Cyclomatic Complexity

Presented by: SUDIP SAMANTA

Contents

What is Software Metric?

Measurement is nothing but quantitative indication of size / dimension / capacity of an attribute of a product / process.

A **Metric** is a measurement of the degree that any attribute belongs to a system ,product or process.

Software metric is defined as a quantitative measure of an attribute a software system possesses with respect to Cost, Quality, Size and Schedule.

What is Cyclomatic Complexity?

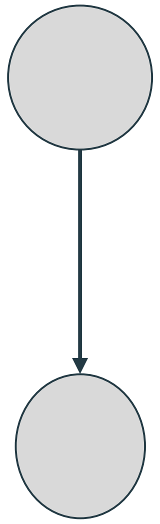
CYCLOMATIC COMPLEXITY is a software metric used to measure the complexity of a program. It is a quantitative measure of independent paths in the source code of the program.

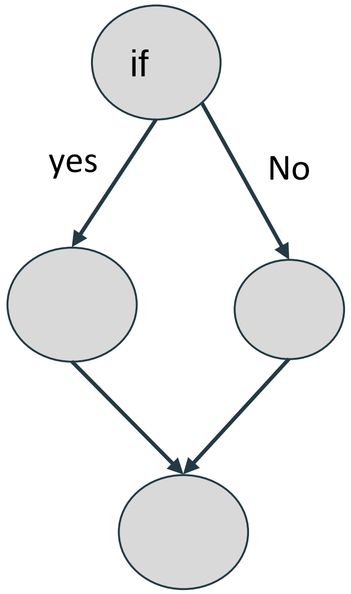
Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths.

1-> 2-> 6-> 1-> 7   1-> 2 -> 3 -> 4 -> 5 -> 2-> 6-> 1-> 7

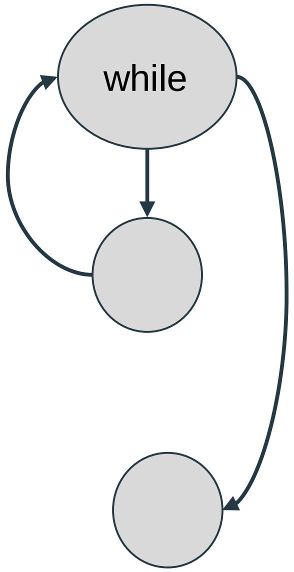
This Cyclomatic Complexity was developed by Thomas J. McCabe in 1976 .Cyclomatic Complexity is computed using control flow graph of the program.

Flow graph notation for a program:

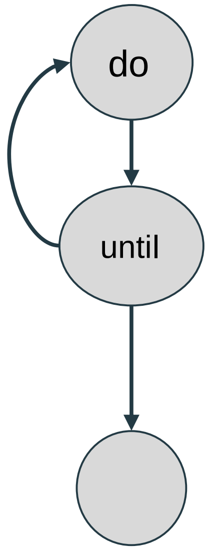

Sequence



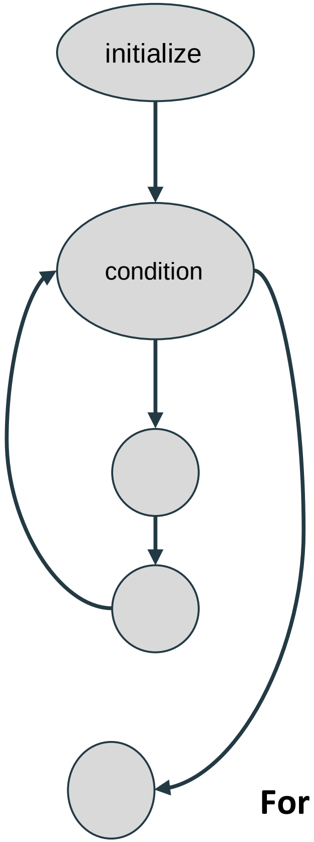
If-else



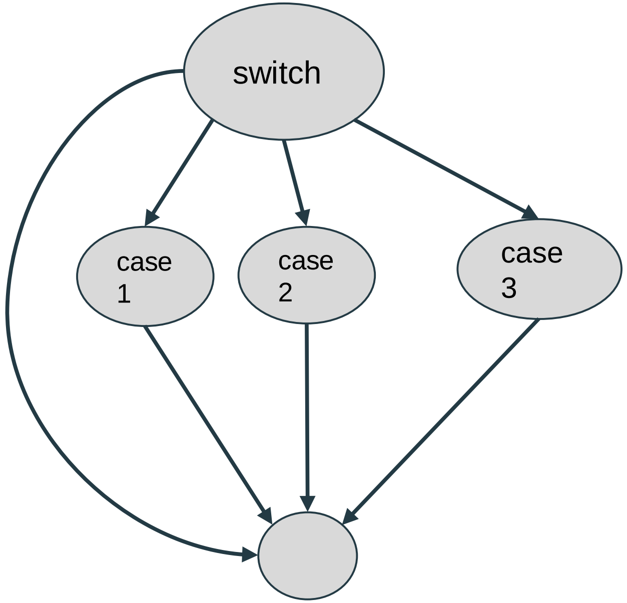
While loop



Until loop



For loop



Switch case

Formula for Calculating Cyclomatic Complexity-

Cyclomatic complexity is calculated using the control flow representation of the program code.
There are 3 commonly used methods for calculating the cyclomatic complexity-

<u>Method 1</u>:-

Cyclomatic Complexity = E – N + 2*C
E = Number of edges   N =Number of Nodes C= the number of connected components

<u>Method 2</u>:-

Cyclomatic Complexity = P+1
P=Number of predicate nodes (node that contains condition)

<u>Method 3</u>:-

Cyclomatic Complexity = Total number of closed regions in the control flow graph + 1

```
int func(int x; int y){
1.      while(x != y){
2.          If (x>y) then
3.              x=x-y;
4.          else y=y-x;
5.          }
6.   Return x; }
```

Calculation:-

Method 1:-

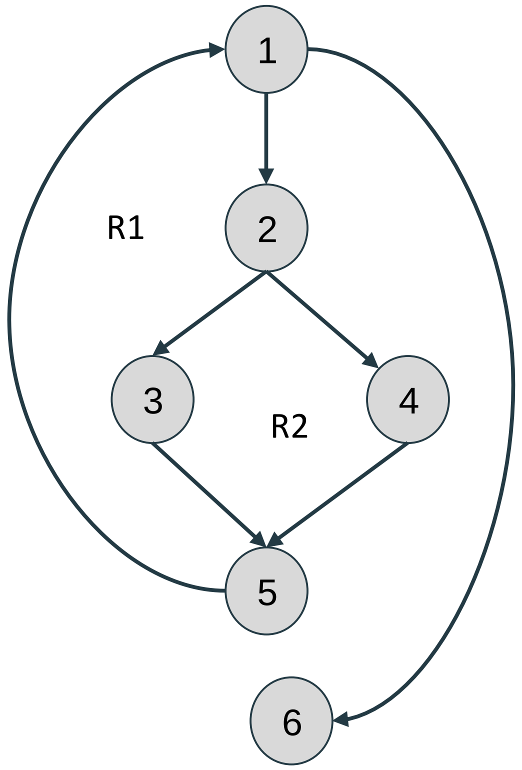
Cyclomatic Complexity = 7 - 6+ 2 * 1
=3

Method 2 :-

Cyclomatic Complexity= 2+1 =3

Method 3 :-

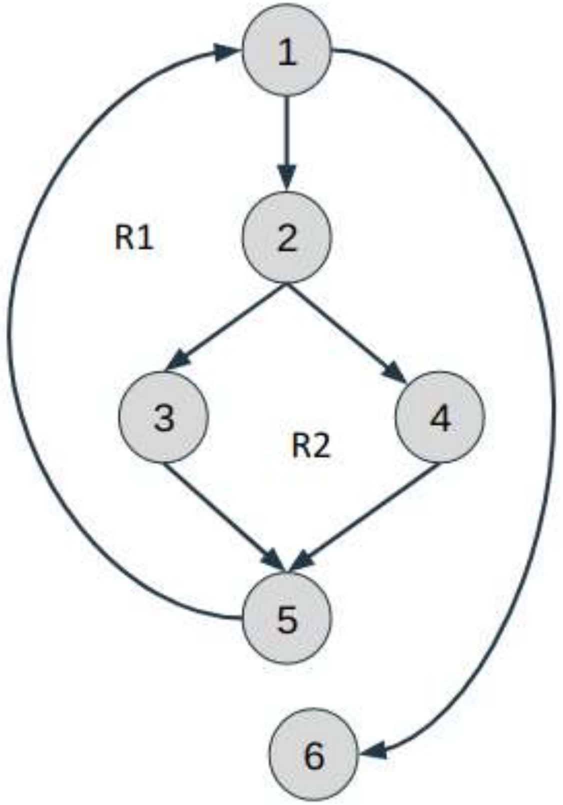
Cyclomatic Complexity= 2+1 =3



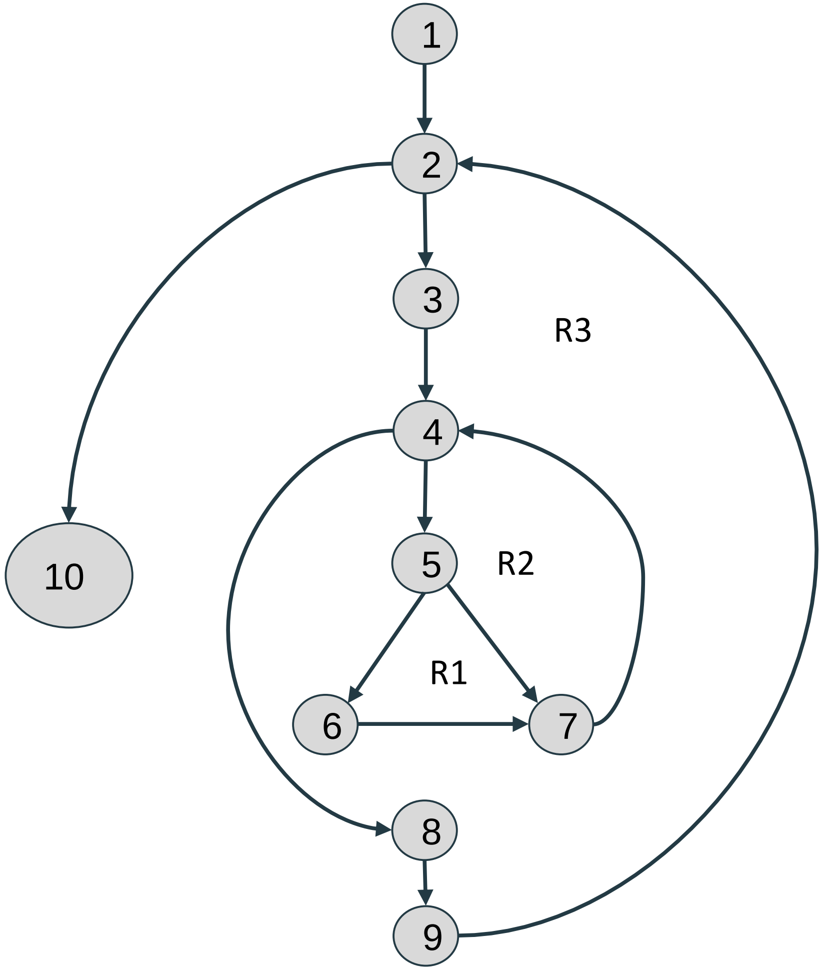
Number of independent paths:

- 1,2,3,5,1,6
- 1,2,4,5,1,6
- 1,6

Total number of independent paths = 3



```
1. i=0,n=4;
2. while (i<n-1) do
3.   j = i + 1;
4.   while (j<n) do
5.     if A[i]<A[j] then
6.        swap(A[i], A[j]);
7.   end do;
8.   i=i+1;
9. end do;
10. Exit
```



<u>Calculation</u>:-

<u>Method 1</u>:-

Cyclomatic Complexity = 12 - 10 + 2 * 1
=4

<u>Method 2</u> :-

Cyclomatic Complexity= 3+1 =4

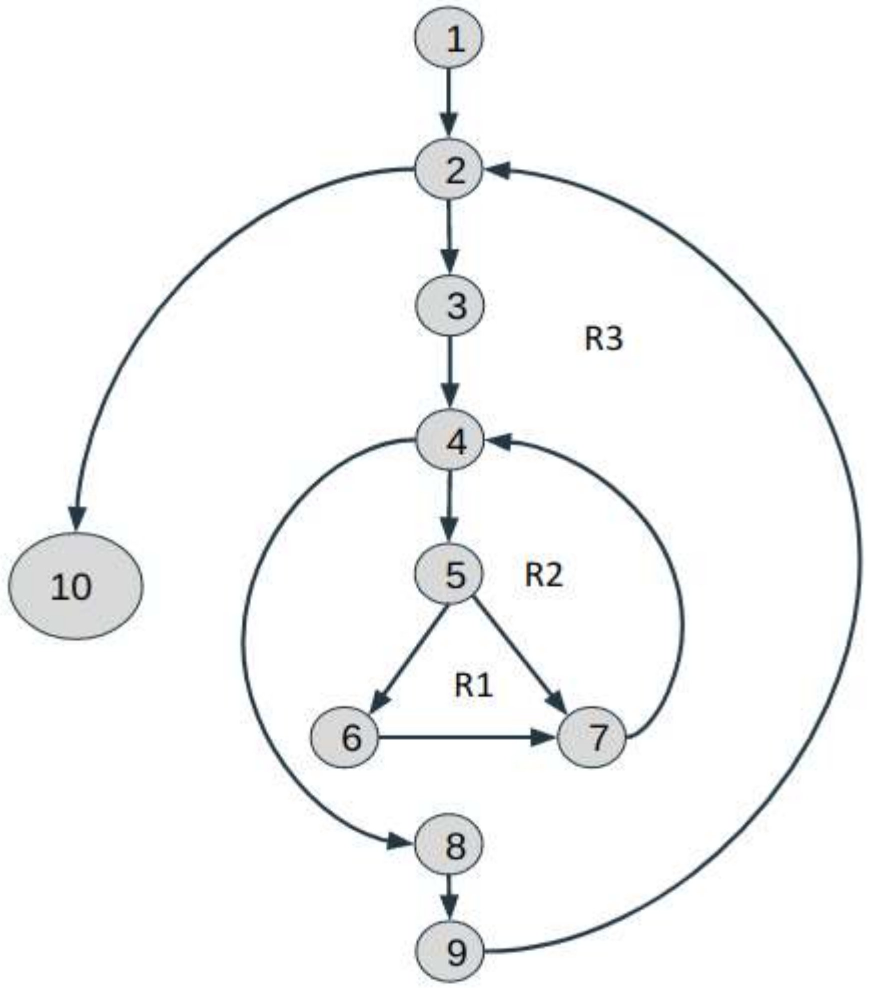
<u>Method 3</u>:- Cyclomatic Complexity= 3+1 =4

Number of independent paths:

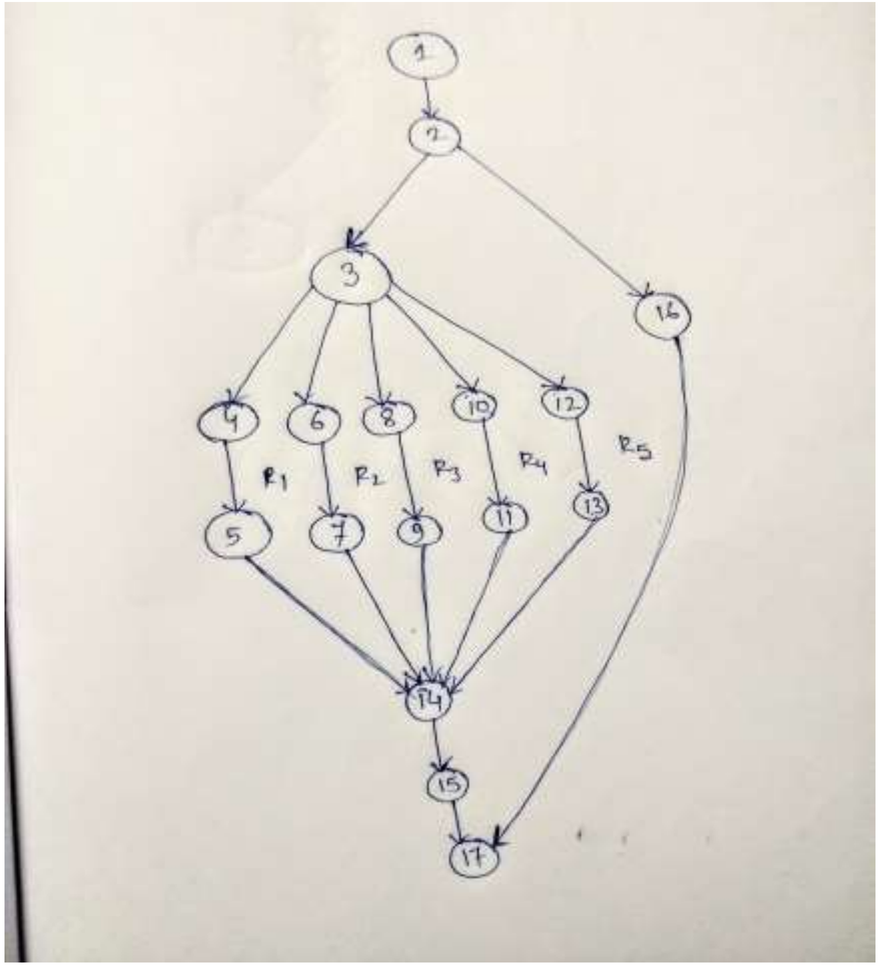
- 1,2,3,4,8,9,2,10
- 1,2,3,4,5,6,7,4,8,9,2,10
- 1,2,3,4,5,7,4,8,9,2,10
- 1,2,10

Total number of independent paths

=4



```
1.  Int n;
2.  if (n >= 0) {
3.     switch(n) {
4.        case 0:
5.             printf("zero \n"); break;
6.         case 1:
7.             printf("one\n"); break;
8.         case 2:
9.              printf("two\n"); break;
10.        case 3:
11.             printf("three or four\n"); break;
12.        Default:
13.             printf("none"); break;
14.       }
15.  }
16. else printf("three or four\n");
17. End
```



Calculation:-

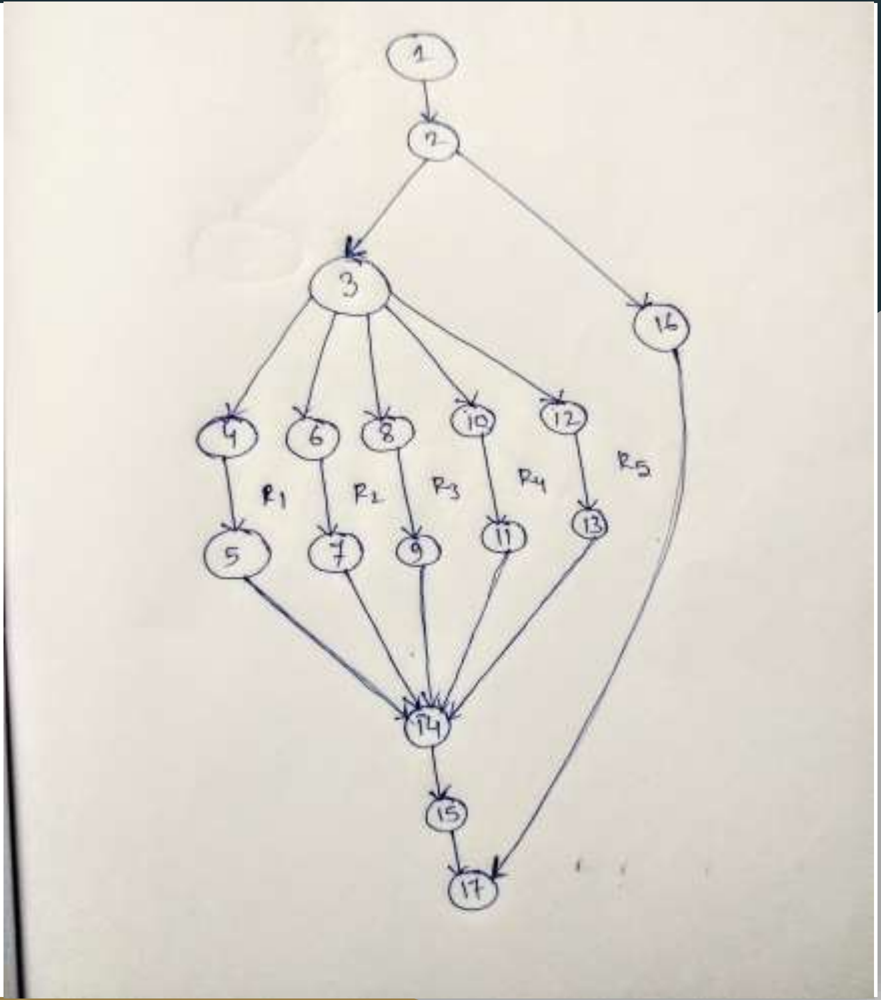
Method 1:-

Cyclomatic Complexity = 21 - 17 + 2 * 1
=6

Method 2 :-

Cyclomatic Complexity= 5+1 =6

Method 3:- Cyclomatic Complexity= 5+1 =6



Calculation:-

Method 1:-

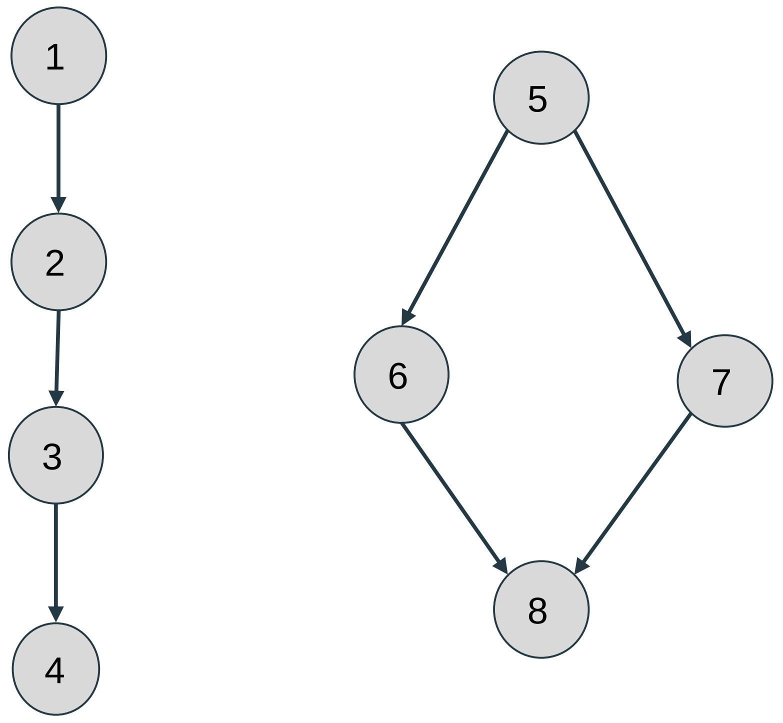
Cyclomatic Complexity = 7 - 8 + 2 * 2
=3

Method 2 :-

Cyclomatic Complexity= 1+(1+1)
=3

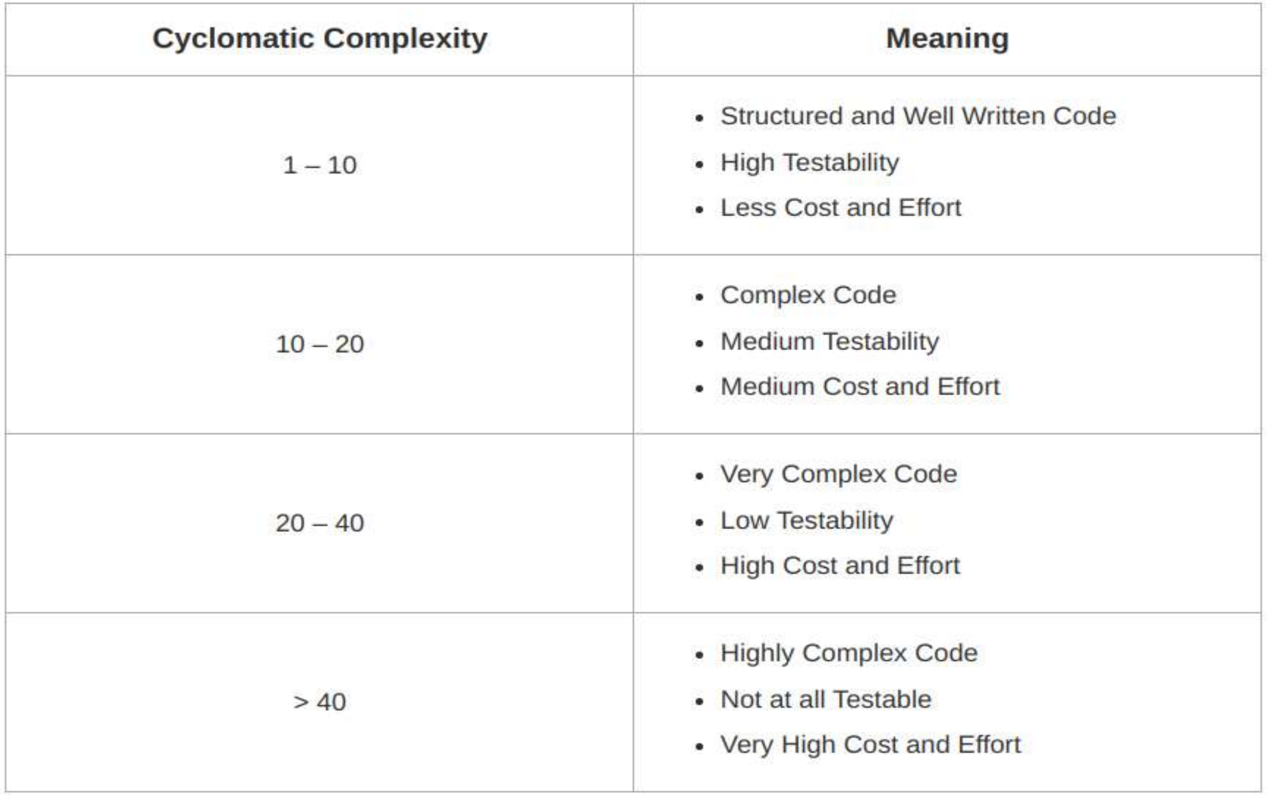
Method 3 :-

Cyclomatic Complexity= 1+(1+1)
=3


$$\begin{tikzcd} 1 \arrow[d] & & 5 \arrow[dl] \arrow[dr] & \\ 2 \arrow[d] & 6 \arrow[dr] & & 7 \arrow[dl] \\ 3 \arrow[d] & & 8 & \\ 4 & & & \end{tikzcd}$$

Overview on the complexity number :

Cyclomatic complexity indicates several information about the program code-



Cyclomatic Complexity	Meaning
1 – 10	• Structured and Well Written Code • High Testability • Less Cost and Effort
10 – 20	• Complex Code • Medium Testability • Medium Cost and Effort
20 – 40	• Very Complex Code • Low Testability • High Cost and Effort
> 40	• Highly Complex Code • Not at all Testable • Very High Cost and Effort

Uses of Cyclomatic Complexity:

- It helps in determining the software quality.
- It is an important indicator of program code’s readability, maintainability and portability.
- It helps the developers and testers to determine independent path executions.
- It evaluates the risk associated with the application or program.
- Developers can assure that all the paths have been tested atleast once

Tools for Cyclomatic Complexity calculation:

Many tools are available for determining the complexity of the application. Some complexity calculation tools are used for specific technologies. Complexity can be found by the number of decision points in a program. The decision points are if, for, for-each, while, do, catch, case statements in a source code.

Examples of tools are

- OCLint - Static code analyzer for C and Related Languages
- Reflector Add In - Code metrics for .NET assemblies
- GMetrics - Find metrics in Java related applications

Disadvantage of Cyclomatic Complexity

McCabe is easy to interpret for a single function. But its computation for a source file, a class or an application is difficult to interpret. The reasons for this is that the function call is not represented in the flow diagram. The McCabe metric is computed independently for each function and it is as if there are interactions between the functions present in the source code.

McCabe’s measure is only of code complexity and not the complexity of the data structures.

Conclusion:

Cyclomatic Complexity is software metric useful for **White Box Testing**. It is mainly used to evaluate complexity of a program. If the decision points are more, then complexity of the program is more. If program has high complexity number, then probability of error is high with increased time for maintenance .

Thank You