软件测试

宫云战 编著

国防工业出版社
National Defense Industry Press

软件测试

宫云战　编著

国防工业出版社

·北京·

内 容 简 介

本书全面论述了软件测试的基本原理和一般方法。全书共分 10 章，分别为：软件测试的基本概念，软件缺陷数目的预测方法，黑盒测试方法，白盒覆盖测试技术，其他白盒测试方法，基于故障的软件测试方法，与软件开发阶段相关的各种测试（包括集成测试、系统测试、需求测试、设计测试与其他专向测试）方法，软件可靠性测试方法，面向对象的软件测试方法和软件测试性分析方法。

本书全面论述了软件测试技术近些年来发展的最新成果，有相当一部分内容是首次在著作中出现。本书可作为研究生教材，也可以作为软件测试工作人员的参考书。

图书在版编目(CIP)数据

软件测试／宫云战编著. —北京：国防工业出版社，2006.1

ISBN 7-118-04176-9

Ⅰ.软… Ⅱ.宫… Ⅲ.软件—测试 Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字(2005)第 110601 号

国防工业出版社出版发行

（北京市海淀区紫竹院南路 23 号）

（邮政编码 100044）

新艺印刷厂印刷

新华书店经售

*

开本 787×1092 1/16 印张 16¾ 385 千字

2006 年 1 月第 1 版 2006 年 1 月北京第 1 次印刷

印数：1—4000 册 定价：32.00 元

国防书店：(010)68428422 发行邮购：(010)68414474

发行传真：(010)68411535 发行业务：(010)68472764

前　言

信息技术极大地促进了社会的进步和人类文明的发展，已成为当今社会发展的主要动力之一。软件是信息技术的重要组成部分。近年来，世界软件产业取得了突飞猛进的发展，软件产业在一些主要发达国家已被提到了国民经济的主导产业。目前，全球软件从业人员已接近1000万人，每年生产数百万个中等规模以上的软件，全球软件及信息服务业市场已达到了近1万亿美元。其中，软件销售额大约为5000亿美元，并以每年约15%的速度在增加。

软件已经应用到国民经济的各个领域，特别是在高科技领域和某些国家安全领域，软件技术显得更加重要。近些年来，软件的规模在大幅度提高，复杂性也在增加，软件不可靠性的矛盾变得越来越突出，由于软件的错误所造成的损失也在大幅度增加，软件不可靠性的矛盾已经到了必须解决的地步。

软件测试是伴随软件工程技术发展起来的。自20世纪70年代诞生至20世纪90年代初期，发展一直比较缓慢，主要原因是社会需求不足、认识不到位、软件测试难度大，表现为软件测试的从业人员不多，测试工具比较简陋。近10年来，随着人们对软件认识的深入，软件危害的事例也越来越多，软件测试技术受到了广泛的重视。软件测试理论得到了快速发展，诞生了一些新的测试方法，出现了一批专门从事软件测试的公司，软件测试的从业人员也在大幅度增加，仅在中国就有大约1200多家软件评测中心，从事软件测试的人员有数万人，但目前尚缺10万～20万人。高效率的软件测试工具也不断涌现，软件测试技术正在形成一种产业，在我国2004年软件测试产业的产值在数十亿元。软件测试技术由于其重要的作用，已经发展成为了一门独立的学科。

传统的软件测试只包括了黑盒测试技术和白盒测试技术，这也是现在软件测试的基础，但由于测试过程复杂且测试效率不高，使得软件测试技术在20世纪徘徊了20余年而进展不大。近10年来，随着人们对软件及软件错误认识的不断深入，新的软件测试方法应运而生，主要包括以下几种。

(1) 软件的故障模型和面向故障的软件测试方法。成熟的故障模型是测试的基础，随着人们对软件错误类型的进一步归纳，软件的故障模型的研究取得了突破性的进展，基于故障的测试方法也应运而生，本书的第6章详细论述了这个方面的最新研究成果。

(2) 面向对象的软件测试技术。面向对象技术是现代软件开发的主流，由于该技术不同于面向过程的软件开发，而且面向对象技术的特性，使得面向对象的测试也有别于传统的软件测试方法，本书的第 9 章论述了面向对象的软件测试技术。

(3) 软件可靠性测试技术。在安全第一的软件中，软件的可靠性测试与评估是必要的。本书的第 8 章详细论述了软件的可靠性测试方法，包括如何获取软件的运行剖面、软件可靠性分配和软件可靠性建模。

(4) 软件测试性分析技术。测试是被动的，从系统科学的角度来看，设计与测试有一个折中，本书的第 10 章论述了软件的测试性分析方法。

(5) 软件测试的自动化技术。这是目前和未来软件测试技术研究的一个重点问题。它对于提高软件测试的客观性和效率有非常重要的意义。

作者近 10 年来一直从事软件测试技术的研究，并参与了多个国家和地方的软件测试的课题研究和工程软件测试的项目，积累了大量的数据，和国内外的同行建立了广泛的学术联系，并在国内主持召开了多次软件测试技术的研讨会。由于在本书的编写过程中，博士研究生万琳、张威、金大海、高传平、黄光燕、张广梅、肖庆等给予了一定的帮助，在此表示谢意。

宫云战

2005 年 7 月写于北京

目　录

第1章 绪 论

1.1 软件生存期

同其他任何事物一样，计算机软件从它的发生、发展到成熟阶段，直至老化和衰亡，是一个历史发展的过程，这个过程称为软件的生存期(Life Cycle)，包括下列6个步骤。

(1) 计划(Planning) 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(软件、硬件、人力)、成本、可取得的效益和开发的进度等做出估计；制定完成开发任务的实施计划。

(2) 需求分析(Requirement Analysis) 由软件人员和用户共同对开发的软件进行详细的定义和确切的描述，其结果是给出软件需求说明书(Software Requirement Specification，SRS)。

(3) 设计(Designing) 软件的设计分为两部分。一是概要设计(Preliminary Design)，是指根据软件的需求说明书，软件设计人员应把需求说明书中各项需求转化为相应的体系结构，在结构中的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。二是详细设计(Detail Design)，是指对概要设计中给出的各个模块所要完成的工作进行具体的描述，为后来的编程打下基础。软件设计的结果是给出设计说明书。

(4) 编码(Coding) 利用某种计算机语言，把设计说明书中规定的内容转化为计算机可以接受程序的过程称为编码。编码应以设计相一致，且结构清晰、易读、易修改。

(5) 测试(Testing) 根据软件的需求说明书、设计说明书和源代码，检验软件开发工作的成果是否符合要求的过程称为软件测试。软件测试是发现软件错误、提高软件可靠性与保证软件质量的重要手段。

(6) 运行与维护(Running and Maintaining) 对已交付用户的软件投入正式使用后便进入运行阶段，这个阶段可能持续若干年。在运行过程中，可能有多种原因需要对它进行修改，包括：运行中发现了软件错误需要修正；为适应变化了的软硬件环境，而需要做相应的变更；为进一步增强软件的功能，或提高其性能，而使它得到进一步的完善和扩充等。

上述6个步骤表明了一个软件从其酝酿开始，直至使用相当长的时间后，被新的软件代替而退役的整个过程。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，如图1.1所示。图中自上而下的箭头代表了问题的一个求解过程，而自下而上的箭头代表了在实际项目的研制中，为确保软件的质量，每一步骤完成后都要进行复查，如发现了问题，就要及时解决，以免问题积压到最后造成更大的困难。运行与维护的箭头表示在运行中可能需要多次维护。另外，图中还指明了6个步骤划分的3个阶段：软件定义阶段、软件开发阶段和软件维护阶段。

值得注意的是，上述软件维护工作不可简单地看成是仅仅修改程序。在运行过程中

若有必要修改，得提出充分的修改理由，经过审核，才能确定下来。接着需要经历制定修改计划、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，则将经历多次这些步骤。可用图 1.2 来表示这一过程，并称为软件的生存周期，也简称为软件的生存期。

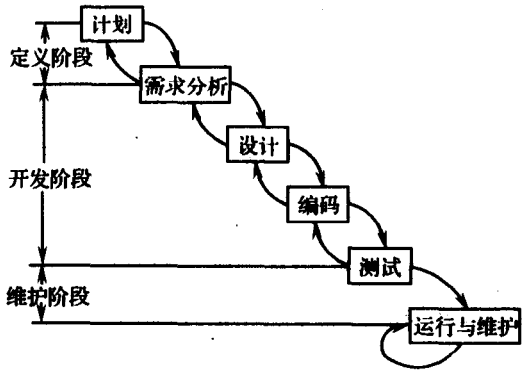

图 1.1　软件生存期的瀑布模型

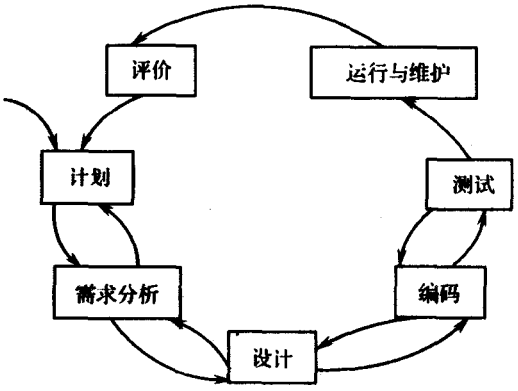

图 1.2　软件的生存周期

1.2 软件危机

计算机系统工程分为硬件和软件两大范畴。计算机硬件的工程技术在过去的 50 多年，已经达到了相当成熟的状态。硬件设计技术和制造技术发展非常迅速，其自动化已经达到相当高的水平，硬件的可靠性已是一种现实的要求，而不再是一种愿望。

软件工程技术方面的情况则不同，软件是最难设计、最少可能成功、最容易出错，也最难管理的系统部分。据报道，在 20 世纪的最后 10 年里，计算机软件已成为系统瘫痪的主要原因。随着以计算机为基础的系统在数量、复杂程度和应用方面的激增，对软件的需要却在不断增加，因此促使供求矛盾日趋激化，这就是人们常说的软件危机。软件危机的来源主要表现在以下几个方面。

(1) 缺乏软件开发的经验　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，使得经费常常突破预算，工期一拖再拖，软件的投资者和用户对开发工作从不满意发展到不信任。

(2) 需求不明确　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，或者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地交换意见，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，导致对多个错综复杂的问题既难于分析，又难于解决。

(3) 缺少开发规范　开发过程中没有统一遵循的、公认的方法论和开发规范，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，使得开发文档很不完整。发现了问题，未能从根本上找原因，只是修修补补。显然，这样开发出来的软件无法维护。

(4) 软件的复杂性　软件的规模一般都比较庞大，大型软件有时会超过 1 亿行源代码。加之人们传统上的误区，往往是硬件难以实现的部分改用软件来完成，这使得软件既庞大，复杂性又高，甚至有时人的大脑已无法理解、无法驾驭人类本身所创造出来的复杂逻辑系统，投入使用后往往错误百出。

(5) 缺乏有效的测试手段　软件的复杂性和软件测试的复杂性，使得难以研制有效的软件测试工具，导致测试效率不高、自动化程度低、测试花费时间多、测试成本高，这使得软件开发者只要求测试人员对软件做简单的测试，而无法保证软件的质量。

软件危机的事例是很多的。最著名的是 20 世纪 60 年代，美国 IBM 公司开发的 IBM 360 操作系统，这一项目在开发期共花费了 5000 万美元，总共投入的工作量是 5000 人年，共写出了 100 万行源程序。由于它太庞大，OS 360 变得相当不可靠，平均每次修改后的新版本大约都存在 1000 个错误，而且有理由认为这是一个常数。另外，美国空军的范登堡中心在 20 世纪 60 年代后期发生过多次导弹试射失败的事故，事后检查几乎都是由于软件有错误而造成的。

与软件危机有关的许多问题都起源于软件本身的特点，软件开发人员的弱点，以及人们对软件开发实质的种种不切实际的误解，计算机软件已经成为以计算机为基础的系统发展的重要瓶径。科学上的危机和其他领域的危机一样，解决危机的过程往往孕育着一种科学理论的诞生。自 20 世纪 70 年代以来，科学家们一直在试图解决软件危机问题，虽然目前尚不能说软件的危机已经过去，但 20 年来，软件技术的迅速发展，包括面向对象的技术、基于知识的软件开发环境、先进的软件测试工具等，为保证大型软件的研制提供了重要的基础。正是这些先进的技术，目前上亿行源代码的软件比比皆是。

1.3 软件质量

质量这一概念有许多不同的定义。在《辞海》中，就把质量一词解释为“产品或工作的优劣程度”。国际标准化组织(ISO)把质量定义为“与一个产品或服务是否能够满足其指定的或蕴涵的需求有关的性质与特征的总合。”同其他产品一样，软件的质量也不是绝对的，在不同的情况下，对不同的人来说，软件质量的含义是不同的。

1. 软件质量要素

软件产品的质量是由许多软件性质构成的，这些性质常称为软件质量要素。一般来讲，软件的质量要素有下列 11 个。

(1) 易使用性　是指软件易于使用的程度。

(2) 完整性　保护软件不被未经同意的存储和使用的能力。

(3) 效率　指软件对计算机资源的使用效率，包括运算时间效率和存储空间效率。

(4) 可靠性　不失败的能力。

(5) 正确性　程序完成其规约的程度。

(6) 易维护性　在程序的操作环境中，确定软件故障的位置并纠正故障的难易程度。

(7) 灵活性　当软件操作环境变化时，对软件做相应修改的难易程度。

(8) 易测试性　对软件测试以保证其无错误和满足其规约的难易程度。

(9) 易移植性　将一个程序从一个运行环境移植到另一个运行环境的难易程度。

(10) 易复用性　复用一个软件或其部分的难易程度。

(11) 互用性　将一个软件系统和其他软件系统组合在一起的难易程度。

2.软件质量要素的衡量标准

软件的每个质量要素又包含一系列的衡量标准,具体为:

(1) 易使用性　包括易操作性、培训、易交流性、输入和输出量、输入输出速度。

(2) 完整性　包括存储控制、存储审查。

(3) 效率　包括运行效率、存储效率。

(4) 可靠性　容错性、一致性、准确性、简洁性。

(5) 正确性　包括易追溯性、一致性、完备性。

(6) 易维护性　一致性、简洁性、简明性、模块性、自我描述性。

(7) 灵活性　模块性、一般性、易扩展性、自我描述性。

(8) 易测试性　简洁性、模块性、检视、自我描述性。

(9) 易移植性　模块性、自我描述性、硬件独立性、软件独立性。

(10) 易复用性　通用性、模块性、自我描述性、硬件独立性、软件独立性。

(11) 互用性　模块性、通信共同性、数据共同性。

每个衡量标准的定义为:

(1) 易追溯性　指在特定的软件开发与操作环境中,能够从软件的需求寻找出其相应的实现的能力与性质。

(2) 完备性　指软件实现了其全部所需功能的性质。

(3) 一致性　指在软件的设计与实现中采用统一的技术与术语的性质。

(4) 准确性　指软件的输出与计算中的精度满足其需求的性质。

(5) 容错性　指在非正常条件下,仍然能够操作软件的性质。

(6) 简洁性　指软件以最容易理解的方式实现其功能的性质。

(7) 模块性　指用一系列在很大程度上相互独立的模块来构成软件的性质。

(8) 一般性　指软件所提供的功能具有应用范围广的性质。

(9) 易扩展性　指易于对软件存储空间和计算功能进行扩充的性质。

(10) 检视　指软件所提供的用于测量使用情况和识别错误的属性。

(11) 自我描述性　指软件中包含它对功能的实现的解析性信息的属性。

(12) 运行效率　指软件使用最少的处理时间的性质。

(13) 存储效率　指软件在操作中对存储空间的需求最少的性质。

(14) 存储控制　指反映对软件及其数据的存储进行控制的能力的性质。

(15) 存储审查　指对软件及其数据的存储进行审查、记录能力的性质。

(16) 易操作性　指决定软件的操作与操作过程的复杂程度与难易程度的性质。

(17) 培训　指支持从初步熟悉到熟练操作软件的过度的性质。

(18) 易交流性　指软件的输入与输出能够被人们理解的程度的性质。

(19) 软件独立性　指决定对软件环境中的其他软件的依赖程度的性质。

(20) 硬件独立性　指决定软件对硬件环境的依赖程度的性质。

(21) 通讯共同性　指软件使用标准的通信协议与界面的性质。

(22) 数据共同性　指软件使用标准的数据表示格式的性质。

(23) 简明性　软件的实现使用最少代码的性质。

每一个软件质量衡量标准又可以有多个不同的度量。软件质量度量有的作用于软件的代码，有的作用于软件开发过程中产生的中间结果和文档。

1.4　软件可靠性

1. 硬件可靠性

所谓系统的可靠性，是指“一个系统在一定的环境下，在所给定的时间内，能按预定的要求完成一定功能的概率。”

设一个具有 N 个元件的系统，运行 t 时间后，有 $F(t)$ 个元件失效，其余 $S(t)$ 个元件仍然保持完好，则分别定义元件的可靠度 $R(t)$ 和不可靠度 $Q(t)$ 为

$$R(t) = \frac{S(t)}{N}, Q(t) = \frac{F(t)}{N}$$

定义元件的失效率 $Z(t)$ 为单位时间内失效的元件数与非失效的元件数之比，即

$$Z(t) = \frac{\mathrm{d}F(t)}{\mathrm{d}t} \times \frac{1}{S(t)}$$

根据理论研究和大量的统计得出，在正常的工作条件下，元件的失效变化趋势如图 1.3 所示。这就是硬件系统可靠性分析中著名的盆式曲线(Bathtub Curve)。它表明系统在使用的开始阶段，其失效率是比较高的，但随着故障的排除，其失效率将稳定在一个基本上保持不变的常数，这个时期称为正常的使用期。由于受到周围环境、元件老化等因数的影响，在使用一定的时间后，系统的故障率开始上升，而且越来越高，直至系统被淘汰。

在正常的使用期，系统的失效率可以近似为一个常数 λ，则

$$R(t) = \frac{S(t)}{N} = \frac{N - F(t)}{N} = 1 - \frac{F(t)}{N}$$

$$\frac{\mathrm{d}R(t)}{\mathrm{d}t} = -\frac{1}{N} \times \frac{\mathrm{d}F(t)}{\mathrm{d}t}$$

$$\lambda = Z(t) = \frac{\mathrm{d}F(t)}{\mathrm{d}t} \times \frac{1}{S(t)} = -\frac{N}{S(T)} \times \frac{\mathrm{d}R(t)}{\mathrm{d}t} = -\frac{1}{R(t)} \times \frac{\mathrm{d}R(t)}{\mathrm{d}t}$$

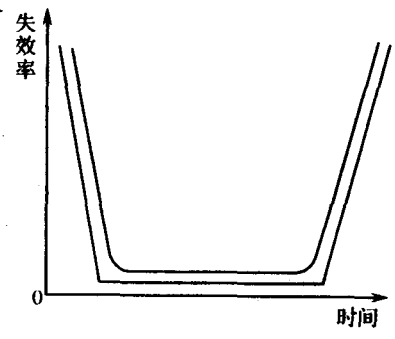

图 1.3　硬件可靠性建模的基本思想

解这个方程，并考虑到 $R(0) = 1$，有

$$R(t) = \mathrm{e}^{-\lambda t}$$

硬件可靠性理论中的一个重要参数是系统的平均故障间隔时间(Mean Time Between Failures, MTBF)和平均故障时间(Mean Time To Failures, MTTF)。MTBF 是指可修复故障的系统，而 MTTF 是指不可修复故障的系统。如果忽略这一点，MTBF 和 MTTF 是可以混用的。因此，根据定义有

$$\mathrm{MTBF} = \int_0^1 t\mathrm{d}R(t) = \frac{1}{\lambda}$$

上述公式是硬件可靠性的基本理论，也是一套非常成熟的理论。

经过几十年的发展，硬件可靠性不但在理论上成熟了，而且在工程上也有许多提高可靠

性的方法和工具。目前,一般出厂的硬件产品都有可靠性指标,如风险函数和 MTBF 等。

2. 软件可靠性

软件的可靠性定义为:在给定的时间内,特定环境下软件正确运行的概率。正如 1.3 节所述,软件可靠性是软件质量的要素之一。对用户而言,同其他软件质量要素相比,软件可靠性是最重要的要素。

从其定义可以看出,软件可靠性涉及 3 个概念:给定的时间、特定的环境、正确运行的概率。

对给定的时间,是指由于软件的可靠性是随时间而变化的,根据软件可靠性的理论,软件每排除一个故障,其可靠性就提高一步。因此,在谈论软件可靠性的时候,一定要指明是什么时间的可靠性。

特定的环境是软件可靠性的一个重要的因素,它包括硬件环境、软件支撑环境和其他为保证软件正常运行所需要的环境。在评估软件可靠性时,一般要假设环境是正确无误的。

正确运行的概率情况比较复杂,软件运行包括选择输入数据和启动程序执行。但问题是如何选择输入数据才能真实地反应软件的可靠性,如何根据错误发生的时间来评估软件可靠性,软件可靠性的变化规律是什么,等,类似的问题在软件可靠性理论的研究中都没有很好的解决。虽然如此,在过去的 30 余年里,科学家在这个方面做了大量的研究,目前比较著名的有 40 余种软件可靠性模型,由于工程数据的来源以及对软件可靠性认识上的不同,这些模型存在较大的差异。一个新的软件究竟用哪种可靠性模型去评估更好,目前尚无定论。只能根据工程背景和个人的喜好去判断。

抛开寻求统一的软件可靠性模型,近几年来,许多科学家试图从工程的角度研究软件可靠性的部分内容,主要是软件的平均无故障时间 MTBF、软件的故障率 λ 等,这些参数对用户来说恐怕是更重要的。目前的许多军用软件、关系到国计民生的一些重要软件都有相应的可靠性参数评估,也是软件生产应达到的目标。

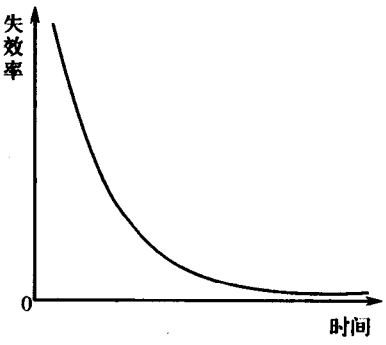

图 1.4 软件可靠性建模的基本思想

根据软件可靠性的理论,一个软件在交付之后,软件的可靠性就是确定的,只不过人们现在还没有很好的办法去认识它。抛开其他因素不谈,惟一能影响软件可靠性的是残留在软件中的错误数目。随着软件在测试过程中或在使用过程中,软件错误的不断发现与排除,残留在软件中的错误会逐步减少,软件的故障率会逐步降低,因此,软件的故障率曲线应该类似于图 1.4 所示的曲线。这是软件可靠性建模的基本思想。

1.5 软件错误

1. 软件错误的概念

定义 1.1 错误(Error):是指人们期望的和系统实际具有的状态或行为之间的偏差。

软件错误包括由系统分析或者程序设计而产生的全部错误。软件错误存在于软件生存期的各个阶段，凡是和预期的状态或行为不相符的统称为软件错误。例如，制定的计划和实际不相符合的、需求说明书有问题的、设计说明书有问题的、程序编码有问题的、测试有问题的、运行结果不正确的、维护有问题的等，都是软件错误。

定义 1.2 失效(Failure)：是指在软件运行期间出现的错误。它是因软件中存在的错误引起的，并在执行期间动态表现出来的和实际不同的状态和行为。

从这里可以看出，失败只是错误的一部分。

定义 1.3 故障(Fault)：是指软件中的物理缺陷。

故障也是错误的一部分，故障是指软件中实实在在存在的错误。故障可能引起软件的失效，也可能不会引起软件的失效，人们总是通过修改软件中的故障来提高软件的可靠性。

定义 1.4 缺陷(Defect，Bug)：凡是没有按照要求所进行的工作统称为缺陷。

缺陷泛指系统中的一切错误。显然，缺陷是比错误范围更大的概念，缺陷除了包括错误所具有的全部属性外，像程序的多余物、难以理解的程序、难以修改的程序、文档不完全等，虽然这些可能并不能使软件失败，但是会使软件有缺陷。通过修改软件中的缺陷来提高软件的质量。

从定义可以看出，上述 4 个概念是彼此相关的，有些文献中也经常是混用的。实际使用时应区分它们之间的层次和范围。

2. 软件的缺陷数目与缺陷密度

软件的缺陷数目是指软件中所存在的所有缺陷的总和。而软件的缺陷密度是指每单位代码中的缺陷数目，这里，单位代码一般是 KLOC 或 KSLOC(千行源代码)，表示为缺陷数目/KLOC，或者缺陷数目/KSLOC。

软件工程师在工作中一般会引入大量的缺陷。统计表明，有经验的软件工程师的缺陷引入率一般是 50 个 ~ 250 个缺陷/KLOC，平均的缺陷引入率在 100 个缺陷/KLOC 以上。即使软件工程师学过软件缺陷管理之后，平均的缺陷引入率也在 50 个缺陷/KLOC，也就是说，每 20 行代码中就有一个缺陷。从理论上看，人的认识能力是有限的，根据 Hatton 模型，对一般的软件，软件的缺陷数目可用下面的公式表示，即

$$N = \log_{10}\Omega(\Omega \leqslant \Omega')$$

$$N = \frac{1}{2}\left(\frac{\Omega}{\Omega'} + \frac{\Omega}{2\Omega'}\right)(\Omega > \Omega')(\Omega' \text{ 是复杂性系数，一般取 200LOC} \sim \text{400LOC})$$

式中：Ω 是软件的复杂度，一般用源代码数目来表示。

从工程实践来看，目前世界上先进的软件开发组织所生产的软件，其缺陷密度可以达到 2 个缺陷/KLOC，一般的软件开发组织所生产的软件其缺陷密度可以达到 4 个 ~ 40 个缺陷/KLOC。据统计，在美国，安全第一的软件开发的代价是每行程序需花费 500 美元。NASA所生产的软件其故障密度可以达到 0.1 个缺陷/KLOC，但其代价是每行源代码高达 1000 美元。

软件的缺陷数目或者缺陷密度可以通过测试或者其他方法来计算。而残留在软件中缺陷数目或者缺陷密度只能通过评估来得到，例如，根据测试的程度，或者根据软件可靠性模型等可以预测软件中残留的缺陷数目。

在安全第一或者一些重要的软件验收中，软件的缺陷数目或者缺陷密度是软件能否通过验收的重要标准。

3. 软件错误的分类方法

软件的错误是非常复杂的，凡是人们能够想到的软件错误都是可能发生的。到目前为止，人们对软件错误的认识尚停留在表面上，其内在规律有待进一步研究。这也是目前软件测试技术发展比较缓慢的原因之一。虽然如此，人们还是可以根据软件错误的表现将其进行分类，常见的分类方法有以下几种。

(1) 根据错误的征兆进行分类　这种分类方法是根据系统的输入和系统对输入的反应结果进行的分类，如表 1.1 所列。这种分类方法的好处是可以提高人们认识错误的严重程度，以便防患于未然。

表 1.1　根据错误的征兆进行的分类

输入 / 系统反应	错误输入	正确输入
对输入的拒绝	正确	第Ⅰ类型的错误(严重)
得出错误的结果	第Ⅱ类型的错误(严重)	第Ⅲ类型的错误(严重)
系统崩溃	第Ⅳ类型的错误(严重)	第Ⅴ类型的错误(严重)

(2) 根据错误的起因进行的分类　计算机系统中的每一个错误，究其原因，大都可以归结为下列 3 种情况之一：硬件或软件中的设计错误；由于环境或部件的老化引起的硬件恶化；错误的输入数据。显然，在软件生存期的不同阶段，每类错误的分布是不一样的，因此，这种分类方法的好处：一是了解错误在不同时期的分布规律，例如，在系统运行的早期所发生的错误，很有可能是设计错误或数据错误所引起的，在运行相当长的时间后所发生的错误可能是由于数据错误引起的，而在运行相当长的时间后，就有可能是硬件恶化引起的，这为判断软件错误的类型并尽快修复提供依据；二是根据这种分类，可以知道哪些错误是可以排除的，而哪些错误又必须通过是更换设备才能解决的。

(3) 根据错误发生的持续时间来分类　一般可将其分为永久性错误和瞬时错误。永久性错误是指经常发生的错误，这类错误是不可怕的，因为它容易被排除。而瞬时错误恰恰相反，由于是偶尔出现，难以掌握其规律，故很难将其排除。

(4) 根据错误的严重程度进行分类　不同的错误所产生的后果是不一样的。较少的错误，如数据格式不对，并不会产生什么后果；而有些错误，如飞机导航软件出错，可能导致机毁人亡的严重后果。一般来讲，根据错误的严重程度可以分为：较少错误、中等错误、较严重错误、严重错误、非常严重错误和最严重错误。这里的分类，是根据错误引起后果的程度来分类的，而不是根据错误对应故障的复杂性来分类的。有时一个较少的故障，例如，程序语句漏掉一个标点符号等，都可能产生严重的后果。

(5) 根据开发阶段进行分类　在软件生存期的每个阶段都可能存在错误，即计划错误、需求分析错误、设计错误、编码错误、测试错误、运行与维护错误。由于生存期的每一个阶段又是下一个阶段的先导，也就是说，前一个阶段正确的设计是保证下一个阶段设计正确的必要条件。因此，要求一旦发现某个阶段存在错误，就要尽快将其排除。否则，越