该等级对应的通用目标和相应的通用实践。为了说明这一点，**项目策划**过程域的通用目标（Generic Goals，GG）和通用实践（Generic Practices，GP）定义如下[CMM07]：

GG 1 达到特定目标

GP 1.1 执行基本实践

GG 2 使已管理过程制度化

GP 2.1 制订组织方针
GP 2.2 策划过程
GP 2.3 提供资源
GP 2.4 分配职责

⊖ 为“项目管理”定义的其他过程域包括：项目监控、供应商协议管理、IPPD的集成项目管理、风险管理、集成团队建立、集成供应商管理以及定量项目管理。

GP 2.5 培训人员
GP 2.6 管理配置
GP 2.7 识别并吸纳相关的利益相关者
GP 2.8 监督和控制过程
GP 2.9 客观地评价遵循情况
GP 2.10 与高层管理者一起评审状态

GG 3 使已定义过程制度化

GP 3.1 建立已定义过程
GP 3.2 收集改进信息

GG 4 使定量管理过程制度化

GP 4.1 建立过程的量化目标
GP 4.2 稳定子过程绩效

GG 5 使优化过程制度化

GP 5.1 保证过程持续改进
GP 5.2 消除造成问题的根本原因

阶段性的CMMI过程模型定义了和持续性模型相同的过程域、目标和实践。主要区别是阶段性模型定义了5个成熟度等级，而不是5个能力等级。要达到某个成熟度等级，就必须实现与一组过程域相关的特定目标和特定实践。成熟度等级和过程域之间的关系如图30-3所示。

等级	焦点	过程域
优化级	持续过程改进	组织创新和部署 原因分析和消除
定量管理级	定量管理	组织过程绩效 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 集成供应商管理 风险管理 决策分析和决定 组织集成环境 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 项目监控 供应商协议管理 测量和分析 过程和产品质量保证 配置管理
初始级⊖		

图30-3 达到成熟度等级所需的过程域（来源：[Phi02]）

⊖ 译者注：原书此处错为“已执行级”。

INFO

CMMI——应不应该使用？

CMMI是一个过程元模型。它（用700多页）定义了软件组织想建立完整的软件过程应该具备的过程特性。已经争论了十多年的问题是："CMMI是否执行过度了？"像日常生活中（以及软件中）大多数事情一样，答案不是简单的"是"或"否"。

SPI的精神应该总是采用。为避免过于简化的风险，CMMI认为软件开发过程必须严肃对待——必须计划周全，必须统一控制，必须准确跟踪以及必须专业化地执行。必须关注项目利益相关者的需要、软件工程师的技能以及最终产品的质量。任何人都不应该质疑上述观点。

如果软件组织要构建大型复杂的系统，此系统需要几十人或几百人参与、历时很多月或很多年，就应该认真考虑CMMI的具体要求。如果组织的文化符合标准过程模型，并且管理者又承诺要使其获得成功，这也许正是适合使用CMMI的场合。然而，在其他情况下，CMMI的内容可能是太多了，组织不能很好地采纳。这意味着CMMI是"坏的"或"过于官僚"或"老式"吗？不……不是这样。它只是意味着适用于一种组织文化的东西可能并不适用于另一种组织文化。

CMMI是软件工程的一项重要成就。在构建计算机软件时到底采用哪些活动和动作，CMMI对此提供了全面的讨论。即使软件组织不采用它的细节，每个软件团队都应该接受它的精神，并且从CMMI对软件工程过程和实践的讨论中得到启发。

30.4 人员CMM

KEY POINT

人员CMM建议通过实践提高劳动力的技能和文化。

不管设想得多好，如果没有具备才能和积极性的软件人才，软件过程不会成功。人员能力成熟度模型（People Capability Maturity Model）"是实现劳动力实践的路线图，这些实践可以持续提高软件组织的员工能力"[Cur02]。人员CMM的目标是20世纪90年代中期开发的，其目标在于鼓励普通员工在知识（称为"核心能力"）、具体软件工程技能和项目管理技能（称为"劳动力能力"）以及过程相关能力诸方面的持续提高。

像CMM、CMMI以及相关的SPI框架一样，人员CMM定义了组织成熟度的5个等级，提供了员工实践和过程相对成熟的指标。这些成熟度等级[CMM08]与已有的一组关键过程域（key process area, KPA）（在一个组织内）相连。组织等级和相关的KPA概况如图30-4所示

人员CMM通过鼓励组织培养和改进其最重要的资产（即人员）来补充SPI框架。同样重要的是，它在员工中形成一种氛围，使软件组织能够"吸引、培养及留住优秀人才"[CMM08]。

等级	焦点	过程域
优化级	持续改进	持续的员工创新 组织绩效协调 持续的能力提高
可预测级	对知识、技巧和能力进行量化和管理	师徒制 组织能力管理 定量绩效管理 基于能力的估值 授权工作组 能力整合
已定义级	识别和开发知识、技巧和能力	分享文化 工作组开发 基于能力的实践 职业发展 能力发展 员工规划 能力分析
已管理级	可重复的、基本的人员管理实践	薪酬 培训和开发 绩效管理 工作环境 交流和协作 人员配备
初始级	不一致的实践	

图30-4 人员CMM的过程域

30.5 其他SPI框架

尽管卡内基·梅隆大学软件工程研究所的CMM和CMMI是应用最广泛的SPI框架，但仍然有一些其他框架[⊖]被提出及应用，其中广泛应用的一些框架如下：

(1) SPICE——国际倡导的过程标准[SPI99]，支持ISO过程评估和生命周期。

(2) ISO/IEC 15504——用于（软件）过程评价[ISO08]。

(3) Bootstrap——遵循SPICE的适用于中小组织的SPI框架[Boo06]。

(4) PSP 和 TSP——个体和团队特定的SPI框架（[Hum97]，[Hum00]），更侧重小型软件过程，是进行带有度量的软件开发的更严格的方法。

(5) TickIT——一种审计方法[Tic05]，该方法评估一个组织是否符合ISO 9001：2000标准。

除了CMM，还有其他SPI框架可以考虑吗？

以下各段对这些SPI框架作一简要介绍。每种SPI框架都有大量的印刷版和网络版资源，有兴趣的读者可以参考。

SPICE。SPICE（Software Process Improvement and Capability dEtermination）模型提供了遵循ISO 15504：2003和ISO 12207的SPI评估框架。

⊖ 合理地说，这些框架中的一些并不是“替代物”，因为它们是SPI方法的补充。更多有关SPI框架的综合列表可以在www.geocities.com/lbu_measure/spi/spi.htm#p2找到。

SPICE文档套件[SDS08]描述了一个完整的SPI框架，包括过程管理模型，对考虑中的过程进行评估和评级的指南，对评估设备和工具的构造、选择和使用，以及评估师的培训。

Bootstrap。Bootstrap SPI框架“已经被开发出来，目的是确保与最新的软件过程评估和改进（SPICE）的ISO标准相一致，并且与ISO 12207相协调”[Boo06]。Bootstrap的目标是使用一系列软件工程最好的实践作为评估的基础，评价一个软件过程。像CMMI一样，Bootstrap根据收集到的软件工程和软件项目“现状”的调查问卷情况，给出过程成熟度等级。SPI的指南是基于成熟度等级和组织目标的。

PSP和TSP。虽然一般将SPI描述为组织活动，但没有理由说明不能在个人或团队级别上进行过程改进。PSP 和TSP（第2章）都强调需要持续地收集进行中的工作数据，并且利用这些数据开发改进策略。PSP 和TSP方法的提出者Watts Humphrey[Hum97]给出了如下评论：

“软件组织已经表现出他们从已完成的项目中获取经验的能力明显不足。” —— NASA

PSP[和TSP]展示了如何计划和跟踪工作，如何始终如一地生产高质量的软件。使用PSP[和TSP]会提供数据表明工作效率及标识你的强项和弱项……要想获得成功和高薪的职业，就需要知道自己具备的技巧和能力，并努力去提高，在所从事的工作中利用好自己的独特才能。

TickIT。TickIT审计方法确保和用于软件的ISO 9001:2000相一致。ISO 9001:2000是通用标准，它适用于任何想改进其产品、系统或者服务质量的组织。因此，这个标准可直接应用于软件组织和公司。

ISO 9001:2000提出的基本策略如下[ISO01]：

ISO 9001:2000强调对于一个组织识别、实施、管理和持续提高过程有效性的重要性，这对于质量管理体系是必须的，为了达到组织的目标还要管理这些过程之间的相互作用……过程有效性和效率可以通过内部和外部的评审过程来评估以及根据成熟度等级来评价。

WebRef

关于ISO 9001:2000的优秀总结可以在http://praxiom.com/iso-9001.htm找到。

ISO 9001:2000已经采用了“计划－实施－检查－行动”循环，它适用于软件项目的质量管理要素。在软件的环境中，为了获得高质量软件并达到客户满意，“计划”要建立所需要的过程目标、活动和任务。“实施” 执行软件过程（包括框架活动和普适性活动）。“检查”监控和检测过程，以保证可以达到质量管理所提出的所有要求。“行动”着手于软件过程改进活动，使改进过程持续地进行。可以在整个“计划－实施－检查－行动”周期中使用TickIT，以保证SPI的进展。作为ISO 9001:2000认证的先驱，TickIT审核员评估上述循环的应用情况。关于ISO 9001:2000和TickIT更详细的讨论可以参考[Ant06]、[Tri05]或[Sch03]。

30.6 SPI的投资收益率

SPI是艰苦的工作，需要投入大量的财力和人力。那些批准SPI预算和资源的管理者总是会问这样的问题：“怎样才能知道我们所投入的资金会取得合理的回报？”

在定性的层次上，SPI的拥护者认为，改进软件过程将会带来软件质量的提高。他们主张，改进了的过程将导致实施更好的质量筛选（结果是减少缺陷的传播）、更好地变更控制（结果是减少项目造成的混乱）、更少的技术返工（从而降低成本并且能获得更好的上市时机）。但是这些定性的收益能转变成定量的结果吗？经典的投资收益率（return on investment, ROI）等式如下：

$$ROI = \left[\frac{\Sigma(benefits) - \Sigma(costs)}{\Sigma(costs)}\right] \times 100\%$$

其中，benefits代表“收益”，costs代表“成本”。

收益包括与更高的产品质量（更少的缺陷）、更少的返工、变更方面减少的工作量相关的成本节省，以及从缩短上市时间中获得的收入。

成本包括直接的SPI成本（例如，培训费、测量费），也包括与强调质量控制和变更管理活动以及应用更严格的软件工程方法相关的间接成本（例如，设计模型的创建）。

在现实世界中，这些定量的收益和成本有时是难以准确计量的，并且所有都是可以开放解释的。但这并不意味着软件组织应该对增加的成本和收益不经过仔细的分析就可以实施SPI程序。关于SPI的投资收益率更全面的叙述可以在David Rico唯一的一本书中找到[Ric04]。

30.7 SPI趋势

在过去的20年中，很多公司都在尝试应用SPI框架改进他们的软件工程实践，这些SPI框架影响了组织的变化和技术的变迁。正如本章前面提到的，超过一半的尝试失败了。不管成功还是失败，都耗费了大量的金钱。David Rico[Ric04]报道了SPI框架的一个典型应用，如SEI CMM的花费是每人2万5千美元到7万美元之间，并且需要数年才能完成。其实，这毫不奇怪，SPI的未来应该强调一种成本较低并且费时较少的方法。

为了使21世纪软件开发更有效率，未来的SPI框架必须变得更加敏捷。不是以组织为关注点（这可能需要很多年才能够成功完成），当前的SPI努力应集中在项目层面上，努力在几个星期内改进团队过程，而不是几个月或几年。要想在很短的时间内取得有意义的成果（即使在项目级别上），复杂的框架模型可能要让位于简单的模型。要求做到十几个关键实践和数以百计的补充实践，不如强调仅有的几个核心实践（例如，类似于本书所讨论的框架活动）。

SPI方面的任何尝试都需要具有一定知识的人员，但是教育和培训费用可能是昂贵的，应该将其最小化（使其流线化）。未来的SPI实施工作应该依靠基于网络的培训，定位于核心的实践，而不是课堂上的课程（昂贵和费时）。不要去做改变组织文化（这可能带来涉及组织方针的风险）的深远尝试，正如现实世界中一样，在某一时刻在一个小的团体内直到出现了一个转折点，文化变革才会发生。

在过去的20年中，SPI工作具有重要的价值。已经开发的框架和模型代表了软件工程界重要的智力资产。但是像所有的事情一样，这些资产对SPI未来尝试的指导并不是可重复性的教条，而是作为更好的、更简单的和更敏捷的SPI模型的基础。

30.8 小结

软件过程改进框架定义了一些特性（要获得有效的软件过程，就必须具备这些特性）、一种评估方法（有助于确定是否具备了这些特性）以及一种策略（辅助软件工程组织实现那些薄弱或缺失的过程特性）。无论发起SPI的人是谁，其目标是提高过程质量，进而提高软件的质量和产品交付的及时性。

过程成熟度模型从总体上体现了软件组织呈现的“过程成熟度”，对于正在使用的软件过程的相对有效性提供了定性的认识。

SPI的路线图开始于评估——一系列的评价活动，通过组织应用的现有软件过程和作用于这些过程的软件工程实践的方式，既揭示了优势，也揭露了缺陷。作为评估的结果，软件组织可以制定一个整体的SPI计划。

任何SPI计划的关键要素之一是教育和培训，该活动主要关注于提高管理者和从业人员的知识水平。一旦工作人员精通了目前的软件技术，选择和合理性判定就可以开始了。这些任务导致对软件过程体系架构、采用的方法和支持工具的选择。设置和评价是SPI活动，这些活动将过程的改变具体化，并可以评估其有效性和影响。

要想成功地改进软件过程，一个组织必须具备以下的一些特征：管理者对SPI的承诺和支持，工作人员参与SPI的整个过程，过程集成到整个组织文化，制定适应本单位要求的SPI策略以及SPI项目的可靠管理。

今天，已经有很多SPI框架在使用。卡内基·梅隆大学软件工程研究所的CMM和CMMI已经广泛应用。对人员CMM进行定制可用以评估组织文化质量和其中的人员。SPICE、Bootstrap、PSP、TSP和TickIT是另外一些有效的SPI框架。

SPI是艰苦的工作，要求投入大量的财力和人力。为了保证获得合理的投资回报，一个组织必须估计与SPI相关的费用和直接利益。

习题与思考题

30.1 为什么软件组织在开始努力改善当地的软件过程的时候经常陷入斗争？

30.2 用你自己的语言描述“过程成熟度”这个概念。

30.3 做一些研究（查看SEI站点），确定美国和全世界的软件组织的过程成熟度分布情况。

30.4 你在一个非常小的软件组织工作——只有11个软件开发人员。SPI适合你吗？解释你的答案。

30.5 评估类似于每年的体检。用体检作为比喻，描述一下SPI评估活动。

30.6 “当前”过程、“过渡”过程和“目标”过程三者之间的区别是什么？

30.7 在SPI环境中如何实现风险管理？

30.8 选择30.2.7节中的一个关键成功因素，做一些研究，并撰写一篇简短的论文说明它是如何实现的。

30.9 做一些研究，并解释CMMI与其前身CMM在哪些方面存在差异。

30.10 从30.5节讨论的SPI框架中选择一个，并撰写一篇简短的论文给出更详细的描述。

推荐读物与阅读信息

SPI方面的最容易获取的、综合信息资源之一是由卡内基·梅隆大学软件工程研究所（SEI）开发的，其网址是www.sei.cmu.edu。SEI的网站包含数百篇论文、研究报告和详细的SPI框架描述。

过去的几年中，在以往20年开发的广泛的图书文献中又增加了一些有价值的书籍。Land（《Jumpstart CMM/CMMI Software Process Improvements》，Wiley-IEEE Computer Society, 2007）将SEI CMM和CMMI的一部分需求与IEEE软件工程标准相融合，强调软件过程和实践的交叉。Mutafelija和Stromberg（《Systematic Process Improvement Using ISO 9001:2000 and CMMI》，Artech House Publishers, 2007）讨论了ISO 9001:2000和CMMI SPI框架及它们之间的“协同作用”。Conradi和他的同事（《Software Process Improvement: Results and Experience from the Field》，Springer, 2006)介绍了一系列案例研究和相关的SPI实验结果。Van Loon（《Process Assessment and Improvement: A Practical Guild for Managers, Quality Professionals and Assessors》，Springer, 2006）在ISO/IEC 15504环境下讨论了SPI。Watts Humphrey（《PSP》，Addison-Wesley, 2005；《TSP》，Addison-Wesley, 2005）在两本不同的书籍中描述了个人团队过程SPI框架和团队软件过程SPI框架。Fantina（《Practical Software Process Improvement》，Artech House Publishers, 2004）以CMMI/CMM为重点提供了务实的操作方法指导。

更广泛的关于软件过程改进的各种信息来源可以通过因特网获取。最新的SPI相关的WWW参考文献列表可以在SEPA站点www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm找到。

第31章

Software Engineering：A Practitioner's Approach (Seventh Edition)

软件工程的新趋势

要点浏览

概念：没有人能够绝对准确地预测未来。但是推测一下软件工程领域未来的趋势并从中给出一些关于技术可能的发展方向的建议，的确是可以做到的。这也正是本章要达到的目的。

人员：任何愿意花时间研究软件工程问题的人都可以尝试预测技术的未来发展方向。

重要性：为什么古代的国王会雇佣占卜者？为什么大多数的跨国公司雇佣咨询公司和智囊团进行预测？为什么相当多的公众相信算命？大家都是想知道什么将要发生，以便我们能做好准备。

步骤：预测前面的路没有什么固定的公式。我们试图通过收集数据，组织这些数据提供出有用的信息，为抽取知识要检查细微的联系，从这些知识中给出可能趋势的建议，这些趋势可以预测未来事情会变得怎么样。

工作产品：对近期的一种看法，可能对，也可能不对。

质量保证措施：预测未来的道路是一门艺术，不是科学。实际上，对未来认真的预测绝对正确或错误（幸好，对世界末日的预言例外）都是十分罕见的。我们寻找趋势并尽量推算它们。我们只能随着时间的流逝评估推算的正确性。

关键概念
构造模块
协同开发
复杂度
应急需求
趋势周期
创新生命周期
模型驱动的开发
开放软件
开放源码
后现代设计
需求工程
软趋势
技术方向
技术演变
测试驱动的开发
工具

在整个软件工程相对短暂的历史中，从业人员和研究人员已经开发了一系列过程模型、技术方法和自动化工具，努力促进构造计算机软件方式的根本变化。尽管过去的经验表明有一个心照不宣的愿望要找到“银弹”——神奇的过程或卓越的技术，使我们很容易地构造大型复杂的基于软件的系统，而没有混淆、没有错误、没有拖延——没有继续困扰软件工作的众多问题。

但是历史表明我们寻找银弹好像注定要失败。新技术不断地引入，作为软件工程师面临的很多问题的“解决方案”大肆地宣传，成为大型或小型项目的一部分。业界权威强调这些“新的”软件技术的重要性，软件界的行家满腔热情地采用这些技术，最终，它们确实在软件工程世界中起了作用。但它们往往并没有履行其承诺，因此，人们的探索还在继续。

Mili和Cowan[Mil00b] 对我们面临的挑战发表评论，试图分离出这些有意义的技术趋势。

什么因素决定趋势的成功？成功的技术趋势具有什么样的特征：它们的技术价值？它们开拓新市场的能力如何？它们改变现有市场经济的能力如何？

趋势遵循什么样的生命周期？尽管传统的看法是，趋势沿着定义好的、可预测的生命周期演变，通过一个传递过程，从调查所得的想法到产生一个成品。但我们发现，目前的很多趋势要么在这个周期上行不通，要么符合另一个。

一个成功的趋势怎样能做到早期识别？如果我们知道怎样识别成功要素，和（或）我们理解趋势的生命周期，那么我们希望找出趋势成功的早期迹象。夸张地说，我们想要寻求能够在其他人之前认识未来趋势的能力。

? 当我们考虑技术演变时，“大问题”是什么？

演变的哪些方面是可控的？公司能利用他们的市场影响力强加于趋势吗？政府能利用其资源强加于趋势吗？在判定趋势中标准起了什么作用？例如，对Ada和Java进行仔细分析，应该在这方面有所启发。

这些问题没有现成的答案，在判定有意义的技术方面，可能无需争辩，过去的尝试充其量是最平庸的。

在本书过去的版本中（过去30年间），我讨论了一些新技术以及它们对软件工程的预期影响。一些技术已经被广泛采用，但另外一些却从没有达到它的潜能。我的结论是：技术来来去去，你和我应该探索的真正趋势是软趋势。我的意思是，软件工程的进展将遵循业务、组织、市场和文化的趋势。这些趋势导致了技术的变革。

在这一章中，我将讨论几种软件工程技术方面的趋势，更多的是讨论在商业方面、组织方面、市场方面以及文化方面的一些趋势，这些可能在未来的10到20年对软件工程技术有着很重要的影响。

31.1 技术演变

在一本有吸引力的书中提出了引人注目的观点：计算（和其他相关的）技术是如何发展的？Ray Kurzweil [Kur05] 认为技术的发展，类似于生物进化，但其增长速度却比生物进化快几个数量级。演进（不管是生物或技术）作为正反馈的结果出现——“从进化过程的一个阶段所产生的更好方法可用于创建下一个阶段”[Kur06]。

21世纪最大的问题是：（1）技术怎样才能快速演进？（2）积极反馈有多么重要的影响？（3）必然要发生的变化将有多么深远的意义？

“做出预测是非常困难的，特别是对未来的预测。”——Mark Twain

当引入一种成功的新技术时，最初的概念转化成合理的预言“创新生命周期”[Gai95] 如图31-1所示。在突破阶段，一个问题是公认的，并且不断试图尝试一个可行的解决方案。在某种程度上，一个解决方案显示了希望。最初的突破性工作在复制阶段获得再生，并获得更广泛的使用。经验会导致经验规则的创造，这些规则支配技术的使用，多次成功导致了广泛使用的理论，它是在自动化阶段由自动化工具创建的。最后，技术成熟并被广泛使用。

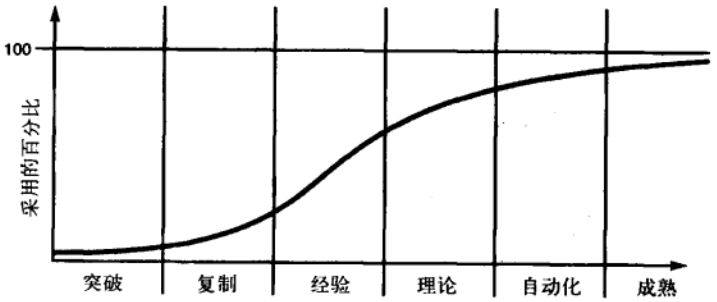


图31-1 技术演变的生命周期

应该注意到很多研究和技术趋势从来都没有成熟。实际上，绝大多数软件工程领域“有希望的”技术在最近几年得到了广泛关注，然后通过专注的拥护者得到恰当的使用。这一点并不是说这些技术缺乏价值，而是表明通过创新生命周期的途径是漫长而艰难的。

Kurzweil [Kur05] 认为计算技术是以“S-曲线”形式演进的，这表明在技术的形成期增速是相当慢的，在增长期加速度很快，然后随着技术达到极限，达到稳定期。但是计算技术和其他

相关技术在图31-1的中间阶段呈现爆炸性（指数的）增长，并且会继续这样。此外，作为S-曲线的结束，在它的增长期，另一个更具爆炸性的增长取而代之[⊖]。今天，我们正处于现代计算技术S-曲线的中下位置，在早期的增长和接下来的爆炸性增长之间的过渡阶段。这意味着，在未来的20至40年，我们将看到计算能力戏剧性（甚至令人难以置信）的变化。今后几十年，在计算速度、规模、容量和能源消耗（仅举几个特性）等方面将会出现数量级规模的重大变化。

KEY POINT
计算技术以指数速度不断演进，其增长可能很快会成为爆炸性的。

Kurzweil [Kur05] 认为：在20年内，技术演变的加速度将会越来越快，最终导致非生物智慧的时代，它将融合并扩展人类的智慧，这是值得人们认真思考的。

相比之下，无论怎样演变，所有这一切，都需要软件和系统，这些软件和系统会使我们当前的努力显得十分幼稚。到2040年，极值理论、纳米技术、大规模高带宽的普适网络和机器人的结合将带给我们一个完全不同的世界[⊖]。软件，可能以我们现在还无法理解的形式，继续成为这个新世界的核心。软件工程不会消失。

31.2 观察软件工程的发展趋势

"我认为这个世界市场有5台计算机就够了。"
—— Thomas Watson（IBM董事长，1943）

在31.1节中简单地探讨了计算技术和相关技术在长期发展趋势中有可能产生引人注目的事情。但是什么是近期的趋势呢？

Barry Boehm[Boe08]认为："软件工程师[将]经常面对令人生畏的挑战，处理快速的变化、不确定性和突发事件、可信性、多样性以及互相依赖等问题，但他们还有机会做得更好"。然而，今后几年我们面对这些挑战的趋势是什么呢？

在这一章的引言中，我提到"软趋势"对软件工程的整个发展方向有重要的影响。但是另外一些（"硬的"）面向研究和技术的趋势依然是重要的。研究趋势"是由技术发展水平和实践发展水平、从业者需要的研究人员的看法、能集成特定战略目标的国家拨款的程序以及纯粹的技术兴趣这些笼统的观念驱动的"[Mil00a]。当满足工业界需求和市场机制需要形成的研究趋势被推断出来时，技术趋势就产生了。

KEY POINT
"趋势周期"表示短期技术集成现实的视图，然而长期趋势是指数级的。

在31.1节中，曾讨论了技术演变的S-曲线模型。当技术演变时，S-曲线适合考虑核心技术的长期影响。但是什么是更适度的、短期的创新、工具和方法呢？Gartner Group [Gar08]——涉及许多行业研究技术发展趋势的顾问机构——已经开发了新兴技术的趋势周期，如图31-2所示。Gartner Group 周期显示了5个阶段：

(1) 技术触发——研究有所突破或创新推出了新产品，吸引了媒体的报道和公众的热情。

(2) 膨胀期望的顶点——基于有限的但广为人知的成功产生的过度热情和对影响过于乐观的预测。

(3) 幻灭——对影响过于乐观的预测没有达到，批评家开始抨击，该技术在专家中已不再流行。

⊖ 例如，在未来十年可能会受到集成电路的限制，但这种技术可以通过分子计算技术被另一个加速S-曲线所取代。

⊖ Kurzweil [Kur05] 提出了合理的技术论据，预测到2029年会出现强大的人工智能（会通过图灵测试），并建议对人类和机器的演变将会在2045年开始融合。这本书的绝大多数读者会活到那时，看看实际上是否是这样。

(4) 启蒙倾斜——大量公司不断增长的使用使得对该技术的真正潜力有了更好的认识，出现了支持该技术的成熟方法和工具。

(5) 生产力的稳定期——这时，该技术给现实世界带来的益处已经很明显了，其使用占据了潜在市场的很大比例。

并不是每项软件工程技术都要经历这种趋势周期。在某些情况下，幻灭是合理的，技术归于相形见绌。

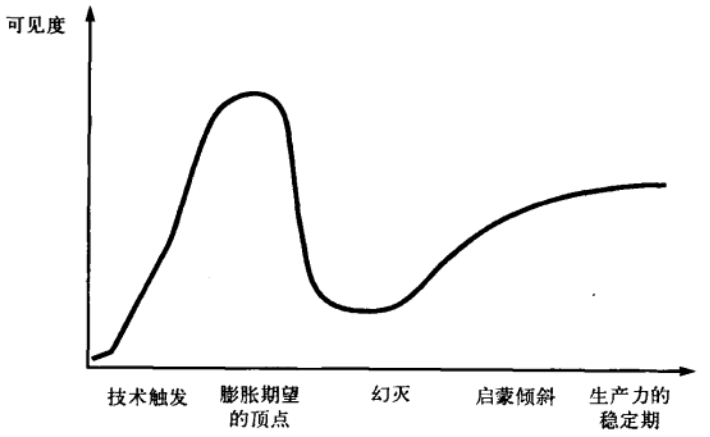


图31-2 Gartner Group关于新兴技术的趋势周期 （源自[Gar08]）

31.3 识别“软趋势”

“对于任何人，640k应该足够了。”——Bill Gates（微软董事长，1981）

每个有着强大IT产业的国家都具有独特的特征，这些特征决定了业务运作的方式，公司内呈现的组织动力，面对本地客户的显著销售额，决定所有人员互动的强势文化。然而，在这些领域中的一些趋势是普遍的，在社会学、人类学、群体心理学（通常称为“软科学”）方面做的研究和学术界与产业界方面做的研究差不多。

哪些软趋势会影响与软件工程相关的技术？

连接和协作（通过高带宽通信做到）可以使软件团队不占用同样的物理空间（实现远程交换和当地条件下的兼职工作）。一个团队可以与在时区、主要语言和文化不同的其他团队协同工作。软件工程必须以贯穿过程模型来应对“分布式团队”，也就是要足够敏捷，以满足即时的需求，而且要遵守纪律以协调不同的群体。

全球化导致了多种多样的劳动力（在语言、文化、问题求解、管理理念、交流偏好以及个人间的相互影响方面都存在差异）。这反过来又需要一个灵活的组织结构。不同的团队（在不同的国家）必须以某种方式对工程问题做出反应，这种方式最好能适应他们独特的需要，同时又促进了某种程度的统一性，使得全球的项目得以进行下去。这种类型的组织建议尽量少的管理层次，并且更重视团队级别的决策。这可以导致更大的灵活性，但前提是沟通机制已经建立，使每个团队在任何时候可以了解项目和技术状况（通过联网的群组软体）。软件工程方法和工具可以帮助实现某种程度的统一性（通过具体的方法和工具可以使团队讲同一种“语言”）。软件过程能为这些方法和工具的实例提供框架。

在世界的某些地区（例如，美国和欧洲），人口老化。不可否认的人口统计数字（和文化

趋势）意味着许多经验丰富的软件工程师和管理人员在未来十年会离开这个领域。软件工程界必须采取切实可行的机制保留住这些上了年纪的管理人员和技术人员所具有的知识[例如，模式（第12章）的使用就是向正确的方向迈出的一步]，使得未来新一代软件工作者能够获得这些知识。在世界上的其他一些地区，从事软件业的年轻人的数量正在迅速增长。这为铸造软件工程文化提供了机会，而没有50年来“陈旧学派”的偏见所带来的负担。

据估计，未来十年将有超过10亿的新消费者进入世界市场。消费者的支出在“新兴经济体将增加一倍，超过9万亿美元”[Pet06]。毫无疑问的是，这个非同一般的支出比例将适用于那些由数码组件构成的产品和服务，这些都是基于软件或软件驱动的。这将表明对新软件的需求在日益增长。接下来的问题是，“能开发出新的软件工程技术来满足这一全球的需求吗”？现代市场趋势通常是由供方驱动的[⊖]。另一种情况是需方的需求推动市场。无论哪种情况创新周期和需求进展在某种程度上有时很难确定哪个是领先的！

最后，人类文化本身将会影响软件工程的方向。每代人都建立了具有自己烙印的本地文化，任何人也不会例外。Faith Popcorn [Pop08]，一个专门研究文化趋势的著名顾问，总结了下面一些特征：“我们的趋势不是时尚的，而是持久的、发展变化的。他们代表了根本的力量，其首要原因是人类的基本需求、态度和愿望。他们帮助我们游览世界，了解正在发生的事情及其原因，并且为未来做好准备”。关于将会对软件工程产生影响的现代文化趋势的详细讨论，最好留给那些“软科学”的专业人士。

31.3.1 管理复杂性

“任何人都希望家里有电脑，这不需要什么理由。”——Ken Olson（数字设备公司创始人、总裁、董事长，1977）

写本书的第一版时（1982年），今天我们知道的这些数码消费产品当时还不存在。包含上百万行代码（LOC）的基于主机的系统就被认为是相当大型了。今天，小型数码设备包含6万到20万行代码的客户软件曾配有几百万行代码的操作系统，这种情况已经很常见了。现代计算机系统包含1千万到5千万行代码也是很常见的[⊜]。在不远的将来，需要超过1亿行代码的系统[⊜]将开始出现[⊜]。

想想那一刻。

考虑具有10亿行代码的系统，其接口既要连接外部的世界、其他互操作的系统、互联网（或其继承者），又要连接内部几百万个必须共同工作的构件，使得这个计算巨人成功运行。有没有一个可靠的方法，以确保所有这些连接都允许信息正确流动呢？

考虑项目本身。我们怎样管理工作流并跟踪进展？传统方法的规模是否能上升几个数量级？需要考虑的问题包括工作人员的人数（和他们的工作地点），人员和技术的协调，无情的变化，多平台的可能性，多操作系统环境。有没有一种方法来管理和协调在一个大型项目中工作的人员呢？

再来考虑工程的挑战。我们如何分析成千上万的需求、约束和限制，确保能够发现和纠正不一致性和不确定性、遗漏和错误？我们如何才能设计出一个强大的体系结构足以处理这种规

⊖ 供方采用了“构造它，消费者会来的”的方式进入市场。有时候，一旦发明了独特的技术，消费者就会趋之若鹜。

⊜ 例如，现代PC机操作系统（例如，Linux、MacOS以及Windows）都有3千万到6千万行代码。移动设备的操作系统软件也超过了200万行代码。

⊜ 在现实中，这个“系统”实际上是具有很多系统的系统——为了取得某个总体目标，数百个互操作的应用系统一起工作。

⊜ 并非所有复杂的系统都是大型系统。一个相当小的应用系统（如小于10万行代码）也可能相当复杂。

模的系统呢？软件工程师如何才能建立一个变更管理系统，可以处理成千上万的变更呢？

还需考虑质量保证的挑战。我们如何才能以一种有意义的方式进行确认和验证呢？又怎么测试一个10亿行代码的系统？

在早期，软件工程师试图以一种特别的方式管理复杂性。今天，我们使用过程，方法和工具保持复杂性可控。但是明天呢？我们当前的方法能胜任这些任务吗？

31.3.2 开放世界的软件

诸如情境智能[⊖]，上下文敏感应用以及普适计算等所有这些概念都将集中在把基于软件的系统集成到比个人电脑、移动计算设备或任何其他数字设备广泛得多的环境中。这些关于计算的不远未来的不同观点都集中提出了“开放世界软件”的概念，这种软件被设计成“通过自组织结构和自适应行为”适应不断变化的环境。

开放世界软件包括情境智能，上下文敏感应用和普适计算。

为了有助于说明软件工程师在可预见的未来所要面对的挑战，考虑一下情境智能（ambient intelligence, amI）这个概念。Ducatel [Duc01]定义情境智能如下：“人们被嵌入到很多种物体上的智能的和直观的界面所包围。情境智能环境能够以一种无缝、无障碍方式识别和响应不同个体[在工作时]的存在”。

让我们来想象这样的情形，在不远的将来，amI已经变得无处不在。你恰好购买了一款个人通讯设备（称做P-com，一种小型移动设备）并且已经用了几周创建[⊖]你的“图像”——你的日常安排中的每件事：待办事项、地址簿、医疗记录、商务相关的信息、旅行证件、愿望清单（你正在找的一些东西，如某一本书、一瓶不太好找的酒、有关玻璃吹制的本地课程）以及“Digital-Me”（D-Me）——一种数码产品，能在一定程度上把你详细地介绍给别人(可以和你一起移动的一类MySpace 或 FaceBook)。P-com中包含个人识别码，称做“关键的钥匙”——一个多功能的个人识别码，可以在很广范围的amI设备和系统中提供访问和查询功能。

显而易见,重要的隐私和安全问题要在其中起作用。一个“信任管理系统”[Duc01] 将是amI的必要组成部分，并且还要有管理权限，以保证和网络、卫生、娱乐、金融、就业以及个人系统之间的沟通。

新的具有amI能力的系统将不断地增加到网络中，每个系统都提供有用的能力并且要求能访问你的P-com 。因此，必须设计好P-com软件，使得它能适应由于一些新的amI系统上线而出现的需求。有很多方式可以做到，但是底线是：P-com 软件必须是灵活的、健壮的，而这些特点是传统软件不能相比的。

31.3.3 意外需求

在软件项目开始的时候，有一个同样适用于每个利益相关者的老生常谈的话题：“你不知道自己不知道什么”。这意味着，客户很少定义“稳定的”需求。也意味着，软件工程师也不可能总是预见哪里含糊不清以及矛盾的所在。需求变更本来就不是什么新鲜事。

因为意外需求已成为现实，你的组织应该考虑采用增量过程模型。

由于系统变得越来越复杂，因此，即使是对陈述全面需求的初步尝试也是注定要失败的。总体目标的陈述是可能的，中间目标的描述也可以实现，但稳定的需求——却不可能！随着每个参与复杂系统的工程设计和建设的人对

⊖ 关于情境智能有价值的并且相当详细的介绍可以在www.emergingcommunication.com/volume6.html找到。可在www.ambientintelligence.org/获得更多信息。

⊖ 所有与P- com的交互作用通过连续语音识别命令和声明出现，其准确性已达到99%。

系统本身、系统所处的环境和与系统交互的用户具有了更多的了解，需求就出现了。

这一现实暗示了很多软件工程趋势。首先，过程模型的设计必须包含变更，并采取敏捷哲学的基本原理（第3章）。其次，必须明智地使用产生工程模型的方法（例如，需求模型和设计模型），因为这些模型随着更多知识的获取将不断地改变。最后，支持过程和方法的工具必须很容易地适应和改变。

但对于意外需求还有另一个方面。今天绝大多数的软件开发都假设软件系统和外部环境之间存在的边界是稳定的。边界可能会改变，但这种改变会以一种受控制的方式进行，也就是把软件作为一个普通的软件维护周期的一部分进行调整。这个假设正在开始改变。开放世界软件（31.2.2节）要求计算机系统“动态地适应和响应变更，即使这些变更是意料之外的”[Bar06]。

就其性质而言，意外需求带来转变。我们如何控制广泛使用的应用程序或系统在其生命周期的演变，并且这对我们设计软件的方式有什么影响呢？

随着变更数量的不断增多，意料之外的副作用出现的可能性也增大了。这应该是为具有意外需求的复杂系统考虑规范的原因。软件工程界必须开发一些方法，帮助软件开发团队预测变更对整个系统的影响，从而减轻意料之外的副作用。今天，我们做到这一点的能力是极为有限的。

31.3.4 人才结构

随着基于软件的系统变得越来越复杂，随着全球异地团队之间的通信和协作变得越来越普遍，随着意外需求（产生变更流）变得越来越规范，软件工程团队的真正性质可能发生变化。作为各种复杂系统的一部分，每个软件团队必须拥有创新的人才和技术技能，整个过程必须允许这些人才孤岛的工作结果能够有效地合并。

Alexandra Weber Morales [Mor05] 提出了人才组合的“软件梦之队”。大脑是总设计师，他能够掌控利益相关者的需求，并将这些需求映射到一个既可扩展又可实现的技术框架中。数据Grrl是一个数据库和数据结构大师，他“通过对谓词逻辑和集合论的深刻理解，将其关连到关系模型，大量使用行和列”。Blocker 是一位技术领导者（经理），他允许团队自由地工作，不受其他团队的影响，同时确保合作的进行。Hacker是一位出色的程序员，他在家工作，可同时有效地使用模式和语言。Gatherer“灵巧地发现系统的需求，具有……人类学的洞察力”，并清晰准确地表达出来。

31.3.5 软件构造块

“对数字技术正确的艺术反应是将其作为永恒人类一切事物的新窗口，并热情、智慧、无畏和幸福地使用它。”
——Ralph Lombreglia

促进软件工程哲学的所有人都重视重用的必要性——源代码、面向对象的类、构件、模式以及商业成品软件（COTS）的重用。虽然软件工程界已经取得进展，试图捕获过去的知识以及重复使用可靠的解决方案，但是当今制造的软件很大一部分仍然是“从零开始”。部分原因是利益相关者和软件工程人员对“独特解决方案”的持续渴望。

在硬件界，数码产品的原始设备制造商（Original Equipment Manufacturers, OEM）使用几乎完全由芯片厂商生产的特定应用的标准产品（Application-Specific Standard Products, ASSP）。这种“商业硬件”提供了实现任何数码产品（从移动电话到HD - DVD播放器）所必需的构造块。越来越多的OEM厂商使用“商业软件”——专为独特的应用领域[例如，VoIP设备]设计的软件构造块。Michael Ward [war07] 评论道：

使用软件构件的一个优点是OEM可依提升软件所提供的功能，而无需具备特定功能的内部开发经验，也无需将开发人员的时间投入到实施和验证这些构件。其他优点包括只需要获取和部署系统需要的一组特定功能的能力，以及将这些构件集成到已有体系结构中的能力。

然而，软件构件方法也有缺点，需要一定的工作量才能将单个构件集成到整个产品中去。如果构件来自多家供应商，每家都拥有自己的接口方法，这种集成挑战可能更加复杂。至于使用其他来源的构件，管理不同供应商所需要的工作量会增加，而且在不同来源构件间的相互作用方面遇到问题的风险会更大。

除了作为商业软件的构件包，越来越倾向于采用软件平台解决方案，这种解决方案“将相关功能的集合合并在一起，这些功能通常在一个集成的软件框架中提供”[War07]。一个软件平台不受一起工作的开发基本功能的OEM的影响，而且允许OEM将软件工作集中在有别于其产品的那些特性上。

31.3.6 对“价值”认识的转变

在20世纪最后的25年，在讨论软件时，商务人士通常问的问题是“为什么它值那么多钱？”这个问题今天已经很少有人再问了，取而代之的是“为什么我不能快点得到它（软件和（或）基于软件的产品）？”。

在考虑计算机软件时，对价值的最时髦认识已经从商业价值（价格和收益率）向客户价值转变，包括：交付的速度、功能的丰富性以及总体产品质量。

31.3.7 开源

谁拥有你或你的组织所使用的软件？日益增加的答案是“每个人”。对“开源”运动的描述如下[OSO08]：“开源是一种软件开发方法，运用了分布的同行审查的力量和过程的透明性。开放源码的承诺是更好的质量，更高的可靠性，更大的灵活性，更低的成本，而且也是垄断性厂商的坟墓”。开源这个术语应用到计算机软件，意味着对于软件工程的工作产品（模型、源代码、测试套件）都是公开的，任何感兴趣并得到允许的人都可以（有控制地）对其评审和扩展。

一个开源“团队”可能有很多全职“梦之队”成员（31.3.4节），但是，随着对应用系统的兴趣的增强和减弱，软件工作的人数会随之扩大或减少。在开源团队的力量来自不断的同行评审和设计（代码）重构，其结果是朝着最优的解决方案缓慢进展。

如果您有进一步的兴趣，Weber [Web05] 提供了一个有价值的介绍，Feller和他的同事们[Fel07] 已编辑了一本全面、客观的文集，其中考虑了与开源相关的利益和问题。

INFO

技术观察

许多新兴技术很可能对演变中的基于计算机系统的类型产生重大影响。这些技术增加了软件工程师所面临的挑战。以下技术值得注意：

网格计算——这项技术（今日可用的）创建了一个网络，把网络上的来自于每台机器的数十亿未使用的CPU组织在一起，并可完成极其复杂的计算工作，而不需要专用的超级计算机完成。对于现实生活中的一个超过450万台计算机的实例，可以访问http://setiathome.berkeley.edu/。

开放世界计算——“这是情境的、含蓄的、无形的和适应性的。它可以在任何时候在网络设备的嵌入式环境中提供不显眼的连接和服务”[McC05]。

微商务（Microcommerce）——电子商务的一个新分支，以很少的收费访问或购买各种形式的知识产权。苹果iTunes是一种广泛使用的例子。

认知机（Cognitive machines）——机器人领域中的“圣杯（holygrail）”是要开发感知其环境的机器，这些机器能“拾起线索，应对不断变化的情况，并与人自然地互动”[PCM03]，认知机仍然处于发展的初期阶段，但潜力是巨大的。

OLED显示器——OLED“使用基于碳的设计分子，当电流通过它时可以发出光线。将很多分子结合在一起，就得到了质量惊人的超薄显示屏——没有背光耗用动力的需要”[PCM03]。结果——可以卷起来或折叠，展示到一个曲面或以其他方式适应特定环境的超薄显示器。

RFID——无线射频识别把开放世界计算带到了工业基地和消费品行业。从牙膏管到汽车引擎，任何物体当它通过供应链移动到最终目的地时都可以被识别出来。

Web2.0——一种广泛使用的Web服务，可以将Web更好地集成到商务和个人计算中。

关注技术有关方面更进一步的讨论，可以在录像和出版物的独特组合上获取，请访问消费者电子协会（Consumer Electronics Association）网站 www.ce.org/Press/CEA_Pubs/135.asp。

31.4 技术方向

“但是，好处究竟是什么呢？”——IBM 高级计算机系统部的工程师就芯片发表的评论

我们总是认为，软件工程似乎比它的实际变化更快。本节将介绍一个新的“广泛宣传的”技术（它可能是一个新的过程、一个独特的方法或者一个令人兴奋的工具），专家们认为“一切”都会改变。但是，软件工程不只是技术——软件工程是有关人以及交流他们的需求、不断创新、使这些需求成为现实的能力。每当有人参与，变化就会时断时续地慢慢发生。只有达到了一个“转折点”[Gla02]，跨越整个软件工程界的技术阶梯和基础广泛的变化才会真正发生。

在这一节中，我将讨论在过程、方法和工具几个方面的一些趋势，这些可能会对未来十年的软件工程产生影响。它们会导致转折点出现吗？我们将拭目以待。

31.4.1 过程趋势

可以说，所有的商业、组织和31.3节中讨论的文化趋势增强了对过程的需要。但是第30章讨论的框架提供了通向未来的路线图吗？过程框架将会演进，能在纪律性和创造性之间找到更好的平衡吗？软件过程会适应那些采购软件的、构建软件的和使用软件的利益相关者的不同要求吗？能否提供一个同时对三组人减少风险的手段？

这些以及其他许多问题依然悬而未决。然而，一些趋势已经开始显露出来。Conradi 和 Fuggetta [Con02] 提出6篇“关于如何加强和更好地应用SPI框架的论文”。他们开始用下面的陈述来讨论：

软件中介的目标是客观和理性地选择最好的承包商。软件公司的目标是在激烈竞争的市场中生存和发展。最终用户的目标是以一个可接受的价格获得能够在恰当的时候解决恰当问题的软件产品。我们不能期望相同的SPI方法和相应的工作都适合所有这些观点。

在下面的段落中，对Conradi和Fuggetta [Con02]提出的论点进行了修改，给出了未来十年可能的过程趋势。

未来十年最可能的过程趋势是什么？

1. 随着SPI框架的发展，他们将强调“关注目标定位和产品创新的策略”[Con02]。在软件开发的快节奏的世界里，长期的SPI策略很少在动态的商业环境中生存下来。太多的变化来得太快。这意味着，稳定的、按部就班的SPI路线图可能被强调短期有产品定位目标的框架所取代。如果经过了一系列增量产品的发布（通过网络交付给最终用户），对新的软件产品线的需求就会出现，软件组织可能认识到有必要改进管理变更的能力。与变更管理相关的过程改进必须以某种方式与产品的发布周期相协调，这种方式在改进变更管理的同时本身不被破坏。

2. 因为软件工程师对一个过程中哪里存在弱点是比较清楚的，过程改变一般应按他们的要求驱动，并且应该自底向上进行。Conradi和Fuggetta [Con02] 建议未来的SPI活动应该“以一种简单的、聚焦的积分卡开始，而不是大规模的评估”。通过严密地关注SPI工作以及自底向上的工作，业务人员将能看到早期实质性的变化——在做出的软件工程工作中产生了真正的差别。

3. 自动的软件过程技术（software process technology, SPT）将不去做全局性过程管理（整个软件过程的广泛支持），而是侧重于软件开发过程中从自动化中最能受益的那些方面。没有人会反对工具和自动化，但在很多情况下，SPT 并没有履行承诺(见31.2节)。要想最有效，应侧重于普适性活动（第2章）——软件过程中最稳定的元素。

4. 更强调SPI活动的投资收益率。在第30章，已经学习了投资收益率（ROI）定义如下：

$$\text{ROI} = \left[\frac{\Sigma(\text{benefits}) - \Sigma(\text{costs})}{\Sigma(\text{costs})}\right] \times 100\%$$

迄今为止，软件组织一直在努力以量化方式明确界定“收益”。可以说 [Con02] “我们因此需要一个标准化的市场-价值模型来解释软件改进，如在Cocomo II (见第26章) 中所使用的”。

5. 随着时间的推移，软件界可能逐渐认识到，如其他技术性更强的学科一样，社会学和人类学的专业知识对于成功的SPI有很多或更多的事情可做。除此之外，SPI改变着组织的文化，文化的改变涉及个人和群体。Conradi和Fuggetta [Con02] 正确地谈到：“软件开发人员是知识工作者。他们往往对高层就如何做工作或改变过程的指挥反应消极”。可以通过考察群体社会学学到更多的知识，更好地了解引进变革的有效方法。

6. 学习的新模式可能有助于向更有效的软件过程转变。在这种情况下，“学习”是指从成功和犯错误中学习。收集度量 (第23章和25章) 的软件组织可以让自己明白过程因素如何影响最终产品的质量。

31.4.2 巨大的挑战

有一个趋势是不可否认的——毋庸置疑，随着时间的推移，基于软件的系统将变得越来越大，越来越复杂。正是这些大型、复杂的系统工程，无论交付平台还是应用领域，都构成了对软件工程师的“巨大挑战”[Bro06]。Manfred Broy [Bro06] 建议软件工程师通过创建新的方法理解系统模型，并利用这些模型作为基础构建高质量的下一代软件以满足“复杂软件系统开发的严峻挑战”。

随着软件工程界开发新的模型驱动方法（在这节的后面讨论）来表示系统需求和设计，下面的一些特征[Bro06]值得关注：

对于未来的应用系统，分析师和设计师必须考虑的系统特性是什么？

- 多功能性——随着数码产品已经进入第二代和第三代，它们已开始提供一套内容丰富、有时不相关的功能。移动电话，一度被认为是通信设备，现在已用于拍照，提供日历，导航旅行以及音乐播放器。如果将来实现了开放世界的接口，这些移动设备将在未来许多年使用。因此，Broy [Bro06] 指出，“工程师必须详细描述所交付功能的背景，而最重要的是，必须确定系统的不同特性之间可能有害的相互作用”。
- 反应性和及时性——数码产品日益与现实世界互动，必须对外部刺激做出及时反应。他们必须与各种各样的传感器接口，必须在时间框架内做出反应，这些特性都适合于手头的工作任务。必须开发新方法：(1) 帮助软件工程师预测各种反应特性的时间；(2) 以一种使功能较少依赖于机器并且更便携的方式实现这些特征。

- *用户交互的新方式*——键盘和鼠标在PC机环境下工作得很好，但是软件的开放世界趋势意味着必须对新的交互方式进行建模和实现。这些新方法是否使用触控界面、语音识别或直接智能接口[⊖]，数码产品的新一代软件必须为这些新的人机界面建模。
- *复杂的体系结构*——超过2000个功能的豪华汽车已经由软件控制，包含在复杂的硬件架构中，包括多处理器、先进的总线结构、执行器、传感器、日趋复杂的人机界面和许多安全相关的组件。更复杂的系统就在眼前，这对软件设计师提出了重大的挑战。
- *异构的分布式系统*——任何现代嵌入式系统的实时组件都可以经过内部总线、无线网络或者因特网（或者所有这三者）连接起来。
- *关键性*——在几乎所有的商业关键系统和大多数安全关键系统中，软件已经成为其中的中枢组件。然而，软件工程界仅仅开始应用软件安全的最基本原理。
- *维护可变性*——在一个数码产品中，软件的寿命很少能超过3～5年，但是，飞机中复杂的航空系统至少可以使用20年。汽车软件介于两者之间。这些对设计有影响吗？

Broy [Bro06] 认为，只有在软件工程领域开发出更有效的分布式和协作软件工程的理念、更好的需求工程方法、更健壮的模型驱动的开发方法以及更好的软件工具，这些软件特点和其他软件特点才能成为可控制的。接下来的几节中将简要地探讨这些领域。

31.4.3 协同开发

显而易见，但无论如何都应该说：软件工程是一门信息技术学科。从任何软件项目开始，每个利益相关者都必须共享信息——有关基本的商业目标和目的，有关特定系统的需求，有关体系结构的设计问题，有关软件构造的几乎每个方面。

协作涉及信息的及时传播以及沟通和决策的有效过程。

今天，软件工程师可以跨越时区和国界进行合作，他们中的每个人都要共享信息。这同样适用于开源项目，其中数百或数千名软件开发人员一起工作建立一个开源应用程序。同样，信息必须传播，以保证开放的协作成为可能。

未来十年的挑战是开发有助于协作的方法和工具。今天，我们继续努力使协作更为容易。Eugene Kim [Lim04] 谈到：

> 考虑一个基本的协作任务：文档共享。很多应用系统（商业上的或开源的）都声称要解决文档共享问题，然而，文件共享的主要方法是来回发邮件。这是网络的计算等价物。 如果声称解决这些问题的工具好用，我们为什么不用它们呢？

我们看看其他基本领域中类似的问题。我可以拿着一页纸去世界的任何地方参加会议，可以肯定，人们会去阅读、在上面做标记、传递它并归档。对于电子文件我不能说同样的事情。我不能在网页上做标注，也不能对电子邮件和Word文档使用相同的归档系统，至少还没有哪种方式能够保证在自己和别人的机器上使用应用系统进行互操作。为什么不能呢？

……为了在协作空间上产生实际影响，工具不仅要很好，而且必须能互操作。

但是，缺少全面的协作工具只是那些必须协同开发软件的软件工程师所面临挑战的一部分。

今天，很大比例[⊜]的IT项目在国际上是外包的，在未来十年内，这一数字将大幅增长。毫不奇怪，Bhat和他的同事 [Bha06] 主张需求工程是外包项目的关键活动。他们确定了一些成功因素，可以使协作的努力获得成功：

⊖ 直接智能接口的简单讨论可以在http://en.wikipedia.org/wiki/Brain-computer_interface 找到，在http://au.gamespot.com/news/6166959.html 上描述了一个商业例子。

⊜ 目前典型的大公司的IT预算大约20%都用于外包，这个比例每年还在增加。（源自：www.logicacmg.com/page/400002849.）

- *共享目标*——项目目标必须清楚地阐明，所有的利益相关者必须了解这些目标，并同意目标的意图。
- *共享文化*——文化差异应该明确界定，应该研究教育方法（将有助于减轻那些差异）和沟通方法（将有利于知识转移）。
- *共享过程*——在某些方面，过程是协作项目的框架，提供了一个统一评估进展和方向的手段，并为所有团队成员引入了共同的技术“语言”。
- *共享责任*——每个团队成员都要认识到需求工程的重要性，并且努力给系统提供最可能的定义。

把这些成功的因素结合起来就会形成彼此“信任”——建立一个全球性团队，可以依靠互相不同的群体来完成分配给他们的工作。

31.4.4 需求工程

在第5章到第7章中已经介绍了基本的需求工程活动——需求引导、细化、协商、规格说明和确认。这些活动的成功和失败对于整个软件工程过程的成功和失败有着重要的影响。然而，已将需求工程（requirements engineering, RE）比作“试图将软管夹放在凝胶物周围”[Gon04]。正如本书很多地方都提到的，软件需求有不断变更的趋势，并且随着开放世界系统的出现，意外需求（和近乎持续的变更）可能会变得普遍。

今天，绝大多数“非形式化的”需求工程方法以创建用户场景（例如，用例）开始。更形式化的方法创建一个或更多的需求模型，并以此作为设计的基础。形式化方法能使软件工程师通过使用可验证的数学符号来表示需求。当需求稳定时，所有这一切都可以合理地工作，但对于动态需求或意外需求问题就不容易解决了。

有许多不同的需求工程研究方向，包括：从翻译文本描述到更结构化的表示（例如，分析类）的自然语言处理；为了结构化和理解软件需求需要更多地依赖数据库；当执行需求工程任务时，使用RE模式来描述典型的问题和解决方案以及面向目标的需求工程。然而，在行业层面，RE活动还停留在非正式和令人不可思议的基础阶段。为了改善需求定义的方式，当执行RE时，软件工程界会实施3种不同的子过程 [Gli07]：(1) 改进知识获取和知识共享，使得更完整地理解应用领域的限制和利益相关者的需求；(2) 在定义需求时，更加强调迭代；(3) 更有效的沟通和协调工具，使所有利益相关者进行有效的合作。

KEY POINT
新的RE子过程包括：(1) 改进的知识获取；(2) 更多的迭代；(3) 更有效的交流和协作工具。

对于前面段落中声明的RE 子过程，当这些子过程已恰当地集成到一个软件工程不断改进的方法中时才会成功。随着基于模式的问题求解和基于构件的解决方案开始支配许多应用领域， RE必须适应敏捷性（快速增量交付）和由敏捷性所导致的内在的意外需求。许多软件工程过程模型的并行性质意味着RE将与设计和构造活动相结合。因此，一个静态的“软件规格说明”的概念开始消失，被“价值驱动的需求”[Som05] 取代，这是响应利益相关者提出早期软件增量要求交付的性能和功能的必然产物。

31.4.5 模型驱动的软件开发

几乎在软件工程过程的每一步，软件工程师都使用抽象手段。随着设计的开始，体系结构级和构件级的抽象得到表达和评估。然后它们必须被翻译成一种编程语言表示，即把设计（较高级别的抽象）转换成具有特定的计算环境（较低级别的抽象）的可操作的系统。模型驱动的软件开发[⊖]以某种方式将特定领域的建模语言与转换引擎和产生器相结合，有助于对较高层次

⊖ 术语模型驱动工程（model-driven engineering, MDE）也会使用。

KEY POINT
模型驱动方法定位于所有软件开发者面临的挑战——怎样在一个比代码级抽象程度更高的级别上表示软件。

抽象的表示，并将其转换成较低层次的表示[Sch06]。

特定领域的建模语言（DSML）)描述“应用程序结构、行为和特定应用领域的需求”，并使用元模型进行描述。元模型“定义在领域中概念之间的关系，准确地描述关键语义以及与这些领域概念相关的约束”[Sch06]。DSML与通用的建模语言如UML(附录1)的主要区别是DSML协调应用领域的设计概念，因此可以以一种有效率的方式表示设计要素之间的关系和约束。

31.4.6 后现代设计

在一篇有趣的“后现代时代”软件设计的文章中，Philippe Kruchten [Kru05]给出了以下观点：

计算机科学还没有达到可以解释一切的大境界，宏观上我们没有发现像物理或其他工程学科发挥作用的基本规律一样的软件基本规律。我们仍然生活在互联网泡沫破灭和Y2K数字末日苦涩的回味中。因此，在这个后现代的时代，这里发生的一切事情似乎都有点重要，却又不是真正的重要，软件设计未来的方向是什么呢?

任何企图了解软件设计趋势的尝试都是要建立设计的边界。需求工程在哪里停止，设计从哪里开始呢？在哪里停止设计，从哪里开始生成代码呢？对这些问题的回答并不容易，因为问题可能第一次提出。即使需求模型应侧重于“做什么”，而不是“如何做”，每个分析师做了一点设计，几乎所有的设计师都做了一些分析。同样，随着软件构件的设计更接近算法细节，设计师开始以更接近代码级的抽象表示构件。

后现代设计将继续强调软件体系结构（第9章）的重要性。设计者必须说明体系结构的问题，做出解决问题的决定，然后明确地定义假设、约束和说明相关问题，这些工作决定了整个软件。但是，有可以在其中描述问题、定义体系结构的框架存在吗？面向方面的软件开发（第2章）或模型驱动的软件开发（31.4.4节）可能会成为未来几年重要的设计方法，但这么说还为时过早。可能在基于构件的开发（第10章）上取得突破，基于构件的开发可能导致强调现有组件装配的设计理念。如果说过去是序曲，那么极有可能会出现许多“新”的设计方法，但很少会超过趋势周期(31.2节)中的“幻灭”。

31.4.7 测试驱动的开发

需求驱动设计，设计成为构造的基础。简单的软件工程实际运作相当起作用，对于创建软件体系结构是不可或缺的。然而，当考虑构件级的设计和构造时，巧妙的变更可以提供极大的好处。

在测试驱动的开发（Test-Driven Development, TDD）中，将软件构件的需求作为生成一组测试用例的基础，以测试用例检查接口，并试图找到数据结构中和构件提供的功能中的错误。TDD 并不是一种真正的新技术，而是一个趋势，它强调生成源代码之前测试用例的设计[⊖]。

KEY POINT
TDD是一种趋势，强调在创建源代码之前设计测试用例。

TDD过程遵循简单的流程，如图31-3所示。在第一小部分的代码生成之前，软件工程师引入测试以备检测代码（尽量发现代码的错误）。然后写下的代码要顺应测试。如果通过了，再创建新的测试来检测要开发的下一段代码。此过程持续下去，直到各构件完全编好代码，并且所有的测试都没有错误地执行了。然而，如果任何测试成功地发现了错误，现有的代码重构（修正），所有和错误相关的测试都重新执行。这种重复的流程继续进行，直到没有测试需要创建，这意味着该构件符合定义的所有要求。

⊖ 回忆一下极限编程（第3章），作为敏捷过程模型的一部分强调这种方法。

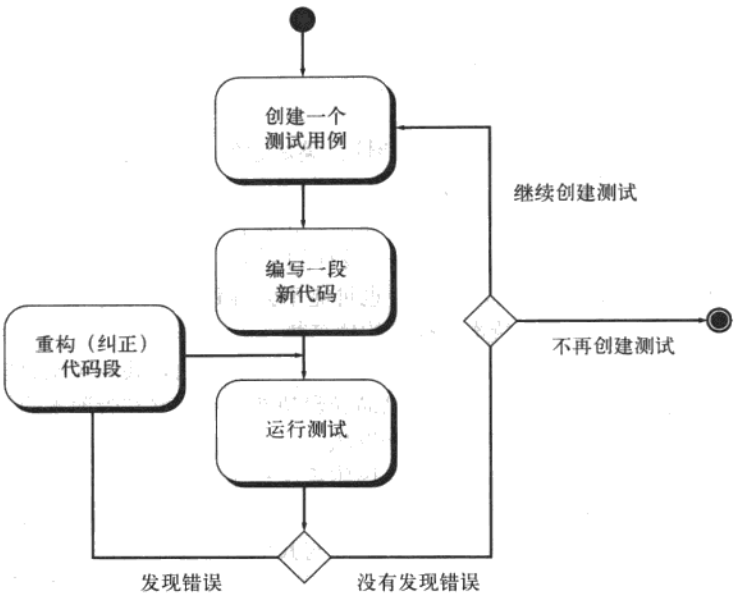


图31-3 测试驱动的开发过程流

在TDD中，代码是以非常小的增量（一次一个子函数）开发的，直到存在测试要检查时，才编写代码。应该注意到，每个迭代都会产生一个或多个新的测试被添加到回归测试集中，每次变更都要运行回归测试集。这样做是为了确保新代码没有在旧代码中产生导致错误的副作用。

在TDD中，测试导出详细的构件设计和得到的源代码。这些测试的结果使得要立即修改构件的设计（通过代码），更重要的是，得到的构件（当完成时）已经以独立的方式通过了验证。如果对TDD有进一步的兴趣，请参见[Bec04b]或[Ast04]。

31.5 相关工具的趋势

每年都有数以百计的工业级软件工程工具被采用。其中大多数是由工具制造商提供的，他们声称这些工具会改进项目管理、或需求分析、或设计建模、或代码生成、或测试、或变更管理、或本书讨论的任何软件工程活动、行为和任务。其他工具已经作为开源产品被开发出来。大多数开源工具集中在特别强调构建活动（特别是代码生成）的“编程”活动上。还有一些工具来自于大学和政府实验室的研究工作。虽然他们在非常有限的应用问题中很吸引人，但是大多数不具备广泛的行业应用。

在行业层面，最全面的工具包形成了软件工程环境（software engineering environments, SEE）[⊖]，它围绕一个中心数据库（存储库）集成了特定的工具集。当作为一个整体考虑时，SEE通过软件过程整合了信息，并且对许多大型、复杂的基于软件的系统所需的协作提供帮助。但当前的环境是不容易扩展的（很难集成COTS工具，它不是软件包的一部分），而且往往是通用的（即它们不是针对具体的应用领域）。新技术解决方案（例如，模型驱动软件开发）的推出和切实可行的支持新技术SEE的可用性之间也存在一个很长的时间间隔。

软件工具的未来趋势将遵循两种不同的途径——一种是以人为本的途径，对应了31.3节中

⊖ 也使用术语集成开发环境（integrated development environment, IDE）。

讨论的“软趋势”；另一种是以技术为中心的途径，随着新技术被引进和采用而关注新技术(31.4节)。以下两小节中，将简要地研究每种途径。

31.5.1 顺应软趋势的工具

31.3节中讨论的软趋势——需要管理复杂性、满足意外需求、建立包含变更的过程模型、协调具有不断变化的人才组合的全球团队等——提出一个新的时代，在这个时代，支持利益相关者协作的工具将变得和支持技术的工具同样重要。但是什么样的工具集支持这些软趋势呢？

在这方面研究的一个例子是GENESIS——设计的一个广义的、开源的环境，支持协作的软件工程工作[Ave04]。GENESIS环境可能会也可能不会得到广泛的使用，但它的基本元素是协作SEE这个方向上的代表，将支持本章提到的软趋势。

协作SEE “支持分布式开发团队的软件工程师的合作和沟通，共同实施建模、控制以及测量软件开发和维护过程。此外，它包括一个制品管理功能，可存储和管理不同的团队在他们的工作过程中生产的软件制品（工作产品）”[Bol02]。

图31-4说明了一个协作SEE的体系结构。该体系结构基于GENESIS环境[Ave04]，它是集成在一个通用的Web客户端的子系统，通过基于服务器的构件来实现，对所有的客户端提供支持。每个开发组织都有自己的客户端子系统，可以与其他的客户端通信。参考图31-4，资源管理子系统管理人力资源如何分配给不同的项目或子项目；工作产品管理系统“负责创建、修改、删除”索引，搜索和存储所有的软件工程工作产品[Ave04]；工作流管理子系统负责协调定义、实例化以及软件过程活动、行动和任务的实现；事件引擎“收集在软件过程中发生的事件（例如，工作产品的成功评审，构件的单元测试的完成）”，并通知其他相关人员；通信系统支持分散的团队之间同步和异步的沟通。

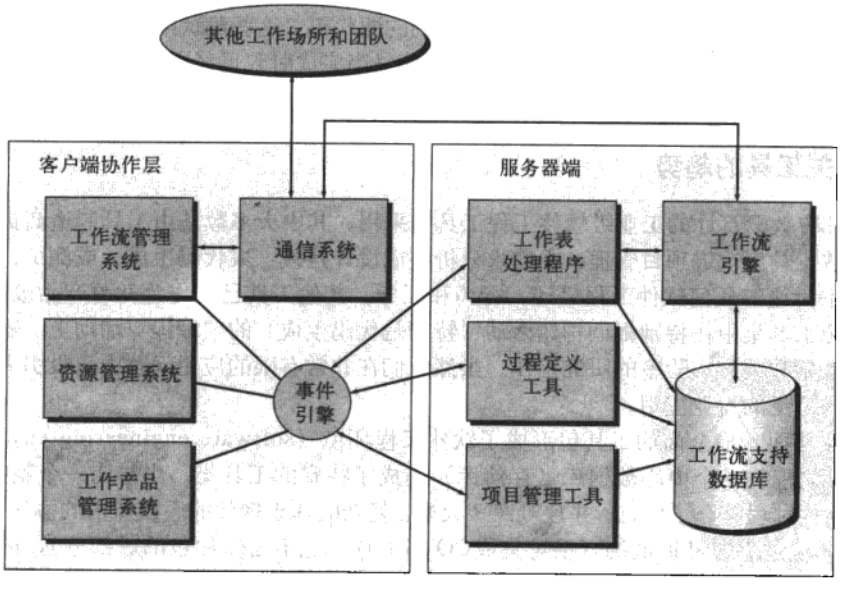


图31-4 协同SEE体系结构 （来源：摘自[Ave04]）

在服务器端，四个构件共享一个工作流支持数据库。这些构件实现以下功能：

- 过程定义——一个工具集，能使团队定义新的过程活动、行动或任务，并定义支配这些元素交互的规则以及生产的工作产品。

- *项目管理*——一个工具集，允许团队建立项目计划，并与其他团队或项目协调该计划。
- *工作流引擎*——“与事件引擎交互，传播与其他工作场所实施的协作过程有关的事件”[Ave04]。
- *工作表处理程序*——与服务器端数据库进行交互，给软件工程师提供当前任务相关的信息，或从目前正在执行的工作中产生的任何未来的任务。

尽管协作SEE的体系结构可能和本节讨论的不同，将会出现基本功能元素（管理系统和构件）将能达到分布式软件工程项目所需要的协调程度。

31.5.2 涉及技术趋势的工具

当利益相关者作为一个团队工作时，敏捷软件工程（第3章）是可以实现的。因此，即使各自在本地开发软件，面向协同SEE（31.5.1节）的趋势也是有益的。哪些技术工具可以补充系统和构件，使得能够更好地协作？

技术工具的主要趋势之一是建立工具集，支持突出体系结构驱动设计的模型驱动开发（31.4.4节）。Oren Novotny [Nov04]建议，模型应成为软件工程的中心焦点，而不是源代码：

可以使用UML 建立平台独立的模型，然后进行不同程度的改造，最终形成一个特定平台的源代码。那么，不言而喻，是这个模型，而不是文件，应该成为新的输出单元。模型在不同的抽象层次上会有很多不同的视图。在最高层次，可以确定分析中平台独立的构件；在最低层次，特定平台的实现可以分解到代码形式的一组类上。

Novotny 认为，新一代工具将与存储库联合工作，在所有需要的抽象层次上创建模型，建立不同模型之间的关系，从其中一个抽象层次的模型转换到另一个层次的模型（例如，将设计模型转换成源代码），管理变更和版本，针对软件模型协调质量控制和质量保证行动。

除了完整的软件工程环境，关注从收集需求、到设计/代码重构、再到测试的任何事情的单点解决方案工具将继续发展，并提供更多的功能。在有些情况下，与通用工具相比，针对特定应用领域的工具将提供更多的好处。

31.6 小结

对软件工程技术有影响的趋势经常来自于商业、团体、市场和文化等领域。这些“软趋势”能指导研究方向以及作为研究结果的技术。

当引入了一项新技术，它就会经历一个生命周期，新技术并不总是被广泛采用，即使最初的期望很高。任何软件工程技术获得广泛应用的程度与解决软趋势和硬趋势问题的能力相连的。

软趋势——对连接和协作、全球项目、知识转让、新兴经济的影响以及人类文化本身的影响力等不断增长的需要，导致了跨越管理复杂性和意外需求的一系列挑战，需要调整分散在各地的软件开发团队不断变化的人才结构。

硬趋势——技术变化的步伐在日益加快——决定软趋势，并影响软件结构、过程范围和表示过程框架特色的方式。协同开发、需求工程的新形式、基于模型的开发和测试驱动的开发以及后现代设计会改变方法的前景。对于不断增长的沟通与协作需要，工具环境将做出回应，同时集成了特定领域点的解决方案，这些可能会改变目前软件工程任务的性质。

习题与思考题

31.1 获得Malcolm Gladwell的畅销书《The Tipping Point》（通过Google图书搜索），讨论如何将他的理论应用于新软件工程技术的采用。

31.2 为什么开放世界软件对传统的软件工程方法提出了挑战?

31.3 回顾Gartner Group关于新兴技术的趋势周期。选择一个众所周知的技术产品，并给出了它的简要的发展史，说明它如何沿着曲线发展的。选择另一个著名的技术产品，它不遵循趋势周期的发展。

31.4 什么是“软趋势”?

31.5 当你面对着一个极其复杂的问题，这需要一个漫长的解决方案。你如何去处理这种复杂性并给出巧妙的解决方案?

31.6 什么是“应急需求”，它们为什么对软件工程师提出了挑战?

31.7 选择一个开放源码的开发工作（除了Linux），并提交其演变和相对成功的简要发展史。

31.8 描述你认为软件过程在未来十年将会怎样改变?

31.9 你在洛杉矶并且和全球软件工程团队一起工作。你和在伦敦、孟买、中国香港和悉尼的同事必须为一个大系统编辑245页的需求规格说明。必须在3天内完成初稿。描述一套理想的在线工具，可以使你们能够有效合作。

31.10 用自己的话描述模型驱动的软件开发及测试驱动的开发。

推荐读物与阅读信息

讨论软件和计算前进道路的书籍涉及大量的技术、科学、经济、政治和社会问题。Kurweil （《The Singularity Is Near》, Penguin Books, 2005）提出了一个引人注目的观点，到本世纪中叶，对世界的看法将发生真正深刻的变化。Sterling（《Tomorrow Now》, Random House, 2002）提示我们：真正的进展是很少有秩序和有效率的。Teich （《Technology and the Future》, Wadworth, 2002）对社会有影响的技术以及如何改变文化状态的技术给出了详细的论述。Naisbitt, Philis 和 Naisbitt （《High Tech/High Tough》, Nicholas Brealey, 2001）指出，我们许多人都“陶醉于”高科技，并且“在高科技时代的巨大讽刺是，我们已经成为了设想给我们自由的设备的奴隶”。Zey（《The Future Factor》, McGraw-Hill, 2000）讨论了5种力量，将对本世纪人类的命运形成重大影响。Negroponte的书（《Being Digital》, Alfred A. Knopf, 1995）是20世纪90年代中期的畅销书，并继续提供对计算及其全面影响的独到见解。

随着软件成为我们生活中几乎每个方面的一部分，“网络空间伦理”已经演变为一个重要的讨论话题。Spinello （《Cyberethics: Morality and Law in Cyberspace》, Jones & Bartlett Publishers, 2002）、Halbert 和Ingulli （《Cyberethics》, South-Western College Publishers, 2001）以及Baird和他的同事（《Cyberethics: Social and Moral Issue in the Computer Age》, Prometheus Books, 2000）的书详细考虑了这个主题。美国政府以CD光盘的形式发布了一份长篇报告（《21st Century Guide to Cybercrime》, Progressive Management, 2003），考虑了计算机犯罪、知识产权问题以及国家基础设施保护中心(the National Infrastructure Protection Center, NIPC)的所有方面。

软件相关技术和软件工程的未来方向的大量信息源可以在互联网上找到。最新的有关软件工程的未来发展趋势的WWW参考资料可在SEPA站点www.mhhe.com/engcs/compsci/ pressman/ professional/ olc/ser.htm上找到。

结 束 语

要点浏览

概念：当我们即将结束这个关于软件工程的相对漫长的旅程时，该阐明一些观点并提出一些结论性意见了。
人员：与本书作者一样的人。当你即将结束这本漫长的、富有挑战性的书的时候，很高兴能以一种有意义的方式做个总结。
重要性：记住我们曾经处的位置和考虑我们要达到的目标总是有价值的。
步骤：我将会考虑，我们所处的位置和要解决的一些核心问题以及未来的一些发展方向。
工作产品：帮助你理解全局的讨论。
质量保证措施：立竿见影是很难的。只有经过若干年以后，你或我才能够分辨出本书中讨论的这些软件工程的概念、原理、方法和技术是否已经帮助你成为了更好的软件工程师。

关键概念
道德规范
未来
信息范围
知识
人员
软件回顾

在前面的31章中，已经探讨了软件工程过程，包括管理规程和技术方法、基本概念和原理、专门技术、面向人员的活动和适合于自动化的任务、用纸和笔表示的符号以及软件工具。本人认为，对敏捷度和质量的测量、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，我从未允诺软件工程是万能的。

当我们走向这个新世纪的第二个十年之时，软件和系统技术对于构造计算机系统的软件专业人员和公司仍是一种挑战。虽然Max Hopper [Hop90]怀着对21世纪的展望写下了下面这些话的，但却准确地描述了当前的情形：

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……细想起来对技术“是多么无奈”，公司不得不拼命地追赶，才能有立足之地。

软件工程技术方面的变化确实“快速和不可遏制”，但同时进展又经常很慢。在决定采用一种新过程、方法或一种新工具的时候，为了理解其应用，需要进行必要的培训，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。

在这个领域有一件事我琢磨了多年，就是软件工程从业人员是“赶时髦”的人。前面的道路将充满令人兴奋的新技术（最新的潮流），这些技术还从来没有真正使用过（尽管进行了大量宣传）。这将形成更合适的技术，以某种方式修改前进道路的方向和宽度。

其中的少数内容我在第31章已经讨论了。

在这最后一章，我将从更哲学的角度扩展我的观点并进一步考虑，在软件工程实践领域我们今天所处的位置以及我们的目标。

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。当从整体上考虑的时候，软件工程的工作产品产生了任何个人、商业或政府都能获取的最重要的日用品——信息。

在第31章中，我简单地讨论了开放世界的计算——情境智能、环境敏感应用以及普适计算——这个方向将从根本上改变我们对电脑的看法，我们处理它们所做的事情（和它们为我们所做的事情）以及我们对信息的看法会成为一种指导、一种商品甚至一种必需品。我还注意到支持开放世界计算的软件将会给软件工程师提出新的、激动人心的挑战。但更为重要的是，未来普遍存在的计算机软件将给整个社会提出更加急剧的挑战。每当技术具有广泛的影响——这种影响可以挽救或危害生命、建立或摧毁多个行业以及告知政府领导人或误导他们，那就必须要“小心处理”了。

32.2 人员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“随着程序规模的增长，必须为此程序投入的人员数量也要增加”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。

“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

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，就像个人之间的交流一样。更糟糕的是，交流（在个人间或项目团队间）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”。

如果软件工程界有效地解决了交流的困境，软件工程师的未来之路必定会涉及个人之间及项目团队之间相互交流方式的根本性改变。在31.1节中，我们讨论了协同工作环境可以在团队交流方面提供引人注目的改进。

最后，交流是知识的传递。知识获取（和传递）的方式正在出现深刻改变。随着搜索引擎更加广泛地使用，以及Web 2.0应用提供了更好的协同能力，世界上最大的关于软件工程方面的研究论文、报告、指南、评论及参考资料的图书馆变得更易于访问和使用。

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。

32.3 表示信息的新模式

“做好明天工作的最好的前提就是做好今天的工作。”——Elbert Hubbard

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，这才是永久的目标，然而，术语上的改变反映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”、“信息”和“内容”这些词反复地出现，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——事实集合，它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“信息”的4种视图。

目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[⊖]。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，即应该如何将来自一系列不同源（它们之间可能没有明显的联系）的信息联系起来，并将它们组合在一起，为我们提供某些独特的收益[⊜]。

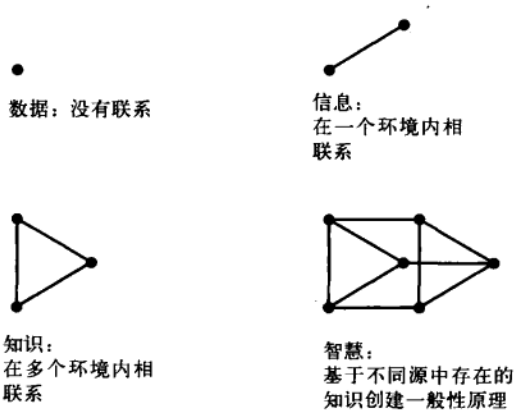


图32-1 “信息”谱系

“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

为了说明从数据到知识的发展过程，以人口普查数据为例：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。可以组合所有这些信息，然后表示为知识——在21世纪的早期，美国的教育系统将面临巨大的压力，这种压力将持续几十年之久。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、更廉价的学习模式。

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沿着信息谱的发展趋势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。

32.4 远景

在32.3节中，我提出未来的道路将通向建立“处理知识”的系统。但是，未来的通用计算

⊖ 数据挖掘和数据仓库技术的快速发展反映了这种增长趋势。

⊜ 语义Web（Web 2.0）允许这种mashups的创建，它提供了一种容易的获取知识的机制

和特殊的基于软件的系统可能会导致相当深刻的事件发生。

在一本有吸引力的涉及计算技术的书（一定适合每个人阅读）中，Ray Kurzweil[Kur05]建议，我们已经达到了这样一个时代——“技术变化的步伐是如此之快，影响是如此之深，以至于人们的生活将不可逆转地跟着改变”。Kurzweil[⊖]给出了令人信服的论证，我们目前正处在指数增长曲线的“转折点”，在未来20年中，计算能力将得到极大提升。随着纳米技术、遗传学和机器人技术方面的进展，我们可能在本世纪中期的某个时候，人类（如同我们知道他们今天的样子）和机器之间的区别变得很模糊——这个时候人类正以某种既可怕（对一些人来说）又壮观（对另外一些人来说）的方式加速进化。

Kurzweil认为，在21世纪30年代的某个时候，计算能力和必要的软件将足以对人脑的每个方面进行建模——所有的物理连接、模拟过程和化学覆盖。这种情况发生时，人类将获得“强人工智能”，因此，机器确实可以思考（使用今天世界的常规用法）。但是，存在一个根本的差别。人脑的处理是相当复杂的并且和外部的信息源仅仅以一种松散的方式连接。即使与今天的计算技术相比，人脑的计算也是很慢的。当人类的大脑能完全仿真时，机器的“思想”将以比人脑快数千倍的速度更迅速地与相应的海量信息相连接（作为一个简单的例子，想一想今天的网络）。其结果是……如此荒诞，最好是留给Kurzweil来叙述。

值得注意的是，不是每个人都相信Kurzweil描述的未来，这是一件好事。在著名的题为《The Future Doesn't Need Us》的论文里，Bill Joy[Joy00]（Sun公司的创始人之一）认为“机器人技术、遗传工程、纳米技术正在威胁着人类这个濒临灭绝的物种”。他对技术的不良预测与Kurzweil所预言的理想国未来形成了对立。双方都应认真考虑，作为一个软件工程师在确定人类的长远未来时应起什么样的领导作用。

32.5 软件工程师的责任

软件工程已经发展成为令人尊敬的、全球性的行业。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]规定：

软件工程师应履行其承诺，使软件的分析、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。依照他们对公众健康、安全和利益的承诺，软件工程师应当坚持以下八项原则：

WebRef

关于ACM/IEEE的职业道德规范的完整讨论可在seeri.etsu.edu/codes/default.shtm找到。

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赞成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。

⊖ 值得注意的是，Kurzweil不是一个磨坊中的科幻作家或没正事的未来主义者。他是一位严肃的技术主义者，“已经成为光学符号识别（optical character recognition，OCR）、文字语音合成、语音识别技术以及电子键盘设备等领域的先驱”（来自Wikipedia）。

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：

- 决不将数据据为己有。
- 决不散布或出售在软件项目中工作时所获得的私有信息。
- 决不恶意毁坏或修改别人的程序、文件或数据。
- 决不侵犯个人、小组或组织的隐私。
- 决不闯入一个系统胡闹或牟取利益。
- 决不制造或传播计算机病毒。
- 决不使用计算技术去助长偏见或制造麻烦。

在过去的十年中，软件产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策方面，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。

32.6 结束语

自本书第一版的编写，至今30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的情形，当时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市场。幸运的是，McGraw-Hill出版社决定尝试一下[⊖]，其余的如他们所说的那样已经成为了历史。

在过去的30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）成熟起来。

计算机软件开发的工程方法目前仍然是传统的知识，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被广泛采用呢？

我认为答案是由于技术转变和伴随而来的文化变化的困难，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）。为了使转变更容易，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，实践者更好地接受和管理者的支持，以及大量的教育。

你不可能同意本书中描述的每个方法。某些技术和观点是相互矛盾的；为了在不同的软件

⊖ 实际上，这要归功于Peter Freeman和Eric Munson，他们使McGraw-Hill相信，这是值得一试的。销售超过了100万册，公平地说，他们做了一个好决定。

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。

当我们更深入地走进21世纪之时，软件依然是世界上最重要的产品和最重要的产业。它的影响和重要性已经经历了一段很长的路。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挑战。让我们期待迎接挑战的人们——软件工程师——他们有更多的智慧去开发改善人类条件的系统。

UML[⊖]简介

关键概念
活动图
类图
通信图
依赖
部署图
泛化
交互框架
多重性
对象约束语言
OCL
顺序图
状态图
构造型
泳道
用例图

统一建模语言（Unified Modeling Language，UML）是“绘制软件蓝图的标准化语言。UML用来可视化、描述、构造和文档化软件密集系统的人工制品”[Boo05]。换句话说，就像建筑设计师需要为建筑公司设计蓝图一样，软件设计师也需要创建UML图帮助软件开发者开发软件。如果了解UML的词汇（图示元素和它们的含义），就可以很轻松地理解和描述一个系统，并向他人解释该系统的设计。

Grady Booch、Jim Rumbaugh和Ivar Jackson在20世纪90年代中期开发出了UML语言，引起了广大软件开发人员的强烈反响。UML融合了大量当时软件产业所使用的具有竞争力的建模表示方法。1997年，UML 1.0被提交给对象管理组织（Object Management Group，OMG）——一个致力于维护计算机产业使用规范的非盈利性组织。同年，UML1.0修订成UML1.1后不久就被对象管理组织采用了。目前的标准是UML2.0[⊜]，也是ISO标准。因为这个标准很新，很多旧的资料，如[Gam95]，没有使用UML符号。

UML2.0提供了13种不同的图供软件建模使用。在本附录中，将仅讨论类图、部署图、用例图、顺序图、通信图、活动图和状态图。本书使用了这些图。

应该注意，UML图有很多可选功能。UML语言提供了这些选项（有时候是隐藏的），使得软件工程师能表达系统的所有重要方面。同时，还可以灵活地隐藏图中那些与建模无关的部分，避免无关的细节将图弄得杂乱。因此，一种特殊功能的省略并不意味着该功能的缺失，它可能意味着该功能被隐藏了。在本附录中，没有介绍UML图的所有功能，而是重点介绍标准选项，尤其是本书用到的那些选项。

类图

为了对类建模（包括类的属性、操作和关系以及和其他类的联系[⊜]），UML提供了类图，类图提供了系统的静态或结构视图。它并不显示图中类的对象之间通信的动态特性。

类图的主要元素是方框，方框是一些用来描述类和接口的图符。每个方框都被水平地划分为多个部分。顶层部分包含类的名字，中间部分列出了类的属性。属性是指类的对象所知道并能一直提供的内容。属性通常被实现为类的域，但并不需要如此。属性可以是一些数值，这些数值是由类从它的实例变量中计算出来的，或是从组成它的其他对象中得到的。比如，对象可以时刻知道当前时间并在你查询时反馈给你。因此，它适合列出当前时间作为该类对象的属性。然而，对象不可能将该时间存储在它的实例变量中。如果这样的话，它需要一直不断地更新该字段。作为替代，对象更可能在你请求时间时计算出当前时间（例如，通过查询其他类的对象）。

⊖ 本附录由Dale Skrien提供并改编自他的著作《An Introduction to Object-Oriented Design and Design Patterns in Java》（McGraw-Hill，2008）。所用内容已经授权。

⊜ OMG已经于2009年2月发布了UML2.2。可在http://www.omg.org/找到。——译者注

⊜ 如果你不熟悉面向对象的概念，可参考附录2中给出的简要介绍。

类图的第三部分包含类的操作或行为。操作是指类的对象所能做的事情，通常实现为类的方法。

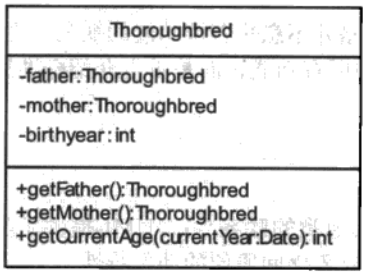
图A1-1介绍了一个简单例子，用Thoroughbred类对优良种马建模。它有3个属性——mother、father和birthyear，还有3个操作：getCurrentAge()、getFather()和getMother()。图中也有一些隐藏的属性和操作没有显示。

每个属性都有名字、类型和可见性级别。类型和可见性都是可选的。类型放在名字后面，并用冒号进行分隔。可见性由前面的－、#、~或＋指定，分别代表私有、受保护、包或公有可见性。在图A1-1中，所有属性都是私有的，由前面的减号（－）指出。你也可以通过下划线表示属性是否为静态属性或类属性。可以用可见性级别、带名字和类型的参数以及返回类型来表示每个操作。

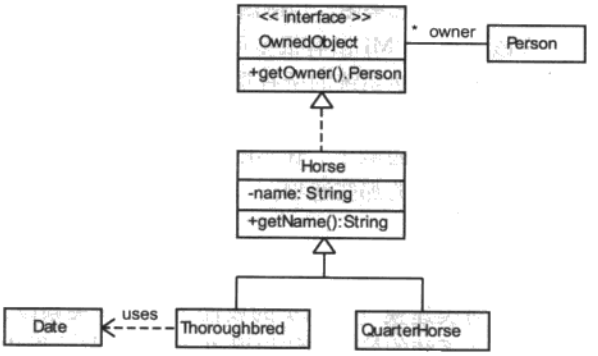
在类图中，名字设成斜体表示抽象的类或方法。例如，图A1-2中的Horse类就是抽象类。通过在名字的上方添加短语“《interface》”（称为构造型）来指定接口。请看图A1-2中的OwnedObject接口。也可以用空心圆来表示接口。

值得一提的是，表示类的图符可以有其他可选的部分。例如，处于类方框底层的第四部分可以列出类的责任。在产生属性和操作之前，CRC卡片上列出的责任如能添加在UML图中类方框的第4部分，则这一部分在CRC卡片（第6章）向类图转换时特别有用。本附录中的所有图均未显示第4部分。

类图也可以表示类之间的关系。类和它的子类之间用实线加上空心的三角箭头连接。箭头的方向是从子类指向父类。在UML中，这样的关系称为泛化。比如，在图A1-2中，类Thoroughbred和类QuarterHorse显示为Horse抽象类的子类。带虚线的箭头表示接口的实现。在UML中，这样的关系称为实现。例如，图A1-2中，Horse类实现了OwnedObject接口。



图A1-1 Thoroughbred类的类图



图A1-2 有关马的类图

两个类之间的关联是指它们之间存在着结构上的关系。关联用实线表示，有很多可选部分。可以在它的每一端都加上标签，指定关联中每个类的角色。例如，在图A1-2中，OwnedObject和Person之间存在关联，Person在此关联中扮演所有者（owner）的角色。关联线的一端或两端有箭头表示可导航性。关联线的每一端也会有多重数值显示。可导航性和多重性将在这一节的后面部分详细讲述。关联也可以通过循环连接它本身，这样的关联表明同一类所创建的不同对象之间可以相互连接。

一端带箭头的关联表明这是一个单向导航。箭头表明从第一个类你可以很容易访问关联方向所指的第二个类，但是从第二个类你却不能很容易地访问第一个类。关于这种现象的另外一种解释是，第一个类可以察觉到第二个类，但是第二个类的对象将不能直接察觉到第一个类。

没有箭头的关联通常表明它是一个双向的关联，这就是图A1-2中所指的。也有可能是仅仅意味着可导航性并不重要，所以省略了。

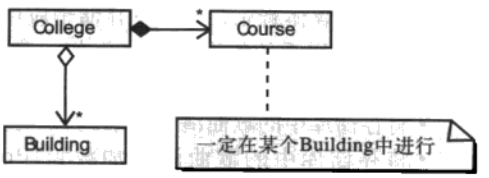
应当注意到，一个类的某个属性和类（其属性所属的类型是类）的关联是一样的。就是说，想表达类中的叫做“name”的String类型的特性，需要如同属性一样地显示它，如图A1-2中的Horse类。或者，也可以构建一个从Horse类到String类的单向关联，在此关联中，String类的角色是“name”。对于原始数据类型，用属性的办法会更好些，而当特性类在设计中起重要作用时，用关联的方法通常比较好一些，在那种情况下，有那种类型的类方框更有价值。

依赖关系表示类之间的另一种连接，由一条虚线（可选的段末箭头和可选的标签）表示。一个类依赖于另一个类，则改变另一个类也需要改变这个类。从一个类到另一个类的关联就自动表明了一种依赖性。如果类之间已存在关联则不需要虚线。然而，对于短暂的关系（即一个类不需要同另一个类维持长时间的连接，但确实偶尔会用到另一个类），我们应该从第一个类画一条虚线到第二个类。例如，在图A1-2中，当Thoroughbred类的getCurrentAge()方法被调用时，它需要使用Date类，所以依赖性被标识为“uses”。

关联一端的多重性是指类关联于其他类的对象的数量。多重性由非负整数或整数范围描述。通过“0..1”描述的多重性是指在关联的一端存在0或1个对象。例如，世界上的每个人要么有社会保险号，要么没有，因此，多重性0..1就可以在类图中的Person类和SocialSecurityNumber类之间的关联中使用。由“1..*”描述的多重性是指一个或更多，由“0..*” 或只是“*”描述的多重性是指0个或更多。在图A1-2中，与Person类关联的OwnedObject端的多重性使用了“*”，因为Person可以拥有0个或更多的对象。

如果关联的一端有比1大的多重性，那么该端涉及的对象将可能被存储于收集中，如集合或有序列表。在UML图中，也可以包括收集类本身，但是，由于关联的多重性，这样的类通常被省略并假定存在那里。

聚合是一种特殊的关联，通过图符一端的空心钻石表示。它表明一种“整体/部分”关系，箭头所指的类是关联的菱形端的一“部分”。组合表明聚合对部分的强所有权。在组合中，部分随着所有者而生存或消亡，因为它们在独立于所有者的软件系统中没有作用。请看图A1-3中关于聚合和组合的例子。



图A1-3 College、Course和Buiding之间的关系

College有一个包含Building对象的聚合，这表示建筑构成了学院。学院也有一个包含课程的集合。如果学院倒闭了，建筑仍将存在（假定该学院不是物理上消失）并可用于其他事情，但是Course对象在学院之外则毫无用处，它是由学院提供的。如果学院作为一个事务实体不复存在了，那么Course对象也就没用了，将不复存在。

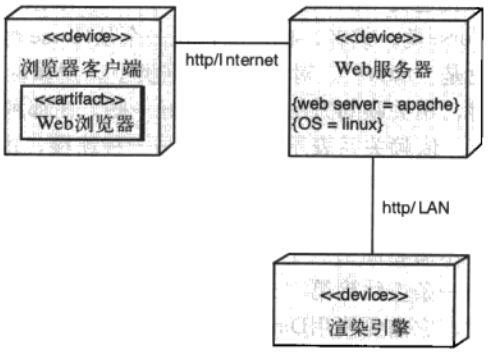
类图中另一个共同元素是注释，由具有狗耳角的方框表示，通过虚线连至其他图符。它可有任意内容（文字和图形），类似于编程语言中的注释。它包括类的作用的解释或是该类的所有对象应遵守的约束。如果内容是约束，内容外面将括着大括号。注意图A1-3中Course类所受到的约束。

部署图

UML部署图关注软件系统的结构，用于显示软件系统在硬件平台和运行环境中的物理分布。例如，设想你正在开发一个基于Web的图形显示包，包的用户将使用他们的Web浏览器访

问你的网站，得到渲染信息。你的网站应根据用户的描述渲染图像并将其返回给用户。因为图像渲染需要大量的计算，所以你决定将渲染从Web服务器中分离出来，放在一个单独的平台上。这样，系统涉及3个硬件设备：Web客户端（运行浏览器的用户电脑），Web服务器所在的电脑和渲染引擎所在的电脑。

图A1-4显示了此包的部署图。在此图中，硬件部件放在标有“《device》”的方框中，硬件部件之间的通信路径用带有可选标签的线表示。在图A1-4中，路径是用连接设备的通信协议和网络类型标记的。



图A1-4 部署图

部署图中的每个节点也用设备的细节注释。例如，在图A1-4中，浏览器客户端被描述为由Web浏览器软件构成的人造制品。人造制品是指包含在设备上所运行软件的文件。你也可以描述标记值，就像图A1-4中的Web服务器节点一样。这些值定义了服务器所采用的Web服务器厂家和服务器所使用的操作系统。

部署图也可以显示运行环境节点，将运行环境节点绘制为包含标签“《execution environment》”的方框。这些节点表示可以运行其他软件的系统，如操作系统。

用例图

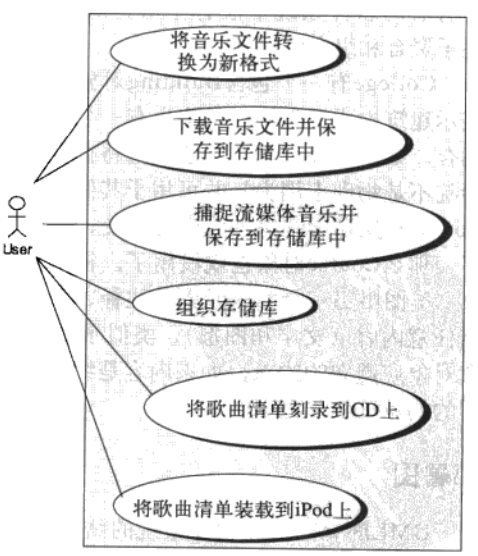
用例（第5、6章）和UML用例图可帮助你从用户的角度决定软件的功能和特点。为了让读者了解用例和用例图是如何工作的，以下为数码音乐文件管理软件创建一些用例和用例图。这个软件类似于Apple的iTunes软件。软件可能要做的一些事情包括：

- 下载MP3音乐文件，并将其存储于应用系统的存储库中。
- 捕捉流媒体音乐，并将其存储于应用系统的存储库中。
- 管理应用系统的存储库（例如，删除歌曲或将其添加到播放列表中）。
- 将存储库中的歌曲清单刻录到CD上。
- 将存储库中的歌曲清单加载到IPod或MP3播放器。
- 将一首歌曲从MP3格式转换为AAC格式，反之亦然。

这并不是一个全面的列表，但它已足够使你理解用例和用例图的作用。

用例通过定义实现明确目标所需的步骤描述了用户和系统之间的交互（例如，将歌曲清单刻录到CD上）。一系列步骤的变动描述了各种不同的场景（例如，如果歌曲清单中的所有歌曲不适合放在一张CD上怎么办？）。

UML用例图是所有用例和用例之间关系的视图。它提供了系统功能的整体图示。数码音乐应用系统的用例图如图A1-5所示。

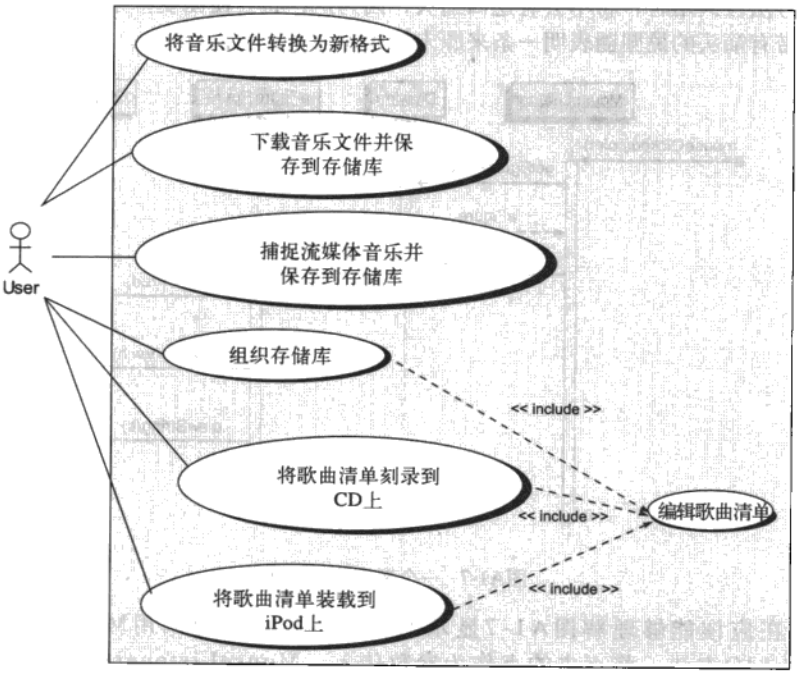


图A1-5 音乐系统的用例图

在这个图中，木棍小人表示和其他类型的用户（或其他交互元素）相关联的参与者（第5章）。复杂的系统通常不止一个参与者。例如，自动售货机应用系统可有3个参与者，表示客户、维修人员和装货人员。

在用例图中，用例用椭圆形显示。角色通过线连接到所执行的用例上。注意，图中并不包括用例的细节，用例的详细信息需要单独存储。另外，还需要注意，可将用例放在矩形框中，而不能将角色放在矩形框中。矩形框表示的是系统的边界，而角色在系统之外。

系统的一些用例会互相关联，例如，将歌曲清单刻录到CD和将歌曲清单存储到iPod具有相似的步骤。这两种情况，用户都会首先建立一个空表，然后再将存储库中的歌曲添加进来。为避免用例中的重复，需建立新用例来表示重复活动，然后让其他用例包含这个新用例，作为其他用例的一个步骤。在用例图中，如图A1-6所示，这种包含关系用标有《include》的虚线箭头来连接用例和被包含的用例。



图A1-6 具有包含用例的用例图

由于用例图显示了所有的用例，因此，用例图有助于确保覆盖系统的所有功能。在数码音乐系统中，确实需要更多的用例，如播放库中歌曲的用例。但是请记住，用例在软件开发过程中最有价值的地方是对每个用例的文字说明，而不是总体用例图[Fow04b]。通过用例的说明，才能对所开发系统的目标形成清晰的理解。

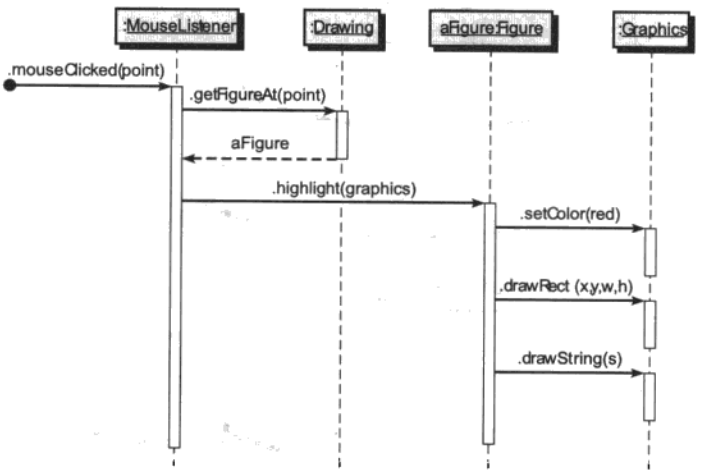
顺序图

与类图和部署图不同，类图和部署图显示系统构件的静态结构，而顺序图显示任务执行过程中对象之间的动态通信。它显示了完成任务的对象之间消息发出的先后顺序。顺序图可以显示某个用例或软件系统的某个场景中存在的交互。

在图A1-7中，可以看到一个画图程序的顺序图。该图显示了在一幅图中点击一个图形时

高亮度显示所涉及的步骤。图顶端行中的每个方框通常对应一个对象，虽然方框也有可能模拟其他东西，比如类。如果方框表示对象（如我们所有例子中的情况），在方框内，可选地规定冒号前的对象类型。也可在冒号之前写上类名，如图A1-7中第三个方框所示。在每个方框下面都有一条被称为对象生命线的虚线。顺序图中的垂直轴代表着时间，随着时间逐渐增加，线条逐渐向下移动。

顺序图使用从调用者到被调用者的水平箭头来表示方法调用，箭头上标有方法名称，可选部分包括参数、参数的类型和返回类型。如图A1-7所示，MouseListener调用Drawing的getFigureAt()方法。当对象在执行方法时（即在堆栈里有活动帧时），在对象的生命线下面可选择性地显示一条空白的条，称为活动条。在图A1-7中，对于所有方法的调用都绘制了活动条。图也可以通过虚线箭头和可选标记选择性地显示方法调用的返回结果。在图A1-7中，getFigureAt()方法的返回结果由返回对象的名字标记。普遍的做法，如图A1-7所示，当一个无返回结果的方法被调用时，将不会有返回箭头，因为这样将会使图变得杂乱，而且不会起任何重要作用。带有箭头的黑圆圈表明一条来源未知或无关的发现消息。



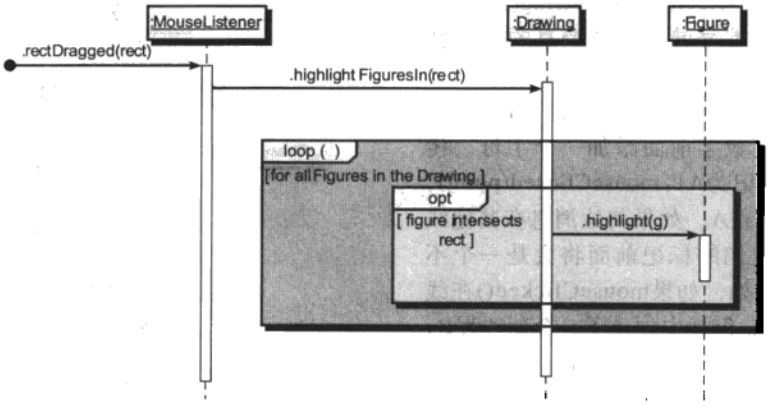
图A1-7 一个顺序图的例子

读者现在应该能够理解图A1-7显示的任务了。未知源调用MouseListener中的mouseClicked()方法，将点击的点作为参数传入。MouseListener然后调用Drawing的getFigureAt()方法，并得到返回的一个Figure对象。接着MouseListener调用Figure的highlight()方法，传递一个Graphics对象作为参数。作为响应，Figure调用Graphics对象的3个方法用红色画出图形。

图A1-7比较简单，不包括附加条件和循环。若需逻辑控制结构，最好为每种情况都绘制一张单独的顺序图。即当消息根据条件可有两条不同的路径时，则需要绘制两张分开的顺序图，为每种可能性绘制一张。

若想要在一张顺序图上包括循环、条件和其他控制结构，可以使用交互框。交互框是矩形，包围图的一部分，并用其所表示的控制结构的类型做标记。图A1-8描述了这种情况，高亮显示给定矩形中所有图形所涉及的过程。将消息rectDragged发送给MouseListener，MouseListener通知绘图部分高亮显示矩形中的所有图形，具体做法是调用Drawing对象的highlightFiguresIn()方法，并传递矩形作为参数。此方法要对Drawing对象中的所有Figure对象

进行循环，如果Figure和矩形相交，则需要Figure高亮显示自身。在方括号内的短语称为守卫，守卫是布尔条件，如果交互框中的动作要继续，则守卫必须为真。



图A1-8 带两个交互框的顺序图

顺序图还有很多其他特点。例如：

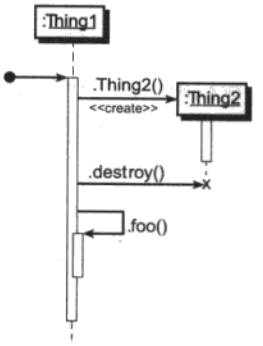
1. 可清楚地分辨出同步和异步消息。同步消息用实体箭头表示，而异步消息用棒形箭头（stick arrowhead）显示。

2. 可用箭头表示对象向它本身发送消息。所用的箭头需从该对象发出，然后折向下，再指回对象本身。

3. 通过绘制指向对象方框的带有适当标记（例如，带有《create》标签）的箭头来表示对象的创建。这种情况下，方框将出现在比行动开始时已存在对象对应的方框低一些的位置。

4. 也可通过对象生命线末端的大写X显示对象的销毁。其他对象可以销毁一个对象，在这种情况下，一个箭头从其他对象指向X。X通常表示该对象已不再有用，因此可以准备进行垃圾回收。

最后3个特点都显示在图A1-9所示的顺序图中。



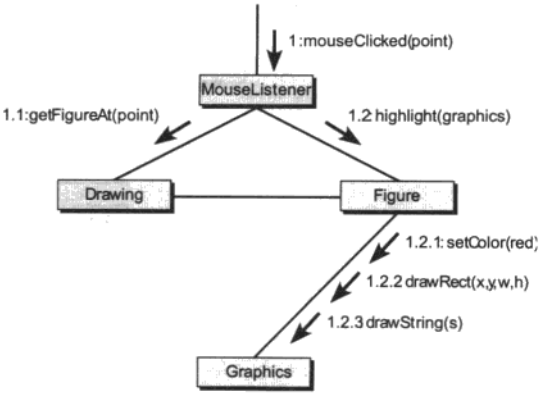
图A1-9 顺序图中的创建、销毁和循环

通信图

UML通信图（在UML1.x中称为“协作图”）提供了通信时间顺序的另一种表达形式，但是强调对象和类之间的关系，而不是时间顺序。图A1-10所示的通信图与图A1-7中的顺序图显示的是相同的动作。

在通信图中相互作用的对象由矩形表示。对象之间的关联由矩形之间连接的线表示。对于图中启动消息传递序列的对象，通常有传入箭头指向该对象。箭头由数字和消息名称标记。如果传入消息被标记为数字1并且它使接收对象调用其他对象的其他消息，则这些消息将沿关联线从发送者到接收者的箭头表示，并按照它们被调用的顺序，标以数字1.1、1.2等。如果这些消息又调用了其他消息，另外的十进制小数点和数字将被添加到标记这些消息的数字中，说明消息传递的进一步嵌套。

在图A1-10中，可以看到mouseClicked消息调用getFigureAt()方法，然后又调用highlight()方法。highlight()消息调用其他3个消息：setColor()、drawRect()和drawstring()。每个标签上的数码编号就像每条消息的时序性质那样显示嵌套。



图A1-10 UML通信图

有很多可选特点被添加进箭头标记中。例如，可以在数字前面添加一个字母。传入箭头可以标记为A1：mouseClicked(point)，指明可执行线程A。如果其他消息在其他线程中执行，它们的标记前面将会是一个不同的字母。例如，如果mouseClicked()在线程A中可执行，但是它创建了一个新线程B，并且调用该线程中的highlight()，那么从MouseListener到Figure的箭头将被标记为1.B2：highlight(graphics)。

如果对显示对象之间的关系及对象间传递的消息感兴趣，通信图可能是比顺序图更好的选择。如果对命令传递的时间顺序感兴趣，那么顺序图也许更好。

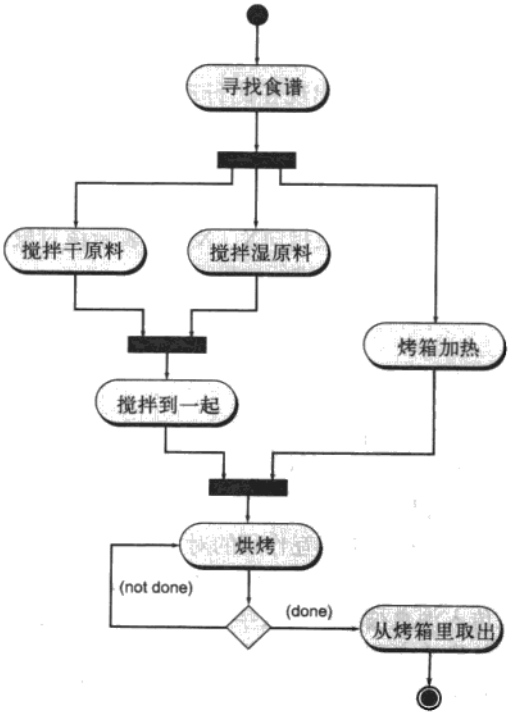
活动图

UML活动图通过系统所执行动作之间的控制流来描述系统或部分系统的动态行为。它类似于流程图，但活动图可以显示并行流。

活动图的主要构件是动作节点，由圆角矩形表示，对应于软件系统执行的任务。从一个动作节点到另一个动作节点的箭头表示控制流。也就是说，在两个动作节点之间的箭头意味着第一个动作完成后，第二个动作才开始。实心黑点表示活动开始的初始节点。被黑圆圈包围的黑点表示活动结束的最终节点。

分叉表示活动分为两个或更多并行活动，被绘制成水平黑条，并带有一个指向自己的箭头、两个或更多出来的箭头。每个发出的箭头都代表一个控制流，此控制流可与其他发出的箭头所对应的控制流并行执行。这些并行活动可在一台电脑上使用不同的线程执行，甚至使用不同的电脑执行。

图A1-11显示了一个烤蛋糕的样本活动图。第一步是寻找食谱。一旦找到了食谱，就可以称量和搅拌干原料和湿原料，并且可以预热烤箱。干原料的搅拌可与湿原料



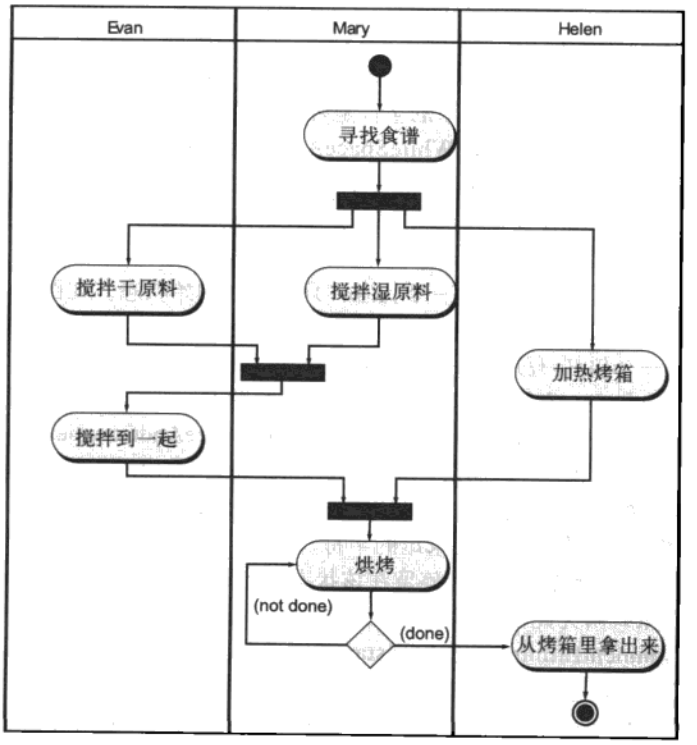
图A1-11 烤蛋糕的UML活动图

的搅拌以及烤箱的预热并行进行。

合并是同步并行控制流的一种方式。它由水平黑条表示，黑条带有两个或更多的流入箭头和一个流出箭头。直至所有流入箭头所表示的控制流均已完成，流出箭头表示的控制流才能执行。在图A1-11中，在将干湿原料搅拌在一起的动作之前有一个合并。该合并表明在两种原料搅拌在一起之前，必须先各自搅拌。图中第二个合并表明，在能够烘制蛋糕之前，所有的原料必须搅拌到一起，并且烤箱必须处于合适的温度。

判定节点与依赖条件的控制流分支相对应。这样的节点用菱形表示，带有一个流入箭头和两个或更多的流出箭头。每个流出箭头由守卫标记（括号里的条件）。控制流沿守卫为真的流出箭头方向进行。确保条件覆盖所有分支是合理的，这样使得每次到达决策节点时，其中有一个条件为真。图A1-11显示了判定节点沿着烘制蛋糕过程的进行状况。如果蛋糕烤好了，将会从烤箱中拿出来。否则，还需再烤一段时间。

图A1-11中的活动图并未说明的是谁或什么做了每次的动作。通常没有必要准确地区分动作的主体。但是，如果想表明动作在参与者中是如何划分的，则可以采用泳道活动图，如图A1-12所示。正如名字所暗示的那样，将图划成条或“道”就形成了泳道，每道都对应一个参与者。一个泳道内的所有动作都由该泳道对应的参与者完成。在图A1-12中，Evan负责搅拌干原料，然后将干原料与湿原料搅拌到一起，Helen负责加热烤箱和取出蛋糕，Mary负责其余的所有事情。



图A1-12 增加了泳道的蛋糕烘制活动图

状态图

对象在特殊点的即时动作取决于当时的状态，也就是当时变量的值。作为一个小例子，想

想带有布尔实例变量的对象。当实施操作时，如果变量为真，则对象可以做某件事，如果变量为假，则对象可以做其他事情。

UML状态图模拟了对象的状态，图中执行的动作取决于这些对象的状态和状态之间的转换。

作为例子，考虑Java编译器的一部分状态图。输入编译器的是文本文件，文本可以认为是长字符串。编译器每次读一个字符，并且根据读入的字符决定程序结构。读字符过程中忽略了“空白”字符（例如，空格、制表、换行和返回字符）以及注释中的字符。

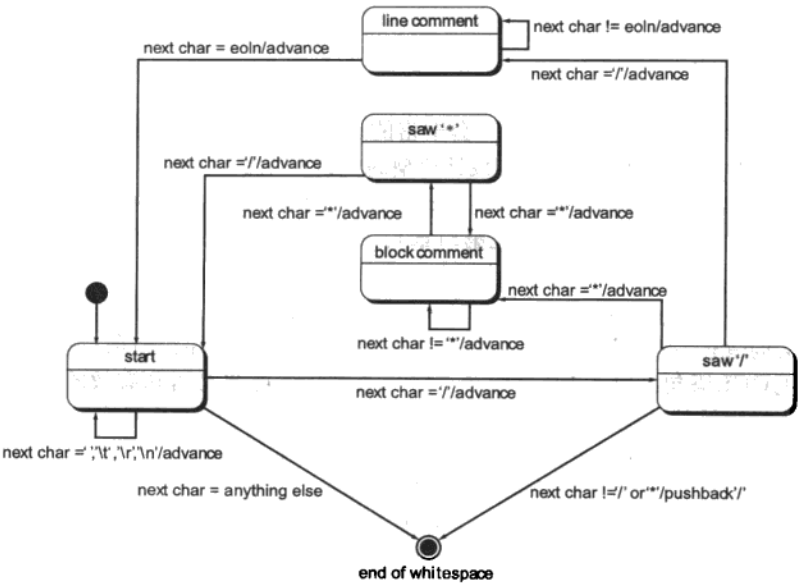
假定编译器将越过空白和注释字符的任务授权给WhiteSpaceAndCommentEliminator类。也就是说，该对象的任务是读取输入字符，直到所有空白和注释字符都被读出，这时它会将控制权转交给编译器去读取和处理非空白和非注释的字符串。思考一下WhiteSpaceAndCommentEliminator对象是怎样读取字符的，并判断下一字符是否为空白或注释。对象可通过测试下一字符的“”、“\t”、“\n”和“\r”来检查空白字符。但对象如何判断下一字符为注释的一部分呢？举个例子，当它第一次看到“/”时，它并不知道该字符是表示除法操作符、/=操作符的一部分、或者是行注释或块注释的开始。为了做出决定，WhiteSpaceAndCommentEliminator需要记住这样的事实：即它看到了一个“/”，然后移动到下一字符。如“/”后的字符为另一个“/”或者“*”，那么WhiteSpaceAndCommentEliminator就会知道它正在读取注释，能直接到达注释末端而不用处理或保存任何字符。如果第一个“/”后的字符是除了“/”和“*”的任意字符，那么WhiteSpaceAndCommentEliminator就会知道“/”表示除法操作或/=操作的一部分，它就会停止向前检查字符。

总之，WhiteSpaceAndCommentEliminator在读取字符的同时，还需要追踪几个事件，包括当前字符是否为空白字符，前面读取的字符是否为“/”，正在读取的字符是否为注释，是否到达了注释的末端等。这些分别对应WhiteSpaceAndCommentEliminator对象的不同状态。在每个状态中，WhiteSpaceAndCommentEliminator对读入的下一字符采取不同的行为。

为了帮助观察对象的所有状态以及状态的改变，可以利用图A1-13所示的UML状态图。状态图通过圆角矩形表示状态，每个矩形的上半部分有它的名字。状态图中还有一个称为“初始伪态”的黑色圆圈，它并不真正表示一个状态，只是指向初始状态。在图A1-13中，start状态即为初始状态。从一个状态指向另一个状态的箭头表明对象状态之间的转换或改变。每个转换由一个触发事件、一个斜杠（/）和一项活动标记。转换标记的所有部分在状态图里是可选的。如果对象正处于某一状态，它的一个转换的触发事件发生了，则执行转换活动，对象将呈现由转换所确定的新状态。例如，在图A1-13中，如果WhiteSpaceAndCommentEliminator对象处于start状态，并且下一字符为“/”，那么WhiteSpaceAndCommentEliminator将越过该字符并且转换为saw ‘/’ 状态。如果“/”之后的字符是另一个“/”，那么对象将前进到line comment状态并将一直停留在该状态，直至读到行末尾的字符。如果“/”后的下一字符是“*”，那么对象将前进到block comment状态，并一直停留在该状态直到看见另一个“*”，后面跟有一个“/”，这表明块注释的结束。研究此图，确保你已经理解了。请注意，当前进到空白字符或注释时，WhiteSpaceAndCommentEliminator返回到start状态并从头开始。这个动作是必需的，因为在Java源代码中，任意字符之前都可能有几段连续的注释和空白字符。

对象可以转换到最终状态，通过用白色圆圈里的黑色圆圈来表示最终状态，这表明没有任何转换了。在图A1-13中，下一字符不是空白或注释的一部分时，WhiteSpaceAndCommentEliminator对象结束。注意所有的转换中除了两个指向最终状态的转换之外都有包括前进到下一字符的活动。这两个指向最终状态的转换不可以前进到下一字符，因为下一字符是编译器感兴趣的单词或符号的一部分。注意，如果对象处于saw ‘/’ 状态，但下一字符不是“/”或“*”，那么“/”是一个除法操作或/=操作的一部分，因此我们不想前进。实际上，我们

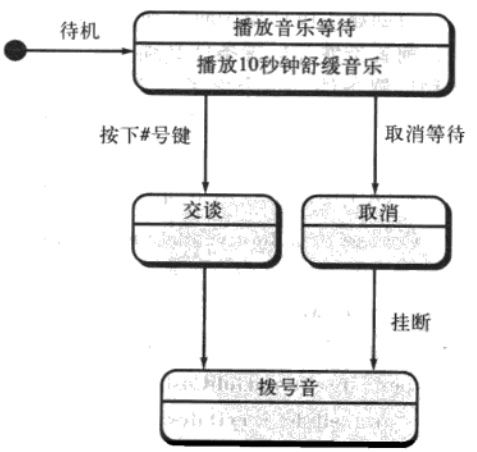
更希望后退一个字符，使得“/”成为下一字符，这样“/”可以被编译器使用。在图A1-13中，后退活动被标记为pushback‘/’。



图A1-13 Java中越过空白和注释的状态图

状态图有助于发现遗漏或意外情况出现。也就是说，使用状态图易于保证对于所有可能的状态，已经考虑了所有可能的触发事件。例如，在图A1-13中，很容易验证每一状态均包括所有可能字符的转换。

UML状态图还包括许多图A1-13不具有的其他特性。例如，当对象处于某一状态时，它通常不做任何事情，只是等待触发事件的发生。然而，也存在一种特殊状态，称为活动状态，在这种状态下，对象执行某些活动，称为do-activity。要表明一种状态是状态图中的活动状态，将短语“do/”添加进状态圆角矩形的下半部分，短语“do/”的后面跟有该状态下将要完成的活动。do-activity可以在任何状态转换发生之前结束，在此之后活动状态类似于正常的等待状态。如果活动状态之外的转换在do-activity结束之前发生，则do-activity中断。



图A1-14 具有活动状态和无触发转换的状态图

由于在转换发生时触发事件是可选的，就可能出现转换的标记中没有触发事件的情况。这种情况下，对于正常的等待状态，对象会立即从该状态转换到新状态。对于活动状态，这样的转换在do-activity完成后立即进行。

图A1-14通过商业电话的状态描述了这种情况。当呼叫者处于等待状态时，呼叫进入播放音乐等待状态（舒缓的音乐将会播放10秒钟）。10秒之后，该状态的do-activity完成，状态行

为如同正常的不活动状态。如果呼叫者在播放音乐等待状态下按下“#”键，则来电转换至取消状态，接着立即转换至拨号音状态。如果“#”键在10秒钟音乐播放完之前就被按下，则do-activity中断，音乐也会立即停止。

对象约束语言概述

UML的多种图为设计模型提供了一套丰富的表达方式。然而，图形表达通常是不够的，需要明确和正式的表示信息的机制约束设计模型的元素。当然，可能会使用自然语言（如英语）来描述约束，但是，这种方法必定导致不一致性和歧义性。因此，一种更为形式化的语言——借鉴集合论和形式化规格说明语言（见第21章）、但具有编程语言的少量数学语法——似乎是恰当的。

对象约束语言（Object Constrain Language, OCL）是对UML的补充，方法是允许使用形式语法和语义构造有关不同设计模型元素（例如，类和对象、事件、消息、接口）的无歧义语句。最简单的OCL语句由以下4部分构造：（1）语境。语境定义语句有效的限制情况；（2）特性。特性表示语境的某些特点（例如，如果语境是类，特性可能是属性）；（3）操作。操纵或取得特性的操作（例如，面向算术，面向集合）；（4）关键词。关键词（例如，if、then、else、and、or、not、implies）是用来具体说明条件表达式的。

举个简单的OCL表达式的例子，考虑第10章讨论的印刷系统。守卫条件位于jobCostAccepted事件上，该事件在PrintJob对象的状态图中可以引起computingJobCost和formingJob状态之间的转换（图10-9）。在该图中（图10-9），守卫条件以自然语言表达，表明只有在顾客同意了工作成本时才进行授权。在OCL中，表达式可采取如下形式：

```
customer
  self.authorizationAuthority='yes'
```

其中，*Customer*类（实际上是类的一个具体实例）的布尔属性authorizationAuthority对于需要满足的守卫条件必须设为“yes”。

当创建了设计模型时，通常有这样一些实例，在这些实例中，设计说明的一些动作完成之前，前置条件和后置条件必须得到满足。OCL提供了强大的工具，以形式化的方式说明前置条件和后置条件。举个例子，考虑对印刷车间系统的扩展（第10章讨论的例子），在该系统中，顾客为印刷工作提供了成本上限，同时，当指定了其他印刷工作特性时，客户提供“下拉式”交付日期。如果成本和交付估计超过了这些界限，该工作将不会提交，并且客户一定要得到通知。在OCL中，可以用下面的方式来描述前置条件和后置条件：

```
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
```

OCL语句定义了一个不变式（inv）——某些行为之前（pre）和之后（post）必须存在的条件。一开始，要建立前置条件，客户必须说明有限制的成本和交付日期，并且授权也必须设

置为“no”。在成本和交付确定之后，就可以应用后置条件了。也应该注意到，表达式：

self.jobAuthorization='yes'

不是将值设置为“yes”而是声明jobAuthorization必须在操作结束时被设置成“yes”。对OCL的完整描述已经超出了本附录的范围。完整的OCL描述可以在www.omg.org/technology/documents/formal/ocl.htm获得。

推荐读物与阅读信息

现在有大量的书讨论UML。讨论UML最新版本的书包括：Miles 和 Hamilton（《Learning UML 2.0》，O'Reilly Media, Inc., 2006）；Booch、Rumbaugh和 Jacobson（《Unified Modeling language User Guide》），2d ed., Addison-Wesley, 2005）；Ambler（《The Elements of UML 2.0 Style》，Cambridge University Press, 2005）以及Pilone和 Pitman（《UML 2.0 in a Nutshell》，O'Reilly Media, Inc., 2005）。

网上有大量在软件工程建模中使用UML的信息资源。最新的WWW列表可以在SEPA网站www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm中的“analysis”和“design”栏目下找到。

面向对象概念

关键概念
属性
类
　边界
　特性
　控制器
　定义
　设计
　实体
封装
继承
消息
方法
操作
多态性
服务
子类
超类

什么是面向对象（Object-Oriented，OO）观点？为什么一个方法被认为是面向对象的？什么是对象？在20世纪80年代和90年代，面向对象概念赢得了广泛的传播，在那时，关于这些问题的答案有很多不同观点，但是今天关于面向对象概念的一个统一观点已经形成。本附录将提供这个重要课题的简要概述，并介绍基本概念和术语。

为了理解面向对象的观点，首先考虑一个现实世界对象的例子——你正在坐着的东西——一把椅子。Chair类是更大的类PieceOfFurniture的子类，个体椅子是Chair类的成员（通常称为实例）。可以将一组一般属性关联到PieceOfFurniture类的每个对象上。例如，在很多可能的属性中，所有家具都有成本、尺寸、重量、位置和颜色。不管我们在讨论桌子还是椅子、沙发还是衣橱，这些属性都适用。因为Chair是PieceOfFurniture的一个成员，所以Chair继承了PieceOfFurniture类的所有属性。

我们通过描述类的属性来定义类，但是会缺少一些东西。PieceOfFurniture类的每个对象都可以通过很多方法进行操作。它可以被买卖，可以被物理地改变（例如，你可以锯掉它的一条腿或将它涂成紫色），也可以将它从一个地方移到另一个地方。每个操作（其他术语为服务或方法）都会修改该对象的一个或多个属性。例如，如果位置属性为一个组合数据项，其定义如下：

location = building + floor + room

那么名为move()的操作将会修改形成属性location的一个或多个数据项（building、floor或room）。为此，move()必须具有这些数据项的“知识”。move()操作可以用在椅子或桌子上，只要它俩都是PieceOfFurniture类的实例，PieceOfFurniture的有效操作——buy()、sell()、weigh()——被描述为类定义的一部分，并被类的所有实例继承。

Chair类（通常和所有对象）封装数据（定义椅子的属性值）、操作（改变椅子属性的动作）、其他对象、常量（设值）和其他相关信息。封装意味着该信息的所有部分被打包在一个名字下，并且可以作为一份规格说明或程序构件加以复用。

现在我们已经介绍了一些基本概念，面向对象的更正式的定义会更有意义。Coad和Yourdon[Coa91]是这样定义面向对象的：

$$\text{面向对象} = \text{对象} + \text{类} + \text{继承} + \text{通信}$$

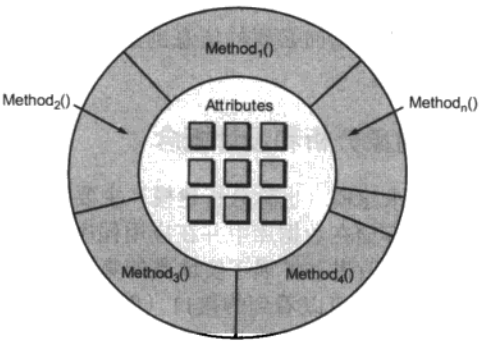
其中，有3个概念都已经介绍过了，通信概念将在本附录的后面部分讨论。

类和对象

类是面向对象的概念，它封装了描述现实世界实体的内容和行为所需要的数据抽象和过程抽象。描述类的数据抽象被能以某种方式操作数据的过程抽象“墙”所围绕[Tay90]（如图A2-1所示）。在设计良好的类中，到达属性（和其上操作）的唯一途径是通过组成墙的方法，如图中所示，因此，该类封装了数据（在墙里面）和操作数据的处理（组成墙的方法）。这样

做可以获得信息隐藏（第8章），并减少由变更引起的副作用。

因为方法倾向于操作一组受限的属性，它们的内聚性得到了改进，并且因为通信仅仅通过组成“墙”的方法才能进行，类趋向于减弱与系统[⊖]其他元素的连接。



图A2-1 类的图解表示

换种说法，类是一般化的描述（例如，模板或蓝图），它描述了相似对象的集合。通过定义，对象是特定类的实例并继承类的属性和控制属性的操作。超类（通常称为基类）是对与其关联的一组类的泛化。子类是超类的特殊化。例如，超类MotorVehicle是类Truck、SUV、Automobile和Van的泛化。子类Automobile继承MotorVehicle的所有属性，并且还有特定于汽车的附加属性。

这些定义意味着存在一种类的层次结构，在这种结构中，超类的属性和方法由子类继承，每个子类又可以增加附加的“私有”属性和方法。例如，操作sitOn()和turn()对于Chair子类是私有的。

属性

属性关联于类并以某种方式描述类。属性可以具有由枚举域定义的数值。在大多数情况下，域仅仅是一组特定的值。例如，假定Automobile类有一个color属性。color的值域为{white，black，silver，gray，blue，red，yellow，green}。在更复杂的情况下，域可能是一个类。继续这个例子，Automobile类还有一个powerTrain属性，该属性本身就是一个类。PowerTrain类将包括描述车辆的特定发动机和变速器的属性。

可以通过将缺省值（特征）赋给属性来扩大特征（值域）。例如，color属性的缺省值为white。通过分配{值，概率}对，将某个概率与特定的特征联系起来也是有用的。考虑汽车的color属性。在某些应用系统中（例如，制造计划），就有必要为每种颜色分配一个概率（例如，白色和黑色作为汽车颜色的概率非常高）。

操作、方法和服务

对象封装数据（表示为属性的集合）和处理数据的算法。将这些算法称为操作、方法或服务[⊖]，并可以视为处理构件。

每个由对象封装的操作都提供了对象行为的一种表示。例如，Automobile对象的GetColor()操作可以取出存储在color属性中的颜色。该操作的含义是：Automobile类已被设计为接收一个刺激（我们称这个刺激为消息），该刺激请求类的特定实例的颜色。只要对象收到该刺激，它就会初始化一些行为。这样的行为可以简单到检索汽车的颜色，也可以复杂到初始

⊖ 然而，值得注意的是，在面向对象系统中，耦合会成为一个严重的问题。当系统中不同部分的类用作属性的数据类型和方法的参数时，耦合就产生了。即使只能通过过程调用进入对象，也并不意味着耦合必然很低，只不过比直接进入对象的程度低而已。

⊖ 在本段的上下文中，使用了术语操作，但术语方法和服务同样流行。

化连接大量不同对象的刺激链。在后一种情况下，思考一个例子，由Object1对象收到的初始化刺激产生两个不同的刺激分别送往Object2和Object3。由第二个和第三个对象封装的操作作用于刺激，返回必要的信息到第一个对象。Object1然后使用返回信息以满足初始化刺激所要求的行为。

面向对象分析和设计概念

需求建模（也称分析建模）主要集中在由问题陈述中直接抽取的类上。这些实体类通常表示的是存储在数据库中并在应用程序持续时间内一直存在的事物（除非特意将其删除）。

设计改进和扩展了实体类的集合。边界类和控制器类在设计期间得到了开发和改进。边界类创建用户可以看到的接口（例如，交互屏幕和输出报告），该接口可在使用软件时与用户进行交互。设计边界类是为了管理实体对象显示给用户的方式。

设计控制器类用来管理（1）实体对象的建立或更新。（2）当边界对象从实体对象获取信息时边界对象的实例化。（3）对象集合之间的复杂通信。（4）对象之间或用户与应用系统之间通信数据的有效性。

以下段落讨论的概念在分析和设计工作中十分有用。

继承。继承是传统系统和面向对象系统之间的主要区别之一。子类Y继承其超类X的所有属性和操作，这意味着原来为X设计和实现的所有数据结构和算法都可以立即为Y所用——不需要做进一步的工作，重用可直接完成。

超类的属性或操作的任何改变都可以立即被所有子类继承。因此，类的层次结构成为一种机制，（在高层次上的）改变可以立即在系统中传播。

值得注意的是，在类继承的各个层次上，对于那些继承自更高层次的类，还可以增加新的属性和操作。实际上，每当创建一个新类时，你都可以有很多选择：

(1) 可以从头开始设计和构建类，也就是说，可以不使用继承。

(2) 可以对类的层次结构进行查找，以确定在较高层次的类中是否包含了大多数需要的属性和操作。新类可以根据需要继承较高层的类，然后进行增加。

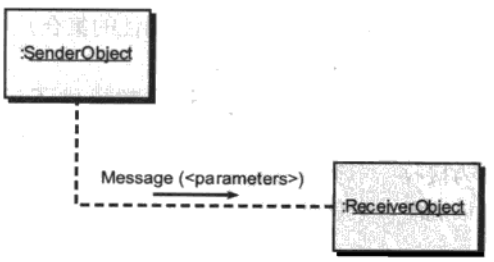
(3) 可以对类的层次结构进行调整，使得所需要的属性和操作可以被新类继承。

(4) 已有类的特征可以被重写，对于新类，可以实现不同版本的属性或操作。

像所有基本的设计概念一样，继承可为设计带来显著好处。但是，若使用不恰当[⊖]，就会使设计变得不必要的复杂，并导致难以维护和易于出错的软件。

消息。类之间必须彼此交互来完成设计目标。消息刺激接收对象产生某种行为，当操作执行时，完成该行为。

对象之间的相互作用由图A2-2做了概要描述。SenderObject内的操作产生一条message (<parameters>)形式的消息，其中，参数(parameters) 指定ReceiverObject作为消息刺激的对象、将要接收这条消息的ReceiverObject中的操作以及为了使操作获得成功需要提供信息的数据项。所定义的类之间的协作是需求模型的一部分，在消息设计中提供了十分有用的指导。



图A2-2 对象之间的消息传递

⊖ 例如，设计子类继承多个超类的属性和操作（有时称为“多重继承”）会让大多数设计者费解。

Cox[Cox86]以如下方式描述了类之间的交互：

通过给对象[类]发送消息告诉对象做什么来要求其执行某个操作。接收者[对象]通过选择对应消息名字的操作去执行，然后将控制权交还给调用者响应消息。消息提供了单个对象和整体的面向对象系统行为的洞察力。

多态。多态是一种特性，这种特性可以显著减少扩展已存在的面向对象系统的设计所需要的工作量。为了理解多态，考虑一个传统的应用程序，此程序必须绘制4种不同类型的图：线图、饼图、直方图和Kiviat图。理想情况下，只要针对某种特殊类型的图收集数据，该图就可以自行绘制。要在传统程序中完成这项工作（并维护模块的内聚性），就有必要为每类图形都开发绘图模块。在设计中必须嵌入类似下面的控制逻辑：

```
case of graphtype:
  if graphtype=linegraph then DrawLineGraph(data);
  if graphtype=piechart then DrawPieChart(data);
  if graphtype=histogram then DrawHisto (data);
  if graphtype=kiviat then DrawKiviat (data);
end case;
```

虽然设计很简单，但是添加新图类型会很棘手。要为每种图创建新的绘图模块，然后还要修改控制逻辑以反映新的绘图类型。

为了在面向对象系统中解决这个问题，所有图形都成为一般类Graph的子类。使用重载的概念[Tay90]，每个子类都定义draw操作。一个对象可以发送一条draw消息给由任意子类实例化的任意对象。接收消息的对象将会调用它本身的draw操作去创建适当的图，因此，将设计精简为：

```
draw<graphtype>
```

当系统中增加一种新的画图类型时，就会创建一个带有它独有的draw操作的子类。但是，在想要绘图的任何对象内不需要任何改变，因为消息draw〈graphtype〉没有改变。总之，多态能够使很多不同的操作拥有相同的名字。反过来也使对象之间彼此分离，使每个对象更加独立。

设计类。需求模型定义了分析类的完整集合。每个分析类都描述了问题域的某种元素，集中在用户或客户可见的问题方面。分析类的抽象等级相对比较高。

随着设计模型的进展，软件团队必须定义一组设计类：(1) 通过提供使类得以实现的设计细节对分析类进行细化；(2) 创建一组新的设计类实现支持业务解决方案的软件基础结构。建议采用5种不同类型的设计类，每种设计类表示设计架构的不同层[Amb01]：

(1) 用户界面类定义所有人机交互（HCI）所必需的抽象。

(2) 业务域类通常为早先定义的分析类的改进。类确定实现业务域的某些元素所需要的属性和操作（方法）。

(3) 处理类执行需要管理业务域类的低层业务抽象。

(4) 持久类表示持续时间超过软件执行时间的数据存储（例如，数据库）。

(5) 系统类实现软件管理和控制功能，使系统能在内部计算环境和外部世界中操作和交流。

随着体系结构设计的进展，软件团队应当为每个设计类开发出完整的一组属性和操作。随着每个分析类转化为设计表示，抽象程度会降低。也就是说，分析类使用业务域术语表示对象（及应用于对象上的相关方法）。作为实现的指南，设计类在一定程度上给出了更多的技术细节。

Arlow和Neustadt[Arl02]建议对每个设计类进行评审，以确保它是“组成良好”的。组成良好的设计类的4个特征为：

完整性和充分性。设计类应当完整地封装期望可能存在于类中的所有属性和方法（基于对类名的合理解释）。例如，对于为视频编辑软件定义的Scene类，仅当它包括所有与创建视频场景有关的属性和方法时，才是完整的。充分性确保设计类仅包括那些能够完成类的意图的方法，不会多也不会少。

原始性。与设计类有关的方法应集中在完成类的一个特定功能上。一旦该功能由一个方法实现了，该类不应当再提供其他方法去完成同样的事情。例如，视频编辑软件的VideoClip类也许由start-point和end-point属性来表明剪辑的开始点和结束点（注意，载入到系统中的未加工视频也许会比剪辑过的要长一些）。方法setStartPoint()和setEndPoint()提供了唯一的手段建立剪辑的开始点和结束点。

高内聚。一个内聚的设计类是专一的。也就是说，它有一组小的、集中的责任，并专一地应用属性和方法来完成这些责任。例如，视频编辑软件的VideoClip类也许包括一套方法用来编辑视频剪辑。只要每个方法单独集中于与视频剪辑有关的属性上，就维持了内聚性。

低耦合。在设计模型中，设计类彼此协作是有必要的。然而，协作应保持在一个可接受的最低限度。如果设计模型高度耦合（所有设计类都与所有的其他设计类协作），随着时间的推移，系统将很难实现、测试和维护。总之，在子系统中，设计类应该对其他类只有有限的了解。这个限制称为Demeter原则[Lie03]，建议一个方法应该只能给相邻类的方法发送消息。⊖

推荐读物和阅读信息

在过去的30年里，有成百上千本书讲述面向对象的编程、分析和设计。Weisfeld（《The Object-Oriented Thought Process》，2d ed.，Sams Publishing，2003）介绍了对一般的面向对象概念和原则的有用讨论。McLaughlin和他的同事（《Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D》，O'Reilly Media，Inc.，2006）提供了面向对象分析和设计方法的可行的和轻松的论述。Booch和他的同事（《Object-Oriented Analysis and Design with Applications》，3d ed.，Addison-Wesley，2007）对面向对象分析和设计进行了更深入的讨论。Wu（《An Introducation to Object-Oriented Programming with Java》，McGraw-Hill，2005）编写了一本面向对象编程的综合书籍，在众多为不同编程语言编写的书籍中，这一本是很经典的。

网上有大量的面向对象技术的信息源。最新的WWW参考文献列表可以在SEPA站点www.mhhe.com/engcs/ compsci/pressman/professional/olc/ser.htm的“analysis”和“design”栏目下找到。

⊖ 叙述Demeter原则的一种不太正式的说法是“每个单元应当只与它的朋友交谈，不要与陌生人谈话。”

参考文献

[Abb83] Abbott, R., "Program Design by Informal English Descriptions," *CACM*, vol. 26, no. 11, November 1983, pp. 892–894.

[ACM98] ACM/IEEE-CS Joint Task Force, *Software Engineering Code of Ethics and Professional Practice*, 1998, available at www.acm.org/serving/se/code.htm.

[Ada93] Adams, D., *Mostly Harmless*, Macmillan, 1993.

[AFC88] *Software Risk Abatement*, AFCS/AFLC Pamphlet 800-45, U.S. Air Force, September 30, 1988.

[Agi03] The Agile Alliance Home Page, www.agilealliance.org/home.

[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.

[Aka04] Akao, Y., *Quality Function Deployment*, Productivity Press, 2004.

[Ale77] Alexander, C., *A Pattern Language*, Oxford University Press, 1977.

[Ale79] Alexander, C., *The Timeless Way of Building*, Oxford University Press, 1979.

[Amb95] Ambler, S., "Using Use-Cases," *Software Development*, July 1995, pp. 53–61.

[Amb98] Ambler, S., *Process Patterns: Building Large-Scale Systems Using Object Technology*, Cambridge University Press/SIGS Books, 1998.

[Amb01] Ambler, S., *The Object Primer*, 2d ed., Cambridge University Press, 2001.

[Amb02a] Ambler, S., "What Is Agile Modeling (AM)?" 2002, www.agilemodeling.com/index.htm.

[Amb02b] Ambler, S., and R. Jeffries, *Agile Modeling*, Wiley, 2002.

[Amb02c] Ambler, S., "UML Component Diagramming Guidelines," available at www.modelingstyle.info/, 2002.

[Amb04] Ambler, S., *"Examining the Cost of Change Curve,"* in *The Object Primer*, 3d ed., Cambridge University Press, 2004.

[Amb06] Ambler, S., "The Agile Unified Process (AUP), 2006, available at www.ambysoft.com/unifiedprocess/agileUP.html.

[And06] Andrews, M., and J. Whittaker, *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*, Addison-Wesley, 2006.

[ANS87] ANSI/ASQC A3-1987, *Quality Systems Terminology*, 1987.

[Ant06] Anton, D., and C. Anton, *ISO 9001 Survival Guide*, 3d ed., AEM Consulting Group, 2006.

[AOS07] AOSD.net (Aspect-Oriented Software Development), glossary, available at http://aosd.net/wiki/index.php?title=Glossary.

[App00] Appleton, B., "Patterns and Software: Essential Concepts and Terminology," February 2000, available at www.cmcrossroads.com/bradapp/docs/patterns-intro.html.

[App08] Apple Computer, *Accessibility*, 2008, available at www.apple.com/disability/.

[Arl02] Arlow, J., and I. Neustadt, *UML and the Unified Process*, Addison-Wesley, 2002.

[Arn89] Arnold, R. S., "Software Restructuring," *Proc. IEEE*, vol. 77, no. 4, April 1989, pp. 607–617.

[Art97] Arthur, L. J., "Quantum Improvements in Software System Quality," *CACM*, vol. 40, no. 6, June 1997, pp. 47–52.

[Ast04] Astels, D., *Test Driven Development: A Practical Guide*, Prentice Hall, 2004.

[Ave04] Aversan, L., et al., "Managing Coordination and Cooperation in Distributed Software Processes: The GENESIS Environment," *Software Process Improvement and Practice*, vol. 9, Wiley Interscience, 2004, pp. 239–263.

[Baa07] de Baar, B., "Project Risk Checklist," 2007, available at www.softwareprojects.org/project_riskmanagement_starting62.htm.

[Bab86] Babich, W. A., *Software Configuration Management*, Addison-Wesley, 1986.

[Bac97] Bach, J., "'Good Enough Quality: Beyond the Buzzword," *IEEE Computer*, vol. 30, no. 8, August 1997, pp. 96–98.

[Bac98] Bach, J., "The Highs and Lows of Change Control," *Computer*, vol. 31, no. 8, August 1998, pp. 113–115.

[Bae98] Baetjer, Jr., H., *Software as Capital*, IEEE Computer Society Press, 1998, p. 85.

[Bak72] Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM*

Systems Journal., vol. 11, no. 1, 1972, pp. 56–73.

[Ban06] Baniassad, E., et al., "Discovering Early Aspects," *IEEE Software*, vol. 23, no. 1, January–February, 2006, pp. 61–69.

[Bar06] Baresi, L., E. DiNitto, and C. Ghezzi, "Toward Open-World Software: Issues and Challenges," *IEEE Computer*, vol. 39, no. 10, October 2006, pp. 36–43.

[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.

[Bas03] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, 2d ed., Addison-Wesley, 2003.

[Bec00] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[Bec01a] Beck, K., et al., "Manifesto for Agile Software Development," www.agilemanifesto.org/.

[Bec04a] Beck, K., *Extreme Programming Explained: Embrace Change*, 2d ed., Addison-Wesley, 2004.

[Bec04b] Beck, K., *Test-Driven Development: By Example*, 2d ed., Addison-Wesley, 2002.

[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, downloadable from http://jeffsutherland.com/scrum/scrum_plop.pdf.

[Bei84] Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand-Reinhold, 1984.

[Bei90] Beizer, B., *Software Testing Techniques*, 2d ed., Van Nostrand-Reinhold, 1990.

[Bei95] Beizer, B., *Black-Box Testing*, Wiley, 1995.

[Bel81] Belady, L., Foreword to *Software Design: Methods and Techniques* (L. J. Peters, author), Yourdon Press, 1981.

[Bel95] Bellinzona R., M. G. Gugini, and B. Pernici, "Reusing Specifications in OO Applications," *IEEE Software*, March 1995, pp. 65–75.

[Ben99] Bentley, J., *Programming Pearls*, 2d ed., Addison-Wesley, 1999.

[Ben00] Bennatan, E. M., *Software Project Management: A Practitioner's Approach*, 3d ed., McGraw-Hill, 2000.

[Ben02] Bennett, S., S. McRobb, and R. Farmer, *Object-Oriented Analysis and Design*, 2d ed., McGraw-Hill, 2002.

[Ber80] Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management*, Prentice Hall, 1980.

[Ber93] Berard, E., *Essays on Object-Oriented Software Engineering*, vol. 1, Addison-Wesley, 1993.

[Bes04] Bessin, J., "The Business Value of Quality," IBM developerWorks, June 15, 2004, available at www-128.ibm.com/developerworks/rational/library/4995.html.

[Bha06] Bhat, J., M. Gupta, and S. Murthy, "Lessons from Offshore Outsourcing," *IEEE Software*, vol. 23, no. 5, September–October 2006.

[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.

[Bin93] Binder, R., "Design for Reuse Is for Real," *American Programmer*, vol. 6, no. 8, August 1993, pp. 30–37.

[Bin94a] Binder, R., "Testing Object-Oriented Systems: A Status Report," *American Programmer*, vol. 7, no. 4, April 1994, pp. 23–28.

[Bin94b] Binder, R. V., "Object-Oriented Software Testing," *Communications of the ACM*, vol. 37, no. 9, September 1994, p. 29.

[Bin99] Binder, R., *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley, 1999.

[Bir98] Biró, M., and T. Remzsõ, "Business Motivations for Software Process Improvement," ERCIM News No. 32, January 1998, available at www.ercim.org/publication/Ercim_News/enw32/biro.html.

[Boe81] Boehm, B., *Software Engineering Economics*, Prentice Hall, 1981.

[Boe88] Boehm, B., "A Spiral Model for Software Development and Enhancement," *Computer*, vol. 21, no. 5, May 1988, pp. 61–72.

[Boe89] Boehm, B. W., *Software 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.

[Boe98] Boehm, B., "Using the WINWIN Spiral Model: A Case Study," *Computer*, vol. 31, no. 7, July 1998, pp. 33–44.

[Boe00] Boehm, B., et al., *Software Cost Estimation in COCOMO II*, Prentice Hall, 2000.

[Boe01a] Boehm, B., "The Spiral Model as a Tool for Evolutionary Software Acquisition," *CrossTalk*, May 2001, available at www.stsc.hill.af.mil/crosstalk/2001/05/boehm.html.

[Boe01b] Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*, vol. 34, no. 1, January 2001, pp. 135–137.

[Boe08] Boehm, B., "Making a Difference in the Software Century," *IEEE Computer*, vol. 41, no. 3, March 2008, pp. 32–38.

[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.

[Boh00] Bohl, M., and M. Rynn, *Tools for Structured Design: An Introduction to Programming Logic*, 5th ed., Prentice Hall, 2000.

[Boi04] Boiko, B., *Content Management Bible*, 2d ed., Wiley, 2004.

[Bol02] Boldyreff, C., et al., "Environments to Support Collaborative Software Engineering," 2002, downloadable from www.cs.put.poznan.pl/dweiss/site/publications/download/csmre-paper.pdf.

[Boo94] Booch, G., *Object-Oriented Analysis and Design*, 2d ed., Benjamin Cummings, 1994.

[Boo05] Booch, G., J. Rumbaugh, and I. Jacobsen, *The Unified Modeling Language User Guide*. 2d ed., Addison-Wesley, 2005.

[Boo06] Bootstrap-institute.com, 2006, www.cse.dcu.ie/espinode/directory/directory.html.

[Boo08] Booch, G., *Handbook of Software Architecture*, 2008, available at www.booch.com/architecture/systems.jsp.

[Bor01] Borchers, J., *A Pattern Approach to Interaction Design*, Wiley, 2001.

[Bos00] Bosch, J., *Design & Use of Software Architectures*, Addison-Wesley, 2000.

[Bra85] Bradley, J. H., "The Science and Art of Debugging," *Computerworld*, August 19, 1985, pp. 35–38.

[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.

[Bre02] Breen, P., "Exposing the Fallacy of 'Good Enough' Software," informit.com, February 1, 2002, available at www.informit.com/articles/article.asp?p=25141&rl=1.

[Bro95] Brooks, F., *The Mythical Man-Month*, Silver Anniversary edition, Addison-Wesley, 1995.

[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.

[Bro01] Brown, B., *Oracle9i Web Development*, 2d ed., McGraw-Hill, 2001.

[Bro03] Brooks, F, "Three Great Challenges for Half-Century-Old Computer Science," *JACM*, vol. 50, no. 1, January 2003, pp. 25–26.

[Bro06] Broy, M., "The 'Grand Challenge' in Informatics: Engineering Software Intensive Systems," *IEEE Computer*, vol. 39, no. 10, October 2006, pp. 72–80.

[Buc99] Bucanac, C., "The V-Model," University of Karlskrona/Ronneby, January 1999, downloadable from www.bucanac.com/documents/The_V-Model.pdf.

[Bud96] Budd, T., *An Introduction to Object-Oriented Programming*, 2d ed., Addison-Wesley, 1996.

[Bus96] Buschmann, F., et al., *Pattern-Oriented Software Architecture*, Wiley, 1996.

[Bus07] Buschmann, F., et al., *Pattern-Oriented Software Architecture, A System of Patterns*, Wiley, 2007.

[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, downloadable from www.dsic.upv.es/~west/iwwost02/papers/cachero.pdf.

[Cai03] Caine, Frarber, and Gordon, Inc., *PDL/81*, 2003, available at www.cfg.com/pdl81/lpd.html.

[Car90] Card, D. N., and R. L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990.

[Cas89] Cashman, M., "Object Oriented Domain Analysis," *ACM Software Engineering Notes*, vol. 14, no. 6, October 1989, p. 67.

[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.

[CCS02] CS3 Consulting Services, 2002, www.cs3inc.com/DSDM.htm.

[Cec06] Cechich, A., et al., "Trends on COTS Component Identification," *Proc. Fifth Intl. Conf. on COTS-Based Software Systems*, IEEE, 2006.

[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.

[Cha93] de Champeaux, D., D. Lea, and P. Faure, *Object-Oriented System Development*, Addison-Wesley, 1993.

[Cha03] Chakravarti, A., "Online Software Design Pattern Links," 2003, available at www

.anupriyo.com/oopfm.shtml.
[Che77] Chen, P., *The Entity-Relationship Approach to Logical Database Design*, QED Information Systems, 1977.
[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.
[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.
[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.
[Cig07] Cigital, Inc., "Case Study: Finding Defects Earlier Yields Enormous Savings," 2007, available at www.cigital.com/solutions/roi-cs2.php.
[Cla05] Clark, S., and E. Baniasaad, *Aspect-Oriented Analysis and Design*, Addison-Wesley, 2005.
[Cle95] Clements, P., "From Subroutines to Subsystems: Component Based Software Development," *American Programmer*, vol. 8, no. 11, November 1995.
[Cle03] Clements, P., R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2003.
[Cle06] Clemmons, R., "Project Estimation with Use Case Points," *CrossTalk*, February 2006, p. 18–222, downloadable from www.stsc.hill.af.mil/crosstalk/2006/02/0602Clemmons.pdf.
[CMM07] *Capability Maturity Model Integration (CMMI)*, Software Engineering Institute, 2007, available at www.sei.cmu.edu/cmmi/.
[CMM08] *People Capability Maturity Model Integration (People CMM)*, Software Engineering Institute, 2008, available at www.sei.cmu.edu/cmm-p/.
[Coa91] Coad, P., and E. Yourdon, *Object-Oriented Analysis*, 2d ed., Prentice Hall, 1991.
[Coa99] Coad, P., E. Lefebvre, and J. DeLuca, *Java Modeling in Color with UML*, Prentice Hall, 1999.
[Coc01a] Cockburn, A., and J. Highsmith, "Agile Software Development: The People Factor," *IEEE Computer*, vol. 34, no. 11, November 2001, pp. 131–133.
[Coc01b] Cockburn, A., *Writing Effective Use-Cases*, Addison-Wesley, 2001.
[Coc02] Cockburn, A., *Agile Software Development*, Addison-Wesley, 2002.
[Coc04] Cockburn, A., "What the Agile Toolbox Contains," *CrossTalk*, November 2004, available at www.stsc.hill.af.mil/crosstalk/2004/11/0411Cockburn.html.
[Coc05] Cockburn, A., *Crystal Clear*, Addison-Wesley, 2005.
[Con96] Conradi, R., "Software Process Improvement: Why We Need SPIQ," NTNU, October 1996, downloadable from www.idi.ntnu.no/grupper/su/publ/pdf/nik96-spiq.pdf.
[Con02] Conradi, R., and A. Fuggetta, "Improving Software Process Improvement," *IEEE Software*, July–August 2002, pp. 2–9, downloadable from http://citeseer.ist.psu.edu/conradi02improving.html.
[Con93] Constantine, L., "Work Organization: Paradigms for Project Management and Organization, *CACM*, vol. 36, no. 10, October 1993, pp. 34–43.
[Con95] Constantine, L, "What DO Users Want? Engineering Usability in Software," *Windows Tech Journal*, December 1995, available from www.forUse.com.
[Con03] Constantine, L., and L. Lockwood, *Software for Use*, Addison-Wesley, 1999; see also www.foruse.com/.
[Cop05] Coplien, J., "Software Patterns," 2005, available at http://hillside.net/patterns/definition.html.
[Cor98] Corfman, R., "An Overview of Patterns," in *The Patterns Handbook*, SIGS Books, 1998.
[Cou00] Coulouris, G., J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 3d ed., Addison-Wesley, 2000.
[Cox86] Cox, Brad, *Object-Oriented Programming*, Addison-Wesley, 1986.
[Cri92] Christel, M. G., and K. C. Kang, "Issues in Requirements Elicitation," Software Engineering Institute, CMU/SEI-92-TR-12 7, September 1992.
[Cro79] Crosby, P., *Quality Is Free*, McGraw-Hill, 1979.
[Cro07] Cross, M., and M. Fisher, *Developer's Guide to Web Application Security*, Syngress Publishing, 2007.
[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.
[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.
[Cur01] Curtis, B., W. Hefley, and S. Miller, *People Capability Maturity Model*, Addison-Wesley, 2001.
[CVS07] Concurrent Versions System, Ximbiot, http://ximbiot.com/cvs/wiki/index.php?title=Main_Page, 2007.

[DAC03] "An Overview of Model-Based Testing for Software," Data and Analysis Center for Software, CR/TA 12, June 2003, downloadable from www.goldpractices.com/dwnload/practice/pdf/Model_Based_Testing.pdf.

[Dah72] Dahl, O., E. Dijkstra, and C. Hoare, *Structured Programming*, Academic Press, 1972.

[Dar91] Dart, S., "Concepts in Configuration Management Systems," *Proc. Third International Workshop on Software Configuration Management*, ACM SIGSOFT, 1991, downloadable from 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 www.perforce.com/perforce/conf99/dart.html.

[Dar01] Dart, S., *Spectrum of Functionality in Configuration Management Systems*, Software Engineering Institute, 2001, available at www.sei.cmu.edu/legacy/scm/tech_rep/TR11_90/TOC_TR11_90.html.

[Das05] Dasari, R., "Lean Software Development," a white paper, downloadable from www.projectperfect.com.au/downloads/Info/info_lean_development.pdf, 2005.

[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.

[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.

[Dav95a] Davis, M., "Process and Product: Dichotomy or Duality," *Software Engineering Notes*, ACM Press, vol. 20, no. 2, April, 1995, pp. 17–18.

[Dav95b] Davis, A., *201 Principles of Software Development*, McGraw-Hill, 1995.

[Day99] Dayani-Fard, H., et al., "Legacy Software Systems: Issues, Progress, and Challenges," IBM Technical Report: TR-74.165-k, April 1999, available at www.cas.ibm.com/toronto/publications/TR-74.165/k/legacy.html.

[Dem86] Deming, W. E., *Out of the Crisis*, MIT Press, 1986.

[DeM79] DeMarco, T., *Structured Analysis and System Specification*, Prentice Hall, 1979.

[DeM95] DeMarco, T., *Why Does Software Cost So Much?* Dorset House, 1995.

[DeM95a] DeMarco, T., "Lean and Mean," *IEEE Software*, November 1995, pp. 101–102.

[DeM98] DeMarco, T., and T. Lister, *Peopleware*, 2d 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.

[Den73] Dennis, J., "Modularity," in *Advanced Course on Software Engineering* (F. L. Bauer, ed.), Springer-Verlag, 1973, pp. 128–182.

[Dev01] Devedzik, V., "Software Patterns," in *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Co., 2001.

[Dha95] Dhama, H., "Quantitative Metrics for Cohesion and Coupling in Software," *Journal of Systems and Software*, vol. 29, no. 4, April 1995.

[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.

[Dij76a] Dijkstra, E., "Structured Programming," in *Software Engineering, Concepts and Techniques*, (J. Buxton et al., eds.), Van Nostrand-Reinhold, 1976.

[Dij76b] Dijkstra, E., *A Discipline of Programming*, Prentice Hall, 1976.

[Dij82] Dijksta, E., "On the Role of Scientific Thought," *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, 1982.

[Dix99] Dix, A., "Design of User Interfaces for the Web," *Proc. User Interfaces to Data Systems Conference*, September 1999, downloadable from www.comp.lancs.ac.uk/computing/users/dixa/topics/webarch/.

[Dob04] Dobb, F., *ISO 9001:2000 Quality Registration Step-by-Step*, 3d ed., Butterworth-Heinemann, 2004.

[Don99] Donahue, G., S. Weinschenck, and J. Nowicki, "Usability Is Good Business," Compuware Corp., July 1999, available from www.compuware.com.

[Dre99] Dreilinger, S., "CVS Version Control for Web Site Projects," 1999, available at www.durak.org/cvswebsites/howto-cvs/howto-cvs.html.

[Dru75] Drucker, P., *Management*, W. H. Heinemann, 1975.

[Duc01] Ducatel, K., et al., *Scenarios for Ambient Intelligence in 2010*, ISTAG-European Commission, 2001, downloadable from ftp://ftp.cordis.europa.eu/pub/ist/docs/

istagscenarios2010.pdf.
[Dun82] Dunn, R., and R. Ullman, *Quality Assurance for Computer Software*, McGraw-Hill, 1982.
[Dun01] Dunaway, D., and S. Masters, *CMM-Based Appraisal for Internal Process Improvement (CBA IPI Version 1,2 Method Description)*, Software Engineering Institute, 2001, downloadable from www.sei.cmu.edu/publications/documents/01.reports/01tr033.html.
[Dun02] Dunn, W., *Practical Design of Safety-Critical Computer Systems*, William Dunn, 2002.
[Duy02] VanDuyne, D., J. Landay, and J. Hong, *The Design of Sites*, Addison-Wesley, 2002.
[Dye92] Dyer, M., *The Cleanroom Approach to Quality Software Development*, Wiley, 1992.
[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.
[Eji91] Ejiogu, L., *Software Engineering with Formal Metrics*, QED Publishing, 1991.
[Elr01] Elrad, T., R. Filman, and A. Bader (eds.), "Aspect Oriented Programming," *Comm. ACM*, vol. 44, no. 10, October 2001, special issue.
[Eri05] Ericson, C., *Hazard Analysis Techniques for System Safety*, Wiley-Interscience, 2005.
[Eri08] Erickson, T., *The Interaction Design Patterns Page*, May 2008, available at www.visi.com/~snowfall/InteractionPatterns.html.
[Eva04] Evans, E., *Domain Driven Design*, Addison-Wesley, 2003.
[Fag86] Fagan, M., "Advances in Software Inspections," *IEEE Trans. Software Engineering*, vol. 12, no. 6, July 1986.
[Fel89] Felican, 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.
[Fel07] Feller, J., et al. (eds.), *Perspectives on Free and Open Source Software*, The MIT Press, 2007.
[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.
[Fer97] Ferguson, P., et al., "Results of Applying the Personal Software Process," *IEEE Computer*, vol. 30, no. 5, May 1997, pp. 24–31.
[Fer98] Ferdinandi, P. L., "Facilitating Communication," *IEEE Software*, September 1998, pp. 92–96.
[Fer00] Fernandez, E. B., and X. Yuan, "Semantic Analysis Patterns," *Proceedings of the 19th Int. Conf. on Conceptual Modeling, ER2000*, Lecture Notes in Computer Science 1920, Springer, 2000, pp. 183–195. Also available from www.cse.fau.edu/~ed/SAPpaper2.pdf.
[Fir93] Firesmith, D. G., *Object-Oriented Requirements Analysis and Logical Design*, Wiley, 1993.
[Fis06] Fisher, R., and D. Shapiro, *Beyond Reason: Using Emotions as You Negotiate*, Penguin, 2006.
[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.
[Fle98] Fleming, Q. W., and J. M. Koppelman, "Earned Value Project Management," *CrossTalk*, vol. 11, no. 7, July 1998, p. 19.
[Fos06] Foster, E., "Quality Culprits," InfoWorld Grip Line Weblog, May 2, 2006, available at http://weblog.infoworld.com/gripeline/2006/05/02_a395.html.
[Fow97] Fowler, M., *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
[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, www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm.
[Fow02] Fowler. M., "The New Methodology," June 2002, www.martinfowler.com/articles/newMethodology.html#N8B.
[Fow03] Fowler, M., et al., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.
[Fow04] Fowler, M., *UML Distilled*, 3d ed., Addison-Wesley, 2004.
[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.
[Fra03] Francois, A., "Software Architecture for Immersipresence," IMSC Technical Report IMSC-03-001, University of Southern California, December 2003, available at http://iris.usc.edu/~afrancoi/pdf/sai-tr.pdf.
[Fre80] Freeman, P., "The Context of Design," in *Software Design Techniques*, 3d ed. (P. Freeman and A. Wasserman, eds.), IEEE Computer Society Press, 1980, pp. 2–4.
[Fre90] Freedman, D. P., and G. M. Weinberg, *Handbook of Walkthroughs, Inspections and Technical Reviews*, 3d ed., Dorset House, 1990.
[Gag04] Gage, D., and J. McCormick, "We Did Nothing Wrong," *Baseline Magazine*, March 4, 2004,

available at www.baselinemag.com/article2/0,1397,1544403,00.asp.

[Gai95] Gaines, B., "Modeling and Forecasting the Information Sciences," Technical Report, University of Calgary, Calgary, Alberta, September 1995.

[Gam95] Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gar84] Garvin, D., "What Does 'Product Quality' Really Mean?" *Sloan Management Review*, Fall 1984, pp. 25–45.

[Gar87] Garvin D., *"Competing on the Eight Dimensions of Quality," Harvard Business Review*, November 1987, pp. 101–109. A summary is available at www.acm.org/crossroads/xrds6-4/software.html.

[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.

[Gar08] GartnerGroup, "Understanding Hype Cycles," 2008, available at www.gartner.com/pages/story.php.id.8795.s.8.jsp.

[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, Wirschaftsuniversität Wien, November 2001, downloadable from wwwai.wu-wien.ac.at/~hahsler/research/virlib_working2001/virlib/.

[Gil88] Gilb, T., *Principles of Software Project Management*, Addison-Wesley, 1988.

[Gil95] Gilb, T., "What We Fail to Do in Our Current Testing Culture," *Testing Techniques Newsletter* (online edition, ttn@soft.com), Software Research, January 1995.

[Gil06] Gillis, D., "Pattern-Based Design," tehan + lax blog, September 14, 2006, available at www.teehanlax.com/blog/?p=96.

[Gla98] Glass, R., "Defining Quality Intuitively," *IEEE Software*, May 1998, pp. 103–104, 107.

[Gla00] Gladwell, M., *The Tipping Point*, Back Bay Books, 2002.

[Gli07] Glinz, M., and R. Wieringa, "Stakeholders in Requirements Engineering," *IEEE Software*, vol. 24, no. 2, March–April 2007, pp. 18–20.

[Glu94] Gluch, D., "A Construct for Describing Software Development Risks," CMU/SEI-94-TR-14, Software Engineering Institute, 1994.

[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.

[Gon04] Gonzales, R., "Requirements Engineering," Sandia National Laboratories, a slide presentation, available at www.incose.org/enchantment/docs/04AprRequirementsEngineering.pdf.

[Gor02] Gordon, B., and M. Gordon, *The Complete Guide to Digital Graphic Design*, Watson-Guptill, 2002.

[Gor06] Gorton, I., *Essential Software Architecture*, Springer, 2006.

[Gra87] Grady, R. B., and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, 1987.

[Gra92] Grady, R. B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.

[Gra99] Grable, R., et al., "Metrics for Small Projects: Experiences at SED," *IEEE Software*, March 1999, pp. 21–29.

[Gra03] Gradecki, J., and N. Lesiecki, *Mastering AspectJ: Aspect-Oriented Programming in Java*, Wiley, 2003.

[Gru02] Grundy, J., "Aspect-Oriented Component Engineering," 2002, www.cs.auckland.ac.nz/~john-g/aspects.html.

[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.

[Gut93] Guttag, J. V., and J. J. Horning, *Larch: Languages and Tools for Formal Specification*, Springer-Verlag, 1993.

[Hac98] Hackos, J., and J. Redish, *User and Task Analysis for Interface Design*, Wiley, 1998.

[Hai02] Hailpern, B., and P. Santhanam, "Software Debugging, Testing and Verification," *IBM Systems Journal*, vol. 41, no. 1, 2002, available at www.research.ibm.com/journal/sj/411/hailpern.html.

[Hal77] Halstead, M., *Elements of Software Science*, North-Holland, 1977.

[Hal90] Hall, A., "Seven Myths of Formal Methods," *IEEE Software*, September 1990, pp. 11–20.
[Hal98] Hall, E. M., *Managing Risk: Methods for Software Systems Development*, Addison-Wesley, 1998.
[Ham90] Hammer, M., "Reengineer Work: Don't Automate, Obliterate," *Harvard Business Review*, July–August 1990, pp. 104–112.
[Han95] Hanna, M., "Farewell to Waterfalls," *Software Magazine*, May 1995, pp. 38–46.
[Har98a] Harmon, P., "Navigating the Distributed Components Landscape," *Cutter IT Journal.*, vol. 11, no. 2, December 1998, pp. 4–11.
[Har98b] 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.
[Her00] Herrmann, D., *Software Safety and Reliability*, Wiley-IEEE Computer Society Press, 2000.
[Het84] Hetzel, W., *The Complete Guide to Software Testing*, QED Information Sciences, 1984.
[Het93] Hetzel, W., *Making Software Measurement Work*, QED Publishing, 1993.
[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.
[Hig95] Higuera, R. P., "Team Risk Management," *CrossTalk*, U.S. Dept. of Defense, January 1995, pp. 2–4.
[Hig00] Highsmith, J., *Adaptive Software Development: An Evolutionary Approach to Managing Complex Systems*, Dorset House Publishing, 2000.
[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.
[Hil05] Hildreth, S., "Buggy Software: Up from a Low Quality Quagmire," *Computerworld*, July 25, 2005, available at www.computerworld.com/developmenttopics/development/story/0,10801,103378,00.html.
[Hil08] Hillside.net, *Patterns Catalog*, 2008, available at http://hillside.net/patterns/onlinepatterncatalog.htm.
[Hob06] Hoberman, S., *Data Modeling Made Simple*, Technics Publications, 2006.
[Hof00] Hofmeister, C., R. Nord, and D. Soni, *Applied Software Architecture*, Addison-Wesley, 2000.
[Hof01] Hofmann, C., et al., "Approaches to Software Architecture," 2001, downloadable from http://citeseer.nj.nec.com/84015.html.
[Hol06] Holzner, S., *Design Patterns for Dummies*, For Dummies Publishers, 2006.
[Hoo96] Hooker, D., "Seven Principles of Software Development," September 1996, available at http://c2.com/cgi/wikiSevenPrinciplesOfSoftwareDevelopment.
[Hop90] Hopper, M. D., "Rattling SABRE, New Ways to Compete on Information," *Harvard Business Review*, May–June 1990.
[Hor03] Horch, J., *Practical Guide to Software Quality Management*, 2d ed., Artech House, 2003.
[HPR02] Hypermedia Design Patterns Repository, 2002, available at www.designpattern.lu.unisi.ch/index.htm.
[Hum95] Humphrey, W., *A Discipline for Software Engineering*, Addison-Wesley, 1995.
[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 www.stsc.hill.af.mil/crosstalk/1998/apr/dimensions.asp.
[Hum00] Humphrey, W., *Introduction to the Team Software Process*, Addison-Wesley, 2000.
[Hun99] Hunt, A., D. Thomas, and W. Cunningham, *The Pragmatic Programmer*, Addison-Wesley, 1999.
[Hur83] Hurley, R. B., *Decision Tables in Software Engineering*, Van Nostrand-Reinhold, 1983.
[Hya96] Hyatt, L., and L. Rosenberg, "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality," NASA SATC, 1996, available at http://satc.gsfc.nasa.gov/support/STC_APR96/qualtiy/stc_qual.html.
[IBM81] "Implementing Software Inspections," course notes, IBM Systems Sciences Institute, IBM Corporation, 1981.
[IBM03] IBM, *Web Services Globalization Model*, 2003, available at www.ibm.com/developerworks/webservices/library/ws-global/.

[IEE93a] *IEEE Standards Collection: Software Engineering*, IEEE Standard 610.12-1990, IEEE, 1993.
[IEE93b] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1993.
[IEE00] IEEE Standard Association, IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems*, 2000, available at http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html.
[IFP01] *Function Point Counting Practices Manual*, Release 4.1.1, International Function Point Users Group, 2001, available from www.ifpug.org/publications/manual.htm.
[IFP05] Function Point Bibliography/Reference Library, International Function Point Users Group, 2005, available from www.ifpug.org/about/bibliography.htm.
[ISI08] iSixSigma, LLC, "New to Six Sigma: A Guide for Both Novice and Experiences Quality Practitioners," 2008, available at www.isixsigma.com/library/content/six-sigma-newbie.asp.
[ISO00] ISO 9001: 2000 Document Set, International Organization for Standards, 2000, www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html.
[ISO02] *Z Formal Specification Notation—Syntax, Type System and Semantics*, ISO/IEC 13568:2002, Intl. Standards Organization, 2002.
[ISO08] ISO SPICE, 2008, www.isospice.com/categories/SPICE-Project/.
[Ivo01] Ivory, M., R. Sinha, and M. Hearst, " Empirically Validated Web Page Design Metrics," ACM *SIGCHI'01*, March 31–April 4, 2001, available at http://webtango.berkeley.edu/papers/chi2001/.
[Jac75] Jackson, M. A., *Principles of Program Design*, Academic Press, 1975.
[Jac92] Jacobson, I., *Object-Oriented Software Engineering*, Addison-Wesley, 1992.
[Jac98] Jackman, M., "Homeopathic Remedies for Team Toxicity," *IEEE Software*, July 1998, pp. 43-45.
[Jac99] Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
[Jac02a] Jacobson, I., "A Resounding 'Yes' to Agile Processes—But Also More," *Cutter IT Journal*, vol. 15, no. 1, January 2002, pp. 18–24.
[Jac02b] Jacyntho, D., D. Schwabe, and G. Rossi, "An Architecture for Structuring Complex Web Applications," 2002, available at www2002.org/CDROM/alternate/478/.
[Jac04] Jacobson, I., and P. Ng, *Aspect-Oriented Software Development*, Addison-Wesley, 2004.
[Jal04] Jalote, P., et al., "Timeboxing: A Process Model for Iterative Software Development," *Journal of Systems and Software*, vol. 70, issue 2, 2004, pp. 117–127. Available at www.cse.iitk.ac.in/users/jalote/papers/Timeboxing.pdf.
[Jay94] Jaychandra, Y., *Re-engineering the Networked Enterprise*, McGraw-Hill, 1994.
[Jec06] Jech, T., *Set Theory*, 3d ed., Springer, 2006.
[Jon86] Jones, C., *Programming Productivity*, McGraw-Hill, 1986.
[Jon91] Jones, C., *Systematic Software Development Using VDM*, 2d ed., Prentice Hall, 1991.
[Jon96] Jones, C., "How Software Estimation Tools Work," *American Programmer*, vol. 9, no. 7, July 1996, pp. 19–27.
[Jon98] Jones, C., *Estimating Software Costs*, McGraw-Hill, 1998.
[Jon04] Jones, C., "Software Project Management Practices: Failure Versus Success," *CrossTalk*, October 2004. Available at www.stsc.hill.af.mil/crossTalk/2004/10/0410Jones.html.
[Joy00] Joy, B., "The Future Doesn't Need Us," *Wired*, vol. 8, no. 4, April 2000.
[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.
[Kan93] Kaner, C., J. Falk, and H. Q. Nguyen, *Testing Computer Software*, 2d ed., Van Nostrand-Reinhold, 1993.
[Kan95] Kaner, C., "Lawyers, Lawsuits, and Quality Related Costs, 1995, available at www.badsoftware.com/plaintif.htm.
[Kan01] Kaner, C., "Pattern: Scenario Testing" (draft), 2001, available at www.testing.com/test-patterns/patterns/pattern-scenario-testing-kaner.html.
[Kar94] Karten, N., *Managing Expectations*, Dorset House, 1994.
[Kau95] Kauffman, S., *At Home in the Universe*, Oxford, 1995.
[Kaz98] Kazman, R., et al., *The Architectural Tradeoff Analysis Method*, Software Engineering Institute, CMU/SEI-98-TR-008, July 1998.
[Kaz03] Kazman, R., and A. Eden, "Defining the Terms Architecture, Design, and Implementation," *news@sei interactive*, Software Engineering Institute, vol. 6, no. 1, 2003, available at www.sei.cmu.edu/news-at-sei/columns/the_architect/2003/1q03/architect-1q03.htm.
[Kei98] Keil, M., et al., "A Framework for Identifying Software Project Risks," *CACM*, vol. 41, no. 11, November 1998, pp. 76–83.

[Kel00] Kelly, D., and R. Oshana, "Improving Software Quality Using Statistical Techniques, Information and Software Technology," Elsevier, vol. 42, August 2000, pp. 801–807, available at www.eng.auburn.edu/~kchang/comp6710/readings/Improving_Quality_with_Statistical_Testing_InfoSoftTech_August2000.pdf.

[Ker78] Kernighan, B., and P. Plauger, *The Elements of Programming Style*, 2d ed., McGraw-Hill, 1978.

[Ker05] Kerievsky, J., *Industrial XP: Making XP Work in Large Organizations*, Cutter Consortium, Executive Report, vol. 6., no. 2, 2005, available at www.cutter.com/content-and-analysis/resource-centers/agile-project-management/sample-our-research/apmr0502.html.

[Kim04] Kim, E., "A Manifesto for Collaborative Tools," *Dr. Dobb's Journal*, May 2004, available at www.blueoxen.com/papers/0000D/.

[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.

[Kiz05] Kizza, J., *Computer Network Security*, Springer, 2005.

[Knu98] Knuth, D., *The Art of Computer Programming*, three volumes, Addison-Wesley, 1998.

[Kon02] Konrad, S., and B. Cheng, "Requirements Patterns for Embedded Systems," *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, IEEE, September 2002, pp. 127–136, downloadable from http://citeseer.ist.psu.edu/669258.html.

[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.

[Kra95] Kraul, R., and L. Streeter, "Coordination in Software Development," *CACM*, vol. 38, no. 3, March 1995, pp. 69–81.

[Kru05] Krutchen, P., "Software Design in a Postmodern Era," *IEEE Software*, vol. 22, no. 2, March–April, 2005, pp. 16–18.

[Kru06] Kruchten, P., H. Obbink, and J. Stafford (eds.), "Software Architectural" (special issue), *IEEE Software*, vol. 23, no. 2, March–April, 2006.

[Kur05] Kurzweil, R., *The Singularity Is Near*, Penguin Books, 2005.

[Kyb84] Kyburg, H. E., *Theory and Measurement*, Cambridge University Press, 1984.

[Laa00] Laakso, S., et al., "Improved Scroll Bars," *CHI 2000 Conf. Proc.*, ACM, 2000, pp. 97–98, available at www.cs.helsinki.fi/u/salaakso/patterns/.

[Lai02] Laitenberger, A., "A Survey of Software Inspection Technologies," in *Handbook on Software Engineering and Knowledge Engineering*, World Scientific Publishing Company, 2002.

[Lam01] Lam, W., "Testing E-Commerce Systems: A Practical Guide," *IEEE IT Pro*, March–April 2001, pp. 19–28.

[Lan01] Lange, M., "It's Testing Time! Patterns for Testing Software, June 2001, downloadable from www.testing.com/test-patterns/patterns/index.html.

[Lan02] Land, R., "A Brief Survey of Software Architecture," Technical Report, Dept. of Computer Engineering, Mälardalen University, Sweden, February 2002.

[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, downloadable from www.ece.utexas.edu/~perry/work/papers/feast1.pdf.

[Let01] Lethbridge, T., and R. Laganiere, *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*, McGraw-Hill, 2001.

[Let03a] Lethbridge, T., Personal communication on domain analysis, May 2003.

[Let03b] Lethbridge, T., Personal communication on software metrics, June 2003.

[Lev95] Leveson, N. G., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[Lev01] Levinson, M., "Let's Stop Wasting $78 billion a Year," *CIO Magazine*, October 15, 2001, available at www.cio.com/archive/101501/wasting.html.

[Lew06] Lewicki, R., B. Barry, and D. Saunders, *Essentials of Negotiation*, McGraw-Hill, 2006.

[Lie03] Lieberherr, K., "Demeter: Aspect-Oriented Programming," May 2003, available at www.ccs.neu.edu/home/lieber/LoD.html.

[Lin79] Linger, R., H. Mills, and B. Witt, *Structured Programming*, 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.

[Lis88] Liskov, B., "Data Abstraction and Hierarchy," *SIGPLAN Notices*, vol. 23, no. 5, May 1988.

[Liu98] Liu, K., et al., "Report on the First SEBPC Workshop on Legacy Systems," Durham University, February 1998, available at www.dur.ac.uk/CSM/SABA/legacy-wksp1/report.html.

[Lon02] Longstreet, D., "Fundamental of Function Point Analysis," Longstreet Consulting, Inc., 2002, available at www.ifpug.com/fpafund.htm.

[Lor94] Lorenz, M., and J. Kidd, *Object-Oriented Software Metrics*, Prentice Hall, 1994.

[Maa07] Maassen, O., and S. Stelting, "Creational Patterns: Creating Objects in an OO System," 2007, available at www.informit.com/articles/article.asp?p=26452&rl=1.

[Man81] Mantai, M., "The Effect of Programming Team Structures on Programming Tasks," *CACM*, vol. 24, no. 3, March 1981, pp. 106–113.

[Man97] Mandel, T., *The Elements of User Interface Design*, Wiley, 1997.

[Mar94] Marick, B., *The Craft of Software Testing*, Prentice Hall, 1994.

[Mar00] Martin, R., "Design Principles and Design Patterns," downloadable from www .objectmentor.com, 2000.

[Mar01] Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering*, 2d ed., Wiley, 2001.

[Mar02] Marick, B., "Software Testing Patterns," 2002, 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.

[McC77] McCall, J., P. Richards, and G. Walters, "Factors in Software Quality," three volumes, NTIS AD-A049-014, 015, 055, November 1977.

[McC94] McCabe, T. J., and A. H. Watson, "Software Complexity," *CrossTalk*, vol. 7, no. 12, December 1994, pp. 5–9.

[McC96] McConnell, S., "Best Practices: Daily Build and Smoke Test", *IEEE Software*, vol. 13, no. 4, July 1996, pp. 143–144.

[McC98] McConnell, S., *Software Project Survival Guide*, Microsoft Press, 1998.

[McC99] McConnell, S., "Software Engineering Principles," *IEEE Software*, vol. 16, no. 2, March–April 1999, available at www.stevemcconnell.com/ieeesoftware/eic04.htm.

[McC04] McConnell, S., *Code Complete*, Microsoft Press, 2004.

[McC05] McCrory, A., "Ten Technologies to Watch in 2006," SeachCIO.com, October 27, 2005, available at http://searchcio.techtarget.com/originalContent/0,289142,sid19_gci1137889,00 .html.

[McDE93] McDermid, J., and P. Rook, "Software Development Process Models," in *Software Engineer's Reference Book*, CRC Press, 1993, pp. 15/26–15/28.

[McG91] McGlaughlin, R., "Some Notes on Program Design," *Software Engineering Notes*, vol. 16, no. 4, October 1991, pp. 53–54.

[McG94] McGregor, J. D., and T. D. Korson, "Integrated Object-Oriented Testing and Development Processes," *Communications of the ACM*, vol. 37, no. 9, September, 1994, pp. 59–77.

[Men01] Mendes, E., N. Mosley, and S. Counsell, "Estimating Design and Authoring Effort," *IEEE Multimedia*, vol. 8, no. 1, January–March 2001, pp. 50–57.

[Mer93] Merlo, E., et al., "Reengineering User Interfaces," *IEEE Software*, January 1993, pp. 64–73.

[Mic08] *Microsoft Accessibility Technology for Everyone*, 2008, available at www.microsoft.com/ enable/.

[Mic04] Microsoft, "Prescriptive Architecture: Integration and Patterns," *MSDN*, May 2004, available at http://msdn2.microsoft.com/en-us/library/ms978700.aspx.

[Mic07] Microsoft, "Patterns and Practices," *MSDN*, 2007, available at http://msdn2.microsoft .com/en-us/library/ms998478.aspx.

[Mil72] Mills, H. D., "Mathematical Foundations for Structured Programming," Technical Report FSC 71-6012, IBM Corp., Federal Systems Division, Gaithersburg, MD, 1972.

[Mil77] Miller, E., "The Philosophy of Testing," in *Program Testing Techniques*, IEEE Computer Society Press, 1977, pp. 1–3.

[Mil87] Mills, H. D., M. Dyer, and R. Linger, "Cleanroom Software Engineering," *IEEE Software*, September 1987, pp. 19–25.

[Mil88] Mills, H. D., "Stepwise Refinement and Verification in Box Structured Systems," *Computer*, vol. 21, no. 6, June 1988, pp. 23–35.

[Mil00a] Miller, E., "WebSite Testing," 2000, available at www.soft.com/eValid/Technology/ White.Papers/website.testing.html.

[Mil00b] Mili, A., and R, Cowan, "Software Engineering Technology Watch," April 6, 2000, available at www.serc.net/projects/TechWatch/NSF%20TechWatch%20Proposal.htm.

[Min95] Minoli, D., *Analyzing Outsourcing*, McGraw-Hill, 1995.

[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.
[Mor05] Morales, A., "The Dream Team," *Dr. Dobbs Portal*, March 3, 2005, available at www.ddj.com/dept/global/184415303.
[Mus87] Musa, J. D., A. Iannino, and K. Okumoto, *Engineering and Managing Software with Reliability Measures*, McGraw-Hill, 1987.
[Mus93] Musa, J., "Operational Profiles in Software Reliability Engineering," *IEEE Software*, March 1993, pp. 14–32.
[Mut03] Mutafelija, B., and H. Stromberg, *Systematic Process Improvement Using ISO 9001:2000 and CMMI*, Artech, 2003.
[Mye78] Myers, G., *Composite Structured Design*, Van Nostrand, 1978.
[Mye79] Myers, G., *The Art of Software Testing*, Wiley, 1979.
[NAS07] NASA, *Software Risk Checklist*, Form LeR-F0510.051, March 2007, downloadable from http://osat-ext.grc.nasa.gov/rmo/spa/SoftwareRiskChecklist.doc.
[Nau69] Naur, P., and B. Randall (eds.), *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*, NATO, 1969.
[Ngu00] Nguyen, H., "Testing Web-Based Applications," *Software Testing and Quality Engineering*, May–June 2000, available at www.stqemagazine.com.
[Ngu01] Nguyen, H., *Testing Applications on the Web*, Wiley, 2001.
[Ngu06] Nguyen, T., "Model-Based Version and Configuration Management for a Web Engineering Lifecycle," *Proc. 15th Intl. World Wide Web Conf.*, Edinburg, Scotland, 2006, download from www2006.org/programme/item.php?id=4552.
[Nie92] Nierstrasz, O., S. Gibbs, and D. Tsichritzis, "Component-Oriented Software Development," *CACM*, vol. 35, no. 9, September 1992, pp. 160–165.
[Nie94] Nielsen, J., and J. Levy, "Measuring Usability: Preference vs. Performance," *CACM*, vol. 37, no. 4, April 1994, pp. 65–75.
[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.
[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 Graduate School, Monterey, CA, June 2000, downloadable from www.dodccrp.org/2000CCRTS/cd/html/pdf_papers/Track_4/075.pdf.
[Nor70] Norden, P., "Useful Tools for Project Management" in *Management of Production*, M. K. Starr (ed.), Penguin Books, 1970.
[Nor86] Norman, D. A., "Cognitive Engineering," in *User Centered Systems Design*, Lawrence Earlbaum Associates, 1986.
[Nor88] Norman, D., *The Design of Everyday Things*, Doubleday, 1988.
[Nov04] Novotny, O., "Next Generation Tools for Object-Oriented Development," *The Architecture Journal*, January 2005, available at http://msdn2.microsoft.com/en-us/library/aa480062.aspx.
[Noy02] Noyes, B., "Rugby, Anyone?" *Managing Development* (an online publication of Fawcette Technical Publications), June 2002, www.fawcette.com/resources/managingdev/methodologies/scrum/.
[Off02] Offutt, J., "Quality Attributes of Web Software Applications," *IEEE Software*, March–April 2002, pp. 25–32.
[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.
[Ols06] Olsen, G., "From COM to Common," *Component Technologies*, ACM, vol. 4, no. 5, June 2006, available at http://acmqueue.com/modules.php?name=Content&pa=showpage&pid=394.
[OMG03a] Object Management Group, *OMG Unified Modeling Language Specification*, version 1.5, March 2003, available from www.rational.com/uml/resources/documentation/.
[OMG03b] "Object Constraint Language Specification," in *Unified Modeling Language*, v2.0, Object Management Group, September 2003, downloadable from www.omg.org.
[Orf99] Orfali, R., D. Harkey, and J. Edwards, *Client/Server Survival Guide*, 3d ed., Wiley, 1999.
[Osb90] Osborne, W. M., and E. J. Chikofsky, "Fitting Pieces to the Maintenance Puzzle," *IEEE Software*, January 1990, pp. 10–11.
[OSO08] OpenSource.org, 2008, available at www.opensource.org/.
[Pag85] Page-Jones, M., *Practical Project Management*, Dorset House, 1985, p. vii.

[Pal02] Palmer, S., and J. Felsing, *A Practical Guide to Feature Driven Development*, Prentice Hall, 2002.
[Par72] Parnas, D. L., "On Criteria to Be Used in Decomposing Systems into Modules," *CACM*, vol. 14, no. 1, April 1972, pp. 221–227.
[Par96a] Pardee, W., *To Satisfy and Delight Your Customer*, Dorset House, 1996.
[Par96b] 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.
[Pat07] Patton, J., "Understanding User Centricity," *IEEE Software*, vol. 24, no. 6, November–December, 2007, pp. 9–11.
[Pau94] Paulish, D., and A. Carleton, "Case Studies of Software Process Improvement Measurement," *Computer*, vol. 27, no. 9, September 1994, pp. 50–57.
[PCM03] "Technologies to Watch," *PC Magazine*, July 2003, available at www.pcmag.com/article2/0,4149,1130591,00.asp.
[Per74] Persig, R., *Zen and the Art of Motorcycle Maintenance*, Bantam Books, 1974.
[Pet06] Pethokoukis, J., "Small Biz Watch: Future Business Trends," *U.S. News & World Report*, January 20, 2006, available at www.usnews.com/usnews/biztech/articles/060120/20sbw.htm.
[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.
[Phi98] Phillips, D., *The Software Project Manager's Handbook*, IEEE Computer Society Press, 1998.
[Phi02] Phillips, M., "CMMI V1.1 Tutorial.," April 2002, available at www.sei.cmu.edu/cmmi/.
[Pol45] Polya, G., *How to Solve It*, Princeton University Press, 1945.
[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.
[Pop03] Poppendieck, M., and T. Poppendieck, *Lean Software Development*, Addison-Wesley, 2003.
[Pop06a] Poppendeick, LLC, *Lean Software Development*, available at www.poppendieck.com/.
[Pop06b] Poppendieck, M., and T. Poppendieck, *Implementing Lean Software Development*, Addison-Wesley, 2006.
[Pop08] Popcorn, F., *Faith Popcorn's Brain Reserve*, 2008, available at www.faithpopcorn.com/.
[Pot04] Potter, M., *Set Theory and Its Philosophy: A Critical Introduction*, Oxford University Press, 2004.
[Pow98] Powell, T., *Web Site Engineering*, Prentice Hall, 1998.
[Pow02] Powell, T., *Web Design*, 2d ed., McGraw-Hill/Osborne, 2002.
[Pre94] Premerlani, W., and M. Blaha, "An Approach for Reverse Engineering of Relational Databases," *CACM*, vol. 37, no. 5, May 1994, pp. 42–49.
[Pre88] Pressman, R., *Making Software Engineering Happen*, Prentice Hall, 1988.
[Pre05] Pressman, R., *Adaptable Process Model*, revision 2.0, R. S. Pressman & Associates, 2005, available at www.rspa.com/apm/index.html.
[Pre08] Pressman, R., and D. Lowe, *Web Engineering: A Practitioner's Approach*, McGraw-Hill, 2008.
[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.
[Pyz03] Pyzdek, T., *The Six Sigma Handbook*, McGraw-Hill, 2003.
[QAI08] *A Software Engineering Curriculum*, QAI, 2008, information can be obtained at www.qaieschool.com/innerpages/offer.asp.
[QSM02] "QSM Function Point Language Gearing Factors," Version 2.0, Quantitative Software Management, 2002, www.qsm.com/FPGearing.html.
[Rad02] Radice, R., *High-Quality Low Cost Software Inspections*, Paradoxicon Publishing, 2002.
[Rai06] Raiffa, H., *The Art and Science of Negotiation*, Belknap Press, 2005.
[Ree99] Reel, J. S., "Critical Success Factors in Software Projects," *IEEE Software*, May 1999,

pp. 18–23.
[Ric01] Ricadel, A., "The State of Software Quality," *InformationWeek*, May 21, 2001, available at www.informationweek.com/838/quality.htm.
[Ric04] Rico, D., *ROI of Software Process Improvement*, J. Ross Publishing, 2004. A summary article can be found at http://davidfrico.com/rico03a.pdf.
[Roc94] Roche, J. M., "Software Metrics and Measurement Principles," *Software Engineering Notes*, ACM, vol. 19, no. 1, January 1994, pp. 76–85.
[Roc06] *Graphic Design That Works*, Rockport Publishers, 2006.
[Roe00] Roetzheim, W., "Estimating Internet Development," *Software Development*, August 2000, available at www.sdmagazine.com/documents/s=741/sdm0008d/0008d.htm.
[Roo96] Roos, J., "The Poised Organization: Navigating Effectively on Knowledge Landscapes," 1996, available at www.imd.ch/fac/roos/paper_po.html.
[Ros75] Ross, D., J. Goodenough, and C. Irvine, "Software Engineering: Process, Principles and Goals," *IEEE Computer*, vol. 8, no. 5, May 1975.
[Ros04] Rosenhainer, L., "Identifying Crosscutting Concerns in Requirements Specifications," 2004, available at http://trese.cs.utwente.nl/workshops/oopsla-early-aspects-2004/Papers/Rosenhainer.pdf.
[Rou02] Rout, T (project manager), *SPICE: Software Process Assessment—Part 1: Concepts and Introductory Guide*, 2002, downloadable from www.sqi.gu.edu.au/spice/suite/download.html.
[Roy70] Royce, W. W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proc. WESCON*, August 1970.
[Roz05] Rozanski, N., and E. Woods, *Software Systems Architecture*, Addison-Wesley, 2005.
[Rub88] Rubin, T., *User Interface Design for Computer Systems*, Halstead Press (Wiley), 1988.
[Rum91] Rumbaugh, J., et al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991.
[Sar06] Sarwate, A., "Hot or Not: Web Application Vulnerabilities," *SC Magazine*, December 27, 2006, available at http://scmagazine.com/us/news/article/623765/hot-not-web-application-vulnerabilities.
[Sca00] Scacchi, W., "Understanding Software Process Redesign Using Modeling, Analysis, and Simulation," *Software Process Improvement and Practice*, Wiley, 2000, pp. 185–195, downloadable at www.ics.uci.edu/~wscacchi/Papers/Software_Process_Redesign/SPIP-ProSim99.pdf.
[Sce02] Sceppa, D., *Microsoft ADO.NET*, Microsoft Press, 2002.
[Sch95] Schwabe, D., and G. Rossi, "The Object-Oriented Hypermedia Design Model," *CACM*, vol. 38, no. 8, August 1995, pp. 45–46.
[Sch96] Schorsch, T., "The Capability Im-Maturity Model," *CrossTalk*, November 1996, available at www.stsc.hill.af.mil/crosstalk/1996/11/xt96d11h.asp.
[Sch98a] Schneider, G., and J. Winters, *Applying Use Cases*, Addison-Wesley, 1998.
[Sch98b] Schwabe, D., and G. Rossi, "Developing Hypermedia Applications Using OOHDM," *Proc. Workshop on Hypermedia Development Process, Methods and Models, Hypertext '98*, 1998, downloadable from http://citeseer.nj.nec.com/schwabe98developing.html.
[Sch98c] Schulmeyer, G. C., and J. I. McManus (eds.), *Handbook of Software Quality Assurance*, 3d ed., Prentice Hall, 1998.
[Sch99] Schneidewind, N., "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics," *IEEE Trans. SE*, vol. 25, no. 6, November–December 1999, pp. 768–781, downloadable from www.dacs.dtic.mil/topics/reliability/IEEETrans.pdf.
[Sch01a] Schwabe, D., G. Rossi, and Barbosa, S., "Systematic Hypermedia Application Design Using OOHDM," 2001, available at www-di.inf.puc-rio.br/~schwabe/HT96WWW/section1.html.
[Sch01b] Schwaber, K., and M. Beedle, *Agile Software Development with SCRUM*, Prentice Hall, 2001.
[Sch02] Schwaber, K., "Agile Processes and Self-Organization," Agile Alliance, 2002, www.aanpo.org/articles/index.
[Sch03] Schlickman, J., *ISO 9001: 2000 Quality Management System Design*, Artech House Publishers, 2003.
[Sch06] Schmidt, D., "Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 2, February 2006, pp. 25–31.
[SDS08] Spice Document Suite, "The SPICE and ISO Document Suite," ISO-Spice, 2008, available at www.isospice.com/articles/9/1/SPICE-Project/Page1.html.
[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.

[SEE03] The Software Engineering Ethics Research Institute, "UCITA Updates," 2003, available at http://seeri.etsu.edu/default.htm.

[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 www.sei.cmu.edu/publications/documents/00.reports/00tr009.html.

[SEI02] "Maintainability Index Technique for Measuring Program Maintainability," SEI, 2002, available at www.sei.cmu.edu/str/descriptions/mitmpm_body.html.

[SEI08] "The Ideal Model," Software Engineering Institute, 2008, available at www.sei.cmu.edu/ideal/.

[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.

[Sha96] Shaw, M., and D. Garlan, *Software Architecture*, Prentice Hall, 1996.

[Sha05] Shalloway, A., and J. Trott, *Design Patterns Explained*, 2d ed., Addison-Wesley, 2005.

[Shn80] Shneiderman, B., *Software Psychology*, Winthrop Publishers, 1980, p. 28.

[Shn04] Shneiderman, B., and C. Plaisant, *Designing the User Interface*, 4th ed., Addison-Wesley, 2004.

[Sho83] Shooman, M. L., *Software Engineering*, McGraw-Hill, 1983.

[Sim05] Simsion, G., and G. Witt, *Data Modeling Essentials*, 3d ed., Morgan Kaufman, 2005.

[Sin99] Singpurwalla, N., and S. Wilson, *Statistical Methods in Software Engineering: Reliability and Risk*, Springer-Verlag, 1999.

[Smi99] Smith, J., "The Estimation of Effort Based on Use Cases," Rational Software Corp., 1999, downloadable from www.rational.com/media/whitepapers/finalTP171.PDF.

[Smi05] Smith, D, *Reliability, Maintainability and Risk*, 7th ed., Butterworth-Heinemann, 2005.

[Sne95] Sneed, H., "Planning the Reengineering of Legacy Systems," *IEEE Software*, January 1995, pp. 24–25.

[Sne03] Snee, R., and R. Hoerl, *Leading Six Sigma*, Prentice Hall, 2003.

[Sol99] van Solingen, R., and E. Berghout, *The Goal/Question/Metric Method*, McGraw-Hill, 1999.

[Som97] Somerville, I., and P. Sawyer, *Requirements Engineering*, Wiley, 1997.

[Som05] Somerville, I., "Integrating Requirements Engineering: A Tutorial," *IEEE Software*, vol. 22, no. 1, January–February 2005, pp. 16–23.

[SPI99] "SPICE: Software Process Assessment, Part 1: Concepts and Introduction," Version 1.0, ISO/IEC JTC1, 1999.

[Spl01] Splaine, S., and S. Jaskiel, *The Web Testing Handbook*, STQE Publishing, 2001.

[Spo02] Spolsky, J, "The Law of Leaky Abstractions," November 2002, available at www.joelonsoftware.com/articles/LeakyAbstractions.html.

[Sri01] Sridhar, M., and N. Mandyam, "Effective Use of Data Models in Building Web Applications," 2001, available at www2002.org/CDROM/alternate/698/.

[SSO08] Software-Supportability.org, www.software-supportability.org/, 2008.

[Sta97] Stapleton, J., *DSDM—Dynamic System Development Method: The Method in Practice*, Addison-Wesley, 1997.

[Sta97b] Statz, J., D. Oxley, and P. O'Toole, "Identifying and Managing Risks for Software Process Improvement," *CrossTalk*, April 1997, available at www.stsc.hill.af.mil/crosstalk/1997/04/identifying.asp.

[Ste74] Stevens, W., G. Myers, and L. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, 1974, pp. 115–139.

[Ste93] Stewart, T. A., "Reengineering: The Hot New Managing Tool," *Fortune*, August 23, 1993, pp. 41–48.

[Ste99] Stelzer, D., and W. Mellis, "Success Factors of Organizational Change in Software Process Improvement," *Software Process Improvement and Practice*, vol. 4, no. 4, Wiley, 1999, downloadable from www.systementwicklung.uni-koeln.de/forschung/artikel/dokumente/successfactors.pdf.

[Ste03] Stephens, M., and D. Rosenberg, *Extreme Programming Refactored*, Apress, 2003.

[Sto05] Stone, D., et al., *User Interface Design and Evaluation*, Morgan Kaufman, 2005.

[Tai89] Tai, K. C., "What to Do Beyond Branch Testing," *ACM Software Engineering Notes*, vol. 14, no. 2, April 1989, pp. 58–61.

[Tay90] Taylor, D., *Object-Oriented Technology: A Manager's Guide*, Addison-Wesley, 1990.

[Tha97] Thayer, R. H., and M. Dorfman, *Software Requirements Engineering*, 2d ed., IEEE Computer Society Press, 1997.

[The01] Thelin, T., H. Petersson, and C. Wohlin, "Sample Driven Inspections," *Proc. of Workshop*

on Inspection in Software Engineering (WISE'01), Paris, France, July 2001, pp. 81–91, downloadable from http://www.cas.mcmaster.ca/wise/wise01/ThelinPeterssonWohlin.pdf.

[Tho92] Thomsett, R., "The Indiana Jones School of Risk Management," *American Programmer*, vol. 5, no. 7, September 1992, pp. 10–18.

[Tic05] *TickIT*, 2005, www.tickit.org/.

[Tid02] Tidwell, J., "IU Patterns and Techniques," May 2002, available at http://time-tripper.com/uipatterns/index.html.

[Til93] Tillmann, G., *A Practical Guide to Logical Data Modeling*, McGraw-Hill, 1993.

[Til00] Tillman, H., "Evaluating Quality on the Net," Babson College, May 30, 2000, available at www.hopetillman.com/findqual.html#2.

[Tog01] Tognozzi, B., "First Principles," *askTOG*, 2001, available at www.asktog.com/basics/firstPrinciples.html.

[Tra95] Tracz, W., "Third International Conference on Software Reuse—Summary," *ACM Software Engineering Notes*, vol. 20, no. 2, April 1995, pp. 21–22.

[Tre03] Trivedi, R, *Professional Web Services Security*, Wrox Press, 2003.

[Tri05] Tricker, R., and B. Sherring-Lucas, *ISO 9001: 2000 In Brief*, 2d ed., Butterworth-Heinemann, 2005.

[Tyr05] Tyree, J., and A. Akerman, "Architectural Decisions: Demystifying Architecture," *IEEE Software*, vol. 22, no. 2, March–April, 2005.

[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

[Ull97] Ullman, E., *Close to the Machine: Technophilia and Its Discontents*, City Lights Books, 2002.

[UML03] The UML Café, "Customers Don't Print Themselves," May 2003, available at www.theumlcafe.com/a0079.htm.

[Uni03] Unicode, Inc., *The Unicode Home Page*, 2003, available at www.unicode.org/.

[USA87] *Management Quality Insight*, AFCSP 800-14 (U.S. Air Force), January 20, 1987.

[Vac06] Vacca, J., *Practical Internet Security*, Springer, 2006.

[Van89] Van Vleck, T., "Three Questions About Each Bug You Find," *ACM Software Engineering Notes*, vol. 14, no. 5, July 1989, pp. 62–63.

[Van02] Van Steen, M., and A. Tanenbaum, *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.

[Ven03] Venners, B., "Design by Contract: A Conversation with Bertrand Meyer," *Artima Developer*, December 8, 2003, available at www.artima.com/intv/contracts.html.

[Wal03] Wallace, D., I. Raggett, and J. Aufgang, *Extreme Programming for Web Projects*, Addison-Wesley, 2003.

[War74] Warnier, J. D., *Logical Construction of Programs*, Van Nostrand-Reinhold, 1974.

[War07] Ward, M., "Using VoIP Software Building zBlocks—A Look at the Choices," TMNNet, 2007, available at www.tmcnet.com/voip/0605/featurearticle-using-voip-software-building-blocks.htm.

[Web05] Weber, S., *The Success of Open Source*, Harvard University Press, 2005.

[Wei86] Weinberg, G., *On Becoming a Technical Leader*, Dorset House, 1986.

[Wel99] Wells, D., "XP—Unit Tests," 1999, available at www.extremeprogramming.org/rules/unittests.html.

[Wel01] vanWelie, M., "Interaction Design Patterns," 2001, available at www.welie.com/patterns/.

[Whi95] Whittle, B., "Models and Languages for Component Description and Reuse," *ACM Software Engineering Notes*, vol. 20, no. 2, April 1995, pp. 76–89.

[Whi97] Whitmire, S., *Object-Oriented Design Measurement*, Wiley, 1997.

[Wie02] Wiegers, K., *Peer Reviews in Software*, Addison-Wesley, 2002.

[Wie03] Wiegers, K., *Software Requirements*, 2d ed., Microsoft Press, 2003.

[Wil93] Wilde, N., and R. Huitt, "Maintaining Object-Oriented Software," *IEEE Software*, January 1993, pp. 75–80.

[Wil97] Williams, R. C, J. A. Walker, and A. J. Dorofee, "Putting Risk Management into Practice," *IEEE Software*, May 1997, pp. 75–81.

[Wil99] Wilkens, T. T., "Earned Value, Clear and Simple," Primavera Systems, April 1, 1999, p. 2.

[Wil00] Williams, L., and R. Kessler, "All I Really Need to Know about Pair Programming I Learned in Kindergarten," *CACM*, vol. 43, no. 5, May 2000, available at http://collaboration.csc.ncsu.edu/laurie/Papers/Kindergarten.PDF.

[Wil05] Willoughby, M., "Q&A: Quality Software Means More Secure Software," *Computerworld*,

March 21, 2005, available at www.computerworld.com/securitytopics/security/story/0,10801,91316,00.html.

[Win90] Wing, J. M., "A Specifier's Introduction to Formal Methods," *IEEE Computer*, vol. 23, no. 9, September 1990, pp. 8–24.

[Wir71] Wirth, N., "Program Development by Stepwise Refinement," *CACM*, vol. 14, no. 4, 1971, pp. 221–227.

[Wir90] Wirfs-Brock, R., B. Wilkerson, and L. Weiner, *Designing Object-Oriented Software*, Prentice Hall, 1990.

[WMT02] Web Mapping Testbed Tutorial., 2002, available at www.webmapping.org/vcgdocuments/vcgTutorial/.

[Woh94] Wohlin, C., and P. Runeson, "Certification of Software Components," *IEEE Trans. Software Engineering*, vol. SE-20, no. 6, June 1994, pp. 494–499.

[Wor04] World Bank, *Digital Technology Risk Checklist*, 2004, downloadable from www.moonv6.org/lists/att-0223/WWBANK_Technology_Risk_Checklist_Ver_6point1.pdf.

[W3C03] World Wide Web Consortium, *Web Content Accessibility Guidelines*, 2003, available at www.w3.org/TR/2003/WD-WCAG20-20030624/.

[Yac03] Yacoub, S., et al., *Pattern-Oriented Analysis and Design*, Addison-Wesley, 2003.

[You75] Yourdon, E., *Techniques of Program Structure and Design*, Prentice Hall, 1975.

[You79] Yourdon, E., and L. Constantine, *Structured Design*, Prentice Hall, 1979.

[You95] Yourdon, E., "When Good Enough Is Best," *IEEE Software*, vol. 12, no. 3, May 1995, pp. 79–81.

[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.

[Zah94] Zahniser, R., "Timeboxing for Top Team Performance," *Software Development*, March 1994, pp. 35–38.

[Zha98] Zhao, J, "On Assessing the Complexity of Software Architectures," *Proc. Intl. Software Architecture Workshop*, ACM, Orlando, FL, 1998, pp. 163–167.

[Zha02] Zhao, H., "Fitt's Law: Modeling Movement Time in HCI," *Theories in Computer Human Interaction*, University of Maryland, October 2002, available at www.cs.umd.edu/class/fall2002/cmsc838s/tichi/fitts.html.

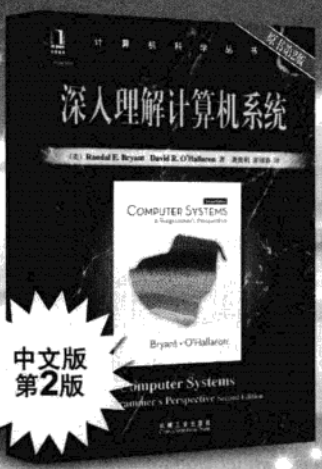
[Zul92] Zultner, R., "Quality Function Deployment for Software: Satisfying Customers," *American Programmer*, February 1992, pp. 28–41.

[Zus90] Zuse, H., *Software Complexity: Measures and Methods*, DeGruyter, 1990.

[Zus97] Zuse, H., *A Framework of Software Measurement*, DeGruyter, 1997.

计算机科学丛书经典推荐

精心的遴选与移译，擘划研究的范畴，揭示学术的源变



深入理解计算机系统（原书第2版）
中文版：978-7-111-32133-0
作者：Randal E. Bryant
David R. O'Hallaron
定价：99.00元

英文版：978-7-111-32631-1
定价：128.00元

作者：Alfred V. Aho 等
译者：赵建华 等
中文版：7-111-25121-7
定价：89.00
本科教学版 7-111-26929-8
定价：55.00

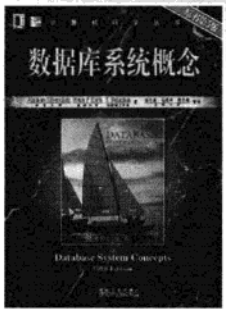
英文版：978-7-111-32674-8
定价：78.00元

作者：William Stallings
译者：陈向群 等
书号：978-7-111-30426-5
定价：69.00

■操作系统领域的经典之作，
全球著名高校竞相采用

作者：Roger S. Pressman
译者：郑人杰 等
中文版第7版：978-7-111-33581-8
定价：79.00
英文版第7版：978-7-111-31871-2
定价：75.00
本科教学版：978-7-111-23443-2
定价：49.00

■全球上百所大学和学院采用，
最受欢迎的软件工程指南



作者：Abraham Silberschatz 等
译者：杨冬青 等
中文版：7-111-19687-2
定价：69.00
本科教学版：978-7-111-23422-7
定价：45.00

■数据库系统方面的经典教材，
被美誉为“帆船书”

作者：John L. Hennessy 等
译者：郑纬民 等
中文版：7-111-20214-1
定价：75.00
英文版：7-111-19339-3
定价：85.00
中文版第4版预计2011年出版

■本书采用MIPS处理器作为展示
计算机硬件技术基本核心

作者：James F. Kurose 等
译者：陈鸣
中文版第4版：978-7-111-16505-7
定价：66.00

■全球上百所大学和学院采用，
被译为10多种语言并被世界上
数以万计的学生和专业人士采用

更多经典请点击 www.hzbook.com 查询

[General Information]
书名=软件工程　实践者的研究方法　原书第7版
作者=（美）普雷斯曼著
页数=642
出版社=北京市：机械工业出版社
出版日期=2011.03
SS号=12764337
DX号=000008075967
URL=http://book1.duxiu.com/bookDetail.jsp?dxNumber=000008075967&d=1C03B6E032875EA3CD16741EE07CC172