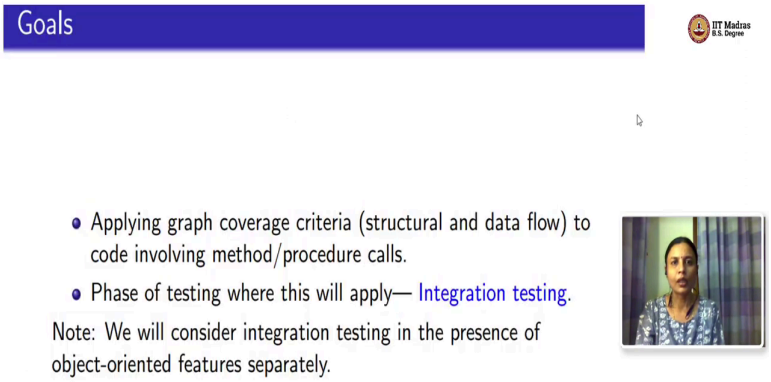
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Technology, Bangalore
Design Integration Testing and Graph Coverage

Welcome back. We begin week 4 lectures, we are in week 4 lectures, as I told you is the first lecture of week 4, we have now moved out of unit testing into what we call the next phase integration testing. And this lecture is about how to use graphs, as we learned for doing design integration test.

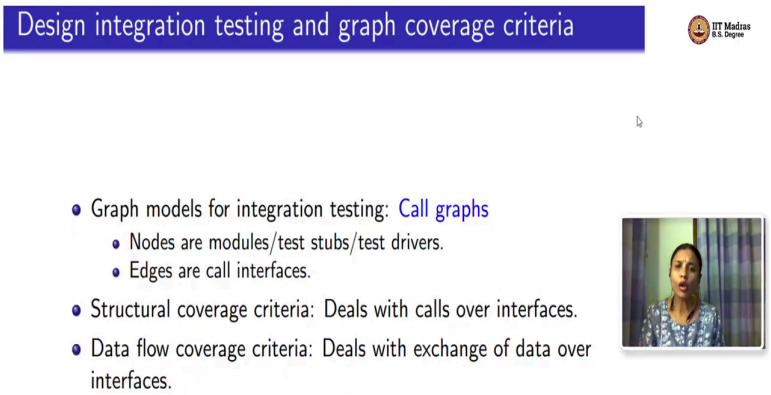
(Refer Slide Time: 00:43)



So, what are our goals? What are we going to learn in this module, be some graph coverage criteria, structural criteria, data flow criteria last week, and we applied that within a method or within a function. Now, we are going to apply them across method calls. So, the graphs that we derive are going to be different. So, but the coverage criteria were more or less be the same. So, we will revisit structural coverage criteria and data flow coverage criteria on different kinds of graphs. And in this process, we will be considering integration testing.

I would like to make a small node here that we assume that the code that we are testing, it could be Java programs, but it does not use any sophisticated object oriented features like for example, the methods that are being called are not polymorphic. There is no inheritance. Those integration testing in the presence of object oriented features involves slightly different graph models and that will be taught as a separate module later in the course.

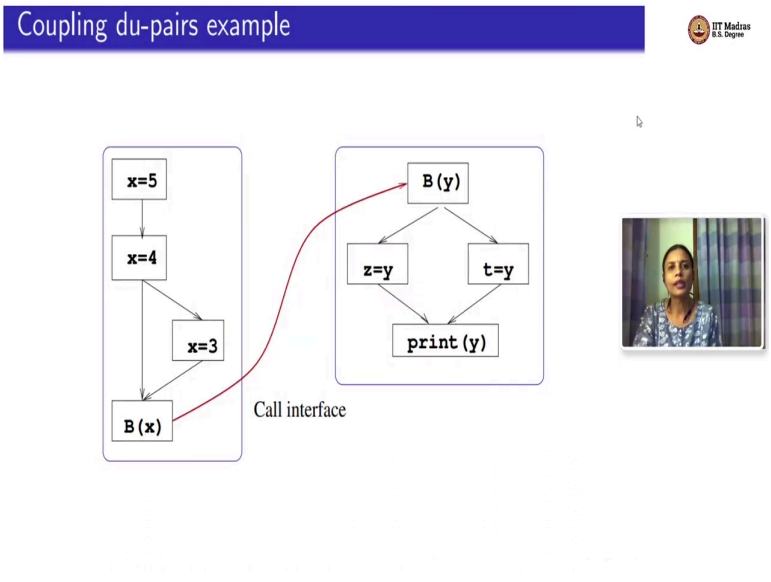
(Refer Slide Time: 01:55)

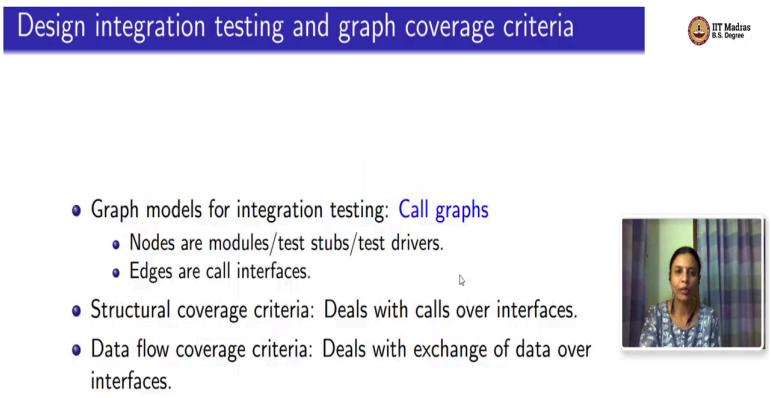


So, graph models for integration testing, are called call graphs, because their edges in these graphs are going to focus on call interfaces. So, like all graphs, these call graphs also have nodes, they also have edges. Nodes are not going to be basic blocks, like in control flow graph or data flow graph, but nodes are going to be entire modules, the entire method itself, entire function or entire procedure. Sometimes the method may not be ready, as we saw last time, in which case you write a dummy method like the stub or a driver. And nodes would be those either methods or their stubs, or driver, which are written for the purpose of testing.

Edges, or call interfaces. The method calls, calling each other for a stub written to call the method that is ready to be tested. In this, we will see structural coverage criteria and data flow coverage criteria. Structural coverage criteria, given nodes or methods or stubs or drivers their edges or call interfaces is directly going to deal with call interfaces. Data flow will involve exchange of data at the call interfaces. And it is going to be different from the normal data flow testing that we saw at the unit level.

(Refer Slide Time: 03:15)





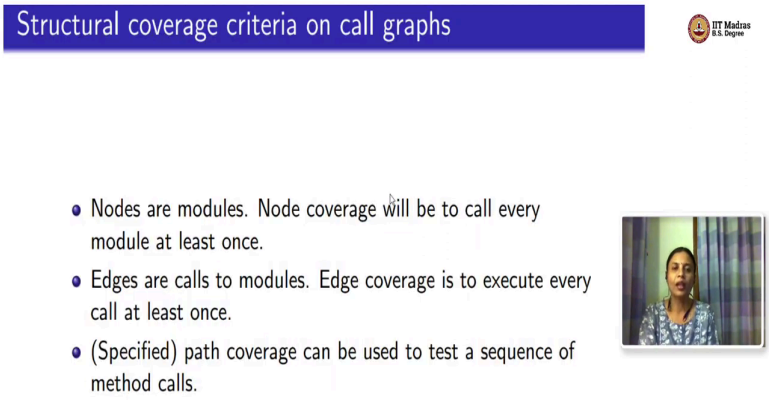
So, here is an example of how the graph looks like. So, let us say there is a control flow graph that you see on the left hand side some small thing it says this is equal to 5 then there is some x is equal to 4, maybe there is a decision statement, and then there is things that happen and then there is a call interface here. So, there is a call to a procedure B by this method, and that changes the control flow to be getting invoked. And in B, there is some code that runs and maybe some return will happen.

So, this call interface is what we are going to focus on. This would be edges of our graph. And this whole blue colored outer node would be the vertex or the nodes of our graph. So, that is what I meant by a call graph here. A call graph nodes are modules, methods, functions,

procedures, test stubs, test drivers is written in replacement of them and edges that these calls a particular function calling another function. This is how call graphs are going to look like.

We may or may not expand into a call graph. It will depend on the context, but there are going to be specific edges that involve call interfaces which we are going to focus on.

(Refer Slide Time: 04:35)



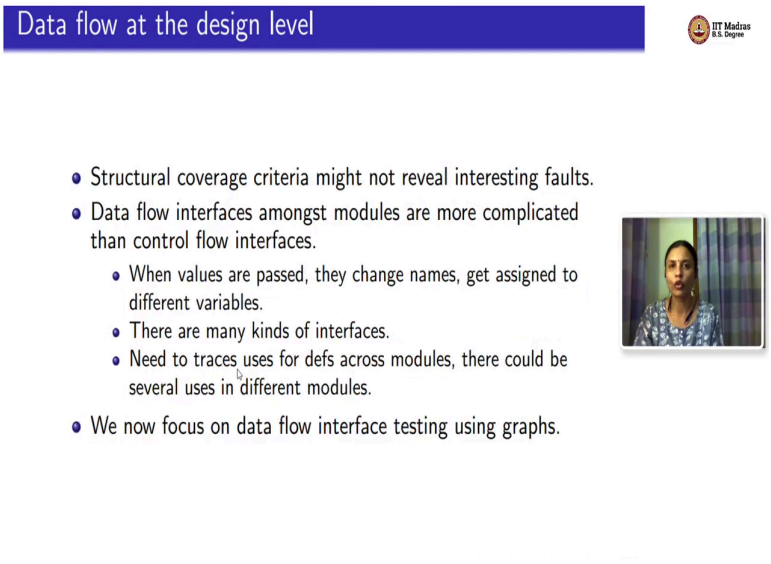
Now let us revisit the structural coverage criteria. We saw node coverage, edge coverage, edge pair coverage, prime part coverage, if you remember. Prime path coverage does not make sense here because the goal of prime path coverage there was to be able to test for the presence of loops. Call interfaces may not have loops or even if there are loops, it is naturally written as a part of the call. There is nothing like skipping a loop and executing a loop and stuff like that.

So, we will look at node coverage. Node coverage means a call is made to every module at least once. That is what node coverage is because nodes of this graph are function calls. Edges are the call interfaces. So, edge coverage means execute. Make sure you test every call interface at least once. Make sure your test cases contain the test requirements is test cases contain every call interface at least once.

Sometimes, you might want to test a sequence of call interfaces, method A calling method B, which in turn call C, which under certain conditions, calls a method D and so on and so forth. So, you could have specified path coverage, which checks for specific sequence of method calls. So, structurally, these three coverage criteria only make sense. Nodes, node coverage

means call every module at least once. It coverages test every call interface once that is your test requirements, specified path coverages test a particular sequence of method calls. The interesting thing is data flow coverage criteria over call graphs.

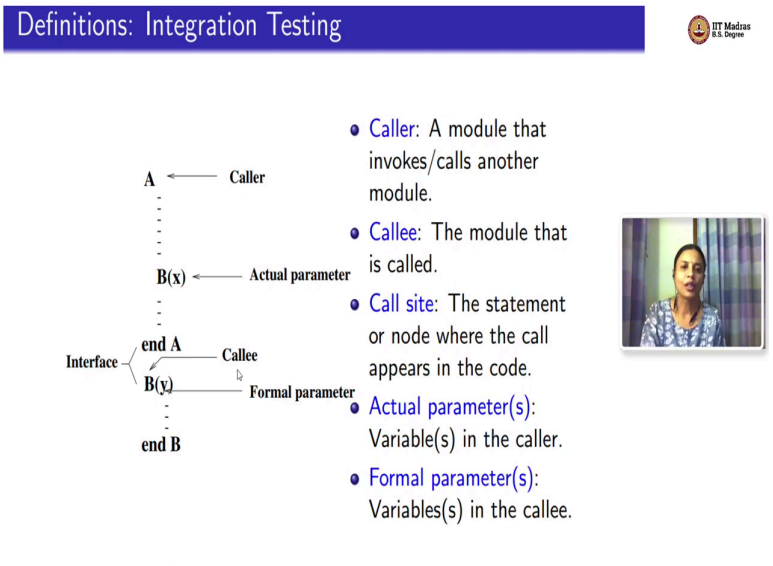
(Refer Slide Time: 06:19)



So, that is what we are going to spend a majority of our time on. So, when it comes to design integration testing, structural coverage may not reveal interesting faults, but data flow interfaces actually involve passing of data across a call interface. So, the data variables might change name, they will take a new name and the method that is being called return values might take new name, so we have to keep track of all these things. And data flow testing at call graphs is a very interesting area.

So, data flow interfaces amongst modules are complicated than control flow, as I told you, when you pass values, they can change names, they can get assigned to new variables internal to the other method. There could be several different kinds of interfaces. So, we have to figure out how the data changes. Definition in one module could be used in the next method. And something defined in that method could be passed back for use in the collie method and so on. So, let us understand the data flow interfaces and see how we can test for this.

(Refer Slide Time: 07:29)

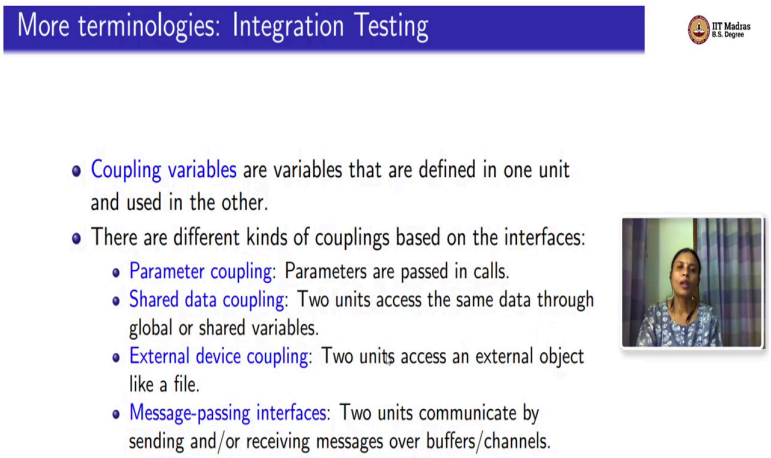


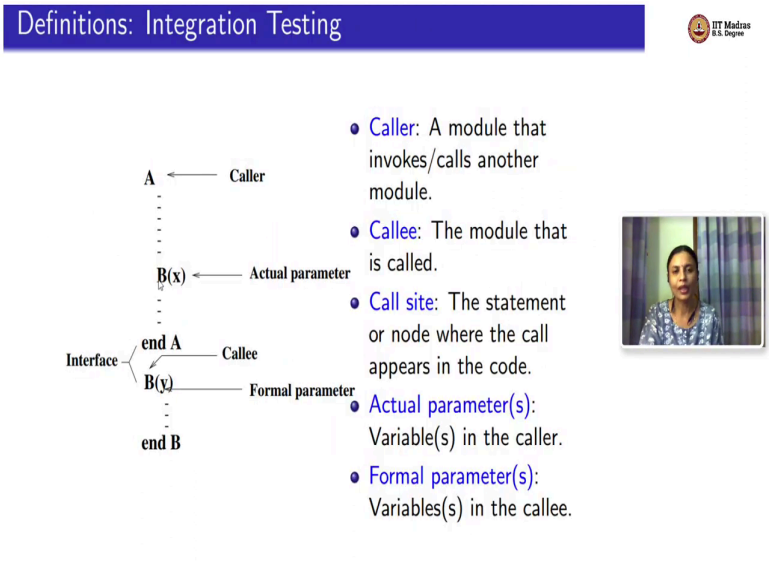
So, here is an example. Let us say there is a method or a function A and A has some code, these dash dash dash lead this is our code. We do not really want to know what it is but there is some sequence of statements in A. At some point as a part of one of his statements, function A calls another function B and passes a parameter x as a part of the call.

In terms of program execution, control now transfers to B through the call interface. x gets passed as y. The code for B executes and when B ends control gets passed back to A and the next statement of A after B. A resumed its execution from that statement. So, there are some terminologies that we use here. A is the caller which is a module that invokes or calls another module. B is the callee which is the module that is being called by another module. Call site is the actual statement or node where the call appears, it is this case in this case displays B of x.

Actual parameters are the variables that are being used in the call by the caller. This example x is an actual parameter. Formal parameter are the variables that are being used in the callee. So, y becomes the formal parameter here. So, these are terms that you have to remember when I do design integration testing. Caller, callee, call site and the parameters that are being passed, actual parameters in the collar and formal parameters in the callee.

(Refer Slide Time: 09:11)





So, now, what we say is the actual parameter here x is coupled with its formal parameter y. So, what is a coupling variable? A coupling variable is that variable which is defined in one unit and used to the other. It could be as a part of call interface. Typically, as a part of all interfaces, but there could be other ways of coupling also. coupling variables are variables that are defined in one unit and used to the other. Based on the kind of call interfaces, we could face several different kinds of coupling.

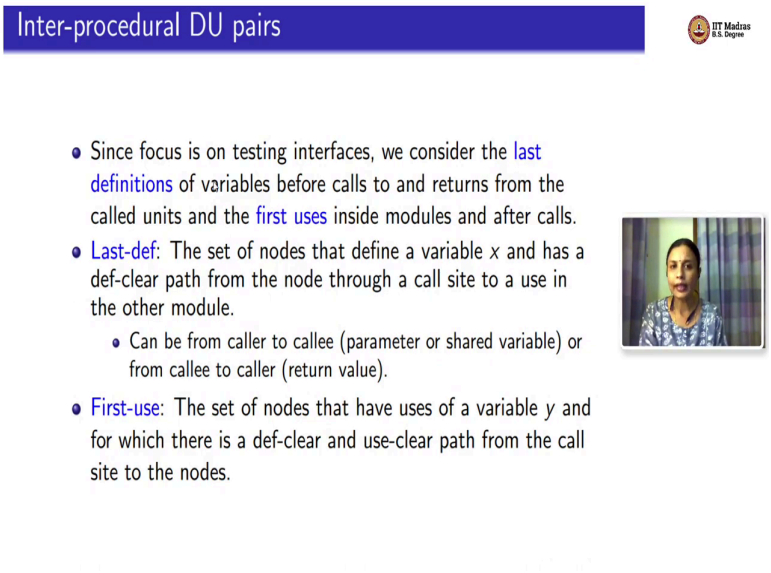
First one is parameter coupling, like we saw here. There is at some point A call B with actual parameter x and x gets passed as y to B. B runs and returns maybe some value to A. So, this parameter coupling is parameters, formal and actual parameters are passed across in calls. What we saw was an example of parameter coupling.

The other one is shared data copy, maybe there was a global variable, which was common to both A and B. A and B could be two different threads could be two different methods. But they had a global variable that was common to them. And they could talk to each other communicate with each other by reading from and writing to the global variable. So, that in that case, we say it is shared data coupling. Two units access the same variable, or the data in the variables by reading from and writing to global variable.

The next could be external device coupling, two units, two methods, two functions or procedures could access a completely external object, like a particular file found somewhere else in a server or in a database and things like that, that is called external device coupling. Two entities could also exchange by sending and receiving messages across dedicated buffers or channels, as we call it. So, those kinds of coupling or interfaces are calling Message Passing Interface, they also exist. So, there could be several different kinds of coupling that can be possible based on the interfaces based on the kind of code that you write.

Now, what we are going to bother is when coupling variables like for example, if I take here, x is coupled with y. I am interested in this relationship at the call interface.

(Refer Slide Time: 11:40)



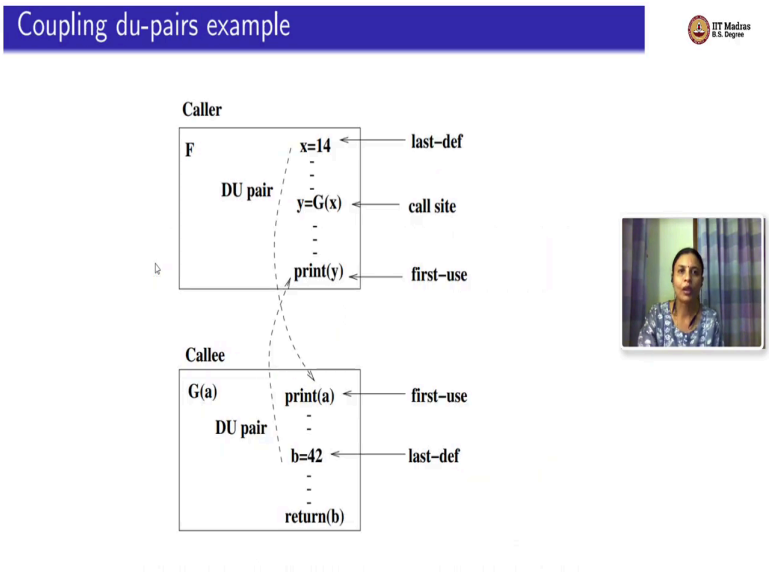
So, focus is on testing interfaces, what we consider is not every possible definition of x, we consider the last definition of x that occurred at the call just before the call interface and the returns from the call units. And we consider the first uses of the coupling variable inside the module after the call. So, what is the last definition? Last definition of a variable is as follows. The set of nodes that define a variable x and has a def clear path from the node

through a call site to a use in the other module, then you say that the last definition of a variable.

We will see an example in the next slide. So, take particular variable x that is going to be passed as a parameter, x get defined at several different points. But consider the last statement before the call where x was defined. After that there are no further re-definitions of x, that is called the last definition of x. x gets passed as a parameter. And its first use, maybe it gets renamed as y. And its first use on the other side is what we call the first use. So, the first use is a set of nodes that have uses of a variable y for which there is a definition clear and a use clear path across the call site to the other site.

So, amongst all the definitions of the parameters that are passed, the last definition is the one just before the call site, there are no further definitions. First use is the one that is just after the call site, there are no abuses before that and the callee module and no further re-definition. So, this last definition can be both way. It can be from the caller to the callee, or it can be the callee passing a return value back to the caller.

(Refer Slide Time: 13:43)



So, here is an example. So, let us say caller is this x, callee is a. So, F has several statements, let us say amongst these, this F is equal to 14 is one assignment statement that happens along the way. Then you have these dash dash dash some more statements at some point the call site is there. So, F calls a method G with parameter x, now control gets transferred to G, x gets renamed as a within G. So, the coupling pair is x comma a, x is coupled with a and one

of the first things that G does, for some reason in this example is to print a. So, this will be the first use of it.

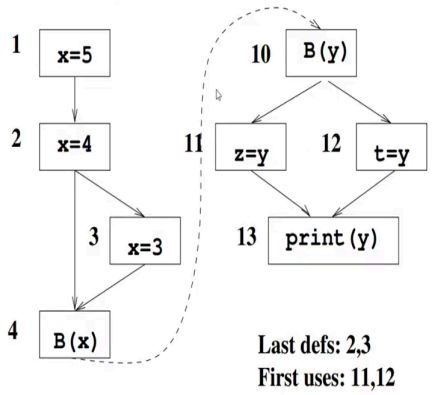
The last definition of x was the definition that was there before the call site. Assume that these three statements which I have not elaborated on. Do not redefine x. So, this was the last definition of x which assign x to 14. At the call site method G is called with x as a parameter. x is coupled with a. One of the first things that G does control gets transferred to G whereas the first thing that G does is to print a. So, this is the first use of a. Then G goes about with its execution.

At some point, let us say it returns the value b. Now we go back into G's code and say which was the last definition of b before this return happened. Let us say this is the last definition of b, b got assigned 42. So, because this is the last def of b. After this b is not redefined and b gets passed back in this return statement to the callee F. And what does b do? It gets coupled with y and it gets printed out by F. So, this is considered the first use for this b. So, in this example, F is the caller. G is the callee. Parameter x is coupled with a. And a, b, G, that is some code, G returns b, which gets coupled mean, which gets assigned to y at the call site.

And after that, it is printed. So, the last definition of b at 42 got first used here. Similarly, the last definition of x, which was 14, got first used and after the call site at print a. We do not consider last def and first used to be the actual call site. But the statements that occurred before the call site and after call site without any intervening definitions or users in between. I hope this is clear.

(Refer Slide Time: 16:28)

Last defs and first uses: Example

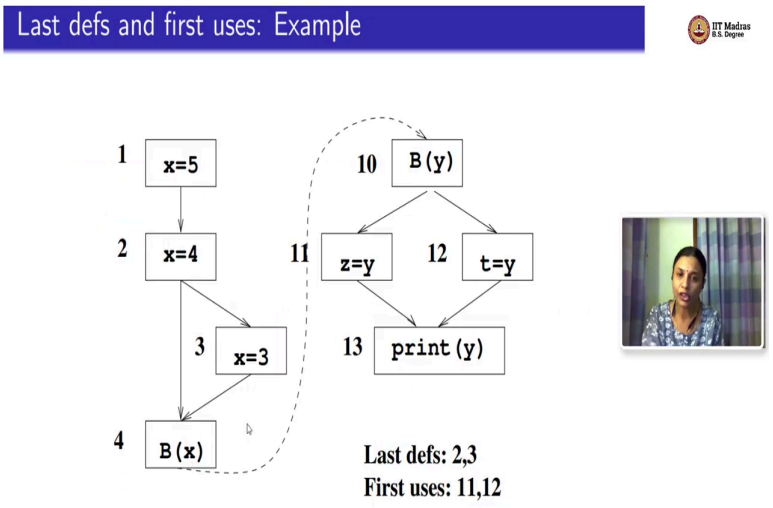


So, some more example. It is the same code that I took earlier. So, x is equal to 5, the same thing y is equal to 4 and then something happens method B x called. This one, this dashed interface is the call interface. x gets coupled with y. Things happen. And then y gets printed. So, the last definitions of x could be statement 2 or statement 3 based on whichever executed and the first uses are statements 11 or statement 12 in the function B. Is this clear, what a last definition and the first use is?

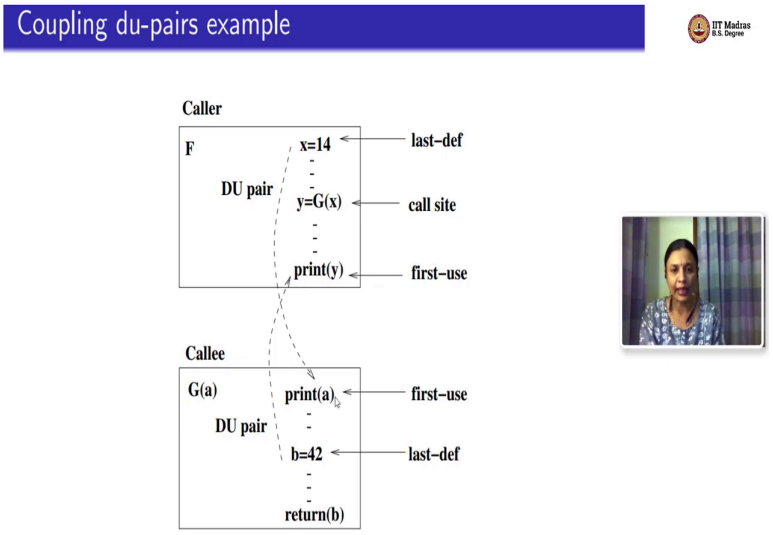
(Refer Slide Time: 17:08)

Coupling data flow coverage criteria

- A coupling du-path is from a last-def to a first-use.
- Data flow coverage criteria can now be extended to coupling variables:
  - All-coupling-def coverage: A path is to be executed from every last-def to at least one first-use.
  - All-coupling-use coverage: A path is to be executed from every last-def to every first-use.
  - All-coupling-du-paths coverage: Every simple path from every last-def to every first-use needs to be executed.
- The above criteria can be met with side trips.

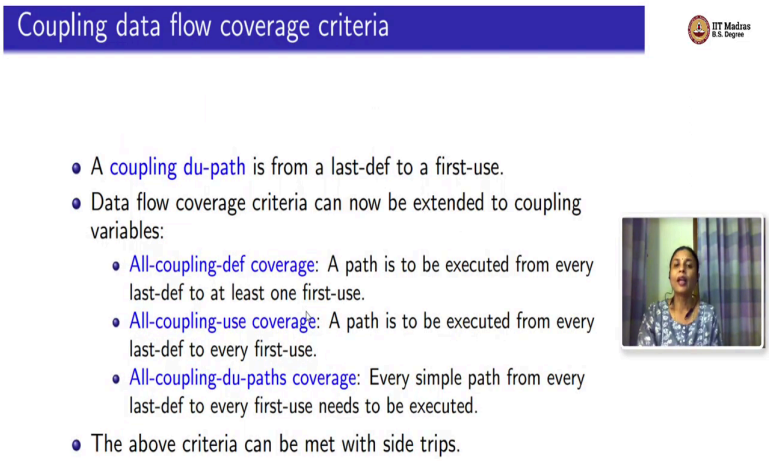


17:27



So, now we are interested in the same data flow coverage criteria that is tracking from definition to the use. But we are not going to do it for variables within the method. We are going to do it for coupling variables. Here x gets coupled with y. In this example, x gets coupled with a and b gets coupled with y. So, we are going to consider coupling variables which are basically the same variable. They just assume a new name and the other callee site. And we consider the last definition and the first use and this is the data that flows through the call interface.

(Refer Slide Time: 17:49)



So, our data flow coverage criteria is going to be bothered about du-path, definition use path but not from any variable to any other variable, but over some coupling variables. So, we say coupling du-path is a du-path from a last definition to a first use. Is this clear? A coupling du path is a du path from a last definition to a first use.

Now the same data flow coverage criteria carries over but for coupling variables using coupling du path. So, remember, we had all definitions coverage, all uses coverage and all du paths coverage last week in data flow coverage criteria.

They are going to take the same thing. And now say we say all coupling variables definitions coverage, that is a part is to be executed from every last definition to at least one first use. The next is coupling all coupling use coverage apart is to be executed from every last definition to every first use. Then all coupling du paths variables, consider all possible simple paths from every last definition to every first use. That is your test requirement.

So, same thing, all definitions all users and all du paths, add coupling variables and last definitions and first user. That is about what we are doing. And like and data flow criteria, if needed. If you phase infeasibility, you can meet this with side trips.

(Refer Slide Time: 19:29)

Example: Quadratic Root

```
1  // Program to compute the quadratic root for two numbers
2  import java.lang.Math;
3  class Quadratic {
4  private static float Root1, Root2;
5  public static void main (String[] argv)
6     { int X, Y, Z;
7     boolean ok;
8     int controlFlag = Integer.parseInt (argv[0]);
9     if (controlFlag == 1)
10    { X = Integer.parseInt (argv[1]);
11      Y = Integer.parseInt (argv[2]);
12      Z = Integer.parseInt (argv[3]); }
13    else
14    { X = 10; Y = 9; Z = 12
15      Y = 9;
16      Z = 12; }
17    ok = Root(X, Y, Z);
18    if (ok)
19      System.out.println(''Quadratic roots:''  + Root1, + Root2);
20    else
21      System.out.println (''No Solution'');
22 }
```

Example: Quadratic Root, contd.

```
23 // Three positive integers, finds quadratic root
24 private static boolean Root (int A, int B, int C)
25 {
26    double D;
27    boolean Result;
28    D = (double)(B*B)-(double)(4.0*A*C);
29    if (D < 0.0)
30    {
31      Result = false;
32      return (Result);
33    }
34    Root1 = (double) ((-B + Math.sqrt(D))/(2.0*A));
35    Root2 = (double) ((-B - Math.sqrt(D))/(2.0*A));
36    Result = true;
37    return (Result);
38 } // End method Root
39 } // End class Quadratic
```

Full code available here: https://cs.gmu.edu/~offutt/softwaretest/edition1/programs/ch02/Quadratic.java

More terminologies: Integration Testing

- Coupling variables are variables that are defined in one unit and used in the other.
- There are different kinds of couplings based on the interfaces:
  - Parameter coupling: Parameters are passed in calls.
  - Shared data coupling: Two units access the same data through global or shared variables.
  - External device coupling: Two units access an external object like a file.
  - Message-passing interfaces: Two units communicate by sending and/or receiving messages over buffers/channels.

Here is another example that I will walk you through. The full code of the example is available here in this URL. But I have put the code across two slides in a slightly incomplete way and will understand coupling data flow coverage criteria using this example. So, this is an example that computes the quadratic root for two numbers. You might want to recollect how what is the formula for quadratic roots with two numbers. It is this minus B plus or minus square root of B squared minus 4 AC by 2 A. That is the formula is the standard high school mathematics middle school mathematics formula that computes the quadratic root of equation.

So, this is a programme that computes quadratic roots for those two numbers. The class is called quadratic. These are the two numbers root 1, root 2 the floating point numbers. Then there is a main function where int x, y, z are considered. And then they are taken as inputs and some if there is some exception, then they are assigned some standard value. Now this main method calls this root method and display in line number 17, where you can see my mouse.

So, main call root with parameters x, y, z in line number 70. So, this is the call interface first. Let us go. Now control transfers to root. So, root, the found method root takes x, y, z, which are three integers as A, B and C. So, x is coupled with A, y is coupled with B, z is coupled with C. And this is just compute the quadratic roots. So, it is say it has a local variability and other local variable result. First, it says D is B squared minus 4 AC, the usual formula, that is what it says. If D is less than 0, it says sorry, I cannot compute roots.

Otherwise, it just uses the formula minus B plus or minus B squared square root of B squared minus 4 AC by 2A, that is 34 and 35. One is assigned to the return value root 1, the other is assigned to return value root 2, and the flag result is set to true and the result is the returned. Result is taken as okay back. So, the return variable result coupling variable is paired coupled with okay in line number 17. It could either be true or be false. It is false if quadratic roots cannot be computed, that is line number 31 executes. It is true if quadratic roots can be computed and line number 36 execute.

Along with this, the global variables, root 1 and root 2 are also assigned values. Root 1 is this line number 34. Root 2 is line number 35. So, here in this example, you see two kinds of coupling interfaces, one is parameter passing, x, y and z are passed as A, B and C. And then result, the Boolean variable is passed back as okay parameter passing coupling. The next is shared variable coupling. Root 1 and root 2 a global variables that are shared by both the

methods main and updated by this function called root which are actual values that is not passed and passed back, but they are directly written onto.

So, I will just for a minute, go back to that slide where I was talking about kinds of couplings. So, that example has parameter coupling and shared data. I hope this is clear.

(Refer Slide Time: 23:25)

Inter-procedural DU pairs

- Since focus is on testing interfaces, we consider the last definitions of variables before calls to and returns from the called units and the first uses inside modules and after calls.
- Last-def: The set of nodes that define a variable x and has a def-clear path from the node through a call site to a use in the other module.
  - Can be from caller to callee (parameter or shared variable) or from callee to caller (return value).
- First-use: The set of nodes that have uses of a variable y and for which there is a def-clear and use-clear path from the call site to the nodes.

Example: Quadratic Root

```
1  // Program to compute the quadratic root for two numbers
2  import java.lang.Math;
3  class Quadratic {
4  private static float Root1, Root2;
5  public static void main (String[] argv)
6     { int X, Y, Z;
7     boolean ok;
8     int controlFlag = Integer.parseInt (argv[0]);
9     if (controlFlag == 1)
10    { X = Integer.parseInt (argv[1]);
11      Y = Integer.parseInt (argv[2]);
12      Z = Integer.parseInt (argv[3]); }
13    else
14    { X = 10; Y = 9; Z = 12
15      Y = 9;
16      Z = 12; }
17    ok = Root(X, Y, Z);
18    if (ok)
19      System.out.println(''Quadratic roots:''  + Root1, + Root2);
20    else
21      System.out.println (''No Solution'');
22 }
```

Example: Quadratic Root, contd.

```
23 // Three positive integers, finds quadratic root
24 private static boolean Root (int A, int B, int C)
25 {
26    double D;
27    boolean Result;
28    D = (double)(B*B)-(double)(4.0*A*C);
29    if (D < 0.0)
30    {
31      Result = false;
32      return (Result);
33    }
34    Root1 = (double) ((-B + Math.sqrt(D))/(2.0*A));
35    Root2 = (double) ((-B - Math.sqrt(D))/(2.0*A));
36    Result = true;
37    return (Result);
38 } // End method Root
39 } // End class Quadratic
```

Full code available here: https://cs.gmu.edu/~offutt/softwaretest/edition1/programs/ch02/Quadratic.java

So, now let us just apply our data flow criteria to this code. And so the shared variables as I told you are root 1 and root 2. x, y and z are coupled with A, B and C. So, I am just mapping out the last definitions and the first uses for all these variables. Last definitions for x, y and z occur at lines 10, 11, 12 or at lines 14, 15 and 16. So, let us go back x is state x, y and z that taken as inputs in 10, 11 and 12. If there is an issue, then x, y and z are assigned some default value. That is the definitions for x.

Similarly, for result, it is line 31, which assigns result false or line 36 which assigns results to be true and as passed back. Lines 34 and 35 compute root 1 and root 2 as I do. First uses is the result is assigned back okay in line number 80 sorry, this should be line 18. Not line 17 and x, y, z are assign back values A, B, is this okay?

(Refer Slide Time: 24:42)

Quadratic Root Example: Coupling du-pairs

Legend: Pairs of locations as (method name, variable name, statement number)

- (main(),X,10) — (Root(),A,28)
- (main(),Y,11) — (Root(),B,28)
- (main(),Z,12) — (Root(),C,28)
- (main(),X,14) — (Root(),A,28)
- (main(),Y,15) — (Root(),B,28)
- (main(),Z,16) — (Root(),C,28)
- (Root(),Root1,34) — (main(),Root1,19)
- (Root(),Root2,35) — (main(),Root2,19)
- (Root(),Result,31) — (main(),ok,18)
- (Root(),Result,36) — (main(),ok,18)

So, now I am just listing the coupling du pairs by in a different style. So, it says method main calls the method root. X is coupled with A. X is defined at line 10. A is defined at line 28. Similarly, method main calls root for Y and from Z or it could be x, y and z in lines 14, 15 and 16, which are used in line 28.

Similarly, root 1 and root 2 are 34, 35 and 19. Result is coupled with okay. And these are the line numbers that match to coupling.

(Refer Slide Time: 25:19)

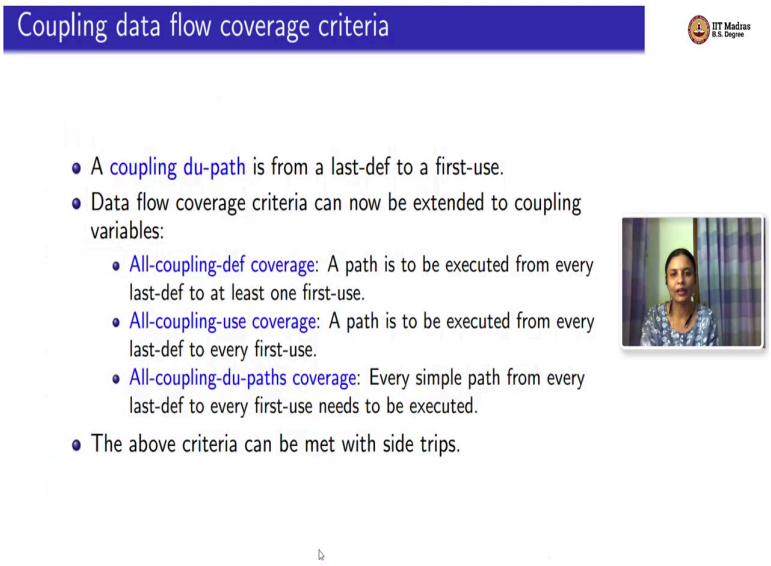
Coupling data flow criteria

- Only variables that are used or defined in the callee are considered for du-pairs and criteria.
- We need to consider implicit initialization of class and global variables. Such default initial values that are given need to be considered as definitions.
- Transitive du-pairs (A calls B, B calls C and there is a variable defined in A and used in C) is not supported in this analysis.
- For arrays, a reference to one element is considered as a reference to the entire array.

Quadratic Root Example: Coupling du-pairs

Legend: Pairs of locations as (method name, variable name, statement number)

- (main(),X,10) — (Root(),A,28)
- (main(),Y,11) — (Root(),B,28)
- (main(),Z,12) — (Root(),C,28)
- (main(),X,14) — (Root(),A,28)
- (main(),Y,15) — (Root(),B,28)
- (main(),Z,16) — (Root(),C,28)
- (Root(),Root1,34) — (main(),Root1,19)
- (Root(),Root2,35) — (main(),Root2,19)
- (Root(),Result,31) — (main(),ok,18)
- (Root(),Result,36) — (main(),ok,18)



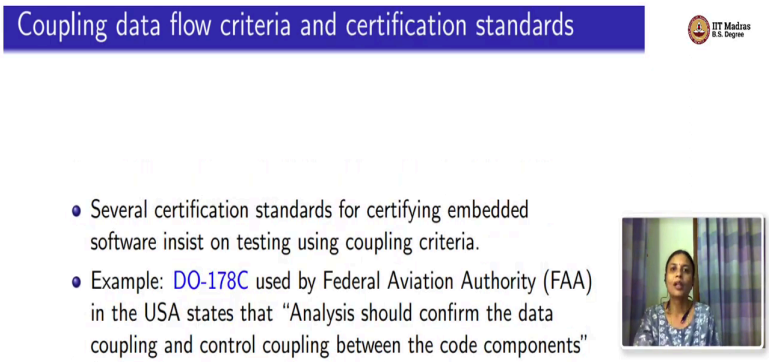
Now coupling data flow criteria, I mean, this is just about it. And you could define, you know, all defs criteria, these three criteria that we saw all defs, all uses and all du path criteria, you could define this.

Now, the variables just a small set of nodes before we wind up. Variables that are used or defined in the callee are considered for du pairs and criteria. Sometimes, even if you do not consider explicitly initialize a variable inside a method, we might have to consider an implicit initialization that happened for class variables, global variables and all. And these implicit initializations have to be considered as du pairs has to be considered as definitions for du pairs.

And sometimes they could be translated du pairs. By transitivity, we mean a scenario like this. There are three functions A, B and C. And it could be the case that A calls B, B calls C. And there might be a variable that is defined in A and used actually only in C. And it is very difficult to do coupling dataflow analysis for transitive pairs like these, we do not have automated support for this. You might have to do them manually. And a few other things in case you are passing an entire array as a parameter.

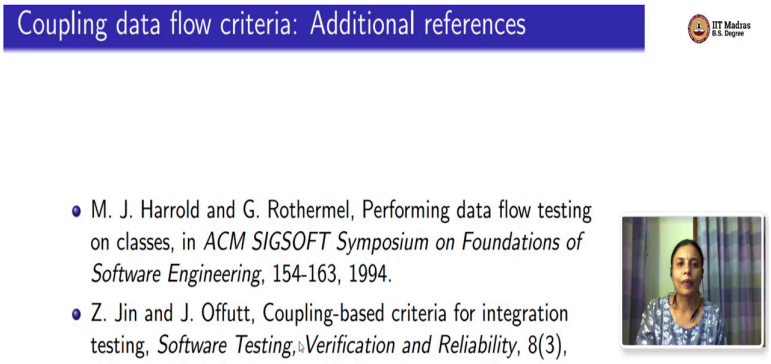
Please remember that reference to one element in the array is considered as a reference to the entire. This is what your IDE will end up doing.

(Refer Slide Time: 26:53)



So, coupling data flow criteria is actually a very important criteria and several certification standards for which people do certify software demand that we test for coupling data flow criteria. One such big standard is this DO-178C standard, which is used by Federal Aviation Authority. And I am just stating this line from that standard, just as an example to tell you that it is an important testing criteria, which is mandated to test it for.

(Refer Slide Time: 27:28)



Some references from which we took this material and you could refer if you want to know more information on data flow criteria, anyway, otherwise, this textbook is always there. So, I will stop here for now. We will continue in the next class.