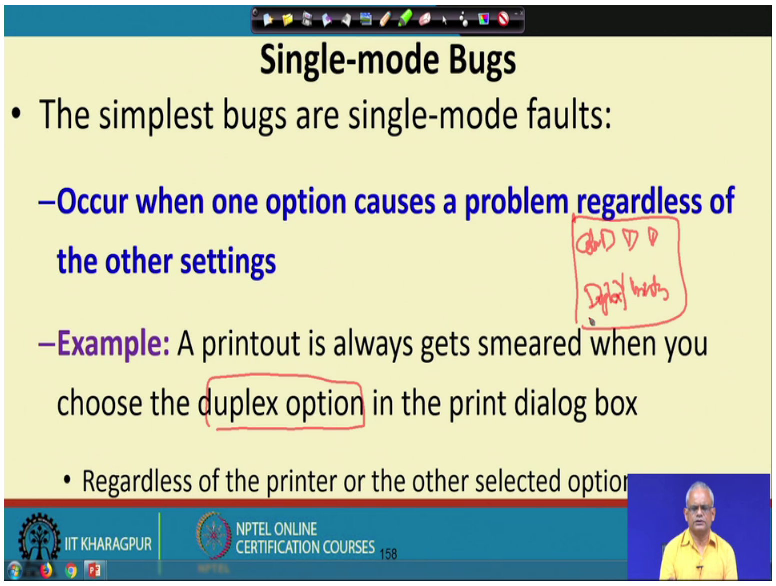
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 54
Pairwise Testing

Welcome to this lecture. In the last lecture we discussing Pairwise Testing and, we had mentioned that the combinatorial testing techniques like a decision table based testing. If, the tendency to generate a large number of test cases, when the number of input conditions are large, actually the number of test cases are exponential in the number of input conditions.

(Refer Slide Time: 00:47)



Single-mode Bugs

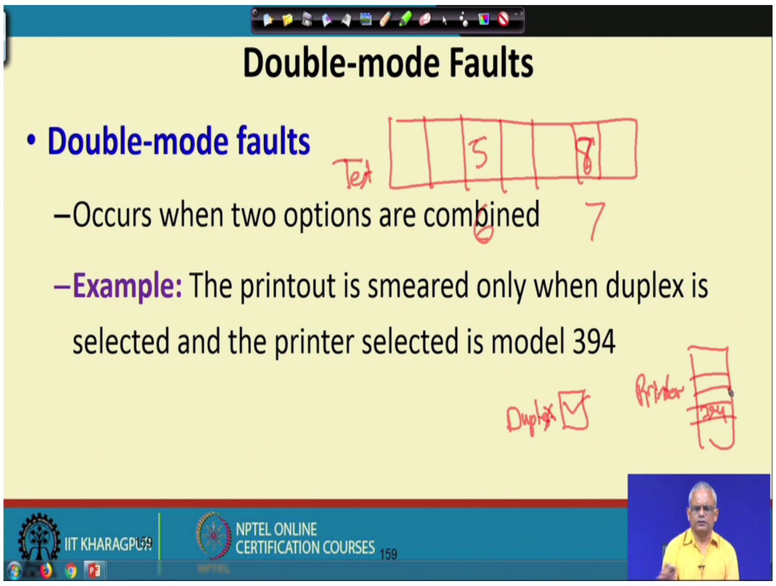
- The simplest bugs are single-mode faults:
  - **Occur when one option causes a problem regardless of the other settings**
  - **Example:** A printout is always gets smeared when you choose the duplex option in the print dialog box
    - Regardless of the printer or the other selected optio

And, we are discussing about reducing the number of test cases and pairwise testing is a very promising way of reducing the number of test cases. And, we are trying to get some insight into how does the pairwise testing produce such a thorough testing with a very very small number of test cases; a drastic reduction from a million test cases to about 7 or 8 test cases and so on.

And, for that we are trying to discuss the characteristics of the bugs. And, we are saying that the bugs are such that as long as input condition has some value, if the bug is expressed as a failure then we call it as a single mode bug. So, a single mode bug, when there is a certain input value is given, then irrespective of all other input the failure occurs. Just to give an example: as long as the duplex option is chosen on the printer dialogue box. There may be other options this duplex this is the printer type color or black and white and so on. There are many options here. Irrespective of all other options as long as you duplex is selected: yes, then the print gets smeared.

So, if we think of the code if this becomes yes, then in the printer code there is some problem as long as this becomes yes. So, this is called as a Single-mode bug. The bug expresses itself, when as long as some input condition is given value, certain value.

(Refer Slide Time: 03:16)



Double-mode Faults

- **Double-mode faults**
  - Occurs when two options are combined
  - **Example:** The printout is smeared only when duplex is selected and the printer selected is model 394

A double-mode fault occurs when 2 options have a specific combination.

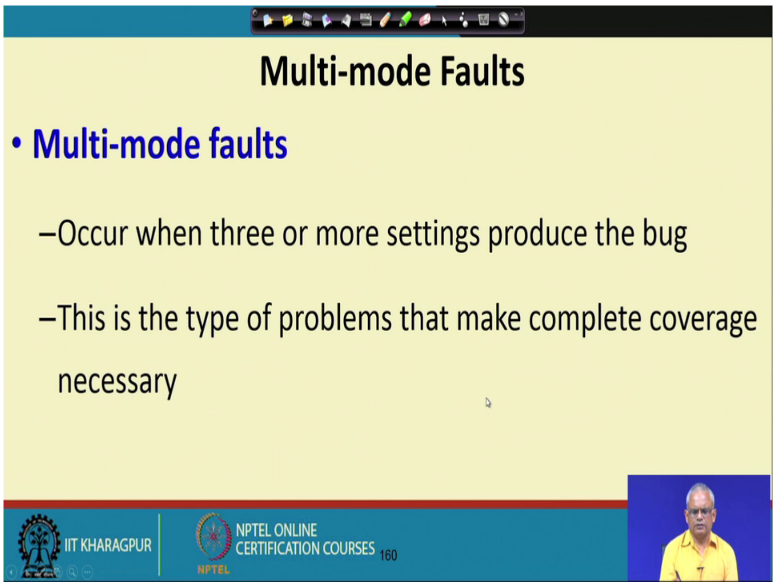
Let us say, we have many inputs many input parameters. And, as long as this is given a value of 5 and this is given 7 then the problem occurs, but if we have given them separately a value of 5 and at that time it is 8 and this is given 6 and this is 7. So, even though the 5 7 it is tested, but a test case. So, these are test cases, which have values for all the parameters defined.

As long as they are getting the value 5 and 7 for these 2 parameter there is a failure, but for all the other combinations it works. An example is that the printout smeared only when the duplex is selected and the printer model is 394. So, a duplex option is checked yes and also the printer model there are many printer, the printer model is selected to be

394, but if same 394 works when it is not duplex and with duplex any other printer will work.

But, then to get a further insight into how does this? How does this happen? That a specific combination of condition causes the problem, some input variable a specific combination of the values of some 2 3 or so, input values. They express themselves as a failure whereas, individually they may be taking the value, but that does not cause the problem.

(Refer Slide Time: 05:32)



Multi-mode Faults

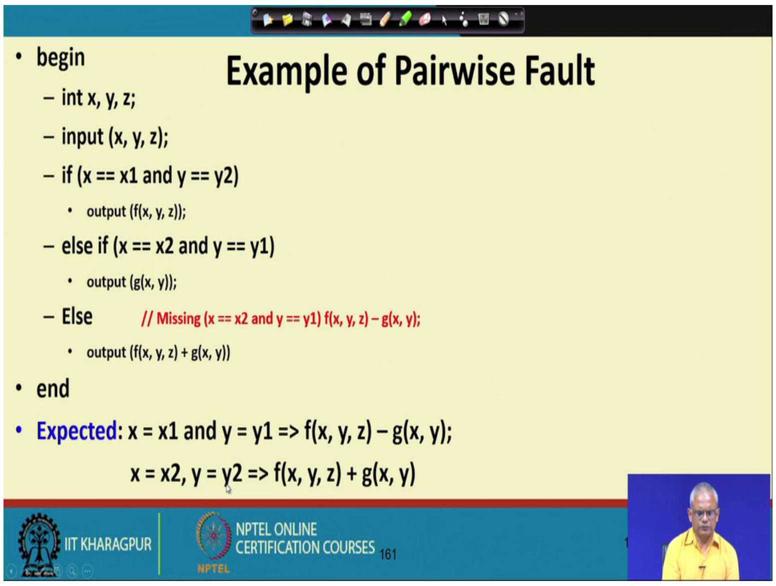
- **Multi-mode faults**
  - –Occur when three or more settings produce the bug
  - –This is the type of problems that make complete coverage necessary

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
160

We will see a code example and understand why that situation occurs?

In general you can have multi-mode faults. As long as all the input parameters that are given, they have some specific combination of all the input parameter causes the problem. It can be 3 way faults when 3 specific inputs have some value combination; it can be 4 way fault, and so on.

(Refer Slide Time: 06:10)



Example of Pairwise Fault

- begin
  - int x, y, z;
  - input (x, y, z);
  - if (x == x1 and y == y2)
    - output (f(x, y, z));
  - else if (x == x2 and y == y1)
    - output (g(x, y));
  - Else // Missing (x == x2 and y == y1) f(x, y, z) – g(x, y);
    - output (f(x, y, z) + g(x, y))
- end
- Expected: x = x1 and y = y1 => f(x, y, z) – g(x, y);
  x = x2, y = y2 => f(x, y, z) + g(x, y)

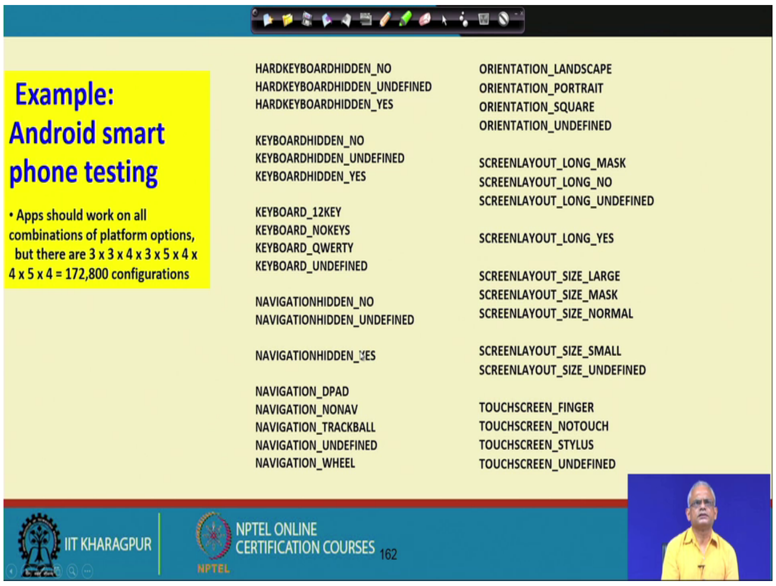
Now, let us try to gain the insight into why this occurs? That we have pair wise that is single mode and double mode faults are the majority of the faults. Let us look at this code here a simple code here and we have some 3 inputs x y z. And, then we have caused we have conditions defined on this input.

If, x is equal to x 1 so, so, we are reading 3 values x y z these are our input and if x is x 1 and y is y 2 then output is some f x y z. If x is x 2 and y is y 1 then output is g x y, but then the programmer has missed one of the condition here. If x is x 2 and y equal to y 1 then the output should be f x y z minus g x y, but then he has written here else output is f x y z g x y. So, this will work fine as long as we give x x 1 y y 2 etcetera, etcetera all possible combinations it will work, but it will fail only when x is x 2 and y is y 1. So, a specific combination between x and y to it will fail and that is here.

Instead of displaying f x y z minus g x y it will come here under the else and it will display f x y z g x y. And, this is a typical code that appears in many programs. And therefore, the expressions are formed using pair or 3 combinations of conditions. And, if there is a problem under that then if specific values are given such that this becomes true, then only the condition in the failure will appear.

So, in this case when x is x 2, when x is x 2 I should make it x 2, when x is x 2 and y equal to y 1 f x y z minus g x y should have been updated, but it actually displays f x y z plus g x y, that is the problem and unless this x is given x 2 and y is given y 1 the problem will not be detected.

(Refer Slide Time: 09:30)



Example: Android smart phone testing

• Apps should work on all combinations of platform options, but there are 3 x 3 x 4 x 3 x 5 x 4 x 4 x 5 x 4 = 172,800 configurations

HARDKEYBOARDHIDDEN_NO
HARDKEYBOARDHIDDEN_UNDEFINED
HARDKEYBOARDHIDDEN_YES

KEYBOARDHIDDEN_NO
KEYBOARDHIDDEN_UNDEFINED
KEYBOARDHIDDEN_YES

KEYBOARD_12KEY
KEYBOARD_NOKEYS
KEYBOARD_QWERTY
KEYBOARD_UNDEFINED

NAVIGATIONHIDDEN_NO
NAVIGATIONHIDDEN_UNDEFINED

NAVIGATIONHIDDEN_YES

NAVIGATION_DPAD
NAVIGATION_NONAV
NAVIGATION_TRACKBALL
NAVIGATION_UNDEFINED
NAVIGATION_WHEEL

ORIENTATION_LANDSCAPE
ORIENTATION_PORTRAIT
ORIENTATION_SQUARE
ORIENTATION_UNDEFINED

SCREENLAYOUT_LONG_MASK
SCREENLAYOUT_LONG_NO
SCREENLAYOUT_LONG_UNDEFINED

SCREENLAYOUT_LONG_YES

SCREENLAYOUT_SIZE_LARGE
SCREENLAYOUT_SIZE_MASK
SCREENLAYOUT_SIZE_NORMAL

SCREENLAYOUT_SIZE_SMALL
SCREENLAYOUT_SIZE_UNDEFINED

TOUCHSCREEN_FINGER
TOUCHSCREEN_NOTOUCH
TOUCHSCREEN_STYLUS
TOUCHSCREEN_UNDEFINED

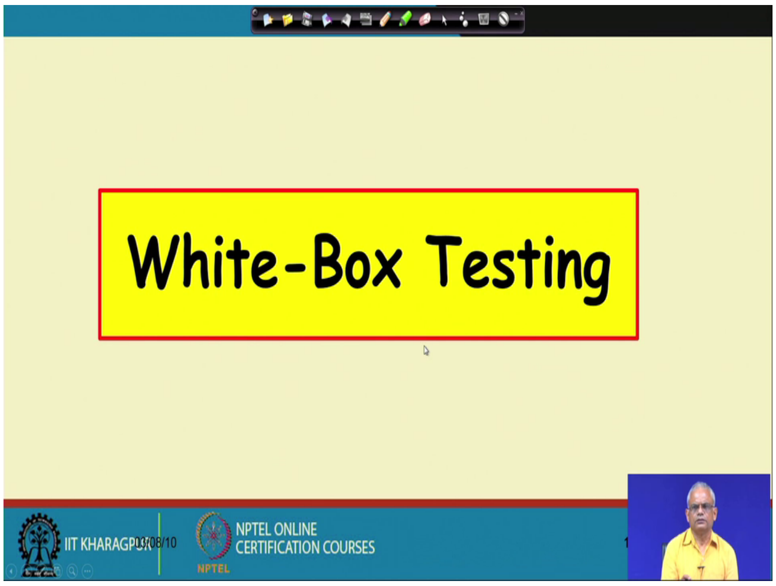
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 162

This occurs in many many examples; let us consider android based smart phone testing.

The android has many environmental variables global variables. And the global variables are whether the hard keyboard hidden no undefined yes keyboard, keyboard is 12 key, nokey, qwerty, navigation, navigation touch screen, whether it is a finger no touch, screen it is a stylus or undefined screen layout small, large etcetera, etcetera. So, just see here that the number of options required to do an exhaustive testing is if we see that this is 3 option 3 option, and so on.

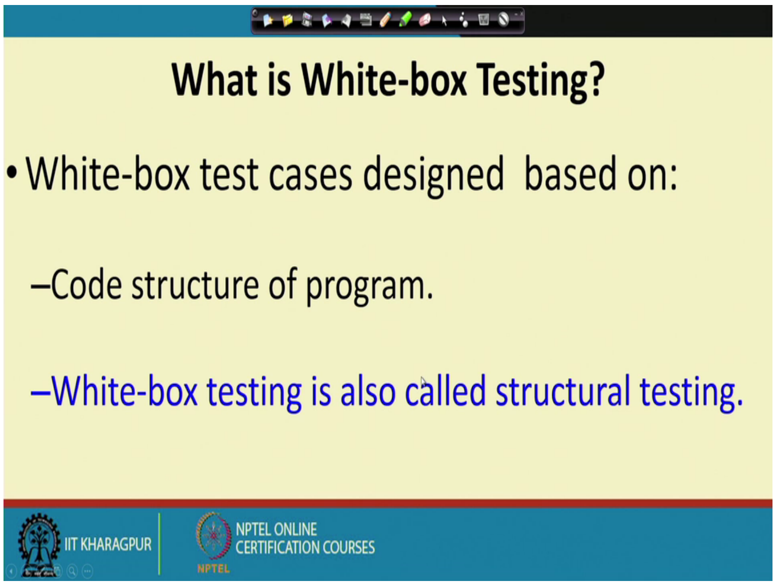
To do an exhaustive testing we need 172800 test cases, which is a large number, but if we consider pairwise or 3 way testing or 4 way testing, then the number can drastically reduced to a manageable number. And we would have also detected almost every bug and that is why the 3 way testing is a very attractive way, it makes combinatorial testing techniques practical.

(Refer Slide Time: 10:58)


White-Box Testing

Now, let us discuss about white box testing. So, far we have been discussing black box testing. And, we looked at several test strategies of black box testing. Now, let us look at white-box testing.

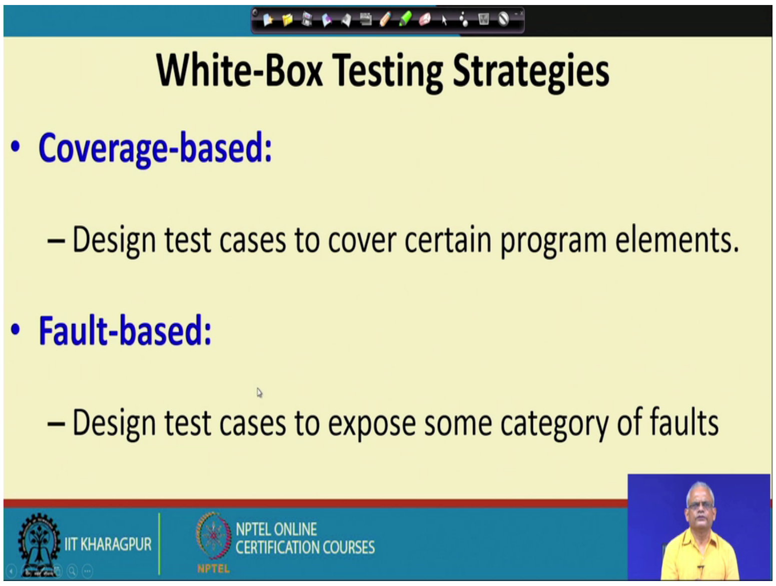
(Refer Slide Time: 11:21)


What is White-box Testing?

- White-box test cases designed based on:
  - Code structure of program.
  - White-box testing is also called structural testing.

White-box testing as we mentioned earlier, here the test cases are designed based on the code structure of the program. We must have the code available to us we examine the code and based on the code we designed this test cases, the white box test cases. And therefore, it is also called as the structural testing, because the test cases are designed based on the code structure.

(Refer Slide Time: 11:52)



White-Box Testing Strategies

- **Coverage-based:**
  - Design test cases to cover certain program elements.
- **Fault-based:**
  - Design test cases to expose some category of faults

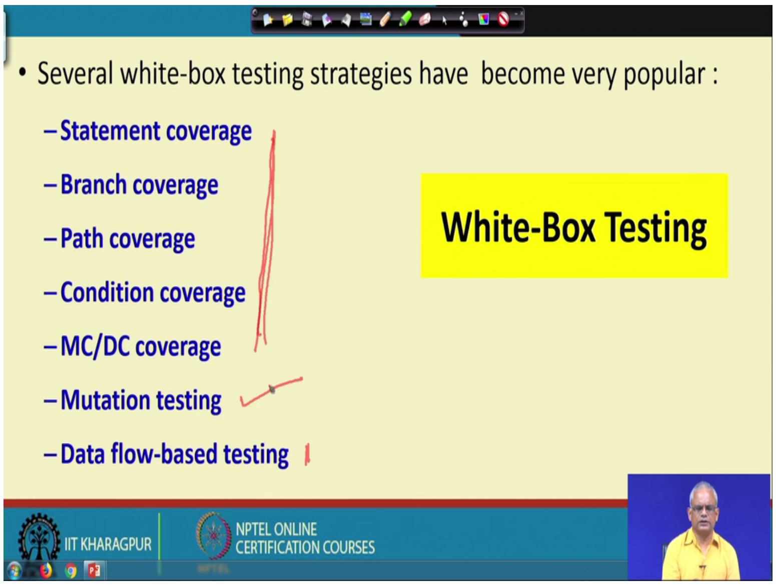
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Again, we have 2 major categories of white box testing; one is called as coverage-based testing, the other is fault-based. In coverage-based testing, we see the program elements, we observe the program elements and based on that, we design test cases such that the program elements are covered. The program element can be a statement, it can be some decisions that occur in the software in the program, it can be some paths control flow in the program and so on. So, the program element is a generic term we have used and based on what we consider as program element? We will have different strategies of coverage based testing.

We can also have fault based testing here we design test cases to expose some category of faults. We observe the program and if we find that there are arithmetic operators let us say plus and so on. We can using fault based testing, we can check whether the there are any problems with respect to the plus operator becoming some other operator are those getting tested. So, in coverage based testing we will try to design test cases to cover certain program element and program element can be a statement it can be a decision can be a control flow and so on.

Whereas, a fault-based testing we observe the code and identify what are the ways in which faults can occur? The faults may get occur introduced, because an arithmetic operation was not done correctly, it may be because a variable type was not correct maybe that some statement was missed; so the faults here at the typical faults that a programmer makes writing the program. And here in fault based testing we check whether the test cases are able to expose these specific types of faults that the programmer might commit.

(Refer Slide Time: 14:46)



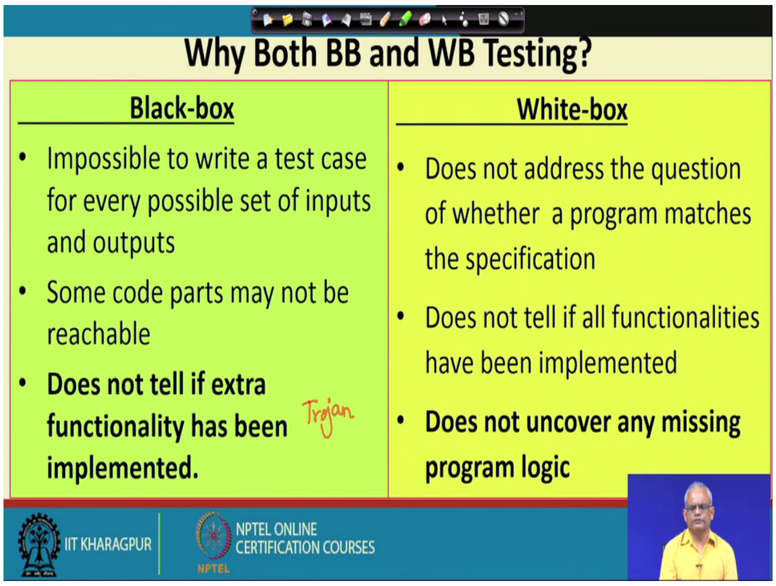
- Several white-box testing strategies have become very popular :
  - Statement coverage
  - Branch coverage
  - Path coverage
  - Condition coverage
  - MC/DC coverage
  - Mutation testing
  - Data flow-based testing

White-Box Testing

Now, first let us look at the coverage based testing. We will discuss statement coverage, branch coverage, path coverage condition coverage, MC DC coverage and also data flow coverage. So, these are the different coverage based testing techniques that we will discuss. And, we will also discuss a fault based testing which is called as the mutation test.

So, these are all coverage based testing, data flow testing, statement, branch, path, condition, MC DC etcetera. We will discuss each of this testing technique and also we will discuss one fault based testing called as mutation testing.

(Refer Slide Time: 15:35)



Why Both BB and WB Testing?

Black-box

- Impossible to write a test case for every possible set of inputs and outputs
- Some code parts may not be reachable
- **Does not tell if extra functionality has been implemented.** Trojan

White-box

- Does not address the question of whether a program matches the specification
- Does not tell if all functionalities have been implemented
- **Does not uncover any missing program logic**

But, before we discuss about the white box testing the coverage based testing techniques a fault base testing and so on. Let us be clear about one is so, that if we are doing black box testing, do we need to do white box testing or if we are doing white box testing is it necessary to do black box testing.

If, we want to answer that ok, we need both this testing. Then we should be able to give an example, that what cannot be detected by black box testing will be detected by white box testing? And also an example of what cannot be detected by white box testing, but will be detected by black box testing? That will justify our claim that we need to do both the testing; if, we cannot even give one example of a bug, which can only be detected by white box testing not by black box testing.

And, similarly some category of bugs, which can only be detected by the black box testing and not the white box testing, then we would have somehow justified that we need both. But, if we can give no example that, which can be detected by only black box or only white box then we will not be justifying why we need both testing? We might as well do just one case testing.

So, let us based on that motivation, let us discuss the difference between black box and white box and why we need both? One of the main shortcoming of the black box is that that, we cannot do exhaustive testing black box testing. And, also even if we do a thorough testing based on the techniques that we discussed, we cannot guarantee that all code parts have been executed. It may be possible that the programmer who has written the code? He has a written some extra code which is a Trojan.

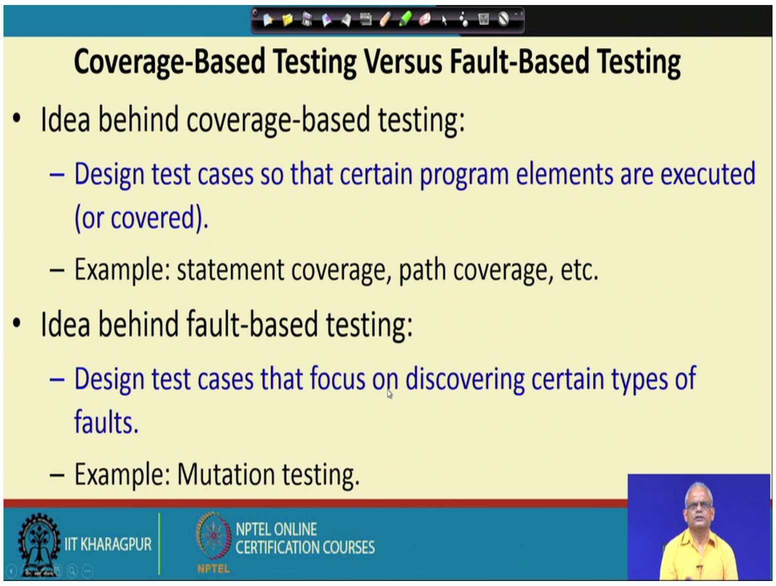
Only the programmer knows, we tested according to the test cases designed based on the input output behavior that is specified, but the Trojan will not get exposed. Because the programmer has written a secret key, only when that key is pressed the Trojan gets activated. It is very difficult to discover Trojans by only black box testing, we need to examine the code and see whether all parts of the code are thoroughly been tested.

So, this is the justification, why we need to do white box testing? Because, if the programmer has written some extra functionality not there in the specification, we cannot detect that using black box testing, because it is not listed in the specification and we cannot design test cases for that. But, let us see why we need to do black box testing. One problem with white box testing is that we have been given the code and we design test cases exactly based on the code, but then just by examining the code we design test cases, but we can find whether all the code is executed and so on. But, we cannot tell if all functionality has been implemented, this is there in the requirement specification.

If the programmer has forgotten to write the code for some program logic, we cannot detect that using only white box testing, because the code itself is absent, unless we check the black box specification and see design test cases for every function, we cannot check whether some functionality has been omitted by the programmer.

So, that gives us the justification, why we need both black box and white box testing? Using just black box testing, we cannot know whether Trojans have been implemented or the programmer has added some extra functionality. Whereas, using white box testing we cannot know whether there is a missing functionality, the programmer has forgotten to write code for some specific functionality.

(Refer Slide Time: 21:14)



Coverage-Based Testing Versus Fault-Based Testing

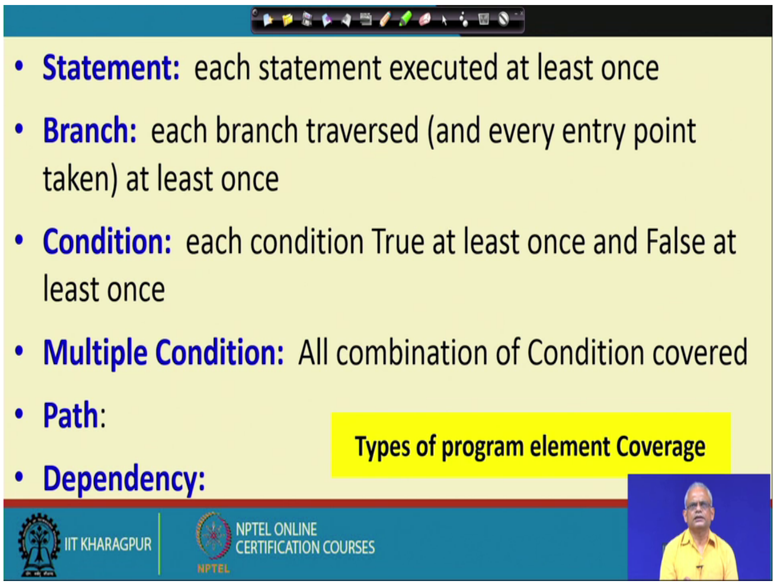
- Idea behind coverage-based testing:
  - Design test cases so that certain program elements are executed (or covered).
  - Example: statement coverage, path coverage, etc.
- Idea behind fault-based testing:
  - Design test cases that focus on discovering certain types of faults.
  - Example: Mutation testing.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now, we said that the white box testing mainly the coverage based testing techniques and the fault based testing techniques. In the coverage based testing techniques, we said that when we check whether the designed test cases, execute some program elements. For example, whether all the statements are getting executed, we call it as statement coverage. All the paths in the program are executed that, we call as path coverage. So, here we cover or execute all the program elements that we consider.

In fault based testing, we focus on discovering certain types of faults. That is whether, if the programmer has done a mistake, in writing the logical expressions are the test cases thorough enough to design all errors in logical expressions and so on. One prominent example of fault based testing is the mutation testing.

(Refer Slide Time: 22:26)



- **Statement:** each statement executed at least once
- **Branch:** each branch traversed (and every entry point taken) at least once
- **Condition:** each condition True at least once and False at least once
- **Multiple Condition:** All combination of Condition covered
- **Path:**
- **Dependency:**

Types of program element Coverage

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

And, we said that in the coverage based testing the type of coverage is defined based on the type of the program element. And, we will consider statement based coverage, branch coverage, condition coverage, multiple condition coverage, path coverage, and dependency coverage.

Thank you.