软　　件　　工　　程

软件工程

王家华　编著

NEUPRESS
东北大学出版社

软件工程

王家华　编著

东北大学出版社

图书在版编目（CIP）数据

软件工程/王家华编著. —沈阳：东北大学出版社，2001.3
ISBN 7-81054-492-6

Ⅰ.软… Ⅱ.王… Ⅲ.软件工程-高等学校-教材 Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字（2000）第 87333 号

内 容 简 介

本书是根据编者多年来从事软件工程研究与教学经验，在参阅了大量国内外最新资料的基础上编写而成的，包括了传统的结构分析方法和正在发展的面向对象的开发方法学，涵盖了从可行性论证直到软件维护各阶段的内容，主要有：可行性论证技术；半形式化和形式化规范技术；软件项目质量、配置管理及软件项目的特征量；模块化原理；系统结构优化设计；输入/出接口优化设计；结构程序设计；测试案例设计；面向对象原理；面向对象的建模；OO 系统设计与测试；以及某些开发阶段特征量的概念、计算和应用。本书可作为本科生、研究生软件工程课的教材。

（沈阳市和平区文化路 3 号巷 11 号　　邮政编码 110006）
电话：(024)23890881　　传真：(024)23892538
网址：http://www.neupress.com　E-mail:neuph@neupress.com

东北大学印刷厂印刷	新华书店经销
开本：787mm×1092mm　1/16　　字数：478 千字	印张：19.625
印数：1～3000 册	
2001 年 3 月第 1 版	2001 年 3 月第 1 次印刷
责任编辑：高　媛	责任校对：米　戎
封面设计：唐敏智	责任出版：秦　力

定价：30.00 元

前　言

软件工程是在20世纪60年代创立的一门工程学，目的在于解决“软件危机”或者称为“软件困境”的问题。经过数十年的努力，人们总结了大量的经验和发现了许多规律，提出了软件开发及其管理的方法和技术。这些成为软件工程的主要内容。

鉴于软件工程是一种特殊的工程，它不像机械工程、建筑工程那样具有明显的可测性和可计算性，因此长期以来在软件工程的可测性和可计算性及抽象理论方面的研究所取得的进展很少，从而导致了软件工程的有关资料中可计算性和形式化的内容较少，一般主要是技术性的。

有很多长期从事软件工程教学工作的人都有一种同感：目前的软件工程教材中包含的可计算和形式化的内容太少，叙述性的内容太多，因此有关的习题要么是难以完成的，如作一个项目的可行性论证，和写一个用户需求报告等；要么是死记硬背书本内容的。在这种情况下，学生不知如何学，老师不知怎么考，最终的结果是学生对于软件工程的整体知识素质难以有所提高，更遑论熟练地掌握和应用这门知识技术。

为改变这种现状，本书尽量结合计算机教学实践编写，并有相当部分的计算和半形式化的内容以及适量的形式化内容，从而更加符合软件开发实际工作的需要。

本教材强调了下列内容：

在项目管理技术内容中介绍了使用代码行和功能点(FP)技术估算程序规模，使COCOMO模型计算项目工作量、开发时间和进度以及确定开发队伍结构更加合理。

在可行性论证内容中增加了最成本有效解和最大可行解的概念和应用。

在用户需求和规范中介绍了有限状态机，Petri网等半形式化工具、数据流程图的形式化定义、代数公理和Z规范技术。

在总体设计中讨论了软件系统设计及其优化设计方法，大型软件系统的设计与集成以及总体设计的特征量的概念与应用。

在详细设计中讨论了结构化设计的概念与技术，人—机接口的优化设计以详细设计软件特征量。

在集成测试章节中详细介绍了测试案例设计和集成测试的特征量计算。

在面向对象方法学中介绍了面向对象原理，在建模方法中讨论了一般的OO建模方法，并增加了高级对象、高级状态和高级跃迁建模设备等内容。

此外每一章中都附有和这些内容相关的可操作的习题。

希望通过本书，广大读者能够掌握这些理论与方法并能正确应用到软件工程实践中去。与此同时，通过加强理论学习与训练，能提高创新能力。

尽管本书努力吸取有关著作中的优秀内容，但是限于编者水平、能力及时间的限制，书中定会存在某些错误和不当之处，希望广大读者不吝赐教。

编　者

2000年11月

目　录

1 软件工程引论

学习软件工程是为了掌握软件系统的开发技术和如何着手编写一个大型程序。

有一个例子可以说明软件工程知识在系统开发中的重要作用:20 世纪 70 年代的最后一年,有两个学生接受了两个不同的毕业设计任务。甲学生的题目是实现一个分布查询系统的原型,所涉及的环境有 DOS 系统、关系数据库管理系统以及一个自行设计的通讯程序;乙同学的题目是"远程文件存取",所需要的环境为 DOS 系统以及一个自行设计的通讯程序。他们几乎学习了全部主要的计算机课程,包括计算机原理、汇编语言程序设计、数据结构、编译方法、高级语言程序设计以及数据库引论,但是惟独没有学习过软件工程。在甲乙两学生之间的知识背景差异是:甲学生自学了结构程序设计技术,接触了一点点软件工程知识,主要是自顶向下的设计方法和结构程序设计方法。他们同时开始了他们的设计工作。甲学生首先进行系统结构设计,确定了组成他的系统近 60 个模块中每一模块的名字、功能、模块间的调用关系、他的系统和 DOS 系统以及本地关系数据库之间的相互作用的方式,然后为每个主要模块绘制程序流程图。就在这个时候甲发现乙已经编了近 2000 条语句,几乎完成了程序量的一半。甲大大地落后于乙的编程进度。但甲继续实现他的系统,按程序流程图为每个模块编程,每个模块大约为 60 条汇编指令,最大的模块不超过 100 条指令。在他们开始调试程序时,整个事情发生了戏剧性的变化,甲的程序模块很快地一个个调试完毕并组装成系统。而乙此时连一个模块也没有调试出来,原因是他的 3000 多条语句仅由两个模块组成,一个模块居然包含 1000 多条语句! 程序中,大量地使用了 GO TO(Jump 指令)语句! 直到开发工作结束,他也没能调出一个模块! 为了答辩,他补了一个程序流程图,逻辑复杂得竟像一个蜘蛛网! 当有人问他:你的程序调出来没有,乙的回答是:"我的程序已经装入内存"。

由此可见,即使在一个很小的系统开发中,软件工程知识也是非常重要的。下面我们将就此进行详细的阐述与讨论。

1.1 软件困境

在 20 世纪 70 年代初期,极少有人对软件有所了解。而在 21 世纪初的今天,大多数职业工作者和许多社会成员都在某种程度上认为自己理解了软件。什么是计算机软件呢? 有人认为一个计算机程序是一个软件。显然这是不全面的。软件应该取下列形式的定义:

软件是:①当执行时能提供需要的功能与性能的计算机程序;②程序能被充分地操作的数据结构;③描述程序使用和操作的文档。为了理解软件,最终理解软件工程,重要地是要理解软件的特征。

1.1.1 软件的特征

研究软件的特征，有助于理解软件生产的困难，进而去克服它们。

(1) 复杂性。

一个软件产品是和计算机系统联系在一起的。人们赋予计算机系统的期望与功能不同于任何其他产品。除了计算机系统以外的任何产品，不过是人们手和脚功能的延伸。例如，一个普通机床，人们利用它将金属材料加工成机器零件。某种意义上说，机床是人们手的功能的延伸，相当于使人的手变得更有力，更锋利，能够将坚硬的金属材料加工成需要形状和尺寸的产品。一个汽车的功能即使再复杂，性能再优越，它不过是人腿功能的延伸和人类负重能力的加强。但是计算机不同，人类希望它能够加强和解放人类的最高级器官——大脑的功能。人们使用计算机监视生产过程，根据生产情况发出调度指令，生产出高质量的产品。实践证明：在这方面计算机做得甚至比人类自身更好。计算机用于事务管理，例如财务管理，可以快速存取大量的数据，更快更准确地计算，将财会人员从繁重的脑力劳动中解放出来。计算机能用于科学计算，求解复杂的超越方程。这类方程靠人类的大脑和手是无法精确求解的。在这方面计算机是人类大脑的延伸。计算机代替人脑的应用例子不胜枚举。一个普通计算机只是一个通用的计算装置，它只能快速执行其指令集合的指令。而这些应用功能的实现是由这些指令构造的计算机程序去操纵有关的数据实现的，即由计算机软件实现。软件要实现的功能是人类大脑的部分功能或某部分功能的加强。因此从本质上决定了和其他产品相比，软件是一种更为复杂的产品。

(2) 难以描述性。

这个特性指的是某些软件算法的难以描述的特性。科学家和工程师在解决问题时，总是采用抽象的模型描述现实世界，使用数学公式准确地表达客观规律。例如航天专家们使用运动学定律描述飞行体的运动；化学家应用化学方程描述化学反应。当计算机应用于这些领域时，软件能够采用这些抽象模型来表达其算法，软件算法具有同样的可描述性。但是当计算机应用于其他领域时，例如建立一个企业的应用系统，计算机软件的算法将是复杂的，很难用一个或一组公式准确地表示它。它往往需要自然语言结合数学公式以及其他表示方法，如逻辑公式等来表示。由于自然语言中存在着二义性，因此用自然语言所表达的算法是很难准确的。另一方面是因为解决这些领域中问题的算法是由领域专家直接或间接提供的。由于人们对客观世界的认识具有不完全性，甚至存在着某些错误认识，因此依赖于领域专家知识的计算机软件很难准确地反映客观世界的规律。

(3) 不可见性。

在生活中，我们所接触到的产品其外形几乎都是可见的。例如一个房屋、一辆汽车，它们的外观是可见的。即使对于设计中的产品，其外形的优劣亦可通过三维模型或者实物造型来观察和判断。对于软件产品则是不同的。首先，由于它由一些计算机程序和其操作的数据以及相应的文档构成，它可能存储在纸介质上，或者磁介质上。我们能观察到的只是介质的形体，而不是软件的形状。如果我们把程序运行中显示给用户的接口定义为软件的外形的话，那么我们只能通过运行程序才可以观察到其外形。因此，对于设计中的软件则是无法观察其外形的。正是由于这种不可见性造成了开发者和客户之间通讯的困难。

(4) 软件的变化性。

改变一个普通产品是不容易的，若把一个建筑物移开原地数米不是一件普通的事情。一辆最高速度为 100km/h 的汽车用过数年后，若要将它的时速增加到 150km/h 是一件非常困难的事。但是对于一个运行了几年的操作系统，重写其中半数的编码，以改进其功能却是很正常的活动。

改变软件比改变运行软件的硬件是更容易的，这就是术语“软件”和“硬件”字面所表达的含意。一个计算机系统的功能是由该系统的软件部分决定的。改变一个计算机系统的功能只能通过改变它的软件部分来实现。因此改变性是软件的固有特性，它是不能被逾越的。软件需要不断加以改变主要是由下述三个原因引起的：①软件是一个现实世界的模型，现实世界变化了，软件必须随之变化；②软件是人们对客观世界认识的反映，人们对客观世界的认识是逐渐加深的，因此相应软件必须改变以反映人们认识上的变化；③一个成功软件的存在周期长于运行它的硬件，当硬件改变时，软件必须要加以改变，以适应新的硬件环境。

1.1.2 软件开发问题

软件产品的固有特性造成软件产品的开发比较困难，存在着长周期、高成本、低质量的问题。

(1) 软件产品开发的长周期与高成本。

现代社会中各行各业都通过应用计算机系统实现自动化，提高了生产率。如生产过程自动化控制、办公自动化、信息处理自动化等等。但令人遗憾的是，计算机软件人员却没有能够从繁重的脑力劳动与手工操作中解放出来，仍然主要靠双手开发软件，不能实现自动化。尽管近些年来，软件生产中计算机的应用有了某种发展，但是远远不能满足实际需求。造成这种状况的原因是软件的复杂性，软件生产需要具体问题的具体分析与处理，因此需要更多人的智力和创造能力。尽管计算机具有某些“智能”，但是没有创造力，它仅仅能按人们预先设计的程序工作。由于软件生产的这种高智力的投入以及手工操作方式，导致了它的长生产周期与高生产成本等问题的出现。

(2) 软件产品生产的低质量。

由于软件产品的难以描述性，因此很难准确地表达用户的需求和精确地表达解决问题的算法，所以产生了所谓的规范错误以及设计错误。实践表明软件产品中的大部分错误属于这两种类型。另外将一个设计转变成对应的编码时，可能产生编码错误。如果在软件开发中不能检测所存在的错误并加以校正，无疑将产生低质量的软件。由于软件的难以描述性和不可见性造成用户和开发者之间的通讯障碍，在极端的情况下，甚至可能开发出完全无用的产品。

(3) 软件开发进度难以控制。

一般说来，由于软件产品的开发周期较长，用户在这个过程中不断加深对其应用环境以及计算机系统的理解，通常是不断地改变对软件的功能要求，这将导致修改规范和设计，因此将增加开发工作量、加长软件的开发周期。软件产品的改变性，反映在软件开发中呈现出开发进度难以控制的现象，经常使成本增加，交货期延长。

软件产品交付之后的主要工作是产品维护。如果软件产品不是利用科学的原理与方法开发的，该软件产品将是难以维护的，这包括缺陷难以定位、错误难以改正、功能难以扩充、编码难以修改等，这些也反映了软件的困境问题。

1.2 软件工程

为了克服软件困境问题，NATO 的研究组于 1967 年使用了术语“软件工程”。而建立软件使用类似于其他工程任务方法的主张是 1968 年 NATO 软件工程会议认可的。该会议的结论是软件工程应该采用已经建立的其他工程的原理和范例解决软件危机问题，即我们称之为软件困境的问题。

对于软件工程，Fritz Bauer 给出了一个定义：“软件工程是建立和应用完善的工程原理以便经济地得到在真实机器上可靠和有效运行的软件。”这个定义为我们提供了讨论的基础，即软件工程是建立和应用关于软件生产的工程原理，但是在技术上涉及得很少，没有提到测量和定量方法的重要性。

IEEE 开发了一个更进一步的定义：“软件工程：①应用系统的有规则的定量的方法开发，使用和维护软件；即应用工程于软件。②研究①中的方法。”在这个定义中明确地提到应用定量的方法。在过去几十年的软件实践中基本是定性研究，很少涉及数量问题。任何一个工程几乎都与测量和数量相关。软件工程也必须和测量紧密地联系起来，并进行定量的研究。一个新的概念“软件的特征量(metrics)”已经被提出来，本章的最后一节将加以介绍。

软件工程的目的在于生产出满足预算、按期交付、用户满意的无缺陷的软件，进而当用户需求改变时，所生产的软件必须易于修改。为了达到这些目标，软件工程师必须掌握广范围的知识，包括技术上和管理上的知识，并且应用于软件开发的始终。

1.2.1 软件的生命周期法

宇宙间任何事物都是运动的，都要经历发生、发展和消亡的过程。一个软件从它的发生到其消亡的过程被称为软件的生命周期。软件的生命周期一般可以划分为 3 个阶段：软件定义、软件开发和软件维护。软件定义可进一步划分为问题定义、可行性论证和用户需求阶段；软件开发可进而划分为：总体设计、详细设计、编码和单元测试与集成测试阶段；维护阶段不再细分。

(1) 问题定义。

在这个阶段中，首先由用户(高层管理者)提出需要解决的问题；此后由系统分析并提出要解决这个问题的系统的目标规模以及完成这个系统的时间、成本等约束条件的书面报告；最后和客户座谈，得到客户的认可。这个报告的篇幅为 1～2 页纸，所用时间将不多于 1 天。

(2) 可行性论证。

可行性论证是任何工程项目的重要环节。任何项目在开始实施之前，必须进行可行性研究。对于一个软件产品项目的可行性论证，通常要对包含此软件的计算机系统进行论证，也就是要对问题定义阶段中所提出的问题进行分析，以决定所提出的问题是否有“可行”的解。所谓可行，首先在技术上要可行，即现有技术能否解决所提出的问题；其次是经济上可行，即用户能否支付实施项目所需的费用，项目建成后能否取得预期的经济效益；最后进行操作上和法律上的可行性研究。综合这些方面的论证，最终决定是否实施该项目。

(3) 用户需求分析。

在这个阶段中，用户将向开发者描述所需要的产品。在开发者看来客户对产品的描述

可能是模糊的、不合理的，甚至是矛盾的。开发者的任务是准确地确定客户所需要的功能；为了实现这些功能，需要哪些处理，需要处理什么数据；需要存储哪些数据文件；得到哪些信息；输入输出数据的格式以及有关处理的具体算法等。此外需要找出为了实现既定功能所需要的约束条件，典型的约束为成本约束和时间约束。例如某项目需要投资 500 万元人民币，并且要求在 18 个月内完成。在这个阶段中我们需要一个工具能够准确地描述用户的需求，并易于和用户交流。数据流程图能够方便地描述用户需要的功能、数据的流动和处理方式等，数据字典能够规定数据的格式和处理的具体算法。这两个设备结合起来，就可以规定用户的具体需求。

在需求阶段的另一个重要任务是要规定系统的生效准则，规定在目标系统建成后验收目标系统的一些原则以及具体的验收方法等。

(4) 总体设计。

在这个阶段中的主要任务是如何得到一个能够实现用户需求的理想的系统结构。所谓理想的结构指的是：①能够实现需求阶段中规定的功能；②系统容易设计和实现；③系统易于维护。在结构设计中应采用模块化原理，把目标系统设计成由一些独立功能的模块组成的形式，模块之间的联系要尽可能地弱。当修改一个模块时，只要保持模块的接口不变，将不影响其他模块。此外为了使独立实现的模块最终能够装配成一个成功的系统，必须仔细地设计各模块的接口。模块的接口指的是和该模块对应的过程的名字以及与其联系的参数等。接口设计的理想工具是系统的结构图，它是在层次图的基础上增加在模块间流动的参数的描述。

(5) 详细设计。

在这个阶段中的任务是进行系统所需要的数据结构设计或者数据库的物理设计(数据库的逻辑设计通常是在总体设计阶段完成)以及模块逻辑的开发。模块逻辑的开发是依据需求分析中建立的数据字典进行。数据字典含有过程的定义，包括过程的算法描述。

在详细设计中，应该使用结构程序设计技术表示模块的逻辑，结构程序有确定的控制结构，使程序逻辑简单清晰，使程序易于理解，便于调试和维修，因而有更高的生产率。

表达模块逻辑最常用的工具有程序流程图和过程描述语言。前者的描述是程序逻辑的形象表示；后者的形式更接近于程序编码。无论使用哪一种方式，它们所表达的程序逻辑必须符合结构程序特点。总体设计和详细设计是一个纯粹的设计过程，它们不涉及具体的程序设计语言。

(6) 编码和单元测试。

本阶段是软件系统的实现过程，要进行模块编码和模块测试。模块测试又称为单元测试。在这个阶段中要根据应用的性质选用合适的程序设计语言。在可能的情况下，应选用适当的第 4 代程序设计语言(4GL)，模块编码是严格按详细设计阶段开发的模块逻辑进行，并使用具有结构程序设计特点的语言进行实现。模块程序设计完成之后，应经过严格的测试。程序测试应该采用科学的方法，主要是黑盒法和白盒法进行测试，从而尽可能多地发现模块中错误，并加以改正。

(7) 集成测试。

把经过彻底测试的模块组装起来形成需要的应用系统。在集成测试中采用黑盒测试方法，主要测试模块接口之间可能存在的错误。集成的方式可采用自顶向下集成或自底向上

集成，或者二者结合的方法。

(8) 维护。

维护阶段开始于软件产品交付给用户之后，主要任务包括改正性维护、适应性维护和完善性维护：改正性维护是改正程序中所遗漏的错误，保证用户程序能正常运行；完善性维护是增加程序的功能以满足用户的进一步需求；适应性维护是当用户改变硬件或软件支持环境时，满足程序需要的修改工作。维护阶段是一个非常重要的阶段，维护工作的好坏直接影响到用户的满意程度，关系到开发者的信誉。

1.2.2 两种软件开发范例

在软件的开发中存在两种范例：一种是结构分析和设计方法；另一种是面向对象的开发方法。这两种范例都遵从生命周期方法。

(1) 结构分析和设计的范例。

这是一种传统的开发方法，自 20 世纪 70 年代末期以来这种方法逐渐完善，并且被人们所广泛接受。这种方法已应用了二三十年，是一种可以信赖的方法，进行了大量成功的实践，但是也有不成功的例子。

这种方法需经过上述的问题定义、需求分析、设计和测试等阶段。用户首先提出需要的功能，开发者用某种工具记录下用户需要的功能，一般使用数据流程图；然后对数据流程图进行细化，并翻译成目标系统的模块层次结构；最后用一种程序设计语言为所有模块设计程序并进行程序调试。支持这种方法学的理论有数据流程图的形式化理论、模块化设计理论和结构程序设计技术。

这种开发范例支持自顶向下的开发方法，在开始时，可以把目标系统看成由一个圆组成，圆的内部有系统的名字。然后对该系统进行细化，把系统看成由一些圆组成，每个圆代表系统的一个功能。然后再对每个功能细化，直到系统的每一个组成部分都是容易实现时为止。这个从抽象到具体的处理过程符合人们的一般思维方法。

这种方法的一个严重缺点是系统一旦建成以后，对用户需求变化的适应能力差。由于这种方法是基于功能分解的方法，用户的功能确定了系统的体系结构。因此当用户需求功能改变，有时需显著地改变系统的层次结构，但这种重大的改变一般不是很容易做到的。而用户需求往往是容易改变的，一方面由于用户对计算机系统逐渐加深了解，会提出进一步需求；另一方面也可能由于用户更深刻地理解了其本身的事务，从而提出更高的要求。这些要求可能导致开发中的软件需要做巨大的调整。

(2) 面向对象的软件开发范例。

这是一个 20 世纪 90 年代兴起的软件开发方法学。在 90 年代初期面向对象的范例看起来是有前景的，仅如此而已。而到 90 年代中期一些专业人士认为面向对象的开发范例优于经典的软件开发方法。

这种方法经由系统分析、设计和试验等阶段，同样遵从软件生产周期的规律。在这种方法中，把软件看成是由一些相互作用的对象组成，每个对象有它的性质和行为(它的性质是指构成该对象的属性，行为是指对这些属性施加的操作)。例如，程序设计中使用的队列是一个对象，其属性为该队列的存储单元；其操作为插一个元素进队列、从队列中消去一个元素和测试该队列是否为空等。我们所讨论的对象都是被动对象，这些操作是对对象本身进

行的，这些操作又称为方法。对象之间的相互作用是通过“消息”实现的。当一个对象需要另一个对象的操作时，前者发送一个消息给后者，后者接到消息后利用消息中规定的方法对其自身进行操作。系统的功能是通过构成该系统的对象的操作实现的。

这种方法的优点是：①系统适应用户需求变化的能力强。当用户需要改变时，只须增加或减少某些对象或改变对象间的相互作用方式，这比结构方法中要改变根模块和绝大多数中间控制模块的逻辑要容易得多。②这种方法的再使用性好，每个对象本身就可能是一个再使用的资源。另外面向对象方法中的继承机制是一种软件再使用方式。

这种方法的缺点是首先缺少一个明确的用户需求分析阶段，没有一个形式化或半形式化的方式规定用户需求，所谓的用户案例方法是用自然语言规定用户的需求，缺少应有的清楚性与可读性。另外这种方法缺少一种模型工具支持从抽象到具体的思维过程，如结构方法中数据流程图所具有的性质。目前流行着多种面向对象的开发方法，尚无一种被普遍接受。

1.2.3 软件开发的过程模型

为了解决软件生产中的问题，软件工程师必须采用某种策略把软件开发过程、开发方法以及开发工具和软件的开发生命周期阶段结合起来。这种策略称为过程模型或软件工程范例。这样一个模型决定了软件的开发过程。一个过程模型的选择依赖于应用或项目的本质、开发方法和可用的工具等。为了理解过程模型对软件开发的影响，我们首先来考查一个最简单的模型建造与修改模型(Building and Fix Model)。

许多产品的开发使用了这个建造与修改模型。在建造这些产品的时候没有任何规范和设计尝试，代替的是开发者简单地构造一个产品，然后去修改它许多次，直到客户满意。这种方法示于图 1.1。

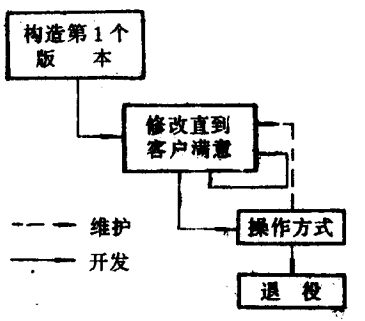

图 1.1　建造与修改模型

这样的模型对于长度为 100～200 行的小程序是满意的，但是对于任何合理规模的程序开发是不适用的。因为开发过程中错误的改正在规范和设计阶段是更容易的，即成本是低的。而在编码阶段修改的成本则相对较高，特别是在维护阶段修改的成本将更高，它将涉及到用户现场的费用以及为所有持有需修改版本客户更新版本的费用。一些更高级的模型将在下一节中进一步讨论。

1.3 某些广泛使用的过程模型

现行流行的软件开发生命周期模型有瀑布模型、原型法模型、增量模型、螺旋模型和快速应用开发模型，本书将仅讨论前 3 种。

1.3.1 瀑布模型

瀑布模型是在 1970 年由 Royce 首先提出的，20 世纪 80 年代一直被广泛地应用。该模型的一个版本显示于图 1.2 中。

按照这个模型，软件开发首先进行项目或产品的可行性论证，并估算项目的规模、成本及完成的时间等。然后通过专家的评审。接着进行用户需求分析，并用用户需求规范文档规定产品的功能及其性能。这个文档经用户和 SQA (Software Quality Assurance，软件质量保证)组审查合格后，开始进入设计阶段。设计阶段产生的设计文档将规定产品如何实现。

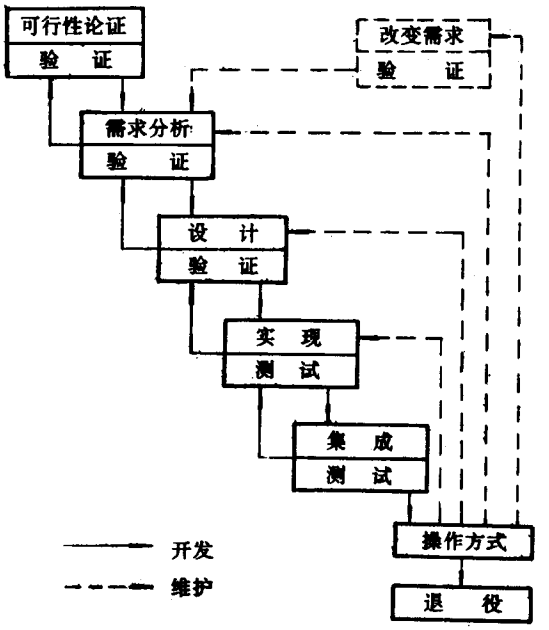

图 1.2 瀑布模型

在设计阶段期间，可能发现规范文档中的某些缺陷，如产品规范可能是不完整的(某些特征漏掉了)、矛盾的(文档中某些叙述不一致)，或者是二义性的(有多个解释)。在继续软件开发之前，必须改正规范文档中所出现的不完整、矛盾或二义性的地方。在图 1.2 中由设计箱中左侧回到需求规范箱的箭头说明了这种情况，同时应该把这些修改结合进设计文档。当开发者对设计文档满意时，提交审查并传递给程序员去实现。

在实现期间可能发现设计阶段中的缺陷。例如一个实时系统实现时，发现它太慢。出现这样一个设计缺陷可能是由于设计过程中采用了不适当的数据库组织和数据存取方法。这样的设计缺陷在编码之前应该校正。具有反馈环的瀑布模型允许实现阶段，如果需要可以修改设计文档、产品规范文档。实现阶段主要为模块编码和测试，编制模块的文档。在集成阶段将已实现的模块装配起来。

对于瀑布模型，关键的一点是对于开发中的每一个阶段的完成都必须有完整的文档和 SQA 组织批准的该阶段的产品。如果某阶段的工作由于反馈的原因需修改，那么仅当修改完成并且被批准后，那个阶段才算完成。

当开发者认为产品已经成功地完成，便会把它交给客户进行接收测试。在这个阶段提交的材料包括用户手册和其他合同中规定的文档。当客户认为产品的确满足它的规范文档，该产品交付给客户、安装并进入运行方式。

一旦客户接收了产品，对产品的任何修改，包括改正错误和扩充功能便构成了维护。维护不仅要求修改实现，而且要修改设计和规范。如果扩充功能，则需修改规范、设计直到编码。瀑布模型是一个动态模型，它的反馈环在这个动态机制中起了重要作用，随规范、设计和编码的修改，加以精确地维护。

瀑布模型的优点是它实施训练有素的方法，规定每个阶段都要有相应的文档，并要经 SQA 审查。这些文档使软件的维护更容易。瀑布模型是一个文档驱动的模型。这个模型的缺点也是来自文档驱动的特点，文档使用的是用户难以理解的工具来描述用户的需求和目标产品，直到新产品开发出来，用户难以想像他的投资将得到一个什么样的产品。在极端的情况下开发出来的产品不是用户需要的。

1.3.2 迅速原型模型

在建筑行业中，传统的方法是在建造一个大型建筑物之前，先制造一个小的木质模型。该模型反映了将要建筑的建筑物的外观以供用户评价。在计算机技术高速发展的今天，建筑行业开始使用 CAD 系统绘出要建筑的建筑物的三维模型，使用户在开始建造之前对将要建筑的建筑物有一个形象的了解。类似于建筑行业中使用的建筑模型，计算机软件在开发之前先建造一个目标产品的原型，供用户评价，以使开发出的产品更好地满足用户的需求。这就是软件开发过程的迅速原型模型的开发方法。

迅速原型是一个工作模型，它在功能上等价于产品的子集。例如目标产品是一个仓库管理系统，包括入库处理、出库处理和库存管理。于是这个迅速原型应该是一个简化的产品，它包括接收数据的屏幕处理、计算和显示结果、打印报表等，但是不包括文件处理、输入数据的合法性检查、系统出错处理等等。

迅速原型模型示于图 1.3 中。该模型的第一步是建造一个迅速原型，让用户使用和评价这个原型。一旦用户认为这个模型确实具有他需要的大部分功能，那么开发者能够提出规范文档，在某种程度上保证目标产品将是用户真正需要的。

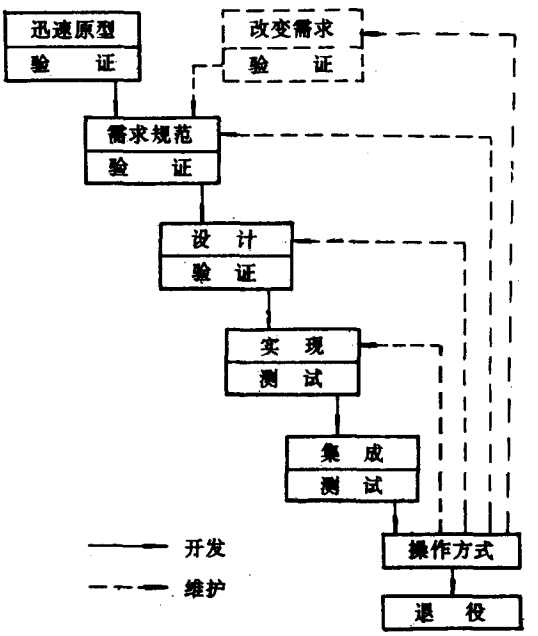

图 1.3 迅速原型法模型

建造了原型以后，软件的开发过程按图 1.3 所示进行。迅速原型的主要优点是线性的，从原型开始进行直到交付目标产品，瀑布模型中的反馈环在原型模型中基本上是不需要的。首先，开发者使用原型开发规范文档，用户已经认可了该原型，因此要求规范文档正确是合理的。其次，在设计阶段，设计者能从原型得到直观的认识和启发，可避免某些设计中的错误。因此在该模型中，反馈是不甚重要的。

在实现阶段中，由于存在设计错误的概率要比使用瀑布模型的情况下更小，因此在实现之后再修改设计的可能性就更小。一旦产品交付用户并安装之后，产品的维护开始，取决于维护的性质，可能需要重新进入需求、规范、设计或实现阶段。

迅速原型模型的本质是在于“迅速”。开发者应尽可能快地建造原型以加快开发过程。因此迅速原型应能快速建造、快速修改以反映客户的需求。一旦用户需求确定之后，原型应该抛弃。

1.3.3 增量模型

像建造一个建筑物一样，软件的开发也是逐步进行的。增量模型规定软件的开发过程是一次开发产品的一个部分。首先应该开发产品的基本部分，然后再逐步开发产品的附加

部分。为了使所开发的产品的各个部分最后能有机地组合起来，首先应该有一个统一的体系结构设计。增量模型的开发过程示于图1.4中，在该模型中，产品的设计、实现、集成和试验是以一系列增量构件为基础进行的，构件是由一些模块的编码构成并能提供特定的功能。例如，在操作系统中，调度程序是一个构件，文件管理系统也是一个构件。在增量模型的每一个阶段，都要编码一个新的构件，然后集成到先前已构成的产品中并作为一个整体进行测试，当这个产品满足规定的功能，即满足它的需求规范时，这个过程停止。开发者可以任意分解目标产品以得到一些构件，但是这些构件必须能集成为一个满足需要功能的产品。如果目标产品只能分解为很少的构件，那么增量模型可能退化为建造与修改模型。如果目标产品分解为太多的构件，那么每个阶段结束时，用户可能得不到需要的功能，并且有更多的时间浪费在集成上，因此构件的大小应适度，这取决于用户的需求。一个典型的产品通常由10～50个构件组成。

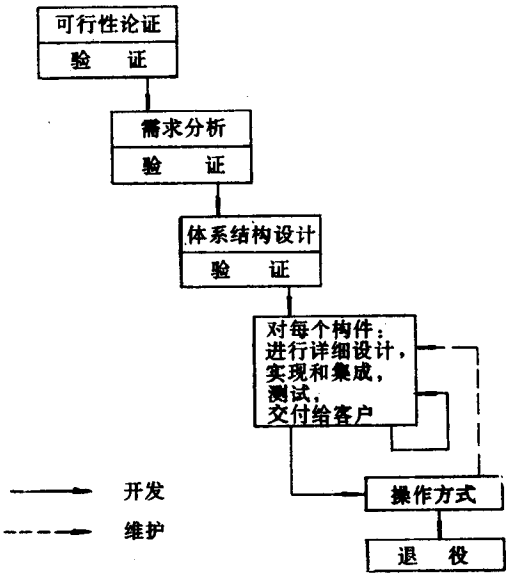

图 1.4　增量模型

瀑布模型和迅速原型模型都是提交完整的可操作质量的软件。客户能够指望交付的产品满足所有的需求，并且应该得到全面的和正确的文档，这些文档能应用于各种维护目的。但是客户为了得到其产品，必须等数月或者数年。

与此相反，增量模型确实在每个阶段都交付一个可操作的产品，但是它仅仅满足客户需求的一个子集。完整的产品划分成一些构件，产品是以一次一个构件的方式开发的。在每个阶段之后，客户都能得到一个产品，从第一个构件交付开始，客户就能做有用的工作。使用增量模型，整个产品的某些部分在数量期之内是可用的。当使用增量模型时，第一个增量经常是核心产品，它满足用户的基本需求，而许多附加功能将在以后的增量中交付。当没有足够的人员在规定的期限内开发完整的产品时，或者由于不可克服的客观原因而把交付期限规定得太短时，增量开发方式是特别有用的。

对于增量模型的一个难点是后来开发的构件必须能够集成到先前已开发的产品中而不毁坏已开发的功能。进而，现存的产品必须容易扩充，后开发的构件必须是简单和直观并容易集成。因此，对于增量模型，产品的体系结构的设计必须是开放的。

1.4　小　结

计算机软件的各种特性导致了软件生产的长周期、高成本、低质量等一系列问题，即所谓的软件困境问题。为了把软件生产从困境中解脱出来，人们使用了工程的方法，即软件工程。软件工程的生命周期法把软件开发划分成确定的阶段，每个阶段完成确定的任务。结构分析和面向对象的开发范例是目前流行的两种开发方式。结构分析方法应用时间较长，