

Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 32
Input Space Partitioning

Hello again. Welcome to the third lecture of week 3. What did we do last time? If you remember I had introduced you to functional testing, how is functional testing done popularly? And we looked at various techniques of functional testing, an overview of functional testing. In that one of the testing techniques was equivalence class partitioning.

Today I had like to spend some time on that. We call equivalence class partitioning as input space partitioning because it involves partitioning the input domain. So, it is practically the same as equivalence class partitioning. So, I will introduce you to that domain of equivalence class partitioning or input space partitioning in detail today.

(Refer Slide Time: 00:49)

Partitions of a set

- Given a set S , a **partition** of S is a set $\{S_1, S_2, \dots, S_n\}$ of subsets of S such that
 - The subsets S_i of S are **pair-wise disjoint**, i.e., $S_i \cap S_j = \emptyset$.
 - The union of the subsets S_i is the entire set S , i.e., $\cup_i S_i = S$.

S

The diagram shows a large rectangle labeled S at the top. Inside this rectangle, four smaller regions are defined by curved lines, labeled S_1 , S_2 , S_3 , and S_4 . These regions are mutually disjoint and their union covers the entire area of the rectangle S . A small mouse cursor is visible near the bottom left of the rectangle. In the bottom right corner of the slide, there is a small video inset showing a woman speaking.

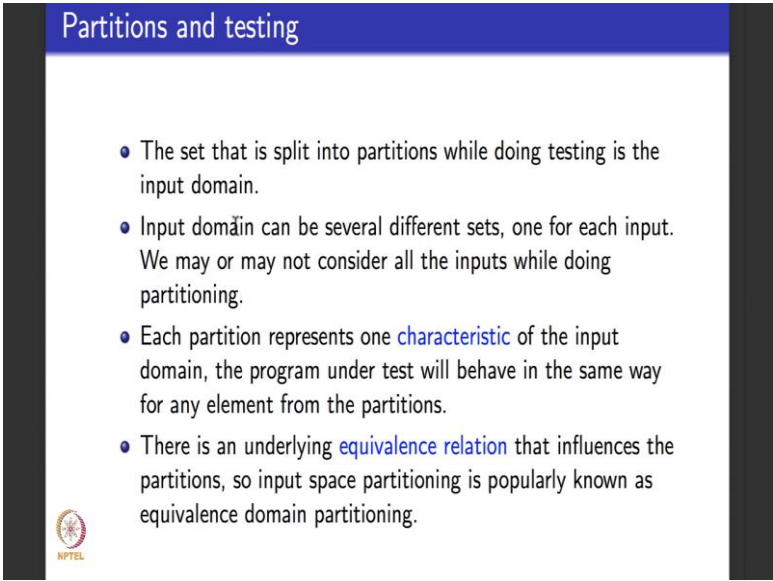
So, we begin by understanding what are partitions of a set. So, I am given a set S , set S could be a finite set, an infinite set, a countable finite set, we really do not worry about what kind of a set it is. I want to understand what the partitions of the set are. So, partition of a set is a collection of subsets of the set S . Here it is a collection of n subsets such that each subset is disjoint pair-wise with the other one that is they are mutually

disjoint. S_i intersection S_j is empty, they do not have any elements in common and put together the union of all the subsets you get back the entire set. So, that intuitively explains the word partition also.

Partition means I take a set S divide it into various subsets, the various subsets that I divide it into should satisfy a few properties. The number of subsets that I divide it into should be finite, in this case it is n . Each of these subsets should not have anything in common with the any other subset that is they should be pair wise disjoint and put together all the subsets together constitute the entire set S . Nothing is left out after partitioning the set S .

So, in this small figure here, the set S is partitioned into 4 partitions: S_1 , S_2 , S_3 and S_4 . If you see put together these 4 partitions constitute the entire set and they are pair wise disjoint. So, this is small example will illustrate how partitions work.

(Refer Slide Time: 02:21)



The slide has a blue header with the text "Partitions and testing". Below the header, there are four bullet points. The first bullet point states that the set split into partitions is the input domain. The second bullet point states that the input domain can be several different sets, one for each input, and that we may or may not consider all inputs while doing partitioning. The third bullet point states that each partition represents one characteristic of the input domain, and that the program under test will behave in the same way for any element from the partitions. The fourth bullet point states that there is an underlying equivalence relation that influences the partitions, and that input space partitioning is popularly known as equivalence domain partitioning. In the bottom left corner of the slide, there is a small circular logo with the text "NPTEL" below it.

- The set that is split into partitions while doing testing is the input domain.
- Input domain can be several different sets, one for each input. We may or may not consider all the inputs while doing partitioning.
- Each partition represents one characteristic of the input domain, the program under test will behave in the same way for any element from the partitions.
- There is an underlying equivalence relation that influences the partitions, so input space partitioning is popularly known as equivalence domain partitioning.

Now, how are we going to use partitions as far as testing is concerned. So, the set, this set S that I defined for a partition of S the set that we are going to partition is what is the input domain. So, take a program, the program will have several inputs all the inputs could be a various types. The input domain as we discussed in the last lecture constitutes all the types of the inputs that the program has along with the domain types as declared within the program. It can be thought of as a Cartesian product or a cross product of all the input types. For example, if a program has a let us say two Boolean inputs and let us

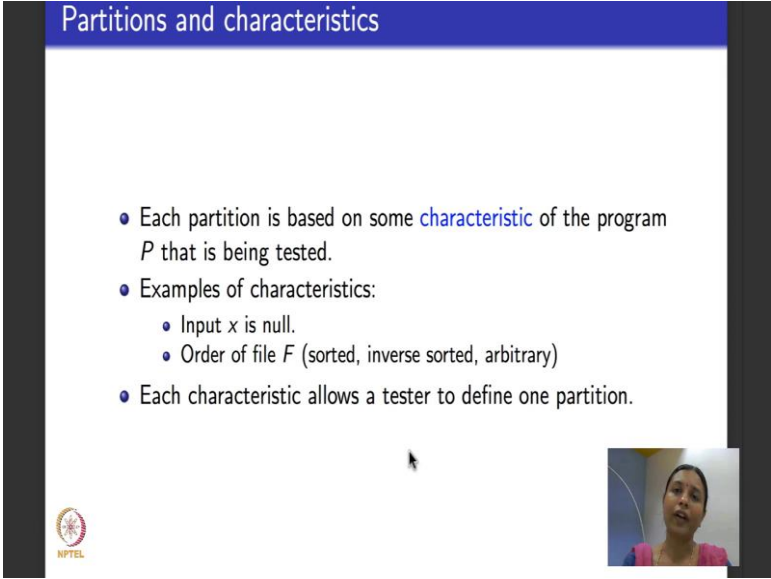
say one integer input, then the input domain will be the set $\{0, 1\}$ for one Boolean input cross product-ed at with $\{0, 1\}$ for the second Boolean input, cross product-ed with the domain of integers based on the kind of computer the program is going to run on.

So, that is the set that we are going to take and partition. Input domain as I told you can be several different sets typically one for each input, while doing partitioning we may or we may not consider all the inputs. Testers typically sometimes decide to focus only on certain inputs do not consider all the inputs for partitioning, but when they are doing let say integration level testing, they might consider all the inputs for partitioning.

So, it widely varies based on what is being tested and the testers choice. It is believed that each partition of an input domain represents one property or in the testing jargon we call it characteristic. It represents one characteristic of the input domain and the program that is being tested will behave in exactly identical way for any input value from each partition. So, I take as input domain, partition it into several partitions and pick one value to define my test case from each partition, and the belief is the program behavior in terms of the expected out to the program is expected to produce, will be exactly the same if I pickup any input from each partition. But inputs across partitions, the program is expected to behave differently because each partition represents a different characteristic.

And there is an underlying equivalence relation. Equivalence relations are binary relations that satisfy certain properties. We are not interested in defining or understanding what they are, but equivalence here means that the behaviour of the program is identical for any choice of input from a given partition. So, we say that the set of partitions are equivalence classes induced by an equivalence relation.

(Refer Slide Time: 04:53)



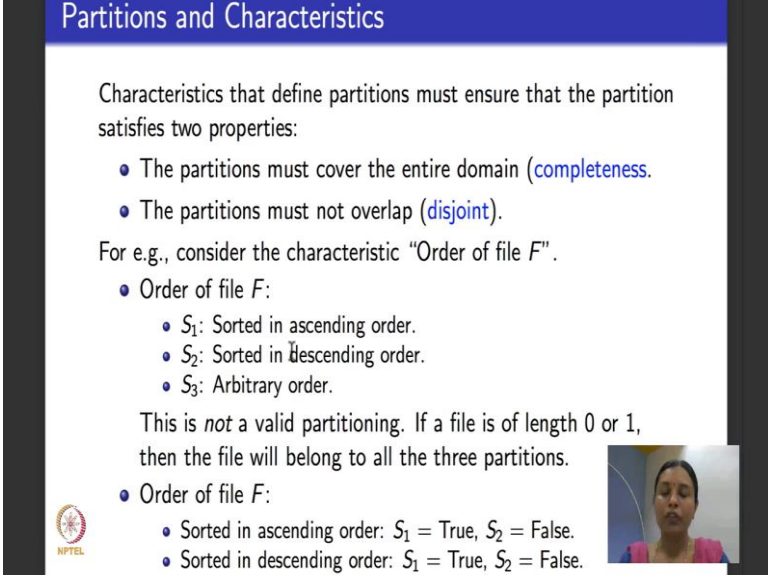
The slide has a blue header with the text "Partitions and characteristics". Below the header, there is a bulleted list of points. In the bottom right corner, there is a small video inset showing a woman speaking. In the bottom left corner, there is a small logo for NPTEL.

- Each partition is based on some **characteristic** of the program P that is being tested.
- Examples of characteristics:
 - Input x is null.
 - Order of file F (sorted, inverse sorted, arbitrary)
- Each characteristic allows a tester to define one partition.

So, how do we define partitions and characteristics? Each partition as I told you is based on a characteristic or some property of input domains. So, here are some example properties or characteristics. You could be worried about a property which asks whether the input is null or not, you could be worried, let us say the input is a file, you would be worried about whether the file is in sorted order or not and if it is in sorted order, what is the order of sorting, is the file in sorted order is it in inverse sorted order or it is an arbitrary order, it is not sorted at all.

So, each characteristic allows a tester to define one partition. So will see a few examples that explain how to define additional characteristics.

(Refer Slide Time: 05:29)



Partitions and Characteristics

Characteristics that define partitions must ensure that the partition satisfies two properties:



- The partitions must cover the entire domain (**completeness**).
- The partitions must not overlap (**disjoint**).

For e.g., consider the characteristic "Order of file F ".

- Order of file F :
 - S_1 : Sorted in ascending order.
 - S_2 : Sorted in descending order.
 - S_3 : Arbitrary order.

This is *not* a valid partitioning. If a file is of length 0 or 1, then the file will belong to all the three partitions.

- Order of file F :
 - Sorted in ascending order: $S_1 = \text{True}$, $S_2 = \text{False}$.
 - Sorted in descending order: $S_1 = \text{True}$, $S_2 = \text{False}$.

So, what are we looking at, we are looking at two properties as I told you if you remember in the first slide I told you what do partitions satisfy? They have to be pairwise disjoint and put together they have to constitute the entire set. So, when we look at characteristic, we are looking at characteristics, the partitions induced by the characteristics must cover the entire domain, put together they must cover the entire input domain. In testing terminology we call it the partitions are complete or completeness property and the disjointness should still hold, pairwise partitions must never overlap, every pair of partitions should be disjoint with the other one.

So, for example, if I consider the characteristic that I showed you in this slide the order of a file: is it sorted is it inverse sorted or is it arbitrary. Here is one example of an attempt to define a characteristic and go ahead and look at partitions based on that characteristic. So, I could say, here are the 3 direct partitions, first partition could ask whether the file is sorted in ascending order, second partition could be all files that are sorted in inverse ascending or descending order, third partition could be all files that are not sorted that is they are in arbitrary order.

But remember if this is the partitioning and we want to go ahead and check whether these partitioning satisfy these properties or not. If the partitions do not satisfy the properties of completeness and disjoint we do not work with them for testing at all, because testing without these two properties can be considered to be have errors, it could

have lot of issues and problems. So, partitions have to be complete, they have to be disjoint.

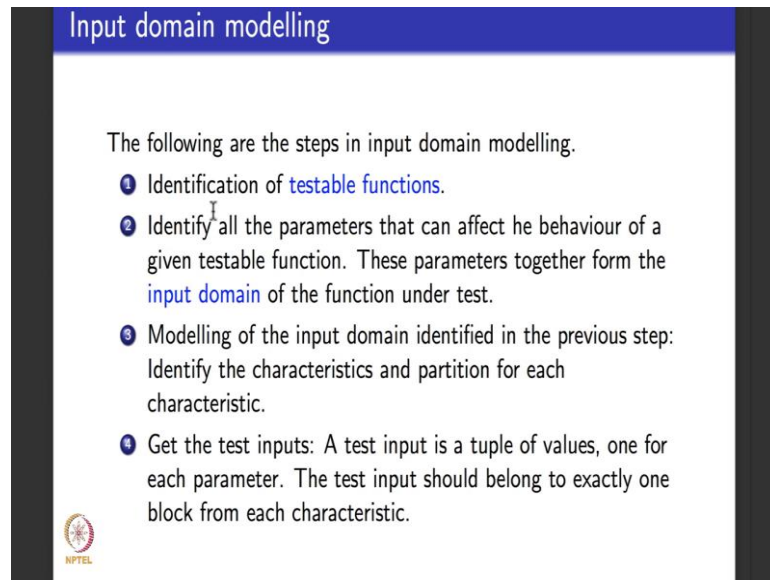
So, here for this example partition, is it complete is it just joint? The problem is it is not disjoint. Why is it not disjoint? It is not disjoint because suppose I consider small files that is files that have length 0 or length 1, then the file will be sorted ascending, the file will be sorted descending and the file will can be thought of as not being sorted at all. So, such files that are of length 0 or length 1 will belong to all the 3 partitions, which means the disjointedness property is not satisfied.

So, what do we do? What we usually do is we consider the same characteristic, but not in exactly this way. We tweak it a little bit and write it is likely differently. So, we write the characteristic of the order of the file as follows. We ask a question for the characteristics. So, we ask is the file sorted in descending order or is the file sorted in ascending order? The answer could be yes in which case it is true and the answer could be no in which case it is false.

So, if I write the same thing like this, you will observe that the disjointedness problem goes away. How does it go away? Take a file of length 0 or length 1. So, it will be sorted in ascending order. So, it belongs to the true part of this partition, it will be sorted in descending order, so, it will belong to the true part of the second partition. But the first characteristic which is been sorted in ascending order induces two partitions: is it true is it false and a file of length 0 or 1 belongs to exactly one of these, and the second characteristic which is the characteristic of being sorted in descending order induces two more partitions: true and false and a file of length 0 or 1 again belongs to exactly one of these partitions.

So, we basically consider partitions the way intuition tells us by looking at properties by looking at specifications, but what is what I am trying to tell you through this slide is that you should be a little careful in way you write it. If you write it directly like this sorted ascending, sorted descending, sort it arbitrary then the partitions may not be valid, but if you consider each one individually and consider it is being true or false and being true or false then the partitions satisfy the disjointedness and completeness property. So, they are valid partitions.


(Refer Slide Time: 09:45)



Input domain modelling

The following are the steps in input domain modelling.

- 1 Identification of **testable functions**.
- 2 Identify all the parameters that can affect the behaviour of a given testable function. These parameters together form the **input domain** of the function under test.
- 3 Modelling of the input domain identified in the previous step: Identify the characteristics and partition for each characteristic.
- 4 Get the test inputs: A test input is a tuple of values, one for each parameter. The test input should belong to exactly one block from each characteristic.



Will move on to considering the input domain and how to model the input domain. What do we mean by modelling the input domain? Modelling the input domain means that I have to be able to understand which are the sets that constitute the input domain for my consideration to be testing now. And how do I go ahead and partition, identify characteristics for these sets and partition them based on these characteristics. Typically what do we do? Here are the steps that we undertake.

The first step that we do is we identify testable functions. When is a function testable? If you remember in the last lecture I had shown you an example of knowing the context right there was a large program p , that was calling a particular function right. So, that function inside the program p was the function under test when we were doing input space partitioning or equivalence partitioning for that functions. So, in general program could have several different testable functions.

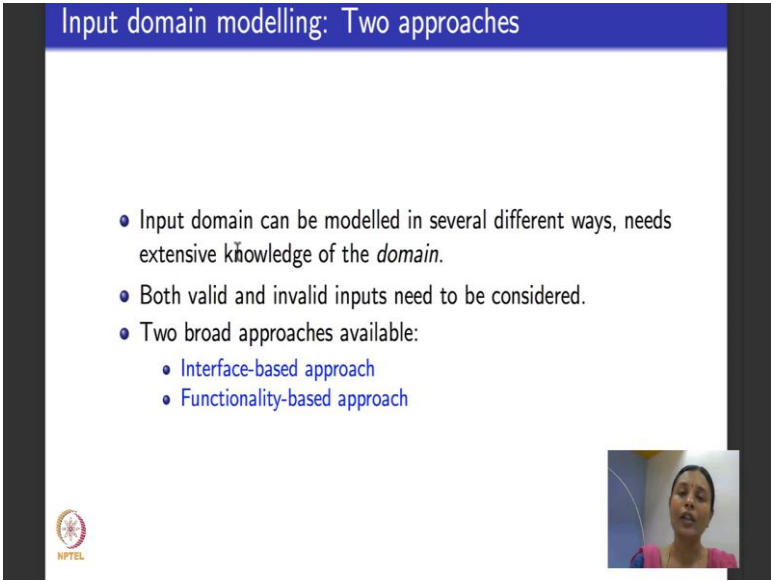
So, we now focus on which are that functions that we want to begin testing on. So, that process is what is called identifying testable functions. So, I begin by identifying which are the functions I want to test. So, those are what are called testable functions. The second step is to identify all the parameters that can affect the behaviour of a testable function. What like for example, if you remember in the previous lecture that particular function f was called only when x was greater than 20 and that property was very important for that. I called it as test in the function in context, that is what this step is

talking about. Once you identify all the parameters these parameters put together constitute the input domain of the function that is being tested.

Now, once I have the input domain I have to be able to model it. Now what do I do here? I look at the properties of the function under test. what do I want to test about this function that is under test? What are the requirements that this function is suppose to satisfy, what is it is functionality? I look at all that, that will help me to identify characteristics, I take the characteristics and partition based on each characteristic. And what do I do for test inputs? I take one partition I told you any set of input values representing that partition is good enough as a test input and each test input should belong to exactly one block from each characteristic.

And please remember when I choose characteristics and partition based on the characteristics, partitions must satisfy the property of being disjoint and complete.

(Refer Slide Time: 12:19)



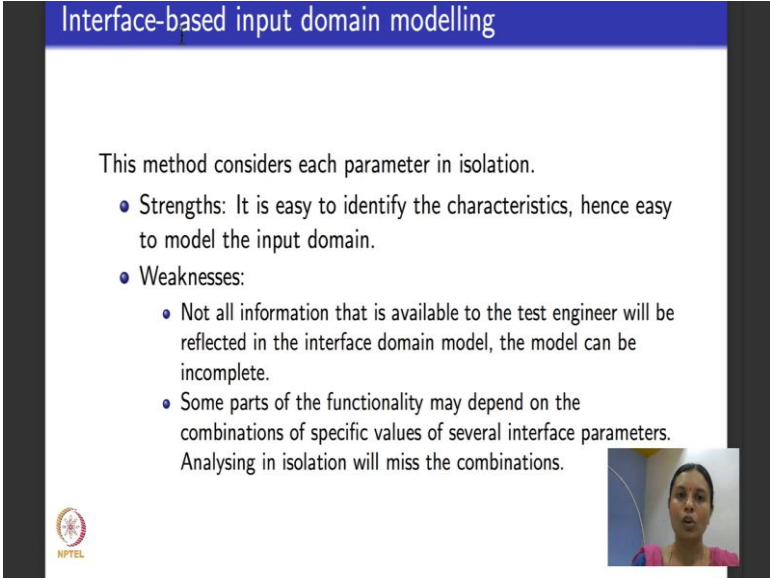
The slide is titled "Input domain modelling: Two approaches" in a blue header. It contains a bulleted list of points. In the bottom right corner, there is a small video inset showing a woman speaking. In the bottom left corner, there is a small circular logo with the text "NPTEL" below it.

- Input domain can be modelled in several different ways, needs extensive knowledge of the *domain*.
- Both valid and invalid inputs need to be considered.
- Two broad approaches available:
 - Interface-based approach
 - Functionality-based approach

So, what are the two approaches of input domain modeling? There are two popular approaches to do input domain modelling, one is called interface based approach and the other one is called functionality based approach. So, we will spend a few minutes understanding what each of these approaches are and two things to be observed, when I modelled the input domain I need to know the domain. What do I mean to mean by need to know the domain? I need to know what the program is going to do.

Let us say the program is a program that it is going to fly an aircraft I need to know a little bit about the application domain of the program. Let us say the program is programs that suppose to be in a banking sector then I need to know a little bit about the application domain. So, I need domain knowledge and my partition should consider both valid and invalid inputs because I am going to test the functionalities.

(Refer Slide Time: 13:11)



The slide is titled "Interface-based input domain modelling" in a blue header. The main content area is white and contains the following text and list:

This method considers each parameter in isolation.

- Strengths: It is easy to identify the characteristics, hence easy to model the input domain.
- Weaknesses:
 - Not all information that is available to the test engineer will be reflected in the interface domain model, the model can be incomplete.
 - Some parts of the functionality may depend on the combinations of specific values of several interface parameters. Analysing in isolation will miss the combinations.

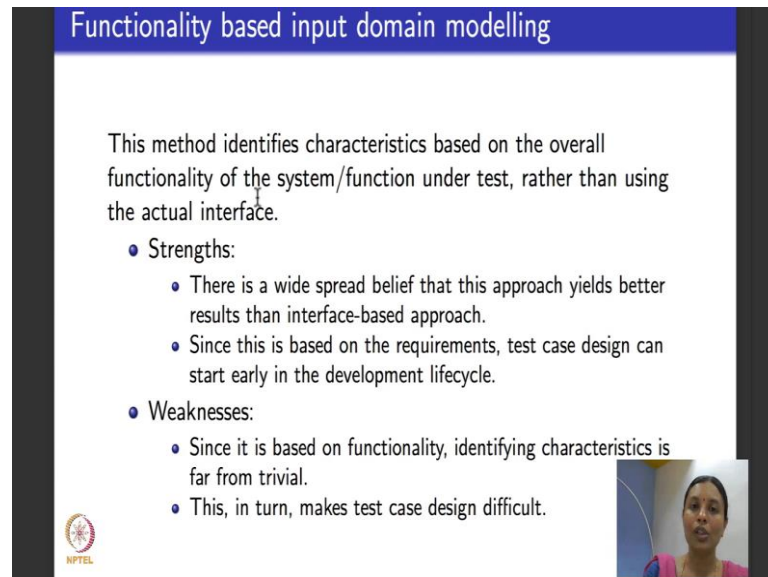
In the bottom left corner, there is a small circular logo with the text "NPTEL" below it. In the bottom right corner, there is a small video inset showing a woman with dark hair, wearing a pink top, speaking.

So, what we spend some time on now is, what are these two approaches to model input domain modelling? So, the first approach is what is called interface based input domain modelling. So, here what it does is that there are several characteristics of an input domain, they could be several parameters induced for each of these characteristics. Interface based approach considers each characteristic of parameter in isolation.

What are the possible strengths of interface based approach? Interface space approach is suppose to make it easier to identify the characteristics because we really do not worry about how do they influence each other, how one influences the other and what are the weaknesses? The weaknesses are, the problem is typically not all information is available to the test engineer that will be reflected correctly in the interface domain model when we consider each characteristic separately. So, there could be a risk of the input domain models that I consider. By models for input domain I mean sets, the sets that I consider for input domain could be incomplete and because I do not consider how they depend on each other, I might miss some functionality.

What I am trying to do in interface based approach is that I am trying to analyze each characteristic in isolation that might miss some functionalities that comes as a part of combination.



(Refer Slide Time: 14:32)



Functionality based input domain modelling

This method identifies characteristics based on the overall functionality of the system/function under test, rather than using the actual interface.

- **Strengths:**
 - There is a wide spread belief that this approach yields better results than interface-based approach.
 - Since this is based on the requirements, test case design can start early in the development lifecycle.
- **Weaknesses:**
 - Since it is based on functionality, identifying characteristics is far from trivial.
 - This, in turn, makes test case design difficult.

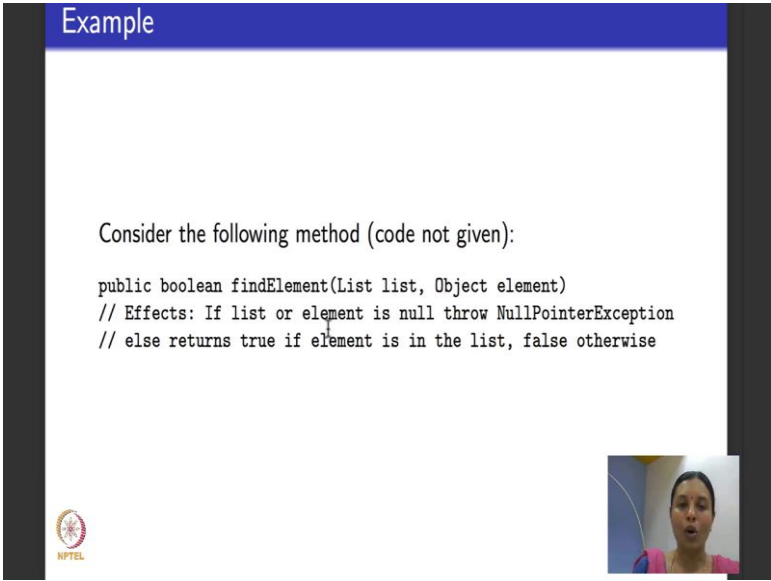
So, we will see examples that will help us to understand both the strengths and the weaknesses in detail. The next approach is functionality based input domain modelling. In this method characteristics are identified based on the overall functionality of a system. So, let us say I have a program under test or a function under test, I look at what that program or the function is supposed to do, what is its main functionality, why is it written, why does it exist, I look at those characteristics and then consider partitions of the input domain based on characteristics that represent the functionality of the considered program or function.

The strengths are that because I consider the main functionality this approach is believed to yield better results than interface based approach. In fact, there are empirical studies, I will give you some references about empirical studies in input domain partitioning, where this is validated and since it is directly based on functionality or requirements test case design can start very early in the development life cycle. Remember functionality caters to requirements and requirements are the first thing that are baselined in a typical development of software.

So, I test engineer can start designing test cases write the requirements face without waiting for the program to be ready. Of course, for execute in this test cases you need the program to be ready, but for designing the test cases I can start up front. What are the weaknesses? The weaknesses are because it is directly based on functionality you need domain knowledge, and identifying, partitioning the input domain modelled can be non-trivial.

So, this is a small example.



(Refer Slide Time: 16:01)



Example

Consider the following method (code not given):

```
public boolean findElement(List list, Object element)
// Effects: If list or element is null throw NullPointerException
// else returns true if element is in the list, false otherwise
```

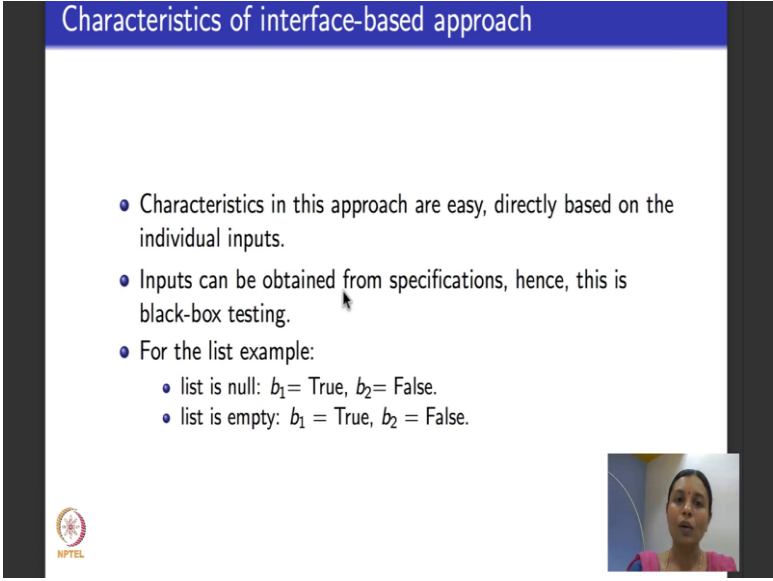


This is an example of a code which is supposed to take a list as an input along with an element. If you remember we had looked at this code once before in one of the earlier weeks and it supposed to find out whether this element is present in the list or not. If the element is present in the list, it will return true that is why it suppose to return a Boolean value. If it is not present in the list it will return false. If the list itself or the element is null then will throw in exception null pointer exception.

Please remember I have not given you the code here. Why have we not given us the full code? We were not given the full code because as I told you in the last lecture we are, our focus is black box testing. In black box testing I do not design test cases based on the code. All the information that I need is which is the program you want me to test, this is the program of find element. What are it is inputs, it is inputs are list and the element to be found in the list. What is this program supposed to return?

It supposed to return a Boolean value true or false and when there are problems it suppose to return in null pointer exception. This is all the information that I need to know. I do not have to know the code for the program because I am doing black box testing. With this information can I use inputs base partitioning to design test cases for the program that is what we going to look at.

(Refer Slide Time: 17:22)



The slide is titled "Characteristics of interface-based approach" in a blue header. It contains three main bullet points. The first states that characteristics are easy and based on individual inputs. The second states that inputs are obtained from specifications, making it black-box testing. The third point, "For the list example:", has two sub-bullets: "list is null: $b_1 = \text{True}, b_2 = \text{False}$." and "list is empty: $b_1 = \text{True}, b_2 = \text{False}$." In the bottom right corner, there is a small video inset showing a woman speaking. The NPTEL logo is in the bottom left corner.

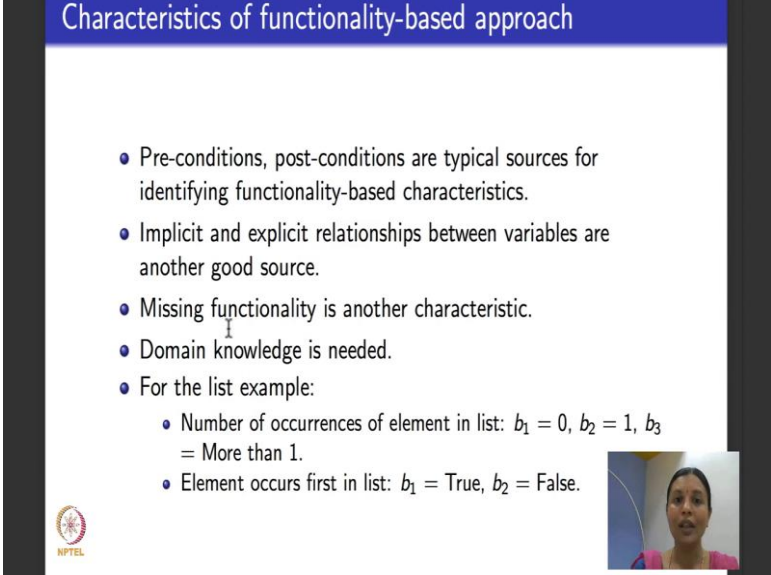
- Characteristics in this approach are easy, directly based on the individual inputs.
- Inputs can be obtained from specifications, hence, this is black-box testing.
- For the list example:
 - list is null: $b_1 = \text{True}, b_2 = \text{False}$.
 - list is empty: $b_1 = \text{True}, b_2 = \text{False}$.

Now, let us look at the characteristics for interface based approach for this program. What do I know? I know that this list, two inputs list and an element two possible outputs, true if the element is in the list, false if the element is not found in the list and an exception. So, simple interface based input domain partitioning will look like this. Is the list null, it will ask and the answer will be true if it is null, false if it is not null and then is the list empty, it will ask answer will be true, answer will be false. You could go ahead and write a few more things: is the list, is the element present in the list, but if when the moment when we talk about is the element present in the list, we are not considering interface based approach. Please remember that interface based approach considers each parameter, input parameter in isolation. For this program there are two inputs list and an element.

So, interface based approach directly considers the list and considers partitions of the kind of list only. It does not ask question about whether the second element input which the element, is it there in the list or not, does not ask that question. When we do

functionality based partitioning, we will consider that property into consideration that is what we do here.

(Refer Slide Time: 18:41)



The slide is titled "Characteristics of functionality-based approach" in a blue header. It contains a bulleted list of characteristics. The first four are: "Pre-conditions, post-conditions are typical sources for identifying functionality-based characteristics.", "Implicit and explicit relationships between variables are another good source.", "Missing functionality is another characteristic.", and "Domain knowledge is needed." The fifth bullet is "For the list example:", which has two sub-bullets. The first sub-bullet is "Number of occurrences of element in list: $b_1 = 0, b_2 = 1, b_3 = \text{More than 1.}$ ". The second sub-bullet is "Element occurs first in list: $b_1 = \text{True}, b_2 = \text{False.}$ ". In the bottom right corner of the slide, there is a small video inset showing a woman with dark hair wearing a pink top, speaking. The NPTEL logo is in the bottom left corner of the slide.

- Pre-conditions, post-conditions are typical sources for identifying functionality-based characteristics.
- Implicit and explicit relationships between variables are another good source.
- Missing functionality is another characteristic.
- Domain knowledge is needed.
- For the list example:
 - Number of occurrences of element in list: $b_1 = 0, b_2 = 1, b_3 = \text{More than 1.}$
 - Element occurs first in list: $b_1 = \text{True}, b_2 = \text{False.}$

So, when I do functionality based approach, for that list example, if you look at the bottom of this slide, we say is the element present in the list, how many times is it present in the list? The number of occurrences of element in the list, if it is 0 which means the element is not present in the list, if it is one then it is present in the list exactly once, if it is more than one then it is present in the list more than once. So, one characteristic based on which I derive the partitions is the number of occurrences of elements in the list: it does not occur which is the first one, it occurs exactly ones which is the second one, it occurs more than ones which is the third one.

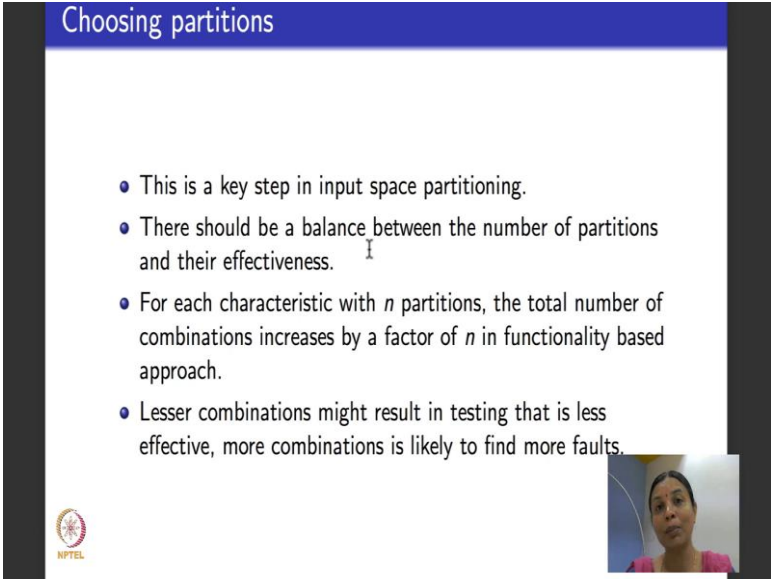
Now, if you see both the inputs to the program, the element and the list, are together considered in functionality based approach. When we did interface based approach, we considered only the list separately. So, another characteristic that we could arrive, this is just as an example for functionality based approach, is the following. We can ask does the element occur first in the list. Not a very interesting characteristic because the program is not worried about asking whether the element is occurring first in the list or not, but let us say I am interested in this characteristic for some weird reason. Then I can again write it I saying if it does occur first in the list, then one characteristic says true if it does not occur first in the list the next characteristics is false.

So, how do I do functionality based approach, where do I get these things from? Usually in a program preconditions and post conditions they are very rich sources to understand functionality. For example, if you look here right it is just a small little comment that acts as preconditions on what the program is expected to do.

So, I do not really need the code of the program. Just by looking at this I am able to arrive at characteristic right. The other things that people usually look, for testers usually look for, while writing characteristics, is to look at implicit and explicit relationships between variables. I will show you an example later in the lecture today about what this means. The other functionality people look for is typically what is missing functionality. So, that they can identify if the program is correctly handling missing functionality, and just to stress it again, to do all this unit extensive domain knowledge. Domain knowledge I mean the domain of this in which the software is running.

So, in the next step is choosing partitions. How do I choose the partitions? As I told you the number of partitions should not be too high. If you choose a very fine grain characteristic

(Refer Slide Time: 21:16)



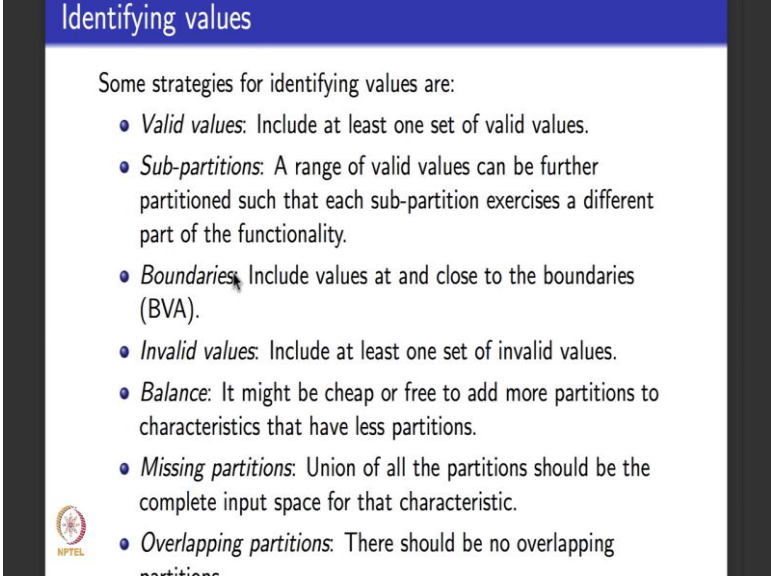
The slide is titled "Choosing partitions" in a blue header. It contains a bulleted list of four points. In the bottom right corner, there is a small video inset showing a woman speaking. In the bottom left corner, there is a small NPTEL logo.

- This is a key step in input space partitioning.
- There should be a balance between the number of partitions and their effectiveness.
- For each characteristic with n partitions, the total number of combinations increases by a factor of n in functionality based approach.
- Lesser combinations might result in testing that is less effective, more combinations is likely to find more faults.

then you will end up in too many partitions and then if you choose one test input for each partition there will be too many test cases. Sometimes it might be needed to get clarity if error is very deep in the program to get focus on the error, but in general it involves an

experienced balance between choosing the partitions wisely such that they are not too large in number, but at the same time very effective in finding faults.

(Refer Slide Time: 21:42)



The slide is titled "Identifying values" in a blue header. Below the header, it says "Some strategies for identifying values are:" followed by a bulleted list of seven strategies. The strategies are: Valid values, Sub-partitions, Boundaries, Invalid values, Balance, Missing partitions, and Overlapping partitions. Each strategy is described in a sentence. There is a small NPTEL logo in the bottom left corner of the slide content area.

Identifying values

Some strategies for identifying values are:

- *Valid values*: Include at least one set of valid values.
- *Sub-partitions*: A range of valid values can be further partitioned such that each sub-partition exercises a different part of the functionality.
- *Boundaries*: Include values at and close to the boundaries (BVA).
- *Invalid values*: Include at least one set of invalid values.
- *Balance*: It might be cheap or free to add more partitions to characteristics that have less partitions.
- *Missing partitions*: Union of all the partitions should be the complete input space for that characteristic.
- *Overlapping partitions*: There should be no overlapping partitions.

How do I identify values within partitions? Typically partition should be such that all these things are used to find the test input values. I should find a few valid values, means there should be in that list example, there should be a nice long list and a case of element being present in the list at least once, and sometimes I might have to consider one partition in isolation and partition it further. There could be needs for that. Sometimes we have to identify values exactly at boundary.

If you remember in the last lecture I told you that once I do equivalence class partitioning I can also do boundary value analysis that is what this is about, and the some partitions have to give me invalid values, because I need to know whether the program has exception handling features for all these things and as I told you there must be a balance between the number of partitions and effectiveness in finding test cases. And there should not be any missing partitions, I should get back the entire input domain that is very important, it should be complete.

And as I told you there should not be any overlapping partitions, the partitions should be disjoint.

(Refer Slide Time: 22:49)

TriTyp Example

- We re-visit the program TriTyp that determines the type of a triangle, given three sides as input.
- Interface-based partitions of inputs:

Partition	b_1	b_2	b_3
q_1 ="Relation of Side 1 to 0"	> 0	$= 0$	< 0
q_2 ="Relation of Side 2 to 0"	> 0	$= 0$	< 0
q_3 ="Relation of Side 3 to 0"	> 0	$= 0$	< 0
- Consider the partition q_1 for Side 1. If one value is chosen from each partition, the result is three tests. Choose test inputs as Side1=7 in the first one, Side1=0 in the second test and Side1=-3 in the third test.
- Some partitions represent valid values, some represent invalid values.

So, what will do for the rest of today's lecture is we will take one example and we will go through what are the various ways in which we can partition the input domain for that example. So the example that I have chosen is the example that we did when we did logic coverage criteria, because that was the most recent example that we visited. So, I thought it will be fresh in your mind. So, it is good to revisit that example. So, the example that we did was the type of a triangle example abbreviated as TriTyp. If you remember we had this program in the last week where we looked at this is TriTyp program which took 3 sides of a triangle as input and determined what was the kind of triangle. Was it in equilateral triangle, isosceles triangle or was it an invalid triangle and so on.

So, here is an example of how an interface based partition for that will look like. For example, I could say that there are 3 sides: side 1, side 2, side 3 and I could consider the relationship of the 3 sides to be with 0. So, I could say what is side one, is side one greater than 0 a side one and number length of side one a number equal to 0, is length of side one a number less than 0 that could be one partition. The next partition could be the same thing for side two is the length of side two a number greater than 0, equal to 0, less than 0. The next partition could be related to the same thing for side 3 is the length of side 3 greater than 0, equal to 0 or less than 0.

So, suppose I consider partition q one for side one what do I do? I choose one value from each partition which will result in 3 tests right. One for side one greater than 0 like for example, you could choose 7 or any number greater than 0 as the value. The second one says side one is equal to 0 that is what I have chosen as a second value. Third one choose a negative number side one less than 0. So, if you see here what do these two represent side 1 equal to 0, side 1 minus 3 which is less than 0, they represent 0 invalid values. So, what do they test about the program? They test whether the program handles these invalid triangles correctly. So that also satisfies this condition that I told you here that some partitions should include few invalid values.

So, this is one partition which checks for greater than 0, equal to 0, less than 0.


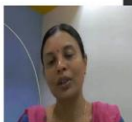
(Refer Slide Time: 25:05)

TriTyp Example, contd.

Another possible interface-based partitioning for TriTyp:

Partition	b_1	b_2	b_3	b_4
q_1 ="Relation of Side 1 to 0"	> 0	$= 0$	$= 1$	< 0
q_2 ="Relation of Side 2 to 0"	> 0	$= 0$	$= 1$	< 0
q_3 ="Relation of Side 3 to 0"	> 0	$= 0$	$= 1$	< 0

- This allows us to consider comparison to 1 too.
- These partitions are complete as the sides are integers. They will not be complete if the sides are floating point numbers.

Sometimes you might want to refine it into doing 4 partitions. You might want to do greater than 0, equal to 0, separately maybe equal to one and less than 0. This is just a choice, just to give you an illustration of an alternative option to do partition. In this case what will happen? There will be 4 test inputs: one for a side one greater than 0, 1 for side one equal to 0, 1 for side one equal to 1 and one for side one being a negative number. Similarly for side 2, similarly for side 3.

Now, another subtle point that I would like to like you all to understand and observe here is that if you remember the program TriTyp in input to the program were 3 integers. If you remember all the 3 sides sorry input to the program were 3 sides and all the 3 sides

what was their data type? They were all integers, these partitions are valid provided the 3 sides are integers. Suppose I will make the 3 sides of the triangle is floating point numbers, then this way of doing partitioning is not valid. Why is it not valid, because it will not be complete. If you see these 4 partitions, they, if they are floating point number then between these 4 partitions none of the partitions test for this length of a side of a triangle being a number between 0 and 1.

Let us say the length of a side of a triangle is 0.7, it does not test for that, it does not belong to any of these partitions. So, it does not satisfy the completeness criteria, but for that program that we had our type of the 3 sides was integers. When I consider integers these partitions are valid.



(Refer Slide Time: 26:54)

TriTyp Example, contd.

Functionality-based partitioning for TriTyp:
This partitioning is based on the type of the triangle, which is the main functionality of the TriTyp program.

Partition	b_1	b_2	b_3	b_4
q_1 ="Geometric classification"	scalene	isoceses	equilateral	invalid

Partitions above are *not* disjoint.

So, you have to be very careful in determining whether partition is valid or not. Based on the type of the domain, one partition could be valid for one domain type and could be invalid for another two domain type. The same partition could be invalid for another domain type. So, when I do functionality based partitioning of TriTyp, I do not directly look at inputs in isolation I start looking at the functionality of the program. What was the program TriTyp supposed to do? It was supposed to determine the type of a triangle. What were the 4 types of outputs that it was suppose to produce? It was supposed to produce numbers as 0, 1, 2, 3 to tell you what the kind of triangle it was. It was supposed to tell you whether the triangle was scalene or isosceles or equilateral or invalid.

So, functionality based approach will consider the functionality of a program, what is the main functionality of the program and try to come up with the characteristic based on the functionality. In the case of type of a triangle the functionality was the type of the triangle itself. So, here is one partitioning. So, I say my criteria for partitioning is the classification for the type of a triangle and then there are 4 partitions: triangle the 3 sides are such that the triangle is scalene, the 3 sides are such that the triangle is isosceles, the 3 sides are such that the triangle is equilateral, and one of the sides or maybe more than one of the sides are such that the triangle is not valid triangle.

So, suppose I consider a partitioning this way. Before we move on in design test cases as I told you the first thing to understand is our partitions complete our partitions disjoint you will realize that these partitions are not disjoint. If you think for a minute can you tell me why these partitions are not disjoint, what is the problem? The problem is if you see every equilateral triangle is also an isosceles triangle. What is an equilateral triangle, it means all the 3 sides are equal. What is isosceles triangle, it says any two sides are equal. So, when all the 3 sides are equal any two sides are equal. So, these partitions suppose I give a triangle that is equilateral it will belong to b_3 , partition and it will belong to b_2 partitions. So, it does not satisfy the property of disjointness.



So, how do I overcome this small issue? The tweak or the correction that I have to do for the partitions is very simple.

(Refer Slide Time: 28:53)

TriTyp Example, contd.

Functionality-based partitioning for TriTyp that is disjoint.

Partition	b_1	b_2	b_3	b_4
q_1 ="Geometric classification"	scalene	isoceses, not equilateral	equilateral	invalid



So, I just rewrite the partitions to mean that it is scalene, it is a isosceles, but not equilateral, it is equilateral and it is an invalid triangle. What will this achieve this will make sure that here for b_2 , I do get a triangle that is only isosceles. It will separate equilateral triangles that also have the scope for being isosceles. So, it says isosceles triangle not equilateral. This way I made sure that the partitions b_2 and b_3 which were overlapping here become disjoint here.

(Refer Slide Time: 29:30)

TriTyp Example, contd.

Possible test inputs for functionality-based partitioning for TriTyp that is disjoint.

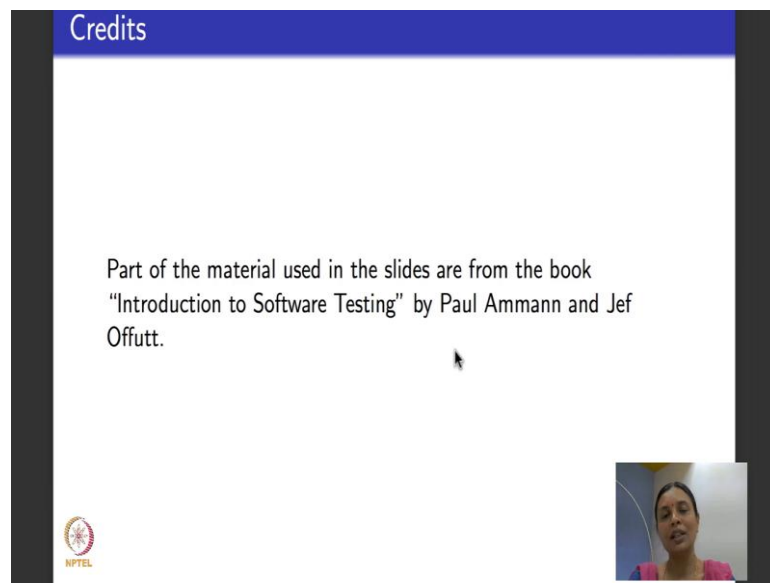
Param	b_1	b_2	b_3	b_4
Triangle	(4,5,6)	(3,3,4)	(3,3,3)	(3,4,8)

Now, I am ready to give test inputs to get to start testing the program. The test inputs that I could give are one set of test cases that will test with other triangle is scalene and for example, 4, 5, 6. This is good enough to test for triangle scalene, it will be this is another set of test cases with two sides being equal to test if a triangle is isosceles, but not equilateral. The third set of test cases with all 3 sides equal have given 3, 3, 3, give any values all of them need to be the same, 4, 4, 4, 5, 5, 5, anything is alright. This tests for equilateral triangles this tests for not a valid triangle is this clear.

So, if you see throughout what we have done we looked purely at the input looked at what input values could be given and how they could be partitioned, based on just the input values which is interface based approach, which was discussed here or based on the overall functionality, which is the functionality based approach which was discussed here.

(Refer Slide Time: 30:28)



So, to design test cases for input domain partitioning, I do not really need the program code. I need knowledge about the inputs and I need to know what the functionality of the program is and I could partition the input domain in several different ways. What will do in the next lecture is, we look at once I have come up with the partitions, how do I design test cases? What are the various coverage criteria to design test cases? Do I pick one value from each partition, do I pick one value from each partition such that the every other value from other partition is covered, do I pick two values from each partition, is there one partition that I have to focus on? All these things we will understand in the next lecture.

Thank you.