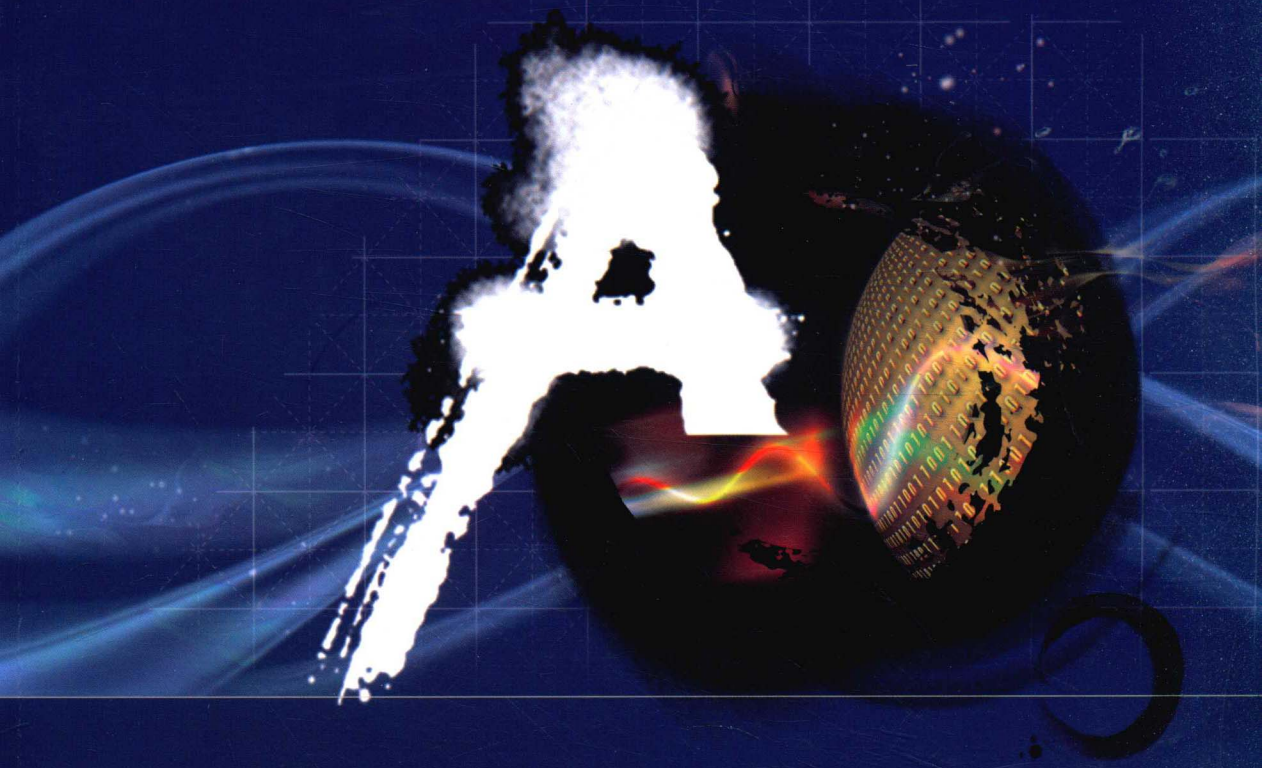
普通高等教育“十二五”规划教材



软件测试程序设计技术

◎孙 晶 杨 波 主编 ◎赵会群 主审

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

普通高等教育“十二五”规划教材

软件测试程序设计技术

孙　晶　杨　波　主编

赵会群　主审

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书从软件测试的基本理论出发，围绕 TTCN-3 核心语言国际测试标准，并结合大量的实际测试案例，对软件测试的相关方法与技术进行了详细的介绍，使读者能够更贴近实际地去了解软件测试。全书共 10 章，主要内容包括软件测试概述，软件测试基础，TTCN 树表描述语言程序设计，TTCN-3 基本语言元素，类型声明，语句、函数、可选步与通信，TTCN-3 核心语言程序设计，测试描述与测试控制，系统测试及测试工具，基于 TTCN-3 的软件测试案例。

本书内容全面、实例丰富、可操作性强，做到了理论与实践的有机结合。

本书适合作为计算机专业高年级本科生和研究生教材或教学参考书，也适合作为软件测试和软件开发相关人员的技术参考书。

图书在版编目（CIP）数据

软件测试程序设计技术 / 孙晶，杨波主编. — 北京：电子工业出版社，2015.10
（普通高等教育“十二五”规划教材）
ISBN 978-7-121-27337-7

I. ①软… II. ①孙… ②杨… III. ①软件－测试 IV. ①TP311.5

中国版本图书馆 CIP 数据核字（2015）第 234529 号

策划编辑：袁 玺
责任编辑：郝黎明　　特约编辑：张燕虹
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1092　1/16　印张：19　字数：536 千字
版　　次：2015 年 10 月第 1 版
印　　次：2015 年 10 月第 1 次印刷
定　　价：39.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010)88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010)88258888。

前　言

计算机技术已经越来越广泛地应用于国民经济和国防建设的各个部门，以不可阻挡之势渗透到人们工作和生活的各个领域，尤其在航天、航空、核能、通信、交通、金融等一些关键领域中，计算机的作用更加至关重要。同时，它们对计算机软件的可靠性和安全性也有严格的要求。近年来，由于软件错误而造成经济损失、导致严重后果的事件屡见不鲜，因此，如何保证软件产品的质量和可靠性就成为人们必须解决的一个重要问题，而软件测试便是保证软件质量的一个重要手段。据统计，国外在软件开发中，开发费用的近一半甚至更多要用于软件测试，由此也可以看出软件测试在软件开发中的重要地位。

本书是为希望了解软件测试的相关技术，尤其是希望了解 TTCN-3 的读者而编写的。依托 TTCN-3，本书给出了丰富且易于实践的案例，让读者能够做到理论与实践相结合，从而能够更加充分地理解和掌握软件测试的相关方法和技术。

本书共 10 章，第 1 章是软件测试概述，第 2 章是软件测试基础，第 3 章是 TTCN 树表描述语言程序设计，第 4 章是 TTCN-3 基本语言元素，第 5 章是类型声明，第 6 章是语句、函数、可选步与通信，第 7 章是 TTCN-3 核心语言程序设计，第 8 章是测试描述与测试控制，第 9 章是系统测试及测试工具，第 10 章是基于 TTCN-3 的软件测试案例。

本书主要介绍如下内容。

(1)介绍软件测试的基本方法，重点讨论软件测试黑盒法和白盒法。对软件测试的过程进行讨论，重点讨论软件测试的设计和测试文档的使用。

(2)介绍 TTCN-3 核心语言的基本概念、语法结构和测试系统结构。通过学习可以掌握 TTCN-3 核心语言测试系统的设计与实现方法，能够设计小型协议软件、应用软件、嵌入式软件的测试系统，为实际从事测试工作奠定理论基础。

(3)介绍如何利用 TTCN-3 进行实际的软件测试，结合多个测试案例进行详细的讲解。

本书详细地讲述了软件测试的有关概念、方法、过程等方面的基础知识，也用大量篇幅讲解了 TTCN-3 的语法知识，并给出了丰富的案例，目的是使读者对软件测试有一个比较全面的了解，并为进一步研究软件测试技术奠定基础。

本书中有大量的算法语句、程序语句及计算公式等，对于其中的变量，为了方便读者阅读，避免歧义，不再区分正、斜体，而是统一采用正体，特此说明。

本书由孙晶、杨波主编，赵会群主审。

由于编者水平有限，书中难免有疏漏和不当之处，敬请广大读者不吝指正。

作　者

目　录

第1章　软件测试概述

本章概述软件测试的有关概念、方法和过程等方面的基础知识。使读者对软件测试有一个比较全面的了解，并为进一步讨论软件测试技术奠定基础。

1.1　软件故障与软件测试

在计算机故障中，有相当一部分是软件故障。下面让我们看两个例子。

例 1：英特尔奔腾浮点除法软件故障

在计算机的“计算器”程序中输入以下算式：

(4195835/3145727)*3145727-4195835

如果答案是 0，则说明计算机没有问题；如果得出的结果不是 0，则说明计算机的工作不正常。看起来这不应该是个问题，可实际上就出现了问题。

1994 年 12 月 30 日，美国 Lynchburg 大学的 Thomas R.Nicely 博士在一台奔腾 PC 上做除法运算时发现，上面的算式不等于 0。后来，他把这一个惊人的发现在 Internet 上发布出去，引起了一场风暴，成千上万的人都发现了同样的问题。那么是什么原因造成这样的算式计算错误呢？这由固化在奔腾 CPU 上的运算器芯片中的软件故障所致。

例 2：千年虫(Y2K)问题

首先介绍一个传说：20 世纪 70 年代一个名叫 Dave 的程序员，负责其公司的工资系统。他使用的计算机存储空间很小，迫使其尽量节省每一个字节。Dave 自豪地将自己的程序压缩得比其他人的更小。他使用的其中一个方法是把 4 位数日期缩减为 2 位，例如将 1973 年缩为 73。因为工资系统极度依赖数据处理，Dave 节省了可观的存储空间。Dave 并没有想到这是个很大的问题，他认为只有在 2000 年时程序计算 00 或 01 这样的年份时才会出现错误。他知道那时会出问题，但是在 25 年之内程序肯定会更改或升级，而且眼前的任务比未来更加重要。这一天毕竟是要来到的。1995 年，Dave 的程序仍然在使用，而 Dave 退休了，谁也不会想到进入程序检查 2000 年的兼容性问题，更不用说去修改了。

关于 Y2K 问题的说法不一，但根本的问题是用 2 位表示年份的问题。这是一个十分典型的软件设计缺陷。Y2K 问题涉及 4 个方面：硬件、操作系统、应用软件及数据。

有关千年的例子很多，给计算机产业带来一次震惊和恐慌。许多国家和大型计算机公司都动用了大量的人力和物力，解决千年虫问题，尤其是解决关系到国家安全、国家支柱产业正常运转和与百姓生活息息相关领域的计算机系统的千年虫问题。

微软作为全球最大的软件供应商，其产品涵盖了操作系统、应用软件及数据等领域，而在 PC 平台上形成最为广泛的应用。关于这一问题，微软对其产品进行了全面的兼容性测试。

微软自 1996 年起开始涉及有关 Y2K 问题的研究，对此，微软采取的是完全对外公开的策略，其中包括产品及其他任何有关 Y2K 的信息。作为一家既面向企业用户也针对广大个人用户的软件产品供应商，微软认为解决 Y2K 的首要问题是对产品进行全面深入的 2000 年兼容性测试。

首先，需要找到一种统一的方法来对不同的产品进行 2000 年兼容性测试；同时，由于微软的产品在全球得到了极为广泛的应用，因此要对产品进行不同语言的测试。至 1999 年年底，微软共

测试了 4052 种产品，这可谓迄今为止历史上最大的软件测试工程之一；其中，97%达到 2000 年兼容，3%不兼容。而不兼容的产品基本都是老产品，如 DOS 版本的 Word 5.0。同时，在整个测试过程中，并未做任何推断性测试，即不能在未对某种语言进行实际性测试的前提下，而从其他语言的同种产品的测试结果进行推断(如假设简体中文的测试结果没问题，则简单推断韩文也不存在任何问题)。英文及其他欧洲语言中只有 5%，而 20%以上的双字节语言(如大多数亚洲国家的语言)采用 UNICODE；同时，世界各地不同的民族采用不同的历法，这些都是测试需要考虑的问题。测试时尽力确保最新产品和最大量使用的产品以及采用关键技术，如 ODBC 的产品的 2000 年兼容测试，然后再来测试客户有特别需求的、采用某项专有技术的产品。

从上面的两个例子中，我们可以看出软件缺陷是造成软件故障的主要问题，也是软件测试的主要对象。那么什么是软件缺陷故障？什么是软件故障？软件测试定义是什么？它们之间的关系是什么？下面先给出一组有关软件测试的相关术语，明确它们之间的关系，进而给出软件测试的概念。

缺陷(bug)	偏差(variance)
缺点(defect)	失败(failure)
问题(problem)	矛盾(inconsistency)
错误(error)	事故(incident)
异常(anomaly)	谬误(fault)

在上述名词中，有一些名词的含义相近，属于一个范畴。第一类是缺陷、缺点和偏差，它们是一类含义相近的概念，我们不妨把它们统称为缺陷。它们都是软件开发过程潜在的缺陷，这些缺陷可能在软件投入运行后出现，使得软件的性能和可靠性等方面与系统的设计需求不符。有时，这些问题可能不出现，软件的性能和可靠性并不会因为它们的存在而受到影响。第二类是错误、谬误、问题、异常和矛盾等，我们把这一类问题统称为错误。这类错误与软件运行状态有关，它们是在软件运行过程中可观测到的软件错误。这些问题出现的原因是软件缺陷所致。第三类是失败、事故或灾难等，我们把这类统称为失败。这是软件运行给用户造成的损失的一类软件故障，它强调软件失败的结果。失败的直接原因是软件系统存在软件错误。并不是所有的软件错误都会导致软件失败，如果对软件错误加以适当的控制，软件错误可以导致安全。

综上所述，软件故障大体上可分为三种类型，每一类对应软件生命周期的不同阶段，贯穿整个软件开发和使用的全部过程。其中，第一类缺陷是软件故障的根源，后两类故障是软件缺陷的直接后果。所以，在软件开发过程中，发现和排除软件故障是一项长期艰苦的工作。而这一项工作的基础是加强软件设计时设计缺陷的检测。

那么什么是软件测试呢？所谓软件测试是为了评价一个软件系统的质量和发现错误而从事的一种工作过程。从软件测试作为软件的执行过程来看，可分为局部软件的局部运行和全部运行；从运行的环境来看，可有仿真运行和实际运行。这就存在一个软件测试中的方式和方法的问题。而方法又与采用的技术相关，技术不同，方法也不同。所以，软件测试技术是测试的关键。

从软件故障的分类中，可以看到软件的故障分布在软件开发的全过程，所以软件测试也就伴随软件开发和使用的整个过程中。在下一节中，我们将分析软件测试与软件生命周期的关系。把握软件测试与开发过程的阶段关系，为有针对性地开展软件测试奠定基础。

1.2 软件测试与软件开发过程

软件开发过程中的各种活动构成软件开发的生命周期，随着这些活动的组织方式和方法不同，就构成不同的软件开发生命周期模型。然而，无论是什么样的生命周期模型，软件开发无一例外

地要经历从软件需求分析到软件测试这样一个过程。也就是说，虽然软件开发的生命周期模型有所不同，但软件开发的阶段性始点和终点是相同的，而且软件测试是不可缺少的一项工作。

软件开发的生命周期并不是独立存在的，它是包含该软件的产品生命周期的一部分。在产品的生命周期内，软件被维护和纠错。当产品就是软件本身时，软件的维护和测试也是相当复杂的一项工作。不同的软件模块被组合成一个大的软件系统，给软件测试工作带来一定的难度。

有许多不同的软件生命周期模型，它们都需要进行测试。本节讨论各种软件开发生命周期模型与软件测试的关系，从而进一步明确软件测试在软件开发中的重要作用；为在采用不同软件开发方法的情况下，灵活运用软件测试的方法和技术奠定基础。

1.2.1　顺序生命周期模型(Sequential Lifecycle Models)

所谓的顺序生命周期模型(简称顺序模型)，是指把软件开发的整个过程定义成有序的开发活动序列，随着开发工作的进展，软件生命周期的状态也在迁移。如图 1-1、图 1-2 所示，顺序模型也称 V 模型或瀑布模型。

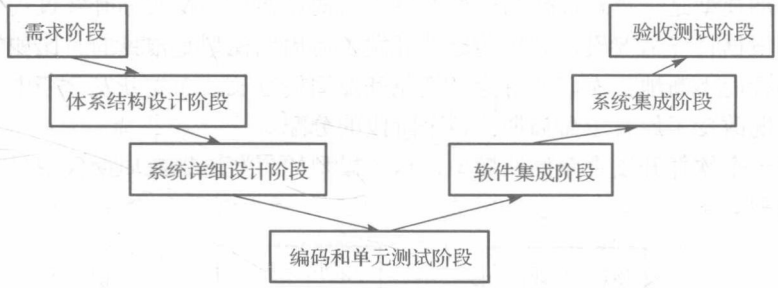


图 1-1　顺序生命周期模型

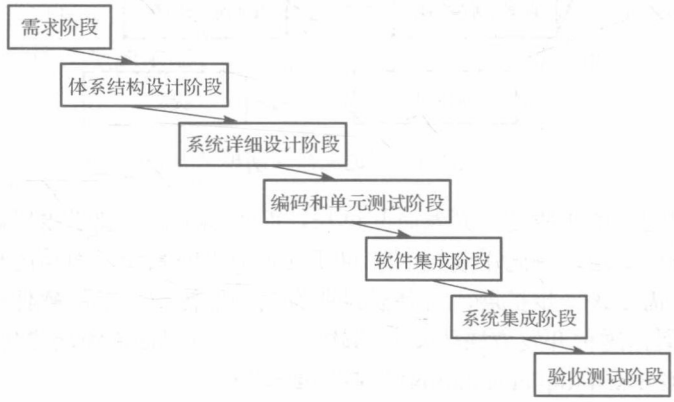


图 1-2　瀑布生命周期模型

瀑布模型也有许多变形，这些模型可能随着软件开发需求的不同，增加新的状态，这些状态有着不同的边界，下面给出一组典型的开发状态描述。

- **需求阶段(Requirements Phase)**：在需求阶段需要对用户需求进行分析，对要开发的软件进行严格的定义，这种定义应该是非二义性的。
- **体系结构设计阶段(Architectural Design Phase)**：在该阶段对软件系统的体系结构进行分析、设计和定义，进而说明体系结构中组件和组件之间的联系。

- **系统详细设计阶段(Detailed Design Phase)**：在该阶段对构成系统的各组件给出详细的设计和说明。
- **编码和单元测试阶段(Code and Unit Test Phase)**：该阶段对设计的组件进行编码，并验证组件代码与详细设计阶段定义的组件细节的正确性。
- **软件集成阶段(Software Integration Phase)**：在该阶段把已经测试过的组件组装到一起，并对集成系统进行测试，直到由组件所构成的软件满足设计要求。
- **系统集成阶段(System Integration Phase)**：在该阶段对软件与其他系统部件进行集成，并进行系统测试，直到系统正常工作为止。
- **验收测试阶段(Acceptance Test Phase)**：在该阶段将按照系统分析、设计定义的各个方法对系统进行测试，从而检验系统开发的正确性。

在上述系统生命周期中，前三个阶段是系统的定义阶段，后面四个阶段都需要对阶段成果进行测试。测试工作同样需要设计和说明，在生命周期的各个层面中，均需要定义该层面的测试点。

1.2.2 渐进(Progressive Development)生命周期模型

顺序生命周期模型是一个理性化的软件开发生命周期模型。在实际开发过程中，可能情况要复杂得多。这样根据特殊情况设计具体的软件开发生命周期模型是常见的。比如软件需求往往随着开发工作的进行而不断地扩充，或者为了避免开发周期过长，而先开发一个中间系统投入运行等，这些特殊情况改变了原有生命周期的各个阶段的分配。

下面介绍另一个软件开发生命周期模型，这个模型称为渐进生命周期模型(如图 1-3 所示)或阶段生命周期模型。

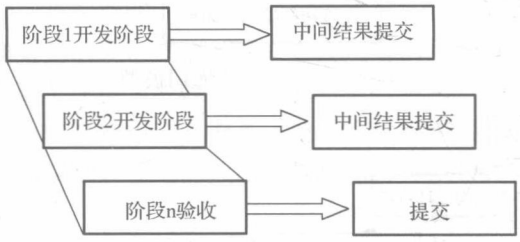


图 1-3 渐进生命周期模型

软件开发过程中的一个矛盾是，开发时间过长，而工期又往往要求很短。解决此问题的一个方法是在两者之间折中处理，首先用较短的时间开发一个中间系统，而功能相对比设计要求少一些。这个中间系统还需要进一步扩展，直到满足所有的功能需求。中间软件的开发可以有效地减少软件开发风险。我们把这种开发方法所建立的软件开发生命周期称为渐进生命周期模型(简称渐进模型)或阶段实现(Phased Implementation)生命周期模型。

在渐进模型中，每一个开发阶段仍然服从顺序模型中的规范，而整个生命周期的阶段可能有所增减，它的实际阶段取决于实际开发过程。

每一个提交的软件都要经过测试，以检验软件是否满足设计要求。测试工作分布在软件开发的各个阶段，不但要对各个阶段的成果进行测试，而且还要对各个阶段的集成成果进行检验。如果这种测试工作能够在控制范围内完成，则该工作将不会影响整个生命周期。然而，测试中发现的问题是严重的，有时需要重新开发软件。这是最坏的情况，但在这种情况下，整个软件的生命周期都将被打乱。无论是哪种情况，测试都需要花费很多时间和精力。所以我们应该尽可能及早地发现问题，减少不必要的开发代价。其中一个解决办法就是利用原型法构造系统的原型。系统

的原型是提供快速实现系统设计功能的方法，开发人员和用户可以在系统原型上测试所开发的系统是否满足设计要求。一个原型系统最终要进一步扩展成一个实际的系统，但在从原型系统到实际系统的过渡过程中，仍然需要测试。

1.2.3　迭代生命周期模型(Iterative Lifecycle Model)

在迭代生命周期模型中，开发工作的最初并不要求对软件需求进行详细的说明，而是随着软件开发工作的进行，逐渐辨别所开发软件的需求，并加以说明。上述过程可能要重复多次，产生新的软件版本。迭代模型就是这样一种开发模型，其生命周期表现为四个不断迭代的阶段，如图 1-4 所示。

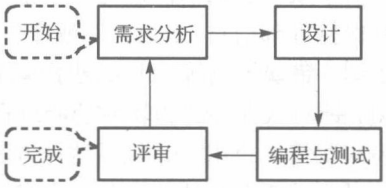


图 1-4　迭代生命周期模型

- **需求分析阶段**：该阶段对软件的需求进行收集并分析。在此基础上，不断的迭代将会得到最终的需求定义。
- **设计阶段**：该阶段根据需求进行设计，设计可能是新的设计，也可能是对早期设计的扩展。
- **编程与测试阶段**：该阶段的中心任务是对软件进行编码、集成和测试。
- **评审阶段**：在该阶段要对软件进行评估，回顾软件的需求，改变或增加新的需求。

对迭代中的每一个周期循环，必须决定软件中哪些部分或全部组件需要在下一次周期循环中被保留，或是被摒弃掉。最终将达到一个目标就是软件需求被满足，这时软件被提交。或者该软件不可能达到设计要求，而不得不从头开始。

迭代模型可以形象地比喻为通过连续的逼近方法来开发软件。这有一些像数学中的逼近法，它通过不断的逼近最终使问题求解。然而，在数学中逼近有时没有解，每一次迭代都在可行解的左右摆动，甚至是发散的。迭代的次数可能会很大，甚至是不可能的。那么在软件开发中，是不是这样呢？情况十分相似。软件中的错误会使得软件生命周期没有尽头，这也是一种发散。

在迭代模型中，成功的关键是在周期内的每一个阶段和循环中都要对结果进行严格的测试。模型的前三个阶段是顺序模型的浓缩，每一个循环中都要对软件进行单元测试，随着生命周期的延续，测试形影不离。

1.3　软件测试方法与测试内容

软件测试的种类很多，大体上可以从以下几个方面来进行划分。

- 从是否需要执行被测软件的角度，可分为静态测试和动态测试。
- 从测试是否针对系统的内部结构和具体实现算法的角度，可分为白盒测试和黑盒测试。
- 从测试范围角度，可分为单元测试、系统测试、集成测试等。
- 从测试目标角度，可分为性能测试、功能测试、可靠性测试等。
- 从测试采用的工具角度，可分自动测试和手工测试等。

软件测试的种类多种多样，测试所采用的测试的方法和技术也是多种多样的。有些测试方法是针对测试对象提出的，有些测试方法是根据测试软件的组织结构而提出的，而有些是从测试工具角度提出的。但无论是哪种方法，其核心都是比较设计需求与软件的实际执行结果的一致性、兼容性。

下面逐一介绍目前我们所听到的各种软件测试方法，通过这些方法的介绍了解软件测试的测试内容。

1.3.1 黑盒测试

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能的情况下，通过测试来检测每个功能是否都能正常使用。在测试时，把程序看成一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确地输出信息，并且保持外部信息(如数据库或文件)的完整性。黑盒测试方法主要有等价类划分、边值分析、因—果图、错误推测等，主要用于软件确认测试。“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

1.3.2 白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是在知道它产品内部工作过程的前提下，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行。按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能，白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

“白盒”法要求全面了解程序内部逻辑结构、对所有逻辑路径进行测试。“白盒”法是穷举路径测试。在使用这一方案时，测试者必须从检查程序的逻辑着手，检查程序的内部结构，得出测试数据。而贯穿程序的独立路径数是天文数字。但由于下面的原因即使每条路径都测试过，仍然可能存在错误。第一，穷举路径测试决不能查出程序违反了设计规范，即程序本身是个错误的程序。第二，穷举路径测试不可能查出程序中因遗漏路径而出现错误。第三，穷举路径测试可能发现不了一些与数据相关的错误。

1.3.3 ALAC(Act-like-a-customer)测试

ALAC 测试是一种基于客户使用产品的知识开发出来的测试方法。ALAC 测试是基于复杂的软件产品有许多错误的原则。最大的受益者是用户，缺陷查找和改正将针对哪些客户最容易遇到的错误。

1.3.4 单元测试

单元测试的对象是软件设计的最小单位——模块。单元测试的依据是详细设计描述，单元测试应对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。单元测试多采用白盒测试技术，系统内多个模块可以并行地进行测试。

1.3.5 综合测试

时常有这样的情况发生：每个模块都能单独工作，但这些模块集成在一起之后却不能正常工作。发生这种情况的主要原因是，模块相互调用时接口会引入许多新问题。例如，数据经过接口可能丢失、一个模块对另一模块可能造成不应有的影响、几个子功能组合起来不能实现主功能、误差不断积累达到不可接受的程度以及全局数据结构出现错误，等等。综合测试是组装软件的系统测试技术，按设计要求把通过单元测试的各个模块组装在一起之后，进行综合测试以便发现与接口有关的各种错误。

1.3.6　确认测试

通过综合测试之后，软件已完全组装起来，接口方面的错误也已排除，软件测试的最后一步——确认测试即可开始。确认测试应检查软件能否按合同要求进行工作，即是否满足软件需求说明书中的确认标准。

1. 确认测试标准

实现软件确认要通过一系列黑盒测试。确认测试同样需要制订测试计划和过程，测试计划应规定测试的种类和测试进度，测试过程则定义一些特殊的测试用例，旨在说明软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、准确人机界面和其他方面(例如可移植性、兼容性、错误恢复能力和可维护性等)是否令用户满意。

确认测试的结果有两种可能：一种是功能和性能指标满足软件需求说明的要求，用户可以接受；另一种是软件不满足软件需求说明的要求，用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正，因此必须与用户协商，寻求一个妥善解决问题的方法。

2. 配置复审

确认测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。

1.3.7　α、β 测试

事实上，软件开发人员不可能完全预见用户实际使用程序的情况。例如，用户可能错误地理解命令，或提供一些奇怪的数据组合，也可能对设计者自认明了的输出信息迷惑不解，等等。因此，软件是否真正满足最终用户的要求，应由用户进行一系列验收测试。验收测试既可以是非正式的测试，也可以是有计划、系统的测试。有时，验收测试长达数周甚至数月，不断暴露错误，导致开发延期。一个软件产品，可能拥有众多用户，不可能由每个用户验收，此时多采用称为 α、β 测试的过程，以期发现那些似乎只有最终用户才能发现的问题。

α 测试是指软件开发公司组织内部人员模拟各类用户对即将面市软件产品(称为 α 版本)进行测试，试图发现错误并修正。α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作并尽最大努力涵盖所有可能的用户操作方式。经过 α 测试调整的软件产品称为 β 版本。紧随其后的 β 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 β 版本，并要求用户报告异常情况、提出批评意见。然后，软件开发公司再对 β 版本进行改错和完善。

1.3.8　系统测试

计算机软件是基于计算机系统的一个重要组成部分，软件开发完毕后会与系统中其他成分集成在一起，此时需要进行一系列系统集成和确认测试。对这些测试的详细讨论已超出软件测试的范围，这些测试也不可能仅由软件开发人员完成。在系统测试之前，软件工程师应完成下列工作。

(1) 为测试软件系统的输入信息设计出错处理。

(2) 设计测试用例，模拟错误数据和软件界面可能发生的错误，记录测试结果，为系统测试提供经验和帮助。

(3) 参与系统测试的规划和设计，保证软件测试的合理性。

系统测试应该由若干个不同测试组成，目的是充分运行系统，验证系统各部件是否都能正常工作并完成所赋予的任务。系统测试中又包括恢复测试、安全测试、强度测试、性能测试、可用性测试、可靠性测试。

(1)恢复测试：恢复测试主要检查系统的容错能力，即当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。恢复测试首先要采用各种办法强迫系统失败，然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化(Reinitialization)、检查点(Checkpointing Mechanisms)、数据恢复(Data Recovery)和重新启动等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。

(2)安全测试：安全测试检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。例如，①想方设法截取或破译口令；②专门定做软件破坏系统的保护机制；③故意导致系统失败，企图趁恢复之机非法进入；④试图通过浏览非保密数据，推导所需信息，等等。理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值。此时，非法侵入者已无利可图。

(3)强度测试：强度测试检查程序对异常情况的抵抗能力。强度测试总是迫使系统在异常的资源配置下运行。例如，①当中断的正常频率为1～2h/s，运行每秒产生10个中断的测试用例；②定量地增加数据输入率，检查输入子功能的反应能力；③运行需要最大存储空间(或其他资源)的测试用例；④运行可能导致虚存操作系统崩溃或磁盘数据剧烈抖动的测试用例，等等。

(4)性能测试：对于那些实时和嵌入式系统，软件部分即使满足功能要求，也未必能够满足性能要求，虽然从单元测试起，每一测试步骤都包含性能测试，但只有当系统真正集成之后，在真实环境中才能全面、可靠地测试运行性能，系统性能测试是为了完成这一测试任务的。性能测试有时与强度测试相结合，经常需要其他软硬件的配套支持。

(5)可用性测试：对“用户友好性”的测试。显然这是主观的，且将取决于目标最终用户或客户。用户面谈、调查、用户对话的录像和其他一些技术都可使用。程序员和测试员通常都不宜作为可用性测试员。

(6)可靠性测试：可靠性测试是为了检验系统软件系统运行是否可靠而进行的一种测试。这类软件系统的失败往往导致不可预料的结果，如航空、航天领域中运行的软件，铁路系统中运行的软件等。可靠性测试的方法关注的是，一旦软件系统出现故障，其系统是否导向安全，所以可靠性测试与安全测试紧密相关。可靠性测试通常采用黑盒测试法。

1.3.9 面向对象的软件测试

面向对象的软件测试(OO Test)是根据面向对象的软件开发方法所设计的软件系统所提出的软件测试方法。OO Test又分为面向对象分析的测试(OOA Test)、面向对象设计的测试(OOD Test)和面向对象的程序测试(OOP Test)。它们是对分析结果和设计结果的测试，主要是对分析设计产生的文本进行，是软件开发前期的关键性测试。OOP Test主要针对编程风格和程序代码实现进行测试，其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用C++语言实现，主要就是对类成员函数的测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，如成员函数间的相互作用、类间的消息传递等。面向对象集成测试不但要基于面向对象单元测试，更要参见OOD或OOD Test结果。面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴OOA或OOA Test结果。