# 软件代码度量-圈复杂度

# 一、历史及定义

**圈复杂度（Cyclomatic complexity）**也称为条件复杂度或循环复杂度，是一种软件度量，是由Thomas J. McCabe, Sr. 在 1976 年提出，用来表示程序的复杂度，其符号为 VG 或是 M。圈复杂度是对源代码中线性独立路径数的定量测量。

**圈复杂度**度量的是程序中线性独立路径的数量；例如：如果程序中不包含控制、判断、条件语句（例如 if，swith 等），那么复杂度就是 1 ；因为整个程序只有一条执行路径；如果程序包含一条IF语句，那么就会有两条路径来执行完整个程序（IF为 TRUE，IF 为 FALSE），所以这时候的复杂度就是 2；两个嵌套的 IF 语句，或者包含两个判断条件的一个 IF 语句，复杂度就是 4；

# 二、度量标准

麦凯布提出圈复杂度时，其原始目的之一就是希望在软件开发过程中就限制其复杂度。他建议程序设计者需计算其开发模块的复杂度，若模块的圈复杂度超过 10，需再分区为更小的模块。NIST（国家标准技术研究所）的结构化测试方法论此作法略作调整，在一些特定情形下，模块圈复杂度上限放宽到 15 会比较合适。此方法论也承认有些特殊情形下，模块的复杂度需要超过上述的上限，其建议为“模块的圈复杂度需在上限范围以内，否则需提供书面数据，说明为何此模块圈复杂度有必要超过上限。”

# 三、计算方法

在开发中常用的检测圈复杂度的工具，Source Monitor, check style都可以检测到高复杂度的代码块。在代码的开发中，配合各种圈复杂度的检测插件，将高复杂度的代码进行适当的拆分、优化，可以大大提高代码整体的质量，减少潜在bug存在。

Source Monitor的圈复杂度(cyclomatic complexity)计算方法：

1 接近Code Complete（代码大全，Steve-McConnell）的方法

在代码大全Table19-2中，是这样描述的，

(1) Start with 1 for the straight path through the routine.

(2) 遇到关键词或其等价，则加1，if while repeat for and or

(3) 每遇到1个case语句就增加1.

2 复杂度是指执行路径的数量。

3 遇到分支则复杂度加1，分支包括 if else foreach while

4 算术性的if语句复杂度加1，例如MyBoolean?IfTrue:IfFalse

5 在if for while或类似逻辑语句中出现一次&&或者||，复杂度加1

6 switch语句每遇到一次从case来的退出就加1.这些退出包括break goto return throw continue或者类似语句。对于default语句即使没有复杂度也要加1。

7 在一个try块中的每个catch或者except复杂度加1，而非try或finally语句

# 四、举例说明：

# 下面这个实例中，单元测试的覆盖率可以达到100%，但是很容易发现这其中已经漏掉了一个NPE的测试用例。case1方法的圈复杂度为2，因此至少需要2个用例才能完全覆盖到其所有的可能情况。

|  |
| --- |
| //程序原代码，圈复杂度为 2  **public** String case1(**int** num) {         String string = **null**;  **if** (num == 1) {             string = "String";         }  **return** string.substring(0);      }   //上面代码的单元测试代码  **public void** testCase1(){         String *test1*=*case1*(1);      } |

圈复杂度主要与分支语句（if、else、，[switch](https://www.baidu.com/s?wd=switch&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd) 等）的个数成正相关。可以在图1中看到常用到的几种语句的控制流图（表示程序执行流程的有向图）。当一段代码中含有较多的分支语句，其逻辑复杂程度就会增加。在计算圈复杂度时，可以通过程序控制流图方便的计算出来。通常使用的计算公式是V(G) = e – n + 2 , e 代表在控制流图中的边的数量（对应代码中顺序结构的部分），n 代表在控制流图中的节点数量，包括起点和终点（1、所有终点只计算一次，即便有多个return或者throw；2、节点对应代码中的分支语句）。

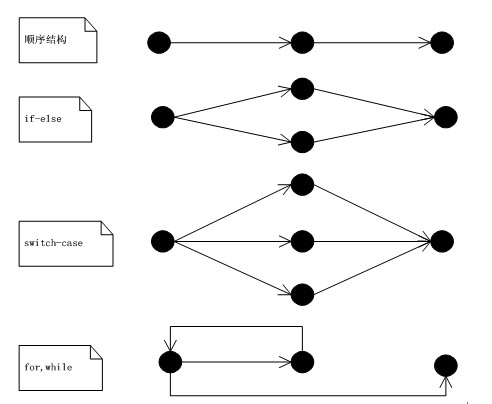


图1、各判断语句的控制流图

知道了如何计算圈复杂度，我们来使用控制流图重新计算一次case1方法的圈复杂度，其控制流图如下图。状态1表示if（num == 1 ）的条件判断，状态2表示string=”String”的赋值操作。可以通过下面的控制流图得到 e = 3 ; n = 3;那么全复杂度V(G) = 3 - 3 + 2 = 2,既case1的圈复杂度为2。

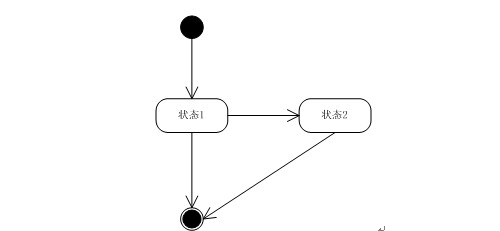


图2、case1的控制流图

在看一个计算全复杂度的例子。程序代码如下：

|  |
| --- |
| **public** String case2(**int** index, String string) {         String returnString = **null**;  **if** (index < 0) {  **throw new** IndexOutOfBoundsException("exception <0 ");         }  **if** (index == 1) {  **if** (string.length() < 2) {  **return** string;             }             returnString = "returnString1";         } **else if** (index == 2) {  **if** (string.length() < 5) {  **return** string;             }             returnString = "returnString2";         } **else** {  **throw new** IndexOutOfBoundsException("exception >2 ");         }  **return** returnString;      } |

程序控制流图：

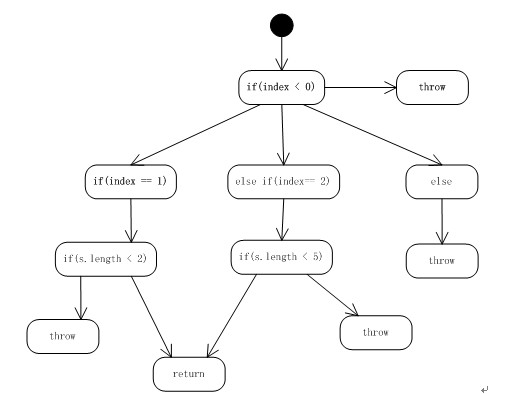
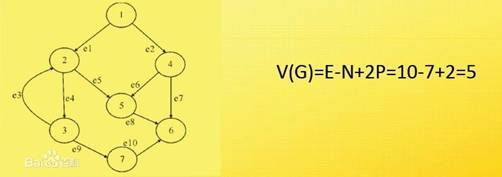


图3、case2的控制流图

根据公式 V(G) = e – n + 2 = 12 – 8 + 2 = 6 。case2的圈复杂段为6。说明一下为什么n = 8，虽然图上的真正节点有12个，但是其中有5个节点为throw、return，这样的节点为end节点，只能记做一个。

计算公式1：V(G)=E-N+2P。其中，E表示控制流图中边的数量，N表示控制流图中节点的数量，P图的连接组件数目（图的组件数是相连节点的最大集合）。因为控制流图都是连通的，所以P为1.



| **圈复杂度** | **代码状况** | **可测性** | **维护成本** |
| --- | --- | --- | --- |
| 1-10 | 清晰、结构化 | 高 | 低 |
| 11-20 | 复杂 | 中 | 中 |
| 21-30 | 非常复杂 | 低 | 高 |
| >30 | 不可读 | 不可测 | 非常高 |