普通高等教育“十一五”国家级规划教材

软件工程

主编 曹哲 高诚
副主编 车进辉 曹晶人 姜卓

中国水利水电出版社
www.waterpub.com.cn

普通高等教育“十一五”国家级规划教材

软件工程

主 编 曹 哲 高 诚

副主编 车进辉 曹晶人 姜 卓

中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

本书根据教育部“普通高等教育‘十一五’国家级规划教材”的精神编写。本书从实用的角度出发系统地介绍了软件工程的基本理论、方法、技术、工具和环境。全书共 14 章。内容主要包括：概述，软件项目管理，计算机系统工程，需求分析，面向数据流的分析方法，面向数据的分析方法与形式化方法，软件设计基础，面向数据流的设计方法，面向数据结构的设计方法，人机界面设计，程序设计语言与编码，软件测试，软件维护，面向对象开发方法等。

本书教学内容的深度、广度定位准确，具有鲜明的应用本科特色，并紧跟新技术，以典型案例和流行工具为依托，突出实用性，结构合理、概念清楚、通俗易懂、内容翔实、实例丰富、习题思考题与内容配合紧密。

本书既可作为高等院校“软件工程”课程的教材或教学参考书，也可作为软件开发人员的参考书。

本书配有免费电子教案，读者可以从中国水利水电出版社网站下载，网址为：http://www.waterpub.com.cn/softdown/。

图书在版编目（CIP）数据

软件工程 / 曹哲，高诚主编. －北京：中国水利水电出版社，2008

普通高等教育“十一五”国家级规划教材

ISBN 978-7-5084-5932-5

Ⅰ. 软… Ⅱ. ①曹…②高… 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（2008）第 153238 号

书　　名	普通高等教育“十一五”国家级规划教材 软件工程
作　　者	主　编　曹　哲　高　诚 副主编　车进辉　曹晶人　姜　卓
出版　发行	中国水利水电出版社（北京市三里河路 6 号　100044） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）63202266（总机）、68367658（营销中心）、82562819（万水）
经　　售	全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京市天竺颖华印刷厂
规　　格	184mm×260mm　16 开本　18.75 印张　460 千字
版　　次	2008 年 10 月第 1 版　2008 年 10 月第 1 次印刷
印　　数	0001—4000 册
定　　价	30.00 元

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

前　言

软件工程是计算机学科中一个非常有价值并具有广阔发展空间的研究领域。多年来，随着计算机硬件技术的迅猛发展，人们开发优质软件的能力远远落后于社会各个领域对计算机软件的需求，也就是说，时至今日，仍然经受着“软件危机”的困扰。为了克服“软件危机”，自20世纪60年代末期以来，人们在这一领域做了大量的研究与实践工作，积累了大量的软件开发技术和方法，进而逐渐形成了系统的软件项目开发与管理理论。于是，一门新兴的学科——软件工程学（简称软件工程）诞生了。软件工程所研究的范围十分广泛，主要包括软件项目开发和软件维护的有关理论、技术、方法、标准、计算机辅助工具和环境以及软件项目管理等诸多方面。软件工程领域的研究成果为缓解软件危机发挥了关键性作用。

“软件工程”课程是高等学校计算机学科教学计划中的一门主干课程。本书正是为普通高校计算机学科“软件工程”课程而编写的教材。本书共14章，其中第1章介绍软件工程的基本概念；第2章介绍软件项目管理，其中增加了新的COCOMO II成本估算模型，并把“软件配置管理”的内容也放在第2章中，但把“软件可靠性度量”的内容放到第12章“软件测试”中，以求达到难点分散，结构更合理的目的；第3章介绍计算机系统工程；第4～6章介绍传统需求分析的有关内容，包括需求分析概念、面向数据流、面向数据、形式化等需求分析方法、需求规格说明与评审等；第7～10章介绍传统的软件设计的有关内容，包括软件设计的概念、设计过程和一般性技术，面向数据流、面向数据以及人机界面的设计方法、技术与工具、设计规格说明与评审等；第11～13章介绍软件的实现、测试与维护，主要包括程序设计语言与编码、软件测试与调试、软件维护等；第14章以案例驱动的方法较系统地介绍了使用UML（统一建模语言）和流行的Rational Rose工具进行面向对象的系统开发方法。

本书具有以下几个特点：

（1）结构更加合理，系统介绍了软件工程的基本原理、概念、方法和工具。

（2）定位准确，具有鲜明的应用本科特色，深度和广度合适，难点分散。

（3）紧跟新技术，以典型案例和流行工具为依托，突出实用性，强调实践动手能力。

（4）概念清楚、通俗易懂、内容翔实、实例丰富，习题思考题与内容配合紧密。

本书可以作为高等院校“软件工程”课程的教材或教学参考书，也可以作为软件开发人员的参考书。

本书由曹哲、高诚任主编，由车进辉、曹晶人、姜卓任副主编，全书由曹哲统稿。具体分工为：第1章由曹哲编写，第5、6、7、8、9、10章由高诚编写，第14章由车进辉编写，第2、12章由曹晶人编写，第3、4、11、13章由姜卓编写。

在本书的编写过程中，北华大学计算机科学技术学院软件工程课程组的全体同志参与了大纲讨论，并提出了许多宝贵意见；张玲玲为本书的可靠性估算例题提供了第二种解法。在此，编者向他们表示衷心的感谢。

由于编者水平有限，书中难免存在疏漏和不妥之处，恳请广大读者批评指正。

编者联系方式：caozhe02@163.com。

编　者

2008年8月

目　　录

第 1 章　概述

随着微电子技术和计算机技术的迅速发展，计算机硬件的性能和质量也在迅速提高，而其体积、功耗、成本却在不断下降。据统计，计算机的性能平均每 18 个月就可以提高一倍。与此同时，随着计算机应用的日益普及和深化、计算机与 Internet 的连接，人类已经进入了以计算机为核心的信息社会。在信息社会中，信息的获取、加工处理和使用都需要大量高质量的软件。多年来，和计算机硬件相比，计算机软件的开发效率远远跟不上计算机应用的普及需求，软件成本在逐年上升，而质量却难以得到可靠的保证，计算机软件已经成为限制计算机系统发展的关键因素。特别是对于大型软件的开发，人们往往显得力不从心，致使进度一拖再拖、成本失去控制、软件质量得不到保证、所开发的软件难以维护。为了扭转这种被动局面，自 20 世纪 60 年代末以来，人们开始重视软件开发方法、工具和环境的研究，逐步形成了“计算机软件工程学”这一计算机科学技术领域中的新兴学科，通常简称为“软件工程”。

本章主要介绍软件和软件工程的基本概念，包括软件、软件工程、软件生命周期、软件开发模型、软件开发方法、工具和环境等。

1.1　软件

1.1.1　软件

计算机软件是指与计算机系统操作有关的程序、数据以及任何与之有关的文档资料。也就是说，软件由两部分组成：其一是机器可执行的程序和数据；其二是与软件开发、运行、维护、使用和培训等有关的文档资料。

1. 软件的特点

硬件属于物理产品，软件则属于逻辑产品。软件的开发、使用和维护与硬件相比存在很大的差别。

（1）软件开发与硬件研制相比，更依赖于开发人员的业务素质和智力，以及开发人员的组织、合作和管理。在大多数场合，软件的开发几乎都是从头开始的，而且开发的成本、进度很难估计。

（2）一般大型软件（比如 Windows 操作系统）在提交使用之前，尽管经过了严格的测试和试用，但仍然存在着潜伏的错误。而硬件产品经过严格的测试、试验、试用后，设计过程中的错误一般都是可以排除的。

（3）软件产品开发成功后，只需对原版软件进行复制，即可生产出任意多的同样的产品；而硬件产品试制成功后，要批量生产还需建厂房和生产线，投入大量的人力、物力和财力，并且需要对每件产品的质量进行控制和检验。

（4）软件在使用过程中的维护工作比硬件要复杂得多。为了修正软件中潜藏的错误，提高软件的可维护性和可靠性，完善软件的性能以适应新的软硬件环境，就需要不断对软件进行

修改。由于软件的内部逻辑关系复杂，很难维护，而且在维护的过程中还可能产生新的错误，因此，软件产品的维护工作远比硬件产品的维护复杂。

（5）由于软件不是物理产品，所以它不会磨损和老化。一个高质量的软件，只要需要，就可以长期使用下去。

软件的上述特性已经构成了一种特殊的文化，即“软件文化”。

2. 软件的发展

自从20世纪40年代第一台电子计算机问世以来，软件的发展可以划分为四个阶段。

第一阶段是20世纪60年代中期以前，这是计算机系统开发的初期阶段。在这一阶段，硬件经历了由电子管计算机到晶体管计算机的变革，而软件还没有系统化的开发方法。计算机系统中的软件多数是由用户自己设计、自己使用、自己维护，软件开发还处于个体化生产状态。由于计算机硬件价格昂贵、体积大、运算速度低、内存容量小，程序设计的目标主要集中在如何提高时空效率上。程序员为减少一条指令、少占用一个内存单元而大动脑筋。程序设计过多地依靠程序员的“技巧”，所开发的“软件”文档不全、开发过程的中间成果大都保留在程序员的记忆中，他人无法维护。初期阶段的大多数计算机系统采用批处理方式进行人机之间的联系。

第二阶段是从20世纪60年代中期到20世纪70年代中期。软件开发已进入了作坊式的生产方式，即出现了“软件车间”。硬件经历了由晶体管计算机到中小规模集成电路计算机的变革，由于CPU速度和内存容量的提高，为了有效地利用计算机系统资源，引进了多用户、多道程序和人机交互等工作方式，出现了实时系统。由于辅助存储设备的发展，出现了第一代数据库管理系统。软件开发开始形成产品。由于软件是逻辑产品，其开发、管理和维护十分复杂，致使供求矛盾加剧，到20世纪60年代末期，“软件危机”变得十分严重。

第三阶段是从20世纪70年代中期到20世纪80年代末期。软件开发进入了产业化生产，即出现了众多大型的“软件公司”。在这一阶段，硬件经历了由中小规模计算机到大或超大规模计算机的变革，致使微处理器、个人计算机、工作站具有相当高的性价比，并广泛应用于生产、生活的各个领域。与此同时，分布式系统、计算机网络、嵌入式计算机系统有了很大发展。所有这些都推动着软件产业的发展，软件开发开始采用“工程”的方法，软件产品急剧增加，质量也有了很大的提高。

第四阶段是从20世纪80年代末期开始的。这是一个软件产业大发展的时期。在这一时期，人工智能、专家系统、人工神经网络软件开始走向实际应用。多媒体技术的发展使计算机可以同时处理文字、声音、图形和图像，从而提高了计算机的信息处理能力，扩大了计算机的应用领域。多处理机系统的发展促进了多指令流多数据流（MIMD）计算机系统并行算法的研究和并行软件的开发。Internet的高速发展推动了在Internet环境下的软件开发技术和分布计算环境下的软件互操作技术的发展，从而涌现出了如Java、ASP.NET等新的开发环境和CORBA（通用对象请求代理结构）标准等。这一阶段也是软件工程大发展的时期，人们开始采用面向对象的技术和可视化的集成开发环境。

1.1.2 软件危机

软件危机是指在计算机软件开发、使用与维护过程中遇到的一系列严重问题和难题。

1. 软件危机的表现

应该说，自从计算机诞生以来，软件危机就一直存在，软件危机主要表现在以下几个方面：

（1）人们对软件开发的成本和进度的估计常常不够准确。由于软件的特殊性，不同类型的软件开发所需的工作量、成本往往差别很大。因此，常常出现实际成本比估算成本高出一个数量级，实际进度比计划进度拖延几个月甚至几年的现象，这大大降低了开发商的信誉，也引起了用户的不满。

（2）用户对已完成的软件不满意的现象时有发生。这主要是由于软件开发者和用户没有进行充分的交流和沟通，开发者与用户对软件功能的理解有偏差，从而开发出不实用或不能完全符合用户需求的软件系统。

（3）软件产品的质量往往是靠不住的。由于软件可靠性和软件质量的定量计算还没有确切可信的方法，加上开发成本和进度的限制，软件质量保证技术（复审、测试等）很难完全贯彻到软件开发的全过程中，因此，软件的质量难以保证。

（4）软件常常是不可维护的。许多软件在开发时就没有考虑到将来的修改。因此，很多软件中的错误是非常难以改正的。实际上，不可能使这些软件适应新的硬件环境，也无法根据用户的需要在原有的软件中增加一些新的功能，软件的维护工作相当困难。

（5）软件通常没有适当的文档资料。软件开发一般没有严格的文档制度，软件开发结束后，保留下来的有效的文档资料很少，特别是与软件版本相匹配的文档资料就更少。需要指出，软件文档资料是开发人员自己及其与用户之间交流信息的依据，也是软件维护人员进行维护的依据，更是软件管理人员对软件开发工程进行管理和评价的根据。文档资料不全或不合格，必将给软件开发和维护工作带来许多难以想象的困难和难以解决的问题。

（6）软件成本在计算机系统总成本中所占的比例逐年上升。随着微电子技术的迅速发展，计算机硬件成本逐年下降。与此相反，随着对软件规模和数量需求的不断增长，软件开发需要投入大量的人力、物力和财力，从而使软件成本占计算机系统总成本的比例逐年上升。特别是软件维护成本迅速增加，已经占到软硬件总成本的 40%～75%，如图 1-1 所示。

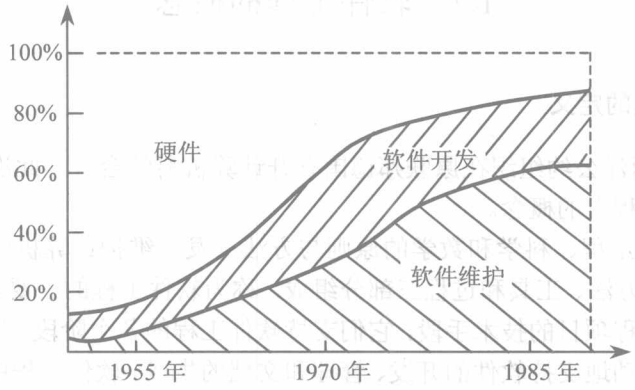

图 1-1　软件、硬件成本变化趋势

（7）软件开发生产率提高的速度远远跟不上日益增长的软件需求。

2. 软件危机的产生原因

造成软件危机的原因很多，其中主要有以下几个：

（1）用户对软件需求的描述不精确。在软件开发的初期，用户对目标软件的功能、性能并不十分了解，因此对软件需求的描述可能有遗漏、二义性或错误，到了软件开发中期，用户

才提出新的要求，这时的变动将付出巨大的工作量和处理相当复杂的逻辑关系，代价巨大。如果到软件完成后再提出修改，其代价比初期要高出 2～3 个数量级。

（2）软件开发人员对用户需求的理解有偏差。软件开发人员很可能并不十分熟悉用户的业务，因此对用户需求的理解有偏差，这将导致软件产品与用户的需求不一致。

（3）缺乏处理大型软件项目的经验。开发大型软件项目需要组织众多人员共同完成。一般来说，多数管理人员缺乏大型软件的开发经验，而多数软件开发人员又缺乏大型软件项目的管理经验，致使各类人员的信息交流不及时、不准确，容易产生误解。

（4）开发大型软件易产生疏漏和错误。大型软件逻辑关系复杂、开发周期长、涉及人员多，因此大型软件产品中很可能存在疏漏和潜伏错误。

（5）缺乏有力的方法学的指导和有效的开发工具的支持，导致软件开发过多地依靠程序员的“技巧”，从而加剧了软件产品的个性化。

（6）面对日益增长的软件需求，开发人员越发显得力不从心。目前，需要开发的软件越来越大，逻辑关系越来越复杂，致使人们无力解决“复杂问题”。从某种意义上说，解决供求矛盾将是一个永恒的主题。

3. 缓解软件危机的途径

到了 20 世纪 60 年代末期，软件危机已相当严重。这促使计算机科学家们开始探索缓解软件危机的方法。他们提出了“软件工程”的概念，即用现代工程的原理、技术和方法进行软件的开发、管理、维护和更新。于是，开创了计算机科学技术的一个新的研究领域——软件工程。几十年来，软件工程在软件开发方法、工具和管理等方面的研究与应用已经取得了很多成果，并已大大缓解了软件危机所带来的影响。当然，要进一步解决软件危机，还有相当多的工作要做。

1.2 软件工程的概念

1.2.1 软件工程的定义

1968 年，北大西洋公约组织在原联邦德国召开计算机科学会议，此次会议上 Fritz Bauer 首次提出了“软件工程”的概念。

软件工程是指用工程、科学和数学的原则与方法开发、维护计算机软件的有关技术和管理方法。软件工程由方法、工具和过程三部分组成，称为软件工程的三要素。软件工程中的各种方法是完成软件工程项目的技术手段，它们支持软件工程的各个阶段。软件工程使用的软件工具能够自动或半自动地支持软件的开发、管理和文档的生成。软件工程中的过程贯穿于整个工程的各个环节，在这一过程中，管理人员应对软件开发的质量、进度、成本等进行评估、管理和控制，包括计划跟踪与控制、成本估算、人员的组织、质量保证、配置管理等。

1.2.2 软件工程的基本原理

著名的软件工程专家 B. W. Boehm 于 1983 年综合了软件工程专家学者们的意见，并总结了开发软件的经验，提出了软件工程的 7 条基本原理。这 7 条原理被认为是确保软件产品质量和开发效率的原理的最小集合，又是相互独立、缺一不可、相当完备的最小集合。下面就简单

介绍一下软件工程的这 7 条原理。

1. 用分阶段的生存周期计划严格管理

根据这条基本原理，可以把软件生存周期（即软件从开发、运行和维护直至退役的全过程）划分成若干个阶段，并相应地制定出切实可行的计划，然后严格按照计划对软件开发与维护进行管理。需要制定的计划有项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划和运行维护计划等。各级管理人员必须严格按照计划对软件开发和维护工作进行管理。据统计，在不成功的软件项目中，有一半左右是由于计划不周造成的。

2. 坚持进行阶段评审

据统计，在软件生存周期各阶段中，编码阶段之前的错误约占 63%，而编码错误仅占 37%。另外，错误发现并改正得越晚，所付出的代价越高。坚持在每个阶段结束前进行严格的评审，就可以尽早发现错误。因此，这是一条必须坚持的重要原理。

3. 实行严格的产品控制

由于外部环境的变化，在软件开发的过程中改变需求是难免的，但决不能随意改变需求，只能依靠科学的产品控制技术来顺应用户提出的需求改变。为了保持软件各个配置成分的一致性，必须实行严格的产品控制。其中主要是实行基准配置管理（又称为变动控制），即凡是修改软件的建议，尤其是涉及基本配置的修改建议，都必须按规程进行严格的评审，评审通过后才能实施。这里的“基准配置”是指经过阶段评审后的软件配置成分，即各阶段产生的文档或程序代码等。

4. 采用现代程序设计技术

实践表明，采用先进的程序设计技术既可以提高软件开发与维护的效率，又可以提高软件的质量。多年来，人们一直致力于研究新的“程序设计技术”。比如，20 世纪 60 年代末提出的结构化程序设计技术；后来又发展出各种结构分析（SA）技术和结构设计（SD）技术；之后又出现了面向对象分析（OOA）和面向对象设计（OOD）技术等。

5. 结果应能清楚地审查

软件产品是一种看不见、摸不着的逻辑产品。因此，软件开发小组的工作进展情况可见性差，难以评价和管理。为了更好地进行评价和管理，应根据软件开发的总目标和完成期限，尽量明确地规定出软件开发小组的责任和产品标准，从而能清楚地审查所得到的结果。

6. 开发小组的人员应少而精

软件开发小组人员的素质和数量是影响软件质量和开发效率的重要因素。实践表明，素质高的人员与素质低的人员相比，其软件开发的效率可能高几倍甚至几十倍，而且所开发的软件中的错误也要少得多。另外，开发小组的人数不宜过多，因为随着人数的增加，人员之间交流情况、讨论问题的通信开销将急剧增加，这不但不能提高生产效率，反而会由于误解等原因而增加出错的概率。

7. 承认不断改进软件工程实践的必要性

遵循上述六条基本原理，就能够较好地实现软件的工程化生产。但是，软件工程不能停留在已有的技术水平上，应积极主动地采纳或创造新的软件技术，要注意不断总结经验，收集工作量、进度、成本等数据，并进行出错类型和问题报告的统计。这些数据既可用来评估新的软件技术的效果，又可用来指明应优先进行研究的软件工具和技术。

1.2.3 软件工程的目标

软件工程的目标是在给定成本、进度的前提下，开发出具有可修改性、有效性、可靠性、可理解性、可维护性、可重用性、可适应性、可移植性、可追踪性和可互操作性并满足用户需求的软件产品。

（1）可修改性（modifiability）：允许对软件系统进行修改而不增加其复杂性。它支持软件的调试与维护。

（2）有效性（efficiency）：指软件系统的时间效率和空间效率。这是一个应该努力追求的重要目标。一般来说，时间效率和空间效率往往是矛盾的，即两者不能兼得。因此，软件设计人员常根据实际系统的资源和性能要求进行适当地取舍或折衷，以便更好地实现用户的需求。

（3）可靠性（reliability）：是指在给定的时间间隔内，程序成功运行的概率。可靠性是衡量软件质量的一个重要目标。尤其是对于实时嵌入式计算机系统，可靠性是一个极其重要的目标。比如，载人宇宙飞船的导航，一旦出现问题就会酿成灾难性的后果。而要开发出可靠性高的系统，在软件设计、编码和测试的过程中，必须将可靠性放在重要地位。

（4）可理解性（understandability）：指系统具有清晰的结构，能直接反映问题的需求。可理解性有助于控制软件系统的复杂性，并支持软件的维护、移植和重用。

（5）可维护性（maintainability）：是指软件产品交付使用后，在实现改正潜伏的错误、改进性能等属性、适应环境变化等方面工作的难易程度。由于软件的维护费用在整个软件生存周期中占主要的比重，因此，可维护性是软件工程中的一个十分重要的目标。软件的可理解性和可修改性支持软件的可维护性。

（6）可重用性（reusability，或称可复用性）：是指软部件可以在多种场合使用的程度。概念或功能相对独立的一个或一组相关模块可构成一个软部件。软部件应具有清晰的结构和注释、正确的编码和较高的时空效率。可将各种软部件按照某种规则放在软部件库中供开发人员选用。广义地讲，可重用性还应包括应用项目、规格说明、设计、概念和方法等的重用。一般来说，重用的层次越高，带来的效益越大。可重用性有助于提高软件产品的质量和开发效率，降低软件开发和维护的费用。

（7）可适应性（adaptability）：是指软件在不同的系统约束条件下，使用户需求得到满足的难易程度。选择广为流行的软硬件支持环境，采用广为流行的程序设计语言编码，采用标准的术语和格式书写文档均可增强软件产品的可适应性。

（8）可移植性（portability）：是指软件从一个计算机系统或环境移植到另一个计算机系统或环境上去的难易程度。采用通用的运行支持环境和尽量通用的程序设计语言的标准部分均可提高可移植性。而将依赖于计算机系统的低级（物理）特征部分相对独立、集中起来，就可以使与处理机无关的部分能够很方便地移植到其他系统上。可移植性支持软件的可重用性和可适应性。

（9）可追踪性（traceability）：是指根据软件需求对软件设计、程序进行正向追踪，或根据程序、软件设计对软件需求进行逆向追踪的能力。软件开发各阶段的文档和程序的完整性、一致性、可理解性支持软件的可追踪性。

（10）可互操作性（interoperability）：是指多个软件元素相互通信并协同完成任务的能力。

追求上述目标即可提高软件产品的开发效率，降低维护难度，保证软件产品的质量。

1.2.4　软件工程的原则

从宏观的角度讲，要实现软件工程的目标，必须遵循前面叙述的软件工程的 7 条基本原理；从具体实现的角度讲，在软件开发的过程中，还必须遵循下列软件工程的原则。

（1）抽象（abstraction）：抽取各个事物中共同的最基本的特征和行为，暂时忽略它们之间的差异。一般采用分层次抽象的方法来控制软件开发过程的复杂性。抽象使软件的可理解性增强并有利于开发过程的管理。

（2）信息隐藏（information hiding）：将模块内部的信息（数据和过程）封装起来。其他模块只能通过简单的模块接口来调用该模块，而不能直接访问该模块内部的数据或过程，即将模块设计成“黑箱”。信息隐藏的原则可使开发人员把注意力集中于更高层次的抽象上。

（3）模块化（modularity）：把一个程序划分成若干个模块，每个模块完成一个子功能，将这些模块组装成一个整体，即可完成该程序指定的功能。其中每个模块是程序中相对独立的成分，是独立的编程单位。如 C 语言中的函数、Ada 语言中的程序包等。采用模块化原则有助于信息隐藏和抽象。

（4）局部化（localization）：即在一个物理模块内集中逻辑上相互关联的计算资源。局部化支持信息隐藏，从而保证模块之间具有松散的耦合、模块内部有较强的内聚。这有助于控制每一个解的复杂性。

（5）一致性（consistency）：整个软件系统（包括程序、数据和文档）的各个模块应使用一致的概念、符号和术语；程序内部接口应保持一致；软件与环境的接口应保持一致；系统规格说明与系统行为应保持一致；用于形式化规格说明的公理系统应保持一致。

（6）完全性（completeness）：软件系统不丢失任何重要成分、完全实现所需系统功能的程度。为了保证系统的完全性，在软件的开发和维护过程中需要严格的技术评审。

（7）可验证性（verifiability）：开发大型软件系统需要对系统逐层分解。系统分解应遵循易于检查、测试、评审的原则，以使系统可验证。

抽象、信息隐藏、模块化和局部化的原则支持可理解性、可修改性、可靠性等目标，并可提高软件产品的质量和开发效率；而一致性、完全性和可验证性等原则可以帮助软件开发人员去实现一个正确的系统。

1.3　软件生存周期

我们知道，人的一生要经过胎儿、幼年、童年、青年、中年、老年，直至死亡，这是人的生存周期。同样，软件从定义开始，经过开发、使用和维护，直到最终退役的全过程称为软件生存周期。软件生存周期各阶段的划分目前尚不统一。为了便于软件开发过程的管理和软件质量的控制，可将软件生存周期划分为 3 个过程共 9 个阶段。3 个过程是：软件定义过程、软件开发过程、软件使用与维护过程。9 个阶段有：可行性研究、需求分析、概要设计、详细设计、实现、组装测试、验收测试、使用与维护、退役。它们之间的关系如图 1-2 所示。需要指出的是，软件生存周期和人的生存周期是不同的。人不可能从中年返回到童年，而软件开发如果在编码阶段发现需求分析有错，就应返回去重新进行需求分析，这一过程称为迭代过程。下面就来简单介绍每个阶段的基本任务、完成任务的途径以及应提交的阶段性成果。

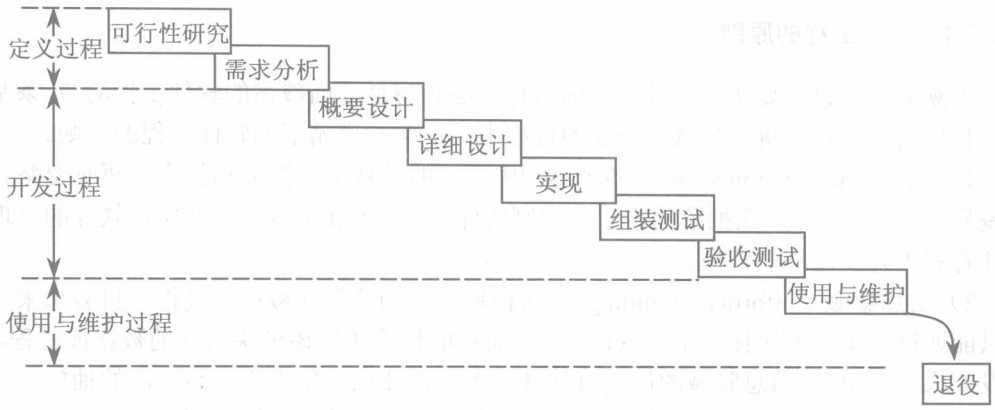

图 1-2 软件生存周期各阶段的划分

1.3.1 软件定义

软件定义的基本任务是确定软件系统的工程需求，也就是要搞清“做什么”。软件定义过程可以通过软件系统的可行性研究和需求分析两个阶段来完成。

1. 可行性研究

本阶段的任务是根据用户提出的工程项目的性质、目标和规模，进一步了解用户的要求及现有的环境及条件，从技术、经济和社会等多方面研究并论证该项目的可行性。即该项目是否值得去解决，是否存在可行的解决办法。此时，系统分析人员应在用户的配合下对用户的要求和现有的环境进行深入调查并写出调研报告，进而进行可行性论证。可行性论证包括经济可行性、技术可行性、操作可行性、法律可行性等。在此基础上还要制定初步的项目计划，包括需要的软硬件资源、定义任务、风险分析、成本/效益分析以及进度安排等。可行性研究的结果将是部门负责人做出是否继续进行该项目决策的重要依据。

2. 需求分析

（1）需求分析的任务。需求分析的任务是确定待开发的软件系统“做什么”。具体任务包括确定软件系统的功能需求、性能需求和运行环境约束，编制软件需求规格说明书、软件验收测试计划和准则、初步的用户手册。其中，功能需求应给出软件必须具备的全部功能；性能需求是指软件的安全性、可靠性、可维护性、精度、容错能力、运行效率等；运行环境约束是指软件将来运行时必须满足的运行环境方面的限制。

（2）需求分析的实现途径。软件系统需求一般由用户提出。用户虽然了解他们所面对的问题，但是由于缺乏软件开发的经验，通常不能完整准确地表达出他们的要求；而系统分析员和软件开发人员虽然知道怎样用软件实现人们的要求，但对特定用户的具体要求并不完全清楚。因此，系统分析员和开发人员在需求分析阶段必须与用户反复讨论、协商，充分交流信息，并用某种方法和工具构建软件系统的逻辑模型。比如，可以用数据流图、数据字典和简要的算法来表达系统的逻辑模型，也可以采用目前较流行的面向对象的分析方法构建系统的逻辑模型等。为了使开发方与用户对待开发软件系统达成一致的理解，必须建立相应的需求文档。有时对大型、复杂的软件系统的主要功能、接口、人机界面等还要进行模拟或建造原型，以便向用户和开发方展示待开发软件系统的主要特征。确定软件需求的过程有时需要反复多次，以便使

用户需求逐步精确化、一致化、完全化，最终得到用户和开发方的确认。

（3）需求分析的阶段成果。需求分析阶段的主要成果有软件需求规格说明书、软件验收测试计划和准则、初步的用户手册等。其中，软件需求规格说明书（Software Requirements Specification，SRS）是一个关键性的文档。一方面，在多数场合，面向开发者的软件需求用需求规格说明语言来描述，它是软件开发人员进行软件设计的依据；另一方面，从某种意义上讲，SRS 又起到与用户签定合同的合同书的作用。因此，在 SRS 中应包括软件系统的全部功能需求、性能需求、接口需求、设计需求、基本结构、开发标准和验收原则等。

1.3.2　软件开发

软件生存周期中的软件开发过程由概要设计、详细设计、实现（即编码与单元测试）、组装测试、验收测试 5 个阶段组成。其中，概要设计和详细设计统称为设计；编码即编程；单元测试、组装测试和验收测试统称为测试。软件开发过程就是软件开发人员按照需求规格说明书的要求，由抽象到具体，直到生成程序，并进行各项测试，最后产生软件产品的过程。在这一过程中，开发者通常可提出多种设计方案，并对各种方案在功能、性能、成本、进度等方面进行比较和折衷，从中选出一种“最佳方案”，以便在满足用户对软件总体目标需求的基础上最大限度地降低成本。下面将简单介绍软件开发过程中各阶段的任务、实现的途径和阶段成果。

1. 概要设计

概要设计又称总体设计。其任务是对需求规格说明书中提供的软件系统逻辑模型进行进一步的分解，从而建立软件系统的总体结构和各子系统之间、各模块之间的关系，定义各子系统接口界面和各功能模块的接口，设计全局数据库或数据结构，规定设计约束，制定组装测试计划，进而给出每个功能模块的功能描述、全局数据定义和外部文件定义等。

要完成概要设计，必须选择某种方法和工具。比如，可以采用面向数据流的设计方法，将需求分析得到的软件系统数据流图进一步分解，然后以结构图为工具，将数据流图映射为软件系统层次结构图；也可以采用面向对象的设计方法和工具等。需要指出，设计的软件系统应具有良好的总体结构、尽量降低模块接口的复杂度，并力争做到各功能模块之间的联系较松散（低耦合度），而功能模块内部具有较高的内聚度。

概要设计的阶段性成果包括概要设计说明书、数据库或数据结构说明书、组装测试计划等文档。

2. 详细设计

详细设计阶段的任务是将概要设计产生的功能模块进一步细化，形成可编程的程序模块，然后设计程序模块的内部细节，包括算法、数据结构以及各程序模块间的接口信息，并设计模块的单元测试计划。详细设计可以采用结构化的设计方法，采用结构化的程序流程图、N-S 图、过程设计语言（Procedure Design Language，PDL）等工具进行描述，也可以采用面向对象的设计方法等。详细设计的阶段成果应提供“详细设计规格说明”（或称“模块开发卷宗”）和单元测试计划等详细设计文档。

3. 实现

实现即编码和单元测试。编码的主要任务是根据详细设计规格说明用某种选定的程序设计语言把详细设计的结果转化为机器可运行的源程序模块，这是一个编程和调试程序的过程。一般来说，对软件系统所采用的分析方法、设计方法、编程方法以及所选用的程序设计语言应