高等院校计算机应用系列教材

软件测试技术

朱居正　编著

清華大學出版社
北　京

内 容 简 介

本书系统地介绍了软件测试的各个方面，本书共分11章，涵盖了软件测试概述、软件测试计划、黑盒测试与测试用例设计、白盒测试、软件测试过程、测试报告与测试评估、软件测试项目管理、面向对象软件测试、Web应用测试、软件测试自动化以及测试项目案例等内容。通过详细的内容介绍和丰富的实例，本书为读者提供了一个全面的软件测试知识体系，旨在帮助读者深入掌握软件测试的原理、方法和实践经验。

本书内容丰富、结构合理、思路清晰、语言简练流畅、示例翔实，不仅适合作为高等院校计算机相关专业软件测试课程的教材，也可作为软件测试培训班的教材或软件测试人员的自学参考书。

本书配套的电子课件、习题答案和实例源文件可以到http://www.tupwk.com.cn/downpage网站下载，也可以通过扫描前言中的二维码获取。

图书在版编目(CIP)数据

软件测试技术 / 朱居正编著. -- 北京 : 清华大学出版社, 2025. 7. -- (高等院校计算机应用系列教材).

ISBN 978-7-302-69797-8

Ⅰ. TP311.55

中国国家版本馆CIP数据核字第2025P2T796号

责任编辑：胡辰浩
封面设计：高娟妮
版式设计：妙思品位
责任校对：成凤进
责任印制：沈　露

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.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：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**19.5　　**字　　数：**462 千字
版　　次：2025 年 9 月第 1 版　　**印　　次：**2025 年 9 月第 1 次印刷
定　　价：79.80 元

产品编号：103186-01

前言

随着信息技术的飞速发展，软件在各个领域的应用日益广泛，软件质量已成为衡量软件成功与否的关键指标之一。作为保证软件质量的重要手段，软件测试的重要性日益凸显。然而，软件测试并非一项简单的工作，它涉及多个方面和环节，要求测试人员具备扎实的理论基础和丰富的实践经验。

本书正是在这一背景下编写的，旨在系统地介绍软件测试的各个方面，帮助读者深入理解软件测试的原理、方法和实践。本书涵盖了软件测试的基本概念、测试计划、测试技术、测试过程、测试用例设计、测试报告与评估、测试项目管理、面向对象软件测试、Web应用测试、软件测试自动化以及实际项目测试案例等内容。

全书共分为11章，主要内容如下。

第1章为软件测试概述，介绍了软件、软件危机和软件工程的基本概念，阐述了软件缺陷与软件故障的区别，以及软件质量与质量模型的重要性。同时，概述了软件测试的基本原则、目标和主要内容。

第2章为软件测试计划，详细讲解了软件测试计划的作用、制订原则和方法，包括如何确定测试范围、选择测试方法、制定测试标准以及编写测试计划文档等内容。

第3章为黑盒测试与测试用例设计，介绍了黑盒测试的基本概念和方法，包括等价类划分、边界值分析、因果图法和决策表法等测试用例设计技术，并通过实例展示了如何设计有效的测试用例。

第4章为白盒测试，深入探讨了白盒测试的原理和方法，包括逻辑覆盖测试、数据流测试、路径测试及变异测试等。通过详细分析和实例，帮助读者掌握白盒测试的核心技术。

第5章为软件测试过程，概述了软件测试过程的各个阶段，包括单元测试、集成测试、系统测试、验收测试、回归测试等，并介绍了每个阶段的测试目标、任务和方法，以及如何进行有效的测试管理。

第6章为测试报告与测试评估，讲解了测试报告的编写方法和测试评估技术，包括软件缺陷的报告、跟踪和管理，以及测试覆盖率和质量评测的方法。通过实例，展示了如何编写高质量的测试报告。

第7章为软件测试项目管理，介绍了软件测试项目管理的基本概念、原则和方法，包括测试项目的范围管理、进度管理、风险管理及成本管理等。通过实例，展示了如何进行有效的测试项目管理。

第8章为面向对象软件测试，针对面向对象软件的特点，介绍了面向对象软件测试的原理和方法，包括面向对象分析测试、面向对象设计的测试以及面向对象编程测试等内容。

第9章为Web应用测试，专门介绍了Web应用测试的技术和方法，包括性能测试、功能测

试、界面测试、客户端兼容性测试及安全性测试等。通过实例，展示了如何对Web应用进行全面测试。

第10章为软件测试自动化，讲解了软件测试自动化的基本概念、作用和实施方法，并介绍了主流的自动化测试工具及其应用。通过实例展示了如何实现测试自动化，以提高测试效率。

第11章提供了一个项目测试的综合案例，选取了实际的医院信息管理系统(HIS)进行测试，展示了软件测试理论在实践中的应用。内容包括测试计划的制订、测试用例的设计、缺陷报告的编写，以及测试结果的总结与分析等。

在编写本书的过程中，我们注重理论与实践的结合，通过详细的章节划分和丰富的实例，力求帮助读者轻松掌握软件测试的核心知识和技术。同时，我们注重内容的先进性和实用性，介绍了当前软件测试领域的新近技术和方法，以及主流的自动化测试工具及其应用。

本书内容丰富、结构合理、思路清晰、语言简练流畅、示例翔实。每章开头的引言部分概述了本章的作用和主要内容。在正文中，结合关键技术和难点，穿插了大量实用的示例。每章末尾都安排了有针对性的思考题和练习题，思考题有助于读者巩固基本概念，而练习题则旨在培养读者的实际动手能力，增强对基本概念的理解和实际应用能力。

本书既适合作为高等院校计算机相关专业的软件测试课程教材，也可作为软件测试培训班的教材或软件测试人员的自学参考书。通过本书的学习，读者能够系统地掌握软件测试的基本知识和技术，提高实践能力，为未来的职业发展奠定基础。

在本书的编写过程中，我们得到了许多专家和学者的支持与帮助，在此表示衷心的感谢。由于作者水平有限，书中难免存在不足之处，恳请广大读者批评和指正。在编写本书的过程中参考了相关文献，在此向这些文献的作者表达诚挚的感谢。我们的电话是010-62796045，邮箱是992116@qq.com。

本书配套的电子课件、习题答案和实例源文件可以到http://www.tupwk.com.cn/downpage网站下载，也可以通过扫描下方的二维码获取。

配套资源

作者

2025年3月

目录

第 1 章

软件测试概述

在当今智能化时代，人们的工作和生活已经离不开软件，每天都会与各种各样的软件打交道。与其他产品一样，软件也有质量要求。为了确保软件产品的质量，除了要求开发人员严格遵循软件开发规范，软件测试是最重要的手段之一。

软件测试是软件工程中的重要部分，是确保软件质量的重要保障。随着软件产业的不断发展和软件复杂度的增加，软件测试的重要性越加凸显，并随着技术的进步而不断演进。本章将对软件与软件测试的基础知识进行讲解。

本章学习目标：

- 了解软件、软件危机和软件工程的概念。
- 了解软件缺陷、软件故障的概念，以及相关案例和产生原因。
- 了解软件质量与质量模型的概念。
- 了解软件测试的含义、原则、过程模型、分类、流程，以及软件测试的发展历程与趋势，并熟悉软件测试人员应具备的基本素养。

1.1 软件、软件危机和软件工程

1.1.1 软件、软件危机和软件工程的基本概念

计算机系统由硬件系统和软件系统两大核心部分组成。在过去的半个多世纪中，得益于微电子技术的飞速发展，计算机硬件技术取得了显著的进步。与此同时，计算机软件作为与硬件系统相辅相成的重要组成部分，是程序、数据以及相关文档的集合。其中，程序指的是根据特定功能和性能需求设计的可执行指令序列；数据是指程序能够操作和处理的信息及其数据结构；而文档则涵盖了与程序设计、开发、维护和使用相关的所有图文资料。

20世纪60年代之后，随着计算机技术的飞速发展，软件功能日益复杂，软件需求也急剧上升。软件开发逐渐从早期以个人为主的手工作坊模式，转变为以团队为主的集体开发模式。在这一转型过程中，国际软件开发人员在开发大型软件系统时面临诸多挑战，部分项目最终以失败告终；有的项目虽然完成，但延迟了数年，成本也远超预算；还有一些项目未能完全满足用户的需求；部分系统难以进行后续的修改和维护。例如，美国IBM公司的OS/360系统和美国空军的某后勤系统，耗费了大量资源，历尽艰难，最终结果却令人沮丧。

落后的软件生产方式无法满足计算机软件需求的快速增长，导致了软件开发和维护过程中出现了一系列严重问题，这一现象被称为“软件危机”。软件危机主要体现在以下几个方面。

(1) 软件生产无法满足日益增长的需求，其生产率远低于硬件和计算机应用的增长速度，导致社会面临软件供不应求的困境。更为严重的是，随着软件规模的扩大和复杂性的提升，软件生产效率急剧下降。

(2) 随着规模和复杂性的增加，软件生产率逐渐降低。此外，智力密集型行业的劳动力成本不断攀升，这些因素共同作用，使软件成本在计算机系统总成本中的占比迅速上升。

(3) 软件开发的进度和成本往往难以控制。通常情况下，预算会大幅超出，项目计划也频繁延期。为了赶进度和节约成本，开发单位往往牺牲软件质量，导致软件开发陷入高成本、低质量、用户不满和信誉下降的恶性循环。

(4) 软件系统实现的功能常常与实际需求不符。由于开发人员对用户需求理解不够深入，急于编码，导致闭门造车，最终交付的软件与用户实际需求存在较大差距。

(5) 软件难以维护。程序中的错误难以修正，软件难以适应新的运行环境，且在使用过程中难以增加用户所需的新功能。许多开发人员在重复开发相似的软件，造成了资源浪费。

(6) 软件文档配置未得到足够重视。软件文档涵盖开发各阶段的说明书、数据词典、程序清单、使用手册、维护手册、测试报告和测试用例等。不规范和不完善的文档是导致软件开发进度和成本失控，以及软件维护和管理困难的关键因素。

软件危机实际上涵盖了软件开发与维护过程中普遍存在的诸多问题。在过去的四十年中，计算机科学家和软件行业的从业者们投入了巨大的努力来解决这些问题。

软件危机的成因可以从两个维度来理解：一方面是软件产品本身的固有属性；另一方面则是软件专业人员自身存在的不足。

软件的不可预测性是其产品固有属性之一。与硬件产品不同，软件构成了计算机系统中的逻辑组件。在程序代码执行之前，开发工作的质量与进度难以准确评估。软件产品的最终使用价值通常在其运行过程中显现。软件故障往往隐蔽，其可靠性难以量化，而对已有问题的修复可能会引发新的错误。

软件产品的另一个核心特性是其庞大的规模和复杂的逻辑结构。现代软件产品通常具有广泛的功能和复杂的逻辑关系。从软件开发管理的视角来看，随着软件规模和复杂性的增加，软件生产率往往呈现下降趋势。根据当前的软件技术水平，软件开发的工作量随着软件规模的扩大而呈指数级增长。

软件开发人员面临的挑战主要源于软件产品是人类思维的产物，因此其生产质量在

很大程度上依赖于开发者的教育背景、专业训练和经验累积。对于规模庞大的软件项目，通常需要团队协作，甚至要求开发人员深入理解特定领域的复杂问题。这就要求用户与开发人员之间，以及开发团队成员之间进行有效的沟通。然而，在这一过程中，理解上的偏差难以避免，这些偏差可能会导致设计或实现上的错误，而纠正这些误解和错误往往需要巨大的努力和成本。此外，由于计算机技术和应用领域的快速发展，知识更新的周期变得越来越短，软件开发人员必须不断适应硬件的更新换代，并应对不断扩展的应用领域。因此，软件开发人员在进行每一项开发工作时，几乎都需要调整自己的知识结构，以适应新的问题解决需求。

为克服软件危机，必须采取技术和管理双重措施。软件工程作为一门研究如何更高效地开发和维护计算机软件的学科，正是从这两个方面入手的。

软件工程涉及利用计算机科学、数学以及管理科学的原理来指导软件开发的工程实践。简而言之，它是一套用于构建大型程序的指导原则和方法论，将其他工程领域的成熟知识应用于软件开发，以确保在开发过程中遵循工程化的原则和方法。

1.1.2　软件工程的目标及其一般开发过程

狭义上，软件工程致力于开发出符合预算、按时交付且令用户满意的无瑕疵软件产品，并确保这些产品在用户需求发生变化时能够轻松调整。广义上，软件工程的目标是提升软件产品的质量和生产效率，以达成软件产业化的终极目标。

在软件工程中，生存周期方法学被高度重视，它借鉴了人类解决复杂问题时的分解策略，即将问题拆分成若干子问题，逐一解决。软件工程中的生存周期方法学正是从时间维度对软件开发和维护的复杂性进行拆解，将软件的整个生命周期细分为多个阶段，每个阶段都有其独立的任务，通过逐步完成这些任务来实现目标。从软件产品概念的形成到开发、测试、使用和维护，再到最终的退出使用，整个过程被称为软件生存周期。

生存周期方法在软件工程中的应用，对确保软件产品质量和组织开发工作具有重大意义。首先，它允许将整个开发过程清晰地划分为多个步骤，从而能够分阶段解决复杂问题，并确保每个阶段都有明确的目标，包括对问题的理解与分析、解决方案的选择，以及具体实施方法。其次，将软件开发过程分阶段进行，为中间产品的检验提供了依据。通常，软件生存周期包括问题定义、软件开发、软件测试、软件使用和维护等关键阶段。

1. 问题定义

问题定义包括软件系统的可行性研究与需求分析，其核心任务是明确软件系统的工程需求。

可行性研究的主要目标是全面了解用户需求及其执行背景，从技术、经济和社会三个层面深入研究和评估软件系统的可行性。软件系统开发团队需与用户紧密合作，对用户需求和系统执行环境进行彻底的调查研究，并依据调查结果编写调查报告。随后，团队将依据调查报告及其他相关资料进行可行性分析。一般而言，可行性分析包括技术可行性、操作可行性和经济可行性三个重要方面。根据可行性分析的结论，团队将拟定初步的项目开发计划，规划出软件系统开发所需的时间、资金和人力资源。

需求分析旨在明确软件开发的功能、性能及运行环境限制，并制定软件需求规格说明书和软件系统的测试标准。功能需求需要详细描述软件应实现的具体功能，而性能需求则涵盖软件的灵活性、安全性、稳定性、可维护性及错误处理等方面。运行环境约束指的是软件系统必须遵循的运行环境要求。软件需求既是开发的基准，也是验收的标准。因此，明确软件需求是软件开发的核心任务和挑战，通常需要开发人员与用户之间多次深入的交流和讨论，以达成共识。完成需求分析是一项极为繁重的工作。

2. 软件开发

软件开发遵循需求规格说明书，从抽象层面逐步具体化，直至完成整个开发流程。该过程通常包括设计和实现等关键阶段。设计阶段细分为概要设计与详细设计，核心在于依据需求说明书构建软件系统的架构、算法、数据结构以及程序模块之间的接口信息，并设定设计限制，为源代码编写提供必要的指导。在实现阶段(即编码阶段)，开发者将设计成果转化为计算机可执行的代码。在编码过程中，开发者需遵循统一且标准化的编码规范，以确保代码的可读性、易于维护性，并提升运行效率。

3. 软件测试

软件在设计过程中可能出现的问题，必须通过严格的测试来发现并修正。测试流程包括单元测试、集成测试、系统测试和验收测试4个阶段。测试技术主要包括白盒测试和黑盒测试。在测试阶段，必须制订详尽的测试计划，并严格遵循该计划，以降低测试的随机性。统计数据显示，软件测试所占的工作量通常超过软件开发总工作量的40%，在某些情况下，软件测试的费用甚至可能达到软件工程其他环节总成本的3~5倍。

4. 软件使用和维护

经过测试阶段后，软件将被部署到用户指定的运行环境中并交付给用户。软件维护涉及对软件系统进行必要的调整或响应软件需求的变更。每当软件中出现潜在错误、用户需求发生变化或软件运行环境发生改变时，软件维护就显得尤为重要。维护的效果会直接影响软件的应用成效和生命周期。鉴于软件的可维护性与设计紧密相关，因此在开发过程中重视软件的可维护性是至关重要的。

软件生存周期的最终阶段是结束对软件系统的支持，即软件的退役。

1.1.3 软件过程模型

遵循恰当的软件过程模型是确保软件开发质量的关键。在软件开发中，由于众多复杂风险因素的存在，开发人员通过长期实践总结出多种软件工程方法，即软件过程模型。这些模型包括瀑布模型、螺旋模型、增量模型、快速原型模型及敏捷模型等。作为软件开发的指导原则和总体架构，这些模型不仅展示了人们对软件开发流程理解的深化，也标志着认识上的重大进步。

1. 瀑布模型

瀑布模型主要用于软件工程的初期阶段，体现了当时人们对软件开发的线性思考方

式。该模型将软件开发过程划分为固定的阶段和顺序，并强调各阶段任务和文档的完整性。这是一种遵循严格线性顺序、逐步细化的开发方法，如图1-1所展示。

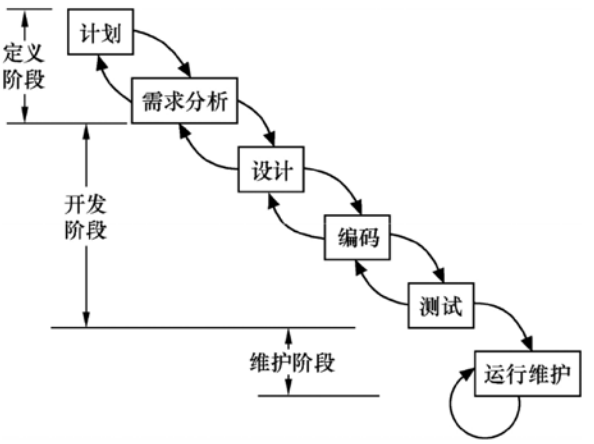

图1-1　瀑布开发过程

瀑布模型由一系列顺序相连的活动构成，包括计划、需求分析、设计、编码、测试及运行维护等关键阶段，各阶段的核心任务如下。

- 在计划阶段，主要任务是确立软件开发的总体目标，包括软件的功能、性能、可靠性及接口；评估软件开发的可行性，探讨解决方案；评估可用资源(包括硬件、软件和人力等)、成本、预期效益及开发进度；并制订详尽的开发实施计划。
- 需求分析阶段的主要任务是收集软件需求，对软件进行明确的定义。软件开发人员与用户共同讨论确定可实现的需求，提供精确的描述；编写软件需求规格说明书等文档并提交审核。
- 设计阶段分为概要设计和详细设计，是软件开发过程中的核心环节。概要设计将确定的需求转化为相应的系统架构，确保架构中每个部分都是功能清晰的模块，每个模块都能体现相应的需求。详细设计阶段则对每个模块应完成的工作进行具体描述，为编程阶段的实施打下基础。
- 编码阶段主要是将设计阶段各模块的描述编写成可执行的代码。
- 测试阶段的目的是发现软件中的错误。
- 运行维护阶段主要负责对运行中的软件进行必要的修复和修改，以确保其持续有效运行。

在软件开发过程中，各阶段的执行并不总是遵循图1-1所示的顺序进行，导致各阶段间存在一定的重叠现象。这种重叠现象的产生是因为在一个阶段结束之前，下一个阶段往往已经开始了。因此，软件开发人员很少采用图1-1所示的纯粹瀑布模型，除非是针对小型项目或者开发的产品与他们之前的工作相似。这主要是由于软件项目的复杂性和非线性的特点所致。

2. 螺旋模型

螺旋模型涉及反复进行需求分析、设计、实现和测试等活动。螺旋模型的核心理念是利用前一版本的成果来构建新版本，这种持续的迭代过程形成了螺旋式上升的轨迹，如图1-2所示。螺旋模型的主要目的是降低风险，同时在软件开发初期阶段，将部分成果展示给客户以获取他们的反馈，从而避免瀑布模型中一次性编写大量代码所带来的问题。

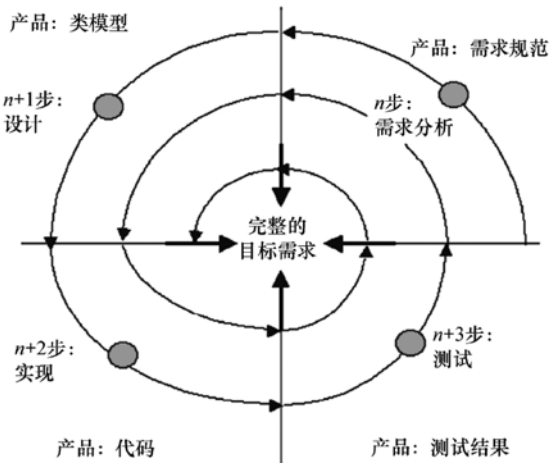

图1-2　螺旋开发过程

螺旋模型允许在每个循环阶段收集各种度量数据。例如，在初始循环阶段，可以记录团队在设计和实施过程中所花费的时间，从而优化后续阶段的时间估算方法。这对于缺乏历史数据的开发团队尤为重要。

尽管螺旋过程模型与典型软件项目的发展趋势相契合，但与简单的瀑布模型相比，它需要更多的精力来进行精细的过程管理。因为每次循环结束后，都必须确保文档的一致性，尤其是代码必须符合文档中描述的设计和记录的需求。为了提升开发团队的效率，通常会在前一个循环结束前启动新的循环，这使得保持文档一致性变得更加复杂。

螺旋开发过程需要进行多少次循环？这取决于具体情况。例如，一个由3人组成的团队，耗时4个月的项目，可能需要进行2到3次循环。然而，如果项目采用5次循环，那么所需的管理成本通常会超过新增循环带来的价值。

3. 增量模型

在软件开发过程中，每次迭代仅添加少量功能时，这种开发方式被称为增量模型。增量模型通过持续的小步骤推进项目，如图1-3所示。增量模型特别适合项目的后期阶段，例如在维护阶段，或者当新立项的产品与已开发产品结构高度相似时。

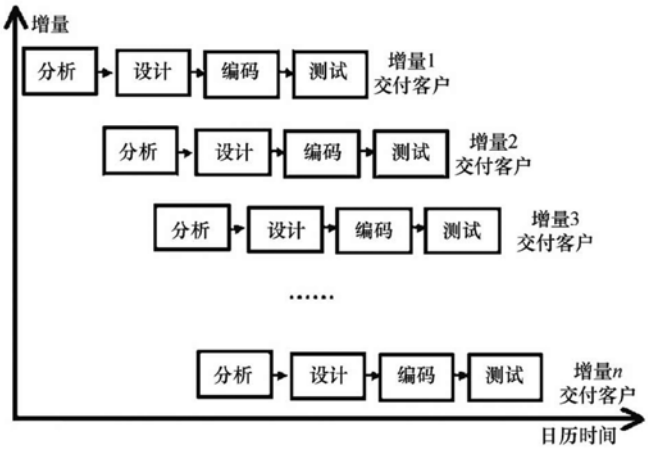

图1-3　增量开发过程

4. 快速原型模型

在快速原型模型中，首先是迅速开展需求分析工作。需求分析团队与用户紧密合作，快速明确软件的核心需求，并构建一个具有基础功能但可操作的原型系统。随后，通过逐步优化和扩展，不断完善该原型系统，最终形成完整的软件系统。

原型系统作为应用系统的示例，允许用户在开发者的协助下体验并评估原型的性能，检验其是否符合规格要求和用户期望。开发人员依据用户的反馈，纠正交互误解和分析错误，补充新需求，并针对环境变化或用户新想法导致的系统需求变动提出修改建议。在大多数情况下，原型系统中的不足之处可以得到修正，这些修正成为新模型的基础。开发者和用户在迭代过程中持续改进原型系统，直至软件性能满足用户需求。因此，快速原型开发模型能够帮助开发人员迅速实现目标系统。

快速原型模型的主要优势在于其用户导向性，它鼓励用户在系统生命周期的设计阶段积极参与，并有助于降低系统开发过程中的风险。特别是在大型项目开发中，由于需求分析难以一次性完成，快速原型模型的应用效果尤为显著。

5. 敏捷模型

敏捷模型采用迭代增量的方式进行软件开发。敏捷开发方法强调以人为本，通过迭代和逐步完善的方式进行开发。在敏捷开发实践中，软件项目被拆分成多个子项目，每个子项目的成果都经过测试，以确保它们能够独立运行并顺利集成。换句话说，敏捷方法将一个大型项目分解为多个既相互关联又可独立运作的小项目，并分别完成这些小项目。在整个过程中，软件始终保持可用性。

敏捷软件开发旨在取代传统的瀑布模型，后者严格按照预设的顺序执行需求分析、设计、编码和测试等步骤。在瀑布模型中，通过各个阶段的成果来衡量项目进度，例如需求规格书、设计文档、测试计划和代码审查等。瀑布模型的主要缺陷在于其严格的阶段划分限制了灵活性，使得一旦项目初期的分析完成，后期需求发生变化时难以适应，调整成本也因此高昂。因此，在需求不明确或可能发生变化的项目中，瀑布模型通常不适用。

相较于其他方法，敏捷方法更倾向于在数周或数月内完成较小的功能模块，其核心在于尽可能早地提供最小可用功能，并在项目持续期间不断进行优化和增强。

敏捷方法特别适合于小规模任务，这些任务在每个迭代周期以及迭代结束时发布的软件中得以体现。其主要优点在于能够完全适应用户需求的变化，并对产品进行持续的迭代更新。敏捷方法更重视交付可运行的软件，而不仅仅是满足需求规格书中的要求。

上述五种模型仅是众多软件过程模型中的一部分，此外还有喷泉模型、统一软件开发模型等其他模型。探讨这些模型的目的是强调软件过程模型在软件工程中的重要性，因为从某种角度来说，不熟悉软件过程模型就无法真正理解软件工程的核心。

同时，我们应意识到，构建一个全面且成熟的软件开发流程并非一朝一夕之事。它涉及从无序到有序、从特殊到普遍、从定性到定量的逐步演进，最终实现从静态到动态的转变。换言之，在达到成熟的软件流程之前，软件开发组织必须经历一系列探索阶段。因此，建立一个软件过程成熟度模型显得尤为必要，它能够对流程进行客观和公正的评估，从而推动软件开发组织优化其软件开发流程。这正是软件能力成熟度模型(CMM)所致力于实现的目标。

1.2 软件缺陷与软件故障

1. 什么是软件缺陷和软件故障

软件的制作离不开人的参与，而人的参与就意味着无法做到尽善尽美。软件开发是一项极其复杂的任务，开发人员在任务过程中难免会出现错误。尽管开发人员投入了大量努力，错误仍然不可避免。因此，软件中出现错误是其固有特性，无法完全消除。

通常，软件测试的目标就是尽可能地发现软件缺陷，并通过修正错误来提升软件的整体质量。

软件错误通常指的是在软件生命周期中出现的非预期或不可接受的人为失误，这些错误会导致软件缺陷的产生。

软件缺陷是指软件(包括文档、数据和程序)中出现的非预期或不可接受的偏差，这些缺陷会在软件运行于特定条件下时引发软件故障(此时缺陷被激活)。

软件故障是指软件在运行时产生的非期望或不可接受的内部状态，若没有及时采取适当的措施(如容错技术)来处理这些故障，软件可能会失效。

软件失效是指软件在执行过程中产生的非预期或不可接受的行为结果，这通常导致功能模块无法完成其预设的任务。软件失效的机制可概括为：软件错误→软件缺陷→软件故障→软件失效。

2. 软件缺陷和软件故障案例

软件缺陷和软件故障案例有很多，以下是一些经典的例子。

1) 千年虫问题

在20世纪70年代末期，为了节省宝贵的内存和硬盘空间，程序员在保存日期信息时，仅保留了年份的最后两位数字。例如，“1980年”被简化为“80”。然而，当2000年到来之际，这一问题开始显现。以银行存款系统为例，当计算利息时，系统会用当前日期“2000年1月1日”减去存款时的日期，比如“1979年1月1日”，理应得到21年的存款期限。假设利息率为3%，那么每100元存款，银行应支付给客户大约86元的利息。但是，如果程序未能修正仅存储年份最后两位的缺陷，计算出的存款年数会变成负的89年，导致客户不仅无法获得利息，反而需向银行支付高达1288元的费用。因此，随着2000年的临近，为解决这一设计上的小疏忽，全世界不得不花费数十亿美元来应对。

2) 美国东北部大停电

2003年，美国东北部和加拿大部分地区发生大规模停电，软件缺陷和错误警报是这次事故的重要原因之一。能量管理系统的软件未能准确监控电网状态，并在问题发生时产生了大量错误警报。

3) 日本“希望”号火星探测器失踪

1998年，日本发射的“希望”号火星探测器在飞往火星的途中失踪，原因是软件错误导致推进器未能正确点火。

4) Ariane 5火箭爆炸

Ariane 5火箭在升空后40秒后发生爆炸，原因是系统软件中的整数溢出漏洞引发软件崩溃，最终导致火箭爆炸。

5) 爱国者导弹软件错误

在第一次海湾战争期间，爱国者导弹系统因软件缺陷未能及时发现并拦截伊拉克导弹。

6) Windows计算器软件缺陷

Windows系统自带的计算器在计算$\sqrt{9}-3$时得出的答案不正确，这个问题在各个操作系统中均存在。

7) UNIX系统中的时间终结

UNIX系统中的时间显示将在2038年1月19日无法正常运行。

8) 阿里云上海地域可用区N网络异常

2024年7月2日，阿里云上海地域可用区N网络访问异常，导致多个应用出现网络故障。

9) 腾讯云服务器故障

2024年4月8日，腾讯云出现服务故障，网页显示504错误，服务器无法连接，故障持续近87分钟。

10) 美团App故障

2024年4月26日，美团App因系统升级出现主页面无法加载以及外卖等服务不可用的情况。

以上案例展示了软件缺陷和故障在不同场景下的具体表现和影响，提醒我们在软件开发和维护过程中需要高度重视软件质量和稳定性。

3. 软件产生错误的原因

软件产生错误的原因多种多样，为了有效避免这些错误，我们必须了解其成因，主要包括以下几点。

(1) 软件的复杂性。软件是思维的产物，其复杂性日益加剧。随着计算机技术的飞速发展，软件功能和结构日益繁复，算法难度也在不断提升。尽管软件对精确度要求极高，但任何环节的失误都可能引发软件错误。因此，软件缺陷层出不穷。

(2) 沟通不足、存在误解或缺乏沟通。软件的复杂性意味着随着项目规模的扩大，个人难以独立完成任务，团队合作成为必然。然而，确保团队成员间思想统一是一项重大挑战。由于人与人之间思想的差异，沟通不足、误解或缺乏有效沟通，都会在软件开发和维护过程中引发一系列严重问题。

(3) 程序设计失误。程序员也是人，难免会犯错。有些错误可能是偶然发生的，而程序设计失误往往源于一时疏忽。

(4) 需求变更。需求变更的影响是多方面的，客户可能未意识到需求变更带来的后果。需求变更可能导致系统重新设计，设计人员的日程也需要重新调整。已完成的工作可能需要重做或完全废弃。此外，需求变更可能影响其他项目，并导致硬件需求的变化。无论是多个小的调整还是一次重大的变更，项目各部分之间已知或未知的依赖性都可能会加剧，从而引发更多问题。因此，需求变更带来的复杂性往往会导致错误的发生。

(5) 时间压力。软件项目的进度表往往难以精确预测，通常需要估计和推测。当截止日

期临近或关键时刻到来时，错误往往会随之出现。

(6) 代码文档不足。缺乏规范的代码文档使得代码维护和修改变得异常困难，这可能导致许多错误的发生。

(7) 软件开发工具。包括可视化工具、类库、编译器和脚本工具在内的各种软件开发工具，可能会将自身的缺陷带入应用软件中。实际上，无论采用何种技术或方法，软件中仍然可能存在错误。虽然使用新技术、先进的开发方法和完善开发流程可以减少错误的引入，但无法完全消除软件中的错误。这些错误需要通过测试来发现和评估。

1.3 软件质量与质量模型

软件质量问题是软件工程领域的核心。那么，何为软件质量？这实际上是一个多维度的概念，不同个体从各自视角审视软件质量问题时，往往会有各自的解读。通常，人们会用诸如某软件易用、功能完备、架构合理、层次清晰、运行迅速等模糊表述来评价软件质量，但这些并不能构成对软件质量的科学评估。随着计算机软硬件技术的进步，人们对软件质量的认识日益加深，相关标准随之演变。根据ISO/IEC 9126-1991(GB/T16260-1996)“信息技术软件产品评价质量特性及其使用指南”标准，软件质量的定义如下。

软件质量涉及软件产品满足既定或潜在需求的能力，涵盖了特性与属性的综合。其核心意义体现在以下几个方面。

- 符合特定需求的软件特性。软件需求是评价软件质量的关键，不满足需求的软件无法称得上高质量。因此，软件必须在功能、性能等方面达到预期，并能够稳定运行。
- 达到预期属性组合的程度。这意味着软件结构合理，系统资源得到合理运用，代码易于阅读、理解，并便于修改，从而方便进行软件维护。
- 满足用户综合期望的程度，即软件系统拥有用户友好的界面，使得用户使用起来更加便捷。
- 软件的综合特性。在软件的整个生命周期中，各阶段的文档都应完整且规范，以利于配置管理。

软件质量特性，即影响软件质量的多种因素，构成了一个复杂的集合。这些特性揭示了质量的核心，而探讨软件质量问题，本质上是要明确软件质量特性的定义。

要深入探究影响软件质量的关键因素及其评价方法，首先需要确立一套科学的质量评估标准，以确保软件产品达到预期的质量要求。为了解决这一问题，构建一个易于理解和操作的质量模型显得尤为重要，这不仅有助于全面评估软件质量，还能有效识别和管理潜在风险。目前，学术界和产业界已提出多种质量模型，这些模型从不同维度定义了软件质量的各项属性。其中，McCall模型、Boehm模型和ISO/IEC 9126模型尤为突出，它们的共同特点在于采用了分层结构来构建软件质量特性体系。这些被广泛认可的质量特性模型通常采用双层架构：顶层由若干基本质量特性构成，这些特性按照主要类别进行划分；底层则进一步细化为各个主要类别下的子类质量特性；最后，在每个子类中详细列出与之对应的

具体评价标准或相关指标。这种层次化的结构设计不仅使质量评估更加系统化，也为软件质量的持续改进提供了清晰的指导框架。

1976年，Boehm及其同事首次提出了软件质量模型的层次结构。随后，在1979年，McCall及其团队对Boehm的质量模型进行了优化，提出了一个新的软件质量模型。该模型的质量定义基于11个关键特性，这些特性涵盖了软件产品的运行、修正和转移等方面。这些特性与软件质量的关系在图1-4中进行了展示。McCall等人坚信，特性是衡量软件质量的指标，而软件属性则可以作为评估标准。通过量化地测量这些属性，可以准确地评估软件质量的高低。

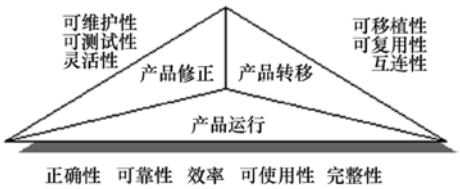

图1-4　McCall质量模型

对于那些缺乏开发经验的个体而言，软件能够顺利运行似乎已经足够。然而，软件工程的终极目标在于开发出符合质量标准的软件产品。产品的质量要求是多维度的，其中正确性仅仅是衡量软件质量的一个方面。软件质量的考量因素众多，包括但不限于正确性、可靠性、可使用性、效率、完整性、可维护性、可测试性、灵活性、可移植性、可复用性以及互连性等。

国际标准化组织(ISO)基于广泛的用户视角，提倡并促进了对软件质量特性统一认识的讨论，逐步达成了共识并持续进行优化。1991年发布的ISO/IEC 9126《软件质量特性与产品评价》标准，初步展示了这一国际性探讨的成果。ISO/IEC 9126-1991标准所规定的软件质量模型由三层构成，如图1-5所示。该模型中，第一层被称为质量特性(SQRC)，第二层被称为质量子特性(SQDC)，第三层被称为度量(SQMC)。此模型定义了8个质量特性，包括正确性、可靠性、可维护性、效率、安全性、灵活性、可使用性及互连性，并推荐了21项子特性，例如完备性和准确性等，但这些子特性并不构成标准的一部分。度量质量子特性的方法没有统一的标准，各组织可根据实际情况自行制定。正如前文所述，软件质量的评价应以用户需求为基准，通常以用户的“满意度”作为衡量标准。

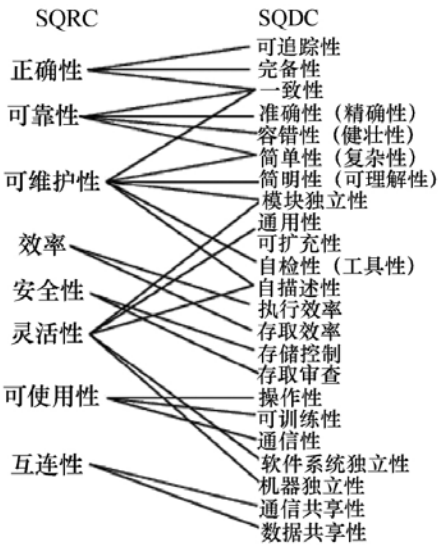

图1-5　ISO的软件质量评价模型

自1997年以来，国际标准化组织(ISO)以软件生命周期为视角，提出了软件质量度量的概念。在这一概念的指导下，ISO对1991年发布的ISO/IEC 9126标准进行了修订，并推出了新的9126系列标准，包括ISO/IEC 9126-1、-2、-3和-4。2001年发布的ISO/IEC 9126《产品质量—质量模型》中，软件质量被定义为包含内部质量、外部质量和使用质量三个维度，如图1-6所示。换言之，软件满足既定或潜在用户需求的能力，需要通过其在内部、外部及使用过程中的表现进行综合评估。

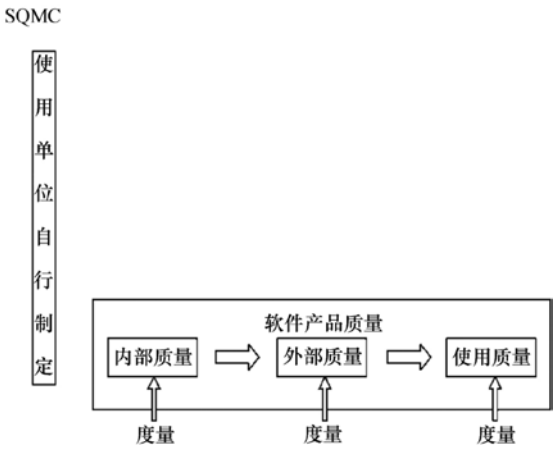

图1-6　新的ISO/IEC 9126软件质量模型

所谓内部质量，是指从软件产品内部视角出发，对其特性的综合评价。这一评价是基于对内部质量需求的精确测量和深入评估。依据新版ISO/IEC 9126《产品质量——质量模型》标准，内部质量被定义为软件产品在特定条件下使用时满足需求的能力属性，它体现了软件开发过程中(涵盖需求开发、设计、编码等阶段)的质量特性。

内部质量特征主要包括以下几个方面。

- 可维护性：指的是对软件系统进行修改以增强其性能或修正错误的能力。
- 灵活性：涉及对系统进行调整以适应多样化用途或环境，而无须进行专门设计的能力。
- 可移植性：意味着对系统设计的某些部分进行调整，以便其能够在其他环境中运行。
- 可重用性：指的是将系统中的某些部分应用于其他系统的难易程度。
- 可读性：涉及理解系统源代码的能力。
- 可测试性：指进行单元测试或系统测试，以验证系统满足所有性能需求的难易程度。
- 可理解性：指从整体或细节层面理解整个系统的难易程度。

软件产品的外部质量指的是在特定环境下使用时，其满足需求的程度。它反映了从外部观察到的软件特性，主要通过在模拟条件下使用外部度量和模拟数据进行测试来评估，即在既定系统环境中可能达到的质量水平。

外部质量特性主要包括以下方面。

- 准确性：指系统受说明、设计和实现错误影响的程度。
- 易用性：指用户学习和使用系统的难易程度。
- 性能：涉及系统对资源的最小化利用，包括存储和执行时间。
- 稳定性：指系统在特定条件下执行功能的能力。
- 安全性：指防止非法或不当访问的能力。
- 兼容性：指系统在不同应用或其他环境中无须修改即可使用的程度，无须经过特定设计。
- 精确度：指系统不受错误影响的程度，特别是在数据输出方面(精确度与准确性不同，它衡量的是系统完成任务的能力，而非设计的正确性)。

- 鲁棒性：指系统在面对无效输入或压力环境时继续执行功能的能力。

软件产品的使用质量被定义为在特定的使用环境下，软件产品满足特定用户实现既定目标的能力。这一定义基于用户视角，旨在评估软件在特定环境和条件下的表现，反映了其在满足用户需求方面的效能。

使用质量通过有效性、生产效率、安全性和用户满意度等特性进行描述。

在实际的软件项目管理中，同时提升所有质量特性是一项极具挑战性的任务。项目管理的三大核心要素——质量、资源和时间相互制约，提高质量往往需要投入相应的成本和时间，这可能导致需要更多资源或项目延期。因此，项目管理应依据项目的具体特性，平衡这三大要素，制定合理且可行的质量目标。

任何产品都是由多个过程构成的，因此，管理和控制这些过程是提升产品质量的关键。对于软件项目而言，为了提高产品质量、缩短开发周期、降低成本，必须对软件开发过程的各个阶段和步骤进行有效的管理和控制。

1.4　软件测试

软件测试作为软件质量保证过程中的核心环节，发挥着至关重要的作用。通过实施软件测试，我们能够对产品的质量进行全方位的评估，从而为软件产品的发布、系统的部署、产品的鉴定以及其他关键决策提供坚实的信息支持。

1.4.1　软件测试的定义与目的

1. 软件测试的定义

软件测试是软件开发流程中不可或缺的环节，其主要任务在于对软件产品进行验证与确认，以评估其质量、发现缺陷或错误，并确保其满足既定需求和预期效果。该过程包括一系列经过精心规划的测试活动，如单元测试、集成测试、系统测试、验收测试等，旨在确保软件在各种环境中能够准确无误、稳定运行。软件测试不仅关注功能的正确实现，还涉及性能、安全性、易用性等多个方面。通过软件测试，可以提升软件的可靠性和用户满意度，同时降低软件发布后的维护成本。

根据IEEE所制定的软件工程标准术语，软件测试被定义为：“运用人工或自动化手段执行或测试系统的过程，旨在检验其是否满足既定需求，或揭示预期结果与实际结果之间的差异。”软件测试与软件质量紧密相关，其核心目标在于确保软件产品的质量。通常，软件质量的评估以“满足需求”为基本准则，而IEEE对软件测试的定义也明确指出，其目标在于验证软件是否符合既定需求。

在软件的整个生命周期中，软件测试扮演着至关重要的角色。根据传统的瀑布模型，测试活动通常被安排在运行维护阶段之前，作为确保软件质量的关键步骤。然而，随着软件工程领域新理念的引入，每个生命周期阶段都应包含测试活动。软件工程的生命周期方法将开发过程细分为多个阶段，这为检验中间产品提供了机会，而完成的文档则成为评估

软件质量的关键。

通常，程序中的错误并非仅由编码引起，它们可能源自详细设计、概要设计甚至需求分析阶段的问题。因此，即便是在源代码层面进行测试，问题的根源可能位于开发过程的早期阶段，解决和纠正这些错误需要追溯到这些前期工作。基于此，测试工作应覆盖整个软件生命周期，特别关注编码之前的各个开发阶段，以确保软件质量。换言之，测试应从软件生命周期的起始阶段开始，并贯穿整个生命周期，以检验各阶段成果是否达到预期目标，并尽可能早地发现并修正错误。若不在早期阶段进行测试，错误的延迟扩散往往会导致最终成品测试的极大困难。美国软件质量安全中心在2000年对100家知名软件公司进行的调查显示：在开发早期发现软件缺陷，与在开发后期发现相比，可以节省90%的资金和人力资源；在产品推向市场前发现缺陷，与推出后发现相比，同样可以节省90%的资金和人力资源。因此，软件测试不应仅仅被视为传统意义上产品交付前的单一“找错”过程，而应是一个贯穿软件生产全过程的科学质量控制过程。从软件项目的需求分析、概要设计、详细设计到程序编码等各个阶段产生的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明及源程序，都应成为软件测试的对象。在整个软件生产过程中，都需要软件测试工程师的参与。

软件测试的核心需求涉及两个主要方面：首先，确保软件产品的精确性和完整性；其次，保障软件在其生命周期各个阶段的逻辑协调性与一致性，以及软件内部结构的统一性。软件生命周期中各开发阶段的主要测试活动如表1-1所示。

表1-1　软件生命周期中的主要测试活动

开发阶段	主要测试活动
需求分析	根据项目需求规格说明书，确定项目中需要进行的测试类型，并据此制订系统测试计划
设计	确认设计是否符合需求，制订集成测试计划和单元测试计划
编码	开发相应的测试代码和测试脚本
测试	执行测试，并提交详尽的测试报告
安装	将经过测试的系统投入生产
维护	修改缺陷并进行重新测试

在需求分析阶段，主要验证需求定义是否与用户需求相符；在设计与编码阶段，确保设计编码遵循需求定义；在测试与安装阶段，检查系统运行是否符合规格；在维护阶段，需要重新测试系统，以确保修改和未修改部分均能正常运作。

2. 软件测试的目的

软件测试的核心目的是确保产品质量，它在开发流程中发挥着关键的质量控制作用。

测试的主要目标是发现程序缺陷，而非保证程序完全无误。软件测试覆盖整个生命周期，用户在使用后也会参与测试。Glen Myers在*The Art of Software Testing*中提出了三个测试原则。

- 测试是执行程序以发现错误的过程。
- 良好的测试用例能发现新的错误。

- 成功的测试是指能够发现新错误的测试。

软件测试的主要宗旨在于以最小的资源消耗(包括人力、物力和时间成本)，揭示软件内部潜在的错误与缺陷。通过此方法，可以对这些错误和缺陷进行修正，从而提升软件的整体质量，并降低因潜在问题引发的商业风险。由于软件是由人类开发的，其固有的不完美性使得错误的出现不可避免。此外，软件开发过程的复杂性使得错误易于产生。尽管软件领域的专家和学者已投入大量努力，但完全避免软件错误仍然是一项挑战。因此，普遍认为软件中存在错误是其固有特性，难以彻底根除。通常，软件测试的目标是尽可能多地发现这些缺陷，并通过修正它们来提升软件质量。

此外，测试的目的不仅限于发现缺陷和错误，还包括对软件质量的评估和度量，以进一步提高软件质量。软件测试是一项旨在评价程序或系统的特定属性的活动，它验证软件是否满足用户需求，并为用户选择和接受软件提供坚实的依据。

同时，分析错误产生的原因有助于识别当前软件开发流程中的不足，为软件流程改进提供方向。通过对测试结果的深入分析和整理，可以优化软件开发规范，并为软件可靠性分析提供支持。

1.4.2　软件测试的原则

为确保测试成效，测试工程师必须深入理解软件测试的核心原则。以下原则通常被视为测试的基石。

(1) 早期测试。将“早期和持续的测试”作为行动准则。鉴于软件的复杂性和程序性，错误可能在软件生命周期的各个阶段出现，因此不应仅将软件测试视为软件开发过程中的一个独立阶段，而应将其融入软件开发的每个阶段。在需求分析和设计阶段，测试工作应当开始，并编写相应的测试文档。同时，要坚持在软件开发的每个阶段进行技术评审和验证，尽早开展测试执行工作。代码模块一旦完成，就应立即进行单元测试；当代码模块集成为相对独立的子系统后，应进行集成测试；一旦有Bug提交，就应进行系统测试。由于测试执行工作提前进行，测试人员能够更早地发现软件缺陷，从而显著降低Bug修复的成本。但需要注意，“早期测试”并不意味着无目的地提前进行测试活动，测试活动的开展应基于达到必要的测试准备条件。

(2) 全面测试。软件由程序、数据和文档构成，因此测试软件不仅限于程序本身，还应涵盖对软件“副产品”的“全方位测试”。需求文档和设计文档作为软件开发过程中的阶段性成果，对软件品质有着直接的影响。阶段性产品的质量是软件整体质量的基础，若无法确保这些阶段性产品的质量，最终软件质量将难以控制。

“全方位测试”涵盖两个方面的内容：首先，它要求对软件的所有组成部分进行彻底测试，包括需求、设计文档、代码和用户文档等。其次，它需要软件开发和测试团队(有时还包括用户)的全面参与。例如，在需求验证和确认的过程中，开发人员、测试人员和用户都应共同参与。因为测试不仅仅是确保软件的正常运行，更重要的是确保软件能够满足用户的需求。

“全方位测试”有助于全面掌握软件质量，并尽可能地消除可能导致软件问题的因素，从而确保软件能够达到既定的质量标准。

(3) 全程测试。“全程测试”包含两层含义：其一，测试人员需密切关注开发流程，并对流程中的任何变化迅速做出响应。例如，开发进度的调整可能需要测试进度和策略的相应调整，而需求变更也可能影响测试的执行。其二，测试人员应全面追踪测试的整个流程，例如建立一套完善的度量和分析体系，以便通过对流程的度量及时掌握过程信息，并据此调整测试策略。

“全程测试”有助于及时应对项目中的变动，降低测试风险。同时，对测试流程的度量与分析也有助于掌握测试流程，调整策略，从而优化测试过程。

(4) 独立性与迭代性的测试。这一概念包含两层含义：其一，测试流程应当与开发流程适度分离，形成一个独立的管理流程。传统的软件开发瀑布模型只是一个理想化的框架。为了应对不断变化的需求，软件开发实践中出现了螺旋模型、迭代模型等多种开发模型。在这些模型中，需求分析、系统设计、编码等环节可能会相互重叠并多次进行。因此，测试工作也应当是迭代和反复的。若无法将测试从开发流程中独立出来进行管理，测试管理将不可避免地陷入困境。其二，测试工作应由独立的专业软件测试机构执行。通常情况下，程序设计者对自己的程序有着深刻的理解，往往倾向于认为其设计是正确的。然而，如果在设计阶段就存在理解上的偏差，或因不良编程习惯而潜藏隐患，程序员本人往往难以察觉这类错误。

(5) 帕累托原则指出，在测试过程中发现的缺陷中，80%可能源自仅占20%的模块。例如，美国IBM公司的OS/370操作系统中，47%的缺陷仅与系统的4%的程序模块相关。因此，必须重视测试中缺陷集中的现象。若发现某一程序模块的错误频率明显高于其他模块，则应该投入更多的时间和精力进行针对性测试。

(6) 对测试结果的错误必须进行确认。通常，由工程师A发现的错误应由工程师B进行复核。对于严重错误，可以组织评审会进行讨论和分析。

(7) 制订严格的测试计划。应详细制订测试计划，并为测试工作预留充足的时间。切勿期望在极短的时间内完成高质量的测试工作。

(8) 完全测试是不现实的，测试必须有终止点。在有限的时间和资源条件下，试图找出所有软件缺陷和错误，使软件达到完美状态是不可能的。一个中等规模的程序，其路径组合数量接近天文数字，进行穷举测试——即对每一种可能的路径都执行一次测试是不现实的。即使能够进行穷举测试，也无法保证找到程序中所有潜在的错误。同时，成本将大幅上升，而漏掉的软件错误数量并不会因成本增加而显著减少。随着测试的推进，发现错误的成本也会逐渐增加。因此，应根据测试中发现错误的概率以及软件的可靠性要求，合理确定测试的终止时间，而不应无休止地进行测试。

(9) 重视回归测试的相关性。回归测试的相关性必须得到足够的重视，因为修复一个错误可能会导致更多错误的出现，这种情况并不罕见。

(10) 谨慎保存所有测试过程文档。保存所有测试过程文档的重要性不言而喻，测试的可复现性往往依赖于详尽的测试文档。

1.4.3 软件测试与软件开发各阶段的关系

在软件领域，人们常常关注软件开发与测试之间的相互作用。软件开发与测试是软

件生命周期中至关重要的环节，共同构成了软件过程的核心。开发过程主要负责软件的生产，而测试则确保软件的质量。若将软件开发类比于传统制造业，开发人员就像生产线上的工人，而测试人员则类似于质量控制专家。然而，软件测试与开发之间的联系更为紧密，测试在软件开发中扮演着不可或缺的角色。

软件开发遵循自顶向下的原则，逐步进行细化。在软件计划阶段，明确了软件的作用范围；在软件需求分析阶段，确立了软件的信息域、功能和性能需求以及各种约束条件；在软件设计阶段，则将设计思想转化为具体的程序代码，通常采用某种程序设计语言。

相对而言，测试过程则按照自底向上的顺序逐步集成。首先，对每个程序模块执行单元测试，以消除模块内部的逻辑和功能错误；接着进行集成测试，以发现并排除子系统或系统结构上的缺陷；然后进行验收测试，以确保软件满足设计要求；最终，从整个系统的角度进行软件运行，以验证其是否能够满足所有既定需求。软件测试与软件开发各阶段之间的关系如图1-7所示。

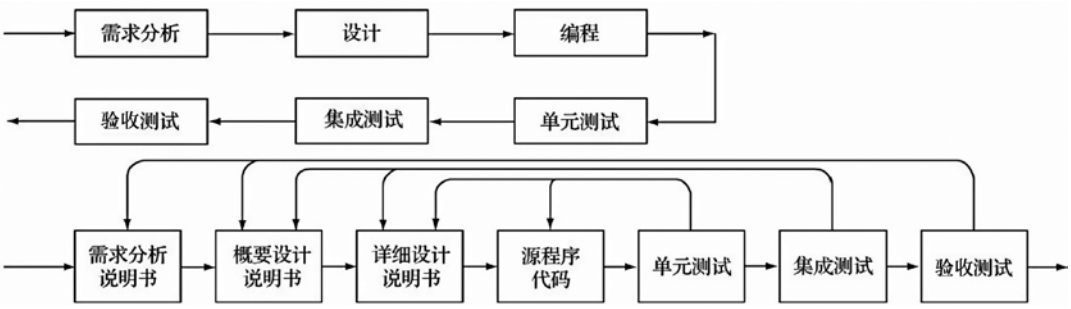

图1-7　软件测试与软件开发各阶段的关系

在软件测试的进程中，通过人工或自动化的执行方式对软件进行检验，并将观察到的行为特性与预期的行为特性进行细致比较。随着对软件测试方法、工具和技术的深入研究，测试的定义已经从一个简单的编程后评估过程，演变为软件开发生命周期中每个阶段不可或缺的活动。

1.4.4　软件测试过程模型

在软件开发的持续实践中，人们积累了宝贵的经验和教训，并对未来发展进行了预测，从而总结出多种开发模型。例如，典型的迭代改进模型、瀑布模型、快速原型模型、螺旋模型、增量模型等。然而，遗憾的是，这些模型中往往对测试环节没有给予足够的重视和充分的阐释。随着软件测试过程模型的出现，这一局面得到了改善。这些测试模型在考虑软件开发过程的同时，实现了开发与测试的有效融合。软件测试模型对于指导软件测试流程、提升测试质量和效率具有重要意义。因此，选择一个恰当的软件测试模型，对于确保测试流程的顺畅进行至关重要。

软件测试流程与软件开发流程一样，是一种抽象模型。它同样需要遵循软件工程原理和管理学原理。测试专家通过实践和不断改进，创建了众多实用的测试模型。这些模型明确了测试与开发之间的关系，使测试流程与开发流程产生互动，从而成为测试管理的重要参考。下面将对一些主要的测试模型进行简要介绍。

1. V模型

V模型作为具有显著代表性的测试模型，最初由Paul Rook于20世纪80年代末提出，旨在提高软件开发的效率与效果。该模型揭示了测试活动与分析设计活动之间的联系。如图1-8所示，它清晰地从左至右展示了基础的开发流程与测试行为，明确标示了测试流程中各种类型的测试，并详细阐释了测试阶段与开发流程中各个阶段的对应关系。图中的箭头指示了时间的流向，左侧下降部分代表开发流程的各个阶段，而与之对应的右侧上升部分则表示测试流程的各个阶段。

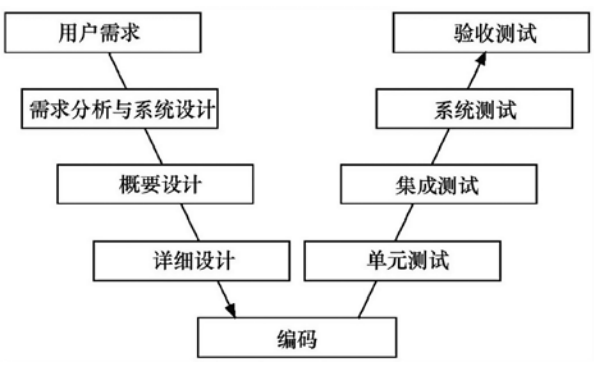

图1-8　软件测试V模型

V模型强调，单元测试与集成测试的宗旨在于验证程序执行是否符合软件设计规范；系统测试的目的在于评估系统功能和性能的质量特性是否满足既定的系统要求；验收测试则确保软件实现是否符合用户需求或合同规定。

然而，V模型也存在一定的局限性，它将测试活动仅视为编码之后的一个阶段，主要关注发现程序运行时的错误，忽略了测试在需求分析、系统设计等前期活动中的验证和确认作用。

2. W模型

W模型是由Evolutif公司提出的，它在V模型的基础上进一步强调了在软件开发过程中各个阶段应并行实施的验证与确认活动。验证活动的目的是通过数据来检验是否正确地执行了产品制造，主要关注过程的正确性；而确认活动则通过数据来确保制造了符合要求的产品，主要关注结果的正确性。W模型如图1-9所示，它由两个V型模型组成，分别代表了测试过程和开发过程，清晰地呈现了测试与开发的并行关系。

W模型强调测试应贯穿整个软件开发周期，测试的对象不仅限于程序，还包括需求和设计等环节，从而实现测试与开发的同步进行。W模型有助于在早期发现潜在问题。例如，在需求分析阶段完成后，测试人员应立即参与需求的验证和确认工作，以便尽早识别缺陷。同时，对需求进行测试也有助于及时评估项目难度和测试风险，从而提前制定应对策略，这将显著缩短总体测试时间，并加速项目进程。

然而，W模型也存在局限性。在W模型中，需求分析、设计、编码等环节被视为顺序进行，测试与开发活动之间维持线性的时间顺序关系，必须在前一阶段完全结束之后才能正式开始下一阶段的工作。这种模式无法适应迭代开发模型的需求。鉴于当前软件开发环境的复杂性和多变性，W模型并不能完全解决测试管理过程中面临的挑战。

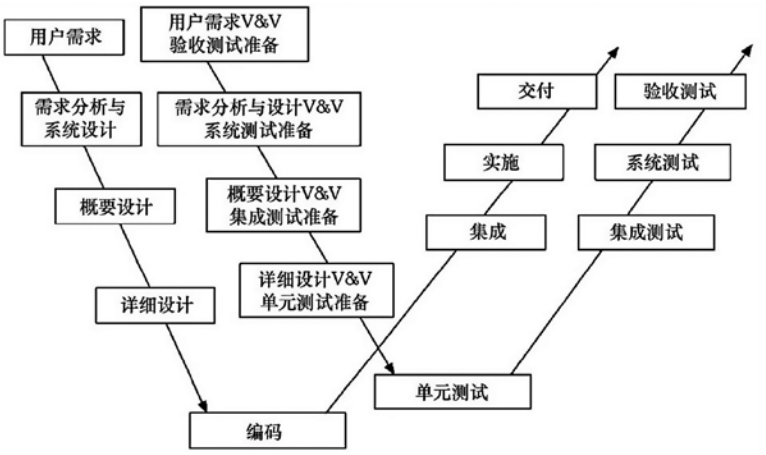

图1-9　软件测试W模型

3. X模型

在X模型的展示中(如图1-10所示)，左侧详细描绘了独立代码块的单独编码与测试流程。这些代码块随后将经历持续的交接过程，并通过集成步骤逐步构建出可执行的程序。这些程序将接受更深入的测试。一旦集成测试完成，产品即可封装并提供给用户，或成为更大规模集成的一部分。图1-10中并行的多条曲线表示变更可以在任何阶段随时发生。从图1-10中可以明显看出，X模型特别突出了探索性测试的重要性，这是一种无须预先规划的测试方法，通常由经验丰富的测试人员执行，旨在发现计划之外的软件缺陷。尽管如此，该测试方法可能导致资源的过度消耗，并且对测试人员的专业技能要求较高。

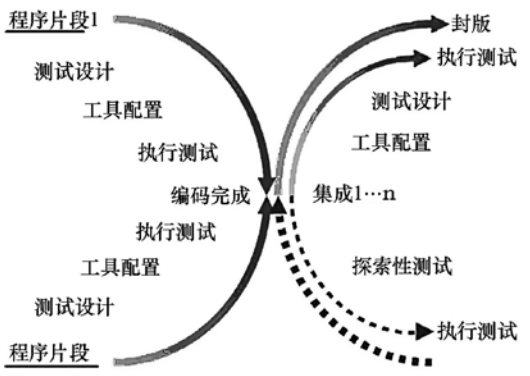

图1-10　软件测试X模型

4. H模型

V模型与W模型均存在一定的局限性。如前所述，这两种模型均将软件开发过程视为需求分析、系统设计、编码等顺序性活动，然而在实际操作中，这些活动往往能够并行推进。因此，相关的测试活动之间并不存在绝对的顺序性。同时，不同层级的测试(如单元测试、集成测试、系统测试等)也展现出反复和迭代的特性。为了应对这些挑战，专家们提出了H模型。该模型将测试活动彻底分离，形成一个独立的流程，并明确区分测试准备和测试执行两个阶段，如图1-11所示。

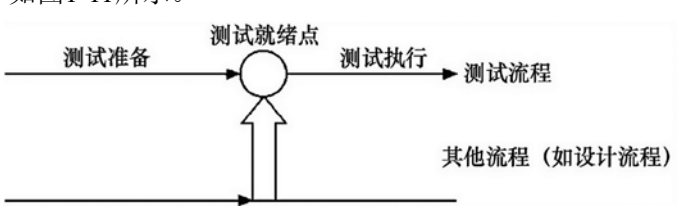

图1-11　软件测试H模型

图1-11展示了在生产周期的特定阶段进行的一次测试“微循环”。图中标识的“其他流程”可能代表任何开发流程，如设计流程或编码流程。换言之，一旦测试条件准备就绪，测试准备工作完成后，测试执行活动即可启动。

H模型揭示了软件测试是一个独立的流程，贯穿于产品的整个生命周期，并与其他开发流程并行执行。该模型强调软件测试应尽早开始并及时执行。不同的测试活动可以按顺序依次进行，也可以循环往复。只要某个测试活动达到准备就绪的状态，测试执行活动即可立即展开。

1.4.5 软件测试的分类

软件测试所涉及的技术与方法极为多样，可以根据不同的视角对其进行分类。

1. 按测试方式分类

根据测试方法的差异，软件测试可分为静态测试和动态测试两大类。

(1) 静态测试涉及在不执行程序的前提下，对代码是否遵循既定规则进行检查，并对程序的数据流及控制流进行详尽审查。

(2) 动态测试包括选取实际的测试用例以执行程序，旨在模拟真实的用户操作。

2. 按测试方法分类

根据测试方法的不同，软件测试可以分为白盒测试和黑盒测试两大类。

(1) 白盒测试。在对软件内部结构有深入理解的基础上，依据程序流程进行测试。白盒测试也被称为结构测试、透明盒测试、水晶盒测试、代码导向测试或设计导向测试。由于白盒测试需要投入较多资源，并且对测试人员的专业能力要求较高，因此通常仅对关键部分实施。

(2) 黑盒测试。依据软件规格说明书，针对系统应实现的功能进行测试。测试人员需对产品的设计概念有所掌握。黑盒测试也被称为行为导向测试、功能导向测试或需求导向测试。

(3) 灰盒测试。灰盒测试结合了白盒测试与黑盒测试的特征，是一种基于对程序内部结构有限了解的软件测试方式。测试人员可能对系统组件之间的交互有一定认识，但对程序内部功能和机制的了解并不深入。灰盒测试既关注输入与输出的正确性，也关注内部表现，但其关注程度不及白盒测试那样深入。测试人员通常通过一些关键现象、事件和标志来推断程序内部运行状态。灰盒测试有助于减少过度测试，从而简化多余的测试用例。

3. 按测试过程分类

在软件交付周期的各个阶段，软件测试是不可或缺的环节。依据不同的目标类型，整个软件测试过程可以细分为四个主要步骤，从独立程序模块的测试开始，一直到最终的验收测试。

(1) 单元测试。该测试阶段在开发初期进行，其主要目的是验证软件中最小的可测试单元。单元测试涉及对单一功能或子程序的测试，包括对每一行代码执行基础测试。它通常用于对模型中的最小组成单元如程序块、方块、函数等进行实施，以确保控制流和数据流得到全面覆盖，并且构件能够按照预期运行。测试内容包括界面测试、局部数据结构测

试、边界条件测试、覆盖条件测试以及错误处理等方面。

(2) 集成测试。集成测试是在单元测试完成后，按照设计要求将各个模块组装起来进行的测试。其主要目的是发现与接口相关的问题。测试重点在于验证模块间的数据传输是否准确无误，模块集成后的功能是否得到正确实现，以及模块接口功能是否与设计要求保持一致。集成测试通常紧随单元测试之后，一旦单元测试通过，即可开始准备集成测试环境。

(3) 系统测试。系统测试将被测试软件视为计算机系统的一个组成部分，与计算机硬件、外部设备、支持软件、数据和人员等其他系统元素结合，在实际运行环境中对整个计算机系统进行全面测试。该测试旨在全面发现系统错误，评估系统的整体性、可靠性和安全性。系统测试从客户或最终用户的角度出发，对系统进行全面评估。

(4) 验收测试。验收测试旨在确认被测试系统是否满足既定需求。测试重点在于评估产品在常规使用条件下的表现。主要由市场、销售、技术支持人员和最终用户根据既定需求共同进行有效性测试，以检验软件的功能、性能及其他特性是否符合用户要求。验收测试通常采用黑盒测试方法。其基本事项包括功能确认(根据用户需求规格说明，检测系统对规定功能的实现情况)和配置确认(检查系统资源和设备的协调情况，确保所有开发文档资料齐全，以支持软件运行后的维护工作) 。配置确认的文档资料包括设计文档、源代码、测试文档和用户文档等。

以上四个过程彼此独立且顺序相连，依次展开。首先，测试人员需单独完成每个单元的测试任务，以确保每个模块能够正常运行。单元测试主要采用白盒测试技术，旨在尽可能地发现模块内部的程序错误。一旦单元测试完成，测试人员将已测试的模块组合在一起，执行集成测试，其目的是检查与软件设计相关的程序结构问题。在这一阶段，更多地使用黑盒测试技术来设计测试用例。集成测试完成后，为了验证被测试软件是否能够与其他系统组件(如硬件、数据库和操作人员)协同工作，必须进行系统测试。最后，验收测试作为检验软件是否满足所有功能和性能需求的最终手段，通常采用黑盒测试方法，根据既定需求对开发初期制定的验收标准进行验证。

4. 按测试目的分类

测试可以根据其目的划分为多种类别。尽管存在超过三十种不同的测试类型，但在实际应用中，许多测试目的常常相互交错。根据测试目的进行分类，主要的测试类型包括以下几种。

(1) 功能测试。该测试专注于根据产品需求说明书对软件进行验证，确保软件功能与需求相一致。测试内容包括既定功能的检验，以及确认软件中是否存在多余或遗漏的功能。

(2) 健壮性测试。该测试着重于程序的容错能力，主要目的是验证程序在遭遇各种异常情况时是否能保持稳定运行。测试内容涵盖数据边界测试、非法数据测试、异常中断测试等。

(3) 接口测试。该测试涉及对各个模块进行系统联调的测试，包括程序内部接口和外部接口的测试。在这一过程中，测试人员会在单元测试阶段执行部分工作，而大部分工作则在集成测试阶段完成。

(4) 性能测试。该测试旨在检验系统性能是否满足用户需求，即在特定运行条件下评估系统的能力。性能测试通常利用自动化工具模拟正常、峰值和异常负载情况，对系统性能

指标进行评估。通过测试获得的负载和响应时间等数据，可以验证软件系统是否达到用户期望的性能标准。

(5) 强度测试。该测试属于性能测试的一种，通过在极端资源配置下运行系统来评估其性能。强度测试的目的是揭示因资源不足或资源竞争导致的错误。例如，当内存或磁盘空间不足时，测试对象可能会暴露出在正常条件下不明显的缺陷，这些缺陷可能是由于共享资源的竞争(如数据库锁或网络带宽) 引起的。如果一个系统在366MB内存下可以正常运行，但当内存容量减少后无法运行并提示内存不足，那么这个系统对内存的最低需求就是366MB。

(6) 压力测试。该测试作为一种性能测试，主要用于在超负荷环境下检验程序的运行稳定性。其目的是评估系统在资源超负荷状态下的表现，通过极限测试手段揭示系统在极端或恶劣条件下的自我保护能力。压力测试旨在确认系统在超过最大预期工作负载时仍能保持正常运行。此外，该测试还涉及评估软件的性能指标，如响应时间、事务处理速率以及其他与时间相关的性能参数。例如，在B/S架构中，用户并发量测试就属于压力测试的范畴。测试人员可以利用Webload工具模拟成百上千的用户同时访问网站，以观察系统的响应时间和处理速度。

(7) 用户界面测试。该测试专注于检验系统的界面设计，确保用户界面的友好性、软件的易用性、系统设计的合理性以及界面元素的正确布局。

(8) 安全测试。该测试着重于评估系统抵御未授权访问的能力。例如，测试系统在面对未经授权的内部或外部用户发起的攻击或恶意破坏时的运行状况，以及系统保护数据安全的能力。

(9) 可靠性测试。该测试旨在确保软件的可靠性水平满足用户需求，并验证软件是否达到软件规格说明书中规定的可靠性标准。通过分析在软件可靠性测试中收集到的失效数据，可以对软件当前的可靠性水平进行评估，并确认其是否符合要求。由于其高投入的特性，软件可靠性测试通常需要执行大量测试。

(10) 安装与反安装测试。安装测试主要检查软件是否能够顺利安装，安装文件的配置是否正确有效，以及安装后是否对整个计算机系统造成影响。反安装测试则是安装测试的逆过程，主要评估软件是否能够被彻底删除，以及删除后是否对计算机系统产生不良影响。

(11) 文档测试。文档测试主要评估内部和外部文档的清晰度与准确性。对于外部文档，测试工作侧重于用户文档，包括需求说明、用户手册和安装手册等，以确保文档内容与实际应用的一致性。同时，还需评估文档是否易于理解，技术术语是否得到了恰当的解释等问题。

(12) 恢复测试。恢复测试旨在检验系统在遭遇崩溃、硬件故障或其他灾难性问题时的表现，以及系统从这些故障中恢复的能力。

(13) 兼容性测试。兼容性测试着重于评估软件产品在不同平台、不同工具软件或相同工具软件不同版本下的兼容性。其目的是确保系统能够与其他软件和硬件设备良好地协同工作。

(14) 负载测试。负载测试通过模拟系统在资源超负荷状态下的表现，以识别设计缺陷或验证系统的负载承受能力。在此类测试中，测试对象将承受不同的工作负载，以评估其

在不同负载条件下的性能表现和持续运行的稳定性。负载测试的目标是确保系统在超出最大预期工作量的情况下，仍能保持正常运行。此外，负载测试还包括对性能指标的评估，例如响应时间、事务处理速率以及其他与时间相关的性能参数。

1.4.6　软件测试流程

软件测试流程贯穿了从测试的启动至完成的整个周期，涵盖了准备、执行和分析等一系列步骤。通常，软件测试活动需经历制订测试计划、设计测试、准备测试、执行测试、评估测试结果等关键环节。软件测试流程的详细步骤如图1-12所示。

接下来，将对测试流程的各个阶段进行详尽阐释。

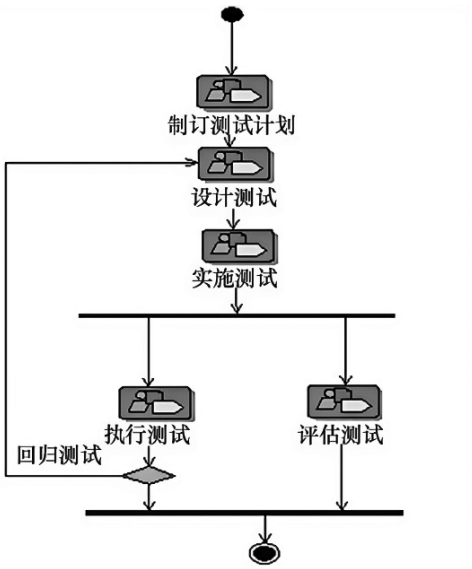

图1-12　软件测试的流程

1. 制订测试计划

制订测试计划是启动测试流程的首要步骤，其核心在于对整个项目的测试活动进行周密规划。测试计划并非单一的时间表，而是一个持续演进的过程，最终以一系列文档的形式固定下来。通常，制订测试计划旨在明确任务、分析潜在风险、规划所需资源并确立进度安排。

测试计划通常由测试主管或具备丰富测试经验的专家来编制。其主要依据是项目开发计划以及测试需求分析的结果。一个完整的测试计划通常包含以下几个关键部分。

(1) 软件测试背景。这部分详细介绍了软件项目的概况、项目团队成员(包括项目经理等)的介绍及其联系信息等。

(2) 软件测试依据。这部分包括了软件需求文档、软件规格说明书、软件设计文档等关键文件。

(3) 测试范围的界定。测试范围的界定指的是明确测试工作应覆盖的领域。在实际操作过程中，测试范围可能会根据项目进度的紧迫性进行调整。例如，在时间有限的情况下，优先考虑对关键功能进行测试。因此，测试计划制订者在接手任务时，需要依据项目计划的时间框架来确定测试范围。若在范围界定上出现偏差，将对测试执行产生负面影响。

在确定测试范围之前，管理人员需要对任务进行细分，这主要出于两个目的：一是识别各个子任务，二是便于评估所需的测试资源。在完成任务细分后，可以依据项目历史数据来估算完成这些子任务所需的时间和资源。通常情况下，执行一次全面的测试是不现实的，测试人员必须策略性地界定测试范围。

(4) 风险识别。在项目执行过程中，不可避免地存在诸多不确定因素，这些因素一旦出现，可能会对项目的顺利进行产生重大影响。因此，在项目开发的初期阶段，首要任务是

识别潜在的风险。风险识别的方法多种多样，但一个普遍适用的原则是：任何可能对项目进度产生显著影响的事件都应被视为一个潜在风险。识别出风险后，需要制定相应的策略来规避这些风险。

(5) 测试资源评估。明确完成测试任务所需的人力资源和物资资源，这包括但不限于测试设备、测试人员、测试环境以及其他相关资源的需求。

(6) 测试策略制定。测试策略涉及选择合适的测试方法、构建必要的测试环境、选用适当的测试工具和测试管理工具，以及对测试人员进行必要的培训。

(7) 时间表编制。在识别子任务并估算出所需测试资源后，可以将任务、资源与时间相结合，制定详细的测试时间进度表。

(8) 其他事项。测试计划应涵盖编写日期、作者信息等细节。尽管详尽的测试计划是理想状态，但在实际操作中，资源限制、人员变动等软件开发过程中的常见问题往往使得按原计划执行变得困难。这要求我们从宏观角度对测试工作进行调整和控制。然而，只要制订了周密的测试计划，测试人员就能在变动中保持镇定，灵活应对。

2. 设计测试

在测试的设计阶段，必须设计详尽的测试用例和测试流程，以确保测试用例能够全面覆盖所有测试需求。

测试用例是针对特定目标而设计的一系列测试输入、执行条件和预期结果的集合。这些特定目标可能包括验证特定的程序路径或确认某项功能是否满足特定需求。

设计测试用例涉及为特定功能或功能组合制定测试策略，并将其编写成文档。在选择测试用例时，不仅要考虑常规情况，还应包括极端情况和边界值情况。测试的主要目的是揭示应用软件中潜在的缺陷。因此，在设计和选择测试用例及数据时，应优先考虑那些有助于发现缺陷的测试用例和数据，并结合复杂的运行环境，在所有可能的输入和输出条件下确定测试数据，以验证应用软件是否能够产生正确的输出。

测试用例的完善并非一次性任务，它们源于测试需求。通常，测试人员会根据在不同阶段确定的测试需求设计测试用例，并随着开发过程的推进，根据测试需求的增补或修改，不断调整测试用例。评估测试用例质量的普遍认可标准有以下两个。

- 测试用例是否能够发现尚未发现的软件缺陷？
- 测试用例是否能够覆盖全部的测试需求？

测试流程通常分为多个阶段，包括代码审查、单元测试、集成测试、系统测试及验收测试等。尽管这些阶段在具体实施细节上存在差异，它们遵循的工作流程却是统一的。设计测试流程意味着构建测试的基本执行步骤，并为每个阶段的工作制定一个基础框架。

3. 测试准备和测试环境的建立

在筹备阶段，必须完成一系列测试前的准备工作，这包括全面而准确地掌握各类测试资料，深入理解并熟悉测试软件，配置测试所需的软硬件环境，搭建测试平台，以及充分掌握测试工具等任务。

测试环境的配置至关重要，一个符合标准的测试环境能够协助测试人员精确地识别软件中的问题并做出恰当的评估。不同的软件产品对测试环境有各自特定的需求。例如，针

对客户端/服务器(C/S)和浏览器/服务器(B/S)架构的软件产品，测试人员需要在多种操作系统环境下进行测试，包括但不限于Windows、UNIX、Linux以及苹果的macOS等，这些环境都是不可或缺的。而对于嵌入式软件(如手机应用)，若需评估特定功能模块的电力消耗或手机的待机时间，测试人员则需构建相应的电流测试环境。

构建测试环境的核心要素之一在于软硬件配置的精确性。唯有深入理解测试对象，才能明确每种测试对象所需的特定软硬件配置，从而搭建一个相对公正、合理的测试环境。在资源许可的条件下，建议构建一个满足待测软件运行所需的最小硬件配置环境。在配置测试的软硬件环境时，还需兼顾其他因素，例如操作系统、办公软件(如用于编写测试计划和规范的文字处理软件和电子表格软件)、视频设备、网络速度、显示分辨率、数据库权限、硬盘容量等。若条件允许，最佳做法是配置几组不同的测试环境。

测试准备是测试人员常常忽略的一个环节。在接到测试任务后，由于多种因素的影响，测试人员往往急于求成，立即投入具体的测试工作，忙于测试、记录和分析。然而，当工作进行到一半时，可能会发现硬件配置不符合要求、网络环境欠佳，甚至软件版本不正确，这些问题都可能对测试工作产生重大影响。这些无不源于测试准备不足。

4. 执行测试

进行测试的实施包括执行全部或部分选定的测试用例，并对结果进行细致观察。测试的执行流程可以进一步细分为以下阶段：单元测试→集成测试→系统测试→验收测试，每个阶段均可能涵盖回归测试等环节。

从测试的角度来看，执行测试不仅涉及数量，更关乎质量，即测试的广度和深度。具体而言，针对一个版本，我们需要测试哪些方面？每个方面又需要达到何种测试深度？执行测试的步骤可以概括为以下4个部分。

(1) 输入：完成工作所必需的初始条件。

(2) 执行过程：从输入到输出的转化过程或所承担的工作任务。

(3) 检查过程：评估输出是否满足既定标准的处理步骤。

(4) 输出：产生的可交付成果。

以程序员进行的单元测试为例，其步骤如下。

(1) 输入程序代码和相应的测试用例。

(2) 执行测试，生成特定产品或中间产品的可交付成果。

(3) 检查工作，确保产品或中间产品符合规范说明和既定标准。

(4) 若检查过程中未发现任何问题，则测试结果将被传递至下一个工作流程；反之，若发现问题，则产品将被退回以进行重新处理。

在测试执行阶段，根据所处的测试周期不同，工作内容会有所区别，主要体现在产品输入、测试策略、所使用的工具以及产品输出等方面。测试活动贯穿软件开发的整个流程。通常情况下，执行测试仅占测试工作总量的约40%。然而，鉴于测试工作通常需要迅速完成，因此常常需要通过长时间的连续工作来应对繁重的工作量。

在测试执行过程中，每个测试用例的结果都必须被详细记录。对于自动化测试，测试工具会自动记录输入数据和测试结果；而对于手动测试，测试结果则应记录在测试用例

文档中。在某些情况下，仅记录测试用例是否通过或失败就已足够。对于未通过的测试用例，需要生成相应的软件缺陷报告。需要注意的是，在测试执行期间，缺陷的记录和报告应当成为测试工程师日常工作的一部分。

5. 评估测试

测试评估的核心方法包括缺陷评估、覆盖评测以及质量评测。

(1) 缺陷评估。缺陷评估采用多种方法，这些方法既多样又广泛，涵盖从简单的缺陷计数到严格的统计建模等不同层面。严格的缺陷评估通常以测试过程中缺陷出现的比率或发现的比率来表示，常用模型假设该比率遵循泊松分布，实际的缺陷率数据也可以适用于这一模型。缺陷评估旨在评估当前软件的可靠性，并预测在持续测试或缺陷排除过程中可靠性将如何变化。缺陷评估也被称为软件可靠性增长建模，是当前研究领域中较为活跃的一个方向。

(2) 覆盖评测。覆盖评测是对测试完整性程度的评估，通过测试需求和测试用例的覆盖以及已执行代码的覆盖来表示。简而言之，测试覆盖是指基于需求(需求驱动)或代码设计/实施标准(代码驱动)对完整性程度的评估。

如果需求已经完全分类，基于需求的覆盖策略可能足以产生可量化的测试完整性评估。例如，若已确定所有性能测试需求，则可以引用测试结果来获得评估，如已验证了75%的性能测试需求。

在实施基于代码的测试覆盖策略时，测试的深度与广度是依据已执行的源代码量来衡量的。对于那些对安全性要求极高的系统，这种测试覆盖策略显得尤为重要。代码覆盖可以通过控制流(包括语句、分支或路径)或数据流来实现。控制流覆盖旨在检验代码行、分支条件、路径或软件控制流的其他关键元素，而数据流覆盖则侧重于通过软件操作来验证数据状态的有效性。

这两种评估方法均可通过手动方式完成，也可借助测试自动化工具来计算得出。

(3) 质量评估。质量评估着重于测试软件的可靠性、稳定性和性能，其依据是对测试结果的分析以及对测试过程中发现的缺陷进行深入分析。在评估测试对象的性能表现时，可采用多种评估方法，这些方法侧重于收集与行为相关的数据，例如响应时间、执行流、操作可靠性和限制条件。这些评估主要在“评估测试”阶段进行，但在“执行测试”阶段也可以利用性能评估来监控测试进度和状态。主要的性能评估方法包括动态监测、响应时间/吞吐量分析、百分位数报告、比较报告，以及追踪和配置文件报告。

6. 总结测试

在测试工作的各个阶段，都应撰写相应的测试总结报告。对于测试软件的每个版本，也应具备详尽的测试总结。测试活动结束后，通常需要对整个项目的测试过程进行回顾，分析存在的不足，总结可为未来测试工作提供借鉴的经验教训。尽管测试总结报告没有固定的格式和字数要求，但其重要性不容忽视。

制定合理的软件测试流程，不仅要求制定者具备深厚的软件测试理论知识，还必须拥有实际的软件测试执行经验、管理经验以及卓越的沟通能力等多方面的专业素养。此外，软件测试流程的完善性也需通过测试人员长期的实践来不断验证。

上述内容概述了测试工作的通用流程。实际上，每个公司或测试部门都有自己独特的测试方法和流程，这些方法和流程之间存在差异。

1.4.7　软件测试发展历程和发展趋势

软件测试的历史与软件本身的发展历程同样悠久，软件的产生和运行自然伴随着测试的不可或缺性。在软件开发的初期阶段，测试的范围相对狭窄，通常被视为“调试”的同义词，其核心目的是纠正软件中已发现的缺陷。在这一时期，测试工作通常由开发人员亲自承担，且对测试的重视程度并不高，测试往往在开发流程的后期才开始介入，通常是在代码编写完毕、产品基本定型之后才进行。

从20世纪50年代末到60年代，随着高级编程语言的出现和普及，程序的复杂性相应增加。然而，受限于当时的硬件条件，软件仍处于从属地位，软件正确性的验证主要依赖于程序员的个人技能。因此，在这一时期，测试理论和方法的发展相对迟缓。

到了20世纪70年代，随着计算机处理速度的迅猛提升以及内存和外存容量的显著增加，软件规模不断扩大，复杂性也急剧增加，软件在计算机系统中的作用日益凸显。这一时期见证了众多测试理论和方法的诞生，逐步构建起一套完善的测试体系。1979年，Glenford Myers所著的《软件测试艺术》成为该领域内极具影响力的著作。Myers将软件测试定义为：“测试是执行一个程序或系统以发现错误的过程”。Myers及其团队在20世纪70年代对软件测试领域的贡献是不容忽视的。

在20世纪80年代初，信息技术(IT)领域经历了飞速发展，软件产品逐渐向大型化和高复杂度演进，软件质量的重要性日益凸显。1982年，美国卡内基梅隆大学举办了首届软件测试技术大会，这标志着软件测试和软件质量研究者与开发人员首次齐聚一堂，该大会成为软件测试技术发展史上的一个转折点。紧接着在1983年，Bill Hetzel在其著作《软件测试完全指南》中提出：“测试是任何旨在评估程序或系统属性的活动，它是对软件质量的一种度量。”这一定义至今仍被广泛引用。同年，IEEE在软件工程术语中对软件测试的定义是：“使用人工或自动手段来执行或评估某个软件系统的过程，目的是验证它是否满足既定需求或揭示预期结果与实际结果之间的差异。”这一定义明确指出了软件测试的核心目的是验证软件系统是否符合需求。软件测试已不再是单一的、仅限于开发后期的活动，而是与整个开发流程紧密融合。软件测试已经发展成为一个专业领域，需要运用特定的方法和工具，由专业人员和专家来执行。

随着20世纪90年代的到来，软件产业迎来了空前的发展，软件项目的规模显著扩大。在众多大规模软件开发项目中，测试环节消耗了大量时间和资源。当时，测试工作几乎完全依赖于手工操作，效率极低；随着软件复杂性的提升，许多测试任务已无法仅靠手工方式应对。因此，许多测试专业人员开始尝试开发自动化测试工具，以支持和辅助测试人员完成特定类型或领域的测试任务，测试工具逐渐变得普及。人们普遍意识到，为了全面测试现代软件系统，自动化测试工具是不可或缺的，测试工具的选择和应用受到了越来越多的重视。测试工具的发展显著提高了软件测试的自动化程度。到了2002年，Rich和Stefan在《系统的软件测试》一书中对软件测试进行了更深入的阐述：“测试是为了评估和提升被测软件的质量，涉及软件设计、执行和维护的整个生命周期过程。”这些经典著作对软件

测试研究的理论化和系统化产生了深远的影响。

在过去的三十年间，计算机与软件技术的飞速发展显著推动了软件测试技术研究的进步。自1982年首次软件测试技术会议在美国卡内基梅隆大学召开以来，该学术会议每两年举行一次。同时，国际软件可靠性会议的规模、论文数量和质量也表明，从事软件测试的专业人员数量显著增加，软件测试技术的研究日益深入。

展望未来，软件测试技术与行业预计将经历更为迅速的发展。软件测试理论和技术将趋于成熟，测试效率有望逐步提升。更多实用的软件测试工具将涌现市场，测试工程师将获得应有的尊重，独立的软件测试部门也将成为更多软件公司的标准配置。然而，随着软件在社会各领域的影响力日益增强，测试任务的复杂性和工作量也随之增加。软件规模不断扩大，功能日益复杂，如何实施全面而有效的测试依旧是软件工程领域亟需深入探索的课题。此外，随着安全问题的凸显，如何对信息系统的安全性进行有效测试与评估，也成为了迫切需要解决的挑战。

1.4.8 软件测试人员的基本素质

软件测试是一项要求极为严格、复杂且充满挑战的工作。随着软件规模的持续扩大和复杂性的日益增加，软件公司已将软件测试视为技术工程中的一个专业职位。随着软件技术的不断进步，对专业化和高效率的软件测试需求日益迫切，对软件测试人员的基础技能和素质要求也在不断提高。简而言之，软件测试人员应掌握以下基本技能和素质。

1. 技能要求

测试人员的技能要求与开发人员截然不同。开发人员可能仅需掌握某种编程语言或开发工具便能胜任其工作，而测试人员则需具备更广泛的知识。测试人员的技能要求可以细分为以下4个主要类别。

1) 业务知识

测试人员对其所测试软件所涉及行业领域的了解称为业务知识。例如，许多IT企业专注于石油、电信、银行、电子政务和电子商务等行业领域的产品开发。掌握行业知识是测试人员成功完成测试任务的关键。只有深入理解产品的业务流程，测试人员才能准确判断开发人员实现的产品功能是否符合要求。测试人员对业务知识的了解越深入，其测试就越能贴近用户的实际需求。此外，测试中发现的缺陷往往也是用户最为关注的。反之，如果对产品涉及的业务领域缺乏理解，测试人员所发现的缺陷可能仅限于功能操作的正确性，甚至可能因为对某些业务知识的误解而导致错误的测试结果。

2) 计算机专业知识

掌握计算机领域的专业知识是测试工程师必须具备的基本素质，也是确保测试工作顺利进行的基础。对于希望拥有更广阔发展空间或持久竞争力的测试工程师来说，扎实的计算机专业技能是不可或缺的。计算机专业技能主要包括以下几个方面。

(1) 软件编程知识。软件编程知识是测试人员必备的技能之一(在微软公司，许多测试人员都具备多年的开发背景)。因此，为了实现更好的职业发展，测试人员必须掌握编程技能。只有具备编程能力，才能胜任单元测试、集成测试、性能测试等更为复杂的测试任务。

(2) 网络、操作系统、数据库和中间件等知识。鉴于测试过程中频繁地需要配置和调试各类测试环境，以及在性能测试环节还需对多样化的系统平台进行深入分析与优化，测试人员必须具备更广泛的网络、操作系统、数据库、中间件等相关知识。

3) 测试专业知识

测试专业知识涵盖广泛，不仅包括基础测试技术，如黑盒测试、白盒测试以及测试用例设计，还涵盖各种测试方法，例如单元测试、功能测试、集成测试、系统测试和性能测试。此外，测试流程管理、缺陷管理和自动化测试技术等也是测试人员必须掌握的重要知识。

4) 用户知识

测试工作应始终以用户和使用者的视角为出发点，而非仅从开发人员或实现者的角度思考。因此，测试人员必须深入了解用户的心理模型和操作习惯。若缺乏这些方面的知识，或思维方式出现偏差，将难以发现用户体验、界面交互、易用性和可用性方面的问题。这类问题虽然看似微小，却是用户极为关注的，有时甚至会成为决定产品成败的关键因素。

2. 素质要求

身为一名杰出的测试工程师，除掌握必要的专业技能和相关知识外，还应具备一些基本的个人素质。

(1) 必须拥有强烈的责任心、自信心，以及在工作中展现出的专注、细致和耐心。

- 责任心：作为完成工作不可或缺的品质之一，测试工程师尤其需要培养强烈的责任心。若在测试过程中未能尽责，甚至草率行事，那么测试任务可能会转嫁给用户，这可能导致极其严重的后果。
- 自信心：自信心是当前许多测试工程师所欠缺的品质，特别是在编写测试代码等任务面前，他们常常感到力不从心。为了职业发展，测试工程师们应当致力于持续学习，树立能够“解决所有测试难题”的自信。
- 专注：测试人员在执行测试任务时必须全神贯注，不可三心二意。经验表明，高度集中的注意力不仅能提升工作效率，还能帮助发现更多软件中的缺陷。在团队中，那些最能集中精力的成员往往业绩最佳。
- 细致：在执行测试工作时，必须细心谨慎，认真对待每一个测试环节，不可忽视任何细节。一些缺陷，如界面样式和文字错误等，若不细心则难以察觉。
- 耐心：许多测试工作可能显得单调乏味，需要极大的耐心才能做好。如果心态浮躁，就难以做到“专注”和“细致”，也难以敏锐地发现那些隐藏较深的软件缺陷。

(2) 测试人员必须具备出色的沟通与交流技巧。在测试过程中，他们需要与不同背景的人互动，包括技术团队成员(如开发者)和非技术人员(如客户和管理层)。这要求测试人员能够与用户建立良好的关系，同时与开发团队有效沟通。在分析故障报告和问题时，他们应能够清晰地表达自己的观点，并在必要时保持冷静与专业，以便与可能情绪化的开发人员协作。当遇到开发人员认为不需修复的软件缺陷时，测试人员应耐心解释缺陷的重要性，并通过实际演示来明确阐述自己的立场。这种能力有助于将冲突和对抗降至最低。

(3) 软件测试人员应展现出卓越的团队合作精神。在软件工程的开发模型和流程中，人员之间的协作是至关重要的。团队合作精神的贯彻执行，从根本上影响着项目开发的成功

与否。测试人员需要与开发人员紧密合作，共同确保项目的顺利进行。在规模较大的软件项目中，测试工作往往需要多名测试人员的参与，单打独斗无法应对复杂的测试任务。因此，所有测试人员必须齐心协力，共同推进工作。缺乏团队合作精神，测试工作将难以顺利进行。

1.5 本章小结

本章全面阐述了软件测试领域的多个方面，从软件危机的背景知识入手，探讨了软件质量的定义，以及软件测试的理论基础与实践应用，旨在为读者构建一个详尽的软件测试知识体系。通过案例分析与数据支持，本章突显了软件测试在保障软件质量方面的核心作用，并明确了软件测试从业者应具备的技能与素质。此外，本章还展望了软件测试的发展趋势，为未来软件测试技术的提升指明了方向。

1.6 思考和练习

一、填空题

1. 软件测试是软件工程中的重要部分，是确保软件质量的_____手段。
2. 本章的学习目标之一是了解软件缺陷和软件故障的概念，及相关_____和产生原因。
3. 敏捷模型采用_____的方式进行软件开发。
4. 敏捷软件开发旨在取代传统的_____模型。
5. 功能测试主要依据_____说明书对软件进行验证。

二、判断题

1. 软件测试只在软件开发完成后进行一次。 (　　)
2. 敏捷模型不适用于需求不明确或可能发生变化的项目。 (　　)
3. 压力测试的目的是评估系统在资源超负荷状态下的表现。 (　　)
4. 安装测试主要检查软件是否能够顺利安装，安装文件的配置是否正确有效。 (　　)
5. 软件可靠性测试通常不需要执行大量测试。 (　　)

三、简答题

1. 简述软件测试的重要性。
2. 什么是敏捷模型？
3. 解释什么是功能测试。
4. 为什么需要进行性能测试？
5. 简述软件缺陷和软件故障的区别。

第 2 章

软件测试计划

软件测试计划是软件测试工作的基础，它明确了测试的目标、范围、方法和重点，是项目启动初期必不可少的规划文档。测试计划对于确保测试工作顺利进行、增进项目成员之间的有效沟通、及早发现并修正软件规格说明书的问题，以及使测试工作更易于管理至关重要。在制订测试计划时，应遵循一些基本原则，例如尽早开始、保持灵活性、简洁易读、多方面评审和计算投入等。此外，本章还详细介绍了制订测试计划时应考虑的内容，包括测试资料的搜集与整理、测试目标的明确、测试方法的制定、测试项通过/失败的标准、测试进度的安排、资源需求、人员职责和培训需求，以及风险及应急措施等。最后，本章通过IEEE 829-1998标准的测试计划模板，给出了测试计划应包含的16个主要部分，为测试计划的编写提供了结构化的指导。

本章学习目标：

- 掌握软件测试计划的核心要素，理解其编制原则和方法，学会应用IEEE 829-1998标准模板，确保测试工作高效、有序进行。
- 能够独立编写科学合理的测试计划，从而提升项目质量，减少后期修正成本，确保软件产品按时交付并满足用户需求。
- 通过案例分析，帮助读者熟悉实际项目中测试计划的编制过程，将理论知识与实践相结合，提升解决实际问题的能力。
- 能够识别并规避常见的测试计划编制误区，优化资源配置，提升团队协作效率，确保测试工作的高效执行。
- 掌握如何根据项目特点灵活调整测试计划，提升应对突发状况的能力，保障软件产品的稳定性和可靠性。

2.1 软件测试计划的目的

软件测试计划是一份详尽阐述测试目标、范围、方法以及测试重点的文件。作为软件项

目计划的一个组成部分，它在项目启动初期至关重要。随着软件开发在众多企业中日益受到重视，软件质量的关注度也逐渐提高，测试流程逐步从一个相对独立的阶段转变为与软件生命周期紧密相连的活动。因此，如何规划整个项目周期内的测试工作，以及如何将测试提升至管理层面，都依赖于测试计划的制订。测试计划因此成为测试工作得以顺利进行的基础。

根据《IEEE软件测试文档标准829-1998》，测试计划被定义为：“一个详细描述预定测试活动的范围、方法、资源和进度安排的文件。它明确了测试项、测试特性、测试任务、人员分配，以及任何潜在风险。”软件测试计划是指导测试流程的纲领性文件，它需要包含所有必须完成的测试工作，包括测试项目的背景、目标、范围、方法、所需资源、进度安排、组织结构以及与测试相关的风险等关键信息。

通过软件测试计划，参与测试的项目成员，特别是测试管理人员，可以清晰地了解测试任务和方法，确保测试实施过程中的沟通顺畅，有效跟踪和控制测试进度，并应对测试过程中出现的各种变更。

具体而言，制订软件测试计划可以在以下方面为测试人员提供帮助。

1. 推动软件测试工作的顺利开展

一份周密的软件测试计划应详尽地包含将要执行的测试活动，涵盖所采用的模式、方法、步骤，以及可能遭遇的问题和风险。这种全面的规划有助于提高测试执行、分析和报告撰写的准备效率，确保软件测试流程更加顺畅。在测试的实际操作过程中，经常会出现一些问题，这些问题可能会阻碍测试工作的顺利进行。然而，许多问题实际上是可以提前预防的。测试计划还应包括潜在的测试风险，这些风险可能包括测试中断、设计规格频繁变更、人力资源短缺、人员流动、测试经验不足、测试进度压缩、软硬件资源不足以及测试方向错误等，这些都是难以预料的风险因素。对于测试计划而言，任何可能影响测试流程的问题都应纳入考虑范围。换言之，针对测试项目的推进应做好最坏的准备，并为这些潜在的最坏情况制定最佳的应对策略，以尽量规避风险，确保软件测试工作能够顺利进行。

2. 提升项目参与人员之间的沟通效率

测试工作需满足特定的先决条件。设想若程序员仅限于编写代码而不为代码添加注释，测试人员将难以完成测试任务。同样，若测试团队成员之间未能就测试对象、所需资源、进度安排等关键信息进行有效沟通，整个测试工作将难以顺利推进。测试计划将测试组织结构和测试人员的工作分配纳入考量，对测试工作进行了明确的划分，这有助于避免工作的重复和遗漏。同时，确保每位测试人员都明确自己应完成的测试任务，并在测试方向、策略等方面达成共识，从而使团队成员之间的沟通更为顺畅，确保沟通中不会出现偏差。

3. 提前识别并修正软件规格说明书中的问题

在软件测试计划编制的初期阶段，首要任务是深入理解软件各部分的规格和要求，这要求测试人员仔细阅读并领会规格说明书的内容。在这一过程中，测试人员可能会发现规格说明书存在的问题，例如论述上的前后矛盾或描述不完整等。越早发现并修正规格说明书中的缺陷，对软件开发的积极影响就越大，因为规格说明书始终是指导软件开发的基础。

4. 使软件测试工作更易于管理

制订测试计划的另一个重要目的是将整个软件测试工作系统化，从而提高其可管理性。

测试计划中包含了两种关键的管理策略：工作分解结构(Work Breakdown Structure，WBS)和监督控制。工作分解结构将所有测试任务细化，以便于测试人员分配工作。在执行软件测试时，管理人员可以运用有效的管理方法来监督和控制测试流程，确保测试进度的透明度。

综上所述，编写一份优质的软件测试计划书在测试开始之前至关重要。

遗憾的是，目前许多软件测试活动仍是在缺少测试计划的情况下进行的。这种“即兴应对”的方法使得测试人员处于不确定的境地，面对问题时往往采取临时性的应对措施，这不仅效率低下，还导致测试人员在相同问题上耗费大量时间，极大地消耗了精力。虽然这种情况下的软件测试也能发现错误和缺陷，但未经计划的测试对软件整体质量的影响令人担忧。实践证明，只有通过精心的规划和有效的管理，才能高效且高质量地完成软件测试工作。

2.2　制订测试计划的原则

制订测试计划是软件测试流程中极具挑战性的环节之一，遵循以下原则将有助于更高效地制订测试计划。

(1) 尽早启动测试计划的制订。即便对所有细节尚不完全掌握，也可以从总体框架入手，随后逐步细化以完成详尽的计划。提前开始制订测试计划有助于大致估算所需资源，并在项目其他部分占用这些资源之前进行测试。

(2) 确保测试计划的可适应性。在制订测试计划时，应考虑到能够轻松地增加测试用例和测试数据等元素，测试计划本身应具备一定的灵活性，但同时也要受到变更控制的约束。

(3) 保持测试计划的简洁性和可读性。测试计划无须庞大且复杂。实际上，越是简洁明了的测试计划，其针对性就越强。

(4) 尽可能地邀请多方面的人员参与测试计划的评审。来自不同背景的人员提供的评审和反馈，对于制订易于理解的测试计划非常有益。测试计划应像项目的其他交付物一样，接受质量控制的管理。

(5) 评估制订测试计划所需的工作量。通常，制订测试计划应占整个测试工作量的大约三分之一，测试计划的质量越高，后续的测试执行过程就会越顺畅。

2.3　如何制订软件测试计划

软件测试计划作为指导测试活动的纲领性文件，要求全面考量影响测试流程的众多因素。为了确保软件测试计划的有效性，需要特别关注以下几点。

1. 详尽地搜集与整理测试资料

搜集与整理测试资料是一项细致而复杂的任务。通常，除了从产品定义中获取资料，测试人员还经常需要直接向程序员询问产品的细节。因此，测试人员与程序设计人员的紧密合作对于提升产品质量至关重要。在测试工作中，除了通过与同事及上级主管的交流来了解与测试相关的人员、事件和工作环境外，还应重点关注技术信息的收集。技术信息主

要包括以下几个方面。

- 软件的类型及其架构。软件的类型及其架构涉及软件的类别与用途(不同类型的软件有不同的测试重点)、软件的结构、支持的平台，以及软件的主要组成部分、各自的功能以及彼此之间的相互联系。此外，还需了解每个组成部分所使用的编程语言信息。如果进行白盒测试，测试人员还需要熟悉各部分已构建的函数库中的函数，包括这些函数的用途、输入和输出值。
- 软件的用户界面。用户界面的风格可能是类似Windows的图形界面、命令行界面或网页类界面。测试人员需要掌握用户界面各部分的功能、相互联系，以及界面中各个组件的特性和操作特点等。
- 当测试的软件涉及第三方软件时，必须对第三方软件的功能及其与待测软件之间的交互有所了解。常见的第三方软件包括各种浏览器，如Internet Explorer、Chrome和FireFox等。

上述所有资料通常可以通过软件的规格说明书、设计说明书或向相关人员咨询来获得。在掌握了所有资料后，接下来的工作是对资料进行整理和归类。

此外，还需搜集和整理的其他信息，包括软件项目当前的主要问题、测试工作将使用的测试软件、缺陷报告软件，以及当前使用的版本控制软件。此外，还应了解哪些计算机专门用于测试，以及哪些关于该软件产品的信息可供参考等。这些信息一般可以通过测试部门的主管获得。

2. 明确测试目标，增强测试计划的实用性

测试目标应当具体明确，能够被量化和度量，而非含糊不清的宏观描述。此外，测试目标需要相对集中，避免列出一系列不分轻重的目标。通过分析用户需求文档和设计规格文档，可以确定被测软件的质量要求和测试应达成的目标。编写软件测试计划的主要目的是使测试工作能够揭示尽可能多的软件缺陷，其价值在于帮助管理测试项目，并识别软件潜在的缺陷。因此，软件测试计划中的测试范围必须全面覆盖功能需求，测试方法必须切实有效，测试工具必须具备高实用性和易用性，生成的测试结果必须直观且准确。

3. 遵循5W原则，确保内容与流程的明确性

5W原则中的每一个W分别代表What(执行何种任务)、Why(为何执行该任务)、When(何时执行任务)、Where(在何处执行任务)以及How(如何执行任务)。运用5W原则来构建软件测试计划，有助于测试团队深入理解测试的目的(Why)，明确测试的范围与内容(What)，设定测试的开始与结束时间(When)，指明测试的方法与工具(How)，并明确测试文档及软件的存放位置(Where)。

为了使5W原则更加具体化，测试团队需要精确掌握被测软件的功能特性、应用领域的专业知识以及软件测试技术。在测试计划中，应着重强调关键环节，分析测试中的风险、特性、场景以及所采用的测试方法。此外，测试人员还需为测试流程的各个阶段、文档管理、缺陷管理、进度管理提供切实可行的策略。

4. 通过评审和更新机制确保测试计划与实际需求相符

在测试计划编写完成后，若未经评审就直接交付给测试团队，可能会存在准确性问题或遗漏关键信息。这可能导致在软件需求发生变更时，测试范围的调整未能得到适当反

映，从而误导测试执行人员。鉴于测试计划内容的复杂性，以及编写者可能受限于个人的测试经验和对软件需求的理解，加之软件开发是一个逐步完善的过程，最初的测试计划往往存在不完整性，需要后续的更新。因此，实施相应的评审机制至关重要，它有助于对测试计划的完整性、正确性及可行性进行全面评估。例如，在测试计划初步完成后，可以提交给一个由项目经理、开发经理、测试经理和市场经理等组成的评审委员会进行审阅，并根据评审委员会的反馈意见和建议对计划进行修正和更新。

2.4　制订测试计划时面对的问题

在制订测试计划的过程中，测试人员可能会遇到以下几类问题。

(1) 意见分歧。测试人员与开发人员在测试工作的理解上往往存在对立，双方均可能认为对方试图占据优势。这种对抗性心态不仅会阻碍项目进展，消耗宝贵资源，还可能损害双方的合作关系，对测试工作本身无任何正面影响。

(2) 测试工具的匮乏。项目管理部门可能未能充分认识到测试工具的重要性，导致在测试工作中过度依赖人工操作，从而提高了测试成本并降低了效率。

(3) 培训不足。许多测试人员未接受过系统的测试培训，这可能导致他们对测试计划产生诸多误解，从而影响测试工作的质量。

(4) 管理层对测试工作的理解和支持不足。测试工作的成功需要管理层的有力支持，这不仅包括资金投入，还包括对测试过程中遇到的问题提供明确的指导和支持。缺乏这些支持，测试人员的工作积极性可能会受到打击。

(5) 用户参与度不足。用户可能被排除在测试流程之外，或者他们可能缺乏参与的意愿。实际上，用户在测试过程中的角色至关重要，他们能够确保软件产品满足实际需求。

(6) 测试周期的紧迫性。测试时间不足是测试团队常有的抱怨。关键在于如何合理安排测试计划的优先级，确保在有限的时间内完成必要的测试任务。

(7) 对测试人员的过度依赖。项目开发人员可能认为测试人员会负责检查他们的代码，因此可能只专注于编码，在代码质量上依赖测试人员。这种心态通常会导致更高的缺陷率和更长的测试周期。

(8) 测试人员面临的两难境地。一方面，如果测试人员报告了过多的缺陷，可能会被指责为项目延期的始作俑者；另一方面，如果未能发现关键缺陷，又会被质疑工作质量。

(9) 必须说“不”的挑战。对于测试人员而言，这是一种极为尴尬的处境，他们有时不得不拒绝项目相关方的要求。然而，项目相关方往往难以接受这个“不”字，因此测试人员在进度和成本的压力下，有时不得不妥协。

2.5　测试计划评估标准

确立测试计划的主要宗旨在于保证测试活动的有序性和目标性。在技术维度上，测试计划需明确测试的目标、范围和深度，并配备详尽的执行策略和测试焦点；在管理维

度上，则要求测试计划能够预估大致的进度和资源需求。一个卓越的测试计划应具备以下要素。

(1) 测试计划应能有效引导整个软件测试流程，确保测试团队与开发团队的协同工作，保障软件质量，并确保产品按时发布。

(2) 测试计划中所采用的方法应促进测试效率，即在较短时间内发现尽可能多的软件缺陷。

(3) 测试计划应明确阐述测试的目标、策略、具体步骤和测试标准。

(4) 测试计划需平衡测试重点与基础覆盖率，两者不可偏废。

(5) 测试计划应充分利用公司提供的资源，包括人力资源和物质资源，并确保方案的可行性。

(6) 测试计划中列出的所有数据必须精确无误，包括外部软件/硬件兼容性要求、输入/输出数据等。

(7) 测试计划应具备一定的灵活性，以便应对突发情况，如需求变更等。

上述内容概述了卓越测试计划应具备的关键要素。鉴于不同软件具有其独特性，测试计划的制订也应充分考虑这些特性。

2.6 制订测试计划

在规划测试计划的过程中，需要考虑到不同软件企业背景的差异，这些差异会导致它们所制订的测试计划文档呈现出一定的多样性。经验表明，采用标准化的文档格式来规划测试计划通常更加理想。为了便于参考，此处提供了一个基于IEEE 829-1998标准的软件测试计划文档模板，如图2-1所示。该模板详细规定了测试活动的范围、方法、资源分配、进度安排、待执行的测试任务以及每个任务的人员配置等关键要素。在实际应用中，可根据具体的测试需求对模板进行适当的调整。

IEEE 829－1998 软件测试文档编制标准

软件测试计划文档模板

目录

1.测试计划标识符
2.简要介绍
3.测试项目
4.测试对象
5.不需要测试的对象
6.测试方法（策略）
7.测试项通过/失败的标准
8.中断测试和恢复测试的判断准则
9.测试完成所提交的材料
10.测试任务
11.测试所需的资源
12.职责
13.人员安排与培训需求
14.测试进度表
15.风险及应急措施
16.审批

图2-1　软件测试计划文档模板

根据IEEE 829-1998软件测试文档编制标准的建议，测试计划应包含16个主要部分，下面将对这些部分进行介绍。

1. 测试计划标识符

测试计划标识符是一个唯一识别码，用于区分不同版本的测试计划、等级以及相关的软件版本等信息。

2. 简要介绍

测试计划的概述部分主要包括测试软件的基本信息及测试范围。测试软件的基本信息包括产品规

格(制造商及软件版本号)、运行平台、应用领域、软件特性以及主要功能模块的特点。此外，还需描述数据存储及传递的方式(通过数据流图展示)，各部分的数据更新机制，以及一些基础的技术要求(如所需的数据库类型)。对于大规模测试项目，测试计划还应明确测试的重点。测试范围的概述可以表述为：“本测试项目包括集成测试、系统测试及验收测试，但不包含单元测试，后者由开发团队负责，不属于本测试计划的范畴之内。”此外，概述中应列出所有经批准的相关文档、主要参考资料及其他测试依据文件，如项目批准文件和项目计划等。

3. 测试项目

测试项目应当详细记录被测试软件的名称及其版本，包括所有测试项目、外部条件对测试特性的影响，以及软件缺陷报告机制等相关信息。

测试项目部分的指导性描述应明确测试的范围，包括具体包含哪些内容，并制定一份详尽的测试项目清单。所有未包含在该清单中的工作均不纳入测试范围。这部分内容可按照程序、单元、模块进行组织，具体要点如下。

- 功能测试：从理论上讲，测试应覆盖所有功能项，例如在数据库中进行添加、编辑和删除记录等操作。尽管这是一项庞大的工程，但对确保测试的完整性至关重要。
- 设计测试：设计测试的目的是检验用户界面、菜单结构、窗体设计等方面的合理性。
- 整体测试：整体测试需确保数据在软件模块间流转的正确性。

根据IEEE标准，可参考以下文档来完成项目测试。

- 需求规格说明。
- 用户指南。
- 操作指南。
- 安装指南。
- 与测试项相关的事件报告。

总体而言，必须对软件的每一部分进行分析，明确其是否需要测试。若软件的某一部分未安排测试，则必须阐述不进行测试的原因。由于误解而未对某些代码进行测试，可能会导致无法发现软件潜在的错误或缺陷。然而，在实际操作中，有时也会选择不对软件产品中的某些内容进行测试，这些内容可能是之前已发布或已测试过的部分。

4. 测试对象

在测试计划的这一部分，应详尽罗列所有待测的单一功能及其组合。需要注意的是，此部分内容与测试项目有所区别。测试项目是从开发人员或项目管理者的视角出发，规划需要进行测试的项目。而测试对象则是从用户的角度出发，规划测试内容。例如，针对某台自动取款机(ATM)软件的测试，其中“需要测试的功能”可能包括取款功能、查询余额功能、转账功能，以及支付电话费和水电费等功能。

5. 不需要测试的对象

本部分指那些不纳入测试计划的单一功能或功能组合，并需明确阐述不进行测试的具体原因。对于某些功能的豁免测试，可能基于多种理由，例如这些功能可能暂时无法启

用，或者它们拥有稳定的运行记录等。然而，一旦某个功能被归入此列，通常意味着它被认为风险较低。这部分内容无疑会吸引用户的关注，因此必须谨慎解释决定不测试某些特定功能的详细原因。

同时，应注意测试进度的可能延后，这可能导致本部分内容的增加。如果风险评估已经明确了每个功能的风险等级，那么在测试进度延后的情况下，可以将那些风险最低的额外功能从“需测试的功能”列表转移到“免测试的功能”列表中。

6. 测试方法(策略)

测试方法是测试计划的核心内容。在测试计划中，需要对测试方法进行描述，并对每个阶段的测试策略进行详尽阐述。因此，许多软件企业更倾向于采用“策略”这一术语。

测试策略应详尽描述测试人员如何对整个软件及其各个阶段进行测试，包括如何确保测试的公正性和客观性。必须综合考虑包括模块、功能、整体、系统、版本、压力、性能、配置和安装在内的多个维度。策略应尽可能详尽，并准备测试记录文档模板，以充分支持即将开展的测试工作。关于测试记录的具体说明如下。

- 公正性声明：测试记录应明确阐述测试的公正性和遵循的标准，以证明测试的客观性。总体而言，软件功能应满足需求并与用户文档描述保持一致。
- 测试用例：测试记录中应详细描述测试用例的性质、所使用的工具及其来源，以及测试用例的执行方式和所用数据。同时，应为未来的回归测试留出空间，并考虑其他同时安装的软件可能对测试软件产生的影响。
- 特殊考虑：在某些情况下，需要针对外部环境的影响对软件进行特定方面的测试。
- 经验判断：可以参考以往测试中常见的问题，并结合当前测试的具体情况来制定测试方法。
- 设想：运用联想性思维，有助于发现新的测试途径。

测试的成功与否，主要取决于是否达到预期的测试覆盖率和精确性。因此，必须提供用于判断测试覆盖率和精确性的技术依据和评估标准。

决策是一项复杂的任务，应由经验丰富的测试人员来完成，因为这直接关系到测试工作的成败。此外，确保项目团队全体成员了解并同意预定的测试策略也极为重要。

7. 测试项通过/失败的标准

本部分需详细阐释“测试项目”中所涉及的每一项测试的合格与不合格判定标准。正如每个测试用例均应设定预期结果，每个测试项目也需要明确其预期成效。通常，合格与不合格的标准会依据测试用例的合格/不合格情况、缺陷的数量、类型、严重性以及缺陷出现的位置，还有系统的可靠性或稳定性等因素来确定。不同的测试阶段和组织机构可能会采用不同的具体标准。以下列出了一些测试项合格/不合格标准的常用指标。

- 测试用例的合格比例，即合格的测试用例与总测试用例的百分比。
- 缺陷的数量、严重程度以及它们在系统中的分布情况。
- 测试用例的覆盖范围，即测试用例覆盖了多少功能点或代码行。
- 用户对测试结果的满意度，即用户对测试成功与否的反馈。
- 文档的完整性，即相关文档是否详尽且准确地记录了测试过程和结果。

- 性能标准的达成情况，即系统是否满足了既定的性能指标。

8. 中断测试和恢复测试的判断准则

本部分详述了测试中断及恢复测试的标准，即在何种情形下应考虑中断测试。例如，当缺陷总数达到既定阈值或出现特定严重级别的缺陷时，测试活动应暂停。同时，对于恢复测试，也应制定明确的指导原则，例如在重新设计系统某部分或修正了错误代码后，如何重启测试流程。此外，本部分还需提供测试的替代方案以及在恢复测试前需要重新执行的测试项目。以下是中断测试的一些常见标准。

- 关键路径上存在未解决的任务。
- 缺陷数量显著增加。
- 出现严重级别的缺陷。
- 测试环境不完善。
- 资源供应不足。

9. 测试完成所提交的材料

提交的测试完成材料应包含测试过程中所涉及的所有开发和设计文档、工具等。具体而言，应包括测试计划、测试设计规格说明、测试用例、测试日志、测试数据、自定义工具、测试缺陷报告以及测试总结报告等。

10. 测试任务

本部分旨在详细说明测试前所需进行的准备工作以及必须执行的测试任务序列。同时，将详细列举所有任务之间的相互依赖性，以及完成这些任务所需的专业技能。在制订测试计划的过程中，通常会将这部分内容与“测试人员的工作分配”一并阐述，以确保每项任务均明确指定责任人。

11. 测试所需的资源

执行测试策略所必需的资源是确保测试成功的关键因素。在测试活动启动之前，必须制订一个全面的项目测试资源计划，该计划应详细列出每个阶段任务所需的具体资源。一旦发生资源过期或资源共享冲突等问题，应立即对计划进行更新。计划中应包含测试过程中可能涉及的所有资源。以下列举了一些测试中常见的资源需求。

- 人员配置：需评估测试团队的规模、成员的专业技能和经验水平。团队成员可能是全职员工、兼职人员、志愿者或学生。
- 设备规格：需考虑计算机、打印机等硬件的性能指标，包括机型要求、内存、CPU、硬盘的最低配置等。同时，还需明确设备用途，如计算机是否用作数据库服务器或Web服务器等。此外，还需考虑特殊限制，例如是否需要开放外部端口、限制某些端口或进行性能测试。
- 办公与实验室空间：需明确办公和实验室的具体位置、空间大小以及布局方式。
- 软件需求：包括文字处理软件、数据库管理软件、自定义工具以及每台设备上安装的自研软件和第三方软件的名称及版本号等。

- 其他资源：需考虑是否需要U盘、各类通信设备、参考书籍、培训材料等辅助工具。
- 特定测试工具：在特定测试工具方面，也需要进行详细的规划。

具体的资源需求会根据项目、团队和公司的具体情况而有所差异，因此在制订测试计划时，精确评估所需资源至关重要。通常，如果在预算初期没有得到妥善规划，项目后期获取额外资源将变得异常困难，甚至可能无法实现。因此，编制一份详尽的资源清单是至关重要的。

12. 职责

测试人员的职责必须明确界定，以确保每位测试人员都对自己的工作责任有清晰的认识。测试团队由众多成员构成，职责界定的模糊性将直接影响整个测试项目的进展，进而导致测试效率的降低。有时，测试任务的性质并不总是一目了然，它们不像编程任务那样易于界定，复杂的任务可能涉及多个执行者或需要团队协作完成。因此，建议参考表2-1所示的工作职责表，以清晰、直观的方式明确展示每位测试人员的具体职责。

表2-1　测试人员的工作职责表

任务	管理	编程	测试	技术作者	用户	产品支持
编写产品可视化说明	—				×	
建立产品部件清单	—			×		
建立联系	—			—		
产品设计/功能划分	—			×		
主项目进度	—			×		
制定和修改产品说明书	—			×		
审查产品说明书	×	—	—	—	—	—
内部产品体系化	—	×				
设计和编制产品		×		—		
测试计划	—		×			
审查测试计划	×	—		—		—
单元测试		×				
通用测试			×			
建立配置清单		—	—	×	—	—
配置测试			×			
定义性能指标	—		—			
执行指标测试			×			
内容测试			×	—		
外部代码小组测试		—	×			
建立β程序	×				×	
管理β程序	×					—
审查打印材料	×	—	—	—	—	—

(续表)

任务	管理	编程	测试	技术作者	用户	产品支持
定义演示版	—				×	
制作演示版	—				×	
测试演示版			×			
软件会议	×	—	—		—	—

在表2-1中，任务被置于左侧，而潜在的执行者则横向排列于表格顶部。符号“×”代表任务的执行者，而一字线“—”则代表任务的参与者。空白处则表明测试人员无须承担该任务。

在实际操作过程中，表格中应包含哪些任务，取决于制表者的专业判断。理想的做法是由小组中资深的测试人员对任务清单进行一次审核，以确保其准确性。然而，鉴于每个项目都具有其独特性以及每位测试人员的能力和特点各异，一个更为有效的方法是咨询经验丰富的测试人员，了解他们过去参与的项目情况，特别是那些容易被忽略的任务。

13. 人员安排与培训需求

前面介绍的职责旨在明确不同角色(例如管理、测试人员和程序员等)应承担的具体任务。而“人员安排与培训需求”则进一步细化，明确测试人员在软件测试工作中具体负责的环节以及他们必须掌握的技能。表2-2所示为一个极度简化的测试人员任务分配示例。在实际操作中，任务分配表应更为详尽，以确保软件的每个部分都得到充分的测试覆盖。每位测试人员都应明确自己的职责，并拥有足够的信息以开始设计测试用例。

表2-2　测试人员任务分配表

测试人员	测试人员任务
A	字符格式：字体、字号、颜色、样式
B	布局：项目符号、段落、制表位、换行
C	配置和兼容性
D	UI：易用性、外观、辅助特性
E	文档：联机帮助和滚动帮助
F	压迫和负荷

培训需求通常包括学习特定工具的使用、掌握测试方法、熟悉缺陷跟踪系统、了解配置管理以及掌握与被测试系统相关的业务基础知识。不同的测试项目往往具有独特的培训需求，这主要取决于各自项目的具体情况。

14. 测试进度表

测试进度表详细列出了测试活动的关键时间安排，评估了各项测试任务所需的时间，并提供了测试进程的时间规划。

测试进度的构建基于项目计划中的主要里程碑(如文档交付、模块交付、接口可用性设计等)。在测试计划的制订过程中，合理安排测试进度至关重要。许多在设计和编制阶段看

似简单易行的测试工作，在实际执行时往往需要大量时间。作为测试计划的一部分，完成测试进度的规划可以为项目管理者提供关键信息，从而更有效地安排整个项目的进度。

在实际的测试过程中，测试进度可能会不断受到项目中先前事件的持续影响。例如，如果项目中某一部分的交付比预定计划晚了两周，而按照原计划这一部分只有三周的测试时间，那么会发生什么情况呢？这可能导致将三周的测试压缩到一周内完成，或者将整个项目推迟两周。这类问题被称为进度危机。为了避免陷入进度危机，一种方法是在测试进度计划中避免设定具体的任务启动和结束日期，而是构建一个不包含具体日期的通用进度表。表2-3所示展示了一个可能导致测试团队陷入进度危机的测试进度表示例。

表2-3　设置固定日期的测试进度表

测试任务	日 期
测试计划完成	2024.3.3
测试用例完成	2024.6.1
第1阶段测试通过	2024.6.15~2024.8.1
第2阶段测试通过	2024.8.15~2024.10.1
第3阶段测试通过	2024.10.15~2024.11.15

相反，如果在测试进度表中对各测试阶段采用相对日期，那么测试任务的开始日期将取决于先前事件的交付日期，如表2-4所示。

表2-4　采用相对日期的测试进度表

测试任务	开始日期	期限
测试计划完成	说明书完成之后7天	4周
测试案例完成	测试计划完成	12周
第1阶段测试通过	编码完成	6周
第2阶段测试通过	Beta构造完成	6周
第3阶段测试通过	发布试用版本	4周

适当的进度规划对于简化测试流程的管理具有重要作用。在大多数情况下，项目管理员或测试管理员将承担进度安排的最终责任，而测试人员则负责规划各自具体任务的进度。

15. 风险及应急措施

在测试流程中，识别潜在风险和不利因素至关重要，并且需要制定相应的缓解策略。软件测试人员需清晰地识别计划阶段的风险，并与测试和项目负责人进行沟通。这些风险应在测试计划中明确记录，并在进度安排时予以考虑。尽管有些风险可能最终并未显现，但提前识别它们是必要的，以避免项目后期出现时的惊慌失措。通常，测试团队会发现资源有限，无法覆盖软件测试的所有方面。然而，识别风险有助于测试人员确定测试项的优先级，并集中精力关注那些可能导致严重后果的领域。

以下是一些软件测试中常见的潜在问题和风险。

- 设备和网络资源的限制可能导致测试不全面。应对策略包括测试人员明确指出缺少哪些资源以及这些限制可能带来的影响。
- 现场定制的开发模式和紧迫的上线时间可能导致测试不充分。测试人员应详细说明在这些限制条件下如何进行测试。
- 不切实际的交付日期。
- 系统间接口的不完善。
- 软件开发是一个逐步完善的过程，初始的测试计划可能不完整，需要定期更新。变更可能源于项目计划、需求、测试产品版本或测试资源的变动，这些都增加了测试的不确定性。
- 历史缺陷较多的模块。
- 经历过多次复杂变更的模块。
- 安全性、性能和可靠性问题。
- 难以变更或测试的特性。

尽管风险分析在初次尝试时可能面临诸多挑战，但在软件测试中执行风险分析至关重要，这一点必须铭记。

16. 审批

审批者应是拥有授权宣布测试工作准备就绪并可进入下一阶段的个体或团队。在测试计划的审批流程中，签名页扮演着重要角色。审批者在签署其姓名及日期时，需明确阐述其是否支持通过评审的立场。

前述内容概括了测试计划的基础框架。在实际撰写测试计划时，可以根据待测软件的特性、各测试部门的实际情况及条件，对前述要素进行适当的补充与调整，无须完全遵循固定模式。通常，制订一份详尽的测试计划需耗时数周乃至数月，并且这是一项需要全体测试人员共同参与的工作。

2.7　本章小结

本章详细介绍了软件测试计划的制订原则、方法及其重要性。软件测试计划构成了软件测试工作的基础，明确了测试的目标、范围、方法和重点，是项目启动初期必须规划的关键文档。本章强调，测试计划对于确保测试工作顺利进行、促进项目成员间的沟通、提前发现并修正软件规格说明书中的问题，以及使测试工作更易于管理，具有重要意义。同时，本章也提出了制订测试计划时应遵循的原则，例如尽早开始、保持灵活性、简洁易读、多方面评审和计算投入等。此外，本章还详细阐述了制订测试计划时应考虑的内容，包括测试资料的搜集与整理、测试目标的明确、测试方法的制定、测试项的通过/失败标准、测试进度的安排、资源需求、人员职责和培训需求、风险及应急措施等。最后，文档依据IEEE 829-1998标准的测试计划模板，列出了测试计划应包含的16个主要部分，为测试计划的编写提供了结构化的指导。

2.8 思考和练习

一、填空题

1. 软件测试计划应包含的16个主要部分是根据_____标准的模板给出的。
2. 制订测试计划时应遵循的一个原则是保持测试计划的_____性和可读性。
3. 在测试计划中，测试对象是从_____的角度出发规划测试内容。
4. 测试计划中应详细描述使用到的测试方法，并对每个阶段的测试策略进行_____阐述。
5. 测试计划中应明确测试项的通过/失败标准，通常依据测试用例的合格/不合格情况、缺陷的数量、类型、严重性以及缺陷出现的位置，还有系统的_____或稳定性等因素来确定。

二、判断题

1. 测试计划的制订可以完全依赖于测试人员的经验，无须遵循任何标准模板。 (　　)
2. 测试计划的制订应尽早开始，即使对所有细节尚不完全掌握，也可以从总体框架入手。 (　　)
3. 测试计划中不需要考虑测试进度的安排。 (　　)
4. 测试计划的制订应保持灵活性，但同时也要受到变更控制的约束。 (　　)
5. 测试计划的制订不需要考虑资源需求。 (　　)

三、简答题

1. 制订软件测试计划的目的是什么？
2. 制订测试计划时应遵循哪些原则？
3. 测试计划中应包含哪些关键要素？
4. 测试计划中如何处理测试中断和恢复测试的标准？
5. 测试计划的审批流程中，签名页扮演着什么角色？