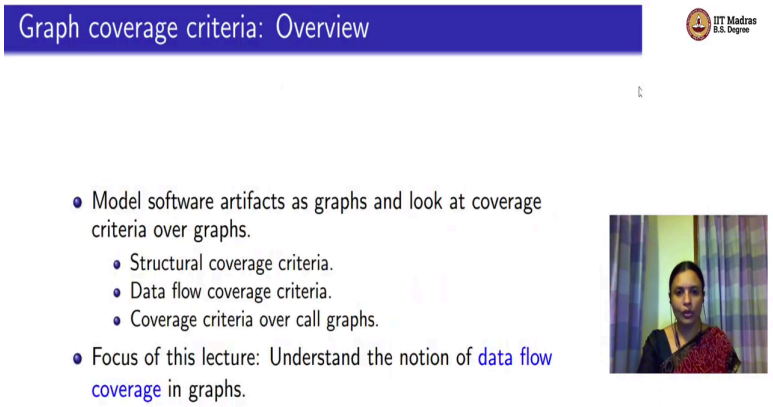
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Technology, Bangalore
Data Flow Graph Coverage Criteria: Applied to test code

(Refer Slide Time: 00:16)

Hello, everyone. Welcome back to week 3. We will continue with Graph Based Coverage Criteria. We saw the definition of data flow and also the data flow coverage criteria, 3 simple coverage criteria and also saw how they compare to the earliest structural coverage that we saw.

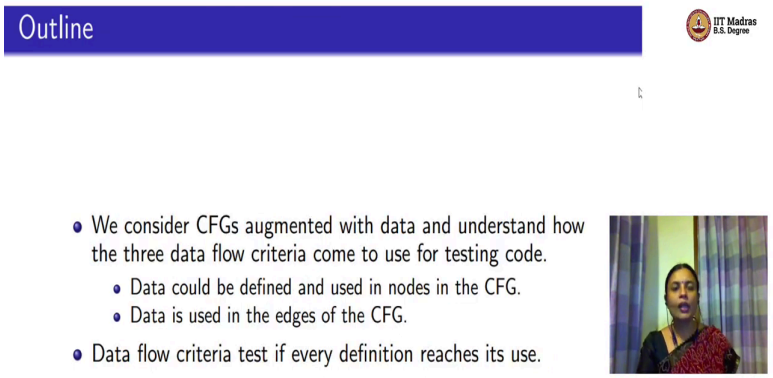
Now, our goal is to take like we did for structure and coverage, take an example of mandate with definitions and uses and learn how to apply the data flow coverage criteria that we learned to test this code.

(Refer Slide Time: 00:50)



So, this is the same overview that I have been keeping throughout these lectures will continue into it for some more time. So, we saw structure and coverage, we saw data flow coverage. And now we are going to apply data flow coverage criteria over an example. Later next week, we will see call graphs.

(Refer Slide Time: 01:11)

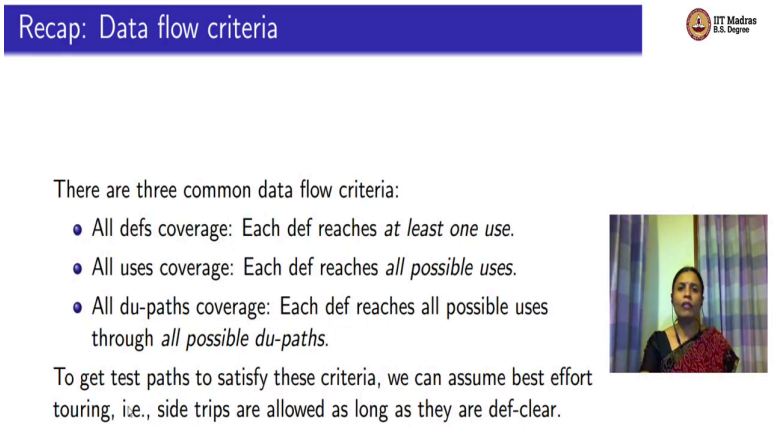


So, what did we do, we took the control flow graph, we augmented with data definitions and data uses and understood how the data flow criteria can be used to test code. Data flow criteria basically ensures that every variable that is defined reaches its use in the right way that it is meant to be used. Some recap from the definitions from the previous 2 lectures, or definition use path short form du-path with respect to a variable v, with a simple path that

begins with the definition of v and reaches the use v such that along the way, v is definition clear.

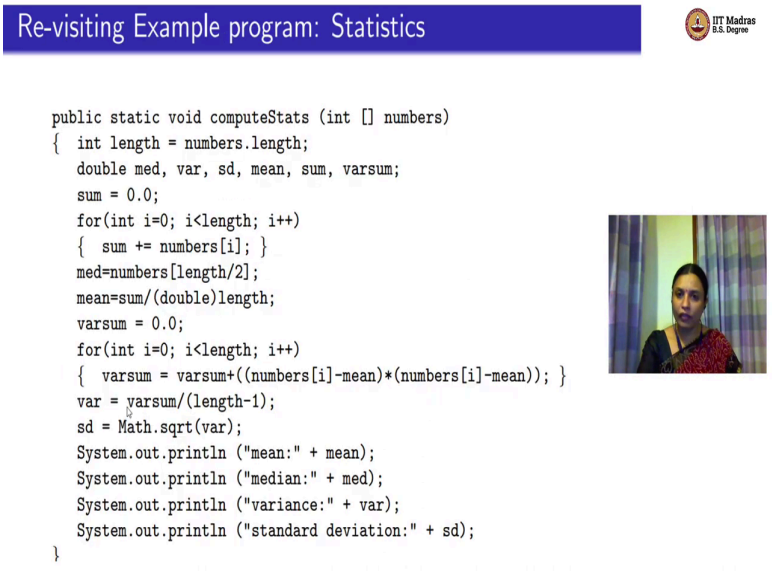
There are no more redefining definitions of v from the definition of v to the use of v. There could be intervening uses along the path. Sometimes, we write a d u n i, n j, v denote the set of all du-paths for the variable v from its definition at n i to the use at n j. We write d u n I, v. If we let go the uses but for the variable v if we want to consider all the du-paths starting at being defined at n i.

(Refer Slide Time: 02:31)



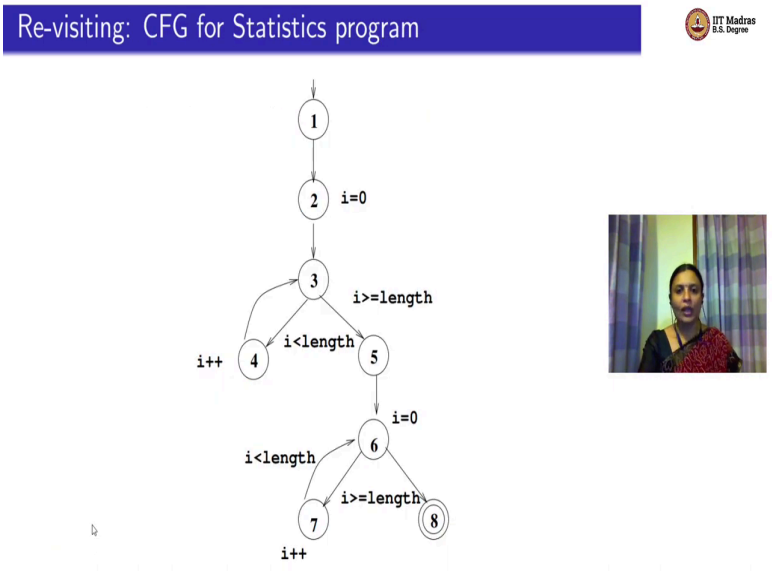
These are 3 common data flow criteria. All definitions coverage, which ensures that each definition reaches at least 1 use that does not go waste all uses coverage, which ensures that every definition reaches all possible uses. We write test set of test cases to check for every definition reaching all possible uses. All du-path coverage, where test requirement demands that for every definition, we consider all possible uses through all possible different paths from the definition to the uses. And like we do for structural coverage criteria, we always consider best effort touring. We permit side trips, as long as they are definition clear.

(Refer Slide Time: 03:18)



We saw this example earlier statistics program in the context of structure and coverage criteria. It is a program that takes an array of numbers and computes all these parameters median, variance, standard deviation, mean, sum and variant sum. So, it has one for loop to compute some uses that value to compute mean and median. Another for loop to compute varsum uses that to compute variance and standard deviation and prints out all the values.

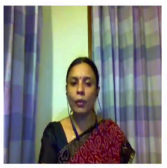
(Refer Slide Time: 03:51)



Re-visiting Example program: Statistics

IIT Madras B.S. Degree

```
public static void computeStats (int [] numbers)
{  int length = numbers.length;
   double med, var, sd, mean, sum, varsum;
   sum = 0.0;
   for(int i=0; i<length; i++)
   {  sum += numbers[i]; }
   med=numbers[length/2];
   mean=sum/(double)length;
   varsum = 0.0;
   for(int i=0; i<length; i++)
   {  varsum = varsum+((numbers[i]-mean)*(numbers[i]-mean)); }
   var = varsum/(length-1);
   sd = Math.sqrt(var);
   System.out.println ("mean:" + mean);
   System.out.println ("median:" + med);
   System.out.println ("variance:" + var);
   System.out.println ("standard deviation:" + sd);
}
```

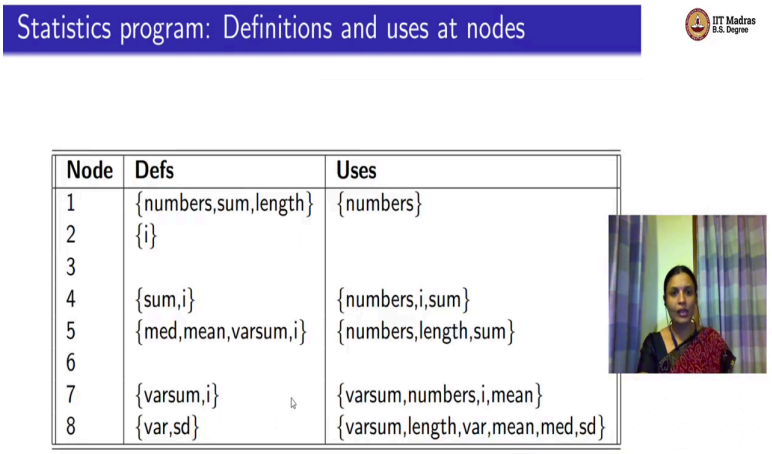


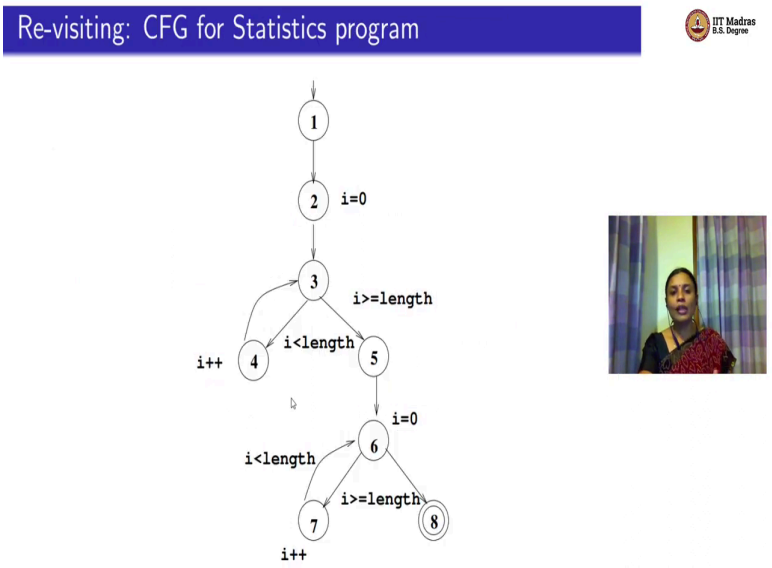
Remember, we had generated the control flow graph for this program. And the control flow graph basically looks like this. Statement number 1 is the initial path. 2, 3, 4 is the first for loop. 2, 3, 4, 3, 4, repeats it. 3, 5 exits the first for loop. 6, 7 is the next for loop. 6, 7, 8 exits the second for loop and 8 also represents a basic block where all the statements are printed. So, this is the control flow graph of that program. We will take the same control flow graph.

Look at all the variables that we have here. We have the array of numbers. Then we have all these statistical parameters that were computed median, variance, standard deviation, means, sum, variance sum. In addition to that, there are these array indices, the index i here that the first for loop uses and the same index i the second for loop again uses and all of them are printed. So, many variables are being defined and used in this program.

So, our goal is to take the control flow graph, augment it, the nodes at the edges of this graph with information about definitions and uses of the variables.

(Refer Slide Time: 05:03)





So, it gets too much to annotate the graph with definitions and uses like we did for pattern matching. Instead here I have followed the book and I have given you as a table. So, if you see this graph has 8 nodes node. 1 is the initial node goes on like this. Node 8 is the double circled final. So, per node in this table clearly see the first column this is the set of all nodes. Per node in this table, we have given what are all the variables that are defined in the second column and what are all the variables that are used in the third column. If you see the node number 1, which is right in the first node, which is the beginning of the fore loop, the variable array is defined that is what is given here.

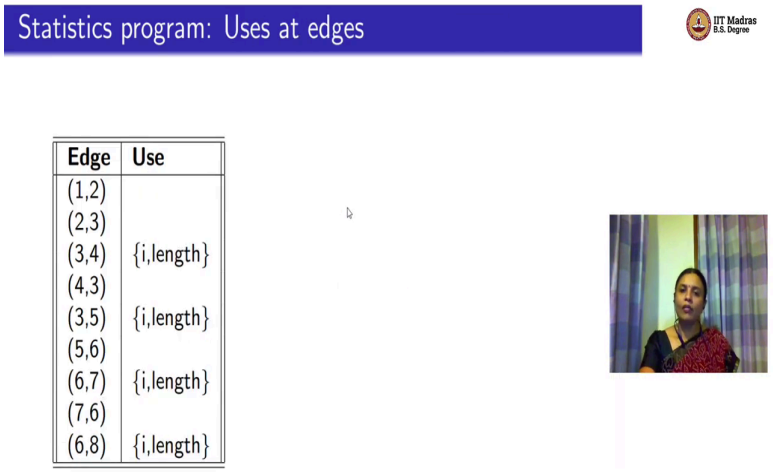
The variable sum is initialized and the length of the array is computed and uses length the variable array is used to compute the length of the array. Similarly, node 2 the for loop is

initialized. So, that is the variable i defined. Node 3 is the dummy node representing the beginning of for loop. Node 4, the variable some is defined i is also redefined. Numbers are used array numbers is used, i is used and sum is used to define sum. In variant statement number a node number 5 all these variables are computed mean, medians, variant sum is initialized and i is re-initialized.

So, they are defined again and the uses are the array is used. The length of the array is used and the value of the sum is used. 6 represents again the dummy node for that for loop. Similarly, for 7 varsum and i are defined. Varsum, numbers, i and mean I used. And 8 variance and standard deviation are defined and all these values are printed. So, they are used.

So, this is a table that takes such a control flow graph looks at the definitions and uses from a code like this and augments the node of this control flow graph with definitions and uses at the nodes.

(Refer Slide Time: 07:14)

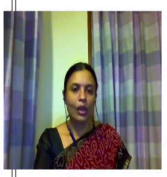


Will do a similar exercises for the edges. So, there are all these edges in the control flow graph. Edge 1, 2 has no use so it is left empty, so does edge 2, 3, it is also left empty. Edges 3, 4, 3, 5, 6, 7 and 6, 8 have all these variables that are being used. The length of the array and the array index. Edge 4, 3 you also uses the same thing because it is that back edge that goes back so they are not explicitly writing it. Similarly, for edge 6, 7 and 7, 6 also.

(Refer Slide Time: 07:50)

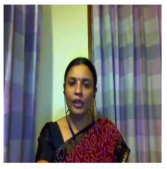
Statistics program: Definitions and uses at nodes

Node	Defs	Uses
1	{numbers,sum,length}	{numbers}
2	{i}	
3		
4	{sum,i}	{numbers,i,sum}
5	{med,mean,varsum,i}	{numbers,length,sum}
6		
7	{varsum,i}	{varsum,numbers,i,mean}
8	{var,sd}	{varsum,length,var,mean,med,sd}



Statistics program: Uses at edges

Edge	Use
(1,2)	
(2,3)	
(3,4)	{i,length}
(4,3)	
(3,5)	{i,length}
(5,6)	
(6,7)	{i,length}
(7,6)	
(6,8)	{i,length}



So, this table documents the definitions and uses corresponding to nodes. This table documents the uses corresponding to edges. As I told you earlier in control flow graph, we typically do not have definitions at edges. Now what we do is we mark the nodes and we mark the edges. Now we are going to pick up per variable what are the definition used pairs. So, the variable numbers which is the array input array gets defined at 4 mean defined at node 1 used at 4 and the definition at node 1 also gets used at 5 and the definition at 1 gets used at 7.

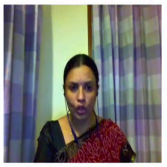
Similarly, the length of the variable array gets defined at 1 and gets used at nodes 5 and 8. And that edges 3, 4, edge 3, 5, edge 6, 7 and edge 6, 8. Similarly, the median variable gets defined at 5 use that 8. Variance and standard deviation gets defined and used at node 8.

Mean gets defined at node 5 and gets used at nodes 7 and 8. So, this table marks the definition use pairs for some of the variables.

(Refer Slide Time: 09:11)

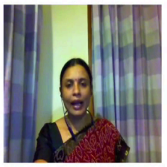
Statistics program: du-pairs, contd.

Variable	du pairs
sum	(1,4),(1,5),(4,4),(4,5)
varsum	(5,7),(5,8),(7,7),(7,8)
i	(2,4),(2,(3,4)),(2,(3,5)),(2,7),(2,(6,7)),(2,(6,8))
	(4,4),(4,(3,4)),(4,(3,5)),(4,7),(4,(6,7)),(4,(6,8))
	(5,7),(5,(6,7)),(5,(6,8))
	(7,7),(7,(6,7)),(7,(6,8))



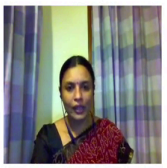
Statistics program: DU paths

Variable	DU pairs	DU paths
number	(1,4),(1,5),(1,7)	[1,2,3,4],[1,2,3,5],[1,2,3,5,6,7]
length	(1,5),(1,8)	[1,2,3,5],[1,2,3,5,6,8]
	(1,(3,4)),(1,(3,5)	[1,2,3,4],[1,2,3,5]
	(1,(6,7)),(1,(6,8))	[1,2,3,5,6,7],[1,2,3,5,6,8]
med	(5,8)	[5,6,8]
var	(8,8)	No path needed
sd	(8,8)	No path needed
mean	(1,4),(1,5)	[1,2,3,4],[1,2,3,5]
	(4,4),(4,5)	[4,3,4],[4,3,5]



Statistics program: du-pairs

Variable	du pairs
numbers	(1,4),(1,5),(1,7)
length	(1,5),(1,8),(1,(3,4)),(1,(3,5)), (1,(6,7)),(1,(6,8))
med	(5,8)
var	(8,8)
sd	(8,8)
mean	(5,7),(5,8)



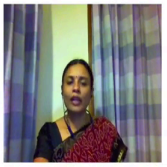
There are of course a lot more. So, we continue for the variable sum the edge gets defined at nodes 1 and 4. It gets used at nodes 4 and 5 respectively. Varsum gets defined that node 5 and use that node 7 and 8 and again gets defined that node 7 and gets used at node 7 and 8. i is this grand for loop index which gets used in both the for loops. So, all the definitions and uses for i are marked out here, is once used in the first for loop. So, define that 2 and used in all these places. And define that 4 and used to know all these places. Similarly, it is used at defined at 5 used in all these places and defined at 7 and used and all these places.

So, just to repeat these 2 tables, identify per variable, which are the node edge or node node pairs where the variables get defined and used. Now we go ahead and for these du-pairs that we identified in these tables here, per variable, we are going to mark the paths that take from the definition to the use. So, take the variable array of numbers, it i's Du-pairs were 1, 4, 1, 5 and 1, 7. The du-paths are 1, 2, 3, 4, which takes the definition at 1 to the use at 4. 1, 2, 3, 5, which takes the definition at 1 to the use at 5. And the definition at 1 gets used at 7 through this path 1, 2, 3, 5, 6, 7.

Similarly, we do it for all the other variables length, median, variance, standard deviation and mean. For variance and standard deviation. If you notice the definition and use come at the same node, which means it gets computed and printed. That is like one basic block that does the computation and the printing. So, we do not explicitly mark a du-path for these 2 variables. The rest of it is a fairly routine exercise.

(Refer Slide Time: 11:24)

Variable	DU pairs	DU paths
mean	(5,7),(5,8)	[5,6,7],[5,6,8]
varsum	(5,7),(5,8)	[5,6,7],5,6,8]
	(7,7),(7,8)	[7,6,7],[7,6,8]
i	(2,4),(2,(3,4)),(2,(3,5))	[2,3,4],[2,3,4],[2,3,5]
	(4,4),(4,(3,4)),(4,(3,5))	[4,3,4],[4,3,4],[4,3,5]
	(5,7),(5,(6,7)),(5,(6,8))	[5,6,7],[5,6,7],[5,6,8]
	(7,7),(7,(6,7)),(7,(6,8))	[7,6,7],[7,6,7],[7,6,8]

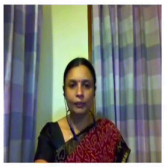


We continue for mean, the du-pairs 5, 7 and 5, 8 correspond to the du-paths 5, 6, 7, 5, 6, 8 same holds of varsum along with the du-path 7, 7 and 7, 8. Pair 7, 7 and 7, 8 which correspond to the du-path 7, 6, 7 and 7, 6, 8. And for i which knew that the number of du-pairs was large per pair there is a du-path that is listed out in the third column of this table.

(Refer Slide Time: 11:52)

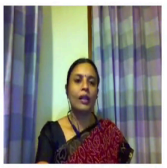
There are 38 du-paths for Stats, but only 12 of them are unique.

- Paths that skip a loop:
  - Four paths: [1,2,3,5], [1,2,3,5,6,8], [2,3,5], [5,6,8]
- Paths that require at least one iteration of a loop:
  - Six paths: [1,2,3,4], [1,2,3,4,6,7], [4,3,4], [7,6,7], [4,3,5], [7,6,8]
- Paths that require at least two iterations of a loop:
  - Two paths: [4,3,4], [7,6,7]



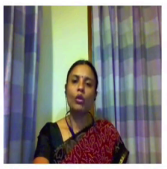
Statistics program: DU paths

Variable	DU pairs	DU paths
mean	(5,7),(5,8)	[5,6,7],[5,6,8]
varsum	(5,7),(5,8)	[5,6,7],5,6,8]
	(7,7),(7,8)	[7,6,7],[7,6,8]
i	(2,4),(2,(3,4)),(2,(3,5))	[2,3,4],[2,3,4],[2,3,5]
	(4,4),(4,(3,4)),(4,(3,5))	[4,3,4],[4,3,4],[4,3,5]
	(5,7),(5,(6,7)),(5,(6,8))	[5,6,7],[5,6,7],[5,6,8]
	(7,7),(7,(6,7)),(7,(6,8))	[7,6,7],[7,6,7],[7,6,8]

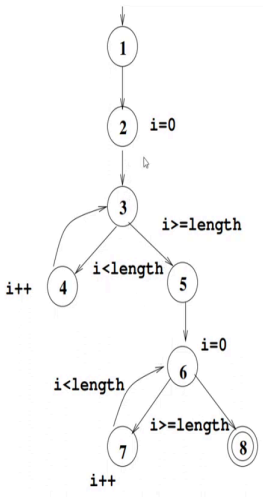


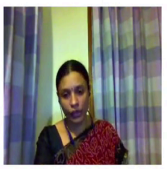
Statistics program: DU paths

Variable	DU pairs	DU paths
number	(1,4),(1,5),(1,7)	[1,2,3,4],[1,2,3,5],[1,2,3,5,6,7]
length	(1,5),(1,8)	[1,2,3,5],[1,2,3,5,6,8]
	(1,(3,4)),(1,(3,5)	[1,2,3,4],[1,2,3,5]
	(1,(6,7)),(1,(6,8))	[1,2,3,5,6,7],[1,2,3,5,6,8]
med	(5,8)	[5,6,8]
var	(8,8)	No path needed
sd	(8,8)	No path needed
mean	(1,4),(1,5)	[1,2,3,4],[1,2,3,5]
	(4,4),(4,5)	[4,3,4],[4,3,5]



Re-visiting: CFG for Statistics program





So, now, if you see if you count all these paths here, across these tables, you will realize that totally there are 38 different du-paths for statistics, each one corresponding to one variable, but several of them are duplicates. If you see 5, 6, 7 comes as a du-path for both mean and for varsum. 5, 6, 8 also comes as the du-path for mean varsum. In fact, it again comes as a du-path for the variable i below. So, many duplicates are there. If we remove the duplicates, we can count 12 unique du-paths for all the variables in the statistics program.

So, there are so many variables that are listed in this table. Let us count number, length, median, variance, standard, deviation, mean. Six of them here. And 3 more, 9 variables. This is the a du-pairs, the nodes and the edges where they get defined and used. These are the paths that take you from the definition to the uses. And some paths are repeating. We remove the repetitions we get 12 unique du-path for all the 9 variables. I am going to bucket that paths into 3 categories for a clarity.

The first category is paths that skip the loop. You remember, we had 2 for loops. These are paths that skip the loop, 1, 2, 3, 5. 1, 2, 3, 5, 6, 8 skips both the loops. 2, 3, 5 and 5, 6, 8 are also paths that skip the loop. The loops were down here, let me just go up once that we have them clear loops are from 3 to 4 and then from 6 to 7. So, when I do 1, 2, 3, 5, I skip the first for loop. When I do 1, 2, 3, 5, 6, to 8, I skip the second for loop also. That is what is given here in this slide. 4 paths skip the loop. 6 paths out of these 12 du-paths that require at least 1 iteration of the loop. 1, 2, 3, 4 interest the first for loop. 1, 2, 3, 4, 6, 7 comes out of the first for loop after iterating it once. 4, 3, 4 can be used to iterate over the for loop.

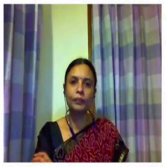
Similarly, 7, 6, 7 can also be used to iterate over the second for loop. 4, 3. 5 and 7, 6, 8 are du-paths that are used to exist the first for loop and the second for loop. This is at least one iteration. You can repeat the fore loops as I told you by using these du-paths 4, 3, 4 and 7, 6, 4. So, these paths like in other prime paths can be repeated to get two or more iterations of the while loops. So, this bucketing is useful because then we know what each kind of du-path means as far as testing loops are concept.

So, now we will go ahead and generate test cases. We will generate a set of test paths for those that skip a loop, generate a set of test cases as paths for those that require at least one iteration of the loop and generate set of test paths as test cases, for those that require at least two iterations of the loop.

(Refer Slide Time: 15:21)

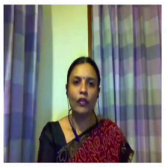
Statistics program: Test case #1

- Test case: numbers = (44), length = 1.
- Test path: [1,2,3,4,3,5,6,7,6,8].
- Additional DU paths covered (without side trips): [1,2,3,4], [2,3,4], [4,3,5], [5,6,7], [7,6,8].
- Note: All these require at least one iteration of the two loops.



Re-visiting Example program: Statistics

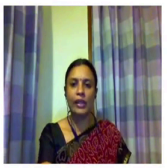

```
public static void computeStats (int [] numbers)
{  int length = numbers.length;
   double med, var, sd, mean, sum, varsum;
   sum = 0.0;
   for(int i=0; i<length; i++)
   {  sum += numbers[i]; }
   med=numbers[length/2];
   mean=sum/(double)length;
   varsum = 0.0;
   for(int i=0; i<length; i++)
   {  varsum = varsum+((numbers[i]-mean)*(numbers[i]-mean)); }
   var = varsum/(length-1);
   sd = Math.sqrt(var);
   System.out.println ("mean:" + mean);
   System.out.println ("median:" + med);
   System.out.println ("variance:" + var);
   System.out.println ("standard deviation:" + sd);
}
```



Statistics program: Du-paths without duplicates

There are 38 du-paths for Stats, but only 12 of them are unique.

- Paths that skip a loop:
  - Four paths: [1,2,3,5], [1,2,3,5,6,8], [2,3,5], [5,6,8]
- Paths that require at least one iteration of a loop:
  - Six paths: [1,2,3,4], [1,2,3,4,6,7], [4,3,4], [7,6,7], [4,3,5], [7,6,8]
- Paths that require at least two iterations of a loop:
  - Two paths: [4,3,4], [7,6,7]



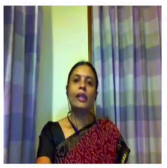
Let us start with the simple case. Let us say array is just the single element array. Let us say it has the number 44. And length of this array is a single array is a single element array. So, length is 1. The test path that this test case will cover will be this one 1, 2, 3, 4, 3, 5, 6, 7, 6, 8. That is, if I give let me go back to the code for a minute for your sake. If I give value of this numbers as single element array 44, compute the length as 1 and let this test case run. And on the CFG, the path that this test case is going to trace is a path that looks like this. It will enter the first fore loop; it will enter the second for loop. And it will enter both these at least once. exactly once, in fact, because this array is of length 1.

So, this test path will cover these du-paths 1, 2, 3, 4, 2, 3, 4, 4, 3, 5, 5, 6, 7, 7, 6, 8. From this list, some of these will be covered here, at least one iteration of the loop, the second list here.

(Refer Slide Time: 16:37)

Statistics program: Test case #2

- Test case: numbers = (2,10,15), length = 3.
- Test path: [1,2,3,4,3,4,3,4,3,5,6,7,6,7,6,7,6,8].
- Additional DU paths covered (without side trips): [4,3,4], [7,6,7].
- Note: All these require at least two iterations of both the loops.



Statistics program: Du-paths without duplicates

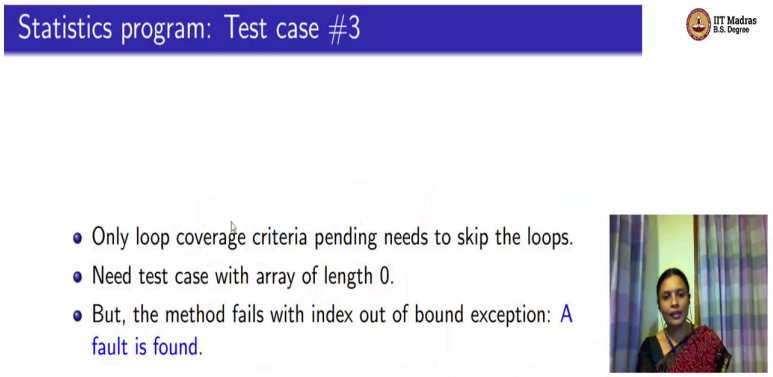
There are 38 du-paths for Stats, but only 12 of them are unique.

- Paths that skip a loop:
  - Four paths: [1,2,3,5], [1,2,3,5,6,8], [2,3,5], [5,6,8]
- Paths that require at least one iteration of a loop:
  - Six paths: [1,2,3,4], [1,2,3,4,6,7], [4,3,4], [7,6,7], [4,3,5], [7,6,8]
- Paths that require at least two iterations of a loop:
  - Two paths: [4,3,4], [7,6,7]

Similarly, suppose I give another test case. In this case, I am giving my array of numbers is an array of 3 elements, 2, 10 and 15. So, its length is 3. And if I run the code on this input is going to take a longer test path. Here, it is going to cover both loops, and take at least 2 iterations of each of the loops. And in this process, the du-paths that it is covering from this list are these 2 paths that are given at the end.

So, now, if you see we have test cases, for paths that require at least 1 iteration of the loop, which was a singleton element array, we have test cases for paths that require at least 2 iteration of the loop, which was at 3 element array. In this slide, what is left out is a test case for paths that skip the loop. So, let us give it a test case.

(Refer Slide Time: 17:30)

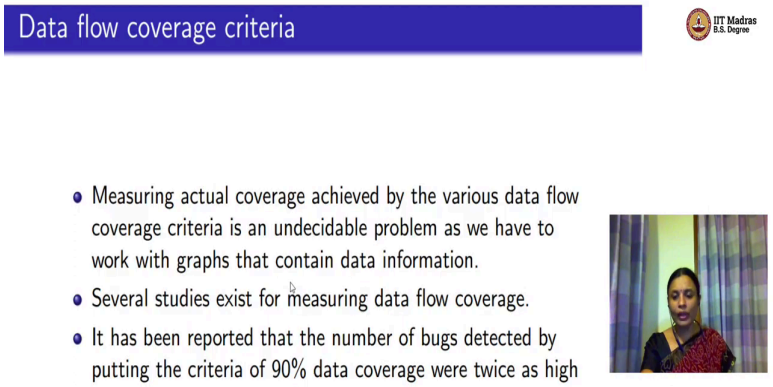


Only loop coverage criteria pending needs to skip the loops. So, which means to skip the loop, I need a test case, which will be an array of length 0. So, that is also a possible input perfectly valid input, but then there lies a problem in the code. If you give an array of length 0, the method will fail with an index out of bounds exception, so a fault is found. So, what have we achieved, by listing the data flow definitions and uses and the du-paths and trying to bucket the du-paths into various categories and generate test cases for each of those du-paths the test requirements, we have found that in the while catering to du-paths that skip a loop, we are getting an exception in the method.

And so the method has an error, this error could have been found independently by you, if you look at the code. We realize that in corner cases, which is an array of length 0, it does the wrong divisions and it does not know how to handle it, it does not have an exception handling mechanism. And that is precisely the error that is there in this program.

Of course, we found about found out this error in a slightly longish way, by doing data flow testing, but none the less, the error that is found is a valid error.

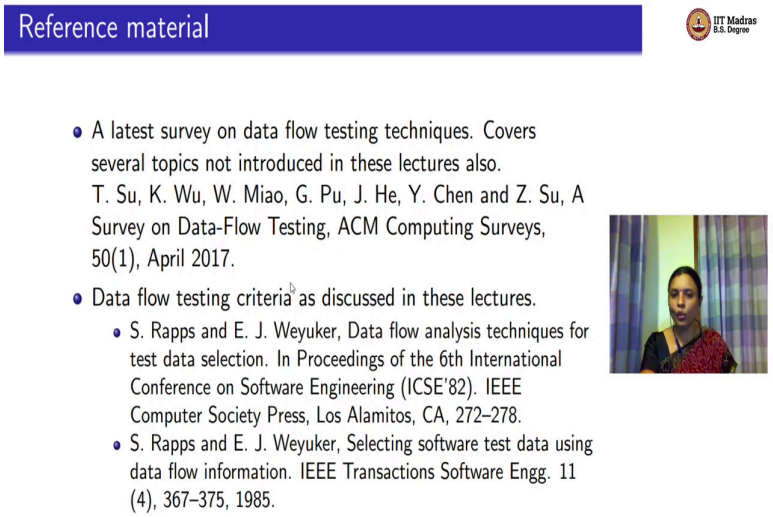
(Refer Slide Time: 18:51)



So, I hope this slide and this example would have illustrated the use of data flow coverage criteria for you. Here we did it for this example. But in general, suppose I have a piece of program. Can I do this in a systematic way? And can I measure how much data flow coverage criteria I have achieved? This is unfortunately a difficult problem or bordering program analysis and software testing. And in general, it is an undecidable problem in the sense that we do not have an automated algorithm that will compute all the du-paths automatically and measure coverage for data flow criteria.

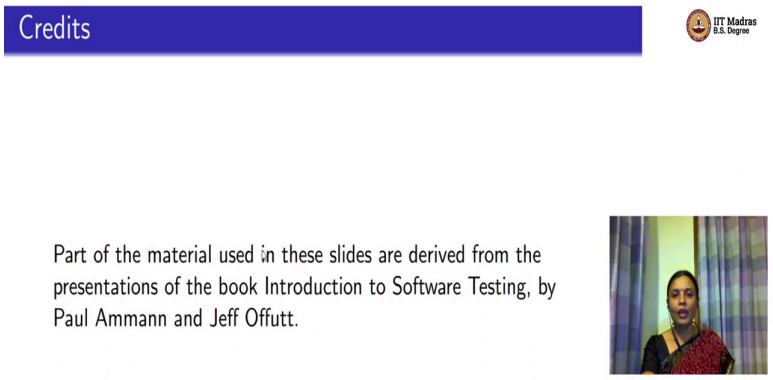
There are several heuristics based studies that exist that will measure data flow coverage and empirically studies that do statistical based surveys and user studies in software engineering have found that if I do data flow criteria based testing and achieve a high coverage rate like 90 percent, then I am more effective in finding bugs than let us say I say branch coverage or edge coverage is my criteria. And I try to achieve high coverage criteria for branch coverage. It is known that data flow coverage is an effective way of testing code.

(Refer Slide Time: 20:11)



Whatever I told you today is partly taken from this Rapps and by Weyuker's paper as I had told you earlier. And just to reiterate, if you would like to know more about testing, feel free to pick up and read this ACM Computing service paper.

(Refer Slide Time: 20:26)



I will stop here for now. But I urge you to take small methods of code yourself and try to work out the control flow graph, augment it with definitions and uses, take the du paths and then try to come up with test cases yourself. We will work out a few exercises as we move on. Thank you.