计算机软件测试

JISUANJIRUANJIANCESHI

主编：毛靖添 景晓宇 王春慧

黑龙江地图出版社

计算机软件测试

JISUANJI RUANJIAN CESHI

主编　毛靖添　景晓宇　王春慧

哈尔滨地图出版社

·哈尔滨·

图书在版编目(CIP)数据

计算机软件测试/毛靖添,景晓宇,王春慧主编.-哈尔滨:哈尔滨地图出版社,2007.7
ISBN 978-7-80717-682-4

Ⅰ.计... Ⅱ.①毛... ②景... ③王...Ⅲ.软件-测试
IV.TP311.5

中国版本图书馆 CIP 数据核字(2007)第 110331 号

哈尔滨地图出版社出版、发行
(地址:哈尔滨市南岗区测绘路 2 号 邮政编码:150086)
哈尔滨市民强印刷厂印刷
开本:787 mm×1 092 mm 1/16 印张:12.75 字数:318 千字
2007 年 7 月第 1 版 2007 年 7 月第 1 次印刷
ISBN 978-7-80717-682-4
印数:1~1 000 定价:22.00 元

前　言

计算机软件测试是软件工程实践中重要的分支，也是软件工程学中的重要内容。因此，本书不仅深入讲述了软件测试的各个方面，包括软件测试的基本理论和方法（单元测试、集成测试、系统和验收测试、测试文档的编写、测试用例的设计和软件度量），而且还详细介绍了软件测试的解决方案，从而指导读者在软件生命周期各个阶段合理地选择恰当的测试技术与测试工具，并有效地运用到软件开发项目中，通过测试确保最终开发出高质量、高可靠性的软件，以供相关人员借鉴和参考。

全书共 10 章，既有一般的理论知识，又有相关的设计用例分析。编写人员分工如下：第 1，2，6 章由毛靖添编写（约 10.3 万字）；第 4，5，8，10 章由景晓宁编写（约 10.3 万字）；第 3，7，9 章由王春慧编写（约 10.1 万字）；全书最后由毛靖添统稿定稿。

本书的出版得到了各方面的支持，在此对参考文献的作者及帮助本书顺利出版的同志表示深切的谢意。

由于时间、学识有限，书中不当之处在所难免，欢迎读者提出宝贵建议和意见，以便再版时更正。

作　者

2007 年 6 月

1 软件测试概述

根据传统的测试理论和角度，会认为测试是软件生存周期中的一个环节。虽然这种提法现在看并不确切，因为为了保证软件的质量，测试应该着眼于整个软件生存周期。但是，这样在提到测试的时候不可避免地涉及到软件工程的相关理论和实践。因此，本章先介绍软件工程的基本理论、方法和原则，然后简单介绍测试技术的发展。

1.1 软件

计算机软件是整个计算机系统中具体实现各种功能和操作的核心部分。软件工程即采用工程的概念、原理、技术和方法来开发和维护软件，将工程管理技术成功的经验和思想与具体软件的开发过程、研究技术相结合，形成一整套适合于计算机软件开发的方法、规范和技术。

1.1.1 软件的概念、特点

计算机软件是程序、数据和相关文档的完整集合。其中，程序是按实现设计的功能和性能要求执行的指令序列，数据是使程序能正确运行的数据结构，文档是与程序开发、维护和使用有关的图文资料，记录软件开发活动和阶段成果，具有永久性，可供人或机器阅读的特点，可用于专业人员和用户之间的通信和交流。

为了要深入地进行计算机软件的开发和研究，首先要了解计算机软件的特点和开发规律。计算机软件可归纳为以下几个特点：

（1）软件是逻辑实体，而不是物理实体，因此软件具有抽象性。这个特点使它与计算机硬件或其他工程对象有着明显的差别。人们可以把它记录在介质上却无法看到它的形态，而必须通过测试、分析、思考、判断去了解它的功能、性能及其他特性。

（2）软件开发更依赖于开发人员的业务素质、智力、人员的组织、合作和管理。软件开发、设计几乎都是从头开始，投入的成本是比较高的，而且成本和进度很难估计。

（3）软件开发是一个复杂过程，这种复杂性可能来自它反映的实际问题的复杂性，也可能来自程序逻辑结构的复杂性。

（4）软件开发成功后，只需对原版进行复制，其研制成本远远大于生产成本。

（5）软件的开发和运行往往受到计算机系统的限制，对计算机系统有着不同程度的依赖性。为解决这种依赖性，在软件开发中提出了软件移植问题，并且把软件的可移植性作为衡量软件质量的因素之一。

（6）软件在使用过程中没有硬件那样的磨损、老化问题。然而软件存在退化问题并且它存在潜伏错误，造成维护工作复杂：

① 纠错性维护——改正运行期间发现的潜伏错误；

② 完善性维护——提高或完善软件的性能；
③ 适应性维护——修改软件，以适应软硬件环境的变化；
④ 预防性维护——改进软件未来的可维护性和可靠性。

1.1.2 软件的发展

1.程序设计阶段

计算机软件发展的早期阶段(20 世纪 60 年代中期以前)是程序设计阶段。这个阶段硬件已经通用化，而软件开发处于个体化生产状态。在这一阶段中，软件还没有系统化的开发方法。目标主要集中在如何提高时空效率上，文档根本不存在。

2.程序系统阶段

计算机软件系统发展的第二阶段(20 世纪 60 年代中期到 70 年代末期)是程序系统阶段。引入了多道程序设计、多用户系统和人机交互的新概念。实时系统和第一代数据库管理系统出现了。软件开发已进入了作坊式生产方式，即出现了“软件车间”。软件开发开始形成产品，可以在较宽广的范围中应用。

在软件的使用过程中，当发现错误时，当用户需求或硬件环境发生变化时都需要修改软件，这些活动称为软件维护。在软件维护上的花费以惊人的速度增长。更为严重的是软件的个体化特征使得它们根本不能维护，到 20 世纪 60 年代末，“软件危机”出现了。

“软件危机”使得人们开始对软件及其特性进行更深一步的研究，人们改变了早期对软件的不正确看法。早期那些被认为是优秀的程序常常很难被别人看懂，通篇充满了程序技巧。现在人们普遍认为优秀的程序除了功能正确，性能优良之外，还应该容易看懂、容易使用、容易修改和扩充。

3.软件工程阶段

计算机系统发展的第三阶段(20 世纪 70 年代中期到 20 世纪 80 年代末期)跨越了近 10 年。在这一阶段，以软件的产品化、系列化、工程化、标准化为特征的软件产业发展起来，有了可以遵循的软件工程化的设计原则、方法和标准。软件产品急剧增加，质量也有了很大的提高。分布式系统、计算机网络通信以及对“即时”数据访问需求的增加都对软件开发者提出更高的要求。

4.面向对象的软件工程阶段

计算机系统发展的第四阶段(20 世纪 80 年代末期开始至今)是一个软件产业大发展的时期，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。

1.1.3 软件危机

落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题和难题的现象称为软件危机。1968 年北大西洋公约组织的计算机科学家在前联邦德国召开的国际学术会议上第一次提出了“软件危机”(software crisis)这个名词。也是在这次会议上 Feitz Bauer 首先提出了“软件工程”概念，引入了现代软件开发的方法，希望用工程化的原则和方法来克服软件危机。

概括来说，软件危机包含两方面问题：

① 如何开发软件,以满足不断增长,日趋复杂的需求;

② 如何维护数量不断膨胀的软件产品。

1. 软件危机的表现

(1)对软件开发成本和进度的估计常常很不准确。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,从而降低了开发商的信誉,引起用户不满。

(2)用户对已完成的软件不满意的现象时有发生。

(3)软件产品的质量往往是靠不住的。

(4)软件常常是不可维护的。

(5)软件通常没有适当的文档资料。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居高不下,且逐年上升。由于微电子学技术的进步和硬件生产自动化程度不断提高,硬件成本逐年下降,性能和产量迅速提高。然而软件开发需要大量人力,软件成本随着软件规模和数量的剧增而持续上升。从美、日两国的统计数字表明,1985 年度软件成本大约占总成本的 90%。

(7)开发生产率提高的速度远跟不上软件需求。软件产品"供不应求"的现象使人们不能充分利用现代计算机硬件提供的巨大潜力。

概括地讲,软件危机主要矛盾集中体现在软件开发周期、软件开发成本和软件质量三个方面。

2. 产生软件危机的原因

产生软件危机的原因,一方面是与软件本身的特点有关;另一方面是与软件开发和维护的方法不正确有关,例如:

① 用户对软件需求的描述不精确。

② 软件开发人员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。

⑤ 缺乏有力的方法学的指导和有效的开发工具的支持。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,人们显得力不从心。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。

此外,人工智能与软件工程的结合成为 20 世纪 80 年代末期活跃的研究领域。基于程序变换、自动生成和可重用软件等软件新技术研究也已取得一定的进展,把程序设计自动

化的进程向前推进一步。在软件工程理论的指导下，发达国家已经建立起较为完备的软件工业化生产体系，形成了强大的软件生产能力。软件标准化与可重用性得到了工业界的高度重视，在避免重用劳动，缓解软件危机方面起到了重要作用。

1.2 软件工程的概念

1.2.1 软件工程的定义

软件工程诞生于20世纪60年代末期，它作为一个新兴的工程学科，主要研究软件生产的客观规律性，建立与系统化软件生产有关的概念、原则、方法、技术和工具，指导和支持软件系统的生产活动，以期达到降低软件生产成本、改进软件产品质量、提高软件生产率水平的目标。软件工程学从硬件工程和其他人类工程中吸收了许多成功的经验，明确提出了软件生命周期的模型，发展了许多软件开发与维护阶段适用的技术和方法，并应用于软件工程实践，取得了良好的效果。

在软件开发过程中人们开始研制和使用软件工具，用以辅助进行软件项目管理与技术生产，人们还将软件生命周期各阶段使用的软件工具有机地集合成为一个整体，形成能够连续支持软件开发与维护全过程的集成化软件支持环境，以期从管理和技术两方面解决软件危机问题。

Feitz Bauer 在 NATO(North Atlantic Treaty Organization 北大西洋公约组织)会议上给出的定义：

“软件工程是为了经济地获得可靠的和能在实际机器上高效运行的软件而确立和使用的健全的工程原理(方法)。”

IEEE【IEE83】给出的软件工程定义：

“软件工程是开发、运行、维护和修复软件的系统方法。”

IEEE【IEE93】给出了一个更加综合的定义：

“将系统化的、规范的、可度量的方法应用于软件的开发、运行和维护的过程，即将工程化应用于软件中。”

被国内软件工程专家和业界普遍认可的定义：

“软件工程是用计算机科学、数学管理科学和工程的原则与方法开发、维护计算机软件的有关技术和管理方法。其中，计算机科学、数学用于构造模型与算法，工程科学用于制定规范、设计范型、评估成本及确定权衡，管理科学用于计划、资源、质量、成本等管理。”

软件工程由方法、工具和过程三部分组成，称软件工程的三要素。

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。

软件工程中的过程是指为了完成软件产品，在软件工具的支持下由软件开发人员执行的一系列软件工程活动，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。通常包括 4 种活动：

P(Plan)——软件计划及规格说明过程:规定软件的功能及其运行时的限制。

D(Do)——软件开发过程:产生满足规格说明的软件。

C(Check)——软件确认过程:确认软件能够完成客户提出的要求。

A(Action)——软件演进过程:为满足客户的变更要求,软件必须在使用的过程中演进。

事实上,软件工程过程是一个软件开发机构针对某类软件产品为自己规定的工作步骤,它应当是科学的、合理的,否则必将影响软件产品的质量。

1.2.2 B.W.Beohm 基本原则

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 条基本原则。这 7 条原则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。下面就简单介绍软件工程的这 7 条原则:

(1)用分阶段的生存周期计划严格管理

这条基本原理是应该把软件生存周期划分成若干个阶段,并相应地制订出切实可行的计划,然后严格按照计划对软件开发与维护工作进行管理。应该制订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。各级管理人员都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,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,所花费的代价越高。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,从而可以最小的代价改正错误。因此,这是一条必须坚持的重要原则。

(3)实行严格的产品控制

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改变需求的要求。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)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,难于评价和管理。为了更好地进行评价与管理,应根据软件开发的

总目标和完成期限，尽量明确地规定软件开发小组的责任和产品标准，从而使所得到的结果能清楚地审查。

（6）合理安排软件开发小组的人员

软件开发小组人员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，开发效率可能高几倍至几十倍，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率，反而由于误解等原因可能增加出错的概率。

（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则，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 软件工程的知识结构与目标

1.软件工程知识结构

2001 年 5 月 ISO/IEC JTC 1（ISO 和 IEC 的第一联合技术委员会）发布了《SWEBOK 指南 V0.95（试用版）》（Guide to the Software Engineering Body of Knowledge，简称 SWEBOK）SWEBOK 把软件工程学科的主体知识分为 10 个知识领域。

①软件需求；②软件设计；③软件构造；④软件测试；⑤软件维护；⑥软件配置管理；⑦软件工程管理；⑧软件工程过程；⑨软件工程工具和方法；⑩软件质量。

2. 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，允许对软件系统进行修改而不增加其复杂性。它支持软件调试与维护。

（2）有效性，指软件系统的时间和空间效率。这是一个应当努力追求的重要目标。

（3）可靠性，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。

（4）可理解性，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，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，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费用。

(7)可适应性,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可增强软件产品的可适应性。

(8)可移植性,是指软件从一个计算机系统或环境移植到另一个上去的难易程度。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可提高可移植性。而应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。可移植性支持软件的可重用性和可适应性。

(9)可追踪性,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,是指多个软件元素相互通信并协同完成任务的能力。

1.2.4 软件工程的原则

(1)抽象,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,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)局部化,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。

(4)一致性,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;系统规格说明应与系统行为保持一致;用于形式化规格说明的公理系统应保持一致。

(5)完全性,软件系统不丢失任何重要成分,完全实现所需的系统功能的程度。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。

(6)可验证性,开发大型软件系统需要对系统逐层分解。分解是人类分析解决复杂问题的重要手段和基本原则,其基本思想是从时间上或从规模上将一个复杂抽象问题分成若干个较小的、相对独立的、容易求解的子问题,然后分别求解。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。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 软件生存周期与软件开发模型

1.3.1 软件生存周期

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。研究软件生存周期是为了更科学、有效地组织和管理软件的生产,从而使产品更可靠、更经济。

采用软件生存周期来使软件开发分阶段进行,前一阶段任务的完成是后一阶段任务的前提和基础,而后阶段通常是将前阶段提出的方案进一步具体化。每个阶段的开始与结束都有严格的标准,每个阶段结束之前都要接受严格的技术和管理评审。

采用软件生存周期的划分方法,使每阶段的任务相对独立,有利于简化问题且便于人员分工协作。而且其严格科学的评审制度保证了软件的质量,提高了软件的可维护性,大大提高了软件开发的成功率和生产率。可将软件生存周期划分为 3 个阶段共 8 个过程。

3 个阶段是:计划阶段、软件开发阶段、维护阶段。

8 个过程有:问题定义、可行性研究、需求分析、概要设计、详细设计、编码、测试、使用与维护。

1.计划阶段

计划阶段包括 3 个过程:问题定义、可行性研究、需求分析。

(1)软件定义或称系统定义,基本任务是确定软件系统的工作需求,为软件系统的提供功能和性能的依据。

(2)可行性研究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。在此基础上还要制订初步的项目计划,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使部门负责人作出是否继续进行该项目决定的重要依据。

(3)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、软件系统的验收测试准则和初步的用户手册。

软件系统需求一般由用户提出。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。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,最终得到用户和开发者的确认。

需求分析阶段的主要成果有软件需求规格说明、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(Software Requirements Specification,即 SRS),是一个

关键性的文档。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、开发标准和验收准则等。

2. 软件开发阶段

软件开发阶段由概要设计、详细设计、编码、测试 4 个过程组成。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。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”。

下面将简单地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——总体设计

①任务：是对需求规格说明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，制订组装测试计划，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。

②实现的途径：使用某种方法和工具，可以选择 SD 或 Jackson 方法等(1.5.1)。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低耦合度，而功能模块内部具有较高的内聚度。

③阶段性成果：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

①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。

②实现的途径：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(PDL，Procedure Design Language)、判定表或判定树等工具进行描述，也可以采用面向对象的设计方法等。

③阶段性成果：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)编码

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尽可能保持一致。

编码阶段应注意遵循编程标准、养成良好的编程风格，以便编写出正确的便于理解、调试和维护的程序模块。

(4)单元测试

每编写出一个程序模块的源程序，调试通过后，即对该模块进行测试，这称为单元测试。

阶段性成果:按一定规则存在盘上的通过单元测试的各功能模块的集合、详细的单元测试报告等文档。

(5)组装测试—集成测试

根据概要设计提供的软件结构、各功能模块的说明和组装测试计划,把经过单元测试检验的模块按照某种选定的策略逐步进行组装和测试。

①主要任务:测试系统各模块间的连接是否正确,系统或子系统的正确处理能力、容错能力、输入/输出处理是否达到要求。

②阶段成果:

第一,应是满足概要设计要求、可运行的软件系统和源程序清单;

第二,组装测试报告等文档。

(6)验收测试——确认测试

①任务:按照验收测试计划和准则对软件系统进行测试,看其是否达到了需求规格说明中定义的全部功能和性能等方面的需求。

②阶段性成果:确认测试结束时,应生成验收测试报告、项目开发总结报告,并向用户提交源程序清单、最终用户手册、操作手册等文档资料。

最后,由专家、用户负责人、软件开发和管理人员组成的软件评审小组要对软件验收测试报告、测试结果和软件进行评审,通过后,软件产品正式通过验收(即完成了开发合同),可以交付用户使用了。

3. 软件的使用与维护

软件使用与维护阶段的任务是通过各种维护活动使软件系统持久地满足用户的需求。每项维护活动实质上都是一次压缩和简化了的软件定义和软件开发过程,都要经历提出维护要求、分析维护要求、提出维护方案、审批维护方案、确定维护计划、修改软件设计、修改程序、测试程序、评审、验收等步骤。

应当指出,软件在使用的过程中,应及时收集被发现的软件错误,并定期撰写"软件问题报告";而每一项维护活动都应该准确地记录下来,并作为正式的文档资料保存。

据统计,软件维护人员为了分析和理解原软件系统所花费的工作量约占整个维护工作量的60%以上。在软件开发的过程中应重视对软件可维护性的支持。

1.3.2 软件开发模型

为了反映软件生存周期内各种工作应如何组织及软件生存周期各个阶段应如何衔接,需要软件开发模型给出直观的图示表达。归纳起来说,软件开发模型(又称为软件生存周期模型)是软件项目开发和维护的总体过程思路的框架。它指出了软件开发过程各阶段之间的关系和顺序,是软件开发过程的概括。它为软件开发过程提供原则和方法,并为软件工程管理提供里程碑和进度表。因此,软件开发模型也是软件工程的重要内容。

1.瀑布模型

瀑布模型(waterfall model)是由 W. Royce 于 1970 年提出来的,又称为软件生存周期模型。其核心思想是按工序将问题化简,将功能的实现与设计分开,便于分工协作。瀑布模型严格按照软件生存周期各个阶段来进行开发,瀑布模型将软件生命周期划分为制订计

划、需求分析、软件设计、程序编写、软件测试和运行维护等六个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。上一阶段的输出即是下一阶段的输入，并强调每一阶段的严格性。它规定了各阶段的任务和应提交的成果及文档，每一阶段的任务完成后，都必须对其阶段性产品（主要是文档）进行评审，通过后才能开始下一阶段的工作。因此，它是一种以文档作为驱动的模型，如图 1-1 所示。

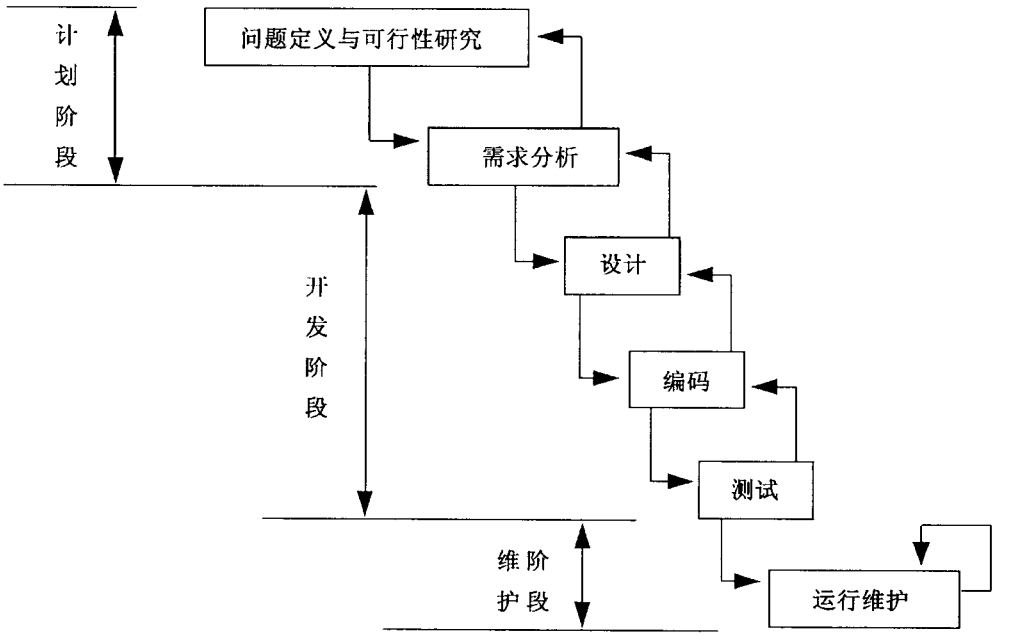

图 1-1 瀑布模型示意图

瀑布模型包括以下活动：

①系统 / 信息工程建模：因为软件总是一个大系统的组成部分，所以一开始应该建立所有系统成分的需求，然后再将其中某个子集分配给软件。整个系统基础是以软件作为其他成分如硬件、人及数据库的接口。系统工程和分析包括了系统级收集的需求，以及一小部分顶层分析和设计。信息工程包括了在战略商业级和商业领域级收集的需求。

②软件需求分析：需求收集过程集中于软件之上。要理解待建造程序的本质，软件工程师必须了解软件的信息领域，以及需求的功能、行为、性能和接口。系统需求和软件需求均要文档化，并与用户一起复审。

③设计：软件设计实际上是一个多步骤的过程，集中于程序的四个完全不同的属性上：数据结构、软件体系结构、界面表示及过程细节。设计过程将需求转换成软件表示，在编码之前可以评估其质量。象需求一样，设计也要文档化，并且是软件配置的一部分。

④代码生成：设计必须转化成机器可读的形式。代码生成这一步就是完成这个任务的。如果设计已经表示得很详细，代码生成可以自动完成。

⑤测试：一旦生成了代码，就可以开始程序测试。测试过程集中于软件的内部逻辑－保证所有语句都测试到，以及外部功能－即引导测试去发现错误，并保证定义好的输入能够产生与预期结果相同的输出。

⑥维护：软件在交付给用户之后不可避免地要发生修改，软件维护重复以前各个阶段，不同之处在于它是针对已有的程序，而非新程序。

瀑布模型提供了软件开发的基本框架，有利于大型软件开发过程中人员的组织、管理，有利于软件开发方法和工具的研究与使用。

但是，这种模型的线性过程太理想化，已不再适合现代的软件开发模式，几乎被业界抛弃，其主要问题在于：

(1)瀑布模型强调文档的作用，并要求每个阶段都要仔细验证，各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量；

(2)由于开发模型是线性的，在需求分析阶段，当需求确定后，无法及时验证需求是否正确、完整。用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险；

(3)作为整体开发的瀑布模型，由于不支持产品的演化，缺乏灵活性，对开发过程中很难发现的错误，只有在开发后期的测试阶段才能发现，才能暴露出来，从而使软件产品难以维护，进而带来严重的后果。

瀑布模型一般适用于功能、性能明确、完整、无重大变化的软件系统的开发。例如，操作系统、编译系统、数据库管理系统等系统软件的开发。应用有一定的局限性。

应该认识到，“线性”是人们最容易掌握并能熟练应用的思想方法。当人们碰到一个复杂的“非线性”问题时，总是千方百计地将其分解或转化为一系列简单的线性问题，然后逐个解决。一个软件系统的整体可能是复杂的，而单个子程序总是简单的，可以用线性的方式来实现。线性是一种简洁，是一种简单。在瀑布模型基础上，活用线性模型的原理，不断改进和提高形成了多种使用于不同场合的模型。例如，增量模型实质就是分段的线性模型，螺旋模型则是接连的弯曲了的线性模型。

2.原型模型

原型模型(prototyping model)的基本框架是软件开发人员根据用户提出的软件基本需求快速开发一个原型，以便向用户展示软件系统应有的部分或全部功能和性能，在征求用户对原型的评价意见后，进一步使需求精确化、完全化，并据此改进、完善原型，如此迭代，直到软件开发人员和用户都确认软件系统的需求并达成一致的理解为止。软件需求确定后，便可进行设计、编码、测试等以后的各个开发步骤，如图 1-2 所示。

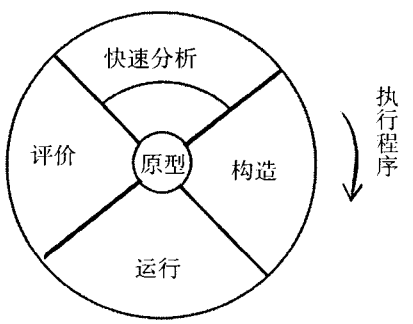

图 1-2　原型模型

快速原型的开发途径有三种：

(1)仅模拟软件系统的人机界面和人机交互方式。

(2)开发一个工作模型，实现软件系统中重要的或容易产生误解的功能。

(3)利用一个或几个类似的正在运行的软件向用户展示软件需求中的部分或全部功能。

总之,建造原型应尽量采用相应的软件工具和环境,并尽量采用软件重用技术,在运行效率方面可作出让步,以便尽快提供。同时,原型应充分展示软件系统的可见部分,如人机界面、数据的输入方式和输出格式等。

原型模型比瀑布模型更符合人们认识事物的过程和规律,是一种较实用的开发框架。它适合于那些不能预先确切定义需求的软件系统的开发,更适合于那些项目组成员(包括分析员、设计员、程序员和用户)不能很好交流或通信有困难的情况。

3.螺旋模型

螺旋模型(spiral model)是 B. Boehm 于 1988 年提出的。它综合了瀑布模型和原型模型的优点,即将两者结合,并加入了风险分析机制。螺旋模型的基本框架如图 1-3 所示。螺旋模型的每一个周期都包括计划(需求定义)、风险分析、工程实现和评审 4 个阶段。

①计划(需求定义)

第一周期开始利用需求分析技术理解应用领域,获取初步用户需求,制订项目开发计划(即整个软件生命周期计划)和需求分析计划。经过一个周期后,根据用户和开发人员对上一周期工作成果评价和评审,修改、完善需求,明确下一周期软件开发的目标、约束条件,并据此制订新一轮的软件开发计划。

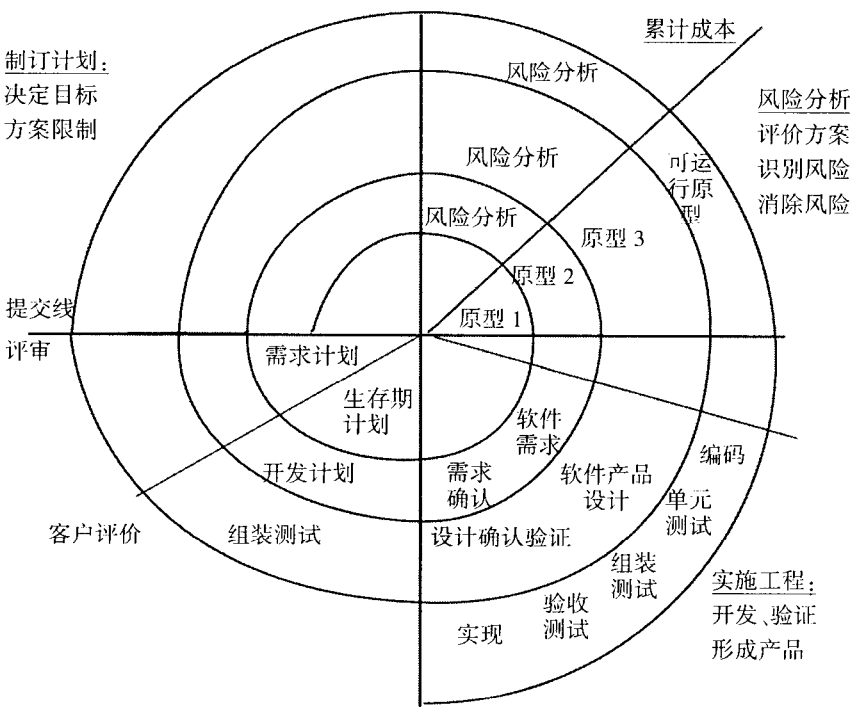

图 1-3 螺旋模型示意图