计算机软件测试技术

郑人杰

华大学出版社

计算机软件测试技术

郑人杰　主编

清华大学出版社

内 容 提 要

软件测试是软件开发的重要环节，也是保证软件质量的重要手段。本书系统地介绍了软件测试的基本概念、常用的测试方法。内容包括软件错误分析，软件质量保证，程序排错方法，软件测试策略，黑盒测试与白盒测试方法，验收测试与测试文档，测试工具与测试环境，以及程序正确性证明。

本书为读者提供了大量的参考文献。附录中还给出了软件审查用表的格式及软件测试用语的解释。

本书可作为大专院校计算机有关专业师生的教学参考书，也可供计算机软件科研人员、开发人员和计算机用户参考。

(京)新登字 158 号

计算机软件测试技术

郑人杰 主编

☆

清华大学出版社出版

北京 清华园

中国科学院印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/16 印张：17 字数：400 千字

1992 年 12 月第 1 版 1992 年 12 月第 1 次印刷

印数：00001～10000

ISBN 7-302-01042-0/TP・382

定价：10.00 元

前　言

正如任何生产过程离不开产品质量检验一样，在计算机软件的开发中测试工作是必不可少的一步。软件测试工作做得怎样，决定着软件产品质量的好坏。然而，人们重视软件测试的原因还不仅于此。事实说明，在软件测试阶段投入的成本和工作量往往要占软件开发总成本和总工作量的一半，甚至更多。尽管这样，测试技术仍然一直落后于人们对它的期望。到现在为止作为软件工程学科中的一个分支，它远未成熟。不仅测试理论，而且已有的测试方法都无法满足当前软件开发的实际需求。这就要求我们密切关注它的发展，继续研究它的规律。

十分遗憾的是在国内大量的出版物中，至今很少见到有关软件测试的技术资料。几年前我们翻译的 Myers 著作(《计算机软件测试技巧》，清华大学出版社，1985)。在普及测试技术方面起过一定作用。近年来，在完成教学和科研任务的过程中，我们围绕着软件测试开展了调查、研究、分析和实践活动，同时也积累了一些资料。我们愿意把这些资料整理出来，与同行们共享，使其发挥应有的作用。

由于本书编写的时间十分紧迫，有些部分深入研究得不够，其中的错误和不妥之处在所难免。我们真诚地希望读者批评指正，批评意见请寄北京清华大学软件技术中心(邮政编码：100084)。

参加本书编写工作的还有姜凡(第一章的软件测试发展的历史回顾、第四章的功能测试、第五章的测试覆盖准则、域测试及程序变异)，徐仁佐(第九章的程序可靠性模型及软件可靠性在软件测试中的应用)，马素霞(第八章程序正确性证明)，殷人昆(第四章的正交实验设计法)和刘挺(第七章 COSTE 简介)。郑晨参加了部分书稿的誊写。在此对他们的辛勤劳动和密切配合表示衷心的感谢。

郑人杰

1990 年底于清华园

目　　录

第一章 绪 论

本世纪中计算机刚一问世，尽管当时还没有软件这一名称，但软件的重要组成部分——计算机程序就开始为我们服务了。然而，计算机软件作为一种人类创造的复杂逻辑实体和人们长期以来已经熟悉的大多数产品有着许多不同的特点。认识和掌握这些特点需要大量的实践和研究。只是70年代以来形成了软件工程的概念，才使软件产业得以初步建立。

软件测试是保证软件质量的重要手段。本章将对软件生存期、软件测试的目的和意义、软件测试的发展简史以及应该如何正确对待软件测试的心理学等问题作一概括的描述。其目的在于，使读者在着手研究和掌握具体的测试技术以前，对软件测试建立起正确的、全面的基本概念；同时澄清在用户甚至在计算机界中仍然流行着的一些错误的和有害的观点。

为了便于读者进一步地研究，本章提供了十分详尽的参考资料目录。

1.1 软件危机和软件生存期

计算机软件在近30多年来经历了曲折的发展道路。在这期间值得一提的是在60年代出现的"软件危机"。

我们知道，随着计算机硬件技术的进步，计算机的元器件质量逐步提高，整机的容量、运行速度以及工作可靠性有了明显的提高。与此同时，硬件生产的成本降低了。计算机价格的下跌为它的广泛应用创造了极好的条件。在此形势下自然要求软件与之相适应。一方面适应高速度、大容量、高可靠度的高性能硬件；另一方面要适应在广泛应用情况下出现的大型、复杂问题对软件技术提出的迫切需求。然而，事实上软件技术的发展未能满足这些需求。和硬件技术的快速发展相比，软件的确大大地落后了。多年来由于问题未得到及时解决，致使矛盾日益尖锐。这些矛盾归结起来主要表现在以下几个方面：

① 由于缺乏大型软件开发的经验和软件开发数据的积累，使得开发工作的计划很难制定。主观盲目地制定计划，执行起来和实际情况有很大差距，使得经费使用常常突破预算。由于工作量的估计不够准确，进度计划难以遵循，开发工作完成的期限一再拖延。延期的项目，为了尽快赶上去，便要增加人力，结果适得其反，不仅进度未能加快，反而更加延误了。在这种情况下，软件开发的投资者和用户对开发工作从不满意发展到不信任。

② 作为软件设计依据的软件需求，在开发的初期提得不够明确，或是未能做出确切的表达。开发工作开始后，软件人员又未能和用户及时交换意见，使得一些问题得不到及时解决而隐藏起来，造成开发后期矛盾的集中暴露。这时对多个错综复杂的问题既难于分析，又难于解决。

③ 开发过程中没有遵循统一的、公认的方法论或是开发规范，参加工作的人员之间

的配合不够严密，约定不够明确。加之不重视文字资料工作，使得开发文档很不完整。发现了问题，未能从根本上去找原因，只是修修补补。显然，这样开发出的软件无法维护。

④ 缺乏严密有效的软件质量检测手段，交付给用户的软件质量差，在运行中暴露出各种各样的问题。在各个应用领域的不可靠软件，可能带来不同程度的严重后果。轻者影响系统的正常工作，重者发生事故，甚至酿成生命财产的重大损失。

这些矛盾表现在具体的软件开发项目上。最为突出的实例便是美国 IBM 公司在 1963 年至 1966 年开发的 IBM 360 机操作系统。这一项目在开发期中每年花费五千万美元，总共投入的工作量为 5 千人-年。参加工作最多时有 1 千人。总共写出了一百万行源程序。尽管如此多的开销，却拿不到开发成果。这是一次失败的记录。项目负责人 F. P. Brooks 事后总结了他在组织开发过程中的沉痛教训，写成《神秘的人-月》一书。这个反映软件危机的典型事例成为软件技术发展过程中一个重要的历史性标志。

从上述软件危机的现象和发生危机的原因分析，要摆脱危机不是一件简单的事，要从多方面着手解决，把握好软件的特点，抓住它与其它产业部门工作对象的相同与相异之处加以对比分析，排除人们的一些传统观念和某些错误认识是非常重要的。

除去那些规模很小的项目以外，通常开发一个软件要在不同层次的多个开发人员的配合与协作中完成；开发各阶段之间的工作应当有严密的衔接关系；开发工作完成以后，软件产品应该面向用户，接受用户的检验。所有这些活动都要求人们根本改变那种把软件当作个人才智产物的看法，抛弃那些只按自己的工作习惯，不顾与周围其它人员密切配合的作法。事实上，这里所列举的情况与研制计算机硬件，甚至与完成一项建筑工程项目并没有本质的差别。任何参加工程项目的人员，他的才能只能在工程的总体要求和技术规范的约束下充分发挥和施展。既然我们已经积累了几千年的工程学知识，能不能把它运用在软件开发工作上呢？实践表明，按工程化的原则和方法组织软件开发工作是有效的，也是摆脱软件危机的一个主要出路。

参照工程学的概念，研究软件工程工作的特点进一步改变了原来受到束缚的传统观念。当我们全面分析软件开发工作的各个"工序"时，认识到程序编写只是整个工作的一部分。在它的前后还有更重要的工序。正如同其它事物一样，从它的发生、发展到达成熟阶段，以至老化和衰亡，有一个历史发展的过程，任何一个计算机软件有它的生存期（Life Cycle）。这个生存期包括 6 个步骤，即：

① 计划（Planning）

② 需求分析（Requirement Analysis）

③ 设计（Design）

④ 程序编写（Coding）

⑤ 测试（Testing）

⑥ 运行与维护（Run and Maintenance）

这些步骤的主要任务是：

① 计划：确定软件开发的总目标；给出软件的功能、性能、可靠性以及接口等方面的设想。研究完成该项软件任务的可行性，探讨问题解决的方案；对可供开发使用的资源（如计算机硬、软件、人力等）、成本、可取得的效益和开发的进度作出估计；制定完成开发

任务的实施计划 (Implementation Plan)。

② 需求分析：对开发的软件进行详细的定义，由软件人员和用户共同讨论决定，哪些需求是可以满足的，并且给予确切的描述；写出软件需求说明书 (Software Requirement Specifications) 或称软件规格说明书，以及初步的用户手册 (System User's Manual)，提交管理机构审查。

③ 软件设计： 设计是软件工程的技术核心。在设计阶段应把已确定的各项需求转换成相应的体系结构，在结构中每一组成部分是功能明确的模块。每个模块都能体现相应的需求。 这一步称为概要设计 (Preliminary Design)。 进而进行详细设计 (Detail Design)，即对每个模块要完成的工作进行具体的描述，以便为程序编写打下基础。上述两步设计工作均应写出设计说明书，以供后继工作使用并提交审查。

④ 程序编写：把软件设计转换成计算机可以接受的程序，即写成以某个程序设计语言表示的源程序清单。这一工作也称为编码。当然，写出的程序应该是结构良好、清晰易读，且与设计相一致的。

⑤ 测试： 测试是检验开发工作的成果是否符合要求，它是保证软件质量的重要手段。通常测试工作分为三步，即：

单元测试 (Unit Testing)——单独检验各模块的工作。

集成测试 (Integrated Testing)——将已测试的模块组装起来进行检验。

确认测试 (Validation Testing)——按规定的需求，逐项进行有效性测试，以决定开发的软件是否合格，能否提交用户使用。

⑥ 运行和维护： 已交付用户的软件投入正式使用以后便进入运行阶段。这阶段可能持续若干年，甚至几十年。在运行中可能有多种原因需要对它进行修改。其原因包括：运行中发现了软件中有错误，需要修正；为了适应变化了的软件工作环境，需要作相应的变更；为进一步增强软件的功能，提高它的性能，使它进一步完善和扩充。

以上 6 步表明每个软件从它的酝酿开发，直至使用相当长时间以后，被新的软件代替而退役的整个历史过程。按此顺序逐步转变的过程可用一个软件生存期的瀑布模型加以形象化描述。图 1.1 给出了这一瀑布模型。 从图中可看出，从左上到右下如同瀑布流水

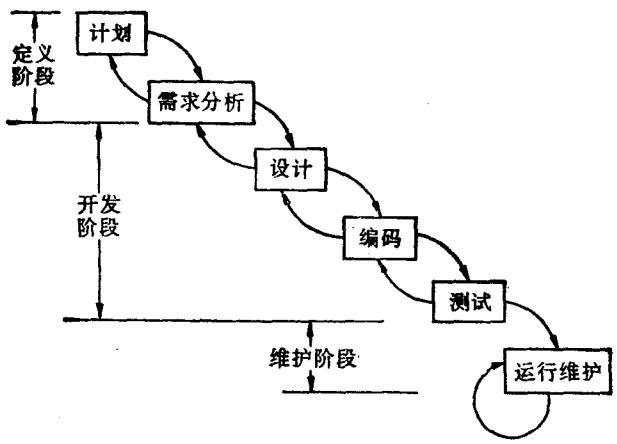

图 1.1 软件生存期的瀑布模型

逐级下落。在最后的运行中可能需要多次维护，图中用环形箭头表示。此外，在实际的项目开发中，为了确保软件的质量，每一步骤完成以后都要进行复查，如果发现了问题就要及时解决，以免问题积压到最后造成更大的困难。每一步骤的复查及修正工作正是图中给出的向上箭头。此外，图中还指明了6个步骤划分的3个阶段：软件定义阶段、软件开发阶段及软件维护阶段。

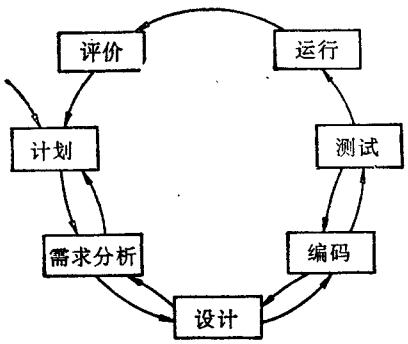

图 1.2　软件生存周期

值得注意的是，上述软件维护工作不可简单地看待。原因在于维护工作不仅仅是修改程序。在软件运行的过程中若有必要修改，得提出充分的修改理由，经过审核，才能肯定下来。接着需要经历制订修改计划、确定新的需求、修改软件设计、修改编码、进行测试以及重新投入运行等一系列步骤。这些步骤正是上述开发一个新软件的步骤。若是运行中多次提出修改，将多次经历这些步骤。我们把这一过程用图1.2来表示，并称此为软件生存周期。实际上软件生存周期和软件生存期表达的是同一内涵，为简便起见通常只称为软件生存期。

1.2　软件测试的意义

软件测试在软件生存期中占有非常突出的重要位置，究其原因是多方面的。

根据 Boehm 的统计，软件开发总成本中，用在测试上的开销要占 30% 到 50%。如果我们把维护阶段也考虑在内，讨论整个软件生存期，开发测试的成本比例会有所降低，但不要忘记，维护工作相当于二次开发，乃至多次开发，其中必定也包含有许多测试工作。因此，有人估计软件工作有 50% 的时间和 50% 以上的成本花在测试工作上。

软件测试究竟是什么意思，它包括哪些工作？这些问题常常被一般人误解，甚至从事计算机工作的人员也可能弄不清楚。

测试（Test）一词最早出于古拉丁字，它有“罐”或“容器”的含义。人们当时用一种特殊的容器检验金属中含有某种元素是多少。到现在测试一词已经普遍使用了，在工业生产和制造业中测试被当作一个常规的生产活动，它常常和产品的质量检验密切相关。在这些行业中测试的含义似乎是明确的，但在计算机软件领域内则不然。比如，什么是程序测试，什么是软件测试，它们之间有什么差别？测试和调试是一回事吗？它们之间又有什么差别？对于这些概念的不同解释可能会涉及到测试的目的和方法。

“程序测试”的说法最早几乎是和“程序编写”同时出现的。从当时的观点来看，谁写出的程序，谁去测试它，谁去使用它，测试被当作编写程序以后很自然要做的一步工作。并且在测试时不仅要发现程序里的错误，而且要排除错误。这时测试与调试混为一谈，完全不加区分。一段时间里人们谈论和写文章所涉及到的测试一词，实际上是调试即排错（debug）（请参看本节最后两段）。其实这两者是完全不同的两个概念。我们还注意到，那时的计算机多用来完成数值计算任务。程序运行后所得的计算结果常常采用以手

工计算其中一部分数据的办法来比较，从而判断程序的正确性。这在当时还是简单易行的。但随着计算机应用领域的拓广，所写程序要解决的问题也不仅仅限于数值计算了。上述手工计算进行校验的办法难于适应了，然而对于程序正确性检验的基本模式并没有改变。这就是给出测试数据，运行被测程序，将所得结果与预期结果进行比较，从而判断程序的正确性。我们称此为程序正确性测试(参看图 1.3)。

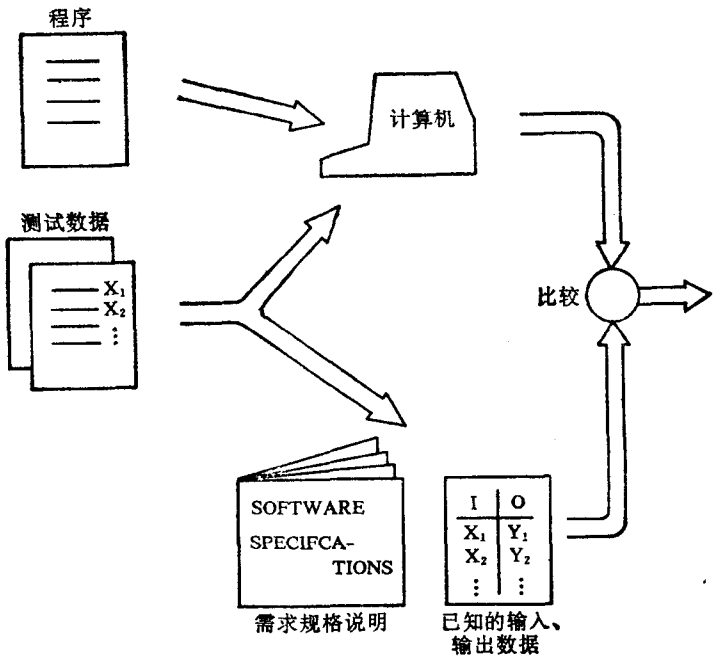

图 1.3　程序正确性测试

70 年代中以来，形成了软件生存期概念。这时人们对于软件测试的认识更广泛也更深刻了。这对于软件产品的质量保障以及组织好软件开发工具有着重要的意义。首先，由于能够把整个开发工作明确地划分成若干个开发步骤，就把复杂的问题按阶段分别加以解决。使得对于问题的认识与分析、解决的方案与采用的方法以及如何具体实现在各个阶段都有着明确的目标。其次，把软件开发划分成阶段，就对中间产品给出了若干个监控点，提高了开发过程的可见度，为各阶段实现目标的情况提供了检验的依据。各阶段完成的软件文档成为检验软件质量的主要对象。这时对软件质量的判断决不只限于程序本身。即使只谈程序本身的正确性，它也和编码以前所完成的需求分析及软件设计工作进行得如何密切相关。很显然，表现在程序中的错误，并不一定是编码所引起的。很可能是详细设计、概要设计阶段，甚至是需求分析阶段的问题引起的。因此，即使针对源程序进行测试，所发现的问题其根源可能在开发前期的各个阶段。解决问题、纠正错误也必须追溯到前期的工作。正是考虑到这一情况，我们通常把测试阶段的工作分成若干步骤进行。这些步骤包括：模块测试，集成测试、确认测试和系统测试。对程序的最小单

位——模块进行测试时，检验每个模块能否单独工作，从而发现模块的编码问题和算法问题；进而将多个模块联结起来，进行集成测试，以检验概要设计中对模块之间接口设计的问题；确认测试则应以需求规格说明书中的规定作为检验尺度，发现需求分析的问题；最后进行的系统测试是将开发的软件与硬件和其它相关因素（如人员的操作，数据的获取等）综合起来进行全面的检验，这样的作法必将涉及到软件的需求以及软件与系统中其它方面的关系。图1.4给出了测试工作与软件开发前期工作的关系。图中开发工作是自上而下进行的，而几种不同的测试都会涉及到前期工作的不同阶段。

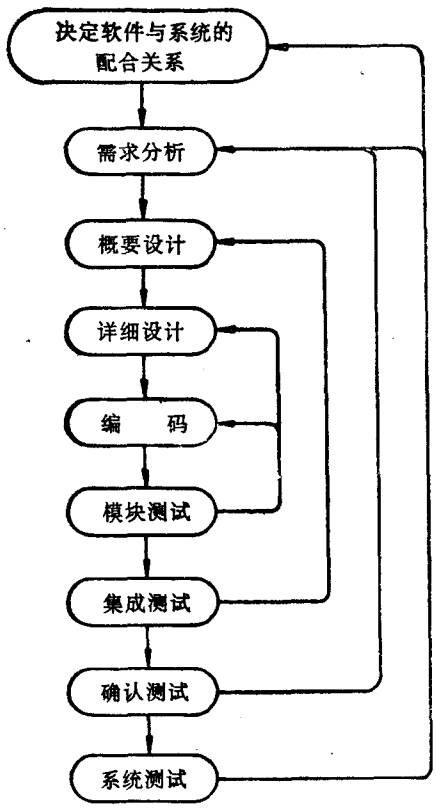

图1.4　测试与开发前期工作的关系

如果我们着眼于整个软件生存期，特别是着眼于编码以前各开发阶段的工作来保证软件的质量，就需要突破原来对测试的理解。也就是在开发的过程中，需要不断地复查与评估、不断地进行检验，以利于把发现的错误和问题得到及时的解决，而不让这些错误和问题隐藏起来，影响后期的开发工作。总之，贯穿在整个开发各阶段的复查、评估与检测活动，远远地超出了程序测试的范围，可以统称为确认、验证与测试活动（V，V&T——Validation, Verification and Testing）有时为了方便，也简称为测试，不过这是广义的测试概念。

所谓确认，它是指如何决定最后的软件产品是否正确无误。比如，编写出的程序相对于软件需求和用户提出的要求是否符合，或者说程序输出的信息是用户所要的信息吗？这个程序在整个系统的环境中能够正确稳定地运行吗？这里自然包含了对软件需求满足程度的评价。在软件产品开发完成以后，为了对它在功能、性能、接口以及限制条件等方面是否满足需求作出切实的评价，需要在开发的初期，在软件需求规格说明书中明确地规定确认的标准。

所谓验证，它是指如何决定软件开发的每个阶段、每个步骤的产品是否正确无误，并与其前面的开发阶段和开发步骤的产品相一致。验证工作意味着在软件开发过程中开展一系列活动，旨在确保软件能够正确无误地实现软件的需求。

确认和验证是有联系的，但也有明显的差别。Boehm 在1981年是这样来描述两者差别的：确认要回答的是：我们正在开发一个正确无误的软件产品吗？而验证要回答的是：我们正开发的软件产品是正确无误的吗？

总之，确认、验证与测试在整个软件开发过程中作为质量保证的手段，应当最终保证软件产品的正确性、完全性和一致性。我们可以把确认、验证与测试活动分为三类（见图

1.5)：

① 完整性检验——验证每一开发阶段（或开发步骤）中产品的完整性；分析每一产品，确保其内部的一致与完全。例如，分析需求规格说明书，以找出不一致的需求或矛盾的需求。比如，其中规定输出报告，但该报告所需要的数据是无法得到的。

② 进展检验——保证各个开发阶段（或开发步骤）之间其规格说明书的完全性和一致性。例如，后一阶段的工作确是前阶段工作的进一步细化。

③ 适用性与充分性检验——把取得的结果与对问题的理解作比较，保证所完成的结果是必要而充分的解。

在整个软件生存期各阶段中确认、验证与测试活动包括：

① 需求分析阶段

• 制定项目的 V,V&T 计划：确定 V,V&T 的目标；安排 V,V&T 的活动；选择采用的方法和工具；制定进度并作出预算。

• 确定与测试用例相关的需求：这些需求构成了一组基础的测试用例。从而有助于澄清并且确定软件需求的可度量性，同时也作为验收测试的基础。

• 复审并分析需求：其目的在于确保规定的需求能够对整个问题取得有指望的和可用的结果，针对问题叙述的清晰性、完全性、正确性、一致性、可测试性和可跟踪性进行复审。

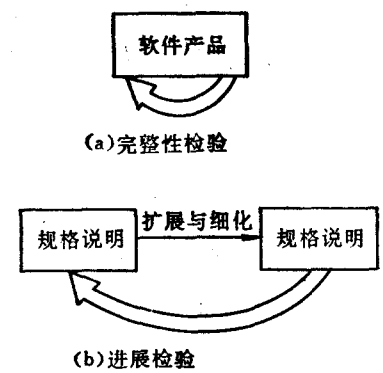

图 1.5 三类确认、验证与测试活动

② 概要设计阶段

• 复审并修订 V,V&T 计划。

• 针对要执行的逻辑功能而生成的测试数据，补充软件需求。

• 复审并分析概要设计：确保内部的一致性、完全性、正确性及清晰性；检验已进行的设计是否满足需求。

③ 详细设计阶段

• 针对功能测试数据进行设计：设计要考虑到基于系统物理结构的测试数据。

• 复审并分析详细设计规格说明：确保内部的一致性、完全性、正确性及清晰性；检验详细设计是否对概要设计做出了正确无误的细化；确认所做的设计满足于需求。

④ 编码及测试阶段

• 完成测试用例规格说明：针对编码过程中对设计的修改补充或修正测试用例规格说明。

• 复审、分析并测试程序：检查是否遵循了编码标准；自动或手工分析程序；运行测试用例，以保证满足验收要求。

• 进行产品验收。

⑤ 运行及维护阶段

·软件评估：对软件的运行情况作出评估，以保证它能继续满足用户的需求。

·软件修改评估：当提出修改的要求时也要进行评估，修改以后要进行复审和测试，以确保修改正确无误。

·回归测试：重新运行前已正确无误的测试用例，以便消除由于软件修改而带来的各种错误。

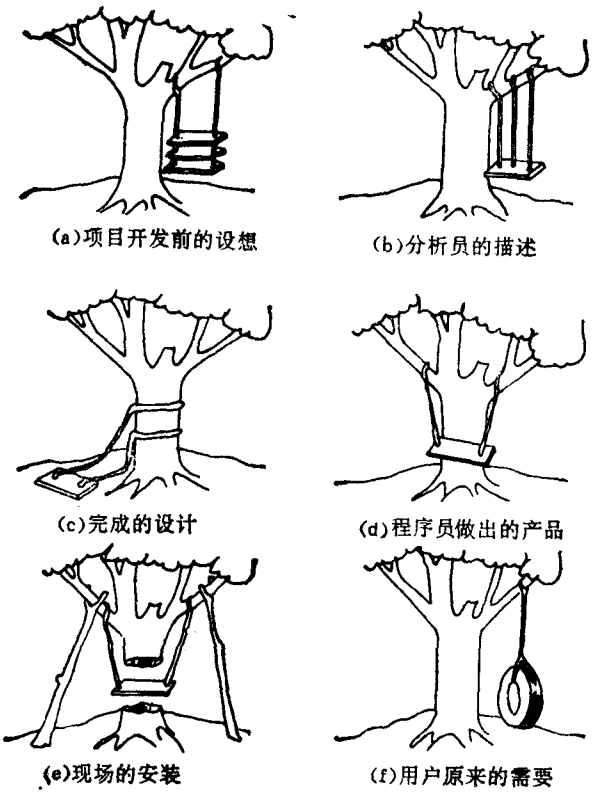

图 1.6　软件开发面临的实际问题

最后，为了较为形象地描述软件开发面临的实际问题，请读者参看图 1.6[注]。虽然这只是一个比喻，但的确在一定程度上反映了实际情况。软件项目的实践一再告诉我们，为了确保软件产品能够符合用户的需要，必须着眼于整个软件生存期，在每个开发阶段采取措施，进行各个阶段的 V,V&T 活动。使之不致在开发完成后，发现和用户的需求有如此大的差距。

在这一节的最后，我们讲一下关于排错（debug）一词的来源。程序中排错，我们现在都称为 debug，其原始意义是捉住小虫（bug）。这里有关于该词的一件轶事，它对理解排错与测试的区别是有益的。

1945 年夏在美国弗吉尼亚某地海军水上武器研究中心运行着 MarkII 计算机，这是以继电器为元件的老式计算工具。由于没有空调设备，夏夜中的机房很热。当时正值大战期间，计算任务十分繁忙。可是 MarkII 突然停止了工作，在多方查找后发现了原因：一只飞蛾从窗外进入，落在继电器的触点上。电磁式继电器触头将其打扁，致使电路中断而停机。机务人员捉到飞蛾，放于机器运行日志，并记载了这一情况。G. M. Hopper 为此创一新词，把排除机器运行的故障统称为“捉虫”——debugging。此后，人们也用该术语称呼程序排错。其实，它和测试一词的含义完全不同。

1.3　什么是软件测试

经过了多年的软件开发实践，积累了许多成功的开发经验，同时也总结出不少失败的

注：图1.6引自伦敦大学《计算中心通讯》，第 53 期(The University Computer Centre Newsletter, No. 53)。

教训。在此过程中，软件测试的重要意义逐渐被人们普遍认识。看到软件测试在开发成本中占有如此大的比例，同时它又是保证软件质量的主要手段，因而逐渐受到重视。然而，究竟什么是软件测试，这一基本概念很长时间以来存在着不同的观点。一些人为软件测试给出了定义，但由于强调的方面不完全一致，至今难于给出统一论述。即使那些公认的测试定义，也还存在问题。特别是目前仍然有许多人对于"什么是软件测试?"持有不正确的观点，这也恰恰是不能很好地做好软件测试工作的原因。

回答这一问题的典型说法有：

- 对照规格说明检查程序；
- 找出程序中的隐藏错误；
- 确定用户接受的可能性；
- 确认系统已经能够提供使用了；
- 取得该软件已能工作的信心；
- 表明程序执行得正确无误；
- 表明程序中错误并未出现；
- 理解程序运行的限制；
- 弄清该软件不能做什么；
- 检验该软件的能力；
- 验证软件文档；
- 使自己确信开发任务已经完成；

等等。

必须承认，以上这些说法并非都错，有的也有其正确的方面，但毕竟含有缺陷。

1973年 W. Hetzel 曾经指出，测试是对程序或系统能否完成特定任务建立信心的过程。这种认识在一段时间内曾经起过作用，但后来有人提出异议。认为我们不应该为了对一个程序建立信心或显示信心而去作测试。此后他又修正了自己的观点，他说："测试是目的在于鉴定程序或系统的属性或能力的各种活动，它是软件质量的一种度量"。这一定义实际上是使测试依赖于软件质量的概念。但软件质量又是什么呢？对于很多从事实际工作的人员来说，质量一词和测试一样抽象，也难以捉摸。事实上，尽管我们可以为软件质量的含意给出确切的解释，但影响软件质量的因素也不是一成不变的。在某一环境下得到的高质量产品，在换了另一环境以后，很可能变成低质量的，甚至是完全不适用的。如果我们把质量理解成"满足需求"，那将是可以接受的。假定软件项目的需求是完全的，开发出的产品又能满足这些需求，那就是高质量的软件产品。1983年 IEEE 提出的软件工程标准术语中给软件测试下的定义是："使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别"。这就非常明确地提出了软件测试以检验是否满足需求为目标。

G. J. Myers 则持另外的观点，他认为："程序测试是为了发现错误而执行程序的过程"。这一测试定义明确指出"寻找错误"是测试的目的。相对于"程序测试是证明程序中不存在错误的过程"，Myers 的定义是对的。因为把证明程序无错当作测试的目的不仅是不正确的，是完全做不到的，而且对做好测试工作没有任何益处，甚至是十分有害

的(关于这一点我们在本章下一节还要进一步说明)。从这方面讲,我们接受 Myers 的定义以及它所蕴含的方法论和观点。不过,这个定义规定的范围似乎过于狭窄,使得它受到很大限制。因为如前所述,除去执行程序以外,还有许多方法去评价和检验一个软件系统。按照 Myers 的定义,测试似乎只有在编码完成以后才能开始。另一方面,"测试"归根结底包含"检测"、"评价"和"测验"的意思,这和"找错"显然是不同的。

以上讨论的软件测试定义都是强调软件的正确。有些测试专家认为软件测试的范围应当包括得更广泛些。J. B. Goodenough 认为测试除了要考虑正确性以外,还应关心程序的效率、健壮性(robustness)等因素,并且应该为程序调试提供更多的信息。他列出了一张测试组成表(图 1.7)。

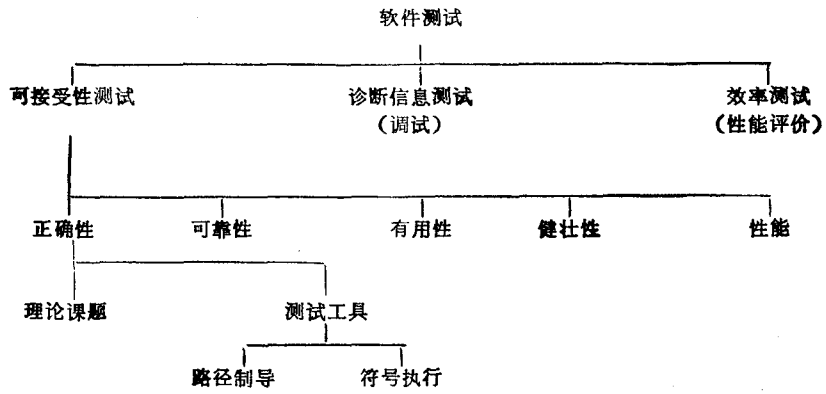

图 1.7 Goodenough 关于软件测试的定义

S. T. Redwine 认为,软件测试应该包含以下几种测试覆盖:

① 功能覆盖;

② 输入域覆盖;

③ 输出域覆盖;

④ 函数交互覆盖;

⑤ 代码执行覆盖;

他还给出了检查表。

至于测试的范围,A. E. Westley 将测试分为四个研究方向,即:

① 验证技术(目前验证技术仅适用于特殊用途的小程序);

② 静态测试(应逐步从代码的静态测试往高层开发产品的静态测试发展);

③ 测试数据选择;

④ 测试技术的自动化。

为了进一步明确测试的范围和具体目标,G. J. Myers 和 B. Beizer 都详细列出了各种软件错误的类型,并指出软件测试的目的就是要找出这些错误。

1.4 应该怎样认识软件测试

如何正确地认识和对待软件测试常常是做好软件测试工作的前提。事实上,到现在

为止仍然有一些不正确的看法和错误的态度，在不同程度地妨碍着测试工作的开展。这包括：

·认为测试工作不如设计和编码那样具有开拓性，也不容易看到进展。在测试中花了多大力气也很难让人看到。这是没有真正认识到软件测试的意义，如果以这种认识指导工作，那是非常有害的。

·以发现软件错误为目标的测试是非建设性的，甚至是破坏性的。简单地以为软件错误都属于责任事故，因此测试中发现了错误是对责任者工作的一种否定。其实，建设性与破坏性之间存在着辩证关系，并且追究错误的个人责任通常无益于问题的根本解决。

·"测试工作枯燥无味，不能引起人的兴趣"。持这种观点的人常常缺乏耐心，也没有认清软件测试的重要意义。

·测试工作的确是艰苦而细致的工作。但有人不愿在这上面花力气，指望着碰运气过关，这当然做不好测试工作。

·"这程序不会有问题，因为是我写的"。实际上，这是没有根据的盲目自信。在发现了程序错误以后，他们便立刻会顾虑别人对自己的看法。这常常是开发小组人员之间配合不好而影响工作开展的原因。要求人们的思维和他们的行为完全附合客观规律，而不犯任何错误是不可能的。一段时间以来，人们常说：要允许别人犯错误，也要允许别人改正错误。这才是我们对待错误的正确态度。

为了澄清认识和端正态度，有必要对以下几个方面的问题加以说明。

一、能够彻底测试程序吗？

如果我们认为测试的目的在于查找错误，并且找出的错误越多越好，很自然就会提出这样的问题：能不能把所有隐藏的错误全部找出来呢？或者问：能不能把所有可能做的测试无遗漏地一一做完，来找出所有的错误呢？

以下举两个例子进行具体的分析，从中可看出"穷举测试"的不现实性。

若一程序 P 有输入量 X 和 Y，并有输出量 Z，在字长为 32 位的计算机上运行。如果 X 和 Y 均只取整数，考虑把所有可能的 X 值和 Y 值作为测试数据，逐个用以驱动程序 P，进行穷举测试。力图全面、无遗漏地"挖掘"出程序中的所有错误。

这样做可能采用的测试数据组 X_i 和 Y_i，其中 i 的最大值为：

$$2^{32}\times 2^{32}=2^{64}\approx 10^{20}$$

如果程序 P 测试一组 X、Y 数据需用 0.001 秒，要完成 10^{20} 组测试，共需 5 亿年的时间！

若另一程序 Q 有四个条件判断（图 1.8 中的 d_1、d_2、d_3 和 d_4）和一个最多重复 20 次的循环。在循环体内有五条路径。若考虑到循环，从入口到出口总计有路径数为：

$$5^{20}\approx 10^{14}$$

假定我们为每条路径设计一组测试数据，再对其实施测试。如果这一过程需时两分钟，则测试完 10^{14} 条路径也要用 5 亿年的时间。

以上两种情况说明，由于穷举测试工作量过大，需用时间过长，致使实施成为不现实，因而也就失去了实用价值。

我们知道，软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地