软件关联缺陷的一种检测方法*

景 涛+, 江昌海, 胡德斌, 白成刚, 蔡开元

(北京航空航天大学 自动控制系,北京 100083)

An Approach for Detecting Correlated Software Defects

JING Tao+, JIANG Chang-Hai, HU De-Bin, BAI Cheng-Gang, CAI Kai-Yuan

(Department of Automatic Control, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)

+ Corresponding author: Phn: +86-10-82315087, Fax: +86-10-82317328, E-mail: yjingtao@sohu.com, http://www.buaa.edu.cn

Received 2004-04-29; Accepted 2004-07-29

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. http://www.jos.org.cn/1000-9825/16/17.htm

Abstract: Software engineers always find the state of some defects can influence the detecting rate of other defects. These defects are correlated defects. This paper gives the definition of correlated software defects, gets some property of correlated defects by experiment, and then proposes an approach named testing approach with defects replacement, to detect correlated defects. It also analyzes the capability and efficiency of the approach by experiment. The data show that the approach is efficient to detect correlated software defects.

Key words: software testing; failure correlation; correlated software defect; random testing; defects replacement

摘 要: 软件中的关联缺陷是一种比较普遍的现象,某些缺陷的存在与否可能导致其他缺陷检测率的变化.软件关联缺陷是造成软件失效关联的根源.给出了关联缺陷的定义,通过一个软件实例验证了缺陷的关联关系,提出了一种缺陷放回的测试方法用来剔除关联缺陷,并通过实验数据分析了缺陷放回方法的能力和效率.实验数据表明,该方法能有效检测软件关联缺陷.

关键词: 软件测试;失效关联;软件关联缺陷;随机测试;缺陷放回

中图法分类号: TP311 **文献标识码**: A

软件在计算机系统中起着日益重要的作用,而随着计算机系统的广泛应用,软件可靠性问题在软件工程领域乃至整个计算机工程领域的重要性不断增加,在这方面已有许多研究.目前的软件可靠性模型主要有[1]Halstead 模型、Jelinski-Moranda 模型、Littlewood-Verrall 模型、Cai 模糊增长模型等等.已有的模型假设大都符合下述假设的 1 个或几个[2]:(a) 失效是相互独立的;(b) 发现缺陷立即剔除(及剔除缺陷的时间可忽略不计);(c) 剔除缺陷过程中不会引入新的缺陷.

* Supported by the National Natural Science Foundation of China under Grant No.60233020 (国家自然科学基金); the Aeronautics Basic Science Foundation of China under Grant No.01F51025 (航空基础科学基金)

作者简介: **景涛**(1975－),男,山东济南人,博士,主要研究领域为软件测试;**江昌海**(1983－),男,学士,主要研究领域为软件可靠性评估;**胡德斌**(1981－),男,硕士生,主要研究领域为软件可靠性分析;**白成刚**(1965－),男,博士,副教授,主要研究领域为软件可靠性分析,软件测试;**蔡开元**(1965－),男,教授,博士生导师,主要研究领域为软件可靠性工程,智能控制.

软件测试是保证软件质量、提高软件可靠性的重要手段.工程师与专家学者提出了很多软件测试方法和策略,包括功能测试、分区测试、路径测试、状态测试以及软件可靠性工程测试[3−6],针对这些测试方法的测试标准和特点也有大量的理论研究[7−10].

在测试过程中人们发现,软件缺陷间存在某种关联关系.Katerina 和 Trivedi[11]引入了失效关联的概念,认为实际测试过程中软件失效不是相互独立的,并提出了一种 Markov 更新模型来对有失效关联的软件可靠性进行建模.Chen 等人[12]认为测试回合不是相互独立的,在此基础上提出了一种二进制 Markov 过程模型,用来预测随机测试策略发现的软件失效数.Bishop 等人[13]指出可以用失效屏蔽效应来解释软件失效之间的关联关系,并可以通过适当的软件设计来杜绝失效关联.Mehmet[14]则根据失效关联/分支覆盖关联定义了一种 Bayesian 测试停止规则,以决定何时停止测试.

在工程领域也已经有人注意到了缺陷关联现象,如由上海微创软件有限公司出品的软件开发管理系统:微创 BMS XP[15]中提供了关联缺陷管理功能,定义了缺陷的 5 种关联,包括缺陷的依赖关联、重复关联、相关关联、文件关联和附件,主要是用来刻画缺陷与缺陷之间、缺陷与其相关的文件之间的关系.当若干个缺陷之间存在某种关系时,或者缺陷需要和某个文件相关联时,就需要使用缺陷关联.

总的来说,人们已经认识到了软件失效之间存在关联关系,但对这一现象还缺乏深入的研究,没有从软件失效的根源,也就是软件缺陷的角度来分析关联的原因.工程领域对于关联缺陷也缺乏有效的检测方法.

我们认为关联缺陷是指缺陷间相互影响的现象.本文给出了关联缺陷的定义,通过一个软件实例验证了缺陷的关联关系,提出了一种缺陷放回的测试方法用来剔除关联缺陷,并通过实验数据分析了缺陷放回方法的能力和效率.

本文第 1 节给出关联缺陷的定义.第 2 节分析关联缺陷的性质.第 3 节给出关联缺陷的检测方法:缺陷放回的测试方法.第 4 节通过实验,分析缺陷放回方法的能力与效率.第 5 节做进一步讨论.最后给出结论.

1 关联缺陷的定义

以 space 软件[16]为例,space 软件有 36 个缺陷,13 498 个测试用例,是一个公认的典型测试对象.在对 space 软件进行测试的过程中我们发现,某些情况下有的缺陷很难触发.例如将 space 软件的所有缺陷编号 1~36 号,若测试过程中除了 2 号缺陷,其余缺陷都被剔除,则即使用穷举法遍历测试用例库中的所用测试用例也无法触发 2 号缺陷.由于软件的测试剖面与运行剖面经常有较大差别,用户极有可能发现测试人员无法发现的缺陷.对于安全关键软件一旦出现这种情况将会导致灾难性的后果,因此有必要加以研究.

经过初步分析发现,单独注入 2 号缺陷虽然无法触发,但如果注入全部 36 个缺陷,使用随机测试策略进行测试,出现缺陷不能触发的几率仅为 2%(400 次测试过程中有 8 次不能触发 2 号缺陷).也就是说大部分情况下 2 号缺陷是可以触发的,能否触发 2 号缺陷与其他缺陷的状态有关.可以假定缺陷间存在某种关联关系,一种情况是某些缺陷存在时无法触发或很难触发另一些缺陷;第 2 种情况是剔除某些缺陷后无法触发或很难触发另一些缺陷.

假定对于一个软件 S,

(1) 软件中含有 n 个缺陷,第 i 号缺陷记为 e_i,i=1,2,…,n;

(2) 软件的状态用 X 表示,$X=[x_1,x_2,...,x_n]$,其中 $x_i=\begin{cases}1, & \text{第}i\text{号缺陷打开}\\ 0, & \text{第}i\text{号缺陷关闭}\end{cases}$,$i$=1,2,…,$n$;

(3) 记软件状态为 X_i,若只有 $x_i=1$,其他分量都为 0;软件状态为 X_{ij},表示 $x_i=1,x_j=1$,其他分量都为 0;

(4) $Y=[x_{l_1},x_{l_2},...,x_{l_m}]$ 是 X 的子向量;

(5) $A_X=\{0,1\}^n$, $A_Y=\{0,1\}^m$;

(6) 如果输入确定,输出就是确定的,不含随机因素;

(7) $\beta_i\in\{0,1\}$ (0 表示不能触发缺陷,1 表示能触发缺陷)表示用例库对第 i 号缺陷的检测能力. $\beta_i=\sum_{X\in A_X}p_X\beta_i(X)$,其中 $\beta_i(X)$ 表示软件处于 X 状态时第 i 号缺陷的检测能力, p_X 表示 X 状态出现的概率.

定义.

定义 $e_i + e_j$ 表示第 i 号缺陷与第 j 号缺陷都处于打开状态,其他所有缺陷处于关闭状态.显然"+"运算符满足交换率和结合率.

若对于任意 $X \in A_X$ 有 $\beta_i(X) = \beta_i(x_i)$,则第 i 号缺陷只与自身相关,其检测能力与其他缺陷的状态无关,$\beta_i(x_i) = \begin{cases} \beta_i, & x_i = 1 \\ 0, & x_i = 0 \end{cases}$.

若 $\beta_i(X_i) \neq \beta_i(X_{ij})$,则称 e_j 是 e_i 的关联缺陷,记为 $e_i \,\&\, e_j = 1$,&称为关联运算符.称第 i 号缺陷与第 j 号缺陷是一对关联缺陷.

若 $\beta_i(X_i) = 1$且$\beta_i(X_{ij}) = 0$,这种关联关系我们称之为第 1 类关联;若 $\beta_i(X_i) = 0$且$\beta_i(X_{ij}) = 1$,这种关联关系我们称之为第 2 类关联.

若 $\beta_i(X_i) = \beta_i(X_{ij})$,则 $e_i \,\&\, e_j = 0$.

显然 $e_i \,\&\, e_i = 1 (i = 1,...,n)$,即关联运算符&满足自反率,我们今后的讨论中不再考虑缺陷与自身关联的问题.

若对于任意 $X \in A_X$ 有 $\beta_i(X) = \beta_i(Y)$,则第 i 号缺陷与缺陷 $l_1, l_2, ..., l_m$ 是相关的,它们共同称为一组关联缺陷.

2 关联缺陷的性质

我们通过实验验证 space 软件中含有关联缺陷,并发现关联运算符&不满足交换率($e_i \,\&\, e_j = e_j \,\&\, e_i$)、分配率($e_k \,\&\, (e_i + e_j) = e_k \,\&\, e_i + e_k \,\&\, e_j$).

实验步骤为

(1) 对 space 软件的 36 个缺陷进行编号,编号为 1,2,…,33,35,36,37.保留 34 号;

(2) 令 i=0;

(3) 若 $i < 36$,则令 i=i+1,否则转第(11)步;

(4) 设定软件状态 X_i.打开第 i 号缺陷,关闭其他所有缺陷;

(5) 使用穷举法顺序测试用例库中每个测试用例,估计 $\beta_i(X_i)$.若所用测试用例都不能触发第 i 号缺陷,则 $\hat{\beta}_i(X_i) = 0$,否则 $\hat{\beta}_i(X_i) = 1$;

(6) 令 j=1;

(7) 设定软件状态 X_{ij}.打开第 i,j 号缺陷,关闭其他所有缺陷;

(8) 使用穷举法顺序测试用例库中每个测试用例,且不剔除第 j 号缺陷,估计 $\beta_i(X_{ij})$.若所用测试用例都不能触发第 i 号缺陷,则 $\hat{\beta}_i(X_{ij}) = 0$,否则 $\hat{\beta}_i(X_{ij}) = 1$;若 $\hat{\beta}_i(X_i) \neq \hat{\beta}_i(X_{ij})$,则 $e_i \,\&\, e_j = 1$,记录结果;

(9) 若 $j < 36$ 令 j=j+1;否则转第(3)步;

(10) 若 $j \neq i$,则转第(7)步;否则转第(9)步;

(11) 结束.

我们把实验数据整理为关联关系矩阵,见表 1.

从表 1 容易看出关联运算符&不满足交换率,例如 $e_1 \,\&\, e_3 = 1$但 $e_3 \,\&\, e_1 = 0$,也就是说 3 号缺陷对 1 号缺陷有影响,反之则不然.另外,数据表明 1 号、2 号和 32 号缺陷单独存在时,当前的测试用例库无法检测到它们,也就是说,测试过程中如果只剩下一个缺陷而这个缺陷是 1 号或 2 号或 32 号时就可以停止测试了;它们也是最容易受到其他缺陷影响的,不考虑自身,分别有 30,30,20 个缺陷与它们关联.

Table 1 Matrix of correlated defects
表 1 关联关系矩阵

e_i&e_j (i \ j)	1	2	3	4	5	6	7	8	9	…	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	35	36	37
1	1	0	1	1	0	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1
2	0	1	1	1	1	1	1	1	1		1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1
3	0	0	1	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	1	1	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
8	0	0	0	1	0	1	0	1	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
9	0	0	0	1	0	1	0	0	1		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
10	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
11	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
12	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
13	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
14	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
15	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
16	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
17	0	0	0	1	0	1	0	0	0		1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
18	0	0	0	1	0	1	0	0	0		0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
19	0	0	0	1	0	1	0	0	0		0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
20	0	0	0	1	0	1	0	0	0		0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
21	0	0	0	1	0	1	0	0	0		0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
22	0	0	0	1	0	1	0	0	0		0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0
23	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0
24	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
25	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
26	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
27	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0
28	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
30	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
32	0	0	0	0	0	0	1	1	1		1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0
33	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0
35	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0
36	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
37	0	0	0	1	0	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1

当软件中只有两个缺陷时,32 号缺陷最难触发,有 3/7 的概率找不到它.

3,4,5,6,29,31,36 号缺陷最容易触发,它们不受其他缺陷的关联影响,任何时候都可以触发;26,28,30 号缺陷也比较容易触发,它们仅受 4,6 号缺陷的关联影响.

从表 1 可以看出,4,6,30,28 号缺陷对大部分缺陷有关联作用,不考虑自身,受其影响的缺陷分别为 28,28,25,20 个.15 号缺陷也对多达 10 个缺陷有关联作用.如果不剔除这几个缺陷,大部分缺陷都无法触发.幸运的是这几个缺陷都是很容易剔除的,4 号、6 号缺陷在任何时候都可以触发.

Space 软件中存在 183 个关联缺陷对(不考虑与自身关联).

还可以验证关联运算符&不满足分配率,对于同时打开 3 个缺陷的情况做了部分验证,实验数据见表 2(数值运算规则:0+0=0,0+1=1,1+1=1).

Table 2 Data of experiment on assignment operator
表 2 分配率验证数据

i=35	e_i & e_j	e_i & e_k	e_i & (e_j+e_k)
j=14,k=15	0	0	1
j=15,k=14	0	0	1
i=37	e_i & e_j	e_i & e_k	e_i & (e_j+e_k)
j=14,k=15	0	0	1
j=15,k=14	0	0	1

3 关联缺陷的检测方法:缺陷放回的测试方法

对于测试人员,最终关心的不是找到更多缺陷的关联关系,而是能否找到更多缺陷.也就是说尽量避免第 2 类关联缺陷,一旦出现第 2 类关联缺陷依然能够检测、剔除.为此我们设计了一种缺陷放回的测试方法,当出现第 2 类关联缺陷时能够有效地检测剔除.为此先对测试过程做一些基本的假定,并给出随机测试的步骤,在此基础上定义缺陷放回的测试方法.假定,

(1) 测试过程中每发现一个缺陷就剔除掉这个缺陷,且不会引入新的缺陷;

(2) 使用过的测试用例立即重新放回测试用例库.

以上假定接近实际测试过程中的单元测试,但不适合系统测试(发现一批缺陷后再进行剔除).

随机测试策略的步骤如下:

第 1 步.根据需要设置缺陷状态.若要测试全部缺陷,令全部缺陷为打开状态.

第 2 步.从测试用例库中随机选一个测试用例进行测试.

第 3 步.如果 orignal.exe(正确文件)与 fault.exe(有缺陷文件)输出不同,说明发现了缺陷.从可能的缺陷中随机剔除一个.

第 4 步.如果已剔除所有缺陷,转第 5 步.否则转第 2 步.

第 5 步.停止.

缺陷放回是指每次输入一个测试用例前,按照一定比例放回一个已剔除的缺陷.缺陷放回的方法需要与其他测试策略相结合,很多测试策略可以方便地引入缺陷放回的方法.带缺陷放回的随机测试策略(简写作 RD1)具体步骤定义如下:

定义缺陷描述数组 $defects[n][2]$,其中,i=1,2,…,n.

$$defects[i][0]=\begin{cases}0, & 第i号缺陷关闭\\ 1, & 第i号缺陷打开,\end{cases}$$

$$defects[i][1]=\begin{cases}0, & 未剔除过第i号缺陷\\ 1, & 曾剔除过第i号缺陷\end{cases}.$$

第 1 步.根据需要初始化 defects 数组.若要测试全部缺陷,令 $defects[i][0]$=1,$defects[i][1]$=0,i=1,2,…,n.

第 2 步.以一定的概率(实验中取 0.01)重新打开一个已剔除的缺陷,即 if($defects[i][1]$=1) then $defects[i][1]$=1.

第 3 步.从测试用例库中随机选一个测试用例进行测试.

第 4 步.如果 orignal.exe(正确文件)与 fault.exe(有缺陷文件)输出不同,从可能的缺陷中随机剔除一个.假定该缺陷编号为 j,则令 $defects[j][0]$=0,$defects[j][1]$=1.

第 5 步.如果对任意整数 $i\in[1,n]$有 $defects[i][1]$=1,则说明已找到所有缺陷,转第 6 步.否则转第 2 步.

第 6 步.停止.

4 缺陷放回的测试方法的能力及效率研究

下面将通过实验对缺陷放回的测试方法进行研究,分析其能力和效率.

使用 space 软件,分别单独打开 1 号、2 号、32 号缺陷,设定其他缺陷为已知缺陷,使用缺陷放回的测试方法进行 40 次测试,实验数据见表 3.

Table 3 Capability of defects-replacement test strategy
表 3 缺陷放回方法的检测能力

	1	2	32
Detecting rate (%)	100	100	100
Average of testing runs	380.23	831.45	549.75
Stdev of testing runs	336.821	884.351	417.115

实验数据表明,缺陷放回的测试方法能够100%地剔除第2类关联缺陷.相比通常的测试方法,同样条件下剔除第 2 类关联缺陷的比例为 0.

考察 2 000 步内剔除缺陷的比例,可见随着测试步数的增加,剔除缺陷的比例呈指数趋势上升.同一测试步数内剔除 1 号缺陷的几率最高,其次是 32 号,然后是 2 号.这 3 个缺陷按照检测率大小从高到低排列应是 1 号、32 号和 2 号.

图 1 中横轴为测试步数 x,纵轴为缺陷检测率 y.

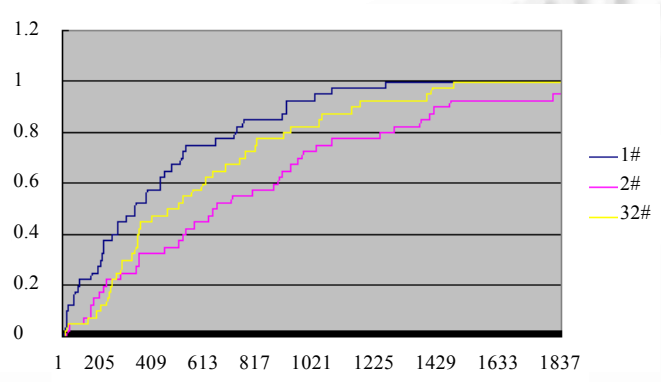

Fig.1 Detecting rate of software defects
图 1 缺陷检测率曲线

下面考察一下缺陷放回的测试方法的效率.使用 space 软件,打开全部 36 个缺陷,分别用常规的随机测试策略和带缺陷放回的随机测试策略进行 400 次测试.随机策略下出现 8 次 2 号缺陷不能剔除;带缺陷放回的随机策略下只出现 4 次 2 号缺陷单独存在的情况,且最终全部剔除.也就是说,带缺陷放回的随机策略不但能够在出现第 2 类关联缺陷时剔除缺陷,而且能够减少第 2 类关联缺陷出现的几率.其原因是缺陷放回降低了 2 号缺陷单独存在的几率.

令 $b_i(k)$ 为第 i 次测试中,从剔除第$(k-1)$个缺陷到剔除第 k 个缺陷所用的测试步数,令

$$bn(k)=\sum_{i=1}^{400} b_i(k),$$

并令

$$tn(k)=\sum_{j=1}^{k} bn(j),$$

$$\bar{t}(k)=\frac{1}{400}\sum_{i=1}^{400} t_i(k),$$

$$dn(k)=\sqrt{\frac{1}{399}\left[\left(t_1(k)-\bar{t}(k)\right)^2+\ldots+\left(t_{400}(k)-\bar{t}(k)\right)^2\right]}.$$

均值相对误差 $$\Delta_{\bar{\Sigma}(k)}=\frac{\bar{t}(k)(\mathrm{R1})-\bar{t}(k)(\mathrm{RD1})}{\bar{t}(k)(\mathrm{R1})}\times 100\%.$$

均方差相对误差 $$\Delta_{D(k)}=\frac{dn(k)(\mathrm{R1})-dn(k)(\mathrm{RD1})}{dn(k)(\mathrm{R1})}\times 100\%.$$

其中 R1 代表随机测试策略,RD1 代表缺陷放回测试策略.

累计测试步数的相关实验数据见表 4.

Table 4 Data of random testing and defects-replacement testing
表 4 纯随机测试与带缺陷放回的随机测试数据

k	$\bar{t}(k)$ (R1)	$\bar{t}(k)$ (RD1)	$\Delta_{\bar{\Sigma}(k)}$ (%)	$dn(k)$ (R1)	$dn(k)$ (RD1)	$\Delta_{D(k)}$ (%)
1	1.05	1.03	1.90	0.209 576	0.170 801	18.50
2	2.1	2.1	0.00	0.296 252	0.340 978	−15.10
3	3.15	3.17	−0.63	0.407 149	0.433 837	−6.55
4	4.23	4.25	−0.47	0.501 68	0.526 247	−4.90
5	5.3	5.33	−0.57	0.554 173	0.596 04	−7.55
6	6.4	6.42	−0.31	0.623 741	0.710 814	−13.96
7	7.52	7.53	−0.13	0.732 642	0.781 426	−6.66
8	8.7	8.69	0.11	0.829 253	0.869 906	−4.90
9	9.92	9.93	−0.10	1.034 629	1.023 582	1.07
10	11.15	11.2	−0.45	1.180 42	1.189 754	−0.79
11	12.5	12.57	−0.56	1.410 592	1.354 506	3.98
12	13.96	14	−0.29	1.526 699	1.551 133	−1.60
13	15.51	15.53	−0.13	1.789 419	1.803 351	−0.78
14	17.08	17.2	−0.70	2.002 165	2.200 478	−9.90
15	18.82	19.03	−1.12	2.316 832	2.566 752	−10.79
16	20.64	20.79	−0.73	2.622 235	2.886 724	−10.09
17	22.65	22.94	−1.28	2.989 674	3.289 475	−10.03
18	24.88	25.19	−1.25	3.481 997	3.915 837	−12.46
19	27.57	27.84	−0.98	4.407 08	4.719 468	−7.09
20	30.93	31.06	−0.42	5.573 044	5.801 097	−4.09
21	34.72	34.44	0.81	6.987 876	6.604 555	5.49
22	39.26	39.19	0.18	8.115 688	8.457 018	−4.21
23	45.35	44.89	1.01	10.215 18	10.526 01	−3.04
24	52.47	52.03	0.84	13.076 36	13.760 35	−5.23
25	61.72	61.49	0.37	16.217 44	17.944 35	−10.65
26	72.18	73.77	−2.20	19.607 97	24.018 76	−22.49
27	85.58	86.69	−1.30	26.532 98	27.858 3	−5.00
28	100.68	102.65	−1.96	31.525 44	33.276 89	−5.56
29	117.61	123.19	−4.74	36.571 32	40.015 79	−9.42
30	138.67	144.74	−4.38	44.648 19	49.431 55	−10.71
31	166.37	172.12	−3.46	60.201 74	58.480 95	2.86
32	207.04	210.51	−1.68	79.591 1	76.076 86	4.42
33	302.08	302.21	−0.04	203.29 58	160.748 3	20.93
34	555.78	562.87	−1.28	485.043 2	436.913 2	9.92
35	1104.95	1 136.61	−2.87	846.241 9	804.203 9	4.97
36	2212.3	2 052.22	7.24	1 467.511	1 275.007	13.12

图 2 及图 3 是根据表 4 的数据画出的.从图 2 可以看出加入缺陷放回后不论是均值还是均方差都有所下降,测试效率提高了.均值提高 7.24%,均方差提高了 13.12%.

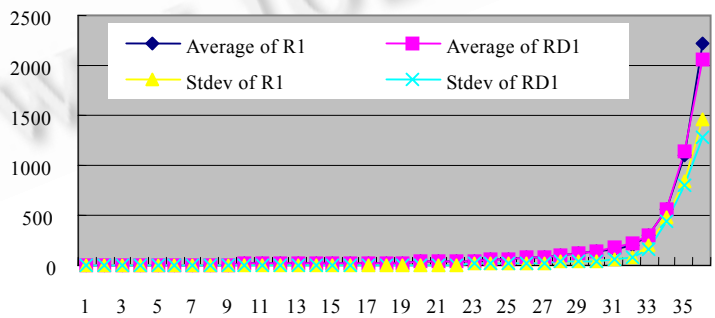

Fig.2 Efficiency of random testing and defects-replacement testing
图 2 随机测试与缺陷放回测试效率比较

从图 3 可以看出,缺陷放回方法与随机测试策略相比,在测试前期和中期相近(累计测试步数均值相近),在后期 25~35 个缺陷时累计测试步数均值明显下降,但在剔除最后一个缺陷时累计测试步数均值明显上升.可见

缺陷放回对于缺陷检测率是有影响的.

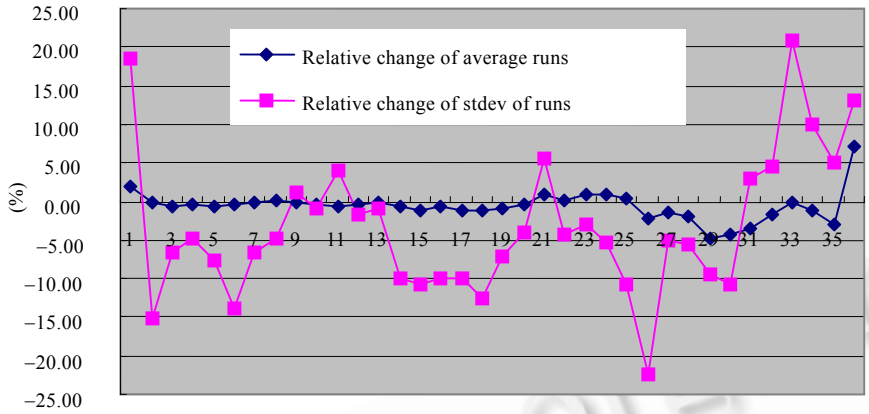

Fig.3 $\Delta_{\bar{\Sigma}(k)}$ and $\Delta_{D(k)}$

图 3 $\Delta_{\bar{\Sigma}(k)}$ 及 $\Delta_{D(k)}$

为了考察缺陷放回方法对缺陷检测率的影响,我们考虑最后一个缺陷.通过对随机测试数据的分析,可得最后一个被剔除的缺陷的信息,主要是 12 号缺陷 132 次、18 号缺陷 129 次和 33 号缺陷 113 次,可见这 3 个缺陷最难剔除,在最后被剔除的几率最大,测试方法对缺陷检测率的影响主要体现在这 3 个缺陷上.分别打开 12、18 及 33 号缺陷,用随机策略和带缺陷放回的随机策略分别进行 40 次实验,统计测试数据如图 4~图 6 及表 5 所示.

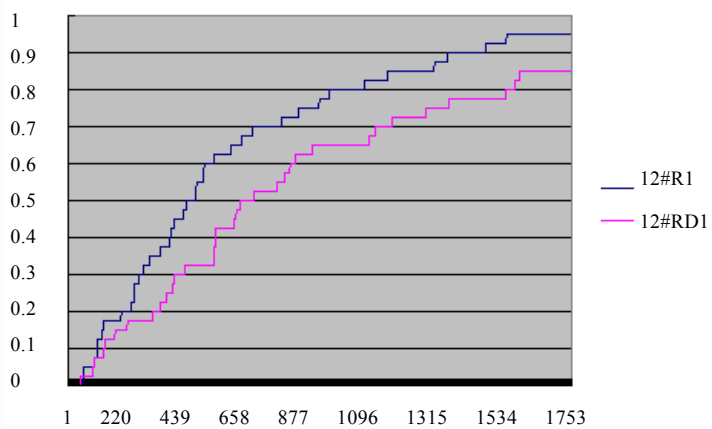

Fig.4 Detect rate of 12th defect under two test strategies

图 4 两种策略下 12 号缺陷检测率对比

图 4 中,加入缺陷放回后,12 号缺陷的检测率降低了;图 5 中,缺陷放回对 18 号缺陷检测率的影响比较小;图 6 中,加入缺陷放回后 33 号缺陷的检测率有大幅度上升.图 4~图 6 分别代表了缺陷放回方法对缺陷检测率的 3 种典型影响.哪一种占主导地位决定了最终缺陷放回测试方法对测试效率的影响是上升还是下降.检测率升降的趋势是由软件及缺陷的固有特性决定的,但升降的幅度也会受缺陷放回方法的参数(缺陷放回概率)的影响.缺陷放回方法对于同一个软件中的不同缺陷的检测率可能会有不同影响,但对同一个缺陷的检测率影响趋势应是不变的.如果缺陷放回方法与某种测试策略结合降低了某个缺陷的检测率,则缺陷放回方法与其他测试策略结合同样会降低该缺陷的检测率.缺陷放回方法对不同缺陷的检测率影响的总和决定了最终对整个软件检测效率的影响,测试效率有可能提高,也有可能不变或者降低,由被测软件决定.

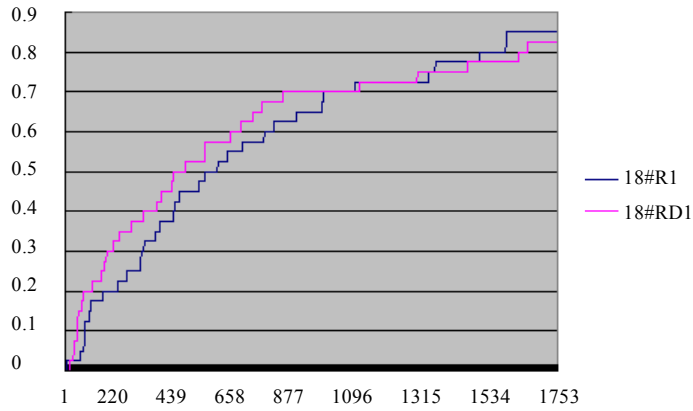

Fig.5 Detect rate of 18th defect under two test strategies
图 5 两种策略下 18 号缺陷检测率对比

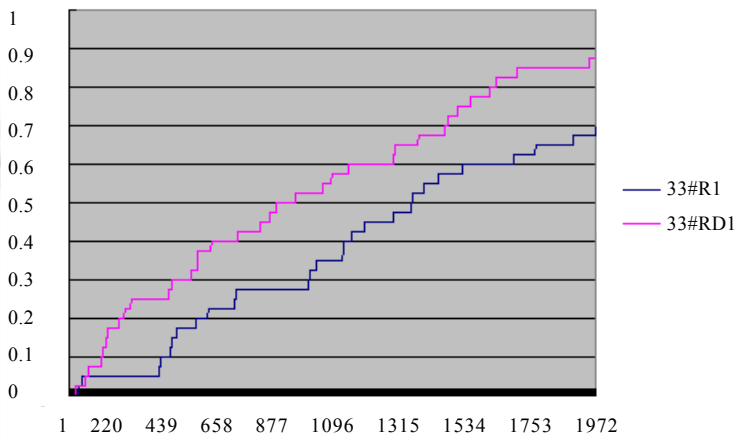

Fig.6 Detect rate of 33th defect under two test strategies
图 6 两种策略下 33 号缺陷检测率对比

Table 5 Test data of 12th, 18th, 33th defects under two test strategies
表 5 两种测试策略下的 12 号、18 号、33 号缺陷检测数据

i	$t(i)$	$E(i)$	$E'(i)$
12	132	704.17	1 056.35
18	129	887.42	934.7
33	113	2 412.5	974.4

从表 5 可以看到,缺陷放回方法大大提高了 33 号缺陷的检测率,测试步数均值从 2 412.5 降低到 974.4.假定不同测试方法下最后一个缺陷出现的几率不变,并假定 40 次实验与 400 次实验对测试结果影响不大,则最终随机测试步数均值为 $E_{R1}=\frac{\sum E(i)*t(i)}{\sum t(i)}$=1283.53,缺陷放回方法测试步数均值 $E_{RD1}=\frac{\sum E'(i)*t(i)}{\sum t(i)}$=989.63,缺陷放回测试效率有明显提高.这就说明了图 2 中缺陷放回方法在测试的最后提高了效率的原因.

5 讨 论

5.1 缺陷放回概率的选取

经过上面的一系列实验我们发现,缺陷放回技术相对于随机测试最明显的优势在于能够发现关联缺陷.问题在于,前面的实验中缺陷放回概率都定为 0.01,这是根据经验设定的.假定每个测试回合进行 2 000 步测试,则每个回合平均会放回 20 个缺陷,对于 space 软件来说有半数缺陷可能被放回.从表 1 可以看到,与 1 号、2 号及

32 号缺陷相关联的缺陷都很多,在 2 000 步内放回与 1 号、2 号及 32 号缺陷相关联的缺陷,从而剔除 1 号、2 号及 32 号缺陷的概率还是很大的.如果提高缺陷放回的概率,剔除 1 号、2 号及 32 号缺陷的概率会更大.我们把缺陷放回概率提高为 0.02,仍然使用 space 软件,分别单独打开 1 号、2 号、32 号缺陷,设定其他缺陷为已知缺陷.每测试 200 步统一剔除已发现的缺陷,且不引入新的缺陷,使用缺陷放回的测试方法进行 40 次测试,考察 2 000 步内剔除缺陷的情况,如图 7 所示.与图 1 缺陷放回概率为 0.01 相比,剔除缺陷的速度提高了.

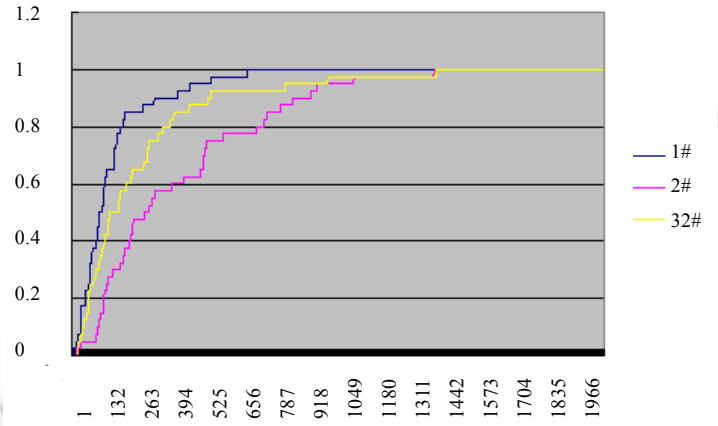

Fig.7 Detecting rate of software defects

图 7 缺陷检测率曲线

选取放回概率应遵循哪些原则,是不是缺陷放回概率越高越好?由于软件的复杂性,目前无法给出结论,我们准备在将来的工作中继续加以研究.

5.2 关于基本假设

目前测试的一个基本假设是测试过程中每发现一个缺陷就剔除掉这个缺陷,且不会引入新的缺陷,这一假设比较符合单元测试的情况,但对于系统测试并不适用.系统测试中通常测试人员按照一定的测试计划进行测试,经过一定的时间,发现一定数量的缺陷后交给开发人员修改,有的缺陷会被剔除,有的缺陷可能无法剔除,还有可能引入新的缺陷.修改完成之后再重新测试.因此比较符合系统测试的假设是测试过程中每测试 x 步之后才剔除已发现的部分缺陷,并有可能引入新的缺陷.对于缺陷放回技术来说,引入新的缺陷不会产生很大影响,在这里我们只想简单验证一下发现缺陷后不立即剔除,而是等到一定测试步数时再剔除会不会影响对关联缺陷的检测.

仍然使用 space 软件,分别单独打开 1 号、2 号、32 号缺陷,设定其他缺陷为已知缺陷.每测试 200 步统一剔除已发现的缺陷,且不引入新的缺陷,使用缺陷放回的测试方法,缺陷放回概率为 0.01,进行 40 次测试,考察 2 000 步内发现缺陷的情况(注意是发现而不是剔除缺陷).如图 8 所示,大多数情况下所有缺陷会在 2 000 步内发现,可见即使改变了基本假设,缺陷放回方法仍然有效.

对比图 1、图 7 和图 8 可以发现,不论测试策略怎样调整,1 号缺陷总是较容易发现,2 号最难发现,说明这是软件本身的固有特性.

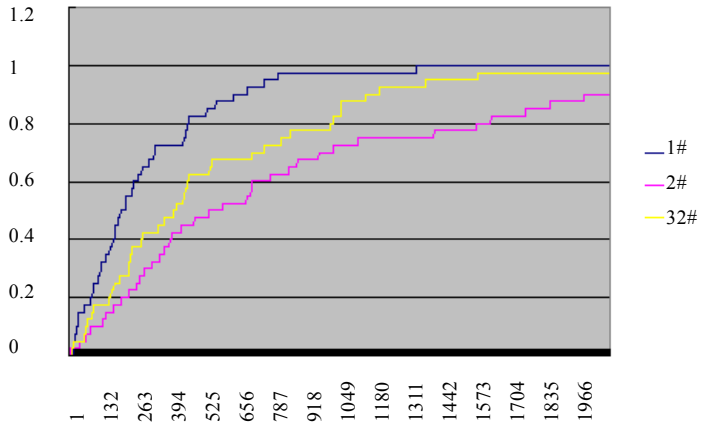

Fig.8 Detecting rate of software defects

图 8 缺陷检测率曲线

6 结论及未来工作

关联缺陷现象在软件测试中并不少见,可以把关联分为第 1 类关联和第 2 类关联.缺陷放回的测试方法对于处理关联缺陷是非常有效的,同时缺陷放回对于测试效率也有一定的影响.在 space 软件中缺陷放回对于提高测试效率是有益的.

未来还有大量工作,包括研究不同的缺陷放回概率对于关联缺陷检测能力的影响、对测试效率的影响;缺陷放回测试方法对于更多其他软件的测试效果;缺陷放回方法与其他测试策略相结合的研究;如何利用被测软件的特性,结构化、面向对象还是面向构件,来调整缺陷放回的参数?限于篇幅无法在这里讨论,将在以后的工作中做进一步研究.

References:

[1] Cai KY. Elements of Software Reliability Engineering. Beijing: Tsinghua University Press, 1995 (in Chinese).

[2] Chen XJ, Su ZH, Gao G. Research on software reliability models. Reliability and Environment Experiment on Electronic Products, 1996,5:17−24 (in Chinese with English abstract). http://www.edu.cnki.net/

[3] Beizer B. Software Testing Techniques. 2nd ed., Van Nostrand Reinhold, 1990.

[4] Myers GJ. The Art of Software Testing. John Wiley & Sons, 1979.

[5] Musa JD. Software-Reliability-Engineered testing. Computer, 1996,29(11):61−68.

[6] Binder RV. Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley, 2000.

[7] Chen TY, Yu YT. On the expected number of failures detected by subdomain testing and random testing. IEEE Trans. on Software Engineering, 1996,22(2):109−119.

[8] Frankl PG, Weyuker EJ. A formal analysis of the fault-detecting ability of testing methods. IEEE Trans. on Software Engineering, 1993,19(2):202−213.

[9] Gutjahr WJ. Partition testing vs. random testing: The influence of uncertainty. IEEE Trans. on Software Engineering, 1999,25(5):661−674.

[10] Zhu H, Hall PA, May JHR. Test coverage and adequacy. ACM Computing Surveys, 1997,129(4):366−427.

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

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

[13] Bishop PG, Pullen FD. PODS revisited−A study of software failure behavior. In: Proc. of the IEEE Int'l Symp. on Fault Tolerant Computing. Tokio, 1988. 1−8.

[14] Sahinoglu M. An empirical Bayesian stopping rule in testing and verification of behavioral models. IEEE Trans. on Instrumentation and Measurement, 2003,52(5):1428−1443.

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

[16] Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. on Software Engineering, 2001,27(10):929−948.

附中文参考文献:

[1] 蔡开元.软件可靠性工程基础.北京:清华大学出版社,1995.

[2] 陈学军,苏振华,高工.软件可靠性模型研究.电子产品可靠性与环境试验,1996,5:17−24. http://www.edu.cnki.net/

敬 告 作 者

《软件学报》创刊以来,蒙国内外学术界厚爱,收到许多高质量的稿件,其中不少在发表后读者反映良好,认为本刊保持了较高的学术水平.但也有一些稿件因不符合本刊的要求而未能通过审稿.为了帮助广大作者尽快地把他们的优秀研究成果发表在我刊上,特此列举一些审稿过程中经常遇到的问题,请作者投稿时尽量予以避免,以利大作的发表.

1. 读书偶有所得,即匆忙成文,未曾注意该领域或该研究课题国内外近年来的发展情况,不引用和不比较最近文献中的同类结果,有的甚至完全不列参考文献.

2. 做了一个软件系统,详尽描述该系统的各个方面,如像工作报告,但采用的基本上是成熟技术,未与国内外同类系统比较,没有指出该系统在技术上哪几点比别人先进,为什么先进.一般来说,技术上没有创新的软件系统是没有发表价值的.

3. 提出一个新的算法,认为该算法优越,但既未从数学上证明比现有的其他算法好(例如降低复杂性),也没有用实验数据来进行对比,难以令人信服.

4. 提出一个大型软件系统的总体设想,但很粗糙,而且还没有(哪怕是部分的)实现,很难证明该设想是现实的、可行的、先进的.

5. 介绍一个现有的软件开发方法,或一个现有软件产品的结构(非作者本人开发,往往是引进的,或公司产品),甚至某一软件的使用方法.本刊不登载高级科普文章,不支持在论文中引进广告色彩.

6. 提出对软件开发或软件产业的某种观点,泛泛而论,技术含量少.本刊目前暂不开办软件论坛,只发表学术文章,但也欢迎材料丰富,反映现代软件理论或技术发展,并含有作者精辟见解的某一领域的综述文章.

7. 介绍作者做的把软件技术应用于某个领域的工作,但其中软件技术含量太少,甚至微不足道,大部分内容是其他专业领域的技术细节,这类文章宜改投其他专业刊物.

8. 其主要内容已经在其他正式学术刊物上或在正式出版物中发表过的文章,一稿多投的文章,经退稿后未作本质修改换名重投的文章.

本刊热情欢迎国内外科技界对《软件学报》踊跃投稿.为了和大家一起办好本刊,特提出以上各点敬告作者.并且欢迎广大作者和读者对本刊的各个方面,尤其是对论文的质量多多提出批评建议.