Computing McGraw-Hill

# HANDBOOK OF SOFTWARE RELIABILITY ENGINEERING

# 软件可靠性工程手册

〔美〕MICHAEL R. LYU主编

刘喜成　钟婉懿　等译

IEEE COMPUTER SOCIETY PRESS
50 YEARS OF SERVICE · 1946 - 1996

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.co.cn

# 软件可靠性工程手册

# HANDBOOK OF SOFTWARE RELIABILITY ENGINEERING

〔美〕Michael R. Lyu 主编

刘喜成　钟婉懿　等译



電子工業出版社

PUBLISHING HOUSE OF ELEGTRONICS INDUSTRY

## 内 容 提 要

本书内容主要包括三个部分十七章。分别叙述了软件可靠性工程基础、实践与经验和工程技术。本书以大量的文献资料和丰富的实践经验说明了软件工程的数据收集、系统操作、防错、排错、容错、失效预测、有关理论、建立模型、测试、度量标准、维护、可靠性分析和估计等多方面知识。书中采用的估计方法、实例、测试结果均来自实际工作系统。本书是软件工作者不可多得的软件可靠性测试的工具书。



书　　名: 软件可靠性工程手册
HANDBOOK OFSOFTWARE RELIABILITY ENGINEERING
著　　者: [美] Michael R. Lyu 主编
译　　者: 刘喜成　钟婉懿　等译
责任编辑: 和德林
责任校对: 李清明
排版制作: 电子工业出版社排版室
印 刷 者: 北京市顺义县天竺颖华印刷厂印刷
装 订 者: 三河市赵华装订厂装订
出版发行: 电子工业出版社出版、发行　URL: http://www.phei.co.cn
北京市海淀区万寿路173信箱　邮编100036　发行部电话(010) 68214070
经　　销: 各地新华书店经销
开　　本: 787 × 1092　1/16　印张: 35.5　字数: 906千字
版　　次: 1997年3月第1版　1997年3月第1次印刷
印　　数: 3000　册
书　　号: ISBN 7-5053-3884-6
TP·1671
定　　价: 68.00元
著作权合同登记号　图字: 01-96-0374

# 前　言

在复杂的软件系统中，可靠性是软件质量最重要的标志，但它经常又是最捉摸不定的。由于世界上的许多工程项目和系统都按软件进行开发，因此软件可靠性应达到相当的水平、既稳定又经济是至关重要的。软件失效可以是最大的新闻、因为它给人们带来很多麻烦，最坏的情况还可以危及人们的生命。

“软件可靠性工程手册”是一本让人喜欢的书，它从各方面对软件可靠性进行了深入地论证。本书在软件可靠性工程发展历史上是一个重要的里程碑。Michael R. Lyu 集中了一批领先专家用大量的文献资料大大丰富了这个领域的实践。本书内容广泛，包括故障防止、故障排除、容错、失效预测；对理论、模型、测试、度量标准、进程、处理、分析和估计技术均有详细叙述。本书提到的方法均已证实，实例均有说服力、测试结果来自现场的工作系统。更要重的是书中有多种可靠性工具、使用它们可提供技术服务。

本书内容处理严格、以慎重的工程科目为特点，十分重视综合研究和评价可靠性的数学方法，说明这些方法如何应用于开发软件系统。学习本手册主要理解实践中的软件可靠性工程，是怎样用图表公式表示、应用和评价的。可靠性十分明显地关系到软件产品的许多特点和开发过程。本书试图以定量定性的方法进行各方面地分析。

本书是为实际工作者或科学工作者而设计的，而不是初学者。读者对象是广大的需要软件可靠性工程信息的人员、小组或组织，其中包括：

1. 需要一般了解软件可靠性的人员，使用软件的高级管理者、专业工程师、软件接口设计人员，有技能、想购买、搞发行或使用软件的各种人员。

2. 软件开发人员、测试人员和质量管理人员。可向他们提供软件可靠性工程技术，包括系统工程原则、可靠性管理、风险分析、决策和软件维护等。

3. 软件工程、可靠性分析、应用统计、操作探讨领域的开发研究人员，及一切想深入了解软件可靠性及其他工程技术的人员。

本书每一个章节的主题可以看成是统一和可分的一部分。这些主题代表了有关软件可靠性工程的基本原则和实践。它们都可在评价和改进软件可靠性工程方面提供多种框架和技术服务，这是从实际和经验中获得的特别技术。

计算机软件的开发和应用正以飞快的速度发展着，越来越显示出它的巨大威力，因而人们对它的可靠性要求越来越高。我希望随着本书的出版，为软件界的读者奉献一本有用的译著。

本书的原作者有几十位专家学者。翻译此书的主要有刘喜成、钟婉懿、刘岩、袁建铬、王会、潘宏伟、王湘文、王涛、高玉洁、黄晓钊、施海虎、胡海潮、张亚平、张静、李军、王平庆、倪荣华等。钟婉懿、张亚平、张静、贾宝崎、董彬对全书进行了校正和补译工作。另外北京计算机学院的几位老师进行了部分审读工作，在此表示感谢。

本书英文版最早是样稿，译者多半以样稿译成中文。到 1996 年才收到正式出版的版本，内容有些改动，所以又进行了大量的校正和补译。对于这样一部技术含量很高的著作，不管是译者或是审校者都觉得有一定难度，加之原著也是多作者，这样就给本书译文的完整统一带来不少烦麻。错误和不妥之处一定存在。敬请读者批评指正。

编者　1996.10

# 第一部分　技术基础

## 第一章　导　　论

Michael R. Lyu
AT&T Bell Laboratories(贝尔中心)
译者:刘　岩

### 1.1　可靠软件的需求

自从第一台计算机从五十多年前诞生以来[Burk46a],人们的日常生活越来越离不开计算机。计算机革命无疑成了当今世界上最快的技术变革。今天,计算机软硬件遍布于人们的现代生活。离开计算机人们将无法控制与操作最新式的照相机、盒式录音机和汽车。计算机已应用于手表、电话、家用电器乃至飞机中。科学技术的进一步发展急需高性能的软硬件。事实上我们可以看到几乎所有的工业,如汽车、航空、石油、交通、电信、半导体、制药业等,即使不是全部,也在很大程度上依靠计算机来提高产品质量及竞争力。

随着对计算机的需求和依赖的与日俱增,计算机出现故障引起危机的可能性也逐渐增加。因软硬件出错所产生的影响,从造成诸多不方便(如家用电器失灵),带来经济损失(如银行系统中断)到危及生命(如飞行系统和医疗系统失灵)不断出现。勿庸置疑,计算机系统的可靠性已成为社会所广泛关注的问题。

在计算机革命中,存在着不均衡发展的现象。与硬件技术飞速发展形成鲜明对比,软件技术的发展却没有得到与之相适应的发展,例如质量、产量、耗费及性能等方面都无法与硬件同步。据报道,在本世纪最后十年里,计算机软件已成为系统瘫痪的主要原因[Gray90a]。事实证明,系统的硬件器件变得越来越可靠,软件已成为导致系统失败和停机的主要因素。随着对软件需求的增加,其规模、复杂性、重要性也相应增加。事实上,软件成分在应用中的增加正是许多系统综合复杂性提高的重要原因。正是由于软件的集成潜力,使得设计者可以设计出更多的、适用范围更广的、跨领域的优秀软件系统。

在过去的十年里,工程系统的规模和复杂性急剧提高,且该趋势还将保持下去。更高更复杂的软/硬件系统常见于宇航局、国防部、联邦航空局、电信业和很多其它私人行业所承担的工程中。例如,航天飞机的机载系统有近 500,000 行代码的软件,且地面控制和处理系统也有大约 350,000 行的代码。在对原始计划进行了大规模削减之后,国际航天局 ALPHA 仍然需要数百万行的软件代码来操纵导航、通迅和实验用的大量硬件设备。在美国电信业中,电信线路的正常运转需要数百个软件系统的支持,其代码总量超过一亿行。在航空电子产业领域,几乎所有新的载荷仪器都有它们自己的含有丰富内嵌式软件的微处理器系统。在联邦航空局(FAA)

的“先进自动化系统”(AAS),新一代航空飞行控制系统中,也包含着极大规模的硬件和复杂的软件。

对复杂的软硬件系统的需求急剧增加,其速度远远超过软/硬件的设计、实现、测试及维护能力。结果是,近期文献中常出现关于可怕的工程事故的报导。据统计,其中大多数是由于软件出错所致[Lee92a]。下面给出几个因主要程序出现软件错误而导致未完成目标任务的实例。在宇航局的旅行者计划中,天王星探测器就因为深度太空网络软件发送信息迟缓和能力衰减而处于危险中,由于发生硬/软件干扰问题,一些航天飞机推迟发射时间。在某国防部(DOD)工程中,软件问题导致 AFTI/F-16 的首航时间耽误了一年,事先设计好的先进程序无一可用。关键系统的失败也影响了大量民用项目和科学工作。如果不是数据分析程序由于“越界溢出”而隐藏了异常数据,南极上空的臭氧空洞本来可以早些引起科学界的注意。自动行李处理系统的小小软件错误使丹佛国际机场在飞行受阻和进入跑道后空等了九个月。

毫无疑问,软件还能杀死人。大规模的 Therac-25 放射治疗仪曾以其安全性著称于世,直到有一天由于软件出错而使其控制系统失灵,导致多名病人失去生命。1992 年 10 月 26 日,伦敦救护服务中心的计算机辅助发送系统刚启动就崩溃了,致使这个全世界最大,每天要接受五千个待运病人求救(其中,很多病人情况危急)的救护服务机构全部瘫痪。在航空业中,过去的几年里,尽管机械原因仍然是造成客机坠毁的可怕因素,但专家指出,软件控制系统对飞行员在异常飞行期发出的紧急查询反应不当,可能是某些事故的主要原因。

软件失效也可导致商务灾难。1990 年 1 月 15 日,通信中转系统新投入使用的软件发生了错误,导致主干线远程网大规模崩溃。1991 年夏季,一系列局域电话网由于软件问题而中断。这些严重事故使数以千计依靠电讯公司运营业务的公司损失了巨额资金。

至此,许多大公司已认识到应该投入大量的工程开发费用以确保设计和推出可靠的软件。使用软件可靠性工程技术的系统方法已有了迫切的需求。显然,开发用于软件可靠性工程的技术是对计算机工程师、软件工程师以及现在和即将来临的下个十年各学科的工程师们的一个极大的挑战。

## 1.2 基本定义

软件可靠性工程(SRE:Software Reliability Engineering)是以软件一系列重要特性为中心而展开的:

可靠性(reliability) 软件可靠性指的是在给定时间内,特定环境下软件无错运行的概率[ANSI91a]。软件可靠性是将在第二章详细讨论的系统可依赖性(system dependability)概念中的一个内容,也是软件质量属性之一。全面衡量软件质量还需要其他一些用户满意的因素,如功能(functionality)、可用性(usability)、性能(performance)、可服务性(serviceability)、能力(capability)、可安装性(Installability)、可维护性(maintainability)和文档[Grad87a, Grad92a]。

软件可靠性已被公认为是系统可依赖性的关键因素,因为它可以定量地衡量软件的失效性——可导致软件失去作用甚至有害于整个系统,这是人们最不期望看到的情况。软件可靠性也可被认为是从软件质量方面满足用户需求的最重要的因素。

因此,软件可靠性工程可定义为定量地按用户对于可靠性的需求,进行研究基于软件系统的操作行为。它包括:

(1) 软件可靠性度量,以文献中建立的软件可靠性模型为基础进行的评价和预测。

(2) 产品设计、开发过程、系统结构、软件操作环境等要点与度量标准及它们对可靠性的影响。

(3) 应用这些知识说明和指导系统软件设计、开发、测试、查询、使用和维护。

注意,软件可靠性定义的三个要素为:失效(failure)、时间(time)、操作环境(operational environment)。还应区别在软件可靠性度量(measurement)中评价(estimation)与预测(prediction)。下节给出这些术语的详细定义。

### 1.2.1 错误与失效

首先应大概了解软件系统及其所期望的服务:

(1) 软件系统是内嵌于计算环境的软件子系统的一个相互作用集,该环境为软件系统提供输入且接受软件的服务(输出)。软件子系统本身又是由其它子系统组成的,以此类推,达到一个期望的分解级分解成有意义的最小元。

(1) 所期望的软件系统服务(行为)是一个输出状态的时间相关序列(time-dependent Sequence),该输出序列与实现软件(按确定目的)所遵从的初始约定说明相一致,亦即与系统用户认同的正确值(为实现有效目的)相一致。

现在我们可以说,一个名为程序(P)的软件系统发送一个所期望的服务(S)给某个名为用户(U)环境或人。

失效　用户发现程序未产生所期望的服务即为失效。如下例所示:

(1) 某预定代理商(U)发现数据库(P)接收了一个查询,却未产生任何反应(S);

(2) 某 银行出纳(U)要求计算机(P)给出某一支票帐户的余额(S)。计算机回答"读错"以示读余额操作未成功;

(3) 某飞机驾驶员(U)发现,自动飞机控制系统(P)未能按指令保持某确定高度(S);

(4) 某电话用户(U)拿起电话拨号,但转接系统(P)却未按要求联接到正确线路(S)。

用户可以根据他们对系统服务的影响选择判定各种不同的失效严重程度级别,如:灾难性失效、重大失效、微小失效。这些严重级别的定义因系统而异,而在某一给定系统内应保持术语的一致性。

一种特殊的失效情况是运行中断。运行中断可定义为在一定时间内(中断期)对客户的服务停止或降级。通常,中断可以由硬件或软件失效,人为错误和环境变化(如雷电、电源失灵、火灾等)引起。由失效而导致整个系统功能的损失称为"系统中断"。例如,在电信业中对系统中断进行定量化,如下:

定义中断时间为"大于三秒(对于导致稳定呼号丢失的失效)或大于三十秒(对于不导致稳定呼号丢失的失效)"[Bell90a]。

错误在既不引起程序失灵,程序又未检测出内部错误(如状态不对)时,错误量不明显地暴露出来。引起失效或内部错称为错误又称"缺陷"。在大多数情况下,错误可被查出并排除;在某些情况下,它仍然存在。一个假设认为,无法充分证明错误的存在(如,分布式系统的分时错误)。错误例子如下:

(1) 启动错:程序不按要求启动。

(2) 输入范围错:程序不能正常检测输入数据范围。

(3) 说明错:输入数据的约束范围说明模糊。

(4) 算法错:计算给定数学方程的算法非正常执行。

(5) 边界错:数组索引越界。

(6) 分时错:分时源的必要连续性丢失,延迟或非正常处理。

简言之,软件失效是最后结果与有关规定不相符或用户在软件系统边界觉察到不希望的软件错误行为,而软件错误是被识别出或假设软件产生失效引起的。当这一区别不明显时经常用缺陷(defect)这一更一般的术语来指错误(原因)或失效(结果)。第九章从各个不同角度,给出了软件缺陷的全面而实际的分类。

另一个常用的术语是"错误"(error),它有两种不同的含义:

(1) 计算、观察或测量值或状态与真实的、规定的或理论上的正确值或状态之间不相符。计算机软件某些部分产生不希望的状态,亦即发生了差错。例如,存在的软件错误被激活引起异常状态和由意外的外部干扰使计算机产生不正常状态。该术语尤其适用于在容错计算中描述错误和失效的中间状态。在非容错系统中,错误等于失效。

(2) 人为因素使软件出现错误。如在软件说明书中缺少或误解了用户需求,在设计说明书中错误解释或缺少了需要的描述。这还不是最流行的用法,术语"过错"(mistake)常用来替代它,以避免混淆。

注意,两种含义的主要区别在于所考虑的"系统"不同。在(1)中"系统"指的是计算机系统,而在(2)中"系统"指的是程序员。除此之外,错误的注记都一致[lapr92a]。

### 1.2.2 时间

可靠性量化通常用相关时间来定义,尽管有时还可能用一些其它的,如程序运行等来定义。这里的时间与三类时间有关。软件系统的执行时间是运行软件时计算机实际花费的 CPU 时间。第二类时间为日历时间,指通常以年、月、周、日计的时间。第三类为时钟时间,是指在运行软件时计算机从头到尾所花去的时间。在测量计算时钟时间时,计算机停机时间不计在内。如果计算机被程序连续占用,该程序占用执行软件的 CPU 的一个时间段,则执行时间与时钟时间成正比。

对于软件可靠性测量和建模而言,通常认为执行时间比日历时间更充分。然而,为了对用户更有意义,可靠性量化必须最终与日历时间相联系。当经理、工程师和其他用户们要将结果与不同系统做比较时,尤其如此。因此,需要在日历时间与执行时间之间进行转换。这类转换技术见[Musa87a]。如果不容易得到执行时间,可用其它时间近似值,如时钟时间、加权时钟时间、主要运行时间或其它应用程序易得的计时单位,如可以使用处理或测试箱等。

一旦时间基准确定后,失效可以用三种方式表示:累积失效函数(the cumulative failure function),失效密度函数(the failure intensity function)和失效平均时间函数(the mean time to failure function)。累积失效函数(也记为均值函数)表示与每一时间点相关的平均累积失效。失效密度函数(也记为失效率函数,failure rate function 或失效发生率函数 rate of occurrence of failures)表示累积失效函数的变化率。平均失效时间(meantime to failure MTTF)表示观察到下次失效的期望时间。显然,上述三种度量标准是密切相关且可相互转化的。例如,失效密度是由累积失效函数相于对时间推导所得到的瞬时值。进而,如果可靠性函数成指数分布,其平均失效时间为失效密度函数的导数。

与时间有关的另一个量为平均修复时间(mean time to repair MTTR),它表示在观察到失效后,修复系统所需要的时间。对硬件部件来说,平均修复时间是一个很容易估计的评价标准,因为典型的硬件修复过程是通过诊断系统出错部件,用同样的或等效的部件替换它。这一标

准过程致使对平均修复时间的估计十分精确。然而对于软件，情况就不同了。软件失效的暂时恢复可以是简单的重装软件或重运行软件直到失效再次出现，而永久性地恢复需要调试、修改、确认、重装该软件。因此，软件的 MTTR 因系统而异且因时间而异。

当系统的 MTTF 和 MTTR 测定后，其可用性(availability)即可求得。可用性是指在需要时系统可用的概率。通常，它可如下计算：

$$\text{可用性} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

第十一章(11.8 节)给出了一些测量实例。

### 1.2.3 操作环境

软件操作环境通常以操作概图(profile)的概念来描述，它指明软件操作所需的环境(如用户初始的输入状态或影响计算的系统状态)。系统的操作概图定义为由软件可执行的操作及其发生的概率组成的集合。操作指一组包含相同处理的运算。典型的操作概图如图 1.1 所示，一般性的操作可以在 X—轴按其发生的概率排序。

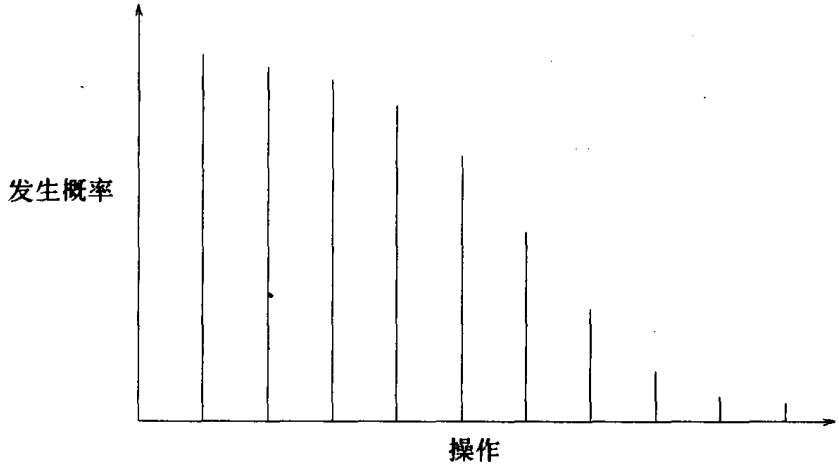


图 1.1 操作概图

第五章将详细论述关于操作概图的结构、发展、说明、工程应用。通常，软件操作数量可能极大。当确定所有操作及其概率的全部细节已不实用时，常使用基于输入状态(或系统状态)分组或划分成域的操作。第五章也有详细论述。当无法得到操作概图或只能得到近似图时，可以利用可靠性增长测试时得到的代码覆盖数据来评价可靠性。

第十三章给出了使用此策略的一些方法。

### 1.2.4 失效数据采集

为评测软件可靠性，通常采集两类失效数据：失效—计数数据(failure-count data)和失效间隔时间数据(time-between-failures)。

□ 失效计数数据

此类数据跟踪单位时间内检测到的失效次数，常见于工程报告和安装说明中。某些软件可靠性模型的公式中直接采用的该数据。

表 1.1 给出了一个典型的失效计数数据集。大多数组织相应地有某种结构管理程序。做为该程序的一部分，有一个报告失效的过程和一个对消除软件错误的修改进行认可的过程。

表 1.1 失效计数数据

| 时间（小时） | 周期内失效数 | 累积失效数 |
|---|---|---|
| 8 | 4 | 4 |
| 16 | 4 | 8 |
| 24 | 3 | 11 |
| 32 | 5 | 16 |
| 40 | 3 | 19 |
| 48 | 2 | 21 |
| 56 | 1 | 22 |
| 64 | 1 | 23 |
| 72 | 1 | 24 |

软件问题报告机制既可以是手工的也可以是自动的。而问题报告(如软件毛病报告等)可以存储在计算机数据库系统中或手工文件系统中,关键在于数据必须易于提取。在规定有时间间隔时容易出错,要百般小心,目的是在特定环境下建立检测每一时间单元之内失效次数的模型。时间单位应在 CPU 速度、人员占用时间、测试强度等方面保持一致。通常无法得到用于检查一致性的有关信息,所能得到的仅是每一时间间隔内的失效计数。而在每一时间间隔内的测试强度可能不一致。例如,某间隔内的测试成员可能为其它间隔内测试成员的二倍。得到这一信息需要很多附加工作量来与测试者交流信息,甚至还要重新检查所关心的过去时间记录。总的来说,形成失效计数的时间越长、结果就越稳定。短时间间隔内产生的偏差在长时间单元内被平均掉了。

另一个难点是,由于错误修复通常不是即时的,因此可能由一个相同错误导致多个失效。其结果是很难采集数据以确定哪次失效与哪个单一错误有关。这给问题报告的及时性和精确性增添了不确定因素。

问题报告中的其它难点包括如何确定问题是真的出现软件失效——因为某些组织用问题报告来反映任何一种形式的异常。另外,问题报告上所反映时间可能不是失效检测的时间,而是填报告的时间。失效计数数据可以在以系统测试开始的开发周期的任一点采集。至于报告率(每周、每月或每年报告失效数),可视用户的总体测量目标而定。建议采用与目标一致的最小时间单元内所报告的失效次数。如果结果不够好,将时间间隔组合到下一级。例如,日到周,周到季等。前面提到的稳定结果有助于模型的建立。

□ 失效间隔时间数据

典型的失效间隔数据详见表 1.2。此类数据可以通过自动在线过程监测生效并记录失效次数而直接得到,也可间接地人工跟踪失效记录,细察失效的后序时间来得到。如果可能的话,最好采集程序的实际执行时间,而不是花费的时钟时间,这是处理器在执行软件指令时所花费的实际时间。它可以给出使用软件的更真实的描述。有时,可能花费了大量的时钟时间,但在此期间内却仅做了很少的计算。这导致了较少的执行时间,因而倾向于给出对软件可靠性过分乐观的估计。使用执行时间建模,较之于单纯使用时钟时间更有优越性。但我们的确要因数据更难得到而多花一些代价,因为必须要有一个实际操作系统的监视器。

另一种得到此类数据的可能方法是通过代表每一时间单元内平均利用率的算法来调整时钟时间。如果无法得到失效间隔时间(time-between-failure)(时钟或执行时间)数据,而仅有组数据(每一时间单元发生失效的次数),失效间隔时间仍可通过以后将要论及的方法间接得到。

其它要考虑的因素还有:

(1) 调整失效次数以反应进化程序(随时间而改进的软件程序)的行为。

(2) 处理多地址、多版本的软件。

在第一种情况下,系统测试初期,当所有软件都不可用时,失效强度在软件开发早期可以忽略不计,进而产生过度乐观的可靠性预测结果。针对此问题,通常存在三种解决方法

〔Musa87a〕:

表 1.2 失效间隔时间数据

| 失效数 | 失效间隔(小时) | 失效时间(小时) |
|---|---|---|
| 1 | 0.5 | 0.5 |
| 2 | 1.2 | 1.7 |
| 3 | 2.8 | 4.5 |
| 4 | 2.7 | 7.2 |
| 5 | 2.8 | 10.0 |
| 6 | 3.0 | 13.0 |
| 7 | 1.8 | 14.8 |
| 8 | 0.9 | 15.7 |
| 9 | 1.4 | 17.1 |
| 10 | 3.5 | 20.6 |
| 11 | 3.4 | 24.0 |
| 12 | 1.2 | 25.2 |
| 13 | 0.9 | 26.1 |
| 14 | 1.7 | 27.8 |
| 15 | 1.4 | 29.2 |
| 16 | 2.7 | 31.9 |
| 17 | 3.2 | 35.1 |
| 18 | 2.5 | 37.6 |
| 19 | 2.0 | 39.6 |
| 20 | 4.5 | 44.1 |
| 21 | 3.5 | 47.6 |
| 22 | 5.2 | 52.8 |
| 23 | 7.2 | 60.0 |
| 24 | 10.7 | 70.7 |

(1) 根据成员结构改变检查程序变化,包括增、删独立子系统。

(2) 忽略变化。因为模型会调整其参数,自动解决问题以适应变化。

(3) 如果全部最终程序都存在的话,调整失效间隔数据到其应有状态。然后按对静态程序采用的办法来处理失效时间。

在后一情况中,存在着从多地址中组合可靠性数据的问题。该情况产生于多个代码版本以不同的处理速度,在不同位置或不同计算机上执行。通常采用的办法,是将不同计算机上的失效按其发生顺序交替排序。然后,求不同计算机上并行执行的时间之和,以求得后续失效之间的总时间。如果时间单元是日历时间或计算机有相同的平均指令执行率,选择一台"参照机"(reference computer),并根据该机调整全部执行时间和失效强度。注意,这通常需假设失效发生而且导致失效的错误是可修复的,对于所有软件备份也可同时修复。事实上,修复时间上可能有差别,但由于复制错误通常不计算在内。同时,修复仍可视为很好的近似。

□ 两类输入间的转换

许多可靠性建模程序都有根据失效计数或失效时间间隔数据估计模型参数的能力,而统计建模技术可兼施于两者。如果程序只提供了一种类型的数据,可能有必要变换另一种。

如果期望输入是失效计数数据,可以先将失效时间间隔变换成累积次数数据,然后简单地计算在规定时间内发生失效的累积次数。显然,将表 1.2 中的失效时间间隔数据转化成表 1.1 中的失灵次数数据的过程十分直观。如果期望输入数据为失效时间间隔数据,将其转化成失效计数数据需要某些假设。可以通过以下两种办法完成转化:

第一种方法,在规定时间间隔内随机分配失效。随机性对某些模型不会造成 15%以上的估计差错〔Musa87a〕。

第二种方法最易实现,在时间长度内均匀分配失效。例如,设时间间隔为三小时,在此期间出现三次失效。可以认为失效时间间隔为一小时。图 1.2 给出了三种在三个一小时期间分配三个失效的可能方法。在(a)中每次失效发生在一小时期间的开始,(b)每次失效发生在一小时期间的中间,(c)每次失效发生在一小时期间的结尾。

应该注意,通常并不提倡从失效计数到失效时间间隔的转化,除非十分必要(如模型

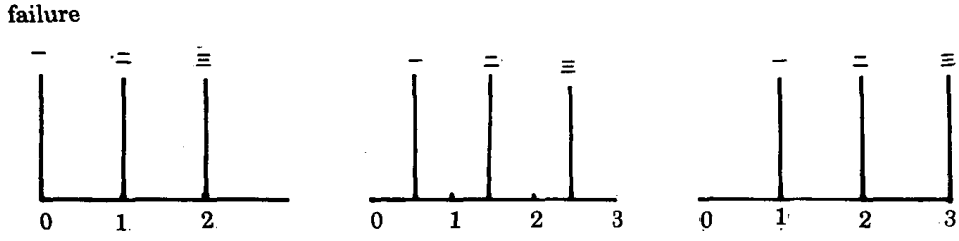


图 1.2 根据一致分布分配失效的方法

需要失效间隔时间),因为它会引起模型操作的噪音。显然,它假设在失效计数数据采集过程中无任何可靠性增长,该假设并不是真的。在任何情况下,当所考虑的时间间隔合理的短时,噪音可以减小。

附录 A 中提到的一些软件可靠性工具(如 CASRE)具有进行数据转换的能力。

### 1.2.5 评价与预测

软件可靠性测试包括两种行为。可靠性评价和可靠性预测,如下所示:

评价——应用统计推理技术于系统测试和系统运行期间得到的失效数据,断定系统当前的软件可靠性。它是对从过去到当前点所得到的可靠性的度量。其主要目的是评价当前可靠性,并确定一个可靠性模型是否为回溯的正确依据。

预测——利用可知的任何软件度量与规程确定未来软件的可靠性。根据软件开发的不同阶段,可采用不同的预测技术:

(1) 在可知失效数据时(如软件在系统测试与操作期),可用评价技术来定参和检证软件可靠性模型,可完成对未来可靠性的预测。

(2) 在不知失效数据时(如软件在设计与代码生成阶段),软件开发过程中得到的规程和所得产品的特性,可被用来确定所测试和提供的系统的软件可靠性。

前一定义又称"可靠性预测"(reliability prediction),后一定义又称为"早期预测"(early prediction)。当上下文意义清楚时,通常只用"预测"一词。

为进行可靠性预测(依赖于可靠性评价技术),考察某些软件产品的实际失效数据是十分必要的。它可以是,比如说在系统测试早期记录的数据(失效计数或失效时间间隔)。该数据用于校准统计评价模型,使之适于应用,如根据数据选择模型。校准后的模型可用来测量和预测软件产品的可靠性。

而另一方面,进行早期预测无需真实数据。而用其它参数来刻画软件产品及其开发、测试和维护过程。早期预测技术通常用来求静态量度标准如固有错误密度、期望错误总数和初始错误率。通常在系统进入测试期之前执行一次。早期预测对软件管理中的耗费评价(Cost estimation)、资源计划(resource planing)方案确证(schedule validation)和质量预报(quality forecasting)都十分重要。

可靠性预测法需要执行软件,而早期预测法则可在执行软件之前进行。换言之,在确定软件可靠性方面早期预测的及时性(timeliness)较优。然而,作为连续适应过程的可靠性预测,因

为评价模型根据失效数据精确度量软件可靠性，所以在对软件可靠性的度量精度方面较优。另一方面，早期预测模型不考虑可靠性的增长，也不精确建立软件可靠性模型。从这些模型出发很难建立作为依赖时间的度量标准的可靠性和作为静态测量标准的错误密度之间的关系。表 1.3 概括比较了这两个技术。

当前大多数软件可靠性模型属于评价类。文献中也给出了几种早期预测模型。第三章给出了现有的评价模型和一些早期预测模型的一览表。第十二章给出了一些用于早期预测的开发和生产规程。

**表 1.3 早期预测和可靠性预测的关系**

| | 早期预测 | 可靠性预测 |
|---|---|---|
| 何时使用? | 预测前期 | 测试运行期 |
| 测试目的 | 初始可靠性和失效率 | 连续可靠性评价 |
| 依赖失效数据? | 不依赖 | 依赖 |
| 需要辅入数据 | 过程和产品评价 | 失效数据 |
| 输出数据 | 错误密度，总错误数 | 全部可靠性相关测度 |
| 性质 | 静态、单时测度 | 动态，连续适应 |
| 可靠性预测 | 直接换算 | 直接可测 |
| 时间性 | 早期 | 晚期 |
| 精确度 | 不确定 | 更好 |

### 1.2.6 其它术语

其它软件可靠性工程(SRE)术语在本书的各章第一次出现时将详细介绍。以下文献中也给出了标准词汇表和辅助信息。

(1) ANSI/IEEE STD T29 - 1991，“IEEE Standard Glossary of Software Engineering Termindogy”。

(2) MIL-STD 756，“Reliability Modeling and Prediction”.

(3) ANSI/AIAA R-013-1992，“Recommended Practice for Software Reliability”，AIAA，Washington D.C，1993.

(4) IEEE STD 982.1 - 1988。“IEEE Standard Dictionary of Measures to Produce Reliable software”.

(5) IEEE STD 982.1 - 1988，“IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software”。

### 1.2.7 实例

例 1.1 给出了软件可靠性对系统可靠性影响的一个例子。

例 1.1

某军用分布式处理系统需要一百小时的 MTTF 和 0.99 的可用率。图 1.3 给出了系统的总体结构，系统包括三个子系统，SYS1、SYS2、SYS3、一个局域网(LAN)、和一个 10kW 的发电机 GEN。为使系统运行，所有的成分都得正常工作。在系统测试早期，可根据军用可靠性手册 Mil-217 来预测硬件可靠性，且给出每个组件的可靠性。在图 1.3 的每个组件块上有两个数

字,上面的代表该组件的 MTTF,下面的代表它的 MTTR,单位为小时。例如,SYS1 的 MTTF 为 280 小时,MTTR 为 0.53 小时,而 SYS2 和 SYS3 的 MTTF 为 387 小时,MTTR 为 0.5 小时。注意到在点划线中框住的 SYS2 被构造成一个三模块冗余系统 La triple module redundant system),只要两个以上的模块能正常工作,该子系统就能正常工作。由于具有了容错能力,其 MTTF 改进为 $5.01 \times 10^4$ 小时,而 MTTR 变为 0.25 小时。

为计算总系统的可靠性,必须考虑系统的所有组件。我们假设软件不出错(这是系统可靠性工程师常犯的一个错误),结果是系统 MTTF 为 125.9 小时,而 MTTR 为 0.62 小时,达到系统可用率为 0.995。这似乎已经达到了初始要求。

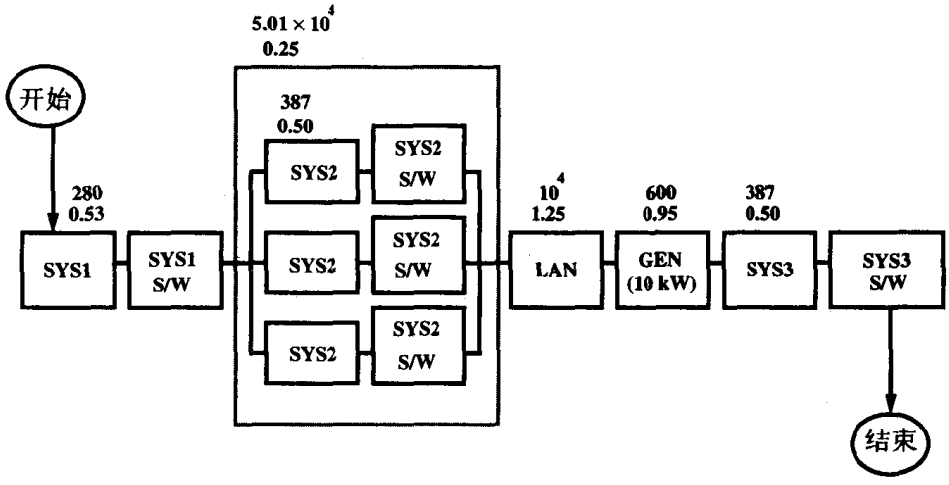


图 1.3 预测系统可靠性实例

事实上,软件是会出错的。SYS2 和 SYS3 软件都有 300,000 行源代码,根据第三章(3.8.3 节)和[RADC87a]中给出的预测模型,SYS2 和 SYS3 软件的预期初始失效率都为每个执行小时 2.52 个。即使不考虑 SYS1 的软件失效,系统 MTTF 也将变成 11.9 个 CPU 分钟! 假设 MTTR 仍为 0.62 小时(尽管由于修复软件需要更多的时间,将导致更高的 MTTR 值),且 CPU 时间与日历时间十分相近(对于此分布系统,此假设为真),系统的可用性变为 0.24,远低于先前的预测!

注意到例 1.1 给出的是一个现实世界上的实例,可靠性参数评价采用实用的军用标准手册[Lyu89a]。该例子并非是一个极端的情况。事实上很多现存的大系统的确面对相同的情况:

软件可靠性是系统可靠性的瓶颈,软件的成熟总是滞后于硬件的成熟。由于能为大多数工程的决策制定和可靠性设定提供关键信息,精确地构造软件可靠性模型并预测其趋势已变得十分重要。

## 1.3 软件可靠性建模

软件可靠性和硬件可靠性一样,都是随机过程且都可用概率分布描述。但软件可靠性与硬件可靠性的不同之处是软件不会老化(衰退或退化)亦即,其可靠性不随时间而减少。而且,在软件测试和运行过程中,由于软件失效时软件错误可以被检测出来并被排除掉,因而软件的可靠性会增高。另一方面,软件可靠性也可能因应用项的意外改变和对软件的不正确修改而降低。软件在其整个生命周期中也是需要不断更新的。软件的可扩展性,也使得人们不可避

免地要考虑不同的失效率。

大多数硬件错误是物理故障(physical faults)。与此不同的是,软件错误为设计错误,它更难看到,更难分类、更难检测,也更难修改。因此,与硬件可靠性相比,软件可靠性是一个更难得到,更难分析和测量难解题。由于只考虑物理故障,硬件可靠性理论通常依赖于分析静态过程。然而,随着系统复杂性的增加和软件设计错误的引入,基于静态过程的可靠性理论变得不适于表示诸如软件中的可靠性增长或衰减之类的非静态现象。这使得软件可靠性测量问题成了一个更具挑战性和不断发展的技术。

### 1.3.1 建模原理

软件可靠性模型说明了失效过程对影响它的主要因素的通用依赖形式:错误引入、错误消除、操作环境。图 1.4 给出了软件可靠性建模的基本思想。

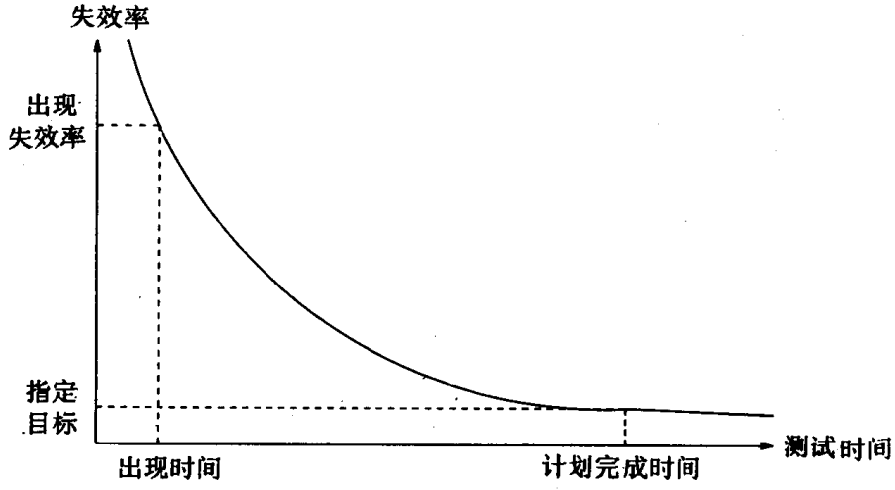


图 1.4 软件可靠性建模的基本思想

在图 1.4 中,软件系统的失效率总体上随着软件失效的发现与消除而减小。在任一给定时间(如标有当前时间的点),可以观察到软件失效率的历史。软件可靠性建模技术通过统计结果预测失效率曲线。其目的有两个:

(1) 预测达到规定目标还需要多个测试时间;

(2) 预测测试结束时软件的期望可靠性。

### 1.3.2 软件可靠性模型评价

自第一个软件可靠性模型于二十多年前提出以来[Jeli72a],文献中已记述了四十多个软件可靠性模型。导致如此的模型泛滥是尚未被人们充分认识的软件工程领域非常复杂,而软件生产及其过程又丰富多彩。因此,必须给出简化假设以保证建立的可靠性模型易用数学及统计学处理。为此,不同模型分别引入了一些(但非全部),软件的过程/产品特性,这些特征与可靠性评价有关。同时给出了关于软件发展、测试和错误修改过程的不同假设。

对不同模型的应用经验表明,没有一个普遍适用的模型能对所有产品都做出最好的可靠性评价和预测[Lyu92a]。如,模型 A 可能对产品 1 精确而对产品 2 不精确,而模型 B 可能对产品 2 精确而对产品 1 不精确。由于没有在所有情况下都优秀的单个模型,很有必要为实践者建立关于选择哪一个模型以及对软件可靠性测量采用什么过程的指导性准则与推荐性意见。

我们将在第三章详细讨论这些模型及其基本假设。第四章给出了一个在具体工程中选择和应用各种适用模型的高精技术。第六章与第七章论述了在模型应用方面当前的实践与经验。

为建立对软件模型的总体评价,建立了以下评价标准:

(1) 论断有效性(predictive validity);

(2) 测量参数的简易性(Ease of measuring pavameters);

(3) 假设质量(Quality of assumptiors);

(4) 能力(Capability)

(5) 应用性(Applicability);

(6) 简单性(Simplicity);

(7) 对噪音的不敏感性(Insensitivity to noise)。

以下详细论述这些评价标准:

(1) 论断有效性

论断有效性包括对当前失效密度的测量精度,以相关数据与耗费完成测试的时间预测,以及对操作失效率的预测。

为客观而定量地比较不同模型,采用了四种形式化定义的测度。这些测度(精确度、倾向性、趋势和噪音)代表对某一特定模型的软件可靠性测量质量的各种量化。第四章将给出这些测度的数学细节。

(2) 测量参数的简易性

该评价标准指的是模型所需要的参数数量以评价这些参数的难度。大多数模型引入两到三个参数。通常,三参数模型较之于二参数模型与失效数据曲线更吻合。但对评价软件可靠性来说,由于在某些情况下二参数模型已能精确地处理可靠性,因此不一定要采用三参数模型。当两模型确定度相同时,需要参数较少的模型为好。这不仅仅因为模型的参数越少越容易处理,还因为软件可靠性工程师能更定量化分析参数的物理意义并为软件开发过程提供适当的反馈。经验表明,以相当简单的假设给模型参数一个物理解释不是一件容易的事情,尽管参数在一开始有一定的物理含义。例如,当应用一个含与初始错误数相关的参数的模型时会发现从一个预测阶段到下一个预测阶段该参数剧烈波动,即使后续的可靠性预测相当准确,也是如此。

从评价参数的过程来看,最大似然法(Maximum Likelihood)和最小幂法(least Square)是两个最常用的方法。它们的数学形式见附录 C。第三章将论述最大似然评价及其相关各模型。软件可靠性模型的论断性行为可能因使用不同的参数评价过程而明显不同。同样,如果结果没有不同,评价过程越简单越好。

(3) 假设质量

软件可靠性模型所作的假设应该尽量与现实工程测试和操作环境相近。软件可靠性模型中常用的假设有:

(a) 测试输入随机地遇到错误;

(b) 所有失效结果是相互独立的;

(c) 测试空间覆盖使用空间(即操作概图);

(d) 所有失效在其发生时都可恢复;

(e) 错误会在失效时被及时地清除或不会再遇到;

(f) 软件失效密度与软件中存在的错误数有关;软件可靠性模型规定这一关系。

如果假设是可以验证的,应该用数据来证明此假设。如果假设不可检验。应从逻辑一致性和软件工程经验的角度检验它。另外,应判断模型所采用假设的明析性与精确性。这有助于确定在软件工程环境中是否采用某一特定模型。第三章详细讨论几个特定的模型假设。

(4) 能力

能力指模型评价软件系统中与可靠性相关的量的能力。这些量包括:

(a) 在给定时间内被评价的固有错误,期望的全部失效或残留的错误;

(b) 当前可靠性 MTTF 或失效密度;

(c) 达到规定可靠性所期望的数据 MTTF,失效密度目标,或遗留的错误数;

(d) 与实现目标有关的人、机资源及耗费需求。

除具有在测试与运行期进行软件可靠性预测的能力外,模型具有在系统设计与早期开发期进行早期预测的能力也很重要。这些预测必须通过软件规程,软件开发环境及操作图来检验。

(5) 可用性

软件模型的可用性应根据不同的代码规模、结构、功能和应用领域来检验。某特定模型的一个优点是它可用于不同的开发环境,不同的操作环境,及不同的软件工程阶段。在应用软件可靠性模型时,模型应可处理以下情况:

(a) 进化软件;

(b) 失效严重程度分类;

(c) 不完全失效数据;

(d) 软件在各种不同速度计算机上的执行;

(e) 多重软件安装;

(f) 与模型假设有差别的工程环境;

(6) 简单性

在选择模型时,应注意简单性。模型越简单,参数越少,收集工程数据,从数据库中选择通用参数以及理解与解释结果就越容易。直到已经进行了几次可靠性预测与评价之后,才需要更复杂的模型,也才会再有数据支持更多更复杂的模型。

简单性表现在模型的三个方面:数据采集过程、建模概念及其使用软件工具的实现过程。数据采集简单可以减少测量耗费、增加数据精度、使模型易于应用。建模概念简单则更容易理解假设、评价参数,使用模型和解释结果。模型易于实现有利于充分利用计算机解决通常的计算密集型(Computation-intensive)问题。

但应该注意,不应过分强调简单性标准在论断有效性标准中的作用。当引入一个额外的模型参数能非常有效地改进论断有效性时,即使参数评价过程会因此而难度大增,使用新模型仍是十分必要的。

(7) 对噪音的敏感性

软件可靠性数据通常包含与模型处理过程不相干的噪音。最常见的噪音是由于软件失效数据通常是基于工程日历时间记录的,而不是以执行时间记录的。即使精确地以执行时间(或至少以对其足够的近似)来记录软件失效,真实的软件执行过程也往往可能与模型假设不一致(如,软件测试不随机)。因此,一个好的模型不仅能在使用纯净数据的理想情况下表现其有效性,还应能在失效数据集(the set of failure adta)不完全或含有测量不确定性时,完成精确测量。

该标准可被认为是模型可用性评价标准的特殊情况。可以想象;对模型的噪音不敏感进