高等院校计算机应用系列教材

软件测试技术
（微课版）

宰光军　刘　燕　主　编
石　磊　陈志坚　副主编

清華大學出版社
北　京

内 容 简 介

本书全面介绍了软件测试的相关技术。本书共分为10章，首先介绍了软件测试的基本概念，并基于SWEBOK V3对整个知识领域进行细致分解。随后介绍了黑盒和白盒软件测试方法，总结了测试方法的实施策略。接下来，本书详细阐述了软件测试的过程，包括各类软件测试过程模型。根据典型的软件测试过程阶段，分别介绍了单元测试、集成测试、系统测试、验收测试四个阶段。每个测试阶段采用不同的测试方法：在单元测试和集成测试中，主要使用前面章节讲解的白盒测试方法，而系统测试则常用黑盒测试方法。接下来，系统介绍了软件测试管理、软件测试工具与自动化。以敏捷开发为例介绍测试管理体系，并介绍了常用的项目管理软件、软件配置管理、缺陷管理等内容。最后，本书介绍了软件测试的相关领域(包括软件测试环境、容器技术、软件测试评估、软件质量保证等)，以及目前流行的人工智能和大数据技术在软件测试中的应用及相关知识。

本书在内容组织上力求条理清晰、内容丰富、语言流畅、通俗易懂，结合目前流行的技术趋势，使理论和实践能够有机地结合起来，更好地满足软件工程学科的特点。本书适合作为高等学校软件工程等计算机类专业的教材，也可以作为软件测试技术的培训教材。

本书配套的电子课件和习题答案可以到http://www.tupwk.com.cn/downpage网站下载，也可以扫描前言中的二维码获取。扫描前言中的视频二维码可以直接观看教学视频。

图书在版编目(CIP)数据

软件测试技术 : 微课版 / 宰光军 , 刘燕主编 .
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 978-7-302-69559-2

Ⅰ. TP311.55

中国国家版本馆 CIP 数据核字第 2025N5U148 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：三河市龙大印装有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**16.75　　**字　　数：**397 千字
版　　次：2025 年 8 月第 1 版　　**印　　次：**2025 年 8 月第 1 次印刷
定　　价：79.80 元

产品编号：104259-01

前言

随着大数据、人工智能、云计算等技术的迅猛发展，软件开发行业正经历着前所未有的变革。这些技术不仅极大地提升了软件开发的效率和质量，也迫使软件测试领域必须紧跟时代步伐，不断创新和进化，主要表现在以下几个方面。

1. 智能测试与人工智能的融入

人工智能和机器学习技术正被广泛应用于软件测试领域，涵盖了智能缺陷预测、测试用例自动生成、测试数据优化等方面。人工智能技术可以帮助测试团队更精准地识别潜在问题，优化测试资源分配，提高测试覆盖率和测试效率。

2. DevOps与测试左移

DevOps强调开发、测试、运维等团队之间的紧密协作，推动快速且高质量的软件交付。测试左移是DevOps理念在测试领域的具体实践，即在软件开发早期就引入测试活动，尽早发现并解决问题。这要求测试团队与开发团队紧密合作，共同制定测试策略，确保软件质量从源头开始把控。

3. 持续集成与持续交付(CI/CD)

CI/CD流程要求软件开发和测试过程高度自动化和集成化，实现代码的频繁提交、自动构建、测试和部署。这要求测试团队能够快速响应开发团队的变更，确保每次提交都能通过自动化测试，从而保持软件的高质量和高可用性。

4. 自动化测试的全面普及

自动化测试已成为现代软件开发流程中不可或缺的一部分。采用自动化测试，可以显著减少重复性工作，提高测试的效率和准确性，加快软件的持续集成和持续交付过程。自动化测试不仅限于单元测试、集成测试，还包括接口测试、性能测试、安全测试等多个层面，从而逐步构建一个全面、系统的自动化测试体系。

5. 云原生测试

随着云原生技术的兴起，越来越多的应用被部署在云平台上。云原生测试强调在云环境中进行测试，以验证应用在云环境下的性能、稳定性和安全性。这要求测试团队具备云计算相关的知识和技能，能够利用云平台的优势开展测试活动。

6. 安全性测试与合规性测试

随着网络安全和数据保护的重要性日益凸显，安全性测试和合规性测试已成为软件测试的重要组成部分。测试团队需要关注软件的安全漏洞和潜在风险，确保软件符合相关法律法规和行业标准的要求。

7. 性能测试与压力测试

在大数据和云计算环境下，软件的性能和稳定性对于用户体验至关重要。性能测试和压力

测试已成为评估软件质量的重要手段。测试团队需要模拟真实或极端的用户场景，对软件进行全面的性能测试和压力测试，以确保软件在高负载下仍能稳定运行。

因此，软件测试技术需要不断适应新技术和新模式的发展，通过自动化、智能化、持续集成和云原生等手段提升测试效率和质量，确保软件能够快速、稳定、安全地交付给用户。

在这一技术背景下，本书立足于软件测试的基础理论和知识体系，以SWEBOK V3提出的15个知识域为指引，分析了软件测试相关的学科，并按照不同的方法对软件测试进行了系统的分类。为了让读者更好地理解软件测试知识体系，本书还介绍了ISTQB、CSTQB、软件评测师、CSTE、LoadRunner ASP等流行的软件测试资质认证体系。

在软件测试概述的基础上，本书介绍了经典的黑盒和白盒测试方法，其中黑盒测试方法主要包括等价类划分法、边界值分析法、判定表、因果图、正交实验法、场景法、状态迁移法和错误推测法等，而白盒测试方法包括逻辑覆盖、基本路径测试、循环测试、程序插桩、域测试等。在软件测试过程知识体系中，除基本的软件测试过程模型和过程管理外，本书还介绍了国内开源的项目管理工具禅道，并针对新技术和新模式的发展，介绍了敏捷和DevOps测试。

在单元测试中，本书除介绍驱动程序、桩程序和Mock技术外，还介绍了流行的单元测试工具。集成测试部分则介绍了微服务架构的集成测试方法。在系统测试中，除基本的功能测试外，本书还特别介绍了泽众软件科技有限公司推出的性能测试工具。在安全性测试方面，本书将信息安全知识融入软件测试知识体系，例如基于故障注入的安全性测试、基于渗透的安全性测试等。

在可靠性测试方面，本书将可靠性工程引入软件测试领域，深入讲解可靠性模型。在易用性测试方面，本书将人机交互的软件工程引入软件测试领域，为易用性提供了更好的参考依据，并同时介绍了兼容性、本地化和验收测试。在软件测试管理方面，本书引入PMPOK作为知识领域的指引，并详细介绍了国际和国内相关的软件测试文档标准。

本书总结了软件测试工具的能力、分类和选择策略，并介绍了软件测试工具的研发技术。在自动化软件测试中，介绍了流行的自动化测试框架。在软件测试环境搭建过程中，介绍了主流的容器技术。本书还探讨了高质量编程与软件测试的关系，并提供了安全编程的建议。

在新技术应用领域，本书介绍了人工智能和大数据的测试，特别是人工智能领域的测试技术，包括对算法、数据集和性能的测试。同时，借助人工智能中的机器学习、深度学习和自然语言处理等算法，可以快速而精准地生成测试用例，显著提升测试的质量和覆盖率。

由于作者水平有限，书中难免有不足之处，恳请专家和广大读者批评指正。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示感谢。我们的电话是010-62796045，邮箱是992116@qq.com。

本书配套的电子课件和习题答案可以到http://www.tupwk.com.cn/downpage网站下载，也可以扫描下方左侧的二维码获取。扫描下方右侧的视频二维码可以直接观看教学视频。

配套资源

扫描下载

看视频

扫一扫

编　者

2025年3月

目录

第 1 章

软件测试概述

随着计算机系统的规模和复杂性的快速增长，软件开发成本以及因软件故障而造成的损失也在不断增加，软件质量问题已成为人们共同关注的焦点。软件是信息系统的核心，软件安全是确保信息安全的前提，而软件质量是信息系统的重要属性。软件测试是对软件产品或软件服务进行验证和确认的过程，是保障软件安全和确保软件质量的重要手段，贯穿于整个软件生命周期。随着软件系统规模和复杂性的增加，进行专业化高效软件测试的要求也越来越严格，软件测试职业的价值进一步得到了认可。软件测试技术作为一门新兴产业，正在迅速发展。本章将全面介绍软件缺陷、软件质量、软件测试定义、软件测试学科、软件测试目的、软件测试原则、软件测试分类和测试用例等内容。

本章的学习目标：

- 理解软件缺陷的定义和特点
- 理解软件质量的定义和特点
- 理解软件测试的定义和特点
- 理解软件测试的学科发展
- 掌握软件测试的目的、原则和分类
- 掌握测试用例的定义和特点

1.1 软件缺陷

1.1.1 Bug与软件缺陷

在第二次世界大战期间，第一代真空管计算机被广泛使用，其中一台名为Mark II的计算机代表了当时的顶尖技术。这种计算机通过控制电流来改变开关状态，从而实现控制功能。唯一不足的是，这些早期的计算机会产生大量的光和热。某天，天气非常炎热，工作

人员为了通风打开了窗户。然而，一只蛾子被光线吸引，飞入了机器内部，最终卡在了继电器的70号位置。这导致电路断开，计算机发生故障，无法得出正确的计算结果。

经过将近一天的检查，Grace Murray Hopper找到了故障的根源——那只蛾子。她使用发夹将蛾子取出，并将其尸体粘贴在自己的管理日志上，旁边写道：“就是这个Bug，害我们今天的工作无法完成。”(见图1-1)自此，在计算机科学中，“Bug”一词从“虫子”变成了“程序错误”的代名词，而“Debug”也成为调试修复错误的专业术语。Grace Murray Hopper是一位杰出的计算机科学家，她不仅赋予了“Bug”这个词新的含义，还领导了著名的计算机语言COBOL的开发。

图1-1　第一个Bug记录

与“Bug”这个词相比，更为正式的术语是软件缺陷(Defect)。IEEE 729-1983对软件缺陷提供了标准定义：从产品内部看，缺陷是软件产品开发或维护过程中存在的错误、问题等；从产品外部看，缺陷则是系统未能实现某个必需功能或功能失效。

通常认为，至少满足以下五条规则中的一条，就可以称为发生了软件缺陷。

(1) 软件未实现规格说明书中要求的功能。

(2) 软件超出规格说明书中指明的范围，例如实现了未提到的功能。

(3) 软件未达到规格说明书中规定的目标。

(4) 软件运行时发生错误。

(5) 软件难以理解、不易使用、运行速度慢，或最终用户认为使用体验不佳。

在一些书籍和开发文档中，软件测试执行过程中发现的问题通常称为Bug，而将软件需求和设计阶段引入的错误称为Defect(缺陷)，编码错误称为Error(错误)，将软件交付用户使用过程中出现的错误称为Failure(故障)。

1.1.2　软件缺陷的普遍性

“软件危机”是指在计算机软件开发和维护过程中遇到的严重问题，这些问题一直困扰着计算机系统的发展。随着互联网的发展，越来越多的设备连接到网络中，这些设备上运行的软件面临着越来越多的攻击者和更容易实施的攻击方式，导致软件面临的安全风险不断增加。1968年，北大西洋公约组织的计算机科学家在联邦德国召开的国际会议上首次

提出了“软件危机”的概念，并讨论了解决方案。在此次会议上，“软件工程”这一概念正式提出，并由此开创了一门新兴的学科——软件工程学，旨在研究和解决软件危机。

造成软件危机的重要因素是软件缺陷。历史上曾多次出现灾难级的软件缺陷事件，以下是其中一些典型的案例。

(1) 1988年，小罗伯特·莫里斯(Robert Morris，Jr.)发布了Internet蠕虫，首次暴露了互联网的脆弱性，并引发了对更高安全性的需求。他使用gets( )函数在Berkeley Unix finger守护程序中引发了缓冲区溢出，导致数千台计算机瘫痪。

(2) 1991年2月，美国在沙特阿拉伯部署的“爱国者”导弹系统未能成功拦截一枚来袭的“飞毛腿”导弹，造成了重大人员伤亡。经过深入调查，发现的原因是系统时钟存在软件缺陷——随着系统运行时间的延长，累积的时钟漂移会导致系统性能逐渐恶化。

(3) 1992年，部分驾驶员报告他们的丰田汽车出现意外的加速问题。调查发现，该问题源于硬件故障和软件缺陷的共同作用，其中涉及的软件问题包括缓冲区溢出、无效指针、竞争条件和堆栈溢出。

(4) 1994年，一位数学教授发现并公布了英特尔广受欢迎的奔腾处理器存在一个缺陷。英特尔回应称，只要用户能够证明他们确实受到了影响，便会根据要求更换芯片。英特尔表示，由缺陷引起的错误极为罕见，绝大多数用户不会察觉。然而，愤怒的用户要求英特尔为所有提出要求的人更换芯片，英特尔最终同意了。这一事件让英特尔公司损失了4.75亿美元。

(5) 1996年6月，阿丽亚娜5号火箭进行了首次飞行，即501航班。火箭发射后37秒自毁，导致任务失败，损失约3.7亿美元。事故的原因是数据从64位浮点值转换为16位带符号整数值时发生了整数溢出。

(6) 1999年，由美国国家航空航天局喷气推进实验室建造的火星气候轨道器项目以错误的角度接近了红色星球，导致了航天器的损毁。最终发现，工程团队的不同部门使用了不同的度量单位。一个研究推进器的部门使用的是英制磅力秒；另一个部门则使用公制牛顿秒。

(7) “千年虫”或“2000年问题”是指与2000年开始的日历数据的存储和格式化有关的事件。由于许多程序在表示四位数年份时只用最后两位数字表示，导致2000年和1900年无法区分，进而引发了系统故障。此问题影响到了银行、核电站、医院、交通运输等关键领域。为纠正这一错误，全世界耗费了数十亿美元来升级计算机系统。

(8) 2014年2月，苹果公司发布了有关SSL/TLS的安全更新。问题源于一行代码“goto fail;”，该行代码导致后面的语句无法执行。这一漏洞引发了大量投诉和经济损失。

(9) 2020年8月，花旗集团由于使用一个过时的软件系统造成了巨大损失。金融软件系统不及时进行维护和更新，将面临极大风险。首先，缺乏安全更新增加了黑客发现并利用安全漏洞的可能性。其次，软件系统未进行维护升级，容易与新操作系统、新设备以及第三方应用出现兼容性问题。

软件的实现过程是一个极其复杂的系统工程，几乎不可能做到零缺陷。通过软件测试能够尽早发现并修复这些缺陷，从而有效保障软件产品的质量。

1.2 软件质量

软件产品与其他产品一样，具有明确的质量要求。软件质量直接影响到软件的使用广泛程度与使用寿命。一款高质量的软件不仅能够满足客户的显式需求，往往还满足其隐式需求。

软件质量是指软件产品满足基本需求和隐式需求的程度。根据软件质量的定义，可以将其分为三个层次。

(1) 满足需求规定：软件产品符合开发者明确定义的目标，并且能可靠运行。

(2) 满足用户需求：软件产品的需求是由用户产生的，软件最终的目的就是满足用户需求，解决用户的实际问题。

(3) 满足用户隐式需求：除了满足用户的显式需求，软件产品如果能够满足用户的隐式需求(即可能需要在将来开发的功能)，将会极大地提升用户满意度，从而提升软件质量。

高质量的软件不仅需要满足上述需求，还应便于内部人员维护与升级。软件开发过程中，统一的编码规范、清晰合理的代码注释，以及详细的需求分析、软件设计和测试文档，都是后期维护与升级的重要保障。同时，这些资料也是软件质量的一个重要体现。

软件质量是使用者与开发者共同关心的问题，但全面、客观地评价一个软件产品的质量并不容易，它并不像普通产品一样，可以通过直观的观察或简单的测量来判断。目前，已有多项标准用于评价软件产品的质量。例如ISO/IEC 9126:1991国际标准，可作为评估软件质量的重要依据。

ISO/IEC 9126:1991是早期最通用的一个评价软件质量的国际标准。该标准不仅对软件质量进行了定义，还制定了软件测试的规范流程，包括测试计划的撰写和测试用例的设计等。ISO/IEC 9126:1991标准由6个特性和27个子特性组成，主要包括以下6个特性。

(1) 功能性：在指定条件下，软件满足用户显式需求和隐式需求的能力。

(2) 可靠性：在指定条件下使用时，软件产品维持规定性能级别的能力。

(3) 易用性：在指定条件下，软件产品被使用、理解和学习的能力。

(4) 效率：在指定条件下，相对于所有资源的数量，软件产品可提供适当性能的能力。

(5) 可维护性：软件产品被修改的能力，包括修正、优化和功能规格变更的说明。

(6) 可移植性：软件产品从一个环境迁移到另一个环境的能力。

2011年，ISO发布了国际软件质量评价标准ISO/IEC 25010:2011。与ISO/IEC 9126相比，25010将质量模型从原来的6个属性扩展到8个属性，新增加了安全性和兼容性两个方面。此外，该标准还对功能性、易用性和可维护性进行了修改。

2016年，国家标准化管理委员会发布了国家标准GB/T 25000，该标准由21个部分组成。其中GB/T 25000.10和GB/T 25000.51是建立软件测试技术体系的重要参考部分。GB/T 25000.51-2016《系统与软件工程 系统与软件质量要求和评价(SQuaRE)第 51部分：就绪可用软件产品(RUSP)的质量要求和测试细则》确立了就绪可用软件产品(RUSP)的质量要求，并规定了测试RUSP所需的测试计划、测试说明等文档要求，以及RUSP的符合性评价细则。GB/T 25000.51-2016涵盖软件产品的八大特性：功能性、性能效率、兼容性、易用性、可靠性、信息安全、维护性和可移植性。这些特性作为软件产品测评的主要依据和标准，如图1-2所示。

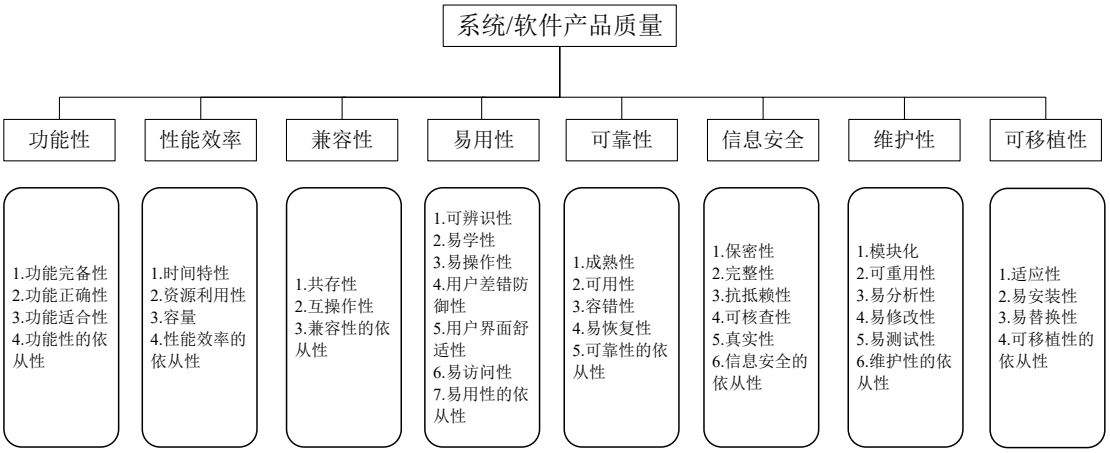

图1-2　GB/T 25000.51-2016涵盖的软件产品八大特性

1.3　软件测试定义

在软件测试的早期阶段，曾经出现过关于软件测试正反两方面的争论，代表人物是软件测试领域的两位先驱Bill Hetzel和Glenford J. Myers。正向思维认为，软件测试是顺应软件运行逻辑，以确保其最终能够正常运行为目标展开的各种活动。而反向思维则持怀疑态度，认为软件一定存在缺陷，基于这一出发点，反向思维认为软件测试是逆向推理软件的内在逻辑，以证明其存在缺陷为目标开展的各种活动。

1973年，Bill Hetzel给出了软件测试的定义：“软件测试是为了程序能够按照预期设想运行而建立足够的信心。”1983年，为了更清晰地描述软件测试，Bill Hetzel将软件测试的定义修改为：“软件测试是为了评估程序或软件系统的特性或能力，并且确定其是否达到预期结果的一系列活动。”上述两个定义代表了一种正向思维方式，认为软件测试的目的是验证软件是否能正常工作，也就是验证软件功能是否按照需求定义和软件设计的结果正确执行。

1979年，Glenford J. Myers出版了对软件测试行业影响深远的著作《软件测试的艺术》，他在书中提出了具有开创性意义的软件测试定义：“测试是为了发现错误而执行程序或系统的过程”。Myers认为，软件测试应当侧重于证明软件存在错误，应当用逆向思维的方式去发现尽可能多的错误。他强调，通过软件测试能够提高程序的可靠性和质量，而实现这一目标的途径是发现并修改程序错误。以发现错误为导向，才能更好地设计测试用例，从而有效地攻击系统的弱点，甚至摧毁系统，以此发现系统中的各种问题。另外，测试是为了证明程序存在错误，而不能保证程序没有错误，因为仍可能存在尚未发现的潜在错误。

1983年，IEEE给出了相对标准的软件测试定义：“使用人工或自动手段来运行或测定某个系统的过程，其目的是检验它是否满足规定需求，或弄清预期结果和实际结果之间的差异。”该定义明确说明了软件测试以检验软件是否满足需求为目标。预期结果和实际结果之间的差异，即软件错误，是测试过程的结果，而不是最终目标。发现软件错误只是实现这一目标的手段。

软件测试可以被定义为由验证(Verification)和确认(Validation)活动构成的整体。该定义反映了对软件测试的广义理解，有别于只把软件测试看作代码实现后的一项软件工程活动的狭义理解。软件测试贯穿于整个软件生命周期，需求评审和设计审查等软件质量保证活动都属于软件测试的范畴。测试对象不仅限于程序，还包括各类软件开发文档。

验证是指检验软件生命周期的每个阶段和步骤的产品是否符合产品规格说明中定义的功能和特性要求，并且与前面的阶段和步骤所产生的产品保持一致性。验证通过数据和证据来确认每个软件生命周期活动是否已正确完成，判断软件产品是否在正确地开发，以及是否可以开始后续的生命周期活动，重点在于过程的正确性。1981年，Boehm给出了有关验证和有效性确认的简洁解释，说明了两者之间的区别。

- 验证(Verification)：Are we building the product right？即是否正确地构造软件，检验软件开发过程中阶段性产品与软件规格说明书的一致性。
- 确认(Validation)：Are we building the right product？即是否构造了符合用户需求的正确软件产品。

1.4 软件测试学科

1.4.1 软件测试的发展历程

在20世纪50年代，软件规模较小，开发过程相对简单，因此软件测试基本等同于调试。然而，到了1957年，软件测试开始与调试区别开来，逐渐被视为一种发现软件缺陷的活动。由于长期存在着“为了让我们看到产品在工作，测试工作往往被推迟”的思想，测试仍然是紧随开发之后的活动。在潜意识中，测试的目的是使自己确信产品能够正常工作。

20世纪60年代中期到70年代中期，软件行业取得了快速发展，软件开始作为一种产品被广泛使用，这个时期出现了“软件作坊”。然而，随着软件数量的急剧膨胀和需求的日益复杂，开发成本也越来越高。随着用户对软件的要求越来越高，市场对软件质量的各个方面提出了新的要求，但是软件开发的管理水平却没有跟上，导致开发成本越来越高，最终引发了软件危机。

1968年北大西洋公约组织在联邦德国召开国际会议，会议上正式提出了“软件工程”这一名词，标志着软件工程学科的诞生。随着软件开发在软件工程方法指导下不断正规化，软件测试理论和方法也不断完善。1972年，北卡罗莱纳大学举办了历史上首届正式的软件测试会议，标志着软件测试作为一个学科正式诞生了。1973年，William C. Hetzel整理出版了软件测试的第一本著作*Program Test Methods*，对测试方法和测试工具进行了论述。1975年，Goodenough和Gerhart首次提出了软件测试的理论，使得软件测试成为具有理论指导的实践性学科。1979年，Glenford Myers在其著作《软件测试艺术》(*The Art of Software Testing*)中给出了当时较为准确的软件测试定义：“测试是为发现错误而执行程序或系统的过程。”

20世纪80年代早期，“质量”的号角开始吹响。软件测试的定义发生了改变。测试不再仅仅是一个发现错误的过程，而是开始涵盖软件质量评估的内容。软件开发人员和测试人员开始坐在一起探讨软件工程和测试问题。各类标准相继制定，包括IEEE(Institute of Electrical and Electronic Engineers)标准、美国ANSI(American National Standards Institute)标准以及ISO(International Organization for Standardization)国际标准。1983年，Bill Hetzel在《软件测试完全指南》(*Complete Guide of Software Testing*)一书中指出：“测试是以评价一个程序或系统属性为目标的任何活动，是对软件质量的度量。”Myers和Hetzel的定义至今仍被引用。

20世纪90年代，测试工具逐渐盛行。人们普遍意识到，工具不仅仅是有用的，而且对于充分测试软件系统必不可少。1996年，提出了测试能力成熟度模型(Testing Capability Maturity Model，TCMM)、测试支持度模型(Testability Support Model，TSM)和测试成熟度模型(Testing Maturity Model，TMM)。2002年，Rick和Stefan在其著作《系统的软件测试》(*Systematic Software Testing*)中对软件测试做了进一步定义：“测试是为了度量和提高被测软件的质量，对测试对象进行工程设计、实施和维护的整个生命周期过程。”这些经典论著对软件测试研究的理论化和体系化产生了巨大的影响。

1.4.2　软件工程与软件测试

1968年，NATO(北大西洋公约组织)的科技委员会召集了近50名顶尖的编程人员、计算机科学家和工业界巨头，讨论并制定摆脱“软件危机”的对策。在那次会议上第一次提出了软件工程(Software Engineering)这个概念。软件工程是一门研究如何用系统化、规范化、数量化等工程原则和方法进行软件开发和维护的学科。

软件工程包括两方面内容：软件开发技术和软件项目管理。软件开发技术包括软件开发方法学、软件工具和软件工程环境。软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划、软件测试等内容。此后，软件工程理论开始蓬勃发展，提出了许多软件开发模型，比如瀑布模型、增量开发、螺旋模型和敏捷开发等。软件工程的发展趋势如图1-3所示。

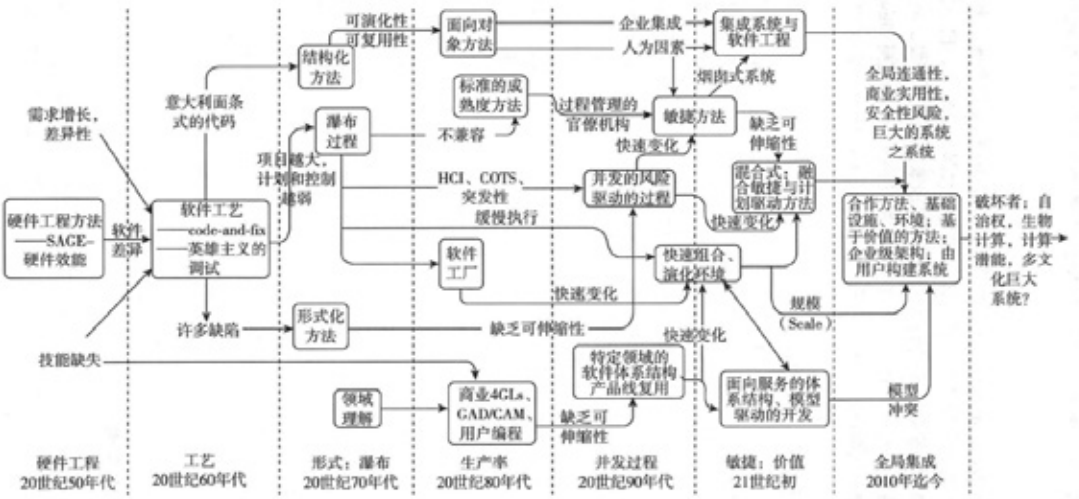

图1-3　软件工程发展趋势图

软件测试学科的发展始终围绕着软件工程学科。IEEE-CS和ACM联合组建的软件工程协调委员会(SWECC)发布了软件工程知识体系和推荐实践的SWEBOK 2004，为软件工程职业实践建立了适当的准则和规范。基于SWEBOK，SWECC进一步定义了可纳入教育项目的知识体系，包括本科生软件工程教育计划SE2004中的SEEK、研究生软件工程教育计划GSwE2009中的CBOK、软件工程职业道德规范和职业实践。

SWEBOK的最新版本V3共包括15个知识域，其中包含11个软件工程实践知识域，分别是软件需求、软件设计、软件构造、软件测试、软件维护、软件配置管理、软件工程管理、软件工程过程、软件工程模型和方法、软件质量、软件工程职业实践，以及4个软件工程教育基础知识域，包括软件工程经济学、计算基础、数学基础和工程基础。软件工程的理论基础主要是计算机科学中的程序理论和计算理论，以及求解问题的数学理论与方法。这些理论不仅关注构造软件的理论、模型与算法及其在软件开发与维护中的应用，也关注求解问题的数学理论与方法及其在软件建模、分析、设计和验证中的应用。

国内软件工程作为一个独立的一级学科，代码为0835，与计算机科学与技术、管理科学、数学等其他一级学科紧密相连，是基础性和技术性并重的新兴学科。软件工程所包含的二级学科如下。

(1) 083501软件工程理论与方法：在计算机科学和数学等基本原理的基础上，研究大型复杂软件开发、运行和维护的理论和方法，以及形式化方法在软件工程中的应用，主要包括软件语言、形式化方法、软件自动生成与演化、软件建模与分析、软件智能化理论与方法等内容。

(2) 083502软件工程技术：研究大型复杂软件开发、运行与维护的原则、方法、技术，以及相应的支撑工具、平台与环境。主要包括软件需求工程、软件设计方法、软件体系结构、模型驱动开发、软件分析与测试、软件维护与演化、软件工程管理，以及软件工程支撑工具、平台与环境等内容。

(3) 083503软件服务工程：研究软件服务工程原理、方法和技术，构建支持软件服务系统的基础设施和平台，主要包括软件服务系统体系结构、软件服务业务过程、软件服务工程方法、软件服务运行支撑等内容。

(4) 083504领域软件工程：研究软件工程在具体领域中的应用，并在此基础之上形成面向领域的软件工程理论、方法与技术，主要包括领域分析、领域设计、领域实现、应用工程等内容。

虽然软件测试不是一级学科，但从技术和学科的角度来看，软件测试是软件工程的重要组成部分，是提升软件产品质量的重要过程。国内大部分高等院校的软件工程专业教学计划中，把“软件测试”或“软件质量保证与测试”等课程作为核心的专业课程。部分院校还将“软件测试”作为“软件工程”课程的独立章节进行专门讲授。

1.4.3 软件测试学派

近几年，敏捷测试、探索式测试、精益测试和基于模型的测试等方法越来越受到关注。《软件测试：经验与教训》一书的作者Bret Pettichord在2003年将软件测试归纳为四大学派。四年后，他又增加了一个敏捷测试学派，将软件测试分为以下五个学派。

(1) 分析学派(Analytic School)：该学派认为软件是逻辑性的，将测试看作计算机科学和数学的一部分。结构化测试和代码覆盖率就是其中一些典型的例子。分析学派认为测试工作是技术性很强的工作，侧重使用类似UML工具进行分析和建模。

(2) 标准学派(Standard School)：该学派从分析学派分支而出，并得到IEEE的支持。它将测试看作侧重于劣质成本控制并具有可重复标准的工作，旨在衡量项目进度。标准学派认为测试是对产品需求的确认，每个需求都需要得到验证。

(3) 质量学派(Quality School)：该学派主张软件质量需要规范，测试就是过程的质量控制活动，能够揭示项目质量风险。该学派认为测试的目的是确保开发人员遵守规范，测试人员在此过程中扮演产品质量的守门员角色。

(4) 上下文驱动学派(Context-Driven School)：该学派认为软件是人创造的，测试所发现的每一个缺陷都和相关利益者密切相关。它认为测试是一种有技巧的心理活动，强调人的能动性和启发式测试思维，其中探索性测试是其典型代表。

(5) 敏捷学派(Agile School)：该学派认为软件开发就是持续不断的对话，而测试就是验证开发工作是否完成。敏捷学派特别强调自动化测试，TDD(Test-Driven Development)是其典型代表。

标准学派和质量学派相对比较成熟，相关流程、过程规范等基本已建立，例如TPI和TMMi等模型，虽然未来会有一些修改。而上下文驱动学派则是比较自然的思路，其他学派或多或少也会从上下文去考虑，也存在融合的可能性。虽然分析学派和上下文驱动学派、敏捷学派有一定对立关系，但它们相互之间又会有更多的交融。敏捷方法主要以实践为基础，敏捷测试不是原发性的，而是源于敏捷开发。开发者被动地寻求测试方法和技术来适应敏捷开发。敏捷测试缺乏自己独立的理论根基，更多地依赖于上下文驱动学派的支持，包括探索式测试和自动化测试。其中，自动化测试是敏捷测试主推的，没有自动化测试就没有敏捷测试，而自动化测试与持续集成和持续测试也高度契合。

对软件测试影响最大的因素是软件发布模式和软件开发技术。在SaaS模式中可以实现持续发布，从而使敏捷测试和探索式测试受到更多的关注。同时，SaaS的发展促进了各种基于云计算的服务模式诞生，软件测试的云服务模式应运而生并快速发展起来。测试公有云提供公共的开放测试服务，例如UTest、SOASTA、SauceLab和Testin等，这些平台可以完成手机应用、Web应用或其他应用的功能测试、兼容性测试、配置测试和性能测试等。而测试私有云是某个企业为自己建立的云测试服务，将测试机器资源、测试工具等都放在云端，公司的各个团队都可以共享所有测试资源，完成从自动分配资源、自动部署到测试结果报告生成的测试过程，并且还能将测试流程和测试管理等固化在私有云中。

1.5　软件测试目的和原则

1.5.1　软件测试目的

基于不同的立场，存在着两种完全不同的测试目的。从用户的角度出发，用户普遍希

望通过软件测试暴露软件中隐藏的错误和缺陷，以评估是否可以接受该产品。而从软件开发者的角度出发，则希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。因此，开发者往往会选择那些导致程序失效概率小的测试用例，避免使用那些易于暴露程序错误的测试用例，并且不太关注检测和排除程序中潜在的副作用。

关于软件测试的目的，Glenford J. Myers提出了如下观点。

- 软件测试是程序的执行过程，其目的是发现错误。
- 优秀的测试用例很可能会发现至今尚未发现的错误。
- 成功的测试是那些发现了至今尚未发现错误的测试。

尽早且尽量多地发现被测对象中的缺陷，是测试人员测试过程中最常提起的一个测试目标，也是测试价值的重要体现。发现错误是软件测试的过程之一，但不是软件测试的唯一目的。因此，软件测试的目的也包含以下内容。

1. 提升软件质量

软件测试最直接的目的就是提高软件的质量，让用户对产品有较好的体验，并保障产品的高质量。发现缺陷的目的是推动开发人员定位和修复问题。测试人员通过再测试和回归测试，确保开发人员已修复缺陷，并确认修复过程没有影响原来正常的功能区域，从而提高产品质量。开发生命周期的每个阶段，都应该有测试的参与，以尽可能多地发现各阶段的缺陷，这样可以大大提高该阶段的缺陷控制能力，这样可以提高测试效率、降低成本并提高质量。

2. 提供信息

测试过程的每个阶段都在为开发过程提供信息，包括给软件产品的不同成员提供不同维度不同详细程度的信息。提供信息的主要目的是帮助项目组做出正确的决策。

(1) 质量评估：通过测试过程提供的各种数据，可以帮助项目经理评估被测软件产品的质量。例如根据测试过程中发现缺陷的累积趋势、测试执行的进度数据、执行通过率和覆盖率等指标，可以判断软件产品是否满足计划中定义的质量要求。

(2) 进度评估：通过提供的各种数据，可以帮助项目经理判断软件产品是否能够及时发布。这包括评估测试执行进度是否在计划范围内，以及修复缺陷进度是否满足质量标准和发布要求等。

在评估产品质量和进度时，测试过程中提供的数据是非常重要的输入。

3. 预防缺陷

在测试过程中发现的缺陷，以及遗漏到用户现场的缺陷，都应该对它们进行缺陷根本原因分析，以确定引入缺陷的主要原因。从测试角度也要分析为什么能发现缺陷，以及为什么缺陷会遗漏到用户现场。缺陷根本原因分析的目的是从以往的软件开发和测试过程中吸取经验和教训，避免同样的问题重复发生，从而改进开发和测试过程。过程改进反过来可以预防相同的缺陷再次引入或遗漏，从而提高软件产品的质量，这也是软件质量保证的重要一环。

4. 保证产品安全

大部分的软件产品都涉及数据信息，尤其是金融类产品，对个人资金账户的安全性要求极高。因此，确保通过数据加密和安全处理来保障用户的资金流动安全至关重要。如果安全性测试不充分，存在漏洞，其后果将不堪设想。

5. 降低开发成本

高效的测试能够在开发过程中通过跟踪需求、验证质量和提交缺陷等环节，提升开发人员的技术水平。建立一套完整的体系，可以提高整个团队的工作效率，从而降低开发成本，进而把控产品质量。例如，对一款产品在不同终端设备的兼容性进行测试，可以大大降低其他版本开发的成本。

6. 降低商业风险

软件测试除了可以降低开发成本，还可以降低因软件缺陷带来的商业风险。举个例子，如果一款产品使用起来不流畅且缺陷频出，在向合作伙伴展示时难以获得认可和信任，从而可能导致商业损失。

7. 提高用户体验

软件测试主要在软件产品发布前进行，通过提前介入，能够有效保障软件产品的质量。在经过测试和修改之后，再将产品交付给用户，可以显著提高用户体验，增强用户对产品的信心。这可以避免在发布后出现严重影响使用的Bug，减少用户投诉等问题。

8. 树立产品信心

当测试过程中发现的缺陷很少或没有发现缺陷时，测试就可以帮助树立对于软件产品质量的信心。一款没有经过测试的产品发布之后，往往容易出现各种Bug。相反，经过充分测试的产品将使软件开发公司更加自信，从而提升其市场竞争力。

1.5.2　软件测试原则

软件测试经过多年的发展，形成了多种原则用于指导软件测试工作。制定软件测试的基本原则有助于提高测试工作的效率和质量，使测试人员能够以最少的人力、物力和时间，尽早发现软件中存在的问题。测试人员应该在测试原则的指导下进行测试工作。以下是业界公认的一些重要原则。

1. 测试应基于用户需求

所有的测试标准应建立在满足客户需求的基础上。从用户角度来看，最严重的错误是那些导致程序无法满足需求的错误。因此，应依照用户的需求配置环境并且按照用户的使用习惯进行测试并评估结果。如果系统不能满足客户的需求和期望，那么这个系统的研发是失败的。同时，在系统中发现和修改缺陷也是没有任何意义的。在开发过程中，用户的早期介入和接触原型系统就是为了避免这类问题的预防性措施。

2. 软件测试不能证明程序无错

软件测试是发现软件错误的过程，但它并不能证明软件完全无错。即使测试人员运行了大量测试用例而未发现错误，也并不能证明被测软件不存在问题。这是因为软件系统的复杂性和测试的不完备性，导致测试人员很难考虑到所有的可能情况和边界条件。因此，测试人员只能证明某些错误已经被发现，而不能确定不存在其他未发现的错误。

3. 应尽早和不断地进行软件测试

由于原始问题的复杂性、软件本身的抽象性、开发各个阶段工作的多样性，以及参加开发的不同层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。因此，不应把软件测试仅仅看作是软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中。坚持在软件开发的每个阶段进行技术评审，有助于尽早发现和预防错误，把出现的错误克服在早期，杜绝某些隐患。这种方法可以大大降低错误修复的成本，提高软件质量。

4. 做好软件测试计划工作

软件测试是有组织、有计划、有步骤的活动。因此，测试工作必须严格执行测试计划，避免测试的随意性。测试计划应包括：软件的功能、输入、输出、测试内容、各项测试的进度安排、资源要求、测试资料、测试工具、测试用例的选择、测试的控制方法和过程、系统的配置方式、跟踪规则、调试规则、回归测试的规定以及评价标准等。此外，回归测试的关联性一定要引起充分的注意，修改一个错误而引起更多错误出现的现象并不少见。项目测试相关的活动依赖于测试对象的内容。对于每个软件系统，测试策略、测试技术、测试工具、测试阶段以及测试出口准则的选择都可能有所不同。同时，测试活动必须与应用程序的运行环境和使用中可能存在的风险相关联。因此，没有两个系统会以完全相同的方式进行测试。

5. 测试前必须明确定义好产品的质量标准

只有建立了明确的质量标准，才能根据测试的结果对产品的质量进行分析和评估。同时，测试用例应该确定期望输出结果。如果无法确定测试期望结果，则无法进行检验。因此，必须使用预先确立的输入数据和输出结果来对照检查当前的输出结果是否正确，以确保测试的针对性。系统的质量特征不仅仅是功能性要求，还包括了很多其他方面的要求，例如稳定性、可用性、兼容性等。

6. 程序员应避免检查自己的程序

测试工作需要严谨的作风、客观的态度和冷静的情绪。人们常由于各种原因存在一种不愿否定自己工作的心理，认为揭露自己程序中的问题并不是一件愉快的事情。这一心理状态就成为程序员自我测试程序的障碍。另外，程序员对软件规格说明的理解错误也可能导致难以发现错误。如果由他人来测试程序员编写的程序，可能会更客观、更有效，并且更容易取得成功。需要注意的是，这一点不能与程序的调试(Debuging)相混淆。调试工作由程序员自己进行通常更为有效。

7. 充分注意测试中的群集现象

一般来说，程序中已发现的错误数量越多，潜在的错误概率也越高。历史统计数据表明，80%的软件缺陷起源于20%的模块。错误集中发生的现象，可能和程序员的编程水平和习惯有很大的关系。因此，测试时不要以为找到了几个错误问题就已经解决，不需继续测试。经验表明，测试后程序中残存的错误数量与该程序中已发现的错误数量或检错率之间成正比。根据这个规律，应当对错误集中出现的程序段进行重点测试，以提高测试资源的使用效率。

8. 对每一个测试结果进行全面检查

有些错误的征兆在输出实测结果中已经显现，但如果不仔细和全面地检查测试结果，这些错误可能会被遗漏。因此，在测试过程中，必须对预期的输出结果明确定义，对实测的结果仔细分析检查，避免因为疏忽或对结果与预期结果一致性的主观判断而导致错误遗漏。

9. 穷举测试是不可能的

完全测试是指试图找出所有软件缺陷，使软件达到完美状态，但这是不可能实现的。一方面，穷举测试本身是不可行的；另一方面，测试资源是有限的。另外，测试后期发现错误的成本非常大，需要权衡投入与产出之间的关系。不充分的测试是不负责任的，过度测试是浪费资源，同样是不负责任的。因此，需要根据软件质量要求，在满足质量标准的前提下进行有效测试。

10. 测试设计决定了测试的有效性和效率

测试设计决定了测试的有效性和效率，测试工具只能提高测试效率，无法解决所有问题。根据测试的目的，采用相应的方法设计测试用例，可以提高测试的效率，帮助更多地发现错误，从而提高程序的可靠性。除了检查程序是否完成了应执行的任务，还应关注程序是否执行了不应有的操作。此外，测试用例的编写不仅应基于有效和预期的输入情况，还需要考虑无效和未预料的输入情况。

11. 妥善保存测试文档，并注意测试设计的可重用性

应妥善保存测试计划、测试用例、出错统计和测试报告等文档，以便于项目的维护。此外，测试设计可以兼顾更好的通用性，便于整个设计方案可以更好地重用，从而节省开发和设计成本。

12. 软件缺陷的免疫性

频繁使用杀虫剂后，害虫会产生免疫力，使杀虫剂失去效力。在测试中，同样的测试用例反复使用，发现缺陷的能力就会越来越差。这种现象的主要原因在于测试人员没有及时更新测试用例，同时对测试用例及测试对象过于熟悉，导致思维定势。为克服这种现象，测试用例需要定期进行评审和修改，并不断增加新的、不同的测试用例来测试软件或

系统的不同部分，保证测试用例永远是最新状态，反映程序代码或说明文档的最新更新信息。这样，软件中未被测试过的部分或者先前没有被使用过的输入组合就会重新执行，从而发现更多的缺陷。同时，作为专业的测试人员，应具备探索性思维和逆向思维，而不仅仅是对输出与期望结果进行比较。

1.6 软件测试分类

软件测试的分类方式多种多样。本节将从测试方法、测试执行状态、测试执行阶段、用户需求、是否自动化和其他相关维度对软件测试进行分类。通过对软件测试进行分类，可以系统地了解其知识体系，如图1-4所示。

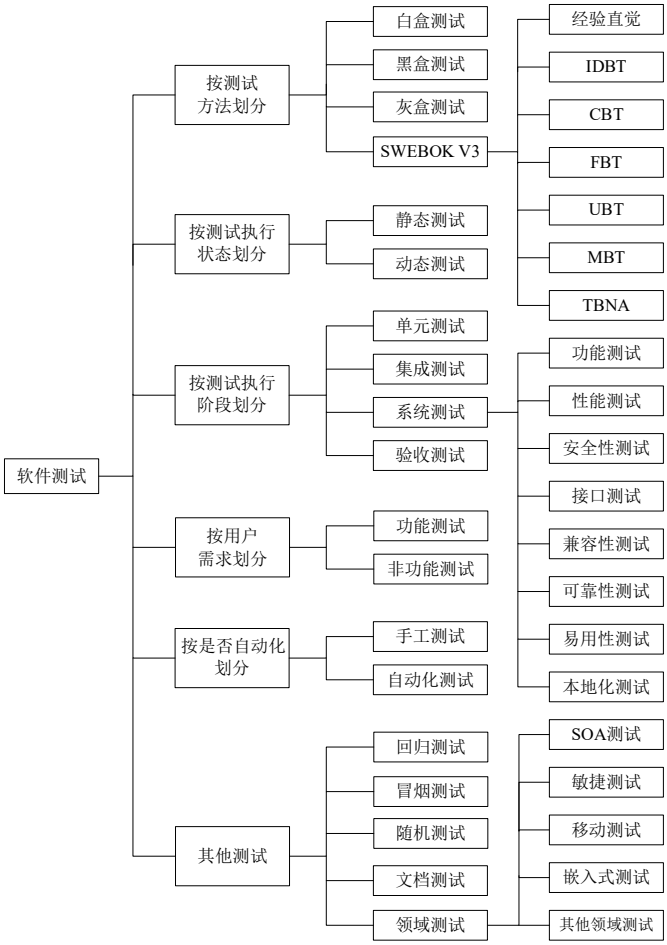

图1-4 软件测试分类图

1. 按测试方法划分

根据测试方法划分，软件测试可以分为白盒测试、黑盒测试和灰盒测试。SWEBOK V3对软件测试的分类方式与这三种方法有所不同。

1) 白盒测试

白盒测试也称结构测试或逻辑驱动测试，是一种根据程序内部的结构对其进行测试的方法。通过这一测试，可以检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。这种方法是把测试视为一个“打开的盒子”，测试人员依据程序内部逻辑结构相关信息，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。白盒测试不仅可以应用于单元测试，还可以覆盖程序的语句、判定、语句判定、判定组合和路径等，还可以扩展到控制流路径的覆盖。

2) 黑盒测试

黑盒测试通过检测每个功能的正常使用情况，发现软件设计规格说明书中的错误和缺陷。在测试过程中，程序被视为一个无法打开的黑盒子。在不考虑程序内部结构和内部特征的情况下，只检查程序功能是否按照要求规范说明书的规定正常使用，程序是否能够适当地接收输入数据并产生正确的输出信息。黑盒测试侧重于程序的外部结构，不考虑内部逻辑结构，主要测试软件界面和软件功能。黑盒测试是从用户的角度来测试输入数据和输出数据之间的对应关系。显然，如果外部特征本身的设计有问题或规格说明有误，则无法用黑盒测试方法发现这些问题。因此，除了要测试所有合法的输入，还要测试那些非法但可能的输入。从这个角度来看，完全测试是不可能的，所以我们应该进行有针对性的测试，通过制定测试案例来指导测试的实施，以确保软件测试有组织、有步骤、有计划地进行。为了真正保证软件的质量，必须量化黑盒测试行为，测试用例是量化测试行为的具体方法之一。黑盒测试用例的具体设计方法包括等价类划分、边界值分析、错误推测、因果图、判定表驱动、正交实验等。

3) 灰盒测试

灰盒测试是一种介于白盒测试和黑盒测试之间的测试方法，主要用于应用程序的集成测试阶段。它不仅关注针对集成系统的输入和输出值的正确性，同时也对程序的内部执行逻辑进行分析、监测和验证。灰盒测试的常见方法是在应用程序系统执行的过程中，利用插装、调试、日志记录或信号监测等多种技术，对软件的内部执行过程进行分析、度量和验证，从而实现对软件内部缺陷的更全面检测。根据不同的应用程序，灰盒测试的目标也会有所差异。灰盒测试正好可以弥补白盒测试和黑盒测试的不足，兼顾了测试的效率，同时又能洞悉系统内部执行过程。灰盒测试也许还没有像白盒测试和黑盒测试那样普遍且标准化地应用到常见的研发流程中，但其思想和方法对大多数软件研发人员来说可能并不陌生。例如使用调试器进行单步执行以观察程序的执行逻辑，或手动插入print( )函数以获取执行日志等。要想将灰盒测试的方案推广到研发流程中，可以借助一个集成化、易用且自动化的解决方案。

4) SWEBOK V3

2014年2月20日，IEEE计算机协会发布了软件工程知识体系(Software Engineering Body of Knowledge，SWEBOK)指南第3版。SWEBOK V3的测试分类方式不同于黑盒测试和白盒测试，主要包括以下测试方法。

- 基于直觉和经验的方法，例如Ad-hoc测试方法和探索式测试等。
- 基于输入域的方法(IDBT)，例如等价类、边界值、两两组合和随机测试等。
- 基于代码的方法(CBT)，例如基于控制流的标准、基于数据流的标准以及CBT参考模型等。
- 基于故障的方法(FBT)，例如故障模型、错误推测和变异测试等。
- 基于用途的方法(UBT)，例如操作配置和用户观察启发等。
- 基于模型的方法(MBT)，例如决策表、有限状态机、形式化验证、TTCN3和工作流模型等。
- 基于应用技术的方法(TBNA)，例如OOS、Web、Real-time、SOA、Embedded、Safe-critical等。

2. 按测试执行状态划分

根据测试执行状态，软件测试可以分为静态测试和动态测试。

1) 静态测试

静态测试是指不运行被测试软件系统的情况下，采用其他手段和技术对被测试软件进行检测的一种测试技术。例如，代码走读、文档评审和程序分析等都是静态测试的范畴。常用的静态分析技术包括控制流分析、信息流分析和数据流分析，但现在这些方法实际应用较少，因为很多问题在编译的时候就解决了。在测试过程中，静态测试中最常用的方法是对文档进行评审。不同类型的文档在评审时关注的问题也各不相同。

几乎所有软件工作产品都可使用静态测试(包括评审和静态分析技术)进行检查，例如：

- 规格说明，包括业务需求、功能需求和安全性需求；
- 用户故事和验收准则；
- 架构和设计规格说明；
- 代码；
- 测试构件，包括测试计划、测试用例、测试规程和自动化测试脚本；
- 用户手册；
- Web 网页；
- 合同、项目计划、进度表和预算计划；
- 配置设置以及基础设施的设置；
- 模型，例如用于基于模型测试的活动图。

评审可以应用于任何工作产品，只要参与者具备阅读和理解的能力。静态分析技术可以有效地应用于具有规范结构(典型代表有代码或模型)的任何软件工作产品，并可运用适当的静态分析工具。此外，静态分析技术还可以借助工具评估以自然语言编写的工作产品，例如需求文档(如检查拼写、语法和可读性)。

2) 动态测试

动态测试是指按照预先设计的数据和步骤来运行被测软件系统，从而对被测软件系统进行检测的一种测试技术。按照阶段划分，单元测试阶段常用的动态测试方法是逻辑覆盖。而在系统测试阶段，进行的测试都属于动态测试，因为必须运行系统才能验证系统功

能是否正确。动态测试是通过观察代码运行时的动作，来提供执行跟踪、时间分析和测试覆盖度等信息。动态测试通过实际运行程序发现错误，并通过有效的测试用例分析对应的输入/输出关系，以评估被测程序的运行情况。

3. 按测试执行阶段划分

根据测试执行阶段划分，软件测试可以分为单元测试、集成测试、系统测试和验收测试。

1) 单元测试

单元测试主要针对最小的软件设计单元(模块)进行验证，目标是确保模块被正确的编码。测试过程中以过程设计描述为指南，对重要的控制路径进行测试以发现模块内的错误。通常情况下，单元测试采用白盒测试的方法，包括对代码风格和规则、程序设计和结构、业务逻辑等进行静态测试，以便及早发现和解决不易显现的错误。

2) 集成测试

集成测试旨在发现与模块接口有关的问题，其目标是在通过单元测试的基础上，将各个模块组合成设计文档中所描述的程序结构。通常建议避免一次性集成(除非软件规模很小)，而采用增量集成。集成方式一般分为自顶向下和自底向上两种方式。自顶向下集成是指模块集成的顺序是首先集成主模块，然后按照控制层次结构向下进行集成。隶属于主模块的模块按照深度优先或广度优先的方式集成到整个结构中。自底向上集成是指从原子模块开始构建和测试。因为模块是自底向上集成的，集成时要求所有隶属于某个顶层的模块始终可用，无须使用稳定的测试桩。

3) 系统测试

系统测试是基于系统整体需求说明书的黑盒类测试，旨在覆盖系统中所有相关的组件。系统测试是针对整个产品系统进行的测试，其目的是验证系统是否满足了需求规格的定义，找出与需求规格不符或存在矛盾的地方。系统测试的对象不仅包括需要测试的产品系统的软件，还包含软件所依赖的硬件、外设，甚至某些数据、支持软件及其接口等。因此，必须将系统中的软件与各种依赖的资源结合起来进行测试。系统测试通常包含功能、性能、安全性、接口、兼容性、可靠性、易用性、本地化等多个方面的测试。

4) 验收测试

验收测试是系统开发生命周期方法论中的一个重要阶段。在此阶段，相关的用户或独立测试人员根据测试计划和结果对系统进行测试和接收。该测试旨在让系统用户决定是否接收系统，并确定产品是否能够满足合同或用户所指定的需求。验收测试包括Alpha测试和Beta测试。Alpha测试由用户在开发者的场所进行，测试在一个受控的环境中进行。Beta测试由软件的最终用户在一个或多个用户场所进行，开发者通常不在现场，用户记录测试中遇到的问题并报告给开发者。开发者对系统进行最后的修改，并开始准备发布最终版本的软件。

4. 按用户需求划分

根据用户需求，软件测试可以分为功能测试和非功能测试。

1) 功能测试

根据国际软件测试资质认证委员会(ISTQB)的定义，功能测试是一种检查组件或整个系

统功能的测试。简而言之，功能测试帮助企业确保软件产品的特定功能完全按照预期正常工作。功能测试一般包括单元测试、集成测试、健全性测试、回归测试和Beta测试等。

2) 非功能测试

ISTQB将非功能测试定义为测试与系统功能无关组件。非功能测试的范围可能是无穷无尽的，并且很大程度上取决于产品的具体情况。非功能性需求更多关注产品如何为最终用户提供服务，而不是仅仅关注预期结果。非功能测试旨在通过各种标准评估应用程序的质量情况，通常包括性能测试、压力测试、负载测试、容量测试、安全测试、易用性测试和本地化测试等。

5. 按是否自动化划分

根据是否自动化，软件测试可以分为手工测试和自动化测试。

1) 手工测试

手工测试是指软件测试的整个过程(如评审、测试设计、测试执行等)均由软件测试工程师手动完成，不使用任何测试工具。狭义上讲，手工测试强调测试执行由人工完成，是最基本的测试形式。

2) 自动化测试

自动化测试是指使用软件来控制测试执行过程，以比较实际结果和预期结果的一致性，同时设置测试的前置条件和其他测试控制条件，并输出测试报告。通常，自动化测试需要在适当的时间使已经形式化的手工测试过程自动化。自动化测试将大量的重复性测试工作交给计算机去完成，可以有效节省人力和时间成本，从而提高测试效率。

6. 其他测试

除了上述分类方法，还有其他一些测试类型，例如回归测试、冒烟测试、随机测试、文档测试和领域测试等。

1) 回归测试

回归测试是指在软件项目中，当开发人员修改软件代码以修复已知的Bug后，测试人员重新测试之前已经测试过的内容，以确认此次修改没有引入新的错误。也就是说，回归测试的目的就是检查开发人员在修复已有Bug时是否导致了新的Bug。

2) 冒烟测试

冒烟测试最初源于电路板测试。当电路板完成后，首先会进行加电测试，如果电路板没有冒烟，才会进行其他测试；否则，电路板必须重新制作。类似地，冒烟测试就是在新版本发布时，将软件的全部功能进行一次快速检查，以发现是否存在重大问题。如果所有功能可以正常运行且不会影响测试进行，则该版本可以进入正式测试阶段。如果功能有重大问题或影响测试进行，那么这个版本就是不合格的，无须进行进一步测试。

3) 随机测试

随机测试的输入数据都是随机生成的，目的是模拟用户的真实操作，对一些特殊使用操作、特殊使用环境、程序并发运行可能造成的问题进行检查。对于软件的重要功能、测试用例未覆盖到的部分、软件更新和新增功能，尤其是前期测试出现严重缺陷的部分，应当进行随机测试，并可以结合回归测试一起进行。每个新的软件版本都需要进行随机测

试，特别需要重视对即将发布的软件版本的随机测试工作。目前，随机测试已经演化为更为系统和专业的探索性测试。

4) 文档测试

软件文档测试是软件测试中非常重要的一个环节，主要涉及各种类型文档的检查和验证。具体而言，文档测试包括了需求文档测试、设计文档测试、用户手册测试、帮助文档测试以及软件源代码的测试等。通过对这些文档的测试，可以确保软件开发的质量和可靠性，从而保障软件产品的稳定运行。

5) 领域测试

软件测试如果具体到技术领域，可以分为SOA测试、敏捷测试、移动测试、嵌入式测试和其他测试领域等。

- SOA测试是针对SOA架构风格的测试。在此测试中，软件组件通过通信协议以网络形式进行交互。它是服务生命周期管理的重要组成部分，支持跨多个SOA服务实现解决服务质量的多个方面。该测试相对复杂，因为复合软件具有许多活动部件和互连关系，对测试具有挑战性。
- 敏捷测试是为了适应敏捷开发而特别设计的一套完整的软件测试解决方案。该解决方案支持持续交付，涵盖所需的价值观、思维方式、测试流程、一系列优秀的测试实践，以及更合适的测试环境、自动化测试框架和工具。敏捷测试可以采用目前已有的各种测试方法，包括手工探索式测试、接口自动化测试、UI自动化测试、边界值/等价类划分方法、组合测试方法等。与传统测试相比，敏捷测试的侧重点有所不同，主要的差别是在价值观、测试思维方式、流程和实践上。
- 移动测试应确保App在所有移动设备及其操作系统上高效运行方面发挥着关键作用。移动App测试通常是指检查App的功能性和非功能性组件，主要针对原生应用、移动Web应用和混合应用进行测试，主要涉及兼容性、操作系统、性能测试、安全性测试、稳定性测试、安装和易用性测试等方面。
- 嵌入式测试是针对嵌入式系统的特殊测试方法。与普通软件测试相比，嵌入式测试的特点在于测试对象的特殊性和测试方法的多样性。嵌入式软件通常集成在设备中，无法独立运行。因此，测试过程中需要考虑设备的特殊硬件和软件环境。
- 其他测试领域包括Web领域、云计算、大数据等，这些技术领域都需要相应的软硬件测试环境和测试方法。

1.7　测试用例

测试用例(Test Case)是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，旨在测试某个程序路径或验证其是否满足特定需求。测试用例是软件测试人员需要具备的基础能力。

测试用例的用途如下：

- 指导测试工作有序进行，使实施测试的数据有据可依；
- 确保所实现的功能与客户的预期需求相符；
- 完善软件不同版本之间的重复性测试；
- 跟踪测试进度，确定测试重点；
- 评估测试结果的度量标准；
- 增强软件的可信任度；
- 分析缺陷的标准。

如何编写测试用例呢？这需要用到相应的测试方法。测试用例一般包含以下内容。

- 用例编号：唯一标识，与需求编号对应，形成多对一关系。
- 用例编写者：设计用例的人员。
- 被测对象：要测试的功能点(模块、系统)。
- 用例标题：对测试项的简短描述。
- 用例级别：确定用例执行的级别。
- 前提条件：执行用例时需要的预置条件。
- 输入条件：执行该动作需要输入的数据。
- 操作步骤：执行该动作需要完成的操作。
- 预期结果：执行完该动作后程序的预期结果。
- 实际结果：实际输出的结果(可选)。
- 问题描述：执行该用例后系统显示的错误信息。
- 验证结果：该测试用例是否执行通过。
- Bug编号：填写Bug管理系统中对应此用例的Bug编号(可选)。
- 需求编号：唯一标识，与用例编号对应，形成一对多关系。
- 测试执行者：按照该用例执行测试的人员。
- 测试日期：执行测试的时间。
- 备注：对未执行或不能执行的用例进行说明。

在设计测试用例时，需要注意以下问题。

1. 测试用例应当包括合理的输入条件和不合理的输入条件

合理的输入条件是指能够验证程序正确性的输入条件，而不合理的输入条件是指异常、临界或可能引起问题的输入。在测试程序时，测试人员常常倾向于过多地考虑合法且预期的输入条件，以检查程序是否按预期执行，而忽视了不合法和意外的输入条件。事实上，软件在投入运行以后，用户的操作往往不遵循预设的规范，可能会输入一些意外的内容，例如用户在键盘上按错了键或输入非法命令。如果开发的软件在遇到这种情况时不能做出适当的反应，给出相应的信息，就容易导致故障，轻则返回错误的结果，重则导致软件失效。因此，软件系统处理非法命令的能力也必须在测试时受到检验。使用不合理的输入条件测试程序时，往往能够发现比使用合理输入条件更多的错误。

2. 测试用例应由测试输入数据及其对应的预期输出结果两部分组成

在测试之前，应根据测试的要求选择在测试过程中使用的测试用例。测试用例主要用

于检验程序员编写的程序，因此不仅需要测试的输入数据，还需要针对这些输入数据的预期输出结果。如果没有给出预期的程序输出结果，就缺乏了检验实测结果的基准，这可能导致将一个似是而非的错误结果当成正确结果。

关于测试用例的编写规范，已经形成了典型的方法供测试人员参考。编写规范一般包含以下设计原则。

- 系统性：应完整说明整个系统的业务需求，包括系统由几个子系统组成以及它们之间的关系；对于模块业务流程，需要说明子系统内部的功能、重点功能以及它们之间的关系。
- 连贯性：对系统业务流程要说明各个子系统之间是如何连接的。若需要接口，必须确保各子系统之间的接口正确；若依靠页面链接，则应验证页面链接的准确性。对模块业务流程，要说明同级模块以及上下级模块如何构成一个子系统，并确保其内部功能接口的连贯性。
- 全面性：应尽可能覆盖各种路径和业务点，同时考虑跨年、跨月的数据情况，以及大数据量的并发测试准备。
- 正确性：输入界面后的数据应与测试文档所记录的数据一致，预期结果也应与测试数据所对应的业务逻辑相符。
- 符合正常业务规则：测试数据要符合用户实际工作中的业务流程，同时也要兼顾各种业务的变化以及当前行业的法律法规。
- 可操作性：测试用例中应明确测试的操作步骤，以及不同的操作步骤对应的测试结果。

编写标准一般包含以下内容。

- 测试用例编写应制定统一的模板，并约定模板的使用方法。
- 测试用例编写应当根据项目实际情况编制测试案例编写手册，包括案例编号规则、案例编写方法、案例编写内容及案例维护等内容。
- 测试用例的编写应根据手册中约定的编写方法和内容进行。
- 测试用例的编写要步骤明确，输入和输出要清晰，并与需求和缺陷相对应。
- 测试用例的编写应严格根据需求规格说明书及测试需求功能分析点进行，确保覆盖全部需求功能点。
- 注重案例的可复用性，使其能够在未来相似系统的测试过程中重复使用，从而减少测试设计的工作量。

编写高质量的测试用例是软件测试的基础。然而，由于软件人员在需求理解和设计等方面存在差异，首次编写的测试用例质量往往会有所不同。因此，对编写的测试用例进行评审是非常必要的。测试用例的评审过程有助于清理用例结构、全面覆盖测试场景，并合理安排用例优先级。

测试用例的评审一般包含以下内容。

- 评估用例设计的结构安排是否清晰合理，以及是否有效覆盖了需求。
- 检查用例的优先级别是否安排合理。
- 确认是否覆盖了测试需求的所有功能点，包括需求中的业务规则及所有用户可能使用的流程或场景。

- 确保用例具有良好的可执行性，包括前提条件、执行步骤、输入数据和预期结果是否清晰且正确。
- 检查是否已经删除了冗余的测试用例。
- 确认是否包含充分的负面测试用例。
- 评估用例是否简洁，并具有较强的复用性。
- 确认用例是否易于管理。

测试工程师的主要工作就是编写测试用例，因此编写测试用例是每一个测试工程师必备的技能。一个优秀的测试用例能够有效指导测试执行的过程。本书将在后续章节重点介绍测试方法，通过应用合适的测试方法与合理的编写规范，帮助用户编写高质量的测试用例。

1.8 软件测试资质认证

在软件测试认证领域，常见的认证体系有ISTQB、CSTQB、软件评测师、CSTE和LoadRunner ASP等。

CSTE(Certified Software Tester)是QAI(Quality Assurance Institute)旗下的重要认证。CSTE是一项广泛认可的专业认证，旨在证明持证人在业务及联营公司的专业能力。这项认证有助于促进职业发展，并为担任管理顾问角色提供更多机会。

LoadRunner ASP(LoadRunner Accredited Software Professional)即性能测试专业人士资格认证，该认证针对性能测试工具LoadRunner进行认证。LoadRunner作为目前性能测试应用最广泛的商用测试工具，最初由Mercury公司开发，2006年被惠普收购并运营了11年，在2017年被Micro Focus收购。为了给测试人员提供统一的行业标准，Micro Focus设立了LoadRunner ASP认证考试。该认证为软件测试人员提供了统一的行业标准，严格评估测试人员的知识和技能。目前，该认证已在全球 100 多个国家和地区推广，并建立了全球统一认证考试系统，是软件性能测试领域含金量最高的认证之一。

下面将简单介绍ISTQB、CSTQB和软件评测师等相关认证。

1.8.1 ISTQB

ISTQB (International Software Testing Qualifications Board) 国际软件测试资质认证委员会是国际权威的软件测试资质认证机构，主要负责制定和推广国际通用资质认证框架，即国际软件测试资质认证委员会推广的软件测试工程师认证项目(ISTQB-Certified Tester)。ISTQB是一个注册于比利时的非营利性组织，该项目由ISTQB授权各国分会在本国组织软件测试工程师的认证，并接受ISTQB质量监控。通过认证的人员将获得全球通用的软件测试工程师资格证书。

ISTQB目前拥有66个分会，覆盖包括中国、美国、德国、英国、法国、印度等在内的120多个国家和地区。来自这些国家和地区的数百位测试领域专家作为志愿者服务于ISTQB

及其倡导的软件测试工程师认证体系。ISTQB已在全球130个国家进行了超过100万次考试，并颁发了超过85万份认证，使其成为测试行业的第一大认证机构，在整个IT行业居第三位(仅次于PMI和ITIL)。

ISTQB在全球范围建立和推广ISTQB-Certified Tester国际认证软件测试工程师培训及国际软件测试认证体系。ISTQB-Certified Tester培训及认证体系分为三个级别：基础级(Foundation Level)、高级(Advanced Level)和专家级(Expert Level)。

ISTQB国际软件测试工程师认证已经成为IT市场需求最大的职业认证之一。获得ISTQB资格认证被视为迈向知名企业的全球通行证。目前，包括IBM、惠普、SAP、西门子、SONY、诺基亚、三星等公司已经要求软件测试人员需要获得ISTQB认证。随着国内软件测试行业的快速发展，获得国际软件测试认证已经成为从事软件测试的“上岗证”。

ISTQB-Certified Tester是一个基于全球统一标准规范和术语大纲的培训及认证体系。该体系分为三个级别：基础级(Foundation Level，CTFL)、高级(Advanced Level，CTAL)、专家级(Expert Level，CTEL)，如图1-5所示。

图1-5　ISTQB认证级别

1. 基础级(CTFL)

CTFL适合有一定测试专业基础知识的测试人员，包括测试工作人员、测试管理人员、质量控制人员(QA/QC)、软件开发人员以及IT部门工作者等。此外，该认证也适合有志于从事软件测试的人员以及具有软件测试基本知识的计算机相关专业学生。

基础级扩展-敏捷测试工程师(CTFL-AT)面向的专业人员包括测试人员、测试分析师、测试工程师、测试顾问、测试经理、用户验收测试人员和软件开发人员等。该认证也适用于在敏捷环境中想要更深入了解软件测试的任何人，例如项目经理、质量经理、软件开发经理、业务分析师、IT总监和管理顾问等。CTFL-AT主要包括软件测试基础理论、软件测试周期、静态测试技术、白盒测试技术、黑盒测试技术、测试设计、测试管理基础和测试工具基础。

2. 高级(CTAL)

CTAL认证适合拥有3到5年以上测试相关经验的测试人员，包括测试员、测试分析师、测试工程师、测试咨询人员、测试经理、用户验收测试人员以及软件开发人员。该认证也适合于希望深入理解软件测试的人员，例如项目经理、质量经理、软件开发经理、业务分析人员、IT主管和管理顾问等。参加该级别认证的人员需要先通过CTFL。

CTAL考试分为三个模块：Test Manager(高级测试经理)、Test Analyst(高级测试分析师)和Technical Test Analyst (高级技术测试分析师)，通过这三个模块才能获得高级证书。考试内容主要包括高级功能测试技术、自动测试技术和应用、高级结构化测试技术、软件测试管理理论和方法。

1) 高级测试经理

通过高级测试经理认证后，测试管理专业人员应有能力完成以下工作。

- 通过实现测试组织设定的使命、目标和测试过程来管理测试项目。
- 组织和领导风险识别与分析，并使用这些结果来进行测试评估、计划、监督和控制。
- 制订并实施与组织政策和测试策略一致的测试计划。
- 通过持续监督和控制测试活动来达到项目目标。
- 向项目利益相关者及时评估和报告相关的测试状态。
- 发现测试团队中的技术和资源缺口，并参与寻找合适的团队成员。
- 确定并规划测试团队所需的技能发展。
- 为测试活动提出一个包括了预期成本和收益的商业提案。
- 保证测试团队内部与其他项目利益相关者的有效沟通。
- 参与并领导测试过程改进活动。

2) 高级测试分析师

通过高级测试分析师认证后，测试管理专业人员应有能力完成以下工作。

- 基于使用的软件开发生命周期，设计并实施合适的测试活动。
- 基于风险分析给出的信息，确定测试活动的合理优先级。
- 根据定义的覆盖标准，选择和应用合适的测试技术，以确保测试能够提供足够的信息。
- 提供与测试活动相关的适当级别的文档。
- 确定要进行的功能测试的合适类型。
- 对项目承担易用性测试的职责。
- 应用工作产品中典型错误的知识，积极参与与利益相关者的正式或非正式评审活动。
- 设计并实施缺陷分类方案。
- 应用工具以支持有效的测试过程。

3) 高级技术测试分析师

通过高级技术测试分析师认证后，测试管理专业人员应有能力完成以下工作。

- 识别与区分软件系统中与性能、安全、可靠性、可移植性和维护性相关的典型风险。
- 制订详细的测试计划，描述测试的设计和执行，以降低性能、安全性、可靠性、可移植性和维护性方面的风险。
- 选择并应用合适的结构设计技术，以确保测试能提供足够的信心，主要基于代码覆盖和设计覆盖。
- 应用对代码和架构中典型错误的知识，积极有效地与开发者和软件架构师一起进行技术评审。
- 识别代码和软件架构中的风险，创建测试计划相关内容，并通过动态分析来降低这些风险。
- 通过应用静态分析，提出代码的安全性、维护性和可测试性方面的改进建议。
- 对于引入特定类型的测试自动化，概述其可能带来的成本和收益。
- 选择合适的工具以实现技术测试任务的自动化。
- 理解在应用测试自动化中可能遇到的技术问题和概念。

3. 专家级(CTEL)

参加专家级实施测试过程改进模块资格认证的人员需满足以下要求。

(1) 必须已经通过高级测试经理模块的认证。

(2) 除了通过资格认证考试，在获得专家级证书之前，还必须提供实际测试工作经验的证明，特别要提供申请认证的专家级模块所在领域的工作经验证明。

(3) 除了通过考试，还要符合以下要求。

- 至少5年的实际测试工作经验(需提交个人简历，并附上2封推荐信)。
- 至少2年的专家级领域工作经验(需提交个人简历，并附上2封推荐信)。
- 至少发表过1篇相关的文章，或在测试大会中进行过专家级模块相关的测试专题演讲。

CTEL内容主要包括改进概要、基于模型的改进、基于分析的改进、选取测试过程改进的方法、改进过程的组织、角色和技能、管理变更、改进成功的要素以及适应不同开发周期模型的方法。

1.8.2 CSTQB

CSTQB是ISTQB在中国的唯一分会，成立于2006年。它全权代表ISTQB在授权区域内推广ISTQB软件测试工程师认证体系、认证，管理培训机构和考试机构，接受ISTQB的全面业务指导和授权。

CSTQB通过市场调研、信息交流、咨询培训、评估认证、版权保护等方面的工作，推动中国软件测试行业的发展，并致力于为CSTQB会员提供优质的服务，面向全行业，发挥政府与企业和事业单位之间的纽带和桥梁作用，为中国的测试行业提供新的测试方法和技术研究与推广的交流平台。同时，CSTQB加强国际交流与合作，积极推进国际通用软件培训和认证体系的建设，致力于建立规范的高端培训和认证平台，促进国际软件测试人才流动和技术交流，使中国软件测试行业与国际接轨。

1.8.3 软件测评师

计算机技术与软件专业技术资格(水平)考试(以下简称计算机软件资格考试)是对原中国计算机软件专业技术资格和水平考试(简称软件考试)的完善与发展。计算机软件资格考试是由国家人力资源和社会保障部、工业和信息化部领导下的国家级考试，其目的是科学、公正地对全国计算机与软件专业技术人员进行职业资格、专业技术资格认定和专业技术水平测试。计算机软件资格考试设置了27个专业资格，涵盖5个专业领域，3个级别层次(初级、中级和高级)。该考试由于其权威性和严肃性，得到了社会各界及用人单位的广泛认同，并在推动国家信息产业特别是软件和服务产业的发展，以及提高各类信息技术人才的素质和能力中发挥了重要作用。具体的资格设置如图1-6所示。

	计算机软件	计算机网络	计算机应用技术	信息系统	信息服务
高级资格	信息系统项目管理师　系统分析师　系统架构设计师　网络规划设计师　系统规划与管理师				
中级资格	软件评测师 软件设计师 软件过程能力评估师	网络工程师	多媒体应用设计师 嵌入式系统设计师 计算机辅助设计师 电子商务设计师	系统集成项目管理工程师 信息系统监理师 信息安全工程师 数据库系统工程师 信息系统管理工程师	计算机硬件工程师 信息技术支持工程师
初级资格	程序员	网络管理员	多媒体应用制作技术员 电子商务技术员	信息系统运行管理员	网页制作员 信息处理技术员

图1-6　计算机软件资格考试

软件评测师属于我国计算机软件资格考试中的中级内容。软件测评师是指能在掌握软件工程与软件测试知识基础上，运用软件测试管理方法、软件测试策略、软件测试技术，独立承担软件测试项目，并具备工程师实际工作能力和业务水平的专职人员。

软件评测师主要考核以下知识体系。

- 操作系统、数据库、中间件、程序设计语言及计算机网络基础知识。
- 软件工程知识。
- 软件质量及软件质量管理的基础知识。
- 软件测试标准、测试技术及方法。
- 软件测试项目管理知识。

除了上述的软件测试资质认证体系，还有多种行业和企业级的认证体系，本章将不再详细介绍。

1.9　思考题

1. 什么是Bug？评判软件缺陷的标准是什么？
2. 软件文档中是否也有软件缺陷？
3. 软件产品的质量主要有哪些特性？
4. 简述软件工程与软件测试的关系。
5. 软件测试主要包含哪些分类？
6. 软件测试的目的和原则是什么？
7. 什么是测试用例？测试用例中应包含哪些信息？
8. 简述测试用例的设计原则。
9. 软件测评师主要考核哪些知识体系？