Mccabe's Cyclomatic Complexity: Calculate with Flow Graph (Example)

To understand Cyclomatic Complexity, lets first understand -

What is Software Metric?

Measurement is nothing but quantitative indication of size / dimension / capacity of an attribute of a product / process. Software metric is also a quantitative measure of an attribute a software system possesses with respect to Cost, Quality, Size and Schedule.

Example-

```
Measure - No. of Errors
Metrics - No. of Errors Found per person
```

In this tutorial, you will learn-

Cyclomatic Complexity in Software Testing

Cyclomatic Complexity in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs or with respect to functions, modules, methods or classes within a software program.

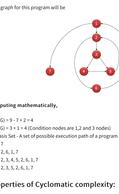
Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths.

This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.

In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.

Flow graph notation for a program:

Flow Graph notation for a program defines several nodes connected through the edges. Below are Flow diagrams for statements like if-else, While, until and normal sequence of flow.



How to Calculate Cyclomatic Complexity

Mathematical representation:

Mathematically, it is set of independent paths through the graph diagram. The Code complexity of the program can be defined using the formula -

```
V(G) = E - N + 2
```

Where,

E - Number of edges

N - Number of Nodes

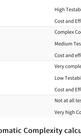
```
V (G) = P + 1
```

Where P = Number of predicate nodes (node that contains condition)

Example -

```
i = 0;
num1 /*-Number of nodes present in the graph

while (i<n-1) do
j = i + 1;

while (j<n) do

if A[j]>A[j] then
swap(A[i], A[j]);

end do;
i=i+1;

end do;
```

Flow graph for this program will be



Computing mathematically,

- V(G) = 9 - 7 + 2 = 4
- V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
- Basis Set - A set of possible execution path of a program
- 1, 7
- 1, 2, 6, 1, 7
- 1, 2, 3, 4, 5, 2, 6, 1, 7
- 1, 2, 3, 5, 2, 6, 1, 7

Properties of Cyclomatic complexity:

Following are the properties of Cyclomatic complexity:

1. V (G) is the maximum number of independent paths in the graph
2. V (G) >=1
3. G will have one path if V (G) = 1
4. Minimize complexity to 10

How this metric is useful for software testing?

Basis Path testing is one of White box technique and it guarantees to execute atleast one statement during testing. It checks each linearly independent path through the program, which **means number test cases, will be equivalent to the cyclomatic complexity of the program.**

This metric is useful because of properties of Cyclomatic complexity (M) -

1. M can be number of test cases to achieve branch coverage (Upper Bound)
2. M can be number of paths through the graphs. (Lower Bound)

Consider this example -

```
If (Condition 1)
Statement 1

Else
Statement 2

If (Condition 2)
Statement 3

Else
Statement 4
```

Cyclomatic Complexity for this program will be 8-7+2=3.

As complexity has calculated as 3, three test cases are necessary to the complete path coverage for the above example.

Steps to be followed:

The following steps should be followed for computing Cyclomatic complexity and test cases design.

Step 1 - Construction of graph with nodes and edges from the code

Step 2 - Identification of independent paths

Step 3 - Cyclomatic Complexity Calculation

Step 4 - Design of Test Cases

Once the basic set is formed, TEST CASES should be written to execute all the paths.

More on V (G):

Cyclomatic complexity can be calculated manually if the program is small. Automated tools need to be used if the program is very complex as this involves more flow graphs. Based on complexity number, team can conclude on the actions that need to be taken for measure.

Following table gives overview on the complexity number and corresponding meaning of v (G):

Complexity Number	Meaning
1-10	Structured and well written code
	High Testability
	Cost and Effort is less
10-20	Complex Code
	Medium Testability
	Cost and effort is Medium
20-40	Very complex Code
	Low Testability
	Cost and Effort are high
>40	Not at all testable
	Very high Cost and Effort

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 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

Uses of Cyclomatic Complexity:

Cyclomatic Complexity can prove to be very helpful in

- Helps developers and testers to determine independent path executions
- Developers can assure that all the paths have been tested atleast once
- Helps us to focus more on the uncovered paths
- Improve code coverage in Software Engineering
- Evaluate the risk associated with the application or program
- Using these metrics early in the cycle reduces more risk of the program

Conclusion:

Cyclomatic Complexity is software metric useful for structured or [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 and minimum time for maintenance and trouble shoot.

Prev
Report a Bug
Next

