CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

INTRODUCTION TO SOFTWARE TESTING

BY
DR. PRAVEEN KANTHA

NUMERICAL PROBLEMS

COUNTING FUNCTION POINT (FP):

Step-1: F = 14 * scale

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

0 - No Influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential

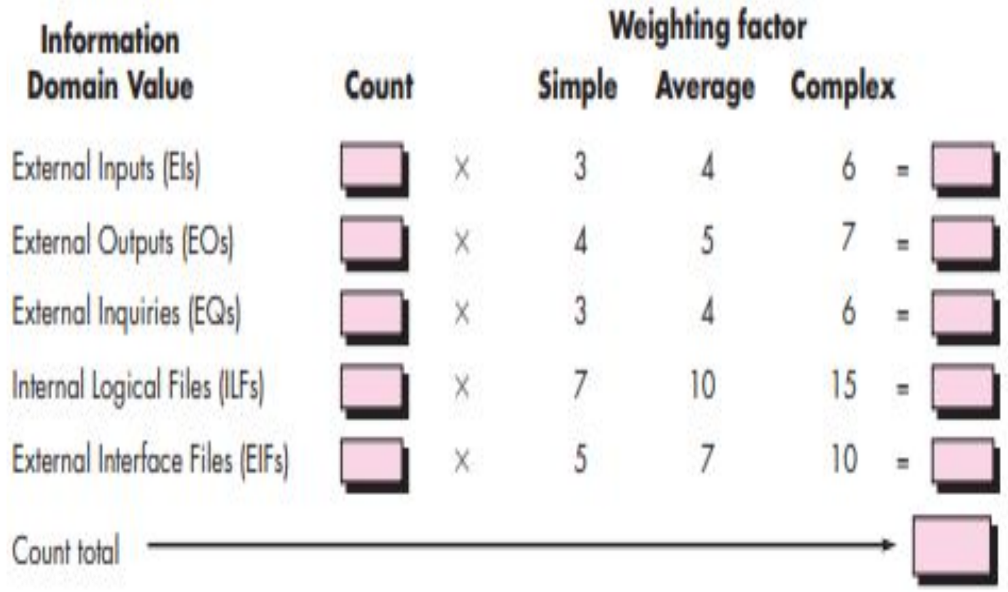
Step-2: Calculate Complexity Adjustment Factor (CAF).

$$\mathbf{CAF = 0.65 + (0.01 * F)}$$

COUNTING FUNCTION POINT (FP):

Step-3: Calculate Unadjusted Function Point (UFP). TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10



Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total							

Multiply each individual function point to corresponding values in TABLE.

COUNTING FUNCTION POINT (FP):

Step-4: Calculate Function Point.

$$FP = UFP * CAF$$

COUNTING FUNCTION POINT (FP):

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input	= 50
User Output	= 40
User Inquiries	= 35
User Files	= 6
External Interface	= 4

COUNTING FUNCTION POINT (FP):

Step-1: As complexity, the adjustment factor is average (given in question), hence,

scale = 3 (because weighting factors are average)

F = 14 * 3 = 42

Step-2:

CAF = 0.65 + (0.01 * 42) = 1.07

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor

0 - No Influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential

Formula for Complexity Adjustment Factor (CAF).

CAF = 0.65 + (0.01 * F)

COUNTING FUNCTION POINT (FP):

User Input = 50
User Output = 40
User Inquiries = 35
User Files = 6
External Interface = 4

Step-3: As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

Calculate Unadjusted Function Point

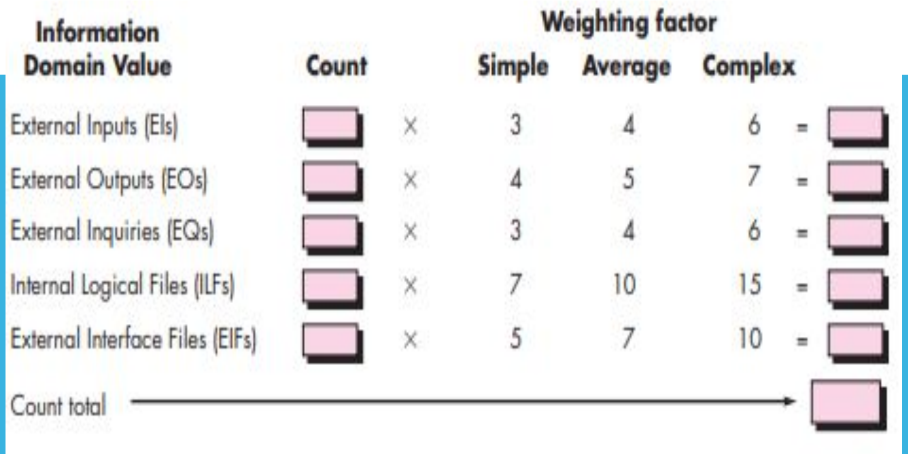
(UFP). = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628

Step-4:

Function Point = 628 * 1.07 = 671.96

Formula of Function Point.

FP = UFP * CAF



Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total							

1. A system has 12 external inputs, 24 external outputs, fields 30 different external queries, manages 4 internal logical files, and interfaces with 6 different legacy systems (6 EIFs). All of these data are of average complexity and the overall system is relatively simple. Compute FP for the system.

2. Compute the function point value for a project with the following information domain characteristics: Number of user inputs: 32 Number of user outputs: 60 Number of user inquiries: 24 Number of files: 8 Number of external interfaces: Assume that all complexity adjustment values are average.

A software project is estimated to have 40,000 LOC. The team will consist of 4 programmers, who can develop 80 LOC per day. The project will run for 20 weeks. What is the total estimated effort for the project in person-months?

To calculate the total estimated effort for the project in person-months, we need to consider

- Number of lines of code (LOC)
- Number of programmers, and
- their productivity rate.

Here's how we can calculate it:

Step 1: Calculate the total number of person-days required for the project:

Person-days = (LOC ÷ LOC per day) ÷ Number of programmers

= (40,000 ÷ 80) ÷ 4 = 125

Step 2: Convert person-days to person-months:

Assuming a month consists of 4 weeks, we can divide the total person-days by 20 (the number of working days in a month) to get the person-months:

Person-months = Person-days ÷ 20

= 125 ÷ 20

= 6.25

A software project is estimated to have 12,000 LOC. The team will consist of 5 programmers, who can develop 74 LOC per day. The project will run for 15 weeks. What is the total estimated effort for the project in person-months?

What is the total number of Lines of Code (LOC) for a software project with three modules, given that Module A has 1200 LOC, Module B has 1000 LOC, and Module C has 6500 LOC?

Ans:

Total Lines of Code (LOC) = LOC of Module A + LOC of Module B + LOC of Module C

= 1200 + 1000 + 6500

= 8700 lines of code

Therefore, the total Lines of Code (LOC) for the entire software project is 8700

How many Lines of Code (LOC) are there in total for a software project with three modules, where Module A has 1700 LOC, Module B has 5800 LOC, and Module C has 4900 LOC?

What is the expected cost of a software project estimated at 700 Function Points (FP), assuming a team of seven consisting of two architects, three programmers, and two testers, with respective monthly salaries of 90,000, 70,000, and 60,000? The team's average productivity is 10 FP per person per month.

Ans:

Total FP=700 FP

Average FP=10 per person-month

Total 7 Person is assigned

So, the time duration to complete 700 FP =700/ (7*10) = 10 month required

Projected cost of the project

= (2 architect salary + 3 programmer's salary + 2 tester salary) *10

= (2*90,000 + 3*70,000 + 2*60,000) *10

=(180,000 + 210,000 + 120,000)*10

=(250,000)*10

=5,100,000

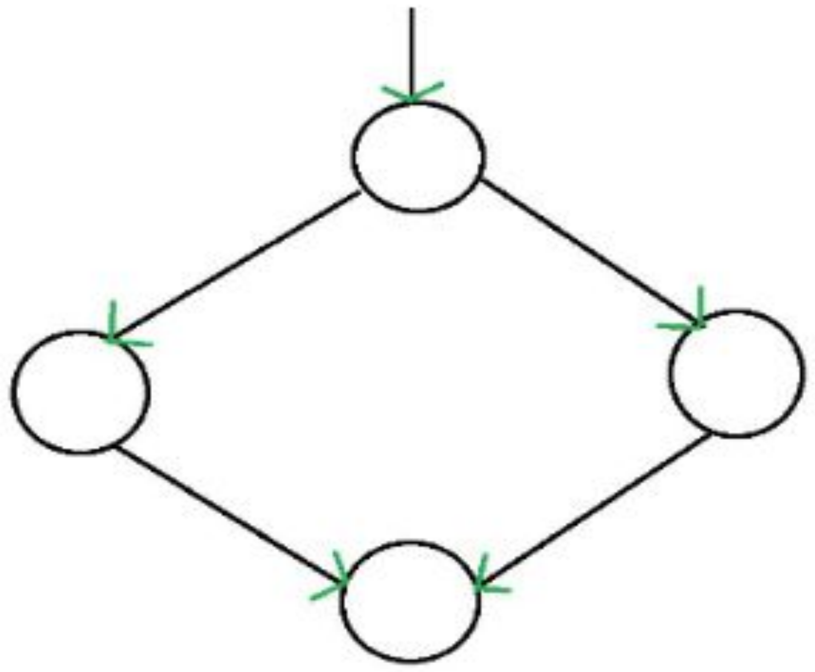
What is the expected cost of a software project estimated at 750 Function Points (FP), assuming a team of 5 consisting of an architect, two programmers, and two testers, with respective monthly salaries of 95,000, 67,000, and 54,000? The team's average productivity is 10 FP per person per month.

How many test cases are needed to comprehensively test the input range of 20 to 80 in a software application, applying boundary value analysis?

Using boundary value analysis, we need to test the boundaries (20 and 80) as well as values just inside and just outside the boundaries (19, 20, 21, 79, 80 and 81). Therefore, the minimum number of test cases required is 6.

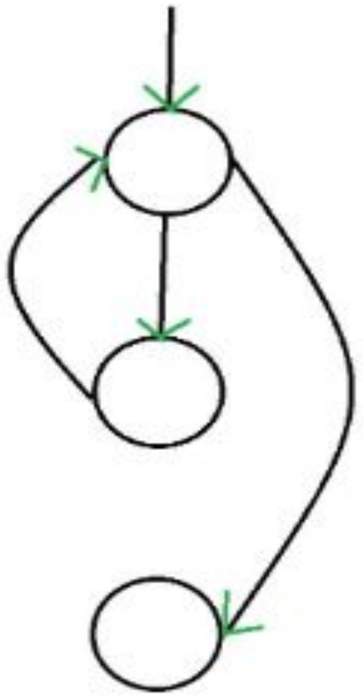
How many test cases are needed to comprehensively test the input range of 75 to 101 in a software application, applying boundary value analysis?

1. If-else



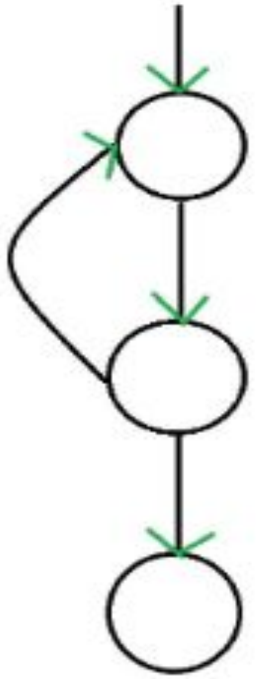
If-then-else

2. While



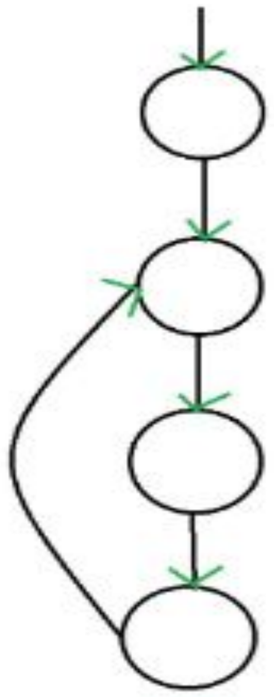
while

3. do-while



do-while

4. for



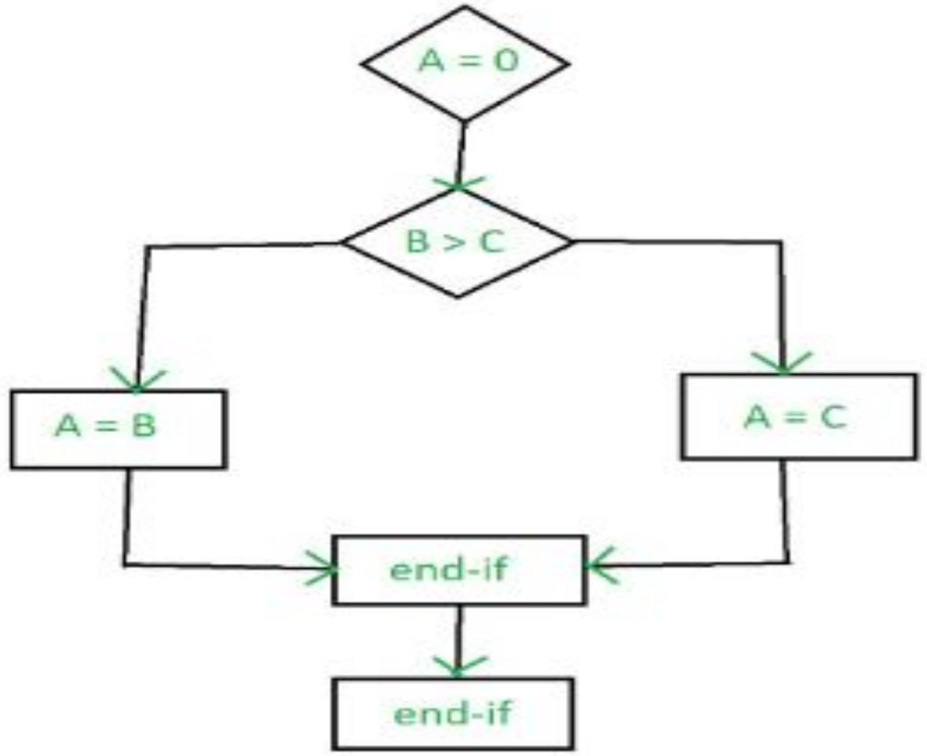
for

Example

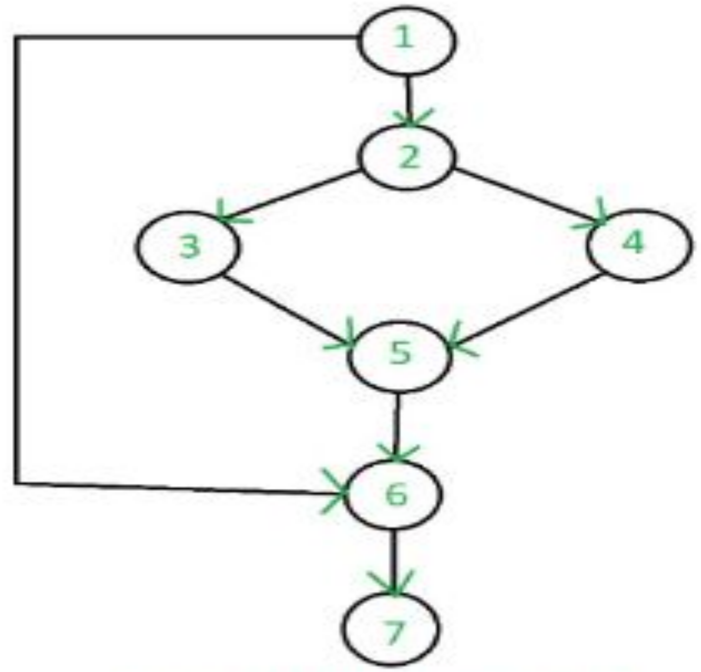
```
if A = 10 then
if B > C
A = B
else A = C
endif

endif
print A, B, C
```

Flowchart of above example will be:



Control Flow Graph of above example will be:



Control Flow Graph

How to Calculate Cyclomatic Complexity?

Steps that should be followed in calculating cyclomatic complexity and test cases design are:

Construction of graph with nodes and edges from code.

- Identification of independent paths.
- Cyclomatic Complexity Calculation
- Design of Test Cases

So, **cyclomatic complexity M** would be defined as,

M = E – N + 2P *where E = the number of edges in the control flow graph*

N = the number of nodes in the control flow graph

P = the number of connected components

In case, when exit point is directly connected back to the entry point. Here, the graph is strongly connected, and cyclometric complexity is defined as

M = E – N + P

where

E = the number of edges in the control flow graph

N = the number of nodes in the control flow graph

P = the number of connected components

In the case of a single method, P is equal to 1. So, for a single subroutine, the formula can be defined as

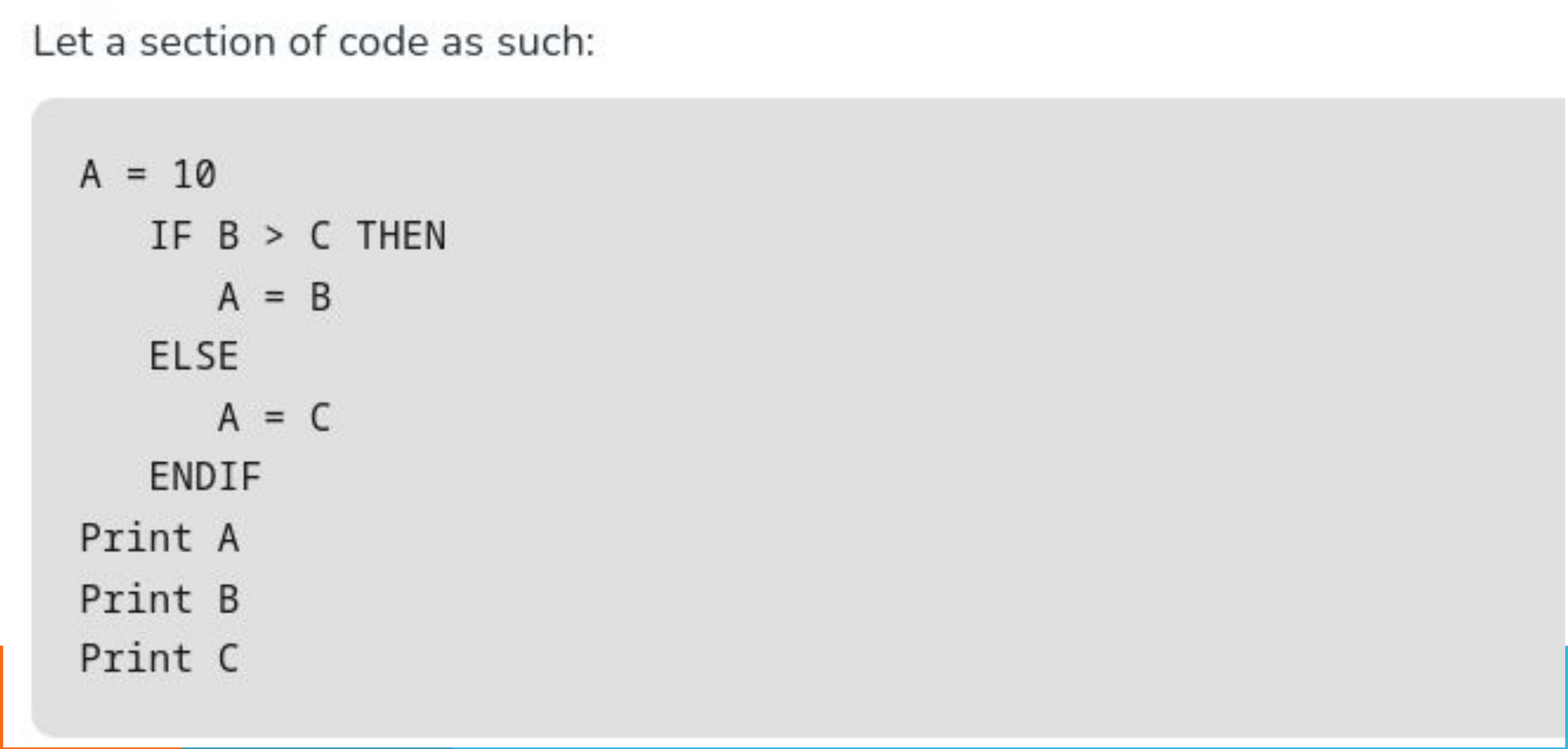
$$M = E - N + 2$$

where

E = the number of edges in the control flow graph

N = the number of nodes in the control flow graph

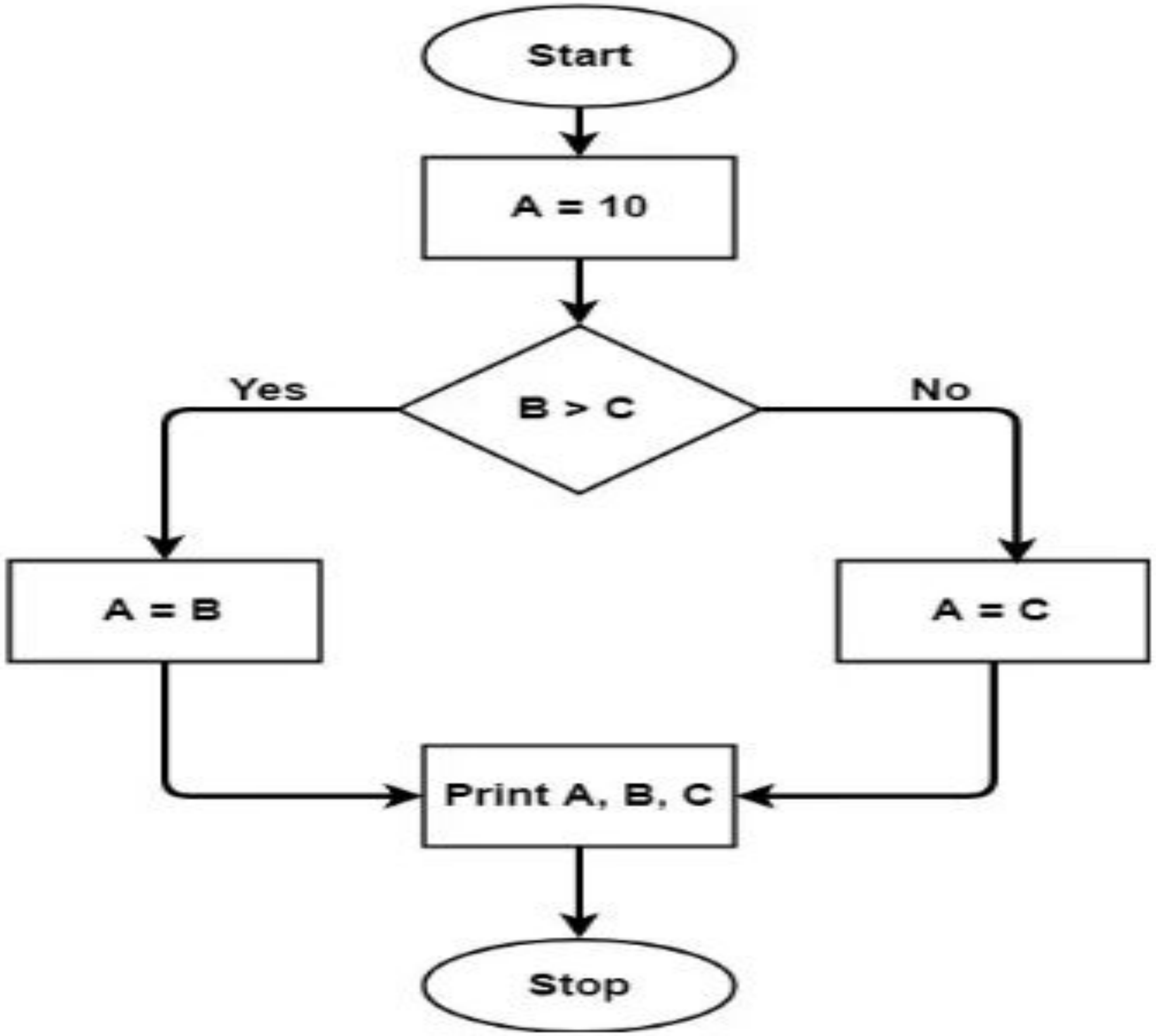
P = the number of connected components



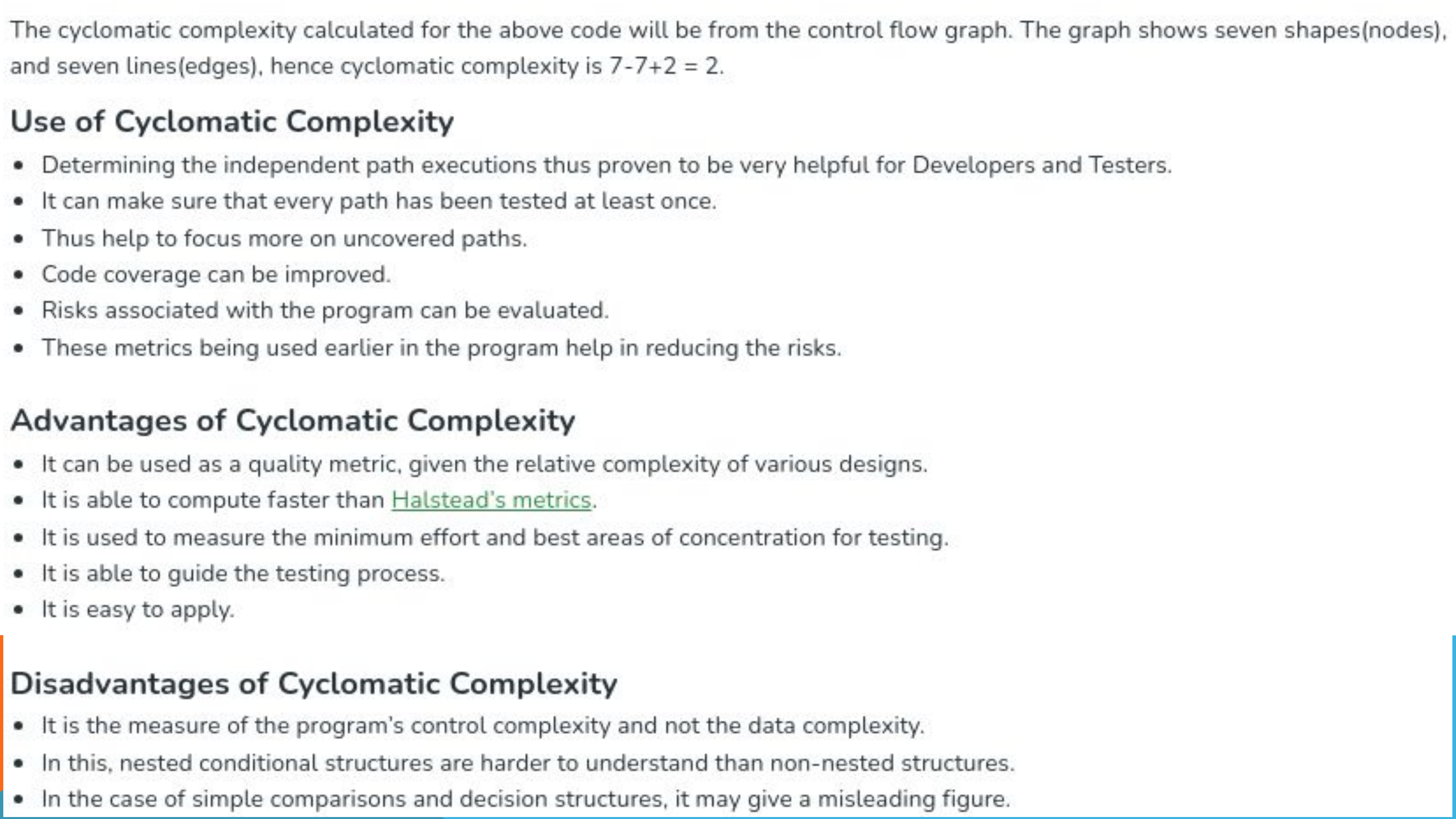
Let a section of code as such:

```
A = 10
   IF B > C THEN
      A = B
   ELSE
      A = C
   ENDIF
Print A
Print B
Print C
```

Control Flow Graph of the above code



Cyclomatic Complexity



The cyclomatic complexity calculated for the above code will be from the control flow graph. The graph shows seven shapes(nodes), and seven lines(edges), hence cyclomatic complexity is 7-7+2 = 2.

Use of Cyclomatic Complexity

- Determining the independent path executions thus proven to be very helpful for Developers and Testers.
- It can make sure that every path has been tested at least once.
- Thus help to focus more on uncovered paths.
- Code coverage can be improved.
- Risks associated with the program can be evaluated.
- These metrics being used earlier in the program help in reducing the risks.

Advantages of Cyclomatic Complexity

- It can be used as a quality metric, given the relative complexity of various designs.
- It is able to compute faster than Halstead's metrics.
- It is used to measure the minimum effort and best areas of concentration for testing.
- It is able to guide the testing process.
- It is easy to apply.

Disadvantages of Cyclomatic Complexity

- It is the measure of the program's control complexity and not the data complexity.
- In this, nested conditional structures are harder to understand than non-nested structures.
- In the case of simple comparisons and decision structures, it may give a misleading figure.

1. Consider the following program module.

```
int module1 (int x, int y) {
   while (x!=y){
      if(x>y)
         x=x-y,
      else y=y-x;
      }
   return x;
}
```

What is the Cyclomatic Complexity of the above module? [GATE CS 2004]

1. Consider the following program module.

```
int module1 (int x, int y) {
   while (x!=y){
      if(x>y)
         x=x-y,
      else y=y-x;
      }
   return x;
}
```

What is the Cyclomatic Complexity of the above module? [GATE CS 2004]

(A) 1

(B) 2

(C) 3

(D) 4

Answer: Correct answer is **(C)**.