软件质量保证与软件测试

--结构性测试回顾

主讲教师:

Email:

测试的效率

■什么时候测试可以停止?多少是足够的测试?

1. 当时间用光时 ——缺少标准
2. 当继续测试没有产生新失效时 } 基于经验的
3. 当继续测试没有发现新缺陷时 } 有效方法
4. 当无法考虑新测试用例时 ——原因?
5. 当回报很小时——基于分析的方法
6. 当达到所要求的覆盖时——结构化测试的指标
7. 当所有缺陷都已经清除时——难以实现

要使用哪些覆盖指标?

在业界实践中最常用的是DD- 路径

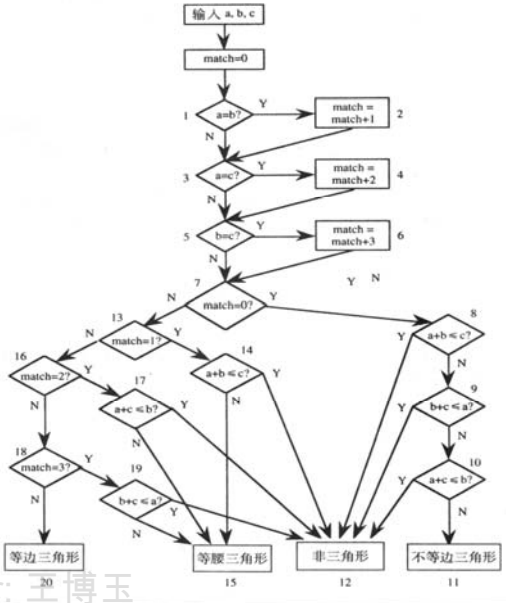
主要内容

■漏洞与冗余

■用于方法评估的指标

■重温案例研究

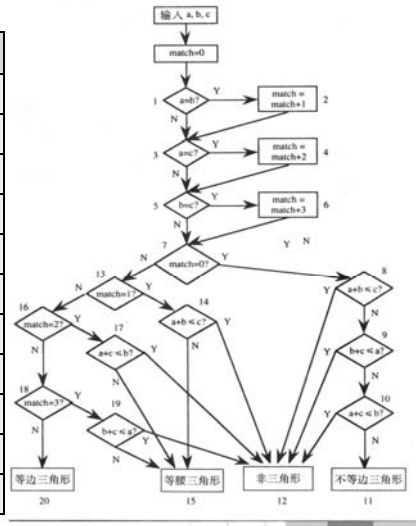
漏洞与冗余--三角形问题的传统实现



三角形程序中的路径

这个实现有11条可行路径

路径	节点序列	描述
P1	1-2-3-4-5-6-7-13-16-18-20	等边三角形
P2	1-3-5-6-7-13-16-18-19-15	等腰三角形(b=c)
P3	1-3-5-6-7-13-16-18-19-12	非三角形(b=c)
P4	1-3-4-5-7-13-16-17-15	等腰三角形(a=c)
P5	1-3-4-5-7-13-16-17-12	非三角形(a=c)
P6	1-2-3-5-7-13-14-15	等腰三角形(a=b)
P7	1-2-3-5-7-13-14-12	非三角形(a=b)
P8	1-3-5-7-8-12	非三角形(a+b ≤ c)
P9	1-3-5-7-8-9-12	非三角形(b+c ≤ a)
P10	1-3-5-7-8-9-10-12	非三角形(a+c ≤ b)
P11	1-3-5-7-8-9-10-11	不等边三角形



边界值的路径覆盖

测试用例	a	b	c	预期输出	路径
1	100	100	1	非三角形	p6
2	100	100	2	非三角形	p6
3	100	100	100	等边三角形	p1
4	100	100	199	等腰三角形	p6
5	100	100	200	非三角形	p7
6	100	1	100	非三角形	p4
7	100	2	100	非三角形	p4
8	100	100	100	等边三角形	p1
9	100	199	100	等腰三角形	p4
10	100	200	100	非三角形	p5
11	1	100	100	非三角形	p2
12	2	100	100	非三角形	p2
13	100	100	100	等边三角形	p1
14	199	100	100	等腰三角形	p2
15	200	100	100	非三角形	p3

路径	节点序列
P1	1-2-3-4-5-6-7-13-16-18-20
P2	1-3-5-6-7-13-16-18-19-15
P3	1-3-5-6-7-13-16-18-19-12
P4	1-3-4-5-7-13-16-17-15
P5	1-3-4-5-7-13-16-17-12
P6	1-2-3-5-7-13-14-15
P7	1-2-3-5-7-13-14-12
P8	1-3-5-7-8-12
P9	1-3-5-7-8-9-12
P10	1-3-5-7-8-9-10-12
P11	1-3-5-7-8-9-10-11

边界值测试和最坏情况测试对比

	P1	p2	p3	p4	p5	p6	p7	p8	p9	P10	P11
边界值（15个用例）	3	3	1	3	1	3	1	0	0	0	0
最坏情况（125个用例）	5	12	6	11	6	12	7	17	18	19	12

- 边界值测试有漏洞
- 最坏情况测试有严重冗余

主要内容

- 漏洞与冗余
- 用于方法评估的指标
- 重温案例研究

用于方法评估的指标

假设功能性测试技术M生成m个测试用例，并且根据标识被测单元中的s个元素的结构性测试指标S来跟踪这些测试用例。当执行m个测试用例时，会经过n个结构性测试元素。

- **定义 1** 方法M关于指标S的**覆盖**是n与s的比值n/s，记作C(M, S)
- **定义 2** 方法M关于指标S的**冗余**是m与s的比值m/s，记作R(M, S)
- **定义 3** 方法M关于指标S的**净冗余**是m与n的比值m/n， 记作NR(M, S)

用于方法评估的指标的解释

- 覆盖指标C(M, S)处理漏洞问题。如果这个值低于1，则说明该指标在覆盖上存在漏洞。如果C(M, S) = 1，则一定有R(M,S) = NR(M,S)
- 冗余性指标很明显，取值越大，冗余性越高
- 净冗余更有用，它指实际经过的元素，而不是要经过的总元素空间
- 将三种指标集合在一起，可给出一种评估功能性测试有效性方法(特殊值测试除外)关于结构性测试指标的定量方法
- **然而，我们实际需要的是要知道测试用例关于缺陷种类的有效性。但是，不能得到这类信息**
- **通过选择关于我们预期(或最担心)的缺陷种类的结构性测试指标，可以接近这个目标**

案例：三角形问题

	P1	p2	p3	p4	p5	p6	p7	p8	p9	P10	P11	∑TestCace
边界值	3	3	1	3	1	3	1	0	0	0	0	15
最坏情况	5	12	6	11	6	12	7	17	18	19	12	125

用于三角形程序的指标

方法	m	n	s	C(M, S) = n/s	R(M, S) = m/s	NR(M, S) = m/n
边界值	15	7	11	0.64	1.36	2.14
最坏情况	125	11	11	1.00	11.36	11.36
目标	s	s	s	1.00	1.00	1.00

案例：佣金问题

（DD路径数量为12）

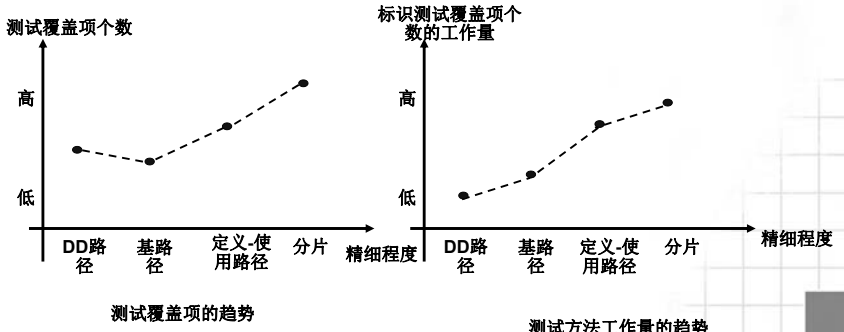
方法	m	n	s	C(M, S) = n/s	R(M, S) = m/s
输出边界值	25	12	12	1	2.08
决策表	3	12	12	1	0.25

输出边界值方法对于于 三个 指标

指标	m	n	s	C(M, S) = n/s	R(M, S) = m/s
DD路径	25	12	12	1	2.08
du-path	25	36	36	1	0.69
片	25	40	40	1	0.625

测试覆盖性数量的趋势线

■下图分别表示的是测试覆盖项(定义中的s)数量的趋势线，以及将覆盖项标识为结构性测试方法的函数的作用。我们不再满足于功能性测试方法的折衷考虑，因为这两张图说明选择合适结构性测试覆盖指标的重要性。



主要内容

■漏洞与冗余

■用于方法评估的指标

■重温案例研究

保险金案例代码（1）

```
Pseudo-code for the Insurance Premium Program
   Dim driver Age， points As Integer
   Dim base Rate， premium As Real
 1. Input (base Rate， driver Age， points)
 2. premium = 0
 3. Select Case driver Age
 4. Case 1: 16 ≤ driver Age < 20
 5.       age Multiplier = 2.8
 6.       If points < 1 Then
 7.                safe Driving Reduction = 50
 8.       End If
 9. Case 2: 20 ≤ driver Age < 25
10.       age Multiplier = 1.8
11        If points < 3 Then
12.               safe Driving Reduction = 50
13.       End If
```


保险金案例代码（2）

```
14. Case 3: 25 ≤ driver Age < 45
15.       age Multiplier = 1.0
16.       If points < 5 Then
17.                safe Driving Reduction = 100
18.       End If
19. Case 4: 45 ≤ driver Age < 60
20.       age Multiplier = 0.8
21        If points < 7 Then
22.                safe Driving Reduction = 150
23.       End If
24. Case 5: 60 ≤ driver Age < 100
25.       age Multiplier = 1.5
26.       If points < 5 Then
27.                safe Driving Reduction = 200
28.       End If
29. Case 6: Else
30.       Output ("Driver age out of range")
31. End Select
32. premium = base Rate * age Multiplier – safe Driving Reduction
33. Output (premium)
```

保险金案例的程序图

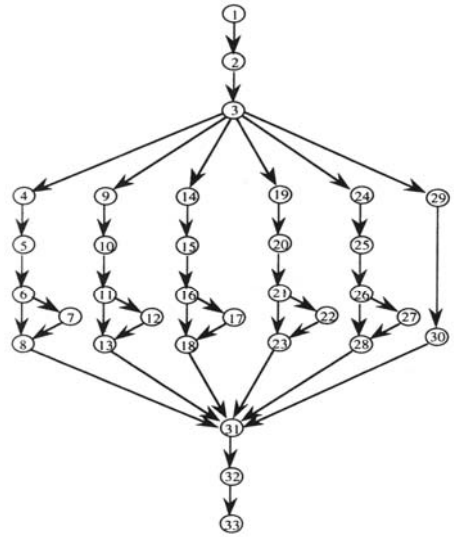


图11-4 保险金程序的程序图

保险金程序中的路径

■ 保险金程序程序图的复杂度是V(G) =11，有11条可行程序执行路径，如表所示

路径	节点序列
P1	1 - 2 - 3 - 4 - 5 - 6 - 8 - 31 - 32 - 33
P2	1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 31 - 32 - 33
P3	1 - 2 - 3 - 9 - 10 – 11 - 13 - 31 - 32 - 33
P4	1 - 2 - 3 - 9 - 10- 11 - 12 - 13 - 31 - 32 - 33
P5	1 - 2 - 3 - 14 - 15 - 16 - 18 - 31 - 32 - 33
P6	1 - 2 - 3 - 14 - 15 - 16 - 17 - 18 - 31 - 32 - 33
P7	1 - 2 - 3 - 19 - 20 - 21 - 23 - 31 - 32 - 33
P8	1 - 2 - 3 - 19 - 20 - 21 - 22 - 23 - 31 - 32 - 33
P9	1 - 2 - 3 - 24 - 25 - 26 - 28 - 31 - 32 - 33
P10	1 - 2 - 3 - 24 - 25 - 26 - 27 - 28 - 31 - 32 - 33
P11	1 - 2 - 3 - 29 - 30 - 31 - 32 - 33

保险金程序中的功能性测试用例

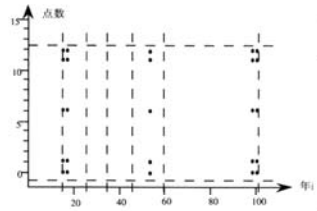


图8-7 保险金计算程序的最坏情况边界值测试

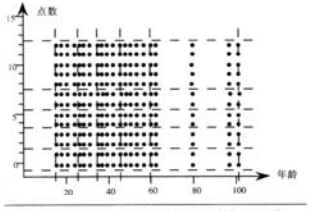


图8-8 保险金计算程序的详细最坏情况边界值测试用例

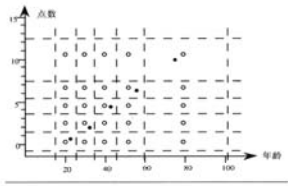


图8-9 保险金计算程序的弱和强等价类测试用例

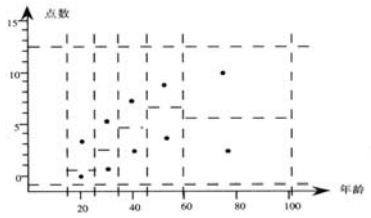


图8-10 保险金计算程序的决策表测试用例

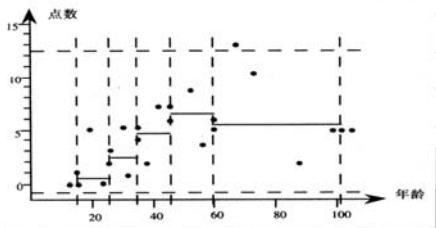


图8-11 保险金计算程序的最终（混合）测试用例

功能性测试方法路径覆盖

■ 如果花一些时间研究第8章各种功能性测试用例集合（上一页图），就会得到如表所示的结果

图号	方法	测试用例	所覆盖的路径
8-7	边界值	25	p1, p2, p7, p8, p9, p10
8-8	最坏情况边界值	273	p1, p2, p3, p4, p5, p6, p7, p8, p9, p10
8-9	弱等价类	5	p2, p4, p6, p8, p9
8-9	强等价类	25	p1, p2, p3, p4, p5, p6, p7, p8, p9, p10
8-10	决策表	10	p1, p2, p3, p4, p5, p6, p7, p8, p9, p10
8-11	混合	25	p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11

■其中的漏洞和冗余问题很明显，只有通过混合方法得到的测试用例才能够产生完全的路径覆盖

练习

- 针对保险金程序，给出：
  - DD-路径
  - 定义-使用路径
  - 片

保险金案例 – 结构性测试

■基于路径的测试

由于程序图是无环路的，因此只存在有限条路径，对于这个例子是11。最佳选择是保留 执行每条路径的测试用例，自然会达到语句和DD-路径覆盖要求。复合条件谓词意味着多条件覆盖，这只能通过最坏情况边界值测试用例和混合测试用例实现，不能使用其他基于路径的覆盖指标。

■数据流测试

这个问题的数据流测试很枯燥， driver Age、points和safe Driving Reduction变量都出现在六个定义清除的定义-使用路径中。driver Age和points的“使用”都是谓词使用。第10章曾经提到过，全路径准则意味着全低层数据流覆盖。

保险金案例—结构性测试

■片测试

片测试也没有提供多少启发。有四个有意思的片(没有列出End If):

S (safe Driving Reduction, 32) = {1,3,4,6,7,9,11,12,14,16,17,19,21,22,24,26,27,31}

S (age Multiplier, 32) = {1, 3, 4, 5, 9, 10, 14, 15, 19, 20, 24, 25, 31}

S (base Rate, 32) = {1}

S (Premium, 31) = {2}

这些片的并(加上End If语句)是整个程序。通过基于片的测试得到的仅有启发，如果失效发生在第32行， safe Driving Reduction和age Multiplier上的片将程序分解为两个不相交的片，这会简化缺陷隔离工作。

总结

■结构性测试都有哪些方法？

■有哪些评估指标？

讨论

- 除了结构化的覆盖率指标外,还有其它的覆盖率指标吗?
  - 需求覆盖率：需求覆盖率=至少被测试用例覆盖一次的需求数/系统总需求数
  - 测试用例覆盖率：测试用例覆盖率=计划执行的测试用例数/测试用例总数
  - 测试用例执行率： 测试用例执行率=实际执行的测试用例数/计划执行的测试用例数
  - 测试用例通过率：测试用例通过率＝(实际执行的测试用例数-测试执行不通过的测试用例数)/实际执行的测试用例数
  - 缺陷修正率：缺陷修正率＝发布前已修正的缺陷数/发布前已知的缺陷总数
  - 等等