软件可靠性综合模型的分析和研究

朱经纷 徐拾义
(上海大学计算机工程与科学学院 上海 200072)

摘 要 软件可靠性是系统可信性的重要属性之一。首先讨论了传统软件可靠性模型的主要弱点,接着提出了一种新的软件可靠性估算模型。当前,传统的软件可靠性度量模型中并未涉及大多数软件的复杂性及测试用例的有效性,从而使得在评估软件可靠性时不够精确,甚至是错误的。因此,提出了一些改进软件可靠性度量的方法。这些方法的主要思想是将影响软件复杂性的因素和测试用例的有效性并入软件可靠性模型中,以便使得该模型能更精确地反映实际情况。最后,大量的实验结果也表明了该模型的合理性及有用性。

关键词 软件可靠性模型,软件复杂性,测试有效性,故障注入,可靠性度量
中图法分类号 TP311 **文献标识码** A

Analysis and Research of a Synthesis Model for Software Reliability

ZHU Jing-fen XU Shi-yi
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

Abstract Software reliability is one of the important attributions of dependable systems. We discussed the major weaknesses of the software reliability models used currently and then proposed a new model of software reliability estimation. For the time being, the assumptions that software reliability measurement does not address the complexity of most software and the effectiveness of test suite, resulting in inaccuracy or even incorrectness in evaluating the software reliability. Therefore, we proposed some strategies of improving the software reliability measurement. The central idea of the strategies proposed in this paper is to incorporate the influence factors of complexity of the software under test and the test effectiveness into the software reliability models so as to make the software reliability models more adequate and accurate to the real situation. Results from substantial experiments have shown the rationality and usefulness of the new model.

Keywords Software reliability models, Software complexity, Test effectiveness, Fault injection, Reliability measurements

1 引言

软件可靠性理论是工业界用来预测软件失效可能性的重要方法之一[1]。软件可靠性度量模型应该顾及到软件的诸多重要特性,如软件的复杂性、测试用例的有效性和软件的运行环境等,否则就会使得评估出来的软件可靠性不够精确。

虽然硬件和软件之间关于可靠性理论和测试的基本概念是类似的[2,3],但是两者之间仍然存在着较大的差别。软件可靠性的预测要比硬件可靠性的预测困难得多,这是因为在软件中,要对软件做充分的评估远不只是简单地给被评估系统赋一个可靠性数值,它是要比把时间和操作规程简单地转换成某些人们可以用数学公式来处理的事务要复杂得多。我们知道,在软件可靠性研究中除了使用硬件系统中常用的方法外,还至少需要处理3个问题:软件的复杂性、测试的有效性和整个软件的运行环境。因此,为了对软件可靠性作更充分更精确的评估,软件可靠性模型及目前所使用的评估方法亟待进一步的改进。

2 基于日历时间的软件可靠性和排错率

通常,传统软件可靠性是基于日历时间、检错率和排错率来进行评估的[4,5]。这些模型中大部分都使用指数递减型的排错率,这是因为指数递减型的排错率能够预测在程序正常时有一段时间几乎很难找到的错误,因而受到众多研究者的亲睐。在本研究中,假设所有检测到的错误都得到正确的修正,并且未因修改而引入新的错误。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,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,于是有:

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

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

$$E_r(t)=\frac{dE_d(t)}{dt} \tag{2}$$

事实上,$E_r(t)$是排错率,可以视为被测软件的失效率。

到稿日期:2008-06-24 本文受国家自然科学基金项目(60473033)资助。
朱经纷(1983-),男,硕士生,研究方向为可信计算、软件测试,E-mail:zjf1983@163.com;徐拾义(1941-),男,教授,博士生导师,CCF高级会员,研究方向为容错计算、可信计算、VLSI设计与测试、软件测试。

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：

$$\sum_{i=1}^{k}[n_i-E_0(1-e^{-\beta t_i})]^2=0 \tag{3}$$

上式两边分别对 E_0 和 β 求偏导后可以得到如下两式：

$$E_0=\sum_{i=1}^{k}n_i(1-e^{-\beta t_i})/\sum_{i=1}^{k}(1-e^{-\beta t_i})^2 \tag{4}$$

$$E_0=\sum_{i=1}^{k}n_i t_i e^{-\beta t_i}/\sum_{i=1}^{k}(1-e^{-\beta t_i})t_i e^{-\beta t_i} \tag{5}$$

由式 (4) 和式 (5)，我们可以计算出 E_0 和 β 的值，同时也可以计算出相应的 $MTTF$ 值，如下所示：

$$MTTF=\frac{e^{\beta t}}{E_0\beta} \tag{6}$$

在给定排错率 E_r 的条件下可以得到相应的软件发布时间为：

$$t=[\ln E_0\beta-\ln Er(t)]/\beta \tag{7}$$

例 1　下表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软件规模为 25,324 行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 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(1)和式(2)所描述的可靠性模型完全适用于被测程序。因而可以得到被测程序的可靠性值计算如下：即，由式(4)和式(5)可以估算出 E_0 和 β 的值，$E_0=285$，$\beta=0.0981$。

表 1 测试过程日志记录

测试天数	检测到的错误数	检测到的累积错误数	测试天数	检测到的错误数	检测到的累积错误数
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 (1) 和式 (2) 的模型中，则可以得到测试期间可能检测到的累积错误数为：

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

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，我们可以大概地估算出该程序在经过 20 天的测试后所剩余的错误数约为 $E_{rem}=285e^{-0.0981\times 20}\approx 40$。如果要求在软件发布前排错率 $E_r(t)$ 低于 0.1/天，那么根据式 (7) 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，排错率才可能下降到 0.1/天。

传统上使用日历测试时间来对被测软件进行可靠性度量，然而测试时间仅仅是反映测试质量的一个因素，传统软件可靠性模型却仅仅建立在测试时间上，而忽略了对软件可靠性具有十分重要影响的软件复杂性和测试用例有效性和测试覆盖率等因素。将可靠性的估算方法仅仅囿于使用类似式(1)、式 (2)、式 (6) 或 式(7) 那样的仅与测试时间相关的公式来度量，实际上所得到的软件可靠性往往是不可靠的结果，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能够体现影响到软件可靠性评估的软件复杂性和测试用例的有效性等重要因素。

3　软件可靠性中的软件复杂性

当前使用的软件可靠性模型由于其在实际评估中并不精确而使得它不能得到应有的广泛使用。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)软件本身的复杂性。一般来说，软件越复杂就越有可能存在更多的故障。因此应尽可能降低软件复杂度。

(2)测试的有效性和彻底性。可靠性模型假定软件测试是基于操作规程和测试有效性的。一个比较好的可靠性仅仅是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。

(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，它驻留于硬件上并与周围环境紧密联系[6]。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，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[7,8]中找到。

通常我们所说的软件复杂性，是指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[9] 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：

$$Hv=(N_1+N_2)\log_2(n_1+n_2) \tag{8}$$

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 1　软件的变量和运算符复杂性被定义为：

$$\ln Hv \tag{9}$$

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[10]（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如下。

定义 2　软件的循环和分支复杂性被定义为：

$$\ln Ml \tag{10}$$

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[11,12]，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 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 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：

$$If=\sum[(fan-in)\times(fan-out)]^2 \tag{11}$$

其中的 $(fan\text{-}in)\times(fan\text{-}out)$ 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。

定义 5 软件的信息流复杂性被定义为：

$$\ln If \tag{12}$$

根据上面介绍的影响软件复杂性的 3 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 6 软件复杂性模型 λ(c)定义为：

$$\lambda(c)=1-e^{-(\ln Hv+\ln Ml+\ln If)/3\ln LOC} \tag{13}$$

其中，LOC 是被测程序的代码行数。

例 2 我们重新来考察例 1 中的程序，表 2 中所列的参数描述了该被测程序的复杂性特征。

表 2 程序复杂性参数

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

由式 (13)，我们可以计算出程序的复杂性为

$$\lambda(c)=1-e^{-(\ln Hv+\ln Ml+\ln If)/3\ln LOC}=0.609$$

4 软件可靠性模型中测试有效性的影响

我们必须考虑的下一个与软件可靠性有关的因素，即测试的有效性。这是因为目前所使用的可靠性模型大部分是基于测试和排错率来计算的，这就说明测试用例的有效性与被测软件的可靠性有密切的关系。但显然又不可能从一个商用软件来直接计算测试的有效性，即便可以也需要花费大量的时间和精力来查找程序中的所有缺陷并计算出测试的有效性。为了解决这个问题，应采用一些切实可行的故障注入(主要是模拟人为故障)方法，如变异测试[13]。通过计算一个测试集所检测到的故障数与注入故障的总数之比，就可以估算出测试用例的有效性。

定义 7 测试集的有效性 $\lambda(e)$定义为被测软件在给定测试时间 t 内由测试用例所发现的注入故障数与注入故障总数之比，即：

$$\lambda(e)=f_d(t)/f_T \tag{14}$$

其中，$f_d(t)$是测试期间所发现的注入故障数，f_T是注入故障的总数。从式(14)可以看出，测试用例的有效性的值应该在[0,1]之间。λ(e)的值越大测试用例就越有效，即测试用例检测到的注入故障数越多。一般总是认为，一个给定测试集在测试期间检测到的注入故障数越多，那么同样的测试集也能检测到更多的非注入故障。从而就可以得到更好的测试有效性。

为了具有可比性，我们仍然以例 1 来阐明上述思想。本例注入的故障只考虑 4 种变异，即参数丢失、参数替换、运算符丢失和运算符替换。此外，一经检测到故障就立即更正过来。对被测程序求测试的有效性步骤如下：

(1)逐个注入故障到被测程序中，使得每次仅注入一个故障。

(2)测试过程中当检测出一个故障后立即记录下来，然后将程序恢复成原来的状态以便接下来可以再注入一个新的故障，启动新的一轮测试。

(3)当所有测试用例都用完了还不能检测到所注入的故障时，就认为这次测试以失败告终，即所注入的故障不能被检测到。同样也将程序恢复成原来的状态，注入一个新的故障同时启动新的一轮测试。

(4)重复步骤(2)和(3)，直到所有要注入的故障都处理了，结束算法。

例 3 对例 1 中的程序使用两个不同的测试集来考察它们的测试有效性，如表 3 所列。注入故障总数为 148 个，第一个测试集能够检测出 117 个注入故障，而第二个测试集仅能够检测出 108 个故障集。因此，可以认为第一个测试集比第二个测试集更有效。

表 3 测试集有效性比较

参数	测试集 1	测试集 2
$f_d(t)$	117	108
f_T	148	148
λ (e)	0.7905	0.7297

5 一个更精确的软件可靠性综合模型

一般而言，软件可靠性模型和潜伏在软件中的残留缺陷(故障)数是密切相关的[14-16]。因此，在排错过程中所得到的失效率的精确性对可靠性模型来说是至关重要的。事实上，如果失效率选择适当，则完全可以产生一个精确的可靠性模型。当然，要做出准确的选择，就应当对失效率、排错率和一些合理的工程假设等方面的历史数据进行充分的研究。

假设我们已经有了应用程序的原始失效率 $\lambda(t)$，那么本节的主要工作则是通过添加上面所讨论的一些新的因素来修改原始失效率 $\lambda(t)$，使得新的可靠性模型更精确和更可靠。

例 4 如图 1 所示，曲线 A 是例 1 中记录的实际测试结果，而曲线 B 所描述的过程则是根据式 (1) 和式 (2) 计算得到的结果。

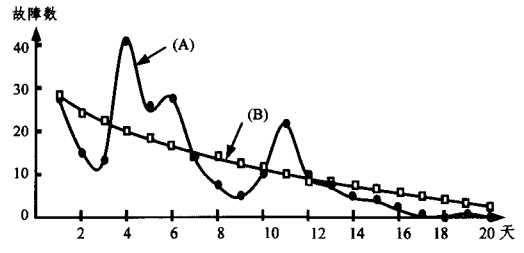

图 1 失效率和排错率

由图 1 可见曲线 B 所描述的理论值与实际测试得到的曲线 A 十分逼近，为了方便我们把曲线 B 近似描述曲线 A，并且在测试到达第 20 天时，曲线 B 几乎成为一条直线，因此，将它近似地处理为一条递减的直线来处理，并将该直线斜率作为程序的失效率。由此可得 $\lambda=0.0014$，而且可以得到经过 20 天测试后程序的可靠性为：

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

然而这个结果是按日历测试时间来计算的，并没有考虑到任何其它方面的因素，通过上述所阐述的，我们可以肯定这个模型的计算结果并不够精确。因此，本节我们引入一个更为精确的可靠性模型。设 $\lambda(t)$ 是原始的失效率，即未考虑软件本身的复杂性和测试有效性，$\lambda(c)$ 和 $\lambda(e)$ 是本文所分别定义的相关因素。现在，我们将影响软件可靠性真实值的这两个因素 $\lambda(c)$ 和 $\lambda(e)$ 加入到可靠性模型中。

此外，我们需要作一些如下约定：

(1)修改后的失效率必须是 $\lambda(t)$，$\lambda(c)$ 和 $\lambda(e)$ 的函数。

(2)如果 $\lambda(e)=1$(表示该测试集的有效性达到最大值)，那么原始失效率 $\lambda(t)$ 已经够精确了而无须修改。

(3)如果 $\lambda(e)=0$(表示该测试集未检测到任何注入故障)，那么这个测试集需要重新修改，且复杂性要完整加入到可靠性模型中。

(4)如果 $0<\lambda(e)<1$，那么原始失效率 $\lambda(t)$ 和复杂性 $\lambda(e)$ 都需要添加到软件可靠性模型中去。

在正常的情况下，测试的有效性 $\lambda(e)$ 不可能取 0 和 1 这两个极端值。因此，根据以上约定的规则及所有的实验结果，我们直接引入一个更精确的可靠性模型，如下：

$$R^*(t)=e^{-\{k_1\lambda(t)+k_2\lambda(c)[1-\lambda(e)]\}} \tag{15}$$

其中，k_1 和 k_2 的值依赖于具体的实验结果和用户需求，通常 $k_1+k_2=1$。在下一节我们将通过实验结果来验证该模型的合理性。

6 实验结果

本节中我们通过一些实验结果来来证明式(15)所提出的可靠性模型确实比传统可靠性模型更精确。在本实验中仍然以例 1 中的程序为例，在第 5 节我们已经知道经过 20 天的测试后，软件的可靠性为 $R(t)=0.972$，原始失效率 $\lambda(t)=0.0014$。在第 3节和第 4 节中又定义了两个因素 $\lambda(c)$ 和 $\lambda(e)$，这样可以直接用式(15)来计算可靠性值。

设 $k_1=2/3$，$k_2=1/3$，使用表 3 中所列的两个测试集的测试有效性来计算可靠性，如表 4 所列。其中测试时间为 20 天，并且原始失效率 $\lambda(t)$ 也是在测试了 20 天后计算得到的。

表 4 修改后的可靠性

测试集	原始失效率/可靠性	复杂性 λ(c)	测试有效性 λ(e)	新的失效率/可靠性
测试集 1	0.0014 / 0.972	0.609	0.7905	0.044 / 0.956
测试集 2	0.0014 / 0.972	0.609	0.7297	0.055 / 0.946

由表 4 可知，在考虑了软件的复杂性和测试有效性以后，该软件的可靠性将从原来的可靠性值 $R(t)=0.972$ 分别下降为 $R_1(t)=0.956$ 以及 $R_2(t)=0.946$。而 $R_1(t)$ 大于 $R_2(t)$ 的原因就是因为测试集 1 的有效性高于测试集 2 的有效性。

结束语 目前所使用的软件可靠性模型的一个主要问题是我们不能对这些模型是否反映软件真实情况进行量化。这是因为，与硬件可靠性不同，目前所使用的软件可靠性模型是基于软件在实际运行时所收集到的一些不全面的信息而计算的。软件常常运行于多用户的复杂环境中，一些可用的资源很有可能突然消失使得运行环境极为复杂。因此，软件可靠性模型应该能够体现出这些情形。不精确的可靠性模型将阻碍可靠性度量的普遍应用。

虽然本文仅是对软件可靠性模型这一领域作了一定的探索，但本文所提出的改进软件可靠性模型的方法对进一步研究软件系统可信性是极为重要的。本文试图将影响软件可靠性的因素，如软件复杂性、测试有效性等引入到可靠性模型中以提高软件可靠性评估的精确性。事实上，我们在现实中会经常遇到这样的情况，这些因素对软件可靠性的影响很大。

参 考 文 献

[1] Whittaker J A, Voas J. Toward a more reliable theory of Software Reliability [J]. Computer, 2000, 33(12): 36-43

[2] Lyu M. Software fault tolerance [M]. New York: John Wiley & Sons, 1995

[3] Lyu M. Handbook of software reliability engineering [M]. New York: IEEE CS Press, Los Alamitos, Calif., and McGraw-Hill, 1996

[4] Musa J D, et al. Software reliability measurement prediction application[M]. New York: McGraw-Hill, 1987

[5] Musa J D, et al. Software reliability engineering [M]. New York: John Wiley & Sons, 1998

[6] Voas J. PIE: A dynamic failure - based technique [J]. IEEE Trans. Software Eng., 1992, 18(8): 717-727

[7] Xu S. Built-in-self-test for software [C]// Xi'an: IEEE Proc. on ATS'03. 2003: 220-224

[8] Xu S. Reconsideration of software reliability measurements [J]// Beijing: IEEE Proc. on ATS'07. 2007: 159-165

[9] Halstead M H. Element of software science [M]. New York: Elsevier, 1977

[10] McCabe T J. A complexity measure [J]. IEEE Trans. on Software Eng., 1976, 2(4): 308-320

[11] Henry S, Kafura D. Software structure metrics based on information flow [J]. IEEE Trans. on Software Eng., 1981, 7(5): 510-518

[12] Brooks F P. The mythical man-month: Essays on Software Engineering [M]. MA: Addison-Wesley, 1975

[13] 徐拾义. 可信计算系统设计与分析[M]. 北京：清华大学出版社，2006

[14] Shooman M L. A micro software reliability model for prediction and test apportionment [C]// Proceeding International Symposium on software Reliability Engineering. New York: IEEE Computer Society Press, 1991: 52-59

[15] Shooman M L. Software reliability models for use during proposal and early design stages [C]// Proceedings ISSRE '99, Symposium on Software Reliability Engineering. New York: IEEE Computer Society Press, 1999

[16] Shooman M L. Reliability of computer systems and networks [M]. New York: John Wiley & Sons, 2002