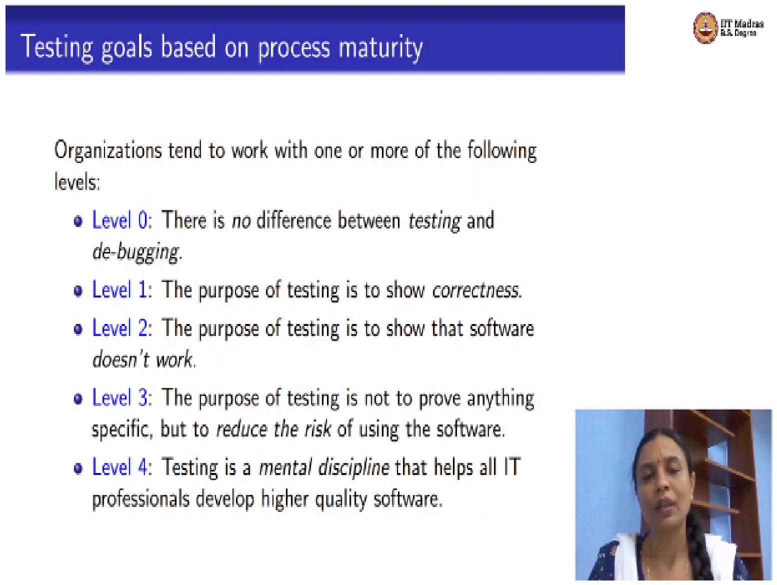
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Technology, Bangalore
Software Testing: Terminologies and Processes

Hello everyone, welcome back, we are in week 1, I will continue with software testing terminologies and processes today and move on to doing test automation in the next lecture.

(Refer Slide Time: 0:36)



Testing goals based on process maturity

Organizations tend to work with one or more of the following levels:

- Level 0: There is *no* difference between *testing* and *de-bugging*.
- Level 1: The purpose of testing is to show *correctness*.
- Level 2: The purpose of testing is to show that software *doesn't work*.
- Level 3: The purpose of testing is not to prove anything specific, but to *reduce the risk* of using the software.
- Level 4: Testing is a *mental discipline* that helps all IT professionals develop higher quality software.

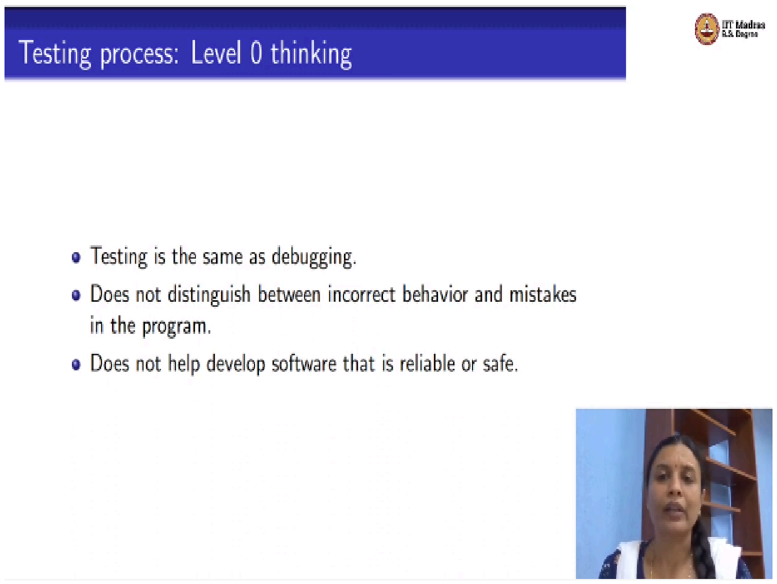
So, we saw all the basic terminologies last week, in the last class let us move on and look at what would be the goals of testing based on what an organization wants to do. Testing is usually heavily dependent on the nature of the software product that is being developed, on the nature of the organization that is developing the software product, how big or small it is, and how mature they are in terms of setting up their quality processes.

So, testing goals can vary to be absolutely trivial to very mature based on how the processes of a particular organization are mature. So, this small module will tell you what would be the various testing process maturity levels as far as software development and testing is concerned.

So, level 0 is absolutely low-level, there is absolutely no difference between testing, and debugging, anybody is free to do what they want to do, followed by testing at level 1 where people think that the purpose of testing is to show that their software product is correct, it goes on till levels 4 as you can see in this slide, at level 2 the purpose of testing converts to showing, producing and showing errors or defects or bugs.

At level 3 the purpose of testing is a little more sophisticated, you know I use testing to show a particular module has bugs, I rectify the bugs to say that the module is now free of those bugs, so there is a bit of testing used to show correctness, this is mature, it is repeated, there are distinguished testing teams and my overall goal is to show that the software works pretty fine, the risks of using it are much less, level 4 where large organizations exist is the most mature level of testing, where the quality and testing processes are so mature, the testing becomes a part of the mainstream work of a developer and all developers including testers, use testing extensively to develop high-quality software.

(Refer Slide Time: 2:52)



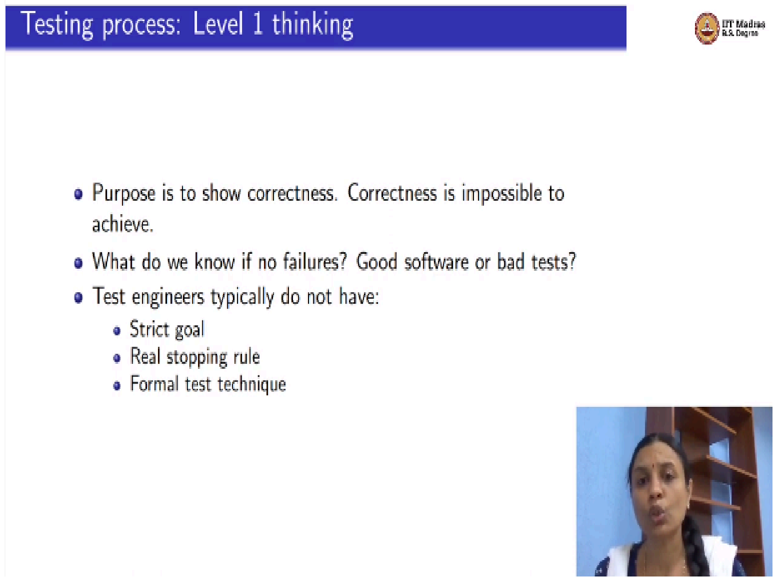
Testing process: Level 0 thinking

- Testing is the same as debugging.
- Does not distinguish between incorrect behavior and mistakes in the program.
- Does not help develop software that is reliable or safe.

So, let us look at each of these levels little by little. So, if I am at level 0 thinking, typical example is somebody who just got a cool startup and he or she is trying to work on a software and their goal is to try and get the product out as early as possible, so they try to debug obviously while they code, they do not maybe go through an exclusive testing phase, they do not have mature, well-defined processes for testing and things like that.

So, it is the same as debugging whatever the developer does to ensure that his or her code works correctly, which is debugging the code becomes the same as testing, no exclusive efforts devoted for testing and we move on, so while this helps it does not give proper confidence, does not ensure that the software is fully reliable or safe.

(Refer Slide Time: 3:53)

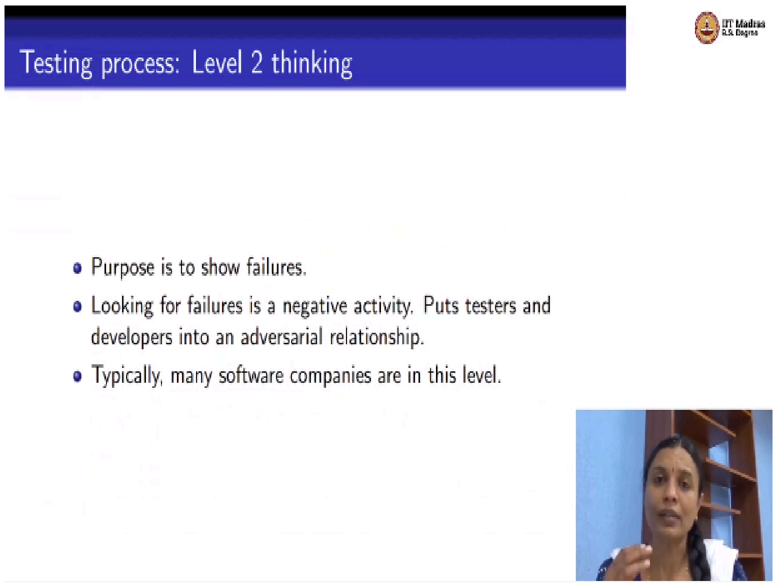


Testing process: Level 1 thinking

- Purpose is to show correctness. Correctness is impossible to achieve.
- What do we know if no failures? Good software or bad tests?
- Test engineers typically do not have:
  - Strict goal
  - Real stopping rule
  - Formal test technique

The next comes level 1 thinking, where the purpose is slightly misunderstood, developer and tester think that the purpose of testing is to show correctness but on the other side as I told you earlier, using testing is almost impossible to make a statement which says that my software is fully correct, so correctness is impossible to achieve using testing because the only thing that we can do with testing is to identify faults and errors and the purpose should be to identify faults and eliminate them and retest. So, if I had set out by using testing to show correctness then I am not following the right path.

(Refer Slide Time: 4:33)

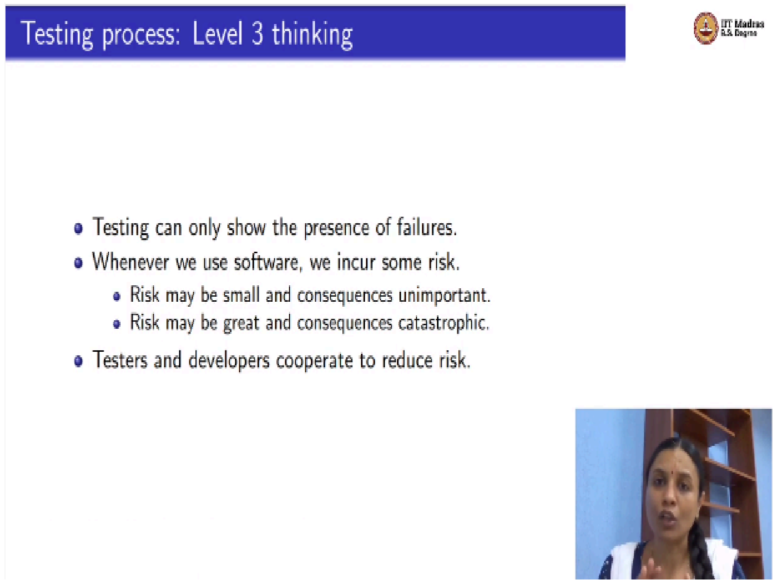


Testing process: Level 2 thinking

- Purpose is to show failures.
- Looking for failures is a negative activity. Puts testers and developers into an adversarial relationship.
- Typically, many software companies are in this level.

The next one is level 2 thinking, where testers and developers are consciously aware of the fact that the goal of testing is to find errors by showing failures in the software and but it is sometimes mistakenly viewed as you know I am a developer, he or she is a tester, they found an error in my code, that is bad you know, their processes are not mature enough to say that finding errors, while testing is good for your software because you can rectify them and ensure that it is correct, so while they consciously realize that the purpose of testing is to find errors, it is not yet mature to accept testing as a robust rigorous process to find and eliminate errors.

(Refer Slide Time: 5:23)

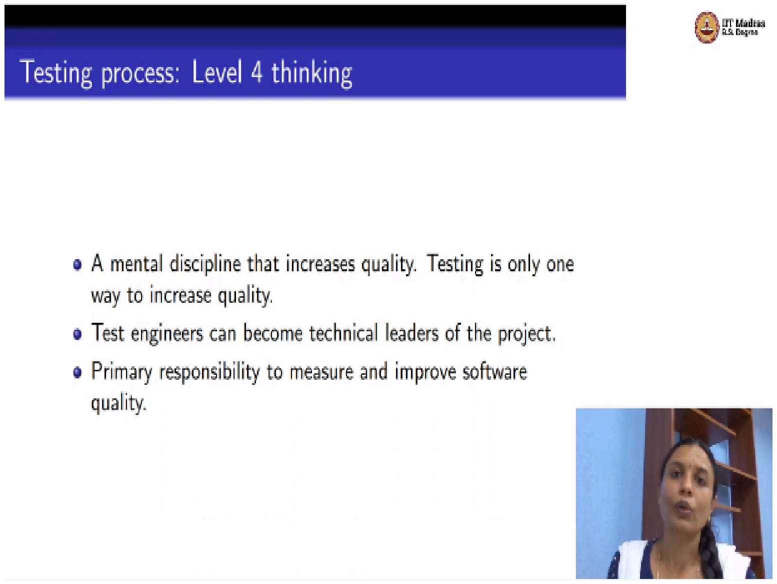


Testing process: Level 3 thinking

- Testing can only show the presence of failures.
- Whenever we use software, we incur some risk.
  - Risk may be small and consequences unimportant.
  - Risk may be great and consequences catastrophic.
- Testers and developers cooperate to reduce risk.

The next is level 3 thinking, where testing is consciously considered by everybody, both developers and testers as a robust technique that is known to show errors and bugs, known to detect faults, so everybody is aware of the fact that when I write code, when I use the code, when I run the code, I am taking some risks, the code is bound to have errors, the earlier I find it as and when I am writing as and when I am testing the better it is, if I release the code before finding a particular bug and if it is found after release then it is risky, so testers and developers work in cooperation to find as many errors as possible and eliminate them, fix them and reduce the risk of the software being defective after release, many organizations typically are at this stage.

(Refer Slide Time: 6:25)

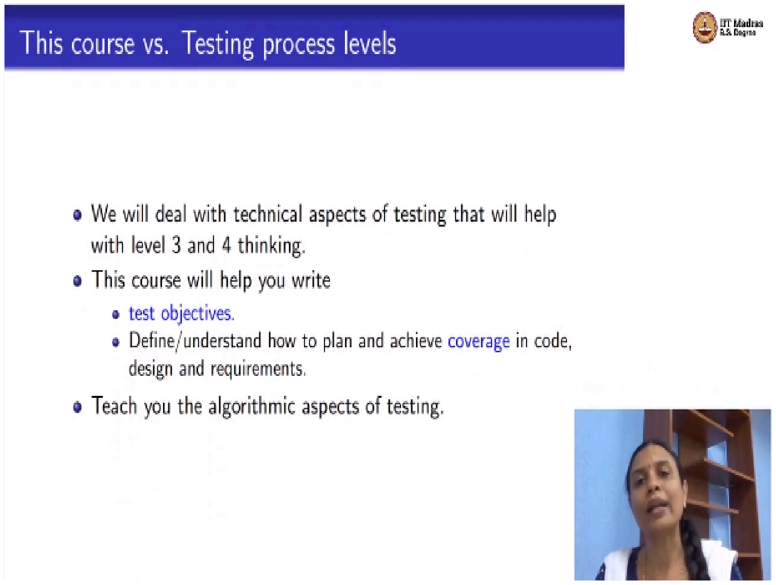


Testing process: Level 4 thinking

- A mental discipline that increases quality. Testing is only one way to increase quality.
- Test engineers can become technical leaders of the project.
- Primary responsibility to measure and improve software quality.

Level 4 is the most mature process-oriented concept for testing. So, if you are in an organization that practices level 4 thinking as far as quality and testing processes are concerned, what is testing? Testing becomes a mental discipline, it is a part of your job that you do to routinely test your code subjected to quality audits and it is the only way that is known to increase the quality of your software module and the ultimate software products, test engineers are developers they become active components and leaders in the project and their primary responsibility is continuous improvement of quality which involves, in turn, continuous measurement of quality, measurement of quality involves finding and eliminating defects, keeping track of the defects, ensuring that they are reduced over time as and when a software is mature, so that is level four thinking.

(Refer Slide Time: 7:29)



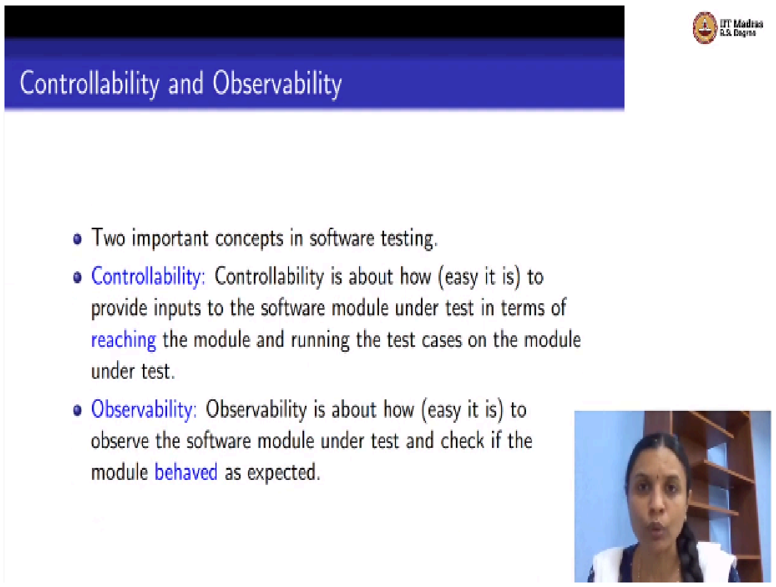
This course vs. Testing process levels

- We will deal with technical aspects of testing that will help with level 3 and 4 thinking.
- This course will help you write
  - test objectives.
  - Define/understand how to plan and achieve coverage in code, design and requirements.
- Teach you the algorithmic aspects of testing.

Now, as far as this course is concerned, how does it matter that was this theoretical thing that we spent a few minutes understanding about testing process levels, but how does it matter as far as this course is concerned, the first thing is it is important for all of you to know that there are something called testing process levels and based on the process level that the organization that you are in is a part of the goals of your testing, the kind of testing that you do will drastically vary, so we in this course are going to deal with all the technical and algorithmic aspects of testing that will help for level 3 and level 4 thinking, where there is maturity as far as testing is concerned.

So, this course will help you to write test objectives, what is my goal, why am I pursuing testing? what is it that I am going to do? while testing a particular software module, it will also help you to design, understand, and implement what it means to achieve coverage in code, so I have a large code base that I have developed, let us say a few thousand lines of code may be much more than that, how do I systematically go about testing this code, how do I plan for it and what do I say I am testing this code for, what are the properties, what am I trying to achieve or cover with testing, the same holds for design, the same holds for requirement documents, this course will teach you how to test design documents, how to test requirements, how to test preconditions, postconditions and how to test code, both at the white box and black box style, you learn all the algorithms that you can that will be used for testing.

(Refer Slide Time: 9:15)


Controllability and Observability

- Two important concepts in software testing.
- Controllability: Controllability is about how (easy it is) to provide inputs to the software module under test in terms of reaching the module and running the test cases on the module under test.
- Observability: Observability is about how (easy it is) to observe the software module under test and check if the module behaved as expected.

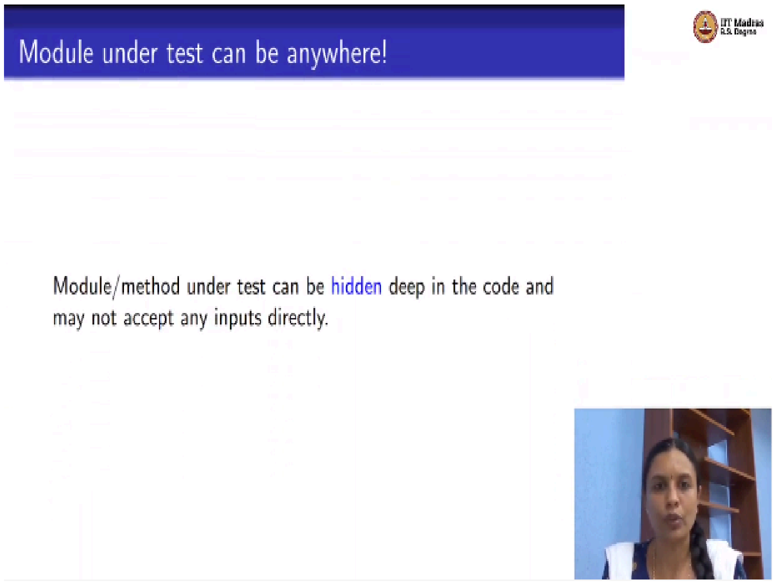
So, that was about process levels, the second module in this video is about one concept that I would like you all to understand before we move on to actually designing test cases, that is the notion of controllability and observability. They are two very important concepts in software testing, so let us see what they mean. So, what is controllability?

So, if you take the English meaning of the word something is controllable, I have control over it, so what is the thing that you need control over as a tester, let us say you are testing a particular software module, a particular code base, it could be one whole class, it could be a fully packaged software, it could just be one method or a procedure inside a software, so controllability is about how as a tester you can provide inputs to the software module that you are testing, run the software module on the test inputs that you have provided and get it up and running.

Now, the next part, I have given inputs to my software module under test, the software module is running on my inputs, what does the software module do, it finishes executing on the inputs and it will produce some outputs, so I should be in a position to observe the outputs, to record the outputs, study them and come to a conclusion about whether there is an error in the software module by studying the outputs that are produced.

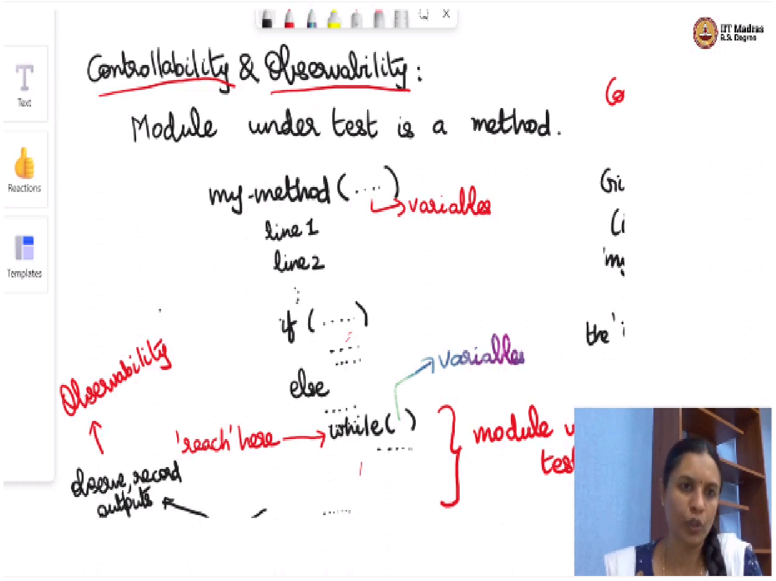
So, you have a software module under test, just to repeat so that could be a full piece of packet software, a class, or a method, I give inputs to that software module, such that the part that I am interested in is exercised, is run, executed or tested, that is controllability and when the software module runs it produces some outputs, so the parts that I do to produce output is called observability. So, these two put together make sure that your testing is successful, that I test my software module, make it controllable, make it run on the test input and then I produce outputs for the software model.

(Refer Slide Time: 11:31)



Module under test can be anywhere!

Module/method under test can be hidden deep in the code and may not accept any inputs directly.



Controllability & Observability:

Module under test is a method.

```
my-method (....)
  line 1
  line 2
  :
  if (....)
    ....
    ....
  else
    ....
    while( )
    .....
```

So, let us understand with an example what is controllability and observability. Let me illustrate through a simple example what I mean by observability and controllability. So, let us switch to the whiteboard for a minute, I hope all of you can see the whiteboard. So, we would like to understand controllability and observability. These are the two concepts I would like to know what it is.

So, let us say the module that I want to test is a method, let us give it a name, so let us call it my method, it has some arguments and then it has code, let us say it has line 1 of code, then line 2 of code, I am not writing code but assume that there is code, and then a few lines and let us say after this there is a if statement, so if something happens then you do something and then there are more lines of code and then there is an else statement here and then there is more lines of code here.

And then as a part of this let us say there is a while loop, as a part of the else which is inside the if, there is a while loop and then there is code here and what I want to test is whether this loop is working correctly, this is the module that I want to test as a tester I suspect or as a developer I suspect that there is something wrong in this module and I would like to test it.

Now, you understand what the problem of observability and controllability will be, I have to first reach this method called my method, that itself could be somewhere inside the code and then I have to go execute all these statements, come to this if statement and what will happen to the condition of the if statement, this should fail because I need to go and execute the else part of the if statement which has a while loop inside that which is the unit that I want to test.

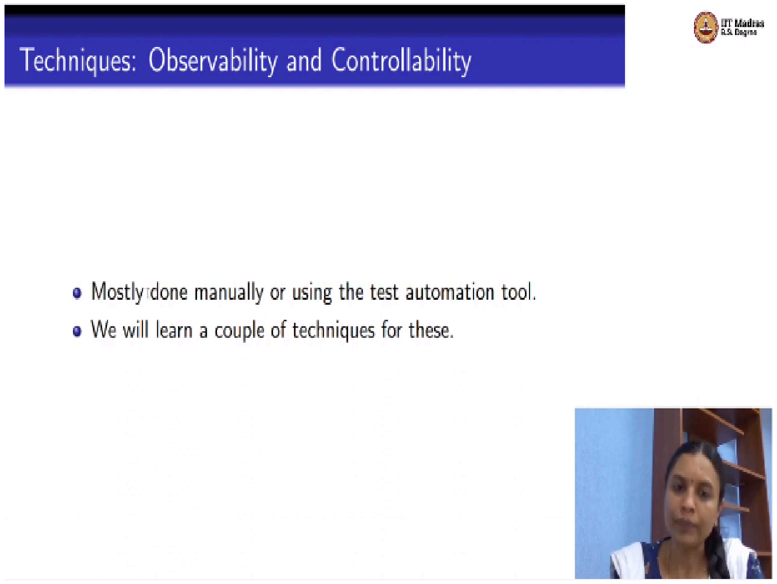
So, I have to first reach this method, reach this if statement, fail whatever the condition of this if statement is, I have to that condition should return false, which will enable this else part to be executed and this while part should be reached, I should reach here, once I reach this then I can test this.

So, for all that you know this while part, the variables that are involved here, they may not even be inputs, there could be some other set of variables that are here, these could be different variables, so the idea is I have to give what are test cases, they are basically inputs and expected outputs, I have to give test cases which are basically inputs that my method, this method can run on, reach the if statement, the statement should reach the if statement, reach the if, fail it, it should return false and then reach the while statement which is nested inside the if.

So, if I do this, this is called controllability, I have succeeded in giving inputs that are able to reach my module under test, now I run this while statement, after I run this while statement, there might be a few other pieces of code, so it goes on like that, it goes on like this and sorry, I just need to open the… anyway so it goes on like this and after this I should be able to observe my outputs, observe and record outputs, once I do this, this part is called observability, is this clear what controllability and observability means. So, this was a simple example to illustrate what controllability and what observability are.

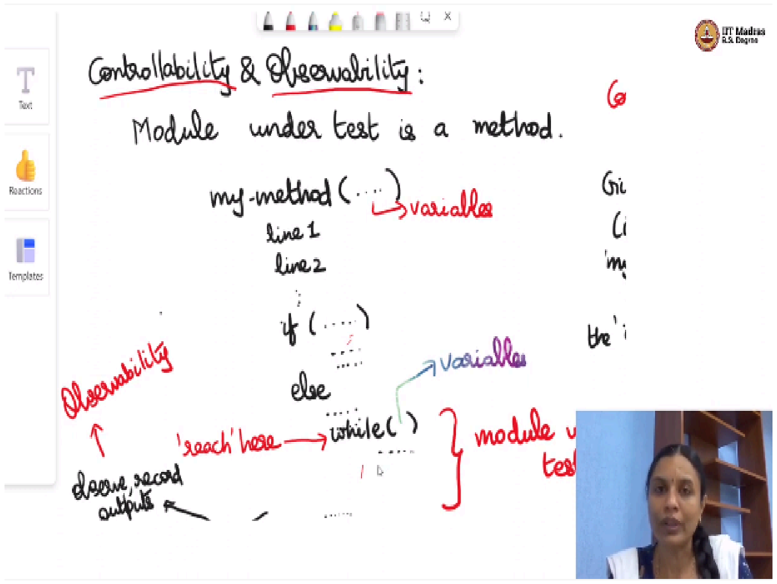
So, let me go back to my slides, so this is what I meant when I said the method that I want to test or just the unit of code that I want to test can be hidden deep in the code and may not accept any inputs directly. So, as a tester I have to ensure that the module under test is reachable, it can get infected and whatever output that it produces, as a unit of code, eventually propagates and becomes the output of the software that as a tester I can observe.

(Refer Slide Time: 18:28)



Techniques: Observability and Controllability

- Mostly done manually or using the test automation tool.
- We will learn a couple of techniques for these.



So, usually people do it manually observability and controllability, they work with a test automation tool where I can do my prefix and post fix annotations as we call it and get the automation tool to manage these properties of reaching the segment of code to be tested and letting the outputs of that segment of code trickle down as outputs to the software.

The test automation tool will provide prefix and postfix annotations for you to be able to do it or you could do it manually, in this course we will learn a couple of techniques for these but remember when I am testing the module that I want to test could be anywhere in the code and as a tester my job is not only to test the module to see how it is, but in that process I also have to be able to first reach the module that is what we call controllability and exercise or execute the model and then move on, record the outputs and observe what happens.

So, these are very important properties that I need to follow for testing so with this in mind the next module I will teach you the basics of one test automation tool that we will be using throughout the course, the tool is called JUnit, we will learn JUnit from a theoretical point of view and along with the instructors you will learn how to use it along with the IDE that you are going to be able to test a real piece of software that you are going to use. So, I will stop here for now with the basics, in the next class we will teach test automation, thank you