软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(3):439−450 [doi: 10.3724/SP.J.1001.2011.03739]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

引入关联缺陷的软件可靠性评估模型*

徐高潮[1,2], 刘新忠[1+], 胡 亮[1,2], 付晓东[1], 董玉双[1]

[1](吉林大学 计算机科学与技术学院,吉林 长春 130012)

[2](吉林大学 符号计算与知识工程教育部重点实验室,吉林 长春 130012)

Software Reliability Assessment Models Incorporating Software Defect Correlation

XU Gao-Chao[1,2], LIU Xin-Zhong[1+], HU Liang[1,2], FU Xiao-Dong[1], DONG Yu-Shuang[1]

[1](College of Computer Science and Technology, Jilin University, Changchun 130012, China)

[2](Symbol Computation and Knowledge Engineer of Ministry of Education, Jilin University, Changchun 130012, China)

+ Corresponding author: E-mail: lxz_jlu@163.com

Xu GC, Liu XZ, Hu L, Fu XD, Dong YS. Software reliability assessment models incorporating software defect correlation. *Journal of Software*, 2011,22(3):439−450. http://www.jos.org.cn/1000-9825/3739.htm

Abstract: Mostly, defect correlation is caused by the defect detected capability masked by other defects. Defect correlation affects the result of the test and makes the results in accurate and distorts the estimated results of software reliability assessment models. From the standpoint of defects themselves, this paper makes a detailed analysis of defect correlation and gives the reasons of the software testing and reliability assessment out of action. By applying the generalized correlation to improve the existing reliability assessment models, this paper puts forward the P-NHPP (phase-nonhomogeneous poisson process) reliability model to make the reliability assessment parameters more coordinated with the actual defect number. Experimental results show that P-NHPP is better and has a fairly accurate prediction capability.

Key words: defect correlation; SRM (software reliability models); NHPP; P-NHPP (phase-nonhomogeneous poisson process); model assessment

摘 要: 关联缺陷的存在很大程度上是由于缺陷的检测能力被其他缺陷所屏蔽,它不仅影响软件测试结果,还扭曲了软件可靠性评估模型的评估结果.从软件缺陷的自身角度来分析软件检测以及可靠性评估失效的原因,对关联缺陷进行了详细分析.为了在软件可靠性评估时获得更符合实际缺陷数量的预测值,将广义关联应用于现有可靠性评估模型的改进上,并提出了 P-NHPP(phase-nonhomogeneous poisson process,简称 P-NHPP)可靠性模型.实验分析表明,P-NHPP 具有较好的拟合效果和预测能力.

关键词: 关联缺陷;软件可靠性模型(SRM);非齐次泊松过程;阶段-非齐次泊松过程(P-NHPP);模型评估

中图法分类号: TP311 **文献标识码**: A

计算机软件技术的发展大大促进了计算机应用领域的发展,尤其是一些关键应用系统的开发和使用(如银行信贷系统、ATM 系统等),软件可靠性已经成为软件产品最重要的属性之一.软件缺陷(defect)普遍存在并具有

* 基金项目: 国家高技术研究发展计划(863)(2007AA01Z123)
收稿时间: 2009-03-30; 修改时间: 2009-07-21; 定稿时间: 2009-10-09

极大危害性,不仅会造成重大的经济损失,甚至危及人身安全.在过去的 30 多年里,大量的软件可靠性模型(software reliability models,简称 SRM)被提出并用来评估软件产品开发过程中的可靠性[1,2].主要包括[3]:软件质量估算模型 COQUALMO(constructive quality model)、缺陷移除矩阵(defect removal matrix,简称 DRE)、Rayleigh 分布模型、指数分布模型和 S 曲线分布模型等.大部分模型都是基于以下一个或多个基本假设[4]:(1) 缺陷都是相互独立的;(2) 缺陷被检测概率相同;(3) 缺陷被检测后立即被修复,修复时间忽略不计;(4) 在缺陷修复过程中不会引入新的缺陷.

SRM 经历了长期的发展并持续保持活跃,在保证和提高软件质量方面起着非常重要的作用.但在实际测试过程中,很多软件缺陷并不是相互独立的,它们存在着某种关联关系,即失效关联(failure correlation).软件关联缺陷是一种普遍存在的现象,"传统可靠性理论失效"的原因之一就是没有考虑到这种关联关系,其存在直接导致其他缺陷检测的效率,是造成软件失效的根源之一[5,6].

目前,仅有少数出版的论文中考虑到了失效关联.复合泊松 SRM[7]主要考虑群集的软件失效,并提出离散复合泊松预测模型来确定随机变量 X_{rem},并将此变量作为软件残留缺陷数量.文献[8]提出 3 种类型的依赖,并结合随机回报网对这些依赖的影响进行了研究.文献[9]在测试回合关联的基础上提出一种二进制 Markov 过程模型,在使用随机测试策略时预测软件的失效数.Bishop 等人[10]认为,软件失效之间的关联关系可以用失效屏蔽效应来解释,如果软件设计适当就可以杜绝失效关联.Katerina 和 Trivedi[4]认为,实际测试过程中软件失效不是相互独立的,提出一种 Markov 更新模型对有失效关联的软件可靠性进行建模.Huang 等人[16]将关联缺陷合并在 SRGM 中,并考虑调试时间滞后带来的影响.景涛等人[5]从软件缺陷角度来考察软件失效关联,提出了关联缺陷的定义,并给出了一种缺陷放回的测试方法来剔除关联缺陷.工程领域内关联缺陷的应用比较稀少,目前只有上海微创软件有限公司的 BMS XP[11]中提供了关联缺陷管理功能,定义了 5 种关联来刻画缺陷之间和缺陷与其他文件内的关联.总的来说,对关联缺陷的研究还处于初步阶段,在工程应用中也没有有效检测方法.

我们认为,关联缺陷的存在很大程度上是由于缺陷的检测能力被其他缺陷所屏蔽,消除这种关联关系能在很大程度上提高 SRM 的评估能力.

本文第 1 节给出关联缺陷的形式化定义.第 2 节和第 3 节在研究 Space 软件缺陷的基础上,对关联缺陷的性质及其影响进行详细分析.第 4 节将广义关联引入对现有软件可靠性模型的改进中,提出 P-NHPP(phase-nonhomogeneous poisson process,简称 P-NHPP)模型,并在第 5 节给出实验数据对 P-NHPP 与 G-O 模型的评估能力进行对比.最后对下一步研究工作进行说明.

1 关联缺陷定义

基本假设:

(1) 缺陷被检测后立即被修复,且修复时间忽略不计;

(2) 缺陷修复不引入新的缺陷.

对一个软件$\mathcal{S}$的检测过程,给定集合[*Defects*,*Cases*],其形式化[12]描述如下:

(1) 软件缺陷 $Defects::=D_1|D_2|\ldots|D_N, N\geqslant 0$,$N$ 为软件缺陷总数;

(2) 测试用例库 $Cases::=C_1|C_2|\ldots|C_M, M\geqslant 0$,$M$ 为软件测试用例总数;

(3) 软件状态可以表示为缺陷的有限集合,用 S 表示,$S:\mathbb{F}Defects$.定义 $\#S=N$ 为测试起始状态,$\#S=0$ 表示软件所有缺陷都被检测,是测试的结束状态;

(4) 软件测试使用 *Cases* 中非重复元素的序列,用 T 表示,$T:iseq\ Cases$.其中回归测试使用部分 *Cases* 元素,用 T_R 表示,$T_R:iseq\ \mathbb{F}Cases$;

(5) 缺陷检测过程满足函数 $TEST=\lambda\ S,R:\mathbb{F}Defects;T:iseq\ Cases|S\times T\bullet R$,根据基本假设 $R\subseteq S$;

(6) 令 $S'=\{D_i\}$,记为 S_i,称为简单状态;令 $S''=\{D_i,D_j\}$,记为 S_{ij},称为序偶状态.

定义 1. 缺陷检测能力(defect detected capability,简称 DDC)描述在状态 $S:\mathbb{F}Defects$ 下,$T:iseq\ Cases$ 对缺

陷 D_i:*Defects* 的检测能力,记为 $DDC_i(S)$.$DDC_i(S)\in\{0,1\}$.其中,1 表示能被检测,0 表示不能被检测.DDC 计算表达式表示为

$$DDC_i(S)=\begin{cases}1, & \exists C_k:Cases\bullet S\times\langle C_k\rangle\mapsto R\in TEST\wedge D_i\in S\wedge D_i\notin R\\ 0, & D_i\notin S\vee(\forall C_k:Cases\bullet S\times\langle C_k\rangle\mapsto R\in TEST\wedge D_i\in R)\end{cases}\tag{1}$$

定义 2. 如果$(D_i,D_j)\in\{D_i,D_j:Defects|\forall C_k:Cases\bullet DDC_i(S_i)\neq DDC_i(S_{ij})\wedge D_i\neq D_j\}$,即缺陷 D_i 的检测能力受 D_j 存在的影响,我们称 D_i,D_j 为一组关联缺陷(defects correlation),记为 $D_i\propto D_j$.其中,符号$\propto$为关联运算符.读作 D_i 关联 D_j 或者 D_j 被 D_i 关联.

定义 3. 如果$D_i\in\{D_i:Defects|\forall D_j:Defects;\forall C_k:Cases\bullet DDC_i(S_i)=DDC_i(S_{ij})\wedge D_i\neq D_j\}$,即 D_i 的检测能力与其他任何缺陷无关,则称 D_i 为独立缺陷(defect independency),记为 $D_i\propto\varepsilon$.

定义 4. 对于 $D_i\propto D_j$,如果 $DDC_i(S_i)=0$,则称 D_i,D_j 为正关联(positive correlation);如果 $DDC_i(S_i)=1$,则称反关联(negative correlation)(文献[5]中将这两种关联分别称为第 2 类关联和第 1 类关联).

定义 5. 令 $A=\{D_j:Defects|D_i\propto D_j\}$,则 A 为缺陷 D_i 的关联缺陷集(defect correlation set),记为 $D_i\propto A$.如果 $D_i\propto\varepsilon$,则有 $D_i\propto\varnothing$.$\#A$ 为关联缺陷集长度,记为 $D_i\|$;令 $B=\{D_i:Defects|D_i\propto D_j\}$,则 B 为缺陷 D_j 的关联影响域(affected domain),记为 $D_j\propto B$.$\#B$ 为影响度,记为$\|D_j$.

定义 6. 令 $A=\{D_j:Defects|D_i\propto D_j\}$,$B=\{D_i:Defects|D_i\propto D_j\}$,则 B 与 A 的关系我们称为广义关联关系(generalized correlation).

定义 7. 缺陷检测效率(defect detected efficiency,简称 DDE)是在所有包含 D_i 的 $S:\mathbb{F}Defects$ 组成的状态集下,用例库 $T:iseq\ Cases$ 对缺陷 D_i 检测能力大小的最大理论估计值,用来评估 DDC 的效率.计算公式为

$$DDE_i=\left(\sum_{S\in S^*}DDC_i(S)\right)/\#S^*\tag{2}$$

其中,$S^*=\{S:\mathbb{F}Defects|D_i\in S\}$.

2 关联缺陷分析

文中采用 Space 软件[13]作为测试对象,它是一个 ADL(array definition language)解释程序,依据 ADL 语法及特定一致性规则,判断读取文件中 ADL 语句的合法性.它包含 6 218 行可执行 C 代码,33 个相关版本,每个版本包含一个缺陷,13 585 个测试用例.我们构建下面实验方法研究简单和序偶状态下的缺陷检测:

(1) 令 $Defects=\{D_1,D_2,\ldots,D_{33}\}$,$D_i$ 对应第 I 版本中的缺陷,令 $Cases=\{D_1,D_2,\ldots,D_{13585}\}$;
(2) foreach D_i in *Defects* {
(3) foreach C_k in *Cases* {
(4) 依据定义 1 计算 $DDC_i(S_i)$;
(5) if($DDC_i(S_i)$==1) {记录结果;break;}
(6) }}
(7) foreach D_i in *Defects* {
(8) foreach D_j in *Defects*\{D_i} {
(9) foreach C_k in *Cases* {
(10) 依据定义 1 计算 $DDC_i(S_{ij})$;
(11) if($DDC_i(S_{ij})$==1) {记录结果;break;}
(12) }
(13) 根据公式(2)计算 DDE_i;
(14) }}

实际测试实验 179 435 次,持续时间 46 天.序偶状态下共检测出关联缺陷 169 组,约占 16%.将缺陷检测结果

整理为关联缺陷统计表,见表 1.其中,√表示 $D_i \propto D_j$ 成立;—表示 $D_i \propto D_j$ 没有定义;0 表示 $D_i \propto D_j$ 不成立.

Table 1 Statistics of defect correlation

表 1 关联缺陷统计表

$DDC_i(S_i)$	$DDC_i(S_i)=0, i \in \{1,2,32\}$;Other value is 1																		
$D_i \propto D_j$	1	2	3	4	5	6	7	…	15	…	27	28	29	30	31	32	33	$D_i\|\|$	*DDE*
1	—	0	√	√	0	√	√		√		√	√	√	√	0	0	√	28	0.88
2	0	—	√	√	√	√	√		√		√	√	√	√	0	0	√	28	0.88
3	0	0	—	0	0	0	0		0		0	0	0	0	0	0	0	0	1.00
4	0	0	0	—	0	0	0		0		0	0	0	0	0	0	0	0	1.00
5	0	0	0	0	—	0	0		0		0	0	0	0	0	0	0	0	1.00
6	0	0	0	0	0	—	0		0		0	0	0	0	0	0	0	0	1.00
7	0	0	0	√	0	√	—		√		0	√	0	√	0	0	0	5	0.84
8	0	0	0	√	0	√	0		0		0	√	0	√	0	0	0	4	0.88
9	0	0	0	√	0	√	0		0		0	√	0	√	0	0	0	4	0.88
10	0	0	0	√	0	√	0		0		0	√	0	√	0	0	0	4	0.88
11	0	0	0	√	0	√	0		0		0	√	0	√	0	0	0	4	0.88
12	0	0	0	√	0	√	0		√		0	√	0	√	√	0	0	6	0.81
13	0	0	0	√	0	√	0		0		0	0	0	√	0	0	0	3	0.91
14	0	0	0	√	0	√	0		0		0	0	0	√	0	0	0	3	0.91
15	0	0	0	√	0	√	0		—		0	0	0	√	0	0	0	3	0.91
16	0	0	0	√	0	√	0		√		0	√	0	√	0	0	0	5	0.84
17	0	0	0	√	0	√	0		√		0	0	0	√	0	0	0	4	0.88
18	0	0	0	√	0	√	0		√		0	√	0	√	0	0	0	5	0.84
19	0	0	0	√	0	√	0		0		0	0	0	√	0	0	0	3	0.91
20	0	0	0	√	0	√	0		√		0	√	0	√	0	0	0	5	0.84
21	0	0	0	√	0	√	0		0		0	√	0	√	0	0	0	4	0.88
22	0	0	0	√	0	√	0		0		0	√	0	√	0	0	0	4	0.88
23	0	0	0	√	0	√	0		√		0	√	0	√	0	0	0	5	0.84
24	0	0	0	√	0	√	0		0		0	0	0	√	0	0	0	3	0.91
25	0	0	0	√	0	√	0		√		0	√	0	√	0	0	0	5	0.84
26	0	0	0	√	0	√	0		0		0	0	0	0	0	0	0	2	0.94
27	0	0	0	√	0	√	0		0		—	√	0	√	0	0	0	4	0.88
28	0	0	0	√	0	√	0		0		0	—	0	0	0	0	0	2	0.94
29	0	0	0	0	0	0	0		0		0	0	—	0	0	0	0	0	1.00
30	0	0	0	√	0	√	0		0		0	0	0	—	0	0	0	2	0.94
31	0	0	0	0	0	0	0		0		0	0	0	0	—	0	0	0	1.00
32	0	0	0	√	0	√	√		√		√	√	0	0	0	—	0	20	0.63
33	0	0	0	√	0	√	0		0		0	√	0	√	0	0	—	4	0.88
$\|\|D_j$	0	0	2	27	1	27	3		11		3	18	2	23	1	0	2	**169**	

分析如下:

(1) 简单状态下,缺陷 $D_1, D_2, D_{32}, DDC_1(S_1)=DDC_2(S_2)=DDC_{32}(S_{32})=0$,即测试用例库无法检测它们;而在序偶状态下,$D_1, D_2, D_{32}$ 关联缺陷集长度都在 20 以上,其 *DDE* 分别为 88%,88%,63%;在其他缺陷存在的情况下极易被检测.说明正关联缺陷的 *DDC* 极易受其关联集缺陷集元素存在的影响,在进行缺陷移除时必须慎重考虑其移除顺序;正关联缺陷的隐蔽性非常高是软件测试最大的障碍,由此产生的测试结果对 SRM 的评估能力也有相当大的影响,必须采取必要的措施减少正关联的产生;

(2) 关联缺陷统计表中共有 6 个独立缺陷:$D_3, D_4, D_5, D_6, D_{29}, D_{31}$,其 *DDC* 不受其他任何缺陷的影响,*DDE* 均为 100%,最易被检测.考察反关联缺陷的关联缺陷集长度,如 $D_{26}\|\|=2, D_8\|\|=4, D_7\|\|=5$.我们可以看出,*DDE* 与其关联缺陷集长度成反比关系.即对反关联缺陷和独立缺陷(关联缺陷集长度为 0)来说,缺陷关联集长度越小,其 *DDE* 值越大,越容易被检测.可以说,只要测试方法恰当,这些关联都是很容易被检测出来的.在 SRM 中,对采集的数据

必须保证非正关联缺陷的检测力,以达到更符合实际的评估结果;

(3) 考察关联缺陷统计表中列的数据,与 D_4 关联的缺陷数量达到 27 个,即$||D_4$=27.如果 D_4 未被剔除,软件内 84%的缺陷将不能被检测.同样还有$||D_6$=27,$||D_{28}$=18,$||D_{30}$=23,关联影响度越大,越要尽早地剔除.可见,关联影响度可以作为缺陷移除策略的重要参考因素.何时对缺陷进行移除,对其他缺陷检测能力都有重要的影响.所以在缺陷移除时,必须对缺陷进行谨慎的分析;

(4) 根据关联缺陷定义 $D_i \propto D_i$ 是不成立的,所以关联运算符∝不满足自反律.从表中数据可以看出,关联运算符∝也不满足交换律,即 $D_i \propto D_j \neq D_j \propto D_i$.如 $D_7 \propto D_4$ 而 $D_4 \propto \varepsilon$,不关联其他任何缺陷;关联运算符∝也没有传递性,即 $D_i \propto D_j \wedge D_j \propto D_k \Rightarrow D_i \propto D_k$ 不成立.如 $D_{32} \propto D_{27} \wedge D_{27} \propto D_{30}$ 而 $D_{32} \propto D_{30}$ 不成立,即它们没有直接的关联关系.但这种间接关联也是需要考虑的问题之一.

3 关联缺陷问题

缺陷在软件中是固有存在的,主要影响到软件开发的设计、测试以及评估方面.但在系统设计、测试及可靠性评估过程中很少考虑到缺陷之间的关系,完全忽略了这些关系对它们造成的影响.从以下几方面进行分析:

(1) 对系统测试的影响.虽然软件测试的目的并不是发现软件中的关联缺陷,但不考虑关联缺陷,我们很难获得完美的检测结果.如 Space 软件中,如果不存在 D_1,D_2,D_{32} 以外的缺陷,则 D_1,D_2,D_{32} 完全不能被用例库检测出来.这种正关联缺陷点必须使用非常规检测方法来进行检测,如文献[5]中的缺陷放回检测技术,虽然这种检测技术还有很大的局限性,但为存在的正关联缺陷检测提供了一种解决方案.另外,缺陷移除策略如果不考虑关联缺陷,将更容易扭曲软件检测结果.以 D_1,D_2,D_{32} 为例,在 Space 软件中,将 D_1,D_2,D_{32} 以外的缺陷全部移除时也能造成它们不可被检测,这种情况在单元测试中尤为突出.适当的移除策略,能够在一定程度上防止关联缺陷过早移除对其检测能力的影响.

(2) 对可靠性评估的影响.系统可靠性评估是在采集系统随机测试数据的基础上对采集的数据以模型进行分析计算,以模型的参数来对系统的可靠性进行评估.现有的可靠性检测模型假设前提之一是缺陷相互独立,但关联缺陷的检测率受其他缺陷影响,一概以常值进行评估很容易导致评估结果失真;另一个假设是缺陷一经检测立即移除,由分析(1)可以看出,这种移除方法很可能获取失真的采样数据,极大地影响软件可靠性评估能力.

(3) 对系统设计的影响.系统测试方案的制定、测试用例的设计和实现与软件开发过程是并行的.在系统设计阶段,对测试用例的设计是关系软件质量的关键.测试用例不仅要达到较好的代码覆盖率,还要在功能逻辑上与用户需求紧密相关.其中,代码覆盖在很大程度上可依赖自动化测试程序中的模块实现,但功能性的测试则需要更多的人为参与.在设计阶段,必须考虑测试对功能的覆盖以及功能之间的逻辑关系,这种关系在缺陷存在时便体现在关联缺陷方面.面向对象技术与组件技术的应用大大增强了功能模块的集成度,但功能模块之间的耦合是不可避免的,功能模块内的缺陷很大程度上提高了集成测试与功能测试的缺陷检测难度.所以,在系统设计阶段必须充分考虑模块相互关联带来的关联缺陷问题,利用适当的设计手段和方法来减低缺陷关联的可能性并提高测试用例的完整性.

在实际应用中,缺陷关联关系的数据是很难采集的.我们的目的并不是发现关联缺陷,而是在软件测试技术与可靠性评估中充分考虑关联缺陷的影响.本文第 4 节从关联缺陷自身的研究出发,将广义关联应用于现有 SRGM 的改进中,提出阶段-非齐次泊松过程模型,以多阶段测试结果进行可靠性参数的迭代估计,以获得更符合实际情况的评估结果.

4 P-NHPP 模型

4.1 NHPP-SRGM

现有的 SRGM 可以分为两类[15]:失效间隔(time between failures,简称 TBF)模型和故障计数(failures count,简称 FC)统计模型,它们从两个角度来对软件质量进行评估.NHPP 类是 SRGM 中应用最广泛的一类,已经成为

软件可靠性工程实践中非常重要的工具,主要包括 G-O 模型[14]、Yamada delayed S-shaped 模型[19]、G-O S-形模型[20]和 Logarithmic Poisson 模型[21].实验结果表明,测试阶段的缺陷数据呈现凹形[22],以上述模型拟合测试阶段的缺陷数据,其中,G-O 模型比其他模型比较具有更好的拟合效果和预测能力[23].因此,本文对测试阶段的失效数据选用 G-O 模型进行拟合对比分析.G-O 模型基本假设:

(1) 缺陷剔除过程是一个 NHPP;

(2) 系统残留缺陷导致软件失效,单位时间内能检测到的缺陷数与软件中残留缺陷数成正比;

(3) 所有缺陷都是独立的,能被检测的概率相同;

(4) 当检测到一个缺陷时,缺陷立刻被剔除,而且不会引入新的缺陷.

根据假设(1)和假设(2),时刻 t 能检测到的缺陷数 $N(t)$服从均值函数(mean value function,简称 MVF)为 $m(t)$的泊松分布,即

$$P(n(t)=k)=\frac{(m(t))^k}{k!}\mathrm{e}^{-m(t)},k=0,1,2,...,$$

其中,

- $n(t)$:$(0,t]$时间段内检测出来的累计缺陷数;
- $m(t)$:时刻 t 累计缺陷数的期望值, $m(t)=\int_0^t\lambda(s)\mathrm{d}s$;
- $\lambda(s)$:失效强度函数$\lambda(s)=r[a-m(t)]$;
- r:缺陷检测率,根据假设(3)为一常量;
- a:缺陷总数,由假设(4),它为一未知常量.

在基本假设下,G-O 为

$$\begin{cases}m'(t)=\lambda(t)\\ m(0)=0\end{cases}\Rightarrow\begin{cases}m'(t)=r(a-m(t))\\ m(0)=0\end{cases}\Rightarrow m(t)=a(1-\mathrm{e}^{-rt}) \tag{3}$$

由公式(3)有 $m(\infty)=a$,即当 t 趋近于无穷时,$m(t)=a$,达到最终可被检测出来的缺陷期望值.假设(4)假定被检测到的缺陷立刻被剔除,这种假设极易在软件内残留正关联缺陷,从而影响模型评估结果;另外,关联缺陷在软件中是普遍存在的,很大一部分缺陷的检测依赖于其他缺陷的检测,G-O 基本假设(3)在分析软件缺陷数据进行软件可靠性评估有很大的局限性.

4.2 基本假设

P-NHPP 模型基本假设:

(1) 缺陷检测过程符合 NHPP;

(2) 系统残留缺陷导致软件失效;

(3) 检测过程是分阶段进行的,每个阶段的缺陷检测过程符合 NHPP,软件缺陷检测过程是阶段 NHPP 的条件迭加;

(4) 后一阶段能检测的缺陷与前面阶段缺陷存在广义关联关系,同一阶段内能被检测的缺陷相互独立的,能够被检测概率为一常数;

(5) $(t,t+\Delta t]$时间段内发现的缺陷期望值与同一阶段内能被检测而未被检测的缺陷正相关,而且与所有残留缺陷所占总缺陷数的比例正相关;

(6) 缺陷修复在每个阶段结束时进行,而且缺陷修复不引入新的缺陷.

4.3 符号及其含义

- k:软件检测阶段序列号;
- a:软件中潜伏的固有缺陷数;
- a_k:阶段 k 固有缺陷数;
- $m(t)$:t 时刻缺陷累计数的期望值;

- $m_k(t)$:t 时刻阶段 k 缺陷累计数的期望值;
- r_k:阶段 k 内缺陷被检测到的概率值;
- $\lambda_k(t)$:t 时刻阶段 k 的失效强度函数,$(t,t+\Delta t]$发现的缺陷数.

4.4 模型建立

由假设(1)、假设(3)

$$a = a_1 + a_2 + ... + a_n = \sum_{k=1}^{n} a_k \ .$$

由假设(4)、假设(5):

$$m(t) = m_1(t) + m_2(t) + ... = \sum_{k=1}^{n} m_k(t) \tag{4}$$

$$\lambda_k(t) = \frac{m_k(t+\Delta t) - m_k(t)}{\Delta t} = r_k \times [a_k - m_k(t)] \times \frac{a - \sum_{i=1}^{k-1} m_i(t)}{a} \tag{5}$$

当 k=1 时,$m_1(t)=a_1(1-\exp(-r_1t))$,与 G-O 模型完全相同.

由公式(4)、公式(5),可得

$$m(t) = \begin{cases} m_1(t) = a_1(1-\exp(-r_1t)), & t=1 \\ \sum_{k=1}^{n} m_k(t), & t>1 \end{cases} \tag{6}$$

4.5 关于假设

P-NHPP 是一种 FC 统计计数模型,对假设分析如下:

① 关于假设(3)、假设(6).将检测过程以阶段来进行区分,这不仅符合软件整合测试,也符合实际软件开发的功能测试和单元测试过程.而现有模型[14,16,19−21,23]显然与实际测试过程不符.现有模型以随机测试策略采集评估所需要的测试数据,很可能造成正关联缺陷无法检测的问题.如 Space 软件中,如果采用阶段检测,并在每阶段结束后集中进行缺陷移除,则能在 4 个阶段 100%地移除全部缺陷;如果采用随机测试方式,并且缺陷在检测后立即移除,有近 10%(40 次完全测试有 3 次没有检测到全部缺陷)的可能会无法检测出 D_1,D_2,D_{32}.显然,在这种假设下采集的测试数据很可能导致模型评估结果失真.而 P-NHPP 充分考虑关联缺陷对测试结果的影响,其采集的数据明显要比现有模型假设更接近于实际值;

② 关于假设(4).充分考虑缺陷在检测过程中相互影响因素,由假设(6),同一阶段的缺陷之间是没有关联关系的.这里假定它们能被检测的概率为一常数,与现有模型假设一致.但后一阶段的测试因为前一阶段缺陷的移除,其失效率必定下降,而缺陷检测率也与上一阶段不同.假设(6)将缺陷间的广义关联关系引入 P-NHPP 模型中,从软件缺陷自身角度考虑来提高模型的评估能力,以计算出与实际观测值更为贴近的评估值;

③ 关于假设(5).由假设(2)可知,软件失效是由于软件残留缺陷所导致.因此假设(5)认为,单位时间内发现的缺陷期望值与软件内残留缺陷占总缺陷数比例正相关.同时,在同一阶段,缺陷的检测符合 NHPP,其缺陷期望值也应与当前阶段内能被检测但未被检测的缺陷正相关;

④ 假设(1)、假设(2)与现有评估模型相同,在此不作进一步分析.

TBF 模型更容易描述软件连续运行中的失效行为,研究趋向于将软件连续失效作为失效依赖来进行研究,但这种关联缺陷与本文研究的关联缺陷有着本质上的区别.将关联缺陷引入 FC 模型的研究比较少,文献[16]将文献[5]定义的两类缺陷引入 FC 模型中,具有较好的预测能力.但其检测结果还在很大程度上依赖于经验性参数 P,这无疑降低了模型结果的拟合效果.软件中,关联缺陷比率因为软件不同有着很大的区别,所以,以文献[16]进行可靠性评估有着很大的片面性.第 5 节我们将以实验对 P-NHPP 的评估效果进行评估.

5 模型评估

5.1 模型评估参数

我们将对 P-NHPP 与 G-O 模型进行实验评估,主要采用下面两种评估参数:

(1) 估计均方误差值(mean square of fitting error,简称 MSE)[18]

$$MSE=\sum_{t=1}^{k}[m(t_i)-m_i]^2\Big/k \tag{7}$$

其中,m_i 为时刻表 t 观测的缺陷数,反映模型的拟合程度.

(2) 预测平均误差(mean error of prediction,简称 MEOP)[17]

$$MEOP=\left(\sum_{i=k}^{n}|n_i-m_i|\right)\Big/(n-k+1) \tag{8}$$

其中,n_i 为时刻 i 观测到的累积缺陷数;m_i 为时刻 i 预测的累积缺陷数,表现模型预测能力.

5.2 数据采集

我们选用 Space 软件的测试数据进行研究.测试用例采用随机选取方式,数据采集策略分两种.Policy *A*:检测到缺陷后立即进行移除,当软件中缺陷全部被移除或者不包含 D_1,D_2,D_{32} 以外的缺陷时停止测试;Policy *B*:采用阶段测试进行数据采集.步骤如下:

(1) 程序初始化:令 $S=\{D_1,D_2,\ldots,D_{33}\}$,$T=\langle C_1,C_2,\ldots,C_{13585}\rangle$,$R=\{\}$;

(2) while (#S>0) {

(3) 从 T 中随机选取 C_i 对软件进行测试;

(4) if (测试不通过) 将检测的缺陷从 S 中移至 R 中;

(5) 将 C_i 移出队列;

(6) if (#T<1) {本阶段结束,修复 R 中的缺陷;重置测试序列 T;}

(7) if (S 中不包含 D_1,D_2,D_{32} 以外的元素) break;

(8) }

为了保证数据的公平,我们以两种策略进行 10 次检测,Policy *A* 平均检测时间 38.6 小时,Policy *B* 平均检测时间 49.4 小时,共 4 个阶段.我们以小时为单位进行数据采集,分别对前 24 小时内所观测到的数据进行统计,然后求采集数据的算术平均值.结果分别以 DS_A 和 DS_B 表示,见表 2 和表 3.其中,t_i 为统计时刻,FC 表示缺陷数,$\sum FC$ 表示累积缺陷数.

Table 2 Dataset of defects while Space software testing by policy *A*

表 2 通过策略 *A* 进行 Space 软件测试缺陷数据集

t_i	FC	$\sum FC$	t_i	FC	$\sum FC$	t_i	FC	$\sum FC$
1	3.6	3.6	9	0.1	13.7	17	0.2	18.9
2	2.1	5.7	10	0.2	13.9	18	0.5	19.4
3	2.6	8.3	11	0.4	14.3	19	0.7	20.1
4	1.7	10	12	0.2	14.5	20	0.2	20.3
5	0.9	10.9	13	1.7	16.2	21	0.6	20.9
6	1.6	12.5	14	1.9	18.1	22	0.2	21.1
7	0.7	13.2	15	0.4	18.5	23	0.3	21.4
8	0.4	13.6	16	0.2	18.7	24	0.2	21.6

Table 3 Dataset of defects while Space software testing by policy *B*

表 3 通过策略 *B* 进行 Space 软件测试缺陷数据集

Phase	t_i	*FC*	∑*FC*	Phase	t_i	*FC*	∑*FC*	Phase	t_i	*FC*	∑*FC*
1	1	2.5	2.5	↑	9	0.8	12.2	↑	17	0.2	16.3
	2	1.9	4.4		10	0.3	12.5		18	0.2	16.5
	3	1.8	6.2		11	1.5	14		19	0	16.5
	4	0.9	7.1	3	12	0.9	14.9	3	20	0.1	16.6
2	5	1.1	8.2		13	0.3	15.2		21	0.2	16.8
	6	0.1	8.3		14	0.3	15.5		22	0.1	16.9
	7	2	10.3	↓	15	0.2	15.7	↓	23	0.2	17.1
	8	1.1	11.4		16	0.4	16.1		24	0.1	17.2

另外,我们选用 Siemens 程序[13,24]的测试数据进行进一步研究.Siemens 程序是西门子公司为了研究不同代码覆盖规范的缺陷检测能力而开发的,由 7 个子程序组成,共 2 481 行可执行代码,132 个相关版本,21 879 个测试用例.实际实验中,每个版本对应的缺陷取相应的测试用例 10 个,实际使用测试用例 1 657 个.数据采集策略同样分为两种:

Policy *C*:检测到缺陷后立即移除,当程序中缺陷全部移除或者在指定时间(实验取 1 小时)内未检测到缺陷则停止测试.

Policy *D*:采用阶段测试进行数据采集,与 Policy *B* 类似,将步骤(8)更改为指定时间未检测到缺陷则停止测试.

以小时为单位采集数据,以两种策略分别进行 10 次检测,Policy *C* 平均检测时间 15.3 小时, Policy *D* 平均检测时间 17.8 小时,共 3 个阶段.对前 12 小时所观测到的数据进行统计,取算术平均值,结果以 DS_C 和 DS_D 表示,见表 4 和表 5.其中,t_i 为统计时刻,*FC* 表示缺陷数,∑*FC* 表示累积缺陷数.

Table 4 Dataset of defects while Siemens program testing by policy *C*

表 4 通过策略 *C* 进行 Siemens 程序测试缺陷数据集

t_i	*FC*	∑*FC*	t_i	*FC*	∑*FC*	t_i	*FC*	∑*FC*
1	28.4	28.4	5	10.4	83.3	9	3.8	103
2	23	51.4	6	6.5	89.8	10	2.1	105.1
3	11.7	63.1	7	4.1	93.9	11	3.2	108.3
4	9.8	72.9	8	5.3	99.2	12	5.2	113.5

Table 5 Dataset of defects while Siemens program testing by policy *D*

表 5 通过策略 *D* 进行 Siemens 程序测试缺陷数据集

Phase	t_i	*FC*	∑*FC*	Phase	t_i	*FC*	∑*FC*	Phase	t_i	*FC*	∑*FC*
1	1	35.1	35.1	1	5	4.7	82.5	2	9	12.6	100.7
	2	23.4	58.5		6	3.3	85.8		10	7.4	108.1
	3	11.9	70.4		7	1.7	87.5		11	4.2	112.3
	4	7.4	77.8		8	0.6	88.1		12	4.1	116.4

5.3 数据分析

所有模型所需参数以极大似然估计(maximum likelihood estimation,简称 MLE)[2,14,17]进行估计,我们进行参数估计的一般形式如下:

$$m(t)=a(1-\exp(-rt)),\lambda(t)=m'(t)\Rightarrow\ln\lambda(t)=\ln a+\ln r-rt.$$

引入均方误差函数作为似然函数,求解公式(9)即可得出 λ,a 计算公式:

$$\begin{cases} L(t,a,r)=\sum_{i=1}^{k}(\ln\lambda(t_i)-\ln\lambda_i)^2 \\ \dfrac{\partial L(a,\lambda)}{\partial a}=\dfrac{\partial L(a,\lambda)}{\partial\lambda}=0 \end{cases} \tag{9}$$

其中,λ_i 为时刻 t_i 测试所取得的缺陷率.

首先,以 DS_A 和 DS_B 利用 G-O 模型使用公式(9)对参数进行最大似然估计,并计算出 MSE 和 MEOP 值,分析阶段测试策略对可靠性评估的能力.取 MSE 值最小的作为评估数据,对比其 MEOP,研究其预测能力.计算结果对比见表 6.

表中仅列出了包括最小 *MSE* 在内的部分数据.使用数据集 DS_A 进行计算,在 t_i=14 时预测值与实际观测值均方误差最小,我们取 t_i=14 时的数据作为 G-O 模型评估结果;使用数据集 DS_B 进行计算,在 t_i=8 时预测望值与实际观测值均方误差最小,我们取 t_i=8 时的数据作为 G-O 模型评估结果.比较其 MEOP 可以看出,使用 DS_B 能获得与观测值更为相近的预测数据.因此,在使用 Policy *B* 进行可靠性评估时比 Policy *A* 有更好的预测能力.同样考查数据集 DS_C 和 DS_D,分别取 t_i=7 和 t_i=8 时的数据对比分析.可以看出,Policy *D* 进行可靠性评估时比 Policy *C* 有更好的预测能力,同时证明 P-NHPP 模型假设(3)、假设(6)的合理性.

Table 6 Comparison results of G-O model with different dataset

表 6 使用不同数据集时 G-O 模型结果对比

Dataset	t_i	FC	$\sum FC$	$\hat{\lambda}$	$\hat{a}$	$m(t_i)$	MSE	MEOP
DS_A	13	16.2	1.7	0.078	32.391	20.607	18.683	5.788
	14	18.1	1.9	0.086	31.737	22.279	18.597	5.934
	15	18.5	0.4	0.092	31.397	23.533	19.046	6.024
DS_B	7	10.3	2	0.104	26.729	13.806	6.906	5.661
	8	11.4	1.1	0.048	31.859	10.218	6.217	5.925
	9	12.2	0.8	0.115	26.091	16.852	7.932	6.004
DS_C	6	6.5	89.8	0.193	130.241	89.246	16.799	3.028
	7	4.1	93.9	0.183	130.366	94.126	14.407	3.184
	8	5.3	99.2	0.239	130.676	111.303	30.916	2.659
DS_D	7	1.7	87.5	0.137	130.626	80.589	28.157	3.298
	8	0.6	88.1	0.154	130.783	92.637	27.211	3.061
	9	12.6	100.7	0.129	130.854	89.711	37.604	3.115

下面使用数据集 DS_B 和 DS_D 对 G-O 模型和 P-NHPP 模型进行参数估计,其中,G-O 使用公式(9),P-NHPP 使用迭代方法分阶段估计.最后,根据估计的参数计算 MSE 和 MEOP.计算结果见表 7.

Table 7 Comparison results of different model with the same dataset

表 7 相同数据集下使用不同模型结果对比

DataSet	Model	t_i	FC	$\sum FC$	$\hat{\lambda}$	$\hat{a}$	$m(t_i)$	MSE	MEOP
DS_B	G-O	8	11.4	1.1	0.048	31.859	10.218	6.217	5.925
	P-NHPP	11	14	1.5	0.059	30.529	14.502	6.108	5.061
		12	14.9	0.9	0.062	32.194	16.849	5.916	5.412
		13	15.2	0.3	0.082	29.867	19.617	6.962	5.701
DS_D	G-O	8	0.6	88.1	0.154	130.783	92.637	27.211	3.061
	P-NHPP	6	3.3	85.8	0.143	130.695	81.583	36.150	3.319
		7	1.7	87.5	0.151	130.124	85.433	26.432	2.974
		8	0.6	88.1	0.154	130.783	94.637	32.537	2.261

使用数据集 DS_B 进行模型参数估计时,在 t_i=12 时预测值与实际观测值的均方误差最小,我们取 t_i=12 时的数据作为 P-NHPP 模型评估结果.从 MEOP 数据对比分析可以看出,使用 P-NHPP 比 G-O 模型进行评估时能获得更好点的预测效果.使用数据集 DS_D 进行模型参数估计时,在 t_i=7 时预测值与实际观测值的均方误差 MSE 最小,我们取 t_i=7 时的数据作为 P-NHPP 模型对数据集 DS_D 的评估结果.从 MEOP 数据对比分析可以看出,使用 P-NHPP 比 G-O 模型进行评估时能具有更好的预测效果.实验结果表明,在对 Space 软件和 Siemens 程序测试数据结果进行估计预测时,P-NHPP 模型相比 G-O 模型具有更好的拟合效果和预测能力.

6 结束语

关联缺陷在软件测试和评估过程中是不可忽略的,其存在很大程度上是由于缺陷的检测能力被其他缺陷

屏蔽所导致.本文给出了关联缺陷的形式化定义,以 Space 软件为例对关联缺陷的性质以及其影响进行了较为详细的阐述,并将广义关联应用于 G-O 模型的改进中,提出了 P-NHPP 评估模型.在对 Space 软件缺陷数据进行评估时,P-NHPP 具有较好的拟合效果和预测能力,但对关联缺陷的研究还有大量的工作要展开:

(1) 文中提出的 P-NHPP 的评估能力还需要有更多的实际测试来分析和证明.

(2) 结合面向对象和组件技术研究关联缺陷,从数据流和控制流的分析上进一步研究更符合实际应用的关联缺陷判定规则.

(3) 在缺陷检测过程中,对正关联缺陷的检测和移除.正关联的检测是关联缺陷检测最重要的组成部分,采用变异测试(mutation testing)技术插入其他缺陷,提高正关联的检测能力.这是下一步研究的重点.

(4) 关联缺陷对回归测试用例集选择的影响.在对用例集的选择或者最小化操作时,考虑关联缺陷影响因子,以提高回归测试效率.

(5) 关联缺陷预防系统的开发.从关联缺陷的角度研究并开发应用关联缺陷预防系统,促使关联缺陷真正达到工程领域的应用.

References:

[1] Xie M. Software Reliability Modeling. World Scientific Publishing Company, 1991. 9−20.

[2] Lyu MR. Hand Book of Software Reliability Engineering. McGraw-Hill, 1996. 71−118.

[3] Wang Q, Wu SJ, Li MS. Software defect prediction. Journal of Software, 2008,19(7):1565−1580 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1565.htm [doi: 10.3724/SP.J.1001.2008.01565]

[4] Katerina GP, Trivedi KS. Failure correlation in software reliability models. IEEE Trans. on Reliability, 2000,49(1):37−48. [doi: 10.1109/24.855535]

[5] Jing T, Jiang CH, Hu DB, Bai CG, Cai KY. An approach for detecting correlated software defects. Journal of Software, 2005,16(1): 17−28 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/17.htm [doi: 10.1360/jos160017e]

[6] Hamlet D. Are we testing for true reliability? IEEE Software, 1992,9(4):21−27. [doi: 10.1109/52.143097]

[7] Sahinoglu M. Compound-Poisson software reliability model. IEEE Trans. on Software Engineering, 1992,18(7):624−630. [doi: 10.1109/32.148480]

[8] Tomek LA, Muppala JK, Trivedi KS. Modeling correlation in software recovery blocks. IEEE Trans. on Software Engineering, 1993,19(11):1071−1086. [doi: 10.1109/32.256854]

[9] Chen S, Mills S. A binary Markov process model for random testing. IEEE Trans. on Software Engineering, 1996,22(3):218−223. [doi: 10.1109/32.489081]

[10] Bishop PG, Pullen FD. PODS revisited—A study of software failure behavior. In: Proc. of the IEEE Int'l Symp. on Fault Tolerant Computing. 1988. 2−8. [doi: 10.1109/FTCS.1988.5289]

[11] 2002. http://software.ccidnet.com/pub/disp/Article?columnID=375&articleID=25560&pageNO=1

[12] Miu HK, Li G, Zhu GM. Software Engineer Language—Z. Shanghai: Shanghai Scientific and Technology Literature Publishing House, 1999 (in Chinese).

[13] Rothermel G, Untch RH, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. on Software Engineering, 2001, 27(10):929−948. [doi: 10.1109/32.962562]

[14] Pham H, Nordmann L, Zhang XM. A general imperfect-software-debugging model with S-shaped fault-detection rate. IEEE Trans. on Reliability, 1999,48(2):169−175. [doi: 10.1109/24.784276]

[15] Yamada S. Software reliability models and their applications: A survey. In: Proc. of the Int'l Seminar on Software Reliability of Man-Machine Systems. Kyoto: Kyoto University, 2000. 56−80.

[16] Huang CY, Lin CT, Kuo SY, Lyu MR, Sue CC. Software reliability growth models incorporating fault dependency with various debugging time lags. In: Proc. of the 28th Annual Int'l Computer Software and Applications Conf. 2004. 186−191. [doi: 10.1109/CMPSAC.2004.1342826]

[17] Zhao M, Xie M. On the log-power NHPP software reliability model. In: Proc. of the 3rd Int'l Symp. on Software Reliability Engineering. North Carolina: Research Triangle Park, 1992. 14−22. [doi: 10.1109/ISSRE.1992.285862]

[18] Kapur PK, Garg RB, Kumar S. Contributions to Hardware and Software Reliability. World Scientific Publishing Company, 1999. 96−109.

[19] Goel AL, Okumoto K. Time-Dependent error-detection rate model for software and other performance measures. IEEE Trans. on Reliability, 1979,28(3):206−221. [doi: 10.1109/TR.1979.5220566]

[20] Yamada S, Ohba M, Osaki S. S-Shaped software reliability growth modeling for software error detection. IEEE Trans. on Reliability, 1983, 32(5): 475−484. [doi: 10.1109/TR.1983.5221735]

[21] Goel AL. Software reliability models: Assumptions, limitations, and applicability. IEEE Trans. on Software Engineering, 1985, SE-11(12):1411−1423. [doi: 10.1109/TSE.1985.232177]

[22] Jeske DR, Zhang XM, Pham L. Adjust software failure rates that are estimated from test data. IEEE Trans. on Reliability, 2005, 54(1):107−114. [doi: 10.1109/TR.2004.842531]

[23] Zhao J, Liu HW, Cui G, Yang XZ. A software reliability growth model considering testing environment and actual operation environment. Journal of Computer Research and Development, 2006,43(5):881−887 (in Chinese with English abstract). [doi: 10.1360/crad20060517]

[24] Hutchins M, Foster H, Goradia T, Ostrand T. Experiments on the effectiveness of dataflow-and controlflow-based test adequacy criteria. In: Proc. of the 16th Int'l Conf. on Software Engineering. 1994. 191−200.

附中文参考文献:

[3] 王青,伍书剑,李明树.软件缺陷预测技术.软件学报,2008,19(7):1565−1580. http://www.jos.org.cn/1000-9825/19/1565.htm [doi: 10.3724/SP.J.1001.2008.01565]

[5] 景涛,江昌海,胡德斌,白成刚,蔡开元.软件关联缺陷的一种检测方法.软件学报,2005,16(1):17−28. http://www.jos.org.cn/1000-9825/16/17.htm [doi: 10.1360/jos160017e]

[12] 缪淮扣,李刚,朱关铭.软件工程语言——Z.上海:上海科学技术文献出版社,1999.

[23] 赵靖,刘宏伟,崔刚,杨孝宗.考虑测试环境和运行环境的软件可靠性增长模型.计算机研究与发展,2006,43(5):881−887. [doi: 10.1360/crad20060517]

徐高潮(1966－),男,湖北孝感人,博士,教授,博士生导师,主要研究领域为网格计算,信息安全,软件可靠性评估.

付晓东(1966－),女,高级工程师,主要研究领域为分布式计算,软件可靠性分析.

刘新忠(1978－),男,博士生,主要研究领域为软件可靠性分析与评估,软件测试,软件形式化.

董玉双(1983－),男,博士生,主要研究领域为分布式计算,软件可靠性分析,软件测试.

胡亮(1968－),男,教授,博士生导师,主要研究领域为信息安全,分布式计算.