全美经典
学习指导系列

软件工程
习题与解答

Schaum's Outlines of Software Engineering

最佳的复习资料，实用的辅助教材

与国外高校计算机水平保持同步

为考研和出国深造奠定坚实基础

(美) David Gustafson 著
钟鸣 王君 等译

本系列丛书全球销售超过3000万册！

Mc Graw Hill Education
机械工业出版社 China Machine Press

全美经典学习指导系列是一套快捷有效的学习指南，该套丛书针对各专业的技术重点提供了数百个实例、习题及答案。通过这些实战练习，不但可以洞悉各门技术精髓，而且能够使考试成绩大幅攀升，更会助你与国外大学生的计算机水平看齐，为将来考研或出国深造奠定坚实基础。

全美经典学习指导系列深得高校学生的喜爱。由于有了这套丛书，在历年的专业考试中，成千上万的学生获得了优异成绩。

想成为一名优等生吗？——请选择全美经典学习指导系列！

如果时间不许却想成绩骄人，这本书可以助你：

- 通过具体范例解决疑难问题
- 考前快速强化
- 迅速找到答案
- 快捷而高效地学习
- 迅速掌握技术重点，无需翻阅冗长的教科书

全美经典学习指导系列以方便快捷的形式提供了考生需要了解的信息，同时不致使你淹没在不必要的细节当中。另外，还可以通过大量的编程练习来测试所学的技巧。该丛书可以与任何教材配合使用，使学生们能够根据各自的进度来学习，从而获得事半功倍的效果！**全美经典学习指导系列**的内容系统而完备，是毕业考试和专业考试的理想参考书。

本书内容包括：

- 计算机软件开发概述
- 软件工程技术和理论的详细解释
- 软件工程学科的极佳教辅材料
- 大量的例子和习题将帮助你解决实际的软件工程问题

如果想获得优异成绩并且能够全面掌握软件工程原理，本书是不可或缺的最佳辅导老师。

C++ 编程习题与解答
C++ 编程习题与解答 (英文版 · 第2版)
Java 编程习题与解答
Java 编程习题与解答 (英文版)
SQL 编程习题与解答
Visual Basic 编程习题与解答
操作系统习题与解答
操作系统习题与解答 (英文版)
关系数据库习题与解答
关系数据库习题与解答 (英文版)
计算机导论习题与解答
计算机导论习题与解答 (英文版)
计算机体系结构习题与解答 (英文版)
计算机图形学习题与解答
计算机图形学习题与解答 (英文版 · 第2版)
计算机网络习题与解答 (英文版)
软件工程习题与解答 (英文版)
软件工程习题与解答
数据结构习题与解答
——Java 语言描述
数据结构习题与解答
——Java 语言描述 (英文版)
数据结构习题与解答
——C++ 语言描述 (英文版)

McGraw-Hill
全球智慧中文化
http://www.mheducation.com

ISBN 7-111-12245-3

网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037
读者服务热线：(010)68995259, 68995264
读者服务信箱：hzedu@hzbook.com
http://www.hzbook.com

ISBN 7-111-12245-3/TP · 2706
定价：29.00元

全美经典
学习指导系列

软件工程习题与解答

Schaum's Outlines of Software Engineering

(美) David Gustafson 著
钟鸣 王君 等译

机械工业出版社
China Machine Press

本书以简明扼要的语言介绍了软件工程的基本概念和基本方法，涉及软件生命周期、软件过程模型和其他模型、软件项目管理、软件度量、风险分析和管理、软件质量保证、软件设计和软件测试等内容。通过阅读本书，读者能迅速了解软件工程的相关知识，并将这些概念和技术用于实际的系统开发中。本书内容全面、实例和习题极为丰富，是软件工程学科的一本极佳的教辅书籍。本书适合软件工程专业的本科生、研究生使用，也是急需了解软件工程知识的技术人员的入门书籍。

David Gustafson: Schaum's Outlines of Software Engineering (ISBN: 0-07-137794-8).

本书版权登记号：图字：01-2002-4821

图书在版编目（CIP）数据

软件工程习题与解答 /（美）古斯塔夫森（Gustafson, D.）著；钟鸣等译. －北京：机械工业出版社，2003.7

（全美经典学习指导系列）

书名原文：Schaum's Outlines of Software Engineering

ISBN 7-111-12245-3

Ⅰ.软… Ⅱ.①古…②钟… Ⅲ.软件工程－解题 Ⅳ.TP311.5-44

中国版本图书馆CIP数据核字（2003）第039045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：朱 劼

北京中加印刷有限公司印刷·新华书店北京发行所发行

2003年7月第1版第1次印刷

787mm×1092mm 1/16·14.25印张

印数：0 001-5 000册

定价：29.00元

译 者 序

软件工程是计算机应用科学中极为重要的一门学科。任何一个大中型信息系统的研制都离不开软件工程的指导。目前有关软件工程或信息系统工程的书籍很多，但这些书籍往往都是大部头，少则几十万字，多则上百万字。

当然，这些书籍之所以占用大量的篇幅是有充分理由的。因为软件工程要研究和处理的内容实在太多，所涉及的学科也不少，而软件工程的各个阶段都有许多问题需要研究。就系统需求分析来说，如何获得领域知识，如何与领域专家面谈，如何把握要点等就存在各种各样的方法。这些方法各有优缺点，适合不同的情况和不同的系统。如何对这些方法进行权衡，选择适合于要研制的系统的方法也需要进行很好的研究。而且，在获得领域知识后，如何表示也有诸多办法。在软件系统研制中如何保证质量？如何安排软件研制的进展？如何计算软件工作量？程序编制出来后，如何调试以保证其正确性？诸如此类的问题不胜枚举。可以说，针对软件工程的每个阶段都可以写出一本大部头的书籍。

但是，在目前这个快节奏的社会中，许多人往往没有时间读大部头的书籍。他们需要一本简明扼要，能很快掌握软件工程知识的书籍。他们学习软件工程的目的不是做学术性的研究，而是要用于工程实践。本书正是适合这种目的的一本书籍。阅读本书，能够迅速对软件工程这门学科有一个总体性的了解。通过完成书中的习题能很快了解和掌握软件工程的大多数概念和方法，并在实际的信息系统研制中加以运用。

本书是作者集25年给研究生、本科生讲授软件工程的教学经验编写而成的，特点是简明扼要，内容全面，用很少的篇幅就介绍了软件工程的主要概念和方法，软件工程的历史研究情况、最新的进展和存在的问题。

本书可以作为研究生、本科生的软件工程课程的习题集。由于本书的理论知识浅显易懂，也可以作为急需了解软件工程知识的技术人员的入门书籍。

参加本书翻译工作的主要成员有：钟鸣、王君。全书由刘晓霞负责审校。同时担任部分翻译及校对工作的还有石永平、张文、田晓涛、梅刚、孙登峰、樊伟、文卫东等同志。

由于译者水平有限，难免有错误或不妥之处，敬请读者批评指正。

前　言

软件工程并不仅仅是技术和术语的综述，它包括了学生必须掌握的技术。本书供在校大学生和研究生学习软件工程使用。笔者一直教授大学的软件工程课程，在25年的教学过程中认识到，要想帮助学生掌握软件工程的技术，必须讲解样例并给予指导。

本书应该与软件工程的课本或讲义结合使用。本书不包括图表、符号及技术的基础知识和说明，仅介绍正确构造图表的规则。本书还给出了使用各项技术的说明，包括运用技术的规则。最重要的是，对于图表、符号及技术，给出了例题和详细的解答。

本书得以撰写成功并不完全源于个人努力。许多人对本书都有贡献。我特别想感谢下列人员：Karen，我的好妻子。在本书的创作过程中，她提供了大力的支持和帮助。没有她的帮助，本书不可能完成。Steve，他从自己的博士学习中抽时间对本书的许多章节提出了意见。我的学生们提供了撰写本书的最初灵感，并阅读了各章，发现了错误，提出了许多建议。我还想谢谢Ramon，是他建议撰写此书的，还要谢谢McGraw-Hill的编辑人员，感谢他们给予的帮助和提出的建议。

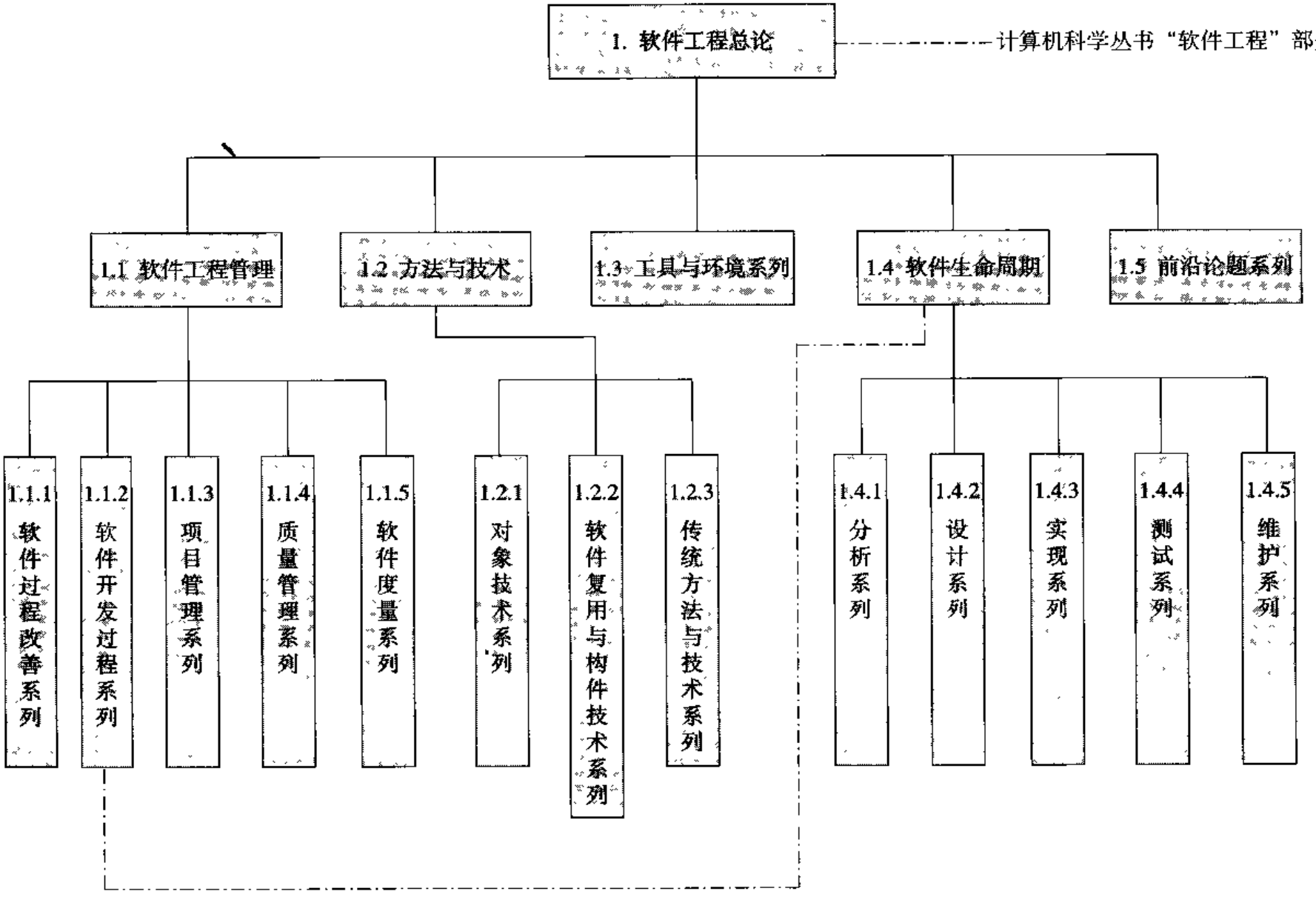
软件工程技术丛书书目

丛书编号	英文书名	中文书名	作　者
1	Object-Oriented and Classical Software Engineering, 5E	面向对象与传统软件工程(原书第5版)	Stephen R. Schach
1	Object-Oriented Software Engineering	面向对象的软件工程	Ian Sommerville
1	Software Engineering: A Practitioner's Approach, 5E	软件工程:实践者的研究方法(原书第5版)	Roger S. Pressman
1	Software Engineering, 6E	软件工程(原书第6版)	Ian Sommerville
1	Software Engineering with Java	软件工程:Java语言实现	Stephen R. Schach
1	Project-Based Software Engineering:An Object-Oriented Approach	基于项目的软件工程:面向对象研究方法	Evelyn Stiller
1.1.1	Software Process Improvement: Practical Guidelines for Business Success	软件过程改进	Sami Zahran
1.1.1	Making Process Improvement Work	软件过程改进简明实践	Neil S. Potter
1.1.2	The Road to the Unified Software Development Process	统一软件开发过程之路	Ivar Jacobson
1.1.2	The Unified Software Development Process	统一软件开发过程	Jacobson/Booch/Rumbaugh
1.1.2	The Rational Unified Process: An Introduction , 2E	统一过程引论(原书第2版)	Philippe Kruchten
1.1.2	UML and The Unified Process: Practical Object-Oriented Analysis & Design	UML和统一过程: 实用面向对象的分析和设计	Jim Arlow
1.1.3	Managing Global Software Projects: How toLead Geographically Distributed Teams.Manage Processes and Use Quality Models	国际化软件项目管理	Gopalaswamy Ramesh
1.1.3	Software Project Management: A Unified Framework	软件项目管理:一个统一的框架	Walker Royce
1.1.3	How to Run Successful Projects III: The Silver Bullet	成功的软件项目管理:银弹方案(原书第3版)	Fergus O'Connell
1.1.3	Successful Software Development, 2E	成功的软件开发(原书第2版)	Scott E. Donaldson
1.1.3	Six Sigma Software Development	6σ软件开发	Christine B. Tayntor
1.1.3	IT Project Management: On Track from Start to Finish	IT项目管理: 从开始到结束的历程	Joseph Phillips
1.1.3	Successful IT Project Delivery: Learning the lessons of Project Failure	IT项目成功交付的秘诀	David Yardley
1.1.3	Software Project Management, 3E	软件项目管理(原书第3版)	Bob Hughes
1.1.3	Architecture-Centric Software Project Management	软件项目管理实用指南: 以体系结构为中心	Daniel J. Paulish
1.1.4	Handbook of Software Quality Assurance, 3E	软件质量保证(原书第3版)	Gordon G. Schulmeyer
1.1.4	Software Rehability Engineering	软件可靠性工程	John Musa
1.1.4	Implementing ISO 9001.2000 The Journey from Conformance to Performance	ISO 9001:2000 实施指南	Tom Taimubg
1.1.4	CMMI Distilled: A Practical Introduction to Integrated Process Improvement	CMMI精粹:集成化过程改进实用导论	Dennis M Ahern
1.1.4	CMM Implementation Guide	CMM实施与软件过程改进	Kim Caputo
1.1.4	Implementing the Capability Maturity Model	CMM实施指南	James R.Persse
1.1.4	Object-Oriented Defect Management of Software	面向对象软件的差错管理	Houman Younessi
1.1.4	Metrics and Models in Software Quality Engineering	软件质量工程:度量与模型	Stephen H. Kan
1.1.4	Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software	软件性能工程	Connie U. Smith

丛书编号	英文书名	中文书名	作　者
1.1.4	Solid Software	Solid Software	Shari Pfleeger
1.1.4	Peer Reviews in Software: A Practical Guide	同级评审	Karl E. Wiegers
1.1.5	Practical Software Measurement	软件度量实用教程	John McGarry
1.1.5	Software Metrics. A Rigorous and Practical Approach, 2E	软件度量(原书第2版)	Norman E. Fenton
1.1.5	Software Assessments, Benchmarks, and Best Practices	软件评估、基准测试与最佳实践	Capers Jones
1.2.1	The Object Primer: The Application Developer's Guide to Object Orientation and the UML. 2E	面向对象软件开发教程(原书第2版)	Scott W. Ambler
1.2.1	UML and C++: A Practical Guide to Object-Oriented Development, 2E	C++面向对象开发(原书第2版)	Richard C.Lee
1.2.1	Object-Oriented Methods: Principles & Practices, 3E	面向对象方法:原理与实践(原书第3版)	Ian Graham
1.2.1	Principles of Object-Oriented Software Development, 2E	面向对象软件开发原理(原书第2版)	Anton Eliëns
1.2.1	Object Solutions: Managing the Object-Oriented Project	面向对象项目的解决方案	Grady Booch
1.2.1	An Introduction To Object-Oriented Programming, 3E	面向对象程序设计导引(原书第3版)	Timothy Budd
1.2.1	The Object Advantage: Business Process Reengineering with Object Technology	对象优势:采用对象技术的业务过程再工程	Ivar Jacobson
1.2.1	The Unified Modeling Language User Guide	UML用户指南	Booch/Rumbaugh/Jacobson
1.2.1	The Unified Modeling Language Reference Manual	UML参考手册	Rumbaugh/Jacobson/Booch
1.2.1	Applying UML and patterns: An Introduction to Object-Oriented Analysis and Design, 1E	UML和模式应用(原书第1版)	Craig Larman
1.2.1	Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process , 2E	UML与模式应用(原书第2版)	Craig Larman
1.2.2	Software Reuse: Architecture, Process and Organization for Business Success	软件复用: 结构、过程和组成	Ivar Jacobson
1.2.2	Software Reuse Techniques: Adding Reuse to the Systems Development Process	软件复用技术: 在系统开发过程中考虑复用	Carma McClure
1.2.2	Practical Software Reuse: Strategies for Introducing Reuse Concepts in Your Organization	实用软件复用方法	Donald J. Reifer
1.2.2	Large-Scale Component-Based Development	大规模基于构件的软件开发	Alan W. Brown
1.2.2	Component-based Product Line Engineering with UML	基于构件的产品线工程:UML方法	Colin Atkinson
1.4.1	Object-Oriented Analysis and Design with Applications, 2E	面向对象的分析与设计: 实例应用(原书第2版)	Grady Booch
1.4.1	Object-Oriented Analysis & Design	面向对象的分析与设计	Andrew Haigh
1.4.1	Analysis Patterns: Reusable Object Models	分析模式: 可复用的对象模型	Martin Fowler
1.4.1	Requirements Analysis and System Design: Developing Information Systems with UML	需求分析与系统设计	Leeszek A. Maciaszek
1.4.1	Systems Analysis and Design in a Changing World	系统分析与设计	John W. Satzinger
1.4.1	Advanced Use Case Modeling, Vol I: Software Systems	高级用例建模 卷1: 软件系统	Frank Armour
1.4.1	Requirements Engineering: A Good Practice Guide	需求工程	Ian Sommerville
1.4.1	Software Requirements and Estimation	软件需求与估算	Swapna Kishore
1.4.1	Effective Requirements Practices	有效需求实践	Ralph R.Young
1.4.1	Applying Use Cases: A Practical Guide, 2E	用例分析技术(原书第2版)	Geri Schneider
1.4.1	Managing Software Requirements	软件需求管理:统一方法	Dean Leffingwell
1.4.1	Writing Effective Use Cases	编写有效用例	Alistair Cockburn
1.4.1	Problem Frames Analyzing and Structuring Software Development Problems	Problem Frames Analyzing and Structuring Software Development Problems	Michael Jackson

丛书编号	英文书名	中文书名	作　者
1.4.2	Object-Oriented Software Construction, 2E	面向对象的软件结构(原书第2版)	Bertrand Meyer
1.4.2	Pattern-Oriented Software Architecture, Vol I: A System of Patterns	面向模式的软件体系结构 卷1:模式系统	Frank Buschmann
1.4.2	Pattern-Oriented Software Architecture, Vol II: Patterns for Concurrent and Networked Objects	面向模式的软件体系结构 卷2:用于并发和网络化对象的模式	Douglas Schmidt
1.4.2	Server Component Patterns	面向模式的软件体系结构 卷3:服务器组件模式	Markus Volter
1.4.2	DesignPatterns: Elements of Reusable Object-Oriented Software	设计模式:可复用面向对象软件的基础	Gamma/Helm/Johnson /Vlissides
1.4.2	Patterns of Enterprise Application Architecture	企业级应用体系结构模式	Martin Fowler
1.4.2	AntiPatterns and Patterns in Software Configuration Management	软件配置管理中的模式与反模式	William J. Brown
1.4.2	Software for Use: A Practical Guide to The Models and Methods of Usage-Centered Design	面向使用的软件设计	Larry L. Constantine
1.4.2	Software Architecture: Organizational Principles and Patterns	软件架构: 组织原则与模式	David Dikel
1.4.2	The UML Profile for Framework Architectures	框架体系结构的UML档案	Marcus Fontoura
1.4.2	Software Architect's Profession: An Introduction	软件架构师入门必读	Marc Sewell
1.4.2	Business Modeling with UML: Business Patterns at work	UML业务建模. 实用业务模式	Hans-Erik Eriksson
1.4.3	Software Fabrication:Automating Application Development	软件构造:自动化软件开发	Jack Greenfield
1.4.3	Building J2EE Applications With The Rational Unified Process	用RUP构建J2EE 应用程序	Peter Eeles
1.4.3	Programming from Specifications	从规范出发的程序设计	Carroll Morgan
1.4.4	Testing IT: An Off-the-Shelf SoftwareTesting Process Handbook	实用软件测试过程之路	John Watkins
1.4.4	Lessons Learned in Software Testing	软件测试经验与教训	Cem Kaner
1.4.4	Testing Computer Software: The Bestselling Software Testing Book Of All Time, 2E	计算机软件测试(原书第2版)	Cem Kaner
1.4.4	Software Testing in the Real World: Improving the Process	软件测试过程改进	Edward Kit
1.4.4	Effective Methods for Software Testing, 2E	有效的软件测试方法(原书第2版)	William E. Perry
1.4.4	Beta Testing for Better Software	软件Beta测试	Michael R. Fine
1.4.4	A Practical Guide to Testing Object-Oriented Software	面向对象的软件测试	John D. McGregor
1.4.4	Managing the Testing Process, 2E	测试过程管理(原书第2版)	Rex Black
1.4.4	Software Testing: A Craftsman's Approach, 2E	软件测试: 工艺师的方法(原书第2版)	Paul C. Jorgensen
1.4.4	Just Enough Software Test Automation	软件测试自动化	Daniel J. Mosley
1.4.4	The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing	软件测试实用技术	Brian Marick
1.5	JAVA Tools for Extreme Programming: Mastering Open Source Tools, Including Ant,JUnit, and Cactus	极限编程的JAVA工具	Richard Hightower
1.5	Agile Software Development Ecosystems	敏捷软件开发生态系统	Tom DeMarco
1.5	Agile Modeling: Effective Practices For eXtreme Programming and The Unified Process	敏捷建模:极限编程和统一过程的有效实践	Scott W. Ambler
1.5	Model-Driven Development: Automating Component Design, Implementation, and Assembly	模式驱动的软件开发	David Frankel
1.5	A Practical Guide to Feature-Driven Development	特征驱动开发方法:原理与实践	Steve R. Palmer

软件工程技术丛书结构图



目　录

译者序
前言

第1章　软件生命周期 ……1
1.1　概述……1
1.1.1　软件生命周期活动的类型 ……1
1.1.2　典型文档 ……2
1.2　软件生命周期模型……2
1.2.1　线性模型 ……2
1.2.2　原型实现模型 ……3
1.2.3　增量模型 ……3
1.2.4　Boehm的螺旋模型 ……3
习题……3
习题答案……4
第2章　软件过程模型和其他模型 ……7
2.1　软件过程模型……7
2.2　数据流程图……8
2.3　petri网模型 ……9
2.4　对象模型 ……10
2.4.1　存在依赖……11
2.4.2　实例图……13
2.5　用例图 ……13
2.6　场景 ……14
2.7　时序图 ……14
2.8　层次结构图 ……15
2.9　控制流程图 ……15
2.10　状态图……16
2.11　网络模型……18
习题 ……18
补充问题 ……19
习题答案 ……21
补充问题答案 ……23
第3章　软件项目管理 ……29
3.1　概述 ……29
3.2　管理方法 ……29
3.3　小组方法 ……29
3.4　重要准则 ……30
3.5　能力成熟度模型 ……32
3.6　个人的软件过程 ……33
3.7　实现值分析 ……33
3.7.1　基本的量……33
3.7.2　进度指示器……34
3.8　错误跟踪 ……35
3.9　事后回顾 ……36
习题 ……37
补充问题 ……37
习题答案 ……39
补充问题答案 ……40
第4章　软件项目计划 ……45
4.1　项目计划 ……45
4.2　WBS：工作分解结构 ……45
4.3　PERT：程序评估和评审技术 ……48
4.3.1　完成时间的算法……49
4.3.2　关键路径……50
4.3.3　构造关键路径的算法……50
4.3.4　宽松时间……51
4.3.5　宽松时间的算法……51
4.4　软件成本估算 ……52
4.4.1　代码行估算……52
4.4.2　基于LOC的成本估算……53
4.4.3　成本构成模型……54
4.4.4　功能点分析……55
4.4.5　生产率……57
4.4.6　判定估算……57
4.4.7　自动估算工具……58
习题 ……58
补充问题 ……59

习题答案 ……61
补充问题答案 ……63
第5章 软件度量 ……69
5.1 概述 ……69
5.2 软件度量理论 ……70
5.2.1 单调性……71
5.2.2 度量尺度……71
5.2.3 统计……72
5.3 产品度量 ……72
5.3.1 McCabe的环数 ……73
5.3.2 Halstead的软件科学 ……75
5.3.3 Henry-Kafura信息流 ……79
5.4 过程度量 ……80
5.5 GQM方法 ……80
习题 ……80
补充问题 ……81
习题答案 ……82
补充问题答案 ……83
第6章 风险分析和管理 ……89
6.1 概述 ……89
6.2 风险确定 ……89
6.3 风险估算 ……89
6.4 风险揭露 ……90
6.5 风险缓解 ……90
6.6 风险管理计划 ……91
习题 ……92
补充问题 ……92
习题答案 ……93
补充问题答案 ……93
第7章 软件质量保证 ……95
7.1 概述 ……95
7.2 形式化检查和技术评审 ……95
7.2.1 检查的角色……96
7.2.2 检查的步骤……96
7.2.3 检查表……96
7.3 软件的可靠性 ……97
7.3.1 错误率……97
7.3.2 概率论……98
7.4 统计质量保证 ……98
7.5 SQA计划的IEEE标准 ……98
习题……101
补充问题……101
习题答案……102
补充问题答案……103
第8章 需求 ……111
8.1 概述 ……111
8.2 对象模型 ……111
8.3 数据流程建模 ……112
8.4 行为建模 ……112
8.4.1 用例 ……113
8.4.2 场景 ……114
8.4.3 状态图 ……114
8.5 数据字典 ……115
8.6 系统图 ……116
8.7 软件需求规格说明的IEEE标准 ……117
习题……118
补充问题……119
习题答案……119
补充问题答案……120
第9章 软件设计……127
9.1 概述……127
9.2 设计过程的各个阶段……128
9.3 设计概念……130
9.4 度量内聚性……132
9.4.1 程序片 ……132
9.4.2 粘合权标 ……133
9.5 度量耦合性……135
9.6 需求的可溯性……136
习题……139
补充问题……140
习题答案……140
补充问题答案……142
第10章 软件测试 ……145
10.1 概述 ……145
10.2 软件测试基础知识 ……145
10.3 测试覆盖准则 ……146
10.3.1 包含……146
10.3.2 功能测试……146

10.3.3 测试度量……147
10.3.4 结构测试……148
10.4 数据流测试 ……152
10.5 随机测试 ……154
10.5.1 操作预置文件……154
10.5.2 测试的统计推断……155
10.6 边界测试 ……155
习题……156
补充问题……157
习题答案……159
补充问题答案……160
第11章 面向对象的软件开发……167
11.1 概述 ……167
11.1.1 继承……167
11.1.2 多态性……168
11.2 确定对象 ……169
11.2.1 noun-in-text方法……169
11.2.2 确定继承……171
11.2.3 确定重用……172
11.2.4 用例方法……173
11.3 确定关联 ……173
11.4 确定多重性 ……175
习题……177
补充问题……177
习题答案……178
补充问题答案……178
第12章 面向对象的度量 ……181
12.1 概述 ……181
12.1.1 传统的度量……181
12.1.2 面向对象的抽象……181
12.2 面向对象设计的度量套件 ……182
12.2.1 度量1：每个类的加权方法 ……182
12.2.2 度量2：继承树的深度 ……182
12.2.3 度量3：孩子的数目 ……182
12.2.4 度量4：对象类之间的耦合 ……182
12.2.5 度量5：类的响应 ……183
12.2.6 度量6：方法缺乏内聚力 ……183
12.3 MOOD度量 ……187
12.3.1 封装……187
12.3.2 继承因子……188
12.3.3 耦合因子……189
12.3.4 多态性因子……190
习题……191
补充问题……191
习题答案……192
补充问题答案……193
第13章 面向对象的测试 ……197
13.1 概述 ……197
13.2 MM测试 ……197
13.3 函数对的覆盖 ……199
习题……203
补充问题……203
习题答案……204
补充问题答案……204
第14章 形式化表示方法 ……207
14.1 概述 ……207
14.2 形式化的规格说明 ……207
14.2.1 前提条件……208
14.2.2 后置条件……208
14.2.3 不变式……208
14.3 对象约束语言 ……208
14.3.1 导航……209
14.3.2 不变式……210
14.3.3 属性……210
14.3.4 预定义操作……210
14.3.5 前提条件和后置条件……211
习题……211
补充问题……211
习题答案……213
补充问题答案……214

第1章　软件生命周期

1.1　概述

软件生命周期是在软件开发中发生的不同活动的序列。在开发过程中会产生不同的交付内容。虽然交付内容可以是协议或评价，但交付内容一般是实物，如源代码或用户手册。通常，活动和交付是密切相关的。里程碑（milestone）是用来说明项目状态的事件。例如，完成用户手册的事件可以是一个里程碑。为了便于管理，里程碑很关键，因为里程碑的完成使管理人员能评价软件开发的进程。

1.1.1　软件生命周期活动的类型

1.1.1.1　可行性——确定所建议的开发是否有价值。

市场分析——确定本产品是否有潜在的市场。

1.1.1.2　需求——确定软件应该包括什么功能。

需求引入——从用户处获得需求。

领域分析——确定本问题中最常见的任务及结构。

1.1.1.3　项目计划——确定怎样开发软件。

成本分析——确定成本估算。

时间规划——建立开发时间表。

软件质量保证——确定有助于保证产品质量的活动。

工作分步结构——确定开发产品所必需的子任务。

1.1.1.4　设计——确定软件应该如何提供功能。

结构设计——设计软件系统的结构。

接口设计——指定系统各部件之间的接口。

详细设计——设计用于软件系统各个部件的算法。

1.1.1.5　实现——建造软件。

1.1.1.6　测试——用数据来运行软件以帮助确保软件正确工作。

单元测试——由原开发人员进行的测试。

集成测试——在软件的集成中进行的测试。

系统测试——在一个与运行环境相匹配的环境中对软件进行的测试。

阿尔法测试——顾客在开发人员处对软件进行的测试。

贝塔测试——在顾客处对软件进行的测试。

接收测试——满足买方需求的测试。

回归测试——对以前的版本进行测试以确保新版本仍然保持以前的功能。

1.1.1.7 交付——给顾客提供一个有效的软件解决方案。

安装——使软件在顾客处可以使用。

培训——教用户使用软件。

帮助桌面——回答用户的问题。

1.1.1.8 维护——更新和改进软件以确保软件继续有用。

1.1.2 典型文档

1.1.2.1 工作陈述——所需功能的初步描述，通常由用户提供。

1.1.2.2 软件需求规格说明——描述最终的软件能做什么工作。

对象模型——说明主要的对象/类。

用例场景——从用户的角度说明可能的一系列行为。

1.1.2.3 项目进度——描述任务的顺序以及估计的时间和工作。

1.1.2.4 软件测试计划——描述怎样测试软件以确保软件行为正确。

接收测试——顾客为确定系统的可接受性而设计的测试。

1.1.2.5 软件设计——描述软件的结构。

体系结构设计——具备互连性的高级结构。

详细设计——低级模块或对象的设计。

1.1.2.6 软件质量保证计划（SQA计划）——描述用来保证质量的活动。

1.1.2.7 用户手册——描述怎样使用最终的软件。

1.1.2.8 源代码——实际的产品代码。

1.1.2.9 测试报告——描述进行了什么测试以及系统运转情况如何。

1.1.2.10 缺陷报告——描述顾客对特定系统行为不满意的地方，一般是指软件故障或错误。

1.2 软件生命周期模型

以下各节给出了四种最常见的软件生命周期模型。

1.2.1 线性模型

这个模型也称为瀑布模型（如图1-1所示），因为其图形像瀑布而得名。此模型由Royce在1970年首先提出，它是第一个描述了任务的标准序列。

瀑布模型有许多版本。虽然某些开发任务在每个开发中几乎都会出现，但划分开发阶段的方法有许多种。请注意，在图1-1所示的瀑布模型中，项目计划活动包含在需求阶段中，且交付及维护阶段也省略了。

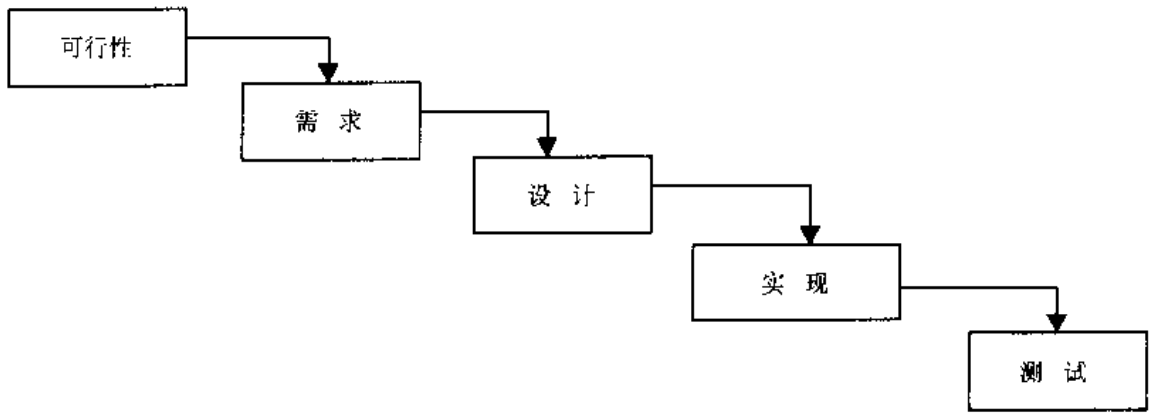


图1-1 瀑布模型

1.2.2 原型实现模型

软件生命周期模型往往先建立一个临时性的版本（原型）。这个原型主要用于测试概念和需求。此原型用来向顾客演示建议的行为。在取得顾客的认可后，相应的软件开发通常遵循与线性模型开发相同的阶段。这样做可避免开发不必要的功能。

1.2.3 增量模型

增量模型是由D. L. Parnas提出的（D.Parnas. "Designing Software for Ease of Extension and Contraction." *IEEE Transactions on Software Engineering (TOSE)* 5:3, March 1979, 128-138）。其目标是设计和向用户提交整个系统的一个最小子集，而且此子集仍然是一个有用的系统。这个设计和提交的过程在整个生命周期中以最小的额外增量重复进行。此模型的优点可以给用户提供一个早期的工作系统以及工作增量。

1.2.4 Boehm的螺旋模型

B. Boehm引入了螺旋模型（B.Boehm. "A Spiral Model for Software Development and Enhancement." *IEEE Computer*. 21:5, May 1988, 61-72）。此模型的图像是一个螺旋结构，此结构从中间开始，不断地重新访问客户沟通（面谈等）、规划、风险分析、工程、构造和释放以及客户评估等基本任务。

习题

1. 阶段性的生命周期模型是怎样支持软件管理的？

2. 里程碑的两个必需的特征是什么？

3. 指出下列文档是在软件生命周期的哪个阶段产生的：最终用户手册、体系结构设计、SQA计划、模块规格说明、源代码、工作陈述、测试计划、初级用户手册、详细设计、成本

估计、项目计划、测试报告、文档。

4. 指出瀑布模型中下列任务的顺序：接收测试、项目计划、单元测试、需求评审、成本估计、高级设计、市场分析、初级设计、系统测试、设计评审、实现、需求规格说明。

5. 画出一个表现循环生命周期模型的图表。

习题答案

1. 阶段性的生命周期模型是怎样支持软件管理的？

阶段性的生命周期提高了项目的可视性。项目可利用各个阶段作为里程碑来进行管理。更详细地划分阶段能够更密切地监控项目的进展。

2. 里程碑的两个必需的特征是什么？

里程碑必须与软件开发的进展相关联且必须在完成时非常明显。

3. 指出下列文档是在软件生命周期中的哪个阶段产生的？

最终用户手册	实现阶段
体系结构设计	设计阶段
SQA计划	项目计划阶段
模块规格说明	设计阶段
源代码	实现阶段
工作陈述	可行性阶段
测试计划	需求阶段
初级用户手册	需求阶段
详细设计	设计阶段
成本估计	项目计划阶段
项目计划	项目计划阶段
测试报告	测试阶段
文档	实现阶段

4. 各任务的顺序如下：

市场分析
项目计划、成本估计、需求规格说明（可同时完成）
需求评审
高级设计
初级设计
设计评审
实现

单元测试
系统测试
接收测试

5. 画出一个表现循环生命周期模型的图表。请参考图1-2。

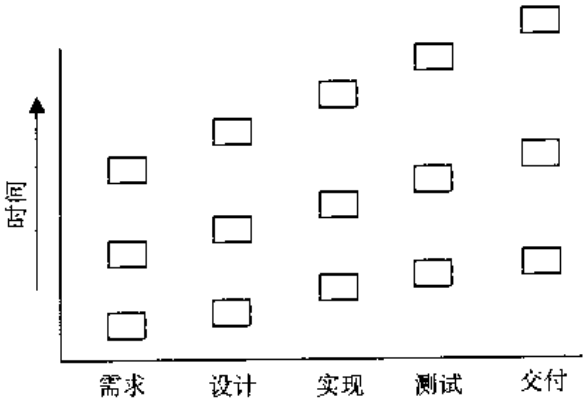


图1-2 循环生命周期模型

第2章　软件过程模型和其他模型

2.1　软件过程模型

软件过程模型（Software Process Model, SPM）描述完成软件开发所执行的过程。软件过程模型一般包括以下内容：

- 任务
- 制品（文件、数据等）
- 参与者
- 判断（可选）

软件过程模型所使用的符号是多种多样的。标准的软件过程模型对任务和过程使用椭圆形，制品由矩形表示，而参与者用小人来表示。许多软件过程模型不包括判断，但用菱形表示判断。流向用弧来表示，其方向一般为从左到右，自顶向下。弧一般不标记。

下面是正确的过程模型的规则和解释：

- 不能用一条弧连接两个任务。任务必须由制品分隔开来。
- 任务在输入制品存在前不可执行。
- 每个模型都有一个或多个开始任务，一个或多个最终任务。
- 所有任务都必须是从开始任务可到达的。
- 从每个任务到达最终任务都存在一条路径。

软件过程模型可以是描述性的，即它们可以描述项目开发过程中已经发生了什么事情。描述模型通常是作为项目的事后分析的组成部分而建立的。这在确定软件开发过程中的问题时非常有用。软件过程模型也可以是事先描述性的，即软件过程模型可以描述应该发生什么事情。事先描述性的软件过程模型可用来描述标准软件开发过程。它们可用于培训新雇员，或用于不常见事件的参考以及用于描述应该发生什么事情。

例子2.1

图2-1是一个用于单元测试软件的过程模型。该模型有两个参与者，分别是测试员和项目组领导。当然，单元测试员负责单元测试。单元测试员利用源代码和测试计划来完成单元测试。这个活动的结果是一个制品，即测试结果。项目组领导评审测试结果，而且相应活动的

结果应该是此单元测试得到认可。这个模型并未明显地说明在此过程不成功时怎么办。不过应该能推断出测试员会不断测试直至满意为止。类似地，如果项目组领导不打算给予认可，那么此过程将返回去重新进行测试。

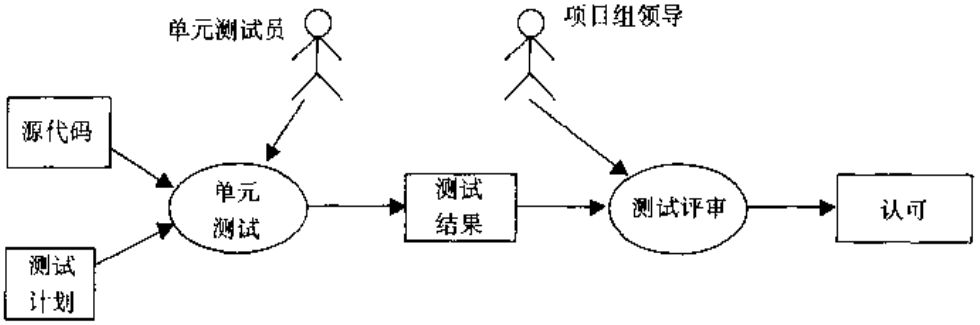


图2-1 单元测试的过程图

例子2.2

画出有判断的过程模型。

增加关于在所有情况下的处理方法的判断会使过程模型更为清楚，如图2-2所示。

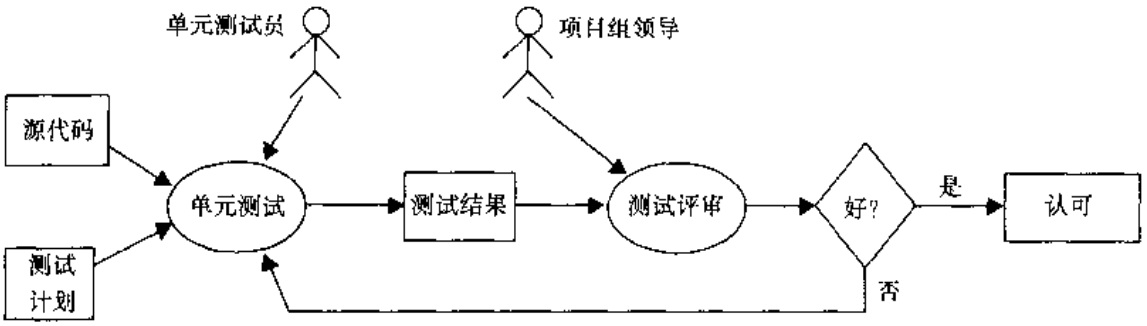


图2-2 带判断的过程模型

2.2 数据流程图

软件开发中一种最基本的图就是数据流程图。数据流程图显示一组组件之间的数据流向。组件可以是任务、软件组件或软件系统中将要包含的功能的抽象。参与者不包含在数据流程图中。动作的顺序通常可从活动框的顺序推断出来。

以下是正确的数据流程图的规则和解释：

- 框表示过程，必须用动词短语标记。
- 弧代表数据，必须用名词短语标记。

- 控制不显示出来。某些顺序可以从次序中推断出来。
- 过程可以是一个一次性的活动，也可以代表一个连续的过程。
- 从一个框流出的两条弧可以表示两个输出都产生，也可以表示产生其中一个输出。

例子2.3

前一节的单元测试例子可以描绘为一个数据流程图，如图2-3所示。

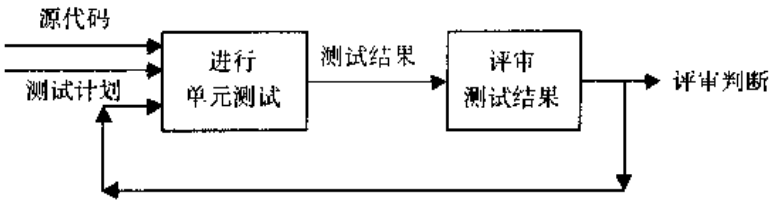


图2-3 单元测试的数据流程图

图2-3说明了某些规则。框中的短语为动词短语。它们代表动作。每个箭头/线用描绘某种制品的名词短语来标记。

此数据流程图没有明确地给出判断。这个例子说明测试的结果会影响进一步的测试，而且测试评审操作的结果也会影响测试（或重新测试）。

例子2.4

可将数学公式$(x + y) * (w + z)$的计算表示为以下的运算序列（如图2-4所示）：

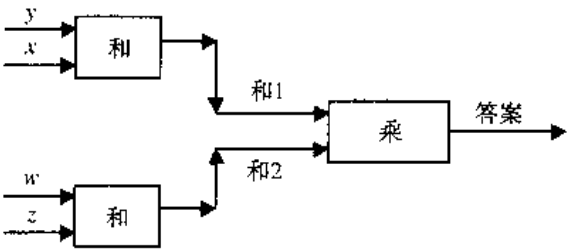


图2-4 数学公式$(x + y) * (w + z)$的计算

2.3 petri网模型

基本的petri网模型由条件结点、弧、事件结点和权标组成。如果一个事件结点的输入条件结点都有权标，则可以激发该事件，可以从输入结点删除权标，并且将权标放在激发位置的所有输出结点上。条件结点一般用圆表示，而事件结点用水平线或矩形表示。

在一个petri网模型中，条件结点一般代表某些必需的条件，如存在测试计划等。条件处

的权标表示该条件是满足的。事件结点（水平线）表示当所有需求满足时（所有条件结点中的权标）所发生（激发）的事件。然后将权标放置在相应事件后的所有条件结点上。

例子2.5

测试的一个petri网模型如图2-5所示。

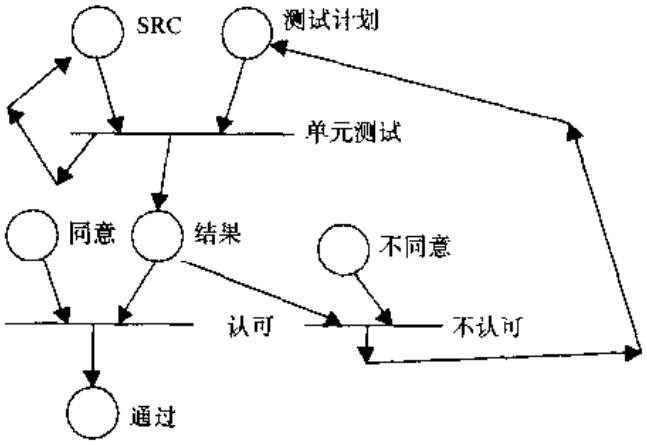


图2-5 petri网模型

基本petri网模型还有许多不同的变形。

2.4 对象模型

在面向对象的并发中（第11章），问题领域的问题及机器空间中的解决方案都用对象来描述。在解决方案中，这些对象一般是类。随着软件开发的进展，从需求到设计阶段，对象从问题领域中事物的表示转换为软件中的程序设计结构。

对象模型可以用来表示实体及实体间的关系。每个框表示一种对象类型，而对象的名称、属性及方法都列在框中。框的顶端部分给出对象名，第二部分给出属性，底部给出方法。两个对象之间的一条弧代表对象之间的一个关系。可在弧中间用关联的名字来标记。角色可在另一端标记。而且，在每一端都可以给出多重性以指出相同的种类允许多少不同的关联。

关系的三种主要类型为继承、聚合、关联。继承关系表示位于弧底部的对象是位于弧顶部对象的一个特殊例子。例如，顶部的对象可能是一种车辆，而底部的对象是一辆小汽车，小汽车也是一种车辆。这通常称为“is-a”关系。聚合关系表示弧底部的对象为弧顶部对象的一个组成部分。例如，顶部对象可能是一辆小汽车，而底部对象可以是引擎。这种关系通常称为“part-of”关系。最后一种关系是关联关系，这种弧表示某个对象与其他对象是关联的。例如，“父－子”关系是一种关联关系。这种关系可以是双向的，也可以

是单向的。

虽然有许多种不同的表示方法，但我们只使用与UML(Unified Modeling Language，统一建模语言）标准（请参阅www.omg.org或www.rational.com，或者用浏览器搜索UML）兼容的表示方法。

例子2.6

构造一个图书馆的对象模型。图2-6所示的简单的图书馆中的对象由图书馆、书、书的副本以及读者组成。

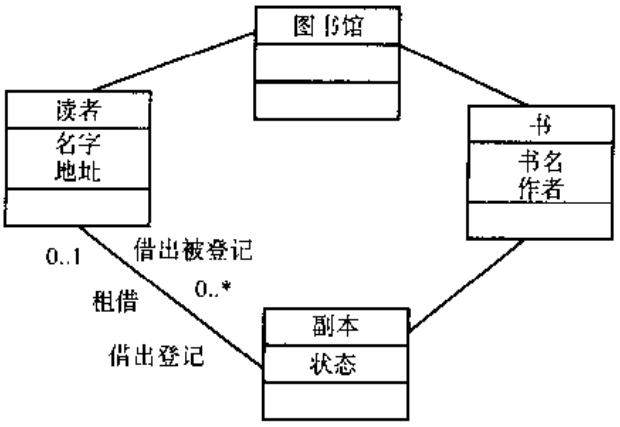


图2-6 简单图书馆的对象模型

图中未给出对象的方法。此图书馆有一个书和读者的聚合关系，即图书馆确实是由书和读者组成。书和副本之间的关系既不是集合关系也不是继承关系。对象“书”表示书的抽象，而副本为借出的实际书籍。读者和副本之间的关系称为“租借”。从副本的观点来看，任务是“借出被登记”，而从读者的角度来看，任务是“借出登记”。多重性指出一个副本可以不进行借出登记，也可以只借给一个读者（“0..1”）。另一多重性“0..*”指出一个读者每次可以具有0个、1个或多个“借出登记”关系。

例子2.7

构造一棵家族树系统的对象模型，它用来存储家族成员的血统信息。

图2-7指出每个人都有一个出生家庭。每桩婚姻都有一对父亲和母亲。图中省略了许多属性，也没有给出函数操作。

2.4.1 存在依赖

理解关系的一种办法是引人另一个称为存在依赖（Existence Dependency, ED）的关系

（Snoeck and Dedene. "Existence Dependency: The Key to Semantic Integrity between Structural and Behavioral Aspects of Object Types." *IEEE TOSE*, April 1998）。存在依赖关系定义如下：如果该较低类（子女）只在较高（双亲）类存在时存在并且较低（子女）类只与较高（双亲）类的一个实例相联系的话，就说一个类（双亲）可以与一个较低类（子女）相联系。该关系和继承可以用来表示任何问题领域。

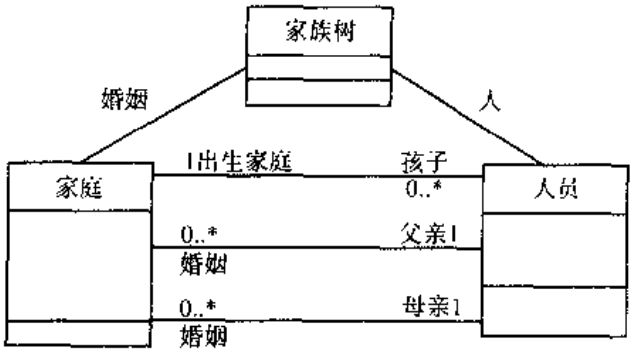


图2-7 家族树对象模型

例子2.8

利用存在依赖关系构造图书馆的一个对象模型。

如例子2.6中的图2-6所示，除"租借"外的所有关系以及图书馆－书都满足存在依赖的要求。但不满足"租借"关系，因为一个副本对象可以存在于登记借阅它的读者对象之前。不过，可以建立一个确实满足ED关系的租借对象。对象"书"不能是图书馆的子女，因为书可在特定的图书馆之前或之后存在。可以将"人员"增加到图中（见图2-8）以显示在"图书馆"上不属于存在－依赖的那部分读者。

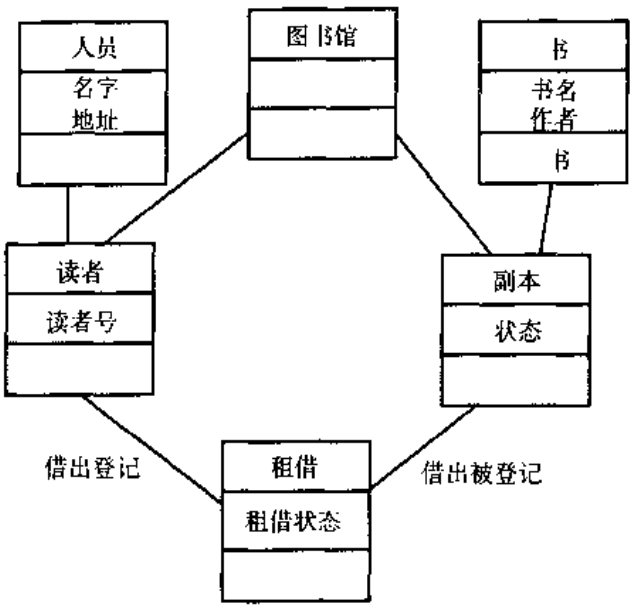


图2-8 使用ED的图书馆对象模型

2.4.2 实例图

对象图表示对象的类型。因此，标记为“小汽车”的框代表所有小汽车的属性和功能。有时，对象的实例之间的关系在对象图中并不是非常清晰。实例图给出了对象的例子实例，能够阐明它们之间的关系。

例子2.9

画出显示Fred、其妻子Sue、他们的孩子Bill、Tom和Mary以及Fred的父母Mike和Jean的实例模型。如图2-9所示。

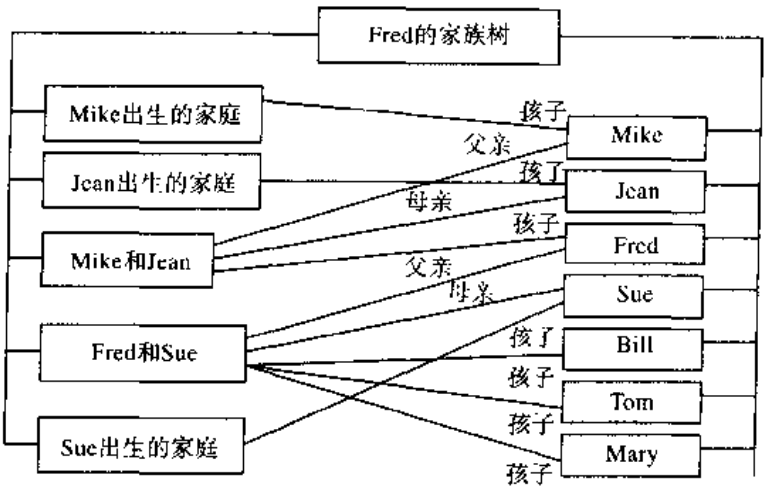


图2-9 Fred的家族树的实例图

2.5 用例图

用例（use case）图为UML图集的一部分。它给出了系统的重要参与者及功能。参与者用小人表示，而功能用椭圆表示。参与者和他们所执行的功能相关联。

例子2.10

画出简单图书馆的用例图，如图2-10所示。

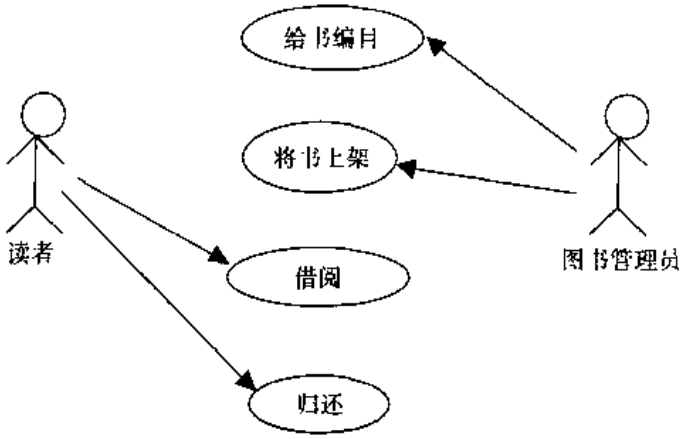


图2-10 简单图书馆的用例图

椭圆中的功能为对象模型中的类的方法。读者对象可以借阅和归还副本。图书管理员参与者不是对象模型上的对象。用例中的图书管理员指示某些功能（如给书编目和将书上架）不是读者可以使用的功能。

2.6 场景

场景是在本问题领域中发生的一个动作序列的描述。

例子2.11

编写图书馆问题的一个场景。

Fred（一位读者）到图书馆借阅一本书。两个月后，他把这本逾期的书返还给图书馆。

2.7 时序图

时序图为UML图集的一部分。时序图中垂直的线代表类的实例。每条垂直线在顶部用类名、后跟一个冒号，最后为实例名来标记。例如，第一条线标记为lib:main，表示类library（图书馆）的实例main。水平箭头描述功能调用。箭头的尾部位于调用类的线上，箭头的头部位于被调用类的线上。功能名位于箭头上。垂直线上的宽方块给出被调用功能的执行时间。返回一般不用给出。对相同功能的多个调用通常只用一个箭头表示即可。

例子2.12

画出例子2.11的时序图，如图2-11所示。

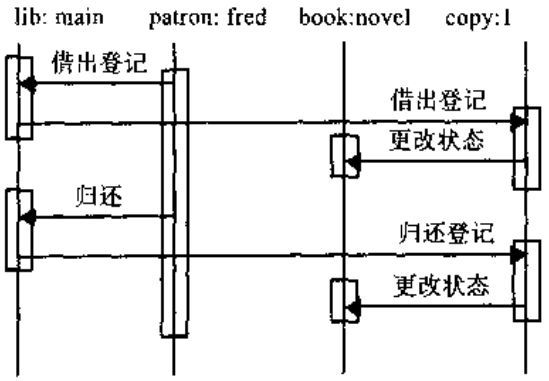


图2-11 借出登记场景的时序图

这个图比例子2.8给出的对象模型更接近设计阶段。这个图中使用的有些功能在前面的对象模型中没有使用。面且，此图中给出的调用时序是由实际的设计所决定的。

2.8 层次结构图

层次结构图给出系统的调用结构。图中的每个框代表一项功能。如果第一个功能可以调用第二个功能，则从第一个功能画一条线到第二个功能。要将所有可能的调用都画出来。

层次结构图不是UML图集之一，通常在面向对象的开发中不使用。但它是用于理解系统动态结构的一个非常有用的图。

例子2.13

画出例子2.12中使用的图书馆程序的层次结构图（参见图2-12）。

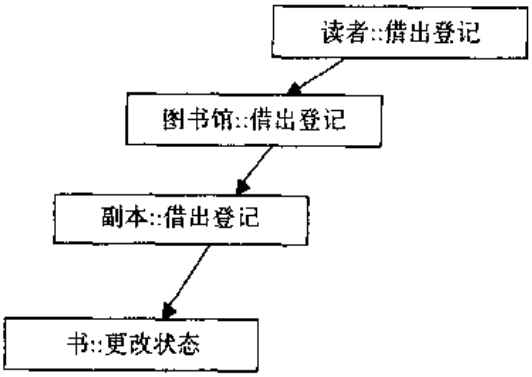


图2-12 层次结构图

2.9 控制流程图

控制流程图（CFG）给出代码的控制结构。每个结点（圆）代表一个代码块，代码块只有一条路通过代码，即在块的开始处只有一个入口，在块的结束处只有一个出口。如果执行块中任意语句，则要执行块中所有语句。结点之间的弧代表可能的控制流。也就是说，如果有可能恰好在块A之后执行块B，则必须存在一条从块A到块B的弧。

以下是正确的控制流程图的规则：

- 必须有一个开始结点。
- 从开始结点到每个结点必须存在一条路径。
- 从开始结点到一个暂停结点必须存在一条路径。

例子2.14

画出下列三角问题的控制流程图。

```
read x,y,z;
type = ''scalene'';
if (x == y or x == z or y == z) type =''isosceles'';
if (x == y and x == z) type =''equilateral'';
if (x >= y+z or y >= x+z or z >= x+y) type =''not a triangle'';
if (x <= 0 or y <= 0 or| z <= 0) type =''bad inputs'';
print type;
```

在图2-13中，“a”结点代表前两条语句和if语句。“type = isosceles”位于标记为“isosceles”的结点中。类似地，“c”结点代表下一条if语句，而“equilateral”结点代表if语句的体。

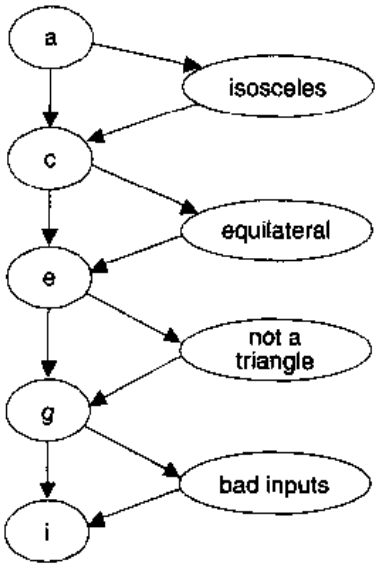


图2-13 三角程序的控制流程图

2.10 状态图

机器或程序的状态为所有变量、寄存器等的所有值的集合。状态图给出系统的状态及这些状态间可能的转换。一个程序或机器所具有的不同状态的数目极大。不过，许多状态在机器对下一个输入以及以后的输入如何反应方面是类似的。具有类似行为的一组状态可以组合成一个状态。可以用图表示这些状态以说明状态间的转换。用状态图可以很好地描述许多程序。

以下是正确的状态图的规则：

- 有一个初始状态。
- 每个状态都可从初始状态到达。
- 从每个状态都有一条路径到达停止状态。
- 状态之间的每个转换都必须用一个导致该转换的事件来标记。

例子2.15

画一个固定尺寸的堆栈的状态图（见图2-14）。

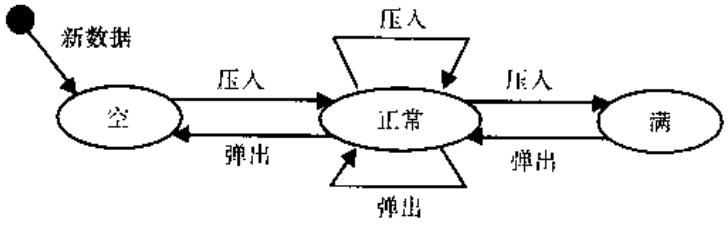


图2-14 固定尺寸的堆栈的状态图

状态图有两种绘制方法。在图2-14中，只给出合法的或非错误的转换。它假定未给出的任意转换都是不合法的。例如，对满状态，没有压入转换。另一种方法是给出所有转换，包括导致错误的转换。

例子2.16

画出带有所有错误转换的堆栈状态图（见图2-15）。

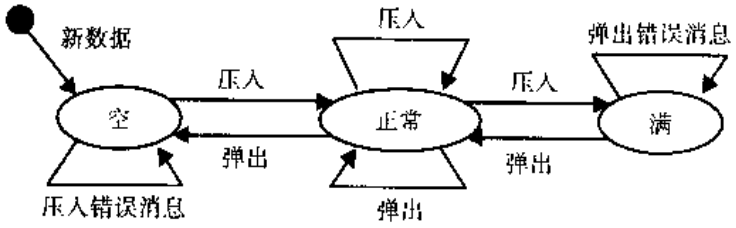


图2-15 给出错误转换的状态图

状态图可根据源代码直接画出。必须检查每个功能以便根据变量的值（系统的状态）进行决策。

例子2.17

下面是一个有限堆栈的压入方法的代码：

```
int push(int item) {
    if (stackindex == MAX) {return ERROR;}
    stack[stackindex++] = item;
    return 0;
    }
```

从这段代码可以标出功能的两个不同行为的状态。请注意，stackindex从零开始。一个状态与条件stackindex == MAX相关，另一状态与条件stackindex !== MAX相关。分析增量可知第二个状态为stackindex < MAX（至少针对这个方法是这样）。分析弹出方法可知此堆栈存在空状态。

2.11 网络模型

网络是一种给出集合关系的数学结构。虽然它在软件开发中不常使用，但在说明功能和属性集之间的关系时经常要使用网络。

例子2.18

画出一个堆栈实现的网络模型，它具有一个数组属性（s-array）和栈的顶元素下标（index）属性以及push、pop、val（显示栈顶值）和depth等方法，如图2-16所示。

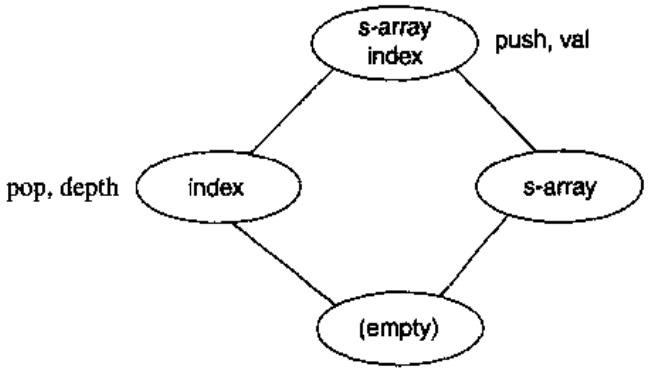


图2-16 堆栈例子的网络模型

最上面的结点代表所有属性的集合，最下面的结点表示空集。各结点用使用该属性子集的函数名来标注。

习题

1. 软件生命周期模型与过程模型之间的区别是什么？

2. 描述性过程模型与事先描述性过程模型之间的区别是什么？

3. 为什么在事先描述性过程模型中判断比在描述性过程模型中判断更常见？

4. 在一个过程模型中任务为什么应该用制品分隔？

5. 为什么过程模型中的任务在输入其制品之前不能开始？

6. 为什么过程模型中的每个结点必须具有一条从开始结点到它本身的路径，以及一条从它自身到终止结点的路径？

7. 在例子2.3中，一个人员怎样才能够区分仅发生在所有单元测试完成之后的测试评审与

发生在每个模块已经进行了单元测试后的测试评审？

8. 关于控制流，数据流程图有哪些规定？

9. 一个petri网激发位置何时激发？

10. 在一个petri网激发位置激发时会发生什么？

11. 问题领域和解决方案空间之间的区别是什么？

12. 对象模型从需求阶段到设计阶段有何改变？

13. 将下面的关系按继承关系、集合关系或一般关联进行分类。

小汽车——Lincoln Town小汽车
人员——学生
图书馆——图书馆读者
书——副本
小汽车——驾驶员
读者——书籍租借
班级——学生

14. 将下列各项分为类或类的实例：

我的汽车
人员
Fred
交通工具
教授
CIS部门

15. 场景与显示了所有可能的动作序列的状态图之间有什么关系？
16. 在一个相互作用图中，调用类还是被调用类位于箭头的头部？
17. 解释为什么集合关系是问题领域中的关系而不是实现领域中的关系？
18. 为什么数据流程图没有关于结点之间的可达性的规则？

补充问题

1. 画出油漆房间墙壁这个任务的一个过程模型。该过程包括下列任务：选择颜色、购买

油漆、清洁墙壁、搅拌油漆、油漆墙壁。

2. 作者在讲授一门有远程学习的学生参与的课程时使用了交互式会话。作者把学生划分成小组并在Web页上贴出问题。各小组利用聊天室讨论问题，利用留言板向指导教师提问，并通过电子邮件提交解决方案。然后，指导教师利用评分表对学生的解决方案进行评分。请画出交互式会话的一个过程模型。

3. 画出简单图书馆问题的一个数据流程图。

4. 画出工厂问题的数据流程图。

5. 画出杂货店问题的数据流程图。

6. 画出二元树的一个对象模型。

7. 画出二元树对象模型的一个实例图。

8. 画出杂货店问题的一个对象模型。

9. 画出工厂问题的一个对象模型。

10. 编写例子2.11中的读者借书的其他场景。

11. 画出具有一个主菜单、一个带有文件打开命令的文件菜单以及在每个菜单上有一条退出命令的图形用户界面的状态图。假定每次只能够打开一个文件。

12. 扩展图2-17中图书馆问题的对象模型，使其包括一个预定对象，以便读者能够预定一本所有副本已经借出的书籍。

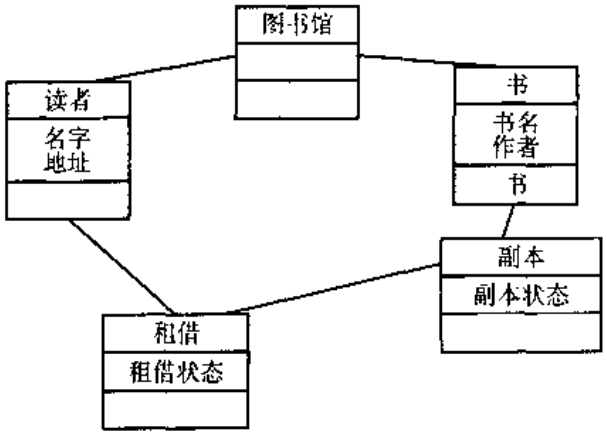


图2-17 图书馆问题的对象模型

13. 建立一个能够预定书籍的图书馆问题的状态机。

习题答案

1. 软件生命周期模型与过程模型之间的区别是什么?

软件生命周期（SLC）模型给出软件开发的主要阶段及主要的交付内容。而过程模型描述基础任务、必需的和生成的制品以及参与者负责的每个基础任务。

2. 描述性过程模型与事先描述性过程模型之间的区别是什么?

描述性过程模型描述软件开发中已经发生了什么事情。它通常是作为事后分析的结果而建立的。事先描述性过程模型描述在软件开发过程中应该做什么，包括对错误情况的响应。

3. 为什么在事先描述性过程模型中判断比在描述性过程模型中判断更常见?

判断更常用于事先描述性过程模型，因为事先描述过程模型要尽量覆盖可选择情形下所做的工作。因此，需要给出用来确定下一步做什么的判断。描述性过程模型描述已经发生的事情，而且一般不包含选择性的动作。

4. 在一个过程模型中任务为什么应该用制品分隔?

如果一个过程跟在另一个过程之后，则第二个过程需要来自第一个过程的信息或文档（这些信息或文档应该标出和文档化）。如果不是这样，这两个过程是相互独立的，一个过程不应该跟在另一个过程之后。

5. 为什么过程模型中的任务在输入其制品之前不能开始?

如果一个过程依赖于来自另一个过程的信息，则第二个过程在第一个过程结束前不能开始执行。有的过程表示法允许两个任务并行执行，此时，第一个过程必须在第二个过程开始之前开始，而第二个过程必须在第一个过程结束之后结束。

6. 为什么过程模型中的每个结点必须具有一条从开始结点到它本身的路径，以及一条从它自身到终止结点的路径?

如果从开始结点到每个结点没有一条路径，则过程模型中的某些结点永远无法到达，因此可以删去这些结点。如果从当前结点没有到终止结点的路径，则存在一个无限循环。上述两种情况都是不令人满意的，需要对它们进行研究。

7. 在例子2.3中，一个人员怎样才能够区分仅发生在所有单元测试完成之后的测试评审与

发生在每个模块已经进行了单元测试后的测试评审？

测试结果箭头上的标记表示单元测试模块的输出具有一个以上的执行结果。因此，这表明测试评审仅发生在测试多个单元之后。

8. 关于控制流，数据流程图有哪些规定？

数据流程图并未规定控制流。但从某些时序信息就可以推断出来。

9. 一个petri网激发位置何时激发？

petri网激发位置在该激发位置的每个输入结点上都有一个权标时激发。

10. 在一个petri网激发位置激发时会发生什么？

放置一个权标到该激发位置的每个输出结点上。

11. 问题领域和解决方案空间之间的区别是什么？

问题领域为现实世界的一部分，由现实世界中的实体组成。而解决方案空间由解决方案实现中的软件实体组成。

12. 对象模型从需求阶段到设计阶段有何改变？

开始时，对象是问题领域中的实体。随着开发进入设计阶段，这些对象变成了解决方案空间中的实体。

13. 将下面的关系按继承关系、集合关系或一般关联进行分类。

小汽车——Lincoln Town小汽车	继承
人员——学生	继承
图书馆——图书馆读者	集合
书——副本	一般关联
小汽车——驾驶员	一般关联
读者——书籍租借	一般关联
班级——学生	集合

14. 将下列各项分为类或类的实例：

我的汽车	实例
人员	类
Fred	实例
交通工具	类
教授	类

CIS部门 实例

15. 场景与显示了所有可能动作序列的状态图之间有什么关系?

场景仅是通过部分或全部状态图的一条路径。

16. 在一个相互作用图中，调用类还是被调用类位于箭头的头部?

被调用类位于箭头的头部。箭头上的功能必须是箭头头部处的类的一个功能。

17. 解释为什么集合关系是问题领域中的关系而不是实现领域中的关系?

在集合关系与其他关联关系的实现中不存在差异。事实上，很难确定某些关系是集合关系或不是集合关系。例如，很显然一辆小汽车是小汽车零件的一个集合。但是，是否应该将商店视为顾客的一个集合就不那么明显了。

18. 为什么数据流程图没有关于结点之间的可达性的规则?

数据流程图不给出控制。因此，在一个数据流程图中可能不连接两个过程。如果每个过程都使用不同的输入数据，并生成不同的输出数据，而且一个过程的输出不用做另一个过程的输入，那么在它们之间就没有弧。

补充问题答案

1. 画出油漆房间墙壁这个任务的一个过程模型。该过程包括下列任务：选择颜色、购买油漆、清洁墙壁、搅拌油漆、油漆墙壁。

见图2-18。

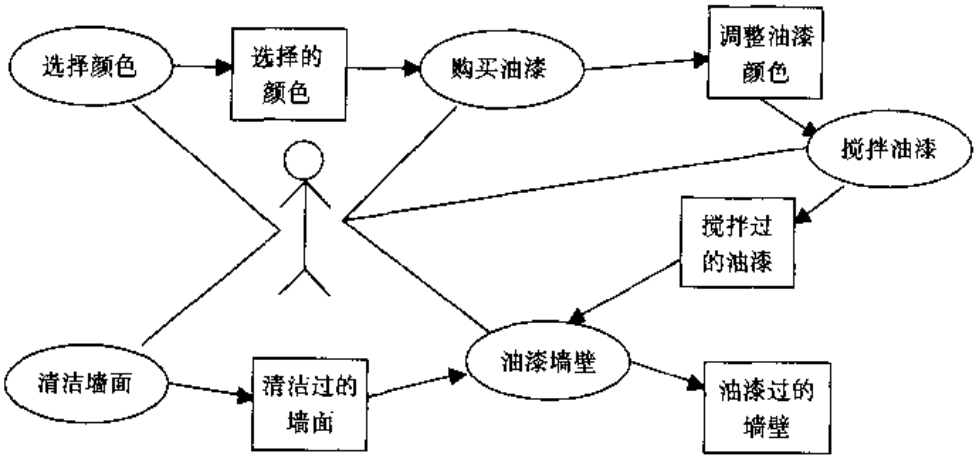
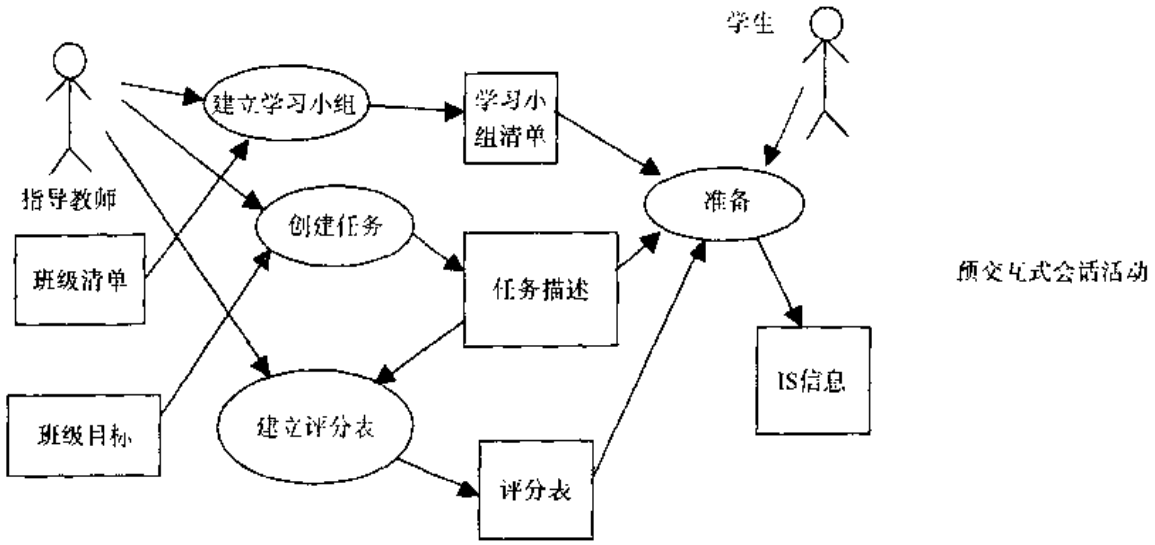


图2-18 油漆墙壁的过程模型

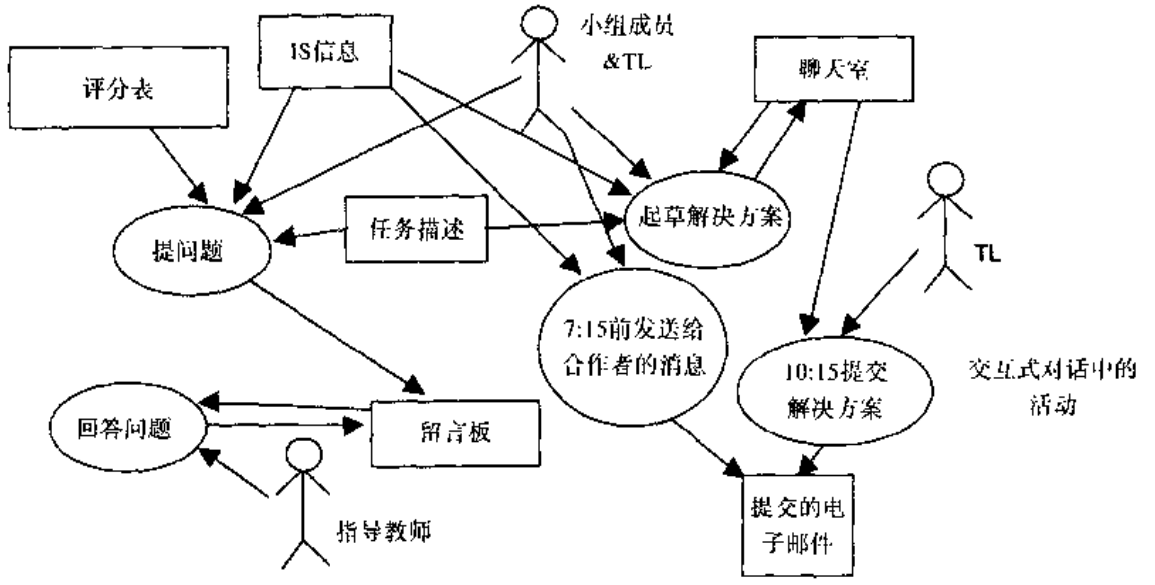
2. 作者在讲授一门有远程学习的学生参与的课程时使用了交互式会话。作者把学生划分

成小组并在Web页上贴出问题，各小组利用聊天室讨论问题，利用留言板向指导教师提问，并通过电子邮件提交解决方案。然后，指导教师利用评分表对学生的解决方案进行评分。请画出交互式会话的一个过程模型。

见图2-19。



准备交互式对话的过程模型



交互式对话中任务的过程模型

图2-19 交互式会话的过程模型

3. 画出简单图书馆问题的一个数据流程图。

见图2-20。

4. 画出工厂问题的数据流程图。

见图2-21。

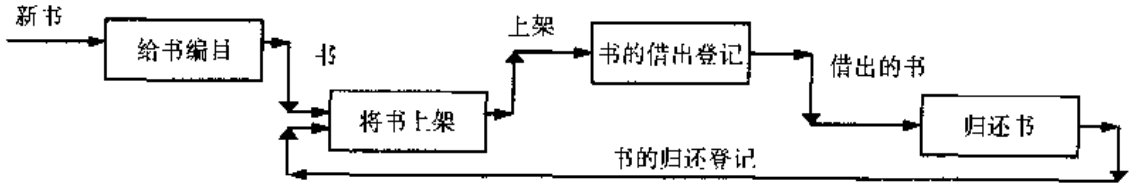


图2-20 图书馆问题的数据流程图

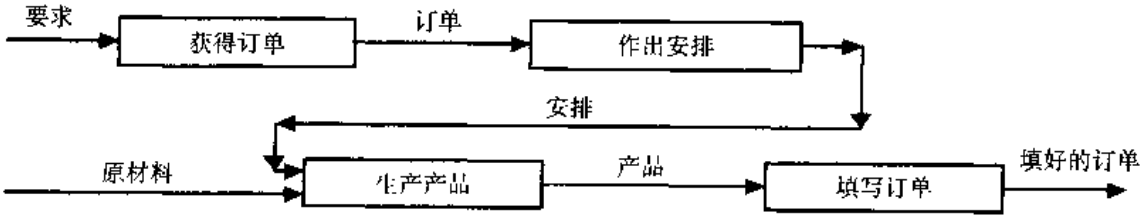


图2-21 工厂问题的数据流程图

5. 画出杂货店问题的数据流程图。

见图2-22。

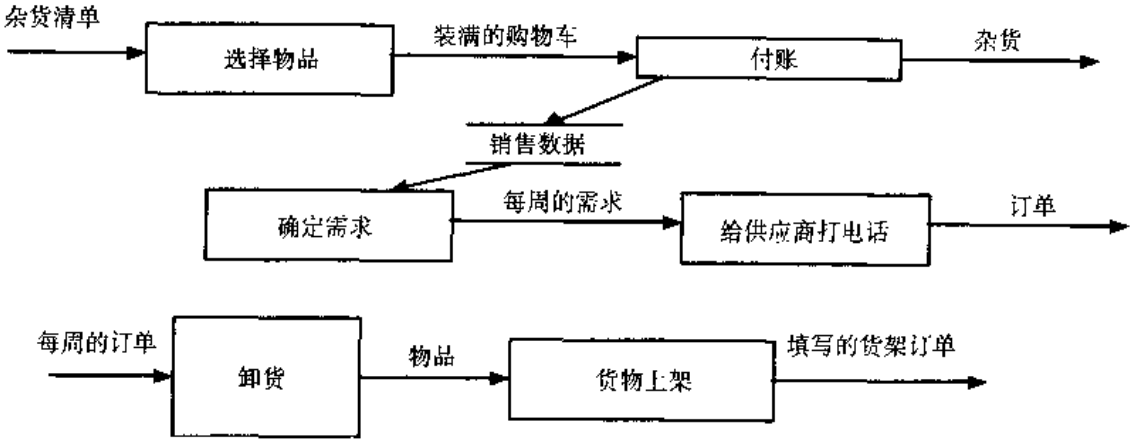


图2-22 杂货店数据流程图

6. 画出二元树的一个对象模型。

见图2-23。

7. 画出二元树对象模型的一个实例图。

见图2-24。

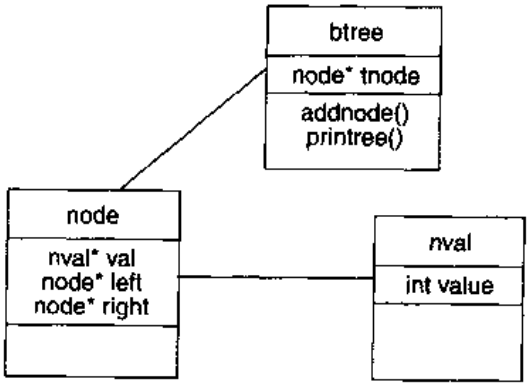


图2-23 二元树对象模型

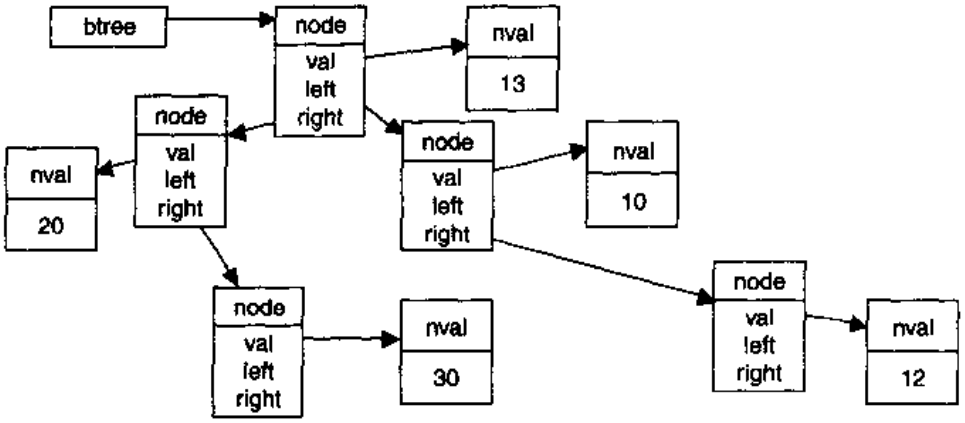


图2-24 二元树实例图

8. 画出杂货店问题的一个对象模型。

见图2-25。

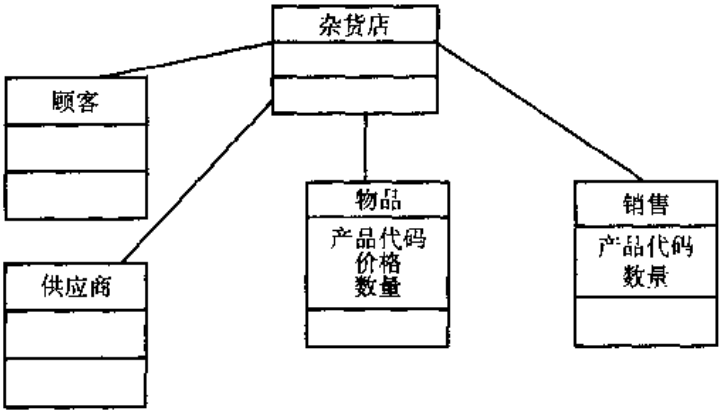


图2-25 杂货店问题的对象模型

9. 画出工厂问题的一个对象模型。

图见2-26。

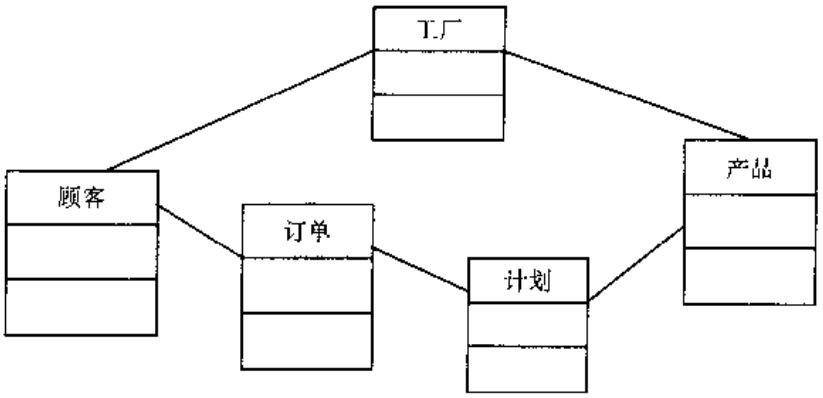


图2-26 工厂对象模型

10. 编写例子2.11中的读者借书的其他场景。

Fred到图书馆，但找不到想借的书。

Fred到图书馆并登记借走两本书。然后，他又回到图书馆登记借出了另外三本书。Fred按时归还了第二次借的三本书。Fred逾期还第一次借的两本书。

11. 画出具有一个主菜单、一个带有文件打开命令的文件菜单以及在每个菜单上有一条退出命令的图形用户界面的状态图。假定每次只有一个文件能够打开。

见图2-27。

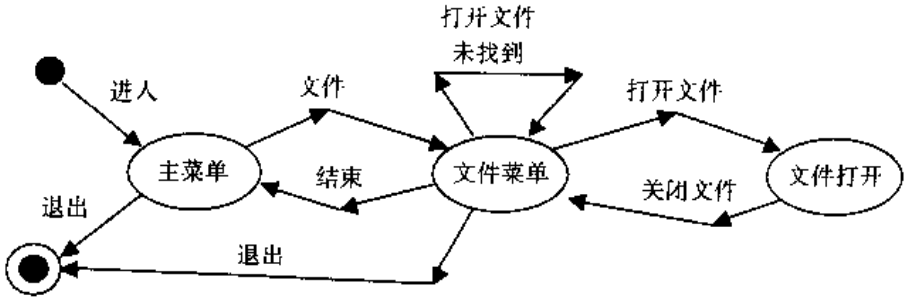


图2-27 GUI的状态图

12. 扩展图2-17中图书馆问题的对象模型，使其包括一个预定对象以便读者能够预定一本所有副本已经登记借出的书籍。

见图2-28。

13. 建立一个能够预定书籍的图书馆问题的状态机。

见图2-29。

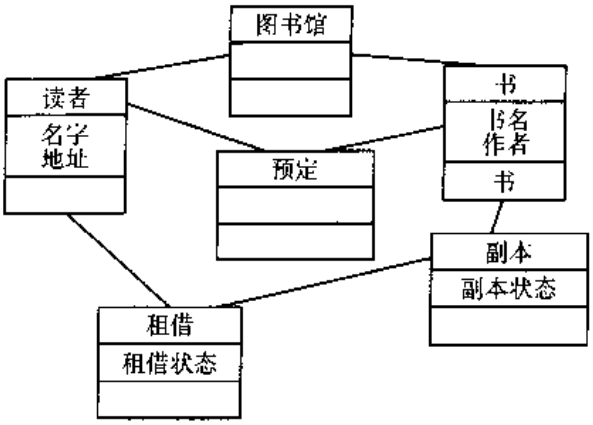


图2-28 图书馆对象模型

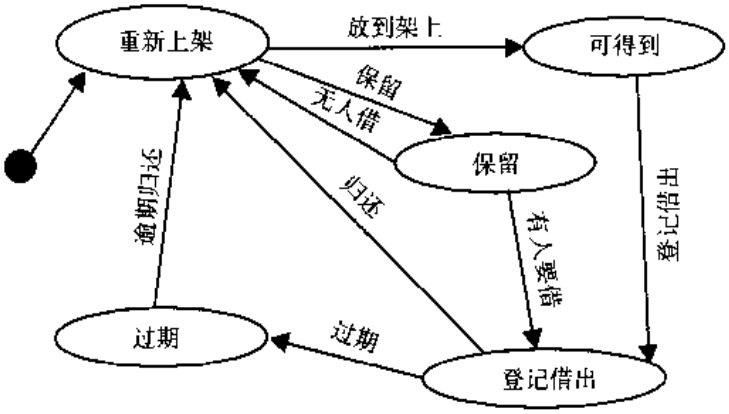


图2-29 图书馆问题的状态机

第3章　软件项目管理

3.1　概述

虽然“经理”一词可能会让许多管理人员想起“Dilbert”的连环画，不过，管理确实很重要。软件项目管理是规划、指导、激励和协调一组职业人员完成软件开发的重要任务。软件项目管理通常会使用许多通用的管理的概念，但也有某些概念是软件开发所独有的。项目的可见度就是这样的一个概念。

在软件开发中缺乏软件产品的可见度将会使其难于管理。在许多其他的领域中，很容易看到进展顺利与否。许多软件项目在完成90%时就停止了。如果问任何一个程序设计人员，他刚发现的问题是否是软件中的最后一个问题，回答都是斩钉截铁的，是。软件管理中许多技术的目的就是弥补缺乏的可见度。

3.2　管理方法

软件项目管理中一个基本的问题是过程或项目是否就是要管理的基本特性。在面向过程的管理中，强调的是软件生命周期中小任务的管理。而在项目管理中，重视的是完成项目的小组。这导致了观点的重要差异。在过程管理方法中，如果小组不遵照所描述的软件生命周期进行工作，将很难管理。而在项目管理方法中，成功与失败可直接归因于小组。

3.3　小组方法

将人员组织成精干高效的小组可能是一件很困难的任务。让一个小组自己建立内部管理模式存在一定的风险。根据项目或小组成员来选择一个小组的组织方式可能会有助于免除灾难。

小组的一个方面是小组中组织结构的情况。虽然有的程序员小组可以独立地工作，但有的小组需要很强的组织结构才能取得进展。下一节要介绍的主程序员小组就是结构性很强的小组的一个例子。在一个结构性强的小组中，将各个小任务分配给每个成员。这些任务通常称为“小鹅卵石”，因为这些任务是一些小里程碑。在一个结构性差的小组中，任务通常要持续更长的时间，而且更不确定。

有的小组由具有类似技能的人员组成。这些小组通常在许多项目中一起工作。而另外一些小组由具有不同技能的人员组成，这些人员根据项目所需的特殊技能而组织在一起。这样的小组通常称为一个矩阵组织。

主程序员小组

IBM建立了主程序员小组的概念。它给小组的成员赋予了特殊的角色。主程序员是最好的程序员并领导小组。小组中也为进行文档和记录等工作雇佣一些非程序人员。小组中还包括初级程序员，由主程序员进行指导。

例子3.1

画出一个有层次结构的小组的高级过程模型。如图3-1所示。

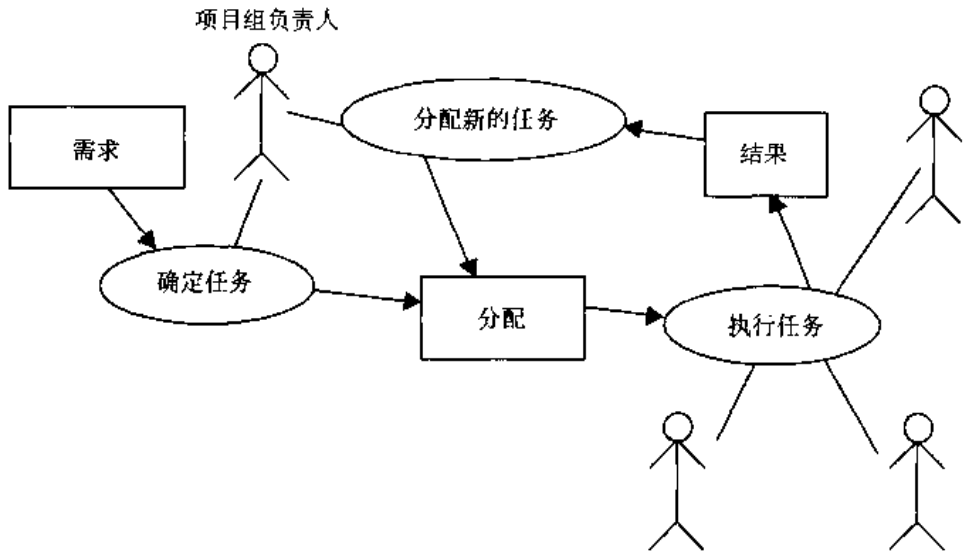


图3-1 有层次结构的小组的高级过程模型

例子3.2

WRT公司有一个IT部门，该部门具有几个有经验的软件开发人员和许多新程序员。IT经理决定利用高度结构化的小组，用过程方法来管理该小组。每个小组将由一个有经验的软件开发人员领导。每周给每个小组成员分配一组任务。小组领导将不断检查进度并分配新任务。

3.4 重要准则

软件开发的大多数研究已经确立了一些重要的准则，这些准则有助于用户进行成功的软件开发。下面16条重要准则来自Software Project Managers Network（www.spmn.com）：

- 进行连续的风险管理。

- 根据经验估计成本和进度。
- 用矩阵进行管理。
- 追踪实现值。
- 追踪缺陷（相对于质量目标来说的缺陷）。
- 将人员作为最重要的资源。
- 采用生命周期配置管理（配置管理工具将保存和维护源代码和文档的多个版本。用户可以检索任何版本）。
- 管理和跟踪需求。
- 使用基于系统的软件设计。
- 确保数据与数据库的互操作性。
- 定义和控制接口。
- 设计第一，编码第二。
- 评估重用的风险和代价。
- 审查需求和设计。
- 把测试作为一个连续的过程进行管理。
- 常常进行编译和烟幕测试。

例子3.3

WRT公司的IT经理需要一个帮助没有经验的软件开发人员成功进行软件开发的软件过程。他使用了最好的准则清单以保证他的软件过程包含重要的活动。

准则1：进行连续的风险管理。清单中包含了整个生命周期中的过程步骤，其中标识和估计了可能的风险，并且包含了减少风险的任务。

准则2：据经验估计成本和进度。清单中包含了在生命周期开始时估计成本的一个过程步骤，还包含了在生命周期中重新估计成本的几个步骤。其中还包含获得进一步估计所需要的数据的步骤。

准则3：用矩阵进行管理。这位经理选择了矩阵方法并在清单中包含了用于矩阵记录的步骤和基于矩阵估计进度的步骤。

准则4：追踪实现值。清单中包含计算实现值（参阅后面的内容）和公布计算结果的步骤。

准则5：追踪缺陷（相对于质量目标来说的缺陷）。对所接收到的缺陷报告数目建立了目标。其中也包括公布缺陷报告的过程步骤。

准则6：将人员作为最重要的资源。经理审查了整个软件过程，以考虑对编程人员产生的影响。

准则7：采用生命周期配置管理。经理在软件过程中为所有文档使用配置管理工具，并且将所有文档输入配置管理工具并在其中进行更改。

准则8：管理和跟踪需求。清单中包括从用户处获得需求的步骤和跟踪当前开发阶段每个需求的步骤。

准则9：使用基于系统的软件设计。清单中包含了保证基于系统的设计的步骤。

准则10：确保数据与数据库的互操作性。清单中包括检查数据与数据库间互操作性的步骤。

准则11：定义和控制接口。清单中包括定义接口和接口基线的步骤。

准则12：设计第一，编码第二。清单中包括设计评审步骤。

准则13：评估重用的风险和代价。清单中包括标识可能重用的范围的步骤以及评估成本和风险的步骤。

准则14：审查需求和设计。清单中包括需求与设计两个阶段的审查步骤。

准则15：把测试作为一个连续的过程进行管理。清单中包括所有阶段的测试步骤。

准则16：常常进行编译和烟幕测试。实现阶段包括经常进行的测试步骤。

3.5 能力成熟度模型

Software Engineering Institute（www.sei.cmu.edu）开发了能力成熟度模型（Capability Maturity Model）。软件工程能力成熟度模型（SE-CMM，Software Engineering Capability Maturity Model）是用来评定一个组织的软件开发过程的。根据一个组织的习惯、方法和组织方式的评估可以将该组织分为以下几种级别：

- 级别1：初级　这是最低级别，该级别的特征为混乱无序。
- 级别2：可重复　这个开发能力级别包括项目成本、安排和功能的跟踪。这种能力能够一再取得初期的成功。
- 级别3：定义　这个级别具有一个已经进行了文档化和标准化的已定义软件过程。所有开发都利用这个标准过程来完成。
- 级别4：管理　这个级别定量地管理开发过程和产品。
- 级别5：优化　这个级别使用定量信息来不断地改进和管理软件过程。

大多数组织起初都被评定为级别1。要升到更高级别，还应在组织和管理方面进行大量的工作。只有少数组织能达到级别5。

3.6 个人的软件过程

Watts Humphrey（W.Humphrey. *Introduction to the Personal Software Process*. Addision-Wesley, 1997）开发了个人软件过程以改进软件工程师个体的技能。他的方法具有监控和度量个体技能的单独的维护的个人时间记录。这样便可以度量个人的生产率。生产率的常用度量方法是每天所产生的代码行数（LOC/day）。此外，该方法还对错误进行计数和记录。这使得个人能够知道在何处犯错误，并评估不同的技术对于其生产率和错误率的影响。而且，生产率还可用来估算所提出安排的合理性。

例子3.4

程序员X记录了如下的时间日志。

日 期	开 始	停 止	中 断	增 量	任 务
1/1/01	09:00	15:30	30分钟午餐	360	编50行代码
1/3/01	09:00	14:00	30分钟午餐	270	编60行代码
1/4/01	09:00	11:30		150	编50行代码
	12:00	14:00		120	测试

此程序员花了360 + 270 + 150 + 120 = 900分钟编写和测试了160行代码的一个程序。假定每天工作5小时（每天300分钟），那么X编写160行代码足足花了三天时间。其生产率是53LOC/day。如果X的经理计划让他编写一个有1 000行代码的程序，X就能够估计出他大约要花4周的时间才能完成这个项目。

3.7 实现值分析

度量软件项目进展的一种方法是计算项目已经完成了多少。这称为实现值分析。它基本上是已经完成的估计时间的百分比。也可以计算别的量。

虽然这种方法是基于估计工作量的，但也可以基于其他可以估计且与进度有关的量。

3.7.1 基本的量

- 工作的预计成本（BCW）：每项工作任务的预计工作量。

- 所计划的工作的预计成本（BCWS）：计划按指定时间完成的每项工作任务的预计工作量之和。
- 完成预算（BAC）：BCWS的总量，因此是项目的总工作量的估计。
- 计划值（PV）：分配给特定工作任务的总预计工作量的百分比，PV = BCW/BAC。
- 所完成工作的预计成本（BCWP）：已经按指定时间完成的工作任务的预计工作量之和。
- 所完成工作的实际成本（ACWP）：已经完成的工作任务的实际工作量之和。

3.7.2 进度指示器

- 实现值（EV）= BCWP / BAC
  = 完成的所有工作任务的PV之和 = PC
  = 完成的百分比
- 进度性能指标（SPI）= BCWP / BCWS
- 进度偏差（SV）= BCWP– BCWS
- 成本性能指标（CPI）= BCWP/ACWP
- 成本偏差（CV）= BCWP– ACWP

例子3.5

LMN公司的项目正进行到一半。下面的工作记录显示了项目的当前状态。

工作任务	估计工作量 (程序员－天)	迄今为止的实际工作量 (程序员－天)	估计完成日期	实际完成日期
1	5	10	1/25/01	2/1/01
2	25	20	2/15/01	2/15/01
3	120	80	5/15/01	
4	40	50	4/15/01	4/1/01
5	60	50	7/1/01	
6	80	70	9/01/01	

BAC为估算之和。BAC = 330天。BAC为总工作量的估算。在01年4月1日，任务1、2和4已经完成。BCWP为这些任务的BCWS之和。因此，BCWP为70天。实现值（EV）为70/330，即21.2%。在01年4月1日，计划完成任务1和2，而任务1、2和4实际上已经完成。所以，BCWP为70天而BCWS为30天。因此SPI为70/30，即233%。SV = 70天–30天 = +40天，即提前了40个程序员－天。ACWP为任务1、2、4的实际工作量之和。因此，ACWP为80个程序员－天。CPI为70/80 = 87.5%。CV = 70程序员－天–80程序员－天=–10程序员－天，即延迟了10个程序员－天。

例子3.6

假如01年7月1日任务3已经完成，实际使用了140天，那么BCWP为190而EV为190/330，即57.5%。在01年7月1日，任务1、2、3、4都已实际完成。这样，BCWP为190天而BCWS为250天。所以，SPI为190/250 = 76%。SV为190个程序员－天–250个程序员－天=–60个程序员－天，或者说推迟了60个程序员－天。ACWP为任务1、2、3、4的实际工作量之和。所以，ACWP为220个程序员－天。本来计划完成任务1到5，但实际上只完成了任务1到4。CPI为190/220 = 86.3%，CV为190–220，或者说推迟了30个程序员－天。

3.8 错误跟踪

错误跟踪是一种很好的管理方法，它用于记录已经出现的错误以及错误间隔时间（发生错误之间的时间）。可利用这种办法来作出何时发行软件的决策。跟踪和公布错误率的另一作用是使软件开发人员意识到错误的严重性并减少错误。也可以通过错误数据看到在软件过程中对软件进行改动的作用。此外，使错误和错误检测可视化能够促使测试员和软件开发员将减少错误作为一个目标。

错误率是错误间隔时间的倒数。如果每两天发生一次错误，则即时错误率为0.5个错误/天。当前即时错误率是当前错误率的一种很好的估计方法。如果在发现错误时不去除导致错误的因素，则累积错误率（将发现的所有错误之和除以总时间）将是对未来错误率的一个很好的估计。通常，如果修正大多数错误（去除错误因素），那么错误率应该下降且错误间隔时间应该增大。将这个数据绘出能够反映出错误率（每单位时间出现的错误数）趋势。把相应的点用直线连接起来是显示趋势的一种很有效的办法。趋势线可用来估计未来的错误率。当趋势线与X轴相交时，错误率的估计为零，或者说没有更多的错误了。如果X轴为错误的编号，则X轴上的截距可用做软件中未来错误总数的估计。如果X轴为测试使用的时间，则此截距为排除所有错误所需的测试时间。后面这条线下的面积为软件中原始错误数的估计。

例子3.7

考虑下面的错误数据（假定为错误的时间间隔）：4、3、5、6、4、6、7。即时错误率为错误间隔时间的倒数：0.25、0.33、0.20、0.17、0.25、0.17、0.14。相对于错误数绘出这些数据，得到一条下降的曲线，如图3-2所示。这表示实际错误率在下降。

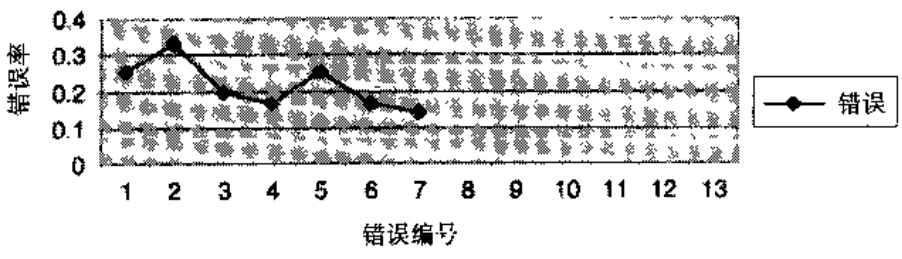


图3-2 错误率图

将点串起来的直线大约在11处与X轴相交。这意味着错误率在第11个错误时将为零，所以软件中错误的总数估计应该为11个。因为已经发现了7个错误，这表示软件中可能还有另外4个错误。

3.9 事后回顾

软件开发的一个重要方面是从所犯的错误和所取得的成功中学习新的东西。在软件开发过程中，这称为事后回顾。这个活动由来自开发组和用户组的关键人物组成。问题由质量、计划和软件过程组成。重要的是每个人都能自由地发表自己的看法。回顾后应该产生和发布一个正式的报表。这个报表不需要很整洁。

例子3.8

JKL公司的事后分析报表，如下所示。

<table>
<tr><td>项目名
项目X</td><td>开始日期：9月5日</td><td>完成日期：12月8日</td></tr>
<tr><td>管理度量</td><td>大小
<table><tr><td>估计</td><td>实际</td></tr><tr><td>3 000 LOC</td><td>5 000 LOC</td></tr></table></td><td>工作量
<table><tr><td>估计</td><td>实际</td></tr><tr><td>12 000 min</td><td>10 000 min</td></tr></table></td></tr>
<tr><td>对估计的主观
评价</td><td>好
总工作量很接近</td><td>不好
低估了重要工作量</td></tr>
<tr><td>对过程的主观
评价</td><td>好</td><td>不好
小组成员未按时完成分配的任务</td></tr>
<tr><td>对计划的主观
评价</td><td>好</td><td>不好
重要工作没有足够时间完成</td></tr>
<tr><td>质量</td><td>发现错误
<table><tr><td>需求</td><td>设计</td><td>单元测试</td><td>集成</td><td>事后排除</td></tr><tr><td></td><td></td><td></td><td>30</td><td></td></tr></table></td><td><table><tr><td></td><td>平均</td><td>最大</td></tr><tr><td>mccabe</td><td>4</td><td>30</td></tr><tr><td>方法/类</td><td>6</td><td>10</td></tr><tr><td>属性/类</td><td>10</td><td>15</td></tr><tr><td>LOC/类</td><td>150</td><td>500</td></tr></table></td></tr>
</table>

（续）

项目名 项目X	开始日期：9月5日	完成日期：12月8日
对质量的主观评价	好	不好 没有很好地测试系统
问题：最初需求不明确	描述： 输入文件格式 -开始就是错的	影响： 浪费了两周时间
问题：	描述：	影响：
问题：	描述：	影响：
问题：	描述：	影响：

习题

1. 可见度的含义是什么？

2. 过程方法和项目方法之间的区别是什么？

3. 对于一个与以前的项目很不相同的新项目，用过程管理方法更好还是项目管理方法更好？

4. 构造许多小的“小鹅卵石”任务的优点是什么？

5. 在一个项目中，可以减少哪些实现值进展度量？

6. 哪些实现值进展度量大于1比较好？

7. 使用SPI和CPI的倒数的优点是什么？

补充问题

1. 画出结构性较差并且依赖于小组讨论确定方向和解决问题的项目小组的一个过程模型。

2. 使用下面的时间记录，以LOC/天为单位计算程序员的生产率。假定项目1为120 LOC，项目2为80 LOC。

日 期	开 始	停 止	中 断	增 量	任 务
2/1/01	08:30	16:30	60分钟午餐		项目1编码
2/2/01	09:00	17:00	30分钟午餐		项目1编码

（续）

日 期	开 始	停 止	中 断	增 量	任 务
2/5/01	09.00	17:30	30分钟午餐 60分钟宏模块测试		项目2编码
2/6/01	07:30	12:00			项目2编码

3. 利用下面的作业记录计算所有基本量和进度指示器。项目按计划完成了吗？假定当前日期为01年5月1日。

工作任务	估计工作量 (程序员－天)	迄今为止的 实际工作量 (程序员－天)	估计完 成日期	实际完 成日期
1	50	70	1/15/01	2/1/01
2	35	20	2/15/01	2/15/01
3	20	40	2/25/01	3/1/01
4	40	40	4/15/01	4/1/01
5	60	10	6/1/01	
6	80	20	7/1/01	

4. 利用电子表格计算下面项目的PV和进度指示器（从1月到9月，间隔时间为半个月）。

工作任务	估计工作量 (程序员－天)	迄今为止的 实际工作量 (程序员－天)	估计完 成日期	实际完 成日期
1	30	37	1/1	2/1
2	25	24	2/15	2/15
3	30	41	3/1	3/15
4	50	47	4/15	4/1
5	60	63	5/1	4/15
6	35	31	5/15	6/1
7	55	58	6/1	6/1
8	30	28	6/15	6/15
9	45	43	7/1	7/15
10	25	29	8/1	8/15
11	45	49	8/15	9/1

5. 教授有40份家庭作业和40份考卷要审阅。审阅考卷的时间一般是审阅家庭作业的时间的三倍。请计算每份家庭作业和每个考卷的PV。在5个小时后，如果该教授审阅完了一半考卷，为了完成全部审阅工作，他还需要多长时间？

6. 给出下面的错误间隔时间（即错误发生之间的时间），用图形估计总的原始错误数目以及完全排除所有错误的时间：6、4、8、5、6、9、11、14、16、19。

7. 项目是1月1日开始的，应该在6月1日完成。现在是3月1日。请完成下表。计算EV、SPI、SV和CV。判断项目是否按时完成。证明你的答案。说明你的工作。

工作#	估计时间	实际花费的时间	PV	预定日期	完 成
1	30	10		2月1日	
2	20	30		3月1日	是
3	50	30		5月1日	是
4	100	5		6月1日	

习题答案

1. 可见度的含义是什么？

可见度是能够看到项目进度如何的属性。

2. 过程方法和项目方法之间的区别是什么？

过程方法类似于一条装配线，其中每个人有一项需要完成的任务。开发人员可能在多个项目中做同样的工作，如测试小组或设计小组的工作。而项目方法强调让小组负责一个项目开发中的所有工作。

3. 对于一个与以前的项目很不相同的新项目，用过程管理方法更好还是项目管理方法更好？

过程管理适用于容易理解的项目。而用强调项目成功的项目方法来管理新的很不相同的项目可能效果更好。

4. 构造许多小的“小鹅卵石”任务的优点是什么？

如果错过了最终期限或任务，项目就被延迟了。最终期限之间的时间越短，那么知道一个项目是否延迟的时间就越早。据说在管理人员能够知道延迟前，一个项目可以只滑过一个任务的时间长度。

5. 在一个项目中，可以减少哪些实现值进展度量？

除了实现值以外的所有进展度量都应减少，只有实现值应当增加。

6. 哪些实现值进展度量大于1比较好？

对于SPI和SV，值大于1表示比计划完成得要多。对于CPI和CV，值大于1表示工作量小于所估计的工作量。所以，这4个量的值大于1时都比较好。

7. 使用SPI和CPI的倒数的优点是什么？

它们的倒数都可用做投影工具。如果SPI的倒数为2，表示项目所花时间为估计时间的两倍。如果CPI的倒数为2，表示项目的工作量为估计工作量的两倍。

补充问题答案

1. 画出结构性较差并且依赖于小组讨论确定方向和解决问题的项目小组的一个过程模型。

见图3-3。

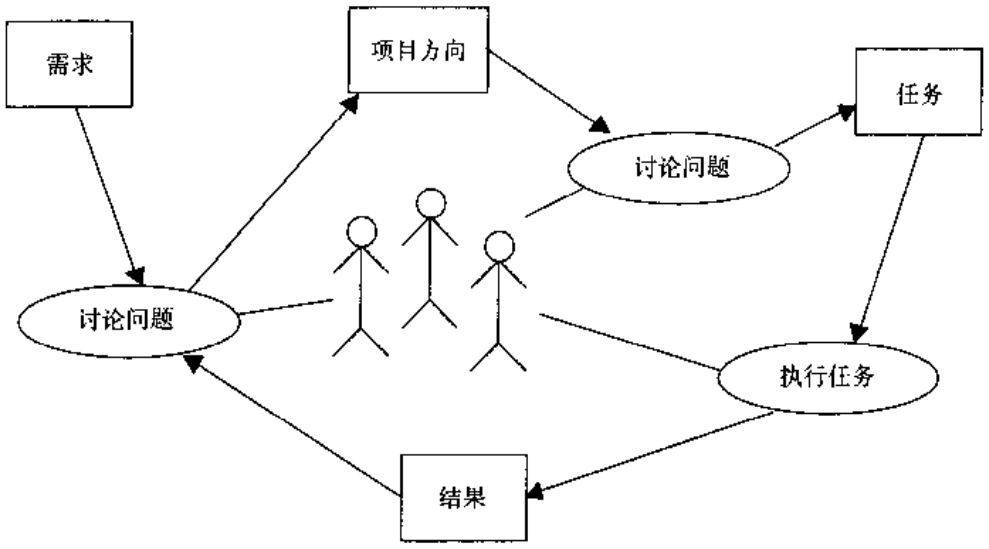


图3-3 结构性较差的小组的过程模型

2. 使用下面的时间记录，以LOC/天为单位计算程序员的生产率。假定项目1为120 LOC，项目2为80 LOC。

日 期	开 始	停 止	中 断	增 量	任 务
2/1/01	08:30	16:30	60分钟午餐		项目1编码
2/2/01	09:00	17:00	30分钟午餐		项目1编码
2/5/01	09:00	17:30	30分钟午餐 60分钟宏模块测试		项目2编码
2/6/01	07:30	12:00			项目2编码

第一天的增量时间为8小时-午饭的1小时 = 420分钟；第二天的增量时间为8小时-30分钟 = 450分钟。因此，项目1的生产率为120LOC/870分钟 = 120 LOC/2.175天 = 55LOC/程序员－天（假定每个程序员每天工作400分钟）。第3天和第4天的增量时间为7小时和4.5小时，即690分钟。相应的生产率为80 LOC/1.725天 = 46.4 LOC/程序员－天。概括起来有，程序员的平均生产率为200 LOC/3.9天 = 51.3 LOC/程序员－天。

3. 利用下面的作业记录计算所有基本量和进度指示器。项目按计划完成了吗？假定当前日期为01年5月1日。

任务工作	估计工作量 (程序员－天)	迄今为止的实际工作量 (程序员－天)	估计完成日期	实际完成日期
1	50	70	1/15/01	2/1/01
2	35	20	2/15/01	2/15/01
3	20	40	2/25/01	3/1/01
4	40	40	4/15/01	4/1/01
5	60	10	6/1/01	
6	80	20	7/1/01	

BCWS为 50 + 35 + 20 + 40 = 145 程序员－天。BAC为 50 + 35 + 20 + 40 + 60 + 80 = 285 程序员－天。工作任务的计划值（PV）为17.5%、12.3%、7.0%、14.0%、21.1%、28.1%。实现值为17.5% + 12.3% + 7% + 14% = 50.7%。01年5月1日的BCWP与这个例子中的BCWS相同，因为计划的工作已经完成。因此，SPI = 145/145 = 1。

进度偏差为145-145 = 0。成本性能指标 = 145/170 = 85.3%。这表示实际工作量大于估计工作量。成本偏差为145-170 = －25。这也表示实际工作量比估计工作量更大。

这个项目看来是按计划进行了，但成本比所计划的要大。

4. 利用电子表格计算下面项目的PV和进度指示器（从1月到9月，间隔时间为半个月）。

任务工作	估计工作量 (程序员－天)	迄今为止的实际工作量 (程序员－天)	估计完成日期	实际完成日期
1	30	37	1/1	2/1
2	25	24	2/15	2/15
3	30	41	3/1	3/15
4	50	47	4/15	4/1
5	60	63	5/1	4/15
6	35	31	5/15	6/1

（续）

任务工作	估计工作量（程序员－天）	迄今为止的实际工作量（程序员－天）	估计完成日期	实际完成日期
7	55	58	6/1	6/1
8	30	28	6/15	6/15
9	45	43	7/1	7/15
10	25	29	8/1	8/15
11	45	49	8/15	9/1

```
    bcw   pv   acw  sched  actual
 1   30 0.070  37   1-Jan  1-Feb
 2   25 0.058  24  15-Feb 15-Feb
 3   30 0.070  41   1-Mar 15-Mar
 4   50 0.116  47  15-Apr  1-Apr
 5   60 0.140  63   1-May 15-Apr
 6   35 0.081  31  15-May  1-Jun
 7   55 0.128  58   1-Jun  1-Jun
 8   30 0.070  28  15-Jun 15-Jun
 9   45 0.105  43   1-Jul 15-Jul
10   25 0.058  29   1-Aug 15-Aug
11   45 0.105  49  15-Aug  1-Sep
BAC 430

       bcws bcwp acwp  ev   spi  sv  cpi  cv
 1-Jan   30    0    0 0.00    0 -30    0   0
15-Jan   30    0    0 0.00    0 -30    0   0
 1-Feb   30   30   37 0.07 1.00   0 0.81  -7
15-Feb   55   55   61 0.13 1.00   0 0.90  -6
 1-Mar   85   55   61 0.13 0.65 -30 0.90  -6
15-Mar   85   85  102 0.20 1.00   0 0.83 -17
 1-Apr   85  135  149 0.31 1.59  50 0.91 -14
15-Apr  135  195  212 0.45 1.44  60 0.92 -17
 1-May  195  195  212 0.45 1.00   0 0.92 -17
15-May  230  195  212 0.45 0.85 -35 0.92 -17
 1-Jun  285  285  301 0.66 1.00   0 0.95 -16
15-Jun  315  315  329 0.73 1.00   0 0.96 -14
 1-Jul  360  315  329 0.73 0.88 -45 0.96 -14
15-Jul  360  360  372 0.84 1.00   0 0.97 -12
 1-Aug  385  360  372 0.84 0.94 -25 0.97 -12
15-Aug  430  385  401 0.90 0.90 -45 0.96 -16
 1-Sep  430  430  450 1.00 1.00   0 0.96 -20
```

5. 教授有40份家庭作业和40份考卷要审阅。审阅考卷的时间一般是审阅家庭作业的时间的三倍。请计算每份家庭作业和每个考卷的PV。在5个小时后，如果该教授审阅完了一半考卷，为了完成全部审阅工作，他还需要多长时间？

假定审阅单位为1个家庭作业，则这个任务总共有 40 × 1 + 40 × 3 = 160个审阅单位。每份家庭作业的计划值为1/160 = 0.625%，而每份考卷的计划值为1.875%。在5小时后，完成了20个考卷的审阅，或完成了37.5%。这样，5/0.375 = 13.33小时为估计的总时间，或者说他还需要8.33个小时才能完成全部审阅工作。

6. 给出下面的错误间隔时间（即错误发生之间的时间），用图形估计总的原始错误数目以及完全排除所有错误的时间。

6、4、8、5、6、9、11、14、16、19

错误间隔时间的倒数为即时错误率。相对于错误编号绘出这些错误率得出一个图，它显示错误率趋于减少，如图3-4所示。

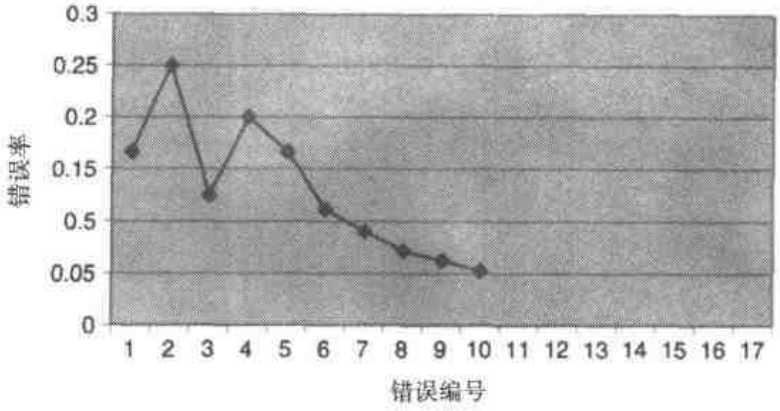


图3-4 错误率图形

画一条直线将显示X截距大约为15。用这个数作为原始错误总数的估计，可以估计出软件中还有5个错误。

也可以相对于经过时间（以前的错误间隔时间之和）绘出错误率，如图3-5所示。

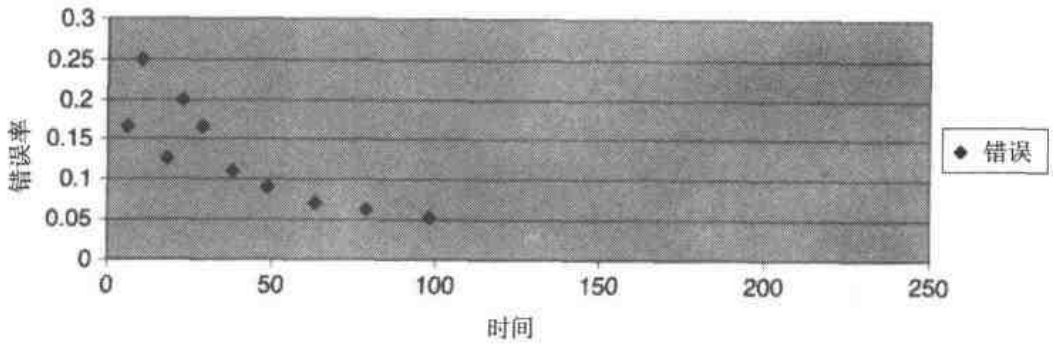


图3-5 错误率与经过时间

把这些点用直线连起来将得到X截距大约为160。这将给出排除所有错误的另外62单位的

测试时间。Y截距大约为0.25。则这条线下的面积为 $0.5 \times 160 \times 0.25$，或20个错误。这表示大约还有10个错误。这两种估计方法之差说明了这种错误估计方法是很粗略的。

7. 项目是1月1日开始的，应该在6月1日完成。现在是3月1日。请完成下表。计算EV、SPI、SV和CV。判断项目是否按时。证明你的答案。说明你的工作。

工作#	估计时间	实际花费的时间	PV	预定日期	完成
1	30	10		2月1日	
2	20	30		3月1日	是
3	50	30		5月1日	是
4	100	5		6月1日	

BAC = 200　BCWS = 50　BCWP = 70　ACWP = 60

EV = 70/200 = 0.35　SV = 70−50 =20　SPI = 70/50 = 1.4　CV = 70 − 60 = 10

此项目的进度提前了。

第4章　软件项目计划

4.1　项目计划

计划是完成工作的不可缺少的步骤，对于软件开发来说也不例外。要使软件开发取得成功需要进行计划。软件项目计划要确定需要完成什么任务，以什么样的顺序完成这些任务，以及为了完成这些任务需要什么资源。

4.2　WBS：工作分解结构

计划的首要工作之一是将大任务分解成小任务。这意味着要找出任务的可以标识的组成部分，还意味着找出可用来度量进展的交付内容和里程碑。

工作分解结构（WBS）应该是一个树形结构。顶层分解一般应该与组织机构中使用的生命周期模型（LCM）相匹配。下一层分解可以与组织机构的过程模型（PM）中的过程匹配。再下面的层次用来将任务划分成更小、更易于管理的任务。

以下是建立适当的工作分解结构的规则：

1) WBS必须为树形结构。WBS中不应该有循环。重复性的活动将在过程模型和/或生命周期模型中给出。

2) 每个任务和交付内容的描述必须是可理解和明确的。WBS可用来与项目小组成员进行沟通。如果小组成员误解了任务或交付内容，将会出问题。

3) 每个任务必须有一个完成标准（通常为一个交付内容）。必须有确定一个任务何时完成的办法，因为没有明确结束标志的子任务会使对进度的期望变成泡影。确定任务是否已经完成的这种标准就称为完成标准。它可以是一个交付内容，如项目的完整设计，然后，同级评审就可以确定该任务是否完成。

4) 必须标出所有交付内容（制品）。一个交付内容必须由某个任务产生，别的任务不会产生它。

5) 完成了所有任务必定意味着整个任务的完成。工作分解计划的目的是标出完成整个任

务所必须的子任务。如果遗漏了重要的任务或交付内容，则无法完成整个任务。

例子4.1

在面包的制作中，烹饪的生命周期模型如图4-1所示。

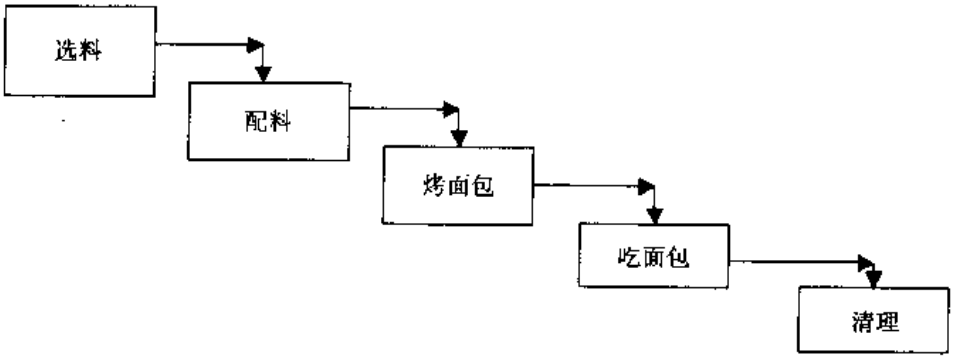


图4-1 面包烹饪的生命周期模型

烹饪的过程模型也可以如图4-2所示。

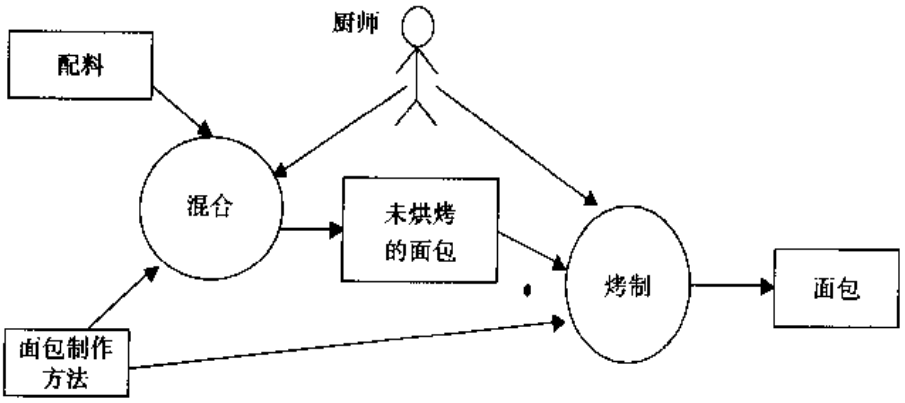


图4-2 面包烹饪的过程模型

各子任务如下：

选择配料，查对配料，混和配料，加水，加发酵粉，加少许面粉，做成软面（酵母和稀面），首先让它发酵，再加面粉，揉成团，再次发酵，做成胚，再次发酵，烘烤，切片，抹黄油，吃，清理。

可将这些子任务划分为生命周期模型的阶段以及过程模型的过程（最左边为LCM，缩进一层的为PM，缩进两层的为任务，右边的是交付内容）：

选料

选择配料	配料清单
查对配料	购物单
混和配料	
混合配料	混合好的配料
做面包	
和面	
加水	碗中的水
加酵母	发酵液体
加少量的面粉	有酵母的稀面
做成软面（酵母和稀面）	软面
第一次发酵	发酵后的软面
再加面粉并揉成团	揉成的生面团
第二次发酵	发酵后的生面团
做成胚	生面包胚
第三次发酵	发酵后的面包胚
烘烤	
烘烤	面包
吃	
切片	面包片
抹黄油	抹了黄油的面包片
吃	很好的味道
清理	
清理	清洁的厨房

例子4.2

XYZ项目小组要开发一套机器人使用的面部识别系统。此系统用来向机器人实验室的参观人员致意。它应该具备适当的可靠性以识别出以前见过的面孔。第一遍工作分解确立了下列子任务：

- 可行性
  - 确定视觉的可实现性。
  - 确定摄像机和软件的可用性。
  - 安排摄像机和视觉软件采购。
- 风险分析
  - 确定视觉风险。

- 需求

  说明需求。

- 设计

  设计原型。

  原型视觉。

- 实现

  对获取的图像进行编码。

  对图像处理进行编码。

  对图像比较进行编码。

  与其他机器人软件集成。

- 测试

  测试图像获取。

- 交付内容

  文档

有些子任务仍然是非常大的。这些任务可能没有明确的、可检查的交付内容。也就是说，不能明确地确定子任务已经完成。它不能让开发人员在完成时感觉到已经完成。一定要有某种办法明确地判定出子任务已经完成。

例子4.3

项目小组把子任务“对获取的图像进行编码”分解成一组更为详细的子任务，每个新的子任务都有一个更明确的交付内容以及完成标准。这组交付内容及子任务如下所示：

安装摄像驱动器	安装好的驱动器
从窗口中测试驱动器并将一个图像保存到文件	映像文件
编写从C++调用驱动器的例程	例程
分别测试C++例程并保存一个图像到文件	来自C++代码的映像
从主机器人控制软件中测试C++例程并获取图像	来自主程序的映像

4.3 PERT：程序评估和评审技术

这种技术建立显示出任务之间依赖关系的图。每个任务都有一个完成任务所需的时间估计，以及可以开始这个任务前所必须完成的其他任务的列表（依赖关系）。这个图可能不只有

一个开始子任务或一个终止子任务。只有在所有子任务都完成时整个任务才算完成。可用此图来计算所有子任务的完成时间，整个任务的最小完成时间以及子任务的关键路径。

4.3.1 完成时间的算法

1. 对于每个结点，执行步骤1.1（直到计算出所有结点的完成时间为止）。
   1.1 如果前面的任务已经完成，则取前面结点的最迟完成时间加上完成本结点所需时间。
2. 具有最迟完成时间的结点决定了项目的最早完成时间。

例子4.4

对表4-1应用这个算法，表4-1给出了任务及依赖关系。图4-3给出了同样的依赖关系。为了使用完成时间算法，应该从子任务a开始。它没有任何依赖关系，所以它在初始时刻开始（即时间0）它在时间0 + 8 = 8完成。类似地，子任务b可以在时间0 + 10 = 10完成。请参看表4-2。请注意，因为这两个子任务不是互相依赖的，它们也不依赖别的子任务，所以它们可以从时间0开始。计算这两个子任务的完成时间不需要别的资源。也就是说，对于这个完成时间的计算，假定有同时完成这两个子任务的人力。

表4-1 子任务

子任务标识	完成任务时间	依赖关系
a	8	
b	10	
c	8	a、b
d	9	a
e	5	b
f	3	c、d
g	2	d
h	4	f、g
i	3	e、f

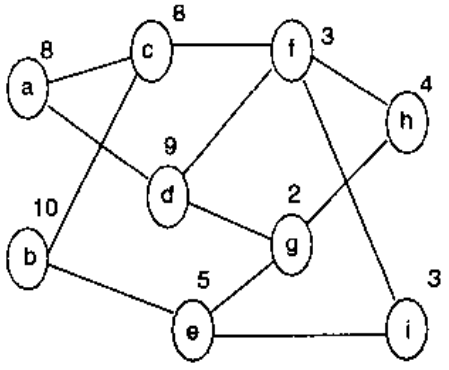


图4-3 PERT图

表4-2 各子任务的开始时间和完成时间

子任务标识	开始时间	完成任务时间	关键路径
a	0	8	
b	0	10	*
c	10	18	*
d	8	17	
e	10	15	
f	18	21	*
g	17	19	
h	21	25	*
i	21	24	

因为子任务a和b的完成时间已经计算出来，可以计算结点c、d、e的完成时间。因为c的前面结点的结束时间为8和10，所以c结点可从10开始，在10 + 8 = 16完成。d的开始时间为8，完成时间为8 + 9 = 17，而e的开始时间为10，完成时间为10 + 5 = 15。

现在可以处理子任务f和g了。它们的开始时间分别为17和16。f的完成时间为17 + 3 = 20，g的完成时间为16 + 2 = 18。子任务h和i的开始时间都是21，h的完成时间为25，i的完成时间为24。表4-2给出了所有子任务的开始时间和结束时间。

4.3.2 关键路径

关键路径是决定可能的最短完成时间的任务集合。如果完成所有并行活动的资源不足，完成时间会更长。但是，完成时间不可能因为增加更多的资源而变短。

4.3.3 构造关键路径的算法

1) 从具有最迟完成时间的结点开始，将它（们）作为关键路径的组成部分。

2) 选择具有最迟完成时间的关键结点作为前趋结点，使它（们）作为关键路径组成部分。重复执行步骤2直到到达开始结点为止。

例子4.5

从表4-2中可看到所有子任务的完成时间。子任务h具有最迟的完成时间25。因此，可以将h作为关键路径的一部分。h的前趋结点为f和g。在这两个子任务中，f具有最迟完成时间，因此将f标为关键路径的一部分。

子任务f以c和d作为前趋结点。因为c的完成时间更迟，所以将c作为关键路径的一部分。子任务c以a和b作为前趋结点，而b的完成时间更靠后，将它作为关键路径的一部分。现在已

到达一个初始子任务，关键路径构造完毕。

4.3.4 宽松时间

位于关键路径上的子任务必须尽早开始，否则整个项目就会延迟。而不在关键路径上的子任务何时开始就具有一定的灵活性了。这种灵活性称为宽松时间。

4.3.5 宽松时间的算法

1) 选取尚未处理过的具有最迟结束时间的非关键路径结点。如果此子任务没有后继结点，则选取所有结点中最迟结束时间。如果此子任务有后继结点，选取后继结点的最迟开始时间中最早的时间。这就是此子任务的最迟完成时间。使此子任务的最迟开始时间反映这个最迟完成时间。

2) 重复步骤1直到处理完所有非关键路径子任务为止。

例子4.6

具有最迟完成时间的非关键子任务为子任务i。因为它没有后继结点，所以采用最迟完成时间25。将其作为i的最迟完成时间。因为25比24迟1，所以将i的开始时间从21改为21,22。现在，最迟未处理的非关键子任务为g。因为h是g的唯一后继结点，而h必须从21开始，所以g必须在21结束。这样g的完成时间成了19,21而开始时间为17,19。下一个要处理的子任务为d。它有后继结点f和g。子任务f必须从18开始，所以d的完成时间为17,18，其开始时间为8,9。要处理的下一个子任务为e。子任务e有后继结点g和i。子任务g的最迟开始时间为19而i的最迟开始时间为22，这样子任务e的开始时间应该是10,14，完成时间为15,19。要处理的最后一个子任务为a。它有后继结点c和d。子任务a必须在9完成，所以a的完成时间为8,9，而开始时间为0,1。表4-3汇总了各个计算结果。

表4-3 带宽松时间的子任务

子任务标识	开始时间	完成时间	关键路径
a	0,1	8,9	
b	0	10	*
c	10	18	*
d	8,9	17,18	
e	10,14	15,19	
f	18	21	*
g	17,19	19,21	
h	21	25	*
i	21,22	24,25	

例子4.7 使用Microsoft Project

打开MS Project（版本98）。在左边菜单上选择PERT视图。在Insert下拉菜单下插入一个新任务。用鼠标右击打开任务信息。将任务的时间改为5天。拖动这个任务建立一个依赖于第一个任务的新任务，或者转到Insert菜单插入另一个新任务。建立例子4.5中的那些任务和依赖关系。关键路径上的任务将显示为红色。用左菜单条查看这个项目的Gannt图视图。

4.4 软件成本估算

软件成本估算的任务是确定完成项目需要多少资源。通常这个估算是以程序员－月（PM）为单位的。

成本估算有两种极不相同的方法。较早的方法称为LOC估算，它基于项目开发所需的代码行数的初步估算。较新的方法基于项目描述中的计算功能点。

4.4.1 代码行估算

基于代码行（LOC）的估算的第一个步骤是估计完成项目的代码行数。这可以根据经验、以前的项目大小、竞争者的解决方案大小进行估计，或者可以将项目分解成更小的项目块，然后估算每个小项目块的大小。对于每个项目块i，标准的方法是估计最大可能尺寸max_i，最小可能尺寸min_i和“最佳猜测”尺寸$best_i$。整个项目的估算为最大、最小和4倍最佳猜测尺寸之和的1/6：

$$S的标准偏差= (sd^2 + sd^2 + \ldots + sd^2)^{1/2}$$

$$E(i)的标准偏差 = (max-min)/6$$

例子4.8

项目组WRT为他们的项目标出了7个子项目块。表4-4给出了他们对每个子项目块的尺寸估算。

表4-4 子项目块尺寸估算（以LOC为单位）

子项目块	最小尺寸	最好猜测	最大尺寸
1	20	30	50
2	10	15	25
3	25	30	45
4	30	35	40
5	15	20	25
6	10	12	14
7	20	22	25

每个子项目块的估算如下：

p1(20 + 4 × 30 + 50)/6 = 190/6 = 31.6

p2(10 + 4 × 15 + 25)/6 = 95/6 = 15.8

p3(25 + 4 × 30 + 45)/6 = 190/6 = 31.6

p4(30 + 4 × 35 + 40)/6 = 220/6 = 36.7

p5(15 + 4 × 20 + 25)/6 = 120/6 = 20

p6(10 + 4 × 12 + 14)/6 = 72/6 = 12

p7(20 + 4 × 22 +25)/6 = 133/6 = 22.17

整个项目的估算为每个部分的估算之和:

整个项目估算 = 31.6 + 15.8 + 31.6 + 36.7 + 20 + 12 + 22.17 = 170.07 LOC

此估算的标准偏差估算如下:

$$\begin{aligned}\text{标准偏差} &= ((50-20)^2 + (25-10)^2 + (45-25)^2 + (40-30)^2 + (25-15)^2 + \\ &\quad (14-10)^2 + (25-20)^2)^{1/2} \\ &= (900 + 225 + 400 + 100 + 100 + 16 + 25)^{1/2} \\ &= 1766^{1/2} = 42.03\end{aligned}$$

4.4.2 基于LOC的成本估算

基本的LOC方法是与历史数据相匹配的一个公式。这个基本公式有三个参数：

$$\text{成本} = \alpha \times \text{KLOC}^{\beta} + \gamma$$

α为每KLOC（千行代码）的边际成本。这是每增加千行代码所需增加的成本。参数β是一个反映关系的非线性指数。β的值大于1表示每KLOC的成本随项目尺寸的增加而增加。这是一种不经济的模式。β值小于1反映了一种经济的模式。有的研究发现β大于1，而有的研究发现β小于1。参数γ反映项目的固定成本。研究已经发现γ可以为正或为零。

例子4.9

LMN公司记录以前的项目数据如下。请估算成本公式的参数并估算一个30 KLOC的项目的工作量有多大（见表4-5）。

表4-5 历史数据

项目ID	尺寸（KLOC）	工作量（PM）
1	50	120
2	80	192
3	40	96
4	10	24
5	20	48

分析或绘出这些数据将会得出尺寸和工作量间的一个线性关系。线的斜率为2.5。这是基于LOC的成本估算公式中的α。因为这条线是直线（线性关系），所以β为1，γ值为零。

4.4.3 成本构成模型

成本构成模型（COCOMO）为经典的LOC成本估算公式。它是由Barry Bochm于1970年建立的。他使用千个交付源指令（KDSI）作为尺寸的单位。KDSI与KLOC是等同的。它的工作量单位为程序员－月（PM）。

Boehm将历史项目的数据划分成三种项目类型：

1) 应用程序（独立的，有组织的，如数据处理，科学计算）。
2) 实用程序（半独立的，如编译程序、链接程序、语法分析程序）。
3) 系统程序（嵌入式的）。

为确定工作量，他定出了成本模型的各个参数值：

1) 应用程序：$PM = 2.4 \times (KDSI)^{1.05}$
2) 实用程序：$PM = 3.0 \times (KDSI)^{1.12}$
3) 系统程序：$PM = 3.6 \times (KDSI)^{1.20}$

例子4.10

计算从5到50 KDSI（见表4-6）的项目的程序员工作量。

表4-6 COCOMO工作量

尺寸	应用程序	实用程序	系统程序
5K	13.0	18.2	24.8
10K	26.9	39.5	57.1
15K	41.2	62.2	92.8
20K	55.8	86.0	131.1
25K	70.5	110.4	171.3
30K	85.3	135.3	213.2
35K	100.3	160.8	256.6
40K	115.4	186.8	301.1
45K	130.6	213.2	346.9
50K	145.9	239.9	393.6

Boehm还算出，在他的项目数据中，基于不同的项目类型和尺寸，存在一个标准的开发

时间。下面列出了开发时间（TDEV）的公式，以程序员－月为单位：

1) 应用程序：$TDEV=2.5*(PM)^{0.38}$

2) 实用程序：$TDEV=2.5*(PM)^{0.35}$

3) 系统程序：$TDEV=2.5*(PM)^{0.32}$

例子4.11

利用COCOMO公式，为5~50KDSI的项目，计算标准的TDEV（见表4-7）。

表4-7 COCOMO开发时间

尺寸	应用程序	实用程序	系统程序
5K	6.63	6.90	6.99
10K	8.74	9.06	9.12
15K	10.27	10.62	10.66
20K	11.52	11.88	11.90
25K	12.60	12.97	12.96
30K	13.55	13.93	13.91
35K	14.40	14.80	14.75
40K	15.19	15.59	15.53
45K	15.92	16.33	16.25
50K	16.61	17.02	16.92

4.4.4 功能点分析

功能点用来确定和量化项目所需功能的数量。其想法是统计出需要处理的外部行为中的事物的数量。要计数的经典项如下所示：

- 输入。
- 输出。
- 查询。
- 内部文件。
- 外部接口。

查询 不改变内部数据的请求－响应对。例如，对特定员工的地址的请求就是一个查询。提问、提供名字和获得地址的整个序列将作为一个查询。

输入 提供给程序的应用数据项。逻辑输入一般视为一个项，单个字段一般不独立计数。例如，一个员工的个人数据的输入视为一个输入。

输出 应用程序数据的显示。输出可以是一个报表，一个屏幕显示或一条错误消息。再次说明，单个字段一般不能认为是独立的输出。如果报表有多行，例如部门中每个员工一行，这些行将合起来作为一个输出。不过，也有人将行的汇总作为独立的输出。

内部文件 必须由系统维护的顾客理解的逻辑文件。如果某个实际的文件包含1 000项个人数据，它可能会被视为一个文件。然而，如果文件包含个人数据、部门汇总数据以及其他部门数据，则为计算功能点，这个文件也可以被视为三个独立的文件。

外部接口 与其他程序共享的数据。例如，个人数据文件可能会被人力资源部门使用以便进行提拔或加薪。因此，这个文件可视为两个系统之间的一个接口。

未调整功能点计数

标出单个的功能点项，把它们分成简单、平均或复杂三类。然后将表4-8的权重赋予每一项，求出总数。这个总数就称为未调整功能点。

功能点计数没有标准。各种书籍中有不同的计数规则。重要的是记住功能点试图度量开发软件所需的工作量。因此，与重要的工作量有关的任务会产生比占用少许工作量的任务更多的功能点。例如，报表底部的汇总行不同，计算功能点的方法也有所不同。有的软件工程师认为，汇总行应计为另一个输出，而另一些软件工程师却只统计报表中的主要条目。具体应该怎样解决应基于汇总行需要多少额外的工作量而定。

专门的规则没有组织机构内的一致性那样重要。一起工作并评价其他功能点分解将有助于建立这种一致性。此外，在项目完成后对估算进行评审有助于确定哪些项比功能点分析所标出的工作量更大，或者说哪些项就功能点来说是重复计数的，它们不需要占用所标出的那么多工作量。

请注意，应该尽量使功能点与处理每项所需的工作量一致。

表4-8 功能点权重

	简 单	平 均	复 杂
输出	4	5	7
查询	3	4	6
输入	3	4	6
文件	7	10	15
接口	5	7	10

例子4.12

学校的某个系需要一个程序用来为每个专业分配时间和教室，并建立课程安排计划。该

系有一个专业列表，该表中指定了教授名字和期望规模。这个系还有一个教室列表，列出每个教室最多能容纳的学生数目，以及不能同时上课的班级集合。此外，一个教授不能同时讲授两门课程。

这个程序比复杂的输入和输出难得多。它有两个主要输入：具有专业列表，指定的教授和期望规模的文件，以及具有带最大规模的教室的列表的文件。这两个文件，虽然读起来很简单，但很难处理，因此将它们的级别评定为复杂。还有不能同时上课的班级集合的另外一个文件。这个文件在结构上也很简单，但难于处理。最后一行只有既不是输入也不是输出的限制。

这个程序有一个输出，即安排计划。这是一个复杂的输出。不存在前面提到的查询或接口，也没有提到要维护的文件。

4.4.5 生产率

生产率是软件开发人员进行度量的一个重要的量。这个量等于最终产品的总尺寸除以所有程序员的总工作量。这个量的单位为LOC/程序员－天。也可以用每个程序员－天的功能点为单位来度量生产率。

请注意，生产率包括了软件生命周期的所有阶段中所有的工作量。

例子4.13

XYZ公司在最终项目的每个生命周期中耗用了如下的工作量（参阅表4-9）。请分别以LOC/程序员－天为单位和功能点/程序员－天为单位计算工作量。功能点估计为50个未调整功能点。最终的项目包括950个代码行。

表4-9 各阶段中的工作量

阶 段	程序员－天
需求	20
设计	10
实现	10
测试	15
文档	10

总的工作量为65程序员－天。生产率为950/65 = 14.6 LOC/程序员－天。利用未调整功能点（fp）方法，生产率为50fp/65天 = 0.77fp/程序员－天。

4.4.6 判定估算

为了判定估算值，需要计算一个量。Tom DeMarco提出了估算质量因子（EQF）。

DeMarco将EQF定义为实际曲线下的面积除以估计值和实际值之间的面积。这就是错误百分比或平均相关错误的倒数。因此，EQF越大，估算就越好。DeMarco认为，该值大于8是比较合理的。

例子4.14

下面是某个项目的估算，该项目在11.5个月后完成，成本为350万美元。

开始	1.5个月	5.5个月	8个月
230万	310万	390万	340万

总面积为11.5月 × 350万美元 = 4025万美元。实际曲线和估算之间的差为|230−350| × 1.5 + |310−350| × 4 + |390−350| × 2.5 + |340−350 × 3.5 = 475万美元。比例为40.25/4.75 = 8.7。

4.4.7 自动估算工具

Internet上有许多可用来计算COCOMO或COCOMO2的工具。大多数工具的界面都很简单。使用浏览器搜索COCOMO，会找到多个站点（两个有用的站点为sunset.usc.edu/research/COCOMOII和www.jsc.nasa.gov/bu2/COCOMO.html）。

习题

1. WBS和过程模型之间的区别是什么？

2. 为什么WBS应该为树型结构？

3. 在WBS中没有任务的完成标准将会出现什么情况？

4. 使用PERT图的优点是什么？

5. 为什么延迟关键路径上的任务会延迟整个项目？

6. 如果做项目的只有一个人，关键路径是否还重要？

7. 宽松时间的重要性是什么？

8. 为什么宽松时间要基于后继任务的最迟开始时间中的最早开始时间？

9. 绘出一个表示因规模扩大而节约成本和因规模扩大而增加成本的图。对此图进行标注，

说明其中各曲线的含义。

10. 常见的是使用默认的估算版本。考虑某人上次要求你给出某样事情的估算。你给出的是默认的估算定义还是给出DeMarco提出的估算定义？

11. 为什么成本估算的参数应该通过公司的数据来确定？

补充问题

1. 建立油漆房间任务的WBS。假定相应的过程模型是用于家庭工作项目的，该项目具有以下活动：计划工作、采购、油漆、清洁。

2. 建立下列牙科诊所软件开发的WBS：

Tom在小镇上开了一个牙科诊所。他有一个牙科助手、一个牙科保健员和一个接待员。Tom需要一个系统来管理预约。

当病人打电话预约时，接待员将查阅日历并安排病人尽早得到诊治。如果病人同意计划的约定时间，该接待员将输入约定时间和病人的名字。系统将核实病人的名字并提供病人的记录数据，数据包括病人的ID号等。在每次检查或清洗后，保健员或助手将标记相应的预约已经完成，增加说明，如果必要的话会安排病人下一次再来。

系统要能够按病人名和按日期进行查询，能够显示病人记录数据与预约信息。接待员可以取消预约，可以打印出前两天预约尚未接诊的通知清单。系统中含有来自病人记录的电话号码。接待员还可以打印出关于所有病人的每天和每周的工作安排。

3. 针对下列的B&B问题，建立软件开发的WBS：

Tom和Sue在新英格兰的一个小镇上开了一个住宿加(次日)早餐的旅店。他们有三间客房，需要一个系统来管理房间预订并监控开支和利润。在某个顾客打电话预订住宿时，他们要查看日历，如果有空位，他们要输入顾客的名字、地址、电话号码、日期、议定的价格、信用卡号以及房间号。预订必须交一天的保证金。

预订在没有保证金的情况下将保存议定的时间。如果到该日期还没有交保证金，将取消预定。

4. 建立下列汽车代理的软件开发的WBS：

某个汽车代理商希望使用系统自动管理其库存情况。系统要能自动记录顾客购买的所有汽车。记录所有修理、所有修理部件发货情况。该代理商希望对总的日常维修、日常销售和总库存给出每日的详细报表。这个报表称为“日报”。该代理商还要记录所有顾客以及拜访本代理店的潜在顾客。这个代理商还想要一个月度报表，按月的每一天列出顾客的来访和购买情况。此代理商还希望能够对顾客或潜在的顾客进行查询。

5. 绘出补充问题1的任务的PERT图。

6. 对于给定的任务集合和依赖关系绘出PERT图。完成给出关键路径和宽松时间的表。

结点	依赖关系	时 间	开始	结束
a		10		
b	a	5		
c	a	2		
d	a	3		
e	b,c	7		
f	b,d	9		
g	c,d	5		
h	e,f,g	6		

7. 绘出给定任务集合及依赖关系的PERT图。完成给出关键路径和宽松时间的表。

结点	依赖关系	时 间	开始	结束
a		10		
b	e	10		
c	d,f	10		
d	a,f,b	20		
e	a,f	8		
f	a	5		

8. 利用给定的数据集估算成本参数：

项 目	尺寸（KLOC）	成本（程序员－月）
a	30	84
b	5	14
c	20	56
d	50	140
e	100	280
f	10	28

9. 根据给定的数据集估算成本参数：

项 目	尺寸（KLOC）	成本（程序员－月）
a	30	95
b	5	80
c	20	65
d	50	155
e	100	305
f	10	35

10. 计算大约有39 800个代码行的有组织的项目的COCOMO工作量、TDEV、平均人员配备及生产率。

11. 计算补充问题2的未调整功能点。

习题答案

1. WBS和过程模型之间的区别是什么？

对于许多项目来说，过程模型描述一般意义下的软件活动。它描述了过程、所使用和生成的制品以及负责相应活动的参与者。过程模型为一个图，它可以有环路。工作分解结构是用针对特定项目的详细、必须的子任务扩展过程模型活动的一种树型结构。出现在每个项目中的WBS任务也应该出现在过程模型中。

2. 为什么WBS应该为树型结构？

如果在WBS中有循环，就表示某个任务递归地依赖于自身，这是不可能的。一般只有在某个任务定义得不恰当时才会出现循环。如果有两条路径到达一个任务，通常表示有两个不同的高级任务依赖于该共同的子任务。它只需要给出（和完成）一次。

3. 在WBS中没有任务的完成标准将会出现什么情况？

这表示不能把握项目进展情况或者不能知道项目何时完成。这还可能表示不知道实际应该做什么。例如，“研究XXX”这样一个子任务就不是很好的任务规定。应该有一个研究的目标并且在应该任务描述中清楚地规定出来。

4. 使用PERT图的优点是什么？

虽然WBS将建立一个任务清单，但利用WBS不太容易看出哪些任务必须先做，哪些任务决定最终的完成时间。决定最终完成时间的任务都在关键路径上。

5. 为什么推迟关键路径上的任务会延迟整个项目？

关键路径定义为一组任务，这组任务决定了完成项目所需的最短时间。如果一个任务位于关键路径上并被延迟，关键路径上的下一个任务的开始时间和结束时间也要延迟。这样依次传递，会波及关键路径上的最后一个任务，从而延迟整个项目。

6. 如果做项目的只有一个人，关键路径是否还重要？

只有当所有任务都位于关键路径上时，关键路径才重要。如果有的任务不在关键路径上，而且因为只有一个人，这些任务不能并行进行，所以这些任务所需的时间将加到关键路径上，构成整个项目的完成时间。

7. 宽松时间的重要性是什么？

虽然不在关键路径上的任务不决定项目最短完成时间，但如果这些任务不能及时完成，则项目的完成时间也要被推迟。宽松时间给出了完成任务的时间范围。

8. 为什么宽松时间要基于后继任务的最迟开始时间中的最早开始时间？

宽松时间是根据这样的最早开始时间确定的，因为如果相应的任务推迟超过该开始时间，则此任务将不能按时完成，产生的波及后果将会推迟整个项目。

9. 绘出一个表示因规模扩大而节约成本和因规模扩大而增加成本的图。对此图进行标注，说明其中各曲线的含义。

向上弯曲的线表示因规模扩大而增加成本。就是说，对于较大的项目，每单位尺寸的成本将会增加。向下弯曲的线表示因规模扩大而节约成本。就是说，项目越大，每单位尺寸的成本就越小。如图4-4所示。

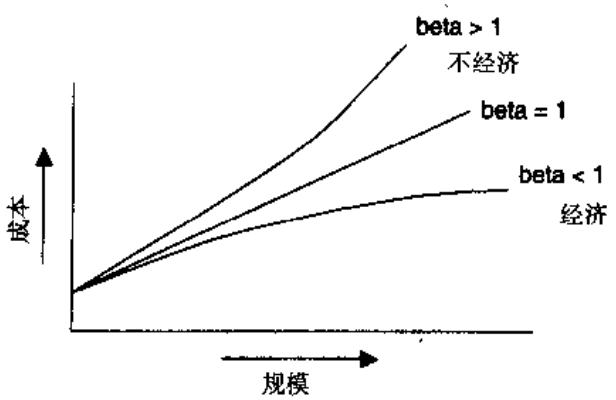


图4-4 习题9的示意图

10. 常见的是使用默认的估算版本。考虑某人上次要求你给出某样事情的估计。你给出的是默认估计定义还是给出DeMarco提出的估算定义？

我的学生问何时对某个作业进行审阅，我常常会说如果我没有别的事且不受打扰时就进行审阅。这个回答是不现实的，因为总是有许多更为紧迫的工作并且常常会有别的事情打扰。

11. 为什么成本估算的参数应该通过公司的数据来确定？

每个公司都有不同的习惯、标准、策略以及开发软件的类型。期望大型的国防承包商找到的成本估算参数与很小的按小时开发的项目的成本估算参数相同是不现实的。

补充问题答案

1. 建立油漆房间的WBS。
（过程模型活动未给出。）

a. 选择房间的颜色	确定的颜色
b. 购买油漆	成罐的油漆
c. 购买刷子	刷子
d. 清理墙面	清理干净的墙面
e. 打开油漆罐	打开的油漆罐
f. 搅拌油漆	搅拌好的油漆
g. 油漆墙面	油漆过的墙面
h. 清理现场	清理并刷过漆的墙壁

2. 建立牙科诊所问题的WBS：
（生命周期阶段和PM活动省略。）

进行风险评估	评估说明
估算工作量	成本估算
规划进度	进度表
与牙科医生一起评审风险、估算以及进度表	牙科医生的认可
设计对象模型	对象模型
与牙科医生一起评审对象模型	牙科医生的认可
撰写病人记录系统的规格说明	规格说明
建立病人记录的原型界面	工作原型
与接待员和牙科医生一道评审催促表单	牙科医生的认可

与医科诊所人员一道评审日报和周报表单	牙科医生的认可
与项目组一起评审类模型设计	评审文档
实现	编译过的代码
成功地用C0覆盖进行单元测试	测试报告
集成系统	编译过的代码
系统测试	测试报告
在牙科诊所进行培训和阿尔法测试	测试报告
与牙科诊所员工一起评审系统	牙科医生的认可
员工的接收测试	牙科医生的认可
提交用户手册和文档	提交的手册

3. 建立B&B问题的WBS。

（PM活动和交付内容省略。）

可行性分析

与Tom和Sue讨论基于Web的选项。

确定是基于Web的系统还是单机系统。

需求

从Tom和Sue处得出需求。

建立需求文档。

与Tom和Sue一起评审需求。

设计

设计原型预订系统。

设计开支和利润部分。

实现

建立原型预订系统。

实现开支和利润部分。

测试

测试原型。

测试开支和利润部分。

交付

与Tom和Sue一起评审原型。

与Tom和Sue一起评审整个系统。

4. 建立汽车代理商问题的WBS。

（生命周期阶段和PM活动省略。）

需求

从代理商处提取需求。

建立需求文档。

与代理商一起评审需求文档。

编写初步的用户手册。

评审初步的用户手册。

建立测试案例。

设计

设计汽车代理系统原型。

设计报表格式。

实现

建立汽车代理系统原型。

实现查询和报表部分。

与代理商一起评审原型。

实现最终的版本。

测试

在代理店测试最终版本。

交付

培训员工。

提交文档。

5. 绘出补充问题1的任务的PERT图：

此PERT图如图4-5所示。

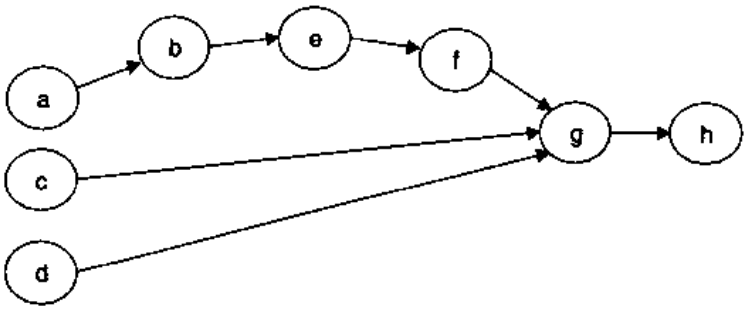


图4-5 油漆房间的PERT图

6. 对于给定的任务集合和依赖关系绘出PERT图。完成给出关键路径和宽松时间的表。（关键路径在表中用星号标出。）见图4-6。

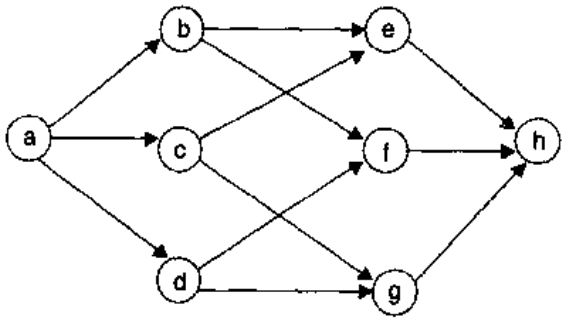


图4-6 PERT图

结　点	依赖关系	时　间	开　始	结　束
a		10	0	10*
b	a	5	10	15*
c	a	2	10,24	12,26
d	a	3	10,21	13,24
e	b,c	7	12,26	19,33
f	b,d	9	24	33*
g	c,d	5	13,28	18,33
h	e,f,g	6	33	39*

7. 绘出给定任务集合及依赖关系的PERT图。

结　点	依赖关系	时　间	开　始	结　束
a		10	0	10
b	e	10	23	33
c	d,f	10	53	63
d	a,f,b	20	33	53
e	a,f	8	15	23
f	a	5	10	15

每项任务都在关键路径上，无宽松时间，见图4-7。

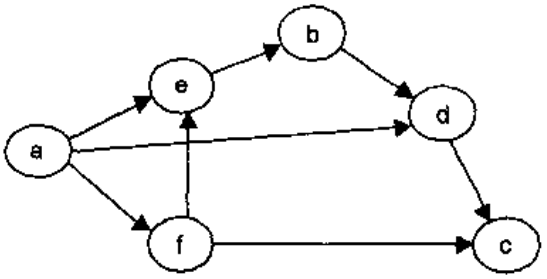


图4-7 PERT图

8. 利用给定的数据集估算成本参数：

$$成本 = 3.8 \times 尺寸(KLOC)$$

9. 根据给定的数据集估算成本参数：

$$成本 = 4.0 \times 尺寸（KLOC）+5.0$$

10. 计算大约有39 800个代码行的有组织的项目的COCOMO工作量、TDEV、平均人员配备及生产率。

有组织的项目采用相应的应用公式。成本 = $2.4 \times (KDSI)^{1.05}$

成本 = $2.4 \times 39.8^{1.05} = 2.4 \times 47.85 = 114.8$程序员 - 月

TDEV = $2.5 \times (PM)^{0.38} = 2.5 \times 6.06 = 15.15$月

平均人员数 = 成本/TDEV = 114.8/15.15 = 7.6个程序员

生产率 = 39 800 LOC/(114.8PM × 20天/月) = 17.3 LOC/程序员 - 天

11. 计算补充问题2的未调整功能点。

类　型	简　单	平　均	复　杂	总　计
输入	病人名	取消预约		13
	完成的预约			
	预约的目的			
输出	说明	日历	日常安排	38
		支持细节	周安排	
		预约信息		
		通知列表		
查询	按名字查询	核实病人		18
	按日期查询	查看日历		
		接受的预约		
文件		病人数据		10
接口				
合计				79

第5章 软件度量

5.1 概述

科学是基于度量的。加快一个过程需要理解数值关系。这就需要度量。

软件度量是对象的符号映射。其目的是量化对象的某些属性。例如，度量软件项目的规模。此外，还有一个目的是预测某些没有正确度量的属性，如研制一个软件项目所需的工作量等。

并非对象的所有符号映射都是有用的。最重要的是度量的有效性。但有效性与度量的使用有关。人的身高就是一个例子。一个人的身高对于预测他是否能通过某扇门而不会碰到他的头很有用。要确认一个度量，仅有度量和属性之间的一种紧密的相关是不够的。例如，鞋的尺码与人的身高有关，但将鞋的尺码作为一个人身高的度量一般是不合适的。

下面列出度量有效的标准（R.Harrison. S.Counsell,R.Nithi. "An Evaluation of the MOOD Set of Object-oriented Software Metrics." IEEE TOSE 24:6, June 1998, 491-496）：

1) 度量必须能够区分不同的实体。
2) 度量必须符合某种表示条件。
3) 属性的每个单位必须给度量贡献一个相等的量。
4) 不同的实体可具有相同的属性值。

很多时候，并不直接度量感兴趣的属性。在这种情况下，可以使用间接度量。间接度量包括一个度量和一个预测公式。例如，密度不是一种直接度量。它是从质量和场强计算出来的，这两者都是直接度量。计算机科学中许多重要的概念（可维护性、可读性、可测试性、质量、复杂性等）都不能直接度量，许多度量程序就是用来间接度量这些属性的。

下面是间接度量有效的标准：

1) 相应的模型必须明确定义。
2) 相应的模型必须在尺度上一致。
3) 不应该有意想不到的不连续情况。
4) 单位和尺度的类型必须正确。

5.2 软件度量理论

度量的代表性理论已经研究100多年了。它涉及一个经验性的关系系统、一个数值关系系统以及这两个系统之间的一个关系 – 保留映射。

经验关系系统（E, R）由两部分组成：

- 实体集合E。
- 关系集合R。

关系一般为“小于或等于”。请注意，并非所有事物都必须有关系，即集合R可能是偏序的（偏序严格定义为一个满足三条公理的序：每个元素自身是相关的，关系不能拥有两个元素之间的两条路径，关系是传递的。关系不必是全序的，即并非每两个元素都是相关的）。

数值关系系统（N，P）也由两部分组成：

- 实体集合N，也称为“答案集合”，这个集合一般为数值（自然数、整数或实数）。
- 关系集合P。这个集合一般已经存在，常常为“小于”或“小于或等于”。

关系 – 保留映射M将（E，R）映射到（N，P）。这个映射的重要限制称为表示条件。有两个可能的表示条件。限制性最强的条件是，如果两个实体在任一系统中相关，则另一系统中的映像（或每个映像）是相关的：

$$x \text{ rel } y \text{ iff } M(x) \text{ rel } M(y)^3$$

（注：当且仅当x的映像与y的映像相关时，x才与y相关。）

限制性较弱的版本认为，如果两个实体在经验系统中相关，则在数值系统中，这两个实体的映像也以相同的方式相关：

$$M(x) \text{ rel } M(y) \text{ if } x \text{ rel } y$$

经典的度量理论的作者曾经使用过这两个版本。第二个版本的优点是经验系统中的偏序可以映射到全序的整数或实数上。

例子5.1 人的身高

人的身高是将经验系统映射到数值系统的一个经典例子。在经验系统中，关于人的身高存在一个很好理解的关系。让两个人站在一起，任何人都能说出这两个人中谁较高。这就是经验系统：人员为实体，众所周知的关系为“较矮或身高相同”。

相应的数值系统为实数系统（米制单位或英制单位），具有小于或等于的标准关系。

映射仅仅是人员的标准度量身高。身高一般是人靠墙光脚站着量出的高度。

满足表示条件（任一版本），因为如果Fred比Bill矮或和Bill一样高，则Fred的度量高度就会小于或等于Bill的度量高度。

例子5.2

建立一个结合了人的身高和体重的度量BIG。

根据经验，如果两个人身高相同，则较重的那个身材更大。如果两人体重相同，则较高的那个身材更大。如果我们采纳这种观点，那么可以得到一个大多数人都同意的偏序。我们不用这个偏序来定序的唯一一对人员是其中一人较重，一人较高的情形。

根据数值，我们可以使用一个<身高，体重>元组。每个元组的成分都是一个实数。如果两个元组中的成分在相同的方向相关，则两个元组是相关的。也就是说，如果x, y为两个元组，且if $x_{身高} =< y_{身高}$ and $x_{体重} =< y_{体重}$，则x在“身材”方面小于或等于y。这也是一个偏序，表示条件的两种版本都得到满足。

5.2.1 单调性

度量的一个重要特性是单调性。这表示一个属性的度量值并不因为该属性在对象中增加而改变方向。例如，代码行数不会因为增加更多的代码而减少。

例子5.3

线性函数是单调的，因为它的方向总是不变。二次函数一般不是单调的。例如，$y=5x-x^2$在$x = 0$到$x = 10$的范围内不是单调的。从$x = 0$到$x = 5$，y是增加的。从$x = 5$到$x = 10$，y是减少的。

5.2.2 度量尺度

有5种不同的度量尺度类型：额定尺度、顺序尺度、间隔尺度、比例尺度和绝对尺度。

限制最小的度量采用额定尺度类型。这种类型指定数值或符号，基本上不考虑数量。额定尺度度量的一个典型例子为运动服上的号码。我们并不仅因为一套运动服上的号码大于或小于另一套运动服上的号码就认为一个运动员比另一个运动员好。不存在从一种额定尺度度量转换到另一种额定尺度度量的公式。

在顺序尺度度量中，存在一种根据分配给实体的号码决定实体次序的隐含规律。典型的

例子为学生在班中的名次。如果一个学生在班中的名次为第一，则他的成绩必定比在班中排第二、第三或任何排名大于1的学生好。但是，我们不能认为排名的数值差很重要。换句话说，我们不能认为排名第一和排名第二的学生的差与排名第一百与排名第一百零一的学生的差相同。对于相同的实体，任何将一种顺序尺度度量转换为另一种顺序尺度度量的公式都必须保持相应的排序。

在间隔尺度度量中，两种度量之间差的量是一个常量。温度是一个例子。有两个经常使用的温度度量实例，即华氏温度和摄氏温度。从摄氏度量转换到华氏度量的公式为$9/5*x + 32$。对于相同属性的任意两种间隔尺度度量，转换公式必定为$ax + b$的形式。

在比例尺度度量中，两种度量之间差的量是一个常量，而且存在一个任何尺度的度量都使用的便于理解的零。例如，钱、长度和高度都是采用比例尺度的度量。这些度量具有很好理解的零概念：钱为零，零高度以及零长度。从一组单位转换为另一组单位（如从厘米转换为英寸）的公式只需要使用一个乘法常数即可。

绝对尺度是一种计数尺度度量。其单位很明显且很好理解。数弹子就是绝对尺度度量的一个例子。

5.2.3 统计

并非所有统计方法都适合于所有尺度。下面指出何种常见统计方法适合于何种尺度：

- 额定尺度：仅适用于模数、中值和百分数。
- 顺序尺度：上述统计方法以及斯皮尔曼相关（Spearman correlation）。
- 间隔尺度：上述统计方法以及平均、标准偏差和皮尔逊相关（Pearson correlation）。
- 比例尺度：所有统计方法。
- 绝对尺度：所有统计方法。

例子5.4 平均数

温度是一种间隔尺度度量。因此，给出平均温度具有统计学意义。但是，棒球运动员服装上的号码为一种额定尺度度量。给出运动队服装上的平均数没有意义。类似地，一个班级中的学生排名的平均数或许多班级中学生排名的平均数也不合理。

5.3 产品度量

产品度量是可以通过文档（不依赖于它怎样生成）进行计算的度量。一般来说，产品度

量涉及源代码的结构。可对其他文档定义产品度量。例如，需求规格说明中图形的数目就是一种产品度量。

例子5.5 代码行

最基本的尺寸度量为代码行度量。代码行有许多不同的计数方法。其定义可能很简单，如文件中换行符的数目等。通常代码行的计数不包括注释。有时，也不包括空行或只有分隔符的行。有时，对语句而不是代码行进行计数。

5.3.1 McCabe的环数

McCabe的环数于1976年引入，是软件开发中继代码行之后最常使用的度量。据原刊物文章的标题，也称为“McCabe的复杂性度量”，它基于图论的环数。McCabe试图度量程序的复杂性。前提是该复杂性与程序的控制流相关。图论使用公式$C = e-n+1$来计算环数。McCabe稍加修改后的公式为：

$$C = e-n + 2p$$

其中：

e = 边的数目

n = 结点的数目

p = 强连通组件的数目（通常为1）

例子5.6

确定图5-1中给出的控制流程图的环数。

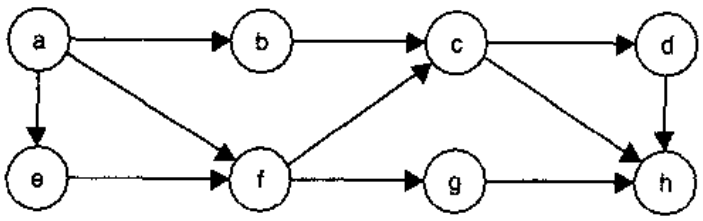


图5-1 控制流程图

图中有8个结点，所以$n = 8$。有11条弧，所以$e = 11$。环数为$C = 11-8 + 2 = 5$。

平面图是没有交叉线的图。瑞士数学家欧拉（1707~1783）证明，在平面图中$2 = n-e + r$，其中r = 区域的数目，e = 边的数目，n = 结点的数目。区域为包含（或定义）在弧中的范围。利用代数变换，可将这个公式转换为$r = e-n + 2$。因此，一个平面图上的区域的数目等于环数。

例子5.7

用罗马数字标记例子5.6的控制流程图中的区域。

如图5-2所示，有五个区域。I为图的外部区域。

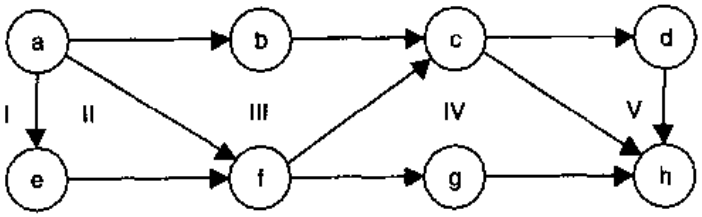


图5-2 带罗马数字的控制流程图

根据控制流程图计算环数很费时间，根据大程序构造控制流程图也是非常费时间的。McCabe找到了一种计算即度量的更直接的方法。他发现区域的数目通常等于程序中判断的数目加1，即$C = \pi + 1$，其中π为判断的数目。

在源代码中，IF语句、WHILE循环或FOR循环都可视为判断。CASE语句或其他多分支语句的判断数目计数为可能的分支数目减1。

控制流程图需要具有一个特定的开始结点和一个特定的终止结点，否则，判断的数目就不会是区域的数目减1。

例子5.8

用小写字母标出例子5.6的控制流程图中判断的数目。

如图5-3所示，从结点a起，有三条弧，所以必定有两个判断。将这两个判断分别标记为a和b。结点c和f都有两条弧，因此每个结点有一个判断。其他结点最多只有一个出口，所以没有判断。这样共有四个判断，即$C = 4 + 1 = 5$。

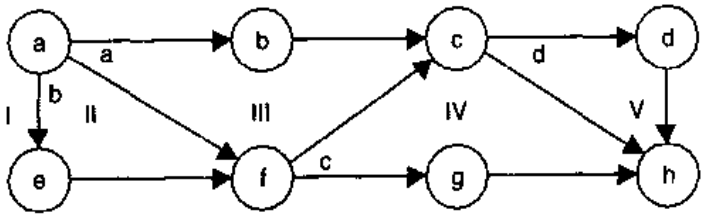


图5-3 带小写字母的控制流程图

例子5.9

计算图5-4所示的不正确的控制流程图的环数。

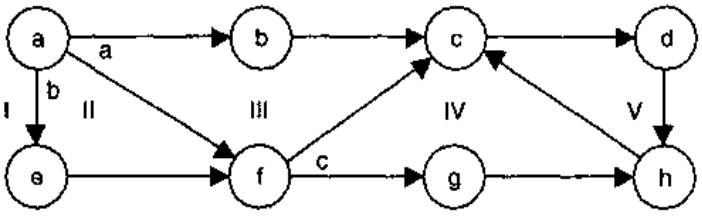


图5-4 不正确的控制流程图

除了弧c-h被替换为h-c弧外，这个控制流程图与上一个例子的控制流程图相同。这样不会改变结点、边或区域的数目。所以，前两个循环计数的方法不改变。不过，判断d消失了，所以第三个方法给不出相同的答案。但这不是一个合法的控制流程图，因为该图没有终止结点。

阈值

度量的一个重要方面是了解值何时合理何时不合理。McCabe分析了大量的项目，发现对于环数超过10的模块，会有许多错误记录，维护起来也困难得多。因此，可将一个模块中环数的阈值定为10。如果环数大于10，必须采取措施以减少这个值，或者将模块分成更小的模块。

5.3.2 Halstead的软件科学

Maurice Halstead是软件度量的第一个研究者。他在20世纪60年代末～70年代进行了许多研究工作。他的目标是确定什么因素导致软件的复杂性。他根据经验寻找内在规模的度量。当他找到了他认为是很好的度量及预测公式后，他还试图建立一种相关的理论。他的简单的度量方法到现在仍然被认为是有效的，而他的较复杂的度量以及预测公式则不是那么准确。

1. 基本实体：操作符和操作数

Halstead得出其良好结果的基本方法是把程序视为一个权标集合，他把这些权标分为操作符或操作数。操作数为具有一个值的权标。通常，变量和常量为操作数。其他的量则是操作符。因此，逗号、圆括号、算术运算符、方括号等全都为操作符。

所有总是成对或以三个一组等方式出现的权标将视为一个权标。例如，一个左圆括号和一个右圆括号是一个权标圆括号。一段具有if-then结构的代码是一个if-then权标。

Halstead关心算术式子而不是声明、i/o语句等。因此，他没有对声明、输入输出语句或注释进行计数。但是，目前多数组织机构都对程序的所有部分进行计数。

Halstead的操作符和操作数定义对许多解释都是开放的。对判定不明确的情形还没有公认的标准。只要一个组织机构保持前后一致，就不会产生问题。但是一个组织机构中的人员不可能将自己的结果与其他组织机构中的结果进行比较。

作者建议一种基于语法的方法，其中所有操作数为用户定义的权标，所有操作符为语言语法定义的权标。

2. 基本度量：η_1和η_2

在程序中，特定操作符的数目为η_1（发音为“eta one”），特定操作数的数目为η_2（发音为“eta two”）。

特定权标的总数为$\eta = \eta_1 + \eta_2$。这是程序尺寸的基本度量方式。

例子5.10

以下代码用重复的加法来完成乘法，确定其特定的操作符和操作数。

```
Z = 0;
while X > 0
        Z = Z + Y ;
        X = X-1 ;
end-while ;
print(Z) ;
```

操作符

=、;、while-endwhile、>、+、−、print、()

操作数

Z、0、X、Y、1

因此，$\eta_1 = 8$，$\eta_2 = 5$。

3. 可能的操作数η_2^*

Halstead希望考虑和比较算法的不同实现。他建立了可能操作数的概念，可能操作数表示给定算法的任意实现所需的值的最小集合。这一般是通过统计算法内不进行初始设置的所有值而得出的。它包括读入的值、传入的参数、算法内访问的全局值等。

4. 长度N

下一个基本度量是操作符的总数N_1和操作数的总数N_2。其和表示程序的权标长度：

$$N = N_1 + N_2$$

例子5.11

计算例子5.10中代码的Halstead长度。

操作符	
=	3
;	5
while-endwhile	1
>	1
+	1
-	1
print	1
()	1

操作数	
Z	4
0	2
X	3
Y	2
1	1

操作符出现了14次，因此N_1为14。同理，可知N_2为12，所以$N = N_1 + N_2 = 14 + 12 = 26$。

5. 长度估计（est N或N_hat）

长度的估计是Halstead预测公式的最基本内容。仅根据估计出的将用于程序的操作符和操作数的数目，便可利用这个公式按权标估计程序的实际尺寸：

$$\text{est N} = \eta_1 * \log_2 \eta_1 + \eta_2 * \log_2 \eta_2$$

例子5.12

估计例子5.10中代码的长度。

$\log_2 x$实际上是求2的几次幂等于x。因此，$\log_2 2$为1，$\log_2 4$为2，$\log_2 8$为3，$\log_2 16$为4：

$$\log_2 \eta_1 = \log_2 8 = 3$$

$$\log_2 \eta_2 = \log_2 5 = 2.32$$

$$\text{estN} = 8 \times 3 + 5 \times 2.32 = 24 + 11.6 = 35.6$$

而实际的N等于26。计算得出的结果可视为一个近似数。对这样的一个小程序来说，这个结果可能是一个不坏的近似值。

从实践中发现，如果N和est N不在大约百分之三十以内，则应用别的软件科学度量可能都不合理。

6. 容量V

当容量真正与对要度量的程序进行编码所占的二进制位数相关时，Halstead把容量视为一个3D度量（要对n个不同的项进行编码，则每个项至少需要$\log_2 n$个二进制位。要对N个项的序列进行编码，则需要$N*\log_2 n$个二进制位。）换句话说：

$$V = N \times \log_2(\eta_1 + \eta_2)$$

例子5.13

计算例子5.10中代码的V。

$$V = 26 \times \log_2 13 = 26 \times 3.7 = 96.2$$

容量给出了对许多不同的值进行编码所需的二进制位数。这个数值是很不容易解释的。

7. 可能的容量V*

可能的容量是用任意语言解决问题的方案的最小尺寸。Halstead假定在最小实现中，只有两种操作符：函数名与分组的操作符。操作数的最小数目为η_2^*。

$$V^* = (2 + \eta_2^*)\log_2(2 + \eta_2^*)$$

8. 实现的级别L

因为我们有实际容量和最小容量，所以自然要取一个比例。Halstead用可能容量除以实际容量。这个比例表示，按可能的容量来度量，当前实现与最小实现之间的接近程度。实现的级别没有单位。

$$L = V^*/V$$

迄今为止描述的这些基本的度量都是很合理的。关于操作数和操作符的许多想法已经用在了许多其他的度量工作中。下面给出的这些度量主要是出于完整性考虑，这并不一定是有用或合理。

9. 工作量E

Halstead希望估计实现算法需要多少时间（工作量）。为此，他采用了元素心理判别（emd）的概念。

$$E = V/L$$

其单位为元素心理判别（emd）。Halstead的工作量是非单调的，换句话说，如果给有的程序增加语句，计算出来的工作量会减少。

10. 时间T

接着，Halstead希望估计实现算法所需的时间。他应用了心理学家John Stroud 1950年的研究成果。Stroud曾经测量过一个被试者观察快速通过自己面前的物体的速度。S为从这些实验中得到的Stroud数（emd/sec）。Halstead用18 emd/sec作为S的值。

$$T = E/S$$

5.3.3 Henry-Kafura信息流

Sallie Henry和Dennis Kafura建立了一个度量源代码的模块之间的复杂性的量。这个复杂性量基于一个模块的信息的流入和流出。对于每个模块，统计流入模块和流出模块的所有信息流，分别标记为in_i和out_i。这些信息流包括参数传递、全局变量以及输入和输出。他们还用每个模块尺寸的一个度量作为乘法因子。也可以用LOC和复杂性度量作这个权重因子。

$$HK_i = weight_i * (out_i * in_i)^2$$

总的度量为每个模块的HK_i之和。

例子5.14

根据下面的信息计算HK信息流度量。假定每个模块的权重为1。

mod#	a	b	c	d	e	f	g	h
in_i	4	3	1	5	2	5	6	1
out_i	3	3	4	3	4	4	2	6

mod#	a	b	c	d	e	f	g	h
HK_i	144	81	16	225	64	400	144	36

整个程序的HK为1110。

5.4 过程度量

生产率

生产率为一种基本的过程度量。生产率等于总的源代码行除以项目所用的程序员－天数。生产率的单位一般为LOC/程序员－天。在20世纪60年代的许多项目中，生产率为2~20 LOC/程序员－天。在较小的单个项目中，生产率可以更高。

例子5.15

项目总计有100 KLOC。20个程序员工作一年才完成此项目。这一年的工作包括了需求、设计、实现、测试和交付阶段的所有工作。假如一年有大约240个工作日（12个月，每个月20天，无假期）。则生产率为100 000 LOC/(20 × 240)天 = 20.8 LOC/程序员－天。

5.5 GQM方法

Vic Basili和Dieter Rombach在Maryland大学建立了这个方法。GQM代表目标、问题和度量。该方法的基本思想是首先确定方法的目标，接着提出与这些目标相关的问题。最后，建立度量以便测量与问题有关的属性的量。

例子5.16

对顾客满意度问题使用GQM方法。

目标——顾客满意度。
问题——在发现问题时顾客感到满意吗?
度量——顾客满意的报告数目。

习题

1. 说明为什么身高的例子符合有效度量的标准。

2. 研究发现，小学生的鞋的尺码与阅读能力之间有很高的相关性。这是否表示鞋的尺码是智力的一个很好的度量?

3. 说明钱币是一种比例尺度度量而不仅仅是一种间隔尺度度量。

4. 说明为什么根据度量理论GPA不合理。

5. 为什么复杂性不易度量?

6. 在经验关系系统上具有一个偏序关系的优点是什么?

7. 为什么判断的数目加1是计算McCabe环数的一个重要方法?

8. 为什么单调性是尺度度量或工作量度量（如Halstead工作量度量）的一个重要特征？

补充问题

1. 标出下列每个度量的合适的尺度：

LOC
McCabe环数
嵌套的平均深度
嵌套的平均深度

2. 用摄氏温度20、30和40说明摄氏和华氏温度尺度为间隔尺度。

3. 说明McCabe环数满足度量的代表性理论。

4. 说明McCabe环数为一种间隔尺度度量。

5. 计算下列源代码的McCabe环数，画出控制流程图并用罗马数字标出区域。

```
read x,y,z;
type = ''scalene'';
if (x == y or x == z or y == z) type =''isosceles'';
if (x == y and x == z) type =''equilateral'';
if (x >= y+z or y >= x+z or z >= x+y) type =''not a triangle'';
if (x <= 0 or y <= 0 or| z <= 0) type =''bad inputs'';
print type;
```

6. 计算图5-5所示的控制流程图的McCabe环数。

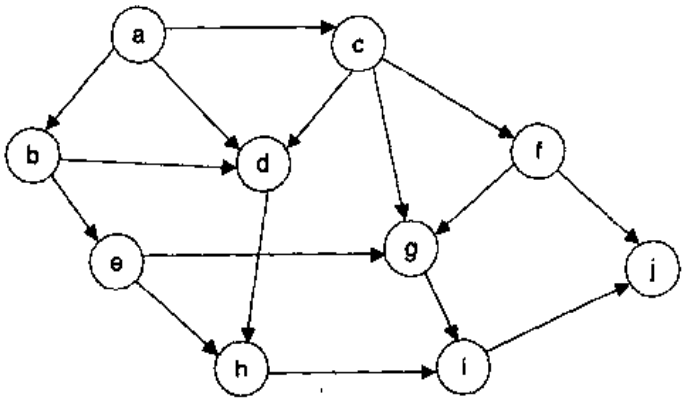


图5-5 控制流程图

7. 计算问题5中关于三角形的代码的Halstead基本度量。

8. 计算下列阶乘代码的Halstead基本度量：

```
int fact (int n) {
        if(n == 0 )
                { return 1 ; }
        else
                { return n * fact (n-1) ; }
        }
```

9. 根据下列代码段画出控制流程图并利用所有三种方法计算McCabe环数。说明哪些代码行由哪些结点表示。在CFG上，用罗马数字标出区域，用小写字母标出判断。

```
cin >> a >> b >> c;
if (a > 10)
{
        cout << ''hello'';
        if (b < a)
        {
                cout << ''part 1'';
                if ( c > a)
                {
                        cout << ''part 2'';
                }
        }
        else
        {
                cout << ''part 3'';
        }
  }
  cout << ''exiting'';
```

10. 在问题9中，统计Halstead的η_1和η_2。计算η和N。以一个名为“串”的操作数统计所有串，给出结果。

习题答案

1. 说明为什么身高的例子符合有效度量的标准。

- 标准1：不同的人可能身高不同。
- 标准2：如果我们认为某两个人在身高方面有关系，则他们的身高数值也有关系。
- 标准3：一个人的身高增加，其身高数值也增加。
- 标准4：不同的人可具有相同的身高。

2. 研究发现，小学生的鞋的尺码与阅读能力之间有高度的相关性。这是否表示鞋的尺码是智力的一个很好的度量？

不对，鞋的尺码与年龄的关系很紧密。年龄又与阅读能力紧密相关。这两者都不能很好地度量智力，因为它们都不满足表示条件。

3. 说明钱币是一种比例尺度度量而不仅仅是一种间隔尺度度量。

钱币有一个很好理解的零概念。在所有货币系统中都存在零，任一系统中的零等于所有别的系统中的零。间隔是固定的。因此，如果你的美元数是我的两倍，那么把我们的钱换为英镑后，你的钱仍然是我的两倍（假定兑换时没有处罚且兑换率相同）。

4. 说明为什么根据度量理论GPA不合理。

要对学分求平均值，那么学分必须是一种间隔尺度。这表示值之间的差是可比的。也就是说，A和B之间的差与D和F之间的差相同。一般来说，在分班时甚至不考虑这一点。

5. 为什么复杂性不易度量?

复杂性没有很好的定义。复杂性包括许多方面，而且对于什么是复杂性，不同的人会有不同的解释。事实上，复杂性可以看做是人与代码之间的交互关系。

6. 在经验关系系统上具有一个偏序关系的优点是什么?

经验关系系统必须具有一种公认的关系。找到某种公认的偏序一般是很容易的，而在所有情形下都得到一致认可可能较为困难。

7. 为什么判断的数目加1是计算McCabe环数的一个重要方法?

构造较大的程序的控制流程图是非常耗时的。

8. 为什么单调性是尺度度量或工作量度量（如Halstead工作量度量）的一个重要特征?

如果增加更多代码会导致工作量的值减少，那么这种度量的行为是不可理解的。工作量度量具有单调性还意味着可以操纵这种度量。

补充问题答案

1. 标出下列每个度量的合适的尺度：

LOC 绝对

McCabe环数	间隔
嵌套的平均深度	顺序
嵌套的最大深度	顺序

2. 用摄氏温度20、30和40说明摄氏和华氏温度尺度为间隔尺度。

相应的华氏温度为68、86和104度。中间温度与最低温度之差为10摄氏度和18华氏度。最高温度与最低温度之差为中间温度与最低温度之差的两倍，分别为20摄氏度和36华氏度。

3. 说明McCabe环数满足度量的代表性理论。

对于经验系统，考虑所有控制流程图的集合。如果一个CFG增加结点或弧可以构造出另一个CFG，则相应的关系是前一个CFG小于或等于第二个CFG。

数值系统（答案集合）可以是整数系统。整数上的关系为标准的小于等于关系。

相应的映射为$e-n+2$。只有两个操作，增加结点和增加弧。增加一条弧表示增加e值。增加一个结点表示在一条弧上增加一个结点。这表示e和n都加1，因此，值也要增加。因此，对于x和y两个CFG，如果x小于y，则y可以通过增加弧和结点而由x得出。这样，映射的值一定是增加或仍然相同，因此满足了不太严格的表示条件。（但不能满足更为严格的表示条件，因为CFG上的序为偏序。）

4. 说明McCabe环数为一种间隔尺度度量。

因为McCabe环数为判断数加1，环数的每个间隔都可以通过增加判断来产生，所以2和3之间的差与10和11之间的差相同。它不是一种比例尺度度量，因为没有明确的零。事实上，环数不可能为零。

5. 计算下列源代码的McCabe环数，画出控制流程图并用罗马数字标出区域。

```
read x,y,z;
type = ''scalene'';
if (x == y or x == z or y == z) type =''isosceles'';
if (x == y and x == z) type =''equilateral'';
if (x >= y+z or y >= x+z or z >= x+y) type =''not a triangle'';
if (x <= 0 or y <= 0 or| z <= 0) type =''bad inputs'';
print type;
```

控制流程图见图5-6。

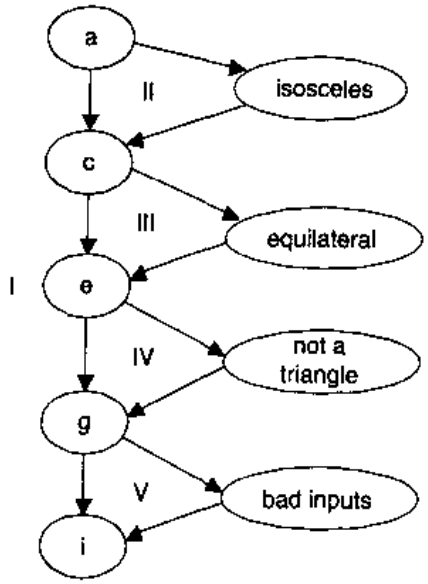


图5-6 控制流程图

区域的数目为5，所以环数为5。也可以用判断来数出环数。离开结点a、c、e、g的路径各有一个判断，因此，共有4个判断。环数为判断数加1，所以环数为5。也可以利用公式$e-n+2$来计算环数。这里，$e=12$，$n=9$，$e-n+2=5$。

6. 计算图5-7所示的控制流程图的McCabe环数。

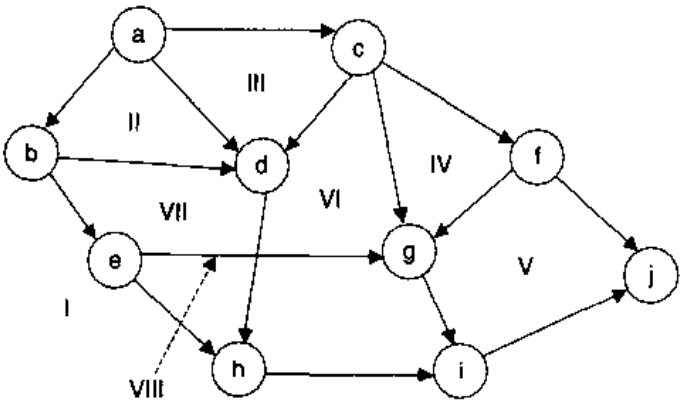


图5-7 控制流程图

请注意，这个图不是平面的（即图中有交叉线）。删除弧直到图为平面图为止，然后为每条删除的弧计一个增加的区域可以得出环数。在这个图中（图5-7），弧e-g被删除，对相应的区域计数，然后为弧e-g增加区域VIII。判断的数目为7（结点a和c上各有两个判断，b、e、f上各有一个判断）。$e-n+2=16-10+2=8$。

7. 计算问题5中关于三角形的代码的Halstead基本度量。

操作符				操作数	
read	1	==	5	string	5
,	2	or	6	x	9
;	7	and	1	y	8
type	6	>=	3	z	8
=	5	<=	3	0	3
if	4	+	3		
( )	4	print	1		

$\eta_1 = 14$ $\eta_2 = 5$ $\eta = 19$ $\eta_2^* = 3$

$N_1 = 51$ $N_2 = 33$ $N = 84$

est $N = 14 \times \log_2 14 + 5 \times \log_2 5 = 14 \times 3.8 + 5 \times 2.3 = 64.7 + 11.5 = 76.2$

$V = 84 \times \log_2 19 = 84 \times 4.25 = 357$

$V^* = 5\log_2 5 = 5 \times 2.3 = 11.5$

$L = 11.5/357 = 0.032$

8. 计算下列阶乘代码的Halstead基本度量：

```
int fact (int n) {
        if(n == 0 )
                { return 1 ; }
        else
                { return n * fact (n-1) ; }
        }
```

操作符：int 2; ( ) 3; { } 3; if 1; == 1; return 2; ";" 2; else 1; * 1; - 1

操作数：fact 2; n 4; 0 1; 1 2;

基本计数：

$\eta_1 = 10$ $\eta_2 = 4$ $\eta = 14$ $\eta_2^* = 1$

$N_1 = 17$ $N_2 = 9$ $N = 26$

est $N = 10 \times \log_2 10 + 4 \times \log_2 4 = 10 \times 3.32 + 4 \times 2.0 = 33.2 + 8.0 = 41.2$

$V = 26 \times \log_2 14 = 26 \times 3.8 = 98.8$

$V^* = 3\log_2 3 = 3 \times 1.6 = 4.8$

$L = 4.8/98.8 = 0.048$

9. 根据本问题描述给出的代码段画出控制流程图并利用所有三种方法计算McCabe环数。说明哪些代码行由哪些结点表示。在CFG上，用罗马数字标出区域，用小写字母标出判断。

见图5-8。

E = 8，N = 6（或者额外的结点可能在if的末尾，因此E = 9，N =7或

E = 10，N =8）=>C = 8−6 + 2 = 4

区域 = 4 => C = 4

判断 = 3 => C = 3 + 1 = 4

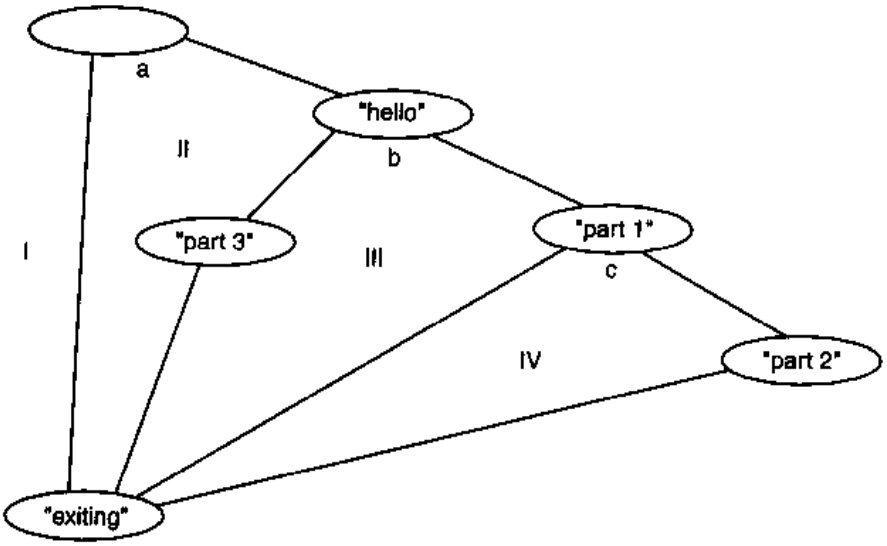


图5-8 控制流程图

10. 在问题9中，统计出Halstead的η_1和η_2。计算η和N。以一个名为“串”的操作数计数所有串，给出结果。

操作符

权 标	计 数	权 标	计 数	权 标	计 数
cin	1	>>	3	;	6
if	3	()	3	>	2
{}	4	cout	5	<<	5
“string”	5	<	1	else	1

操作数

权 标	计 数	权 标	计 数	权 标	计 数
a	4	b	2	c	2
10	1				

请注意，引号可以独立于串而单独计数。else可以作为if-else的一部分计数，所以if出现了两次。

$$\eta_1 = 12 \quad \eta_2 = 4 \quad \eta = 16$$
$$N_1 = 39 \quad N_2 = 9 \quad N = 48$$

第6章 风险分析和管理

6.1 概述

风险是发生不可预见的事件（称风险事件）的可能性。风险包括不确定性（肯定要发生的事件不是风险）和损失（不对项目产生消极影响的事件不是风险）两方面。主动风险管理是试图使风险事件所导致的消极影响降至最低的过程（Roger Pressman, *Software Engineering: A Practitioner's Approach*. 5th ed. McGraw-Hill. New York. 2001.145-163）。

对于风险管理，存在不同的看法。有些专家认为只有当前项目特有的风险才应该在风险分析和管理中考虑。他们的观点是，对于大多数项目来说共同风险的管理应该合并到软件过程中。

6.2 风险确定

这是确定可能的风险的过程。风险可以分为影响项目计划的风险（项目风险）、影响质量的风险（技术风险）或影响产品生命力的风险（商业风险）。有些专家拒绝把所有项目共同的事件在风险管理中考虑，这些专家认为那些共同事件是标准项目规划的一部分。

例子6.1

考虑一个在切削设备上开发安全关键软件的项目。下表中列出了风险及其分类（项目、技术或业务），以及所有项目共同的或对本项目特有的风险。

风　险	项　目	技　术	业　务	共　同	特　殊
硬件不可用		X			X
需求不完整	X			X	
使用特殊方法		X			X
达到所需可靠性的问题		X			X
关键人物的保留	X			X	
低估所需的工作量	X			X	
唯一可能的用户破产			X		X

6.3 风险估算

风险估算在评价一个风险时包括两项任务。第一项任务估算风险发生的概率，即风险概

率（risk probability）。第二个任务是估算风险事件发生的代价，这个代价通常称为风险影响（risk impact）。估算风险概率很困难。已知的风险很容易处理，并且它们已成为软件过程的一部分。当前项目特有的新风险对管理来说是最重要的。风险的成本可以通过以前的项目失败的经验来确定。

6.4 风险揭露

风险揭露是风险事件的预期值。该值等于风险概率乘以风险事件的成本。

例子6.2

考虑两个骰子。假设摇出7为不期望的事件，该事件将使你损失60元。试计算摇出7的风险概率和风险影响。计算风险揭露。

风险概率为两个骰子所有的36种组合中的6种，或1/6。风险影响为60元。风险揭露为1/6乘60元，即10元。

风险决策树

将各种风险进行可视化的技术是建立风险决策树。顶层的分支根据可供选择的风险而分开。下一层的分支依据事件发生的概率。每个叶结点都有某一事件的风险揭露。顶层分支下的所有叶结点的风险揭露的总和为该选择的总风险揭露。

例子6.3

一个朋友提出要与你玩两个打赌游戏之一。游戏A是掷一个硬币两次，如果两次都是正面，他给你10元。每出现一次反面，你给他2元。游戏B仍然是你掷一个硬币两次，但是你玩一次付2元，如果两次都是正面，他付给你10元。那么，你选择玩哪个游戏？

图6-1示出该问题的风险决策树。两个游戏总和都为0.50元。因此，你每玩一次游戏，平均获利50分。选择哪个游戏都一样。

6.5 风险缓解

风险缓解（Risk Mitigation）是试图寻找降低风险事件发生概率或风险发生影响的主动策略。虽然没有减少风险的神奇方法，但有一些常用的方法可用来尽早解决与不确定因素有关的风险。例如，如果对需要使用的特定系统有顾虑的话，最好尽早地调查那些系统。通常，可以通过建立原型来尽早确定问题。

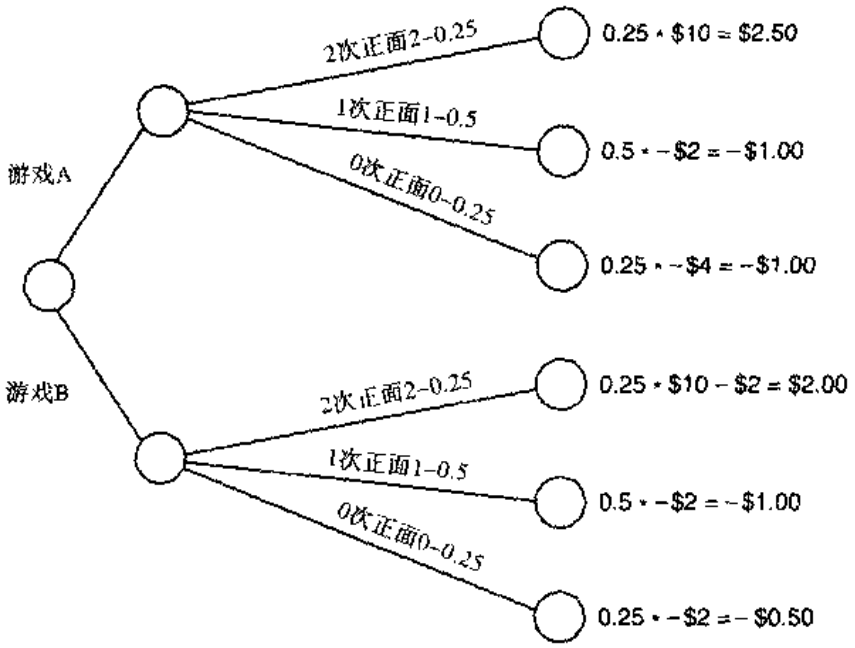


图6-1 例6.3的风险决策树

例子6.4

考虑在风险识别问题中确定的风险。提出一种降低概率或减少影响的方法。

风　险	降低概率	减少影响
硬件不可用	加强硬件设备	建立模拟器
需求不完整	增加需求评审	
使用特殊方法	加强人员训练，雇佣专家	
达到所需可靠性的问题	可靠性设计	
关键人物的保留	多付薪水	雇佣另外的人员
低估所需的工作量	雇佣外部估算人员	提供充裕的时间，反复估算
唯一可能的用户破产	对范围进行外部评估	确定其他可能的客户

6.6 风险管理计划

风险管理计划必须包括风险标识符、风险描述、风险概率的估算、风险影响估算、缓解策略的列表、意外情况计划、风险触发机制（确定何时应该激活意外情况计划）以及责任人。其他内容还包括当前和/或过去的相关度量的情况。

例子6.5

规划一个风险管理的表格，并填写样例数据。

风险ID：1-010-77	概率：10%	影响：很高
描述：指定的硬件可能不可用。		
缓解策略：建立模拟器，加速硬件开发。		
风险触发机制：硬件使进度拖后1周或以上。		
意外情况计划：外购硬件作为备份，在模拟器上交付系统。		
状态/日期/责任人： 建立-01/01/01-Fred Jones Sim.完成-02/10/01-Bill Olson		

习题

1. 为什么说风险管理很重要？

2. 假设你要开车去机场赶一趟从未坐过的航班，那么这次旅行有什么风险？哪些风险可作为一般旅行的风险进行处理？

补充问题

1. 分析第4章补充问题2中的牙科诊所问题所潜在的风险。把风险按一般的或项目特有的分类。

2. 考虑一个未发现错误概率为0.5%的项目，该错误将使公司损失100 000元。计算风险揭露。

3. 考虑对补充问题2使用附加评审方法，即其中有50%的错误为价值100元的错误。使用附加评审计算新的风险揭露。该附加评审方法更好吗？

4. 如果将补充问题3中的附加评审方法的错误概率改为10%，有什么变化？

5. 建立例子6.3中问题的决策树，如果在游戏A中付5元，在游戏B中付4元。你应该选择哪个游戏？

6. X公司的历史数据表明每KLOC的错误率为0.0036。一种新的评审技术表明每100 KLOC价值1000元，并减少错误数50%。假设每个错误平均花费公司10 000元。当前项目的大小估算为50 KLOC。计算每种方法的风险揭露。新的评审方法值得一用吗？

习题答案

1. 为什么说风险管理很重要?

可以管理风险且将风险的影响最小化。然而，最小化风险需要识别风险和管理风险。

2. 假设你要开车去机场赶一趟从未坐过的航班，那么这次旅行有什么风险？哪些风险可作为一般旅行的风险进行处理?

- 一般风险——汽油用光、轮胎漏气、由于天气造成延误、交通事故、忘记手提箱。
- 特殊风险——去机场的高速路正在建设、航班目的地有变化、该航班乘机登记延误。

补充问题答案

1. 分析第4章补充问题2中的牙科诊所问题所潜在的风险。把风险按一般的或此项目特有的分类。

- 一般风险——误解用户的要求、与用户交流不够、用户的硬件有问题、成本超限、项目延迟，等等。
- 特殊风险——与病人记录系统连接。

2. 考虑一个未发现错误概率为0.5%的项目，该错误将使公司损失100 000元。计算风险揭露。

风险揭露为每种概率的风险揭露之和。

$$0.005 \times 100\,000 + 0.995 \times 0 = \$500$$

3. 考虑对补充问题2使用附加评审方法，即其中有50%的错误为价值100元的错误。使用附加评审计算新的风险揭露。该附加评审方法更好吗?

$$0.0025 \times 100\,000 + 0.9975 \times 100 = 250.25 + 99.75 = \$350.00$$

附加评审方法更好。

4. 如果补充问题3中的附加评审方法的错误概率改为10%，有什么变化?

风险揭露将增加。

$$0.0045 \times 100\,000 + 0.9955 \times 100 = 450.45 + 99.55 = 550.00$$

比非附加的评审方法还要差一些。

5. 建立例子6.3中问题的决策树，如果在游戏A中付5元，在游戏B中付4元。你应该选择哪个游戏？

风险决策树如图6-2所示。两个游戏都会损失你的钱。在游戏A中，平均损失0.75元，而在游戏B中将损失1.50元。

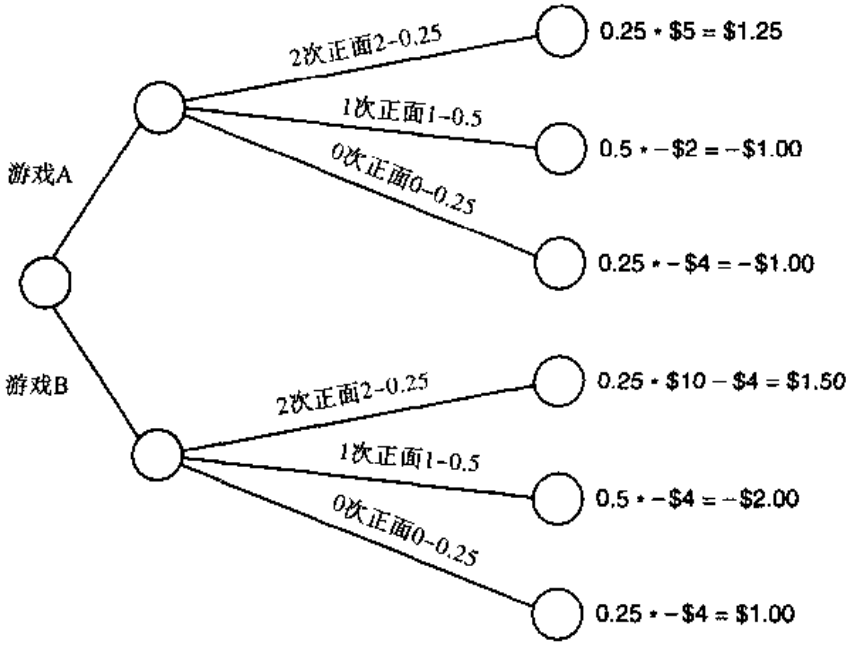


图6-2 补充问题5的风险决策树

6. X公司的历史数据表明每KLOC的错误率为0.0036。一种新的评审技术表明每100 KLOC价值1000元，并减少错误数50%。假设每个错误平均花费公司10 000元。当前项目的大小估算为50 KLOC。计算每种方法的风险揭露。新的评审方法值得一用吗？

• 情况1——无新的评审技术

$$0.0036 \times 50\ \text{KLOC} \times \$10\,000 = \$1800$$

• 情况2——用新的评审技术

$$0.0018 \times 50\ \text{KLOC} \times \$10\,000 + \$500 = \$1400$$

显然，新的评审技术更好。

第7章　软件质量保证

7.1　概述

定义质量有许多方法，但没有一种方法是完美的，就像俗话说的，“眼见为实”。

一种定义是，“质量是能够满足给定需求的一个产品或服务的功能和特性的总和”（英国标准协会）。

另一种定义是，优质软件是能够完成其预定功能的软件。低质量比较容易定义，就是消费者的不满意度。不满意度一般的度量方法是缺陷报告。

达到质量的主要技术是软件评审或走查。检查的目标是发现错误。形式化方法比非形式化方法要好。评价检查最常用的度量单位是：发现的错误/KLOC。可以用发现的错误/花费的小时来度量效率。为确定最佳准备时间，相关科研人员已经做了大量的实验。关于检查会议需要开多长时间等实验工作也已在继续进行。

7.2　形式化检查和技术评审

形式化检查是一种形式的、预先安排的活动。在该活动中，设计者要呈交关于该设计的相关材料以及评审该设计的同级评审组。

进行形式化检查和进行技术评审有很大差别。下面列出形式化检查和其他评审之间的几个区别：

- 使用知识的等级。
- 生产者是活动的参与者。
- 检查一个明确的、完整的产品。
- 主要目的是发现缺陷。
- 在软件开发中按惯例使用形式化检查。
- 赋予特定的角色。
- 检查使用形式化检查的特殊步骤。
- 检查中至少包括三个人。

7.2.1 检查的角色

尽管有各种差异，但大多数检查都具有以下几种基本角色：

- 仲裁者——仲裁者选定检查组，实施检查，报告结果。
- 读者——读者通常不是该产品的生产者，但是读者将引导检查组在检查会议上审查工作成果。
- 记录者——记录者提供检查的记录，并正确报告每项缺陷。
- 生产者——生产者是最初生产该产品的人员。他的角色是在检查中回答问题。生产者还负责修正在检查中发现的问题，然后向仲裁者报告修正结果。

7.2.2 检查的步骤

以下是检查的基本步骤：

1) 概述——当生产者满足入口准则时，安排检查。然后，生产者呈交一份概述。这份概述可以让检查组熟悉要检查的产品。

2) 准备——检查组成员研究该产品。准备阶段所花费的时间根据产品以KLOG为单位的大小来控制。成员可以使用检查表将注意力集中在重大问题上。

3) 检查会——仲裁者负责召开检查会。有些方法使用读者而不是生产者来实际引导检查。记录者对出现的问题做全面记录。检查组的所有成员签署报告。检查组的任何成员如果有不同意见，都可以产生一个少数人报告。

4) 修改——生产者评审该报告，并修正产品。

5) 跟踪——仲裁者评审该报告并修正结果。如果满足出口准则，检查完成。如果不满足标准，仲裁者可以让生产者重新修改产品，或安排重新检查。

7.2.3 检查表

检查表是在评审过程中应检查的项目的列表。有时，这些项目可以以问题形式出现。

检查表的价值是将评审者的注意力集中在潜在的问题上，分析所发现的每个错误是否为检查表的项目。（我记得我曾经为C++中的直接跟在IF语句条件后的分号所引起的问题进行过

很长时间的调试。现在，我编写的C++程序的任何检查表中都包括了对判定条件后面的分号进行检查。）

在检查过程中对不能发现错误的检查表项目应该取缔。过多的检查表项目将削弱检查的效力。

7.3 软件的可靠性

可靠性是在给定时间段中的不失败概率。可靠性通常用R(n)表示，这里的n为时间单位数。如果时间单位为天，则R(1)是在1天中不失败的概率。在给定时间段中的失败概率为1减去该时间段的可靠性（即F (*n*) = 1–R (*n*)）。

软件可靠性是软件遇到数据输入或不能正确处理以产生正确答案的其他情况的频率。软件可靠性与软件的失效无关。软件失败就像是从口袋里选弹子或者蒙着眼睛向墙上的气球投飞镖。

7.3.1 错误率

如果每两天发生一个错误，则即时错误率为每天0.5个错误。错误率是错误间隔的时间（内部错误时间）的倒数。错误率可用来估算失败概率F(1)。除非我们知道某种倾向，否则短期未来行为的最佳估算是当前行为。因此，如果在一天内发现20个错误，则第二天的最佳估计为20。

例子7.1

如果两天后发生一个错误，那么系统在1、2、3和4天中的不失败概率是多少？

如果每两天发生一个错误，我们可以使用0.5作为即时错误率。它通常还用来估算一天的失败概率，即F(1) = 0.5。因此，R(1) = 1–F(1) = 0.5，R(2) = 0.25，R(3) = 0.125，R(4) = 0.0625。

如果我们可以看到错误率的趋势，则可以更好地估算错误率。除了使用等式提供合适的数据外，还可以用错误率的图形形象化地表示其状态。

如果*x*是内部错误时间，则1/*x*是即时错误率。可以绘出即时错误率与相对的错误数或错误的经过时间的图形。用直线连接这些点。当前时间的直线值就是错误率。

该直线与水平坐标轴的交点表示错误率为零的错误数，或者表示删除所有错误需要的时间。如果x轴为经过时间，则该直线下面的区域（单位为时间×错误/时间）表示错误数量。

因此，在测试或观察中，软件失败频率的经验数据通常用来估算当前的错误率，而理论数据将用来提供更长时间范围内的预测。

7.3.2 概率论

F(1)是下一次执行的失败概率。它等于θ，即测试失败情况的百分比（失败通常定义为与需求中指定的内容不同的一个外部行为）。失败概率可以通过当前的即时错误率，或通过错误率图形中估算的错误率来估计。

如果我们知道R (1)，则可以执行n次测试，而不失败的概率为R (n) = R $(1)^n$。

请注意，$F(n)$不是F $(1)^n$。$F(n) = 1 - (1-F(1))^n$。

7.4 统计质量保证

统计质量保证（SQA）是用统计估算软件质量。随机选取一个测试样例的小集合来执行代码，将得到可用于估算质量的结果。有时这种方法称为软件探查（Software Probe）。随机选取的样本的错误率可作为该项目的错误率估算。

如果正确运行的百分比高，则说明软件开发很成功。如果百分比低，则可能需要对开发过程采取补救行动。

7.5 SQA计划的IEEE标准

提高质量的一个重要部分是为质量做规划，即设计有助于提高质量的那些活动。IEEE标准委员会研制了关于软件质量保证计划的标准（Std 730-1989）。

以下是IEEE Std 730-1989中具体说明部分：

1. 目的——此部分列出所涉及的软件以及所涉及的软件生命周期部分。

2. 参考文献——此部分列出在计划中引用的所有文献。

3. 管理

3.1 组织——此部分描述组织的结构和责任，通常包括一张组织结构图。

3.2 任务——此部分列出要执行的所有任务、任务和检查点之间的关系以及任务的顺序。

3.3 职责——此部分列出每个组织单位的职责。

4. 文档

4.1 目的——此部分列出所有必需的文档并说明如何评价文档。

4.2 满足最低要求的文档——此部分描述满足最低要求的文档，此部分通常包括以下内容：

SRS——软件需求说明
SDD——软件设计描述
SVVP——软件验证和验证计划
SVVR——软件验证和验证报告
用户文档——手册、指南
SCMP——软件配置管理计划

5. 标准、准则、约定和度量

此部分确定要使用的标准、准则、约定和度量，以及如何监控和确保一致性。内容至少应包括文档标准、逻辑结构标准、编码标准、测试标准、选择的SQA产品和过程度量。

6. 评审和审计——此部分定义评审/审计所进行的工作，如何完成评审/审计，以及确定所需的进一步的活动。

7. 测试——此部分包括不包含在SVVP中的所有测试。

8. 问题报告——此部分定义报告、跟踪和解决问题的准则和过程，包括组织责任。

9. 工具、技术和方法——此部分将确定特殊的软件工具、技术和方法，并描述其使用方式。

10. 代码控制——此部分将定义维护软件的方法和工具。

11. 媒体控制——此部分将定义标识、存储和保护物理媒体的方法和工具。

12. 厂家控制（针对外购）——此部分将说明厂家提供的产品以保证由厂商提供的软件满足标准。

13. 记录——此部分将确定保留的文档以及收集、维护和保护文档的方法。

14. 培训——此部分将确定必需的培训活动。

15. 风险管理——此部分将指定风险管理的方法和过程。

例子7.2

为一个软件开发项目建立SQA计划的第3和第8部分。假定项目经理是Bill；外部测试组的领导是Tina，成员是Donna和Helen；一个由Mike、Sam和Joe组成的单独配置管理组以及一个由John和James组成的外部质量保证（QA）组。

参见图7-1。

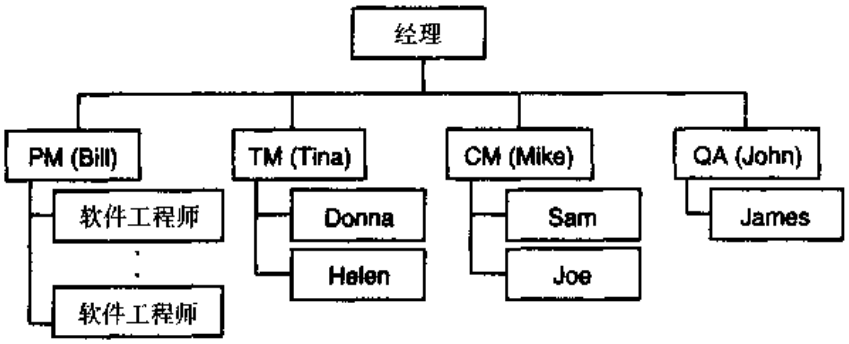


图7-1 SQA计划的第3部分

第3部分

3.1 组织

3.2 任务

评审所有文档。配置管理工具将管理所有文档和源代码模块。所有测试计划将在需求阶段完成，并包含足够多的测试案例。形式化检查将在每个阶段结束时实施。

3.3 职责

项目组负责所有开发，包括需求、设计和实现。项目组提供作为需求的一部分的测试计划。他们还负责所有的文档，其中包括用户手册和培训文档。

测试组负责测试源代码的基准版本。测试组将使用在需求阶段提供的测试计划。将提供附加测试样本以覆盖代码的每个语句。测试计划、需求或测试中的任何矛盾都将报告给项目总经理。

配置管理组将负责接受软件配置项并分配版本号。

质量保证组将负责监视所的评审、走查和检查。QA组将跟踪所有问题报告。

第8部分

在开发单元以外所识别的所有问题都必须报告给QA组，以分配问题报告号。每个组的经理将确认分配给该组的问题报告的修正。QA组将负责跟踪所有问题，并且每周向项目总经理报告一次。

习题

1. 评审和形式化技术评审之间的区别是什么?

2. 如何评价检查表?

3. 检查表应该进行什么样的修订?

4. 影响形式化技术评审效率的因素是什么?

5. 如何度量形式化技术评审的效率?

6. 如果生产者不能解决某个问题会怎样?

7. 如果检查组的一个或多个成员不同意大多数人的意见怎么办?

8. 如果一个普通的骰子被摇了5次，那么任何一次摇动都看不见6的概率是多少?

9. 如果一个骰子被摇了5次，那么至少有一次看见6的概率是多少?

补充问题

1. 为评审C++代码建立一个检查表。

2. 为评审软件设计建立一个检查表。

3. 为形式化检查画一个过程模型。

4. 假定测试表示了操作情况。若在200个测试样例中有10个错误，那么计算此软件系统的可靠性。

5. 假定FTR技术A需要2小时/KLOC的准备时间并分配1小时/KLOC的评审时间，FTR技术B需要1小时/KLOC的准备时间和4小时/KLOC的评审时间。假设在一个有相同源代码的控制试验中，技术A发现12个错误/KLOC，且B发现14个错误/KLOC。试比较这两种技术的效率。

6. 如果某软件在10天的测试中测试100次出现5次错误，那么下一天的软件可靠性的最好估计值是多少？下一周呢？

7. 针对B&B问题提供一个SQA计划（第4章的补充问题3）。

8. 4天后出现一个错误，5天后出现2个错误。画出错误率与错误数量的对比图以及错误率与时间的对比图，并估算系统的错误数并删除所有错误的时间。

习题答案

1. 评审和形式化技术评审之间的区别是什么？

检查或形式化技术评审（FTR）需要明确的、完整的产品。生产者必须是评审/检查中某项活动的参与者。检查/FTR必须是定义的软件过程的一部分。主要目的是发现错误。检查必须遵照有特定角色的特定过程和步骤。

2. 如何评价检查表？

在评审软件代码/文档时使用它。记录与每个检查表项目相关的问题。记录在成功地评审该项之后所发现的错误。

3. 检查表应该进行什么样的修订？

删除与错误检测无关的项目。对生命周期的后阶段中检测到的错误应该生成新的检查表项目。

4. 影响形式化技术评审效率的因素是什么？

准备时间和评审时间。

5. 如何度量形式化技术评审的效率？

通常用错误发现率来度量。该值可以同时用发现的错误/KLOC和发现的错误/评审小时来度量。

6. 如果生产者不能解决某个问题会怎样?

如果项目经理不能解决某个问题，那么要对所有阶段进行重新检查。

7. 如果检查组的一个或多个成员不同意大多数人的意见怎么办?

少数人将提供一份少数人的报告以陈述其观点。

8. 如果一个普通的骰子被摇了5次，那么任何一次摇动都看不见6的概率是多少?

$$(5/6)^5 = 0.4018$$

9. 如果一个骰子被摇了5次，那么至少一次看见6的概率是多少?

$$1-(5/6)^5 = 1-0.4018 = 0.5982$$

补充问题答案

1. 为评审C++代码建立一个检查表。

1) 构造函数中的所有指针都初始化了吗?
2) 所有变量都声明了吗?
3) 每个“{”都有相匹配的“}”吗?
4) 每个全等比较式都有双“=”吗?
5) 所有的while或if条件都以“;”作为结束吗?
6) 每个类声明都以“；”结尾吗?

上述检查表是笔者根据检查C++错误的个人经验建立的样例检查表项目。笔者曾在检查第5项错误时花了很长的时间。

2. 为评审软件设计建立一个检查表。

1) 所有重要的函数都在设计中展示了吗?
2) 所有重要的属性都在设计中说明了吗?
3) 所有名称都与目的和类型相关并且是明确的吗?

4) 指定了类之间的所有关系了吗?

5) 所有函数都有执行该函数所必须的数据吗?

3. 为形式化检查画一个过程模型。

参见图7-2。

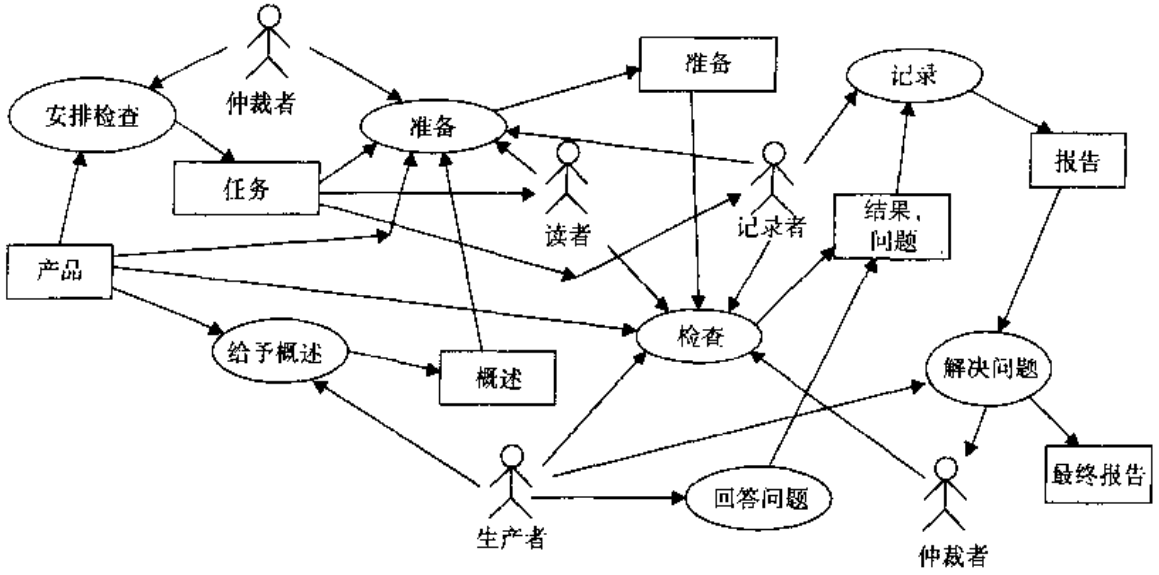


图7-2 检查的过程模型

4. 假定测试表示了操作情况。若在200个测试样例中有10个错误，那么计算软件系统的可靠性。

F(1) = 10 / 200 = 0.05

R(1) = 0.95

5. 假定FTR技术A需要2小时/KLOC的准备时间并分配1小时/KLOC的评审时间，FTR技术B需要1小时/KLOC的准备时间和4小时/KLOC的评审时间。假设在一个有相同源代码的控制实验中，技术A发现12个错误/KLOC，且B发现14个错误/KLOC。试比较这两种技术的效率。

技术A所花的时间是技术B所花的时间的80%，但找到的错误是技术B发现错误的85%。因此，A比B的效率稍高一些。从边缘改善的情况，花2小时/KLOC来查找最后2个错误/KLOC更有效。

6. 如果某软件在10天的测试中测试100次出现5次错误，那么下一天的软件可靠性的最好估计值是多少？下 -周呢?

F(1) = 0.05 R(1) = 0.95

下一天呢?

假设每天测试10次（最近10天的平均值）。$R(10) = R(1)^{10} = 0.95^{10} = 0.598$。

下一周呢?

假设进行70次测试。$R(10) = R(1)^{70} = 0.027$。

7. 针对B&B问题提供一个SQA计划（第4章的补充问题3）。

第1部分 目的

XYZ公司正在为Tom和Sue的提供一夜住宿和早餐的旅馆开发软件。该软件用来维护饭店预定信息并监控费用和利润。该SQA计划是11月1日提交给Tom和Sue的1.0版本。

该SQA计划包括了从需求说明到软件测试的完整的系统的开发生命周期。

第2部分 参考文献

工作语言，B&RSOW1.0版，日期1/1。
XYZ 公司编码标准，4.6版,日期7/8/95。

第3部分 管理

3.1 组织

项目领导（Tom）
开发组（Bill、Jane、Fred）
SQA组（John）

3.2 任务

需求分析及说明
项目计划
成本估算
结构设计
初步设计
实现
测试

项目监控
检查
评审
文档

3.3 职责

项目领导——Tom

职责：项目计划、成本估算、项目监控、所有汇总报告和计划的审批。

需求领导——Bill

职责：需求分析及说明。
项目组成员：Jane、Fred。

设计——Jane

职责：结构设计。

实现领导——Jane

职责：初步设计和实现。
小组成员：Fred、Bill。

测试领导——Bill

职责：全部测试和测试报告。
小组成员：Fred。

文档——Jane

职责：用户手册。

SQA——John

职责：引导完成评审、走查和检查的实施，所有文档和报告的评审，所有问题报告的跟踪，以及每周问题报告的提交。

第4部分 文档

软件需求说明
使用UML和OCL的软件设计
软件测试计划
软件测试报告
用户手册

每个文档将在草案中评审并在最终版本中检查。SQA领导（John）将负责实施所有的评审和检查。

第5部分 标准、实践、约定和度量

该项目将使用XYZ公司编码标准。SQA组将实施所有代码的检查以确保一致，并向项目领导提交一致性报告。

将计算所有组件未调整的功能点，计算所有类的LOC。

第6部分 评审和审计

将完成以下评审。SQA领导将实施每项评审并向项目领导提交以获得审批。

评审：

软件需求
初步设计评审
每个类设计的走查
代码检查

第7部分 测试

测试将按照SQA组开发的并由项目领导同意的测试计划执行。

第8部分 问题报告

所有问题都会报告给相应的领导。如果合适的话，领导将向SQA领导提交一份问题报告，SQA领导将问题报告输入到问题跟踪系统中。问题的处理将报告给SQA领导。向项目领导提交每周的问题跟踪报告。

第9部分 工具、技术和方法

无。

第10部分 代码控制

将使用XYZ公司的配置管理系统。

第11部分 媒体控制

N/A

第12部分 厂家控制（针对外购）

N/A

第13部分 记录——收集、维护和保留

N/A

第14部分 训练

N/A

第15部分 风险管理

N/A

8. 4天后出现一个错误，5天后出现2个错误。画出错误率与错误数量的对比图以及错误率与时间的对比图，并估算系统的错误数并删除所有错误的时间。

如图7-3所示，错误率与错误数量的图形显示了为6的截距。因此，该系统中还剩余4个错误。

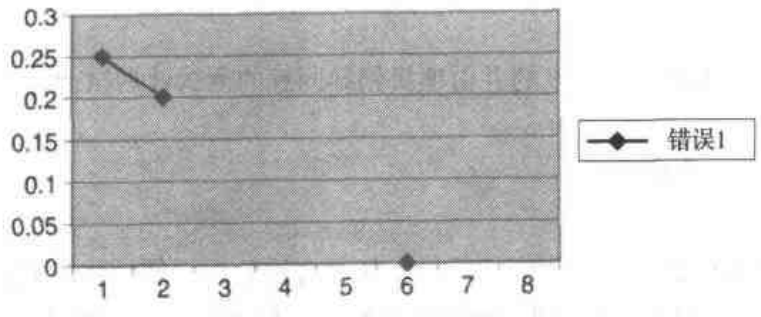


图7-3 错误率与错误数量

如图7-4所示，错误率与时间的图形显示截距为29~30之间。因此，完全消除错误大约需要20个单位时间。

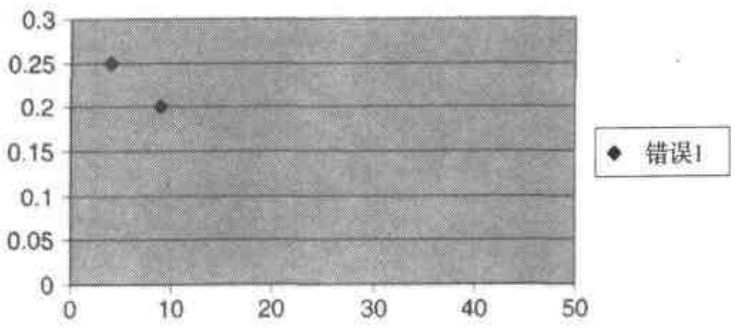


图7-4 错误率与时间

第8章 需　　求

8.1 概述

需求阶段的目标是从用户那里提取需求。这一般是通过与用户讨论后建立图表和需求规格说明来实现的。然后，用户评审相应的图表和规格说明，以确定软件开发人员是否理解了需求。因此，为了与用户沟通要开发的软件所必需的基本组成部分，必须将图表和规格说明返回给用户。

以下小节介绍了实现这种沟通的有用的图表和需求规格说明。

8.2 对象模型

面向对象（OO）方法学中的基本方法是建立对象模型（参见2.4节），这个模型描述问题领域的现实世界的子集。目的是给问题领域建模，但不设计具体的实现。因此，理解问题所必需的实体都应该包括在内，即使不打算把它们包含在解决方案中也是如此。对象模型所包含的属性和方法也是理解问题所需要的内容，而不仅仅是对解决方案最为重要的那些内容。

下面是用于需求的对象模型规则：

- 理解问题领域的所有重要的现实世界的实体都必须包括在内。
- 理解该问题领域的所有重要的方法和属性都必须包括在内。
- 不应该包括只对实现有重要意义的对象、属性和方法。

例子8.1
画出图书馆问题的对象模型（参见例子2.6）。
参见图8-1。

例子8.2
画出简化的类vi编辑器的对象模型。

参见图8-2。

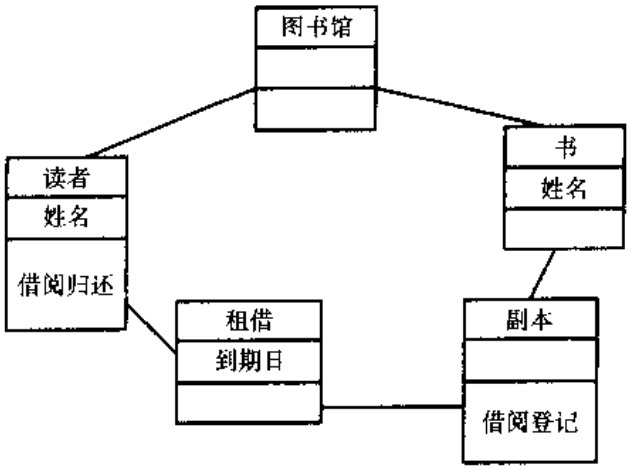


图8-1 图书馆问题的对象模型

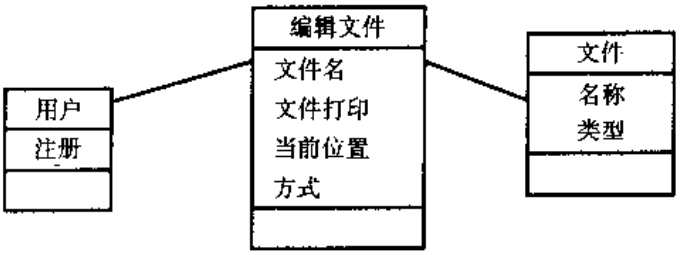


图8-2 简化的类vi编辑器的对象模型

8.3 数据流程建模

尽管数据流程图（参见2.2节）在OO开发中不常使用，但它是OO软件开发之前的重要部分。数据流程图（DFD）在许多系统的规格说明中仍然扮演着重要的角色。数据流程图的重要性在于说明了哪些数据对于哪个组件有效。了解数据的有效性通常有助于理解某个组件应该做什么以及它将如何完成任务。

例子8.3

画出简单的Unix的类vi编辑器的DFD。

参见图8-3。

8.4 行为建模

行为建模指系统的行为，这种行为通常来自用户的观点。这些图表通常用于指明要研制

的系统的范围。重要的是图表能捕获系统的基本方面，并能够及时与开发人员和用户进行沟通，以确保这是用户所期望的系统。

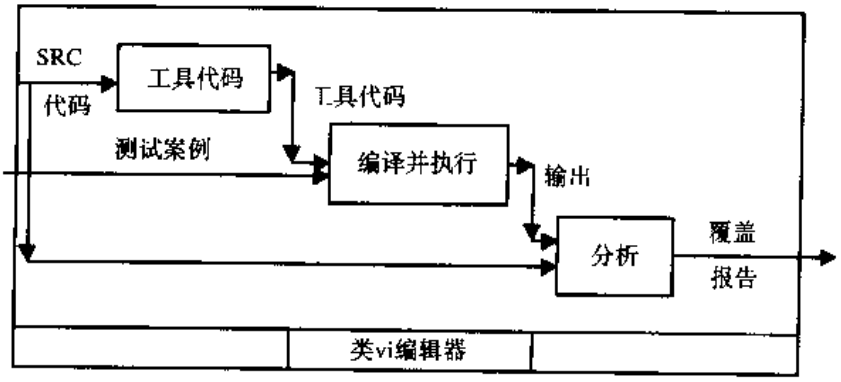


图8-3　Unix的类vi编辑器的数据流程图

8.4.1 用例

用例图从用户的观点表达系统功能（参见2.5节）。所有关键性功能都必须涉及到。但是更高级阶段所隐含的例程函数不必专门提及（明确性可减少误传的危险）。文字需求将详细叙述其中每个功能。

例子8.4

画出类似简化的Unix的类vi编辑器的用例图。

参见图8-4。

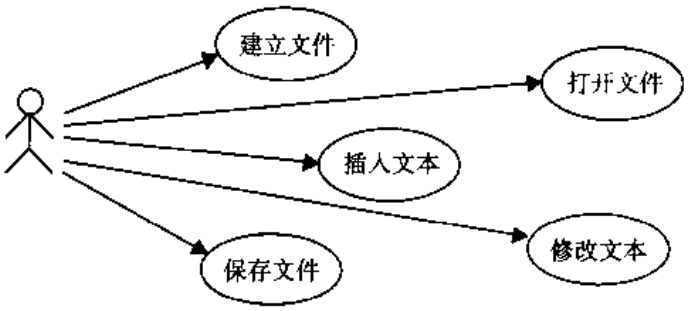


图8-4　编辑器的用例图

该图的基本功能是文件处理（建立、保存及打开）。插入和修改功能是文本编辑功能的较高级部分。请注意，像搜索、复制和移动等功能被忽略了，因为它们没有被明确提到。

8.4.2 场景

场景是完成某个用户任务的一组动作序列。每组动作序列只表示一个单独的场景。场景用来说明系统的一个重要功能或建议的使用方法。

在UML中，使用交互图（参见第2章）指明场景。场景还可以通过列出动作序列来指明。

例子8.5

用例子8.4中的每个用例为简化类vi编辑器编写场景。用分号分隔动作。用圆括号包括注释或条件。

```
Create file
     vi filename (file does not already exist)
Open file
     vi filename (file already exists)
Insert text
     I ; <desired text> ; <esc>
     i ; <desired text> ; <esc>
     O ; <desired text> ; <esc>
     o ; <desired text> ; <esc>
     A ; <desired text> ; <esc>
     a ; <desired text> ; <esc>
Modify text
     cw ; <new text> ; <esc>
     dw
     dd
     x
Save file
     ZZ
```

注意，没有给出所有的动作序列。为简单起见，没有给出所有的操作。在实际的规格说明中，应该给出所有的操作和重要的操作序列。在此例中，每个场景只代表使用的一个部分。另外，每个场景都可能从打开文件运行到关闭文件。

8.4.3 状态图

状态图已在第2章中介绍过了。当状态图作为需求规格说明的一部分时，状态反映了用户可理解的领域条件。只对实现有重要意义的状态应该合并到领域的重要状态中。另外，所允许的转换必须包括场景中的所有允许的转换。不打算包含在该建议系统中的动作序列也不应该包含在状态图中。这样做可能有难度，因为在场景中的一个转换不能阻止其他转换的出现。

以下是在需求规格说明中使用状态图的规则：

- 所有状态必须是领域有效的。
- 场景中的所有序列都允许使用。
- 所有禁止的场景都不允许使用。

例子8.6

画出简化类vi编辑器的状态图。

参见图8-5。

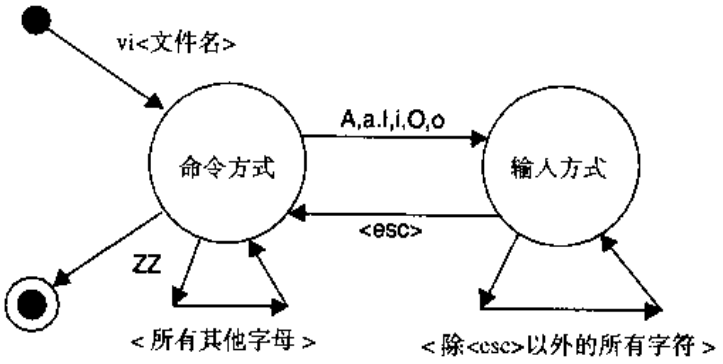


图8-5　简化类vi编辑器的状态图

该状态图是利用对程序的理解将状态合并到具有领域相关性的状态中建立的。该图很好地向用户表达了所建议的简化类vi编辑器的行为。

8.5　数据字典

数据字典是记录系统中每个数据元素的信息的表。最初，在需求阶段，数据字典是问题领域中的数据项。

一个典型的条目应该包括项的名称、属于哪个类、该数据项的类型以及该项的语义。

例子8.7

建立例子2.6中的图书馆问题的数据字典。

Name	Class	Type	Size	Semantics
Author	Book	String	< 40 char	Last name, first name (may be truncated)
Book	Book	Object		Abstract concept of the book
Book ID	Copy	Key		Key to info about the book
Borrower	Loan	Key		Key to patron who made this loan
Copy	Copy	Object		Library's physical copy of a book

Copy ID	Copy	Key		Key to physical copy being borrowed
ISBN	Book	String	10-20 char	International Standard Book Number
Loan	Loan	Object		A borrowing that is still active
Name	Patron	String	< 40 char	Last name, first name (may be truncated)
Patron	Patron	Object		Registered holder of library card
Title	Book	String	< 50 char	First 50 char of title from title page

8.6 系统图

系统图（system diagram）是非形式化定义的图，用来提供要研制的系统的概述。当用更加形式化定义的图都难以表达必要的概述时经常会使用系统图。系统图通常结合数据流和用例图的特点。系统图通常用椭圆表示系统的处理部分，用数据对象表示文件和/或数据库，用方框表示数据，小人表示人。弧表示功能的流入和流出。绘制系统图的难点是保持使用符号的一致性并提供充分的细节。

例子8.8

画出简化类vi编辑器的系统图。

参见图8-6。

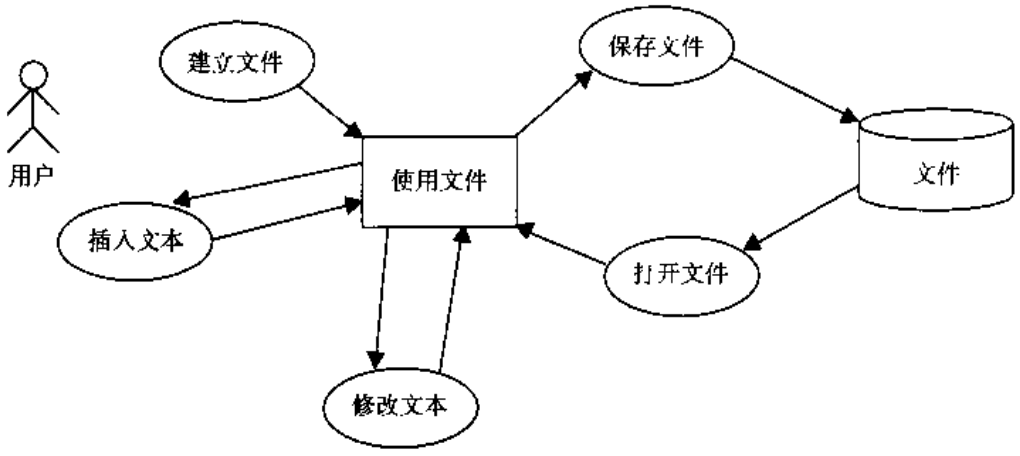


图8-6 简化类vi编辑器的系统图

例子8.9

画出机器编写源代码，编译机器编写的代码，用测试案例执行该代码，然后分析结果的测试工具的系统图。

参见图8-7。请注意，来自编译器过程的输出是另一个过程。

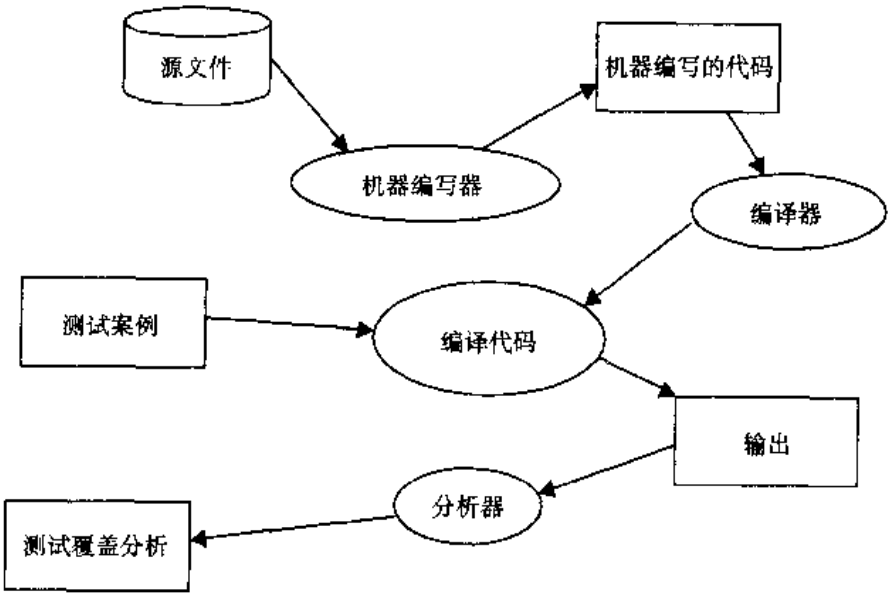


图8-7　测试工具的系统图

8.7　软件需求规格说明的IEEE标准

以下SRS提纲基于IEEE830-1993：

1. 引言——此部分提供规格说明的概要。

1.1　目的——此部分必须描述SRS的目的和所针对的人员。

1.2　范围——此部分必须标识软件产品，解释该产品将要做什么和不做什么，描述该软件的应用，其中包括收益、对象和目标。

1.3　定义——此部分必须确定在规格说明中使用的所有的项目、首字母缩写词和缩写词。

1.4 参考——此部分必须确定在规格说明中引用的所有文献。

1.5 概述——此部分必须描述文档的其余部分所包含的内容。

2. 总体描述——此部分将提供理解需求的其余部分的背景。

2.1　产品透视——此部分必须将该产品与其他产品联系在一起。它通常将包括更大的

系统的一个方块图。它应该说明约束条件（如与其他软件的系统接口）、用户界面（如屏幕格式、定时）、硬件接口、软件接口（如接口软件的版本）、内存约束、操作（如操作方式）以及站点调整约束条件。

2.2 产品功能——此部分必须包括该产品主要功能的概述。

2.3 用户特征——此部分必须包括用户的教育程度、经验和技术专长。

2.4 约束——此部分必须包括未在2.1中列出的任何其他的约束（如规章限制）。

2.5 假设与相关性——此部分必须包括如果不为真则需要改变需求的所有假设。

2.6 需求的分配——此部分必须确定可以延缓到产品今后的版本中实现的需求。

3. 特定的需求——根据IEEE标准830："SRS的此部分应该包含能使设计人员设计出满足需求的系统的所有软件需要，并使测试人员能够测试该系统是否满足那些需求。"这是必须记住的一条重要标准：SRS应该足够详细，以便可以从该SRS中直接构造设计和测试。同样，根据IEEE标准830："这些需求应至少包括进入系统的所有输入的描述（激励源）、来自系统的所有输出（响应）以及由系统响应某个输入或支持某个输出所执行的所有函数。"

3.1 外部接口需求——此部分必须描述系统的所有输入和输出。这是2.1中信息的细节。

3.2 函数——此部分必须描述系统的所有函数。必须包括输入的有效性检查、对异常情况的响应、参数的影响和输出与输入之间的关系。

3.3 性能需求——此部分必须描述静态和动态需求。

3.4 设计约束——此部分必须描述设计中的任何约束条件。

习题

1. DFD图较之其他图的优点是什么？

2. 需求规格说明中的行为规格说明和图的目的是什么？

3. 在用例图中哪些功能是重要的？

4. 评价场景应该使用什么准则?

5. 评价状态图应该使用什么准则?

6. 系统图的优点是什么?

7. 系统图面临的主要问题是什么?

补充问题

1. 画出B&B问题（参见补充问题4.3）的对象模型。

2. 画出牙科诊所问题（参见补充问题4.2）的对象模型。

3. 画出B&B系统（参见补充问题4.3）的DFD。

4. 画出牙科诊所系统（参见补充问题4.2）的DFD。

5. 画出B&B问题（参见补充问题4.3）的用例图。

6. 画出牙科诊所问题（参见补充问题4.2）的用例图。

7. 写出B&B问题（参见补充问题4.3）的场景。

8. 写出牙科诊所问题（参见补充问题4.2）的场景。

9. 画出整个B&B问题（参见补充问题4.3）的状态图。

10. 画出B&B问题（参见补充问题4.3）中的预定数据项的状态图。

11. 画出牙科诊所问题（参见补充问题4.2）的状态图。

12. 画出B&B系统（参见补充问题4.3）的系统图。

13. 画出牙科诊所系统（参见补充问题4.2）的系统图。

习题答案

1. DFD图较之其他图的优点是什么?

通过系统的数据流以及每个过程的可用特定数据都清楚可见。

2. 需求规格说明中的行为规格说明和图的目的是什么?

目的是将所要研制的系统的概述与用户沟通，以保证对所要研制的系统的总行为的理解。

3. 在用例图中哪些功能是重要的?

重要的功能是传达必需的功能性的那些关键功能。

4. 评价场景应该使用什么准则?

每个重要的功能序列都需要根据用户的观点给出。

5. 评价状态图应该使用什么准则?

状态图要显示所有可能的转换。图中的每个弧都要有一个引起转换的事件。从开始结点到每个结点以及从每个结点到最终结点都必须有一条路径。

6. 系统图的优点是什么?

非形式化，可以更灵活地表达总的系统的思想。

7. 系统图面临的主要问题是什么?

缺少形式化可能会因某个符号含义不明确而导致不确定的图。

补充问题答案

1. 画出关于B&B问题（参见补充问题4.3）的对象模型。

参见图8-8。

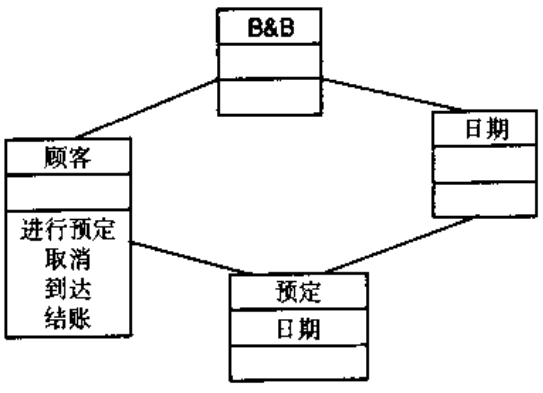


图8-8 B&B对象模型

2. 画出牙科诊所问题（参见补充问题4.2）的对象模型。

参见图8-9。

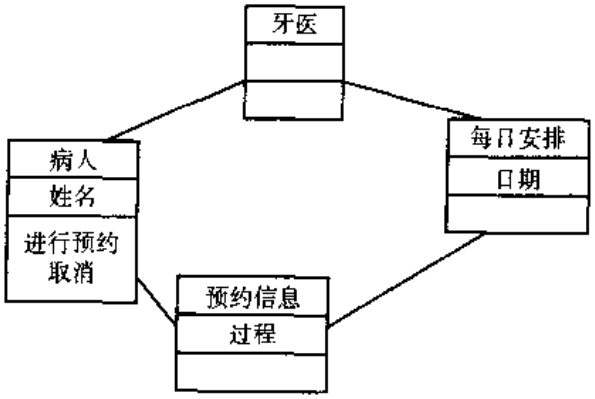


图8-9　牙科诊所对象模型

3. 画出B&B系统（参见补充问题4.3）的DFD。

参见图8-10。

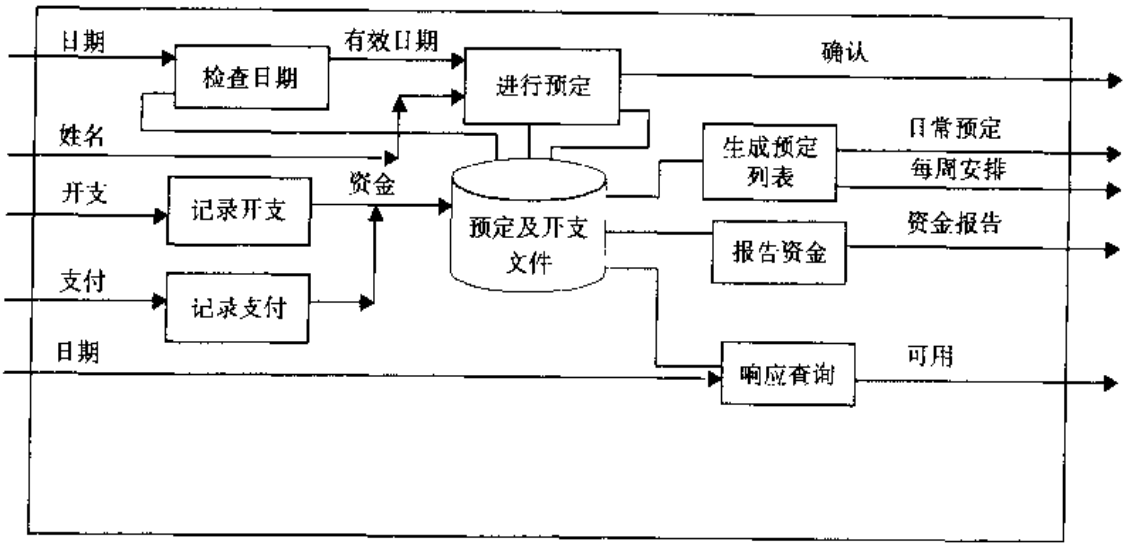


图8-10　B&B数据流程图

4. 画出牙科诊所系统（参见补充问题4.2）的DFD。

参见图8-11。

5. 画出B&B问题（参见补充问题4.3）的用例图。

参见图8-12。

6. 画出牙科诊所问题（参见补充问题4.2）的用例图。

参见图8-13。

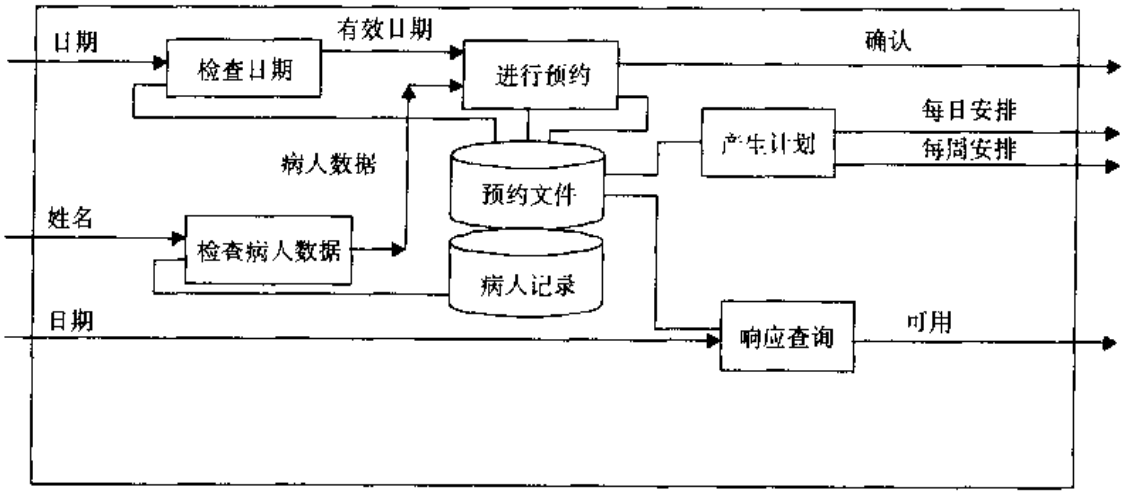


图8-11 牙科诊所的数据流程图

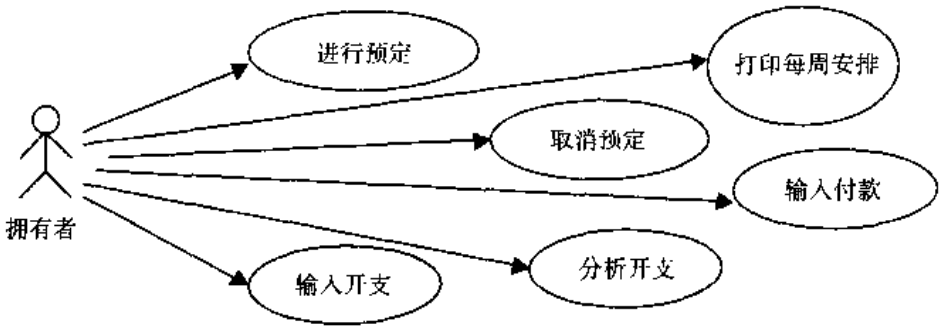


图8-12 B&B的用例图

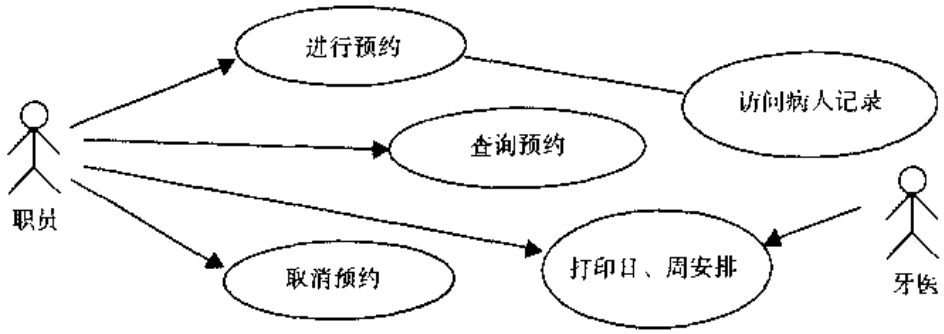


图8-13 牙科诊所的用例图

7. 写出B&B问题（参见补充问题4.3）的场景。

顾客电话1：

顾客打电话询问是否可以预定某指定日期的房间。

Sue拿出那一周的日历。
有一个空闲房间。
Sue报价。
Sue得到顾客的姓名、地址、电话号码和信用卡号。
Sue输入信息。
顾客提供信用卡担保预定。

顾客电话2：
顾客打电话询问是否可以预定某指定日期的房间。
Sue拿出该周的日历。
没有空闲房间。

顾客电话3：
顾客打电话询问是否可以预定某指定日期的房间。
Sue拿出该周的日历。
有一个空闲房间。
Sue报价。
Sue得到顾客的姓名、地址、电话号码和信用卡号。
Sue输入信息。
顾客交预定保证金。
预定日期已过。
另一个顾客请求在该日期预定房间。
删除无保证金的预定。

8. 写出牙科诊所问题（参见补充问题4.2）的场景。

(1) 正常情况

某个病人请求预约。病人的名字由系统识别出。系统建议某个时间。病人接受该时间，接待员输入该预约。在该预约前的两天，接待员得到一份有病人姓名和电话号码的提醒清单。接待员打电话提醒病人。病人如约到来。该预约后，牙医助手安排该病人的下一次预约。

(2) 新病人

某个病人请求预约。系统不认识该病人的姓名，必须将该病人的信息输入到病人记录系统中。

(3) 多个预约

某病人请求在未来两年内进行6个月的预约。接待员将其姓名输入到系统中，在接收后，输入认可的预约。

9. 画出整个B&B问题（参见补充问题4.3）的状态图。

参见图8-14。

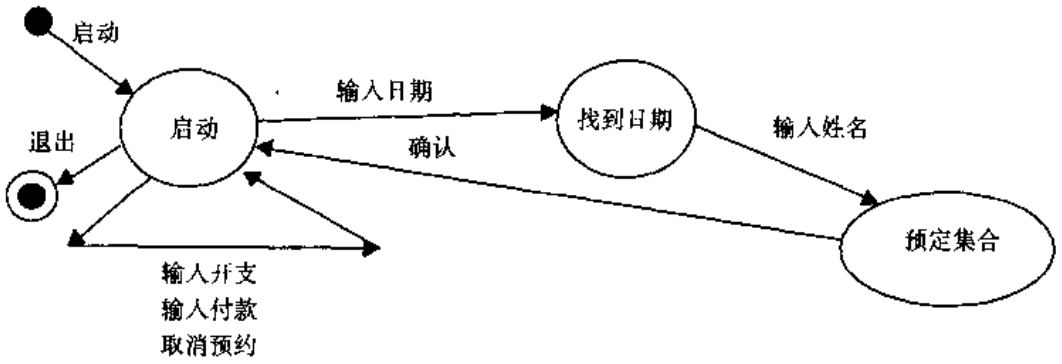


图8-14 整个B&B问题的状态图

10. 画出B&B问题（参见补充问题4.3）中预定数据项的状态图。

参见图8-15。

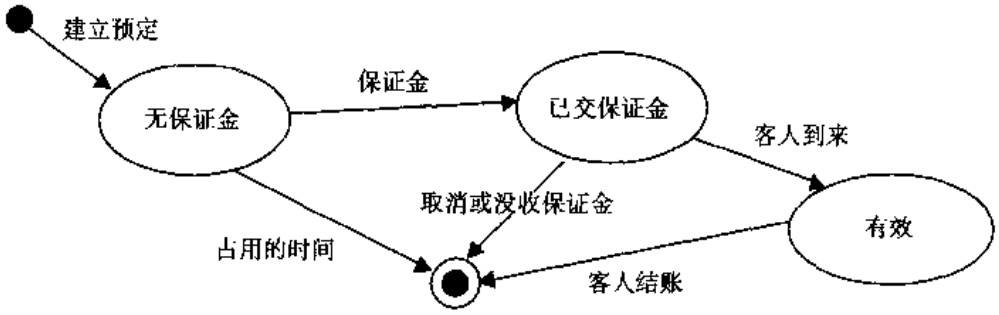


图8-15 预定数据项的状态图

11. 画出牙科诊所问题（参见补充问题4.2）的状态图。

参见图8-16。

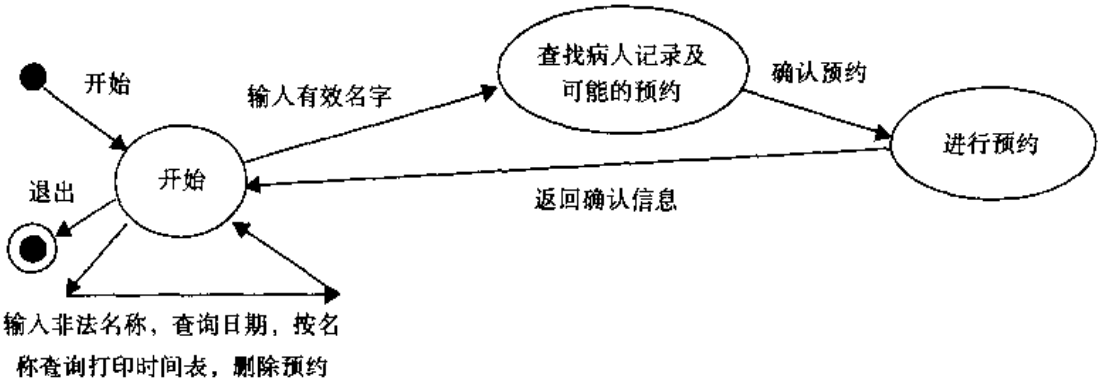


图8-16 牙科诊所的状态图

12. 画出B&B系统（参见补充问题4.3）的系统图。

参见图8-17。

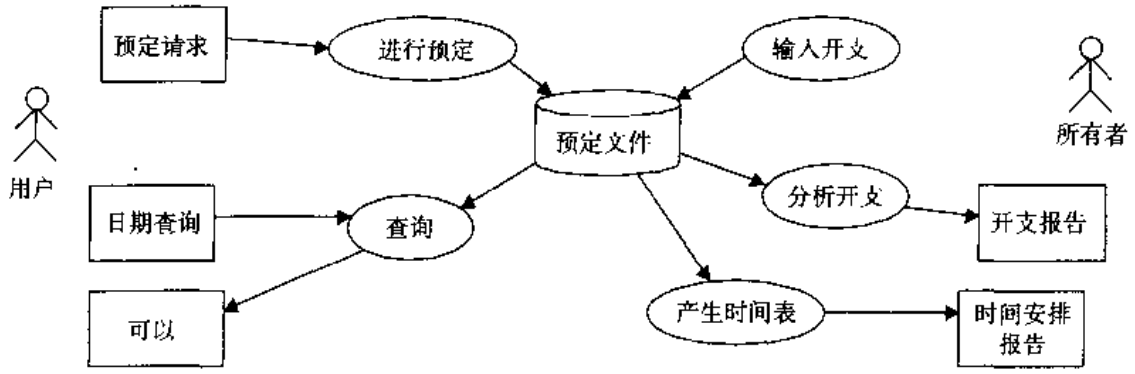


图8-17　B&B的系统图

13. 画出牙科诊所系统（参见补充问题4.2）的系统图。

参见图8-18。

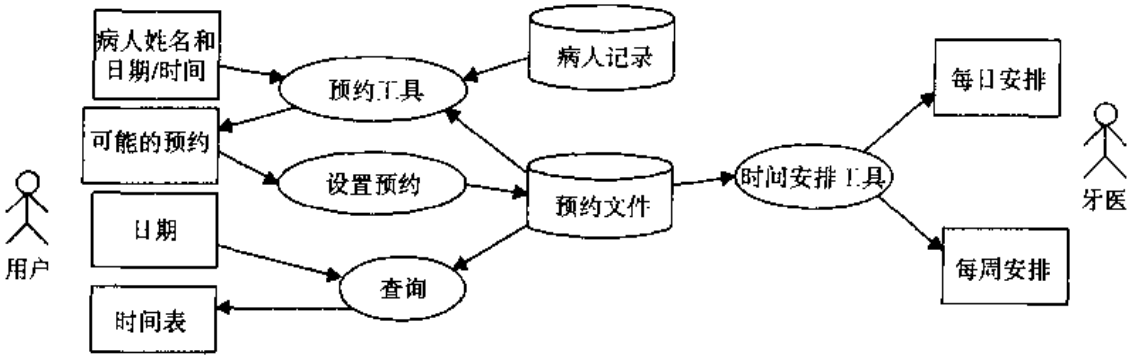


图8-18　牙科诊所的系统图

第9章 软件设计

9.1 概述

设计是“为了充分定义一个设备、一个过程或一个系统使其满足物理实现而使用各种技术和原理的过程”(Taylor，*An Interim Report of Engineering Design*, MIT, 1959.)。设计也是软件开发过程中最富有艺术性或创造力的部分，所以很少能够写出指导设计的规则。

设计过程把需求中的“做什么”转化为设计中的“如何做”。设计阶段的结果应该是生成一个用于实现系统的、有详细细节的、无需在今后与需求编写人或用户进一步沟通的文档。

设计过程还将术语从需求的问题空间转化为实现的解决方案空间。有些作者讨论面向对象的分析（OOA）对象（属于问题/领域空间）以及面向对象的设计（OOD）对象（属于解决方案/实现空间）。例如，在问题空间中，我们讨论诸如人这样的现实世界的对象；而在解决方案空间中，我们讨论称为人的C++类。

Gunter et al.（Gunter, Gunter, Jackson, and Zave. “A Reference Model for Requirement and Specifications.” *IEEE Software*, May/June 2000. 37-43.）写到了关于环境（世界）的现象和实现（机器）的现象。一个现象可以是可视的或隐藏的。面向用户的需求可以按照环境中的隐藏或可视的现象来表达。但是，作为开发基础使用的规格说明必须位于环境和实现之间，并且必须根据可视现象表达。这个规格说明是设计的起点，在本书中将其称为开发规格说明。

例子9.1

要求机器人用黑白相机查找特殊品牌的汽水罐，并将这些铁罐放到某个回收位置。这样的描述便可以是面向用户的需求，并由环境中的一个现象组成。但是，汽水罐是环境中的一个隐藏现象。即实现不知道汽水罐，它只知道汽水罐的黑白图像。这是可视现象。当编写作为设计起点的规格说明时，必须按照这些图像进行讨论。假设（并需要验证）只有真正的汽水罐提供这些图像。例如，如果环境中的墙壁被含有汽水罐图像的广告所覆盖，则问题将相当复杂。

例子9.2

确定哪个是图书馆系统的环境中的现象，哪个是实现中的现象。

实际的图书是一个环境隐藏的现象。系统决不知道具体的书籍。当图书管理员搜索图书

时，实际上是在搜索条形码。此条形码不是ISBN，但必须反映一本书的多个副本。条形码是环境可视的。实现中也许为该书的数据使用其他的标识符或指针。这个内部的标识符是实现隐藏的。

应该根据书的条形码编写开发规格说明。具体的书籍或内部标识符都不应该在开发规格说明中涉及。

9.2 设计过程的各个阶段

以下是设计的各个阶段：

数据设计——该阶段产生数据结构。
结构设计——该阶段产生结构化单元（类）。
接口设计——该阶段说明单元之间的接口。
程序设计——该阶段说明每个方法的算法。

例子9.3

针对图书馆问题（参见例子8.1和2.6），根据如图9-1所示的对象模型中的数据项设计图书馆的类/数据结构。

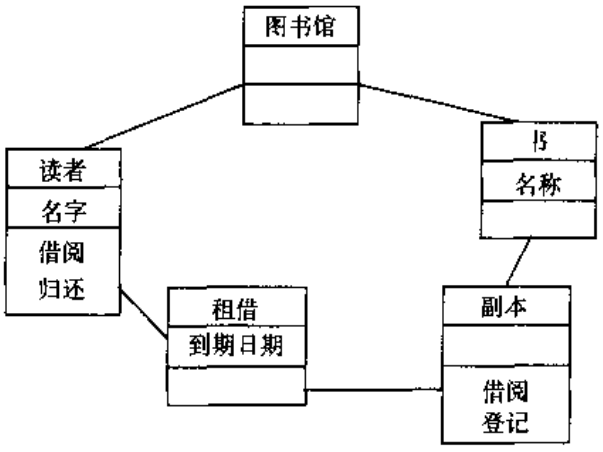


图9-1 图书馆的对象模型

在本例中，数据设计和结构设计阶段已经合并在一起。本例集中在租借和登记借出功能上，而不关心其他必需的任务，如管理、分类、分配过期罚款、下架的书籍以及读者管理。

域实体“书”可能不继续进入设计中。它将与“copy”结合成为一个类/数据结构，该结构存储所有关于副本的信息。可以使用ISBN以及一个副本号作为书的唯一标识符。读者信息

将存储在第二个数据结构中。每个记录可以由一个唯一的读者ID号标识。租借信息或许使用或不使用单独的数据结构。如果借阅信息需要在书籍归还后继续保存，最好是单独的类/数据结构。另外，读者ID以及书的到期日期可以是副本类/数据结构的一部分。

请注意图9-2，与问题/领域空间相比，在实现/解决方案空间中增加了更多的数据项。它认为“ISBN”是问题空间而不是解决方案空间的一部分。但是，许多图书馆系统不允许用户通过ISBN 进行搜索。

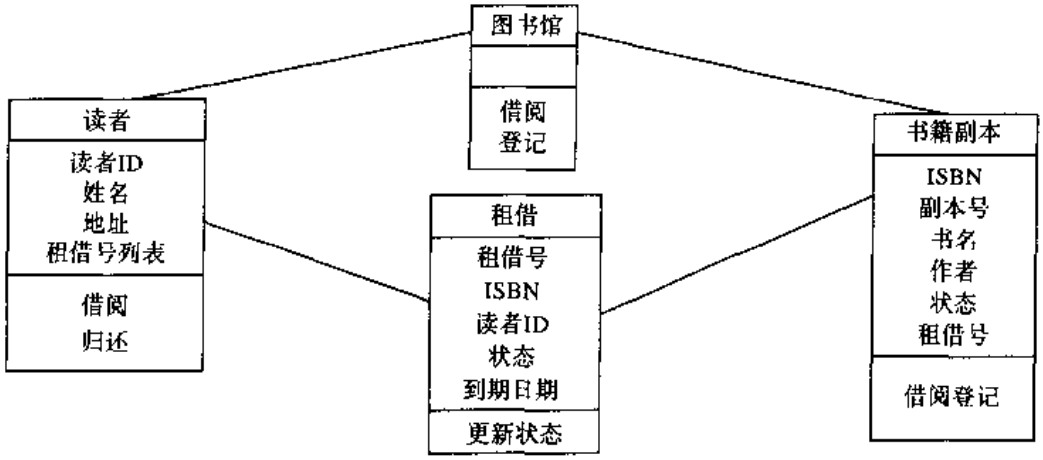


图9-2 图书馆问题的类图

接口

接口规格说明是一个模块的外部行为的规格说明。它应该足够完整，以便调用模块精确地了解被调用模块在各种环境下都做些什么。其完整性还有助于实现者精确地了解必须提供什么信息。

在OO模型中的接口规格说明通常是公共方法及其与方法相关的语义的特征。接口还可以作为整个系统的行为的形式化规格说明的一部分给出。

接口还可以是一个方法的不变量、前提条件和后置条件。

例子9.4

使用例子9.3中产生的类图设计图书馆问题的borrow函数的接口。

patron和bookcopy都有“borrow”方法。调用这两种方法之一都可以建立loan实例。类图中没有表明哪个方法建立此实例。但是，如果指定每个方法的参数和返回类型，就很清楚了。

```
method patron::borrow
     input parameters - isbn
     return type - int
          0 if book is not available
          1 if book is available and loan instance created successfully
          -1 if error condition
method bookcopy::borrow
     input parameter - loannumber
     return type - int
          0 if bookcopy is not available
          1 if bookcopy updated successfully
```

9.3 设计概念

设计的两种方法称为精化和模块化：

精化——此设计方法通过使细节不断分层细化来进行设计。有时此方法称为“自顶向下”的设计。

模块化——此设计方法是构造方法，它将软件划分成一些较小的块。这些块集成在一起满足问题的需求。

例子9.5

精化图书馆问题中的借书函数。

顶层由函数borrow book与两个参数（书的标题和读者姓名）开始。

下一层精化增加loan实体的概念。它可能有以下部分：查找给定标题的书籍，查找给定读者名的读者，建立给定书籍和读者的IDS的借出实例。

再下一层精化将扩展每个部分。如果该书找到并可用，则find book返回ISBN。如果未找到，返回0。如果该书正在使用，返回-1。如果找到读者且遵守一切规则，则find patron返回读者ID。如果未找到，返回0。如果读者没有资格借书，返回-1。如果成功建立，则create loan返回1。

设计属性

以下是设计的三个属性：

抽象性——一个对象如果去掉了不必要的细节就是抽象的。同样，艺术抽象试图只用很

少的细节来传达图像。软件设计抽象试图让设计人员将精力集中于实质问题，而不用关心无关紧要的低层细节。良好的抽象隐藏了无关紧要的细节。

内聚性——如果某种材料是粘在一起的，就是这种材料内聚的。如果过程中的所有语句都与每个输出有关，则该过程是内聚的。如果类中所有的属性都由每个方法使用，则该类是内聚的。也就是说，如果模块中的一切都是相关的，就实现了模块的内聚。通常期望内聚性高。

最初，内聚性是根据内聚类型定义的（W.Stevens, G.Myers, and L.Constantine, "Structured Design," *IBM Systems Journal*, 13 #2, 1974, 115-139.）。这些类型包括一致的、逻辑的、临时的、过程的、通信的、顺序的和函数的。临时内聚是指由于所有的函数必须在同一时间执行而组合在一起的情况。逻辑内聚是指各函数在逻辑上合成一个整体。

耦合性——耦合是实现模块互连的一种方法。如果一个模块中的某个变量的变化可能涉及另一个模块中变量的变化，则说这两个模块是耦合的。通常期望耦合度最低。

例子9.6

给出图书馆问题中借阅函数的抽象性。

borrow函数有三个类：library、patron和bookcopy。最好的抽象是：图书馆的borrow函数尽可能少地知道patron和bookcopy函数。例如，borrow函数需要知道loan类吗？

如图9-3所示，如果library中的borrow函数只调用较低层次上的borrow函数，则它有良好的抽象性。较低层的类将处理建立loan实例以及将指针传递给其他较低层的类的细节。

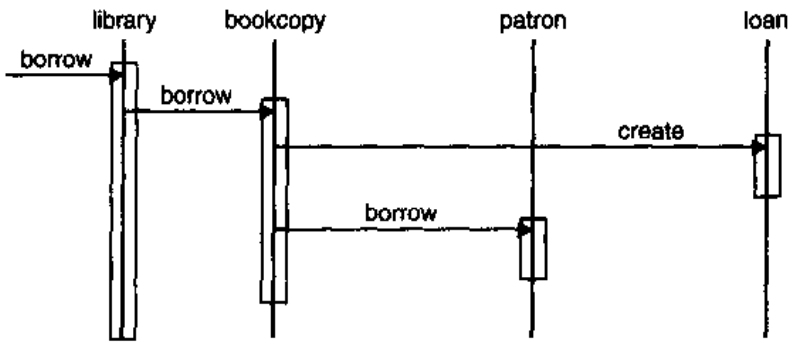


图9-3 图书馆问题中borrow函数的抽象

但是，如果library中的borrow函数了解loan类，则它可以检查该book的可用性，建立loan实例，并同时调用两个较低层的borrow函数设置loan实例的值（参见图9-4）。

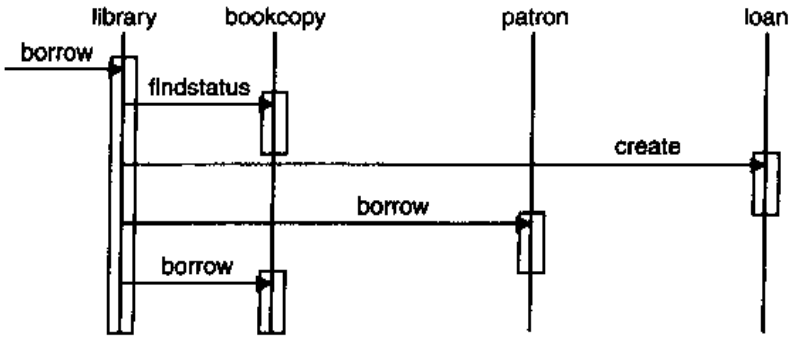


图9-4 library中的borrow函数了解loan类的情况

图9-5的版本未很好地进行抽象。也就是说，较低层类的细节在library的borrow函数中未隐藏。

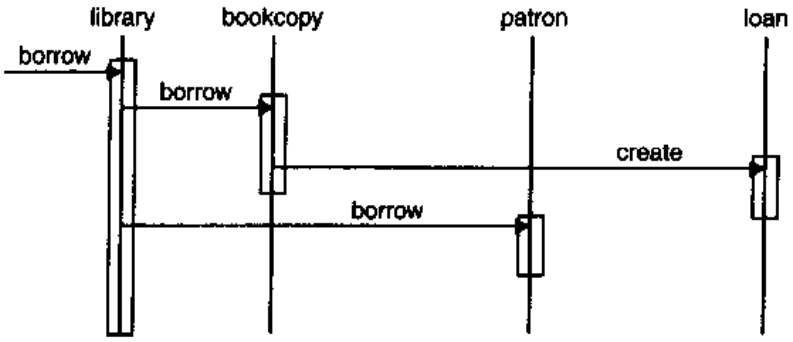


图9-5 借阅交互图——版本1

9.4 度量内聚性

9.4.1 程序片

程序中的变量值依赖于其他变量值。有两种基本依赖性。一种是数据依赖性，即 x 的值通过定义影响 y 的值并成对使用。另一种是控制依赖性，即 x 的值确定包含 y 定义的代码是否执行。

例子9.7

利用重复加计算乘积

以下代码计算 x 和 y 的乘积。输出变量 z 对变量 x 有数据依赖性，因为 x 被加到 z 上。输出变量 z 对变量 y 有控制依赖性，因为 y 控制把 x 加到 z 中的次数。

```
z = 0;
while x > 0 do
      z = z + y;
      x = x -1;
end-while
```

程序片可以从任一角度进行计算。输出片找出影响特定输出的值的每条语句。输入片找出被特定输入值影响的每条语句。

从有向图中更容易计算程序片，在有向图中有一组结点n，每个结点是代码中的一个输入、一个输出或一条语句。弧e为依赖关系。

James Bieman和Linda Ott（James Bieman and Linda Ott, "Measuring Functional Cohesion," *IEEE TOSE*, 20:8 August 1994, 644-657）曾经使用变量定义和引用作为基本单元来代替程序语句。这些定义及引用称为权标（token）。因此，每个常量引用、变量引用和变量定义都是单独的权标。

例子9.8

画出一个有向图说明例子9.7中代码的变量之间的依赖性。用实线表示数据依赖性，用虚线表示控制依赖性。

从图9-6的有向图中，我们可以看到输出片将从唯一的输出 z 开始。语句$z = z + y$和$z = 0$中的权标z、z、y、z和 0可以加入到该片中。接着，加入语句while x>0中的权标 x和0。接下来再加入语句$x = x-1$中的权标x、x和1。这样就穷举了这些语句，程序中的一切内容都在变量z的输出片中。

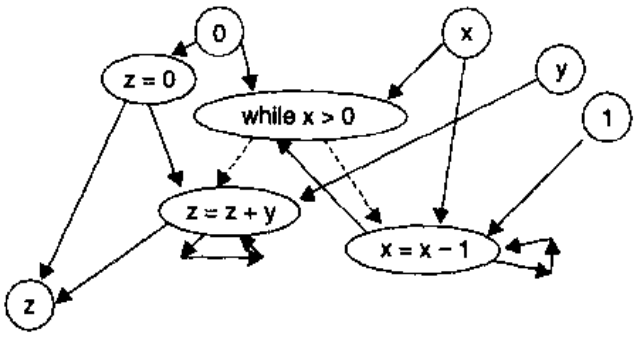


图9-6 例子9.7的有向图

输入片可以从输入变量x开始。语句while x>0和$x=x-1$中的权标x、0、x、x和1加入到该片中。接着，加入语句 $z=z+y$ 中的权标 z、z 和 y。没有其他权标可加入。因此，输入片是除$z=0$以外的一切内容。

变量y的输入片将只包含初始的y 权标和语句 $z=z+y$ 中的权标 z 和 y。

9.4.2 粘合权标

Bieman和Ott还使用输出片定义了某些内聚性度量。该定义一方面基于粘合权标（glue

token），即在多个片中的权标（代码段），另一方面基于超粘合权标（superglue token），即在所有片中的权标。一个权标的粘合率（adhesiveness of a token）是在一个过程中包含该权标的输出片的百分比。

以下是三个功能内聚性度量：

弱功能内聚（Weak Functional Cohesion, WFC）——粘合权标与总权标的比率。
强功能内聚（Strong Functional Cohesion, SFC）——超粘合权标与总权标的比率。
粘合率（Adhesiveness, A）——所有权标的平均粘合率。

例子9.9

计算以下代码段的功能内聚性度量。

```
cin >> a >> b;
int x,y,z;
x=0; y=1; z=1;
if (a > b){
     x = a*b;
     while (10 > a){
       y=y+z;
       a=a+5;
     }
else {
     x=x+b;
}
```

图9-7示出了上述代码中的每个权标。弧从每个权标开始画到受该权标值影响的所有权标上。

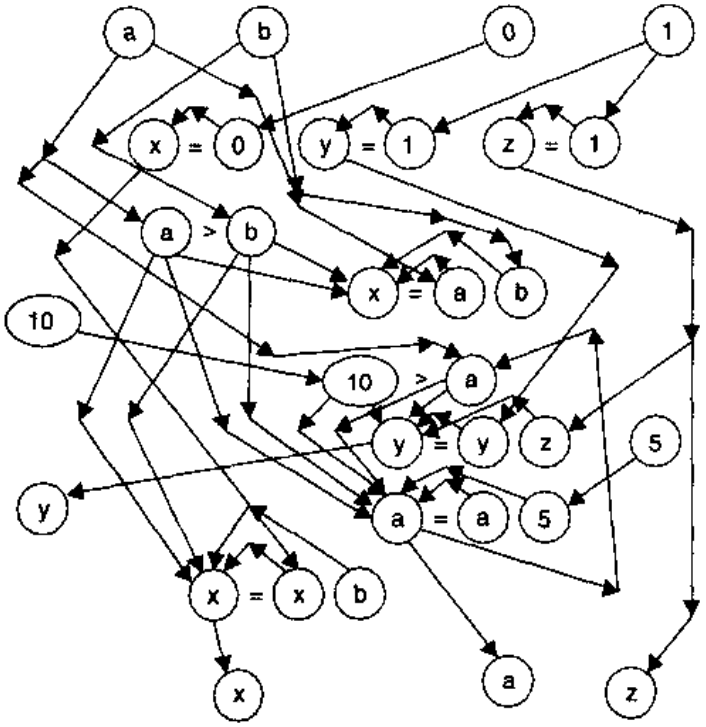


图9-7 显示所有依赖关系的有向图

参见图9-8。

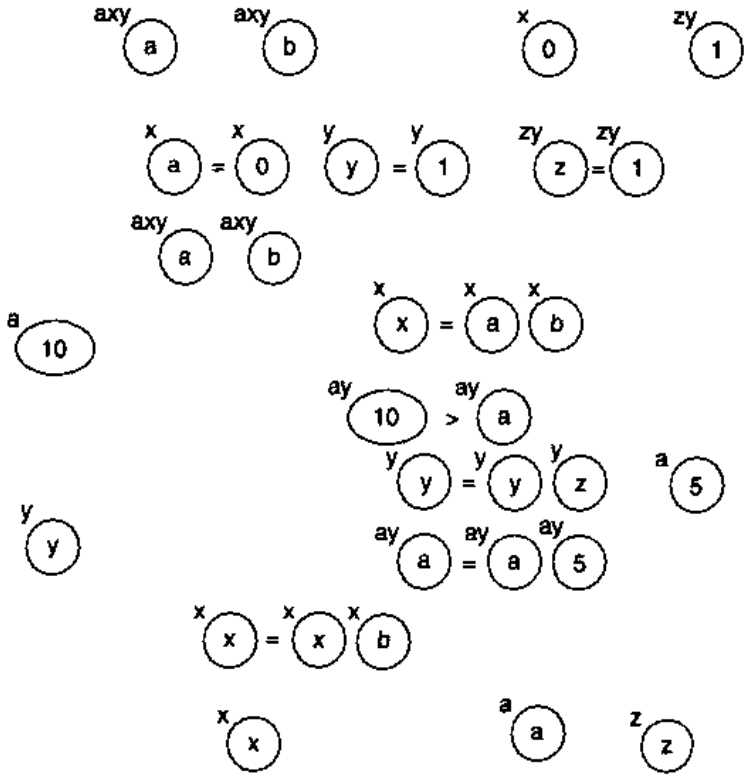


图9-8 带附注的权标显示权标所出现的片

由于没有超粘合权标，因此强功能内聚（SFC）等于0。在31个权标中，有12个粘合权标，因此弱功能内聚为12/31，即0.387。

存在4个片。0权标有100%的粘合率。4个权标在三个片中，因此它们有75%的粘合率。8个权标在两个片中，因此它们有50%的粘合率。剩余的权标有19个，它们都在一个片中，因此它们有25%的粘合率。

粘合率是所有权标的平均粘合率，因此平均粘合率为：

$$(4 \times 0.75 + 8 \times 0.50 + 19 \times 0.25)/ 31 = 11.25/31 = 0.363。$$

9.5 度量耦合性

耦合性（coupling）用来度量两个或多个模块或类联系的紧密程度。耦合性的度量应该指出一个模块的改变对另一个模块有多大的影响。目前已经有许多耦合性度量。

耦合度量的基本形式是建立导致一个模块与另一个模块内部工作相联系的一个项目列表。

Dharma的模块耦合

Dharma（H.Dharma, "Quantitative Models of Cohesion and Coupling in Software," *Journal of Systems and Software*, 29:4, April 1995）建议使用考虑到下列情况的度量：

di = 输入数据参数的数量
ci = 输入控制参数的数量
do = 输出数据参数的数量
co = 输出控制参数的数量
gd = 用于数据的全局变量的数量
gc = 用于控制的全局变量的数量
w = 被调用模块（扇出）的数量
r = 调用该模块（扇入）的模块的数量
Dharma的模块耦合指示器是以上项目乘一个比例常量之和的倒数：

$$mc = k / (di + 2 \times ci + do + 2 \times co + gd + 2 \times gc + w + r)$$

该度量方法有两个困难。一是倒数表示计算的情况数量越多，该模块与其他模块耦合度越大，而mc值则越小。另一个问题是参数和调用计数提出了潜在的问题，即不能确保该模块与其他模块的内部工作相联系。全局变量的使用则确保了该模块与访问同一个全局变量的其他模块相联系。

9.6 需求的可溯性

需求的可溯性试图将每个需求与满足该需求的设计元素相联系。需求应该影响设计。如果一个需求没有相应的设计部分，或一个设计部分没有相应的需求部分，则会存在潜在的问题。当然，有些需求没有反映该需求的特定设计部分，并且有些设计部分太一般以致于不需要需求部分。

检查可溯性的一种方法是画一个矩阵。一个轴列出所有需求项目，另一个轴为设计项的列表。若某个设计项处理某个需求，则在交点处放一个标记。

例子9.10

画一个矩阵说明B&B问题和设计的以下描述中的需求的跟踪。

需求：

Tom和Sue在一个新英格兰小镇上开了一家提供住宿加早餐的旅馆。[1]他们有三间客房。

[2]他们需要一个管理[2.1]预定和监控[2.2]开支和利润的系统。当某个顾客请求[3]预定时，他们将检查[4]日历。如果有空闲房间，他们将输入[6.1]该顾客名、[6.2]地址、[6.3]电话号码、[6.4]日期、[6.5]商定价格、[6.6]信用卡号和[6.7]房间号。预定必须用[7.1]一天的租金[7]做担保。

如果没有对[7.2]商定时间提供担保，则取消预定。如果无该日期的担保，取消预定[7.3]。

设计

```
Class [A] B&B attributes: [A.1] day* daylist[DAYMAX];
   [A.2] reservation* reslist[MAX]; [A.3] transaction*
   translist[TRANSMAX]
   methods: [A.4] display calendar by week
        [A.5] display reservations by customer
        [A.6] display calendar by month
Class [B] day attributes: [B.1] date thisdate
   [B.2] reservation* rooms[NUMBEROFROOMS]
   methods:[B.3] create(), [B.4] addreservation(),
   [B.5] deletereservation()
Class [C] reservation attributes: [C.1]string name
   [C.2] string address [C.3] string creditcardnumber
   [C.4]  date arrival [C.5] date guaranteeby
   [C.6]  int numberofdays [C.7]int roomnumber
   methods: [C.8] create() [C.9] guarantee()
   [C.10] delete()
Class [D] transaction  attributes: [D.1] string name
   [D.2]  date postingdate [D.3] float amount
   [D.4]  string comments
```

	A	A.1	A.2	A.3	A.4	A.5	A.6	B	B.1	B.2	B.3	B.4	B.5	C	C.1	C.2	C.3
1										X							
2	X																
2.1			X							X	X	X	X	X			
2.2				X													
3			X														
4					X	X	X										
5																	
6.1															X		

6.2																X	
6.3																	
6.4																	
6.5																	
6.6																	X
6.7																	
7																	
7.1																	
7.2																	
7.3																	

	C.4	C.5	C.6	C.7	C.8	C.9	C.10	D	D.1	D.2	D.3	D.4
1												
2												
2.1												
2.2								X				
3												
4												
5												
6.1												
6.2												
6.3												
6.4	X		X									
6.5												

6.6												
6.7				X								
7						X						
7.1												
7.2		X										
7.3							X					

在上面的表格中，存在许多空行和空列。需求5与空闲房间相联系。尽管空闲房间应该在预定日期保持空闲，但没有明确的空闲处理。需求6.3是顾客的电话号码，目前也没有。需求6.5是商定价格，预定信息中没有。需求7.1提到了一天的租金，也不在这些属性中。

列A.1是工作日列表，可有助于空闲房间的查询。B和B.1是必需的，但不是某个需求特定的。C.8是一个构造器。D.1~D.4是事务处理的细节，在需求中忽略。

习题

1. 已知以下设计，针对耦合性、内聚性和抽象性思想，说明是希望使用高值还是低值，并说明该设计是如何解释这些内容的。

```
Class college
      student* stulist[MAX]
      course* courselist[MAX]
public:
      addStudent(char* studentname)
      addStudentToCourse(char* studentname, char*
      coursename)
      void displayStudent(char* studentname)
      void displayStudentsInCourse(char* coursename)
Class course
      student* classroll[MAX]
public:
      void displayStudents()

Class student
      char* name
public:
      void displayname()
```

2. 为什么Gunter要限制在规格说明中可以使用的项/事件？用户需求和规格说明之间有何不同？

3. 要研制的系统是基于影像处理的面貌识别系统。该系统有一台照相机，打算通过控制门锁阻止非雇员进入公司的保密设施。当某人试图转动门把手时，系统会拍下图像并将它与现有雇员的图像进行比较。

将以下事件根据其是在环境中还是在系统中以及是隐藏的还是可视的进行分类：

1) 某人试图转动门把手。
2) 门没有被系统上锁。
3) 雇员让非雇员通过该门。
4) 某个雇员有一个双胞胎姊妹。
5) 某个图像与匹配算法有一些类似。

补充问题

1. 画出在音乐商店中一个试图用现金购买某张音乐CD的顾客与店员之间交互的场景。一定要包括所有的可能性。使用用弧表示事件的状态机模型。

2. 计算以下代码段的Bieman和Ott的功能内聚性度量。画出有向图并指出方向。

```
cin >> a >> b;
int x,y,z;
x=0; y=1; z=1;
while (a > 0){
     x = x + b;
     z = z * b;
     if (a > b){
           y=y*a;
     }
     a=a-1;
}
cout << x << a << z << y;
}
```

习题答案

1. 已知以下设计，针对耦合性、内聚性和抽象性思想，说明是希望使用高值还是低值，并说明该设计是如何解释这些内容的。

```
Class college
        student* stulist[MAX]
        course* courselist[MAX]
public:
        addStudent(char* studentname)
        addStudentToCourse(char* studentname, char*
        coursename)
        void displayStudent(char* studentname)
        void displayStudentsInCourse(char* coursename)
Class course
        student* classroll[MAX]
public:
        void displayStudents()

Class student
        char* name
public:
        void displayname()
```

耦合性——期望低耦合性，因为college类中没有其他类的内部构造的内容，并且其他类也不需要了解college，所以该设计有低耦合性。

内聚性——希望高内聚性，由于每个类只涉及自身的属性，因此该设计有高内聚性。

抽象性——该设计显示了良好的抽象性。例如，college的显示方式不包括有关低层显示方法的任何细节。

2. 为什么Gunter要限制在规格说明中可以使用的项/事件？用户需求和规格说明之间有何不同？

Gunter说，用户的需求必须在该用户所知的项/事件中指定并且可以将隐藏到机器中的项/事件包括在内，而规格说明是实现的基础，只能在机器系统和现实世界都可视的项中指定。

3. 要研制的系统是基于影像处理的面貌识别系统。该系统有一台照相机，打算通过控制门锁阻止非雇员进入公司的保密设施。当某人试图转动门把手时，系统会拍下图像并将它与现有雇员的图像进行比较。

将以下事件根据其是在环境中还是在系统中以及是隐藏的还是可视的进行分类：

1) 某人试图转动门把手。　　环境可视的
2) 门没有被系统上锁。　　系统可视的
3) 雇员让非雇员通过该门。　　环境隐藏的
4) 某个雇员有一个双胞胎姊妹。　　环境隐藏的
5) 某个图像与匹配算法有一些类似。　　系统隐藏的

补充问题答案

1. 画出在音乐商店中一个试图用现金购买某张音乐CD的顾客与店员之间交互的场景。一定要包括所有的可能性。使用用弧表示事件的状态机模型。

参见图9-9。

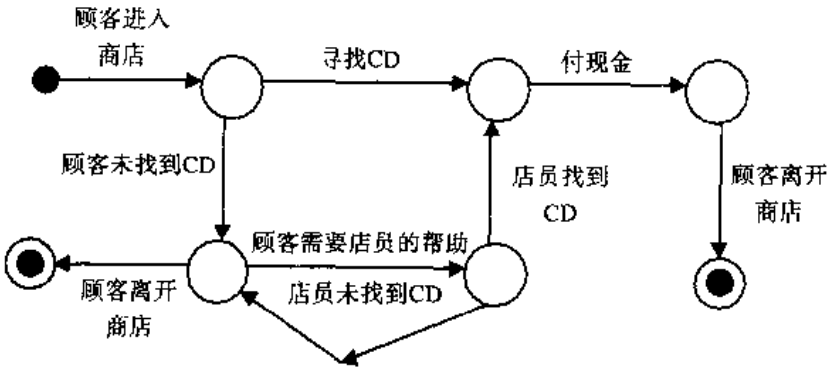


图9-9 有向图

2. 计算以下代码段的Bieman和Ott的功能内聚性度量。画出有向图并指出方向。

```
cin >> a >> b;
int x,y,z;
x=0; y=1; z=1;
while (a > 0){
    x = x + b;
    z = z * b;
    if (a > b){
      y=y*a;
    }
    a=a-1;
}
cout << x << a << z << y;
}
```

参见图9-10。

参见图9-11。

图中有33个权标。4个为超级粘合，6个（包括超级粘合权标）为粘合权标。弱功能内聚（WFC）为6/33 = 18.2%。强功能内聚（SFC）为4/33 = 12.1%。粘合率为(4 × 1 + 2 × 0.75 + 27 × 0.25)/33 = 12.25/33 = 37.1%。

此程序计算了三个单独的量。它在内聚性度量上的得分低毫不奇怪。

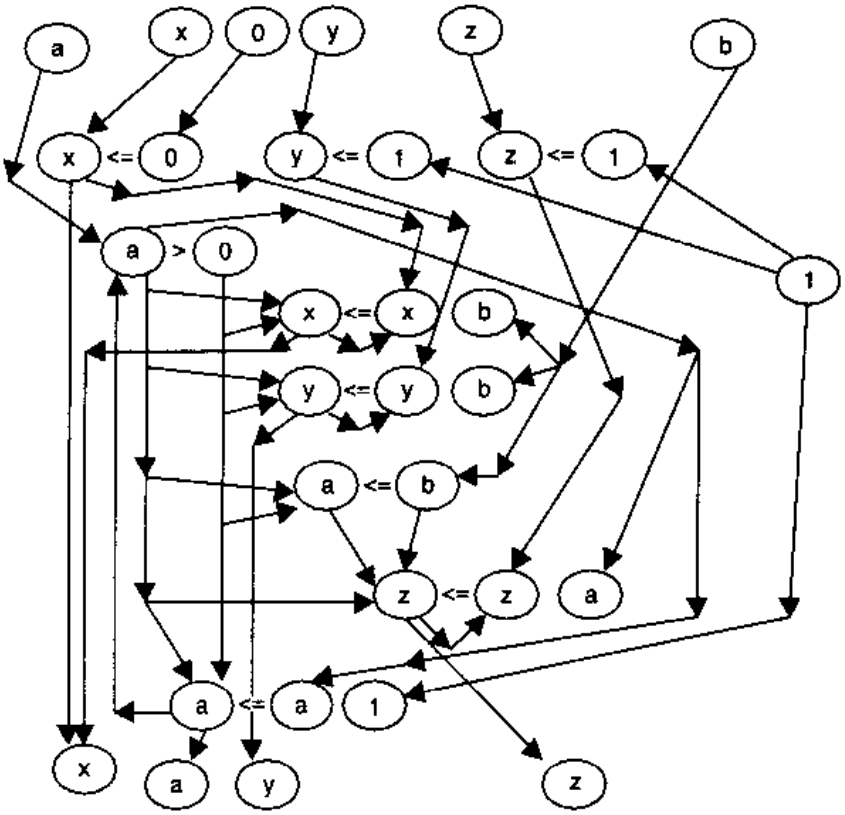


图9-10 控制流程图

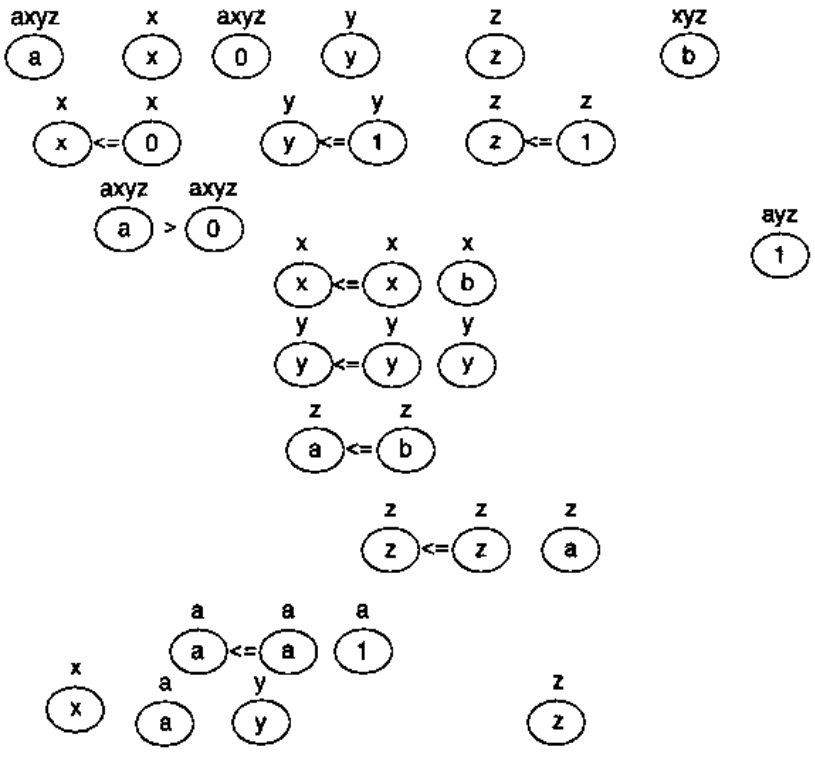


图9-11 控制流程图

第10章 软件测试

10.1 概述

软件测试是用实际的测试数据执行软件。有时也将软件测试称为动态软件测试，以区别于静态分析（静态分析有时也称为静态测试）。静态分析指的是分析源代码以确定是否有问题。虽然也有一些技术可用于测试软件，但用具体的测试数据实际执行软件是最基本的。

10.2 软件测试基础知识

穷举测试会测试每种可能的情况。我们很少进行穷举测试，因为即使是很简单的系统也有许多可能的测试情况。例如，在一台32位字长的机器上，某个程序有两个整数输入，则此程序将具有2^{64}个可能的测试案例（参见习题10.1）。因此，测试总是执行所有可能的测试情况的一小部分。

软件测试中有两个基本的概念，一个是使用什么样的测试案例（测试案例选择）；另一个是有多少测试案例是必须的（停止准则）。测试案例选择可根据规格说明（功能）、代码的结构（结构）、数据流（数据流）以及随机选择测试案例来进行。测试案例的选择可视为一种使测试案例通过输入空间的尝试。在特定的域中有的方面可能特别容易出错，可能需要格外关注。停止准则可基于覆盖准则，如在每个子域中执行n个测试案例，停止准则也可基于行为准则，如一直进行测试直到错误率小于某个阈值x为止。

一个程序可以视为从某个领域空间到答案空间或答案值域的一个映射。给定一个输入，它是领域空间中的一个点，程序生成一个输出，此输出是值域中的一个点。类似地，程序的规格说明也可以看做从领域空间到答案空间的一种映射。

规格说明是软件测试必不可少的项。软件正确性定义为与规格说明映射相同的程序映射。“没有规格说明的程序总是正确的”，这是需要记住的一句名言。一个没有规格说明的程序不能根据规格说明进行测试，程序爱干什么就干什么，所以也就不会违反其规格说明。

测试案例应该包含预期的输出，否则很容易出现看到计算机的输出结果就认为*它正确的*情况。如果预期的输出与实际输出不同，测试人员和/或用户就能够进行判断以确定哪个结果是正确的。

10.3 测试覆盖准则

测试覆盖准则是关于怎样选择测试以及何时停止测试的规则。测试研究的一个基本问题是怎样比较不同的测试覆盖准则的有效性。标准的方法是利用包含关系。

10.3.1 包含

如果满足准则A的任意测试集合也满足准则B，那么就说测试准则A包含测试准则B。这表示，测试覆盖准则A以某种方式包含了测试准则B。例如，如果一个测试覆盖准则要求执行每条语句，而另一个测试覆盖准则要求执行每条语句并进行一些别的测试，那么我们就说第二个准则包含第一个准则。

研究人员已经从大多数常规的准则中得出了包含关系。不过，虽然包含是一个用来比较测试准则的特性，但它并不衡量两个准则的相对有效性。这是因为大多数准则都要说明怎样选择测试案例。选取满足一个准则的最小测试案例集合并不像选择好的测试案例直到满足该准则那样有效。因此，一个满足“较弱”准则的好测试案例集合会比满足“较强”准则但选择得不好的测试案例集合更好。

10.3.2 功能测试

在功能测试中，利用软件规格说明来确定应该测试的子域。前几个步骤之一是为每种不同的程序输出生成一个测试案例。例如，每条错误消息都应该生成。其次，所有特殊的情形都应该具有一个测试案例。对一些棘手的情形也应该进行测试。一些常见的错误和容易引起误解情况也应该进行测试。结果应该是一个实现程序时对其进行彻底测试的测试案例集合。这个测试案例集合也有助于开发人员搞清楚目标软件的某些预期的行为。

Glenford Myers在其经典著作（G. Myers, *The art of Software Testing*, New York: John Wiley, 1979）中提出了下述功能测试问题：给出一个接收三个数a、b、c作为三角形边长并输出三角形的类型的程序，为此程序建立一个良好的测试案例集合。Myers说，据他的经验，多数软件开发人员得不出良好的测试集合。我在软件工程课程中使用这个例子，也得出了相同的结论。有的班级甚至不能在测试集合中给出正确的三角形。

例子10.1

对于这个经典的三角形问题，我们可以将领域空间划分成三个子域，每种不同的三角形为一个子域，它们分别是：不等边三角形（任意两条边都不相等）、等腰三角形（有两条边相

等）和等边三角形（任意两条边都相等）。还可以给出两种错误情形：具有不良输入的一个子域和其中边的长度构不成三角形的子域。此外，因为不给出边的顺序，所以所有组合都应该试一下。最后，每个测试案例都必须给出输出值。

子　域	样例测试案例
不等边三角形	
尺寸渐增	(3, 4, 5 —— 不等边三角形)
尺寸渐减	(5, 4, 3 —— 不等边三角形)
第二个尺寸最大	(4, 5, 3 —— 不等边三角形)
等腰三角形	
a = b且另一边较大	(5, 5, 8 —— 等腰三角形)
a = c且另一边较大	(5, 8, 5 —— 等腰三角形)
b = c且另一边较大	(8, 5, 5 —— 等腰三角形)
a = b且另一边较小	(8, 8, 5 —— 等腰三角形)
a = c且另一边较小	(8, 5, 8 —— 等腰三角形)
b = c且另一边较小	(5, 8, 8 —— 等腰三角形)
等边三角形	
所有边相等	(5, 5, 5 —— 等边三角形)
不是三角形	
第一条边最大	(6, 4, 2 —— 不是三角形)
第二条边最大	(4, 6, 2 —— 不是三角形)
第三条边最大	(1, 2, 3 —— 不是三角形)
不良输入	
一个输入不正确	(-1, 2, 4 —— 不良输入)
两个输入不正确	(3, -2, -5 —— 不良输入)
三个输入不正确	(0, 0, 0 —— 不良输入)

可以增加这个子域清单以区别于其他可能有意义的子域。例如，在不等边三角形子域中，实际上有6种不同的排列，但据程序设计中可能出错的情况来说，最长的边的位置可能最有意义。

请注意，每个子域中一个测试案例通常视为是最小但可接受的测试案例集合。

10.3.3 测试度量

使子域的标识形式化的一种办法是利用可以从规格说明中确定的条件建立一个矩阵，然后系统地确定这些条件的组合为真或假。

例子10.2

三角形问题中的条件为（1）a=b or a=c or b=c;（2）a=b and b=c;（3）a<b + c and b <a + c and c<a + b;（4）a>0 and b>0 and c>0。可将这四个条件作为一个矩阵的行。矩阵的每个列

为一个子域。对于每个子域，条件为真的列放置一个T，条件为假的列放置一个F。所有有效的T和F的组合都将使用。如果有三个条件，则会有$2^3 = 8$个子域（列）。其余的行将用于a, b, c的值以及用于每个子域的预期的输出。

条件	1	2	3	4	5	6	7	8
a = b or a = c or b = c	T	T	T	T	T	F	F	F
a = b and b = c	T	T	F	F	F	F	F	F
a >= b + c or b >= a + c or c >= a + b	T	F	T	T	F	T	T	F
a <= 0 or b <= or c <= 0	T	F	T	F	F	T	F	F
简单的测试案例	0, 0, 0	3, 3, 3	0, 4, 0	3, 8, 3	5, 8, 5	0, 5, 6	3, 4, 8	3, 4, 5
预期的输出	不良输入	等边三角形	不良输入	不是三角形	等腰三角形	不良输入	不是三角形	不等边三角形

可以使用T和F的有效组合。如果有3个条件，那么会有$2^3 = 8$个子域（列）。附加的行可用于a、b、c的值并用于每个子域期望的输出。

10.3.4 结构测试

结构测试是根据源代码的结构进行的测试。最简单的结构测试准则为每条语句覆盖，通常称为C0覆盖（E.F. Miller建立了C0和C1命名系统。他的著作包含了许多别的准则）。

1. C0：每条语句覆盖

这个准则规定，源代码的每条语句都应该由某个测试案例来测试。达到C0覆盖的一般方法是选择测试案例直到某种覆盖工具指出代码中所有语句都已执行完毕为止。

例子10.3

下列伪代码实现了前述三角形问题。矩阵显示了哪些行由哪些测试案例执行。请注意，前三条语句（A、B和C）可视为相同结点的组成部分。

结　点	源代码行	3,4,5	3,5,3	0,1,0	4,4,4
A	read a, b, c	*	*	*	*
B	type = "scalene"	*	*	*	*
C	if (a==b || b==c || a==c)	*	*	*	*
D	type = "isosceles"		*	*	*
E	if (a==b && b==c)	*	*	*	*
F	type = "equilateral"				*
G	if (a>=b+c||b>=a+c||c>=a+b)	*	*	*	*
H	type= "not a triangle"			*	
I	if (a<=0||b<=0||c<=0)	*	*	*	*
J	type= "bad inputs"			*	
K	print type	*	*	*	*

在第四个测试案例后，每条语句都得以执行。这个测试案例集合并不是覆盖每条语句的最小集合。但是，寻找最小的测试集合常常不能找到好的测试集合。

2. C1：每个分支测试

一个更为彻底的测试准则是每个分支测试，该测试也通常称为C1测试覆盖。在这个准则中，目标是经过每个判断的每条分支。

例子10.4

如果把例子10.3的程序，用一个控制流程图（见第2章）表示，那么这个覆盖准则要求覆盖控制流程图中的每条弧。参见图10-1。

弧	3,4,5	3,5,3	0,1,0	4,4,4
ABC-D		*	*	*
ABC-E	*			
D-E		*	*	*
E-F				*
E-G	*	*	*	
F-G				*
G-H			*	
G-I	*	*		*
H-I			*	
I-J			*	
I-K	*	*		*
J-K			*	

3. 每条路径测试

更为彻底的测试是每条路径测试准则。一条路径是利用某个测试案例执行的一个唯一的

程序结点序列。在上述测试矩阵中（例子10.2）有8个子域。这8个子域中每一个子域恰好是一条路径。在该例子中，有16个T和F的不同组合。但是，其中有8个组合是不可行路径。也就是说，对于该问题的条件不存在具有这8种T和F的组合的测试案例。很难确定一条路径是不可行还是难于找到执行该路径的测试案例。

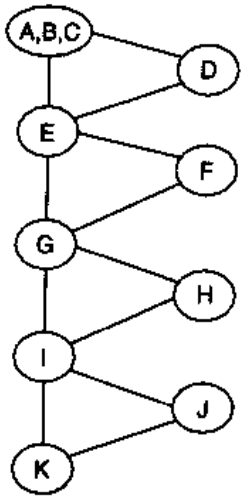


图10-1 例子10.3的控制流程图

大多数具有循环的程序都有无限多条路径。一般而言，使用每条路径测试并不合理。

例子10.5

下面的表给出了例子10.3的三角形伪代码中的8条可行路径。

路 径	T/F	测试案例	输 出
ABCEGIK	FFFF	3,4,5	不等边三角形
ABCEGHIK	FFTF	3,4,8	不是三角形
ABCEGHIJK	FFTT	0,5,6	不良输入
ABCDEGIK	TFFF	5,8,5	等腰三角形
ABCDEGHIK	TFTF	3,8,3	不是三角形
ABCDEGHIJK	TFTT	0,4,0	不良输入
ABCDEFGIK	TTFF	3,3,3	等边三角形
ABCDEFGHIJK	TTTT	0,0,0	不良输入

4. 多条件覆盖

多条件测试准则要求估计每个原始关系条件的真和假。此外，还必须测试一个条件中原始关系的所有T/F的组合。请注意，表达式的惰性判定（一个编译器在不生成不需要的测试代码时称为惰性判定。例如，如果某个“or”表达式的第一个条件为真，则第二个条件就不必测试了）将会排除某些组合。例如，在具有“and”关系的两个表达式中，如果第一个关系为

假，第二个就不进行判定了。

例子10.6

在例子10.3的伪代码中，每个判断语句都有多个条件。由于惰性判定而不执行的条件用“X”来表示，参见以下几个表。

if (a==b || b==c || a==c)

组　合	可能的测试案例	分　支
TXX	3,3,4	ABC-D
FTX	4,3,3	ABC-D
FFT	3.4,3	ABC-D
FFF	3,4,5	ABC-E

if (a==b && b==c)

组　合	可能的测试案例	分　支
TT	3,3,3	E-F
TF	3,3,4	E-G
FX	4,3,3	E-G

if (a>=b + c || b>= a + c || c>= a + b)

组　合	可能的测试案例	分　支
TXX	8,4,3	G-H
FTX	4,8,3	G-H
FFT	4,3,8	G-H
FFF	3,3,3	G-I

if (a<=0 || b<=0 || c<=0)

组　合	可能的测试案例	分　支
TXX	0,4,5	I-J
FTX	4,-2,-2	I-J
FFT	5,4,-3	I-J
FFF	3,3,3	I-K

5. 子域测试

子域测试的思想是将输入域划分为互相排斥的子域，并且要求每个子域中的测试案例的数目相等。这个思想基本上是基于测试矩阵概念的。子域测试在不限制怎样选择子域的情况

下应用更为普遍。一般来说，如果在选择子域时有很好的理由，那么它们在测试时就可能有用。此外，来自其他途径的子域也可以划分为更小的子域。理论研究表明，对子域再进行划分仅在将可能的错误隔离在个别子域中时有效。

每条语句覆盖和每个分支覆盖都不是子域测试。不存在与执行不同语句或分支相关的互相排斥的子域。每条路径覆盖是一种子域覆盖，因为执行特定路径的测试案例的子域与其他任意路径的子域是互相排斥的。

例子10.7

对于三角形问题，我们可能会从每个输入的一个子域开始。如果合适的话，也可以根据最大或不良元素位于第一个位置、第二个位置或第三个位置，将输入进一步划分为新的子域。

子　域	可能的测试案例
等边三角形	3,3,3
等腰三角形——第一	8,5,5
等腰三角形——第二	5,8,5
等腰三角形——第三	5,5,8
不等边三角形——第一	5,4,3
不等边三角形——第二	4,5,3
不等边三角形——第三	3,4,5
不是三角形——第一	8,3,3
不是三角形——第二	3,8,4
不是三角形——第三	4,3,8
不良输入——第一	0,3,4
不良输入——第二	3,0,4
不良输入——第三	3,4,0

6. C1包含C0

例子10.8： C1包含C0

对于三角形问题，我们曾在例子10.3中选择良好的测试案例直到完成C0覆盖为止。这些测试案例为（3,4,5——不等边三角形）、（3,5,3——等腰三角形）、（0,1,0——不良输入）和（4,4,4——等边三角形）。这些测试也覆盖了5个可能输出中的4个。但是，可以用两个测试案例达到C1覆盖，这两个测试案例为（3,4,5——不等边三角形）和（0,0,0——不良输入）。这个测试集合可能没有前一个测试集合好。但是，它达到了C1覆盖，也达到了C0覆盖。

10.4 数据流测试

数据流测试是基于通过程序的数据流进行的测试。数据从定义的地方流向使用的地方。数据定义（或def）将一个值赋予一个变量。变量有两种不同的使用方法。计算使用（或c-use）

表示变量出现在赋值语句右边。c-use出现在赋值语句中。谓词使用（或p-use）表示将变量用于判断语句的条件中。p-use用于判断语句的分支。定义自由路径（或def-free）是一条从变量定义到变量使用的路径，它不包括其他变量的定义。

例子10.9

三角形问题（例子10.3）的控制流程图。

图10-2的控制流程图对变量 a、b、c 的定义和使用作了注释。

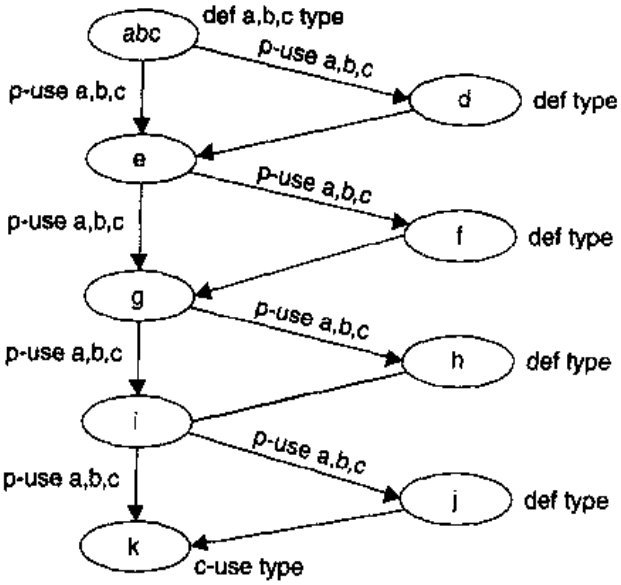


图10-2 三角形问题的控制流程图

有许多数据流测试准则。基本的准则包括dcu。该标准要求从每个定义到一个c-use有一条def-free路径；dpu要求从每个定义到一个p-use有一条def-free路径；du要求从每个定义到每个可能使用有一条def-free路径。测试密度最大的准则是all-du-paths，它要求测试从每个定义到每个可能使用的所有def-free路径。

例子10.10

三角形问题的数据流测试。

dcu——唯一的c-use是对于结点k（输出语句）的变量类型进行的。

从结点abc中的def类型到结点k	路径abc,e,g,i,k
从结点d中的def类型到结点k	路径d,e,g,i,k
从结点f中的def类型到结点k	路径f,g,i,k
从结点h中的def类型到结点k	路径h,i,k
从结点j中的def类型到结点k	路径j,k

dpu——唯一的p-use是对于变量a,b,c进行的，而a,b,c的唯一def在结点abc处。

从结点abc到弧abc-d
从结点abc到弧abc-e
从结点abc到弧e-f
从结点abc到弧e-g
从结点abc到弧g-h
从结点abc到弧g-i
从结点abc到弧i-j
从结点abc到弧i-k

du——所有def到所有用例。
dcu和dpu组合的所有测试案例。

all-du-paths——从所有def到所有使用的def-free路径。
与du测试相同。

10.5 随机测试

随机测试是通过随机选择测试案例来完成的。这种方法速度快，并且还能避开测试人员的偏好。此外，在随机地选择测试案例时，进行统计推断更为容易。通常，测试是随机地从一个操作预置文件（profile）中选择的。

例子10.11

对于三角形问题，我们可以使用随机数生成器，并依次把每三个数作为一个测试组。而且，还要完成确定预期输出的工作。使用随机数生成器的问题之一是生成等边三角形的机会非常小。如果实际发生了这种情形，我们大概会对伪随机数生成器产生疑问了。

10.5.1 操作预置文件

通常，在开发环境中进行测试与在操作环境中执行有很大的不同。使这两者相似的办法是得到某些类型的一个规格说明，以及在常规的操作中遇到这些类型的可能性。这个规格说明称为操作预置文件。通过从操作预置文件中抽出测试案例，测试员将会对测试中程序的行为会预示操作中程序的行为更有信心。

例子10.12

三角形问题的一个可能的操作预置文件如下：

#	描 述	概 率
1	等边三角形	0.20
2	等腰三角形——钝角	0.10
3	等腰三角形——直角	0.20
4	不等边三角形——直角	0.10
5	不等边三角形——全是锐角	0.25
6	不等边三角形——钝角	0.15

为了应用随机测试，测试员可能会生成一个利用概率选择类别的数，然后再生成足够建立测试案例的数。如果所选择的类别为等边三角形，则测试员将为三个输入使用相同的数。等腰三角形——直角的情况下将需要一个随机数作为两腰的长度，然后利用解三角形计算另一条边。

10.5.2 测试的统计推断

如果从一个操作预置文件中随机选择测试案例进行了随机测试，则测试中软件的行为应该与操作环境中软件的行为相同。

例子10.13

如果利用一个操作预置文件随机选择了1000个测试案例，并发现了3个错误，我们可以断定这个软件在操作环境中每执行1000次的错误率小于3。关于使用错误率的详细内容，请参阅3.8节。

10.6 边界测试

通常，错误发生在域之间的边界处。在源代码中，判断语句决定边界。如果一个判断语句写为 x < 1 而不是 x < 0，则边界就改变了。如果一个判断写为 x= <1，则边界 x=1位于真子域中。在边界测试的术语中，我们说on测试位于真子域中，off测试为 x 的值大于1且位于假子域中。

如果一个判断写为 x<1而不是 x<=1，则边界 x=1位于假子域而不是位于真于域中。

边界测试的目的是保证两个子域间的实际边界与指定的边界尽可能接近。因此，测试案例是在边界上和合理地接近边界的边界外选择的。标准的边界测试要在尽可能远的地方进行两个on测试，在接近于边界的中间进行一个off测试。

图10-3示出了一个简单的边界。箭头指出边界的on测试位于边界之下的子域中。两个on

测试位于边界的端点处，而off测试恰好在边界的一半处之上。

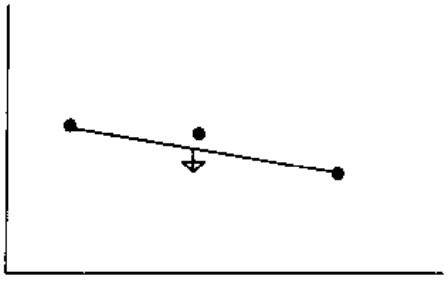


图10-3 边界条件

例子10.14

在三角形的例子中，原始条件a<=b + c or b<= a + c or c<=a + b确定了一个边界。因为这些条件有三个变量，所以这个边界实际上是3D空间中的一个平面。相应的on测试是两个（或多个）具有相等性的单独分开的测试，如（8,1,7）和（8,7,1）。这两个测试结果都为真。off测试应该在另一个域（假域）中，并且应该接近于中间，如（7.9,4,4）。

请注意，关子面向对象（OO）的测试的讨论，请参阅第13章。

习题

1. 软件测试的基本概念是什么？

2. 为了进行测试，为什么需要一个规格说明？

3. 为什么路径测试通常是不切实际的？

4. 路径测试包含语句覆盖吗？

5. 软件测试员有时说“错误发生在边角上”。这表示什么意思？

6. 每条语句覆盖不是一种子域测试准则。其意义是什么？

7. 廉价商店中的销售终端与豪华商店中的销售终端在操作预置文件上有何不同？

8. 软件开发人员可能没有意识到自己的软件测试不彻底。为什么测试覆盖准则有助于改善这种情况？

补充问题

1. 如果一个程序有两个整数输入，每个输入都是一个32位的整数，那么这个程序有多少个可能的输入？

2. 如果一个程序有2^{64}个可能的输入，且每毫秒可进行一个测试，那么对所有可能的输入进行测试需要多长时间？

3. 给定工时数和当前每个工时的工资，工资管理程序将计算出周工资总数。一个工人每周工作不超过80个小时，最高报酬为每小时50美元。建立一个功能测试。

4. 有一个计算三角形面积的程序。其输入为三组x,y坐标。建立一个功能测试。

5. 一个程序接收两个时间（以12小时的格式），输出经过的时间数。建立一个功能测试。

6. 一个折半查找程序搜索按字母顺序排列的名字列表，如果查找的名字在列表中，返回真，否则返回假。建立一个功能测试。

7. 对补充问题10.3中的工资管理程序，确定其条件并构造测试矩阵。

8. 对补充问题10.4中的三角形面积的计算，确定其条件并构造测试矩阵。

9. 对补充问题10.5中经过时间的计算器，确定其条件并构造测试矩阵。

10. 对补充问题10.6中的折半查找程序，确定其条件并构造测试矩阵。

11. 对例子10.3中的三角形问题的伪代码，找出达到C0和C1覆盖的最小测试案例集合。

12. 如果经过的时间小于24小时，下列伪代码实现补充问题10.5中经过时间的问题。选择测试案例直至达到每条语句覆盖为止。选择别的测试案例以达到每个分支覆盖。

```
read hr1 min1 AmOrPm1
read hr2 min2 AmOrPm2
if (hr1 == 12)
      hr1 = 0
if (hr2 == 12)
      hr2 = 0
if (AmOrPm1 == pm)
      hr1 = hr1 + 12
if (AmOrPm2 == pm)
      hr2 = hr2 + 12
```

```
if ( min2 < min1)
        min2 = min2 + 1
        hr2 = hr2 - 1
if( hr2 < hr1)
        hr2 = hr2 + 24
elapsed = min2 - min1 + 60* (hr2 - hr1)
print elapsed
```

13. 对补充问题10.12中的伪代码，找出一个达到C0的测试案例的最小集合并找出一个达到C1的测试案例的最小集合。

14. 对下列代码，确定所有可行路径、路径测试和数据流测试：

```
cin >> a >> b >> c; // node A
x = 5; y = 7;
if ( a > b && b > c) {
        a = a + 1; // node B
        x = x + 6;
        if ( a = 10 || b > 20) {
                b = b + 1; // node C
                x = y + 4;
                }
        if (a < 10 || c = 20) { // node D
                b = b + 2; // node E
                y = 4
                }
        a = a + b + 1; // node F
        y = x + y;
        }
if (a > 5 || c < 10) { // node G
        b = c + 5; // node H
        x = x + 1;
        }
cout >> x >> y; // node I
```

15. 给出下列代码，画出其CFG，并分别生成准则C0、C1、dpu和dcu的一个最小测试案例集合：

```
cin>> a >> b // node A
if (b>a) {
        x = b; // node B
        if (b>20) {
                x = x + 9; // node C
        }
        else {
                x = x + 1; // node D
        }
```

```
        x = x + 1; // node E
}
else {
        x = a // node F
        if (a > 20_ {
                x = x + 15; // node G
        }
        x = x - 5; // node H
}
if (b > a + 20) // node I
{
        x = 20; // node J
}
cout << x; // node K
```

习题答案

1. 软件测试的基本概念是什么?

基本概念是怎样选择测试案例以及何时终止测试。

2. 为了进行测试，为什么需要一个规格说明?

规格说明用来决定程序的实际行为何时正确，何时不正确。

3. 为什么路径测试通常是不切实际的?

多数程序都有通过程序的无数条路径。

4. 路径测试包含语句覆盖吗?

是的，每条语句都位于某条路径上。所以，覆盖了每条路径将覆盖每条语句。

5. 软件测试员有时说“错误发生在边角上”。这表示什么意思?

错误在边界上更容易出现。也就是说，源代码中的故障常常影响到某个判断，从而在边界上产生一个错误。

6. 每条语句覆盖不是一种子域测试准则。其意义是什么?

在使用一个子域测试准则时，很容易通过从每个子域中选取多个测试案例来改进覆盖，这样也容易分析。但是，对于每条语句覆盖来说，情况不是这样。

7. 廉价商店中的销售终端与豪华商店中的销售终端在操作预置文件上有何不同？

在廉价商店中，标记为小于10美元的物品更多。而在豪华商店中，高价物品更多。豪华商店中的物品价格或许还要上舍入到更高的价格。

8. 软件开发人员可能没有意识到自己的软件测试不彻底。为什么测试覆盖准则有助于改善这种情况？

测试覆盖准则有助于促使测试人员测试软件的许多不同部分。

补充问题答案

1. 如果一个程序有两个整数输入，每个输入都是一个32位的整数，那么这个程序有多少个可能的输入？

每个32位的整数具有2^{32}个可能的值。因此，具有两个整数输入的程序应该具有2^{64}个可能的输入。

2. 如果一个程序有2^{64}个可能的输入，且每毫秒可进行一个测试，那么对所有可能的输入进行测试需要多长时间？

每秒可进行10^6个测试，即每天可进行8.64×10^{10}个测试。这等于每年可进行3.139×10^{13}个测试。因为$2^{10} = 1024$大约等于$1000 = 10^3$，所以$2^{64}=(2^{10})^{6.4}$大约等于$(10^3)^{6.4} = 10^{19.2}$。将$10^{19.2}$除以10^{13}得出一个大于10^5的值，所以做完所有测试将至少需要10^5年。

3. 给定工时数和当前每个工时的工资，工资管理程序将计算出周工资总数。一个工人每周工作不超过80个小时，最高报酬为每小时50美元。建立一个功能测试。

此功能测试应该包含常规工资和加班工资的测试，包括错误条件的测试。

工 时 数	报 酬	预期的输出
30	40.00	1200.00
60	50.00	3500.00（假定有加班）
81	50.00	工时数无效
20	60.00	报酬无效

4. 有一个计算三角形面积的程序。其输入为三组x,y坐标。建立一个功能测试。

此功能测试应该包括正确的三角形、非三角形、错误条件以及明显的三角形的定向。

点1	点2	点3	预期的面积
1,1	1,5	5,1	8
1,1	1,5	1,10	不是一个三角形
10,10	0,10	10,0	50
0,0	0,10	10,10	50
0,0	0,0	0,0	0

5. 一个程序接收两个时间（以12小时的格式），输出经过的时间数。建立一个功能测试。

此功能测试应该包括经过的时间小于1个小时、大于1个小时的测试，包括一个经过时间大于12小时的测试和一个需要时间进位的测试，最后还要包括时间颠倒的测试。

开始时间	停止时间	经过的时间
10:00 a.m.	10:40 a.m.	0:40
9:57 p.m.	11:40 p.m.	1:43
3:00 a.m.	9:15 p.m.	18:15
1:50 a.m.	3:40 a.m.	1:50
3:00 a.m.	7:24 a.m.	4:24
5:00 p.m.	4:00 a.m.	错误

6. 一个折半查找程序搜索按字母顺序排列的名字列表，如果查找的名字在列表中，返回真，否则返回假。建立一个功能测试。

此功能测试应该包括下列测试：
列表中第一个名字
列表中最后一个名字
第一个名字后的名字
最后一个名字前的名字
一个位于列表中间的名字
不在列表中且恰好在第一个名字后的名字
不在列表中且恰好在最后一个名字前的名字

7. 对补充问题10.3中的工资管理程序，确定其条件并构造测试矩阵。

条件						
0<小时<=40	T	F	F	F	T	F
40<小时<=80	F	T	F	T	F	F
0<报酬<=50	T	T	T	F	F	F
小时	30	50	90	50	30	-5
报酬	50	30	30	60	70	-5
预期的输出	1500	1650	错误	错误	错误	错误

8. 对补充问题10.4中的三角形面积的计算，确定其条件并构造测试矩阵。

条件		
点在同一条线上	F	T
点1	10,0	0,0
点2	10,10	0,5
点3	0,0	0,10
预期的面积	50	错误

9. 对补充问题10.5中经过时间的计算器，确定其条件并构造测试矩阵。

没有对时间指定的条件。唯一的条件是时间必须是合法的。

10. 对补充问题10.6中的折半查找程序，确定其条件并构造测试矩阵。

没有为搜索指定的条件。

11. 对例子10.3中的三角形问题的伪代码，找出达到C0和C1覆盖的最小测试案例集合。

C0可以用三个小于或等于零的相同的值来达到，即0,0,0。

C1可用两个测试案例来达到，分别是0,0,0和一个不等边三角形（即3,4,5）。

12. 如果经过的时间小于24小时，下列伪代码实现补充问题10.5中经过时间的问题。选择测试案例直至达到每条语句覆盖为止。选择别的测试案例以达到每个分支覆盖。

```
read hr1 min1 AmOrPm1
read hr2 min2 AmOrPm2
if (hr1 == 12)
      hr1 = 0
if (hr2 == 12)
      hr2 = 0
if (AmOrPm1 == pm)
      hr1 = hr1 + 12
if (AmOrPm2 == pm)
      hr2 = hr2 + 12
```

```
if ( min2 < min1)
      min2 = min2 + 1
      hr2 = hr2 - 1
if( hr2 < hr1)
      hr2 = hr2 + 24
elapsed = min2 - min1 + 60* (hr2 - hr1)
print elapsed
```

C0：开始时间	停止时间	预期经过时间
12:00 p.m.	12:40 p.m.	0:40
9:57 p.m.	11:40 p.m.	1:43
5:00 p.m.	4:00 a.m.	12:00

C1：开始时间	停止时间	预期经过时间
上面的测试案例加上后面的这个案例		
8:00 a.m.	12:40 p.m.	4:40

13. 对补充问题10.12中的伪代码，找出一个达到C0的测试案例的最小集合并找出一个达到C1的测试案例的最小集合。

C0的最小测试集至少需要两个测试案例。hr1必须在一个测试中为12，hr2必须在一个测试中为12，hr1必须在一个测试中为p.m.，hr2必须在一个测试中为p.m.，min2必须在一个测试中小于min1，hr2必须在一个测试中小于hr1。这可以在两个测试案例上完成。

最小C0：开始时间	停止时间	预期经过时间
12:00 p.m.	10:40 a.m.	22:40
9:57 a.m.	12:40 p.m.	2:43

这也达到了C1覆盖，因为在这些测试案例上，这些条件每个都为假。

14. 下面是可行路径、路径测试和数据流测试：

路　径	真	a,b,c
AGI	FxxF	4,8,12
AGHI	FxxT	4,8,8
ABDFGI	TFFF	不可行
ABDFGHI	TFFT	12,8,6
ABCDGI	TTFF	不可行
ABCDGHI	TTFT	24,22,8
ABCDEFGI	TTTF	不可行
ABCDEFGHI	TTTT	24,22,20
ABDEFGI	TFTF	不可行
ABDEFGHI	TFTT	6,4,2

结点	def	c-use	puse
A	a,b,c,x,y		
ab,ag			a,b,c
B	a,x	a,x	
bc,bd			a,b
C	b,x	b,y	
D			
de,df			a,c
E	b,y	b	
F	a,y	a,b,x,y	
G			
gh,gi			a,c
H	b,x	c,x	
I		x,y	

见图10-4。

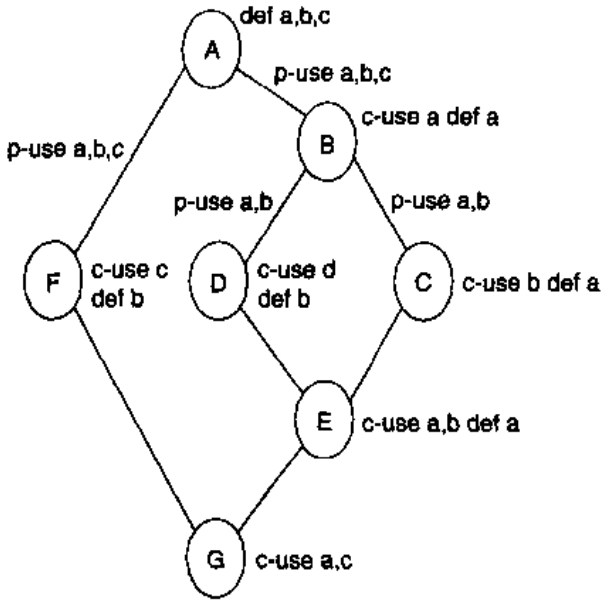


图10-4 10.12的图形表示

15. 下面为相应的路径、CFG和准则C0、C1、dpu和dcu的最小测试案例集合：

路径（可行路径编号）：

1. abceik TTF
2. abceijk TTT
3. abdeik TFF

4. abdeijk TTT
5. afhik FTF
afhijk 不可行
6. afghik FFF
afghijk 不可行
最小测试案例集合：

C0： 输入 输出
2. abccijk 10,30 20
3. abdeik 10,20 22
6. afghik 20,15 30
（必须包括6，但可以是1和4）
C1： 输入 输出
C0测试加上

6. afghik 20,15 30
（必须包括6，但可以为路径1和4）

dpu：唯一的p-usc是针对变量a和b的。a和b的唯一def位于结点A中。最小测试集合等于C1。它必须包括部分路径AB、AF、BC、BD、FH、FG、IJ和IK。这可以用路径1、4、5、6或路径2、3、5、6来完成。

dcu：变量x的def位于结点B、C、D、E、F、G、H和J中。变量x的c-use位于结点C、D、E、G、H和K中。变量a和b的def位于结点A，变量b的c-use位于结点B，变量a的c-use位于结点F。最小测试集必须包括部分路径BC或BD、CE、DE、FG或F..K、GH、H..K或H..K、JK、AB和AF。这可以通过任何C1测试集来完成。

见图10-5。

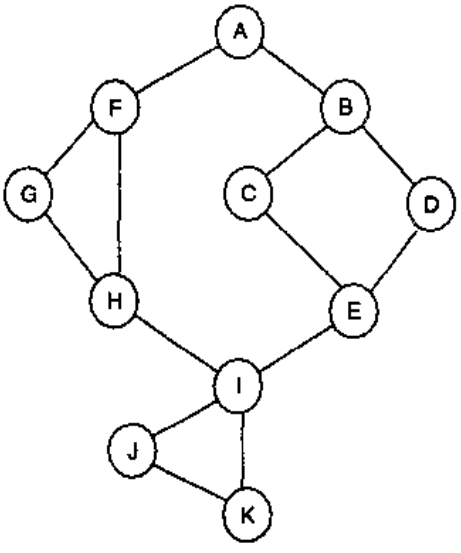


图10-5 10.15题的CFG

第11章　面向对象的软件开发

11.1　概述

面向对象的软件不同于传统的软件。面向对象的软件开发具有许多好处，如需求、设计和实现的简化。这些好处是通过用代表实现实体的对象来为问题领域建模、通过封装函数与数据、通过在项目内和项目间重用对象、通过得到更智能化地接近问题的解决方案而获得的（智能距离是一个用来描述两种思想之间的接近程度的术语。这里指的是现实世界问题的结构与解决方案的结构的接近程度）。

统一建模语言（UML）是面向对象的模型的标准表示法。UML的规范可在Web上找到（登录www.omg.org或搜索关键字“UML”）。

11.1.1　继承

面向对象的软件中，一个创新性的思想是继承。继承来自于对思想和概念的层次性的认识，这些层次/分类涉及的固有重用思想来源于从较高层的概念到较低层的特化。

如果通过使一组实体为另一组实体的特化来关联两组实体，就有可能存在继承关系。在一个继承关系中，基类（较一般的类）将包含所有公共的属性。派生类（较特殊化的类）将继承基类的所有公共的属性。

例如，如果一组实体由车辆组成，而另一组实体由小汽车组成，这时可以使用继承。小汽车可以从车辆进行继承。小汽车可以共享车辆类的许多属性和操作。所有公共属性都可以位于基类中。派生类将自动继承这些属性。这样可以减少工作量。

例子11.1

绘出一个标识小汽车和车辆之间所有共性的对象模型。

图11-1示出了小汽车和车辆，两者都具有车身、引擎、轮子（可能不是四个）、前灯、商标、厂商和价格。（可能还有许多共同的东西。）

例子11.2

绘一个给出处理书籍、杂志、小册子、电影录像带、音乐CD、录音图书磁带和报纸的图书馆系统的共性的对象模型。

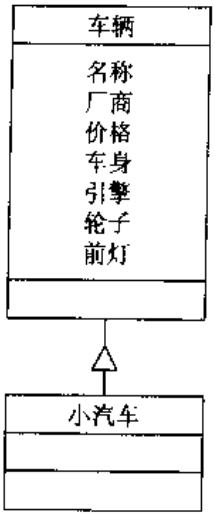


图11-1 小汽车和车辆的共性模型

图11-2示出这些物品都有书名、出版者、获得日期、目录编号、书架位置、借出状态和登记借出限制。

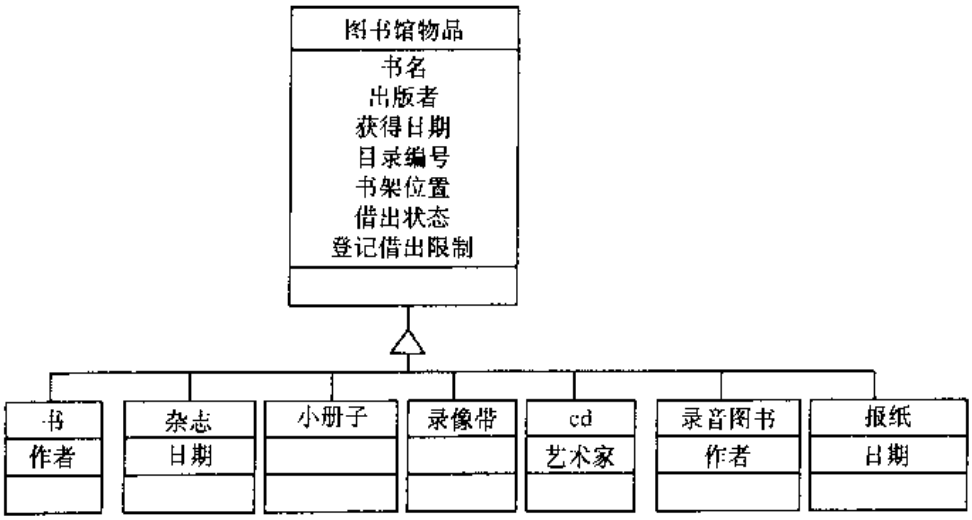


图11-2 图书馆系统共性的对象模型

有的样例属性已经加到某些派生类中。请注意“录音图书”可以从“书”中派生，“报纸”和“杂志”可从名为“连载”或“期刊”的对象中派生出来。

11.1.2 多态性

多态性表示“能够呈现许多形式”。它指可以处理不同版本或形式的对象或参数表的函数。

在面向对象的软件中，这通常表示一个函数可以处理基本类型或派生类型。在小汽车/车辆的例子中，基类具有所有车辆都可以执行的任务的多态函数，而且可能是不同的执行，如转弯等。每个派生类可以使用基类函数也可以提供适合于该派生类的一个版本。

例子11.3

找出图书馆问题（例子11.2）中所有物品的公共函数和对派生类必须专门指定的函数。

公共函数：借出和归还登记函数（登记借出限制除外）。

专门指定的函数：编目函数。

11.2 确定对象

确定建议系统的需求的一个方法是从确定问题领域中的对象开始。这些对象在问题陈述中一般为名词。

11.2.1 noun-in-text方法

在noun-in-text（文本中的名词）方法中，要标出文本中的所有名词。不同的名词可能会用于相同的概念。这些等同的名词和与每个概念相关的名词应该分组存放。有的名词与建议的系统外部的环境相关，可以删除。

在每个名词组中，应该选择代表对象的名词。组中其他名词或者成为属性或者抛弃。

例子11.4

使用noun-in-text描述方法确定下列杂货店问题中的对象：

一个杂货店想使其库存管理自动化。这个杂货店具有能够记录顾客购买的所有物品和数量的销售终端。顾客服务台也有类似的终端，以处理顾客的退货。它在码头有另一个终端处理供应商发货。肉食部和农产品部有终端用于输入由于损耗导致的损失/折扣。

名词：

杂货店、库存、销售终端、物品、数量、顾客、购买的货物、服务台、退货、码头、发货、供应商、肉食部、农产品部、损失、折扣。

名词组：

杂货店

库存、物品、数量、退货、损失、折扣

发货

供应商

肉食部、农产品部

顾客

系统外部的环境实体：

销售终端、服务台、码头、肉食部、农产品部

但是，应该包括肉食品和农产品以反映不同的处理。

对于是使顾客处于系统外部还是使系统知道和跟踪顾客这一问题需要进行选择。所作出的选择是跟踪顾客。

对象和属性的最终列表如下：

grocery store（杂货店）
inventory（库存）
items with an attribute of quantity（物品，具有属性"数量"）
customer（顾客）
purchases（购买的货物）
returns（退货）
shipments（发货）
suppliers（供应商）
losses（损失）
discounts（折扣）
meat items（肉食品）
produce items（农产品）

见图11-3。

例子11.5

利用noun-in-text描述方法确定下列家族树问题中的对象：

Fred正在研究家族谱系，想建立一个存储所找到的自己家族信息的程序。他出生于一个大家庭，有很多伯父、叔父、堂兄弟姊妹。

这个问题不容易利用noun-in-text方法进行解决。第一个句子为动机，相应的只有一个名词"家族"。第二个句子重复了名词"家族"，然后列出了人员间关系的名词。与前一个例子

不同，这些关系不是派生类。叔父不是人员的特化，它是人员间的一种关系。

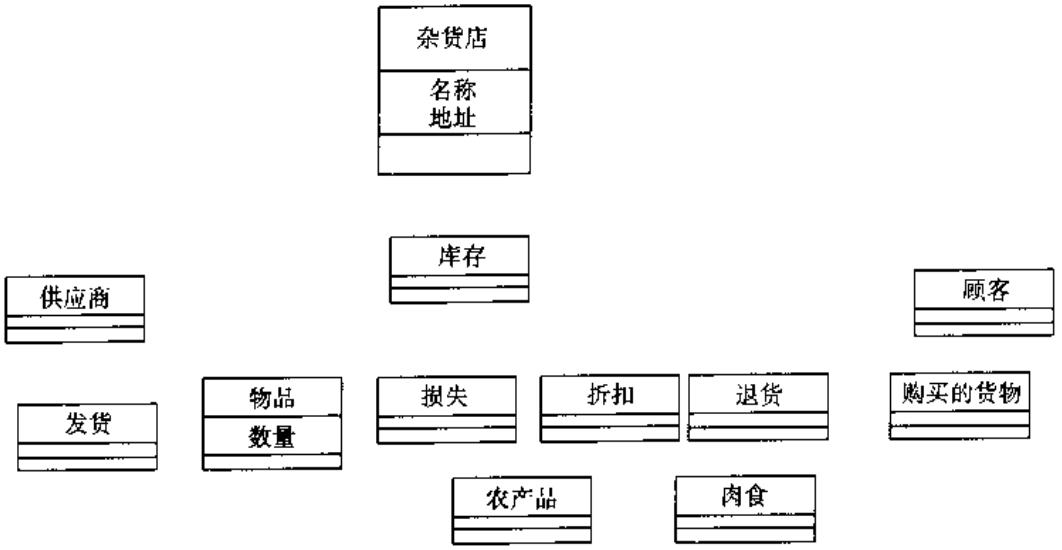


图11-3 例11.4的图形表示

要确定对象，必须熟悉问题领域。这个问题的恰当的对象集合为家族树、人员、家族。请参阅图11-4。

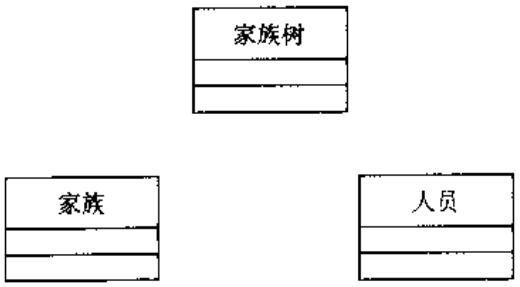


图11-4 家族谱系模型

11.2.2 确定继承

继承是“a-kind-of”（一种）关系。基类为公共的对象，派生类为公共对象的特殊实例。自顶向下的方法要确定有时要进行特殊的处理以及具有特殊的属性的对象。这种方法一般是找出继承关系的一种有效的方法。

有时也采用相反的方法，即自底向上的方法，它对类似的项进行分组，然后寻找共性。所有类似项的交集构成基类。

例子11.6

确定杂货店（例子11.4）的可能继承。

自顶向下的方法将有助于了解肉食部和农产品部具有特殊的物品处理。这将得出一个物品的基类以及肉食和农产品的派生类。杂货店领域内的专家有助于确定能够在杂货店内出现的其他派生类。

此外，自底向上的方法会发现“损失”、“折扣”、“退货”、“购买的货物”等对象的共性。这表示这些对象可以从对象“交易”中派生。请参阅图11-5。

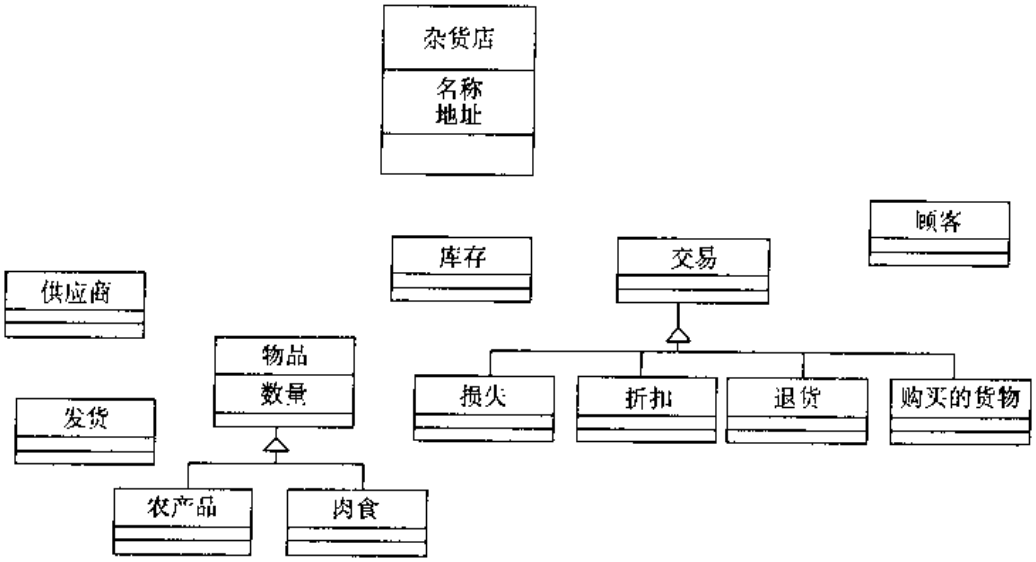


图11-5 杂货店继承的模型

11.2.3 确定重用

重用是面向对象的软件的一种承诺。但是，重用很少自己发生。确定重用的第一个步骤是执行一个名为领域分析的任务。领域分析要审查问题领域以确定什么样的对象和功能对所有活动来说是公共的。没有很好的领域知识，就很难确定领域内所有类似系统之间存在什么样的共性。为使重用起作用，必须确定相应领域内多个解决方案之间有用的部分。这表示要理解潜在的共性。

重用的途径可以是自顶向下，也可以是自底向上的。自底向上的方法寻找领域中大多数解决方案的共同的中低层的类。自顶向下方法寻找解决方案框架中的共性以及在低层对象中的差异。

除非重用是一个目标，并且确定潜在的重用和设计可重用的类很重要，才需要对重用进行着重研究，因为实现重用是比较难的。

例子11.7

确定杂货店领域（例子11.4和11.6）中的重用。

在杂货店领域中，似乎共同性位于低层对象中。大多数杂货店都处理相同种类的物品。有的中层活动也具有许多共性，例如，库存管理系统、进货、跟踪销售情况等。

11.2.4 用例方法

确定系统需求的另一种方法是从确定场景开始。这种方法将确定必须的功能以及支持这些功能所需的对象的活动视为最佳途径。

例子11.8

建立杂货店问题（例子11.7）的场景。从所建立的场景列出对象清单。

大多数场景将基于一般的领域知识，而不是仅从简短的问题陈述中派生而来的。

场景1：库存量现在比较低；给供应商送了一份订单；所订购的货物已到达货运码头；将物品和数量输入库存管理系统。

场景2：顾客购买了货物并付了账；更新顾客数据库（所做的决定还是让系统跟踪顾客）。

场景3：一个新顾客进入商店，要求他填写新顾客信息表单并领取会员卡。

场景4：农产品部的员工检查农产品，并把坏了的蒿苣扔了；更新库存。

从这些场景可以很方便地确定出下面的对象：

库存、供应商、订单、发货、物品、顾客、会员卡、农产品

杂货物品（基类）和农产品（派生类）之间的关系很清楚。

11.3 确定关联

在确定了一个领域中的对象后，下一步就是确定对象间的关联了。关联表示两个对象间的一种关系。2.4节介绍了不同的关联。对象间的一个关联表示在系统实现中，对象间存在一个链接。因此，关联的重要性在于确定了一个对象必须对另一个对象进行什么样的访问。这种访问对于功能的有效实现是非常重要的。

确定对象间关联的方法有多种。一种方法是确定问题领域中存在的关联。

例子11.9

建立例子11.5的家族树问题的关联。

问题的陈述中提到了叔父、伯父和堂兄弟姊妹。这些全都是关联（关系）。没有原始的关联。谱系中基本的关联为母亲、父亲、孩子。这些关联的逆向关联分别为婚姻、婚姻、出生家族。

此外，还有一个从顶层对象“家族树”到“家族”和到“人员”的关联（聚集）。这些关联可分别称为“婚姻”和“人员”，参见图11-6。

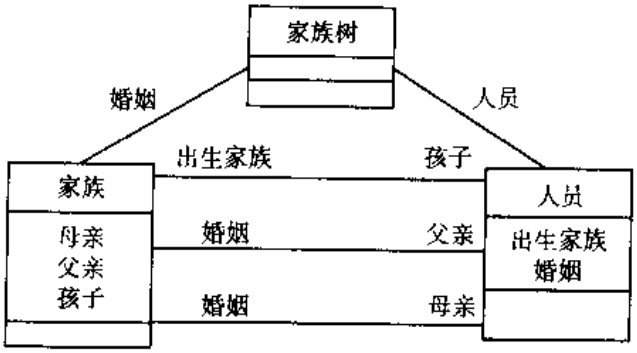


图11-6 家族树的关联模型

确定关联的另一方法是考虑所需功能。如果要求一个对象具有访问其他对象的功能，则在这些对象之间必须存在一个关联或关联序列。

例子11.10

某个学院需要一个能够管理课程、教学区、学生的系统。绘出对象模型并确定对象间的关联。

此学院需要访问学生以便打印出学生信息。为了打印出学生学习的课程，需要进行从学生到教学区的访问。为了打印出可以使用的教学区的安排表，需要访问课程，然后再访问每个课程的教学区。参见图11-7。

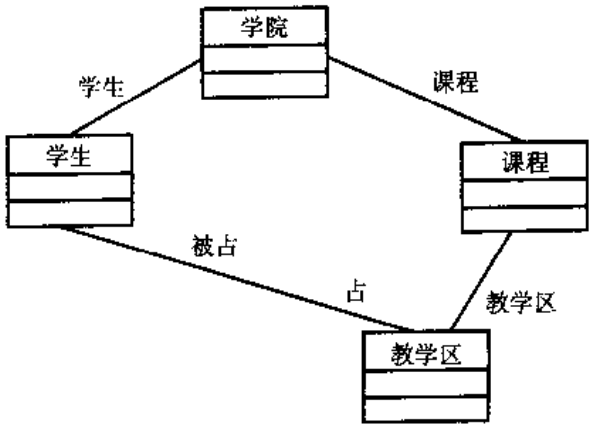


图11-7 该学院管理系统的关联模型

存在依赖

另一方法是利用对象之间的存在依赖（2.4.1节）关系确定所需的关联。如果子对象的存在恰好依赖于父对象的一个实例，则这两个对象有一个存在依赖关系。这表示父实例存在子所建立的子实例之前，而子实例在父实例被删除之前删除。

例子11.11

使用存在依赖构造例子2.6的图书馆样例中的关联。

“书”或“人员”都不对“图书馆”存在依赖。但是，“书”或“人员”分别以“读者”和“副本”项参与图书馆的活动，这确实满足存在依赖要求。参见图11-8。

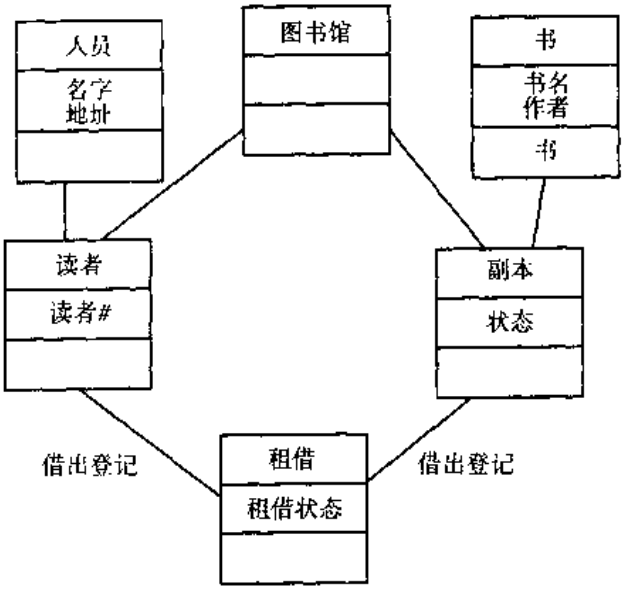


图11-8 图书馆问题的存在依赖模型

例子11.12

利用存在依赖确定例子11.10的学生教学区问题中的关联。

例子11.10中建立的对象模型并不满足存在依赖规则，因为“教学区”并不对“学生”存在依赖，反之亦然。因此，必须使用一个名为“登记”的对象。请参阅图11-9。

11.4 确定多重性

多重性是对象实例间的关联上的约束。多重性是由关联末端处的一个表达式指定的。这个表达式可以是单个值、值的范围、范围列表或单个值的列表。在范围中，两个值用两个句

点分隔。

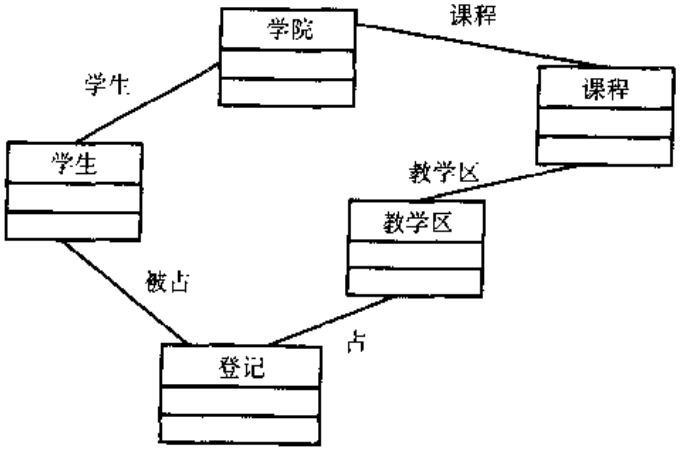


图11-9 教学区问题的存在依赖

一个对象的实例与其他对象的实例有多少关系？问题领域常常对这个问题有约束。

例子11.13

利用多重性限制一本书的副本在给定时间内可以借出多少次。

如图11-10所示，相应关联的租借端处的0..1限制了一个副本一次最多参与一个租借关系。关联的副本端处的1要求一次租借必须与一个副本恰好有一个关联，即不可能存在这样一个租借，它不具有与自己关联的一个副本。

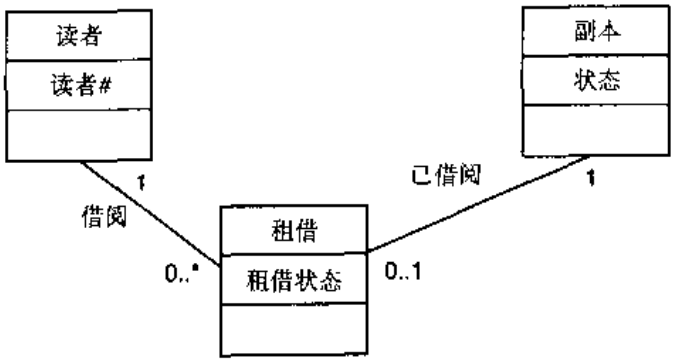


图11-10 借书问题的多重性

借阅关联限制租借实例恰好与一个读者关联。读者可以与零个或多个租借有关联。

例子11.14

确定例子11.12的学生－教学区问题的多重性。

所有实例都必须恰好关联到一个父实例。所有双亲必须关联到1到n个子实例。例如，0..*表示可以有零个或多个关系。而且，每个课程必须与一个学院关联。

见图11-11。

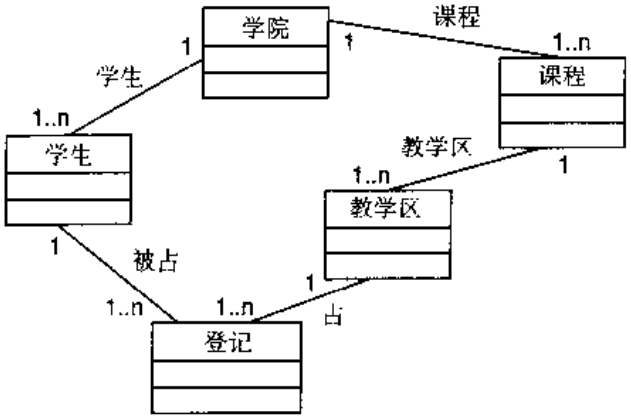


图11-11 学生－教学区问题的多重性

习题

1. 为什么说Ford汽车是小汽车的特化而引擎不是小汽车的特化？

2. 对象和属性之间的区别是什么？

3. 领域分析的目标是什么？

补充问题

1. 确定下列B&B问题陈述的对象：

Tom和Sue在新英格兰的一个小镇上开了一个住宿加早餐的客栈。他们有三间客房，并需要一个系统来管理房间预订并监控开支和利润。在某个顾客打电话预订住宿时，他们要查看日历。如果有空位，他们要输入顾客的名字、地址、电话号码、日期、议定的价格、信用卡号以及房间号。预订必须交一天的保证金。

预订在没有保证金的情况下将保存议定的时间。如果到该日期还没有交保证金，将取消预定。

2. 确定下列牙科诊所问题陈述的对象：

Tom在小镇上开了一个牙科诊所。他有一个牙科助手、一个牙科保健员和一个接待员。Tom需要一个系统来管理预约。

在一个病人打电话预约时，接待员将查阅日历并安排病人尽早得到诊治。如果病人对计划的约定时间感到满意，接待员将输入约定时间和病人的名字。系统将核实病人的名字并提供病人的记录数据，这些记录数据包括病人的ID号等。在每次检查或清洗后，保健员或助手将标记相应的预约已经完成，增加说明，如果有必要的话会安排病人下一次再来。

系统要能够响应按病人名和按日期进行的查询。病人记录数据与预约信息一起显示。接待员可以取消预约，可以打印出前两天预约尚未接诊的通知清单。系统中含有来自病人记录的电话号码。接待员还可以打印出关于所有病人的每天和每周的工作安排。

3. 绘出B&B问题（补充问题11.1）的一个对象模型。
4. 绘出牙科诊所问题（补充问题11.2）的一个对象模型。

习题答案

1. 为什么说Ford汽车是小汽车的特化而引擎不是小汽车的特化？

Ford汽车将具有基类“小汽车”所具有的所有相同的属性和功能。因此，Ford汽车可以从基类“小汽车”中派生出来。而引擎只是小汽车的零件，不是小汽车的特化。许多小汽车的功能和属性引擎都不具有。所以，引擎不能从基类“小汽车”中派生出来。

2. 对象和属性之间的区别是什么？

对象是一个实体，而属性是对象的一个特性。例如，人应该是一个对象，而人的身高应该是属性。有时，这两个概念很难区分。在人/身高的例子中，人可以是一个基类，可能有高个子人、矮个子人和中等身材人等派生类。

3. 领域分析的目标是什么？

领域分析的目标是确定最适合在未来系统中重用的成分。这个方法要找出问题领域中可能的系统之间的共性。

补充问题答案

1. 确定B&B问题陈述的对象。

对象
住宿加早餐的旅店
客房
预约
日历
顾客
保证金
付款
开支

2. 确定牙科诊所问题陈述的对象。

对象
牙科诊所
病人
预约
日历
病人记录
通知清单
每天工作安排
每周工作安排

3. 绘制B&B问题的一个对象模型。
见图11-12。

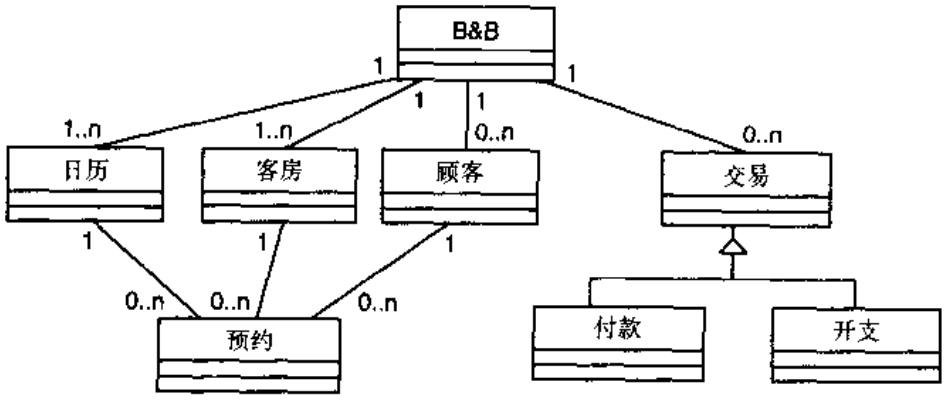


图11-12 B&B对象模型

4. 绘制牙科诊所问题的一个对象模型。

见图11-13。

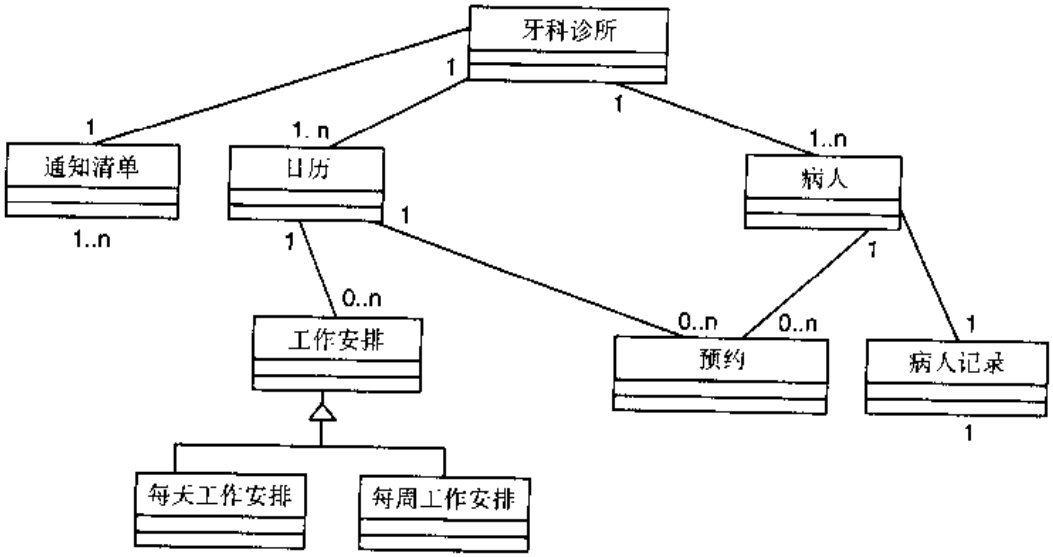


图11-13 牙科诊所的对象模型

第12章　面向对象的度量

12.1　概述

面向对象软件的度量与传统软件的度量有相同的目标（见第5章），即尽量掌握软件的特性。面向对象指的是将函数与数据封装在一起的程序设计语言和程序设计风格。封装是通过限制函数对数据的可访问性以保证数据的完整性来实现的。此外，面向对象的软件还涉及继承和动态绑定。面向对象的软件应该使现实世界模型化，从而更易于理解、更易于修改（可维护性）、更易于重用。不过，面向对象软件的复杂度在源代码的静态结构中并不明显。面向对象软件度量这个领域还是一个探索性的领域。

12.2和12.3节中给出的度量是目前比较重要的观点。12.2节中给出了Chidamber和Kemerer所提出的度量（Shyam Chidamber and Chris Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE TOSE*(Transactions on Software Engineering) 20:6 June 1994, 476-493）。12.3节给出了MOOD度量（Rachel Harrison, Steve Counsell, and Reuben Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics", *IEEE TOSE* 24:6 June 1998, 491-496）。要大家公认面向对象的度量确实有用还需要做进一步的工作。

12.1.1　传统的度量

可以应用传统软件的度量。这些度量在一些大型函数中可能是有用的。非面向对象软件的度量使用控制流程图（以及变体，如数据流程图）作为软件的基本抽象。而控制流程图作为面向对象软件的抽象似乎不太有用。在面向对象的软件开发中，很少有人发表评价利用McCabe环数或Halstead软件科学的著作。应用传统软件度量的直观问题是面向对象软件的复杂度并不表现在控制结构方面。

12.1.2　面向对象的抽象

在大多数面向对象的软件中，方法很小且方法中判断的数目也很少。大多数复杂度表现在方法之间的调用模式上。这方面很少有著作，对什么样的抽象有意义这个问题也没有一致的看法。面向对象的软件开发中最常见的抽象是统一建模语言（UML）中使用的图。

12.2 面向对象设计的度量套件

面向对象设计的度量套件（Metrics Suite for Object-Oriented Design）是评估系统中类的一个全面的方法。这些度量大部分是在每个类的基础上计算的。也就是说，这些度量中没有任何一个将系统作为整体来估计。不清楚怎样把这些度量扩展到整个系统。通常，在一个系统的类上求这些度量的平均值并不恰当。

12.2.1 度量1：每个类的加权方法

度量“每个类的加权方法”（WMC）基于一个直觉，这个直觉认为每个类的方法数目是软件复杂度的一种重要表示。为避免将完整的权重给予一些琐碎的方法，如给予get和set方法，需要作进一步的工作。WMC包括给方法加权的规定。令C为类的一个集合，其中每个类的方法数目分别为$M_1, M_2, \ldots, M_n$。令$c_1, c_2, \ldots, c_n$（c_i的值在相应的文章中没有定义）为各个类的复杂度（权重）。

$$\mathrm{WMC} = \frac{1}{n} \times \sum_{i=0}^{n} c_i \times M_i$$

这是面向对象设计的度量套件中在一个系统的类上求平均的唯一一个度量。在本书的例子中，我们假定c_i等于1。

12.2.2 度量2：继承树的深度

度量“继承树的深度”（DIT）是类的继承树的任意结点到根结点的最大长度。继承会增加软件的复杂度。这个度量是针对每个类进行计算的。

12.2.3 度量3：孩子的数目

不仅继承树的深度重要，继承树的宽度也很重要。度量“孩子的数目”（NOC）为继承层次结构中附属于一个类的即时子类的数目。这个度量是针对每个类进行计算的。

12.2.4 度量4：对象类之间的耦合

应该重视模块之间的耦合（见9.5节）。在面向对象的软件中，我们可以定义耦合为一个类

使用其他类中方法或属性的情况。如果一个类中声明的方法使用另一个类定义的方法或实例变量，则认为这两个类是耦合的。耦合是对称的。如果类A与类B耦合，则类B也与类A耦合。度量“对象类之间的耦合”（CBO）为一个类耦合其他类的数目。

这个度量针对每个类进行计算。

12.2.5 度量5：类的响应

类的响应（RFC）集合{RS}是为了响应类的某个对象所接收到的消息而执行的方法的集合。它是类中所有方法与类中方法所调用的所有方法的并集。只计算一层调用。

$$RFC = |RS|$$

这个度量针对每个类进行计算。

12.2.6 度量6：方法缺乏内聚力

如果一个模块（或类）的所有内容是紧密相关的，就说它是内聚的。度量“缺乏内聚力”（LCOM）试图度量内聚力的缺乏程度。

- 令I_i为方法i使用的实例变量的集合。
- 令P为I_i的成对的空交集。
- 令Q为成对的非空交集。

可通过使用一个双向图来形象化地表示LCOM度量。一个结点集合由属性组成，另一个结点集合由函数组成。如果一个函数访问或设置一个属性，则称该属性连接到该函数。弧的集合为集合Q。如果有n个属性和m个函数，则可能存在$n \times m$个弧，所以P的尺寸为$n \times m$减去Q的尺寸。

$$LCOM = \max(|P| - |Q|, 0)$$

这个度量针对每个类进行计算。

例子12.1

计算下面的样例C++程序的Chidamber度量套件，此C++程序提供矩形的链接列表：

```
class point {
  float x;
  float y;
public:
```

```
    point(float newx, float newy) {x=newx; y=newy;}
    getx(){return x;}
    gety(){return y;}
};
class rectangle {
    point pt1, pt2, pt3, pt4;
public:
    rectangle(float pt1x, pt1y, pt2x, pt2y, pt3x, pt3y, pt4x, pt4y)
        { pt1 = new point(pt1x, pt1y); pt2 = new point(pt2x, pt2y);
        pt3 = new point(pt3x, pt3y); pt4 = new point(pt4x, pt4y);}
    float length(point r, point s){return sqrt((r.getx()-s.getx())^2+
        (r.gety()-s.gety())^2); }
    float area(){return length(pt1,pt2) * length(pt1,pt3);}
};
class linklistnode {
    rectangle* node;
    linklistnode* next;
public:
    linklistnode(rectangle* newRectangle){node=newRectangle; next=0;}
    linklistnode* getNext(){return next;}
    rectangle* getRectangle(){return node;}
    void setnext(linklistnode* newnext){next=newnext;}
};
class rectanglelist {
    linklistnode* top;
public:
    rectanglelist(){top = 0;}
    void addRectangle(float x1, y1, x2, y2, x3, y3, x4, y4) {
        linklistnode* tempLinkListNode; rectangle* tempRectangle;
        tempRectangle = new rectangle(x1,y1,x2,y2,x3,y3,x4,y4);
        tempLinkListNode = new linkListNode(tempRectangle);
        tempLinkListNode->setnext(top);
        top=tempLinkListNode; }
    float totalArea(){float sum; sum=0; linklistnode* temp; temp=top;
        while (temp !=0){sum=sum + temp->getRectangle()->area();
            temp=temp->getNext();}
        return sum;}
};
```

度量1：每个类的加权方法

类	#方法
point	3
rectangle	3
linklistnode	4
rectanglelist	3

WMC=13/4=3.25方法/类

度量2：继承树深度（DIT）
这个例子中没有继承。

度量3：孩子的数目（NOC）
这个例子中没有继承。

度量4：对象类之间的耦合（CBO）
参见图12-1。

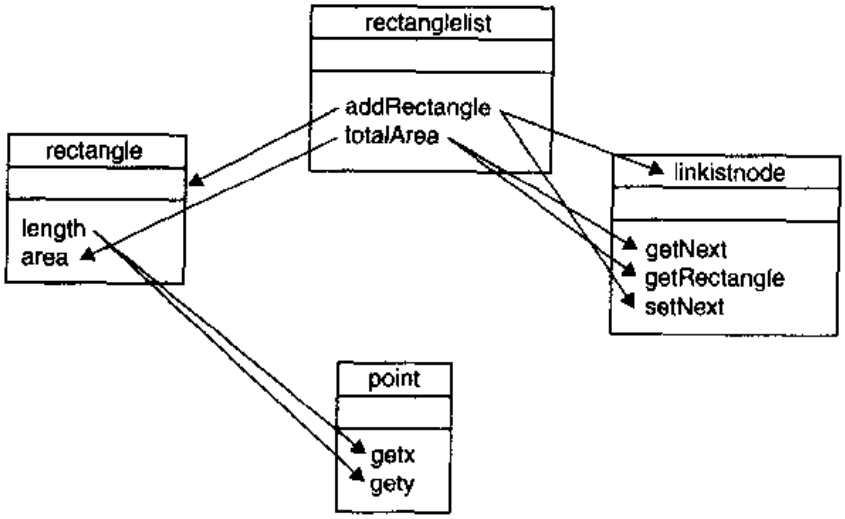


图12-1 对象类之间的耦合

此类图用箭头标注以示出每个函数调用（仅给出其他类中的调用）哪些函数（构造函数）。

类	耦 合 类	CBO
point	rectangle	1
rectangle	point, rectanglelist	2
linklistnode	rectanglelist	1
rectanglelist	rectangle, linklistnode	2

度量5：类的响应（RFC）

类	响应集合	RFC
point	point, getx, gety	3
rectangle	rectangle, point	6
	length, getx, gety,	

（续）

类	响应集合	RFC
	area	
linklistnode	linkListNode,	4
	getNext,	
	getRectangle,	
	setNext	
rectanglelist	rectangleList,	8
	addRectangle,	
	rectangle, setNext	
	totalArea,	
	getRectangle, area,	
	getNext	

度量6：方法缺乏内聚力（LCOM）

参见图12-2。

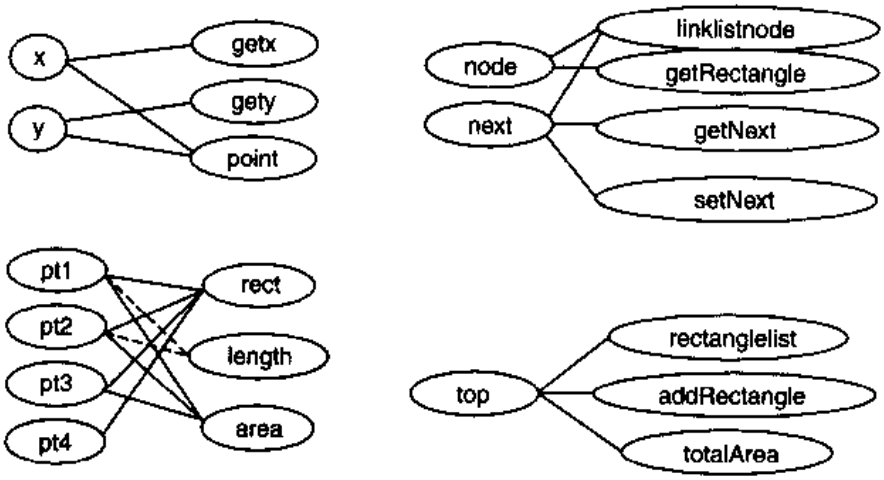


图12-2 LOCOM度量

长度（length）和点（pt）之间的线为虚线，因为它依赖于参数，依赖于这些参数在某个特定的调用中的实际访问情况。

类	LCOM
point	max(0, (6 - 4) - 4) = 0
rectangle	max(0, (12 - 9) - 9) = 0
linklistnode	max(0, (8 - 5) - 5) = 0
rectanglelist	max(0, (3 - 3) - 3) = 0

12.3 MOOD度量

MOOD度量套件也打算成为度量系统的封装、继承、耦合及多态性属性的一个完整的度量集合。

令TC为系统中类的总数。

令$M_d(C_i)$为一个类中声明的方法的数目。

考虑谓词Is_visible($M_{m,i}$,C_j)，其中$M_{m,i}$为类i中的方法m，C_j为类j。如果$i!=J$且C_j可以调用$M_{m,i}$，则这个谓词为1，否则为0。例如，C++中的公用方法对其他类都是可见的，而C++中的私有方法对所有其他类不可见。

方法$M_{m,i}$的可见性$V(M_{m,i})$定义如下：

$$V(M_{m,i}) = \frac{\sum_{j=1}^{TC} \text{Is\_visible}(M_{m,i}, C_j)}{TC - 1}$$

12.3.1 封装

方法隐藏因子（MHF）和属性隐藏因子（AHF）可用来度量封装。

$$\text{MHF} = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{m,i}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

$$\text{AHF} = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{m,i}))}{\sum_{i=1}^{TC} A_d(C_i)}$$

例子12.2

计算下列C++代码的MHF和AHF。

```
Class A{
  int a;
public:
  void x();
  void y();
```

```
};
Class B {
  int b;
  int bb;
  void w();
public:
  void z();
};
Class C {
  int c;
  void v();
};
TC = 3
```

方法	is_vis(A)	is_vis(B)	is_vis(C)	$V(M_{m,i})$
A::x()	0	1	1	1
A::y()	0	1	1	1
B::w()	0	0	0	0
B::z()	1	0	1	1
C::v()	0	0	0	0

MHF = 2/5 = 0.4

属　性	is_vis(A)	is_vis(B)	is_vis(C)	$V(A_{m,i})$
A::a()	0	0	0	0
B::b()	0	0	0	0
B::bb()	0	0	0	0
C::c()	0	0	0	0

AHF = 4/4 = 1.0

12.3.2 继承因子

有两个继承度量，分别为方法继承因子（MIF）和属性继承因子（AIF）。

- $M_d(C_i)$ = 类i中声明的方法的数目
- $Mi(C_i)$ = 类i中继承（且不覆盖）的方法的数目
- $M_a(C_i) = M_d(C_i) + M_i(C_i)$ = 类i的关联中可调用的方法的数目

$$\mathrm{MIF} = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

- $A_d(C_i)$ = 类i中声明的属性的数目
- $A_i(C_i)$ = 类i中可访问的基类的属性的数目
- $A_a(C_i) = A_d(C_i) + A_i(C_i)$ = 类i的关联中可访问的属性的数目

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

例子12.3

计算下列C++代码的MIF和AIF：

```
Class A{
protected:
   int a;
public:
   void x();
   virtual void y();
};
Class B public A {
   int b;
protected:
   int bb;
public:
   void z();
   void y();
   void w();
};
Class C public B {
   int c;
   void v();
};
```

类	Md	Mi	Ad	Ai
A	x(), y()	无	a	无
B	w(), z(), y()	A::x()	b, bb	A::a
C	v()	B::w(), z(), y() A::x()	c	B::bb

MIF = 5/11 AIF = 2/6

12.3.3 耦合因子

耦合因子（CF）度量类之间的耦合性，但不包括由于继承导致的耦合。

如果类i与类j有关系，则令is_client(c_i, c_j) = 1，否则为0。此关系可以是类 i调用类 j的一个方法，或者有一个对类j或类j中的属性的引用。这个关系不能是一个继承。

$$CF = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC} is\_client(c_i, c_j)}{TC^2 - TC}$$

例子12.4

计算图12-3所示的住宿加早餐的旅店问题（补充问题11.4）的对象模型上的耦合因子。仅在图中所示的关联需要时才假定有一个关系。

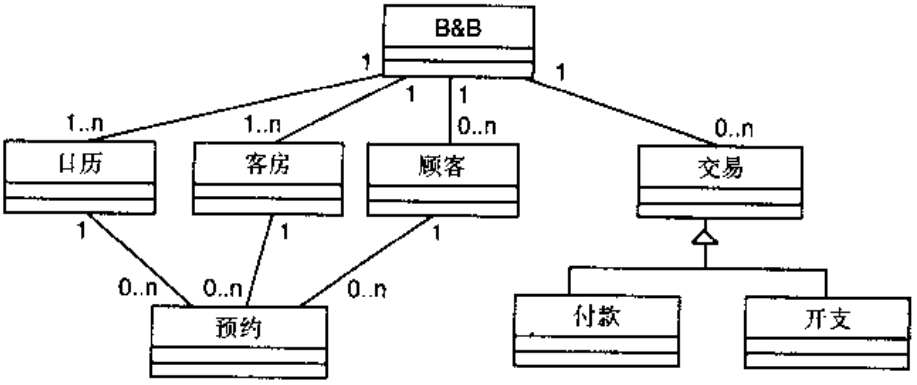


图12-3 11.4问题的对象模型

TC = 7

类	is_client类
B&B	日历、客房、顾客、交易
日历	预约
客房	预约
顾客	预约
交易	无
付款	无
开支	无

CF = 7/42

12.3.4 多态性因子

多态性因子（PF）为可能的多态性的一个度量。

- 令$M_o(C_i)$为类i中覆盖的方法的数目。

- 令$M_n(C_i)$为类i中新方法的数目。
- 令$DC(C_i)$为类i的子孙的数目。

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) \times DC(C_i)]}$$

例子12.5

计算例子12.3的C++代码的多态性因子。

类	M_n	M_o	DC
A	x(), y()	无	2
B	w(), z()	y()	1
C	v()	无	0

$$PF = 1/(2 \times 2 + 2 \times 1 + 1 \times 0) = 1/6$$

习题

1. 为什么McCabe环数与Halstead的软件科学不适合应用于面向对象的软件?

2. 面向对象的设计中可使用的何种抽象可作为面向对象的度量的基础?

3. 整个系统的度量何时不同于对每个类计算的度量的和或度量的平均数?

4. 较高的LCOM是好还是坏?

5. 有人曾经建议在LCOM中，仅使用P和Q的尺寸之差。也就是说，使用零与这个差的最大者并不实际。这种更改的效果是什么?

补充问题

1. 计算下列维护人员/学生数组的代码的Chidamber度量:

```
class person{
  char* name;
  char* ssn;
public:
  person(){name = new char[NAMELENGTH]; ssn = new
      char[SSNLENGTH];}
```

```
  ~person(){delete name; delete ssn;}
  void addName(char* newname){strcpy(name, newname);}
  void addSsn(char* newssn){strcpy(ssn, newssn);}
  char* getName(){return name;}
  void virtual display(){cout << "the person's name is
      "<<name;}
};
class student public person {
  float gpa;
public:
  void addGpa(float newgpa){gpa = newgpa;}
  void display(){cout<<"the student's name is"
  <<getName()<<" and gpa is " << gpa;}

};
class personlist {
  person* list[MAX];
  int listIndex;
public:
  personlist(){listIndex = 0;}
  void addPerson(char* newname, char*
  newssn){list[listIndex]=new person;
    list[listIndex]->addName(newname); list[listIndex]
    ->addSsn(newssn);
    listIndex++;}
  void addStudent(char* newname, char* newssn, float gpa)
    {student* temp = new student;
    temp->addName(newname); temp->addSsn(newssn);
    temp->addGpa(newgpa);list[listIndex++]=temp;}
  void display(){int j; for(j=0; j<listIndex; j++) list[j]
  ->display();}
};
```

习题答案

1. 为什么McCabe环数与Halstead的软件科学不适合应用于面向对象的软件?

这两个度量都是基于写成单个函数的一个算法的尺寸和复杂度的。面向对象的功能一般分布在许多方法中，而且还分布在不同的类中。每个面向对象的函数通常都较小并且相对简单。因此，这两种度量方法一般得不出面向对象的系统的较好的复杂性度量。

2. 面向对象的设计中可使用的何种抽象可作为面向对象的度量的基础?

标准的抽象为UML图，即对象模型、用例图、状态模型和顺序图。这些内容似乎都不能

抓住面向对象软件中基本的复杂性概念。

3. 整个系统的度量何时不同于对每个类计算的度量的和或度量的平均数?

如果个别类的度量基本上是尺寸度量，如LOC或孩子的数目，则求这些个别度量值的和，以获得整个系统的度量值或者每个类的平均尺寸会具有一定的意义。如果个别类的度量是一个平均值，则平均值的平均值也可能是合理的，例如，每个函数的平均参数个数。

但是，和或平均数都不是一个良好的类之间相互作用的度量。

4. 较高的LCOM是好还是坏?

坏．因为这表示高度缺乏内聚力。

5. 有人曾经建议在LCOM中，仅使用 P和 Q的尺寸之差。也就是说，使用零与这个差的最大者并不实际。这种更改的效果是什么?

这种更改的效果是允许区分更有内聚力的类。一个具有内聚力的类仅映射为零。

补充问题答案

1. 计算维护人员/学生数组的问题陈述中代码的Chidamber度量。

度量1：每个类的加权方法

类	方法数目
person	6
student	2
personlist	4

注意，继承的函数不计算在内。

WMC = 12/3 = 每个类4个方法

度量2：继承树的深度（DIT）

类	DIT
person	0
student	1
personlist	0

度量3：孩子数（NOC）

类	NOC
person	1
student	0
personlist	0

度量4：对象类间的耦合（CBO）

见图12-4。

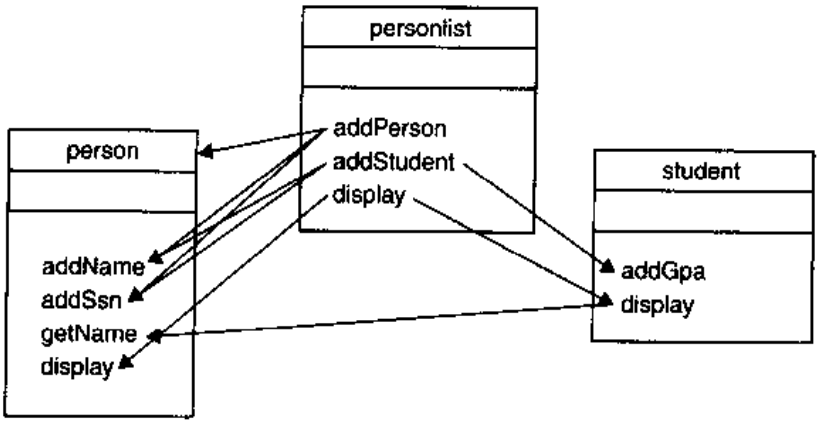


图12-4 对象类间的耦合

这个类图用箭头标注以示出哪些函数被每个函数调用（仅给出其他类中的调用）。

类	耦 合 类	CBO
person	student, personlist	2
student	person, personlist	2
personlist	person, student	2

度量5：类的响应（RFC）

类	响应集合	RFC
person	person, addName, addSssm getName, display	5
student	student, addGpa, person, getName	6
personlist	personlist,addPerson,addStudent,addName,addSsn,addGpa,display	7

度量6：方法缺乏内聚力（LCOM）

参见图12-5。

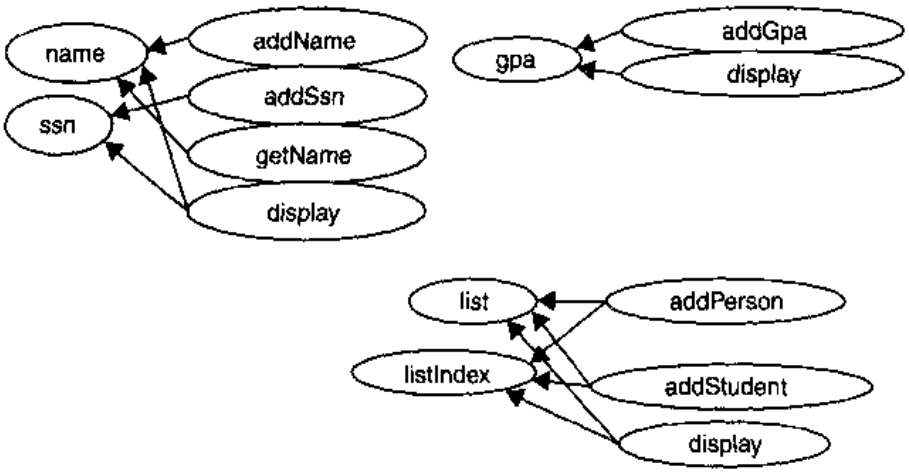


图12-5 LCOM度量

类	LCOM
person	max(0, (8−5)−5) = 0
student	max(0, (2−2)−2) = 0
personlist	max(0, (6−6)−6) = 0

第13章　面向对象的测试

13.1　概述

测试面向对象的软件提出了一些新挑战。许多传统的技术仍然能够使用。例如，面向对象软件的功能测试与传统软件的功能测试没有什么不同。测试案例将根据需求文档中描述的功能来建立。但是，面向对象软件的结构测试与传统软件的结构测试极不相同。我们将介绍面向对象软件的两种结构测试方法——MM测试和函数对测试。

1. 传统软件

传统软件的测试常常是根据在软件结构上定义的覆盖准则进行的。标准的方法（参见第10章）有语句覆盖、分支覆盖和数据流覆盖。这些覆盖准则都是基于控制流程图或改进控制流程图的。

2. 面向对象的软件

面向对象的软件增加了软件测试的复杂性。控制流程图不再是软件结构的一种良好表示，基于对象模型的结构测试会更为合适。但是，至今尚未找到有效的对象模型覆盖度量。

类中的方法应该用已经介绍过的技术进行测试。相同的覆盖准则可应用于面向对象的软件。但在直观上，语句和分支覆盖准则似乎已经不适合彻底地测试面向对象软件的复杂度了。必须测试方法之间的交互作用。

面向对象软件测试的一种方法是覆盖对所有方法的调用。有时，这种方法称为MM测试。

13.2　MM测试

MM测试（方法－消息测试）覆盖要求测试每个方法调用。因此，在每个方法中，对其他方法的调用都必须至少测试一次。如果一个方法多次调用另一个方法，那么每次调用只需测试一次。这似乎是最基本的覆盖准则。MM测试不包含每条语句覆盖（参阅10.3.1）。

例子13.1

确定矩形问题的连接表的MM测试覆盖。

```
class point {
   float x;
   float y;
public:
   point(float newx, float newy) {x=newx; y=newy;}
   getx(){return x;}
   gety(){return y;}
};
class rectangle {
   point pt1, pt2, pt3, pt4;
public:
   rectangle(float pt1x, pt1y, pt2x, pt2y, pt3x, pt3y, pt4x, pt4y)
      { pt1 = new point(pt1x, pt1y); pt2 = new point(pt2x, pt2y);
      pt3 = new point(pt3x, pt3y); pt4 = new point(pt4x, pt4y);}
   float length(point r, point s){return sqrt((r.getx()-s.getx())^2+
      (r.gety()-s.gety())^2); }
   float area(){return length(pt1,pt2) * length(pt1,pt3);}
};
class linklistnode {
   rectangle* node;
   linklistnode* next;
public:
   linklistnode(rectangle* newRectangle){node=newRectangle; next=0;}
   linklistnode* getNext(){return next;}
   rectangle* getRectangle(){return node;}
   void setnext(linklistnode* newnext){next=newnext;}
};
class rectanglelist {
   linklistnode* top;
public:
   rectanglelist(){top = 0;}
   void addRectangle(float x1, y1, x2, y2, x3, y3, x4, y4) {
      linklistnode* tempLinkListNode; rectangle* tempRectangle;
      tempRectangle = new rectangle(x1,y1,x2,y2,x3,y3,x4,y4);
      tempLinkListNode = new linkListNode(tempRectangle);
      tempLinkListNode->setnext(top);
      top=tempLinkListNode; }
   float totalArea(){float sum; sum=0; linklistnode* temp; temp=top;
      while (temp !=0){sum=sum + temp->getRectangle()->area();
         temp=temp->getNext();}
      return sum;}
};
```

调用结构如下所示。列出每个类的函数，然后列出调用其他函数的每个函数，也要列出被调用的函数。对于MM测试，每个调用都必须执行。例如，对point的4个调用都要执行。没有给出判断。不过，这个程序中没有影响调用的顺序的判断。

```
class point
   point()
   getx()
   gety()
```

```
class rectangle
  rectangle()
    point::point()
    point::point()
    point::point()
    point::point()
  length()
    point::getx()
    point::getx()
    point::gety()
    point::gety()
  area()
    length()
    length()

class linklistnode
  linklistnode()
  getNext()
  getRectangle()
  setnext()
class rectanglelist
  rectanglelist()
  addRectangle()
    rectangle::rectangle()
    linklistnode::linklistnode()
    linklistnode::setnext()
  totalArea()
    linklistnode::getRectangle()
    rectangle::area()
    linklistnode::getNext()
```

MM测试：建立至少一个矩形，然后得出总面积的任意测试案例都将执行所有这些调用。

13.3 函数对的覆盖

函数对的覆盖要求对所有可能的方法执行序列，必须测试长度为2的那些序列。这通常是基于状态机的图或基于显示可能的方法执行的正则表达式来完成的。

因为正则表达式可映射为一个有限状态机，所以上述两种方法是等同的。虽然用来描述软件系统行为的有限状态机可能不是最小的，但是有一些额外的状态将会增加测试集的有效性。

例子13.2

确定例子13.1的矩形程序连接表的函数对测试覆盖。考虑这些函数的调用结构。

```
class point
  point()
```

```
    getx()
    gety()

  class rectangle
    rectangle()
      point()point()point()point()
    length()
      getx()getx()gety()gety()
    area()
      length()length()

  class linklistnode
    linklistnode()
    getNext()
    getRectangle()
    setnext()

  class rectanglelist
    rectanglelist()
    addRectangle()
      rectangle()linklistnode()setnext()
    totalArea()
      (getRectangle()area()getNext())*
```

单个函数的正则表达式多数具有固定的方法调用列表。只有totalArea函数在while循环中重复了零次或多次。

把所有这些内容放入一个正则表达式，并先建立rectanglelist，然后可以完成addRectangle或totalArea，这给出下列正则表达式：

```
rectanglelist ((addRectangle rectangle point point point point
linklistnode setnext) | (totalArea (getRectangle area length getx getx
gety gety length getx getx gety gety getNext)*)
```

相应的函数对测试可通过下列测试集完成：

1) 建立一个矩形，然后计算面积。

2) 建立两个或多个矩形，然后计算面积。

3) 不建立矩形，计算面积。

4) 在计算面积后建立矩形。

例子13.3

确定图13-1的状态机中给出的有穷堆栈的函数对测试覆盖。无目的状态的弧表示错误转换。

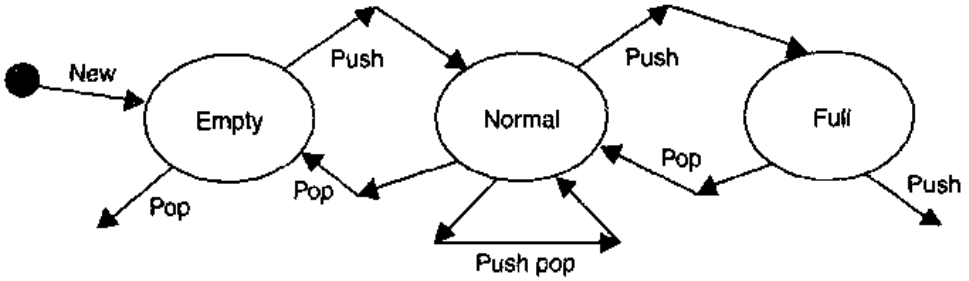


图13-1 状态机模型

这个例子在测试中需要下列的函数对。

1. new pop(on empty – error)
2. new push
3. push (from empty) push
4. push (from empty) pop
5. push (from normal to normal) push (still in normal)
6. push (from normal to normal) push (into full)
7. push (from normal to normal) pop
8. push (from normal to full) push (error)
9. push (from normal to full) pop
10. pop (from normal to normal) push (still in normal)
11. pop (from normal to normal) pop (still in normal)
12. pop (from normal to normal) pop (into empty)
13. pop (into empty) push
14. pop (into empty) pop (error)

例子13.4

对于下列代码样例，确定函数对测试覆盖需要覆盖的函数对。为每个在其体内有方法调用的函数以及整个程序建立正则表达式。

```
class D {
  int x;
  int Db(int s) {return 2*s;}
  int Dc(int s) {return s;}
public:
  D(){x=0;}
  int Da(int s){if (x=1) {x=0; return Db(s);} else {x=1; return Dc(s);}}
};
class A {
  int x;
  int y;
  D* m;
public:
  A(){m=new D;}
  virtual int Aa(int s){cout<<x; return m->Da(y);}
```

```
  void add(int s, int u){x=s;y=u;}
};
class B {
  A* w[MAX];
  int z;
  int q;
public:
  B() {z=0; q =1;}
  int Bread(){cin>>s>>u; if (z= MAX) return 0;
    if(s<q){w[z]=new A; w[z]->add(s,u);}
    else {w[z]=new C; w[z]->add(u,s);w[z]->Atadd(s);}
    z++; return z;}
  int Ba(int s){q=w[s]->Aa(q); return q;}
};
class C public A{
  int t;
public:
  int Aa(int r) {cout<<t; return m->Da(t);}
  void Atadd(int x) {t = x;}
};
```

每个函数及其体内的函数调用的正则表达式为：

```
class D
  Da : (Db | Dc)
class A
  A: D
  Aa: Da
class B
  Bread: e | (A add | C add Atadd)
  Ba : Aa
class C
  C: A
  Aa: Da
```

可能调用的正则表达式为：

```
B (Bread (e | A D add | C A D add Atadd)) | Ba Aa Da (Db | Dc))*
```

函数对覆盖必须包括下列函数对：

```
B Bread
Bread A
Bread C
Bread Bread
B Ba
Da Db
add Bread
add Ba
Da Dc
Atadd Bread
Atadd Ba
Db Ba
```

```
Db Bread
Dc Bread
Dc Ba
```

习题

1. 怎样进行面向对象软件的功能测试?

2. 面向对象软件的语句覆盖有用吗?

3. MM测试包括语句覆盖吗? (参见10.3.1节。)

4. 函数对覆盖的优点是什么?

补充问题

1. 给出下列代码，生成满足MM测试准则和每个函数对测试的测试案例。

```
class Threes {
  char cout;
public:
  Threes(){count = 'a';}
  void PlusOne(){ if(count == 'a') count = 'b'; if(count == 'b')
  count = 'c';
    if(count == 'c') count = 'a'; }
  char* IsDiv() {if(count == 'a'){return 'yes';}else{return
  'no';}}
}
class Num {
  Threes* SumMod3; int last; int number;
  void Digit(int newnum){int j; for (j=1; j<=newnum; j++)
  SumMod3->PlusOne();}
public:
  Num(){SumMod3 = new Threes;}
  void Reduce() {while (number > 0){last = number - (number/
  10)*10;
    Digit(last); number = number/10;}
  char* IsDivisibleBy3(int newnum) {number = newnum; Reduce;
    return SumMod3->IsDiv();}
}
Main() {
  Num* Test = new Num;
  int value;
  char* answer;
```

```
cin >> value;
answer = test->IsDivisibleBy3(value);
cout << answer;
```

习题答案

1. 怎样进行面向对象软件的功能测试?

面向对象软件的功能测试与传统软件的功能测试没什么不同。

2. 面向对象软件的语句覆盖有用吗?

有用，面向对象软件的语句覆盖应该进行。它可能是最小的可接受的覆盖。

3. MM测试包括语句覆盖吗? （参见10.3.1节。）

不包括。MM测试要求测试每个方法调用，而有的源代码段可能不包含方法调用，因此不会被达到MM测试的测试案例集所测试。

4. 函数对覆盖的优点是什么?

函数对覆盖保证执行调用的组合。在堆栈的例子中（13.3），如果通过一个用户接口调用堆栈，那么每个函数可能只有一次调用。所以，利用函数对覆盖，一个简单的create、push、pop序列就可以完成了MM测试。函数对覆盖包含MM测试。

补充问题答案

1. 给出下列代码，生成满足MM测试准则和每个函数对测试的测试案例。

```
class Threes {
  char cout;
public:
  Threes(){count = 'a';}
  void PlusOne(){ if(count == 'a') count = 'b'; if(count == 'b')
  count = 'c';
    if(count == 'c') count = 'a'; }
  char* IsDiv() {if(count == 'a'){return 'yes';}else{return
  'no';}}
}
class Num {
```

```
  Threes* SumMod3; int last; int number;
  void Digit(int newnum){int j; for (j=1; j<=newnum; j++)
  SumMod3->PlusOne();}
public:
  Num(){SumMod3 = new Threes;}
  void Reduce() {while (number > 0){last = number - (number/
  10)*10;
    Digit(last); number = number/10;}
  char* IsDivisibleBy3(int newnum) {number = newnum; Reduce;
    return SumMod3->IsDiv();}
}
Main() {
  Num* Test = new Num;
  int value;
  char* answer;
  cin >> value;
  answer = test->IsDivisibleBy3(value);
  cout << answer;
```

调用的正则表达式：

```
main: Num IsDivisibleBy3
Num: Threes
Reduce: Digit *
IsDivisibleBy3: Reduce IsDiv
Digit: PlusOne*
Threes:
PlusOne:
IsDiv
```

MM测试：

此测试集必须执行所有调用。

一个测试案例，输入5，输出no，应该执行了所有调用。

每个函数对：

测试集必须具有用于每个IsDiv转换的一个测试，因此，输入6和7应该完成覆盖。输出分别为yes和no。

第14章　形式化表示方法

14.1　概述

形式化表示方法是一种基于数学的表示方法。此表示方法的语法和/或语义都以数学概念为基础。形式化表示方法在软件开发中有减少错误的巨大潜力。大家还没有充分认识到这些好处，主要原因是构造和使用形式化的规格说明很困难。

自然语言的问题是它很模糊，有二义性或多义性。通常，使用自然语言的规格说明要依赖于词的语义/含义来表达所要表达的内容。

规格说明应该回答问题。任何规格说明，无论是形式化的还是非形式化的，都用来评价开发人员关于对于特定行为的问题是否回答得良好。形式化的规格说明能够更准确地回答问题。

形式化有三个级别：

- 非形式化：非形式化的技术具有灵活的规则，不包含可以建立的模型。
- 半形式化：半形式化的技术具有适当定义的语法。
- 形式化：形式化的技术具有严格定义的语法和语义。

14.2　形式化的规格说明

形式化的规格说明利用形式化定义的模型来构造软件行为的表述。例如，一个形式化的规格说明可以利用集合表示法构造其模型。必须有一种方法将软件映射到形式化模型，将软件中的过程映射到形式化模型中的过程，以及把形式化模型中的语句映射为软件中的语句。

例如，可以利用序列的数学表示来说明一个堆栈的行为。可以为每个堆栈操作指定数学上等价的东西。然后，给出堆栈上的操作集合，我们可以把这些操作映射为数学序列上的操作。在完成序列上的操作后，可以将结果映射到堆栈。这样，就可以利用数学序列精确地指定堆栈的行为了。

一般来说，形式化规格说明中构造的语句可分为三类，分别是：前提条件、后置条件、

不变式。

14.2.1 前提条件

前提条件是一个与函数相关的条件，在该函数可以执行前，此前提条件必须为真。前提条件有两种解释：

- 不指定错误处理——如果不满足前提条件，要进行某种错误处理。这种风格假定要扩展相应的实现以处理这些错误条件。
- 指定所有错误处理——假定如果不满足条件，则不调用相应的函数。这种规格说明将扩展以指定希望实现的函数要处理的所有错误条件。

14.2.2 后置条件

后置条件也与一个函数有关。后置条件指定函数完成后所出现的变化。一般来说，形式化表示方法有一种指定执行前的状况与执行后的状况的表示法。例如，有的表示方法利用一个省略符来标记变量以代表完成后的变量值，而没有省略号的变量代表函数执行开始前的值。

14.2.3 不变式

不变式是一个总是为真的语句。实际上，在一个函数或语句执行中，它可能不是真的。但是，它在每个函数完成前或完成后为真（甚至在函数内的活动中，不变式可能一直都不是真的。例如，一个循环内的不变式可能描述一种涉及两个变量的关系。但是，这两个变量的值可能在两条不同的语句中更新了。在这两条语句之间，不变式可能不为真）。

这里举出堆栈的两个不变式例子。堆栈包含小于或等于所允许的最大数目的项，这是一个不变式。另一个不变式可以是某些字段不为空。

14.3 对象约束语言

对象约束语言（OCL）为UML规范的组成部分（OCL规范可在Web上找到。在浏览器搜索工具中输入“OCL”或浏览www.software.ibm.com/ad/ocl）。它最初是用来指定UML的成分的。目前在UML规范中没有支持OCL语句分析的工具。

OCL用对象模型来提供规格说明的上下文关系。大多数OCL语句判定一个实体的集合。OCL包含可应用于结果集合的操作和比较。

OCL的语句总是用上下文来编写的。此上下文通常是对象模型中的一个类。上下文由类的加下划线的名称给出。

OCL表达式“self”启动导航。它引用类的一个实例。

14.3.1 导航

OCL表达式可使用关联的另一方的角色名、关联名或类名。结果是一个集合，或者是一个元素。如果多重性是0或1，表示单个对象。否则，表示一个集合。

例子14.1

下面的OCL语句计算当前从图书馆借出的所有书籍的集合（见图14-1所示的对象）。

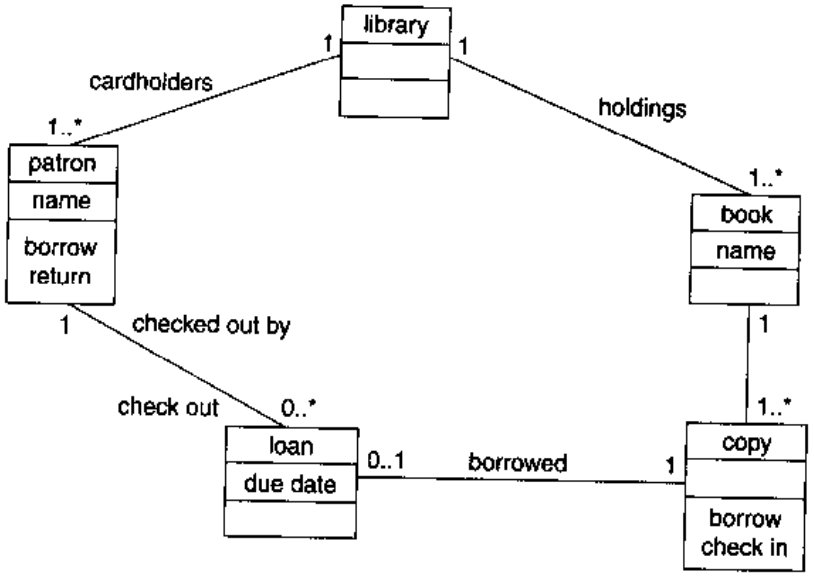


图14-1 图书馆的关系模型

```
library
  self.holdings.copy.borrowed
  self.cardholders.checkedout
```

下划线的library表示上下文。self指出了类库的一个实例。表达式self.holdings判定类book的所有实例的集合。表达式self.holdings.copy判定所有book的所有copy的集合。表达式self.holdings.copy.borrowed判定library中所有book的loan的所有实例的集合。

表达式self.cardholders判定patron的所有实例的集合。表达式self.cardholders.checkedout判

定loan的所有实例的集合。

14.3.2 不变式

OCL中的不变式通常是用上下文编写的，此上下文以下划线的对象名给出。一般来说，用导航确定一个与别的集合或元素进行比较的集合或元素。可为导航的结果使用函数。

例子14.2

利用例子14.1的表达式编写图书馆问题的不变式。

```
library
self.holdings.copy.borrowed = self.cardholders.checkedout
```

带下划线的library给出这个不变式的上下文。此不变式描述持卡者借阅的书籍集合与登记借出的书籍副本集合相同。

```
library
self.holdings.copy = self.cardholders.checkedout.borrowed
```

前一个不变式写法是正确的，但它不为真。后一个表达式self.holdings.copy判定library中书籍的copy的所有实例的集合。另一表达式判定当前登记借出书籍的copy的所有实例的集合。只有在library中所有书籍当前都登记借出的情况下该表达式才为真。

14.3.3 属性

表达式也可以引用属性的值。同样使用点表式法。

例子14.3

编写一个说明“Grapes of Wrath”不在图书馆中的不变式。

```
book
self.name <> ''Grapes of Wrath''
```

上下文为类book。表达式self.name判定书名的值。

14.3.4 预定义操作

OCL有许多集合上的操作，分别为size、count(object)、includes(object)、sum和

includesall(collection)。

例子14.4

编写一个说明任何读者都不能一次借出10本书的不变式。

```
Patron
self.checkedout->size < 10
```

表达式self.checkedout判定与读者关联的借出书籍的集合。操作size返回集合中书籍的数目，不变式要求此数目小于10。

14.3.5 前提条件和后置条件

在OCL中，前提条件和后置条件的上下文必须用一个带下划线的函数给出。语法pre:和post:区分是前提条件还是后置条件。可使用关键字result来指定操作的结果。

在OCL中语法@pre用来指定一个操作前的值。

例子14.5

写出前提条件和后置条件以保证一个读者不能借出9本以上的书。

```
patron::borrow
pre: self.checkedout->size < 9
post: self.checkedout->size < 10
```

或者：

```
post: self.checkedout@pre->size + 1 = self.checkedout->size
```

习题

1. 规格说明能回答何种问题?
2. 为什么二义性是一个问题?
3. 为什么数学表示法（如集合）是规格说明的良好基础?
4. 前提条件、后置条件和不变式之间的区别是什么?

补充问题

1. 给出图14-2所示的对象模型，判定每个给出的OCL语句。如果语句是错误的，说明错

在何处，并给出最简单的纠正方法。

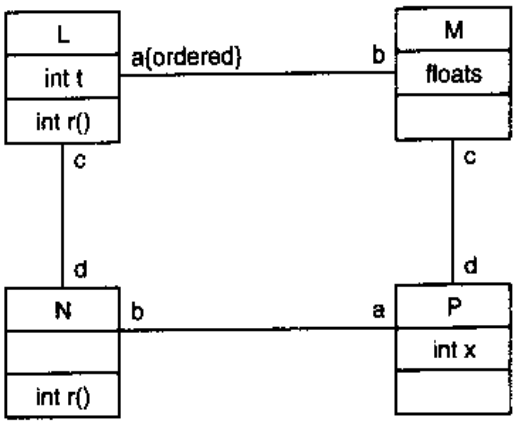


图14-2 问题14-1的对象模型

```
L
self.c->size = 10
self.a = self.c.b.d

          L::r() : int
pre: self.a.b = self.c.a
post: t = t@pre + 1
post: result = self.a->first.s

P
self.a.d->size > max
self.a.b = self.d.c
          N::q() : int
pre: self.b->isEmpty
pre: self.d->forall( l |l.t < 10)
post: result = self.d->size
```

2. 给出图14-3所示的对象模型，解释每条OCL语句。它说明了什么？其OCL不变式合理吗？它总是为真吗？

```
familytree
  a) self.person = self.marriage.child

marriage
  b) self.child.birthfam = self
  c) self.husband.birthdate < self.wife.birthdate

person
  d) self.birthfam.child_include(self)
  e) self.marriage->size = 1
```

f) self.marriage.wife.birthdate < self.birthdate

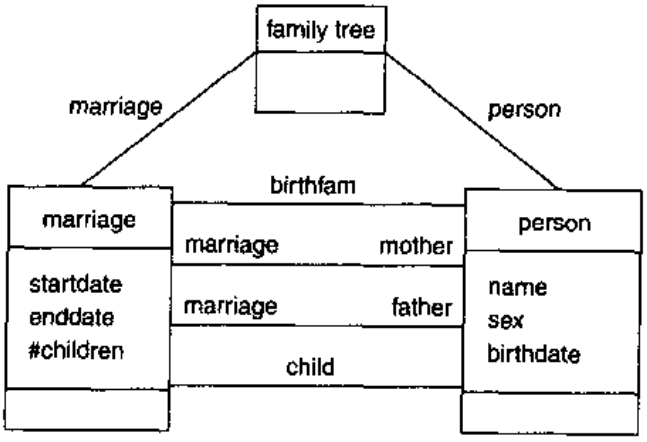


图14-3 问题14-2的对象模型

3. 编写一个餐馆的OCL约束，此餐馆没有吸烟区，顾客按到来的先后顺序入座。

```
Class group
  Char* name
  Int number
  Int arrivalorder
Class waitlist
  Group* list[MAX]
  Int listptr
  Void addtolist(group* newgroup)
  Group* seatnext()
Class restaurant
  Waitlist* waiting
  Void arrive(group* newgroup)
  Group* seat()
```

习题答案

1. 规格说明能回答何种问题?

一般来说，规格说明能回答的问题是关于打算研制的软件的行为的问题。开发人员应该能够利用此规格说明来精确地确定软件在某种特定情形下的行为。

2. 为什么二义性是一个问题?

如果二义性表示开发人员对规格说明的解释与用户的期望有所出入，就会出问题。

3. 为什么数学表示法（如集合）是规格说明的良好基础?

诸如集合这样的数学概念是规格说明的良好基础，因为集合与集合的运算是精确定义的。例如，两个集合的并就很好理解。如果某个函数的行为可定义为特定集合上的运算，就可以很准确地确定该函数应该完成什么工作。

4. 前提条件、后置条件和不变式之间的区别是什么?

前提条件在函数可以执行前必须为真。后置条件在函数完成后必须为真。不变式应该在整个函数的执行中为真。实际上，多数不变式在每个运算之间都为真。

补充问题答案

1. 除pre:self.a.b = self.c.a外，所有语句都是对的。pre:self.a.b = self.c.a应该为pre:self.a.b = self.c.d。

2. 给出图14-4所示的对象模型，解释每条OCL语句。它说明了什么？其OCL不变式合理吗？它总是为真吗?

```
familytree
  a) self.person = self.marriage.child

marriage
  b) self.child.birthfam = self
  c) self.husband.birthdate < self.wife.birthdate
person
  d) self.birthfam.child_include(self)
  e) self.marriage->size = 1
  f) self.marriage.wife.birthdate < self.birthdate
```

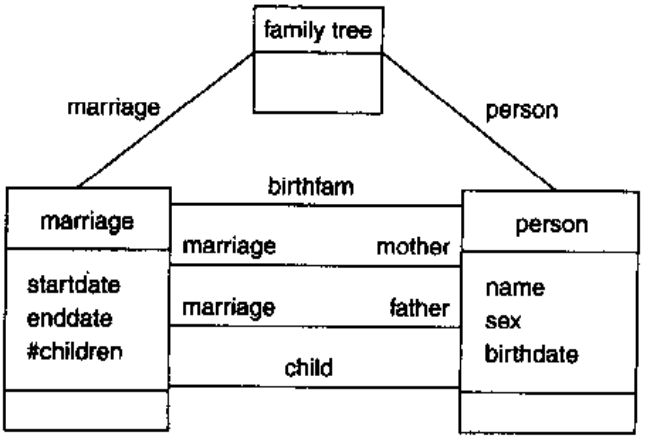


图14-4 问题14-2的对象模型

a. 这个不变式表示，人员集合与所有孩子的集合相同，或者每个人具有导致其出生的婚姻。这是一个合理的不变式，如果数据完整，它为真。

b. 这个不变式表示，每个孩子都具有自己出生的家族，它与指向作为孩子的人员的实例相符。这是合理的并且总为真。

c. 这表示每个丈夫都比其妻子年长。可以描述这个不变式，但它不太符合实际。

d. 这表示可从birthfam达到的孩子（兄弟）的集合包括人员。这是合理的，并且总是为真。

e. 这表示一个人的婚姻的集合只有一个。这不太符合实际。

f. 不管是你自己的生日（如果你为女性）还是你的配偶（如果你为男性）的生日，总是小于你自己的生日。这不太合理，不总是为真。

3. 编写一个餐馆的OCL约束，此餐馆没有吸烟区，顾客按到来的先后顺序入座。

```
Class group
  Char* name
  Int number
  Int arrivalorder

Class waitlist
  Group* list[MAX]
  Int listptr
  Void addtolist(group* newgroup)
  Group* seatnext()
Class restaurant
  Waitlist* waiting
  Void arrive(group* newgroup)
  Group* seat()
```

等待列表：

```
    Self.Listptr = self.list->size
Void waitlist::addtolist(group* newgroup)
    Pre:  self.listptr < MAX
    Post: forall(i | list[i] = list[i-1]@pre)
          List[0] = newgroup
          Listptr = listptr@pre+1
Group* waitlist::seatnext()
    Pre:  self.listprt > 0
    Post: Result = list[listptr@pre]
          Self.listptr = self.listprt@pre - 1
```

```
Group* Restaurant::seat
    Pre:  waiting.waitlist->size > 0
    Post: waiting.waitlist->size = waiting. waitlist@pre->size
    - 1
      Result = waiting.seatnext()
      and forall (x : group | waiting.seatnext().arrivalorder
      <= x.arrivalorder)
```