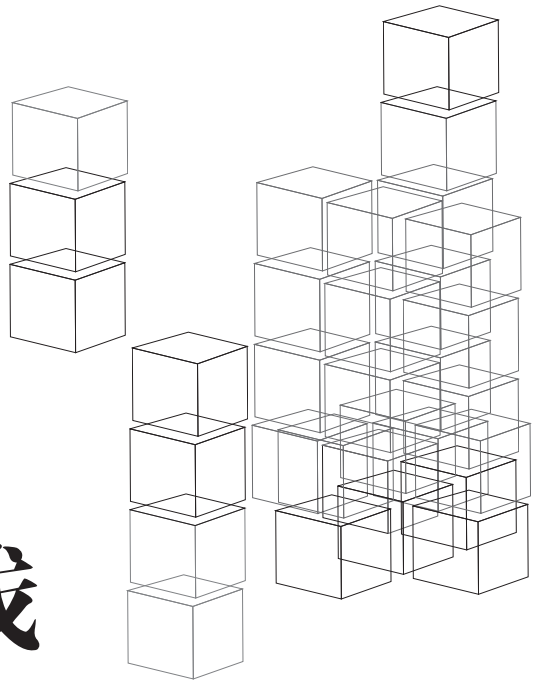
软件开发与测试丛书



软件质量管理实践

刘文红 侯育卓 郭栋 张卫祥 杨隽 沈玥 编著

清華大學出版社
北京

内 容 简 介

本书紧扣软件工程标准规范要求，结合国内软件研制现状，系统地介绍了软件质量管理的要求，涵盖了软件工程、CMMI软件能力成熟度模型和软件测试技术的相关知识。对于帮助软件质量管理人员清楚了解和掌握质量管理精髓具有较强的指导作用。本书是作者多年从事软件工程技术研究和软件质量体系建设的实践经验总结，具有良好的实用性、较强的内容指导性和较高的参考价值。本书可供从事软件研制的技术和管理人员使用，也可供高等院校的研究生及高年级本科生学习和参考。

图书在版编目(CIP)数据

软件质量管理实践/刘文红等编著. —北京：清华大学出版社，2023.6
(软件开发与测试丛书)
ISBN 978-7-302-63494-2

Ⅰ. ①软… Ⅱ. ①刘… Ⅲ. ①软件质量－质量管理 Ⅳ. ①TP311.5

中国国家版本馆CIP数据核字(2023)第085310号

责任编辑：李双双
封面设计：常雪影
责任校对：欧 洋
责任印制：曹婉颖

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦A座 **邮 编：**100084
社 总 机：010-83470000 **邮 购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市龙大印装有限公司
经 销：全国新华书店
开 本：185mm×260mm **印 张：**13 **字 数：**316千字
版 次：2023年8月第1版 **印 次：**2023年8月第1次印刷
定 价：69.00元

产品编号：091894-01

“软件开发与测试丛书”
编审委员会

“软件开发与测试”丛书序

为应对“软件危机”的挑战，人们在20世纪60年代末提出借鉴传统行业在质量管理方面的经验，用工程化的思想来管理软件，以提高复杂软件系统的质量和开发效率，即软件工程化。40多年以来，软件已广泛应用到各个工程领域乃至生活的各个方面，极大地提高了社会信息化水平，软件工程也早已深入人心。

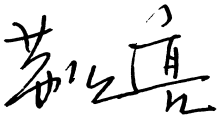
质量是产品的生命，对软件尤其如此。软件的直观性远不及硬件，软件的质量管理相对困难得多；但与传统行业类似，大型复杂软件的质量在很大程度上取决于软件过程质量。质量评估是质量管理的关键，没有科学的评估标准和方法，就无从有效地管理质量，软件评测是质量评估的最有效和最重要的手段之一。

北京跟踪与通信技术研究所软件评测中心是从事软件评测与工程化管理的专业机构，是在我国大力发展航天事业的背景下，为保障载人航天工程软件质量，经原国防科工委批准，国内最早成立的第三方软件评测与工程化管理的技术实体组织之一。自成立以来，软件评测中心出色地完成了以载人航天工程、探月工程为代表的数百项重大工程关键软件评测项目，自主研发了测试仿真软件系统、测试辅助设计工具、评测项目与过程管理软件等一系列软件测试工具，为主制订了GB/T 15532—2008《计算机软件测试规范》、GB/T 9386—2008《计算机软件测试文档编制规范》、GJB 141《军用软件测试指南》等软件测试标准，深入研究了软件测试自动化、缺陷分析与预测、可信性分析与评估、测试用例复用等软件测试技术，在嵌入式软件、非嵌入式软件和可编程逻辑器件软件等不同类型软件测试领域，积累了丰富的测试经验和强大的技术实力。

为进一步促进技术积累和对外交流，北京跟踪与通信技术研究所组织编写了本套丛书。本丛书是软件评测中心多年来技术经验的结晶，致力于以资深软件从业者和工程一线技术人员的视角，融会贯通软件工程特别是软件测试、质量评估与过程管理等领域相关的知识、技术和方法。本丛书的特色是重点突出、实用性强，每本书针对不同方向，着重介绍实践中常用的、好用的技术内容，并配以相应的范例、模板、算法或工具，具有很高的参考价值。

本丛书将为具有一定知识基础和工作经验、想要实现快速进阶的从业者提供一套内容丰富的实践指南。对于要对工作经验较少的初入职人员进行技术培训、快速提高其动手能力的单位或机构，本丛书也是一套难得的参考资料。

丛书编审委员会



2015年5月6日

前　言

随着信息技术的迅速发展，计算机软件的应用日益广泛，软件失效导致的后果也愈加严重，特别是在航空航天、金融保险、交通通信、工业控制等关系国计民生的重要领域，软件一旦失效将造成重大损失，因此对软件质量提出更高的要求。软件质量受到人们越来越多的关注。

本书紧扣软件工程标准规范要求，结合国内软件研制现状，系统地介绍了软件质量管理的要求，涵盖软件工程、CMMI 软件能力成熟度模型和软件测试技术的相关知识。对于软件质量管理人员清楚了解和掌握质量管理精髓具有较强的指导作用。

全书结构如下：第 1 章概述了软件工程和软件生命周期各模型，介绍了软件过程在软件质量管理中的意义；后续各章按照先工程后管理的顺序，第 2～5 章分别介绍了软件需求管理、同行评审、验证与确认的要求；第 6～11 章分别介绍软件缺陷管理、软件配置管理、软件质量保证、软件质量度量、测量与分析及软件质量持续改进。

本书是编写组多年从事软件质量管理工作的技术积累，兼具实用性与前瞻性，系统地介绍了软件质量管理与软件工程化各方面的内容。与软件一样，本书虽然经过了认真的编写和修改，仍然会有一些不足或疏漏存在，而这些不足或疏漏只有在使用时才会被发现。如果您在阅读本书后，愿意将不足或疏漏、意见和建议反馈给我们，我们将非常感激。

编著者

2022 年 4 月

目 录

CHAPTER 1 第1章

软件质量管理概述

1.1 软件工程概述

人类社会已经跨入了21世纪，计算机系统已经渗入人类生活的各个领域，同时计算机软件工程已经发展成为当今世界重要的技术领域之一。对软件本身进行研究使一门重要的学科——软件工程诞生。软件工程的研究领域包括软件的开发方法、软件的生命周期及软件的工程实践等。

1.1.1 软件危机与软件工程的起源

1.1.1.1 计算机系统的发展历程

20世纪60年代中期以前，是计算机系统发展的早期。在这个时期，通用硬件已经相当普遍，软件是为每个具体应用而专门编写的，大多数人认为软件开发是无须预先计划的事情。这时的软件实际上就是规模较小的程序，程序的编写者和使用者往往是同一个(或同一组)人。由于规模小，程序编写起来相当容易，也没有系统化的方法，软件开发工作更没有得到任何管理。这种个体化的软件环境，使软件设计往往只是在人们头脑中隐含进行的一个模糊过程，除了程序清单之外，根本没有其他文档资料保存下来。

20世纪60年代中期到70年代中期，是计算机系统发展的第一代。计算机技术在这10年中有了很大进步。多道程序多用户系统引入了人机交互的新概念，开创了计算机应用的新境界，使硬件和软件的配合上升到一个新的层次。实时系统能够从多个信息源收集、分析和转换数据，从而使进程控制能以毫秒而不是分钟来进行。在线存储技术的进步引领了第一代数据库管理系统的出现。

计算机系统发展的第二代的一个重要特征是出现了广泛使用产品软件的“软件作坊”。但是，“软件作坊”基本上仍然沿用早期形成的个体化软件开发方法。随着计算机应用的日益普及，软件数量急剧膨胀，在程序运行时发现的错误必须设法改正；用户有了新的需求时必须相应地修改程序；硬件或操作系统更新时，通常需要修改程序以适应新的环境。上述各种软件维护工作，以令人吃惊的比例不断地耗费资源。更严重的是，许多程序的个体化特性使它们最终变得不可维护。“软件危机”就这样开始出现了。1968年，北大西洋公约组织的计算机领域的科学家们在联邦德国召开国际会议，讨论软件危机问题。这次会议正式提出并使用了“软件工程”这个名词，一门新兴的工程学科就此诞生。

1.1.1.2 软件危机简介

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，以满足对软件日益增长的需求；如何维护数量不断膨胀的已有软件。鉴于软件危机的长期性和症状不明显的特征，近年来有人建议把软件危机更名为“软件萧条”(depression)或“软件困扰”(affliction)。不过“软件危机”这个词强调了问题的严重性，而且也已为绝大多数软件工作者所熟悉，所以本书仍将沿用它。

具体来说，软件危机主要有以下一些典型表现。

(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有可能高出一个数量级，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。而为了追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定量概念刚刚出现不久，软件质量保证技术还没有被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。“可重用的软件”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”(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(milestone)，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，这些文档资料在软件开发过程中，对于软件维护人员准确地交流信息更是至关重要、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。由于微电子学技术的进步和生产自动化程度不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀及软件规模和数量的不断扩大而持续上升。美国在 1985 年付出的软件成本大约已占计算机系统总成本的 90%。

(7) 软件开发生产率提升的速度，既跟不上硬件的发展速度，也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现代计算机硬件提供的巨大潜力。

以上列举的仅仅是软件危机的一些明显的表现，与软件开发和维护有关的问题远远不止这些。

1.1.1.3　产生软件危机的原因

在软件开发和维护的过程中存在这么多严重问题，一方面与软件本身的特点有关，另一方面也和软件开发与维护的方法不正确有关。

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。由于软件缺乏“可见性”，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，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错误，因此，软件维护通常意味着改正或修改原来的设计，这就在客观上使软件较难维护。

软件不同于一般程序，它的一个显著特点是规模庞大，且程序复杂性将随着程序规模的增加而呈指数上升。为了在预定时间内开发出规模庞大的软件，必须由许多人分工合作。然而，如何保证每个人完成的工作合在一起确实能构成一个高质量的大型软件系统，更是一个极端复杂、困难的问题，不仅涉及许多技术问题，诸如分析方法、设计方法、形式说明方法、版本控制等，更重要的是必须有严格而科学的管理体系。

软件本身独有的特点确实给开发和维护带来了一些客观困难，但是人们在开发和使用计算机系统的长期实践中，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，目前相当多的软件专业人员对软件开发和维护还存在不少糊涂观念，在实践过程中或多或少地采用了错误的方法和技术，这可能是使软件问题发展成软件危机的主要原因。与软件开发和维护有关的许多错误认识和做法的形成，可以归于在计算机系统发展的早期阶段软件开发的个体化特点。错误的认识和做法主要表现为忽视软件需求分析的重要性，认为软件开发就是编写程序并设法使之运行，轻视软件维护等。

事实上，对用户要求没有完整、准确的认识就匆忙着手编写程序是许多软件开发工程失败的主要原因之一。只有用户才真正了解他们自己的需要，但是许多用户在开始时并不能准确、具体地叙述他们的需要，软件开发人员需要做大量深入、细致的调查研究工作，反复多次地和用户交流信息，才能真正全面、准确、具体地了解用户的要求。对问题和目标的正确认识是解决任何问题的前提和出发点，软件开发同样也不例外。急于求成，仓促上阵，对用户要求没有正确认识就匆忙着手编写程序，这就如同不打好地基就盖高楼，最终必然垮台。事实上，越早开始编写程序，完成它所需要用的时间往往越长。

一个软件从定义、开发、使用和维护，直到最终被废弃，要经历一个漫长的时期，这就如同一个人要经过胎儿、儿童、青年、中年和老年，直到最终死亡的漫长时期。软件经历的这个漫长的时期通常被称为生命周期。软件开发最初的工作应是问题定义，也就是确定要求解决的问题是什么；然后要进行可行性研究，决定该问题是否存在一个可行的解决办法；接下来应该进行需求分析，也就是深入、具体地了解用户的要求，在所要开发的系统（不妨称之为目标系统）必须做什么这个问题上和用户取得完全一致的看法。经过上述软件定义时期的准备工作才能进入开发时期，而在开发时期首先需要对软件进行设计（通常又分为概要设计

和详细设计两个阶段),然后才能进入编写程序的阶段,程序编写完之后还必须经过大量的测试工作(需要的工作量通常占软件开发全部工作量的40%～50%)才能最终交付使用。所以,编写程序只是软件开发过程中的一个阶段,而且在典型的软件开发工程中,编写程序所需的工作量只占软件开发全部工作量的10%～20%。

另外,还必须认识到程序只是完整的软件产品的一个组成部分,在上述软件生命周期的每个阶段都要得出最终产品的一个或几个组成部分(这些组成部分通常以文档资料的形式存在)。也就是说,一个软件产品必须由一个完整的配置组成,软件配置主要包括程序、文档、数据等成分。必须清除只重视程序而忽视软件配置其余成分的糊涂观念。

做好软件定义时期的工作,是降低软件成本提高软件质量的关键。如果软件开发人员在定义时期没有正确、全面地理解用户需求,直到测试阶段或软件交付使用后才发现"已完成的"软件不完全符合用户的需要,这时再修改就为时晚矣。

在软件开发的不同阶段进行修改需要付出的代价差别很大。在早期引入变动,涉及的面较少,因而代价也比较低;在开发的中期,软件配置的许多成分已经完成,引入一个变动要对所有完成的配置成分都做相应的修改,不仅工作量大,而且逻辑上也更复杂,因此付出的代价剧增;在软件"已经完成"时再引入变动,当然需要付出更高的代价。根据美国一些软件公司的统计资料,在后期引入一个变动比在早期引入相同变动所需付出的代价高2～3个数量级。图1-1所示为在不同时期引入同一个变动需要付出的代价随时间变化的趋势。

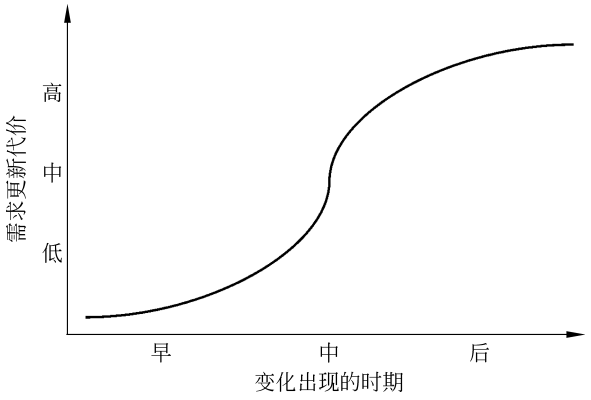


图1-1 在不同时期引入同一个变动付出的代价随时间变化的趋势

通过上面的论述不难认识到,轻视维护是一个最大的错误。许多软件产品的使用寿命长达10年甚至20年,在这样漫长的时期中不仅必须改正使用过程中发现的每一个潜伏的错误,而且当环境变化时(如硬件或系统软件更新换代)还必须相应地修改软件以适应新的环境,特别是必须经常改进或扩充原来的软件以满足用户不断变化的需要。所有这些改动都属于维护工作,而且是在软件已经完成之后进行的,因此,维护是极端艰巨复杂的工作,需要花费很大代价。统计数据表明,实际上用于软件维护的费用占软件总费用的55%～70%。软件工程学的一个重要目标就是提高软件的可维护性,减少软件维护的代价。

了解产生软件危机的原因,纠正错误认识,建立起关于软件开发和维护的正确概念,仅仅是解决软件危机的开始,全面解决软件危机需要采取一系列综合措施。

1.1.1.4　消除软件危机的途径

为了消除软件危机，首先应该对计算机软件有一个正确的认识。正如 1.1.1.3 节中讲过的，应该彻底清除在计算机系统早期发展阶段形成的“软件就是程序”的错误观念。一个软件必须由一个完整的配置组成。事实上，软件是程序数据及相关文档的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列；数据是使程序能够适当地处理信息的数据结构；文档是开发、使用和维护程序所需要的图文资料。1983 年电气和电子工程师协会(Institute of Electrical and Electronics Engineers，IEEE)为软件下的定义是：计算机程序、方法、规则、相关的文档资料及在计算机上运行程序时所必需的数据。虽然表面上在这个定义中列出了软件的 5 个配置成分，但是，方法和规则通常是在文档中说明并在程序中实现的。

更重要的是，必须充分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。必须充分吸取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硬件研究和开发的经验教训。

应该推广和使用在实践中总结出来的开发软件成功的技术和方法，并且研究探索更好、更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误概念和做法。

应该开发和使用更好的软件工具。正如机械工具可以“放大”人类的体力一样，软件工具可以“放大”人类的智力。在软件开发的每个阶段都有许多烦琐重复的工作需要做，在适当的软件工具辅助下，开发人员可以把这类工作做得既快又好。把各个阶段使用的软件工具有机地集合成一个整体，支持软件开发的全过程，称为软件工程支撑环境。

总之，为了消除软件危机，既要有技术措施、方法和工具，又要有必要的组织管理措施。软件工程正是从管理和技术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。

1.1.2　软件工程

1.1.2.1　什么是软件工程

概括地说，软件工程是指导计算机软件开发和维护的工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，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，经济地开发出高质量的软件并有效地维护它，这就是软件工程。

下面给出软件工程的几个定义。

1983 年，IEEE 给软件工程下的定义是：软件工程是开发、运行维护和修复软件的系统方法。这个定义相当概括，它主要强调软件工程是系统方法而不是某种神秘的个人技巧。

Fairly 认为：“软件工程学是为了在成本限额以内按时完成开发和修改软件产品所需要的系统生产和维护技术及管理学科。”这个定义明确指出了软件工程的目标是在成本限额内按时完成开发和修改软件的工作，同时也指出了软件工程包含技术和管理两方面的内容。

Fritz Bauer 给出了下述定义：软件工程是为了经济地获得可靠的且能在实际机器上有效地运行的软件，而建立和使用的完善的工程化原则。这个定义不仅指出软件工程的目标是经济地开发出高质量的软件，而且强调了软件工程是一门工程学科，它应该建立并使用完善的工程化原则。

1993 年，IEEE 进一步给出了一个更全面的定义。

软件工程是：①把系统化的、规范的、可度量的途径应用于软件开发、运行和维护的过程，也就是把工程化应用于软件中；②研究①中提到的途径。

认真研究上述这些关于软件工程的定义，有助于我们建立起对软件工程这门工程学科的全面的整体性认识。

1.1.2.2 软件工程的基本原理

自从 1968 年在联邦德国召开的国际会议上正式提出并使用了“软件工程”这个术语以来，研究软件工程的专家学者们陆续提出了 100 多条关于软件工程的准则或信条。著名的软件工程专家 Barry W.Boehm 综合这些学者们的意见并总结了 TRW 公司多年来开发软件的经验，于 1983 年在一篇论文中提出了软件工程的 7 条基本原理。他认为这 7 条原理是确保软件产品质量和开发效率的最小集合。这 7 条原理是互相独立的，其中任意 6 条原理的组合都不能代替另一条原理，因此，它们是缺一不可的最小集合。然而这 7 条原理又是相当完备的，人们虽然不能用数学方法严格证明它们是一个完备的集合，但是可以证明在此之前已经提出的 100 多条软件工程原理都可以由这 7 条原理的任意组合蕴含或派生。

下面简要介绍软件工程的 7 条基本原理。

1）用分阶段的生命周期计划严格管理

有人经统计发现，在不成功的软件项目中有一半左右是由于计划不周密造成的，可见把建立完善的计划作为第 1 条基本原理是吸取了前人的教训而提出来的。在软件开发与维护的漫长生命周期中，需要完成许多性质各异的工作。这条基本原理意味着，应该把软件生命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的开发与维护工作进行管理。Boehm 认为，在软件的整个生命周期中应该制订并严格执行 6 类计划，它们是项目概要计划、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。

不同层次的管理人员都必须严格按照计划各尽其职地管理软件开发与维护工作，绝不能受客户或上级人员的影响而擅自背离预定计划。

2）坚持进行阶段评审

当时人们已经认识到，软件的质量保证工作不能等到编码阶段结束之后再进行。这样说至少有两个理由：第一，大部分错误是在编码之前造成的，如根据 Boehm 等人的统计，设计错误占软件错误的 63%，编码错误仅占 37%；第二，错误发现与改正得越晚，所需付出的代价也越高，参见图 1-1。因此，在每个阶段都进行严格的评审，以便尽早发现在软件开发过程中所犯的错误，是一条必须遵循的重要原则。

3）实行严格的产品控制

在软件开发过程中不应随意改变需求，因为改变一项需求往往需要付出较高的代价。但是，在软件开发过程中改变需求又是难免的。由于外部环境的变化，相应地改变用户需求

是一种客观需要，显然不能硬性禁止客户提出改变需求的要求，而只能依靠科学的产品控制技术来顺应这种要求。也就是说，当改变需求时，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。所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分（各个阶段产生的文档或程序代码）。基准配置管理也称为变动控制：一切有关修改软件的建议，特别是涉及对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。绝对不能是谁想修改软件（包括尚在开发过程中的软件），就随意进行修改。

4）采用现代程序设计技术

从提出软件工程的概念开始，人们一直把主要精力用于研究各种新的程序设计技术。20 世纪 60 年代末提出的结构程序设计技术，已经成为公认的先进的程序设计技术。在这之后又进一步发展出各种结构分析（structured analysis，SA）与结构设计（structured design，SD）技术。近年来，面向对象技术已经在许多领域中迅速地取代了传统的结构化开发方法。实践表明，采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件产品的质量。

5）结果应能清楚地审查

软件产品不同于一般的物理产品，它是看不见、摸不着的逻辑产品。软件开发人员（或开发小组）的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比一般产品的开发过程更难于评价和管理。为了提高软件开发过程的可见性，更好地对软件进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使所得到的结果能够被清楚地审查。

6）开发小组的人员应该少而精

这条基本原理的含义是，软件开发小组的组成人员的素质应该好，而人数则不宜过多。开发小组人员的素质和数量，是影响软件产品质量和开发效率的重要因素。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。此外，随着开发小组人员数目的增加，因为交流情况讨论问题而造成的通信开销也急剧增加。当开发小组人员数为 N 时，可能的通信路径有 $N(N-1)/2$ 条，可见随着人数 N 的增大，通信开销将急剧增加。因此，组成少而精的开发小组是软件工程的一条基本原理。

7）承认不断改进软件工程实践的必要性

遵循前 6 条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产。但是，仅有前 6 条原理并不能保证软件开发与维护的过程能赶上时代前进的步伐、跟上技术的不断进步，因此，Boehm 提出应把承认不断改进软件工程实践的必要性作为软件工程的第 7 条基本原理。按照这条原理，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验，如收集进度和资源耗费数据，收集出错类型和问题报告数据等。这些数据不仅可以用来评价新的软件技术的效果，而且可以用来指明必须着重开发的软件工具和应该优先研究的技术。

1.1.2.3　软件工程包含的领域

IEEE 在 2014 年发布的《软件工程知识体系指南》中将软件工程知识体系划分为以下

15 个知识领域。

(1) 软件需求(software requirements)。软件需求涉及软件需求的获取、分析、规格说明和确认。

(2) 软件设计(software design)。软件设计定义了一个系统或组件的体系结构、组件、接口和其他特征的过程及这个过程的结果。

(3) 软件构建(software construction)。软件构建指通过编码、验证、单元测试、集成测试和调试的组合,详细地创建可工作的和有意义的软件。

(4) 软件测试(software testing)。软件测试是为评价、改进产品的质量、标识产品的缺陷和问题而进行的活动。

(5) 软件维护(software maintenance)。软件维护指由于一个问题或改进的需要而修改代码和相关文档,进而修正现有的软件产品并保留其完整性的过程。

(6) 软件配置管理(software configuration management)。软件配置管理指支持性的软件生命周期过程,它是为了系统地控制配置变更,在软件系统的整个生命周期中维持配置的完整性和可追踪性,而标识系统在不同时间点上的配置的学科。

(7) 软件工程管理(software engineering management)。软件工程的管理活动建立在组织和内部基础结构管理、项目管理、度量程序的计划制订和控制三个层次上。

(8) 软件工程过程(software engineering process)。软件工程过程涉及软件生命周期过程本身的定义、实现、评估、管理、变更和改进。

(9) 软件工程模型和方法(software engineering models and methods)。软件工程模型特指在软件的生产与使用、退役等各个过程中的参考模型的总称,诸如需求开发模型、架构设计模型等都属于软件工程模型的范畴;软件开发方法,主要讨论软件开发各种方法及其工作模型。

(10) 软件质量(software quality)。软件质量特征涉及多个方面,保证软件产品的质量是软件工程的重要目标。

(11) 软件工程职业实践(software engineering professional practice)。软件工程职业实践涉及软件工程师应履行其实践承诺,使软件的需求分析、规格说明设计、开发、测试和维护成为一项有益和受人尊敬的职业;还包括团队精神和沟通技巧等内容。

(12) 软件工程经济学(software engineering economics)。软件工程经济学是研究为实现特定功能需求的软件工程项目而提出的在技术方案、生产(开发)过程、产品或服务等方面所做的经济服务与论证、计算与比较的一门系统方法论学科。

(13) 计算基础(computing foundations)。计算基础涉及解决问题的技巧抽象、编程基础、编程语言的基础知识、调试工具和技术、数据结构和表示、算法和复杂度、系统的基本概念、计算机的组织结构、编译基础知识、操作系统基础知识、数据库基础知识和数据管理、网络通信基础知识、并行和分布式计算基本的用户人为因素、基本的开发人员人为因素和安全的软件开发和维护等方面的内容。

(14) 数学基础(mathematical foundations)。数学基础涉及集合、关系和函数,基本的逻辑、证明技巧、计算的基础知识、图和树、离散概率、有限状态机、语法、数值精度、准确性和错误、数论和代数结构等方面的内容。

(15) 工程基础(engineering foundations)。工程基础涉及实验方法和实验技术、统计分

析、度量工程设计，建模、模拟和建立原型，标准和影响因素分析等方面的内容。

软件工程知识体系的提出，让软件工程的内容更加清晰，也使其作为一个学科的定义和界限更加分明。

1.2　软件过程

软件工程过程是为了获得高质量软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

在完成开发任务时必须进行一些开发活动，并且使用适当的资源（人员、时间、计算机硬件、软件工具等），在过程结束时将输入（如软件需求）转化为输出（如软件产品），因此，ISO 9000 定义过程为“把输入转化为输出的一组彼此相关的资源和活动”。过程定义了运用方法的顺序、应该交付的文档资料、为保证软件质量和协调变化所需要采取的管理措施，以及标志软件开发各个阶段任务完成的里程碑。为获得高质量的软件产品，软件工程过程必须科学、合理。

本节讲述在软件生命周期全过程中应该完成的基本任务，并介绍各种常用的过程模型。

1.2.1　软件生命周期的基本任务

概括地说，软件生命周期由软件定义、软件开发和运行维护 3 个时期组成，每个时期又可进一步划分成若干个阶段。

软件定义时期的任务是确定软件开发工程必须完成的总目标；确定工程的可行性；导出实现工程目标应该采用的策略及系统必须完成的功能；估计完成该项工程需要的资源和成本，并且制订工程进度表。这个时期的工作通常又称为系统分析，由系统分析员负责完成。软件定义时期通常进一步划分为 3 个阶段，即问题定义、可行性研究和需求分析。

软件开发时期具体设计和实现在前一个时期定义的软件，它通常由下述 4 个阶段组成：概要设计、详细设计、编码和单元测试、综合测试。其中前两个阶段又称为系统设计，后两个阶段又称为系统实现。

运行维护时期的主要任务是使软件持久地满足用户的需要。具体地说，当软件在使用过程中发现错误时应该加以改正；当环境改变时应该修改软件以适应新的环境；当用户有新要求时应该及时改进软件以满足用户的新需要。通常对维护时期不再进一步划分阶段，但是每一次维护活动本质上都是一次压缩和简化了的定义和开发过程。

下面简要介绍上述各个阶段应该完成的基本任务。

1）问题定义

问题定义阶段必须回答的关键问题是：“要解决的问题是什么。”如果不知道问题是什么就试图解决这个问题，显然是盲目的，只会白白浪费时间和金钱，最终得出的结果很可能是毫无意义的。尽管确切地定义问题的必要性是十分明显的，但是在实践中它却可能是最容易被忽视的一个步骤。

通过调研，系统分析员应该提出关于问题性质、工程目标和工程规模的书面报告，并且需要得到客户对这份报告的确认。

2）可行性研究

这个阶段要回答的关键问题是："上一个阶段所确定的问题是否有行得通的解决办法。"并非所有问题都有切实可行的解决办法，事实上，许多问题不可能在预定的系统规模或时间期限之内解决。如果问题没有可行的解，那么花费在这项工程上的任何时间、资源和经费都是无谓的浪费。

可行性研究的目的就是用最小的代价在尽可能短的时间内确定问题是否能够解决。必须记住，可行性研究的目的不是解决问题，而是确定问题是否值得去解决。要达到这个目的，不能靠主观猜想而只能靠客观分析。系统分析员必须进一步概括地了解用户的需求，并在此基础上提出若干种可能的系统实现方案，对每种方案都从技术、经济、社会因素（如法律）等方面分析其可行性，从而最终确定这项工程的可行性。

3）需求分析

这个阶段的任务仍然不是具体地解决客户的问题，而是准确地回答"目标系统必须做什么"这个问题。

虽然在可行性研究阶段已经粗略了解了用户的需求，甚至还提出了一些可行的方案，但是，可行性研究的基本目的是用较小的成本在较短的时间内确定是否存在可行的解法，因此许多细节被忽略了。然而在最终的系统中却不能遗漏任何一个微小的细节，所以可行性研究并不能代替需求分析，它实际上并没有准确地回答"系统必须做什么"这个问题。

需求分析的任务不是确定系统怎样完成它的工作，而仅仅是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰和具体的要求。

用户了解他们所面对的问题，知道必须做什么，但是通常不能完整准确地表达出他们的要求，更不知道怎样利用计算机解决他们的问题；软件开发人员知道怎样用软件实现人们的要求，但是对特定用户的具体要求并不完全清楚。因此，系统分析员在需求分析阶段必须与用户密切配合，充分交流信息，以得出经过用户确认的系统需求。

这个阶段的另外一项重要任务，是用正式文档准确地记录对目标系统的需求，该文档通常称为规格说明（specification）。

4）概要设计

这个阶段的基本任务是：概括地回答"怎样实现目标系统"。概要设计又称为初步设计、逻辑设计、高层设计或总体设计。

首先，应该设计出实现目标系统的几种可能的方案。软件工程师应该用适当的表达工具描述每种可能的方案，分析每种方案的优缺点，并在充分权衡各种方利弊的基础上，推荐一个最佳方案。此外，还应该制订出实现所推荐方案的详细计划。如果客户接受所推荐的系统方案，则应该进一步完成本阶段的另一项主要任务。

上述设计工作确定了解决问题的策略及目标系统中应包含的程序。但是，对于怎样设计这些程序，软件设计的一条基本原理指出，程序应该模块化，也就是说，一个程序应该由若干个规模适中的模块按合理的层次结构组织而成。因此，概要设计的另一项主要任务就是设计程序的体系结构，也就是确定程序由哪些模块组成及模块间的关系。

5）详细设计

概要设计阶段以比较抽象的、概括的方式提出了解决问题的办法。详细设计阶段的任务就是把解法具体化，也就是回答"应该怎样具体地实现这个系统"这个关键问题。

这个阶段的任务还不是编写程序，而是设计出程序的详细规格说明。这种规格说明的作用类似于其他工程领域中工程师经常使用的工程蓝图，它们应该包含必要的细节，程序员可以根据它们写出实际的程序代码。

详细设计也称为模块设计、物理设计或低层设计。这个阶段将详细地设计每个模块，确定实现模块功能所需要的算法和数据结构。

6）编码和单元测试

这个阶段的关键任务是写出正确的，容易理解、容易维护的程序模块。

程序员应该根据目标系统的性质和实际环境，选取一种适当的高级程序设计语言（必要时用汇编语言），把详细设计的结果翻译成用选定的语言书写的程序，并且仔细测试编写出的每一个模块。

7）综合测试

这个阶段的关键任务是通过各种类型的测试（及相应的调试）使软件达到预定的要求。最基本的测试是集成测试和验收测试。所谓集成测试是根据设计的软件结构，把经过单元测试检验的模块按某种选定的策略装配起来，在装配过程中对程序进行必要的测试。所谓验收测试则是按照规格说明书的规定（通常在需求分析阶段确定），由用户（或在用户积极参加下）对目标系统进行验收。必要时还可以再通过现场测试或平行运行等方法对目标系统做进一步测试检验。为了使用户能够积极参加验收测试，并且在系统投入生产性运行以后能够正确、有效地使用这个系统，通常需要以正式的或非正式的方式对用户进行培训。通过对软件测试结果的分析可以预测软件的可靠性；反之，根据对软件可靠性的要求，也可以决定测试和调试过程可以结束的时机。应该用正式的文档资料把测试计划、详细测试方案及实际测试结果保存下来，作为软件配置的一个组成部分。

8）软件维护

维护阶段的关键任务是，通过各种必要的维护活动使系统持久地满足用户的需要。通常有 4 类维护活动：改正性维护，也就是诊断和改正在使用过程中发现的软件错误；适应性维护，即修改软件以适应环境的变化；完善性维护，即根据用户的要求改进或扩充软件使其更完善；预防性维护，即修改软件为将来的维护活动预先做准备。

虽然没有把维护阶段进一步划分成更小的阶段，但是实际上每一项维护活动都应该经过提出维护要求（或报告问题），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，复查验收等一系列步骤，因此，实质上是经历了一次压缩和简化了的软件定义和开发的全过程。

每项维护活动都应该被准确地记录下来，作为正式的文档资料加以保存。我国国家标准《计算机软件开发规范》（GB 8566—1988）也把软件生命周期划分成 8 个阶段，这些阶段是：可行性研究与计划，需求分析，概要设计，详细设计，实现，组装测试，确认测试，使用和维护。其中，实现阶段即是编码与单元测试阶段，组装测试即是集成测试，确认测试即是验收测试。可见，国家标准中划分阶段的方法与前面介绍的阶段划分方法基本相同，差别仅仅是：因为问题定义的工作量很小而没有把它作为一个独立的阶段列出来；由于综合测试的工作量过大而把它分解成了两个阶段。

在实际从事软件开发工作时，软件规模种类、开发环境及开发时使用的技术方法等因素，都影响阶段的划分。事实上，承担的软件项目不同，应该完成的任务也会有差异，没有一

个适用于所有软件项目的任务集合。适用于大型复杂项目的任务集合，对于小型且较简单的项目而言往往过于复杂。因此，一个科学、有效的软件工程过程应该定义一组适合所承担的项目特点的任务集合。一个任务集合通常包括一组软件工程工作任务、里程碑和应该交付的产品(软件配置成分)。

生命周期模型规定了把生命周期划分成哪些阶段及各个阶段的执行顺序，因此，也称为过程模型。

实际从事软件开发工作时应该根据所承担的项目的特点来划分阶段，但是，下述软件过程模型并不针对某个特定项目，因此，只能使用“通用的”阶段划分方法。由于瀑布模型与快速原型模型的主要区别是获取用户需求的方法不同，因此，下面在介绍生命周期模型时把“规格说明”作为一个阶段独立出来。此外，问题定义和可行性研究的主要任务是概括地了解用户的需求，为了简洁地描述软件过程，把它们都归并到需求分析中。同样，为了简单起见，把概要设计和详细设计合并在一起称为“设计”。

1.2.2 瀑布模型

1970 年，人们整理了第一个软件生命周期，即瀑布型生命周期(waterfall model)。瀑布型生命周期包括可行性分析与开发项计划、需求分析、设计(概要设计和详细设计)、编码、测试、维护等阶段。瀑布模型一直是被广泛采用的生命周期模型，现在它仍然是软件工程中应用最广泛的过程模型。图 1-2 所示为传统的瀑布模型。

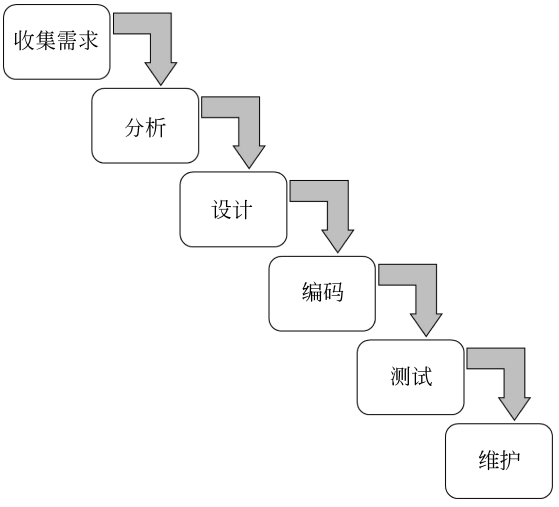


图 1-2 传统的瀑布模型

1.2.2.1 瀑布型生命周期

典型的瀑布型生命周期，有如下 6 个阶段。

1) 问题的定义及规划

此阶段由软件开发方与需求方共同讨论，主要确定软件的开发目标及其可行性。

2）需求分析

在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析阶段是一个很重要的阶段，这一阶段做得好，将为整个软件开发项目的成功打下良好的基础。“唯一不变的是变化本身”。同样地，需求也是在整个软件开发过程中不断变化和深入的，因此我们必须制订需求变更计划来应付这种变化，以保护整个项目的顺利进行。

3）软件设计

此阶段主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计，数据库设计等。软件设计一般分为总体设计和详细设计。好的软件设计将为软件程序编写打下良好的基础。

4）程序编码

此阶段是将软件设计的结果转换成计算机可运行的程序代码。在程序编码中必须要制定统一、符合标准的编写规范，以保证程序的可读性和易维护性，提高程序的运行效率。

5）软件测试

在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。整个测试过程分单元测试、组装测试及系统测试三个阶段进行。测试的方法主要有白盒测试和黑盒测试两种。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性。

6）运行维护

软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件不能继续适应用户的要求。要延续软件的使用寿命，就必须对软件进行维护。软件的维护包括纠错性维护和改进性维护两个方面。

1.2.2.2　传统的瀑布模型开发软件的方式的特点

1）阶段间具有顺序性和依赖性

这个特点有两重含义：①必须等前阶段的工作完成之后，才能开始后一阶段的工作；②前一阶段的输出文档就是后一阶段的输入文档。因此，只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果。但是，万一在生命周期某阶段发现了问题，很可能需要追溯到在它之前的一些阶段，必要时还要修改前面已经完成的文档。然而，在生命周期后期改正早期阶段造成的问题，需要付出很高的代价。这就好像水已经从瀑布顶部流泻到底部，再想使它返回到高处需要付出很大的能量一样。

2）推迟实现的观点

缺乏软件工程实践经验的软件开发人员，接到软件开发任务以后常常急于求成，总想尽早开始编写程序。但实践表明，对于规模较大的软件项目来说，编码开始得越早，最终完成开发工作所需要的时间反而越长。这是因为，前面阶段的工作没做或做得不扎实，过早地考虑进行程序实现，往往导致大量返工，有时甚至发生无法弥补的问题，带来灾难性后果。

瀑布模型在编码之前设置了系统分析与系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。

清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软

件的一条重要的指导思想。

3）质量保证的观点

软件工程的基本目标是优质、高产。为了保证所开发软件的质量，在瀑布模型的每个阶段都应坚持两个重要做法。

（1）每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。完整、准确的合格文档不仅是软件开发时期各类人员之间相互通信的媒介，也是运行时期对软件进行维护的重要依据。

（2）每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。事实上，越是早期阶段犯下的错误，暴露出来的时间就越晚，排除故障、改正错误所需付出的代价也越高。因此，及时审查是保证软件质量、降低软件成本的重要措施。

传统的瀑布模型过于理想化。事实上，人在工作过程中不可能不犯错误。在设计阶段可能发现规格说明文档中的错误，而设计上的缺陷或错误可能在实现过程中显现出来，在综合测试阶段将发现需求分析、设计或编码阶段的许多错误。因此，实际的瀑布模型是带“反馈环”的，如图 1-3 所示（图中实线表示开发过程，虚线箭头表示维护过程）。当在后面阶段发现前面阶段的错误时，需要沿图中左侧的反馈线返回前面的阶段，修正前面阶段的产品之后再回来继续完成后面阶段的任务。

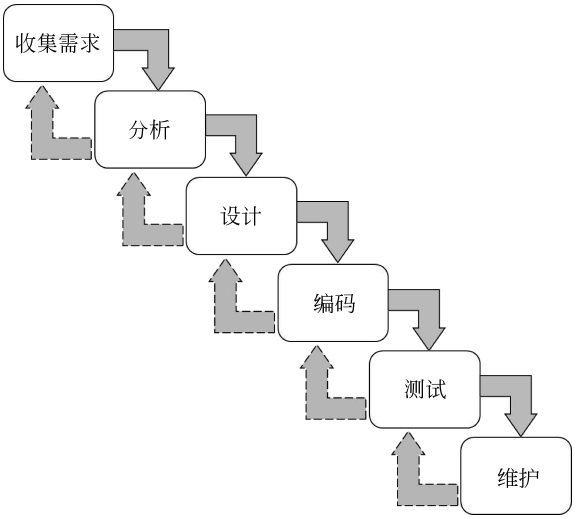


图 1-3　加入迭代过程的瀑布模型

瀑布模型有许多优点：可强迫开发人员采用规范的方法（如结构化技术）；严格地规定了每个阶段必须提交的文档；要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证。

各个阶段产生的文档是维护软件产品时必不可少的，没有文档的软件几乎是不可能维护的。遵守瀑布模型的文档约束，将使软件维护变得比较容易一些。由于绝大部分软件预算都花费在软件维护上，因此，使软件变得比较容易维护就能显著降低软件预算。可以说，瀑布模型的成功在很大程度上是由于它基本上是一种文档驱动的模型。

但是，“瀑布模型是由文档驱动的”这个事实也是它的一个主要缺点。在可运行的软件

产品交付给用户之前，用户只能通过文档来了解产品的概貌。但是，仅仅通过写在纸上的静态的规格说明，很难全面、正确地认识动态的软件产品。而且事实证明，一旦一个用户开始使用一个软件，在他的头脑中关于该软件应该做什么的想法就会或多或少地发生变化，这就使得最初提出的需求变得不完全适用。其实，要求用户不经过实践就提出完整准确的需求，在许多情况下都是不切实际的。总之，由于瀑布模型几乎完全依赖书面的规格说明，很可能导致最终开发出的软件产品不能真正满足用户的需要。

1.2.3 节将介绍快速原型模型，它的优点是有助于保证用户的真实需要得到满足。

1.2.3　快速原型模型

快速原型(rapid prototype)是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。如图 1-4 所示(图中实线箭头表示开发过程，虚线箭头表示维护过程)，快速原型模型(rapid application development，RAD)的第二步是快速建立一个能反映用户主要需求的原型系统(prototype)，让用户在计算机上试用它，通过实践来了解目标系统的概貌。通常，用户试用原型系统之后会提出许多修改意见，开发人员按照用户的意见快速地修改原型系统，然后再次请用户试用，一旦用户认为这个原型系统确实能完成他们所需要的工作，开发人员便可据此书写规格说明文档，根据这份文档开发出的软件可以满足用户的真实需求。

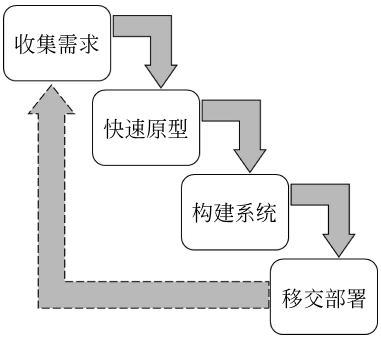


图 1-4　快速原型模型

从图 1-4 可以看出，快速原型模型是不带反馈环的，这正是这种过程模型的主要优点：软件产品的开发基本上是按线性顺序进行的。能做到基本上按线性顺序开发的主要原因如下。

(1) 原型系统已经通过与用户交互而得到验证，据此产生的规格说明文档正确地描述了用户需求，因此，在开发过程的后续阶段不会因为发现了规格说明文档的错误而进行较大的返工。

(2) 开发人员通过建立原型系统已经学到了许多东西(至少知道了“系统不应该做什么，以及怎样不去做不该做的事情”)，因此，在设计和编码阶段发生错误的可能性也比较小，这自然减少了在后续阶段需要改正前面阶段所犯错误的可能性。

软件产品一旦交付给用户使用，维护便开始了。根据用户使用过程中的反馈，可能需要返回到收集需求阶段，如图 1-4 中虚线箭头所示。

快速原型的本质是“快速”。开发人员应该尽可能快地建造出原型系统，以加速软件开发过程，节约软件开发成本。原型的用途是获知用户的真正需求，一旦需求确定了，原型将被抛弃。因此，原型系统的内部结构并不重要，重要的是，必须迅速地构建原型然后根据用户意见迅速地修改原型。UNIXShell 和超文本都是广泛使用的快速原型语言。快速原型模型伴随着第四代语言(Power Builder，Informix-4GL 等)和强有力的可视化编程工具(Visual Basic，Delphi 等)的出现而成为一种流行的开发模式。

当快速原型的某个部分是利用软件工具由计算机自动生成时，可以把这部分用到最终的软件产品中。例如，用户界面通常是快速原型的一个关键部分，当使用屏幕生成程序和报表生成程序自动生成用户界面时，实际上可以把这样得到的用户界面用在最终的软件产品中。

1.2.4 增量模型

增量模型也称为渐增模型，如图 1-5 所示。使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码、集成和测试。每个构件由多个相互作用的模块构成，并且能够完成特定的功能。使用增量模型时，第 1 个增量构件往往实现软件的基本需求，提供最核心的功能，如使用增量模型开发字处理软件时，第 1 个增量构件可能提供基本的文件管理、编辑和文档生成功能；第 2 个增量构件提供更完善的编辑和文档生成功能；第 3 个增量构件实现拼写和语法检查功能；第 4 个增量构件完成高级的页面排版功能。把软件产品分解成增量构件时，应该使构件的规模适中，规模过大或过小都不好。最佳分解方法因软件产品特点和开发人员的习惯而异。分解时必须遵守的约束条件是：当把新构件集成到现有软件中时，所形成的产品必须是可测试的。

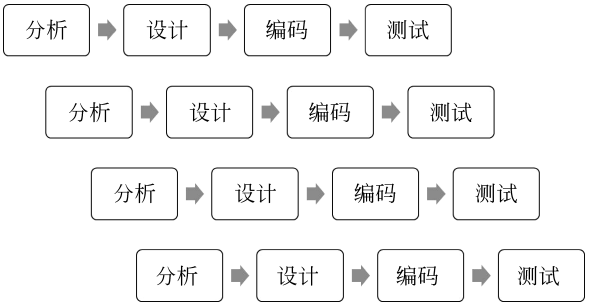


图 1-5 增量模型

采用瀑布模型或快速原型模型开发软件时，目标都是一次就把一个满足所有需求的产品提交给用户。增量模型则与之相反，它分批地逐步向用户提交产品，每次提交一个满足用户需求子集的可运行的产品。整个软件产品被分解成许多个增量构件，开发人员一个构件接一个构件地向用户提交产品。每次用户都得到一个满足部分需求的可运行的产品，直到最后一次得到满足全部需求的完整产品。从第一个构件交付之日起，用户就能做一些有用的工作。显然，能在较短时间内向用户提交可完成一些有用的工作的产品，是增量模型的一个优点。增量模型的另一个优点是，逐步增加产品功能可以使用户有较充裕的时间学习和适应新产品，从而减少一个全新的软件可能给客户组织带来的冲击。

使用增量模型的困难是，在把每个新的增量构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。此外，必须把软件的体系结构设计得便于按这种方式进行扩充，向现有产品中加入新构件的过程必须简单、方便。也就是说，软件体系结构必须是开放的。从长远观点看，具有开放结构的软件拥有真正的优势，这种软件的可维护性明显好于封

闭结构的软件。尽管采用增量模型比采用瀑布模型和快速原型模型需要更精心的设计，但在设计阶段多付出的劳动将在维护阶段获得回报。如果一个设计非常灵活而且足够开放，足以支持增量模型，那么，这样的设计将允许在不破坏产品的情况下进行维护。事实上，使用增量模型时开发软件和扩充软件功能（完善性维护）并没有本质区别，都是向现有产品中加入新构件的过程。

从某种意义上说，增量模型本身是自相矛盾的。它一方面要求开发人员把软件看作一个整体，另一方面又要求开发人员把软件看作构件序列，每个构件本质上都独立于另一个构件。除非开发人员有足够的技术能够协调好这一明显的矛盾，否则用增量模型开发出的产品可能并不令人满意。

1.2.5　螺旋模型

软件开发几乎总要冒一定风险，例如，产品交付给用户之后用户可能不满意，到了预定的交付日期软件可能还未开发出来，实际的开发成本可能超过预算，产品完成前一些关键的开发人员可能“跳槽”，产品投入市场之前竞争对手发布了一个功能相近、价格更低的软件，等等。软件风险是任何软件开发项目中都普遍存在的实际问题，项目越大，软件越复杂，承担该项目所冒的风险也越大。软件风险可能在不同程度上损害软件开发过程和软件产品质量，因此，在软件开发过程中必须及时识别和分析风险，并且采取适当措施以消除或减少风险的危害。

构建原型是一种能使某些类型的风险降至最低的方法。为了降低交付给用户的产品不能满足用户需要的风险，一种行之有效的方法是在需求分析阶段快速地构建一个原型。在后续的阶段中也可以通过构造适当的原型来降低某些技术风险。当然，原型并不能“包治百病”，对于某些类型的风险（例如，聘请不到需要的专业人员或关键的技术人员在项目完成前“跳槽”），原型方法是无能为力的。

螺旋模型的基本思想是，使用原型及其他方法来尽量降低风险。理解这种模型的一个简便方法，是把它看作在每个阶段之前都增加了风险分析过程的快速原型模型，如图 1-6 所示，图中带箭头的点画线的长度代表当前累计的开发费用，螺线旋过的角度值代表开发进度。螺旋线每个周期对应一个开发阶段，每个阶段开始时（左上象限）的任务是：确定该阶段的目标、为完成这些目标选择方案及设定这些方案的约束条件。接下来的任务是：从风险角度分析上一步的工作结果，努力排除各种潜在的风险。通常用建造原型的方法来排除风险，如果风险不能排除，则停止开发工作或大幅削减项目规模，如果成功地排除了所有风险，则启动下一个开发步骤（见图 1-6 右下象限）。在这个步骤的工作过程相当于纯粹的瀑布模型。最后是评价该阶段的工作成果并计划下一个阶段的工作。

螺旋模型有许多优点：对可选方案和约束条件的强调有利于已有软件的重用，也有助于把软件质量作为软件开发的一个重要目标：减少了过多测试（浪费资金）或测试不足（产品故障多）所带来的风险，更重要的是，在螺旋模型中维护只是模型的另一个周期，在维护和开发之间并没有本质区别。

螺旋模型主要适用于内部开发的大规模软件项目。如果进行风险分析的费用接近整个

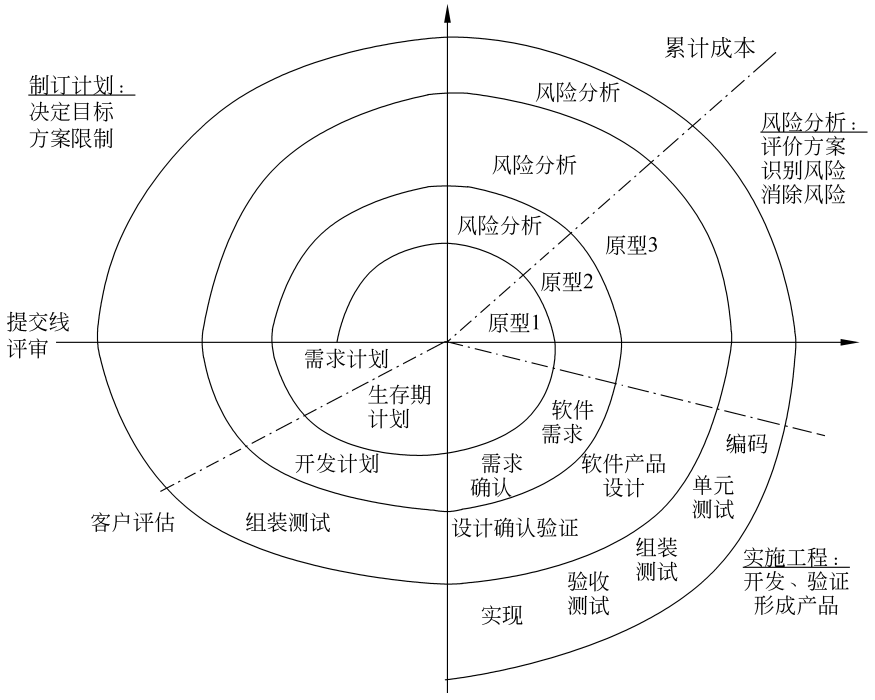


图 1-6　螺旋模型

项目的经费预算，则风险分析是不可行的。事实上，项目越大，风险也越大，因此，进行风险分析的必要性也越大。此外，只有内部开发的项目，才能在风险过大时方便地终止。

螺旋模型的主要优势在于，它是由风险驱动的；但是，这也可能是它的一个弱点。除非软件开发人员具有丰富的风险评估经验和这方面的专门知识，否则将出现真正的风险：当项目实际上正在走向灾难时，开发人员可能还认为一切正常。

1.2.6　喷泉模型

迭代是软件开发过程中普遍存在的一种内在属性。经验表明，软件过程各个阶段之间的迭代或一个阶段内各个工作步骤之间的迭代，在面向对象范型中比在结构化范型中更常见。

图 1-7 所示的喷泉模型是典型的面向对象生命周期模型。“喷泉”这个词体现了面向对象软件开发过程迭代和无缝的特性，图 1-7 中代表不同阶段的圆圈相互重叠，这明确表示两个活动之间存在交迭；而面向对象方法在概念和表示方法上的一致性，保证了各项开发活动之间的无缝过渡。事实上，用面向对象方法开发软件时，在分析、设计、编码等开发活动之间并不存在明显的边界。图 1-7 中在一个阶段内的向下箭头代表该阶段内的迭代（或求精）；较小的圆圈代表维护，圆圈较小象征着采用了面向对象范型之后维护时间缩短了。

为避免使用喷泉模型开发软件时开发过程过分无序，应该把一个线性过程（例如，快速原型模型或螺旋模型中的中心垂线）作为总目标。但是，同时也应该记住，面向对象范型本身要求经常对开发活动进行迭代或求精。

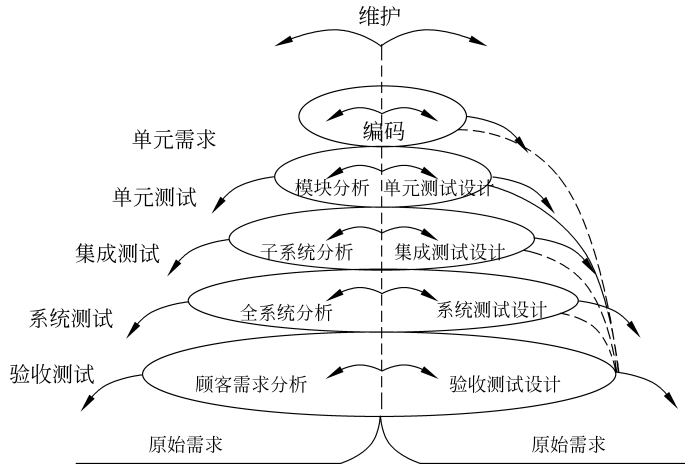


图 1-7　喷泉模型

1.2.7　Rational 统一过程

Rational 统一过程(Rational unified process,RUP)是由 Rational 软件公司(已被 IBM 并购)推出的一个软件开发过程框架。所谓软件开发过程框架指团队根据具体的项目组或软件开发企业的不同需求,能够定义、配置、定制和实施一致的软件开发过程。

通过总结经过多年实践和验证的各种软件开发最佳实践,RUP 框架提出了一组丰富的软件工程原则的指导信息。它既适用于不同规模和不同复杂度的项目,也适用于不同的开发环境和领域。

RUP 包含以下 3 个核心元素。

(1) 用于成功开发软件的一组基本观念和原则。这些观念和原则是开发 RUP 的基础,包含了后面要讲述的 6 条“最佳实践”和 10 个“流程要素”。

(2) 一套关于可重用方法内容和过程构建的框架。可以在这个框架之下定义自己的开发方法和过程。

(3) 基础的方法和过程定义语言。这就是统一方法架构元模型(unified method architecture, UMA)。该模型提供了用于描述方法内容及过程的语言。这种新语言统一了不同方法和过程工程语言。

1.2.7.1　最佳实践

软件开发是一项团队活动。理想情况下,此类活动包括在贯穿软件生命周期的各阶段中配合默契的团队工作。此类活动既不是科学研究也不是工程设计——至少从基于确凿事实的可量化原则的角度来说不是。软件开发工作假设开发人员可以计划和创建单独片段并稍后将它们集成起来(就如同在构建桥梁或宇航飞船),经常会在截止期限、预算和用户满意度的某一方面失败。

在缺少系统的理论指导时,就必须依靠称为“最佳实践”的软件开发技术,其价值已在不

同软件开发团队的多年应用中经过反复验证。RUP 的“最佳实践”描述了一个指导开发团队达成目标的迭代和递增式的软件开发过程，而不是强制规定软件项目的“计划—构建—集成”这类活动顺序。以下分别讲述 RUP 的 6 条最佳实践。

1）迭代式开发

采用传统的顺序开发方法（瀑布模型）是不可能完成客户需要的大型复杂软件系统的开发工作的。事实上，在整个软件开发过程中，客户的需求会经常改变，因此，需要有一种能够通过一系列细化、若干个渐进的反复过程而得出有效解决方案的迭代式方法。迭代式开发如图 1-8 所示。

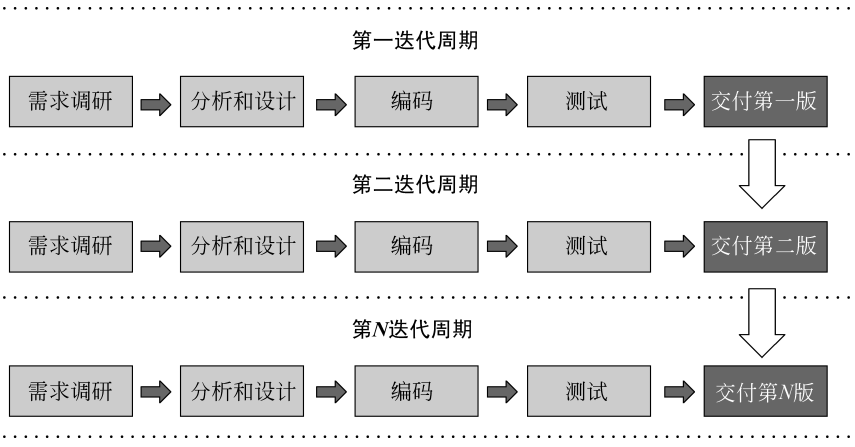


图 1-8　迭代式开发

迭代式开发允许需求在每次迭代过程中发生变化，这种开发方法通过一系列细化来加深对问题的理解，因此能更容易地容纳需求的变更。

也可以把软件开发过程看作一个风险管理过程，迭代式开发通过采用可验证的方法来减少风险。采用迭代式开发方法，每个迭代过程以完成可执行版本结束，这可以让最终用户不断地介入和提出反馈意见。同时，开发团队根据产生的结果可以频繁地进行状态检查以确保项目能按时进行。迭代式方法同样使需求、特色和日程上战略性的变化更为容易。

2）管理需求

在开发软件的过程中，客户需求将不断地发生变化，因此，确定系统的需求是一个连续的过程。RUP 描述了如何提取、组织系统的功能性需求和约束条件并把它们文档化。经验表明，使用用例和脚本是捕获功能性需求的有效方法，RUP 采用用例分析来捕获需求，并由它们驱动设计和实现。

3）使用基于组件的架构

所谓组件就是功能清晰的模块或子系统。系统可以由已经存在的、由第三方开发商提供的组件构成，因此组件使软件重用成为可能。RUP 提供了使用现有的或新开发的组件定义架构的系统化方法，从而有助于降低软件开发的复杂性，提高软件重用率。

4）可视化建模

为了更好地理解问题，人们常常采用建立问题模型的方法。所谓模型，就是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。由于应用领域不同，模型

可以有文字、图形、数学表达式等多种形式，一般说来，使用可视化的图形更容易令人理解。

RUP 与可视化的统建模语言(unified modeling language，UML)紧密地联系在一起，在开发过程中建立起软件系统的可视化模型，可以帮助人们提高管理软件复杂性的能力。

5) 验证软件质量

某些软件不受用户欢迎的一个重要原因是其质量低下。在软件投入运行后再去查找和修改出现的问题，比在开发的早期阶段就进行这项工作需要花费更多的人力和时间。在 RUP 中，软件质量评估不再是一种事后的行为或由单独小组进行的孤立活动，而是内嵌在贯穿整个开发过程的、由全体成员参与的所有活动中。

6) 控制软件变更

在变更是不可避免的环境中，必须具有管理变更的能力，才能确保每个修改都是可接受的而且能被跟踪的。RUP 描述了如何控制、跟踪和监控修改，以确保迭代开发的成功。

1.2.7.2　RUP 的十大要素

通常我们在软件的质量和开发效率之间需要达到一个平衡。这里的关键就是我们需要了解软件过程中一些必要的元素，并且遵循某些原则来定制软件过程以满足项目的特定需求。下面讲述 RUP 的十大要素。

1) 前景：制定前景

有一个清晰的前景(vision)是开发一个满足项目干系人(stakeholder)需求的产品的关键。

前景给更详细的技术需求提供了一个高层的、有时候是合同式的基础。正像这个术语隐含的意义那样，前景是软件项目的一个清晰的、通常是高层的视图，它能在过程中被任意一个决策者或实施者借用。前景捕获了非常高层的需求和设计约束，让它的读者能够理解即将开发的系统。前景向项目审批流程提供输入信息，因此与商业理由密切相关。前景传达了有关项目的基本信息，包括为什么要进行这个项目，以及这个项目具体做什么，同时前景还是验证未来决策的标尺。

前景的内容将回答以下问题：

(1) 关键术语是什么？(词汇表)

(2) 我们要尝试解决什么问题？(问题声明)

(3) 谁是项目干系人？谁是用户？他们的需要是什么？

(4) 产品的特性是什么？

(5) 功能性需求是什么？(用例)

(6) 非功能性需求是什么？

(7) 设计约束是什么？

制定一个清晰的前景和一组让人可以理解的需求，是需求规程的基础，也是用来平衡相互竞争的项目干系人之间的优先级的一个原则。这里包括分析问题，理解项目干系人的需求，定义系统，以及管理需求变化。

2) 计划：按计划管理

产品的质量是和产品的计划息息相关的。

在 RUP 中，软件开发计划(software development plan，SDP)综合了管理项目所需的各种信息，也许会包括一些在先启阶段开发的单独的内容。SDP 必须在整个项目中被维护和

更新。

SDP 定义了项目时间表(包括项目计划和迭代计划)和资源需求(资源和工具),可以根据项目进度表来跟踪项目进展。同时也指导了其他过程内容的计划:项目组织、需求管理计划、配置管理计划、问题解决计划、QA 计划、测试计划、评估计划及产品验收计划。

软件开发计划的格式远远没有计划活动本身及驱动这些活动的思想重要。正如 DwightD.Eisenhower 所说:“计划并不重要,重要的是实施计划。”

计划、风险、业务案例、架构及控制变更一起成为 RUP 中项目管理流程的要点。项目管理流程包括以下活动:构思项目、评估项目规模和风险、监测与控制项目、计划和评估每个迭代及阶段。

3) 风险:降低风险并跟踪相关问题

RUP 的要点之一是在项目早期就标识并处理最大的风险。项目组标识的每个风险都应该有一个相应的缓解或解决计划。风险列表应该既作为项目活动的计划工具,又作为组织迭代的基础。

4) 业务案例:检验业务案例

业务案例从业务的立场提供了确定该项目是否值得投资的必要信息。

业务案例主要用于为实现项目前景而制订经济计划。计划制订之后,业务案例就用来对项目提供的投资收益率(rate of return on investment,ROI)进行精确的评估。它能够提供项目的合理依据,并确定对项目的有关经济约束。它向经济决策者提供关于项目价值的信息,并用于确定该项目是否应继续前进。

业务案例的描述不应深挖问题的细节,而应就为什么需要该产品树立一个有说服力的论点。它必须简短,这样就容易让所有项目团队成员理解并牢记。在关键里程碑处,将重新检验业务案例,以查看预期收益和成本的估计值是否仍然准确,以及该项目是否应继续。

5) 架构:设计组件架构

在 RUP 中,软件系统的架构指系统关键部件的组织或结构,组件之间通过接口交互,而组件是由一些更小的组件和接口组成的。

RUP 提供了一种设计、开发、验证架构的系统化的方法。在分析和设计流程中包括以下步骤:定义候选架构、精化架构、分析行为(用例分析)和设计组件。

要陈述和讨论软件架构,必须先定义一种架构表示法,以便描述架构的重要方面。在 RUP 中,架构由软件架构文档通过多个视图表示。每个视图都描述了某组项目干系人所关心的系统的某个方面。项目干系人有最终用户、设计人员、经理、系统工程师、系统管理员等。软件架构文档使系统架构师和其他项目组成员能够与架构相关的重大决策进行有效的交流。

6) 原型:增量地构建和测试产品

RUP 是为了尽早排除问题、解决风险和问题而构建、测试和评估产品的可执行版本的一种迭代方法。

递增地构造和测试系统的组件,这是实施和测试规程及原则,通过迭代证明价值的“要素”。

7) 评估:定期评估结果

顾名思义,RUP 的迭代评估审查了迭代的结果。评估得出了迭代满足需求规范的程度,同时还包括学到的教训和实施的过程改进。

根据项目的规模、风险及迭代的特点，评估可以是对演示及其结果的一条简单的记录，也可能是一个完整的、正式的测试评审记录。

这里的关键是既关注过程问题又关注产品问题。越早发现问题就能减少越多的问题。

8）变更请求：管理并控制变更

RUP 的配置和变更管理流程的要点是当变化发生时管理和控制项目的规模，并且贯穿整个生命周期。其目的是考虑所有的涉众需求，在尽量满足需求的同时又能及时地交付合格的产品。

用户拿到产品的第一个原型后（往往在这之前就会要求变更），他们会要求变更。重要的是，变更的提出和管理过程始终保持一致。

在 RUP 中，变更请求通常用于记录与跟踪缺陷和增强功能的要求，或者对产品提出的任何其他类型的变更请求。变更请求提供了相应的手段来评估一个变更的潜在影响，同时记录就这些变更所作出的决策。他们也帮助确保所有项目组成员都能理解变更的潜在影响。

9）用户支持：部署可用的产品

在 RUP 中，部署流程的要点是包装和交付产品，同时交付有助于最终用户学习、使用和维护产品的所有必要的材料。

项目组至少要给用户提供一个用户指南（也许是通过联机帮助的方式提供），可能还有一个安装指南和版本发布说明。

根据产品的复杂度，用户也许还需要相应的培训材料。最后，通过一个材料清单（bill of materials，BOM）清楚地记录哪些材料应该和产品一起交付。

10）过程：采用适合项目的过程

选择适合正开发的产品类型的流程是非常必要的。即使在选定一个流程后，也不能盲目遵循这个流程，必须应用常理和经验来配置流程和工具，以满足组织和项目的需要。

1.2.7.3　RUP 生命周期

1）核心工作流

RUP 中有 9 个核心工作流，如图 1-9 所示。其中前 6 个为核心过程工作流程，后 3 个为核心支持工作流程。下面简要地叙述各个工作流程的基本任务。

业务建模：深入了解使用目标系统的机构及其商业运作，评估目标系统对使用它的机构的影响。

需求：捕获客户的需求，并且使开发人员和用户达成对需求描述的共识。

分析与设计：把需求分析的结果转化成分析模型与设计模型。

实现：把设计模型转换成实现结果（形式化地定义代码结构；用构件实现类和对象；对开发出的构件进行单元测试；把不同实现人员开发出的模块集成为可执行的系统）。

测试：检查各子系统的交互与集成，验证所有需求是否都被正确地实现了，识别、确认缺陷并确保在软件部署之前消除缺陷。

部署：成功地生成目标系统的可运行的版本，并把软件移交给最终用户。

配置与变更管理：跟踪并维护在软件开发过程中产生的所有制品的完整性和一致性。

项目管理：提供项目管理框架，为软件开发项目制订计划、人员配备、执行和监控等方

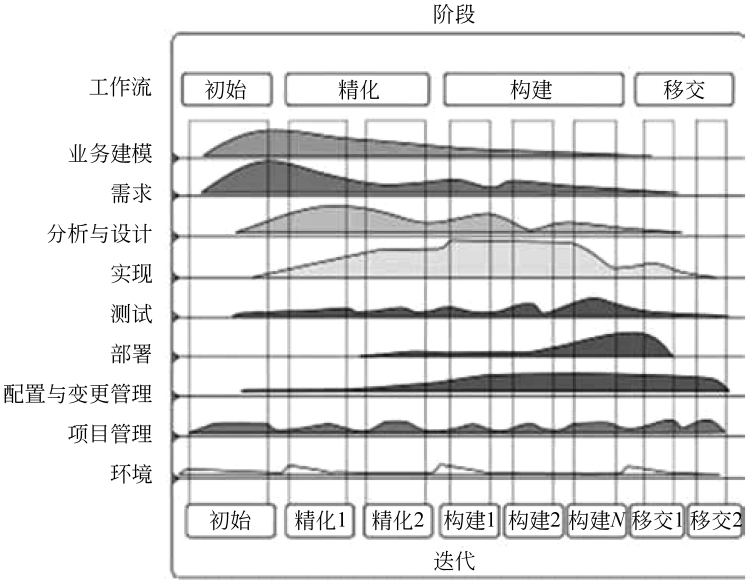


图 1-9　方法内容定义与方法内容在流程中的应用

面的实用准则，并为风险管理提供框架。

环境：向软件开发机构提供软件开发环境，包括过程管理和工具支持。

2）工作阶段

RUP 的软件生命周期按时间分成 4 个顺序阶段，每个阶段以一个主要里程碑结束；每个阶段的目标通过一次或多次迭代来完成。在每个阶段结束时执行一个评估，确定是否达到了该阶段的目标。如果评估令人满意，则允许项目进入下一个阶段。如果未能通过评估，则决策者应该作出决定，要么中止该项目，要么重做该阶段的工作。

下面简述 4 个阶段的工作目标（见图 1-10）。

先启阶段：建立业务模型，定义最终产品视图，并且确定项目的范围。

精化阶段：设计并确定系统的体系结构，制订项目计划，确定资源需求。

构建阶段：开发出所有构件和应用程序，把它们集成为客户需要的产品，并且详尽地测试所有功能。

移交阶段：把开发出的产品提交给用户使用。

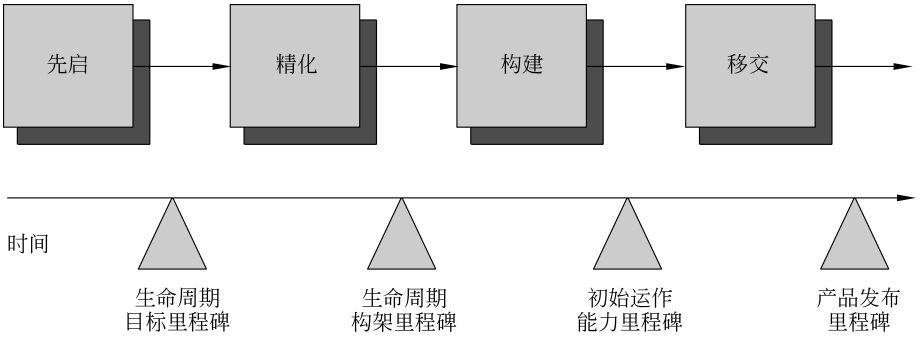


图 1-10　迭代过程的阶段和里程碑

3）RUP 迭代式开发

RUP 强调采用迭代和渐增的方式来开发软件，整个项目开发过程由多个迭代过程组成。在每次迭代中只考虑系统的一部分需求，针对这部分需求进行分析、设计、实现、测试、部署等工作，每次迭代都是在系统已完成部分的基础上进行的，每次给系统增加一些新的功能，如此循环往复地进行下去，直至完成最终项目。

事实上，RUP 重复了一系列组成软件生命周期的循环。每次循环都经历一个完整的生命周期，每次循环结束都向用户交付产品的一个可运行的版本。前面已经讲过，每个生命周期包含 4 个连续的阶段，在每个阶段结束前有一个里程碑来评估该阶段的目标是否已经实现，如果评估结果令人满意，则可以开始下一阶段的工作。

每个阶段又进一步细分为一次或多次迭代过程。项目经理根据当前迭代所处的阶段及上次迭代的结果，对核心工作流程中的活动进行适当的参见，以完成一次具体的迭代过程。在每个生命周期中都轮流访问这些核心工作流程，但是，在不同的迭代过程中是以不同的工作重点和强度对这些核心工作流程进行访问的。例如，在构件阶段的最后一次迭代过程中，可能还需要做一点需求分析工作，但是需求分析已经不像初始阶段和精化阶段的第 1 个迭代过程中那样是主要工作了，而在移交阶段的第 2 个迭代过程中，就完全没有需求分析工作了。同样，在精化阶段的第 2 个迭代过程及构件阶段中，主要工作是实现，而在移交阶段的第 2 个迭代过程中，实现工作已经很少了。

1.3　软件过程在软件质量管理中的意义

1.3.1　软件过程的定义

软件过程指软件生存周期中的一系列相关过程，是将用户需求转化为可执行系统的演化过程所进行的软件工程活动的全体，是用于生产软件产品的工具、方法和实践的集合。值得提出的是，软件过程中的“过程”是创建一个产品或完成某些任务的一种系统化的方法和工作过程，其执行者不再仅仅是计算机，而经常是由具体承担任务的软件开发人员使用给定的开发工具来执行，它甚至可以是一个无法在计算机上运行的过程，完全由人工或人工借助计算机以外的工具来完成。软件过程是关系复杂的软件活动的集合，各活动之间有着严格、密切的关系。有的是异步并行的，有的互为条件，因此实际的软件过程中的软件活动存在复杂的网状关系。如何正确、有效地对软件活动进行管理成为软件过程管理的一个很重要的方面。

软件过程是改进软件质量和组织性能的主要因素之一。Dowson 曾指出：“软件产品质量在很大程度上依赖软件过程，尤其是大规模的软件开发更是如此。”因此，不少软件开发企业力图通过改进开发过程来改善软件产品的质量，提高软件生产率、缩短产品的开发时间，从而增加企业的竞争力和效益。

1.3.2　软件过程描述

软件过程描述通过某种形式化的手段对软件开发过程加以系统、严格的描述，为软件开

发人员提供一个标准的、无歧义的软件开发规范，并以此为基础辅助和指导开发人员的工作，同时对实际的软件开发过程进行监督和控制，从而保证软件产品的质量和软件生产率。通过过程描述所表示的过程模型可以看作软件开发过程的脚本，它指导开发者按照严格工程化的方法一步步地进行开发工作，软件则是依据过程描述进行的一系列软件过程活动的产品。软件过程描述与软件过程之间的关系类似于程序和进程之间的关系，软件过程是软件过程描述的动态执行，而其如何执行则是由过程描述所决定的。软件过程描述是面向过程软件开发环境的基础，因此它历来是软件过程研究的一个重点。

1987 年，Leno Osterweil 提出的“软件过程描述也是软件”思想对后来的过程描述语言产生了极其深远的影响，因为过程描述语言是用以描述一个信息产品系统化开发或修改的方法。该描述也可以使用编译或解释执行的某种程序语言进行编写。所进行的描述本身是可执行的。从这一点出发，它类似于通常的程序，所以过程描述实际上也可以称为过程程序。

1.3.3 软件过程管理

软件过程指软件生存周期所涉及的一系列相关过程。过程是活动的集合；活动是任务的集合；任务要起到把输入进行加工然后输出的作用。活动的执行可以是顺序的、重复的、并行的、嵌套的或者是有条件地引发的。软件过程主要针对软件生产和管理进行研究。图 1-11 描述了软件过程和软件项目管理的关系。软件开发包含软件过程执行和项目计划执行。软件过程确保了需求的实现和项目结果，最终提交出合格的产品。

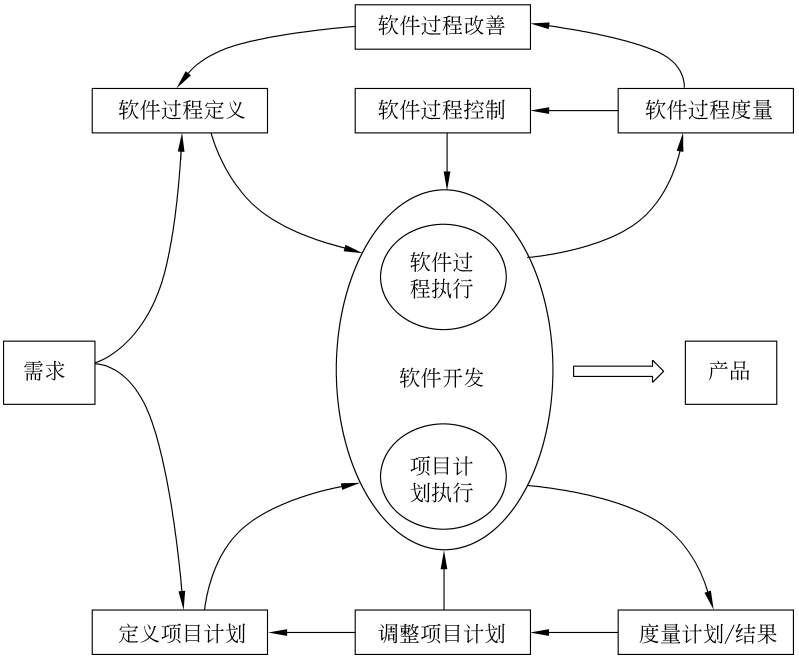


图 1-11 软件全面质量管理模型

从软件过程管理来看，合理的软件过程管理主要包括以下几个方面。

(1) 关注可执行的程序或者系统：大多数软件项目管理表明，软件开发过程中需要各种各样的文档。但是过于关注文档，或者留下一堆文档，我们会发现软件依然有客户难以容忍的“漏洞”(bug)，需要不断地打补丁来解决异常数据问题。从结果来看，软件质量是由执行的程序或系统来展现的。关注执行的程序或系统才是解决问题的核心。

(2) 一开始就充分暴露风险并及时解决：软件项目有各种各样的风险，有进度、成本、客户需求不断变更、人员不稳定等风险。因此应三思而后行。只有当目标和方向确定，充分考虑存在的风险并提出解决办法之后，才能成功。

(3) 软件质量要每时每刻都能够得到验证：在软件项目过程中，通常通过最终用户的测试来把关软件质量。但是最终客户经常关注软件功能，并不知道运行过程中产生的“bug”和其他问题对整个系统的影响。所以在提交软件产品的过程中，不能到最后才来验证软件质量。

(4) 需求的代码要能无缝链接：软件项目管理最常提到的是需求的管理，满足了客户的需求也可以看成提交了合格的软件产品。但客户有的需求是隐性的，有时客户不一定知道他们具体需要的是什么。造成明明是按照客户要求开发出来的代码却不能满足客户需求的后果。如果需求和代码是个非常复杂的跟踪关系矩阵，那么软件质量就很难得到保证。

(5) 软件在面对变更时很“健壮”：变更无所不在，除非是通用的软件。客户的个性化需求是非常多的。不同的客户有不同的需求。面对软件需求的变更，软件要在架构上灵活、“健壮”。首先一定是基于组件的，其次还要符合迭代开发、用例驱动开发和以架构为中心。同时可维护性也是非常关键的。总之，在我们追求最佳软件质量的过程中，合理的软件过程管理是软件质量的基础。要把软件项目管理建立在牢固的基础上，才能交付满足客户需求和可维护的软件产品，也就是合格的软件产品。

1.4　本章小结

软件过程是为了获得高质量软件产品所需要完成的一系列任务框架，它规定了完成各项任务的工作步骤。软件过程必须科学、合理，才能开发出高质量的软件产品。

按照在软件生命周期全过程中应完成的任务的性质，在概念上可以把软件生命周期划分成问题定义、可行性研究、需求分析、概要设计、详细设计、编码和单元测试、综合测试及维护 8 个阶段。实际从事软件开发工作时，软件规模、种类、开发环境使用的技术方法等因素，都影响阶段的划分，因此，一个科学、有效的软件过程应该定义一组适合所承担的项目特点的任务集合。

生命周期模型(软件过程模型)规定了把生命周期划分成的阶段及各个阶段的执行顺序。本章介绍了 5 类典型的软件生命周期模型。

瀑布模型历史悠久、广为人知，它的优势在于其是规范的、文档驱动的方法。这种模型的问题是，最终交付的产品可能不是用户真正需要的。

快速原型模型正是为了克服瀑布模型的缺点而提出来的。它通过快速构建起一个可运行的原型系统，让用户试用原型并收集用户反馈意见的办法，来获取用户的真实需求。

增量模型具有能在软件开发的早期阶段使投资获得明显回报和易于维护的优点。但是，要求软件具有开放结构是使用这种模型时固有的困难。

风险驱动的螺旋模型适用于大规模的内部开发项目，但是，只有在开发人员具有风险分析和排除风险的经验及专门知识时，使用这种模型才会获得成功。

当使用面向对象范型开发软件时，软件生命周期必须是循环的。也就是说，软件过程必须支持反馈和迭代。喷泉模型是一种典型的适合于面向对象范型的过程模型。

能力成熟度模型(CMM)是改进软件过程的一种策略。它的基本思想是，因为问题是由管理软件过程的不恰当方法引起的，所以运用新软件技术并不会自动提高软件生产率和软件质量，应当下大力气改进对软件过程的管理。

对软件过程的改进不可能一蹴而就，因此，CMM 以增量方式逐步引入变化，它明确地定义了 5 个不同的成熟度等级，一个软件开发组织可用一系列小的改良性步骤迈入更高的成熟度等级。

每个软件开发组织都应该选择适合本组织及所要开发的软件特点的软件生命周期模型。这样的模型应该把各种生命周期模型的合适特性有机地结合起来，以便尽量减少它们的缺点，充分利用它们的优点。

为了突出软件质量的各个指标，满足用户对软件的实际需求，进而发挥软件的作用，可借助软件质量管理，实现对软件开发、研究和设计等内容的控制，从而避免各类外界因素对软件质量造成不利影响。进而推动软件的功能性、效率性、可用性和可靠性等指标均符合软件的质量需求，满足人类的实际需要。软件质量管理可以结合用户的基本需求，加强对软件系统需求和质量标准等的控制，并保障在具体软件开发过程中，可以按照相应标准展开对软件的质量控制，从而减少软件质量问题的产生。由此可见，借助软件质量管理，可增加软件开发研究企业的效益与价值，是推动软件研发持续健康发展的基础。

CHAPTER 2 第2章

软件需求管理

需求是系统服务或约束的陈述。完整、正确、稳定和文档化的软件需求是软件开发的基础。然而，在实际的项目中，期望在项目开始阶段就获得稳定的需求是不现实的。许多软件系统的开发是开创性的，面对全新的系统，无论是用户还是开发人员都缺乏完整、准确的认识，随着项目的进行，用户需求才越来越清楚。同时，随着对软件应用需求的日益迫切，激烈的市场竞争带来软件开发窗口的缩小，软件开发不可能等到软件需求完全固化的情况下进行，从而导致软件需求在整个开发过程中处于不断调整和完善的状态，因此，需求的管理是软件质量管理中需要解决的一个关键问题。

2.1 软件需求的层次与要求

软件需求一般可以分为三个不同的层次，包括业务需求、用户需求和功能需求。此外，每个系统还有一些非功能需求，如性能需求、约束等。

(1) 业务需求。业务需求通常来源于产品的策划部门、实际用户的管理者或者项目的投资者，它描述了为什么要开始这样一个软件项目，即组织或客户高层想要达到什么目标。

(2) 用户需求。用户需求描述了用户的目标，即实际用户用产品可以实现的任务。

(3) 功能需求。功能需求是从开发者的角度规定了开发者必须在产品中实现的功能。软件需求规格说明的编写完成意味着功能需求的建立。当软件需求规格说明经过确认和评审后，需求基线正式确立。

(4) 性能需求。性能需求指的是软件系统应达到的技术指标，是对产品的功能描述做的补充。这些技术指标一般包括可用性、可移植性、效率、可靠性和可维护性等。

(5) 约束。约束指的是限制了开发人员设计和构建系统时的选择范围，如开发语言、使用的数据库等。

在理想的情况下，每一条业务需求、用户需求和功能需求的描述都应该具备以下特点。

(1) 完整性。每一项需求都必须完整地描述可交付使用的功能，它必须包含开发人员设计和实现这项功能需要的所有信息。

(2) 正确性。每一项需求都必须准确地描述将要开发的功能。如果某一项软件需求与其相对应的系统需求发生了冲突就是不正确的。

(3) 可行性。需求必须可以在系统及其运行环境的抑制能力和约束条件内实现。

(4) 必要性。每一项需求记录的功能都必须是用户真正需要的，或者是为符合外部系

统需求或某一标准而必须具备的功能。对于每项需求都必须可以追溯至特定的客户需求的来源。

(5) 有优先次序。为每一项功能需求、特性或用例指定一个实现的优先次序,以表明他在产品的某一版本中的重要程度。

(6) 无歧义。一项需求描述对所有读者应该只有一种一致的解释,然而用自然语言描述需求却极易产生歧义。

(7) 可验证性。如果某些需求不可验证,那么判定其实现的正确与否就不是客观分析,而是主观论断。不完备、不一致、有歧义或不可行的需求也是不可验证的。

软件需求规格说明中所包含的整体性需求还必须具有以下特征。

(1) 完整性。不能遗漏任何需求或者必要的信息。

(2) 一致性。需求的一致性指需求不会与同一类型的其他需求或更高层次的业务、系统或用户需求发生冲突。

(3) 可修改性。必须能够对软件需求规格说明作必要的修订,并可以维护每项需求的历史记录。

(4) 可跟踪性。需求必须可以跟踪,这样才能找到它的来源、它对应的设计单元、实现它的源代码及用于验证其是否被正确实现的测试用例。

2.2 软件需求工程

对软件需求的工程化方法的实践和研究,导致软件工程的一个子领域——需求工程(requirement engineering)逐渐形成。并且,人们逐渐认识到对软件需求的分析活动不仅限于软件产品开发初期,而是贯穿整个软件生存周期。

通常在需求工程过程中,需求的获取、分析、规约和验证活动统称为需求开发部分。需求的管理活动,又分为需求确认、需求变更、需求评审和需求跟踪几个子活动,如图 2-1 所示。

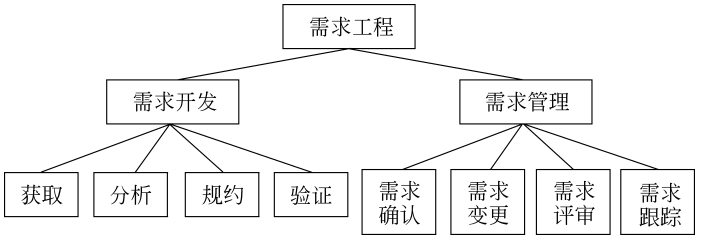


图 2-1 需求工程活动分解

狭义的需求管理并不包括需求的收集和分析,而是认为需求已经明确,仅对其进行管理。广义的需求管理应包括用户需求的收集、分析和验证等内容。

需求开发过程关注的是如何产生、分析和确认用户需求、产品需求和产品构件需求。这些需求说明相关人员的需要,包括与产品生命周期各阶段及产品属性有关的需求,也包括因选择某技术解决方案而产生的限制条件。产品生命周期的各个阶段需要进行标识以细化需

求。除用户需求外，选定的解决方案也可派生出产品及产品构件需求。

需求管理过程的目标是管理产品和产品构件的需求，并识别需求与项目计划及工作产品的不一致之处。需求管理的意义在于使产品的目标可以满足其责任人和用户的需求，当需求发生变更时，要对计划和受影响的工作产品进行及时调整，保持一致性。具体化就是：

(1) 从被认可的需求提供者那里收集需求，并与他们共同评审，以便在将需求纳入项目计划之前，解决不明确的问题并排除误解。

(2) 一旦需求提供者和需求接收者达成了协议，就可以从项目参与者那里获得对需求的承诺。

(3) 随着项目进展，当计划、工作产品和需求之间出现不一致时，要更改需求或更改计划。

(4) 需求管理要将需求的变更及其变更的理由文档化，并维持源需求与所有产品和产品构件的需求之间的双向可跟踪性。

需求管理是分析问题、获取用户需求并将需求进行固化和确认的一个管理活动，其过程是使用户与开发团队对需求达成一致的过程，并作为对需求变更进行管理的一个手段，在软件开发管理中起着积极的作用。

需求管理的目的是使用户对系统的期望和要求是合理的，需求管理应使用户方和实施方对这种期望和要求的内容建立共识并可进行维护。需求管理有如下两个目标：一是规范控制相关需求，为软件系统的范围确立基线。二是通过管理手段来保持软件开发工作与用户需求一致。要实现上述两个目标就必须在需求管理层面做到需求的跟踪管理、需求变更的管理。此外，CMM 还制定了几个关键实践域，执行约定、执行能为、执行活动、测量和分析、验证执行，来达到这两个目标。需求管理严格来说属于需求工程的具体分支，在现代工程及软件实施需求管理中发挥着越来越重要的作用。在各类规范标准中，需求管理都有具体的描述。

对于软件项目而言，其需求的不确定性、二义性等已经成为整个项目的主要风险之一，项目可以开展的重要基础是确定好的需求。系统和软件工程词汇表从系统外部行为、内部特性及文档化要求各个方面描述了需求的定义：用户为解决某个问题或达到某个目标而需具备的条件或能为；系统或系统组件为符合合同、标准、规范或其他正式文档而必须满足的条件或者具备的能力；上述第一项或第二项中定义的条件和能力的文档表达。需求是将用户解决问题的意向和想法经过不断的完善和分析形成可以转化成计算机软件系统的模型，通常具有很大的模糊性和不确定性，一般来说通过标准化规范文档来对需求提出方和开发方进行约束，但文档也可随着深入沟通和交流有条件地变更。一般将需求的相关活动分为需求开发和需求管理。需求开发主要是通过编写代码等方式将用户的需求形成信息系统并通过测试验证确保系统能够正常使用。

需求管理的目的是通过一系列方法将用户的需求充分挖掘，使用户和开发团队之间达成对需求的共同理解与确认，并应该在整个软件生命周期中对需求变更和需求跟踪进行有效的管理。

因此软件需求管理是一项能将需求提取、组织并形成文档化的系统方法，是在出现变更需求时能够将相关干系人对变更范围、目的、影响保持一致的过程。需求管理主要包括需求获取、需求确认、需求状态跟踪、需求变更管理等工作环节，是软件开发过程中控制和维持需

求约定的主要活动。

2.3 需求开发

需求开发过程指从情况收集、分析和评价到编写文档等一系列产生需求的复杂活动，主要分为用户需求获取、需求分析、需求规约和需求验证等过程。这几个阶段不一定是遵循线性顺序进行的，其中的活动经常是相互穿插和反复交互的。

2.3.1 需求获取

需求的获取是需求分析人员与系统用户一起工作以明确用户需求的过程。一个软件开发部门的业务分析人员要和业务部门的领导、主管、业务人员进行访谈和讨论，从而在宏观上把握需求，同时逐步了解客户需要及业务流程，与业务人员通过不断的交流沟通对项目需求达成一致。

对于开发工程团队来说，应该利用需求获取来理解用户需要通过软件系统做些什么，并在修改后向用户提供建议。改进有关的问题和要求，逐步实现用户使用软件的目的，并在使用过程中逐步采用相关的方法和工具，以满足用户的实际需要。

捕获需求的方法有很多种，需要在理解用户单位的组织结构的前提下与用户就需求问题进行交流。需求捕获的成败在于需求分析人员能否快速理解用户业务的痛点和需求提出的初衷，获取过程必须遵循由表及内、层层分析的原则，将整个需求流程了解清楚后再针对细节做持续沟通。捕获是个科学性的过程，捕获的方法主要有：用户访谈、问卷调查、供应商交流、同业调研、现场观摩和专题讨论，上述方法均有适合自己的应用场景和缺陷，在实际运用中必须结合需求的现实情况进行选择和组合搭配。

（1）同业调研

同业调研是公司开发重大项目时常用的一种需求获取的方法，开发部门与业务部门一同通过对有先进经验的同业进行可行性与应用性调研，了解吸收其经验教训，减少在系统规划期间的风险，通过衡量自身的情况快速选择适合的方案。

对于同业调研而言，不一定都选择行业领先的公司，与本公司处于相同发展阶段的公司同样也要积极加入拜访行程中，在了解同业优秀公司的同时再了解其他公司面临的困难和计划的解决思路，这样对自身方案的设计规划有着极强的借鉴作用。调研时要与相关公司的领导从宏观层面了解情况用于建立整体的概念，也要深入基层一线了解他们对相关问题的看法，必要时可以用户问卷调查的方式收集相关资料。调研前需要做好相关准备工作明确调研的目的、关注点、需要调研人的角色等内容。调研后要及时整理相关资料，最好能形成调研报告，调研报告中需要特别指出哪些调研拜访结果可能与实际情况不符，形成客观有效的资料用于公司项目决策的重要参考。

（2）用户访谈

用户访谈是最基本、最常用的技术，是需求捕获的一项重要方法。它非常适合用于公司内部系统的开发工作，可以与目标用户面对面地进行有效沟通，通过访谈的方式将双方最关

心的内容进行深入交流，使需求沟通人员能直观地感受到问题最核心的部分。但用户访谈需要占用双方比较长的时间，而且在需求访谈之前双方应提前做一定准备，才能加强现场访谈的效果。其缺陷在于因访谈用户有限而受单一用户信息提供的影响较大。除此之外，用户访谈还有其他缺陷需要实际操作时进行逐一发现解决。用户访谈最好要与其他方法进行结合使用才能发挥更大的作用。

需求获取人员与用户常常存在知识结构不一致的问题，对于相关术语和知识的存在不同的理解，对专业部分极易交流困难形成理解偏差，这就需要与用户所属机构的资深人士或相关领域专家进行有效交流来弥补。访谈者不要预设立场和答案，否则会在实际访谈过程中出现有诱导性的谈话内容，导致获取错误的信息。访谈时间极易因为被访谈者的工作等原因被极度压缩，导致相较预想的效果大打折扣。

(3) 专题讨论

专题讨论是在获取需求及需求沟通阶段非常重要的一个沟通方法，其可以将所有干系人和开发人员聚集在一起，共同讨论设定好的议题。它的目的在于可以使各干系人充分共享资源和各自的关键信息，在明确的议题范围下讨论解决方案、快速达成共识，对相关分歧也可以准确记录。

专题讨论应用范围很广，在需求管理的实际应用中，专题讨论在日常需求管理和项目需求管理中起到了很大的作用。在日常需求管理中，专题讨论针对版本内的需求方案采用统一会议的方式来进行评估，如果需求文档完善，则需求分析人员可以组织一同讨论分析，如在评估过程中发现需求文档有不完善的情况可以组织业务部门人员参与评估。在实际应用中，这种方式比较普遍而且对需求评估分析的效果起到了良好的沟通确认作用。在项目建设过程中，专题讨论用于项目建设的多个节点，基本贯穿项目需求管理的全过程。

专题讨论也有其缺陷，即涉及人数较多时，不容易控制会谈范围，需要主持人时刻关注讨论范围，及时中止不相干讨论，控制整个讨论的节奏与氛围。特别是在进行项目专题讨论时，必须避免讨论范围的增加和对具体开发设计细节的过度关注，对于会议上不能确定的内容要记录下来不作过多讨论，这样才可以控制整个会议的时长，增加讨论效率。

专题讨论分为以下步骤：会前准备、会中控制并记录、会后总结确认。在专题讨论会开始前，应确认好参会人员，并将会议议题等相关内容提前发给各参会人员，便于参会人员提前思考和准备自身材料并将有关想法和方案及时请示领导。会议进行时必须指定专门的会议记录人员，用于记录会议形成的一致决议和分歧，主持人应对各内容时间和讨论规则作好预先设定并宣导，及时控制会议讨论范围，调动会议气氛。会议结束后，会议记录人员应及时编写会议纪要，会议纪要应对阶段性意见进行精简总结，删除不具可行性的意见和建议，并着重对会议分歧进行具体描述以便后续再讨论一致的解决意见。会议记录人员应将会议纪要发给参会人员确认，根据反馈意见修改后发出最终版的会议纪要并抄送相关领导，作为需求沟通中有约束效力的文档存档。

2.3.2　需求分析

从用户处获取的原始需求，还需要对其进行进一步的分析和整理，对其分散获取的各项具体建立逻辑关系，明确软件类的需求，并对其进行分类，确定其需求的优先级和重要程度

等。主要的需求分析方法如下。

(1) 结构化分析方法的主要特点是“自顶向下、逐层分解”,利用图形、表格等描述方式表达需求,对需求问题进行分析,具体采用的工具有:Data FlowDiagram、Data Dictionary、E-R 图、判定表和判定树、结构化语言。

总体上看,结构化分析方法是一种强烈依赖数据流图的自上而下的建模方法,在具体的项目中,结构化分析方法的具体操作方式如下:①建立系统的物理模型;②建立系统的逻辑模型;③划清人机界限。

(2) 基于用例的分析方法,主要由成熟度高、规模大和分工明确的开发公司采用,针对大型的软件项目,开发方会根据获取的需求来形成可视化的程序实例,模拟出系统的各项功能、使用流程和数据项,建立可供需求分析的用例模型。

使用用例分析方法时可遵循以下步骤:①界定系统使用者;②分析整理需求形成用例;③形成用例图;④对用例进行详细描述。

2.3.3 需求规约

需求规约是以规约化的方式将获取和分析后的需求转化为软件需求规格说明书的过程。

规格说明书需要参与描述用户需求和系统需求的过程。它不仅要反映用户的实际需求,而且要尽量用基本的词汇,简明扼要,在保证其中的整体性、操作性及验证性的基础上用简单的问题来表达。

编写规格说明是清楚、准确地编写用户需要和约束文档的过程,它是最终用户和开发机构之间的技术合同书,是开发者开发软件系统的依据,也是最终用户验收软件系统的依据。

编写需求规格说明书应确保需求的完整性、一致性、正确性、无二义性、易于追溯、可测试性及可行性。

2.3.4 需求验证

需求验证是需求开发过程中的最后一部分,需求验证所包括的活动是为了确定以下几方面的内容:

(1) 软件需求规格说明正确描述了预期的系统行为和特征;

(2) 需求的完整性和高质量;

(3) 需求的一致性;

(4) 软件的需求分析,为接下来的功能说明书和系统详细设计及测试提供了基础。

需求验证过程中,技术人员和业务人员需与客户方面的决策共同进行工作,其主要目的是确保对需求进行审查,并通过这些要求充分体现质量特性。第一,让用户明确 SRS 是否能够完整地描述他们的需求。第二,根据相关文件,验证内容通常包括审查 SRS,以确认不仅需要提交相关要求的人员,还需要分析员和测试员,以及需要修改控制用户的一般需求。第三,确保检测范围,产品批准标准和其他许多方面都是完全符合用户需求的。

需求验证一般通过评审或测试的方式进行,使获取的需求得到相关人员(应用系统的使用者、应用系统的直接验收者、高层经理、项目经理、业务专家、产品经理、项目组成员、测试

人员等)的共识,这种共识是建立在相关人员反复沟通的基础上的。作为需求开发成果的需求规格说明书应具有正确性、无二义性、完整性、可验证性、一致性、可修改性、可跟踪性,注释合理适用,让非计算机人员能够理解。

需求开发过程中还没有形成任何软件,不可能进行任何实质的软件测试,但是可以在软件开发组设计编码之前,以需求为基础建立概念性的测试用例,并使用这些例子来发现软件规格说明书中的错误、二义性,以及是否有遗漏等。

2.4　需求管理

需求管理包括需求的确认、变更、评审和跟踪几个过程。

2.4.1　需求确认

常见的方式是通过召开需求确认会议来对需求进行交流沟通和确认,通常由全体项目利益相关方参加,共同探讨,对项目的需求达成共识。

在需求确认会议上,一定要先针对全局性的问题进行交流,千万不要针对部分人感兴趣的问题进行长时间讨论,然后再对根据原型法得到的需求规格说明书中的内容、差异逐一审核,业务人员通过对项目需求的讲解,对需求可行性的分析,需求优先级的确认等,最终与开发人员达成一致,并且要进行书面确认。

软件项目需求确认的最终书面确认,是需求管理的重要环节,为项目开发过程中的需求变更管理提供了依据。

2.4.2　需求变更

客户需求在项目执行过程中虽然会一直存在,但不一定是绝对的,因此我们需要考虑到如何管理需求的变化而对用户需求进行适当更改,控制和管理。首先,需要根据必须给予满足的需求做一些会影响到整个项目正常交付的重要的变更。另外,也可以做一个不会影响系统交付的改良性变更,但如果用户不满意,则整个项目的价值将会改变。

由于需求分析不全、业务需求不断增加和变更、需求不清楚等原因,需求在项目的整个生命周期都会不可避免地发生变化。需求管理是软件项目开发过程中控制和维持需求约定的活动,包括变更控制、版本控制、需求跟踪和需求状态跟踪等工作。项目业务需求的变更是影响项目进度的主要因素,一定要严格控制变更,避免无限制地进行需求变更。

在项目开发过程中,要做好应对需求变化的准备,需求管理的方法主要有以下几点。

(1) 建立需求变更控制流程。制定一个选择、分析和决策需求变更的过程,所有的需求变更都要遵循此过程。

(2) 进行需求变更影响分析。要及时召集业务人员和开发人员,对项目的需求变更所带来的影响进行分析,明确变更相关模块的工作量,从而帮助需求变更控制部门做出更适当的决策。

(3) 建立需求控制文档。以确定的《软件需求规格说明书》为前提,之后的需求变更要遵循变更控制过程,新的版本以前面版本为基础,要避免两个版本的混淆,确保需求的一致性。

(4) 维护需求变更历史记录。要求用户填写变更申请单发送给项目配置管理员,再通过配置员转交质保小组,负责组织专家小组和项目组成员一起讨论实施变更的可行性及实施后带来的影响。

(5) 跟踪需求状态。要保存每项需求的状态,以便于管理控制。从整体上把握每个需求的进度。

(6) 保持需求稳定性。过多的需求变更会给项目的进度造成不小的麻烦,往往会导致银行软件项目的延期,对于无法实现或是变更会带来巨大影响而将导致的进度延期,这时,我们可以将变更报告提交给用户或邀请用户进行协调会议,讨论变更取舍问题或是项目进度变更问题。在项目的后期和项目完成时间不可更改时,要冻结需求,以保证项目顺利完成,而需要新增的功能可以留待下一个版本完善。

(7) 决定变更之后,由项目经理组织实施变更,测试人员检测变更结果,而质保小组成员监督变更实施过程并协助配置管理员对变更后的成果进行版本控制。变更实施完后,上线前还需要指定人员协助用户一同测试并由用户签字同意后方可上线。

版本的控制一直存在于包括软件本身和相关文件的跟踪记录软件开发的过程中。按照版本控制的要求,可以确保集中地管理空间中的配置项并解决相关的问题,这将导致版本具有特定的可回溯性,开发团队可以以此为基础进行研发以提高开发效率。版本控制是一种固有的需求手段,也是提高开发效率的根本。

2.4.3 需求评审

需求评审的目的有两个,一是提供机会给开发人员和业务人员进行沟通,使双方通过沟通减少对需求不一致的理解;二是双方人员对需求达成共同理解或对认可的部分需要进行承诺。需求的评审可以分为正式评审和非正式评审。

需求的非正式评审是需求开发过程中最常进行的一种活动。当开发人员利用业务流程图和状态转换图来展现业务的流程时,需要多次与业务人员沟通这些模型是否真实地反映了用户需求,请业务人员为模型的正确性进行评估。

正式的评审活动通常是通过召开评审会议来进行的。需求评审会议主要是对软件规格说明进行的一次确认活动。

会议是项目进行过程中一种重要的沟通方式。为了保证评审会议高效、有序地进行,需要考虑以下几个因素。

(1) 明确与会人员的职责分工。需求评审的目的是鉴别出不完整的、错误的、遗漏的或多余的功能需求,以及这些需求是否已经被正确且清晰、无歧义地阐述,它们之间是否相互一致、没有矛盾。如果在评审会议之前不对与会人员的职责进行明确,则更多的注意力将集中到流程的讨论上,从而往往导致忽略了需求是否被正确且清晰、无歧义地阐述。

项目评审会议各方职责如下:

① 作者,即软件需求规格说明的编写者,在审查会议中主要听取其他参与人员对软件需求规格说明的评论,就他人的疑问作出回答,但不参与讨论。

② 决策者为项目领导，负责鉴别出不完整的、错误的、遗漏的或多余的功能需求。

③ 使用者为参与项目编码的人员，负责查看软件需求规格说明是否存在语意不清或者无法理解的描述。

④ 记录者则需要在会议进行中及时记录审查过程中提出的问题及缺陷，并在会议结束时，向与会人员复述记录的内容，使参与人员将精力集中到评审上而且有利于保证抓住问题的本质，以便准确地向作者传达会议评审结果。

(2) 抓住主次，突出评审重点。在评审会议中，各部门的决策者首次参与到项目中，而决策者对项目需求评审的重视程度是项目评审能否顺利开展的关键。因此要抓住决策者最迫切关心的主要问题作为会议讨论的重点。当决策者认识到项目评审的重要性时，需求评审的工作在进程上就有了保障。

(3) 建立良好的评审沟通氛围。评审过程中沟通氛围的好坏直接关系到需求基线的质量，因此建立一个良好的沟通氛围、处理好软件使用方与开发方的关系显得尤其重要。在评审的过程中，难免会因功能的实现发生一些意见分歧，应尽量避免发生争执，从产品应用的角度，找出分歧的解决方式。

围绕需求进行管理最重要的依据就是需求基线，即软件需求规格说明。而软件需求规格说明就是在评审会议中确定的。评审会议是很容易发现问题的一种形式。开发人员和业务人员在进行需求讨论时，更多关注的是业务流程的讨论，而参与的决策者可以从管理、风险的角度给出建议，这些很可能是开发人员考虑不到的。与会人员共同审阅一份文档，易于发现软件需求规格说明中描述不清的、有歧义的，以及存在潜在缺陷的地方。需求评审会议既是对项目阶段工作成果的校验也是对工作结果的一种肯定。

2.4.4　需求跟踪

需求跟踪的目的是通过需求的双向跟踪控制软件需求变化的每个状态。双向跟踪包括正向跟踪和逆向跟踪。正向跟踪指沿着软件生存周期，从分配需求开始，一直跟踪到软件需求分析、软件设计、软件实现和软件测试等后续阶段所产生的各个软件工作产品的相应元素和状态。逆向跟踪指从某个阶段的软件工作产品的某个元素开始，进行反向跟踪，直到分配需求。对于变化的部分进行逆向跟踪可以更好地控制变更所带来的影响。

软件在各个开发阶段的工作产品之间存在清晰的继承关系，以需求跟踪矩阵对这种关系做出准确的表达，那么需求更改无论出现在哪个阶段，都能沿着这条线索进行无遗漏的跟踪，对相关部分实施更改和调整，使软件更改受到控制和管理。表 2-1 所示为软件需求与分配需求的跟踪矩阵示例，还可据此编制软件各阶段的跟踪矩阵。

表 2-1　软件需求跟踪矩阵示例

分配需求	软件需求					
	A_{S1}	A_{S2}	……	A_{Si}	……	A_{Sn}
A_{R1}						
A_{R2}						

续表

分配需求	软件需求					
	A_{S1}	A_{S2}	……	A_{Si}	……	A_{Sn}
……						
A_{Rj}						
……						
A_{Rm}						

将上述跟踪矩阵合而为一，形成一个完整的需求跟踪矩阵，如表 2-2 所示。

表 2-2 完整的需求跟踪矩阵示例

项目名称					项目标识		
序号	分配需求	软件需求	概要设计	详细设计	软件实现	……	备注
1	1.1.2	2.3.1	5.3.2	3.3.1	Init	……	
2	1.1.2	2.3.2	5.3.3	3.3.2	Load	……	
……	……	……	……	……	……	……	
j	2.1.5	4.1.2	6.2.4	7.3.6	Process	……	
……	……	……	……	……	……	……	

2.5 常见的软件需求管理问题

在软件需求的日程管理中，经常出现如下问题。

（1）需求提交质量不高

业务需求提交需求前未经过细致的思考，未设立好需求的目标和范围，只是为了解决当前面临的实际问题，不能从全局考虑整体业务的流程，这使得很多功能开发后因为整体业务的关系而导致使用率不高，造成开发资源的浪费。或者出现需求处于开发中或测试时临时提出需求变更，使得很多需求不能按时上线并出现返工等情况，也进一步加剧了开发与业务之间关系紧张的局面。由于需求提交时未考虑到与其他关联系统之间的联系，再加上开发人员未对需求做出细致的分析，也没有与业务人员进行紧密的沟通，因此会因关联系统未提供支持而导致出现无法测试或改动很大的情况。

（2）需求分析不完整

虽然流程要求必须经过需求分析，但在现实工作中会因为加快进度等原因导致需求分析做得不完整或根本没有进行需求分析。有的开发人员未与需求提出人进行有效的沟通，对需求的目标实现功能会出现误解。对于部分项目类需求来说，虽然编写了相关的需求分析文档，但有些文档属于为了应付流程将已有的需求说明书等文档内容进行简单的复制粘贴，并没有在项目开发中起到与用户沟通确认、内部管理分配开发任务等作用。需求分析不

完整会造成后续相关系统进行维护开发时没有完善的历史文档对其进行回顾和梳理，影响后续需求开发效果与效率。

（3）需求排期不规范

一些开发项目组未对需求进行有效的优先级规划和排期，会引发开发与业务部门的一些争执，很多需求根据想象来进行分配开发，未考虑到需求之间可能存在相互联系或功能覆盖。同时一些开发项目组也没有考虑到业务对不同需求的紧急程度期望有差异，用户也没有深度地参与到需求排期中，往往导致开发人员不停地被不同的需求提交人催促开发上线，不能有效地将紧急且重要的需求排到优先处理，还会出现其他需求开发完毕但重要且紧急的需求又由领导层面布置任务，为了紧急完成任务导致开发人员出现临时性加班的情况。以上这些局面会使业务部分产生依赖情绪，不提前考虑需求提交时间，不善于设置需求的优先级，从而进一步造成开发紧缺。

2.6　需求变化控制及跟踪的应用

在软件开发项目的应用中，以软件任务书为依据，软件工程组制订软件开发计划和需求规格说明。在各阶段工作产品（软件需求规格说明、软件设计文档、源代码、测试说明等）开发过程中，填写与相邻工作产品的“追溯表”，分步进行顺向追踪，这是静态的需求跟踪。但在实际软件开发中，需求更改引起的后续更改，以及设计编码过程中的变化可能引起的需求变化是需求管理中的难点，为了更好地跟踪，采用需求管理活动的动态跟踪方式。一般采取以下三种方法。

（1）将软件任务状态分为“已评审”“已分析”“已设计”“已实现”“已验证”和“已完成”这六种，按照项目进展进行跟踪和记录，掌握进展和变化情况，实现控制项目范围变化的目的，如表 2-3 所示。

表 2-3　需求管理活动跟踪表示例

填写人	张三		填写时间		××××年××月××日		
软件任务状态记录							
任务标识/所在章节	任务名称	已评审	已分析	已设计	已实现	已验证	已完成
rw1/4.1.1	信息输入输出	√	√	√			
rw2/4.1.2	××控制任务	√	√				
rw3/4.1.3	人机交互任务	√	√				
……							
软件任务状态统计/%		100	100	20	0	0	0

（2）在进行需求跟踪时，出现涉及需求变化的情况，除办理相应变更手续外，对相关的工作产品和软件任务书实施跟踪，详细记录每个变化的跟踪情况。如表 2-4 所示的需求跟踪记录。

表 2-4 需求跟踪记录示例

当前阶段/任务		软件设计阶段		跟踪人		×××	
跟踪起始时间		××××年××月××日		跟踪结束时间		××××年××月××日	
序号	软件更改简述	软件阶段	文档条款	前阶段条款	后阶段条款	相关更改	备注
1	概要设计中缺少从文件中读取××数据的设计说明	概要设计	5.3.2	1.1.2，2.3.1	3.3.1，Init	前阶段不涉及。后阶段做相应更改	
2							
……							

跟踪描述：

发现问题，概要设计中缺少从文件中读取××数据的设计说明。对应需求规格说明 2.3.1 节“××数据(rx_info_safedata)输入”一节。

设计人员已在概要设计文档中补充了相应内容，见 5.3.2 节(csc_init_ankong××数据读取)及第 5 章。

(3) 定期对需求变化和跟踪情况进行统计和汇总分析，这里采用的一般是项目技术状态管理的方式，也可以将表 2-4 略作改进，在表中增加对相关变化次数的统计，就可将变化的总体情况一览无余。

2.7 本章小结

本章从软件需求的层次和要求入手，介绍了软件需求工程的概念，并分别从需求开发和需求管理两个方面进行了详细拓展。最后给出了软件需求管理中常见的问题，以及一份需求变化控制和跟踪的应用实例。

CHAPTER 3 第3章

同行评审

提高软件质量如果只依赖软件测试人员在软件产品完成后进行是不科学的。一方面，软件测试在软件研发的末端，发现问题较晚，解决问题所花费的代价也较高。特别是当软件缺陷涉及软件需求、设计时，可能导致整个项目的失败。另一方面，软件测试的覆盖性往往受到时间进度、经费等的限制，不可能无限制地做下去，因此有些潜在的缺陷很难被发现。

同行评审是一种重要、有效的验证方法，它是通过审查、结构化走查等方式对项目开发的工作产品进行的验证。同行评审是一种软件开发人员主动开展的软件质量保证形式，是被许多软件研制专业机构认可的提高软件质量的最佳实践。适时地开展同行评审可以有效地避免在软件研制各活动中引入上一阶段的缺陷，尽可能地减少缺陷的放大，软件缺陷放大示意图如图 3-1 所示。

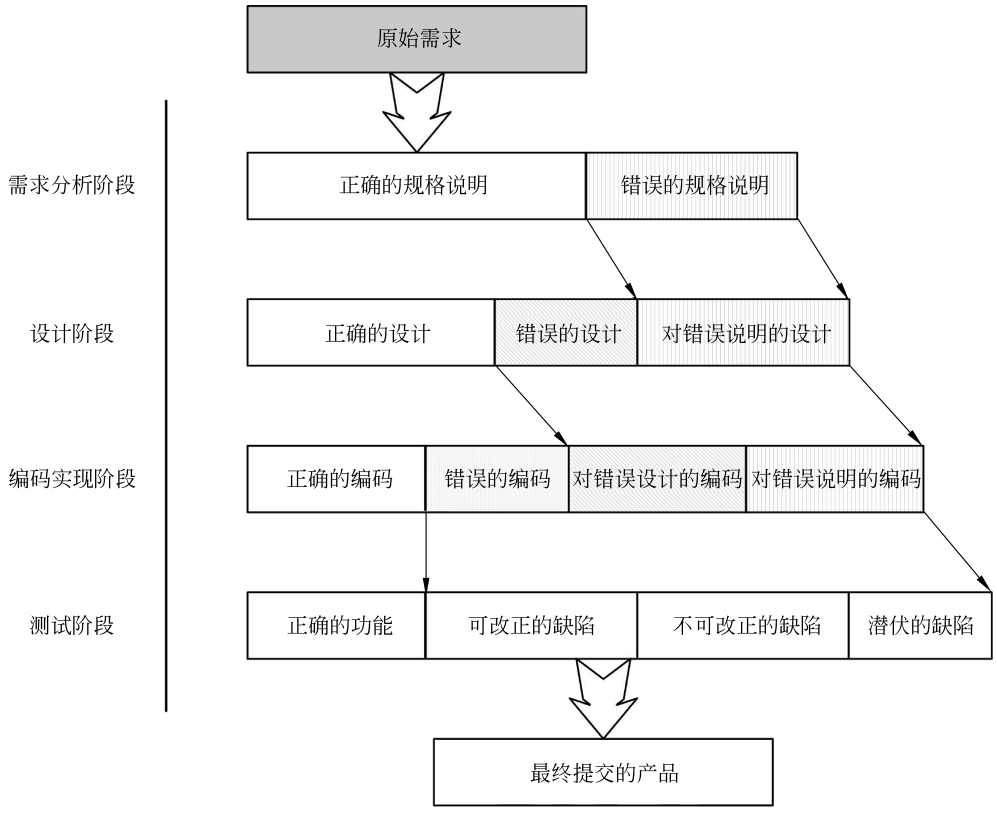


图 3-1　研制活动中缺陷放大示意图

同行评审的主要目的包括：

(1) 发现软件的功能、逻辑或实现上的错误；

(2) 验证软件是否满足其需求、设计等；

(3) 保证软件的实现符合预先指定的标准；

(4) 获得以统一方式开发的软件；

(5) 使项目更易于管理。

另外，同行评审还提供了培训机会，使刚从事软件研发工作的人员能够了解软件分析、设计和实现的不同方法。通过同行评审使软件研制人员对软件系统中不熟悉的部分有更为深入的了解，因此，同行评审发挥了培训后备人员和促进项目连续性的作用。

同行评审的活动主要包括：同行评审的策划、实施同行评审和分析同行评审数据。

3.1 同行评审的方式和对象

3.1.1 同行评审的方式

同行评审在实践过程中通常包括正式评审、审查和代码走查。对研制过程中编写的文档，一般采用审查或正式评审的方式进行同行评审；对软件代码一般采用代码走查的方式进行同行评审。

(1) 正式评审，通常是由经过同行评审培训的领域专家组成的同行评审组，在工作产品完成后对其进行的正式评审。同行评审组规模一般在 5～7 人为宜。正式评审的目的在于定位并除去工作产品中的缺陷。

(2) 审查，通常是由开发团队内部或研制单位内部的领域专家，对工作产品或工作产品中的部分内容进行的审查。可以是技术审查，也可以是检查工作产品与规程、模板、计划、标准的符合性或者变更是否被正确地执行。审查的目的在于在开发团队内部对工作产品提出改进意见。

(3) 代码走查，一般由 2～3 名经验丰富的开发团队成员，可以以小型会议的形式，对代码共同进行阅读，同时，可用代码审查和静态分析工具等对代码进行审查和分析。主要是审查代码是否正确地实现了软件设计，代码编写质量及编程规则的遵循情况等。对代码规范的审查建议在代码编写的早期进行。

三种同行评审方式的比较，如表 3-1 所示。

表 3-1 三种同行评审方式的比较

内 容	方 式		
	正式评审	审 查	代码走查
对象	对开发活动或软件产品质量有重要影响的软件文档	对开发活动或软件产品质量影响较为轻微的软件文档	关键重要的软件代码
时机	软件文档全部完成	软件文档完成关键重要章节	完成首个模块或关键重要模块的编写

内　　容	方　　式		
	正 式 评 审	审　　查	代 码 走 查
人员规模/人	5～7	3～5	2～3
主持人	开发团队所在部门技术负责人	开发团队负责人	代码编写人
验证	开发团队负责人	指定的开发团队成员	指定的开发团队成员
输出	问题清单 同行评审测量数据	问题清单 同行评审测量数据	走查报告 缺陷清单 同行评审测量数据

3.1.2　同行评审的对象

同行评审的对象包括在软件开发过程中产生的文档、代码等。具体工作产品见表 3-2。

表 3-2　软件开发过程中的同行评审

序号	工 作 产 品	评 审 方 式
1	系统总体设计方案	正式评审
2	软件需求规格说明	正式评审
3	软件接口需求规格说明	正式评审
4	软件接口设计说明	正式评审
5	软件概要设计说明	正式评审
6	软件详细设计说明	正式评审
7	数据库设计说明	正式评审
8	软件源代码	走查
9	测试计划	正式评审
10	测试说明	正式评审
11	测试记录	正式评审
12	问题报告	正式评审
13	测试报告	正式评审
14	版本说明	审查
15	计算机系统操作员手册	审查
16	软件用户手册	正式评审
17	软件程序员手册	审查

续表

序号	工作产品	评审方式
18	固件保障手册	审查
19	软件产品规格说明	审查

3.2 策划同行评审

同行评审的策划和准备活动通常包括确定参与同行评审的人员、准备同行评审要用的材料，以及安排同行评审日程。同行评审的人员包括工作产品的开发人员和必须参与同行评审的相关人员，同行评审的材料包括工作产品、检查单和评审准则。策划同行评审的活动如下。

1. 确定评审对象与评审方式。软件研制过中的工作产品均应进行同行评审。在软件开发过程中需要评审的工作产品和可进行的评审方式可以参考表 3-2 确定。

一般情况下，同行评审应遵循以下原则：

（1）评审组一般应该由适当的同行专家组成，人数参照表 3-1，按照评审方式的不同确定；

（2）在召开评审会之前应进行准备，每个参会人员预先准备的时间不应超过 2 小时；

（3）评审会的时间一般不超过 2 小时。

制定上述原则的原因是同行评审的范围越小，发现问题的可能性就越大。例如，代码走查时每次只关注部分模块的实现是否按照设计进行。

2. 确定同行评审需采集的信息。对同行评审活动需要采集的信息包括：评审准备工作量、评审所用的工作量、修改评审问题所需的工作量、工作产品的规模、缺陷数量、缺陷类型和严重程度等。

3. 建立并维护同行评审的入口准则和出口准则。为了提高同行评审的效率，需要针对具体的工作产品制定评审的入口准则和出口准则。

在准备评审时应根据入口准则进行检查，这些检查可以以检查单的形式体现。例如，根据需求跟踪矩阵确定是否覆盖所有需求，文档的格式是否符合要求，定义和描述是否有二义性等。一般情况下检查的人员应是项目负责人或项目负责人指定的相关人员。

同样在评审完成时，应根据评审准备时制定的评审出口准则进行检查，也就是需要确定什么样的条件下可以通过同行评审，何种条件下需要安排另一次的评审。例如，通过评审的条件可以明确为工作产品不能存在严重级别的缺陷，工作产品必须覆盖上一阶段的功能、性能、接口等要求。

4. 建立评审审查单。在评审前应根据评审的工作产品的特点制定评审审查单。同行评审审查单格式模板如表 3-3 所示。

表 3-3　同行评审审查单格式模板

同行评审审查单	项目名称	
	产品名称	

序号	审查项	满足	不满足	不适用
1				
2				
3				

审查结论：　□合格　□不合格
审查人员签字：　年　月　日

问题记录(可附页)

应根据评审对象的特点制定相应的审查单,代码走查时应制定代码审查规范,也可以选用工具进行代码编码规范的审查,以及代码设计规范方面的分析。

对于文档类审查单的制定,应根据不同类型的软件文档制定不同的审查单。具体审查内容如下。

1) 系统总体设计方案

(1) 总体概述了系统(或项目)的建设背景或改造背景,概述了系统的主要用途;

(2) 引用文件完整准确,包括引用文档(文件)的文档号、标题、编写单位(或作者)和日期等;

(3) 确切地给出所有在本文档中出现的专用术语和缩略语的定义;

(4) 完整清晰描述软件系统的功能需求;

(5) 完整清晰描述软件系统的性能需求;

(6) 完整清晰描述软件系统的外部接口需求;

(7) 完整清晰描述软件系统的适应性需求;

(8) 完整清晰描述软件系统的安全性需求;

(9) 完整清晰描述软件系统的操作需求;

(10) 完整清晰描述软件系统的可靠性需求;

(11) 清晰描述软件系统的运行环境;

(12) 描述了系统的生产和部署阶段所需要的支持环境;

(13) 以配置项为单位(包括软件配置项或/和硬件配置项)设计了软件系统体系结构或系统体系结构;

(14) 软件系统的体系结构合理、可行;

(15) 用名称和项目唯一标识号标识每个 CSCI;

(16) 清晰、合理地为各个软件配置项分配了功能、性能；

(17) 翔实设计各个软件配置项与其他配置项(包括软件配置项、硬件配置项、固件配置项)之间的接口；

(18) 进行了软件系统危险分析，合理确定软件配置项关键等级；

(19) 合理分配了与每个 CSCI 相关的处理资源；

(20) 追踪关系完整、清晰；

(21) 文档编写规范、内容完整、描述准确、一致。

2) 软件系统测试计划

(1) 完整、清晰地描述了引用文件，包括引用文档(文件)的文档号、标题、编写单位(或作者)和日期等；

(2) 测试组织独立、人员组成合理、分工明确；

(3) 测试环境适应测试任务的需求；

(4) 测试资源满足测试任务的需求；

(5) 提出软件系统的每个功能应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖的要求；

(6) 功能测试项划分合理、充分，覆盖了软件系统设计说明定义的所有功能；

(7) 性能测试项划分合理、充分，覆盖了软件系统设计说明提出的所有性能指标；

(8) 接口测试项划分合理、充分；覆盖了软件系统设计说明定义的所有外部接口，包括软件配置项之间、软件系统和硬件之间的所有接口；

(9) 对于每一个接口，提出正常输入和异常输入的测试要求；

(10) 每一测试项的测试要求明确、测试方法合理，详细说明了完成本测试项所需要的测试数据生成方法和注入方法，说明测试结果捕获方法及分析方法等；

(11) 清晰建立测试项与测试依据之间的双向追踪关系；

(12) 提出时限测试要求(测试程序在有时限要求时完成特定功能所需的时间)；

(13) 提出系统各部分之间协调性的测试要求；

(14) 提出系统依赖运行环境程度的测试要求(测试软、硬件环境对系统性能的影响等)；

(15) 提出系统处理容量的测试要求；

(16) 提出系统负载能力的测试要求；

(17) 提出系统运行占用资源情况的测试要求；

(18) 提出边界测试要求；

(19) 对于 A、B 级软件，提出安全性测试的要求；

(20) 文档编制规范、内容完整、描述准确一致。

3) 软件需求规格说明

(1) 完整、清晰地描述了引用文件，包括引用文档(文件)的文档号、标题、编写单位(或作者)和日期等；

(2) 确切给出了所有在本文档中出现的专用术语和缩略语定义；

(3) 以 CSCI 为单位，进行软件需求分析；

(4) 采用了适合的软件需求分析方法；

(5) 总体概述了每个 CSCI 应满足的功能需求和接口关系；

(6) 完整、清晰、详细地描述由待开发软件实现的全部外部接口(包括接口的名称、标识、特性、通信协议、传递的信息、流量、时序,等等);

(7) 完整、清晰、详细地描述由待开发软件实现的功能,包括业务规则、处理流程、数学模型、容错处理要求、异常处理要求等专业应用领域的全部要求;

(8) 分别描述各个 CSCI 的性能需求;

(9) 明确提出软件的安全性、可靠性、易用性、可移植性、维护性需求等其他要求;

(10) 用名称和项目唯一标识号标识每个内部接口,描述在该接口上将要传递的信息的摘要;

(11) 用名称和项目唯一标识号标识 CSCI 的数据元素,说明数据元素的测量单位、极限值/值域、精度、分辨率、来源/目的(对外部接口的数据元素,可引用详细描述该接口的接口需求规格说明或相关文档);

(12) 指明各个 CSCI 的设计约束;

(13) 详细说明在将开发完成了的 CSCI 安装到目标系统上时,为使其适应现场独特的条件和(或)系统环境的改变而提出的各种需求;

(14) 描述运行环境要求,包括运行软件所需要的设备能力、软件运行所需要的支持软件环境;

(15) 详细说明用于审查 CSCI 满足需求的方法,标识和描述专门用于合格性审查的工具、技术、过程、设施和验收限制等;

(16) 详细说明要交付的 CSCI 介质的类型和特性;

(17) 描述 CSCI 维护保障需求;

(18) 描述本文档中的工程需求与“软件系统设计说明”和(或)“软件研制任务书”中的 CSCI 的需求的双向追踪关系;

(19) 文档编制规范、内容完整、描述准确一致。

4) 软件接口需求规格说明

(1) 概述本文档所适用的系统,标识和描述各个接口在系统中的作用;

(2) 列出本文档引用的所有文件;

(3) 提供一个或多个接口示意图,描述和标识各 CSCI、HWCI 和本文档适用的各关键项之间的连接关系和接口;对每个接口应标识其名称和项目唯一标识号;

(4) 详细说明对接口的需求,应规定与各 CSCI 的联接是并发执行还是顺序执行,说明接口使用的通信协议及接口的优先级别;

(5) 对于并发的 CSCI,应规定内部使用的同步方法;

(6) 清晰描述每个接口的数据要求,对每个通过接口的数据元素,应详细说明数据元素的项目唯一标识号、简要描述、来源/用户、度量单位、极限值/值域(若是常数,提供实际值)、精度或分辨率等;

(7) 文档编制规范、内容完整、描述准确一致。

5) 软件配置项测试计划

(1) 测试组织独立、人员组成合理、分工明确;

(2) 测试资源满足测试任务的需求;

(3) 测试环境及其测试环境的安装、验证和控制计划合理可行,满足测试任务的需求;

(4) 提出软件配置项的每个特性应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖的要求;

(5) 功能测试项划分合理、充分,覆盖了软件需求规格说明定义的所有功能;

(6) 性能测试项划分合理、充分,覆盖了软件需求规格说明提出的所有性能指标;

(7) 接口测试项划分合理、充分,覆盖了软件需求规格说明定义的所有软件配置项之间、软件配置项和硬件之间的接口;

(8) 每一测试项的测试要求明确、测试方法合理,详细说明了完成本项测试所需要的测试数据生成方法和注入方法,说明测试结果捕获方法及分析方法等;

(9) 对于每一个外部接口,提出正常输入和异常输入的测试要求;

(10) 提出计算精度测试要求(测试程序在获得定量结果时程序计算的精确性);

(11) 提出时限测试要求(测试程序在有时限要求时完成特定功能所需的时间);

(12) 提出配置项各部分之间协调性的测试要求;

(13) 配置项依赖运行环境的程度的测试要求(测试软、硬件环境对系统性能的影响等);

(14) 提出配置项处理容量的测试要求;

(15) 提出配置项负载能力的测试要求;

(16) 提出配置项运行占用资源情况的测试要求;

(17) 提出边界测试要求;

(18) 对 A、B 级软件配置项,提出了安全性测试的要求;

(19) 对 A、B 级嵌入式软件配置项,提出了目标码覆盖的测试要求;

(20) 建立了测试项与测试依据之间的双向追踪关系清晰;

(21) 文档编制规范、内容完整、描述准确一致。

6) 软件概要设计说明(结构化)

(1) 概述了 CSCI 在系统中的作用,描述了 CSCI 和系统中其他配置项的相互关系;

(2) 以 CSC 为实体进行了软件体系结构的设计;

(3) 软件体系结构合理、优化、稳健;

(4) 应对 CSC 之间的接口进行设计,用名称和项目唯一标识号标识每一个接口,并对与接口相关的数据元素、消息、优先级、通信协议等进行描述;

(5) 为每个接口的数据元素建立数据元素表,说明数据元素的名称和唯一标识号、简要描述、来源/用户、测量单位、极限值/值域(若是常数,提供实际值)、精度或分辨率、计算或更新的频率或周期、数据元素执行的合法性检查、数据类型、数据表示/格式、数据元素的优先级等;

(6) 规定每一个接口的优先级和通过该接口传递的每个消息的相对优先次序;

(7) 描述接口通信协议,分小节给出协议的名称和通信规格细节,包括消息格式、错误控制和恢复过程、同步、流控制、数据传输率、周期还是非周期传送,以及两次传输之间的最小时间间隔、路由/地址和命名约定、发送服务、状态/标识/通知单和其他报告特征及安全保密等;

(8) CSC 内存和处理时间分配合理(仅适用于“嵌入式软件”或“固件”);

(9) 描述 CSCI 中各 CSC 的设计,将软件需求规格说明中定义的功能、性能等全部都分配到具体的软件部件,必要时,还应说明安全性分析和设计并标识关键模块的等级;

(10) 用名称和项目唯一标识号标识 CSCI 中的全局数据结构和数据元素，建立数据元素表；

(11) 用名称和项目唯一标识号标识被多个 CSC 或 CSU 共享的 CSCI 数据文件，描述数据文件的用途、文件的结构、文件的访问方法等；

(12) 建立软件设计与软件需求的追踪表；

(13) 文档编写规范、内容完整、描述准确一致。

7) 软件概要设计说明(面向对象)

(1) 概述了 CSCI 在系统中的作用，描述了 CSCI 和系统中其他配置项的相互关系；

(2) 以包或类的方式在软件体系结构范围内进行逻辑层次分解，将软件需求规格说明中定义的功能、性能等全部进行分配，分解的粒度合理，相关说明清晰；

(3) 采用逻辑分解的元素描述有体系结构意义的用况，使体系结构设计与用况需求之间有紧密的关联；

(4) 描述了系统的动态特征，对进程/重要线程的功能、生命周期和进程间的同步与协作有明确的说明；

(5) 软件体系结构合理、优化、稳健；

(6) 对每个标识的接口都设计有相应的接口类/包，规定每一个接口的优先级和通过该接口传递的每个消息的相对优先次序；

(7) 描述接口和数据元素的来源/用户、测量单位、极限值/值域(若是常数，提供实际值)、精度或分辨率、计算或更新的频率或周期、数据元素执行的合法性检查、数据类型、数据表示/格式、数据元素的优先级等；

(8) 进行安全性分析和设计并标识关键模块的等级；

(9) 为完成需求的功能增加必要的包/类，使层次分解的结果是一个完整的设计；

(10) 实现视图描述 CSCI 的实现组成，每个构件分配了合适的需求功能，构件的表现形式(exe、dll 或 ocx 等)合理；

(11) 部署视图描述 CSCI 的安装运行情况，能够对未来的运行景象形成明确概念；

(12) 建立软件设计与软件需求的追踪表；

(13) 采用的 UML 图形或其他图形描述正确、详略适当，有必要的文字说明；

(14) 文档编写规范、内容完整、描述准确一致。

8) 数据库设计说明

(1) 进行数据库系统概念、逻辑、物理设计；

(2) 数据的逻辑结构满足完备性要求；

(3) 数据的逻辑结构满足一致性要求；

(4) 数据的冗余度合理；

(5) 数据库的备份与恢复设计合理、有效；

(6) 数据存取控制满足数据的安全保密性要求；

(7) 数据存取时间满足实时性要求；

(8) 网络、通信设计合理、有效；

(9) 审计、控制设计合理；

(10) 视图设计、报表设计满足要求；

(11) 文件的组织方式和存取方法合理有效；

(12) 数据的群集安排合理、有效；

(13) 数据在存储介质上的分配合理有效；

(14) 数据的压缩与分块合理有效；

(15) 缓冲区的大小和管理满足要求；

(16) 对数据库访问和操作的软件单元设计合理、描述完整；

(17) 正确提供本文档所涉及的数据库或 CSU 到系统或 CSCI 需求的双向追踪；

(18) 文档编写规范、内容完整、描述准确一致。

9) 接口设计说明

(1) 概述接口所在系统，标识和描述本文档适用的各个接口在该系统中的作用；

(2) 准确给出所有在本文档中出现的专用术语和缩略语的确切定义；

(3) 采用接口示意图描述和标识各 CSCI、HWCI 和本文档适用的各关键项之间的连接关系及接口，对每个接口应标识其名称和项目唯一标识号；

(4) 对每个接口进行设计，包括接口的数据元素、消息、优先级别、通信协议及同步机制；

(5) 对每个通过接口的数据元素，建立数据元素表，表中应为数据元素提供下列信息：数据元素的项目唯一标识号、简短描述、来源/用户、度量单位、极限值/值域(若是常数，提供实际值)、精度或分辨率、计算或更新频率/周期、数据类型、数据表示法和格式、优先级等，以及对数据元素执行的合法性检查；

(6) 应用名称和项目唯一标识号标识接口间的每个消息，描述数据元素对各个消息的功用，并提供每个消息与组成该消息的各数据元素间的交叉引用，而且还应提供每个数据元素与各数据元素间的交叉引用；

(7) 应规定接口优先级和通过该接口传递的每个消息的相对优先次序；

(8) 对每个接口，应描述与该接口关联的商用、军用或专用的通信协议，对协议描述应包括：消息格式、错误控制和恢复过程、同步、流控制、数据传输机制、路由/编址和命名约定、发送服务、状态、标识、通知单和其他报告特征；

(9) 安全保密等；

(10) 文档编写规范、内容完整、描述准确一致。

10) 软件详细设计说明(结构化)

(1) 将软件部件分解为软件单元；

(2) 对每个软件单元规定了程序设计语言所对应的处理流程；

(3) 对每个单元的入口、出口给予清晰完整的设计；

(4) 对于结构化设计，可采用数据流图、控制流图清晰描述软件单元之间的关系；

(5) 每个 CSU 的详细设计信息应包括：输入/输出数据元素、局部数据元素、中断和信号、程序算法、错误处理、数据转换、逻辑流程图、数据结构、局部数据文件和数据库、限制和约束等；

(6) 将包分解到类，用类图、顺序图、活动图或文字等多种方式进行描述；

(7) 对类的名称、属性、操作、动态特性、静态特性等进行说明；

(8) 准确说明类的纵向、横向关系；

(9) 说明类的数据成员,包括量化单位、值域、精度,若是常数,应提供其实际值;

(10) 说明类的操作,包括输入参数、输出参数、处理过程及算法,还应说明其异常处理机制;

(11) 说明类的动态特性,必要时可采用状态机或其他形式予以描述;

(12) 说明本 CSCI 需要用到的数据,包括配置数据设计、数据文件设计及数据库设计;

(13) 准确描述软件详细设计与概要设计的追踪关系;

(14) 文档编写规范、内容完整、描述准确一致。

11) 软件详细设计说明(面向对象)

(1) 将包最终分解到类,并用类图、时序图、活动图或文字等合适的方式进行描述;

(2) 对相关类的组合采用类族方式命名或采用设计模式命名,说明类组合的功能、特征等;

(3) 对每个类说明其类型、功能、在软件结构中的位置;

(4) 准确说明类的纵向、横向关系;

(5) 说明类的每一个属性,每个属性的名称、用途、类型、可访问性、值域、精度和合法性检查等,若是常数,应提供其实际值;

(6) 说明类的每一个操作,包括名称、功能、输入、输出、处理过程及算法、异常处理机制等,并采用适当的文字或图进行说明;

(7) 对数据文件或数据库的包装类,说明类的静态特性,描述数据元素和类属性字段的对应关系;

(8) 对于有状态变化的类,说明类的动态特性,必要时可采用状态机或其他形式予以描述;

(9) 说明本 CSCI 需要用到的数据,包括配置数据设计、数据文件设计及数据库设计;

(10) 准确描述软件详细设计与概要设计的追踪关系;

(11) 文档编写规范、内容完整、描述准确一致。

12) 软件单元测试计划

(1) 测试组织人员组成合理、分工明确;

(2) 将应该测试的所有软件单元全部明确地标识为被测对象;

(3) 测试资源满足单元测试任务的需求;

(4) 测试环境及其测试环境的安装、验证和控制计划合理可行,满足单元测试任务的需求;

(5) 提出软件单元的每个特性应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖的要求;

(6) 对每个被测单元提出圈复杂度(McCabe 复杂性度量值)的度量要求;

(7) 对每个软件单元的扇入、扇出数提出分析和统计要求;

(8) 对软件单元源代码注释率(有效注释行与源代码总行的比率)提出分析检查要求;

(9) 对软件可靠性、安全性设计准则和编程准则提出检查要求 (要求 A、B 级软件应落实全部强制类编程准则);

(10) 对源代码与软件设计文档一致性的分析、检查要求;

(11) 对有特殊要求的软件单元,进行特殊测试,如占用空间、运行时间、计算精度等测

试要求；

（12）对于重要的执行路径，提出路径测试要求；

（13）提出边界测试要求；

（14）提出单元调用关系100%的覆盖要求；

（15）提出语句覆盖率要求（A、B级软件应达到100%的要求）；

（16）提出软件测试分支覆盖率要求（A、B级软件应达到100%的要求）；

（17）对用高级语言编制的A、B级软件，提出修正的条件判定覆盖（MC/DC）覆盖要求（“921”工程要求达到100%）；

（18）对于用高级语言编制的A、B级嵌入式软件，提出测试目标码覆盖率要求（“921”工程要求语句覆盖率达到100%、分支覆盖率达到100%）；

（19）明确提出单元测试的终止条件；

（20）给出单元测试项（条目）到详细设计之间追踪关系；

（21）文档编制规范、内容完整、描述准确一致。

13）软件测试说明

（1）测试用例设计遵循对应的测试计划；

（2）给出与测试活动有关的进度安排，包括测试准备、测试执行、测试结果整理与分析等；

（3）描述测试所需硬件环境的准备过程；

（4）描述测试所需软件环境的准备过程；

（5）逐项审查测试所需的硬件环境和软件环境的就绪状况，如操作系统、测试工具、测试软件、测试数据等；

（6）测试用例设计应覆盖软件测试计划中标识的每个测试项；

（7）保证每个测试项应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖；

（8）对每个测试用例，应详细描述下列内容：

① 测试用例名称和项目唯一标识、测试用例综述、测试用例追踪、测试用例初始化、测试步骤、测试输入与操作、期望测试结果、测试结果评判标准、测试终止条件、前提和约束条件、测试用例设计方法等；

② 测试用例描述表清晰、规范、易理解；

③ 测试用例内容描述准确，与测试计划的相关描述保持一致；

④ 建立测试用例到测试项（条目）的追踪表；

⑤ 文档编写规范、内容完整、描述准确一致。

14）软件测试报告

（1）对测试过程进行了描述；

（2）测试报告中应说明被测软件的版本；

（3）测试报告中应说明测试时间、测试人员、测试地点、测试环境等；

（4）测试报告中应翔实、清晰地说明设计的测试用例数量和实际执行的测试用例数量、

部分执行的数量、未执行的数量；

(5) 对于每个执行的测试用例还应该说明执行结果(通过、未通过)；

(6) 对于未执行和部分执行的测试用例，应说明原因；

(7) 执行过程中如果增加了新的测试用例，则测试报告中应说明新增加的测试用例；

(8) 测试报告中应统计所有测试用例的测试结果，包括用例名称、执行状态、执行结果、出现问题的步骤及问题标识等；

(9) 对每个被测对象(被测软件)的质量分别进行客观评估；

(10) 文档编写规范、内容完整、描述准确一致。

15) 软件测试记录

(1) 每个测试用例的测试记录应包括测试用例名称与标识、测试综述、用例初始化、测试时间、前提和约束、测试用例终止条件等基本信息；

(2) 测试输入、期望测试结果、评估测试结果的标准等应与“软件测试说明”中的相关描述保持一致；

(3) 应记录每个测试步骤的实测结果，当有量值要求时，应准确记录具体的实际测试量值，如果实际测试结果已经存储在文件中，可以只记录文件名；

(4) 测试记录应详细描述实际测试时的操作步骤；

(5) 对于完整执行过的测试用例，应明确给出测试用例的执行结果(通过、未通过)；

(6) 如果在测试中发现软件有问题，除记录实测结果外，还应详细填写“软件问题报告单”；

(7) 对未执行或未完整执行的测试用例，应逐个说明原因；

(8) 测试人员签署测试记录；

(9) 文档编写规范、内容完整、描述准确一致。

16) 软件问题报告

(1) 应详细说明发现的每一个问题，并形成问题报告单；

(2) 问题单对于软件问题的描述明确、清晰；

(3) 合理划分问题类别；

(4) 合理定义问题级别；

(5) 清晰地建立问题的追踪关系，即问题的来源；

(6) 文档编写规范、内容完整、描述准确一致。

17) 计算机系统操作员手册

(1) 概述本文档所适用的系统和软件的用途；

(2) 给出所有在本文档中出现的专用术语和缩略语的确切定义；

(3) 正确列出计算机系统的各个操作指令；

(4) 正确列出操作计算机系统的前提条件，包括通电和断电、启动、关机等步骤；

(5) 正确描述计算机系统的操作过程，包括输入和输出过程、监督过程、恢复过程、脱机程序过程及报警、切换等其他过程。如果采用了一种以上的操作方式，那么对每种方式的操作命令都应进行阐述；

(6) 正确描述计算机系统的诊断过程,应说明为执行每一诊断过程所需的硬件、软件或固件,执行诊断过程所需的每一步指令及诊断消息和采取的相应动作;

(7) 正确描述诊断工具集,用名称和唯一标识号来标识每一个诊断工具,并叙述该工具和它的应用;

(8) 文档编写规范、内容完整、描述准确一致。

18) 软件用户手册

(1) 正确给出所有在本文档中出现的专用术语和缩略语的确切定义;

(2) 准确描述软件安装过程,完整列出安装的有关媒体情况及使用方法;

(3) 准确描述软件的各功能及操作说明,包括初始化、用户输入、输出、终止等信息;

(4) 准确标识软件的所有出错告警信息、每个出错告警信息的含义和出现该错误告警信息时应采取的恢复动作等;

(5) 文档编写规范、内容完整、描述准确一致。

19) 软件程序员手册

(1) 正确给出所有在本文档中出现的专用术语和缩略语的确切定义;

(2) 概述本文档所适用的系统和 CSCI 的用途;

(3) 正确描述宿主机和目标机的操作系统,以及包括编辑、编译、链接等实用程序在内的其他软件;

(4) 准确描述软件的编程特征;

(5) 准确描述软件的编程语言;

(6) 准确描述软件的输入输出控制的编程;

(7) 准确描述软件的附加或专用编程技术;

(8) 提供编程示例;

(9) 准确描述软件的错误检测和诊断功能;

(10) 文档编写规范、内容完整、描述准确一致。

20) 固件保障手册

(1) 准确描述固件设备说明;

(2) 准确描述固件设备的安装修理过程;

(3) 准确描述适用于设备、支持硬件和软件的任何安全保密性;

(4) 准确描述所有固件的操作限制和环境限制;

(5) 准确描述所有固件的编程设备及其过程;

(6) 准确描述由设备提供商提供的供方信息、为设备和软件提供技术保障的所有商用信息;

(7) 文档编写规范、内容完整、描述准确一致。

21) 版本说明

(1) 正确列出所发行产品的文档清单;

(2) 正确列出所发行产品的文件清单;

(3) 正确列出所发行产品的更动清单;

(4) 正确列出所发行产品的修改数据；

(5) 正确说明新版本与旧版本产品之间的接口兼容性；

(6) 正确列出所发行产品的引用文档清单，若有更动，还应说明该清单的更动部分；

(7) 清楚说明更动对旧版本的影响；

(8) 提供产品的安装说明；

(9) 说明所发行产品的可能问题和已知错误及其解决办法；

(10) 文档编写规范、内容完整、描述准确一致。

22) 软件产品规格说明

(1) 准确列出所有在本文档中出现的专用术语和缩略语的确切定义；

(2) 准确提供产品所包含的所有设计文档；

(3) 准确提供产品的源代码列表；

(4) 建立源代码列表与计算机软件部件和单元的索引关系；

(5) 规定编译源代码的编译程序和链接程序；

(6) 规定在交付时产品所用的测量资源；

(7) 文档编写规范、内容完整、描述准确一致。

5. 制订同行评审计划。项目负责人根据项目计划中确定的同行评审要求制订详细的同行评审计划，并将评审计划、待评审的工作产品及相关的评审审查单一同发给评审人员，详细的同行评审计划应包括会议名称、会议时间、地点、评审人员、评审分工、评审审查表及评审通过的准则等。

3.3　实施同行评审

同行评审人员应根据同行评审计划开展同行评审工作。同行评审的有效性的重点在于同行评审人员是否认真履行职责，是否在召开评审会之前有充分的时间对工作产品进行了审查。实施同行评审的主要活动如下。

(1) 预先审查。在召开评审会前，评审人员应该根据分工，对照审查单审查工作产品。记录审查过程中发现的问题。

(2) 召开评审会。项目负责人组织召开评审会，评审人员说明预先审查中发现的问题，评审组与项目相关人员经过讨论形成评审报告。评审报告的内容包括评审意见和问题说明。评审意见和问题记录的表格模板如表 3-4 和表 3-5 所示。

(3) 采集同行评审数据。为了不断提高同行评审的有效性，需要对同行评审数据进行采集。需要采集的数据包括：工作产品名称、规模、所处开发阶段，同行评审组的构成、同行评审类型、评审人员预先审查时间、评审会所用时间、所发现的缺陷数、缺陷类型及其来源，等等。

(4) 同行评审问题的处理与跟踪。项目组应针对同行评审中提出的问题制定相应的初步处理措施，开发团队负责人或制定人员应对这些问题的处理进行跟踪。

表 3-4　评审意见表

评审意见表		项目名称	
		阶段名称	
评审性质	□评审　□复审	评审方式	□内部　□外部
评审日期		评审地点	
组织单位		评审委员会组成	（用附件的形式记录）
审查对象与审查结果	名称		版本
评审结论	□通过　（无问题） □通过　（有_____个问题，“评审问题记录表”附后） □不通过（有_____个问题，“评审问题记录表”附后）		
其他说明			

评审组组长（签字）：

年　　月　　日

表 3-5　评审问题记录表

评审意见表		项目名称	
		阶段名称	
评审性质	□评审　□复审	评审方式	□内部　□外部
评审日期		评审地点	
序号	问题描述		初步处理意见

评审组组长（签字）：

年　　月　　日

3.4　同行评审的数据分析

3.4.1　采集和分析的数据

分析同行评审数据是不断提高同行评审有效性的重要环节，也是不断提高软件产品质量的重要手段。因此，需要对同行评审的数据进行分析，并与研制单位的期望进行比较。另外，还应更深入地分析与问题有关的事件，例如，造成缺陷的原因、缺陷方案的影响等，以便采取切实有效的措施避免同类问题的发生。采集和分析的数据包括：

(1) 文档页数和代码行数；

(2) 工作产品问题或缺陷数；

(3) 工作产品的问题或缺陷密度；

(4) 同行评审所用时间；

(5) 问题或缺陷等级；

(6) 问题或缺陷类型；

(7) 问题或缺陷的原因；

(8) 问题或缺陷注入阶段；

(9) 发现问题/缺陷数的效率。

3.4.2　同行评审的过程控制

根据 Watte Humphrey 于 1998 年提出的经验数据，设计阶段的同行评审工作量应该占到该阶段工作量的 1/3 或以上，代码走查工作量也要占到编码和单元测试阶段工作量的 1/3 以上。如果它们都只占到 15%，此时同行评审的质量系数只能达到 0.5。

同行评审的准则如下。

(1) 设计同行评审工作量应占设计阶段总工作量的 1/3 以上，其质量准则为

设计文档同行评审应该至少发现 3 个缺陷/页，遗漏缺陷密度控制在 0.5 个/页以下。

(2) 代码同行评审工作量应占实现阶段总工作量的 1/3 以上。

(3) 同行评审准备时间等于(或大于)开会时间。

(4) 同行评审期间发现的缺陷数量应该是同行评审准备期间发现的缺陷数量 2 倍以上。

(5) 同行评审发现缺陷的效率是测试发现缺陷的 3 倍。

3.4.3　建议的同行评审效率

如果在软件开发全过程中使用同行评审，建议同行评审的总工作量要占 10%的开发工作量。同行评审的效率：

(1) 每 20 页叙述性文档，需要 40 人·时；

(2) 每 12 页概要设计,需要 30 人·时;

(3) 每 1000 行代码,需要 55 人·时。

同行评审的效率取决于以下因素:

(1) 同行评审的准备情况;

(2) 参与同行评审人员的技能和经验。

3.4.4 同行评审覆盖率

在开发过程中,一般对同行评审有如下的覆盖率要求:

(1) 对需求的同行评审覆盖率要求 100%;

(2) 对设计的同行评审覆盖率要求 100%;

(3) 对确认测试的测试用例的同行评审覆盖率要求 100%;

(4) 对代码的同行评审覆盖率要求不少于 30%,新编代码的关键部位和关键算法要进行 100%的同行评审,非新编代码采取抽查方式,抽查比例建议不少于 25%。

3.5 评审常见问题

根据 Humphrey 的经验,审查不能发挥作用的原因大致如下。

(1) 最大的问题是进度紧张而且对管理重视不足,使得审查流于形式。管理层对进度的推动力远远大于对质量的关注程度,造成同行评审的形式大于内容,没有发挥其应有的作用。

(2) 同行评审数据未被充分使用。从以往项目中获取同行评审数据可以有效提高同行评审效率,避免发生类似问题。同行评审组织者应从历史经验库中获取以往相同工作产品的同行评审数据,以便有效地开展同行评审活动。

(3) 准备不足。

① 同行评审不足主要体现在项目计划中未对同行评审进行有效策划,不仅是单个项目的同行评审准备不足,甚至可能是整个研制单位内部对同行评审工作没有制定相关的规范,没有建立组织级过程。

② 工作产品质量太差。工作产品缺少自我检查,或因计划不合理,提交的工作产品质量太低,需要修改后再提交。

③ 参与同行评审的人员选择不恰当。人员问题可能有下列情形:

a. 由于计划组织不充分,评审资源没有得到保证,资深技术人员或者评审人员忙于其他工作,没有投入足够的时间;

b. 参与同行评审人员一般是领域专家而不是评审活动的专家,他们没有掌握进行同行评审的方法、技巧、过程等,因此需要对评审员进行培训。因此,在同行评审前先对评审员基于评审所需的知识和技能进行培训是很有必要的;

c. 参与人员太多或者参与人员不能胜任,或者有管理人员参与,导致无法开展有效的同行评审活动。

(4) 一次涵盖的内容太多。一次评审太多的内容,导致人员精力等无法长时间集中,发现问题的能力受到影响。建议要评审的对象内容需要有重点,一般按照“2/8”原则确定主要内容进行评审。

(5) 会议上过多地讨论问题如何改正。

同行评审的目的主要在于定位问题,一旦正确地确认了问题,对于大多问题都能很快找到解决方案,对于一时无法给出解决方案的问题可以在评审后研究讨论。因此,一般的同行评审会建议在一个问题上用时不超过 3 分钟;如果评审专家之间有不同意见,先进行记录。当评审专家讨论解决方案时,主持人可以要求他们在会后讨论。

3.6　本章小结

同行评审是由软件工作产品生产者的同行遵循已定义的规程对工作产品进行的技术评审。通过同行评审,有利于及早和高效地从软件工作产品中识别并消除缺陷,减少到产品发布时的缺陷。

同行评审的对象包括所有软件开发的中间工作产品和最终工作产品,同行评审方式分为正式评审、审查和代码走查三类,一般文档采用正式评审、审查的方式,代码采用代码走查的方式。文档的同行评审要对文档的完整性、一致性和正确性进行同行评审。代码走查要对代码的规范性和实现的正确性进行审查。组织应建立基本的同行评审准则,以便保证同行评审的有效实施。

同行评审实施过程中存在许多问题,开发团队应针对问题制订科学、合理的同行评审计划,在实施过程中及时分析同行评审数据,发现问题,并不断提高同行评审的能力。

第 4 章 CHAPTER 4

验　　证

4.1　概述

验证的目的是确保所选择的工作产品满足指定的需求。对工作产品的验证可以有效提高软件产品对客户需求、软件系统需求和软件配置项需求的满足程度。因此，验证贯穿软件开发的整个过程。

验证一般由测试人员和开发人员共同完成，主要包括同行评审、代码审查、静态分析、单元测试等。验证过程包括验证准备和验证实施。第 3 章对同行评审进行了详细说明，本章主要对代码审查、静态分析和单元测试进行详细介绍。

验证是验证工作产品是否恰当地反映了指定的需求。换句话说，验证确保“正确地构造了”工作产品。

本章通过对验证的一般要求进行说明，介绍了代码审查、静态分析、单元测试的实施方法。其中验证过程包括验证准备、验证实施两部分，是所有验证方法的共同要求。验证过程主要任务示意图如图 4-1 所示。

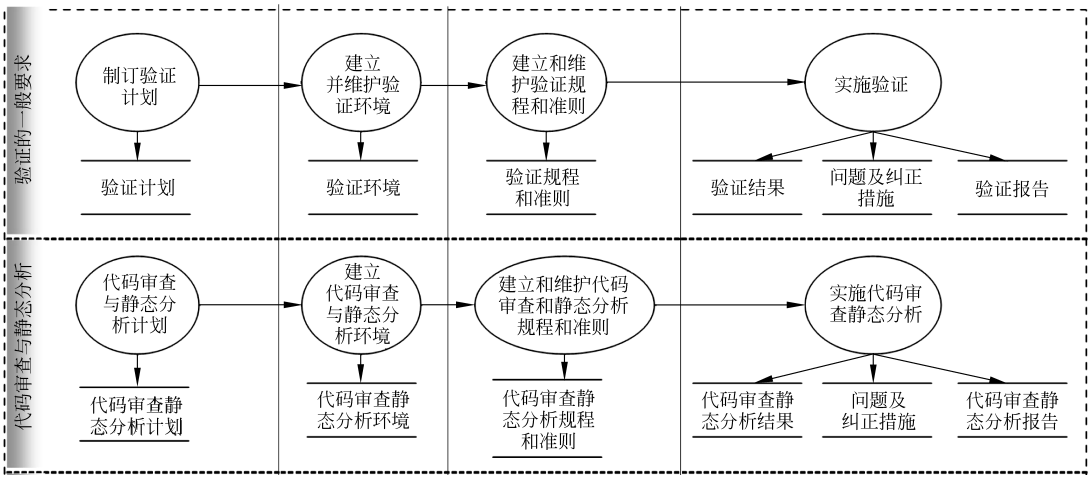


图 4-1　验证过程主要任务示意图

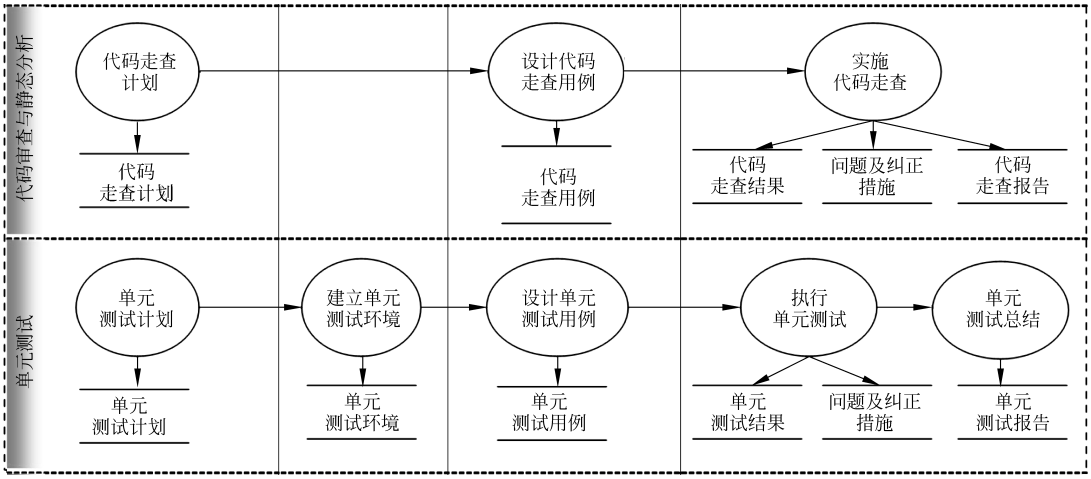


图 4-1(续)

4.2　验证的一般要求

验证活动包括制订验证计划、建立并维护验证环境、建立和维护验证规程和准则、实施验证四个环节。制订验证计划是有效开展验证的必要活动，是确保将验证纳入软件需求、设计、编码等环节而适时进行的必要措施。验证的主要方法包括代码审查、静态分析和单元测试等。验证还应确定验证所需要的环境，包括支持工具、测试设备与软件、仿真、原型和设施等。验证规程和准则的建立是有效实施验证活动的重要保证。实施验证可以帮助发现和解决软件产品存在的问题。

4.2.1　制订验证计划

在项目的初期验证计划的内容主要为标识需要进行验证的工作产品和初步的验证方法。本章主要介绍对代码的验证方法，验证方法一般包括代码审查、静态分析和单元测试等。初步的验证计划可以写入软件开发计划。

验证计划的内容应随着项目的不断深入而更为具体。应包括需要验证的工作产品的具体验证项、每个验证项的验证方法、验证所需要的环境、与相应需求的追踪关系等。

制订软件验证计划是一个循序渐进的过程，随着软件研制过程而逐步实施。

(1) 标识工作产品验证的具体内容。验证人员应根据一定的准则确定要验证的工作产品。标识工作产品验证的准则示例如下。

① 关键算法的实现代码；

② 安全关键的单元。

(2) 标识工作产品要满足的需求。验证活动的目的就是要验证工作产品是否满足相应的需求，因此在制订验证计划时需要借助需求跟踪矩阵梳理工作产品应满足的要求。

(3) 确定工作产品的验证方法。

(4) 编写验证计划。将上述内容写入验证计划。项目的初步验证计划可以与项目管理计划集成,但具体的验证计划应独立成文。例如,单元测试计划等。

4.2.2 建立并维护验证环境

验证环境是开展验证活动的重要保障,验证环境的需求取决于验证所选的工作产品和所使用的验证方法。建立并维护验证环境活动包括明确验证环境需求、建立验证环境。

(1) 明确验证环境需求。根据验证要求和验证方法确定验证所需要的环境需求,包括验证所需要的设备、设施和工具等,并对可重用和修改的验证资源进行说明。应对设备的详细配置和相关工具的版本进行详细描述。这部分内容应写入相关的验证计划中,例如,测试的环境需要可以写入测试计划中。

(2) 建立验证环境。验证人员应根据计划适时建立验证环境,对于测试环境还应对环境是否符合要求进行验证。环境验证的内容包括验证内容、验证方式、验证结论和验证人员等。

4.2.3 建立和维护验证规程和准则

验证规程为实施验证提供过程和步骤要求,准则为评估工作产品是否满足要求提供依据。采用测试作为验证方法时,验证的规程体现在软件测试用例中,验证的准则体现在两个层次,一是每个测试步骤的期望结果和评估标准,二是测试用例通过的准则。

4.2.4 实施验证

实施验证就是依据制订的验证计划、验证方法、验证规程和准则对所选择的工作产品进行验证。实施验证的活动主要包括执行验证、分析验证的充分性、再次执行验证、分析验证结果和编写验证报告。

(1) 执行验证。执行验证就是依据验证计划、验证规程执行验证,并如实、详细记录验证结果。使用的验证方法、环境、数据与验证计划、验证规程不一致时,应如实记录并说明其原因。

(2) 分析验证的充分性。当验证执行完毕后,应根据验证要求和实际验证的结果,分析验证活动是否满足要求。若是因为工作产品的异常导致的不充分,应具体说明未完成的验证。若是因验证工作的不足造成的不充分,应进行补充验证,以便达到验证的要求。例如,当测试用例执行完毕后发现测试覆盖率为 67%,未达到 80%的覆盖率要求时,需要分析原因,若需要就应补充相应的测试用例。

(3) 当对问题的纠正需要对工作产品进行更动时,应进行再一次的验证。再次验证前应对更动的影响进行分析,以便制订合理、可行的再次验证的计划和规程。

(4) 分析验证结果。在获取验证结果后应将实际结果与期望结果进行比较,对验证结果进行分析。当验证结果与期望结果不一致时,应对验证数据进行分析,记录分析的结果,提交问题报告单。问题报告单模板如表 4-1 所示。

表 4-1　软件问题报告单模板

软件名称			问题标识	
问题类别	需求问题□ 编码问题□	设计问题□ 数据问题□	文档问题□ 其他问题□	
问题级别	重大问题□	严重问题□	一般问题□	改进建议□
问题追踪：				
问题描述：				
附注及修改建议：				
报告人		报告日期		

注：软件问题分为重大、严重和一般三个等级。

(1) 重大问题。软件问题导致程序无法继续运行、丧失主要功能或造成重大损失的，视为重大问题：

① 导致系统死机、崩溃或异常退出；

② 主要功能未实现或实现错误；

③ 造成人员、装备、环境等重大损失；

④ 重要数据丢失，且很难恢复。

(2) 严重问题。软件问题对主要功能、性能有较大影响或造成严重损失，视为严重问题：

① 没有完整实现软件需求，对主要功能、性能等有较大影响；

② 没有正确实现软件需求，对主要功能、性能等有较大影响；

③ 造成人员、装备、环境等严重损失；

④ 重要数据丢失，但能以某种方式恢复；

⑤ 软件文档对主要功能、性能描述缺失或错误。

(3) 一般问题。软件问题对软件功能性能有较小影响或造成一般损失，视为一般问题：

① 没有完整实现软件需求，对软件主要功能、性能影响较小，或对一般功能、性能造成影响；

② 没有正确实现软件需求，对软件主要功能、性能影响较小，或对一般功能、性能造成影响；

③ 软件操作与软件使用说明不符；

④ 软件文档存在准确性、一致性、错别字等影响较小的问题。

测试过程中发现的对其他不方便使用或对软件功能有轻微影响的问题可提出改进建议。

若对验证结果进行分析，确定其是由于验证方法、规程、准则和验证环境的问题时也需要进行标识和记录。

(5) 编写验证报告。在分析验证结果的基础上形成验证报告，验证报告应说明验证的结果与验证依据的符合程度。

4.3　代码审查

代码审查是对软件代码进行静态审查的一项技术，目的是检查代码和设计的一致性、代码执行标准的情况、代码逻辑表达的正确性、代码结构的合理性及代码的规范性、可读性。

代码审查应根据所使用的语言和编码规范确定审查所用的检查单。代码审查一般需使用工具完成。审查内容包括(以结构化为例):

(1) 格式;
(2) 程序语言的使用;
(3) 数据引用;
(4) 数据声明;
(5) 计算;
(6) 比较;
(7) 入口和出口;
(8) 存储器使用;
(9) 控制流;
(10) 参数;
(11) 逻辑和性能;
(12) 维护性和可靠性。

4.3.1 实施要点

代码审查的实施要点主要如下。

(1) 对于代码执行标准的情况、代码逻辑表达的正确性、代码结构的合理性及代码的可读性等,应明确规则检查标准,一般采用开发过程中遵循的标准,也可由测试方制定规则检查标准,规则检查标准应得到委托方的确认。

(2) 尽可能选用相应代码的规则检查工具进行测试,对工具设置的检查规则应符合评审通过的规则。对于工具的检查结果,特别是问题部分,需要人工确认。

(3) 检查代码和设计的一致性需要阅读设计文档和代码,以检查代码实现是否与设计一致。

(4) 报告发现的问题,形成代码审查报告。

(5) 由于软件代码的复杂性,代码审查的通过标准不宜设为 100%满足;测试方可用百分比的方式提出建议通过标准,最终由委托方确定。

(6) 有条件时,在回归测试前,可对软件更改前后版本的代码进行比对。

4.3.2 审查过程

代码审查采用自动化测试工具与人工确认相结合的方式进行。

(1) 制定代码审查单。代码审查单应根据所使用的语言和编码规范制定,重点对代码执行标准的情况、代码逻辑表达的正确性、代码结构的合理性及代码的可读性等进行检查,需得到委托方的确认。通常采取在公认度高的通用检查单(如 MISRA C++:2008)的基础上根据具体情况剪裁的方式,制定所需要的代码审查单。代码审查单示例如表 4-2 所示。

表 4-2　代码审查单示例

序号	类别	检　查　项
1	初始化和定义	只读存储器空白单元的处理是否合理
2		随机存储器空白单元的处理是否合理
3		I/O 地址定义是否正确
4		实际地址范围是多少，可寻址范围是多少，对实际地址范围以外的寻址是否进行了正确的处理
5		变量是否唯一定义
6		变量名称是否容易混淆
7	数据引用	是否引用了未经初始化的变量
8		模块中间的数据关系是否符合约定
9	计算	数学模型的程序实现是否正确
10		变量值是否超过有效范围
11		对非法数据有无防范措施（如除法中除数为 0 等情况）
12		数据处理中是否存在累计误差
13		是否对浮点数的上溢和下溢采取了合理的处理方式
14		数据类型是否满足精度要求
15		数组是否越界
16		是否存在比较两个浮点数相等的运算
17	控制流	每个循环是否存在不终止的情况
18		循环体是否存在循环次数不正确的可能，如是否存在迭代次数多 1 或少 1 的情况
19		是否存在非穷举判断，如果输入参数的期望值为 1、2 或 3，那么逻辑上是否可以判定该值非 1、非 2 就必定是 3，这种假设是否正确
20		中断嵌套及现场保护是否正确
21		条件跳转语句中的条件判断是否正确
22		程序是否转错地方
23		控制逻辑是否完整
24		是否使用了 abort，exit 等跳转函数
25		函数中是否存在多个出口
26		循环中是否存在多个出口
27	多余物	用于增加程序的可测试性而引入的必要功能和特征是否经过验证，证明不会因此影响软件的可靠性和安全性
28		是否存在不可能执行到的模块、分支、语句
29		是否存在定义而未使用的变量及标号

续表

序号	类别	检 查 项
30	安全可靠性设计	数据及标志有无防止瞬时干扰的措施，一般应采用“三比二”比对策略、定时刷新存储单元或回送比对后周期数据等措施
31		重要数据的无用数据位是否采用了屏蔽措施
32		对程序误跳转或跑飞是否采取了防范措施，如陷阱处理或路径判断
33		重要信息的位模式是否避免采用仅使用一位的逻辑“1”和“0”表示，一般使用非全“0”或非全“1”的特定模式表示
34		有无必要的容错措施
35	健壮性设计	对误操作是否有防范措施
36		对于软件的重要功能或涉及系统安全性的功能，一旦硬件发生故障时，软件是否能继续在特定程序上维持其功能
37	格式	程序注释是否正确、有意义
38		每个模块的入口处是否有头说明，包括功能、调用说明、入口说明、出口说明等
39		程序模块的注释率是否符合要求
40	数据处理	缓冲区的使用是否合理
41		数据处理流程是否高效、合理
42		数据处理逻辑是否正确、合理
43		数据处理是否通俗易懂
44		数据处理方法是否简洁、高效、合理
45	其他	模块的规模，即代码行数是否符合要求
46		模块的圈复杂度是否符合要求
47		模块的扇入、扇出数是否符合要求
48		模块的参数化率是否符合要求
49		堆栈的处理是否合理，是否存在错误
50		若使用了“看门狗”技术，其时间周期是否合理
51		每个模块是否完成一个主要功能
52	其他	模块的入口、出口是否进行了现场保护
53		全局变量的不恰当使用

（2）根据代码审查单，设定自动化测试工具的规则集，进行自动化代码规则检查。对于C/C++语言，经常使用的自动化测试工具包括TestBed，CodeCast等。自动化测试工具运行后，将生成自动化检查的结果，一般地，自动化检查结果中将包含大量的提示、警告和错误信息，其中可能含有相当大比例的虚警或误报信息。

（3）采用人工方式对工具检查结果进行分析和确认。需对工具检查结果进行逐条分析，确认其指出的相应代码是否存在问题，必要时，可请软件开发人员对代码进行解释，协助

确定问题。如果确实存在问题，应填写问题报告单。

(4) 采用人工方式，检查代码和设计的一致性。这需要阅读设计文档和代码，比较代码实现是否与设计一致，目前只能通过人工方式进行。同样地，如果存在问题，应填写问题报告单。

在采用人工方式对工具检查结果进行确认，以及检查代码和设计的一致性时，可采用会议方式进行，应详细记录分析结果，特别是审查中发现的问题。应对问题的修改情况进行跟踪，必要时组织再次代码审查。

4.3.3　代码审查结果

完成代码审查后应形成代码审查报告。代码审查报告的内容一般包括审查对象概述、审查时间、审查人员、审查使用工具(如有)、代码审查分析与统计结果(软件单元的规模、圈复杂度、扇入扇出数、源代码注释率等静态特性)及审查问题等。

表 4-3 为一个代码审查结果的汇总统计示例。

表 4-3　代码审查情况统计表

审查软件	×××软件		软件标识		××-×××-×× 1.07	
审查人员	×××		审查使用工具		Testbed 9.5	
审查开始时间	××××年××月××日		审查结束时间		××××年××月××日	
模块名	规模行数	圈复杂度	扇入	扇出	注释率/%	违反代码规则处
F_COMMON 模块	96	7	4	2	24	24
F_CTCAD 模块	12	3	4	0	21	3
F_CYCL 模块	35	4	5	1	23	5
F_DLAYER 模块	214	11	7	3	35	57
F_DUF 模块	113	9	2	9	28	44
F_GEM 模块	78	5	11	1	26	23
……	……	……	……	……	……	……

本例结合了代码审查工具进行，所以编码规则检查的具体情况在工具所出具的结果报告中可以进一步查看。如果使用人工审查，可以结合表 4-2 的审查表来对每个模块进行审查，并列写其中每个模块的违反规则情况。

4.4　静态分析

静态分析是一种对代码的机械性的和程序化的特性分析方法，主要目的是以图形的方式表现程序的内部结构，供测试人员对程序结构进行分析。静态分析的内容包括控制流分析、数据流分析、接口分析、表达式分析等，可根据需要进行裁剪，但至少应进行控制流分析和数据流分析。

4.4.1 实施要点

在静态分析中，测试人员通过使用静态分析测试工具分析程序源代码的系统结构、数据结构、内部控制逻辑等内部结构，生成函数调用关系图、控制流图、内部文件调用关系图、子程序表、宏和函数参数表等各种图形图表，可以清晰地展现被测软件的结构组成，并通过对这些图形图表的分析，帮助测试人员阅读和理解程序，检查软件是否存在缺陷或错误。

1）控制流分析

控制流分析中常用的有函数调用关系图和函数控制流图。函数调用关系图通过树形方式展现软件各函数的调用关系，描述多个函数之间的关系，是从外部视角查看各函数；函数控制流图是由节点和边组成的有向图，节点表示一条或多条语句，边表示节点之间的控制走向，即语句的执行，它是从函数内部考察控制关系，直观地反映函数的内部逻辑结构。

函数调用关系图的测试重点主要如下。

（1）函数之间的调用关系是否符合要求。

（2）是否存在递归调用。递归调用一般对内存的消耗较大，对于不是必须的递归调用应尽量改为循环结构。

（3）函数调用层次是否太深。过深的函数调用容易导致数据和信息传递的错误和遗漏，可通过适当增加单个函数的复杂度来改进。

（4）是否存在孤立的函数。孤立函数意味着永远执行不到的场景或路径，为多余项。

函数控制流图的测试重点主要如下。

（1）是否存在多出口情况。多个程序出口意味着程序不是从一个统一的出口退出该变量空间，如果涉及指针赋值、空间分配等情况，一般容易导致空指针、内存未释放等缺陷；同时，每增加一个程序出口将使代码的圈复杂度增加 1，容易造成高圈复杂度的问题。

（2）是否存在孤立的语句。孤立的语句意味着永远执行不到的路径，是明显的编程缺陷。

（3）圈复杂度是否太大。一般地，圈复杂度不应大于 10，过高的圈复杂度将导致路径的大幅增加，容易引入缺陷，并带来测试难度和工作量的增加。

（4）释放存在非结构化的设计。非结构化的设计经常导致程序的非正常执行结构，程序的可读性差，容易造成程序缺陷且在测试中不易被发现。

2）数据流分析

数据流分析最初是随着编译系统有效目标码的生成而出现的，后来在软件测试中也得到成功应用，用于查找如引用未定义变量等程序错误或对未使用变量再次赋值等异常情况。

如果程序中某一语句执行时能改变某程序变量 V 的值，则称 V 是被该语句定义的；如果某一语句的执行引用了内存中变量 V 的值，则说该语句引用变量 V。

数据流分析考察变量定义和变量引用之间的路径，测试重点通常集中在定义/引用异常故障分析上。

（1）使用未定义的变量。如果一个变量在初始化前被使用，其当前值是未知的，可能会导致危险的后果。

(2) 变量已定义,但从未被使用。该类错误通常不会导致软件缺陷,但应对代码中的所有这种类型的问题进行检查和确认。

(3) 变量在使用之前被重复定义。变量在两次赋值之间未被使用,这种情况比较常见,大部分情况下也不会导致软件缺陷,但也应该进行检查和确认。

(4) 参数不匹配。指的是函数声明中的形参的变量类型与实参的变量类型不同,许多编译器对这种情况执行自动类型转换,但在某些情况下是危险的。

(5) 可疑类型转换。指的是为一个变量赋值的类型与变量本身的类型不一致。类型转换时两种类型看起来可能很相似,但赋值结果可能会导致信息丢失。如果无法避免,应使用显式的强制类型转换。

4.4.2　静态分析过程

静态分析主要通过运行静态分析测试工具对程序代码进行自动化分析的方式开展,需使用人工方式对测试工具生成的结果进行分析并得出结论。如果存在问题,应填写问题报告单。静态分析情况统计示例如表 4-4 所示。

表 4-4　静态分析情况统计示例

被测软件	×××软件	软件标识	××-×××-××-1.07
执行人员	×××	执行时间	××××年××月××日
静态分析使用工具	Testbed 9.5		
控制流分析			
函数调用关系图检查		不通过	调用层次最高达 22 层
函数控制流图检查			
F_COMMON 模块		通过	—
F_CTCAD 模块		通过	—
F_CYCL 模块		不通过	该函数存在多出口
F_DLAYER 模块		通过	—
F_DUF 模块		通过	—
F_GEM 模块		通过	—
……		……	……
数据流分析			
变量和常量是否被引用		不通过	存在定义未使用的变量××和××
变量使用前是否初始化		通过	—
传递参数值是否正确		通过	—

如无静态分析工具,也可完全由人工来执行静态分析,但是工作量较大,工作效率也相对较低。人工执行静态分析时可参考如表 4-5 所示的静态分析检查单。

表 4-5 静态分析检查单

静态分析内容		通过准则	结论			说明
			通过	不通过	不适用	
一、控制流分析						
1.1	是否存在任何条件下都不能运行到的代码	否				
1.2	是否存在不影响任何输入/输出的代码	否				
1.3	是否存在不合理的循环结构	否				
1.4	是否存在失败的递归过程	否				
1.5	是否存在无效的函数参数	否				
1.6	是否存在多个函数出口	否				
1.7	是否存在浮点数相等比较	否				
1.8	是否使用 GOTO 语句	否				
1.9	是否使用赋值测试语句	否				
二、数据流分析						
2.1	变量和常量是否被引用	是				
2.2	变量使用前是否初始化	是				
2.3	传递参数值是否正确	是				
三、接口分析						
3.1	各内部接口的定义与使用是否一致	是				
3.2	外部接口的定义是否一致	是				
四、表达式分析						
4.1	表达式中的括号使用是否正确	是				
4.2	存在数组下标越界	否				
4.3	表达式中的除数为 0	否				
4.4	是否有输入/输出数据超出定义域值域范围	否				
4.5	对负数开平方,或对 π 求正切值造成错误	否				
4.6	对浮点数计算的误差对结果造成较大影响	否				
五、质量度量指标						
5.1	文本度量-软件单元的语句数	小于 200				
5.2	注释度量-软件单元得有效注释率	大于 20%				
5.3	扇出数-函数调用的下层函数个数	小于 7				
5.4	局部变量-局部变量个数	小于 8				
5.5	函数参数-函数参数个数	不大于 7				

续表

静态分析内容		通过准则	结论			说明
			通过	不通过	不适用	
五、质量度量指标						
5.6	结构度量-圈复杂度	不大于 10				
5.7	结构度量-基本圈复杂度	不大于 4				

4.4.3　静态分析结果

静态分析的结果包括静态分析测试工具的原始结果，以及人工分析得出的结论。可以形成单独的静态分析报告，或者合并到其他静态测试（如代码审查）的报告中。

4.5　单元测试

4.5.1　概述

单元测试是对软件组成的最小单元进行的测试。一般来说，软件单元是软件设计中的最小单元，一个最小单元应有明确的功能、性能和接口，并且可清晰地与其他单元划分开。根据软件开发使用的编程语言不同，软件单元的划分也不尽相同，单元的选取可参考如下原则：

（1）在结构化编程语言，如 C 语言的程序中，一般认为一个函数就是一个单元；

（2）在面向对象编程语言，如 C++ 、Qt、Java 程序中，可以将一个类作为一个单元；

（3）在可视化编程环境下的面向对象语言，如 Visual Basic、Visual C++ 或 C# 程序中，可以将一个窗体、组件作为一个单元。

有的观点认为软件测试的目的只在于发现软件错误。其实，这种观点是片面的。就软件单元测试而言，单元测试的目的是验证每个软件单元是否满足软件详细设计说明的各项要求，同时，其目的也包括发现软件单元可能存在的错误。根据现在软件工程的要求，单元测试的主要目的包括：

（1）验证代码是否与软件设计一致。

（2）发现软件代码是否存在错误。

（3）跟踪软件的实现。

（4）复核软件设计对软件需求的实现情况。

4.5.2　单元测试原则

在进行软件单元测试时，应该遵循以下原则。

（1）单元测试应该依据软件详细设计进行。

(2) 在对软件单元进行动态测试前,应对软件单元的源代码进行静态测试。静态测试的内容一般包括:代码和设计的一致性、代码执行标准的情况、代码逻辑表达的正确性、代码结构的合理性及代码的可读性等。

(3) 单元测试应覆盖软件设计文档中规定的软件单元的所有功能要求。

(4) 应对单元的健壮性进行测试。

(5) 应对单元的内存使用率、响应时间等进行测试。

(6) 测试用例的输入应至少包括有效等价类值、无效等价类值和边界数据值,既要对正常的处理路径进行测试,也要对异常处理路径进行测试。

(7) 对于安全关键等级较高的软件(如 A、B 级软件),单元测试的语句、分支覆盖率均应达到 100%。对于用高级语言编制的 A、B 级软件,还需进行修正的条件判定覆盖(MC/DC)100%测试。另外,应对没有覆盖到的语句和分支进行分析,说明其未被覆盖的原因。

(8) 应对单元的输出数据及其格式进行测试。

4.5.3 单元测试环境

单元测试的重要特点就是需要开发一定的桩模块和驱动模块,这是单元测试环境区别于其他阶段测试的重要标志。

由于软件单元往往不是一个能够独立运行的程序,所以在进行单元测试时必须开发测试用的桩模块和驱动模块。驱动模块将测试数据注入要测试的软件单元,替代调用被测软件单元的角色,同时驱动模块需要记录相关结果。桩模块的作用是替代被测单元需要调用的软件单元,需要实现与被测单元的接口,模拟相应的操作,提供处理结果的记录。

建立单元测试环境的主要工作包括:

(1) 开发必需的桩模块和驱动模块;

(2) 准备单元测试数据;

(3) 获取单元测试工具,包括购置和开发测试工具。

测试环境需要保证能够开展动态测试外,还应能够支持测试结果记录,测试覆盖率记录等。如图 4-2 所示为单元测试环境示意图,当被测软件单元需要调用多个模块才能完成单元测试时,需要开发桩模块模拟被调用的软件单元,如图 4-2 中桩模块 1～n,若被调用的软件单元已通过测试时,可直接使用而不必开发桩模块,如图 4-2 中软件单元 1～m。

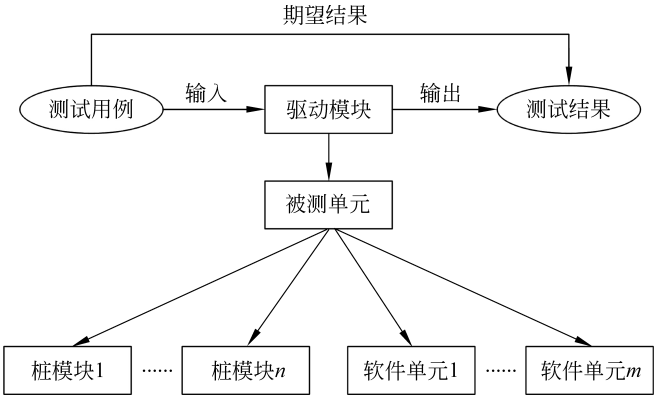


图 4-2 单元测试环境示意图

驱动模块和桩模块的设计原则包括：

(1) 设计驱动模块和桩模块时应考虑测试用例执行所需满足的环境因素，如前置条件、后置条件等；

(2) 应充分考虑驱动模块和桩模块的复用性；

(3) 桩模块的设计应保证功能上与其替代的软件单元的一致性；

(4) 尽量使测试数据与测试程序分离，提高测试数据、测试程序的灵活性和重用性。

在建立单元测试环境时，驱动模块的功能应满足如下要求：

(1) 应可以接收测试数据输入，输入可以包括人工输入、数据文件输入等，测试数据不仅包括注入被测软件单元的数据，还可包括期望结果数据；

(2) 将测试数据注入被测软件单元，一般情况采用调用被测单元的方式进行，利用参数传递将输入数据注入被测单元；

(3) 记录和输出测试结果。

在建立单元测试环境时，桩模块的功能应满足如下要求：

(1) 正确地完成被模拟软件单元的基本功能，这里所谓的完成基本功能并非真正实现被模拟软件单元的功能，而是简单地按照测试用例的需要，将调用被模拟软件单元的结果返回给被测软件单元；

(2) 能够被正确调用，在参数个数、类型、顺序等方面与被模拟软件单元一致；

(3) 有返回值，若被模拟软件单元有返回值，应根据测试用例的要求返回被模拟软件单元应有的返回值。

桩模块和驱动模块开发的工作量较大，有些还较为复杂，此时可以根据实际情况采取相应的单元测试策略。

4.5.4　单元测试内容

单元测试的目的是验证代码是否满足设计的要求，并发现在编码过程中引入的错误。单元测试的主要内容包括功能测试、性能测试、接口测试、局部数据结构测试、边界条件测试、独立路径测试和错误处理测试。

1. 功能测试

软件单元测试的功能测试是对软件设计文档规定的软件单元的功能进行验证。针对软件设计文档分配给软件单元的每一项功能进行测试，确认已实现的功能是否满足软件设计文档的要求。

2. 性能测试

软件单元测试的性能测试是对软件设计文档规定的软件单元的性能进行验证。如精度、时间、容量，应测试软件单元所实现的性能指标是否满足设计要求。

3. 接口测试

软件单元测试的接口测试是验证进出单元的数据流是否正确，接口测试是单元测试的基础。对单元接口数据流的测试必须在其他测试之前进行。

针对单元接口进行的测试，主要涉及如下几方面的内容。

(1) 调用被测单元时的实际参数与该单元的形式参数的个数、属性、量纲、顺序是否一致。

(2) 被测单元调用下层单元时,传递给下层单元的实际参数与下层单元的形式参数的个数、属性、量纲、顺序是否一致。

(3) 是否修改了只作为输入值的形式参数。

(4) 调用内部函数的参数个数、属性、量纲、顺序是否正确。

(5) 被测单元在使用全局变量时是否与全局变量的定义一致。

(6) 在单元有多个入口的情况下,是否引用了与当前入口无关的参数。

(7) 常数是否当作变量来传递。

(8) 输入/输出文件属性的正确性。

(9) OPEN 语句的正确性。

(10) CLOSE 语句的正确性。

(11) 规定的输入/输出格式说明与输入/输出语句是否匹配。

(12) 缓冲区容量与记录长度是否匹配。

(13) 文件是否先打开后使用。

(14) 文件结束条件的判断和处理的正确性。

(15) 输入/输出错误是否检查并做了处理及处理的正确性。

4. 局部数据结构测试

在单元测试中,必须测试单元内部的数据能够保持完整性、正确性,包括内部数据的内容、格式及相互关系不发生错误。单元局部数据结构测试内容重点应考虑:

(1) 局部数据的完整性,包括内容、格式及相互关系;

(2) 数据类型及其说明的正确性和一致性;

(3) 变量名的正确性,如变量名是否有拼写错误或缩写错误;

(4) 变量使用的正确性,如未赋值或未初始化就使用变量;

(5) 初始值或缺省值的正确性;

(6) 是否有下溢、上溢或地址错误;

(7) 全局数据对软件单元的影响。

5. 边界条件测试

在对单元进行边界条件测试时,应采用边界值分析方法来设计测试用例,测试与边界值相关的数据处理是否正确。边界测试主要检查以下内容:

(1) 处理 n 维数组的第 n 个元素时是否出错;

(2) 在 n 次循环的第 0 次、第 1 次、第 n 次是否有错误;

(3) 运算或判断中取最大和最小值时是否有错误;

(4) 数据流、控制流中刚好等于、大于、小于确定的比较值时是否出现错误等。

6. 独立执行路径测试

单元测试中,最主要的测试是针对路径的测试,在测试中应对模块中每一条独立路径进行测试。需要特别注意的测试内容主要包括以下内容。

1）计算错误

（1）算术优先级不正确。

（2）混合类型的运算不正确。

（3）初始化不正确。

（4）算法不正确。

（5）运算精确度不满足精度要求。

（6）表达式的符号表示不正确。

2）比较和控制流错误

（1）不同数据类型的比较。

（2）不正确的逻辑运算符或优先次序。

（3）因浮点运算精度问题而造成的比较不相等。

（4）关系表达式中不正确的变量和比较符。

（5）错误或不可能的循环终止条件。

（6）循环变量修改不正确。

7. 错误处理测试

良好的软件设计应该考虑软件投入运行后可能发生的错误，并在程序中采取相应的措施。错误处理测试的重点是检验如果模块在工作中发生了错误，其中的出错处理是否有效。错误处理常见问题主要有：

（1）所提供的错误描述难以理解；

（2）所提供的错误描述信息无法确定引发错误的位置或原因；

（3）显示的错误提示与实际错误不一致；

（4）对错误条件的处理不正确；

（5）对可能发生的错误未进行异常处理。

4.5.5　单元测试方法

单元测试方法可分为白盒测试方法和黑盒测试方法。

1. 白盒测试

白盒测试方法包括逻辑测试、数据流测试、程序变异、程序插桩、域测试和符号求值等。

一般情况下，单元测试对逻辑测试都会有较为明确的要求。逻辑测试是测试程序逻辑结构的合理性、实现的正确性。逻辑测试应由测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。逻辑测试一般需进行：

（1）语句覆盖；

（2）判定覆盖；

（3）条件覆盖；

（4）条件组合覆盖；

（5）路径覆盖。

2. 黑盒测试

黑盒测试方法主要用于软件单元的功能和性能方面的测试，单元测试中黑盒测试常用的方法和技术包括：

(1) 等价类划分法；

(2) 边界值分析法；

(3) 错误推测法；

(4) 因果图法。

在进行功能测试时，需要考虑使用正常数据、边界数据和异常数据来进行测试。另外，还需要考虑接口测试等。

4.5.6 单元测试用例设计

单元测试用例设计可以从以下 5 个方面考虑。

(1) 为系统运行设计用例。第一个单元测试用例一般应覆盖该软件单元的主要功能，主要基于两个目的：

① 证明单元已具备开始测试的条件；

② 验证单元测试环境的可用性。

常用的测试用例设计方法包括：

① 规范导出法；

② 等价类划分。

(2) 正向测试用例设计。测试人员应该依据软件详细设计文档设计单元测试用例，正向测试用例的作用就是验证详细设计文档中所规定的功能、性能指标是否实现。可使用等价类划分方法设计测试用例。测试用例设计时，应注意需要覆盖所有的功能、性能要求。

(3) 逆向测试用例设计。逆向测试就是验证被测软件单元没有做它不应该做的事情。在设计测试用例前应对软件单元进行静态分析，比较静态分析结果与软件详细设计文档是否一致，验证软件实现是否正确。在此基础上，可使用的测试用例设计方法包括：

① 错误猜测法；

② 边界值分析法。

(4) 特殊要求的测试用例设计。在对安全关键等级较高的软件进行单元测试时，需要对软件单元的性能、安全性、保密性等设计测试用例。常使用的方法是规范导出法，以及在安全性分析的基础上进行用例设计。

(5) 覆盖率测试用例设计。通过上述(1)～(4)进行的测试用例设计已经可以开展单元测试了，但为了达到软件单元测试的充分性要求，需要满足一定的覆盖率指标。当执行完上述测试后，若未能达到覆盖率指标要求，还需要分析覆盖率情况，确定未覆盖的原因，以便有针对性地补充设计测试用例。当确实因用例设计不充分时，可根据分析结果补充相应的测试用例。未达到覆盖率的原因可能有：

① 不可能注入的条件；

② 不可达或冗余代码；

③ 不充分的测试用例。

4.5.7　单元测试过程

单元测试依据软件设计文档，按照图 4-3 所示过程开展测试，单元测试过程分为以下 4 个阶段。

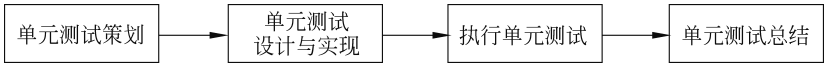


图 4-3　单元测试过程

1. 单元测试策划

依据软件设计文档进行单元测试策划，制订单元测试计划。单元测试策划应尽早进行，一般情况下，可以在软件设计初步完成时就开始进行策划。为了达到质量、进度、效率的平衡，应在单元测试策划时考虑如下因素：

（1）软件的质量要求；

（2）软件研制的进度安排；

（3）软件单元的关键等级；

（4）测试资源的限制。

单元测试策划的内容包括以下内容。

（1）按照软件设计和软件质量要求确定软件单元测试的需求。包括：

① 需要进行测试的软件单元名称、标识；

② 需要测试的内容；

③ 提出每个单元的测试方法，包括驱动模块和桩模块的要求；

④ 提出每个单元测试的充分性要求；

⑤ 测试终止条件；

⑥ 优先级；

⑦ 对软件设计文档的追踪关系。

（2）提出单元测试环境要求，包括测试工具等。

（3）提出单元测试人员安排。

（4）安排单元测试的进度计划。应依据测试需求、测试环境、人员等情况，制订合理可行的软件单元测试进度计划。

（5）制定单元测试通过的准则。单元测试通过的准则如下：

① 软件单元功能与设计一致；

② 软件单元接口与设计一致；

③ 能够正确处理输入和运行中的异常情况；

④ 单元测试发现的问题得到修改并通过回归测试；

⑤ 达到了覆盖率的要求；

⑥ 单元测试报告通过评审。

单元测试策划完成时应编写软件单元测试计划。

2. 单元测试设计与实现

按照单元测试计划规定的内容开展动态测试用例的设计。按照用例要求完成桩模块和驱动模块的设计，完成测试环境的构设，准备动态测试数据。

3. 执行单元测试

按照测试计划、测试说明完成动态测试，记录测试结果、必要时还应完成相应的回归测试。

4. 单元测试总结

软件单元测试完成后应对单元测试工作进行总结，以便评估软件单元测试中的问题是否得到解决，单元测试工作是否达到充分性要求。单元测试总结的内容包括以下内容。

（1）对单元测试过程进行总结。应对单元测试策划、静态测试、动态测试过程进行总结，说明在测试策划过程、静态测试和动态测试中开展的主要工作、参与的人员和工作完成情况。

（2）对单元测试方法进行说明。应说明单元测试所采用的测试方法与策略，并说明采用这些方法的依据。

（3）对单元测试环境进行分析。应说明单元测试所使用的测试环境，包括测试工具、桩模块和驱动模块的情况，并对测试环境的差异性进行分析，说明测试环境是否满足单元测试的要求。

（4）对单元测试结果进行分析。单元测试结果的分析应包括对静态测试、动态测试，以及所有回归测试的情况的分析。主要内容包括：

① 测试时间；

② 测试人员；

③ 测试用例执行情况，内容包括测试用例数、通过的测试用例、未通过的测试用例、完全执行的测试用例、未完全执行的测试用例、未执行的测试用例；

④ 测试覆盖情况，包括对软件单元的覆盖情况，每个单元的覆盖情况（包括语句、分支、路径等的覆盖情况），说明是否满足测试充分性要求；

⑤ 说明单元测试过程中发现的问题，并对问题解决情况进行说明；

⑥ 若有遗留问题应说明遗留问题的处理措施；

⑦ 若有无法执行的测试应说明后续测试的计划等；

⑧ 对软件单元测试的整体情况进行评价，并提出改进的意见建议。

4.6 本章小结

验证活动是保证“正确地构造”产品的重要手段，把握好验证环节，就能够提供高质量的软件产品。因此，需要尽早地开展验证工作。对于高安全性、高可靠性要求的软件，要求进行语句分支覆盖测试，以有效保证测试的充分性。