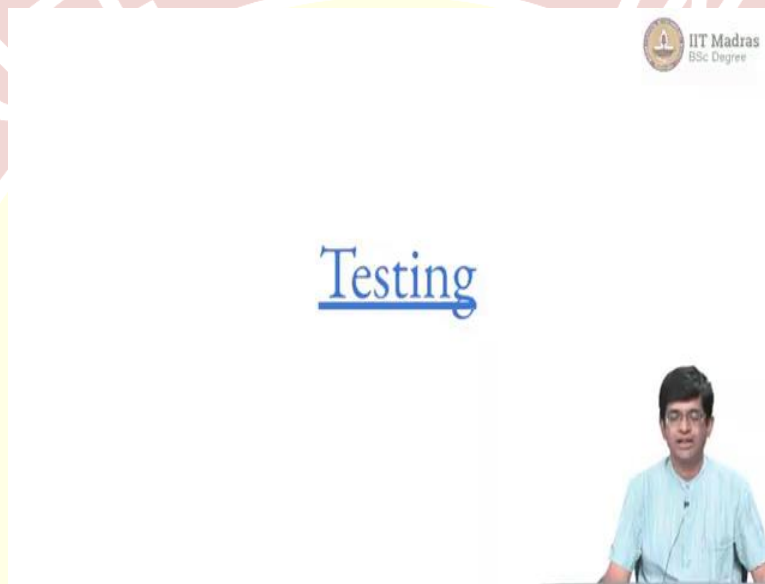


**IIT Madras**  
ONLINE DEGREE

**Modern Application Development – I**  
**Professor. Nitin Chandrachoodan**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**Application Testing**

Hello everyone and welcome to this course on Modern Application Development.

(Refer Slide Time: 00:17)



Hello everyone. Now, we are going to talk about what is an important topic but is not directly related to application development but applies to all kinds of software development in general which is the topic of testing. So, what we are going to look at is; what is testing; why do we need it; what are different ways of doing it. And also we will look at some specific aspects that are related to the concepts of the application development as we have been seeing it so far.

(Refer Slide Time: 00:45)

## Application Testing

- Why?
- What?
- When?
- How?
- Pytest



So, we can basically break it down into a set of questions, why, what, when, how and I will try and explain why I have put up those titles as we proceed and finally what I am going to do is look at one specific way of testing that we have in fact been using as part of this course and is one of the ways by which you can test python programs. I want to emphasize that it is not the only way of doing it.

In general one of the things you would hopefully have noticed throughout this course is that you know we are looking more at certain concepts rather than trying to say that this is the particular way of you know or the only right way of doing something. There are multiple ways of attacking any problem. Pytest happens to be one fairly good way of attacking the problem of how to test python programs and that is what we will go into in a little bit more detail.

(Refer Slide Time: 01:37)



### Why?

*Does something work as intended*

- Requirements - specifications
- Respond correctly to inputs
- Respond within reasonable time
- Installation and environment
- Usability and Correctness



So the first question; why should we test? And the obvious answer is; what is it that we are testing and does it work as it was intended to? So, does something and I am intentionally leaving this as a vague something whatever it is, it need not be software, in particular it need not be a web application that we are talking about. But the crucial part is does it work as intended.

Which means that the intention behind the design of the system whatever it is must be known right and that basically comes from what are usually called the requirements or the specifications of the problem. So, just to give an example it could be something as different from software development as maybe building a car right and at the end of the day you still need a set of specifications for a car right and in the case of the car the specifications would probably be it needs to be able to you know operate based on petrol.

It needs to accept petrol as its fuel, it needs to have turn on when certain conditions are met, it needs to be able to travel at a certain speed, it needs to be able to carry a certain number of people and so on right. And you would have an appropriate set of tests as long as you know if I do not have petrol; will the car operate? No, if I put diesel instead of petrol will it operate? Well it might but you know probably it will cause problems.

If I put in more than the adequate load, what will happen to the car? All those are tests that you need to run in order to find out whether the car is testing as intent or is working as intended. Now obviously we are interested in application development which means that our specifications

would be slightly different, it would probably be what kind of web page gets displayed; how do I decide what goes in which corner of the page?

Do I have the appropriate boxes where I can enter data; does it show me the correct page at the correct time for what I want to do? Or not the correct time as such but under the correct set of circumstances will it show me the correct page where there is a box where I can enter the data or if I need to select some particular thing like let us say a date; does it show me a date picker, something which allows me to select a date.

So, all those are parts of the requirements right and we need to somehow test whether those are met. Another question that we might ask is does it respond correctly to inputs? So, if I enter a date, does it actually take the date and store it somewhere, does it for example maybe the date picker was to find out if I am born in a certain year and does it respond appropriately with whatever is the right kind of message for someone born in that year.

Does it respond within a reasonable time? Now this is slightly more non-standard what I mean by that is it may not directly relate to the functionality of the system, you may be willing to accept a system where you do not really put a limit saying it has to respond within 100 milliseconds or 1 second. But there might be applications where it actually has to respond within a fixed amount of time.

One way of looking at it is you take exams online using a testing software which is running on some computer and those things have tight deadlines on when they are supposed to start and when they are supposed to finish. And they have to the software itself needs to be tested to make sure that once a person clicks on it, it actually registers the response and then moves on to the next question within a certain amount of time.

Because if it does not and it is running very slow that is problematic for the student who is taking the test. Now all these are sort of the functional specification after that we come to can it be installed properly, does it work in the right environment, can the environment it needs be set up in a systematic way? And finally it comes down to the usability and of course the correctness.

Is it doing the right thing and is it also usable by the end user. Now this usability and correctness especially are important for software, the whole idea of software usability right how user friendly is it, can it be actually used by the right kind of audience all of those are questions that we need

to ask. So, testing ultimately is asking these questions right, are all of them satisfied in other words is this something that can be used the way that the original designers intended it to be?

(Refer Slide Time: 06:07)

### Static vs. Dynamic

#### Static Testing:

- Code review, correctness proofs

#### Dynamic Testing:

- Functional tests
- Apply suitable inputs



Now what I am going to do over here is there are a number of different concepts related to testing and I am just going to be putting some of them down because these are terms that you need to know or be aware of as we move further into the domain of testing, so that you understand some of the concepts that we are talking about.

At a very basic level there are one distinction that you can make between different types of testing; is between so called static and dynamic testing. Now what would we mean by static testing? What I mean by static testing over here is something like perhaps what is called a code review. And in a code review what we do is we are not actually running the program, we are just looking at the code.

There are a bunch of people who are hopefully experienced software engineers or programmers and who look at the code and sort of look at it without running it and say okay this looks all right you know I mean the variables seem to be declared okay; the initializations are all right, the structure of the code looks okay and maybe give suggestions.

Look if you refactor this in some way or write it in a different manner, create a function to take care of this process all of those things are things that can be done in a code review. The important point and why it is called static is that the code is actually not running at this point.



Similarly, you there are certain types of software that can actually do proof checking on other software. So, if I write a program I can run it through a so called equivalence checker or a proof checker which will be able to prove that this program will actually do what I expected to do.

Now of course the question becomes how do I tell the proof checker what I expected it to do and how does it check that. So, there are a lot of intricacies over there but they can be dealt with and under certain circumstances at least they are a very strong way of proving the correctness of something that has been implemented.

Now, on the other hand the majority of testing that is done is actually a slightly different form, it is called dynamic testing. And in dynamic testing what we do is we actually run the program, we check its functionality by applying various kinds of inputs, I am saying apply suitable inputs which means that the suitable inputs might either be the correct inputs that the program expects in order to run or it might be inputs that are intended to try and break the functioning of the program.

As we will see later I mean there might be circumstances where you want to test for corner cases, what happens if I give the wrong input? The program should not crash or at least it should crash gracefully, it should allow me to recover from the problem. So, dynamic testing is the majority of the testing that we are looking at in most cases.

(Refer Slide Time: 09:00)

### White-box testing

- Detailed knowledge of implementation
- Can examine internal variables, counters
- Tests can be created based on knowledge of internal structure
- Pro:
  - More detailed information available, better tests
- Con:
  - Can lead to focusing on less important parts because code is known
  - Does not encourage clean abstraction
  - Too much information?



Now while testing a piece of software there are different sort of levels of detail that we can look at okay. At one level we call what is called white box testing. In a white box and the reason is called white box is to contrast it with what is the obvious next step which is the black box, and black box we sort of implicitly understand what it means. A black box is something that you cannot see inside.

So, by the terminology at least white box means something you can look inside right and you know everything about you have detailed knowledge of how the code is implemented. Now the good part about a white box approach to software is that I now know how the code was written I can look at each and every individual variable, counter you know all the different data structures that are there inside the system, maybe put traces on them right and watch the values of different variables and I can also create tests that are based on this knowledge of the internal structure.

Now the obvious benefit of such an approach is that I have a lot of detailed information about the system which means that I can usually construct better tests, I will be able to sort of say okay look this is likely to make the system crash. Let me try it out and you know see how it behaves okay make sure that the system cannot crash under this kind of input okay. And if I did not know that it was internally using a certain kind of variable then I would not have been able to guess something like that.

Now the disadvantage of this approach is that because you know the code sometimes you tend to focus on parts that are less important, you think that you know some particular array declaration or something else is the actual problem in the code. Whereas if you did not know that it was written using this particular array you might have said okay, look, I just need to test functionality and I will focus on just that part alone.

The other problem that happens is it does not encourage the designers to cleanly abstract and what I mean by that is this right from the beginning of this course we have been talking about the separation of concerns, the whole idea of HTML versus CSS was that HTML takes care of the semantic markup, CSS takes care of the styling. Similarly in model view controller, the model takes care of the data model.

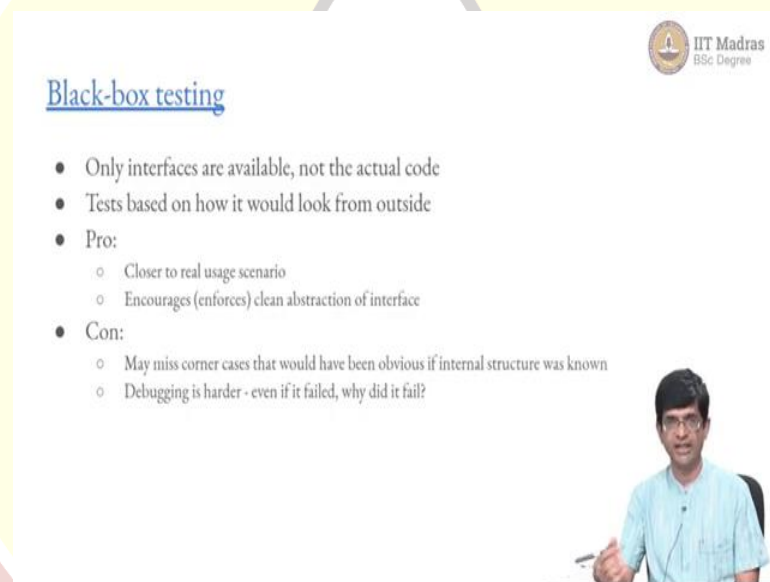
The view takes care of what is displayed to the user and the controller handles the interaction between the two. All those are examples of separations of concerns. Now in the same way when



you are designing something you would like to have a separation of concerns, in the sense that the person writing the software versus the person testing the software should ideally have a very clean interface between them.

And the person who is testing should only ideally be able to look at it and say this is what I expect as the behavior. I really do not want to know what is happening inside, white box testing although it makes certain kinds of debugging easier makes it harder to sort of enforce that kind of a clean abstraction between the two, in some cases. Partly it of course depends on the discipline of the team that is doing the development but you could argue that by enforcing it to start with you would have you know got that benefit all the way from the beginning.

(Refer Slide Time: 12:14)



The slide is titled "Black-box testing" and features a list of bullet points. In the top right corner, there is a logo for "IIT Madras BSc Degree". In the bottom right corner, there is a small video inset showing a man in a light blue shirt speaking. The slide is overlaid on a large, semi-transparent watermark of the IIT Madras logo.

- Only interfaces are available, not the actual code
- Tests based on how it would look from outside
- Pro:
  - Closer to real usage scenario
  - Encourages (enforces) clean abstraction of interface
- Con:
  - May miss corner cases that would have been obvious if internal structure was known
  - Debugging is harder - even if it failed, why did it fail?

So obviously, the opposite of (black) white box testing is what we will call black box testing. Now in a black box test only the interfaces are available, one way of thinking about this in terms of concepts we have already looked at in the course is that you are only given the API. You do not know how the API was implemented but you are given an API and you are given an API specification.

You are told that if you make a request to this particular endpoint with this kind of data, then you will get back this kind of a response. Now that is nice because after all the API is something that is going to be made public and I should be able to test at that level. The good news from the

point of view of the tester is that they are testing something that will actually be seen by the end user.

The problem is that even if I find an issue something does not behave correctly or something crashes I may find it harder to actually go in and debug. All I can say is something went wrong, I do not have any insight into what actually happened. The other problem that can happen is I am looking at it only as an API and it is quite possible that I might miss corner cases, certain kinds of inputs that could actually cause the system to crash.

But which I miss out completely because I do not know how things were implemented internally. So, let us say for example some counter inside the code right of course this does not happen in a language like python but let us say if it was C and it was implemented as a 16 bit short integer. I do not know that as a tester I just basically test out different values, give different inputs and say that this works.

But a person who actually knew it was implemented as a short might actually try out some set of circumstances which would cause that counter to go beyond 16 bits and then they would say yes you know that actually triggers a problem. So, there are circumstances where black box testing is not sufficient, it does not give you enough insight into what might be going on.

On the other hand the biggest benefit is it is closer to the real world usage scenario and of course it sort of enforces a clean abstraction. I am only able to give the test engineer, the interface I cannot tell them how the code was implemented.

(Refer Slide Time: 14:32)



### Grey-box testing

- Hybrid approach between white-box and black-box
- Enforce interface as far as possible
- Internal structure mainly used for debugging, examining variables etc.



Now obviously we would like to also have some kind of hybrids where we call which we call a grey box test which is a mix of black and white. And the idea behind grey box testing is you try and do the black box testing to the extent possible. But the moment you hit a problem you would want to go into the internal structure because otherwise you cannot debug.

Now grey box testing requires that the person doing the testing or the team doing the testing has to have the detailed knowledge of the internal structure as well or at least should be able in close communication with whoever has done that internal implementation.

(Refer Slide Time: 15:08)



### Regressions

- Maintain series of tests starting from basic development of code
  - Each test is for some specific feature or set of features
- **Regression:** loss of functionality introduced by some change in the code
- Future modifications to code should not break existing code
- Sometimes necessary
  - Update tests
  - Update API versions etc.



Now, apart from the box approach to testing another term which comes up often is this concept of regressions or regression testing right and the idea behind the regression is that you know you have implemented certain functionality in your code and you are continuing to do so and the whole idea of software development is seen as a continuous process. It is not just that okay I write code, it is done and complete.

I have a large design that i need to implement and I go towards it step by step. I implement part of the functionality let us say I am doing the grey book I would first be able to create a student, then edit a student, then create a course, edit a course, then finally I would add the ability to combine the two and assign marks for a student for a given course. At each point I want to be able to test and say that look now that I am working on courses, I do not want anything on the student side to break. Just because I have added a new table in the database to handle courses I do not want the student interface to break.

Similarly, when I start doing student course marks I do not want the interface or the view for the students to break. Now each of those would be a test. So, for example I might have a test which says that when I want to add a student it presents me with a certain view and that view should contain the ability you know a button that allows me to add a new user right and so on.

Now what happens if I change something as a result of you know change some internal column in one of the tables in the database and because of that one of the other views in the student view breaks. It does not display as it was supposed to, that is called a regression. It means that some functionality that was previously working has stopped working and regression testing is precisely to take care of such scenarios.

All it says is, anything that has already passed a test keep track of it right and automate it because it is impossible to run each of these tests by hand for every change that you make to the system. But if you can automate it that is great because after making every change I can run a regression test suite, a test suite is basically a collection of tests. So, the regression test suite will go back and make sure that nothing broke as a result of the change that you made now.

Sometimes regressions are necessary meaning that you might actually have to make a breaking change in an API or in a database structure or something else. At which point usually the approach that is followed is, if this is production code that other people are already using you

have to inform them and you know usually what you would do is; you would say okay there is a version change.

This is why for example; API's go from V1 to V2 where they say that look you can no longer use V1 because there have been some changes introduced that will no longer work in V1 right you need to switch over to V2. So, now how do you handle V1? One possible way is you completely break it, you say that any request to the V1 API will just result in an error code which means that whoever had written code using the V1 API will have to go back and fix their code.

The other possibility is, you maintain multiple versions, you say that if you are using V1 yes I will continue to support it but you would not have these features. But the point is a regression test suite allows you to keep track of all of this and tell you what breaks at which point.

(Refer Slide Time: 18:42)

### Coverage

- How much of the code is covered
  - Every line is executed at least once - 100% code coverage
  - Does not guarantee "correctness" in all conditions
  - There may be more complex paths or other conditions that can cause failure
- Branch coverage, condition coverage, function coverage ...



Now there is another important concept in testing which is called code coverage right and basically this refers to how much of the code that you have written is covered by tests. Now what exactly does that mean? What do we mean to say when we say that a certain amount of code is covered by tests, what it ultimately you know the only thing that you can really say is, is every line in the code addressed by some test meaning that will that line get executed as part of some test. And if that is the case then I can say yes I have 100 percent code coverage.

Now you have to be very careful with terminology like this because it is entirely possible to have a hundred percent code coverage but the code coverage does not really check all possible

breaking scenarios for the code. In particular it only means a particular line of code got executed at least once as part of some test. But did it get executed for all possible different inputs or different values of the variables that are over there?

That is obviously impossible, especially if I have a you know real numbers or something like that as the input right, it is not possible to test it for all possible test cases. So, a code coverage of 100 percent is not a guarantee that your code is actually bug free, it is not at all a guarantee of that all it means is you at least looked at it once.

So, please keep that in mind and in addition to code coverage which is just line by line coverage there are a few other forms of coverage which all ultimately have the same problem, they do not guarantee that your code is bug free. All that they can say is okay somebody at least looked at it once and some test over there has at least examined this code, so there are no obvious breaking problems with this. There might still be issues under certain circumstances but it is not that you know the code is fundamentally flawed or syntactically wrong at least.

(Refer Slide Time: 20:46)

Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Function coverage

- Test invokes foo() at least once

From Wikipedia

IIT Madras  
BSc Degree

So, let us take an example; this is an example that is actually treated in Wikipedia, so I am going to use the same thing because it is a nice compact example that illustrates all the points that we want to make. And what we have here is there is some function, this is a programming language which is basically like C and what you can see is that it takes there is a function that takes in 2 inputs x and y, it declares some internal variable.



It has some condition check on x and y. Based on that condition check it updates the value of z and then returns that value. So, if this condition was not satisfied it should return a 0 and so on. So, I can think of some tests that I could run for this. For example, I might find that you know I should be able to call this function and say that if the x and y values do not satisfy this condition, then the zip condition will not be taken and z should get this value 0 which is what it was initialized to.

So, one set of tests that I can create is if  $x=0$  this condition is straight away not satisfied, I should get back  $z=0$ . Then I would have some test which basically says  $x=1$ ,  $y=1$  and at that point I should get back  $z=1$  which is the value of x. So, I can write a few tests saying that  $\text{foo}(0,0)$  should give the value 0,  $\text{foo}(1,1)$  should give the value 1;  $\text{foo}(1,0)$  should give the value 0 because there the y greater than 0 would not be satisfied and so. So, those are a set of tests that I can think of.

Now in terms of testing i can also think at different levels. The first is; do I have function coverage? In other words do I have at least one test somewhere in my set of tests which calls this function foo? Now obviously when I have just this one piece of one function over here; the only test I can think of is something that does call this function foo.

But let us say I am writing a library of routines where there are like hundreds of different functions that could be called, I need to make sure that every function gets called at least once. So, that is function coverage.

(Refer Slide Time: 23:00)



### Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

#### Statement coverage

- Example: foo(1,1)
  - All statements in code will be:



Then I could go to statement coverage and over here in statement coverage what I want to check is; okay I have 1, 2, 3, 4, 5, 6 lines of code inside the function. Is each one of them going to get executed under certain test cases? And the example that we can give over here is if I call it with foo(1,1), what will happen with x=1; y=1? The first line int z = 0, yes will be executed. The if condition? Yes, it will be executed of course right because it is the next statement after that and then because both these conditions x greater than 0 and y greater than 0 are satisfied I will go in here.

Of course the open brackets and close brackets are there but they are not really statements right. So, the next sort of statement is going to be z=x, yes that will be executed as well. And finally the return z will also be executed. So, all the four sort of lines of code are going to be executed and therefore I can say, I have 100 percent statement coverage, as far as these tests are concerned.

With that one test foo(1,1), I have 100 percent statement coverage. Clearly you can see that you know the fact that I have got 100 percent statement coverage does not mean I have actually tested the code because I tested it only for one set of conditions; x=1, y=1, I do not know how it will behave under other conditions and whether it will do the right thing.

(Refer Slide Time: 24:34)



### Example

```
int foo (int x, int y)
{
    z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

### Branch coverage

- At least two tests needed:
- foo(1,1)
  - Branch taken
- foo(1,0)
  - Branch not taken



The next is so called branch coverage, I have an if condition over here which means there are two possibilities; one is, I take the if which means I go inside and execute what is there. The second is; I do not take the if and jump straight to the return statement. So, I need two conditions one of which will take the branch that for example is foo(1,1). And the other one which will make me skip over and go to the return straight away.

For example foo(1,0) will do that okay, so you can see that by having two tests I can ensure 100 branch coverage. At least all the possible branches over here have been exercised at least once.

(Refer Slide Time: 25:18)



### Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

### Condition coverage

- At least two tests needed:
- foo(0,1)
  - First condition fails, second succeeds
- foo(1,0)
  - First condition succeeds, second fails
- Note: does not guarantee branch coverage



And finally there is something else called condition coverage. Now in condition coverage what we are saying is look what were the conditions inside the if statement there was a  $x > 0$ , there was a  $y > 0$ .

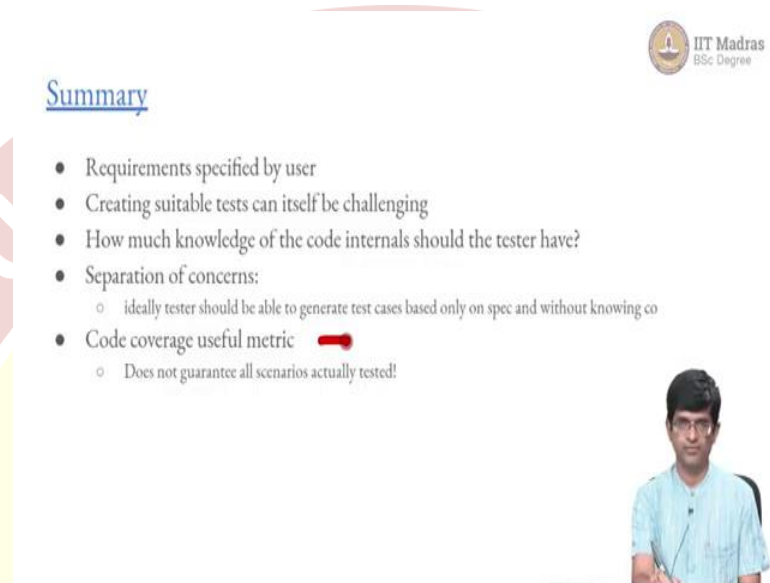
What are the possibilities for  $x > 0$ ? Either it is  $> 0$  or it is  $= 0$  or it is  $\leq 0$ . So, foo(0,1) would for example test, the first condition fails or rather the first condition fails; the second condition succeeds. Whereas, foo(1,0) would say that the first condition succeeds and the second or rather the first condition succeeds and the second condition fails.

Now if you look at those two tests you will realize that under both these conditions the branch will actually not be taken. For that, for the branch to be taken and to go inside and execute this  $z=x$ , I need  $x > 0$  and  $y > 0$  at the same time which is not satisfied in either of these conditions. Which means that the condition coverage over here I have 100 condition coverage, in the previous case 100 branch coverage.

But the 100 condition coverage does not guarantee 100 branch coverage nor vice versa. So, the takeaway from all of this is that you know 100 coverage of tests should not give you a false sense of security, it should not be taken to mean that my program is bug free, not at all. It only means that it has been looked at under certain circumstances, those basic tests that you tried passed hopefully the tests are good enough to look at all possible you know problematic cases.


But you know do not sort of assume that just because somebody says oh they have 100 percent code coverage, the program is great and it is bug free, it is not. And that is not a guarantee of anything of that sort.

(Refer Slide Time: 27:16)



IIT Madras  
BSc Degree

### Summary

- Requirements specified by user
- Creating suitable tests can itself be challenging
- How much knowledge of the code internals should the tester have?
- Separation of concerns:
  - Ideally tester should be able to generate test cases based only on spec and without knowing co
- Code coverage useful metric 
  - Does not guarantee all scenarios actually tested!

So, to summarize everything I mean regarding testing, the requirements for any test ultimately have to be specified by the user. And creating the right kind of tests itself is a huge challenge, it is not at all trivial. One big question that has to be answered is; how much knowledge of the code internal should the tester have? This is white box versus black box testing.

And ideally we should be able to separate out the concerns, the tester should only be looking at it as a white box but at the same time they should also be able to give debug information which means maybe the grey box is the right approach. But grey box is not clearly enough defined, it does not say how much information the tester should have.

So, because of all of this there is always you know some level of subjectivity in how we go about implementing such tests. And finally, there was this metric called code coverage that we looked at which is a useful metric because it tells you that at least you have a certain set of tests, it is by no means enough to guarantee that code is bug free.