西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

第一章　软件测试基础

授课教师：　郑炜

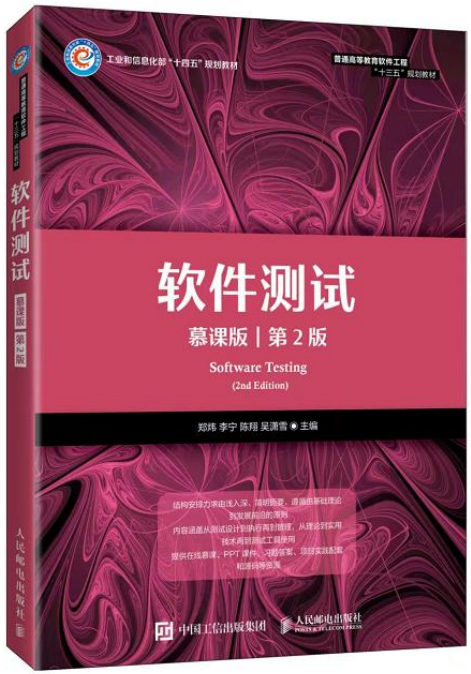
邮箱：wzheng@nwpu.edu.cn

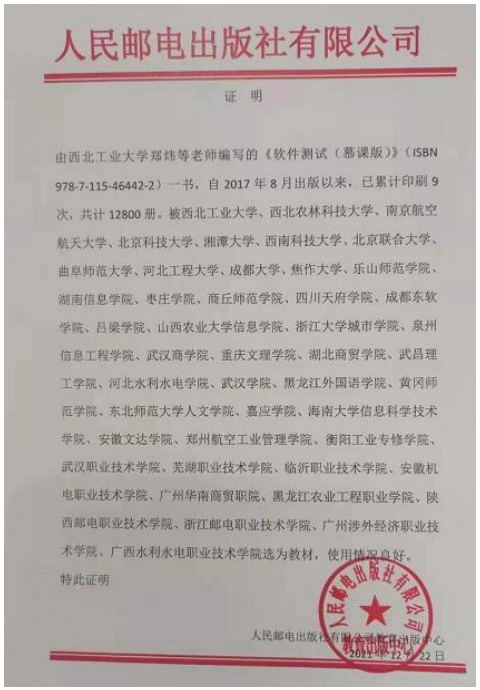
电话/微信 13991976695

1. 笔试 60%

2. 实验 20%

3. 其他 20% 包括 平时成绩，软件测试大赛等





人民邮电出版社有限公司

证 明

由西北工业大学郑炜等老师编写的《软件测试（慕课版）》（ISBN 978-7-115-46442-2）一书，自 2017 年 8 月出版以来，已累计印刷 9 次，共计 12800 册。被西北工业大学、西北农林科技大学、南京航空航天大学、北京科技大学、湘潭大学、西南科技大学、北京联合大学、曲阜师范大学、河北工程大学、成都大学、焦作大学、乐山师范学院、湖南信息学院、枣庄学院、商丘师范学院、四川天府学院、成都东软学院、吕梁学院、山西农业大学信息学院、浙江大学城市学院、泉州信息工程学院、武汉商学院、重庆文理学院、湖北商贸学院、武昌理工学院、河北水利水电学院、武汉学院、黑龙江外国语学院、黄冈师范学院、东北师范大学人文学院、嘉应学院、海南大学信息科学技术学院、安徽文达学院、郑州航空工业管理学院、衡阳工业专修学院、武汉职业技术学院、芜湖职业技术学院、临沂职业技术学院、安徽机电职业技术学院、广州华南商贸职院、黑龙江农业工程职业学院、陕西邮电职业技术学院、浙江邮电职业技术学院、广州涉外经济职业技术学院、广西水利水电职业技术学院选为教材，使用情况良好。

特此证明

人民邮电出版社有限公司教育出版中心

2021 年 12 月 22 日

课程要求

- 1、通过教学使学生了解软件测试的一般原理和方法，并掌握软件测试的策略和经典测试方法，完成软件测试用例设计、测试实验和测试结果分析与测试总结报告编写等实际软件测试工作，掌握流行的自动化测试工具。
- 2、培养目标
- 1）全栈工程师（开发者测试，具有基本测试知识的开发工程师）
- 2）专业测试人员（测试工程师，QA工程师）
- 3）软件工程研究型人才（人工智能相关的自动化测试工具开发，故障自动定位修复，安全测试攻防）
- 软件安全漏洞分析与检测、软件缺陷分析。主要是采用基于神经网络的机器学习方法(Neural Network-based Machine Learning)、交互式机器学习(Interactive Machine Learning)、主动学习（Active Learning）、迁移学习(Transfer Learning)等技术解决目前软件架构或者编码层面所造成的漏洞，并进一步分析以指导重构和漏洞修复。

- 1.1 软件测试的基本概念
  - 1.1.1 软件测试是什么
  - 1.1.2 软件测试的目的
  - 1.1.3 软件测试与软件质量保证
  - 1.1.4 软件测试的必要性
  - 1.1.5 软件测试的基本概念分析
- 1.2 软件测试的分类

- 1.3 软件缺陷管理
  - 1.3.1 软件缺陷的概念
  - 1.3.2 软件缺陷的属性
  - 1.3.3 软件缺陷生命周期
  - 1.3.4 常见的软件缺陷管理工具
- 1.4 软件质量模型与软件测试相关特性
  - 1.4.1 软件质量模型
  - 1.4.2 测试的复杂性和经济性

- 1.5 软件测试充分性和测试停止准则
  - 1.5.1 软件的测试充分性问题
  - 1.5.2 软件测试原则
  - 1.5.3 测试停止准则

1.1.1 软件测试是什么

什么是软件测试呢？

不同的人对软件测试有不同的理解。

Glenford J. Myers提出：

（1）软件测试是程序的执行过程，目的在于发现错误。

（2）软件测试是为了证明程序有错误，而不是证明程序无错误。

（3）一个好的软件测试用例在于能发现至今未发现的错误。

（4）一个成功的软件测试是发现了至今未发现的错误的测试。

Bill Hetzelt 在《软件测试完全指南》中指出：

“软件测试是以评价一个程序或者系统属性为目标的任何一种活动。软件测试是对软件质量的度量。”

1.1.1 软件测试是什么

IEEE给出了以下两个规范的软件测试的定义：

- 在特定的条件下运行系统或构件，观察或记录结果，对系统的某个方面做出评价。

- 分析某个软件项以发现和现存的，以及要求的条件之差别（即错误并评价此软件项的特性）。

1.1.2 软件测试的目的

现对软件测试的目的总结为以下3点：

（1）以最少的人力、物力、时间找出软件中潜在的各种错误和缺陷，全面评估和提高软件质量，及时揭示质量风险，控制项目风险。

（2）有助于发现开发工作中所采用的软件过程的缺陷，通过对软件缺陷进行分析，获得软件缺陷模式，有助于软件缺陷预防，以便进行软件过程改进；同时通过对软件测试结果的分析和整理，可以修正软件开发的规则，并为软件的可靠性分析提供相关的依据。

（3）评价程序或系统的属性，对软件质量进行度量和评估，以验证软件的质量能否满足用户的需求，为用户选择、接受软件提供有力的依据。

1.1.3 软件测试与软件质量保证

- 软件质量保证是**贯穿软件项目整个生命周期**的有计划的系统活动，经常针对整个项目质量计划执行情况进行评估、检查和改进，确保项目质量与计划保持一致。

- 软件质量保证确保软件项目的过程**遵循了对应的标准及规范要求，且产生了合适的文档和精确反映项目情况的报告**，其目的是通过评价项目质量建立项目达到质量要求的信心。软件质量保证活动主要包括评审项目过程、审计软件产品，就软件项目是否真正遵循已经制订的计划、标准和规程等，给管理者提供可视性项目和产品可视化的管理报告。

1.1.3 软件测试与软件质量保证

- 软件质量保证与软件测试是否是一回事？软件测试能够找出软件缺陷，确保软件产品满足需求。但是测试不是质量保证，二者并不等同。测试可以查找错误并进行修改，从而提高软件产品的质量。软件质量保证可以通过测试避免错误以求高质量，并且还有其他方面的措施以保证质量。
- 从共同点的角度看，软件测试和软件质量保证的目的都是尽力确保软件产品满足需求，从而开发出高质量的软件产品。两个流程都是贯穿在整个软件开发生命周期中的。正规的软件测试系统主要包括：

制订测试计划、测试设计、实施测试、建立和更新测试文档。

- 而软件质量保证的工作主要为：

制订软件质量要求、组织正式度量、软件测试管理、对软件的变更进行控制、对软件质量进行度量、对软件质量情况及时记录和报告。

- 软件质量保证的职能是向管理层提供正确的可行信息，从而促进和辅助设计流程的改进。软件质量保证的职能还包括监督测试流程，这样测试工作就可以被客观地审查和评估，同时也有助于测试流程的改进。

1.1.4 软件测试的必要性

软件在人们的日常生活中无处不在。从软件诞生的第一天起，软件缺陷（bug）就成为开发人员的梦魇。第一个有记载的软件缺陷是美国海军开发人员、编译器的发明者格蕾斯·哈珀（Grace Hopper）发现的。哈珀后来成为了美国海军的一位将军，还领导了知名计算机语言COBOL的开发。

1945年9月9日下午三点，哈珀中尉正带领她的团队构造一个称为“马克二型”的计算机，该计算机使用了大量的继电器。当时第二次世界大战还没有彻底结束，哈珀所在的团队夜以继日地工作。机房是一间第一次世界大战时建造的老建筑。那是一个炎热的夏天，房间没有空调，所有窗户都敞开散热。突然，马克二型死机了。技术人员尝试了很多办法，最后定位到第70号继电器出错。哈珀观察这个出错的继电器，发现一只飞蛾躺在中间，已经被继电器打死。她小心地用镊子将蛾子夹出来，用透明胶布粘到“事件记录本”中，并注明“第一个发现虫子的实例”。

1.1.4 软件测试的必要性

软件缺陷经常会给企业带来一定的经济损失，甚至有时候会带来灾难性后果。下面介绍近些年来发生的两起具有代表性的软件质量事故，并以此来强调软件测试的必要性。

1．波音公司星际客机软件故障

据国外媒体报道，美国东部时间2019年12月20日6时36分，美国波音公司的星际客机飞船从卡纳维拉尔角发射升空，6时50分飞船与火箭分离后，飞船出现软件故障，最终无法与国际空间站对接，并于12月22日7时58分提前返回地面。

2020年2月28日，波音副总裁兼星际客机项目经理约翰•穆赫兰德（John Mulholland）承认软件测试验证存在问题，对星际客机飞船的软件测试不充分。

2020年3月6日11时30分，美国国家航空航天局（National Aeronautics and Space Administration，NASA）和波音公司举行媒体电话会议，公布波音公司星际客机飞船首次轨道测试失败的调查结果。NASA和波音联合独立调查小组确定，针对软件异常提出61项纠正和预防措施。

1.1.4 软件测试的必要性

1．波音公司星际客机软件故障

火箭的发射本来一切正常，飞船按计划被送入了远地点92千米、近地点77千米的亚轨道。但是就在飞船与火箭分离后，发现星际客机飞船软件定时器异常，可以理解为飞船的星时错误，表现出的现象是星际客机飞船的48台姿轨控推力器开始疯狂工作，飞船误以为处于提升近地点的变轨过程中，在短时间内消耗了大量燃料。这一错误是由于软件编码错误，飞船在终端计数开始之前（即火箭将指定的T0设置为正确的时间）将时钟与火箭同步。

发现异常后，NASA和波音公司的飞行控制人员在休斯敦组成的联合小组注意到了这个问题，第一时间尝试向飞船注入正确指令，手动消除影响，但不凑巧的是，当时飞船正好处于两颗跟踪与数据中继卫星（Tracking and Data Relay Satellites，TDRS）的覆盖交接区，因此指令没有注入成功。等到波音公司能够发出正确地面指令，被飞船接收执行后为时已晚，星际客机飞船消耗了过多的燃料，与国际空间站的对接试验不得不取消。

2．Uber自动驾驶汽车撞死行人

软件缺陷还会造成更大的悲剧，导致生命危险。2018年3月，Uber成为第一家旗下无人驾驶汽车导致行人死亡的公司。在美国亚利桑那州坦佩市的一次测试中，无人驾驶汽车撞死了正在过马路的伊莱恩赫茨伯格（Elaine Herzberg），撞人前无人驾驶汽车未减速、转向或停车。该事故发生后，Uber将无人驾驶汽车的测试中止了8个月。此外，该起死亡事件还导致一起官司，美国好几个州也因此禁止Uber测试无人驾驶汽车。此事故成为“全球首例无人驾驶汽车致死案”，这让许多人觉得无人驾驶的幸福之路走到了尽头。

Uber无人驾驶系统采用的是福特融合系统，顶部有大量传感器，车辆周围隐藏着更多的传感器，共由6个部分组成：车顶激光雷达360°扫描单元组可以对车周围的环境进行三维扫描；朝前方的摄像阵列管控远、近区域，观察突然闯入的汽车、横跨大街的行人、交通信号灯、交通标志等；前方、后方、两侧（翼门）装有激光雷达模块，用于探测靠近汽车的物体和不容易看到的足够小的物体；侧面和后面成对的立体摄像头协助构建汽车周围持续的（实时的、动态的）影像；侧面和后方装有天线（RTA），负责GPS定位和无线数据传输；可定制的计算机和存储系统相当于指挥、控制中心，能够将这些输入的信息和资源进行汇总，进行数据的实时处理。

1.1.4 软件测试的必要性

这样一个看似强大的系统，似乎不应该发生这样的事故，但为什么发生了呢？Uber的自动驾驶汽车当时车速为61千米/小时，即这辆车是以16.9米/秒的速度前行。系统检测到人这个过程，可能需要1秒的时间，该时间包括扫描间隔时间、数据处理分析并做出正确决策的时间（摄像头捕捉到的信息需要经过复杂的计算机图像识别算法进行识别），而在夜间摄取的视频受光照影响，相对不清晰，图像越不清晰，计算的工作量就越大。也许留给刹车的时间只有2秒，但不到50米，速度又比较快，刹车距离显然不够。如果系统足够智能，判断刹车来不及，就一面刹车、一面急转弯，避免撞到行人。但是，人也不是静止不动的，当时行人看到自动驾驶汽车过来时，也会做出一定的躲闪反应，改变自己的方向，这样又给自动驾驶系统不断产生新的信息、得到新的反馈，会干扰系统做出决策，甚至人的行动方向和汽车转弯产生冲突（即没能避免碰撞），就很可能让系统重新计算，甚至算法出错，无法及时做出决策，导致悲剧。

Uber发生事故时是在进行高级别的自动驾驶测试，主要测试自动驾驶汽车在复杂交通环境下自主认知能力和多系统协同工作的安全性与稳定性。在测试过程中，不仅要考虑道路行人、参与车辆、道路基础设施及交通信号灯等基本的交通因素，而且要在开放的公共道路测试环境中，混合传统驾驶汽车及其他交通（自行车、电动车）参与者，环境复杂性进一步提高，不确定因素更多，还要保证自动驾驶汽车能够很好地融入这样的真实环境中。要完全覆盖所有的场景几乎是不可能的，这也就是我们常说的测试是不能穷尽的。

1.1.5 软件测试的基本概念分析

软件测试里经常提到3个概念：**缺陷（fault）**、**错误（error）**和**失效（failure）**。

具体来说：**缺陷对应于项目内的错误代码，有时候又称为defect或者bug**。错误是指程序在运行时，因为执行到了错误代码而造成程序内部状态出错。失效是指程序在运行结束后，其返回的实际结果与预期结果不一致。

```
public static int numZero( int[] x ){
   int count = 0;
   for ( int i=1; i < x.length; i++ ){
      if ( x [i] == 0 ){
         count ++;
      }
   }
   return count;
}
```

图1-2 一段包含缺陷的代码

1.1.5 软件测试的基本概念分析

我们通过图1-2所示的一段包含缺陷的简单代码对这3个概念做深入的分析。这是一个编程新手经常容易犯的一个错误，其缺陷在第3行，正确的代码应该是“for (int i=0; i<x.length, i++)”。假设我们设计测试用例的输入为“x={1,0,2}”，虽然该输入执行到了缺陷代码，但并未造成程序错误（即变量count的取值没有出错），同时该测试用例的实际输出结果与预期输出结果保持一致，即没有产生失效，因此可见该测试用例无法检测出该程序内的缺陷。通过分析这段代码，我们可以设计出另外一个测试用例，其输入为“x={0,1,2}”，运行该测试用例会造成变量count的取值出错，即造成错误。该错误通过内部传播，会最终影响到程序的输出，并造成程序的实际输出与预期输出不一致，即产生失效。

1.1.5 软件测试的基本概念分析

基于上述分析，我们会思考一个问题，如果被测程序内部包含缺陷，是不是所有测试用例都能触发这个缺陷并产生失效呢？RIP模型针对该问题，给出了答案。

RIP模型认为要确保测试用例触发外在失效，需要满足以下3个必要条件。

- **可达性（Reachability）**：测试用例必须执行到包含缺陷的语句。
- **传染性（Infection）**：缺陷语句执行后必须导致程序内部状态出现错误（即某些变量的取值出现错误）。
- **传播性（Propagation）**：不正确的程序内部状态通过传播语句导致程序的实际输出与预期输出不一致。

- 1.1 软件测试的基本概念
- 1.2 软件测试的分类

1.2 软件测试的分类

目前，软件测试领域有许多测试名称，这些名称来自不同的分类原则，以下是常见的测试分类方式。

- 按测试阶段或测试步骤划分
- 按测试对象划分
- 按使用的测试技术划分
- 按软件质量特性划分
- 按照测试项目划分

1.2 软件测试的分类

- 按测试阶段或测试步骤划分

按测试阶段或测试步骤划分，软件测试分为单元测试、集成测试、确认测试和系统测试。

在确认测试中，按照测试的方式又分为Alpha测试和Beta测试。

- 按测试对象划分

按测试对象划分，软件测试分为单元测试、配置项测试、部件测试和系统测试。

1.2 软件测试的分类

- 按使用的测试技术划分

按使用的测试技术划分，软件测试分为静态测试和动态测试，这两种测试分别代表了程序不同的运行状态。动态测试又分为白盒测试和黑盒测试，白盒测试包括逻辑覆盖测试、域测试、程序变异测试、路径测试、符号测试等，黑盒测试包括功能测试、强度测试、边界值测试、随机测试等。

- 按软件质量特性划分

按软件质量特性划分，软件测试分为功能性测试、可靠性测试、易用性测试、效率测试、维护性测试和可移植性测试。

1.2 软件测试的分类

功能性测试又包含适合性测试、准确性测试、互操作性测试、安全保密性测试、功能性的依从性测试；可靠性测试包含容错性测试、成熟性测试、易恢复性测试、可靠性的依从性测试；易用性测试包括易理解性测试、易学习性测试、易操作性测试、吸引性测试、易用性的依从性测试；效率测试包括时间特性测试、资源利用率测试、效率的依从性测试；维护性测试包括易改变性测试、稳定性测试、易测试性测试、易分析性测试、维护性的依从性测试；可移植测试包括适应性测试、易安装性测试、易替换性测试、共存性测试和可移植性的依从性测试。

● 按照测试项目划分

按照测试项目划分，软件测试分为以下几类：

（1）功能测试：主要针对软件/产品需求规格说明的测试，验证功能是否符合需求，如检验原定功能、是否有冗余的功能等。

（2）健壮性测试：侧重于软件容错能力的测试，主要是验证软件对各种异常情况（如数据边界、非法数据、异常中断等）是否进行正确处理。

（3）恢复测试：对每一类导致恢复或重构的情况进行测试，验证软件自身运行的恢复或重构、软件控制系统的恢复或重构，以及系统控制软件的恢复或重构。

（4）人机界面测试：对人机界面提供的操作进行测试，测试人机界面的有效性、便捷性、直观性等，如人机界面是否友好、是否方便易用、设计是否合理、位置是否准确等。

（5）接口测试：测试被测对象与其他软件（包括软件单元、部件、配置项）或硬件的接口。

（6）强度测试：使软件在其设计能力的极限状态下，以及超过此极限下运行，检验软件对异常情况的抵抗能力。

（7）可用性测试：对“用户友好性”的测试。可用性测试受主观因素影响，且取决于最终用户。用户面谈、调查和其他技术都可在该测试中使用。

（8）压力测试：对系统不断施加压力的测试，通过确定一个系统的瓶颈或者不能接受的性能点，获得系统能提供的最大服务级别的测试。例如，测试一个Web站点在大量的负荷下，何时系统的响应会退化或失败。压力测试注重的是外界不断施压。

（9）性能测试：测试软件是否达到需求规格说明中规定的各类性能指标，并满足相关的约束和限制条件。

（10）兼容测试：测试软件在一个特定的硬件、软件、操作系统或网络等环境下的性能如何。

（11）用户界面测试：对系统的界面进行测试，测试用户界面是否友好、是否方便易用、设计是否合理、位置是否正确等一系列界面问题。

（12）安全性测试：测试软件在没有授权的内部或者外部用户攻击、恶意破坏时如何进行处理，是否能保证软件和数据的安全。

（13）可靠性测试：这里指的是比较狭义的可靠性测试，它主要是对系统能否稳定运行进行估计。

（14）安装测试：安装测试主要检验软件是否可以正确安装、安装文件的各项设置是否有效、安装后是否影响原系统、卸载后是否删除干净和是否影响原系统。

（15）文档测试：测试开发过程中生成的文档，以需求规格说明、软件设计、用户手册、安装手册等为主，检验文档是否与实际存在差别。文档测试不需要编写测试用例。

1.2 软件测试的分类

软件测试的这些类别之间有着密切的关系，体现在以下几个方面。

（1）在软件开发过程中，不同阶段的测试对应了对不同软件对象的测试，这种对应关系如图 1-3所示。

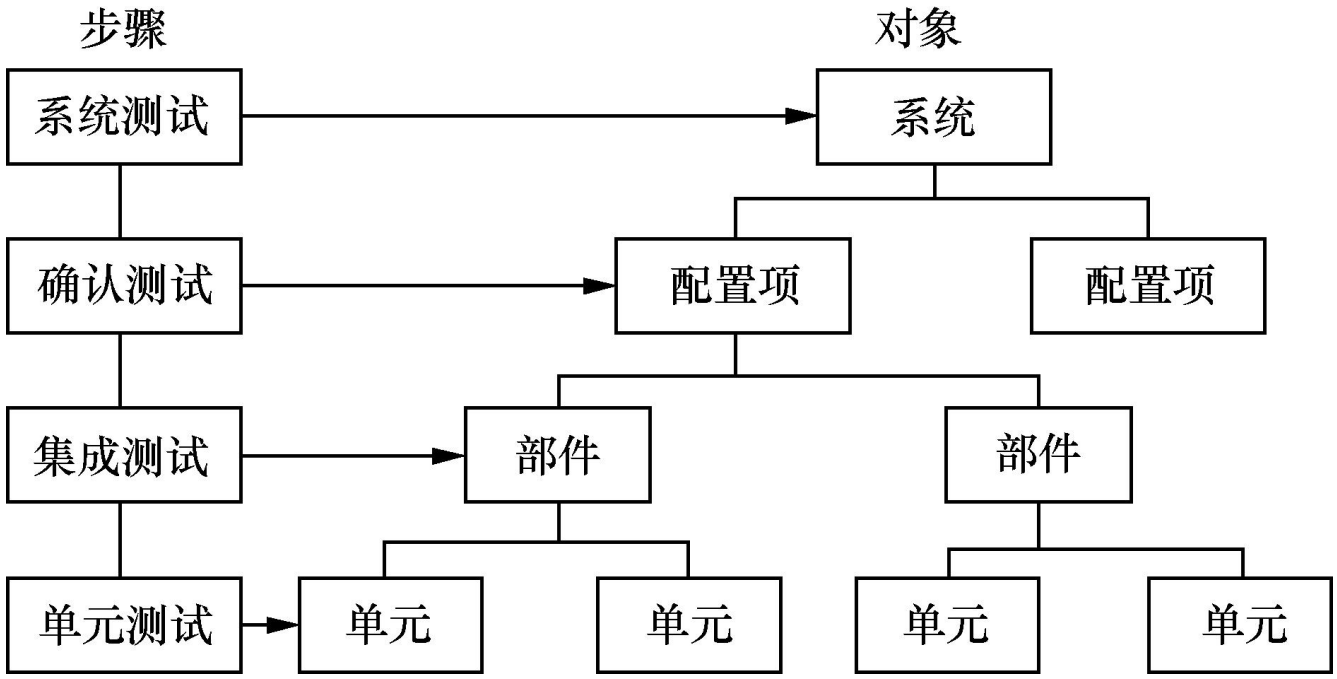


图1-3 软件测试的步骤和对象

1.2 软件测试的分类

（2）在不同的测试阶段，由于测试目标、对象、要求的不同而采用不同的测试技术。一般情况下不同测试对象采用的测试技术如表1-1所示。

表 1-1　测试对象和测试技术

测试对象	测试技术
单元测试	黑盒测试、白盒测试、静态测试
部件测试	黑盒测试、白盒测试、静态测试
配置项测试	黑盒测试、白盒测试
系统测试	黑盒测试

（3）在不同的测试阶段对不同对象的测试包含不同的测试项目。例如，确认测试可包含功能测试、性能测试、人机界面测试；组合测试可包括接口测试；系统测试可包括可靠性测试、强度测试等。同时，对各阶段和对象的测试完整性要求也不同。

- 1.1 软件测试的基本概念
- 1.2 软件测试的分类
- 1.3 软件缺陷管理
  - 1.3.1 软件缺陷的概念
  - 1.3.2 软件缺陷的属性
  - 1.3.3 软件缺陷生命周期
  - 1.3.4 常见的软件缺陷管理工具

1.3.1 软件缺陷的概念

一般看来，满足以下的任意一种情况都可以称为软件缺陷。

（1）软件未达到产品说明书中标明的功能。

（2）软件出现了产品说明书中指明的不应该出现的功能。

（3）软件功能超出了产品说明书中指明的范围。

（4）软件未达到产品说明书中指明的应达到的目的。

（5）软件难以理解和使用、运行速度慢或最终用户认为不好。

1.3.2 软件缺陷的属性

通常情况下，软件缺陷单需要包含以下几种内容。

（1）软件缺陷标识（Identifier）：软件缺陷标识是标记某个软件缺陷的一组符号。每个软件缺陷必须有唯一的标识。

（2）软件缺陷类型（Type）：软件缺陷类型是根据软件缺陷的自然属性划分的软件缺陷种类。软件缺陷类型通常分类情况如表1-2所示。

表 1-2 软件缺陷类型

序号	类别	描述
1	界面	界面错误，如界面显示不符合需求、提示信息不合规范等
2	功能	系统功能无效、不响应、不符合需求
3	性能	系统响应过慢、无法承受预期负荷等
4	安全性	存在安全隐患的软件缺陷
5	数据	数据导入或设置不正确

1.3.2 软件缺陷的属性

（3）软件缺陷严重程度（Severity）：软件缺陷严重程度是指因软件缺陷引起的故障对软件产品的影响程度，如表1-3所示。

表 1-3 软件缺陷严重程度

序号	软件缺陷严重程度	描述
1	严重软件缺陷	不能执行正常工作功能或重要功能，或者危及人身安全、系统安全
2	较大软件缺陷	严重地影响系统要求或基本功能的实现，且没有办法更正（重新安装或重新启动该软件不属于更正办法）
3	较小软件缺陷	影响系统要求或基本功能的实现，但存在合理的更正办法（重新安装或重新启动该软件不属于更正办法）
4	轻微软件缺陷	使操作者不方便或遇到麻烦，但它不影响执行工作功能或重要功能
5	其他软件缺陷	其他错误

1.3.2 软件缺陷的属性

（4）软件缺陷优先级（Priority）：软件缺陷优先级是指软件缺陷必须被修复的紧急程度，如表1-4所示。

表 1-4 软件缺陷优先级

序号	优先级	描述
1	立即解决	严重阻碍测试进行，且没有方法绕过去
2	高优先级	严重影响测试进行，但是有可选方案绕过该功能进行其他内容测试
3	正常排队	软件缺陷需要正常排队等待修复或列入软件发布清单
4	低优先级	软件缺陷可以在方便时纠正

1.3.2 软件缺陷的属性

（5）软件缺陷状态（Status）：软件缺陷状态是指软件缺陷通过一个跟踪修复过程的进展情况，如表1-5所示

表 1-5 软件缺陷状态

序号	软件缺陷状态	描述
1	提交（Submitted）	已提交软件缺陷
2	打开（Open）	确认待处理的软件缺陷
3	已拒绝（Rejected）	被拒绝处理的软件缺陷
4	已解决（Resolved）	已修复的软件缺陷
5	已关闭（Closed）	确认已解决的软件缺陷
6	重新打开（Reopen）	修复验证不通过，被重新打开的软件缺陷

1.3.2 软件缺陷的属性

（6）软件缺陷起源（Origin）：软件缺陷起源是指软件缺陷引起的故障或事件第一次被检测到的阶段，如表1-6所示。

表 1-6 软件缺陷起源

序号	描述
1	由于需求阶段引起的软件缺陷
2	由于构架阶段引起的软件缺陷
3	由于设计阶段引起的软件缺陷
4	由于编码阶段引起的软件缺陷
5	由于测试阶段引起的软件缺陷

1.3.2 软件缺陷的属性

（7）软件缺陷来源（Source）：软件缺陷来源是指引起软件缺陷的起因，如表1-7所示。

表 1-7 软件缺陷来源

序号	描述
1	由于需求的问题引起的软件缺陷
2	由于构架的问题引起的软件缺陷
3	由于设计的问题引起的软件缺陷
4	由于编码的问题引起的软件缺陷
5	由于测试的问题引起的软件缺陷

1.3.3 软件缺陷生命周期

在软件开发过程中，软件缺陷拥有自身的生命周期，软件缺陷在走完其生命周期后最终会关闭。确定的软件缺陷生命周期保证了测试过程的标准化，软件缺陷在其生命周期内会处于许多不同的状态，如图1-4所示。

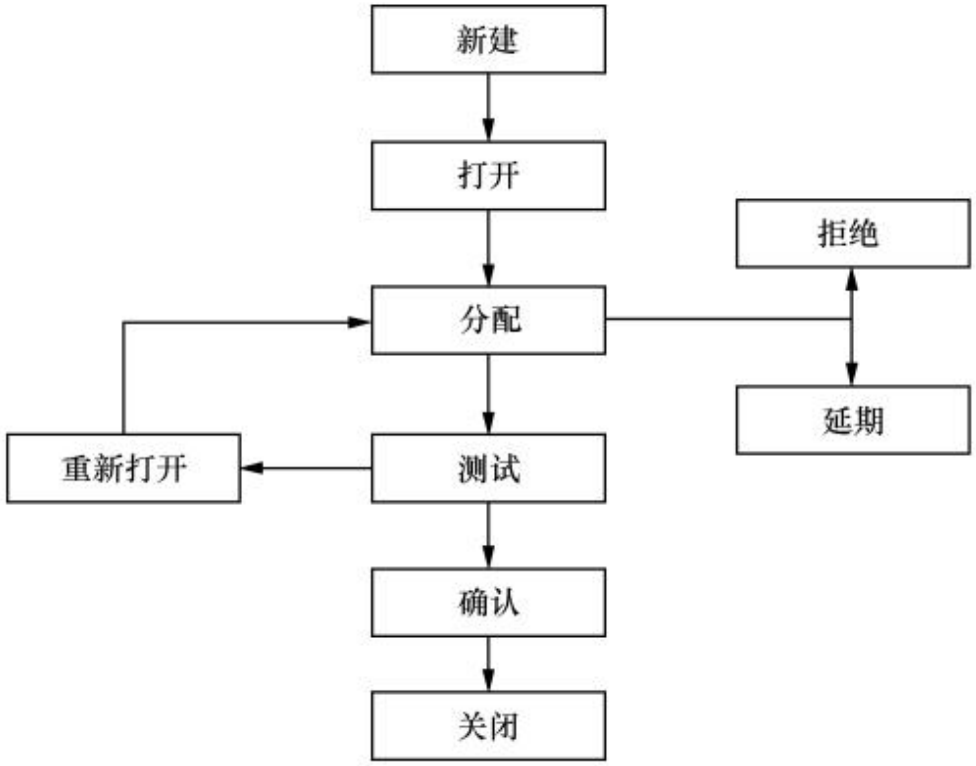


图1-4　软件缺陷的生命周期

1.3.4 常见的软件缺陷管理工具

常见的软件缺陷管理工具

- Bugzilla
- BugFree
- Quality Center
- JIRA
- Mantis
- LaunchPad

1.3.4 常见的软件缺陷管理工具

- **Bugzilla**

  Bugzilla是由Mozilla公司提供的一个开源的、免费的软件缺陷跟踪工具。作为一个软件缺陷记录及跟踪工具，它能够建立一个完善的缺陷跟踪体系，该体系包括报告缺陷、查询缺陷记录并产生报表、处理解决、管理员系统初始化和设置4个部分。Bugzilla的详细使用介绍可参考8.4节的内容。

- **BugFree**

  BugFree是借鉴微软公司的研发流程和缺陷管理理念，使用PHP+MySQL独立写出的一个缺陷管理系统。该系统简单实用、免费，并且开放源代码（遵循GNU GPL）。命名“BugFree”有两层意思：一是希望软件中的缺陷越来越少，直到没有；二是免费且开放源代码，大家可以自由使用、传播。

1.3.4 常见的软件缺陷管理工具

- **Quality Center**

  Quality Center是一个基于Web的商业测试管理工具，可以组织和管理应用程序测试流程的所有阶段，如制订测试需求、计划测试、执行测试和跟踪软件缺陷。此外，通过Quality Center还可以创建报告和图来监控测试流程。

- **JIRA**

  JIRA是Atlassian公司出品的项目与事务跟踪工具，被广泛应用于软件缺陷跟踪、客户服务、需求收集、流程审批、任务跟踪、项目跟踪和敏捷管理等工作领域。JIRA配置灵活、功能全面、部署简单、扩展丰富。JIRA是比较流行的基于Java架构的管理系统，由于Atlassian公司对很多开源项目实行免费提供软件缺陷跟踪服务，因此在开源领域，JIRA的被认知度比其他测试产品要高得多，而且易用性也好一些。

1.3.4 常见的软件缺陷管理工具

- **Mantis**

  Mantis是一个基于PHP技术的轻量级软件缺陷跟踪系统，其功能与前面提及的JIRA系统类似，都是以Web操作的形式提供项目管理及软件缺陷跟踪服务。在功能上可能没有JIRA那么专业，界面也没有JIRA美观，但在实用性上足以满足中小型项目的管理及跟踪需求。更重要的是其开源，用户不需要负担任何费用。不过目前的版本还存在一些问题，期待在今后的版本中能够得以完善。

- **LaunchPad**

  LaunchPad是由Ubuntu的母公司Canonical公司资助架设的网站，最初是一个提供维护、支持或联络Ubuntu开发者的平台。后来越来越多的项目使用该系统进行软件缺陷跟踪管理，如云平台基础架构OpenStack。

- 1.1 软件测试的基本概念
- 1.2 软件测试的分类
- 1.3 软件缺陷管理
- 1.4 软件质量模型与软件测试相关特性
  - 1.4.1 软件质量模型
  - 1.4.2 测试的复杂性和经济性

1.4.1 软件质量模型

软件质量是软件的生命，它直接影响软件的使用和维护。通常从以下几个方面评价软件质量的优劣。

- 软件需求是衡量软件质量的基础，不符合需求的软件就不具备质量。设计的软件应在功能、性能等方面都符合要求，并能可靠地运行。
- 软件结构良好，易读、易于理解，并易于修改、维护。
- 软件系统具有友好的用户界面，便于用户使用。
- 软件生存周期中各阶段文档齐全、规范，便于配置、管理。

1.4.1 软件质量模型

如何评定一个软件呢？

最通用的一个规范就是使用ISO/IEC 9126-1991标准规定的软件质量度量模型。它不仅对软件质量做了定义，还涉及整个软件测试的一些规范流程和测试计划的撰定、制订以及测试用例的设计。

ISO/IEC 9126-1991标准规定的软件质量度量模型由3层组成，其中第1层称为质量特性，第2层称为质量子特性，第3层称为度量，如图1-5所示。

1.4.1 软件质量模型

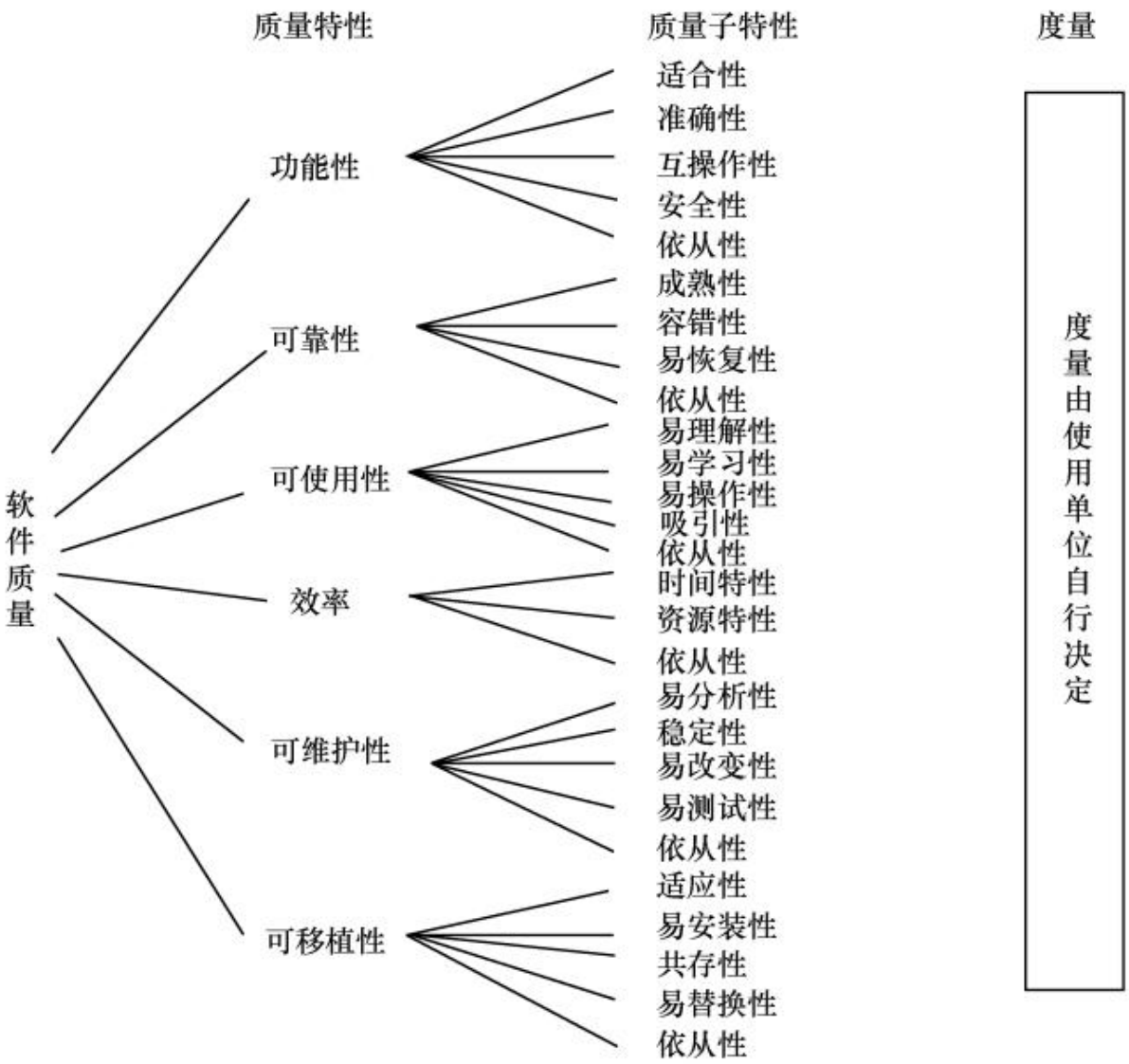


图1-5 ISO软件质量度量模型

其中对质量特性和质量子特性做如下详细说明。

- **功能性：是指当软件在指定条件下使用时，软件产品满足明确和隐含的功能的能力。**

① 适合性：是指软件产品与指定的任务和用户目标提供一组合适的功能的能力。

② 准确性：是指软件产品具有所需精确度的正确或相符的结果及效果的能力。

③ 互操作性：是指软件产品与一个或多个规定系统进行交互的能力。

④ 安全性：是指软件产品保护信息和数据的能力，以使未授权的人员或系统不能阅读、修改这些信息和数据，但不拒绝授权人员或系统对其的访问。

⑤ 依从性：是指软件产品遵循与同功能性相关的标准、约定或法规，以及类似规定的能力。

1.4.1 软件质量模型

●**可靠性：是指当软件在指定条件下使用时，软件产品维持规定的性能级别的能力。**

① 成熟性：是指软件产品避免因软件中错误发生而导致失效的能力

② 容错性：是指在软件发生故障或违反指定接口的情况下，软件产品维持规定的性能级别的能力。

③ 易恢复性：是指在失效发生的情况下，软件产品重建规定的性能级别并恢复受直接影响的数据的能力。

④ 依从性：是指软件产品遵循与同可靠性相关的标准、约定或法规，以及类似规定的能力。

●可使用性：是指当软件在指定条件下使用时，软件产品被理解、学习、使用和吸引用户的能力。

① 易理解性：是指软件产品使用户能理解它是否合适，以及如何能将软件用于特定的任务和使用环境的能力。

② 易学习性：是指软件产品使用户能学习它的能力。

③ 易操作性：是指软件产品使用户能操作和控制它的能力。

④ 吸引性：是指软件产品吸引用户的能力。

⑤ 依从性：是指软件产品遵循与同易用性相关的标准、约定、风格指南或法规，以及类似规定的能力。

1.4.1 软件质量模型

●效率：是指在规定条件下，相对于所用资源的数量，软件产品可以提供适当的性能的能力。

① 时间特性：是指在规定条件下，软件产品执行其功能时，可以提供适当的响应时间和处理时间，以及吞吐率的能力。

② 资源特性：是指在规定条件下，软件产品执行其功能时，可以提供合适的数量和类型的资源的能力。

③ 依从性：是指软件产品遵循与同效率相关的标准或约定的能力。

1.4.1 软件质量模型

●**可维护性：是指软件产品可被修改的能力，修改可能包括修正、改进或软件适应环境、需求和功能规格说明中的变化。**

① 易分析性：是指软件产品诊断软件中的缺陷或失效原因，以及判定待修改部分的能力。

② 稳定性：是指软件产品避免由于软件修改而造成意外结果的能力。

③ 易改变性：是指软件产品使指定的修改可以被实现的能力。

④ 易测试性：是指软件产品使已修改软件能被确认的能力。

⑤ 依从性：是指软件产品遵循与同维护性相关的标准或约定的能力

●可移植性：是指软件产品从一种环境迁移到另一种环境的能力。

① 适应性：是指软件产品无须采用有别于为考虑该软件的目的而准备的活动或手段，就能够适应不同的指定环境的能力。

② 易安装性：是指软件产品在指定环境中被安装的能力。

③ 共存性：是指软件产品在公共环境中同与其分享公共资源的其他独立软件共存的能力。

④ 易替换性：是指软件产品在环境相同、目的相同的情况下替代另一个指定软件产品的能力。

⑤ 依从性：是指软件产品遵循与同可移植性相关的标准或约定的能力。

1.4.2 测试的复杂性和经济性

● 测试的复杂性

例如，要测试一个三角形程序，该程序完成下述功能。

输入3个整数a、b和c，作为三角形的3条边，通过程序判断由这3条边构成的三角形类型是等边三角形、等腰三角形还是一般三角形，并输出相应的信息。

写出自己认为合适的测试输入，然后根据测试输入回答下面的问题，每回答1个“是”加1分，最后看看能得多少分。

1.4.2 测试的复杂性和经济性

（1）是否设计了一种测试输入表示合法的一般三角形？

注意，像(1,2,3)和(2,3,9)这样的测试输入不应该回答“是”，读者可以想想为什么。

（2）是否设计了一种测试输入表示合法的等边三角形？

（3）是否设计了一种测试输入表示合法的等腰三角形？

注意，像(4,4,8)这样的测试输入不应该回答“是”，因为不存在这样的三角形。

（4）是否至少设计了3种测试输入表示合法的等腰三角形，由此检查了2条边相等的3种排列方案？如(3,3,4)、(4,3,3)、(3,4,3)。

（5）是否设计了这样的测试输入，其中三角形的一条边长为0？

1.4.2 测试的复杂性和经济性

（6）是否设计了一种测试输入，其中3个整数都大于0，而其中的2个数之和等于第3个数？

注意，如果把(2,3,5)当成一个一般三角形，则表明程序中有故障。

（7）是否至少设计了3种第6个问题那样的测试输入，检查1条边边长等于另外2条边边长之和的3种排列方案？如(2,3,5)、(5,2,3)、(3,5,2)。

（8）是否设计了一种测试输入，表示3个整数都大于0，而其中某2个数的和小于第3个数？

注意，如果把(2,3,9)当成一个一般三角形，则表明程序中有故障。

（9）是否至少设计了3种第8个问题那样的测试输入，检查了1条边小于另外2条边之和的3种排列方案？如(2,3,9)、(2,9,3)、(9,2,3)。

1.4.2 测试的复杂性和经济性

（10）是否设计了一种测试输入，表示3条边边长都为0，即(0,0,0)？

（11）是否设计了这样的测试输入，其中三角形的一条边长为负数？

（12）是否至少设计了一种测试输入，其中三角形的边长不是整数？

（13）是否至少设计了一种测试输入，其中三角形的边数不是3？例如，给出2条边或4条边。

（14）对于每一种测试输入，是否还给出了预期的输出？

当然，满足上面条件的一组测试输入不能保证查出所有可能的故障，但由于问题（1）～问题（13）代表了该程序实际上可能发生的故障，对程序进行充分的测试应该能检查出这些故障。你得了多少分？一个经验丰富的专业程序开发人员平均只得7.8分（满分14分）。这表明，即使像上面这样简单的程序测试，也不是一件容易的事。何况要测试一个具有十多万条语句的空中交通管制系统，一个编译程序，甚至一个普通的工资开放软件呢？可见，软件测试是一项复杂而艰巨的任务，需要系统地学习、训练和实践。

黑盒测试的复杂性

黑盒测试是一种常用的软件测试方法，在应用这种方法设计测试用例时，测试人员把被测程序看成是一个打不开的黑盒，在不考虑程序内部结构和内部特性的情况下，只根据需求规格说明书，设计测试用例，检查程序的功能是否按照规范说明的规定正确地执行。

如果希望利用黑盒测试方法查出软件中的所有故障，只能采用穷举输入测试。所谓穷举输入测试，就是把所有可能的输入全部都用作测试输入。

例如，要对Microsoft Windows计算器进行测试。

检验了1+1是等于2后，绝不能保证Windows计算器能正确地进行所有的加法运算。很可能当进行1024+1024的运算时，计算结果不正确。由于把被测程序看成了一个黑盒子，所以发现问题的唯一途径只能是测试每一种输入情况。

要穷尽地测试图1-6所示的Windows计算器，就得考虑所有可能的合法输入。

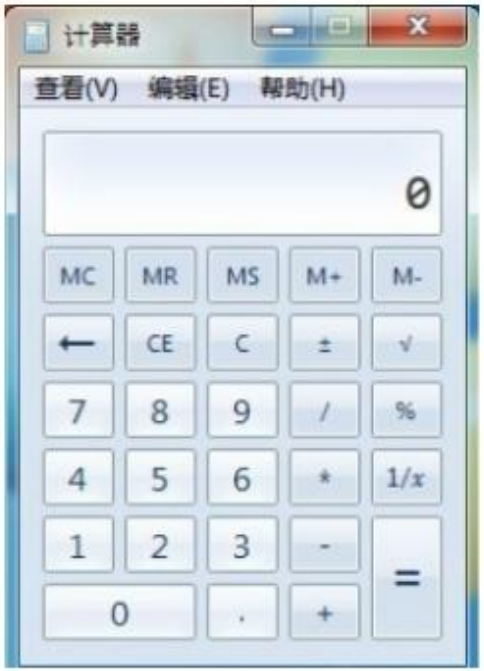


图1-6 Windows计算器

白盒测试的复杂性

黑盒穷举测试不现实，那么，另一种常用的测试方法白盒测试是否可以做到穷举测试呢？白盒测试又称结构测试或基于程序的测试，该方法将被测对象看作一个打开的盒子，允许人们检查其内部结构。测试人员根据程序内部的结构特性，设计和选择测试用例，检测程序的每条路径是否都按照预定的要求正确地执行。

对于没有学过软件测试的人来说，或许认为使程序中每条路径至少执行一次就做到了穷举测试。然而，程序的路径数量可能是个天文数字。

1.4.2 测试的复杂性和经济性

下面来看一个非常简单的程序，并假定程序中所有判断都是相互独立的。它的程序控制流程图如图1-7所示。

图 1-7 中的每个节点代表一条语句，每条边或弧则表示两条语句间的控制转移。该图描述了一个由10～20条语句构成的程序，其中含有一个最多重复20次的循环语句，而在循环体内，则有一些嵌套的条件语句。那么从A点走到B点的所有路径数有520+519+…+51，其中5是贯穿循环体的路径数。大多数人都难以想象这么大的数据是什么概念，可以这样设想：如果每5分钟可以写出、执行并验证一个测试用例，那么测试完所有路径大概要花10亿年。

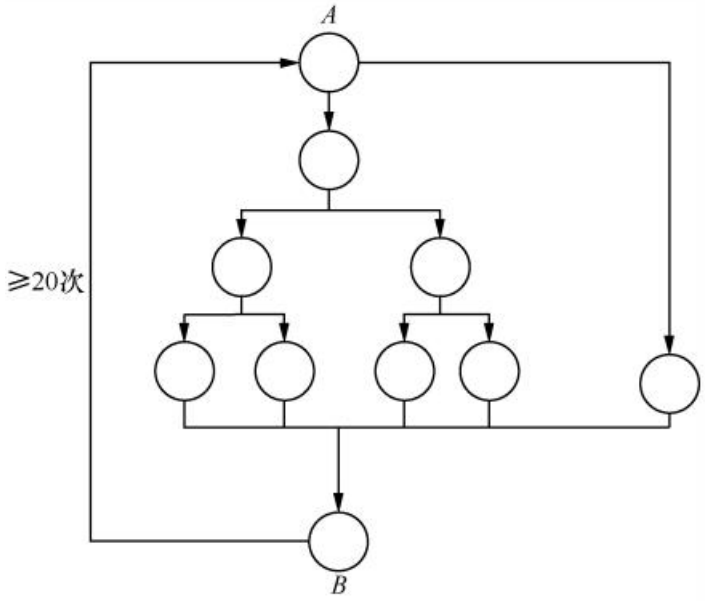


图1-7　程序控制流程图

1.4.2 测试的复杂性和经济性

即使程序中每条路径都测试过了，仍不能保证程序没有故障，原因有以下3点。

（1）穷举路径测试不能保证程序实现符合规格说明的要求。例如，如果要求编写升序排序程序，结果程序被错误地编写成降序排序程序。这时，穷举路径测试是毫无用处的。

（2）穷举路径测试不可能查出程序中因遗漏路径而出现的错误。

（3）穷举路径测试可能发现不了有关数据的故障。例如，假定程序要求实现：

if(abs(x-y))<€

而实际程序则写成：

if(x-y)<€

…

显然，这个语句有错，但使用穷举路径测试并不一定能发现这个错误。

因此，无论穷举输入测试还是穷举路径测试，都不可能对被测程序进行彻底的测试。艾兹格·迪杰斯特拉（E.W.Dijkstra）的一句名言对测试的不彻底性做了很好的注释：“软件测试只能证明故障的存在，但不能证明故障不存在”。

- 1.1 软件测试的基本概念
- 1.2 软件测试的分类
- 1.3 软件缺陷管理
- 1.4 软件质量模型与软件测试相关特性
- 1.5 软件测试充分性和测试停止准则
  - 1.5.1 软件的测试充分性问题
  - 1.5.2 软件测试原则
  - 1.5.3 测试停止准则

1.5.1 软件的测试充分性问题

测试充分性问题是软件测试的另一个重要问题

1．测试充分性准则

测试充分性准则用来评价一个测试数据集（测试输入数据的集合）按照规范说明测试被测软件是否充分。

测试的充分性也可以根据“覆盖率”这一概念进行衡量。覆盖率至少可以通过两种方式来测定。一种方式是基于软件规格说明，测试检测了其中的多少需求；另一种方式是基于程序源代码，测试检测了其中的多少行代码、多少条语句或多少条路径等。这两种方法也反映了两种基本的测试方法—基于规范的测试（黑盒测试）方法和基于结构的测试（白盒测试）方法。

1.5.1 软件的测试充分性问题

测试充分性准则是在测试之前，由相关各方根据质量、成本和进度等因素规定的，表现为对测试的要求与软件需求和软件实现有关，具有以下的一些基本性质。

（1）空测试对于任何软件测试都是不充分的。

（2）对任何软件都存在有限的充分测试数据集，这一性质称为有限性。

（3）如果一个测试数据集对一个软件系统的测试是充分的，那么再增加一些测试用例也是充分的，这一性质称为单调性。

（4）软件越复杂，需要的测试用例就越多，这一性质称为复杂性。

（5）测试得越多，进一步测试所能得到的充分性增长就越少，这一性质称为回报递减律。

2．测试数据集充分性公理

Weyuker（韦约克）将公理系统应用到软件测试的研究中，给出了以下**4**条基于程序的测试数据集充分性公理。

公理**2.1**（非外延性公理） 如果有两个功能相同而实现不同的程序，对其中一个是充分的测试数据集对另一个不一定是充分的。

公理**2.2**（多重修改公理） 如果两个程序具有相同的语法结构，对一个是充分的测试数据集对另一个不一定是充分的。

1.5.1 软件的测试充分性问题

公理2.3（不可分解公理） 对一个程序进行了充分的测试，并不表示对其中的成分都进行了充分的测试。

公理2.4（非复合性公理） 对程序各单元是充分的测试数据集并不一定对整个程序（集成后）是充分的。

1.5.2 软件测试原则

1. 完全测试程序是不可能的

理想情况下，测试所有可能的输入，将提供程序行为最完全的信息，但这往往是不可能的。例如，一个程序若有输入量X和Y及输出量Z，在字长为32的计算机上进行。如果X、Y为整数，按功能测试法穷举，测试数据有232×232=264个。如果测试一组数据需要1毫秒，一年工作365天×24小时，那么完成所有测试需5亿年。

如果由于某些原因将一些测试输入去掉，比如，认为测试条件不重要或者为了节省时间，那么测试就不是完全测试。主要有以下3个方面的原因。

（1）程序的输入量太大。

（2）程序的输出量太多。

（3）软件实现的途径太多。

2. 软件测试是有风险的

如果决定不测试所有的情况，那么就意味着选择了风险。

不能做到完全测试，不测试又会漏掉一些软件故障。测试的目标应该是使有限的测试投资获得最大的收益，即以有限的测试用例检查出尽可能多的软件故障。如果试图测试所有情况，费用将大幅度增加，而漏掉软件故障的数量并不会因费用上涨而显著下降。如果减少测试或者错误地确定测试对象，那么费用很低，但是会漏掉大量软件故障。因此，软件测试的一个主要原则是如何把无边无际的可能输入减少到可以控制的范围，以及如何针对风险制订出一些明智抉择，去粗存精，找到最合适的测试量，使测试做得不多不少。

3. 测试无法找到隐藏的软件故障

在防疫检查工作中，如果检查人员发现一匹马身上或者周围有寄生虫迹象，就可以判断这匹马感染了寄生虫。如果检查人员对另一匹马进行检查，没有找到寄生虫迹象或者找不到这匹马被感染的征兆，也许发现了一些死虫或者废弃的洞穴，但是无法证实有活的寄生虫存在。检查结果无法肯定这匹马没有感染寄生虫，只能说明没有发现活的寄生虫存在。软件测试工作与防疫检查工作极为相似，通过软件测试可以查找并报告发现的软件故障，但是不能保证软件故障全部被找到，也无法报告隐藏的软件故障。继续测试，可能还会发现一些软件故障。

4. 存在的故障数量与发现的故障数量成正比

经验表明，测试后程序中残存的故障数量与该程序中已发现的故障数量成正比，其中原因可能是以下3种。

（1）程序员怠倦。程序员编写一天代码或许情绪还不错，第2天、第3天可能就会烦躁不安了。一个软件故障很可能是暴露附近更多软件故障的信号。

（2）程序员往往犯同样的错误。每个人都有自己的偏好，一个程序员总是反复犯自己容易犯的错误。

（3）某些软件故障可能是冰山一角。某些看似无关的软件故障可能是由一个极其严重的错误造成的。

5. 杀虫剂现象

1990年，鲍里斯·贝泽尔（Boris Beizer）在其编著的《软件测试技术（第二版）》一书中引用了“杀虫剂现象”一词，用于描述软件测试进行得越多，其程序免疫力越强的现象。这与农药杀虫类似，常用一种农药，害虫最后就有抵抗力，农药发挥不了多大的效力。

为了避免杀虫剂现象的发生，应该根据不同的测试方法开发测试用例，对程序的不同部分进行测试，以找出更多的软件故障。

6. 并非所有软件故障都能修复

不修复软件故障的原因可能有以下4种。

（1）没有足够的时间。软件产品开发中，常常在项目进度中没有为测试留出足够的时间，而软件又必须按时交付。

（2）修复风险太大。在软件测试中，这种情况很常见。软件本身很脆弱，修复一个软件故障可能导致其他软件故障的出现。在紧迫的产品发布和进度压力之下，修改软件将冒很大的风险。在某些情况下，暂时不去理睬软件故障，以避免出现新的软件故障或许是一个可选的安全之道。

（3）不值得修复。不常出现的软件故障和在不常用功能中出现的软件故障可以暂不修复；可以躲过和用户有办法预防或避免的软件故障通常也可以不修复。

（4）不算真正的软件故障。在某些特殊场合，错误理解、测试错误或者产品规格说明书变更可以把软件故障当作附加的功能而不当作故障来对待。

7. 一般不要丢弃测试用例

在使用交互系统进行软件测试时，常常出现这样的情况：测试人员坐在计算机前，编写出一些测试用例并用它们对被测程序进行测试，当再次测试程序时（例如，改错后或改进程序后），就得重新编写测试用例。由于重新编写测试用例需要大量的工作，测试人员多半要回避它。因此，对程序的重新测试很少能像原来那样严格。这意味着，如果对程序的修改使原先能正确运行的部分出现了故障，那么这个故障常常发现不了。因此，除非确实没有用，测试人员一般不要丢弃测试用例。

8. 应避免测试自己编写的程序

开发和测试是两个不同的活动。开发是创造或者建立一个模块或整个系统的过程；而测试是为了发现一个模块或者系统中存在故障，不能正常工作的过程。这两个活动之间有着本质的区别，而一个人也不可能把两个截然对立的开发人员和测试人员角色都扮演好。

除了这个心理原因，还有一个重要的原因：程序中可能包含有程序员对问题的叙述或说明的误解而产生的故障。但这并不是说程序员不可能测试自己的程序。只是相比之下，如果由他人来进行测试，可能会更有效、更成功。

9. 软件测试是一项复杂且具有创造性的和需要高度智慧的挑战性任务

以前，软件产品较小，也不太复杂，即使出现软件故障，也很容易修复，不需付出多少代价和破坏性。但是，随着软件规模和复杂性的增加，测试一个大型软件所要求的创造力，可能超过设计那个软件所要求的创造力。现在，生产低质软件的代价太高了，软件行业也发展到强制使用软件测试人员的时代。尽管软件测试不可能发现软件中的所有故障，尽管有一些方法可用来指导测试用例的开发，但使用这些方法仍然需要很大的创造力。

1.5.3 测试停止准则

下面就给出一些实用的停止测试的标准。

第1类标准：测试超过了预定的时间，停止测试。

第2类标准：执行了所有测试用例但没有发现故障，停止测试。

第3类标准：使用特定的测试用例方法作为判断测试停止的基础。

第4类标准：正面指出测试完成的要求，如发现并修改70个软件故障。

第5类标准：根据单位时间内查出故障的数量决定是否停止测试。