工业软件研发、测试与质量管理论丛

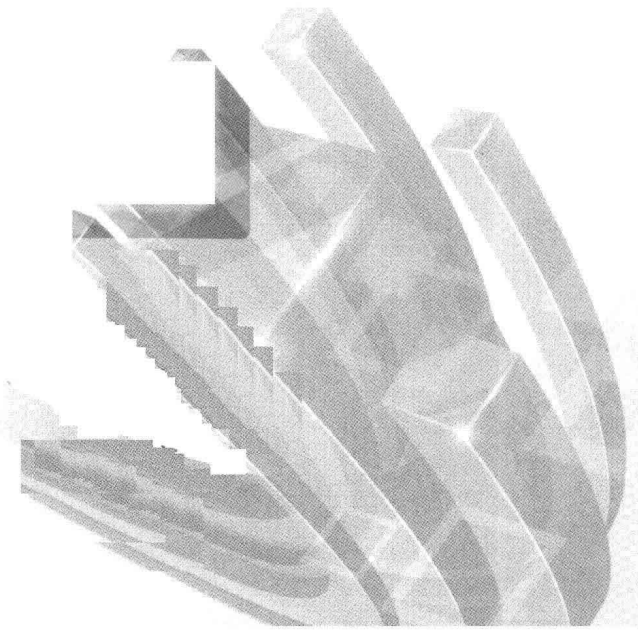
主编 王蕾

上海社会科学院出版社

上海市软件评测中心
上海社会科学院信息研究所 编

工业软件研发、测试与质量管理论丛

主 编 王 蕾
编 委 王 操 孟 艳
朴希望 李世群



上海社会科学院出版社

序　言

随着数字计算技术的迅速发展和广泛应用，数字系统（包括软件和硬件系统）与信息的可靠性和可信性问题已经引起了人们的高度重视与关注。提高可靠性和可信性的关键在于如何消除埋藏在系统和信息中的隐患，而消除隐患的关键则取决于测试理论和测试技术的开发研究与应用。

近年来，社会对计算机软件的需求量高速增长，因此，人们对软件可靠性的认识愈来愈深刻，对它的要求也愈来愈高——质量是软件的生命，测试是软件的灵魂。随着软件产业的高速推进，我国的软件评测企业大幅增加，目前我国已有不下2000家软件测试企业，软件测试专业人员已经接近二十万人，软件测试产业年产值已超过百亿元。国际上，软件产业的巨头不仅在其本国成立了各种大型软件测试和质量管理机构，还纷纷来到我国组建以软件测试外包为主导的软件企业。全世界以软件测试技术和软件测试服务以及软件质量管理为核心的软件研发与软件测试产业每年正以超过20%的速度递增。在此形势下，国内在软件测试技术和质量管理方面的学术交流与科研活动异常活跃，新颖的测试理论和测试技术不断涌现，引人关注的软件质量管理方法层出不穷。

为了加快推进上海信息化与工业化融合，促进上海工业软件产业快速、稳健地发展，近期由上海工业软件工程中心主办的“工业软件测试与质量管理高峰论坛”就是当前软件测试和质量管理方面的一个十分成功的学术交流活动的例子。这次论坛为工业软件领域的广大科研人员和科技工作者提供了一个交流、探讨、学习和分享智慧的平台。

工业软件是在我国推进信息化与工业化融合的进程中产生的概念，是对应用于工业领域里的计算机软件的统称。它是一种专用于或主要用于工业领域，为提高工业企业研发、制造、经营管理水平和工业装备性能的应用软件。工业软件是工业领域信息化系统中血液，也是信息化与工业化融合的纽带和桥梁。工业软件集设计、制造、检测、管理数字化、智能化技术于一体，在各门类工业中已经得到了广

泛的应用。

由上海市软件评测中心和上海社会科学院信息研究所组织编写的《工业软件研发、测试与质量管理论丛》一书就体现了本届高峰论坛的交流成果，也是在人类经济社会信息化进程和信息安全技术高度发展的背景下应运而生的产物。本书精选的二十多篇高质量的论文和技术报告，是高校、科研机构和不同行业的具有丰富经验的软件测试和软件质量管理科研人员、教授、专家的理论研究和工作实践的结晶。本书从理论、方法和应用三个视角透视了上海地区广大软件测试和质量管理科研人员经过长期探索、研究和实践，在软件测试理论、技术和软件质量管理方面所取得的丰硕成果。本书的宗旨就在为不同岗位上从事工业软件测试和质量管理方面的专家和技术人员提供实用的理论知识与技术支撑，为他们从事的科研和技术工作服务。希望本书对促进我国软件测试理论和测试技术以及软件质量管理技术的进一步发展能起到有力的推动作用。

徐拾义

2012 年 3 月

前　言

工业软件是信息化与工业化融合发展的产物，是信息化与工业化的粘合剂，大力发展工业软件是工业领域信息化的重要主题之一。“两化融合”的目标是将信息技术与工业过程紧密结合起来，以实现产品设计数字化、过程控制自动化、企业管理信息化、装备功能智能化，而这一目标的实现必须以工业软件的应用为基础。工业软件，顾名思义就是利用信息技术手段将工业企业的运营和管理流程、工业产品的研发和设计流程以及工业产品加工制造过程的控制逻辑加以软件化、代码化，从而为工业企业的运营管理和工业产品的研发设计提供便捷高效的辅助手段，驱动管理业务、设计业务和制造装备按照既定的逻辑自动高效地运行，以实现预定的管理、技术或控制功能。因此，工业软件可以细分为面向企业经营的管理软件、面向研发与设计过程的辅助软件、面向制造过程的自动化控制软件和面向设备装置的嵌入式软件。随着我国信息化与工业化深度融合的推进，工业软件作为工业领域信息化的基础和灵魂，在“两化融合”的过程中扮演着越来越重要的角色。可以说，没有工业软件就不可能实现以信息化带动工业化的目的，形成信息化与工业化相互促进、融合发展的局面，也就没有“两化融合”。

本论丛是在上海市软件评测中心主办、上海社会科学院信息研究所承办的“工业软件测试与质量管理高峰论坛”的会议论文基础上编辑而成的。编辑出版本论丛，旨在以论坛直接互动平台为基础，为工业软件领域的广大工作者提供一个长期交流、探讨和分享智慧的平台。

2011 年 12 月 19 日，在上海市经济和信息化委员会支持下，在漕河泾新兴技术开发区企业协会协助下，由上海市软件评测中心主办、上海社会科学院信息研究所承办的“工业软件测试与质量管理高峰论坛”在上海社会科学院上海社科国际创新基地顺利召开。论坛邀请了上海市经信委软件处领导，上汽集团、沪东造船、交通电子行业协会和同济大学软件学院等工业领域的高级专家、知名学者、权威人士及相关企业的广大工程技术人员就工业软件的研发、测试和质量管理等主题进行了广泛探讨和交流。

本论丛的如期编辑出版及其所依托的“工业软件测试与质量管理高峰论坛”的圆满成功得到了上海市软件评测中心有限公司“工业软件公共测试与验证平台项目”的支持，是该平台项目的重要成果之一。上海市软件评测中心有限公司工业软件公共测试与验证平台项目于2009年10月获得国家发展改革委、工业和信息化部的“电子信息产业振兴和技术改造项目”立项批准。该项目技术改造的重点是在已有测试环境基础上，购置必要的设备、仪器，建设工业软件公共测试与验证平台。

本论丛汇集了“工业软件测试与质量管理高峰论坛”的主要论文，按照理论、方法和应用三大部分编选，涉及了工业软件的研发、测试与质量管理三个方面的内容。其中，第一部分为理论篇，共选编了5篇论文，分别研究了软件可靠性精确度量方法和模型、Linux进程切换时间和优先级倒置解除时间精确测量方法、优化测试码间距和提高随机测试效率的方法、字节码级Java程序动态故障注入工具原型的设计与实现、软件测试性研究前沿等理论问题。第二部分为方法篇，共选编了7篇论文，主要探讨了工业软件的测试方法与模型、工业软件公共测试与验证平台、第三方工业软件测试项目的管理、嵌入式软件测试规范、工业管理软件的设计与实现、轨道交通运控系统中的车速计算和安全软件研发等方法问题。第三部分为应用篇，共选编了12篇论文，主要剖析、阐述了工业软件的第三方品质保障服务、上海工业软件测试行业的发展对策、面向客户关系管理系统的化工企业客户数据挖掘、质量管理软件的应用等应用问题。

由于本论丛汇集了众多作者的研究成果，不同作者的研究内容、视角、方法及叙述风格存在着不尽一致的情况，研究的范围和深度方面也可能存在较大差异，提出的观点、方法、模型、应用有些还不够成熟，敬请有关专家、学者和广大读者批评指正。

编　者

2012年3月

目　录

理论篇

方法篇

应用篇

理 论 篇

工业软件可靠性精确模型研究*

徐拾义　张　鼎

摘　要：软件可靠性是数字系统可信性的重要属性之一。本文首先讨论了传统软件可靠性模型的主要缺陷。在传统的软件可靠性度量模型中，并未涉及大多数软件自身的复杂性及测试用例的有效性等问题，也没有考虑软件本身的健壮性，从而在评估软件的可靠性时无法获取精确的结果，甚至得出错误的论断，使人们因对所应用软件的可靠性评估有误而导致严重事故。为克服上述缺陷，本文提出了新的针对苛求型工业软件可靠性的度量模型，该方法的主要思想是将可能影响软件可靠性度量模型的相关因素如软件自身的复杂性和测试用例的有效性以及软件本身的健壮性等都融入到软件可靠性度量模型中，以获得一个能反映软件实际工作情况的更为精确的可靠性模型。大量的实验结果表明了该模型的合理性及有效性。

关键词：软件可靠性模型；软件复杂性；测试有效性；健壮性；故障注入；可靠性度量

1　引言

软件可靠性理论是工业界用来预测软件发生故障和失效的重要方法之一。然而，当前应用的软件可靠性度量模型，并未考虑软件自身质量、软件复杂性、软件测试用例有效性和软件运行环境等诸多重要因素对软件可靠性评估所产生的重大影响，从而使人们对软件可靠性的评估很不精确，甚至得到错误的结果。

长期以来，软件可靠性模型和可靠性评估方法是来自于硬件可靠性评估的方法和思想。因此，软硬件测试的基本思想和软硬件可靠性模型评估的方法是十分相似的，但是，在具体实施方面，这两者之间仍然存在着较大的差别。软件可靠性

* 本文系国家自然科学基金资助项目(61076123)成果之一。

的评估要比硬件困难得多。这是因为，要对软件进行充分和精确的评估并不只是简单地用数学公式对被测系统赋予一个可靠性数值的问题，而是需要处理一个与时间和操作规程都密切相关的复杂任务。所以，软件测试和可靠性评估要比硬件复杂得多。传统的可靠性理论在硬件中应用可能获得很好的效果，但是，直接将该理论应用于软件的可靠性评估却会带来许多意想不到的困难问题。事实上，由于软件可靠性评估方法的不精确性和理论的不完善性，在工业界，软件可靠性理论几乎是很少在实际中使用的。为了进一步提高软件可靠性评估模型的精确性，充分发挥软件可靠性理论的指导意义，本文提出在应用传统的硬件系统可靠性理论常用的思想和方法外，至少还应该处理三个重要的并且只有软件可靠性评估中才会发生的症结，即：软件的复杂性和健壮性、软件测试的有效性和软件的运行环境。因此，如果需要对软件可靠性作更充分更精确的评估，则必须对上述三个影响软件可靠性评估的重要因素作认真和详细的研究，从而可以大大提高当前正在应用的软件可靠性模型的精确度。

2　传统的软件可靠性模型和软件故障排错率

一般来说，传统的软件可靠性模型是基于软件的运行时间、软件测试得到的检错率和排错率来进行评估的。而在该模型中大部分都使用所谓指数递减型的排错率来计算的，这是由于指数递减型排错率能够比较精确的反映和预测在程序运行时出现故障或错误的情况，因而受到众多研究者的青睐。在本研究中，假设所有检测到的错误都应得到正确的修正，并且未因修改而引入新的错误（即在今后的回归测试中应该不会再次出现同样的故障），设 E_0、$E_d(t)$ 分别表示潜伏在被测程序中的错误总数（通常，这个数字对测试者来说是未知的，但是，可以在后期进行估算得到）和测试过程中检测到的累积错误数，于是根据排错过程，有：

$$E_d(t) = E_0(1 - e^{-\beta t}) \tag{2.1}$$

其中，β 是与可靠性有关的一个系数。由累积错误数 $E_d(t)$ 对测试时间 t 进行求导可以得到：

$$E_r(t) = \frac{dE_d(t)}{dt} \tag{2.2}$$

事实上，一般称 $E_r(t)$ 为排错率（即单位时间内排除的故障数），由于不考虑其

他因素，则 $E_r(t)$ 也可以视为被测软件的失效率 $\lambda(t) = E_r(t)$。于是，可以十分简单地计算被测软件的可靠性为：

$$R(t) = e^{-\int \lambda} = e^{-\int E_r(t)t} \tag{2.3}$$

如果，$E_r(t)$ 为一常数，则可靠性可以表示为，

$$R(t) = e^{-\lambda} = e^{-E_r t} \tag{2.3-1}$$

现设 $n_i(i=1, \cdots, k)$ 表示第 i 天所检测到的累积错误数，我们可以通过最小二乘法来估算 E_0 的值，构造如下公式(2.4)：

$$\sum_{i=1}^{k} [n_i - E_0(1 - e^{-\beta t_i})]^2 = 0 \tag{2.4}$$

上式两边分别对 E_0 和 β 求偏导后可以得到如下两式：

$$E_0 = \sum_{i=1}^{k} n_i(1 - e^{-\beta t_i}) / \sum_{i=1}^{k} (1 - e^{-\beta t_i})^2 \tag{2.5}$$

$$E_0 = \sum_{i=1}^{k} n_i t_i e^{-\beta t_i} / \sum_{i=1}^{k} (1 - e^{-\beta t_i}) t_i e^{-\beta t_i} \tag{2.6}$$

由式(2.5)和式(2.6)，可以计算出 E_0 和 β 的值，同时也可以计算出相应的 MTTF 的值，如下所示：

$$\mathrm{MTTF} = \frac{e^{\beta t}}{E_0 \beta} \tag{2.7}$$

在给定排错率 E_r 的条件下，也可以得到相应的软件发布时间为：

$$t = [\ln E_0 \beta - \ln Er(t)] / \beta \tag{2.8}$$

表 2.1 描述了一个实际的软件测试过程，是一专业软件测试公司对一个规模为 25，324 行的程序进行测试的详细记录。每天(按 24 小时计算)检测到的错误数都记录在表 2.1 中。

从表中很容易发现，在这 20 天的测试过程中排错率非常接近指数递减型排错率，式(2.1)和式(2.2)所描述的可靠性模型完全适用于被测程序。因而根据式(2.3-1)可以得到被测程序的可靠性值计算如下：由式(2.5)和式(2.6)可以估算出 E_0 和 β 的值，$E_0 = 285$，$\beta = 0.0981$。

表 2.1　应用软件测试过程日志记录

测试天数	日检测到的错误数	累积错误数	测试天数	日检测到的错误数	累积错误数
1	27	27	11	21	207
2	12	39	12	10	217
3	14	53	13	8	225
4	41	94	14	5	230
5	27	121	15	4	234
6	28	149	16	2	236
7	14	163	17	1	237
8	8	171	18	0	237
9	5	176	19	1	238
10	10	186	20	0	238

这就是说，被测程序中总共可能存在着约 285 个错误。将 E_0 和 β 的值分别代入到式(2.1)和式(2.2)的模型中，则可以得到测试期间每天可能检测到的错误数为：

$$E_d(t) = 285(1 - e^{-0.0981t})$$

因此，也可以得到对该软件进行测试时的排错率为：

$$E_r(t) = 24.51e^{-0.0981t}$$

同时，还可以估算出在经过 t 时刻测试后，其剩余错误数估计为：

$$E_{rem}(t) = 285e^{-0.0981t}$$

在本例中，我们可以估算出该程序在经过 20 天的测试后所剩余的错误数约为 $E_{rem} = 285e^{-0.0981\times20} \approx 40$。下图描述了对本例程序测试得到的排错曲线图。

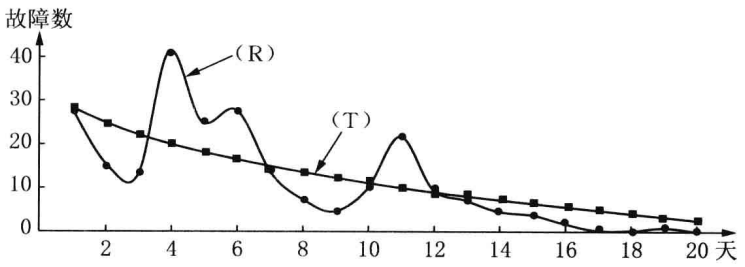


图 2.1　本例的排错曲线图

图 2.1 画出了两条曲线：曲线(R)和曲线(T)。其中，曲线(R)是根据测试运行时间方法得到的实际故障数，而曲线(T)则是按照指数递减型排错模型逼近曲线(R)所作出的排错率理论曲线。本例为了计算上的方便，将曲线(T)在 20 天的测试时间中得到的理论值看成是一条直线。因此，可以将该曲线近似地处理成线性递减型模型，可将其作为直线处理，并将该直线斜率作为被测软件的失效率 λ(即，λ 已经成为常数)，由此可得到该直线的斜率 $\lambda = 0.0014$，因此根据式(2.3-1)不难计算出经过 20 天运行测试后该程序的可靠性为：

$$R(t = 20) = e^{-\lambda t} = e^{-0.0014 \times 20} = 0.972$$

如果要求在软件发布前排错率 $E_r(t)$低于 0.1/天，于是，根据式(2.8)可得到其可能的发布时间为 $t = 52.31$ 天，也就是说，必须经过至少 52 天以上的测试，被测程序排错率(失效率)才可能下降到 0.1/天。

在上述表 2.1 中，实际上是使用传统的以测试运行时间和测试到的故障数为依据对被测软件可靠性度量作出评估的典型例子。该方法可能使人们对被测软件的可靠性度量有一个大致的轮廓，然而依测试运行时间作为依据，却只能反映程序质量的一个方面。传统软件可靠性模型之缺陷的要害就是它仅仅建立在测试运行时间上，而忽略了影响软件可靠性评估的诸如软件复杂性和测试用例有效性和测试覆盖率等其他不可或缺的重要因素。将可靠性的估算方法仅仅囿于使用类似式(2.1)，(2.2)，(2.6)或(2.7)那些仅与测试时间有关的公式来度量，实际上所得到的软件可靠性值往往是不可靠的，甚至还可能对用户或测试者产生误导，引起严重后果。因此，在软件可靠性模型中必须要考虑引入足以能体现可能影响软件可靠性评估的各种重要因素。

3 软件复杂性评估

为了使软件可靠性模型更为精确，被测软件自身的复杂性因素是不能不考虑的。事实上，进一步考察可以发现，评估软件可靠性必须考虑到以下所列的几个因素：

(1) 软件自身的复杂性。一般来说，软件越复杂就越有可能存在较多的故障，因而可能使得软件的可靠性下降，因此应尽可能降低软件复杂度。

(2) 测试的有效性和彻底性(即软件的健壮性)。可靠性模型是假定软件测试是建立在操作规程和测试有效性的基础上的。一个软件被认为具有比较高的可靠

性是仅仅对那些适用于这种操作规程而言的。如果修改测试用例和操作规程，那么对同一软件将会产生不同的可靠性值。操作规程本身不能保证得到一个精确的可靠性。如果输入(或侵入)非正常值，软件仍能保持正常运行或能自行修复，则该软件的可靠性应该比较高，反之，不可能有较高的可靠性。

(3) 运行环境和硬件。软件并不是在一个孤立的空间中运行的，相反，它驻留于硬件中(特别是嵌入式软件)并与周围环境紧密联系。操作系统是一般用户可以接触到的最底层的系统软件，它以特权用户的身份运行着并可以直接访问到硬件。一个实际的问题是目前大多数软件测试工具被设计成仅能处理人为输入，然而这会使我们对软件运行环境产生误解。

为了解决这些问题，本文在以前相关研究的基础上提出了一些新的思想以期能提高软件可靠性的精确性。为了说明方便起见，本节将先简要介绍有关定义和术语，其中有些定义可以在我们以前的研究报告和论文中找到。

通常我们所说的软件复杂性，即是软件模块的逻辑复杂性及模块之间的联系。它包括了模块中的变量数和运算符数，模块中的循环数和分支数，以及软件中的信息流和控制流等。为了简洁性，我们考虑了一些与软件复杂性相关的参数。首先，采用 Halstead 度量来评估程序的词法构成，即计算程序中所包含的操作数和运算符数，同时它也反映了词法构成对复杂性的影响程度。通常，Halstead 度量(记为 Hv)可以定义为：

$$Hv = (N_1 + N_2)\log_2(n_1 + n_2) \tag{3.1}$$

其中 n_1 和 n_2 分别表示程序中的不同运算符个数和不同的操作数个数，N_1 和 N_2 分别表示程序中运算符数出现的总数和操作数出现的总数。于是，我们给出以下与软件复杂性有关的定义。

定义 3.1　软件的变量和运算符复杂性被定义为：

$$\ln Hv \tag{3.2}$$

其中，Hv 是被测程序的 Halstead 度量。

另一个与软件复杂性有关的参数是 McCabe 圈数(记为 Ml)，它通常是用来度量程序所包含的循环数和分支数。我们把循环和分支复杂性定义为如下。

定义 3.2　软件的循环和分支复杂性被定义为：

$$\ln Ml \tag{3.3}$$

其中，*Ml* 是 McCabe 圈数，表示被测程序中的循环分支数。

软件中信息流复杂性度量是用来度量模块间的交互关系的复杂程度，并定义了专门的术语 *fan-in*、*fan-out* 来度量被测软件中信息交互的复杂程度。

定义 3.3 模块 P 的 *fan-in* 数是指从外部传入该模块的参数个数及该程序中所读取的全局变量的个数。

定义 3.4 模块 P 的 *fan-out* 数是指从该模块的返回给外界参数个数及该程序中所改变的全局变量的个数。

程序 P 的信息流量可以被定义为：

$$If = \sum [(fan-in) \times (fan-out)]^2 \tag{3.4}$$

式(3.4)中的(*fan-in*)×(*fan-out*)表示程序中的一个模块(或函数)所有输入源和输出目的之间的乘积。程序中信息流的总数是所有模块信息流的总和。而且，对任何一个模块来说，一般都至少有一个扇入和一个扇出，即 $fan-in \neq 0$，$fan-out \neq 0$，因而 $If \neq 0$。

定义 3.5 软件的信息流复杂性被定义为：

$$\ln If \tag{3.5}$$

根据上面介绍的影响软件复杂性的三个因素，我们现在可以定义一个更为精确的软件复杂度模型如下。

定义 3.6 软件复杂性模型 $\lambda(c)$ 定义为：

$$\lambda(c) = 1 - e^{-(\ln Hv + \ln Ml + \ln If)/3\ln LOC} \tag{3.6}$$

其中，*LOC* 是被测程序的代码行数。

表 3.1 重新考察表 2.1 中的程序，表 3.1 所列的是该被测程序的复杂性特征的相关参数的描述。

表 3.1 程序复杂性参数

参数	实际值	对数化后的值值
LOC	25324	10.140
Hv	76317	11.243
Ml	812	6.670
If	43861	10.689