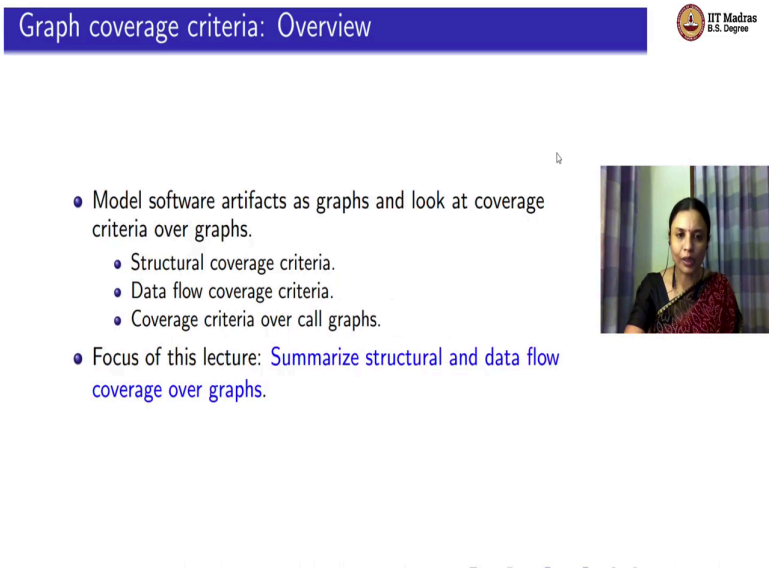
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Technology, Bangalore
Unit Testing Based on Graphs - Summary

Welcome back. This is the last lecture of week 3. Predominant part of this week was devoted to understanding unit testing based on graphs. And I thought before we move on into the next topic, it is useful to devote one lecture trying to summarize and understand for ourselves, how unit testing can be done by using graphs? For be a shorter lecture, we can reuse a lot of the information that we are going, we have learned in the past week, and revise them before we move on to the next topic.

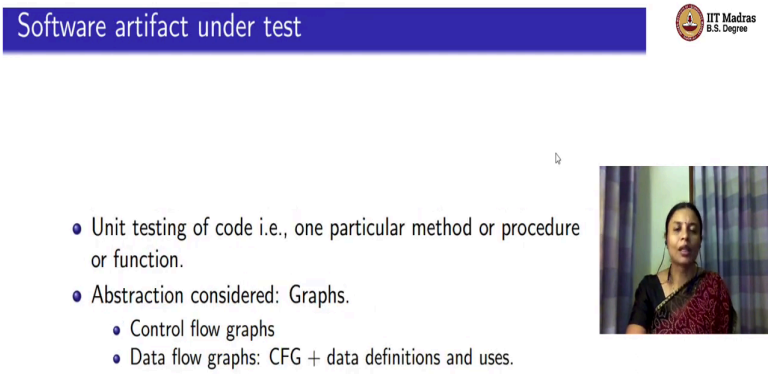
(Refer Slide Time: 00:55)



So, our goal that we followed was take code, which is the software artifact that we considered for last weekend this week. We model the code as a graph, and look at coverage criteria over graphs. We looked at structural coverage criteria, purely based on control flow graph. Then we added data definitions and uses and looked at data flow coverage criteria.

In the next week, I will tell you about call graph. But the goal today is to summarize whatever we have learned by repeating it in a slightly different style.

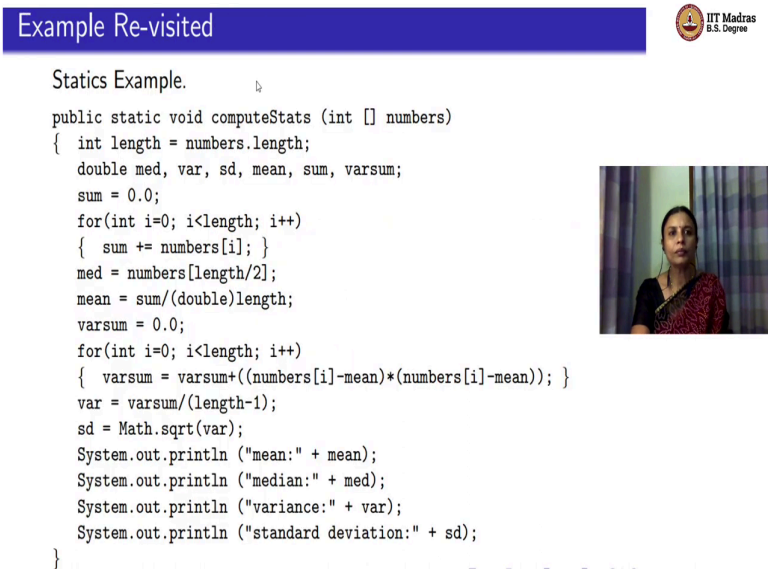
(Refer Slide Time: 01:34)



So, our software artifact under test is code. And the level of testing that we are in is unit testing of code. That is we have one particular function or a procedure or a method. It could be large or small, but we have to unit test that particular entity. And one way we learned to do that was to abstract out paths of the method and model paths of it as graphs and then use the structure of graphs to be able to unit test it, use the structure of graphs, plus the data available in the method to be able to unit test it.

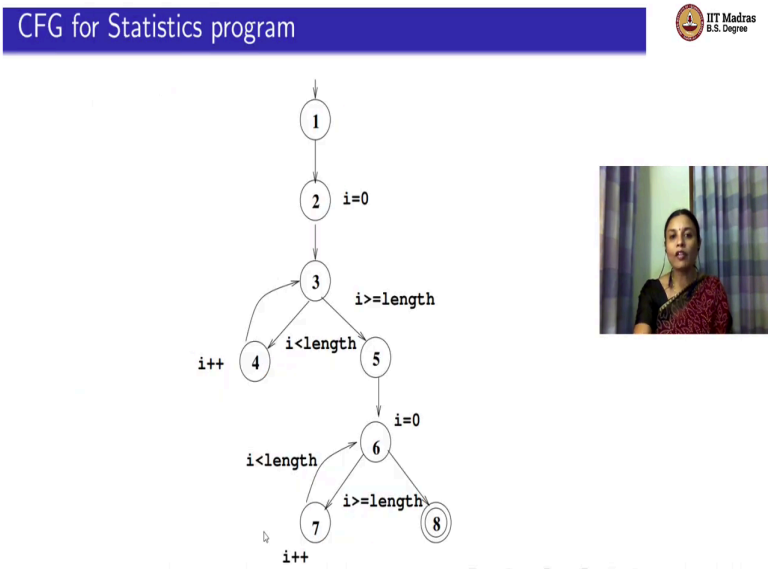
The first abstraction of the model that we consider is what we call control flow graph abbreviated as CFG. And then we learned criteria over them. And then we took CFG, took nodes and edges in the CFG. And augmented it with data added information about data getting defined, data getting used. And that is what we call data flow graphs. And we learn testing based on data flow graphs. So, I will just recap these two. That is the purpose of this lecture. It is essentially a repeat and a summary of what we learned till now.

(Refer Slide Time: 02:59)



Remember, in the process of looking at these examples, sorry, this the typo here, which is that is not static. We looked at code, and I am trying to show you the same code once again. This is our famous statistics example. We had an array of numbers and we want to compute various statistical parameters for it. We have revisited this in the context of control flow graph, we looked at it in the context of data flow graph and found an error in the last lecture. But the idea is that this is the whole thing is a method computeStats is a method that takes input as an array of numbers, called numbers, and computes various statistical parameters and outputs. I want to unit test this method.

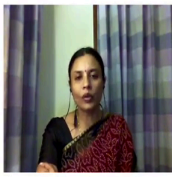
(Refer Slide Time: 03:48)



Example Re-visited

Statics Example.

```
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);
}
```



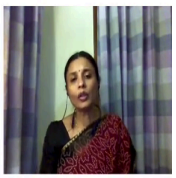
So, we derive this control flow graph to unit test the method. You can go back and forth between this method and this control flow graph. And now in the course, when you use an IDE, let us say Visual Studio or IntelliJ or any other integrated development environment. You should be able to as a part of that IDE generate given a method like this, a control flow graph like this. It is either available as a plugin or you should just look for it within Eclipse or Visual Studio or IntelliJ idea. You will be able to get a control flow graph.

A lot of concepts that we learned in the course will also be followed by the IDE and the control flow graph, will have more or less the same structure. Maybe there could be a couple of small differences in terms of the kind of nodes that they put for loops or the kinds of nodes that they collapse into basic blocks. But outer line control flow structure, which says okay, here are the decision points, here are the loops, there is transfer of control that information every IDE should be able to give.

So, once you get this control flow graph, you keep it with you. And then you can use this for testing.

(Refer Slide Time: 05:09)

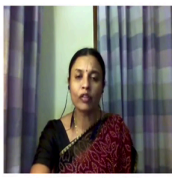
Structural coverage criteria

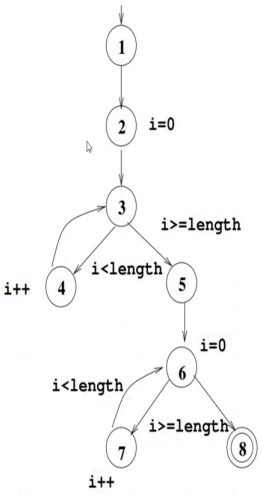


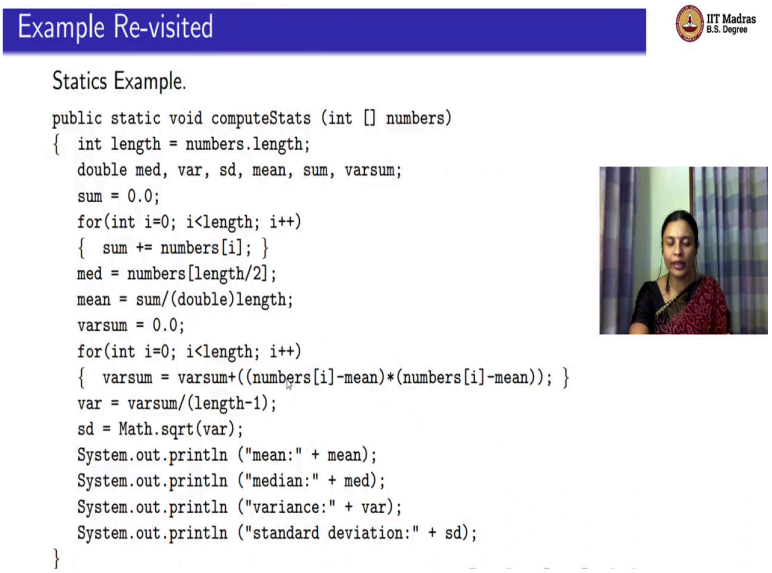
CFG can be thought of as a coarse abstraction of the underlying code. We retain only the control flow information in this graph.

- Node coverage: Corresponds to executing every (basic block) of statement.
- Edge coverage: Corresponds to executing all the decisions/transfer of control.
- Prime path coverage: Corresponds to executing loops (structurally).

CFG for Statistics program







So, this control flow graph can be thought of as an abstraction on the underlying code. It lets go information about the variables. Even though I put it here, just for clarity sake, we do not really keep information about variables. We do not keep information about what are the exact statements that label each of these nodes. So, it is an abstraction of the code. And in fact, it is course abstraction of the code, because we lose some amount of information. But we retain information about the way control flows from one statement to a subsequent statement within a method in a code. So, that is what is control flow information.

Once we have the graph, the structural entities that occur as a part of the graph, or is nodes, or vertices, the edges and some kinds of paths, where I can do node coverage as a structural coverage criteria. In terms of testing the code, we learned that it corresponds to writing a set of test cases that will execute every basic block of statements in the code. It is a simple white box testing exercise that is very useful, basically write a set of test cases that will execute every statement in your method. So, if there is an error, there is a reasonably good chance that by just by executing every statement, you will be able to pull out there.

We also saw edge coverage as a structure and graph coverage criteria. That corresponds to executing every possible transfer of control from one statement to the other. It could be a decision transfer of control. It could be a straightforward transfer of control into the next assignment statement or the print statement. It could be a transfer of control that lets you enter a loop. It could be a transfer of control that lets you exit from a loop basically all possible transfers of control. And if there is a scope to get an error there, edge coverage is a useful coverage criteria to find such error.

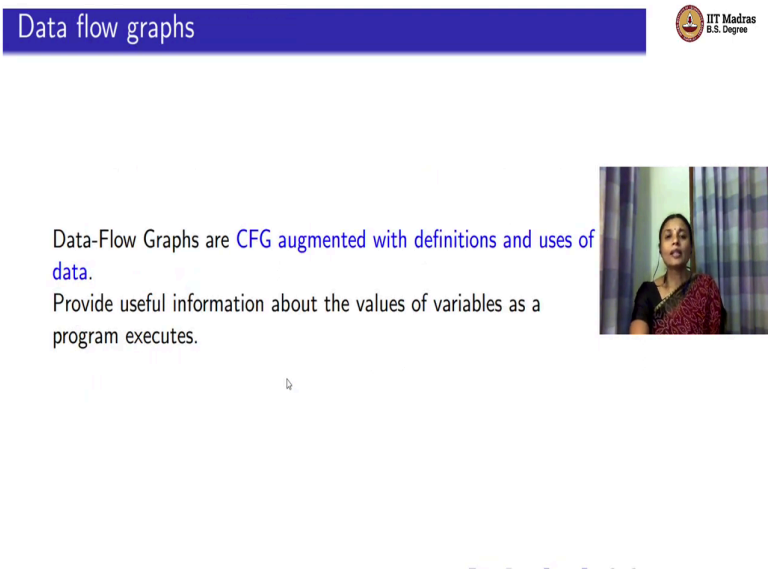
The next is structurally, we also retain information about paths. We learned that if the graph does not have loops, that is if the underlying code does not have loops, then the paths are finite. But if the underlying code has a loop, like as in the case of the statistics example, then the underlying graph corresponding to the code also will have loops.

Structurally, we understood prime paths which are maximal simple path is a good coverage criteria to get all possible ways in which you can execute loops as many times as you want, exactly one, or even skip the loops. So, prime path coverage is a useful structure and coverage criteria that you can use without knowing anything about the loop, loop counter. You can still test the loop by skipping it, by executing it exactly once, and by executing it more than once or as many times as you like.

This very useful information that we could use for testing purely based on the control flow graph. So, just to repeat, you have a method like this, feel free to use your IDE. Look for commands options that let you generate a control flow graph like this. Once you have control flow graph like this, you could apply coverage criteria of your choice structure in coverage criteria and generate test cases. Then I recorded this module. I had shown you the web application from George Mason University where you could feed in your control flow graph and get test cases.

For the purposes of learning in this course, we would do it by hand by drawing the control flow graph, or picking it up from the IDE and then you learning by using the algorithms that we learned we will come up with test requirements and test paths for these kinds of coverage criteria. And they are useful for white box testing.

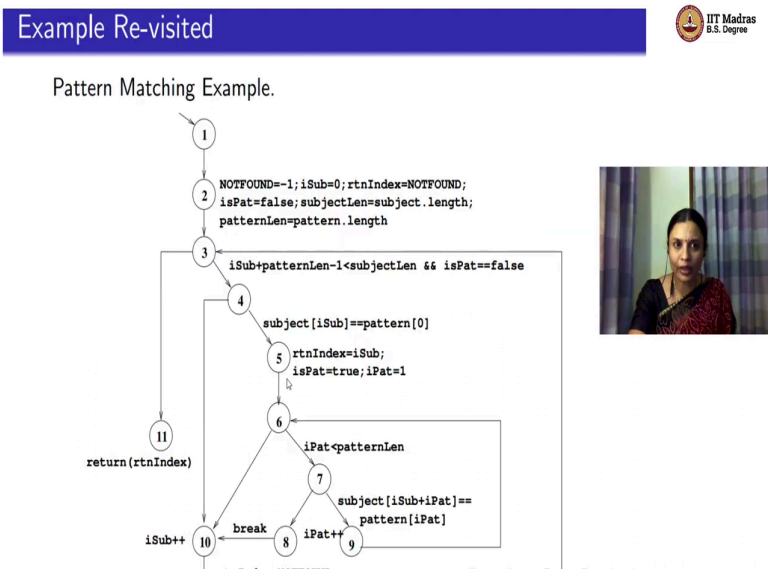
(Refer Slide Time: 09:18)



The next list control flow graphs, let us go all information about the variable maybe we would want to keep them because some of it is useful. So, we consider data flow graph. What are data flow graphs in testing? Their control flow graphs, where some of the nodes and some of the edges have additional information. What is the additional information? The additional information talks about the variable that are under use in the method or the function, which are the nodes where the variables get defined that is the value gets stored or restored into memory and which are the nodes or edges where the variable gets used that is the value gets retrieved or revived from memory.

So, we take CFG. Add information about definitions and uses of variables to the nodes and edges, we get data flow graphs. Data Flow graph provides very useful information about transfer of control, because it has the underlying CFG. It also provides useful information about how the variables that are defined, get used in different different places in the program, as how can we track how a value of the variable progresses from its definition to one or more of its possible uses.

(Refer Slide Time: 10:41)

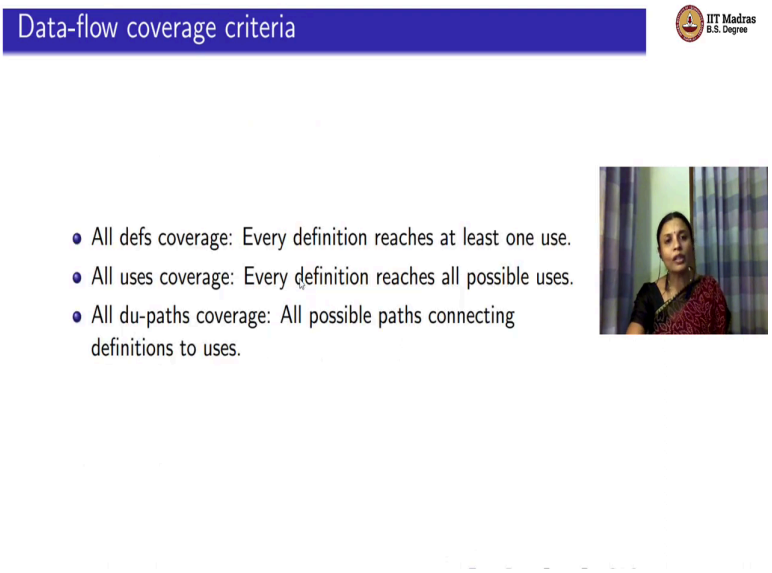


In this context, we saw the example of a pattern matching, taken subject and given a pattern, check if this pattern occurs in the subject or not. I have not given you the code here. But this is the control flow graph that we derived for the same pattern matching example. This has so many variables subject pattern as the character array, index of the subject, length of the subject and length of the pattern, then return index, a Boolean flag saying whether the pattern is found or not and so on and so forth. The control flow graph looks like this.

In this control flow graph, by using these information that I find, I can augment the nodes and edges with the definitions and uses of the respective variables. This also we saw in detail, so I am not repeating the whole thing. So, for example, in the node 2, the following variables are defined not found, isub, return index, isPat, subject length, pattern length. In the node 2, the following variables are also used.

Similarly, in the edge 3, 4 and in the edge 3, 11, which corresponds to the same decision statement while statement. These are the variables that are used. Similarly, for other edges, and so on and so forth. So, I take the control flow graph, augment it with definitions and users and I get the data flow graph which looks like this.

(Refer Slide Time: 12:12)

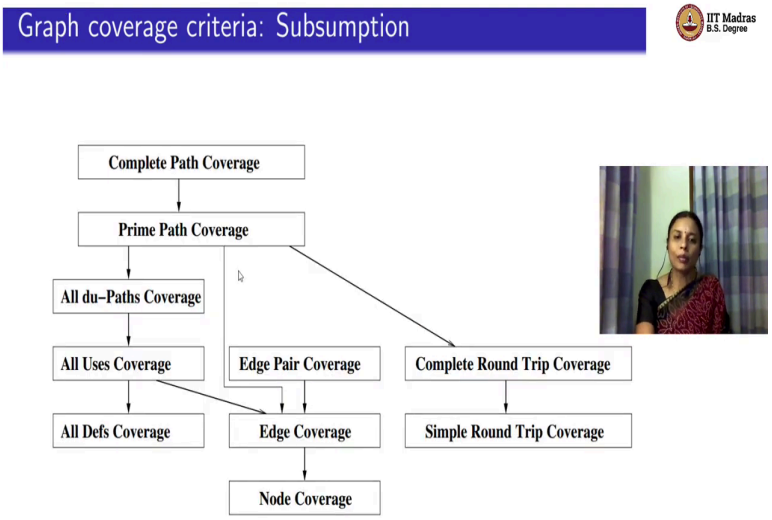


On this data flow graph, we learn three kinds of coverage criteria. The first one is make sure every variable that is defined has at least one use, come up with a test path that takes the definition of a variable to its use. That is all definitions coverage. Second one is all uses coverage. Make sure the definition of a variable reaches all possible uses, right different test cases that will track the definition of a variable to each of its different uses.

The third one is a most sophisticated test requirement says for every variable, make sure you have test cases that will take every definition of a variable to every possible use in all possible execution paths of the code. So, that is like an exhaustive data flow testing. I test how the value of a variable flows from its definition to its uses through all possible execution paths and the code.

We applied it to the statistics example. And we found a very useful bug when the loop was being skipped.

(Refer Slide Time: 13:29)



This summary slide gives you all the data flow and control flow coverage criteria that we have seen till now. On this middle line here, we have no coverage, edge coverage, edge pair coverage. If you go up here, prime paths for loops and then complete path coverage. We also saw special kinds of prime paths, which were round trips. Prime paths that begin and end at the same node. Simple round trip coverage and complete round trip coverage. And these three on the left side are the three data flow criteria.

All definitions, all uses and all du-paths. This how one subsumption relations for these three data flow definitions. Another subsumption relations for the structural coverage criteria. And a picture like this puts together both of them. Like we observed that prime path coverage not only subsumes all du-path coverage. It also subsumes all uses and all defs coverage. So, this is one new extra subsumption.

We also saw that all uses coverage will subsume all the decisions and so it does subsume edge coverage. So, this is the summary slide with you I want you all to remember. So, if you have to pick and choose between a particular coverage criteria, the useful kind of coverage criteria that you could pick from this slide are all du-paths or prime path or all uses or edge coverage or node coverage. The rest of them are there. But they may not be very useful comparatively.

So, I will stop here for now. I just wanted to summarize what we have seen till now in terms of unit testing a particular method or procedure by deriving control flow graphs, augmenting control flow graphs with data information and using coverage criteria on that.

Next week also, we will see graph based testing, but move on to other kinds of software artifacts, like larger fragments of code with call interfaces, specification and so on. So, I will stop here for now. I will see you next week. Thank you.