软件工程

——原理、方法与应用

吴钦藩　编著

人民交通出版社

Ruanjian Gongcheng

软 件 工 程

——原理、方法与应用

吴钦藩 编著

人民交通出版社

内 容 提 要

本书全面、系统地介绍了有关“软件工程”的概念、原理、方法及其应用，并力图反映国内外软件工程领域的最新发展的成果。全书分软件工程概述、系统定义与软件计划、软件需求分析、概要设计、详细设计、软件编码、软件测试、软件维护、软件管理等九章，并选编了大量的习题和思考题。

本书可作为高等学校或继续教育“软件工程”课程的教材和教学参考书，也可供有一定实践经验的软件开发人员、管理人员阅读，对各个级别（尤其是高级程序员、系统分析员）的计算机软件专业技术资格和水平考试人员也有较高的参考价值。

图书在版编目(CIP)数据

软件工程：原理、方法与应用/吴钦藩编著. —北京：人民交通出版社，1997.5

ISBN 7-114-02629-3

Ⅰ.软… Ⅱ.吴… Ⅲ.软件工程 Ⅳ.TP311.5

中国版本图书馆CIP数据核字(97)第07131号

责任印制：孙树田

软 件 工 程

——原理、方法与应用

吴钦藩 编著

插图设计：裘琳 版式设计：崔凤莲

人民交通出版社出版发行

(100013 北京和平里东街10号)

各地新华书店经销

北京云浩印制厂印刷

开本：787×1092 1/16 印张：15.25 字数：390千

1997年9月 第1版

1997年9月 第1版 第1次印刷

印数：0001～3000册 定价：28.00元

ISBN 7-114-02629-3

TP·00007

序

软件工程是计算机科学与技术中的一个重要学科，用以指导软件人员进行软件的开发、维护和管理。软件工程学主要研究软件结构、软件设计方法、软件工具和开发环境、软件工程标准和规范以及与软件工程有关的工程化思想和理论。软件工程的出现是软件产品走向大生产、走向市场的转折，虽然只有几十年的历史，但它同计算机科学与技术一起迅速发展，软件方法学和软件工程已成为软件人员必须掌握的基本知识和技能。

软件工程包括的面很广，有基础理论研究，也有应用研究以及实际开发和维护，除了技术问题之外，它还涉及与软件有关的所有活动，例如计算机可靠性理论、管理学、经济学、心理学、市场学、法律与道德等方面。软件工程还深入到人工智能、人机界面、知识库系统等计算机各专业学科，影响着这些学科的发展。

吴钦藩同志编著的“软件工程——原理、方法与应用”一书，是结合他多年的教学经验和参加交通部“七五”、“八五”CAD 工程及其它科研项目的体会，系统地介绍了软件工程的概念、原理、方法及其应用，并反映了国内外软件工程领域的最新进展和成果。此书内容翔实，通俗易懂，有较强的系统性、可读性和实用性，是高校计算机及其它工科专业学生学习“软件工程”的好教材，也可供有关的软件开发、维护、管理人员阅读参考，作为继续教育以及各个级别的计算机软件专业技术资格和水平考试有关内容的学习教材。

张叔辉

前　言

软件工程是人们用以指导软件开发、维护的一门新技术，国内在80年代初刚兴起。软件工程学主要研究软件结构、软件设计方法、软件工程标准和规范、软件工具以及与软件工程有关的思想和理论。它是计算机专业的新兴学科。

本书系统、全面地介绍了有关软件开发、维护和管理的工程化技术，比较详尽地论述了软件工程的概念、原理、方法及其应用，有较强的系统性、可读性和实用性，并力图反映国内外软件工程的最新发展和技术水平。全书分软件工程概述、系统定义与软件计划、软件需求分析、概要设计、详细设计、软件编码、软件测试、软件维护和软件管理等九章，并选编了大量的习题和思考题，可作为高等学校“软件工程”课程的教材和教学参考书。

本书涉及面广、内容翔实，在编写时力求简明、实用、易懂，能使读者在较短的时间内了解软件工程的基本原理和方法。通过阅读教材中的实例和在实际课题中运用软件工程方法，能达到更好的学习效果。

本书也力求能适应于不同层次读者的需要，供有一定实践经验的软件开发人员、管理人员阅读，作为继续教育的教材以及各个级别的计算机软件专业技术资格和水平考试有关内容的学习教材。

上海海运学院计算机系周广声教授对本书作了认真、细致的审阅，并提出了宝贵的意见，对此表示衷心地感谢。

由于编者水平有限，书中难免还存在一些缺点和错误，殷切希望广大读者批评指正。

编　者

1996年5月

目　录

第一章　软件工程概述

1.1　软件危机和软件工程

自1946年世界上出现了第一台电子计算机以来，计算机技术发展十分迅速。60年代，先进国家计算机应用越来越广泛，几乎涉及到社会生活的各个方面，诸如工厂管理、银行事务、情报检索、飞机订票、办公室自动化(OA)等系统。随着计算机应用日益普及和深化，正在运行使用着的计算机软件的数量以惊人的速度急剧膨胀，而且软件的规模十分庞大，包含数百万行代码、耗资几十亿美元、花费几千人年的劳动才开发出来的软件产品，在70年代已经屡见不鲜。

由于微电子技术的进步，计算机硬件成本每5年下降两至三个数量级，而且质量稳定提高。与此同时，计算机软件成本却在逐年上升(图1-1)，质量没有可靠的保证，软件开发的生产率也远远跟不上普及计算机应用的要求。软件已经成为限制计算机系统发展的关键因素。

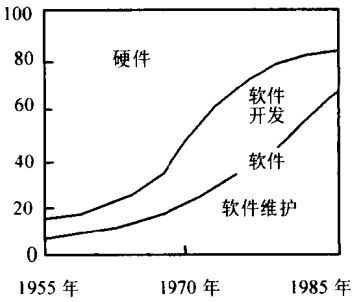

图1-1　软件与硬件成本比

在计算机系统发展的早期时代所形成的一些概念和做法，已经阻碍着计算机软件的开发，更为严重的是，用错误方法开发出来的许多大型软件几乎根本无法维护，只好提前报废，造成大量人力、物力和财力的浪费。西方计算机科学家把软件开发和维护过程中遇到的一系列严重问题统称为“软件危机”，并且在60年代后期开始认真研究解决软件危机的方法，从而逐步形成了计算机科学技术领域中的一门新兴的学科——计算机软件工程学，通常简称为软件工程。

1.1.1　计算机系统的发展

过去人们往往按照计算机硬件的演变来划分计算机系统发展的时期，然而为了了解计算机软件发展演变的过程，特别是为了了解软件危机是怎样产生的，又是如何加剧的，从而探索解决软件危机的途径，我们应该更全面地回顾计算机系统发展的简短历史，按照计算机应用领域的演变而不是仅仅根据硬件特点的演变来划分计算机系统发展的时期。

软件发展的历史是与计算机系统发展紧密相关的。

在计算机系统发展的初期(50年代～60年代)，硬件经历了不断的变化，而软件则被多数人作为一种事后的工作来看待，几乎没有什么系统的方法可以遵循。在此期间，大多数系统采用批处理工作方式，只有个别系统是交互式系统。当时通用硬件已经成为平常的事情，软件却是为每一种用途分别设计的，它们的通用性相当有限，而且，大多数软件是由使用该软件的人或机构研制的，使软件带有较强烈的个人色彩。由于这种个体化的软件环境，使得软件设计通常是在人们头脑里进行的一个隐含的过程，而且，除了程序清单之外，一般没有其他文档资料保存下来。

计算机系统发展的第二个时期跨越了从60年代中期到70年代中期这10年。在这个时期，硬件经历了从晶体管计算机到集成电路计算机的变革，CPU速度和内存容量都有了很大提高，从而为计算机在众多领域中的应用提供了潜在的可能性。为了更有效地利用硬件的能力，计算机系统普遍采用多道程序多用户的分时工作方式，从而引入了人—机对话的新概念，该技术使软件和硬件提高到更为精湛的新水平。实时系统能够在几毫秒内而不是几分钟内收集、分析和变换来自多个信息源的数据，进而控制处理过程并产生输出。联机辅助存贮设备的发展导致了数据库管理系统的出现。

第二个时期也是以产品化软件的使用和“软件车间”的出现为特征的，人们开发软件是为了广泛销售。不过在这个时期，“软件车间”开发软件时，基本上仍然沿用早期时代形成的开发方法。

随着以计算机为基础的系统日益增多，计算机软件库开始膨胀。本单位自己开发的软件和从外部购买来的软件产品堆集了几十万条源程序的语句，所有这些程序，在运行中发现错误时必须加以改正；当用户的要求改变时，必须加以修改；当买进新硬件或操作系统的新版本时，通常必须改变程序以适应新的环境。上述种种软件维护工作，其开销越来越大(图1-1)。更坏的是，许多程序所带有的个人色彩使得它们实际上是不可维护的。“软件危机”就开始出现了！1968年北大西洋公约组织的一次学术会议上，计算机科学家讨论了软件危机问题，首次提出和使用“软件工程”这个名词，一门新兴的工程学科就此出现了。

计算机系统发展的第三个时期从70年代初期开始，一直沿续到80年代。这个时期硬件发展的特点是从集成电路计算机进步到超大规模集成电路计算机，高性能低成本的微处理机大量涌现，发展日新月异。

在这个时期，分布式系统使用多台计算机共同解决一个大问题，各台计算机并行工作分别完成各自的小任务，同时，各台间进行必要的通信和同步，这大大增加了计算机系统的复杂程度。利用计算机硬件提供的简单算术运算和逻辑运算能力，模拟人类复杂的思维解题过程的人工智能系统，需要十分复杂的计算机软件才能实现。

硬件的迅速发展已经超过人们提供支持软件的能力。然而，硬件只提供了潜在的计算能力，如果没有软件来驾驭和开发这种能力，人类并不能有效地使用计算机。在第三个发展时期，软件危机进一步加剧了。用于维护软件的费用占了数据处理总预算的50%以上，软件开发的生产率远远满足不了对于新系统的需求。为了对付日益严重的软件危机，计算机科学家和软件工程师开始认真研究和采用软件工程学。

1.1.2 软件危机

软件危机指在计算机软件的开发和维护过程中所遇到的一系列严重问题。这些问题绝不仅仅是“不能正常运行”的软件才具有的，实际上几乎所有软件都不同程度地存在这些问题。概括地说，软件危机包含下述两方面的问题：如何开发软件，怎样满足对软件的日益增长的需求；如何维护数量不断膨胀的已有软件。具体地说，软件危机主要有下述一些表现：

1) 对软件开发成本和进度的估计常常很不准确。实际成本比估计成本高出一个数量级，实际进度比预期进度拖延几个月甚至几年的现象并不罕见。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满。

2) 用户对“已完成的”软件系统不满意的现象经常发生。软件开发人员常常在对用户要求

只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就仓促上阵匆忙着手编写程序。软件开发人员和用户之间的信息交流往往很不充分，“闭门造车”必然导致最终的产品不符合用户的实际需要。

3）软件产品的质量往往靠不住。软件可靠性和质量保证的确切的定量概念刚刚出现不久，软件质量保证技术（审查、复审和测试）还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

4）软件常常是不可维护的。很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可再用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

5）软件通常没有适当的文档资料。计算机软件不仅仅是程序，还应该有一整套文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是“最新式的”（即和程序代码完全一致的）。软件开发组织的管理人员可以使用这些文档资料作为“里程碑”来管理和评价软件开发工程的进展状况；软件开发人员可以利用它们作为通信工具，在软件开发过程中准确地交流信息。对于维护人员而言，这些文档资料更是至关重要、必不可少的。缺乏必要的文档资料或者文档资料不合格，必然给软件开发和维护带来许多严重的困难和问题。

6）软件成本在计算机系统总成本中所占的比例逐年上升。由于微电子技术的迅速发展，硬件成本骤降，然而软件开发需大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。软件价格昂贵，已成为许多计算机系统中花钱最多的项目。

7）软件开发生产率提高的速度远远跟不上计算机应用迅速普及深入的趋势。软件产品“供不应求”的现象使人类不能充分利用80年代计算机硬件提供的巨大潜力。

导致这一系列问题的重要原因，一方面与软件本身的特点有关，另一方面也和软件开发人员的弱点有关。

软件不同于硬件，它是计算机系统中的逻辑部件而不是物理部件。在写出程序代码并在计算机上试运行之前，软件开发过程的进展情况较难衡量，软件开发的质量也较难评价，因此，管理和控制软件开发过程相当困难。此外，软件在运行过程中不会因为使用时间过长而被“用坏”。如果运行中发现错误，则很可能是遇到了一个在开发时期引入的在测试阶段没能检测出来的故障，因此，软件维护通常意味着改正或修改原来的设计，这就在客观上使得软件较难维护。

软件本身独有的特点确实给开发和维护带来一些客观困难，但是，人们在开发和使用计算机系统的长期过程中，也确实积累和总结出了许多成功的经验。如果坚持不懈地使用经过实践考验证明是正确的方法，许多困难是完全可以克服的，过去也确实有一些成功的范例。但是，由于在计算机系统发展的早期时代软件开发的个体化特点，目前相当多的软件人员对软件开发和维护还有不少糊涂观念，在实践过程中或多或少地采用了错误的方法和技术，这可能是软件问题发展成软件危机的主要原因。

所以，必须认识到软件开发不是某种个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。必须充分吸收和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法，特别要吸取几十年来人类从事计算机研究和开发的经验教训。为了解决软件危机，既要有软件开发和维护的技术措施（软件方法和工具），又要有必要的组织管理措施，软件工程正是从技术和管理两方面研究如何更好地开发

和维护计算机软件的一门新兴学科。

1.1.3 软件工程

针对60年代末期出现的软件危机，人们提供了软件工程(Software engineering)这一新的概念，以便借助于传统工程设计的一些基本思想，来降低软件结构和软件管理的复杂性，提高软件的品质。其目的是突出软件研制所需要的与处理过程有关的方法。经过十多年对这个问题的研究，我们已经了解关于软件研制过程中许多重要知识。根据这些知识，已经发展了许多技术，能够适用于软件研制的整个周期的各个阶段。

从1970年开始，许多人都想对软件工程下定义，Peter Wegner 和 Barry Boehm 认为软件工程可被定义为：科学知识在设计和构造计算机程序，以及开发、运行和维护这些程序所要求的有关文件编制中的实际应用。F. L. Bauer 认为：为了经济地获得软件，这个软件是可靠的并且能在实在的计算机上工作，因此需要健全的工程原理(方法)的确立和使用。

中国大百科全书计算机卷是这样定义的：用工程方法开发与维护软件的过程和有关的技术。根据这一定义，软件工程就是一种开发和维护软件的规范化方法。作为工程科学，就是要采用工程的概念、原理、技术和方法来开发和维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来。软件工程的主要对象是大型软件，强调的是使用生存周期方法学和各种结构分析及结构设计技术。其研究的主要问题有：软件质量保证与评价、开发与维护方法、工具系统、文件资料、用户界面以及软件管理等。

软件工程学是研究软件工程学问的一门科学，其研究的主要内容是："如何应用一些科学理论和工程上的技术来指导软件的开发，从而达到用较少的投资、较短的时间获得高质量的、可靠的软件的目的"。软件工程学包括的面很广，有基础理论研究，也有应用研究以及实际开发。除了技术问题之外，它还涉及与开发软件有关的所有活动，例如计算机可靠性理论、管理学、经济学、心理学、法律与道德等方面。本书只讨论软件工程中的主要技术问题。

当软件工程学还处在学术研究阶段时，已对软件开发的实践产生了巨大的影响。1971年IBM公司运用一些软件工程技术成功地研制了纽约时报情报库系统和空间实验室的飞行模拟系统，这有力地证明了软件工程化是软件开发的正确道路。软件工程的出现是开发软件上从单纯的程序设计的技巧到科学化开发软件的转折，这极大地推动了软件产业的发展。目前，软件工程学已成为计算机科学中一个重要分支，研究软件工程学无疑将促进计算机推广应用的步伐，直接或间接地产生巨大的社会效益和经济效益。

1.2 软件和软件生存周期

计算机技术从50年代到80年代有了突飞猛进的发展，人们对许多问题的看法也发生了根本的变化，我们有必要对软件的一些基本概念重新进行讨论。软件和软件生存周期是软件工程学中的两个重要的概念。

1.2.1 软件

软件(Software)这个术语是在1964年后逐渐被人们采用的，现已为大家所接受并被广泛使用。在以前相当长的一段时间里，软件被认为就是程序或程序系统。随着软件规模增大，软件作为一种社会产品，我们需要对软件进行再认识。

从计算机系统来看，除了硬件之外就是软件。硬件是看得见、摸得着的具体的物理装置，而软件应该是事先编好的程序全体，以及相应的资料或规范书。软件不同于普通的解题程序，它存放在计算机里，是合理使用计算机系统的组成部分，它利用计算机自身的功能，充分发挥计算机的效能。因此，软件也是程序，但又不同于程序。从历史发展来看，软件的概念是程序的概念在认识上的一次飞跃。软件和程序既有联系的一面，又有特性上的差异：

1)软件的工作量和复杂程度远远地超过程序。这一点是容易理解的。软件作为一个程序系统，当然工作量大，复杂性高。

2)程序并不成为社会财富，而软件却是一笔很大的社会财富。这一点很重要。在以前，为了解决特定的问题而编制的程序大多具有个体化特点。可是软件就不同，软件最大的特点就是提供给别人使用，不能提供给别人使用的程序就不是软件。当然，早先人们也注意到这个问题，搞了个叫作程序库的东西，用目前的观点看来，它可以称得上是软件了。

3)程序不需要维护，而软件需要花大力气进行维护和修改。这和前二点相关，道理是非常清楚的。

4)编出的程序是要使它能在计算机上运行，而编出的软件不仅要能在计算机上运行，还要使人们能够看懂。这就说明了由于工作的对象不同，使软件具有不少程序所不具有的属性。

由此可见，所谓软件就是程序以及开发、使用和维护程序所需的所有文档资料。软件由两部分组成，一部分是编好的程序全体，包括程序所处理的全部数据和数据结构，另一部分是资料和规范。

软件既然要给别人使用，又要维护，自然不能光有能运行的程序，还要能被人们所接受和阅读，因此必须提供资料和规范书。另外，在整个软件的研制过程中，各个程序员之间和各个程序模块之间的约定、功能等资料也是一种互相交流和检查的手段。因此，在整个软件研制过程中，及时地按一定规格产生各种文档是研制工作的有机组成部分，必须给予充分重视。

1.2.2 软件的生存周期

在过去很长的时间里，人们认为软件开发就是编制程序。随着计算机应用范围的不断扩大，软件规模越来越大，逻辑越来越复杂，这样的理解是不合适的。作为工程化的一般特征，软件产品和其他工业产品一样，也应有设计、测试、检查等不可缺少的过程，由此，逐步形成了“软件生存周期”(Software Life Cycles)的概念。即它是一个从用户需求开始，经过这项软件开发成功投入使用，在使用中不断增补修订，直到最后决定停止使用，让位于新的软件的过程。

一个软件的生存周期可以划分成若干个互相区别而又彼此联系的阶段，每个阶段中的工作都以上一阶段工作的结果作为依据，同时也为下一阶段的工作提供前提。软件生存周期阶段的划分，使每个阶段的任务相对独立且比较简单，便于不同人员分工协作，从而降低了整个软件开发工程的困难程度。同时，每个阶段都采用经过验证行之有效的管理技术和方法，并且从技术和管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发的全过程以一种有条不紊的方式进行，保证了软件的质量、降低成本、合理使用人才，进而提高软件开发的生产率。

由于工作的对象和范围的不同、经验的不同，对软件生存周期的阶段划分也不尽相同。但是，这些不同划分中软件开发工作的流程基本上是相同的。国家标准 GB 8566—88《计算机软件开发规范》将软件生存周期分为以下八个阶段：可行性研究与计划、需求分析、概要设计、详细设计、实现(包括单元测试)、组装测试(即集成测试)、确认测试、使用和维护。图 1-2 给出了

软件生存周期的瀑布模型。在实际应用中，我们常把这些阶段粗分为软件计划、软件开发和软件维护三个阶段。

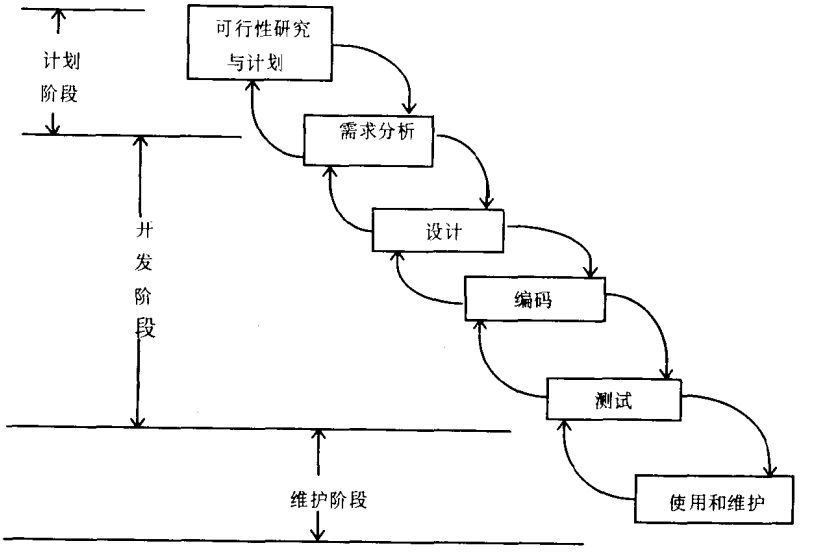

图 1-2　软件生存周期的瀑布模型

软件生存周期的每个阶段都要产生一定规格的文档移交给下一阶段，使下一阶段在所提供的文档基础上继续开展工作。国家标准 GB 8567—88《计算机软件产品开发文件编制指南》建议在软件的开发过程中编制下述 14 种文件，即可行性研究报告；项目开发计划；软件需求说明书；数据要求说明书；概要设计说明书；详细设计说明书；数据库设计说明书；用户手册；操作手册；模块开发卷宗；测试计划；测试分析报告；开发进度月报以及项目开发总结报告。对于一个软件而言，其生存周期各阶段与各种文件编写的关系可见表 1-1 所示。其中有些文件的编写工作可能要在若干阶段中延续进行，先编写初步的，然后随着开发工作逐步详细和完善。

软件生存周期各阶段中的文件编制　　表 1-1

	可行性研究与计划	需求分析	设计	编码	测试	使用与维护		可行性研究与计划	需求分析	设计	编码	测试	使用与维护
可行性研究报告	——						数据库设计说明书			——			
项目开发计划	——	——					模块开发卷宗				——	——	
软件需求说明书		——					用户手册		——	——	——		
数据要求说明书		——					操作手册			——	——		
测试计划		——	——				测试分析报告					——	
概要设计说明书			——				开发进度月报	——	——	——	——	——	
详细设计说明书			——				项目开发总结	——	——	——	——	——	

应当指出，由于理解能力及环境变化等限制，软件系统的实际开发工作不可能直线地通过分析、设计、编码、测试等阶段，出现各阶段间的回复是不可避免的，这就构成了图 1-2 中各步骤间的向上流线。经验表明，失误造成的差错越是发生在生存周期前期，在系统交付使用时造成的影响和损失越大，要纠正它所花费的代价也越高。因此，在每个阶段完成以后都要进行严格的复审，尽量减少工作中的失误。

图 1-3 说明了在典型的情况下，软件生存周期各阶段的工作量所占的比重。图 1-3a）说明维护工作量要占整个生存周期的三分之二，图 1-3b）说明测试阶段工作量约占整个开发期工

作量的一半。

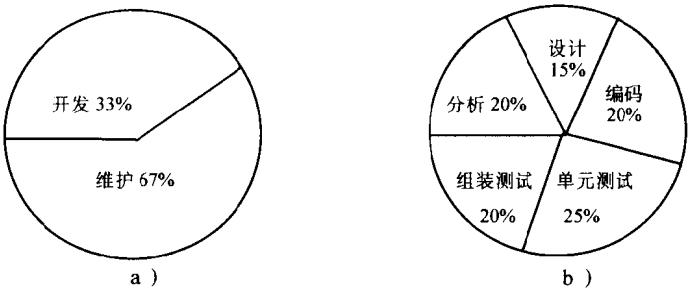

图 1-3　软件生存周期各个阶段工作量分配

下面扼要介绍软件生存周期各个阶段的基本任务和工作概况。

(1) 系统定义和软件计划

确定软件系统必须满足的总体目标，给出它的功能、性能、可靠性以及接口等方面的设想；研究完成该项软件任务的可行性，探讨解决问题的方案；估价系统的投资(包括计算机软、硬件资源和人力等)和收益，拟定完成系统的各个目标的策略以及进度，制定完成开发任务的实施计划。

(2) 需求分析

对开发的软件系统的定义进行详细的调查和分析，充分理解用户的需求，确定哪些需求是可以满足的，明确这些需求的逻辑结构，并加以确切地描述，写出软件需求说明书(Software Requirement Specifications)及初步的用户手册。

软件需求说明书 SRS 是软件系统的逻辑模型，说明软件系统干什么，这是以后各阶段工作的基础，是用户和软件人员双方充分讨论交流后达成的协议书，因此，用户和软件人员都应有代表参加这个阶段的工作。

(3) 软件设计

设计是软件工程的技术核心，其基本任务是将用户要求转换成一个具体的软件系统的设计方案。该阶段包括概要设计(或称总体设计)、详细设计等步骤，每一个步骤考虑的详细程度有所不同。概要设计是在软件需求说明书的基础上建立软件的系统结构，包括数据结构和模块结构。模块结构中的每个模块意义明确且和某些用户需求相对应，进而进行详细设计，对每个模块进行具体的描述，确定模块的功能、接口和实现方法，以便为程序编写打下基础。

所有设计中的考虑都应以设计说明书的形式加以详细描述。

(4) 编码

把软件设计转换成计算机可以接受的程序。如果前段工作搞得好，相对地说，本阶段工作就比较简单，初级程序员也可以参加这个阶段的工作。自然，交付的源程序清单应该是结构良好、清晰易读的，且与设计相一致的。

(5) 测试

软件测试是保证软件质量的重要手段，其任务是发现并排除错误，它通常又可分为单元测试(或称模块测试)，组装测试、确认测试等步骤。测试最好由另一个独立的部门(不参加该软件系统的设计和编写的人员)来完成，这样可以提高测试的效率。经过测试修改就得到了可运行的软件系统，交付用户使用。整个测试过程都要记录在测试分析报告中。

(6) 运行和维护

已交付的软件投入正式使用便进入运行阶段。在运行阶段，需要对软件系统进行修改，其原因可能有：运行中发现了错误需要修正；为了适应变化了的软件工作环境，需作适当变更；为了增强软件功能需作变更。每一项维护活动都应该准确地记录下来，作为正式的文档资料加以保存。

1.3 软件的质量

在60年代末期，所开发的大型程序存在着潜在的错误，已经是司空见惯了，这不是缺乏试验而造成的。直到现在，生产一个大型的没有错误的程序似乎也不太可能。在当时，计算机发展已有20多年，一方面硬件成本每隔2年～3年下降一半，内存和外存的成本每年下降40%左右，硬件的性能价格比每10年改进一个数量级；而另一方面，计算机的软件却很少能在成本、时间进度、功能规模、维护能力等方面达到要求，特别是在可靠性方面不能符合人们的需要。硬软件如此一对比，无怪乎人们要惊呼“软件危机”。抱怨毕竟无济于事，人们开始审查以前对待软件的态度和程序设计中的问题。以往具有个体特性的程序设计常常注重时空的效率，即节省运行时间和存储空间，由程序过渡到软件也就按习惯继承过来了，这对软件是相当不利的。

如何评价一个软件的优劣，也就是衡量软件质量的标准是什么？这是一个十分重要的问题。对于软件质量的标准，由于人们经历的不同，研究问题的对象不同，想达到的目标不同，所以看法不尽一致，提出的要求和用词也不尽一致。P. B. Hansen 认为：一个好的程序软件必须是简明的、可靠的和可适应的。没有简明性就不可能使人理解该软件的目的和细节，没有可靠性就不可能让人们在使用中完全依赖于它，没有可适应性来修改程序的需求，最终将成一个完全僵死的东西。下面我们从可维护性、可靠性、可理解性和效率等方面来讨论软件的质量。

1）可维护性(Maintainability)

软件系统经过测试后仍含有错误，所以在运行阶段还需要继续排错，修改扩充，这种维护的工作量是相当大的，例如IBM公司OS/360系统的每一个版中有1000个大大小小的错误。维护工作也是相当困难的，有人统计，如果程序员一次要修改5～10个语句，则修改成功的可能性是50%，如果一次修改40～50个语句，则修改成功的可能性下降至20%，因此，软件维护并不如想像的那么简单。

随着软件规模的扩大和复杂性的增加，也由于人工费用所占比例的上升，软件维护问题已显得越来越重要了。人们已意识到，一个软件系统即使其他方面都相当理想，但是如果不容易维护，它将不会有什么实际使用价值，所以，“可维护性”成为评价软件质量的重要方面。

“可维护性”通常指对程序的易阅读、易发现和纠正错误、易修改扩充的程度。一个软件系统的结构是否好，各类文档是否完整，对维护起着重要的影响。为了使软件容易维护，在开发早期就必须采用一定的技术，仔细分析、精心设计，并建立完整的文档。如果在早期不认真做好这些工作，就可能在后期造成许多测试和维护上的问题，因此，防患于未然十分必要。

本书第八章还要进一步讨论软件维护问题。

2）可靠性(Reliability)

可靠性通常包括两个概念：正确(Correctness)和健壮(Robustness)。

正确性的含义是指软件系统本身没有错误，在预期的环境条件下能正确地完成期望的功能所能达到规定的程度。

健壮性的含义是指在硬件发生故障或输入数据不合理等意外的环境条件下，系统仍能适当地工作的能力程度。

软件的正确性是受条件限制的概念，它依赖于软件的外部说明和工作状态，也依赖于它的内部结构。对一个小型的程序，我们可以要求它是完全正确的。而对一个大型的软件系统，它所处的环境千变万化，几乎是无限的，例如硬件各部分可能出现多种类型的故障，用户也可能有意无意地输入一些不合理的数据，使系统遭到意想不到的破坏，所以，我们不能期望一个大型软件系统在任何情况下都是正确的，因此就引进了新的概念——健壮性。一个健壮的程序在遇到意外的情况时，能按某种预期的方式作出适当的处理，例如它能意识到发生了意外，及时地通知管理人员，并有效地控制事故的蔓延，不致于丢失重要的信息，事后也能较快地从故障中恢复到正常状态，避免严重的后果发生。

正确性与健壮性是相互补充的。一个正确的程序，它可以不检查输入数据是否合理，这就可能造成严重的后果，所以这个正确的程序并不可靠。一个可靠的程序却不一定是完全正确的，程序设计中的个别很少发生的错误并没有使软件不可使用，在大多数情况下能够可靠地工作。当然，可靠的程序应该是基本正确的。

总的说来，可靠的软件系统在正常情况下能够正确地工作，而且在意外情况下，亦能适当地作出处理。随着计算机使用的范围越来越广，有些软件系统的故障可能造成经济上的巨大损失，甚至是灾难，如核反应堆泄漏等。所以，软件系统的可靠性无疑是绝对重要的，人们宁可在开发时多化些代价，提高系统的可靠性，与发生事故后造成的损失相比，这些代价还是值得的。

3）可理解性(Understandability)

在相当长一段时间中，人们一直认为程序只是提供给计算机的，而不是给人阅读的，所以只要它逻辑正确，计算机能按其逻辑正确执行就足够了，至于它是否易于被人理解则是无关紧要的。

但是随着软件规模的增大，人们逐步看到，在整个软件生存期中，为进行测试、排错或修改，开发人员经常需要阅读本人或他人编写的程序和各种文档。同时，易于理解的软件，无疑将提高维护的工作效率。所以，可理解性应该是评价软件质量的一个重要方面。

可理解性可以有两重含义，一方面是指系统的内部结构清晰，易于软件人员阅读和理解；另一方面也指系统的人－机界面简明清晰，便于用户使用。

4）效率(Efficiency)

效率是指系统是否能有效地使用计算机资源，如时间和空间等。从计算机问世以来，提高效率一直为人们所重视的。近年来，由于硬件价格大幅度下降，追求效率已不如以前那么重要了。值得注意的是追求效率与追求可维护性、可靠性等往往是相互抵触的，片面地强调节省时间和空间，设计出来的系统就可能结构较复杂，难以理解和修改。追求可靠性一般也需付出一定的时间和空间作为代价。在硬件价格下降、人工费用上升的情况下，只要达到性能要求，人们宁肯牺牲一点效率来换取较好的可维护性和可靠性。另外，编制软件本身也存在一个效率问题，程序员的工作效率比程序的效率远为重要。当然，程序结构清晰，其效率自然提高，如果程序逻辑紊乱，其效率肯定低下。

除了上面讨论之外，还可以举出其他一些反映软件质量的特性。图 1-4 是 B. Boehm 提出的软件质量图，下面只解释图中最右边各因素的含义，不再作详细讨论。

1）设备独立性(deviceindependenty)：不依赖于特定的硬件设备而能工作的程度。

2）自包含性(self－containedness)：不依赖于其他程序仅靠自身能实现功能的程度。