Software Testing

Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 15
Data flow graph coverage criteria: Applied to test code

Hello again, now we are in week 4. This is the first lecture of week 4. What are we going to do today? Today I will complete testing of source code using the graph models. If you remember last week we took source code then we looked at control flow graphs over source code and then we saw how the various structural coverage criteria applied for these. What we will be doing today is take source code again, but instead of looking at control flow criteria alone we will consider the CFG or the control flow graph which is augmented with defs and uses of data, and see how the data flow criteria that we had learnt about last week applies to testing of source code.

(Refer Slide Time: 00:56)

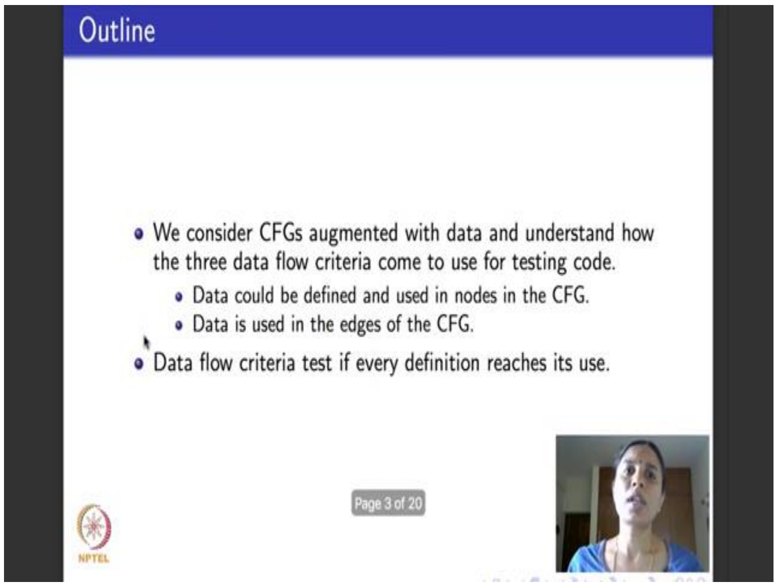
Graph coverage criteria: Overview

- We look at graphs as structures and look at the following two coverage criteria over graphs.
  - Structural coverage criteria.
  - Data flow coverage criteria.
- Later, we consider software artifacts (code, design elements and requirements) modelled as graphs and see how these coverage criteria apply to them.
- Focus of this lecture: Using data flow coverage over graphs to test source code.

This is what we have done till now it is a summary of what we have done. So, we have looked at graphs the structures and we have learnt about 2 kinds of coverage criteria: structural coverage criteria and data flow coverage criteria. We learnt various coverage criteria, see how they subsumed, then what we saw last week was we took source code, we learnt how to draw CFG modules for each bits of design construction, source code and then we took an example, we took the whole CFG for that example and learnt how to apply structural coverage criteria to test that source code by applying the coverage criterion the CFG.

Today what we will do is we will take the source code, take the CFG augment the CFG with defs and uses and see how to apply the 3 data flow criteria that we learnt on those things to test the source code.

(Refer Slide Time: 01:46)



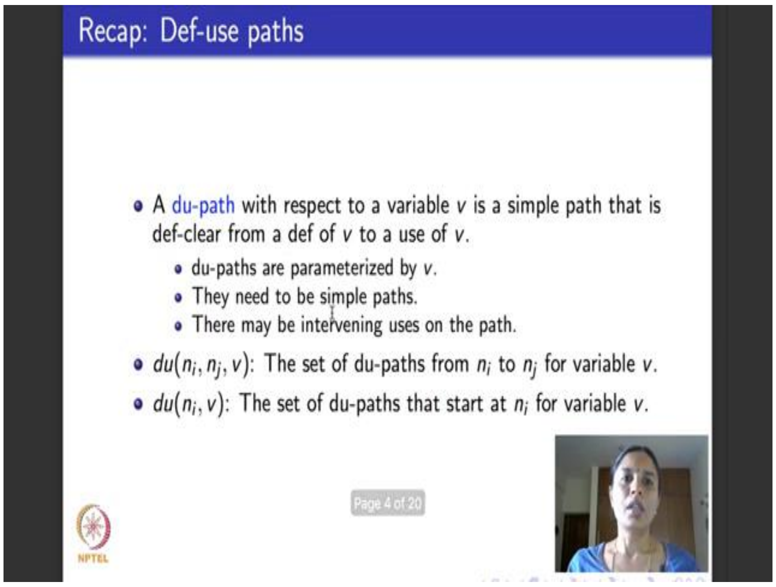
Outline

- We consider CFGs augmented with data and understand how the three data flow criteria come to use for testing code.
  - Data could be defined and used in nodes in the CFG.
  - Data is used in the edges of the CFG.
- Data flow criteria test if every definition reaches its use.

Page 3 of 20

So, to start with I will recap data flow coverage criteria, and then like we did in the previous example, we will take one example code, draw a CFG, look at the defs and uses and see what applying data flow coverage criteria to them will mean. So, as I told you we look at CFGs augmented with data, data is augmented as definitions and uses definitions and uses. Definitions and uses occur in the nodes of the CFG, there are no typically definitions in the edges of this CFG when it comes to modeling code; definitions come only a nodes edges have uses in them. So, what are data flow criteria’s goal? It is to be able to write test paths that check if definition reaches is used in 1 way or the other, right.

(Refer Slide Time: 02:31)



Recap: Def-use paths

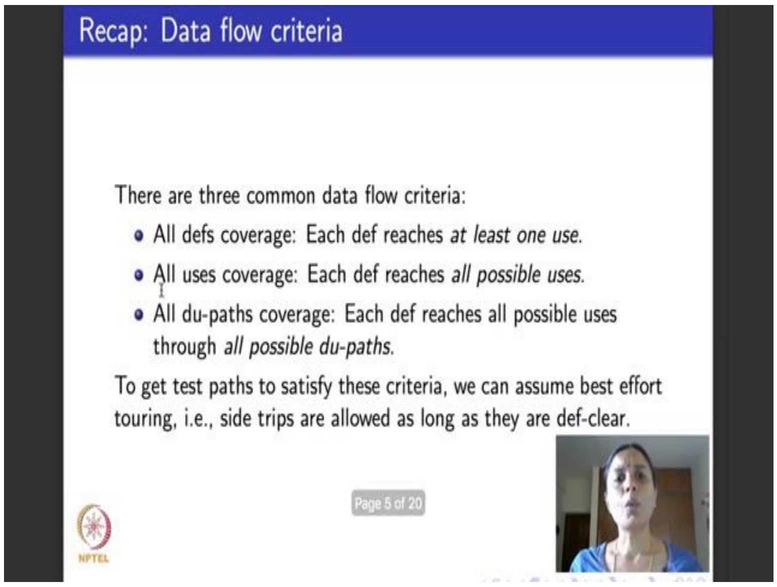
- A du-path with respect to a variable v is a simple path that is def-clear from a def of v to a use of v.
  - du-paths are parameterized by v.
  - They need to be simple paths.
  - There may be intervening uses on the path.
- $du(n_i, n_j, v)$: The set of du-paths from n_i to n_j for variable v.
- $du(n_i, v)$: The set of du-paths that start at n_i for variable v.

Page 4 of 20

So, we recap a few basic concepts about data flow criteria that we learned from the last weeks lectures, we learnt what is called a def use path abbreviated as d u-path; d u-path are with always reference to a fixed variable b.

What sort of paths they are? The first thing to notice that they are simple paths they do not have any cycles, and they trump from a node that contains a definition of v to a node or an edge that contains a use of v. What we insist is that in between these 2 definition and the corresponding use, there are no further definitions of v in all the intermediate vertices along the simple paths. So, this is what is written here we have parameterized by the variable v, they need to be simple, and that there need to be no intermediate definitions, but they could be intervening uses,we really do not worry about that, and for a variable v if there is a du-path going from n_i to n_j we write it as du (n i, n j, v) for a variable v the set of d u paths beginning at the node n_i is written as du (n_i, v).

(Refer Slide Time: 03:40)



Recap: Data flow criteria

There are three common data flow criteria:

- All defs coverage: Each def reaches *at least one use*.
- All uses coverage: Each def reaches *all possible uses*.
- All du-paths coverage: Each def reaches all possible uses through *all possible du-paths*.

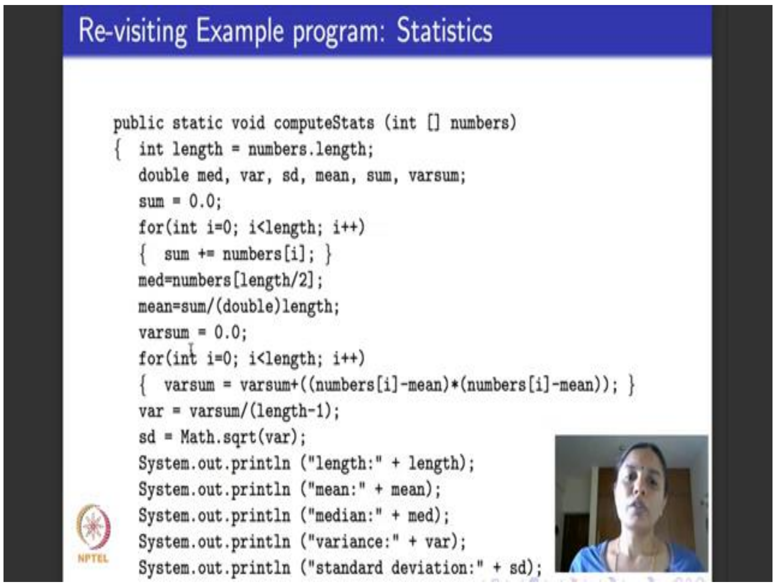
To get test paths to satisfy these criteria, we can assume best effort touring, i.e., side trips are allowed as long as they are def-clear.

Page 5 of 20

What are the 3 dataflow criteria that we learnt? Not as many a number of structural coverage criteria only 3 in number. So, if you remember the 3 criteria deal with how the definition goes to its use. The first one was called all defs coverage, it basically insisted that each definition reaches at least one use; it basically insists that the variable is never defined and not used at all. The second criteria says all uses coverage; it insists that each definitions reaches all possible uses. The third criteria called all du-paths coverage says that each definition reaches all possible uses using all possible different paths that take the definition tools use, and another thing to remember is that when we consider these 3 data flow criteria what are called test requirements, right.

So, when we consider test paths to test these test requirements, we assume that the test paths can be taken with what is called best effort touring. So, which means you are allowed side trips and detours as long as it remains to be a test path and you can use the test paths to satisfy data flow coverage criteria. Oe extra requirement that we have about this side trips that could come in best effort touring, is that this side trip should still be definition clear that is always a condition that we insist.

(Refer Slide Time: 05:09)



Re-visiting Example program: Statistics

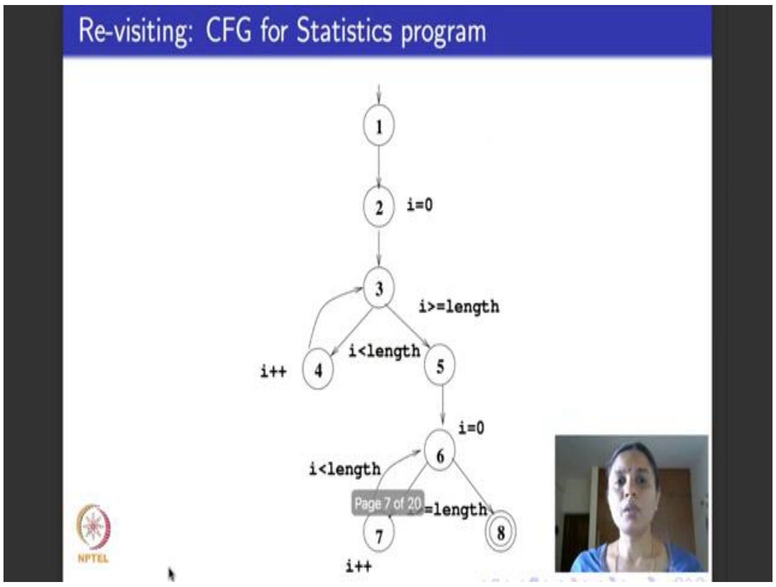
```
public static void computeStats (int [] numbers)
{  int length = numbers.length;
   double med, var, sd, mean, sum, varsum;
   sum = 0.0;
   for(int i=0; i<length; i++)
   {  sum += numbers[i]; }
   med=numbers[length/2];
   mean=sum/(double)length;
   varsum = 0.0;
   for(int i=0; i<length; i++)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

So, if you remember we had looked at the statistics examples when we had seen the control flow graph and how to draw it, I am recapping the same example this is the code for the same statistics program.

We will quickly recap what it does, it takes an array called numbers and then it is idea is to compute various statistical parameters about this array. What are the various statistical parameters that it computes? It computes the median, variance, standard deviation means sum and the variance and how does it go about doing? It is very simple this formulae as straight forward initializes the sum to 0, computes the sum in a for loop, then it computes the median mean and then it initializes varsum to 0 uses another for loop to compute the varsum and then it computes the variance and standard deviation, and it prints all the values that it is computed, right.

So, this is how the program looks like. So, if you remember the control flow structure of the program, there are 2 for loops one here and one here and the whole set of assignment statements before the first for loop, 3 statements in between the 2 for loops, and a few statements and belong with printouts after the third for loops.

(Refer Slide Time: 06:18)



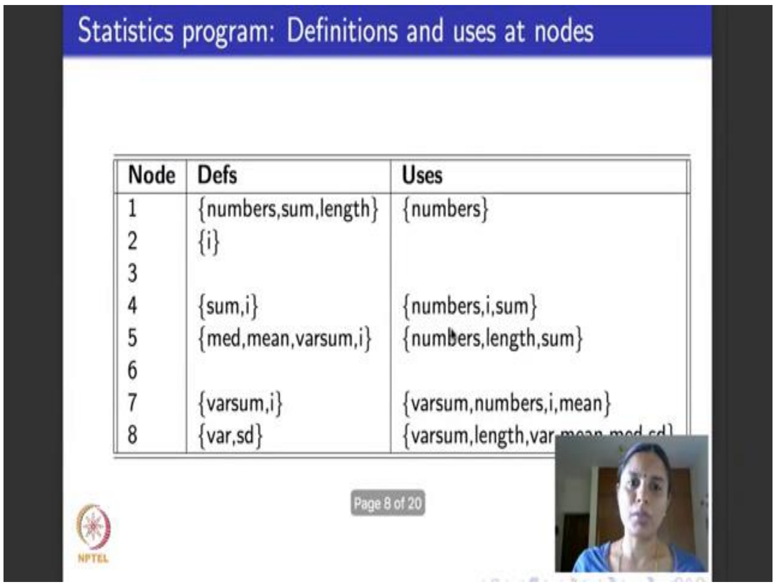
Re-visiting: CFG for Statistics program

So, here was the control flow graph for the statistics program that we had drawn last time. Node 1 is where we begin and the node 2 initializes the for loop, node 3 is this dummy node corresponding to the CFG of the first for loop, this (3, 4) structure here enters and executes the first for loop.

This 3 to 5 means I have finished computing the sum here in the first for loop, and move on to compute the median, mean, variance sum and then go on to the second for loop. So, that is this path--- 3 to 5, I do all the other initializations the other computation, sorry, median and other things here at node 5, and then I move on to node 6 where I begin the second for loop. This cycle of length 2 between 6 and 7 executes the second for loop, and when I take from 6 to 8 I finished my execution, 8 is the node where I go ahead compute this var, standard deviation and do all these printouts.

So, in this CFG that we have drawn here corresponding to the statistics program, we have given labels corresponding only to the length. I have retained the same structure that we used it in the last module, we have not given all other labels just to reduce clutter. But I hope you remember what labels node 1. The statements that label node 1 are all these statements before the beginning of the for loop, 2 begins the for loop 3 to 4 executes the for loop 3 to 5 comes out of the for loop, at node 5 we do all these computations--- median, mean, initialized varsum; node 6 begins the second for loop the cycle 6 to 7 executes the second for loop, 6 to 8 exits the for loop, at node 8 we compute var, standard deviation and do all these printfs, right. So, that is the CFG.

(Refer Slide Time: 08:23)



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}

Page 8 of 20

So, if you look at the CFG it depicts plane control flow graph. My goal is to be able to augment the CFG with data information specifically with information about definitions and uses of data, that is what we are going to be able to do it. Again what I have done is have drawn tables, where I have taken nodes from the CFG we will go back to the control flow graph, if you see the nodes are numbered 1, 2, 3, 4 and so on up to 8. At each node certain variables are defined and certain variables are used. So, this table lists which are the variables that are defined at each of the nodes and which are the variables that are used at each of the nodes.

So, if you see it says at node 1, three variables are defined. What are the 3 variables, numbers sum and length. So, let us go back and see how they are defined at node 1. So, this is node 1 in the CFG, it marks the node where all statements that happen before the execution of the first for loop are. So, we go back one slide which are the statements these are the statements there is this length is numbers.length by compute the length of the array, I do all these initial the declarations and then I initialize. So, it says node 1 represents that and the variables that are defined at node 1 are these array numbers which is taken as input, sum which was initialized to 0 and length which computes the length of the array.

So, that is what these statements correspond to numbers which was taken as input, length which initially which computed the length of the array, and sum which was initialized to 0.

So, it says at node 1, which corresponds to those statements these are the definitions. What are the uses at node 1? It says the array numbers is used at node 1. Why is that so? If you go back to the statement, we have this statement right which says length is numbers.length. So, it says you compute the length of this array numbers and set it to the variable length.

So, the numbers as an input, as a variable, which is an array data type is also used at node 1. Node 2 is very simple it initializes the index corresponding to the for loop. So, the only variable that is defined at node 2 is i, nothing is used at node 2. We go back to the CFG see node 3, you remember node 3 corresponds to the beginning of the for loop and in the module where we discussed how to draw control flow graphs corresponding to loops, I told you that CFG is corresponding to loops will have a dummy nodes. Dummy node in the sense the node that is meant for the loop to come back to; 3 such a node because there is a dummy node corresponding to the for loop there are no definitions and no uses at node 3. So, this whole row is left blank.

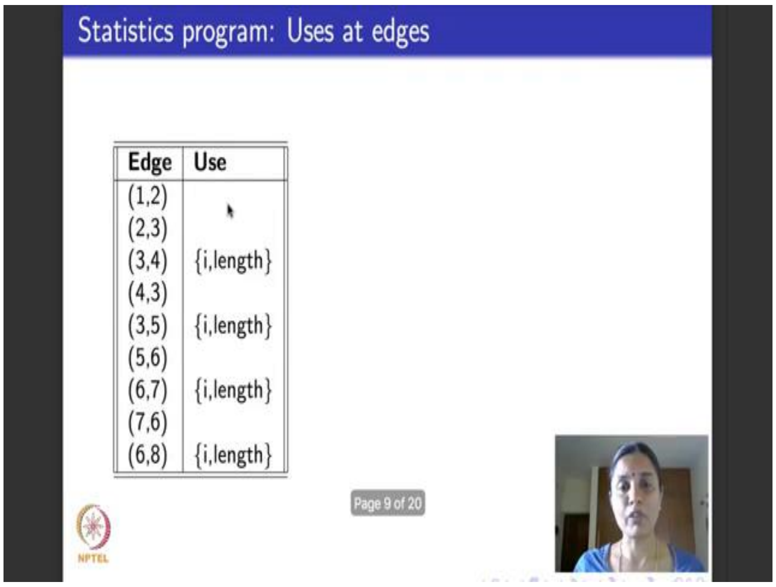
I move on. At node 4 what happens the for loop gets executed. Let us go back to the code and see which is the statement that gets executed in the for loop, that is this. So, it a just take the variable sum, add the value of the next number that you find in the array numbers to the sum, and keep repeating it for the entire length of the array. So, the definitions at sum at node 4 are sum, because it is set back to sum and i and what are the uses? Uses are the array numbers, i and sum again. Is that clear? At node 5 what happens if you go back to the CFG node 5 we are out to the for loop, which means we finished going through this array compute that the sum of all the numbers that we found in the array, we are out of this for loop we are doing these three statements we are computing the median, we are computing the mean, and we are computing, we are initializing varsum.

So, the defs at node 5 are median, mean, varsum and i. Why is i there? I is there because I still came out at node 5 when i exceeded the number of iterations of the for loops, I still gets defined at node 5, the last increment of I happens at node 5. So, the defs at node 5 are all these, the uses at node 5 are numbers length and sum; because these statements use those variables to be able to compute the mean, median and initialize varsum. So, the defs an uses a node 5 I hope are clear. Node 6 if you go back, again is a dummy node that corresponds to the loop.

So, I that is no defs uses for node 6, I move on like that; node 7 corresponds to the execution of the second for loop. So, the defs at node 7 are these, the uses at node 7 are these. If you remember node 7 was inside this for loop where this is computed, varsum is computed. What do I use here? I use the variable mean, I use varsum which I initialize to 0 here, and I use the array number. What do I use them for? I use them to define varsum once again. So, that is what the table indicates. I use varsum numbers I and mean, and I define varsum and I also increment I in node for a for loops that also comes as definitions. At node 8 I am out of this for loop and I do all these statements.

So, the Defs and uses at node 8 are like this. I define variance and standard deviation and I print out all these values. So, because I print out all these values at node 8, they are all come as uses for node 8. So, is it clear please, how this table has arrived at? I take the CFG list all the vertices of the CFG; per vertex, I go back and see what are the definitions, what are the uses.

(Refer Slide Time: 14:11)



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}

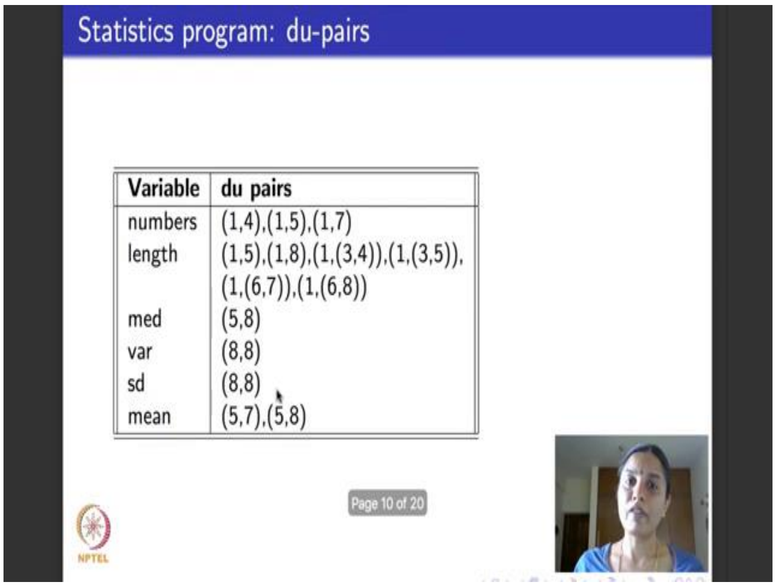
So, what we do now is repeat this exercise for the edges of the CFG. Remember when we talked about edges of a control flow graph there are no definitions associated with edges of a control flow graph corresponding to code.

So, typically it will only be uses and again where do uses come? Uses come only in those edges that involves checking of conditions. So, if you go back and then see the CFG the uses, which are the edges that correspond to the users will be very clear. The edge (3, 5) means that I am exiting the for loop. So, this condition holds the invariant of the for loop is no longer true, the edge (3, 4) means I am executing the next iteration of the for loop. So, this condition holds. So, these correspond to uses at these 2 edges. Similarly we uses this at these 2 edges (6, 8) and (6, 7) are these, that is what is written here and all other edges there are no uses, so they are left blank.

So, the uses at edge (3, 4) are i and length, the uses at edge (3, 5) are i and length, the uses at edge (6, 7) and (6, 8) which correspond to the execution of the second for loop are again i and length. Please remember that in this code we have used the same variable i to represent the first for loop and second for loop and when I document defs and uses, I again reuse the same variable like; it really does not matter, but if you find it very confusing you could call the index of the second for loop using another variable, you could call it as a variable j and then track defs and uses for j. It will still be the same right there is no harm in using the same variable again, right.

So, what have we done till now we have taken the control flow graph, taken each node in the control flow graph written down what which variables are defined at a node which variables are used at node. And then, we took each edge in the control flow graph, and talked about are there any variables that are used at these edges and if there are what are they, that is this table. Now the next step after we have documented the definitions and uses is to compute definition use pairs right. So, now, if you remember du-pairs are always parameterized by a variable. So, per variable, I document to the various def use pairs or du-pairs.

(Refer Slide Time: 16:21)



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)

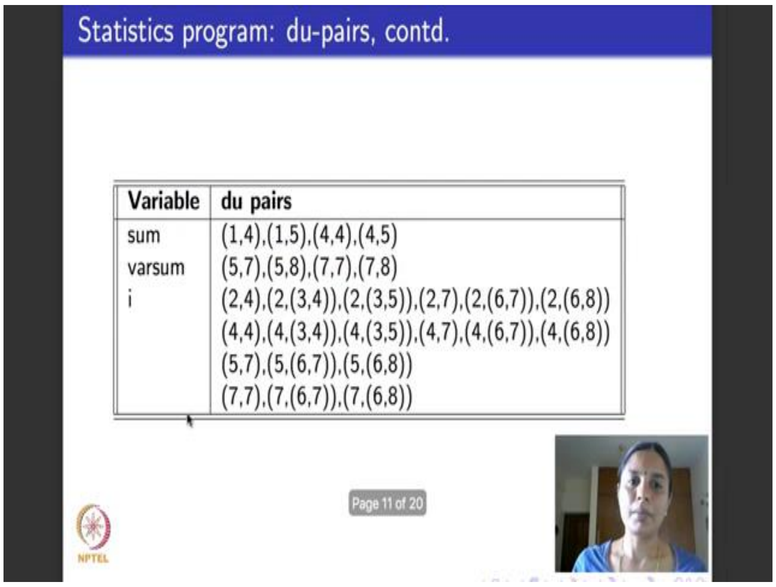
Page 10 of 20

So, for the variable numbers let me go back, numbers is defined at node 1 and where it is used ? It is used at node 4, it is used at node 5, and it is used at node 7, is it clear. It is also used at node 1 right so, I write that in this table. For the variable numbers it is defined at node 1 used at node 4, defined a node 1 used at node 5, defined a node 1 used at node 7. So, you might wonder it is also defined and used at node 1, why have I not listed it in this table? I have not listed it in this table because that will violate the condition that from node 1 to node 1 the path is def clear right because it is defined and used to the same node.

So, when that happens we typically do not list it as a du-path; because it violates the condition of the path being simple and it violates the condition of the path being def clear I list everything else and I do is exercise for each of the variables. So, I repeat this exercise for length, for median, variance, standard deviation mean, sum, varsum and I right this data spread across two slides. So, what is it for length? Let us go back, if you see length is defined at node 1, you see here the length is defined at node 1 whereas, length used again it is used a node 5, it is used a node 8, and when it comes to uses we have to look at edges also it is used at node edge (3, 4), it is used at edge (3, 5), (6, 7) and (6, 8) that is what is listed here.

Its defined at node 1, used at 5, defined at 1 used at 8 and defined at 1 and used at all these various edges. So, please read this as defined at 1 used at the edge (3, 4), right I go on repeating this exercise the variable, median this defined at node 5 used at node 8 that is what is given here similarly for var, standard deviation, mean, sum, varsum.

(Refer Slide Time: 18:48)



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))

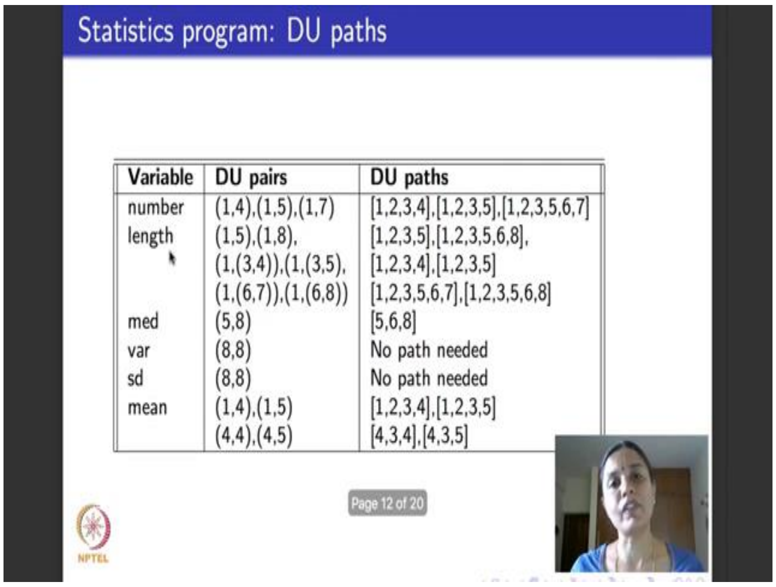
Page 11 of 20

And for i, if you see the list of the d u pairs is quite vast, because I is the index of the variable for loop and it comes in two for loops in the code right.

And if you see here in these tables also in the use of edges this i all over, in the uses, in the defs of nodes there is i here there is i here correct. So, the du-pairs for the variable i is quite a bit. This one that begins at 2 talks about i being used in the first for loop, (2, 4) to an edge (3, 4), to an edge (3, 5), to a 7, to an edge (6, 7) this one talks about it is i being used in second for loop right. So, the du-pairs for i is a large set right.

So, is it clear what we have done till now, we took the CFG augmented the CFG, with Defs and uses augment at each node of the CFG with definition and use augment at each edge of the CFG with uses; then which just repopulated the table we said we will take one variable at a time and used the pairs where the variable is defined and used. The definition of a variable typically comes only from the node; the use of variable could come from a node or from an edge. So, this is the set of all the du-pairs corresponding to each of the variables in the code these two tables depict that.

(Refer Slide Time: 20:22)



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]

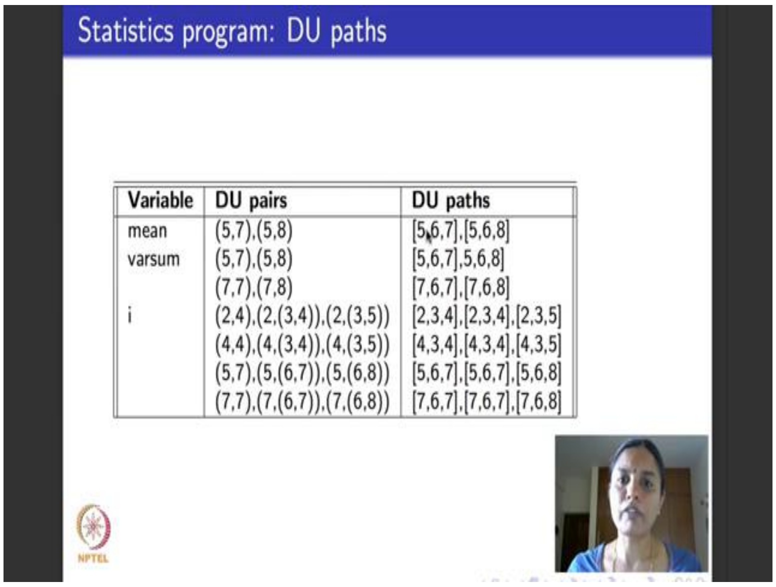
Page 12 of 20

Now, after we have had d u pairs we are ready to define definition use paths. So, we take the same data per variable which are the du-pairs, and then say; which are the paths that correspond to this du-pairs. So, take the variable number it is defined at 1 used at 4, and the paths that takes it from the definition at 1 and to the use at 4 is this path: 1 to 2 to 3 to 4. Similarly the variable number is defined at 1 used at 5, and the path that takes it from the definition at 1 to the use at 5 is this path 1, 2, 3, 5. Take the variable number---defined at 1 used at 7 and this is the path 1, 2, 3, 5, 6, 7 that takes the definition of the variable number from it is definition at 1 to it use at 7.

Similarly, let us say for from median it is defined at 5, and used at 8. So, the paths that takes it from it is definition at 5 to it is use at 8 is the path 5, 6, 8; for var and sd is defined and you study it I as I told you I need not have listed this at all even as I do you pair. So, I do not list du-paths because they are not def free and they are not simple paths I ignore them and get going. So, I go on listing for all the variables in the program this are the pairs this is the paths right.

Now, if you see this, concentrate on this column corresponding to du-paths, you will realize that several paths repeat right. So, if you take this 1, 2, 3, 5; 1, 2, 3, 5 comes here again 1, 2, 3, 5 comes here again, here again similarly the path 1, 2, 3, 4 here once twice thrice so, on right.

(Refer Slide Time: 22:15)

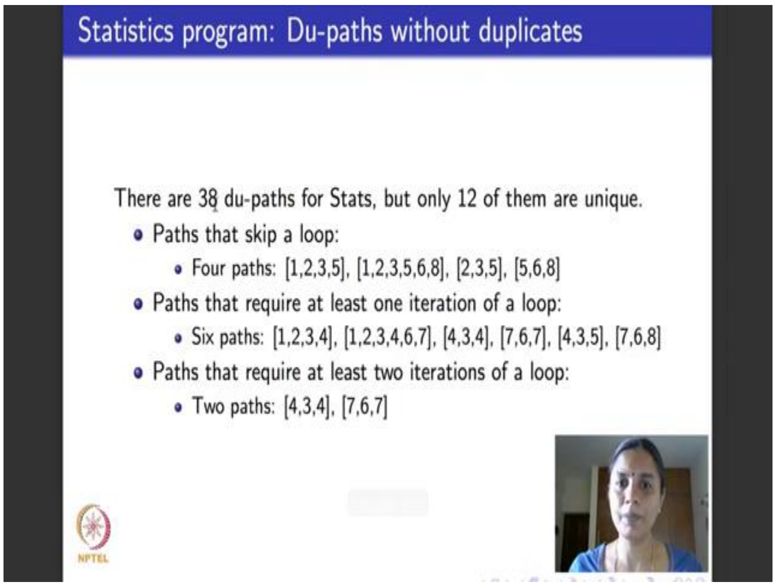


Statistics program: DU paths

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]

And if you go for other variables also if you see the path 5, 6, 7 comes here, here, here and here similarly 5, 6, 8 gets repeated, 7, 6, 7 is repeated, there are several repetitions. So, if you count the du-paths across the last column of these two slides, you will realize that there are totally 38 of them, but lot of them repeat. So, there are only 12 that are unique, right.

(Refer Slide Time: 22:33)



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]

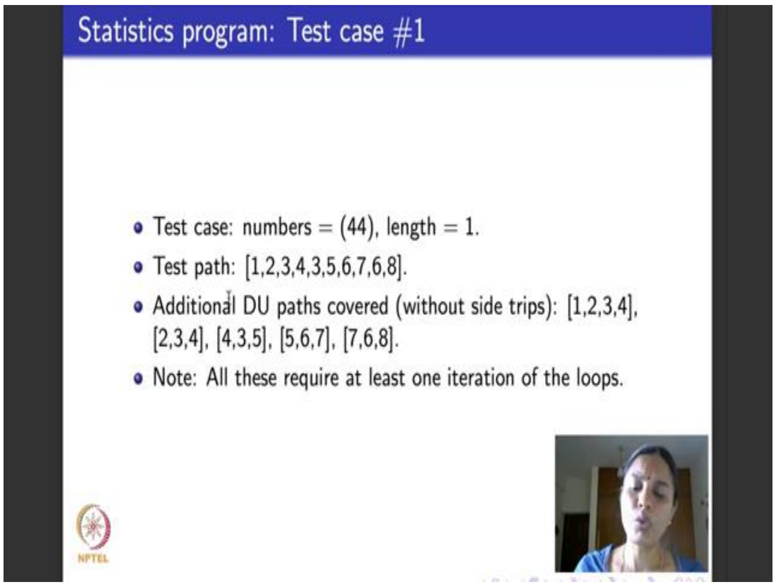
So, these are the 12 unique du-paths, I have grouped them into three sets here. So, the first set which consists of 4 paths a paths that skip one of the loops in the program. Let us look at these paths then go back to the CFG and see if they really skip on loop. So, what are the paths? 1, 2, 3, 5, remember from 3 it goes to 5. So, similarly here from 1 to 3 it again goes to 5 and then does something from 3 it again goes to 5, and from 5 it goes to 6 and 8. So, we will go back now to the CFG and trace out these paths. So, 1, 2, 3, 5 it does skip the for loop, then 1, 2, 3, 5, 6 enters, but it took then take 7 went to 8. So, it skipped this second for loop. These four paths correspond to paths that skip the loop right the remaining paths 1, 2, 3, 4; 1, 2, 3, 4, 6, 7; 4, 3, 4; 7, 6, 7; 4, 3, 5 and 7, 6, 8 what do they do. They require at least one iteration of the loop.

So, we will go back to the CFG 1, 2, 3, 4, I have entered the loop I do 3, 4, I have iterated because once I enter 4 the only way to come out is to execute the loop once, similarly once I enter 6 the only way to come up to execute the loop once, and I do not go back to 4, I go to 3 instead. So, these path need one iteration of the loop, the remaining paths which talk about 4, 3, 4 and 7, 6, 7 talk about walking in the loop. I am again going back to the CFG 4, 3, 4. So, when I do 4, 3, 4 I have come back into the loop. So, I am taking the second iteration of the loop.

Similarly, when I do 7, 6, 7, I have come back to the loop. So, I am taking a second iteration of the loop right what did I do? I have started with defs and then uses and then I listed the def use pairs and then for each def use pairs we listed the du-paths. Several of these du-paths were repetitive. So, there were only 12 unique du-paths, it just so happens that they correspond to some of them correspond to paths that skip a loop, some of them correspond to paths that require at least one iteration of the loop, and the rest of them correspond to paths that require 2 iterations of the loop. So, only these du-paths that we need to cover.

This information that I told you actually tell you what those paths mean. They actually mean something like prime path coverage or loop coverage here; because if you see they nicely skip the loop require at least 1 iteration of the loop and require more than 1 iteration of the loop right. Now what we will do is we will go ahead and write test cases that execute these d u paths right.

(Refer Slide Time: 25:48)



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 loops.

So, what do the test case look like? I want to focus on writing test cases for each of these groups. So, I will write test cases for paths that require at least one iteration of a loop, I will write test cases for paths that require 2 iterations is the loop and then finally, write test cases for paths that skip a loop.

There is a reason why I want to put this at the last it will become very clear soon. So the first test case that I write this for paths that require at least one iteration of the loop, it so turns out that a very simple test case is enough to do this. The simple test case is the array is just this, it as it has just one number. So, it is length is 1 and the single number that it has let us say it is 44 some positive number. So, is it clear? The array is just an array of length 1 containing a single number 44, and what is the test path that it takes? It happens to take this path 1, 2, 3, 4, 3, 5, 6, 7, 6, 8. Please remember that this is a test path a test paths by definition has to always begin in an initial node and end in a final node.

So, we take remember this if I do 1, 2, 3, 4, I am entering the loop right if you remember the CFG, I come back to 3 finish one iteration of the loop I go to node 5 which comes out of the loop then I do 6, 7, 6, 8. So, let us go back and gives this code an input of 44. So, the numbers array is just an array of length 1 which contains the value 44; length will be computed as 1, will enter the for loop some will be computed as 44. Median mean varsum will be initialized, varsum will be computed.

So, in the CFG it will go through 1, 2, 3 do this for loop once we are just sum is computed once, it is added 44 is added to 0 come out compute the various things, and then what it will do, it will do 1 iteration of this for loop varsum is computed and come out. So, it helps to have an array of length 1 right, because I just want to add that 1 number to sum, execute the loop once and come out right that explains why I gave an array of length 1 as a test case to this.

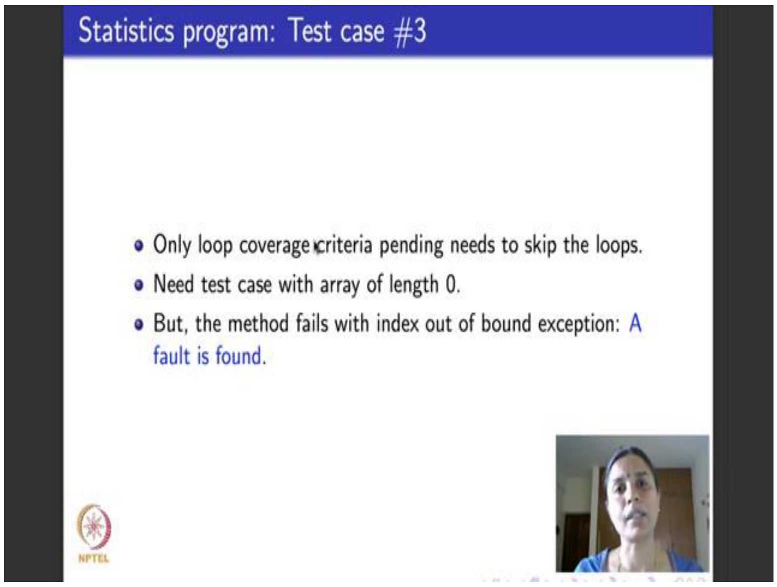
So, this is a test path that it traces and which are the du-paths that it covers? It covers in my list of 12, it covers these paths that needed one iteration of the loop. It covers them correctly and. in fact, the nice thing is that it covers them without any side trips. It covers them directly, you do not need any side trips to towards this right. So, now, I want to write a test case that will cover paths that will need two iterations of the loop, which means what? What do the loops correspond to the first loop corresponds to iterating over the length of the array and adding the numbers to sum. So, I need an array of length more than one any array of length more than one would do.

So, here is a sample array that I give as my test case, which has the numbers 2, 10 and 15 and it is length happens to be 3 right, because it has 3 numbers what it will do when it executes the code is, it will take this for loop 3 times and this for loop 3 times, because it has to go and add the 3 numbers that it finds across the array to the variable sum in this for loop, and in this for loop it has to compute varsum right.

So, what it will do is that you take the paths it will add the first number, it will add the second number, it will add the third number go through this for loop more than twice come again here, go through this for loop also more than twice because there are 3 numbers in my array right. So, the test paths that this one takes will look like this sorry will look like this. So, it does 1, 2, 3, 4, 3, 4, 3, 4 where the 3 numbers are added here in the 3 iterations of the loop first for loop, comes out of the first for loop then computes varsum by doing 3 iterations of the second for loop 6, 7; 6, 7; 6, 7 comes out of the second for loop at 8 it prints, right.

So, in this it covers the paths 4, 3, 4 and 7, 3, 7 which required at least 2 iterations of the loop. Now if I go back and take this segmentation of du-paths I have finished writing test cases for paths that require one iteration of the loop, finish writing test cases for paths that require 2 iterations of the loop, what is pending test case that first path that skips will loop. So, if you see what will happen if we write a test case for path that is skip a loop what sort of a test case will that be; that will be an array of length 0 right.

(Refer Slide Time: 30:42)



Statistics program: Test case #3

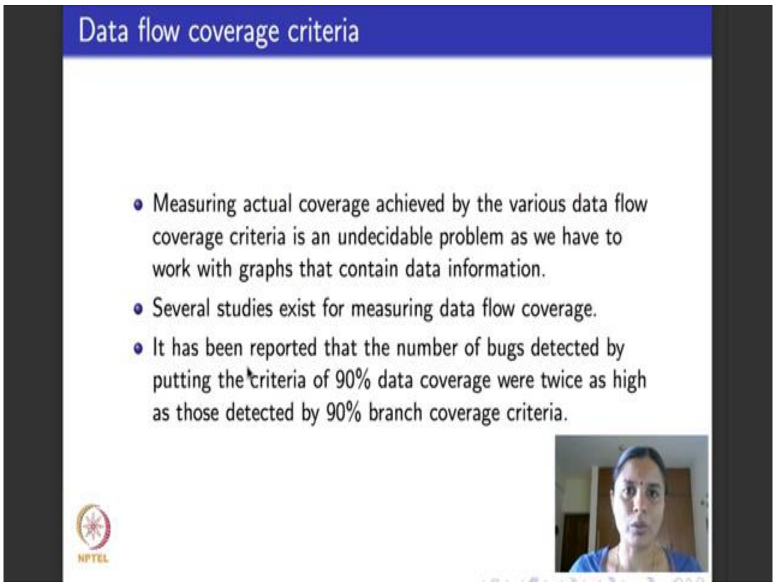
- Only loop coverage criteria pending needs to skip the loops.
- Need test case with array of length 0.
- But, the method fails with index out of bound exception: A fault is found.

What will be a; what will happen if we give an array of length 0? Let us go back and look at the code. There will be a fault in the code now; probably the first fault that we are seeing through the span of this course. You will get to see little more faults which are always interesting to find in testing. So, if you see right here what will happen if I pass the parameter of length 0 right? I do this i pass a parameter of length 0 I come here initialize i to 0, i less than length, it will fail. So, an exception will be produced and the code is not meant to handle arrays of length 0, so the method will fail with index out of bound exception. So, I have found a fault.

So, what was the fault in the code? The fault in the code is very simple. The code did not handle the corner case of what will happen if an array of length 0 is given as input to the code. So, the code was computing all the statistical details correctly, it just did not handle one corner case and that is the fault that we have found by applying data flow coverage criteria to this code. So, I hope you walking through this example in detail would have helped you to learn how to apply data flow coverage criteria step by step, and see if the fault, if a fault if it is there in the code can be found.

It just so happens that for this example there was a fault and the applying data flow coverage criteria helped us to find the fault. We are more or less done with looking at dataflow coverage criteria. I would like to recap some of the points that I told you last week, one is sometimes we are interested in asking how much have we covered in the code when I apply any of the three data flow coverage criteria.

(Refer Slide Time: 32:22)

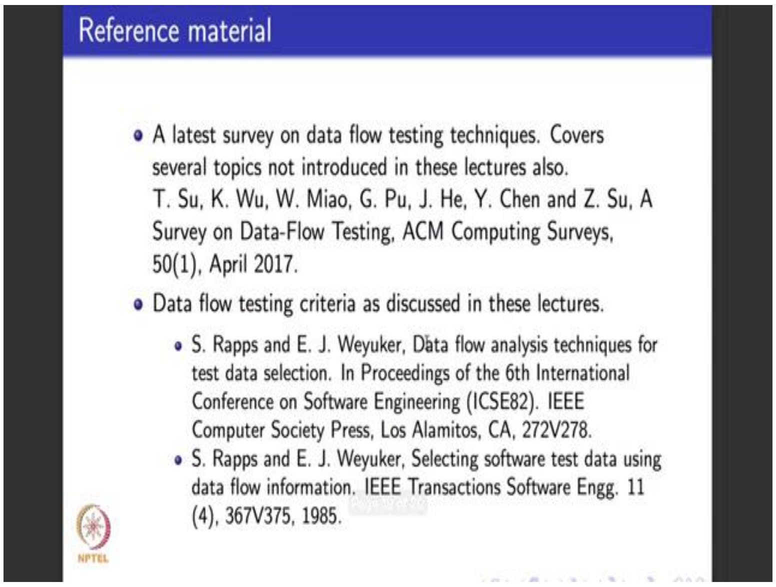


Data flow coverage criteria

- Measuring actual coverage achieved by the various data flow coverage criteria is an undecidable problem as we have to work with graphs that contain data information.
- Several studies exist for measuring data flow coverage.
- It has been reported that the number of bugs detected by putting the criteria of 90% data coverage were twice as high as those detected by 90% branch coverage criteria.

It so turns out that that is a tough problem to solve usually undecidable, and there are several studies that exist for measuring data flow coverage criteria. We will not be able to look at such measurements in detail, but empirically it is known that suppose I take one of the data flow coverage criteria and I insist that 90 percent of the data flow coverage criteria needs to be done right. It so happens that the coverage that it achieves is twice as high as those detected by insisting that I do 90 percent of branch coverage criteria. So, a data flow coverage criterion is indeed more effective and these are purely empirical studies.

(Refer Slide Time: 33:19)



Reference material

- A latest survey on data flow testing techniques. Covers several topics not introduced in these lectures also. T. Su, K. Wu, W. Miao, G. Pu, J. He, Y. Chen and Z. Su, A Survey on Data-Flow Testing, ACM Computing Surveys, 50(1), April 2017.
- Data flow testing criteria as discussed in these lectures.
  - S. Rapps and E. J. Weyuker, Data flow analysis techniques for test data selection. In Proceedings of the 6th International Conference on Software Engineering (ICSE82). IEEE Computer Society Press, Los Alamitos, CA, 272V278.
  - S. Rapps and E. J. Weyuker, Selecting software test data using data flow information. IEEE Transactions Software Engg. 11 (4), 367V375, 1985.

You can look at these papers which I talked about at the end of the previous lectures also, to look at more details about data flow coverage criteria. These two are very seminal, old papers that introduced most of the data flow coverage criteria that we looked at throughout this course, and there is a latest survey on entire data flow testing techniques that is available.

So, what we will do next time is we move on to design. How to model design as graphs and how to look at data flow coverage criteria right. Just to end I took the statistical example from this text book that we have used throughout this course. So, when I see you for the next lecture we will move on from code to design and see how to apply graph coverage criteria for design.

Thank you.