

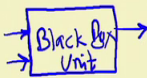

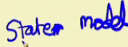
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 46
Unit testing strategies-I

In the last lecture we had started to discuss about unit testing we looked at some aspects of unit testing the stubs and drivers. The stubs and driver are small software that needs to be written by the testers before they can start carrying out the unit testing and of course, the unit testing is done by the developer themselves and therefore, these are the developers who do unit testing and therefore, they write the stubs and drivers. The independent testers they do the integration and system testing who were part of the test team.

(Refer Slide Time: 01:12)

Design of Unit Test Cases

- There are essentially three main approaches to design test cases:
 - Black-box approach 
 - White-box (or glass-box) approach 
 - Grey-box approach 

The slide footer includes the IIT Kharagpur logo, NPTEL ONLINE CERTIFICATION COURSES 256, and a small video inset of the professor.

And the unit testing is performed by the developers themselves and before they can unit test unit they need to write the drivers and stubs and then carry out the unit testing. Now let us look at more detail into unit testing. How does one design unit test cases, let us say we want to do unit testing for a function as I was mentioning that unit testing can also be done for a module, or for a class, or for a component, but let us now assume that we are interested to do unit testing for a function.

Now, what are the main approaches to design unit test cases one is called as the black box approach, the other is white box approach and the third is grey box approach. For a function unit testing is basically a black box and white box we will see how to design those test cases and why these are called as black box and white box, but for class or a module or a component we might also have to do a grey box approach in black box approach we do not need to know the entire detail of the software we just view the unit as a black box.

This is our unit we just know that if we give input to the black box unit it should behave in some way in the sense that it will produce some output. So, the black box approach we design the test cases by observing the inputs that should be given and output that should be produced. On the other hand in the white box or glass box approach we know the details of the code that is there.

And therefore, we can create a model of the code for example, a control flow diagram or something and then based on the internal knowledge of the code we might design some test cases. So, this is called as a glass box, because we need to check the internal code here we do not even bother about what is the code we just look at it is specification, it is what is the input and what output should be produced and based on that we design the test case, on the hand in the grey box approach we have something in between a black box and a white box.

We use the design of the unit may be a glass diagram, may be a state model a state model, a class diagram or may be a call relation between modules and so on. So, these are the design models which are intermediate between a black box and white box do not really need to look at the code and neither is it a black box, we have knowledge which is intermediate between a black box and a white box and that is why it is called as a grey box.


And if it is a pure function that we are testing we need to test only black box and white box, but for glasses components and son we might have to do a grey box testing as well.

(Refer Slide Time: 05:38)

Page 3/3


Black-Box Testing

- Test cases are designed using only **functional specification** of the software:
 - Without any knowledge of the internal structure of the software.
- Black-box testing is also known as **functional testing**.



The diagram illustrates the black-box testing process. It shows a green cloud labeled 'Input' with a green arrow pointing to a blue box labeled 'Software'. From the 'Software' box, an orange arrow points to a pink cloud labeled 'Output'.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 257 | NPTEL



Now, first let us do black box unit testing, the black box unit testing is done based on the functional specification of the software or functional specification of the unit. So, given the unit and then we do not need to look at the code, we just see what is its input output behavior and based on that we design the test cases.

We should not look at the internal of the software only the input output behavior, the black box testing is also called as functional testing, because based on the knowledge of the functionality without looking at the code we test it so, it is also called as a functional testing.

(Refer Slide Time: 06:38)

The slide is titled "What is Hard about BB Testing" and contains the following content:

- Data domain is large
- A function may take multiple parameters:
 - We need to consider the combinations of the values of the different parameters.

Handwritten diagram: $f(p_1, p_2)$ where p_1 and p_2 are represented by boxes containing symbols.

Page 313

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 258

Speaker: A man in a blue shirt is visible in a small video window at the bottom right of the slide.

Now, let us try to appreciate why black box testing can be hard, here we just look at the input data and the output behavior that is what it should output and then design test cases.

The main complexity here is that the data domain for any typical software is large and therefore, we need to decide which test cases to apply because we need to have a optimal number of test cases and we have to select that out of the billions and trillions of test data that are possible. And not only that a function may take multiple parameters so, we need to fix the values of those parameters.

So, if a function takes just one parameter we need to just test with respect to that parameter, but if it takes 2 parameters and there are number of values that are possible for this parameter and for this parameter, this parameter 1 and parameter 2 for a function. Then we have to possibly test with various combinations of values may be fix this as one this and test with respect to this may be fix this and test with respect to this and so on. So, we need to try out various combinations and that makes testing much more challenging and complex.

(Refer Slide Time: 08:43)

The slide is titled "What's So Hard About Testing?". It contains the following content:

- Consider `int check-equal(int x, int y)`
- Assuming a 64 bit computer
 - Input space = 2^{128}
- Assuming it takes 10secs to key-in an integer pair:
 - It would take about a billion years to enter all possible values!
 - Automatic testing has its own problems!

Handwritten notes in blue ink show: $2 \times 2 = 2^{128}$, with "64" written above each "2".

The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES 259, and a small video inset of a man in a blue shirt.

First let us convince our self that, even a very small function just 2 line function can become extremely hard to test if we want to do the black box testing thoroughly that is a exhaustive black box testing with all possible input values. Let us consider a very trivial function name of the function is check equal, takes 2 integer parameters and then checks if these 2 parameters are the same.

Basically just one line are here, if x equal to y written 1 else written 0, but then we are not looking at the internals we are just doing a black box testing. So, what we will do is, we will check whether this function works well for all possible combinations of input data. Now let us say we are using a 64 bit computer, in a 64 bit computer each integer is represented using 64 bit and for simplicity.

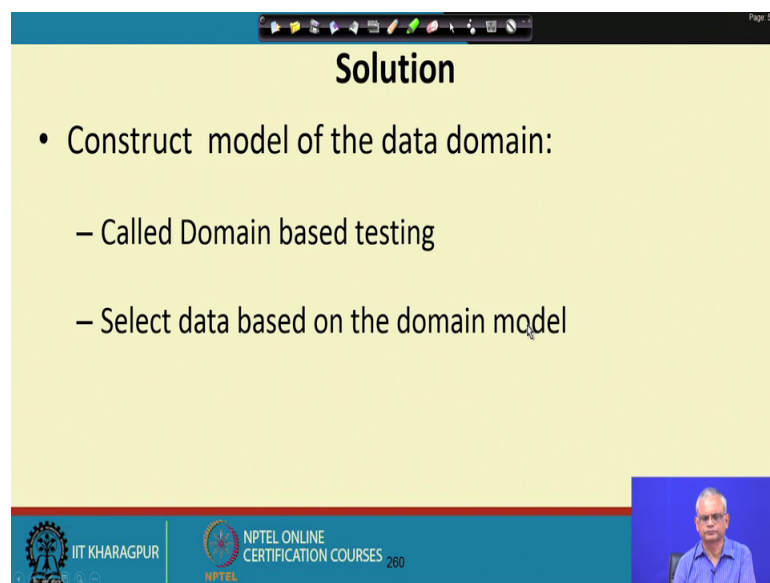
Let us assume because integer representation we do not want to go that level, but we just say that 64 bits are used and therefore, 2 to the power 64 inputs are possible for the first parameter. Similarly for the second parameter 2 to the power 64 different inputs are possible and therefore, the total number of inputs with which the function needs to be checked is all possible combinations of the first parameter and the second parameter which is 2 into 2 to the power 64 is equal to 2 to the power 128.

And this is a huge number we can read 2 to the power 10, 2 to the power 20, 2 to the power 30, but I cannot even read what is 2 to the power 128. It is extremely large 2 to the power 10 is kilo so, 20 is million, 30 is billion trillion and so, on. I do not even know

how to read it is extremely large number and let us assume that we are testing manually this 2^{128} possible values with which we need to test and let us say that we are expert in typing we can type very fast and just take 10 seconds to enter each test data.

And if we compute that the number of hours that will come here is very large and we can easily compute how many hours and then we might take a billion year to enter all possible values and that is just to test this small program. And therefore, the black box testing is hard if we consider all possible inputs. But our objective now is to find out black box test strategies which will drastically reduce the number of test cases may be 3 4 or something, but then they should be almost as effective as the exhaustive testing. So, that is our objective with that objective let us see what are the test strategies that we have.

(Refer Slide Time: 12:48)



Solution

- Construct model of the data domain:
 - Called Domain based testing
 - Select data based on the domain model

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 260

In all this test strategies that we discuss we will create a model of the data domain because we know the input data, the output data, we will create a data model and based on the data model we will design the test cases.

(Refer Slide Time: 13:20)

Black Box Testing


- Software considered as a black box:
 - Test data derived from the specification
 - No knowledge of code necessary
- Also known as:
 - Data-driven or
 - Input/output driven testing
- The goal is to achieve the thoroughness of exhaustive input testing:
 - With much less effort!!!!

Diagram: A blue box labeled "System" with an "Input" arrow pointing into it from the left and an "Output" arrow pointing out of it to the right.

Page 5/5

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES 262**

NPTEL



(Refer Slide Time: 13:24)


White-box Testing

- To design test cases:
 - Knowledge of internal structure of software necessary.
 - White-box testing is also called structural testing.

Page 5/5

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES 261**

NPTEL



On the other hand in the white box testing we know the internal structure that is the code we can check and therefore, it is also called as structural testing because we know the structure of the software.

Now, let us look at a black box testing before and after that we will look at white box testing, in black box testing we look at the system the black box and then we know the input output behavior, it is also called as data driven testing or input output testing we look at this first and we have already seen that exhaustive testing is very difficult. We

need to have some test strategies which are almost comparable with respect to the effectiveness or thoroughness of testing of the exhaustive testing, but the number of test cases should be much less and should take very less effort compared to exhaustive testing.

(Refer Slide Time: 14:34)

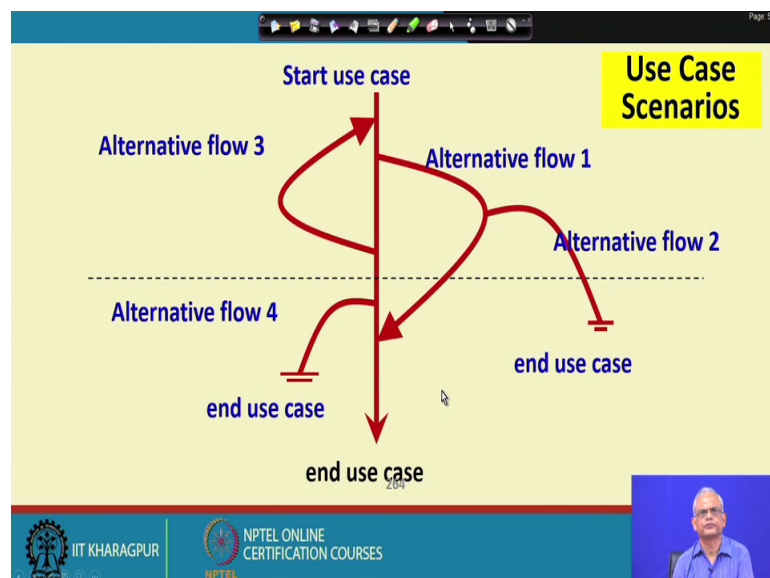
Black-Box Testing

- Scenario coverage
- Equivalence class partitioning
- Boundary value testing
- Cause-effect (Decision Table) testing
- Combinatorial testing
- Orthogonal array testing

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 263

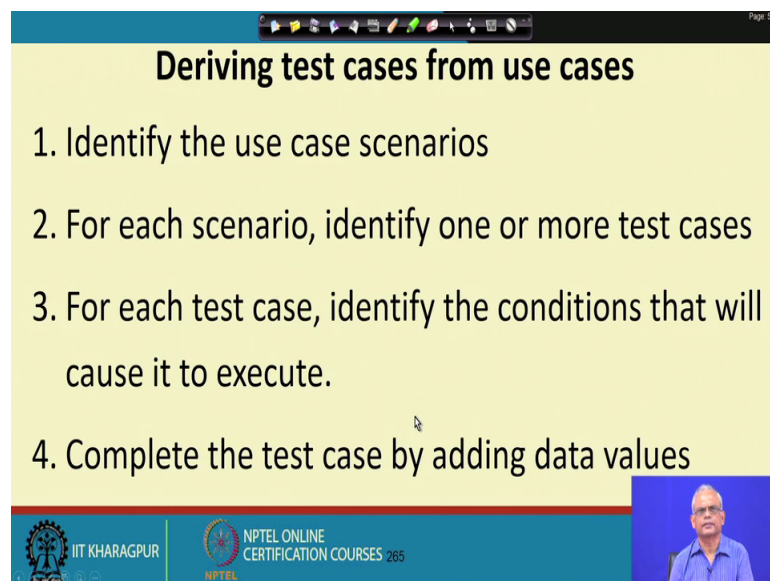
There are a large number of black box test strategies that have been reported we will look at only few of them, the scenario coverage, equivalence class partitioning, boundary value testing, cause effect testing, combinatorial and orthogonal array testing.

(Refer Slide Time: 14:58)



First let us look at the scenario based testing from our discussion on scenarios which we had during the object oriented discussion and object oriented design, we know that a use case consist of many scenarios one is called as the main scenario and then we have the alternate scenarios at some point in the main scenario the alternate flows occur. And these may actually end or they might just have a separate setup actions and then have similar action as the main line. Now if this is a kind of description of a use case with different scenarios then how do we design the scenario based testing.

(Refer Slide Time: 16:00)



Deriving test cases from use cases

1. Identify the use case scenarios
2. For each scenario, identify one or more test cases
3. For each test case, identify the conditions that will cause it to execute.
4. Complete the test case by adding data values



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 265

The first thing is to identify the use case scenarios and typically for requirements document is well written we will have identified the scenarios there in the form a text description. And for each scenario we will identify one or more test cases and these test cases are actually will execute the scenarios, and then we identify these are we identify the test data and then also identify the state at which the system should be there for the test data to be entered and that we say that identify the conditions that will cause this to execute and then complete the test cases by adding the data values.

(Refer Slide Time: 16:56)

Scenario number	Originating flow	Alternative flow	Next alternative	Next alternative
1	Basic flow			
2	Basic flow	Alt. flow 1		
3	Basic flow	Alt. flow 1	Alt. flow 2	
4	Basic flow	Alt. flow 3		
5	Basic flow	Alt. flow 3	Alt. flow 1	
6	Basic flow	Alt. flow 3	Alt. flow 1	Alt. flow 2
7	Basic flow	Alt. flow 4		
8	Basic flow	Alt. flow 3	Alt. flow 4	

Identify use case scenarios: Example

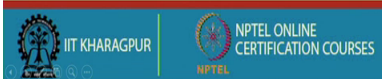

If we represent our scenarios in this form that first is basic flow, then basic flow with alternate flow, 1 alternate flow, 1 with alternate flow, flow 2 etcetera we develop the table and then each of this are a scenario and we need to have that many test cases.

(Refer Slide Time: 17:25)

- Parameters of any test case:
 - Conditions
 - Input (data values)
 - Expected result
 - Actual result

Identify the test cases

Test case ID	Scenario/condition	Data value 1	Data value 2	Data value N	Exp. results	Actual results
1	Scenario 1					
2	Scenario 2					
3	Scenario 3					

And for each scenario we need to identify the conditions or the state at which the input to be given the data input values there may be many inputs to be given the expected result for each of the input and the actual result that was observed.

Let us look at the next black box testing the scenario coverage was rather intuitive even straight forward just identify all the scenarios by looking at the SRS document and then we just list out all the scenarios and then the condition under which these will execute and then the test data and expected output. Now let us look at the equivalence class testing this is another black box testing.

(Refer Slide Time: 18:22)

The slide is titled "Equivalence Class Partitioning" and contains the following text:

- The input values to a program:
 - Partitioned into **equivalence classes**
- Partitioning is done such that:
 - **Program behaves in similar ways to every input value belonging to an equivalence class.**

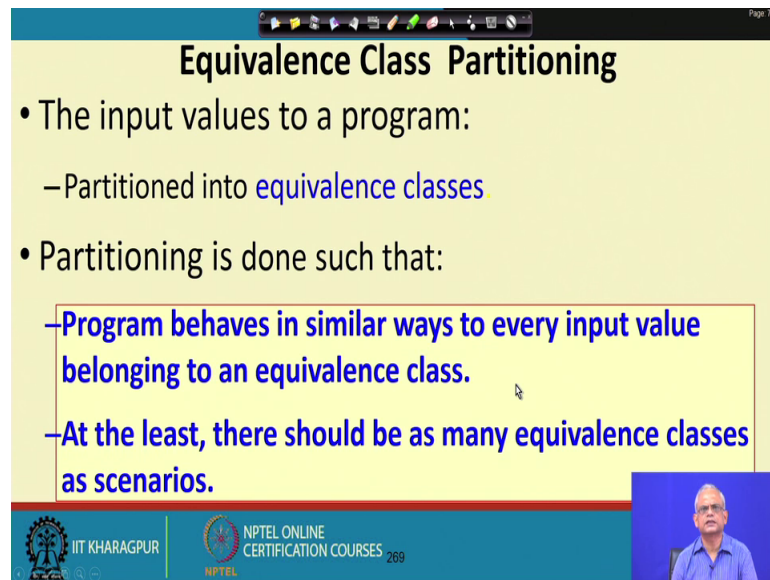
A hand-drawn diagram shows a rectangular box divided into four vertical columns labeled 1, 2, 3, and 4. Column 1 contains the symbols 'x', 'x', and 'x'. Below the entire box, the text "Data Set" is written.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and the number 269. A small video inset of a speaker is visible in the bottom right corner.

In this approach we look at the input data domain and then we partition it into equivalence class. The main idea behind partitioning the input data into equivalence class is that the program will behave similar way for every input belonging to an input equivalence class. So, if this is the set of input to the unit we identify equivalence classes in the input and the main idea here is that for this let us say equivalence class 1.

So, this is the set of data, data set or the data domain of the function and then we have partitioned it into equivalence classes 1 2 3 4 etcetera and for each equivalence class all input should execute the software in similar way, that is the same set of statements may get executed of course, here we do not have a knowledge of the internal of the software, but just by observing the behavior we can guess that these are executed in similar manner.

(Refer Slide Time: 20:12)



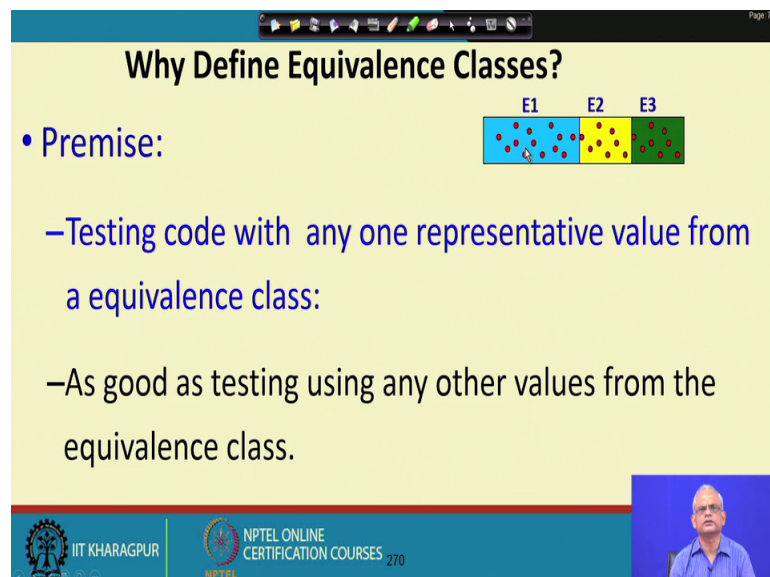
Equivalence Class Partitioning

- The input values to a program:
 - Partitioned into **equivalence classes**
- Partitioning is done such that:
 - **Program behaves in similar ways to every input value belonging to an equivalence class.**
 - **At the least, there should be as many equivalence classes as scenarios.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 269

And from each equivalence class we select one value with which to test and if the scenarios are extremely simple then each scenario need one equivalence class, but then it is much more complicated normally every scenario might need hundreds of equivalence classes or thousands of equivalence classes.

(Refer Slide Time: 20:49)



Why Define Equivalence Classes?

- **Premise:**
 - Testing code with **any one representative value from a equivalence class:**
 - As good as testing using any other values from the equivalence class.

E1 E2 E3

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 270

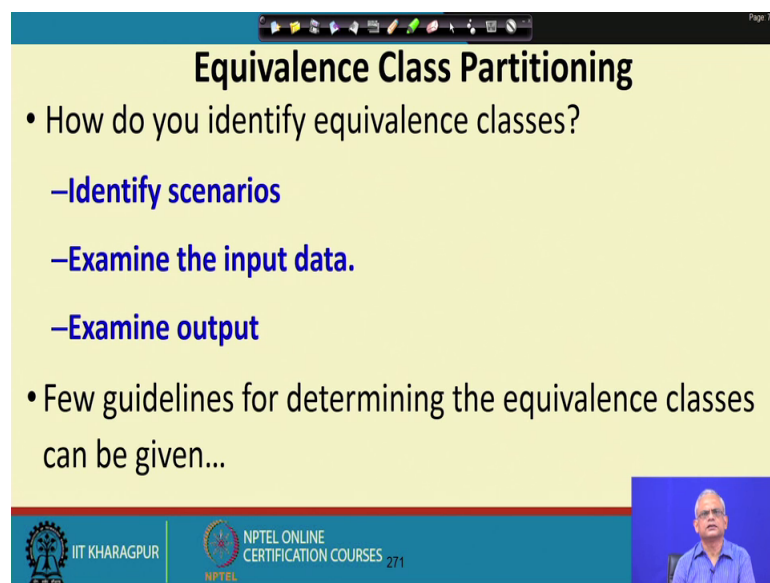
So, we need to pick value from each equivalence class, one of the main assumption in equivalence class is that we want to reduce the number of test cases with which the

system needs to be tested and if these are all the set of test cases that are possible in exhaustive testing we have partitioned it into equivalence classes E1, E2 and E3.

And the way we have partitioned into E1, E2 and E3 is that each element in equivalence class E1 they behave similarly the system execute similarly for each of the inputs. So, if we execute one of the data here then we have tested that behavior of the unit. So, our work now becomes to test with one representative value from each equivalence class and defining the equivalence class really helps.

Because testing using one value is equivalent to testing any other value in that equivalence class and each equivalence class, typically may contain thousands or millions of data points, we are assuming that executing using one of these points. Since the behavior is similar it is same as executing all of them and therefore, the equivalence class testing drastically reduces the number of test cases.

(Refer Slide Time: 22:33)



The slide is titled "Equivalence Class Partitioning" and contains the following content:

- How do you identify equivalence classes?
 - Identify scenarios
 - Examine the input data.
 - Examine output
- Few guidelines for determining the equivalence classes can be given...

The slide also features a small video inset of a presenter in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES 271 at the bottom.

But then even though it is a very good technique effective technique and reduces the number of test cases drastically, but the question that we are confronted is that how does one go about identifying the equivalence classes.

One is we need to identify the scenarios we need to examine the input data, we need to examine the output data, and then based on this we need to design the equivalence

classes, but then that is a bit of a weighed statement can we tell something more concurrent we will give that in the form of a few guidelines.

(Refer Slide Time: 23:20)

•If an input condition specifies a range, one valid and two invalid equivalence class are defined. E.g. 1 to 100

•If an input condition specifies a member of a set, one valid and one invalid equivalence classes are defined. E.g. {a,b,c}

•If an input condition is Boolean, one valid and one invalid classes are defined.

Example:

•Area code: range --- value defined between 10000 and 90000

•Password: set - six character string.

Guidelines to Identify Equivalence Classes

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 272 All Rights Reserved

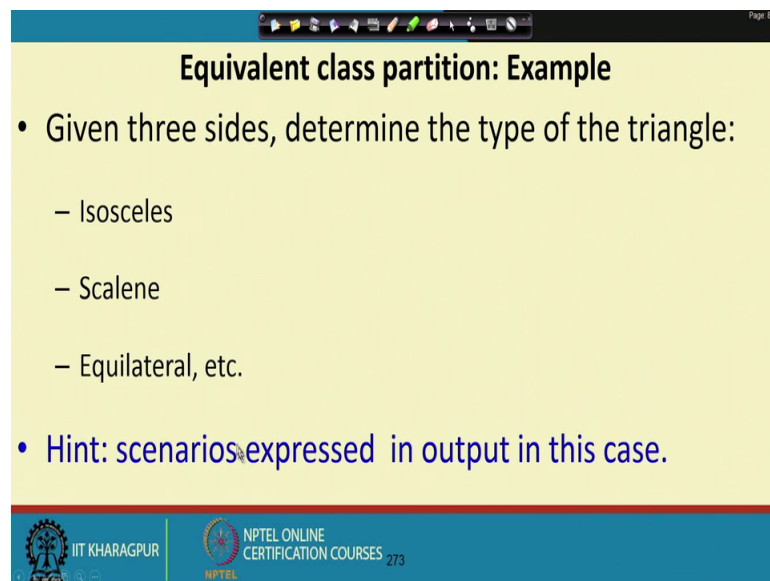
The guideline is that if the input data is a range we will have 2 immediate equivalence classes one is a valid and two invalid. So, the range is 1 to 100 anything between 1 to 100 is a valid data and anything more than 100 is invalid and anything less than 100 is invalid. So, we have one valid equivalence class and one invalid equivalence class which has contents values less than the minimum value that is acceptable and one which has values higher then highest value that is acceptable. So, there are two invalid equivalence classes and one valid equivalence class.

If the input is the set of data like let us say a b c these are all valid data the system will respond to this, but then we have invalid data which are not part of this set anything that is not part of this set is a invalid equivalence class. So, for a set of data we have two equivalence classes, one valid and one invalid, valid belongs to the set, invalid is anything other than that.

If the input condition is a Boolean again we have one valid and one invalid that is we give a Boolean value or do not give a Boolean value, just to give some example if we want to enter a area code let say the area code is between 10000 and 90000. So, the valid values are between these two and then anything more than 90000 are invalid, anything less than 10000 are invalid.

So, one valid and two invalid, what about a password which contains 6 character a string of 6 characters the set of 6 sorry the password containing 6 characters in a string, string of 6 characters we can consider it as a set of values. So, all those strings which contain 6 characters they are the set of values. So, we have one valid password which is any of these members of the set and invalid password which is not part of this may be contains 5 characters or something or may be a special characters.

(Refer Slide Time: 26:27)



The slide is titled "Equivalent class partition: Example" and contains the following content:

- Given three sides, determine the type of the triangle:
 - Isosceles
 - Scalene
 - Equilateral, etc.
- Hint: scenarios expressed in output in this case.

The slide footer includes the IIT KHARAGPUR logo and the text "NPTEL ONLINE CERTIFICATION COURSES 273".

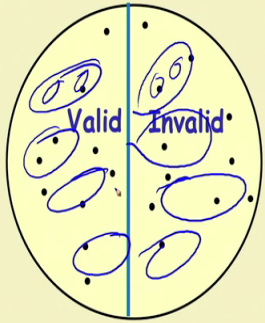
Now, let us look at an example how do we do equivalence class partition? Let us say we have given 3 sides of a triangle and we want to decide whether it is a Isosceles, Scalene, Equilateral etcetera. So, this is a simple function we take argument 3 sides of a triangle and displays what is the type of the triangle and we want to design equivalence class partition for this.

Looking at the output here gives us the hint about the equivalence classes. So, all the data sets for which it will produce isosceles is equivalence class, scalene is another equivalence class, equilateral another class.

(Refer Slide Time: 27:31)

Equivalence Partitioning

- First-level partitioning:
 - Valid vs. Invalid test cases



IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 274

So, here the equivalence classes are easily identifiable from the output, but for any input data we have always 2 equivalence classes one is valid and one invalid and there may be several invalid. So, to start with there is one valid equivalence class and another invalid equivalence class and we might have several invalid equivalence classes here which we might later identify and also there may be several valid equivalence classes out of this.

So, given a problem to design the equivalence tests we first define 2 equivalence classes the valid and invalid and then look at further division the valid into equivalence classes set of equivalence classes and each of this may again contain equivalence classes we will just look at that in the next lecture right now we are we must at the end of this lecture we will stop here.

Thank you.