软件复杂性与测试用例集价值

吴 际,金茂忠,刘 超
(北京航空航天大学计算机学院,北京 100083)

摘 要: 测试用例集价值是测试用例集的重要问题,本文给出了以复杂性度量为基础的测试用例集合价值度量模型,并在此度量基础上定义了三个重要的度量:测试密度,测试用例价值贡献以及测试用例集合执行相似度,并通过实际的例子指出了如何进行度量.

关键词: 软件测试;测试用例价值;测试用例集价值;测试密度;软件复杂性
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12A-2166-03

Software Complexity and Value of Test Cases Satisfied with Testing Criterion

WU Ji, JIN Mao-zhong, LIU Chao
(*School of Computer Science and Engineering of Beihang University, Beijing* 100083)

Abstract: This paper presents the value of test cases set measurement, test density measurement, value of test case measurement and degree of execution comparability of test cases set model based on complexity measurement. A practical measurement of the models is presented too.

Key words: software test; value of test case; value of test cases set; test density; software complexity

1 引言

软件测试是软件质量保证的重要手段,一个好的测试首先需要有效的测试用例.测试用例有效性的研究近几年在国际上受到了相当的重视,其主要研究目标是测试用例在发现软件故障(或缺陷)方面的有效性或效率[1~3].对于一个确定的测试准则,往往存在多组不同的测试用例集合满足该准则.如何评价、比较、选择和改进这些测试用例集合是测试需要回答的重要问题.

2 测试用例集的价值

所有为了满足某个测试准则而设计的测试用例在一起形成了测试用例集合.要进行测试用例集合价值的度量,首先必须分析决定和影响测试用例集合价值的关键要素.软件工程实践表明软件中越复杂的地方隐藏故障的概率就越高,测试必须给予这些部分以更多的重视,因此软件复杂性是测试用例集合价值模型的重要要素.同时,测试用例集合的执行结果也是重要的要素.

定义 1 测试准则(testing criterion):是关于一组有限、可枚举的待测试目标(待测试的软件成分)的判定规则,如果测试通过了判定规则的判定,则认为达到了测试准则,否则没有达到.本文称测试准则所定义的所有测试目标为测试目标向量.

定义 2 测试目标复杂性(testing unit complexity):反映测试目标向量中分量的复杂程度的软件特性.不同测试准则定义的测试目标向量在粒度上可能不同,在同一个测试目标向量中也可能有不同粒度的测试目标,不同粒度的测试目标往往需要不同的复杂性度量模型.如果测试目标是(1)软件结构性测试中的语句、分支、路径等,推荐使用 Halstead 软件科学度量、McCabe 圈复杂度等模型;(2)软件模块关系测试中的调用关系、类关系等,推荐使用方法(函数)、类的复杂性模型;(3)功能测试中的功能点,推荐使用功能复杂性度量模型,如扩展功能点[4].

设存在定义 1 的测试准则 C 和测试目标向量 $\boldsymbol{F}_C=(F_1 \quad F_2 \quad \cdots \quad F_k)$,以及满足 C 的一个测试用例集合,则该测试用例集合的价值 $\boldsymbol{V}_C$ 定义为:

$$\boldsymbol{V}_C=(\omega_1 \quad \omega_2 \quad \cdots \quad \omega_k)\times(e_1 \quad e_2 \quad \cdots \quad e_k)^T \tag{1}$$

其中$(\omega_1 \quad \omega_2 \quad \cdots \quad \omega_k)$是对应于测试目标向量的复杂度向量,$\omega_i$ 为 F_i 的复杂度,$(e_1 \quad e_2 \quad \cdots \quad e_k)$为使用该测试用例集合后的覆盖计数向量,$e_i$ 为对 F_i 的执行覆盖计数.

定义 3 测试密度(Test Density):是对测试目标向量中的一个分量或整个向量进行测试的力度(或强度)的度量.设 $\boldsymbol{D}_C=(\boldsymbol{D}_1 \quad \boldsymbol{D}_2 \quad \cdots \quad \boldsymbol{D}_k)$是对应于给定的测试目标向量的测试密度向量,$D_T$ 是测试用例集合对测试目标向量的平均测试密度,则 D_i 和 D_T 定义为:

$$D_i=\frac{e_i}{\omega_i}, \qquad D_T=\frac{1}{k}\sum_{i=1}^{k}D_i \tag{2}$$

定义 4 测试密度基线(test density baseline):一个可由用户制定的数值,如果某个测试目标的测试密度值低于该值,则认为对该目标的测试力度不够.合理使用测试密度基线可以加强测试准则.

定义 5 测试用例集合执行相似度:是对两个测试用例集合执行获得的测试密度向量的相似性的度量.设有两个测试用例集合,相应的有两个测试密度向量,则这两个测试用例集合执行相似度可以运用统计理论中的相关系数计算试用例集合,相应的有两个测试密度向量,则这两个测试用例集合执

收稿日期:2002-06-10;修回日期:2002-09-02
基金项目:国家自然科学基金(No.60073005)

行相似度可以运用统计理论中的相关系数计算(使用协方差)获得.相似度越接近 1.两个测试密度向量之间的相似性就越强.实践中应尽量减弱不同测试用例集合之间的相似度,以提高测试的有效性.

3 单个测试用例的价值

这里的单个测试用例来自于一个满足某个测试准则的测试用例集合.设有测试准则 C 定义了测试目标向量 $\boldsymbol{F}_C=(F_1\quad F_2\quad\cdots\quad F_k)$,则单个测试用例的价值 V_{stc} 定义为:

$$V_{stc}=(\omega_1\quad\omega_2\quad\cdots\quad\omega_k)\times(e_1\quad e_2\quad\cdots\quad e_k)^T \tag{3}$$

其中$(e_1\quad e_2\quad\cdots\quad e_k)$为使用该测试用例后的覆盖计数向量,其他定义同(1).

定义 6 测试用例贡献度 C_{stc}:测试用例集合中一个测试用例对达到测试准则的贡献程度.设某个测试用例获得的覆盖计数向量为$(e_1\quad e_2\quad\cdots\quad e_k)$,显然至少存在一个 $e_i\neq0$,$1\leq i\leq k$,$\mathrm{sgn}(x)$是符号函数,则

$$C_{stc}=\frac{1}{k}*\sum_{i=1}^{k}\mathrm{sgn}(e_i),\mathrm{sgn}(x)=\begin{cases}1, & x>0\\0, & x=0\end{cases} \tag{4}$$

测试用例贡献度可用来在回归测试中选择测试用例,也可用于评价测试用例设计人员的工作成效.

定义 7 逻辑语句块:单入口,且中间只有一条路径的语句段落,记为 $B(m,n)$,其中 m,n 为该语句段落的起始行号和结束行号.

定义 8 分支语句块:由一个条件判定语句的一个逻辑分支所控制的语句块,用 $BB(m,n)$表示,m,n 为该分支语句块的起始行号和结束行号;用 $BB_L(m)$来表示逻辑上的分支语句块(如 if 语句没有 *else* 分支).

4 模型度量实例

本节的度量针对一个用 C 语言实现的多项式分解函数,见附表 1.任意给定一个整数和一个基数(0 到 9 之间),该函数把整数分解为基数的多项式,且所有项的系数都为 1 或 - 1.

4.1 100%语句覆盖准则测试用例集合

由定义 7 得到测试目标向量(共有 25 个测试目标),逻辑语句块的复杂性度量使用 Halstead 的程序难度度量模型.针对 100%语句覆盖准则设计了两个不同的测试用例集合,TC_1 = ((16,3),(0,2),(8,0)),TC_2 = ((2, - 2),(- 7,3),(20,0),(0,19),(11,3)),得到的覆盖计数向量为 e_1 和 e_2,测试密度向量为 DC_1 和 DC_2,平均测试密度为 D_1 和 D_2,价值为 VC_1 和 VC_2,TC_1 和 TC_2 的执行相似度为 d_{12},数据如下所示.

ω	= (	1	1	1	1	1	1	2	1	2	1	1	1	1	1	3	1	2	3	1	3	1	3	2	2	1)
e_1	= (	3	1	2	1	1	4	4	4	2	2	1	1	4	7	7	4	1	3	4	4	2	1	2	4	1)
e_2	= (	5	1	4	1	3	9	9	9	6	6	4	2	9	13	13	9	2	7	6	6	7	1	6	9	3)
DC_1	= (	3	1	2	1	1	4	2	4	1	2	1	1	4	7	2.3	4	0.5	1	4	1.3	3	0.3	1	2	1)
DC_2	= (	5	1	4	1	3	9	4.5	9	3	6	4	2	9	13	4.3	9	1	2.3	6	2	7	0.3	3	4.5	1)

$VC_1=\omega\times e_1^T=114$ | $VC_2=\omega\times e_2^T=238$ | $d_{12}=0.95$

$D_{T1}=(\Sigma D_i)/25=2.18, D_i$ in $DC_1, 25\geq i\geq1$ | $D_{T2}=(\sum D_i)/25=4.56, D_i$ in $DC_2, 25\geq i\geq1$

第一组测试用例集合只考虑了正常情况的输入.考虑到数学上根本不可能对任何基数都做出如此分解,因此在第二组测试用例集合只用了基数为 2 和 3.不幸的是对于(2, - 2),算法不收敛,只能强行中止.但对于(- 7,3),算法却能得出"- 7 = - 9 + 3 - 1"的正确结论.度量结果说明测试用例集合 TC_2 比测试用例集合 TC_1 的价值要高.数据显示,两个测试用例集合都存在有一些密度分量低于 1.0.由此可知,TC_1 和 TC_2 在相应分量上的测试强度都不足.TC_1 和 TC_2 间的执行相似度为 0.95,这说明两个测试用例集合在测试结果上基本呈线性相关关系,有必要调整测试策略,如增加异常测试等.

4.2 100%分支覆盖准则测试用例集合

分支语句块的复杂度可以定义为该分支所控制的语句块的复杂度,可以采用 McCabe 圈复杂度来度量.由定义 8 可得到测试目标向量(共有 18 个测试目标).这里使用圈复杂度模型来度量分支语句块的复杂度.合理性起见,对该模型作一点小的适应性变动,如下所示:

$$\begin{aligned}CC(BB(m,n))&=k+1\\CC(BB_L(m))&=1\end{aligned} \tag{5}$$

其中 CC 为圈复杂度,k 为在语句块 $BB(m,n)$范围内的条件语句个数.实际 $CC(BB_L(m))$也应该等于 2,但由于这只是逻辑分支语句块,因此定义为 1.

针对测试目标向量设计了两个测试用例集合,TC_1 = ((41,3),(202,3),(30,0),(0,2)),TC_2 = ((- 37,2),(- 77,3),(30,0),(0,2)),覆盖计数向量为 e_1 和 e_2,测试密度向量为 DC_1 和 DC_2,平均测试密度为 D_1 和 D_2,价值度量分别为 VC_1 和 VC_2,TC_1 和 TC_2 的执行相似度为 d_{12},数据如下所示.

ω	= (	2	1	2	1	8	1	3	1	2	2	2	1	2	3	2	1	2	2	)
E_1	= (	1	3	1	2	11	4	3	8	2	1	31	11	2	9	22	6	6	3	)
E_2	= (	1	3	1	2	6	4	3	3	2	1	15	6	2	4	11	3	3	1	)
DC_1	= (	0.5	3	0.5	2	1.375	4	1	8	1	0.5	15.5	11	1	3	11	6	3	1.5	)
DC_2	= (	0.5	3	0.5	2	0.75	4	1	3	1	0.5	7.5	6	1	1.3	5.5	3	1.5	0.5	)

$VC_1=\omega\times e_1^T=295$ | $VC_2=\omega\times e_2^T=201$ | $d_{12}=0.9519$

$D_1=(\sum D_i)/18=4.104, D_i$ in $DC_1, 18\geq i\geq1$ | $D_2=(\sum D_i)/18=2.364, D_i$ in $DC_1, 18\geq i\geq1$

度量结果表明测试用例集合 T_{C1} 的价值比测试用例集合 T_{C2} 高.考察两个测试密度向量,有些分支语句块虽然在覆盖向量上获得了比较高的计数,但是在密度上却低于基线 1.0.密度较低指出了测试用例设计的改进方向.两个测试用例集合具有较高的执行相似度(0.9519),这说明应增加测试用例的多样性.

规模的原因,上面的两个度量实践还不足以说明价值度量的实践意义.但通过测试密度和相似度还是发现了一些常规无法发现的测试问题.实践中,真实测试的目标向量往往相当大,同时使用的测试用例也很多.如果不对测试的执行情况进行细致分析,就很难了解测试的不足和发现的问题.同时这也对测试用例集合价值的度量提出了工具支持的要求.就白盒测试而言,北京航空航天大学软件工程研究所研制的 Safepro 系列测试工具(目前支持 C/C++/VC++/Java 等多种语言版本)可以同时完成这个任务.

通过度量实例发现,测试用例集合给出了测试用例管理的操作方法.通过测试用例集合价值,测试密度和测试用例集合相似度的度量可以为测试建立一个基线.测试密度和测试用例集合执行相似度清晰地指出了一个测试用例集合在哪些方面比另一个测试用例集合好,哪些还不够,需要改进.

参考文献:

[1] Yuri Chernak. Validating and improving test-case effectiveness[J]. IEEE Software, 2001, 18(1-2): 81-86.

[2] J C Huang. Measuring the effectiveness of a test case, Application-specific software engineering technology[A]. Proc. of 1998 IEEE Workshop, ASSET-98[C]. 1998. 157-159.

[3] Von Mayrhauser, et al. Using a neural network to predict test caseeffectiveness[A]. Proceedings of 1995 IEEE Aerospace ApplicationsConference[C]. Snowmass: CO, 1995. 77-91.

[4] 吴际,汤铭端.扩展功能点[J].软件学报,2001,12(2):309-316.

作者简介:

吴 际 男,1973 年 6 月出生,安徽肥西人,博士研究生,主要研究领域为面向对象技术,软件工程,编译技术和自然语言处理技术.

金茂忠 男,1941 年 8 月生于上海,教授,博士生导师,主要研究领域包括软件工程,软件测试,编译技术.

附表 1 多项式分解源代码

源代码(1~35 行)		源代码(36~66 行)	
1	bool polynomial(int *integer*, int *base*)	36	if(*delta* = = 0)
2	{	37	{
3	int power[100];	38	power[*index*] = r;
4	char signs[100];	39	signs[*index*] = *sign*;
5	int *index* = 0;	40	}
6	int *r*, *left*, *delta*, *factor*;	41	else
7	char sign ;	42	{
8	if(*base* = = 0)	43	*delta* = *left* - (*factor* / *base*);
9	return false;	44	int *capacity* = 0;
10	if(*integer* = = 0)	45	for(int *i* = 0; *i* < r - 1; *i* + +)
11	return false;	46	{
12		47	*capacity* * = *base*;
13	*left* = integer;	48	*capacity* + = 1;
14	*sign* = ' + ';	49	}
15	for(; *left* ! = 0;)	50	if(*delta* < = *capacity*)
16	{	51	{
17	*delta* = left;	52	power[*index*] = *r* - 1;
18	*factor* = 1;	53	*factor* = *factor* / *base*;
19	*r* = 0;	54	signs[*index*] = *sign*;
20		55	}
21	if(*left* < 0)	56	else
22	{	57	{
23	*left* = 0 - *left*;	58	power[*index*] = *r*;
24	if(*sign* = = ' + ')	59	signs[*index*] = *sign*;
25	*sign* = ' - ';	60	}
26	else	61	}
27	*sign* = ' + ';	62	*index* + + ;
28	}	63	*left* - = *factor*;
29	*delta* = *left* - *factor*;	64	}
30	while(*delta* > 0)	65	return true;
31	{	66	}
32	*factor* = *factor* * *base*;		/* End of polynomial function */
33	*r* + + ;		
34	*delta* = *left* - *factor*;		
35	}		