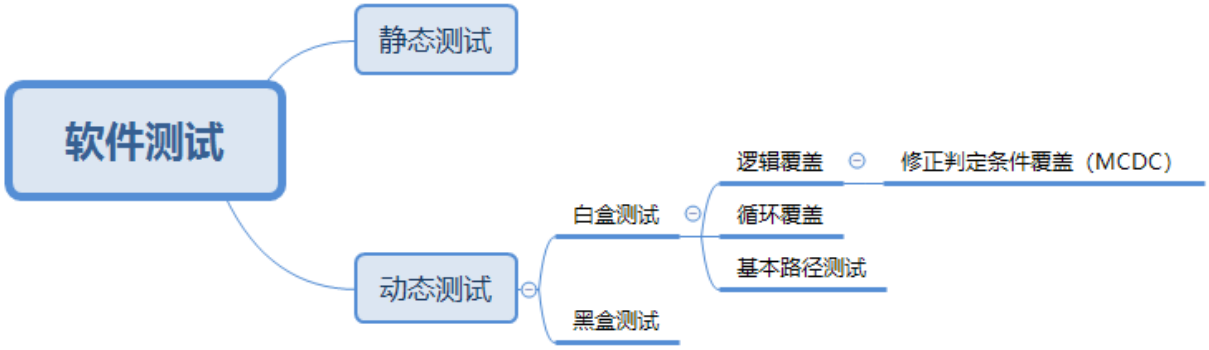
# ChenM666

软件测试-修正判定条件覆盖（MCDC）的一些认识




修正判定条件覆盖(Modified Condition/Decision Coverage即MC/DC)

通过参照上图，我们首先从宏观的角度上理解MCDC在软件测试中的分支。软件测试软件工程开发中中必不可少且非常重要的一部分，软件测试从是否执行代码来看可分为两种测试方法：静态测试和动态测试．静态测试是指不用执行程序的测试，它主要采取方案——代码走查、技术评审、代码审查的方法对软件产品进行测试，通俗来说就是“看”代码；而动态测试是指实际运行程序，并通过观察程序运行的实际结果来发现错误的软件测试技术，从是否关心内部结构来看它分为黑盒测试和白盒测试．黑盒测试是在不知道程序内部结构，只知道程序规格的情况下采用的测试技术或策略；白盒测试是一种在知道程序内部结构的情况下采用的测试技术或策略，就是要选取足够的测试用例，对源代码实现比较充分的覆盖，以便尽可能多地发现程序中的错误．它包括逻辑覆盖法、路径测试法、循环覆盖．MC/DC就是白盒测试方法中的逻辑覆盖中的方法．MC/DC是欧美的航空/航天制造厂商和使用单位联合制定的“航空运输和装备系统软件认证标准”，目前在国防和航空航天领域应用广泛。

MC/DC是DO-178B Level A认证标准中规定的，欧美民用航空器强制要求遵守该标准。

MC/DC定义如下：
Condition —— a Boolean expression containing no Boolean operators：
Decision —— a Boolean expression composed of conditions and zero
Modified Condition/Decision Coverage —— every point of entry and
oked at least once，
every condition in the program has taken all possible outcomes at least once，and each condition in a decision has been shown to independently affect a decision S outcome by varying just that condition while holding fixed all other possible conditions.




< 2024年12月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | 31 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |

**导航**
博客园
首页
新随笔
联系
订阅 XML
管理

条件表示不含有布尔操作符号的布尔表达式；

判定表示由条件和零或者很多布尔操作符号所组成的一个布尔表达式；

修正条件判定覆盖要求在一个程序中每一种输入输出至少得出现一次，在程序中的每一个条件必须产生所有可能的输出结果至少一次，并且每一个判定中的每一个条件必须能够独立影响一个判定的输出，即在其他条件不变的前提下仅改变这个条件的值，而使判定结果改变。

定义显得有些晦涩难懂的，我们可以换一个角度来解读这句话：

MC/DC首先要求实现条件覆盖、判定覆盖，在此基础上，对于每一个条件C，要求存在符合以下条件的两次计算：

1）条件C所在判定内的所有条件，除条件C外，其他条件的取值完全相同；

类似于控制变量；

2）条件C的取值相反；

3）判定的计算结果相反。

核心意思是每个条件都要独立影响判定结果。为什么说“两次计算”，而不是“两个用例”呢？当循环中有判定时，一个用例下同一判定可能被计算多次，每次的条件值和判定值也可能不同，因此，一个用例就可能完成循环中判定的MC/DC。

MC/DC是条件组合覆盖的子集。条件组合覆盖要求覆盖判定中所有条件取值的所有可能组合，需要大量的测试用例，实用性较差。MC/DC具有条件组合覆盖的优势，同时大幅减少用例数。满足MC/DC的用例数下界为条件数+1，上界为条件数的两倍，例如，判定中有两个条件，条件组合需要4个用图表1，

表1

| 用例 | A | B | A&&B |
|---|---|---|---|
| 1 | T | T | T |
| 2 | T | F | F |
| 3 | F | T | F |
| 4 | F | F | F |

而MC/DC只需要3个图表2。



随笔档案
2022年4月(2)

阅读排行榜
1. 软件测试-修正判定条件覆盖（MCDC）的一些认识(12496)
2. C语言写学生信息管理系统（文件版）(543)

[illegible]行榜
[illegible]测试-修正判定条件覆盖（MCDC）的一些认识(1)

推荐排行榜
1. 软件测试-修正判定条件覆盖（MCDC）的一些认识(3)

最新评论
1. Re:软件测试-修正判定条件覆盖（MCDC）的一些认识

还是用逻辑流程图理解好一点，每条导向最终结果的路径都有对应用例的时候就全覆盖了

--一只泥达耶

表2

| 用例 | A | B | A&&B |
| --- | --- | --- | --- |
| 1 | T | T | T |
| 2 | T | F | F |
| 3 | F | T | F |

对于条件A，用例1和用例3，A取值相反，B相同，判定结果分别为1和0；
对于条件B，用例1和用例2，B取值相反，A相同，判定结果分别为1和0；

判定中有三个条件，条件组合覆盖需要8个用例，而MC/DC需要的用例数为4至6个。如果判定中条件很多，用例数的差别将非常大，例如，判定中有10个条件，条件组合覆盖需要1024个用例，而MC/DC只需要11至20个用例。

那么我们该如何写出测试集

思路：逻辑与表达式测试所有条件为真的的情况，然后分别测试每个条件为假，其他条件都为真的情况。逻辑或表达式测试所有条件为假的情况。然后分别测试每个条件为真的情况。

MC/DC效果与多重条件覆盖效果相同，但是用例数确明显减少。

MCDC实例

A||（B&&C）

首先第一层：对于逻辑或表达式，所有条件为假，然后测试每个条件为真

即A为F，（B&&C）为F

A为T，（B&&C）为F

A为F，（B&&C）为T

第二层：逻辑与表达式测试所有条件为真的的情况，然后分别测试每个条件为假，其他条件都为真的情况

即B为T，C为T

B为F，C为T

B为T，C为F

| 用例 | A | B | C | （B&&C） | A||（B&&C） |
|---|---|---|---|---|---|
| 1 | F | F | T | F | F |
| 2 | T | F | T | F | T |
| 3 | F | T | T | T | T |
| 4 | F | T | F | F | F |

对于条件A用例1和2，A取反，B、C相同，结果取反

对于条件A用例1和3，B取反，A、C相同，结果取反

此时我们会发现还差一个C取反，A、B相同的案例，由于前三个案例C全为T，故此时C为F，于是B只能为T即，3、4做对比，A为F。由此我们便写出MC\DC的测试用例

依照上述我们再进行一个测试用例

（A||B）&&（C||D）

首先第一层：

| 用例 | （A||B） | （C||D） | （A||B）&&（C||D） |
|---|---|---|---|
| 1 | T | T | T |
| 2 | T | F | F |
| 3 | F | T | F |



A||B为F

| 用例 | A | B | A\|\|B |
|---|---|---|---|
| 1 | F | F | F |
| 2 | F | T | T |
| 3 | T | F | T |

C||D为F

| 用例 | C | D | C\|\|D |
|---|---|---|---|
| 1 | F | F | F |
| 2 | F | T | T |
| 3 | T | F | T |

| 用例 | A | B | A\|\|B | C | D | C\|\|D | (A\|\|B) &&(C\|\|D) |
|---|---|---|---|---|---|---|---|
| 1 | F | T | T | F | T | T | T |
| 2 | F | T | T | F | F | F | F |
| 3 | F | F | F | F | T | T | F |
| 4 | T | F | T | F | T | T | T |
| 5 | F | T | T | T | F | T | T |

当我们写完1、2、3案例，自然就可以写出A和C，而且这个用例组不唯一

| 用例 | A | B | A\|\|B | C | D | C\|\|D | (A\|\|B) &&(C\|\|D) |
|---|---|---|---|---|---|---|---|
| 1 | T | F | T | T | F | T | T |
| 2 | T | F | T | F | F | F | F |
| 3 | F | F | F | T | F | T | F |
| 4 | F | T | T | T | F | T | T |
| 5 | T | F | T | F | T | T | T |

第二种方法

（1）我们已经知道就最简布尔表达式:逻辑与表达式测试所有条件为真的的情况，然后分别测试每个条件为假，其他条件都为真的情况。逻辑或表达式测试所有条件为假的情况。然后分别测试每个条件为真的情况。

(A||B)&&(C&&D)可以等价看成X&&Y，其中X为(A||B）Y为(C&&D)

（2）针对第一个条件，设计直接作用到结果的第一组测试用例，再对其他条件取值的时候不断使用关于“AND”和“OR”最简布尔表达式的测试用例设计思路；

（3）每个条件应取到所有可能的值；

（4）针对每个条件，在前边已经测试好的测试用例中，选取该条件直接作用到结果的测试用例为参照对象，改变该条件的取值，同时保持其他条件的的值不变。

（5）按照这种方式的获得的最小测试用例组不是唯一的。

| 用例 | （A\|\|B） | （C\|\|D） | （A\|\|B）&&（C\|\|D） |
|---|---|---|---|
| 1 | T | T | T |
| 2 | T | F | F |
| 3 | F | T | F |

A||B为F

| 用例 | A | B | A\|\|B |
|---|---|---|---|
| 1 | F | F | |

| | | | |
|---|---|---|---|
| 2 | F | T | T |
| 3 | T | F | T |

C||D为F

| 用例 | C | D | C\|\|D |
|---|---|---|---|
| 1 | F | F | F |
| 2 | F | T | T |
| 3 | T | F | T |

| 用例 | A | B | A\|\|B | C | D | C\|\|D | (A\|\|B) &&(C\|\|D) |
|---|---|---|---|---|---|---|---|
| 1 | T | F | T | T | F | T | T |
| 2 | F | F | F | T | F | T | F |
| 3 | F | T | T | T | F | T | T |
| 4 | F | T | T | F | F | F | F |
| 5 | F | T | T | F | T | T | T |

这样我们也可以找到最小测试用例，而且更加条理。

以上便是我通过借鉴网络各个博主以及我自己的一些理解与总结，可能难免出现错误，欢迎私信批评指正，如有侵占，联系后我会及时删除。

2022-04-12 22:24:21



分类：软件测试

Chen小M
粉丝 - 3 关注 - 3
+加关注



» 下一篇： C语言写学生信息管理系统（文件版）

posted on 2022-04-12 22:09 Chen小M 阅读(12497) 评论(1) 编辑 收藏 举报

**阅读排行：**

- 2000 Star，是时候为我的开源项目更新下功能了
- 好消息，在 Visual Studio 中可以免费使用 GitHub Copilot 了！
- 工作中这样用MQ，很香！
- 基于.NET WinForm开发的一款硬件及协议通讯工具
- 使用 .NET Core 实现一个自定义日志记录器