第 11 卷 第 1 期
1985 年 1 月

自 动 化 学 报
ACTA AUTOMATICA SINICA

Vol. 11, No. 1
Jan., 1985

软件可靠性预测方法的进展

黄 锡 滋
(成都电讯工程学院)

摘 要

软件可靠性是在 70 年代由软件工程与可靠性学科相结合产生的一个新分支，是一个很有发展前途的新领域. 现在，可靠性学科的一些基本原理和方法. 已逐渐渗透到软件生存期的设计、测试和维护等各个阶段，尤其是用于对软件产品的可靠性作出定量的评估和预测. 本文侧重从可靠性学科的角度，对它的发展状况作了综合性的介绍，希望对促进这一新分支在我国的发展有所裨益.

一、概 述

在计算机系统的开发中，软件的开发占据着重要的位置. 国外的资料表明，60 年代后期软件开发费用就已经超过了硬件开发费用. 实践表明，即使经过精心设计和调试的大型软件系统仍然存在着各种错误，一旦遇到适当条件，这些潜在的错误将导致系统的失效和瘫痪. 美国空军范登堡中心六十年代末曾多次发生过导弹试验失败的事故，事后检查几乎都是由于计算机软件的错误造成的. 然而对于软件可靠性的重要性，在相当长的时间内，科技界和工程界都缺乏足够的认识. 这种状况终于导致了“软件危机”的出现[1]. “软件危机”从反面提高了人们对软件可靠性的认识，从而推动了软件工程和软件可靠性学科的发展.

软件可靠性是软件工程和可靠性工程相结合产生的一个新的分支. 现在可靠性学科的一些原理和方法已经在软件设计这个领域发挥了重要作用. 例如容错软件实质上是冗余技术在软件领域中的应用.在软件的调试和试验阶段,怎样在发现和改正错误的同时,利用获得的信息,对软件产品的可靠性进行估计和预测,就更需要利用可靠性学科的知识.

虽然国外对软件可靠性的开发和研究已有十多年的历史,但是实践已清楚地表明,对于任何一个大型复杂的软件系统,要做到交付使用时就完全没有错误是不可能的. 1978 年的一份资料表明[2]，贝尔实验室设计的一个大型自动换接系统，虽经过精心设计和调试,所发生的故障中仍有 20% 是由软件错误造成. 1979 年的一份资料又表明[3]，美国空军的一个指挥和控制系统，软件指令总数为四十五万. 在 1 千小时的运行时间内软件共发生过二十五次关键性失误,单位指令的故障发生率为 5×10^{-8}/hr(CPU). 鉴于大型系统的指令数众多,这种水平的故障发生率显然是一个不容忽视的严重问题.

本文于 1983 年 11 月 28 日收到.

本文侧重从可靠性学科的角度，介绍国外软件可靠性这个新分支的发展状况.

国外对软件可靠性的定量研究工作，于六十年代中期开始. 1964 年贝尔实验室发表了第一篇电子换接系统可靠性方面的文章，其中包括失效数据的直方图[4]. 1967 年 udson 提出了与软件可靠性有关的概率和确定性两种数学模型[5]. 1970 年 Barney 等人表了他们凭经验观测的电子换接系统的软件可靠性数据[6]. 1971 年 Akigama 对十个同的软件组件的失效数据进行了分析[7]. 从 1972 年开始，各种软件可靠性预测数学模纷纷涌现. 这里所指的预测，包括两方面的内容. 其一是预测已调试完毕即将交付使的软件产品达到的可靠性水平；其二是预测试验调试过程中软件产品在将来可能达到可靠性水平. 要进行软件可靠性预测，就必须找到一个度量软件可靠性的标准. 这个题有三种不同的观点和方法，或简称三种流派.

鉴于软件失效都是由于软件中固有的错误造成的，因此一种自然而又直观的方法就以软件中的固有错误数作为衡量软件可靠性的指标. 这一流派以 Mills 和 Basin 为代

另外一些人从七十年代开始，用概率的观点来评价软件可靠性. 从国外最新资料上以看出，概率的观点由于其内在的合理性，已被广泛地接受和采用. 个别人以软件失效理与硬件不同为理由，对这种观点持批评态度. 作者认为他们忽视了事物的另一方面，无论硬件或软件失效的发生都具有随机的性质，而概率正是描述随机事件的唯一科学法. 因此以失效机理不同为由来否定概率的观点是不恰当的. 在采用概率方法度量软可靠性方面又可区分为两派. 一派以 Jelinski 和 Musa 等人为代表，主张沿用硬件可性的方法，将软件可靠性定义为“在规定的时间内、规定的条件下，完成规定功能的概”. 这是一种面向时间的定义方法. 另一派则以 Nelson 为代表，定义软件可靠性为“对一组输入数据，软件能正常运行不发生错误的概率”. 这是一种面向数据的定义方法.

下面介绍三种流派的主要理论.

二、面向时间的数学模型

这类模型由于采用了与硬件可靠性相似的定义，因而可靠度、失效率、平均无故障工时间等指标都可用来评价软件产品的可靠性.

1. Jelinski-Moranda 模型 (J-M 模型)[8]

这是由 Jelinski 和 Moranda 于 1971 年提出的以指数分布为基础的数学模型. 它假：1) 在两个相继失效构成的时间间隔内，软件系统的失效率为常数；2) 其大小正比于统中残存的错误数目；3) 每次失效后总有并仅有一个错误被排除，因此系统的失效率阶梯下降的. 用 N 表示软件最初的错误数目，用 t_i, $i \in (1, \cdots\cdots, N)$ 表示在第 $i-1$ 错误被改正后，软件投入运行至第 i 次失效发生前这个阶段的时间. 按上述假定可得系统失效的密度函数为

$$f(t_i) = \lambda_i e^{-\lambda_i t_i}, \quad i \in (1, \cdots\cdots, N). \tag{1}$$

效率为

$$\lambda_i = \phi[N - (i-1)]. \tag{2}$$

其中 ϕ 是比例常数。

如果 $x_1, x_2, \cdots\cdots, x_n$ 是试验中实际测得的 n 次失效发生的时间间隔，则用最大似然法，可从下式中求出 N 的估计值 $\hat{N}$ 为

$$\sum_{i=1}^{n} \frac{1}{\hat{N}-(i-1)} = \frac{n\sum_{i=1}^{n} x_i}{\hat{N}\sum_{i=1}^{n} x_i - \sum_{i=1}^{n}(i-1)x_i}. \tag{3}$$

$\hat{N}$ 的值求出后，可从下式中求出 ϕ 的估计值 $\hat{\phi}$：

$$\hat{\phi} = \frac{n}{\hat{N}\sum_{i=1}^{n} x_i - \sum_{i=1}^{n}(i-1)x_i}. \tag{4}$$

在试验和调试结束后，系统残余错误的估计值为 $\hat{N}^* = \hat{N} - n$，系统交付使用后的失效率为

$$\lambda = \hat{\phi}\hat{N}^*. \tag{5}$$

系统的可靠度为

$$R(t) = e^{-\hat{\phi}\hat{N}^* t}. \tag{6}$$

系统的平均无故障工作时间为

$$\hat{\mathrm{M}}\mathrm{TBF} = \frac{1}{\hat{\phi}\hat{N}^*}. \tag{7}$$

Jelinski 和 Moranda 曾利用美国海军舰队计算机程序编制中心的海军战术数据系统（NTDS）的软件失效数据，对他们的模型作了验证．NTDS 包括三十八个不同的模块，每个模块都经过了发展、测试、使用等三个阶段．验证所用的失效数据取自一个大型模块的软件反常报告．这个模块在发展阶段共发现了二十六个错误，测试阶段又发现了四个错误，随后在使用阶段又发现了两个错误． NTDS 的数据示于表 1． J-M 根据交付使用前的数据计算的结果示于表 2． 可以看出计算结果与实际数据大致吻合．

与 J-M 模型相近似的还有 Schick-Wolverton 模型[9]、变态 S-W 模型[9]、苏曼模型[10]等．

2. Musa 模型[11]

该模型由 Musa 于 1975 年提出． 他一方面假定系统失效与系统内残存的错误数成正比，同时又假定在调试阶段错误的改正率与失效率成正比，即

$$\frac{dn}{d\tau} = BC\lambda(\tau). \tag{8}$$

其中 n 为系统残存的错误数；τ 为调试时间；B 为错误减少系数，它表示平均发生一次失效所能纠正的错误数；C 为试验压缩系数．

从（8）式可以看出，Musa 模型已抛弃了一次失效改正一个错误的假设，并且用系数 C 描述在试验阶段可以更有效地发现错误．从（8）式可得在调试阶段的失效率为

$$\lambda(\tau) = \frac{1}{T_0}\exp\left\{-\frac{C}{M_0 T_0}\tau\right\}. \tag{9}$$

表 1 NTDS 数据

	错误数 n	错误间隔时间 x_k（日）	累计时间 $Sn = \sum x_k$（日）
设计阶段（核查）	1	9	9
	2	12	21
	3	11	32
	4	4	36
	5	7	43
	6	2	45
	7	5	50
	8	8	58
	9	5	63
	10	7	70
	11	1	71
	12	6	77
	13	1	78
	14	9	87
	15	4	91
	16	1	92
	17	3	95
	18	3	98
	19	6	104
	20	1	105
	21	11	116
	22	33	149
	23	7	156
	24	91	247
	25	2	249
	26	1	250
测试阶段	27	87	337
	28	47	384
	29	12	396
	30	9	405
	31	135	540
使用阶段	32		798
测试阶段	33		814
	34		849

表 2 J-M 模型预测结果比较

错误数（k）	实际失效时间 x_k（日）	预测失效时间 $\hat{x}_k$（日）（按 26 个数据计算）
1	9	4.7
2	12	4.8
3	11	5.0
4	4	5.2
5	7	5.4
6	2	5.6
7	5	5.8
8	8	6.0
9	5	6.3
10	7	6.6
11	1	6.9
12	6	7.2
13	1	7.6
14	9	8.0
15	4	8.5
16	1	9.0
17	3	9.6
18	3	10.3
19	6	11.1
20	1	12.0
21	11	13.0
22	33	14.3
23	7	15.9
24	91	17.8
25	2	20.3
26	1	23.5
27	87	28.1
28	47	34.8
29	12	45.6
30	9	66.4
31	135	121.7
总的错误数 (34)		$\hat{N} = 31.2$
失效率比例系数		$\hat{\phi} = 0.00685$
残留错误数 (8)		$\hat{N} - 26 = 5.2$

注． 本表摘录自 IEEE Trans. Reliab. Vol. R-28, p 210.

其中 T_0 是系统开始测试时的 MTBF; $M_0 = N_0/B$;

N_0 是系统原有的错误数.

系统交付使用后的可靠度为

$$R(\tau,\tau') = \exp\{-\tau'\lambda(\tau)\}. \tag{10}$$

这个模型的另一个特点是它区分了两种时间的概念．一种是执行时间，它被定义为

机器中心处理机运行的时间．（8）到（10）式采用的就是执行时间．另一种是日历时间，它以周(或日)作为计时单元．由于测试的过程受制于三种资源——故障识别人员、错误改正人员和程序留机时间，Musa 建立了执行时间和日历时间的联系：

$$\Delta t = \int_{\tau_2}^{\tau_1} \max\left(\frac{dt_I}{d\tau} \cdot \frac{dt_F}{d\tau} \cdot \frac{dt_C}{d\tau}\right) d\tau. \tag{11}$$

其中 t_I 是故障识别时间；t_F 是错误改正时间；t_C 是程序留机时间． Musa 总结了各类资源消耗和执行时间的关系：

$$X = \theta\Delta\tau + \mu M_0\left[\exp\left\{-\frac{C}{M_0T_0}\tau_1\right\} - \exp\left\{-\frac{C}{M_0T_0}\tau_2\right\}\right]. \tag{12}$$

其中 X 是资源要求；θ 是资源对执行时间的平均消耗率；μ 是每次失效的资源平均消耗率．用最大似然法可求出表达式(9)中各参数的估计值

$$\hat{T}_0 = \frac{C}{n}\left\{\sum_{i=1}^{n}\tau_i' - \frac{1}{\hat{M}_0}\sum_{i=1}^{n}(i-1)\tau_i'\right\}, \tag{13}$$

$$\sum_{i=1}^{n}\frac{1}{\hat{M}_0-(i-1)} = \frac{C}{\hat{M}_0\hat{T}_0}\sum_{i=1}^{n}\tau_i'. \tag{14}$$

其中 $\tau_1', \tau_2', \cdots\cdots, \tau_n'$ 表示测试中发现的 n 次故障间隔时间．

Musa 于 1979 年发表了他对十六个软件系统进行检验的结果[12]，见表 3．

表 3　试验的软件系统特征

系统名称	目标程序指令数	程序员数	失效样本大小	系统性质
1	21700	9	136	实时指挥和控制
2	27700	5	54	同上
3	23400	6	38	同上
4	33500	7	53	同上
5	2445000	275	831	实时商用系统
6	5700	8	73	商用(子系统)
14C	数十万	110	36	实时
17	61900	8	38	军用
27	126100	8	41	同上
40	180000	8	101	同上
SS1A	数十万	未知	112	操作系统
SS1B	数十万	未知	375	同上
SS1C	数十万	未知	277	同上
SS2	数十万	未知	192	时间分配系统
SS3	数十万	未知	278	文字处理系统
SS4	数十万	未知	196	操作系统

实验中，Musa 仔细地对模型的假设条件作了验证． 并对 5 号系统用公式计算出来的故

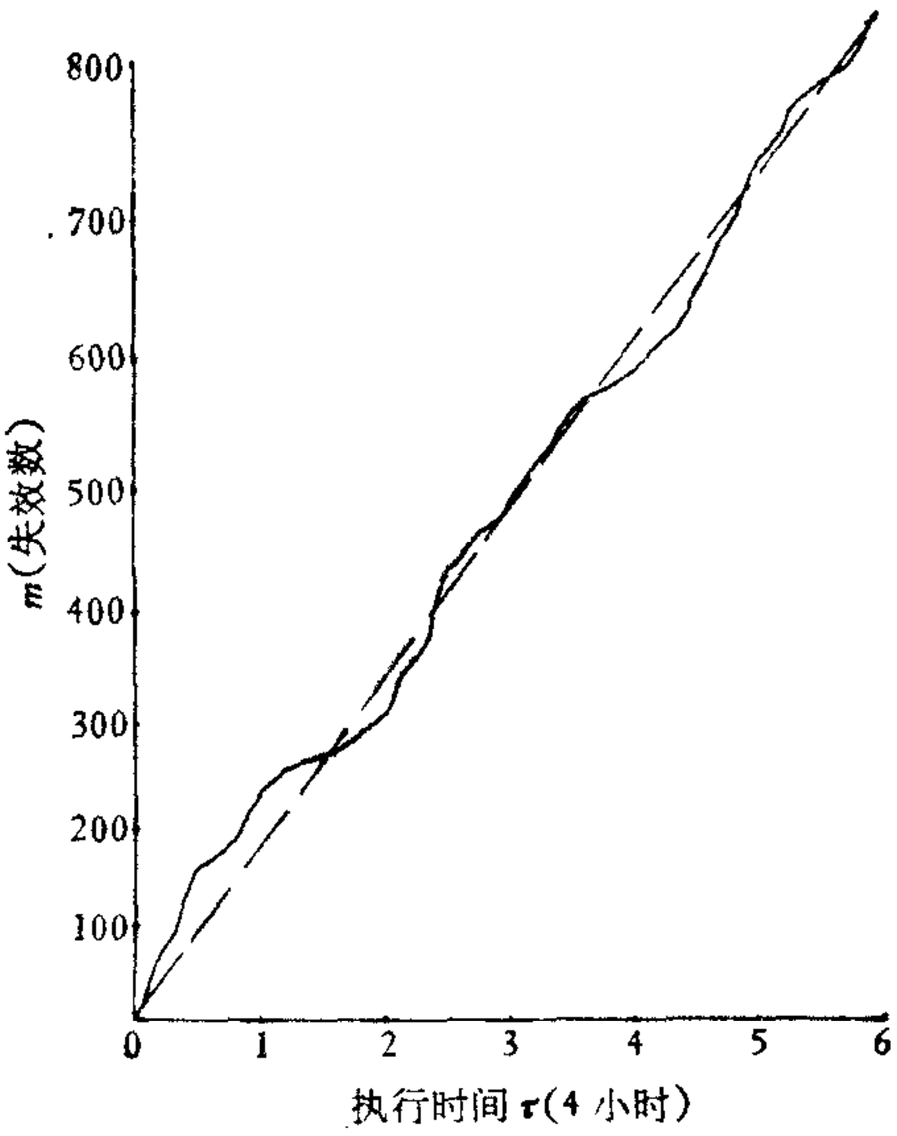

图 1 5 号系统实验结果

障发生次数与实际故障发生的次数进行了比较(见图 1).

图 1 中纵轴的座标已作了相应的变换，以使理论曲线能表示为一条直线. 对 5 号系统进行测试的过程中，在发现了 288 个错误后对原来的程序大约改动了 21%，这是一种十分极端的情况. 图 1 表明,即使在这种情况下理论模型与实际情况仍能很好地吻合.

三、面向数据的数学模型

1973 年 TWR 公司的 Nelson 开创了另一条研究软件可靠性的途径[13]. 他把计算机程序 P 看作是对一个可计算函数 F 的说明. 函数 F 定义在全体输入数据的集合 E 上.

$$E = (E_i, i = 1,2,\cdots,N). \tag{15}$$

其中 E_i 表示第 i 组输入数据,每个 E_i 必有一个函数值 $F(E_i)$ 与它对应. 从计算机运行角度看，每个 E_i 又与"一次运行"有一一对应的关系. 用 $F'(E_i)$ 表示程序 P 运行后的输出，由于程序的不完善性，它不可能完全准确地表达 F，但是在绝大多数情况下，$F'(E_i)$ 和 $F(E_i)$ 之差在可接受范围内,即

$$|F'(E_i) - F(E_i)| \leqslant \Delta i. \tag{16}$$

其中 Δi 是可接受的误差上限.

然而对 E 的某个子集 E_e，程序 P 的运行将产生某种不可接受的输出. 例如程序运行中过早中断,没有结果;程序陷入死循环,不能结束; 程序的输出超过了可接受的限度等,即

$$|F'(E_i) - F(E_i)| > \Delta i. \tag{17}$$

1) 设 N_e 表示属于集合 E_e 的 E_i 数目,则程序运行一次失败的概率为

$$P = N_e/N. \tag{18}$$

其中N是全部输入数据的数目.

因此程序运行一次无故障的概率为

$$R = 1 - P = 1 - N_e/N. \tag{19}$$

程序运行 n 次无故障的概率为

$$R(n) = (1 - N_e/N)^n. \tag{20}$$

以上结果是建立在计算数据的选择服从均匀分布的基础上的. 实际上各类数据被选用的概率并不相同.

2) 用 P_i 表示 E_i 被选用的概率,并定义

$$y_i = \begin{cases} 1, & \text{如 } E_i \in E_e, \\ 0, & \text{如 } E_i \notin E_e, \end{cases} \tag{21}$$

则程序运行一次失效的概率为

$$P = \sum_{i=1}^{n} P_i y_i. \tag{22}$$

程序运行一次不出故障的概率为

$$R = 1 - P = \sum_{i=1}^{n} P_i(1 - y_i). \tag{23}$$

程序运行 n 次不出故障的概率为

$$R(n) = \left[\sum_{i=1}^{n} P_i(1 - y_i)\right]^n. \tag{24}$$

Nelson 把软件的可靠性定义为程序在 n 次运行中不出故障的概率,可用式(20),(24)来表示. 进一步考查计算机的运行情况还可看出, 在不同的运行次数中, E_i 被选用的概率并不相同. 这时第 j 次运行出故障的概率为

$$P_j = \sum_{i=1}^{n} P_{ij} y_i. \tag{25}$$

其中 P_{ij} 是在第 j 次运行中数据 E_i 被选用的概率. 这时软件系统的可靠性被定义为

$$R(n) = \prod_{j=1}^{n} (1 - P_j) = \prod_{j=1}^{n} \left(1 - \sum_{i=1}^{n} P_{ij} y_i\right). \tag{26}$$

(26)式可以改写成指数形式:

$$R(n) = e^{\sum_{j=1}^{n} \ln (1-P_j)}. \tag{27}$$

如果 $P_j \ll 1$, 则得

$$R(n) = e^{-\sum_{j=1}^{n} P_j}. \tag{28}$$

如果对所有的 j, $P_j \equiv P$, 则

$$R(n) = e^{-P_n}. \tag{29}$$

$R(n)$ 还可以用执行时间来表示. 用 Δt_j 表示第 j 次运行的执行时间,则程序运行 i 次的累积执行时间为

$$t_j = \sum_{i=1}^{j} \Delta t_i. \tag{30}$$

令 $h(t_j) = -\ln(1 - P_j)/\Delta t_j$, 则

$$R(n) = e^{-\sum_{j=1}^{n} \Delta t_j h(t_j)} \tag{31}$$

如果 n 很大, Δt_j 很小,则上式可表示为

$$R(n) = \exp\left\{-\int_0^t h(s)ds\right\}. \tag{32}$$

这样就得到了与硬件中的可靠度相对应的表达式. $h(s)$ 与失效率函数相对应.

3) 为评价程序的可靠性,可抽取 n 组数据作试验, 如果 $\hat{n}e$ 次失败,则得出运行一次的可靠性估计值为

$$\hat{R} = 1 - \hat{n}e/n. \tag{33}$$

如果数据是按照概率分布 P_i 抽样的,当 $P_i \ll 1$ 时,上式给出的是 R 的无偏估计.

四、面向"错误数"的数学模型

本文第二部份提到的 J-M 模型、S-W 模型、Musa 模型实际上都给出了软件中错误数目的估计方法, 这些方法都是在失效服从某种特定分布的假设下给出的. Mills 和 Basin 给出了更直接的估计软件中错误数目的方法[9].

Mills 首先于 1970 年提出了估计软件中错误的超几何分布模型. 他的方法是令 N 代表软件中固有的错误总数, n_1 是在测试前人为加入的错误数, r 为测试中发现的错误数, 其中 k 个是人为错误的概率为

$$q_k(N) = \binom{n_1}{k}\binom{N-n_1}{r-k}\Big/\binom{N}{r}. \tag{34}$$

由此得出最大似然估计为

$$\hat{N} = \mathrm{int}[n_1 r/k]. \tag{35}$$

Basin 于 1973 年指出上两式是不正确的. 按照 Mills 的方法,对 N 的最大似然估计应为

$$N_0^* = \mathrm{int}[n_1(r-k)/k]. \tag{36}$$

Basin 提出了两步测试程序,第一次一个程序员测出了 n_1 个错误, 第二个程序员再独立地测试出了 r 个错误. 他认为 (34) 式实际上表示的是在 r 个错误中的 k 个是属于第一次已发现的错误的概率,而 (35) 式给出的是这种两步测试法得出的 N 的最大似然估计.

Mills 和 Basin 的方法非常直观,代表了软件可靠性评价方法的一个流派. 还有其它一些软件可靠性专著也对这种方法作了介绍,但如何准确地给出错误数的区间估计,这一问题仍未得到解决, 使这个方法的应用范围受到限制. 最近成都电讯工程学院的有关研究进一步解决了这个问题. 以程序中引人人为错误的观测方法为例,用 N_0 表示程序中固有的错误的真值,用 N_1 表示引人的人为错误数,用 n 表示测试中发现的错误数,用 r 表示测试中发现的人为错误数,对于给定的置信度 θ, 可以得出

$$P[N_0 \leqslant N_0^+(r,\theta,n,N_1)] \leqslant \theta. \tag{37}$$

式(37)给出了 N_0 的单侧区间估计，其中 $N_0^+(r,\theta,n,N_1)$ 是固有错误数的置信上界，它是 r, θ, n, N_1 的函数，因而也是一个随机变数.

对于给定的 α_1, α_2, 其中 $0<\alpha_1<1$, $0<\alpha_2<1$, $\alpha_2<\alpha_1$. 以 $\alpha_1-\alpha_2$ 作为置信度，可以得出

$$P(N_0^-(r,\alpha_2,n,N_1) \leqslant N_0 \leqslant N_0^+(r,\alpha_1,n,N_1) \leqslant \alpha_1-\alpha_2. \tag{38}$$

式(38)给出的是双侧区间估计，其中 $N_0^+(r,\alpha_1,n,N_1)$ 是固有错误数的置信上界；$N_0^-(r,\alpha_2,n,N_1)$ 是固有错误数的置信下界. $N_0^-(r,\alpha_2,n,N_1)$, $N_0^+(r,\alpha_1,n,N_1)$ 都是随机变数.

文[14]给出了 $N_0^+(r,\alpha_1,n,N_1)$, $N_0^-(r,\alpha_2,n,N_1)$ 的计算公式，对于一些常用的典型数据可直接从文[14]中查出 N_0^+ 和 N_0^- 的数值.

五、可靠性增长预测

在实验和调试阶段往往需要估计使产品达到预定的可靠性水平所需要的时间，以及相应的资源和费用，这就需要进行可靠性增长的预测. 本文第二部份所述各种模型大都能用来达到这个目的. 例如用 J-M 模型的(3)(4)式确定了常数 $\hat{N}$ 和 $\hat{\phi}$ 后，再利用(7)式就可以推出为使软件残存错误减少到给定的水平所需要的调试时间. 使用 Musa 模型则更方便. 从(9)式立刻可以得出测试阶段系统的 MTBF 和测试时间的关系：

$$T = T_0 \exp\left(\frac{C}{M_0 T_0}\tau\right). \tag{39}$$

除此之外还有如下预测方法：

1. Lioyd 和 Lipow 模型[15]

设测试过程共分为 N 个阶段，每个阶段都进行一系列测试，直至发现某个错误并对程序作了修正后结束，然后开始另一阶段的测试，则可靠性增长过程可用下式描述：

$$R(k) = R(n) - A/k. \tag{40}$$

其中 $R(k)$ 是在 k 阶段的实际可靠性；$R(n)$ 是当 R 增长后系统的可靠性极限值，A 是某给定的常数.

按照实际试验所得出的数据，用最小二乘法可得

$$-\sum_{k=1}^{n}\left(\frac{S(k)}{n(k)} - R(n) + \frac{A}{k}\right) = 0, \tag{41}$$

$$\sum_{k=1}^{n}\left(\frac{S(k)}{n(k)} - R(n) + \frac{A}{k}\right)\frac{1}{k} = 0. \tag{42}$$

其中 $n(k)$ 是第 k 阶段测试次数；$S(k)$ 是第 k 阶段测试成功的次数.

2. 威布尔增长模型[9]

Wagoner 在 1973 年收集了某个包括 4000 个 Fortran 语句的程序调试数据，用来研究 J-M 模型、Musa 模型和 S-W 模型的可靠性增长过程. 同时他又用威布尔分布曲线来拟合实际数据，并给出了按威布尔分布的增长曲线、S-W 增长曲线与实际增长曲线的比较结果(见图 2). 分析表明，用威布尔分布能较好地描述可靠性的实际增长过程. 威布尔

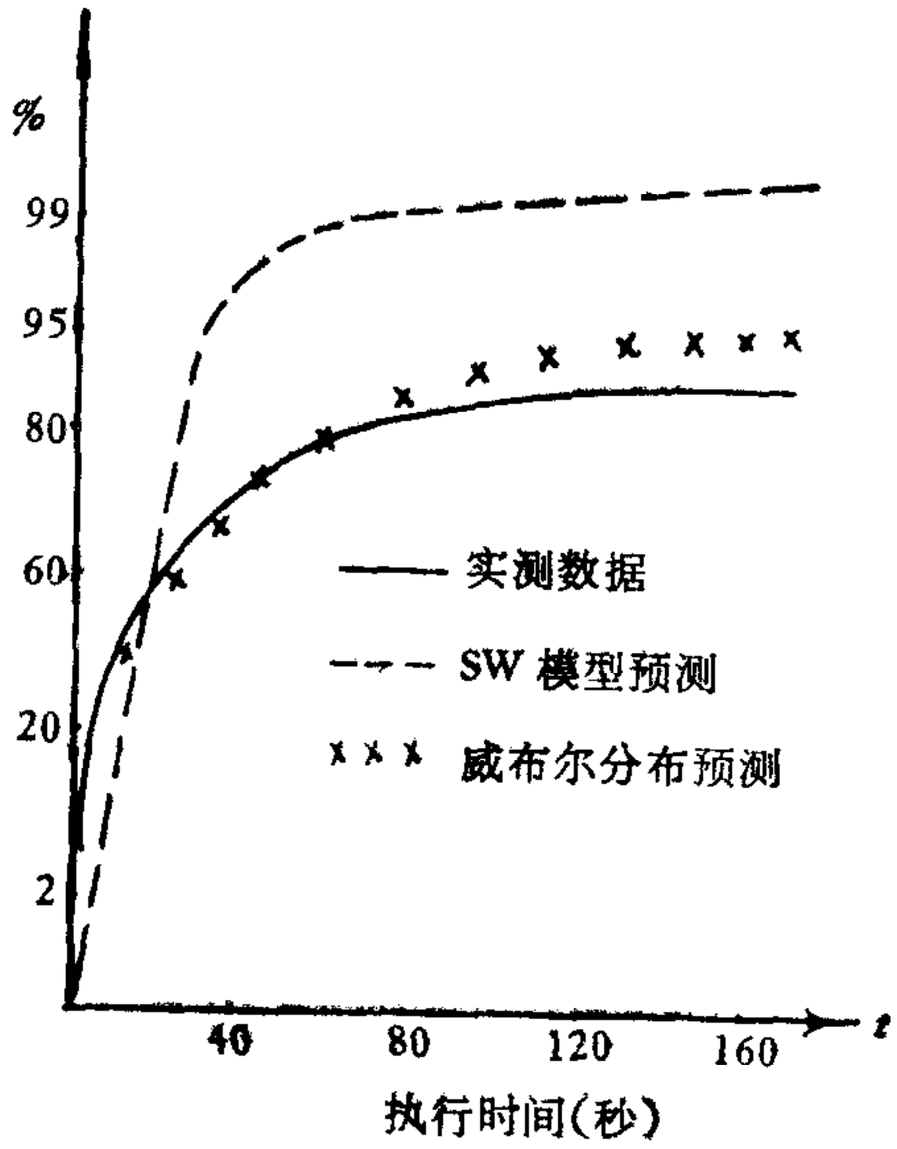

图 2 可靠性增长曲线

分布的参数是用最小二乘法得出的.

六、结 束 语

本文初步概括了迄今为止软件可靠性预测的主要成果. 可以看出，虽然软件可靠性研究在七十年代有了很大的发展,但从整体上看,尤其是与硬件可靠性技术相比，它还不够成熟. 可以预言，软件可靠性的研究在八十年代必将成为软件工程和可靠性工程所共同关心的重要课题. 一些新的模型还会陆续涌现,现有的各种模型也将进一步得到完善. 这些工作又必然会集中到一个核心目标上,即寻找一些经过验证、较为简单、较为正确,并适于工程运用的数学模型. 现行各种模型所用数学工具似有日趋繁复之势，但硬件可靠性发展过程已表明,工程界更乐于采用较简捷实用的方法. 为了达到这个目的,笔者认为应充分注意下面两方面的工作:

首先需要研究各种方法之间的内在联系,以取各家之长,逐步形成一些更完善的预测模型. 在这方面，Nelson 已做了一些工作. 他已着手把自己的模型和面向时间的模型联系起来进行考查. 今后应继续开展这方面的工作.

其次，要结合实际工程大力开发模型的实验验证工作. 模型的正确性实用性归根结底要由实验结果来判断,因为各个模型都是建立在一定假设基础上,这些假设的近似程度会直接影响到模型的正确性. 只有在充分验证的基础上才能进一步明确模型应用的范围和条件. 以面向时间的模型为例,这些模型都是先假设软件失效服从某种特定的规律,这些假设的近似性究竟怎样? 又如，这些模型中除 Musa 模型外都未区别执行时间和日历时间,这样区别究竟有无必要? 这些问题都有待经过充分的实验验证来解决.

实验工作在过去开展不够的一个重要原因是缺乏正确完整的软件失效数据. 国外的实验也多是根据“软件问题报告”中的记载事后进行的. 这些报告的内容往往既不完整也不准确. 1979 年 Suker 利用过去的一些不完整数据对一些模型作了比较,结果很难说明

问题[16]．因此软件失效数据的收集和分析十分重要． 有人认为开展实验工作耗资巨大，很不现实．其实这是误解．这里所说的实验，是与软件发展期间的试验、调试阶段同时进行的．数据的收集、处理和分析，程序的试验及调试，是任何软件系统发展的必经阶段，过去由于缺乏认识，因而白白地浪费了许多宝贵的信息．

随着我国社会主义现代化建设的发展，计算机必将更广泛地得到应用，软件可靠性也必将成为我国科技战线上急需解决的重大课题之一．为此，必须建立我国自己的研究队伍，以尽快深入地开展软件可靠性研究工作．要做到这一点并不难，只要软件工程和硬件可靠性工程这两个领域的专业人员共同努力，各自从自己的领域出发，在使本学科的研究工作向纵深发展的同时，也注意横向发展，就能较快地赶上软件可靠性研究的国际先进水平．

参　考　文　献

[1] Kopetz H., *Software Reliability,* Macmillan Press, 1979.

[2] Toy W. N., *Proc. IEEE* Vol. 66, 1978.

[3] Lipow M., *IEEE Trans. on Reliab.* Vol. R-**28**, No. 3, p. 178, 1979.

[4] Haugk, G. *Bell Syst. Tech. J.* Vol. **43**, p2575, 1964.

[5] Hudson G. R., *SDC Rep.* SP-3011 1967.

[6] Barney D. R., *Bell Syst. Tech. J.* Vol. **49**, p2975, 1970.

[7] Akiyama F. *Proc. IFIP. Congr.* Vol. 1 p353, 1971

[8] Freiberger W., Statistical Computer Performance Evaluation New York, Academic, 1972.

[9] Schick G. J., Wolrerton R. W., *IEEE Trans. on Soft. Eng.* Vol. Se-**4**, No. 2, p104, 1978.

[10] Shooman M. L., Proc. 1973 IEEE Symp. Comput. Soft. Reliab.

[11] Musa J. D., *IEEE Trans. on Soft. Eng.* Vol. SE-**1**, No. 3, Sep., p312, 1975.

[12] Musa J. D. *IEEE Trans. on Reliab.* Vol. R-**28**, No. 3, p181, 1979.

[13] AD-A030 798.

[14] Huang Xizi, *Micro. and Reliab.,* Vol. **24**, No. 1, p11, 1984.

[15] AD-A007 773.

[16] Sukert A. N., *IEEE Trans. on Reliab.,* Vol. R-**28**, No. 3, p199, 1979.

A REVIEW ON THE METHODS FOR PREDICTING RELIABILITY OF SOFTWARES

HUANG XIZI

(Chengdu Institute of Radio Engineering)

ABSTRACT

Software reliability is a new branch of both software engineering and reliability science. Reliability theory is especially valuable for predicting reliability of softwares quantitatively. A brief review of this new branch is given in this paper, mainly from the viewpoint of reliability science. The author hopes it would be helpful to the development of this new brach in our country.