RUANJIAN KEKAOXING GAILUN

软件可靠性概论

郭新峰◎著

中国水利水电出版社
www.waterpub.com.cn

软件可靠性概论

郭新峰◎著

中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

随着软件规模日益庞大、软件应用领域空前广泛，社会对软件的依赖程度日益增长。同时，软件的运行异常给人们生产、生活等各个方面造成严重威胁，成为迫切需要解决的问题。软件可靠性是软件质量的重要内容，本书基于软件可靠性的基础理论和办法，借鉴国内外相关研究成果，针对软件可靠性这个综合性课题，应用系统分析的方法，借助机器学习的最新研究成果，介绍并分析了软件全生命周期各种软件可靠性模型，形成一套较为系统的软件可靠性的理论和方法，可作为软件可靠性工程的相关指导。

图书在版编目（CIP）数据

软件可靠性概论 / 郭新峰著. -- 北京 : 中国水利水电出版社, 2015.8
ISBN 978-7-5170-3627-2

Ⅰ. ①软… Ⅱ. ①郭… Ⅲ. ①软件可靠性－研究
Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2015)第206369号

书　　名	**软件可靠性概论**
作　　者	郭新峰　著
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路1号D座　100038） 网址：www.waterpub.com.cn E-mail：sales@waterpub.com.cn 电话：（010）68367658（发行部）
经　　售	北京科水图书销售中心（零售） 电话：（010）88383994、63202643、68545874 全国各地新华书店和相关出版物销售网点
排　　版	北京时代澄宇科技有限公司
印　　刷	北京九州迅驰传媒文化有限公司
规　　格	184mm×260mm　16开本　11.75印张　279千字
版　　次	2015年8月第1版　2015年8月第1次印刷
定　　价	**42.00**元

凡购买我社图书，如有缺页、倒页、脱页的，本社发行部负责调换

版权所有·侵权必究

前　言

随着社会发展和科技进步，计算机软件更加广泛地应用于人类生产、生活的各个领域，人们对软件的依赖程度已经空前紧密。然而，由于软件的复杂性，软件中存在错误是难以避免的，软件错误常引发软件异常，进而造成严重的危害。近年来，软件质量保证已经逐渐引起人们重视，人们认识到软件质量就是软件生命，软件可靠性是软件最重要的品质。

本书第1章主要描述了“软件”的基本概念和特点，从软件产品本身和软件的开发过程两个方面进行分析，引导读者从软件工程的视角认识计算机软件的特性。从软件工程的视角，软件具有产品特性和过程特性，软件的形成和使用体现了管理和技术两方面的内容，软件的质量取决于软件的开发过程和开发方法。第2章介绍了软件质量的含义和软件质量标准，从软件工程的视角论述了软件质量工程最核心的工作——软件质量保证，介绍了软件质量保证活动的组织与管理，同时介绍了软件质量保证的常用技术与方法。第3章介绍了软件测试技术，软件测试目的是发现软件缺陷，是保证软件质量的主要手段。第4章介绍了软件度量，通过测量软件的属性值，获取并定量描述软件质量，软件的度量是评估软件好坏的重要依据。第5章详细介绍了软件不可靠的根源——软件缺陷，分析了软件缺陷的产生原因和软件失效机理，概述了软件缺陷分析技术和方法。依据经验说明了软件全生命周期各阶段的活动中人的心智错误产生了软件缺陷，并指出软件需求是产生软件缺陷的“源头”。第6章提出了一种面向软件可靠性的需求模型，即ROB模型。需求是所有软件开发活动的源头，ROB需求建模方法易理解、便于沟通，支持需求变化，保证需求的一致，从源头上为软件可靠性提供保障。第7章阐述了软件可靠性的统计概念，提出软件可靠性定义建立在数学基础上。第8章主要介绍软件可靠性工程各项活动，以及这些工程活动的管理、组织、技术和方法。在第9章，重点介绍了软件可靠性评测方法和软件可靠性模型的主要研究成果。软件可靠性研究的焦点是软件可靠性的评测方法，软件可靠性评测的核心是软件可靠性模型。通过介绍当前最新的理论和研究，书中对在软件可靠性模型引入智能计算方法等工作做了介绍和展望。

本书引导读者从软件本质特性出发，从不同视角认识软件可靠性保障技术和措施。软件可靠性以及软件其他质量属性，取决于软件全生命周期各阶段的开发行为和工程活动，软件可靠性保障需要一套系统的工程技术、方法和过程管理。软件可靠性工程是为使系统满足可靠性要求而进行的一系列软件工程活动，主要包括需求的一致性建模和软件可靠性要求的确定、软件的可靠性分析与设计和软件可靠性的预测与分析、软件可靠性测试与验证和软件可靠性评测。

为保证软件可靠性，需要在整个软件开发过程中选取不同的时机进行可靠性评价。通常在软件设计、编码和测试阶段，甚至软件交付使用之后，根据不同的目的均可进行可靠性评估，但是适用的可靠性评估方法或者软件可靠性模型通常是不同的。目前，在软件可靠性领域，软件可靠性模型是研究的核心，本书引入了当前最新的研究和争论，并结合实际的软件工程实践讨论评测性能的优劣。

本书旨在从软件工程的角度讨论提高软件可靠性的方法，通过在软件全生命周期实施软件可靠性工程，以期形成一整套系统的软件可靠性评测方法，引导读者认识软件的可靠性，并对软件可靠性保障技术形成完整的认识。

郭新峰

2014 年 12 月

试读结束：需要全本请在线购买： www.ertongbook.com

目　录

任何研究，首先必须清楚地描述和理解课题，包括课题中包含的各种概念和问题，软件可靠性理论及评测方法是近年来逐渐被人们重视的课题，围绕什么是软件、什么是可靠性等问题展开，课题研究对象是软件，下面首先从软件的概念开始讨论。

第1章 软 件 概 述

1.1 软件的概念

计算机系统包括计算机硬件系统与计算机软件系统，软件是硬件的功能扩充，硬件是软件的运行基础。计算机系统的功能由软件的执行完成，软件是社会发展的关键技术。美国国家关键技术委员会将软件列为六大关键技术之一；欧洲共同体将软件和信息处理列为关键技术；我国把IT产业列为优先发展的战略性产业。

软件是人类智慧的结晶，其本质是逻辑系统，是知识性的产品集合。软件用来模拟和解决实际问题，是对物理世界的一种抽象，或者是某种物理形态的虚拟化，由于现实世界各种事物和现象的复杂性，软件是抽象的复杂系统。

1.1.1 软件的定义

各种论著关于软件一般性的认识和科学、规范的定义为

$$\text{软件}=\text{程序}+\text{文档}$$

而

$$\text{程序}=\text{算法}+\text{数据结构}$$

大多学者由这种观点给出软件的解释，软件包括：

①能够完成预定功能和性能的、可执行的程序。

②使程序能够适当操作信息的数据结构。

③与程序操作和使用相关的描述文档。

即

$$\text{软件}=\text{程序}+\text{数据}+\text{文档}$$

程序是为解决特定问题而用计算机语言编写的命令序列集，是反映具体逻辑的代码形式。软件包括具有某种体系结构的一定规模的可执行的程序，以硬复制和电子形式存在的文档，以及包含有数字、文本、图形、视频、音频等不同形式的数据。

在当今所处的网络信息时代，随着人们对软件相关概念的深入理解，有些学者认为，上述定义中应增加一项内容，即服务。由此，软件定义为

$$\text{软件}=\text{程序}+\text{数据（库）}+\text{文档}+\text{服务}$$

从用户的角度，软件按照用户期望，针对用户的使用方式，提供“正常”服务的功

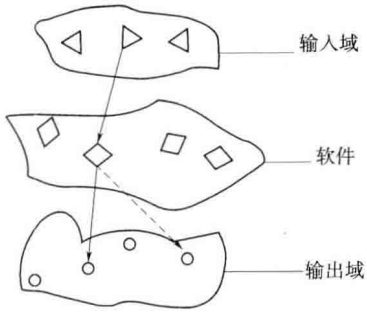

图 1-1 软件看作输入域到输出域的函数

能。因此，软件本质上可看作是一个从输入空间映射到输出空间的函数，输入空间是若干输入状态的集合，输出空间则是若干输出状态的集合，特定的输入状态由软件映射为特定的输出，如图 1-1 所示。

软件的定义很难精确地描述出来，随着人们对软件认识的不断提高，对“什么是软件”这一问题的回答也更加深刻。陆汝钤院士曾引述马希文教授的话，“软件是人类知识的结晶”，并提出：“硬件是逻辑运算，软件则执行逻辑，而‘知件’即领域知识。” Howard Baetjer, Jr. (1998) 对软件的描述是：“软件同其他资产一样，是知识的具体体现，而软件开发是一个社会学习的过程。”

1.1.2 软件的特点

软件在社会发展中起着重要的作用，应用广泛。由于社会进步与科技发展的需要，软件应用领域空前广泛。在计算机系统甚至嵌入式系统中，软件功能所支持的范围日渐增长，人们对软件的依赖程度越来越强烈。

软件的特征很难简单地概括，软件的复杂性使软件缺陷不可避免，为下一步分析讨论软件可靠性，这里从软件多方面特点讨论软件复杂性。

软件是一种特殊的产品，其目的是为用户提供服务，满足用户需求。软件是人类智慧的结晶，是复杂的逻辑系统。与一般的产品不同，软件不是有形的，而是看不见、摸不着的逻辑系统，自由度大；软件是脑力劳动的结晶，软件是有价值的，软件具有质量特性和功能特性。软件是被开发或设计出来的，而不是传统意义上被“制造”或“生产”出来的；软件不会“磨损”或者“老化”，软件可以多次复制、重复使用；软件持续运行会引起资源（如内存、CPU、硬盘、带宽等）消耗和争夺，可导致性能下降，甚至崩溃。软件规模大、复杂、标准化差；软件开发周期长，需求变化快；软件具有时效性，新的软件需求和新的软件技术易导致软件过时；软件生命周期中参与人员多、流动性大；软件具有演化性，软件演化可引起软件可靠性、性能和可维护性等特性的“退化”；软件载体等成本低、复制简单，但需投入大量的、高技术人力，因此，软件开发、维护和服务的成本很高。

软件工程的复杂性使软件具有相当大的复杂程度。软件是经过复杂的生产过程形成的产品，软件开发过程是人类脑力劳动的转化过程，需要经过一系列复杂的过程。从可行性分析、调研开始，经过需求获取、需求分析、需求确认等活动，对客观事物在计算机中进行抽象、建模，根据需求进行设计、编码实现，一直到软件交付并测试、维护等复杂过程。软件开发过程有阶段性，而各阶段无时界；软件开发过程各项活动均可能需要修改，即使是需求也可能发生变化，而需求的变化可能造成一系列更为复杂的变化。

软件运行剖面是多样的，软件的使用具有复杂性。软件应用于不同领域，其功能、用途、使用规程等也不同。软件由用户使用，由于用户的个性、习惯等不同，即使是相同软件，可能的使用路径也是千差万别。软件使用的复杂性更加剧了软件的复杂程度。

由于软件的复杂性和人类认识问题的局限性，软件错误不可避免，而及早地发现并消除所有这些错误是一个至今未解决的难题。软件错误是在软件开发过程中产生的。随着应用领域日益广泛，软件规模越来越大，复杂程度越来越高，软件错误越来越多，也越来越难以察觉。人们逐渐认识到，软件可靠只是相对的，软件不可靠才是绝对的。软件错误具有隐蔽性，潜藏在软件中，在特定软件运行剖面被激活，表现为软件使用过程中的运行异常。软件运行异常造成的危害也越来越严重。实际应用中，软件需经历逐步完善的过程。软件投入运行初期常存在隐藏的错误，具有较高的故障率，随着使用过程中不断发现错误并改正，软件逐渐完善，故障率逐渐降低。软件维护的周期长，维护软件不能简单地替换或使用冗余技术，而是通过开发补丁程序不断地解决问题，逐渐提高软件系统的稳定性和可靠性。

1.2 软件工程

1.2.1 软件工程概述

自1968年第一届NATO会议提出软件工程概念以来，已形成的一系列理论、方法和工具等，指导并规范了软件开发和维护的全生命周期的各种活动。通过应用于实践，解决了软件开发过程中的若干问题，在很大程度上提高了软件开发效率和软件质量。

软件工程是运用工程学的基本原理和方法来组织和管理软件生产的一项工程。软件工程的目的是在规定时间内，高效率、低成本地开发出符合用户所要求的高质量的软件产品。

软件工程的要素包括：①软件的用户和市场、需求与维护；②人力、物力和财力等资源的有效利用和协同；③软件开发过程的管理与控制；④软件开发、实现和维护的方法和技术；⑤对相关人员的组织与培训；⑥自动化的工具与环境；⑦软件工程的标准与规划，形式化描述和交流方式；⑧软件质量的度量与控制。其中最基本的三要素是方法、工具和过程。软件工程方法包括项目计划与估算、软件系统需求分析、数据结构设计、系统总体结构的设计、算法过程的设计、编码、测试及维护等方法。自动/半自动的软件工具、开发机器和软件工程数据库组合起来形成了软件工程环境。软件工程的过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理以及软件开发各个阶段完成的里程碑。软件工程以软件质量保证为基础，软件过程的改进导致了更加成熟的软件工程方法的不断出现。

随着社会发展，人们对软件开发效率和软件质量不断提出更高要求，由于软件的复杂性进一步增强，软件的演化性要求进一步强烈，软件的逻辑复杂性、不可见性等固有特性，使软件开发效率及软件质量仍然不能满足需要，软件工程中存在的许多问题仍然迫切需要研究和解决。

1.2.2 软件过程

随着软件动态开放环境日渐复杂，软件越来越复杂且规模不断变大，迫切需要采用工

程的方法管控软件开发过程。模型即对事物的一种抽象，去掉与问题无关的、非本质的内容，更简明认识和易于把握事物本质。软件开发过程模型化的目的就是为了深刻理解和把握软件工程。

典型软件开发模型包括瀑布模型、螺旋模型、原型法、基于构件的开发、敏捷开发（快速反馈、快速迭代、弱化文档、增量开发）。

1.2.2.1 软件生命周期

软件生命周期（Systems Development Life Cycle，SDLC）是指从软件概念的提出直到软件失去价值而报废的整个过程和时期。为描述软件开发及维护过程全生命周期的各种活动，人们建立了抽象的软件开发模型，即软件生命周期模型。

瀑布模型是软件开发过程的基本模型，现在已被普遍接受，是对软件开发过程最为直观的描述。其核心思想是：按工序将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护等阶段。要求软件开发过程按照工序先后相互衔接，逐级展开，如图1-2所示。瀑布模型中软件开发、维护的各项活动按线性方式进行，以文档为驱动，某一阶段的活动依赖于上一阶段的工作结果，当前阶段的工作结果验证并审核通过后才进入下一阶段。瀑布模型的特点概括为建立里程碑、文档驱动和无迭代。按照瀑布模型现行过程，软件测试可以分为单元测试、集成测试、系统测试。瀑布模型在实践中过于理想化，实际应用中受到很大的制约。

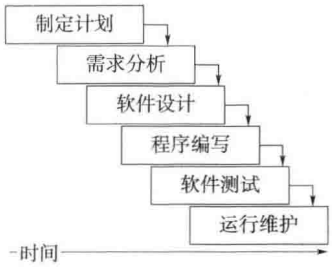

图1-2 瀑布模型

1.2.2.2 原型模型

原型模型在初步获取、分析需求后，快速开发出基本的原型系统，然后基于原型进一步确认、获取和分析需求，不断修改原型直到需求确认后，进行后续的工作，即详细地分析、设计和编码，最终形成客户满意的产品。原型模型的主要思想基于“样品”：根据初步需求或已有的不完善软件系统，通过对“样品”不断改进，使产品满足用户需要。原型模型有利于开发人员与用户沟通，强调用户的参与，采用逐步求精的方法完善原型，使得原型能够“快速”开发，避免了像瀑布模型一样在冗长的开发过程中难以对用户的反馈做出快速的响应。相对瀑布模型而言，原型模型更符合人们开发软件的习惯，但是由于用户缺乏软件过程管理系统方法的认识，用户通过原型提出和定义需求之后，开发人员必须充分考虑软件质量和可维护性等方面进行后续工作。

1.2.2.3 RAD模型

RAD（Rap Application Development）模型，即快速应用开发模型，由于其模型结构图形看起来像英文字母“V”，故也称V模型，如图1-3所示。RAD模型基于构件的开发方法可以缩短开发周期，提高开发速度。RAD模型包含以下几个开发阶段：

（1）业务建模。业务活动信息流的模型化，模型描述了什么信息驱动业务流程，业务流程生成什么信息，这些信息由谁生成、由谁处理、流向何处等问题。

（2）数据建模。业务所需数据对象的模型化，对业务建模定义的信息流进一步细化，形成支持业务的数据对象及描述这些对象的属性及对象间的关系。

（3）处理建模。业务功能数据流的模型化，将数据对象转换为完成一个业务功能所需

试读结束：需要全本请在线购买： www.ertongbook.com

的信息流，定义处理描述对数据对象的获取以及增、删、改。

（4）应用生成。基于第四代4GL技术，使用自动化工具，创建可复用构件。

（5）测试及反复。对构件及其所有接口进行测试，通过复用已经通过测试的构件来创建软件。

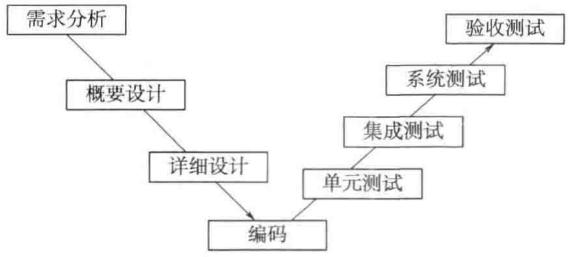

图1-3　RAD（V）模型

为进一步提高软件开发效率，RAD改进型模型将“编码”与“单元测试”水平对齐，如图1-4所示。从水平方向在分析与设计活动同时，进行相应的验证与测试活动或准备工作，从竖直方向分为软件外部工作与软件内部工作两个部分。模块化是形成构件的基础，因此RAD模型更适合信息系统开发，而不适合高性能、技术风险高或不易模块化的系统开发。。

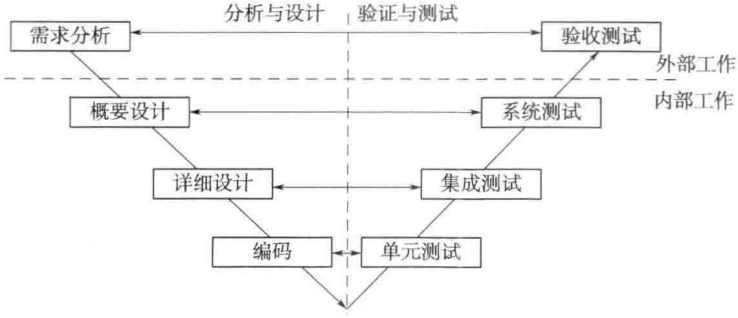

图1-4　改进的RAD模型及其解释

1.2.2.4　演化模型

演化模型的思想是渐进开发、逐步完善。演化模型在实际开发过程中按阶段进行。

1. 增量模型

软件开发按照软件功能分为多个阶段进行，先开发主要功能或用户最需要的功能，然后随着时间推进，不断增加新的辅助功能或次要功能，最终开发出一个功能强大完善、高质量、性能稳定的产品。

2. 迭代模型

软件开发按照细化程度分为多个阶段进行，先建立产品的整个框架，在系统初期就已具有全部功能，然后随着时间推进，不断细化已有的功能或完善已有功能，这个过程好像是一个迭代的过程，最终实现一个功能强大完善、高质量、性能稳定的产品。

3. 螺旋模型

螺旋模型将原型实现的迭代、增量特征结合起来，使软件开发成为一系列的增量发

布，在每一个迭代中形成更完善的软件版本。螺旋模型分为若干框架活动，称为任务区域。图 1-5 形象地描述了包含 4 个任务区域的螺旋模型：①系统目标受预算、时间等条件的限制，作出取舍；②风险评估与管理，确定实施方案；③开发和测试，包括需求分析、概要设计、详细设计、编码、单元测试、系统测试和验证测试等；④分析与计划，定义资源、进度及其他任务。

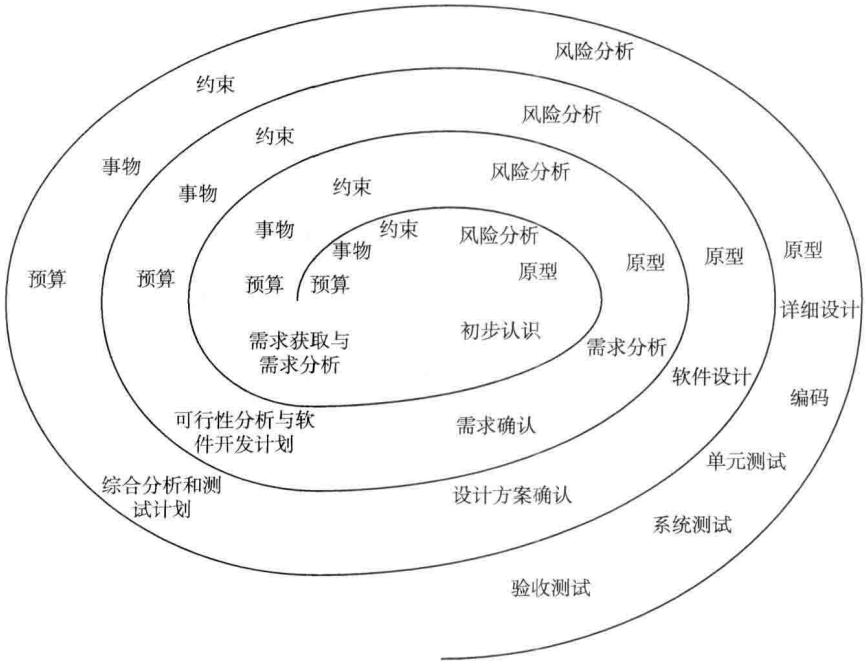

图 1-5　螺旋模型

螺旋模型指出，在项目定义、需求、设计等阶段均存在风险，需要重点考虑，并通过多次迭代的原型主动诱发风险。螺旋模型基于“判定计划—风险分析—实施工程—客户评估”过程，其特点概括为风险驱动、过程迭代。

1.2.2.5　标准软件开发过程与文档

标准的软件开发过程分为计划期（项目计划、需求分析）、开发期（概要设计、详细设计、编码实现、单元测试、集成测试）、维护期（维护）。相关软件文档主要包括可行性分析、软件开发计划书、需求规格说明书、设计说明、测试计划、测试报告、用户使用手册。

1. 可行性与计划研究阶段

（1）可行性研究报告。在可行性研究与计划阶段内，要确定该软件的开发目标和总的要求，要进行可行性分析、投资和收益分析、制订开发计划，并完成应编制的文件。

（2）项目开发计划。编制项目开发计划的目的是用文件形式把开发过程中各项工作的负责人员、开发进度、所需经费预算、所需软硬件条件等问题记录下来，以便根据计划开展和检查工作。

2. 需求分析阶段

（1）软件需求说明书。软件需求说明书的编制是为了使用户和软件开发者双方对软件达成一致的理解，包括功能和性能等各方面的规定。

（2）数据要求说明书。数据要求说明书是关于数据的描述和采集要求的技术信息。

（3）初步的用户手册。使用通俗、易理解的语言，充分地描述系统所具有的功能及基本使用方法。使用户或潜在用户了解软件用途，并能够确定使用的方法。

3. 设计阶段

（1）概要设计说明书。其包括系统的基本处理流程、系统的组织结构、模块划分、功能分配、接口设计、运行设计、数据结构设计和出错处理设计等的说明，为程序的详细设计提供基础。

（2）详细设计说明书。对软件各个层次中每一个程序或子程序、模块的设计进行说明。

（3）数据库设计说明书。对数据库的所有标识、逻辑结构和物理结构作出具体的设计规定。

（4）测试计划初稿。它指组装测试和确认测试的测试计划，包括对每项测试活动的内容、进度安排、设计考虑、测试数据的整理方法及评价准则。

4. 实现阶段

（1）模块开发卷宗。在模块开发过程中逐步编写，每完成一个模块或一组密切相关的模块的复审时编写一份，应该把所有的模块开发卷宗汇集在一起。编写的目的是记录和汇总低层次开发的进度和结果，以便于对整个模块开发工作的管理和复审，并为将来的维护提供非常有用的技术信息。

（2）用户手册和操作手册。操作手册的编制是为了向操作人员提供该软件每一个运行的具体过程和有关知识，包括操作方法的细节。

本阶段最终完成测试计划的终稿。

5. 测试阶段

（1）测试分析报告。分析并以文件形式记录组装测试和确认测试的结果。

（2）项目开发总结报告。总结项目开发工作经验，说明实际取得的开发结果以及对整个开发工作的各个方面的评价。

本阶段完成模块开发卷宗。

6. 运行与维护阶段

开发进度月报：及时向有关管理部门汇报项目开发的进展和情况，以便及时发现和处理开发过程中出现的问题。一般以项目组为单位每月编写。对于一项软件而言，有些文件的编写工作可能要在若干个阶段中持续进行。

1.2.3　软件配置管理

软件配置管理（Software Configuration Management，SCM）基于构件等先进思想和技术，为软件研发提供了一套管理办法和活动原则，支持软件全生命周期的资源管理需求，确保软件工作产品的完整性、可追溯性。软件变更的原因多样，而软件变更常使软件处于混乱状态。软件配置管理的目标是标识变更、控制变更、确保变更正确实现，并向其他有关人员报告变更，通过使用标识、组织和控制修改的技术，使错误降为最小并更有效地提高效率。配置管理系统支持对软件的配置标识、变更控制、状态纪实、配置审核、产品发布管理等功能，实现核心知识产权的积累和开发成果的复用。配置管理系统支持建立

和维护项目储存库，包括开发库、受控库、产品库。

软件配置管理主要包括3个方面的内容，即版本控制、变更控制和过程支持。关键活动包括配置项、工作空间管理、版本控制、变更控制、状态报告、配置审计等。

软件配置管理配置项包括软件全生命周期中的要素：代码、文档、工具等。软件基线是软件文档或源代码（或其他产出物）的一个稳定版本，是项目储存库中每一道工序的标准，是进一步开发的基础。每一个新的基线或变更后的基线结合上次基线建立以来的工作存入工作区，进一步的开发工作采用新的基线。主要的三类基线包括：①功能基线，是通过正式评审和批准，并由项目供、需双方签字同意的对开发软件系统的规格说明，是最初批准的功能配置标识；②分配基线，是经过正式评审和批准的软件需求规格说明，是最初批准的分配配置标识；③产品基线，是经过正式评审和批准的有关软件产品的全部配置项的规格说明，是最初批准的产品配置标识。

1.2.4 UML及类图

统一建模语言（Unified Modeling Language，UML）是通用的、可视化的建模语言。UML不是程序设计语言，不是工具或知识库的规格说明，而是一种建模语言、方法和标准，是一套描述系统和需求的通用的图形语言和符号体系。

目前常见的UML建模工具有Rational Rose、Microsoft Visio、Enterprise Architect、Visual UML等。UML中模型工具是图形表示，可分为：①结构性图形，包括静态图（如类图、对象图、包图）、实现图（如组件图、部署图）、剖面图和复合结构图；②行为式图形，包括活动图、状态图和用例图；③交互式图形，包括通信图、交互概述图、时序图和时间图。这几种基本图形也可以分为：①用例视图（如用例图）；②逻辑视图，包括静态结构图（如类图、对象图），动态行为图（如状态图、顺序图、活动图、合作图）；③构件视图（如构件图）；④并发视图，包括动态图（如状态图、合作图、活动图）、实现图（如构件图、配置图）；⑤部署视图，如部署图。

用例图中的要素包括系统、执行者、用例、关联（包含、扩展）、执行者间的泛化、文档（用例编号、用例名称、相关角色）。类图中的要素包括关联（关联、组合、聚合、依赖、泛化）、属性、操作、约束规则。状态图是对类描述的补充，用以说明某类对象所有可能的状态以及哪些事件将导致状态的迁移。状态图中的要素包括状态（初态、终态、中间状态、复合状态、并发状态）、迁移（触发条件、守护条件、效果）、注释、历史标识、条件判断标识。顺序图中的要素包括对象（对象、生命线、激活的生命线）、消息（简单、同步、异步、创建、销毁、返回）、守护条件、注释。活动图中的要素包括活动（组合活动、特殊活动、条件判断、同步条件）、对象、起点和终点、泳道和时标。

UML模型检查要点主要是：①用例图中用例之间关联正确，角色和用例之间关联正确，用例之间不要重叠，检查用例文档内容是否齐全，前置和后置条件是否满足，分支是否封闭；②类图中检查类中的属性是否消除冗余，关联是否有冲突、是否不存在环，不要有get和set，通过实例化来验证类图是否正确；③状态图中检查迁移条件（触发条件、守护条件、效果），检查状态中的动作，通过实例化一个对象进行检查；④活动图中检查活动迁移，检查并发的分支和汇聚；⑤顺序图和类图中对象能否处理发来的消息，参数定义

是否正确，守护条件是否满足；⑥状态图和顺序图中状态链和顺序图是否一致。

1.2.5　软件开发方法

在软件开发的过程中，软件开发方法关系到软件开发的成败。软件开发方法是软件开发所遵循的方法和步骤，期望通过这些方法保证软件/文档满足要求。最为基本及影响最为深远的软件开发方法是结构化方法和面向对象的方法。下面介绍几种具有代表性的软件开发方法。

1.2.5.1　Parnas 方法

1972 年，由于软件在可维护性和可靠性方面存在严重问题，D. Parnas 针对这两个问题提出 Parnas 方法。

(1) 信息隐蔽原则。在概要设计时列出将来可能发生变化的因素，并在模块划分时将这些因素放到某些模块内部，以便将来在这些因素变化需修改软件时，仅需修改这些模块而不需修改其他模块。这样做的好处是避免错误的蔓延，提高软件的可维护性，改善软件的可靠性。

(2) 软件设计时预先对可能出现的故障采取措施。

1.2.5.2　SASD 方法

1978 年，E. Yourdon 和 L. L. Constantine 提出了结构化方法，即 SASD 方法。1979 年 Tom De Marco 对此方法作了进一步的完善。结构是指软件系统内各组成要素之间的相互联系、相互作用的框架。结构化方法主要面向功能和数据流，是传统的软件开发方法。物理数据流关注系统中的物理实体，以及具体的文档、报告和其他输入输出硬复制等；物理数据流图说明系统中数据的加工和存储细节，是构造和实现系统的技术性蓝图。逻辑数据流关注系统与参与者的活动，以及支撑这些活动必需的控制资源和数据基础；逻辑数据流图说明系统中需要的加工和存储等活动有哪些。

数据流图描述数据流动、存储和加工的逻辑关系。结构化开发方法强调系统结构/软件结构的合理性，把数据流图映射为软件结构图，而数据流图的类型决定了映射方法。

1. 两类数据流图

数据流图通常将系统物理模型（如“业务流程图”）进行功能建模，转化为系统的“逻辑模型”，将数据和处理进行逐层分解为若干层次的数据流图。

结构化方法更适合于信息系统，信息系统的数据流图有两种类型：变换型数据流图和事务型数据流图。数据流图三要素包括输入、处理和输出。数据流图以处理为中心，为加以区别，变换型数据流图中处理称为数据加工或中心变换；事务型数据流图中处理称为事务或事务处理中心。

变换型数据流图中，信息以外部形式沿物理输入通道进入系统，变换为内部形式并沿输入通道到达变换中心，通过变换中心对数据加工处理后，沿输出通道最终变换为外部形式并离开系统。变换型数据流图呈线性结构，如图 1-6 所示。

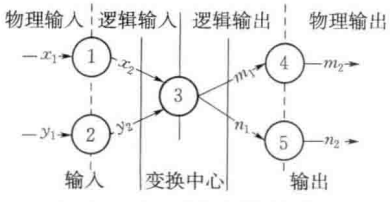

图 1-6　变换型数据流图示例

事务型数据流图中，信息沿输入到达某个事务处理

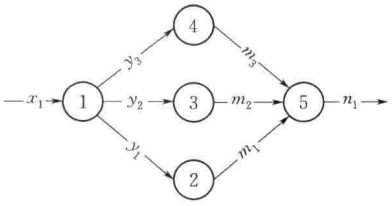

图 1-7　事务型数据流图示例

中心，事务处理中心根据输入数据类型，从若干个动作序列中选出一个来执行，即选择某类事务进行处理。事务型数据流图呈束状结构。如图 1-7 所示，即一束数据流平行流入或流出，可能同时有几个事务要求处理。

2. 结构化设计方法

结构化设计方法的主要思想是自顶向下、逐步求精、模块化。按照数据流图的不同类型分别通过“变换分析”和“事务分析”技术，最终得到一个满足数据流图所表示的系统模块初始结构图，即系统的物理模型。

(1) 结构化设计的步骤。

1) 根据数据流图类型确定软件结构基本框架。

对于变换型数据流图，确定逻辑输入和逻辑输出的边界，找出变换中心，映射为变换结构的顶层和第一层。

如果是事务型，确定事务中心和活动路径，映射为事务结构的顶层和第一层，建立软件结构的基本框架。

2) 分解上层模块，设计中下层模块结构。

3) 根据软件结构设计准则对软件结构求精并改进。

4) 导出接口描述和全程数据结构。

5) 复审。

如果有错，转入修改完善；否则进入下一阶段详细设计。

(2) 变换型结构设计方法和步骤。

1) 分析底层逻辑，确定输入和输出的边界，识别出变换中心。

a. 从物理输入端开始一步步向系统中心移动，直到识别出第一个不能看作系统外部输入的数据流为止，这一数据流离物理输入端最远，即作为逻辑输入。

b. 从物理输出端开始逆数据流方向，向系统中心移动，直到识别出第一个不能看作系统物理输出的数据流为止，作为逻辑输出。

c. 在逻辑输入与逻辑输出间的处理被识别为顶层变换中心或顶层处理。顶层处理常在数据流的汇合处，并包括多个处理。为便于处理，可以先不考虑数据流图中可能的支流。

2) 设计软件结构的顶层和第一层。

顶层处理作为主模块位于软件结构最顶层，其功能即整个软件的功能。主模块中，输入模块协调对所有输入数据的接受，变换中心管理对内部形式的数据的所有操作，输出模块协调输出信息的产生过程。主模块设计完成后，开始设计第一层模块，其步骤如下：

a. 为每一个逻辑输入添加一个输入模块，其功能是向主模块提供数据。

b. 为每一个逻辑输出添加一个输出模块，其功能是把主模块提供的数据输出。

c. 为顶层处理添加一个变换模块，其功能是把逻辑输入变换成逻辑输出。

3) 设计中、下层模块。

在顶层模块和第一层模块的基础上，可以按照自顶向下、逐步求精的思想来画出以下各层模型。输入模块也需要一个数据来源，且其功能是为调用它的模块提供数据，所以应

试读结束：需要全本请在线购买：　www.ertongbook.com