

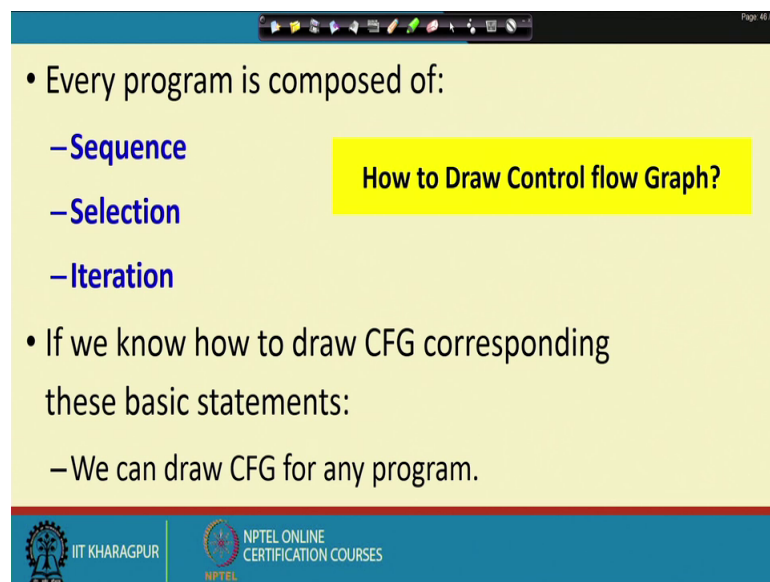
**Software Engineering**  
**Prof. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 59**  
**Path Testing**

Welcome to this lecture. In the last lecture, we were discussing about the Path Testing. And, we said that in path testing, we need all linearly independent paths to be exercised by the test suite. Then, only we can say that the test suite achieves path coverage, but we said that to define linearly independent paths. And, paths we need to be able to draw the control flow graph of the program. And we are discussing how to draw the control flow graph of the program. And, then we said that number the statements.

And, then each number becomes a node and we need to draw the edges among the nodes. And, we were discussing about a systematic technique about how to draw the control flow graph. And, we said that there are 3 types of statements in any program the sequence, selection and iteration.

(Refer Slide Time: 01:24)



The slide content is as follows:

- Every program is composed of:
  - **Sequence**
  - **Selection**
  - **Iteration**
- If we know how to draw CFG corresponding these basic statements:
  - We can draw CFG for any program.

**How to Draw Control flow Graph?**

Page 40/46

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

If we know, how to represent the edges for these three types of statements, then we should be able to draw the control flow graph for any program. Now, let us see how to draw the edges for these three types of statements.

(Refer Slide Time: 01:43)

**How to Draw Control flow Graph?**

- **Sequence:**  
1 a=5;  
2 b=a\*b-1;

The diagram shows a control flow graph with two nodes, 1 and 2, connected by a downward arrow, representing the sequence of statements.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

First let us look at sequence type of statement; this is an example of sequence type of statement. And, here the control just flows from one statement to the other in sequence. So, this the easiest of all 3 types of statements and we have numbered this 1 and 2 and we just draw the edge showing the control flow occurs from statement 1 to statement 2.

(Refer Slide Time: 02:11)

**How to Draw Control Flow Graph?**

- **Selection:**  
1 if(a>b) then  
2 c=3;  
3 else c=5;  
4 c=c\*c;

The diagram shows a control flow graph with four nodes, 1, 2, 3, and 4, connected by arrows forming a diamond shape, representing the selection statement.

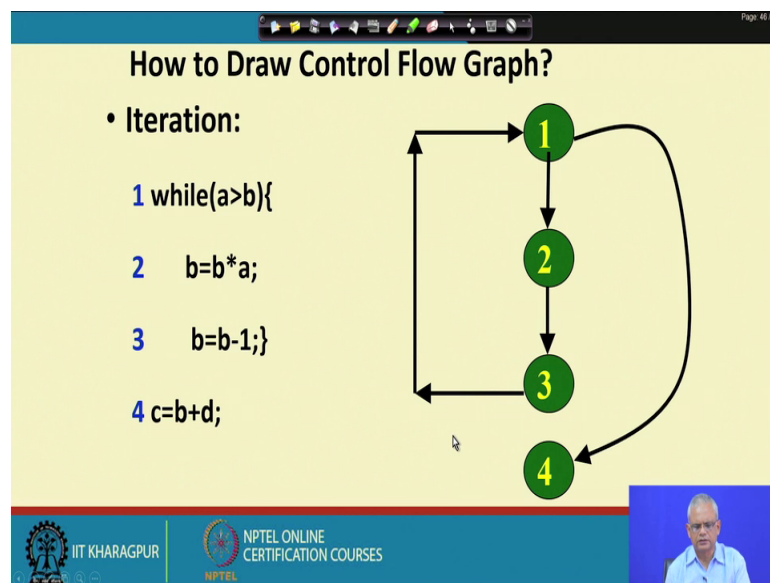
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at a selection type statement, if a greater than b then c equal to 3 else c equal to 5, so this is the statement. And, then the subsequent statement is c equal to c into

c. So, this is the selection statement that follows the sequence statement based on the outcome of the decision here the control transfers either to 2 or to 3.

But, after 2 and 3 complete 2 or 3 complete the control goes to the next statement in sequence. So, drawn that 1 2 either 2 or 3 and then from both 2 and 3 control comes to 4 so, this also easy to draw.

(Refer Slide Time: 03:16)



Now, let us look at iteration type of statement. So, this is a while loop and here if the while condition is true, then it enters the loop and statement 2 will be executed, but if the while condition is false it goes out of the loop and goes to the next statement in sequence, but the only thing to mark here is that each time it completes a loop the control comes back to check what is the outcome of the decision. Only when the decision becomes false it comes to the next statement.

The decision never transfers from 3 to 4, but from 3 it always goes back to 1 where the decision is evaluated and only if it is false it comes to 4. So, the same thing we have represented here, if the decision is true enters the loop 1 2 3 and it compulsorily transfers to 1 to check what is the outcome of the decision. And, if it is true again it look continuous the loop and only when evaluates to false goes to the next statement.

So, if you draw edge between 3 and 4, that will be erroneous many who start to draw control flow graphs make that mistake they draw an edge between 3 and 4, but see here

that control does not transfer from 3 to 4, but it goes back to 1 where the decision is checked and only when it is false the control transfers to 4.

(Refer Slide Time: 05:15)

## Path

- A path through a program:
  - A node and edge sequence from the starting node to a terminal node of the control flow graph.
  - There may be several terminal nodes for program.

The diagram shows a control flow graph with four nodes labeled 1, 2, 3, and 4. Node 1 is at the top, 2 is below it, 3 is below 2, and 4 is at the bottom. Directed edges connect 1 to 2, 2 to 3, 3 to 4, and 3 back to 1, forming a loop.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, once we draw the control flow graph, we can define a path. A path is any node and edge sequence from the start node to a terminal node of the control flow graph. For this example, from 1 we can have a path 1 2 3 and then 4 another may be 1 2 3, 1 2 3 4. So, several paths are possible given a program. And also there can be several terminal nodes in a program, because there may be written statements and so on.

(Refer Slide Time: 06:12)

## All Path Criterion

- In the presence of loops, the number paths can become extremely large:
  - This makes all path testing impractical

The diagram shows a control flow graph with four nodes labeled 1, 2, 3, and 4. Node 1 is at the top, 2 is below it, 3 is below 2, and 4 is at the bottom. Directed edges connect 1 to 2, 2 to 3, 3 to 4, and 3 back to 1, forming a loop.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The all path criterion says that, in presence of loops the number of paths can be infinite or very large. The reason for that is that for given this kind of a loop, we can have 1 2 3 4, 1 2 3, 1 2 3 4, 1 2 3, 1 2 3, 1 2 3, 3 times and 4 and so on all are all qualifiers paths. So, we can say that in presence of loops the number of paths becomes infinite.

And therefore, if you trying to achieve all path criterion, it will be very difficult, even if we know how many what is the maximum number of iterations that can occur here, but sometimes we cannot determine for example, it might be that while true repeat. So, all path criterion even though it is a strong criterion, but then it is impractical because we do not know how many paths are required and the paths are can be extremely large.

(Refer Slide Time: 07:47)

**Linearly Independent Path**

- Any path through the program that:
  - Introduces at least one new edge:
  - Not included in any other independent paths.

The diagram shows a graph with four nodes labeled 1, 2, 3, and 4. Node 1 is at the top, node 2 is below it, node 3 is below node 2, and node 4 is at the bottom. There are directed edges: 1 to 2, 2 to 3, 3 to 4, and 4 to 1. There is also a curved edge from 1 to 3. A mouse cursor is pointing at the edge between nodes 2 and 3.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

From this consideration that all path criterion is not a practical criterion the linearly independent path was defined, the linearly independent path is defined as a path belongs to the set of linearly independent paths.

If, it introduce at least one new edge, which is not included in any other independent path so, 1 2 3 4 will be a independent path linear linearly independent path, but 1 2 3, 1 2 3 4 will not, because it does not cover any new edge not covered by the previous test case that is 1 2 3 4. So, the same edges are repeating and get excluded by linearly independent path.

(Refer Slide Time: 08:47)

The slide is titled "Independent path" in a yellow box. It contains the following text:

- It is straight forward:
  - To identify linearly independent paths of simple programs.
- For complicated programs:
  - It is not easy to determine the number of independent paths.

A green graph diagram is shown on the right side of the slide, consisting of several nodes connected by edges, forming a complex network.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video feed of the presenter.



Given a control flow graph finding a linearly independent path for very very simple programs, we can easily find it, but for complicated programs it becomes extremely difficult.

For example, if I draw a graph like this. So, just for this finding the number of independent path is nontrivial. And just remember that the number of statements can be 100 and the number of edges can be let us say 150. And then even if you spend a month trying to find out what is the number of linearly independent paths you may still be struggling at the end of the month. So, just from visual inspection trying to compute, linearly independent path set is an extremely complicated task for nontrivial programs.

(Refer Slide Time: 10:22)

### McCabe's Cyclomatic Metric

- An upper bound:
  - For the number of linearly independent paths of a program
- Provides a practical way of determining:
  - The maximum number of test cases required for basis path testing.


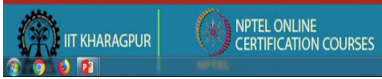
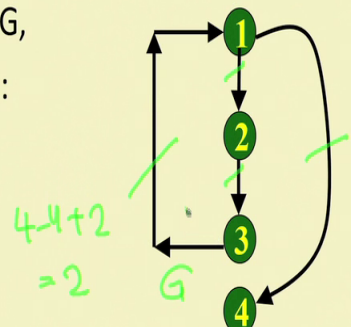


McCabe based on graph theoretical results, he could find the number of linearly independent paths in a straight forward manner. Rather than trying to count all the linearly independent paths, he used graph theoretic results. And extremely simple to use his result to find the number of linearly independent paths in a program, and that provides a practical way of determining the maximum number of test cases required for the basis path testing. So, the set of linearly independent paths are also called as basis paths and that is the reason McCabe's Cyclomatic Metric is a important metric for any program.

(Refer Slide Time: 11:22)

### McCabe's Cyclomatic Metric

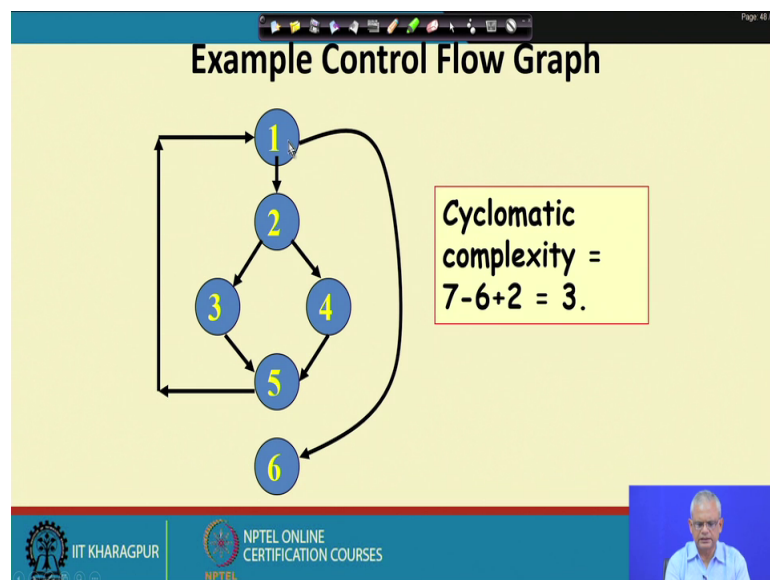
- Given a control flow graph G, cyclomatic complexity  $V(G)$ :
  - $V(G) = E - N + 2$
  - N is the number of nodes in G
  - E is the number of edges in G



Let us see how to compute the McCabe's metric. Given a control flow graph the McCabe's metric is defined as  $V(G)$ ,  $G$  is the graph and  $V(G)$  is the metric, which is the number of edges in the graph minus the number of nodes plus 1. Here the number of edge is 1 2 3 4, 4 edges and there are 4 nodes for. So, 4 minus 4 plus 2 is equal to 2. So, the McCabe metric for a control flow graph for a loop or iterative statement is 2. Given any complex control flow graph we can just compute the number of edges number of nodes and then you can just use the formula  $E$  minus  $N$  plus 2 and get the cyclomatic complexity value.

This is also easily automated because given a graph, we can easily compute the number of nodes and the number of edges and then compute the McCabe's metric.

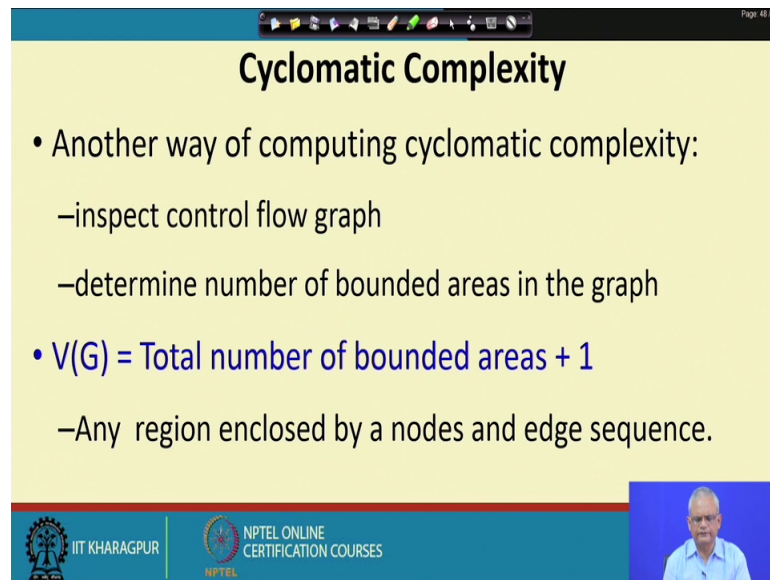
(Refer Slide Time: 12:59)



Let us take this example. So, for this program we find that the number of edges is 7, number of nodes is 6 you can count and then plus 2 which is 3.



(Refer Slide Time: 13:20)



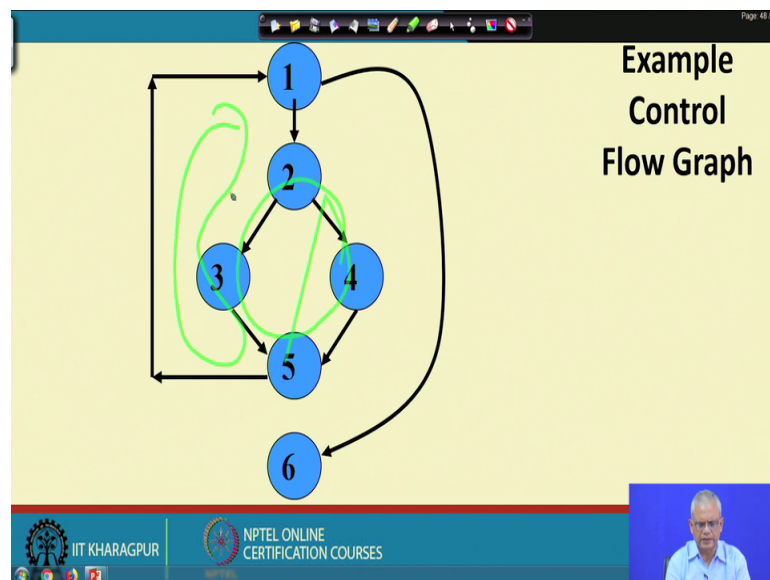
**Cyclomatic Complexity**

- Another way of computing cyclomatic complexity:
  - inspect control flow graph
  - determine number of bounded areas in the graph
- $V(G) = \text{Total number of bounded areas} + 1$ 
  - Any region enclosed by a nodes and edge sequence.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

There are alternate definitions of the cyclomatic complexity, we can visually examine the number of bounded areas and add 1 and that also will give the cyclomatic complexity.

(Refer Slide Time: 13:40)



**Example Control Flow Graph**

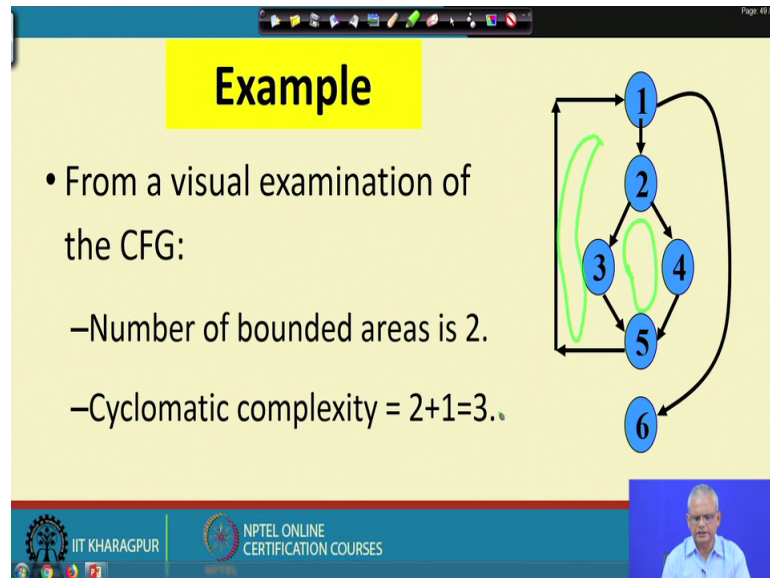
The graph consists of 6 nodes (1-6) and several edges. Three bounded areas are highlighted in green: a triangle formed by nodes 2, 3, and 4; a quadrilateral formed by nodes 2, 3, 4, and 5; and a pentagon formed by nodes 2, 3, 4, 5, and 6.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For example, for the same 1 same control flow graph. We can find the number of bounded areas. So, this is a bounded area, this is a bounded area and this is a bounded area. So, 2 plus 1 number of bounded areas plus 1.

So, 2 plus 1 is 3 and this is easy to compute from a visual examination of the CFG. The other  $1 E \text{ minus } N \text{ plus } 1 E \text{ minus } N \text{ plus } 2$  so, that is easy to compute using a computer program.

(Refer Slide Time: 14:23)



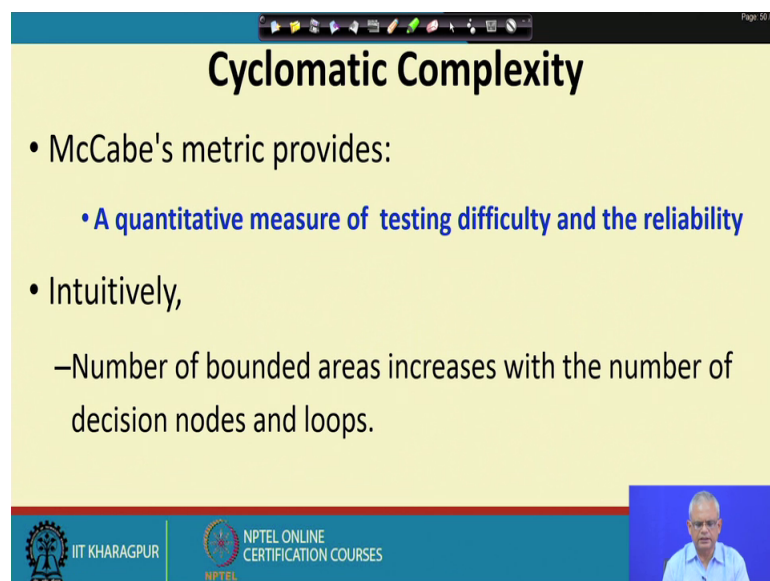
**Example**

- From a visual examination of the CFG:
  - Number of bounded areas is 2.
  - Cyclomatic complexity =  $2+1=3$ .

The diagram shows a control flow graph with 6 nodes (1-6) and several edges. Two bounded areas are highlighted in green: one is a loop between nodes 1, 2, 3, 4, and 5; the other is a loop between nodes 2, 3, 4, and 5. Node 6 is a terminal node reached from node 5.

So, here the number of bounded areas are 2 and 2 plus 1 is 3 and that is the cyclomatic complexity we might also compute  $E \text{ minus } N \text{ plus } 2$ , which will also give the same result.

(Refer Slide Time: 14:50)



**Cyclomatic Complexity**

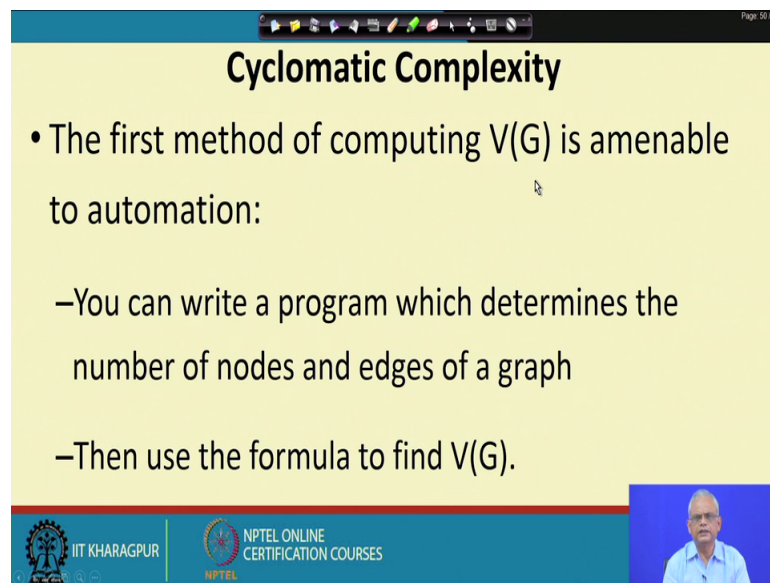
- McCabe's metric provides:
  - **A quantitative measure of testing difficulty and the reliability**
- Intuitively,
  - Number of bounded areas increases with the number of decision nodes and loops.

The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

The McCabe's metric provides a quantitative measure of the testing difficulty, because if the McCabe's metric is 15; that means, there are 15 basis paths or linearly independent paths.

And, we need 15 test cases. So, the McCabe's metric value provides the difficulty level of testing and also the reliability, because that many paths are there and we might miss bugs and that also corresponds to the reliability.

(Refer Slide Time: 15:38)



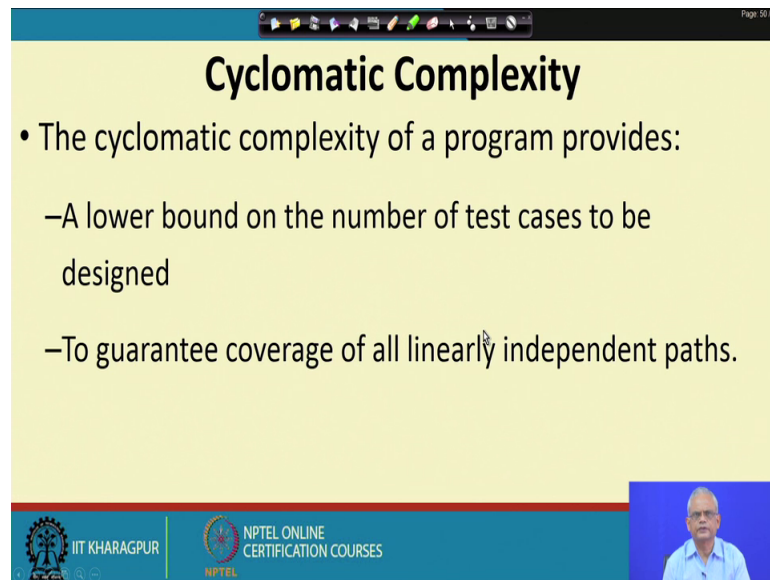
The slide is titled "Cyclomatic Complexity" and contains the following text:

- The first method of computing  $V(G)$  is amenable to automation:
  - You can write a program which determines the number of nodes and edges of a graph
  - Then use the formula to find  $V(G)$ .

The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a speaker is visible in the bottom right corner.

Computing the bounded areas is easy from a visual inspection and the  $E$  minus  $N$  plus 2 formula easy to compute using a computer program.

(Refer Slide Time: 15:52)



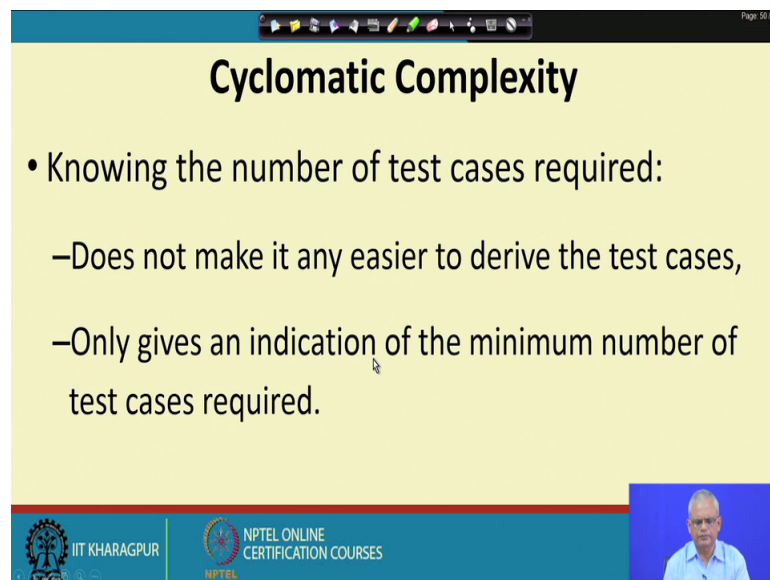
## Cyclomatic Complexity

- The cyclomatic complexity of a program provides:
  - A lower bound on the number of test cases to be designed
  - To guarantee coverage of all linearly independent paths.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, one thing we must remember that it provides a bound on the number of test cases to be designed, we can test even more test cases.

(Refer Slide Time: 16:12)



## Cyclomatic Complexity

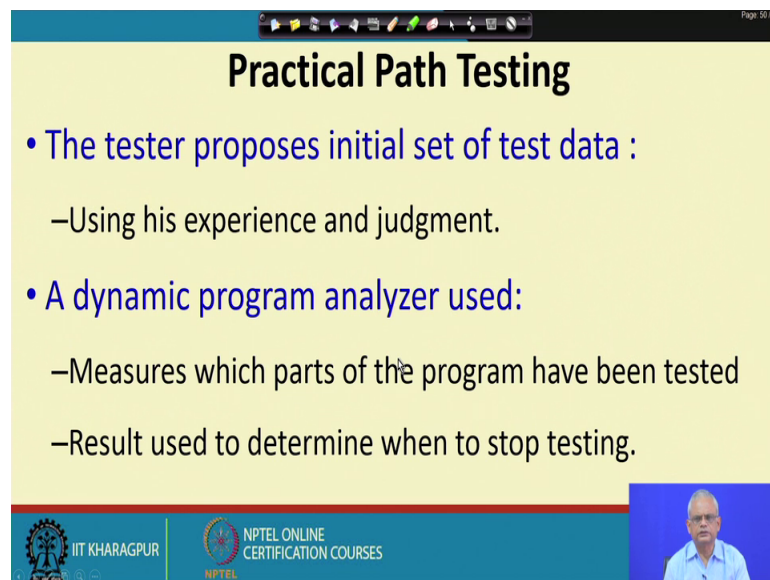
- Knowing the number of test cases required:
  - Does not make it any easier to derive the test cases,
  - Only gives an indication of the minimum number of test cases required.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, the problem is that knowing the number of test cases required does not make it any easier to derive the test cases, because if we say that there are 15 linearly independent paths, but then how do you identify those paths? We know that there are 15, but then identifying those 15 is a extremely tough problem.

So, one way is that we keep on testing and see if a new edge is covered by a test case, which was not covered by earlier test cases then we say that a new linearly independent path is being executed. And, we can count the number of linearly independent paths and we can find the percentages of linearly independent paths that have been executed.

(Refer Slide Time: 17:11)



**Practical Path Testing**

- The tester proposes initial set of test data :
  - Using his experience and judgment.
- A dynamic program analyzer used:
  - Measures which parts of the program have been tested
  - Result used to determine when to stop testing.

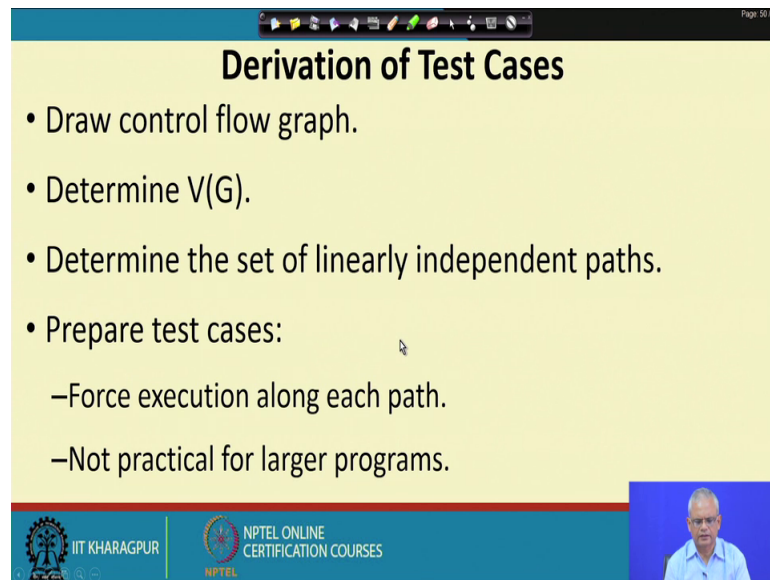
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, in that the tester proposes a initial setup test data and then use a dynamic program analyzer, which counts the number of linearly independent paths being executed. And, then it displays the percentage of linearly independent paths, that have been covered and that gives the path coverage metric.

(Refer Slide Time: 17:40)

### Derivation of Test Cases

- Draw control flow graph.
- Determine  $V(G)$ .
- Determine the set of linearly independent paths.
- Prepare test cases:
  - Force execution along each path.
  - Not practical for larger programs.

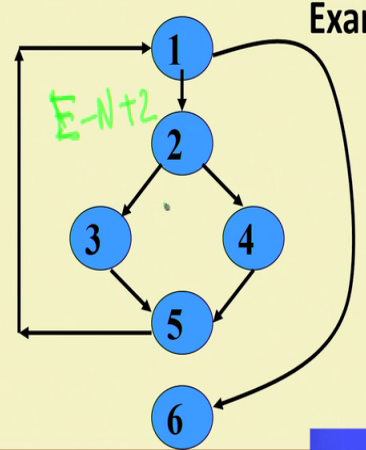


But, can we derive the test cases for very trivial programs yes, if there are 1 or 2 control statements and so on becomes easy, but if there are dozens of control statements becomes extremely difficult to derive the test cases.

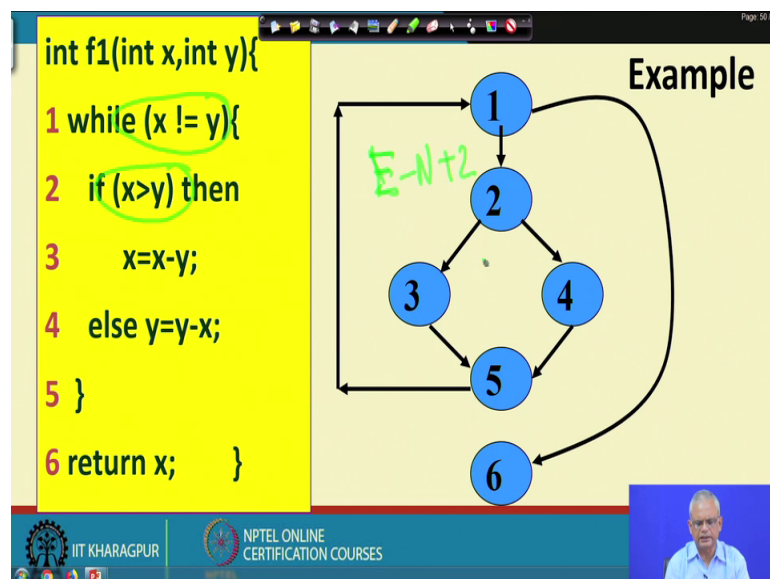
(Refer Slide Time: 18:05)

```
int f1(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; }
```

**Example**



The control flow graph consists of nodes 1 through 6. Node 1 is the entry point to the while loop. Node 2 is the decision point for the if statement. Node 3 is the execution point for the 'x=x-y;' statement, and node 4 is the execution point for the 'y=y-x;' statement. Both nodes 3 and 4 lead to node 5, which loops back to node 1. Node 6 is the exit point from the function. A handwritten note 'E-N+2' is next to node 2.

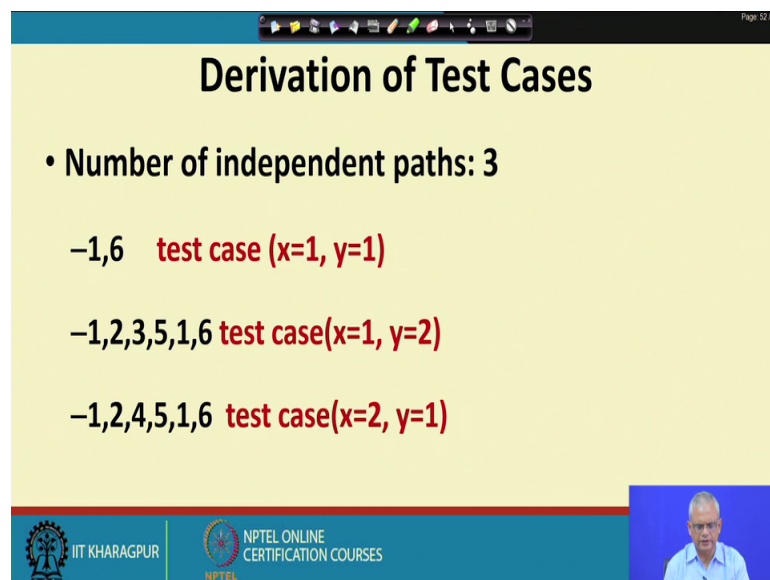


For this example we see that there are 3 linearly independent paths; one occurs when it enters in it executes the statement 3 and the other one when it executes the statement 4. And, we can check that 2 test cases should be able to cover all the linearly independent paths here. Even though the McCabe's metric is 3, that is an upper bound and we might

need 2 or 3 test cases we will design test cases to achieve path coverage, but just need to mention 1 shortcut here to count the number of linearly independent paths, other than drawing the CFG counting the number of edges number of nodes and performing  $E - N + 2$ .

Or computing the number of bounded areas plus 1, the other method is look through the program statement; whenever you find a decision statement count it, number of decision statements plus 1. So, that is also equal to the McCabe's metric. Let me repeat again to be able to easily compute the McCabe's metric look through the program code, whenever you encounter a decision statement like while if etcetera, count it and then add 1 that will also be equal to  $E - N + 2$  or number of independent bounded areas plus 1.

(Refer Slide Time: 20:08)



The slide is titled "Derivation of Test Cases" and lists three independent paths with their corresponding test cases. The paths are represented as sequences of nodes: -1,6; -1,2,3,5,1,6; and -1,2,4,5,1,6. The test cases are (x=1, y=1), (x=1, y=2), and (x=2, y=1) respectively. The slide also includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker.

**Derivation of Test Cases**

- Number of independent paths: 3
- 1,6 test case (x=1, y=1)
- 1,2,3,5,1,6 test case(x=1, y=2)
- 1,2,4,5,1,6 test case(x=2, y=1)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for the previous program we can we have number of independent paths here 3 and we can use 3 test cases for achieving the path coverage.

(Refer Slide Time: 20:29)

The slide features a title "An Interesting Application of Cyclomatic Complexity" at the top. Below the title is a bulleted list of relationships. To the right of the list is a hand-drawn diagram in green ink showing a circle with two vertical lines inside, representing a loop structure. At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker.

**An Interesting Application of Cyclomatic Complexity**

- Relationship exists between:
  - McCabe's metric
  - The number of errors existing in the code,
  - Time required to correct the errors.
  - Time required to understand the program.

The cyclomatic complexity metric other than given an estimate of the number of linearly independent paths, it has several other applications. It co relates with the number of errors existing, in the code after testing, it co relates with the time required to correct an error and it co relates with time required to understand the program. You might ask why there is a co relation between the McCabe's metric and these 3 issues.

It is not had to make out why it is so? Remember, that McCabe's metric gives the number of paths in the program and to understand the program you need to trace all the paths. So, if there are independent paths to be able to understand this behavior of the program, you need to check that from the output how do you trace to the input? So, all paths you have to check in the code and see how the control it gets the values and so on and that is the reason why the understanding difficulty co relates well with the McCabe's metric.

And, that is also the reason, because to able to correct the error you need to understand the program. And therefore, it co relates with the cyclomatic metric and similar is the error existing in the code.



(Refer Slide Time: 22:30)

**Cyclomatic Complexity**

- Cyclomatic complexity of a program:
  - Indicates the **psychological complexity of a program.**
  - Difficulty level of understanding the program.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The cyclomatic complexity corresponds to the difficulty of understanding a program. And therefore, we say that it indicates the psychological complexity of a program or the structural complexity of a program.

(Refer Slide Time: 22:51)

**Cyclomatic Complexity**

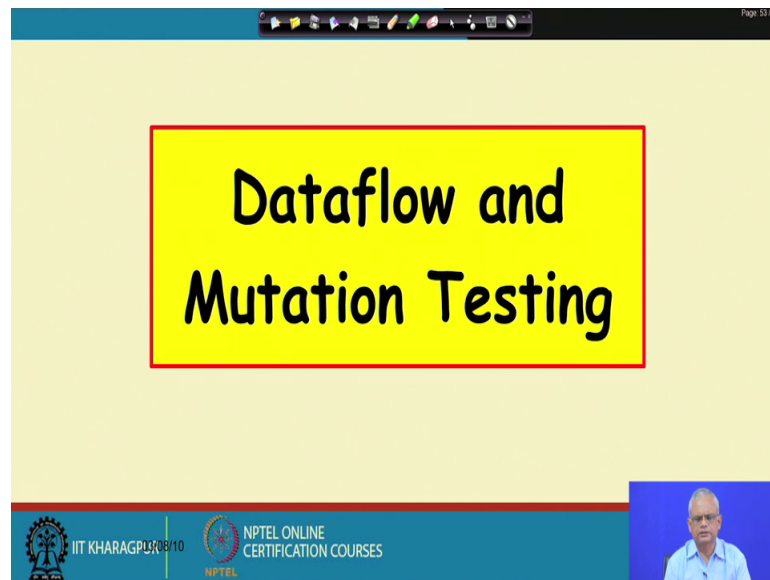
- From maintenance perspective,
  - Limit cyclomatic complexity of modules
    - To some reasonable value.
  - Good software development organizations:
    - Restrict cyclomatic complexity of functions to a maximum of ten or so.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Normally, the companies require their programmers to keep the cyclomatic complexity to a reasonable value. And of course, if you think of it means that every function should have only a few loops and decision statements.

Many organizations require that programmer should not write functions having more than 10 cyclomatic complexity, because otherwise the complexity of understanding debugging becomes extremely large.

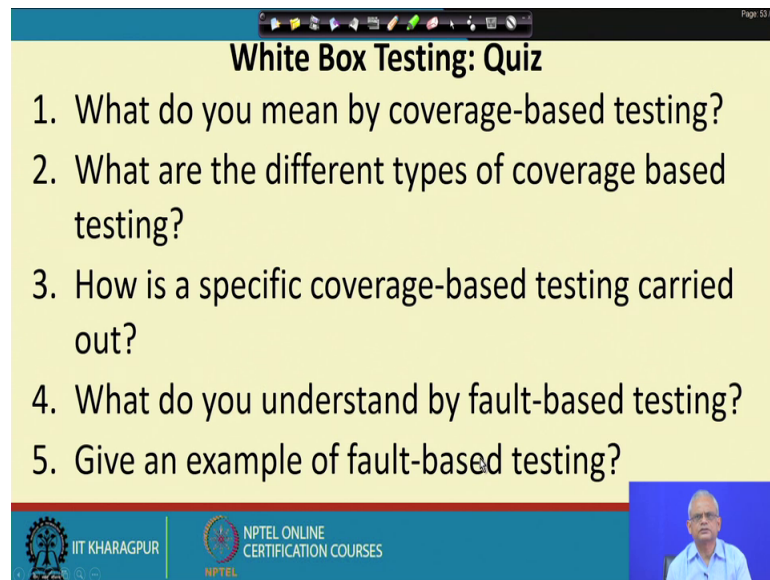
(Refer Slide Time: 23:40)



So, we saw that the cyclomatic complexity metric has many applications. And, we saw how to measure the cyclomatic metric for a program and we saw the applications of the cyclomatic metric, it indicates the difficulty level of testing the program, the difficulty level of understanding the program, it co relates with the final reliability of the program and so on.

And, we saw that the cyclomatic metric is important many organizations restrict it to 10 or so for a unit and it is called as a structural complexity metric and or the psychological complexity metric. The structural metric is a more popular term that is used. Now, let us look at another test technique that is a white box test technique called as the data flow testing and then we look at the mutation testing.

(Refer Slide Time: 24:58)



**White Box Testing: Quiz**

1. What do you mean by coverage-based testing?
2. What are the different types of coverage based testing?
3. How is a specific coverage-based testing carried out?
4. What do you understand by fault-based testing?
5. Give an example of fault-based testing?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, before that a small quiz here what do you mean by coverage based testing? We had seen that coverage based testing implies that certain program elements should be executed by the test suite.

And, we measure the extent to which the program elements are executed by test case and that we give a percentage coverage achievement. And, the elements can be the statements the control transfer edges and so on paths and so on. Now, what are the different types of coverage testing that we discussed? We discussed statement coverage, we discussed decision coverage, the basic condition coverage, condition decision coverage, MC DC coverage, multiple condition coverage, and then the path coverage.

How is a specific coverage-based testing carried out? We saw that designing test cases for coverage based testing is non-trivial except for extremely small programs. And therefore, the test cases are designed and as they are executed impossibly random test cases random values. And, as they are executed the dynamic analyzer tool keeps track of what is the path coverage achieved.

And, we keep on giving inputs until the path coverage becomes acceptably large, but one thing need to mention here is that 100 percent path coverage may not be achievable for non-trivial programs, because there can be in infeasible paths. What do you understand by fault based testing? That is the next question. In a fault based testing we introduce faults into the program and run the test cases and see if those faults are getting detected.

And, the way to introduce fault is called as mutation each time introduce a fault we call it as a mutation of the program.

And, then for on the mutated program, we run the test cases and see if the test cases are able to detect the fault that we have introduced. Give an example of a fault based testing? So, mutation testing is an example of a fault based testing. We are at the end of this lecture we will stop here and we will discuss about the data flow and mutation testing in the next lecture.

Thank you.