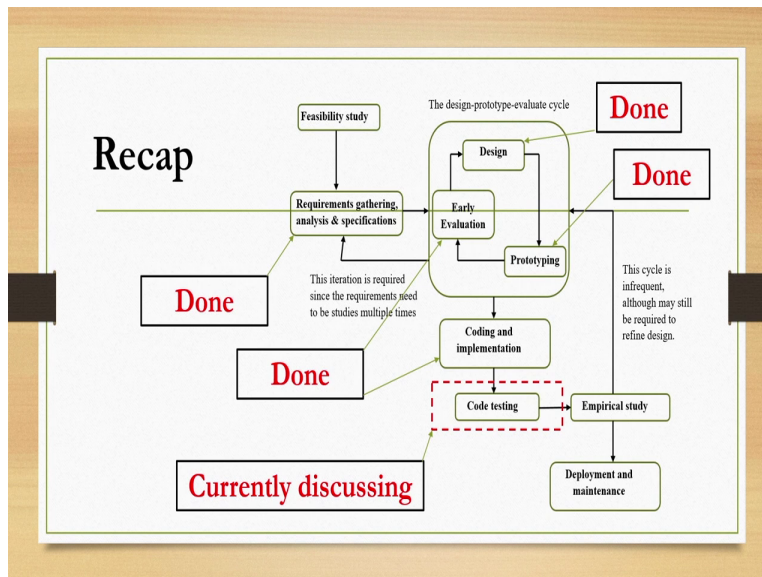


Design and Implementation of Human – Computer Interfaces
Prof. Dr. Samit Bhattacharya
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture: 32
Black-Box Testing II

Hello and welcome to the NPTEL MOOCS course on design and implementation of human computer interfaces lecture number 28 where we will continue our discussion on black box testing. So, before we start let us quickly recap what we learned and where we are now.

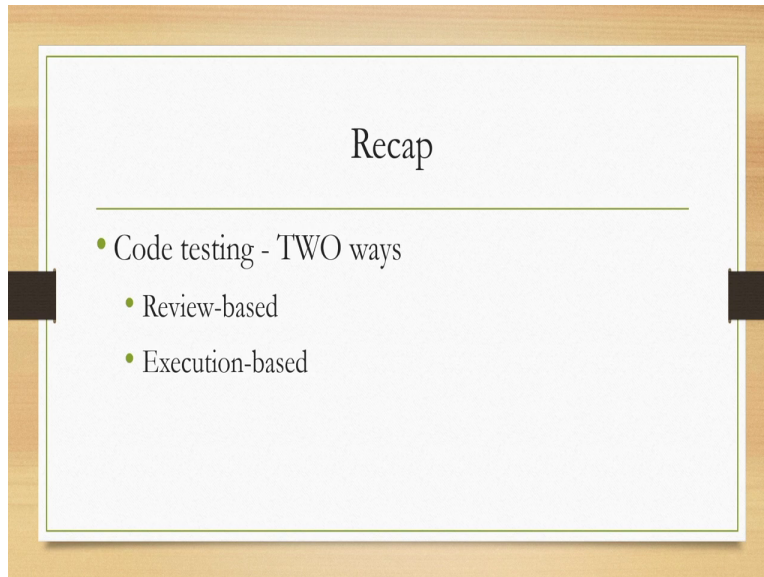
(Refer Slide Time: 01:03)



So, as you may recollect we are discussing the interactive system development life cycle there are several stages in the life cycle among these stages we have already covered some of the stages and we are currently focusing on a particular stage. So, we have already covered the requirement gathering stage design prototype evaluate cycle as well as the code design stage coding and implementation stage and currently we are discussing code testing.

That is once the system is implemented or the code is written for the system implementation what we do for testing the efficacy of the code.

(Refer Slide Time: 01:53)



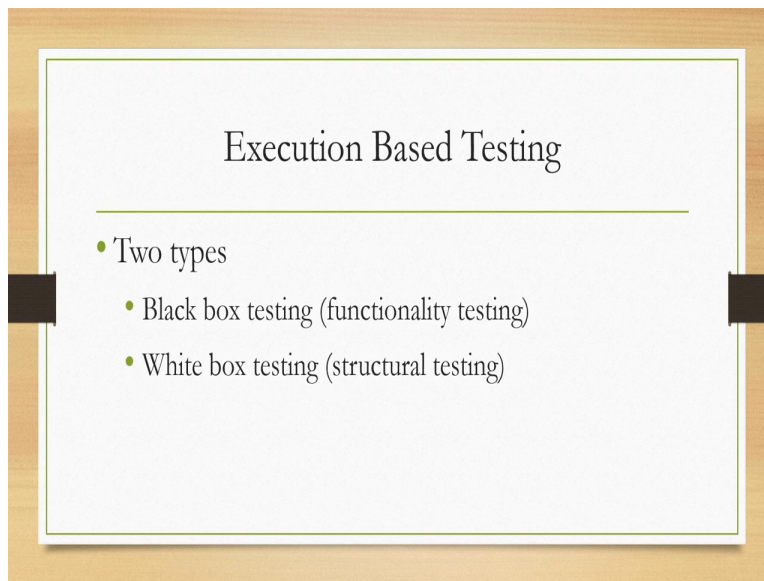
There are broadly two ways using which we can design our testing process or there are broadly two ways using which we can test our code one is review based and the other one is execution based. In review based code testing again there are broadly two types code walkthrough and code inspection. Primarily in review based code testing what we do we basically ask the evaluators to go through the code and identify issues that they feel are present in the code.

Here it is not mandatory to execute the code although in the code walkthrough method some test cases may be provided for hand execution of the code. What we get as output of review based code testing is essentially qualitative issues that are there in the code which include conformation with the coding standards conventions and presence of common errors mostly these type of issues. The purpose of review based code testing is to identify as many issues as possible before we go for more rigorous and formal testing.

It may be noted that more rigorous and formal code testing involves more effort than review based code testing. So, if we can clean the code as much as possible before we go for this formal code testing then our effort may be less the effort required to test the code formally may be less. So, that is the objective of review based code testing to give us a quick review of issues that are present in the code.

Once that is done we go for execution based code testing. Of course it is not mandatory to follow both the methods any one is also fine as long as we have the ability to deal with the effort required. In execution based code testing which is also the term used for referring to formal code testing. We generally use a set of test cases which is input output pair for a given input the output is mentioned and then we execute the code with the input to see whether the desired output is generated or not if not then there are issues.

(Refer Slide Time: 04:48)



And then we try to rectify those issues in execution based code testing there are broadly two ways of doing that one is black box testing or functional testing other one is white box testing or structural testing. Now currently we are discussing black box testing what it is.

(Refer Slide Time: 05:03)

BBT - Idea

- Design test cases based on input/output values ONLY
[no knowledge of design or code required]

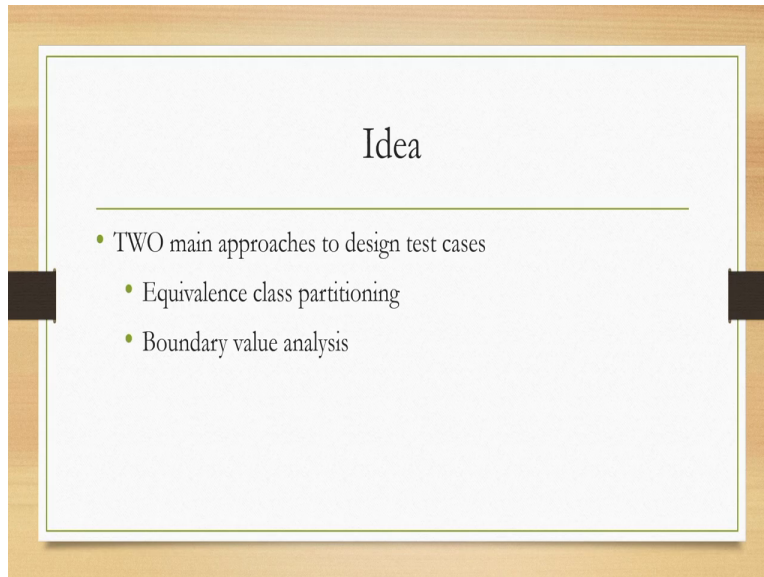
The basic idea is that here in this case we try to design test cases based on only input output values rather than the design or the internal structure of the code. So, as we have mentioned in the previous lectures it is very important when we go for formal code testing to design appropriate test cases. Now randomly designed large number of test cases not necessarily reveal errors with the program what we need is a more systematic approach to design suitable test cases.

Now there are broadly two ways to do that one as we mentioned is a functional approach or also known as black box testing method and the other one is the structural approach which is also known as white box testing method. In functional approach what we do is basically we assume the code to be a collection of functions. Now each function has its own input and it produces some output even if that is null but still that is a void output.

When we are trying to test the code that means when we are trying to design the suitable test cases if we are bothered only about the input and output of a function rather than how the function actually processes the input to produce the output that is the internal code of the function then we are going to use black box testing method that means we are looking at the functions as black boxes which takes only input and produces output without any knowledge of how the output is produced.

In contrast when we are actually trying to design test cases with the full knowledge of how the output is produced that means we are aware of the internal structure of the code then what we are following is called structural testing method or white box testing method. So, currently we are discussing black box testing method.

(Refer Slide Time: 07:13)



In black box testing method again there are two ways using which we can design the test cases one is the equivalence class partitioning idea and the other one is boundary value analysis. We started our discussion on these two concepts both are important when we are going for designing test cases.

(Refer Slide Time: 07:45)

Equivalence Class & Partitioning

- Domain of input values partitioned into sets, each called 'equivalence class', such that program behaves the same for all the member of an (equivalent) class

In case of equivalence class partitioning what is the idea we assume that the input domain is a very large set of values it is. So, large that it is not possible to actually use each and every possible input for testing the program. So, what we do instead is we try to partition it into subsets each subset is called an equivalence class with the idea that the behaviour of the program for any element of an equivalence class is same.

That is suppose the class contains 20 elements or input values for each value the program behaves in the same way. So, to test the program if we take any value rather than all 20 values then that would be sufficient to represent the entire class behaviour. So, instead of 20 input values we can settle with one input value and the corresponding output value as a test case to test the behaviour of the program for the entire equivalence class of inputs that is the idea that means we are trying to manage the number of inputs to be considered for testing and we are trying to reduce the number to a manageable number.

(Refer Slide Time: 09:10)

Equivalence Class & Partitioning

- Testing code with any ONE value of an equivalence class is as good as testing with ALL input values belonging to that class

And why this works because we are already assuming that testing code with any one value of an equivalence class is as good as testing with all input values belonging to that class. So, this is very significant. So, we have to be very careful while we are partitioning the input domain into equivalence classes. If we do not do it properly then definitely this assumption will not hold and our testing will be flawed. Let us try to understand this concept with some examples.

(Refer Slide Time: 09:40)

Example 1

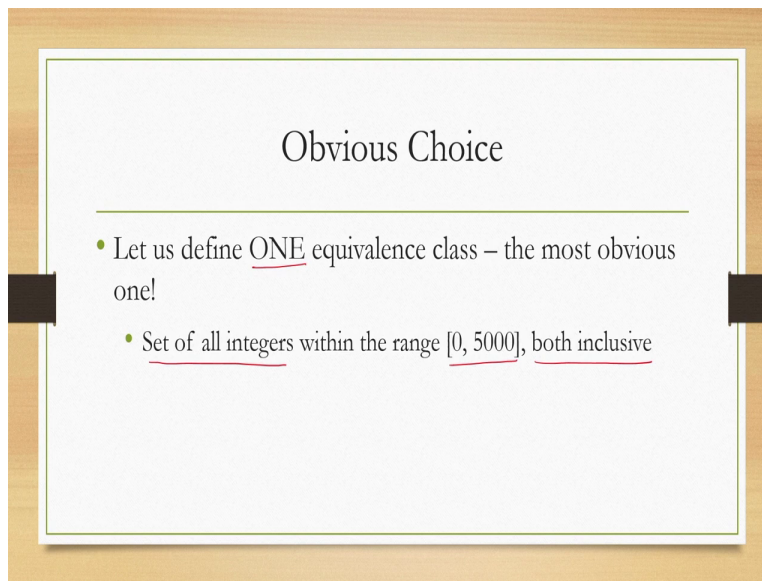
- Consider a code to compute square root of an input integer in the range [0, 5000]
 - Input – integer within the range, output – square root
 - What is/are the equivalence class(es)?

We will see few examples where the idea of equivalence class how to choose it and how to select equivalence classes so, that our testing gives us as many errors as possible that happens. So, with few examples we will try to understand that. So, let us start with our first example. Suppose you

have written a code to compute square root of an input integer in the range 0 to 5000 both inclusive. Now you may think that is a very odd program why it should be restricted within this range.

If I write a program that should be able to take any value as input and produce any output. Let us for the time being ignore those concerns and let us assume that this is what our code does for the sake of simplicity in discussion. So, here the input is an integer within the specified range that is between 0 to 5000 and the output is the square root of that integer. Now if this is the given scenario then what would be the equivalence classes.

(Refer Slide Time: 11:04)



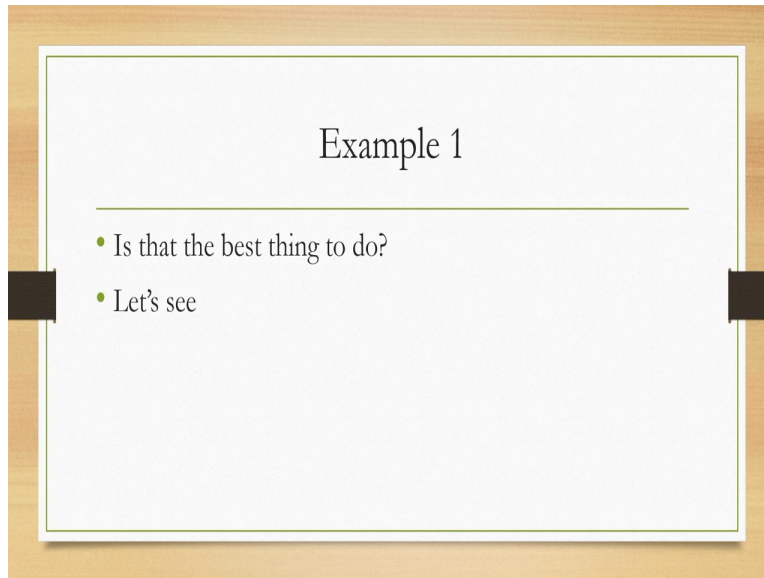
The slide is titled "Obvious Choice" and contains a list of two points. The first point is "Let us define ONE equivalence class – the most obvious one!". The second point is "Set of all integers within the range [0, 5000], both inclusive".

Let us start with defining only one equivalence class. So, if this problem is given to you then the most obvious answer would be to come up with an equivalence class with a single equivalence class which is set of all integers within the range 0 to 5000 both inclusive. So, that is the most obvious answer that probably most of us will come up with when this problem is given to us. So, what is our problem our problem is to identify the equivalence classes for the particular program to be tested.

Now the program takes as input an integer within the range 0 to 5000 both inclusive and when we are asked to come up with the equivalence class partitioning for this particular problem most

of us most likely to come up with this answer that there will be only one class that is the set of all integers between this range 0 to 5000 both inclusive.

(Refer Slide Time: 12:19)

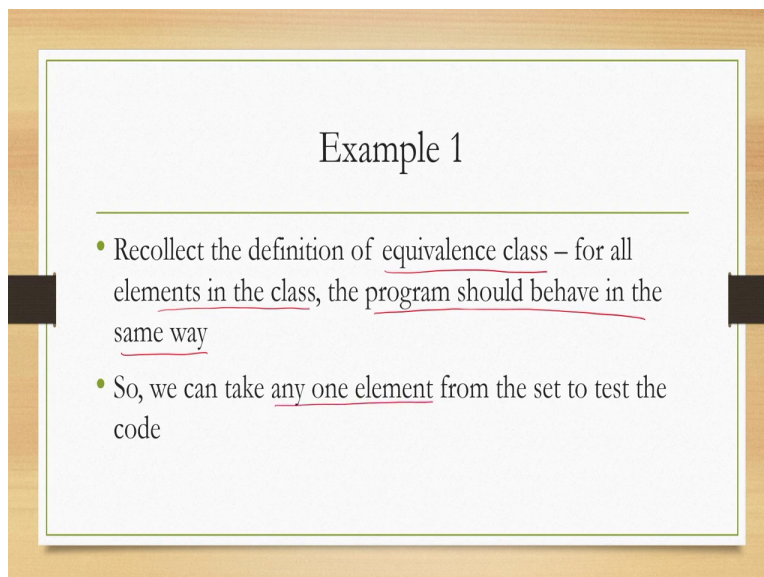


Example 1

- Is that the best thing to do?
- Let's see

Now of course is that the best thing to do will that give us what we are looking for that is a suitable test suit or the set of test cases that will be able to unearth all the problems that are there in the program. Can this single equivalence class give us that assurance let us see whether that is indeed the case.

(Refer Slide Time: 12:49)



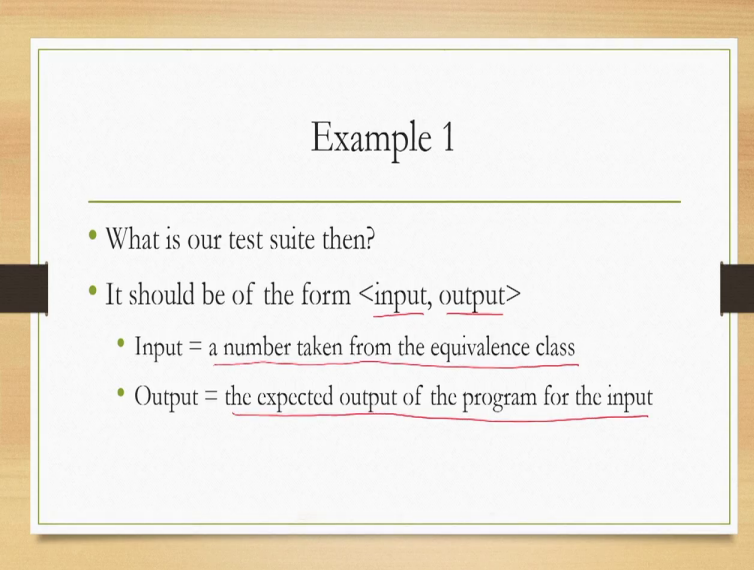
Example 1

- Recollect the definition of equivalence class – for all elements in the class, the program should behave in the same way
- So, we can take any one element from the set to test the code

So, what is the definition of equivalence class it is a set of elements where for all elements in that particular set or the class the program should behave in the same way. So, if we are defining something as an equivalence class then any element in that equivalence class when used as input should lead to the same behaviour in the program that means the program should be able to produce the same output.

In other words in an equivalence class we can choose any element to test our program it is not necessary to choose all the elements we can choose any one element because that is representative of all the elements that are present in the class. Then in our example if we assume that there is single equivalence class which is the set of all integers within the range 0 to 5000 then what can be our test cases and test suit.

(Refer Slide Time: 14:06)



Example 1

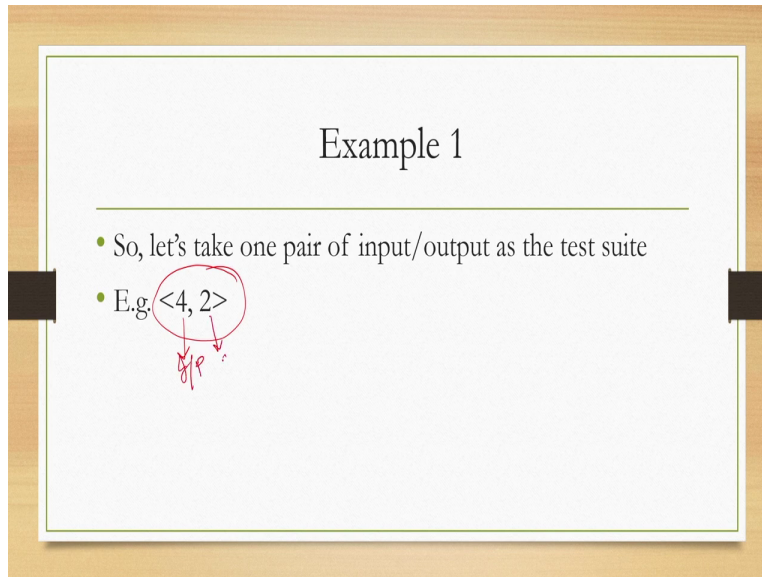
- What is our test suite then?
- It should be of the form <input, output>
 - Input = a number taken from the equivalence class
 - Output = the expected output of the program for the input

Now our test cases should be in the form input and output as we have already seen earlier. So, this is the doublet of the form input and output where the input is any number taken from the equivalence class a single number of course and output is the expected output of the program for the given input. So, we are choosing a number as input and if that number is given to the program it is supposed to produce some output.

So, the correct output that is supposed to be produced by the program is chosen as the output of the test case. Now in this example if we have chosen a single equivalence class which is the set

of all integers between the range 0 to 5000 both inclusive then we can choose any one test case because all the other inputs will lead to the same program behaviour. So, no need to test for other inputs.

(Refer Slide Time: 15:24)



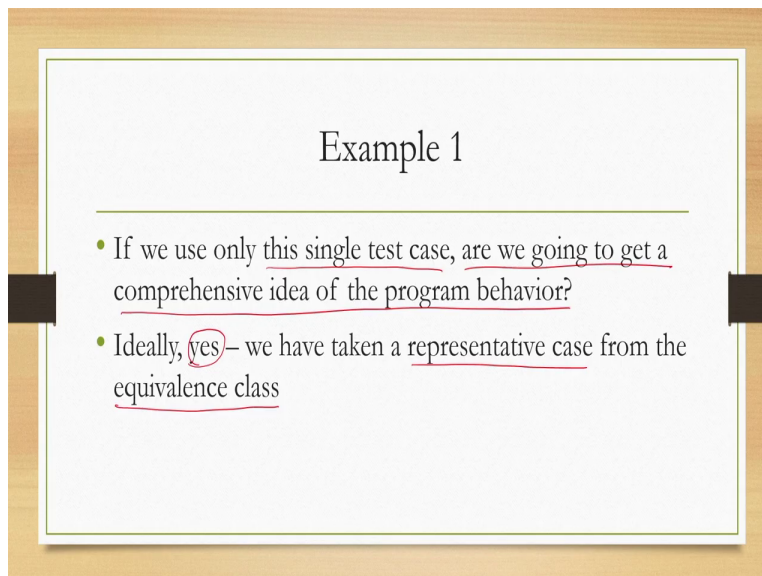
Example 1

- So, let's take one pair of input/output as the test suite
- E.g. <4, 2>

Handwritten annotations: A red circle around the test case <4, 2>, a red arrow pointing from the circle to the number 2, and the handwritten text 'input' below the circle.

So, let us choose this one 4, 2 where 4 is the input and 2 is the output that is 4 if given as input to the program produces the square root of 4 that is 2.

(Refer Slide Time: 15:41)



Example 1

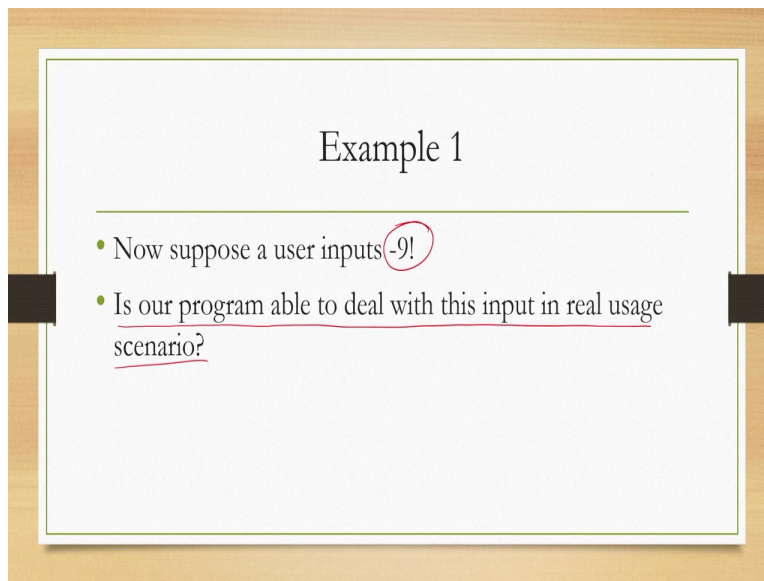
- If we use only this single test case, are we going to get a comprehensive idea of the program behavior?
- Ideally, yes – we have taken a representative case from the equivalence class

Now if we use only this single test case. So, our test suit consists of a single test case are we going to get a comprehensive idea of the program behaviour. Here of course we are assuming

that we have a single equivalence class that means by definition we should be okay with a single test case but does that really mean that with that single test case we will be able to identify all issues that are there with the program.

Ideally it should be yes we should be able to identify all issues that are present with the program because we have taken a representative case from the equivalence class. Now here of course the term representative is not very significant because any element from the equivalence class is a representative case. So, we do not need to do anything extra but let us see one more example of an input.

(Refer Slide Time: 16:49)



Example 1

- Now suppose a user inputs -9!
- Is our program able to deal with this input in real usage scenario?

Suppose the user inputs -9 is our program able to deal with this input in real usage scenario. Now our program can produce output when the input is given within the range 0 to 5000 in real usage scenario a user can make mistakes the user can erroneously put -9 in that case have we tested for that input with our test suit definitely no. We have not tested for this case which may happen in real world scenario because users can make mistakes.

(Refer Slide Time: 17:33)

Example 1

- Answer – a big NO!
- We never tested for this input – it's outside the range we considered
- So, we missed something in our equivalence class formulation

We never tested for this input it is outside the range we considered. So, what it implies it implies that we missed something in our equivalence class formulation. It is not necessary that when we are thinking of equivalence class we only think of situations where there will be no errors. To our eye is human that we all know. So, while providing input it is possible to make mistakes and we should be also careful about handling those scenarios where user input is not within the limit that is specified for correct behaviour of the program.

(Refer Slide Time: 18:31)

Next Obvious Choice

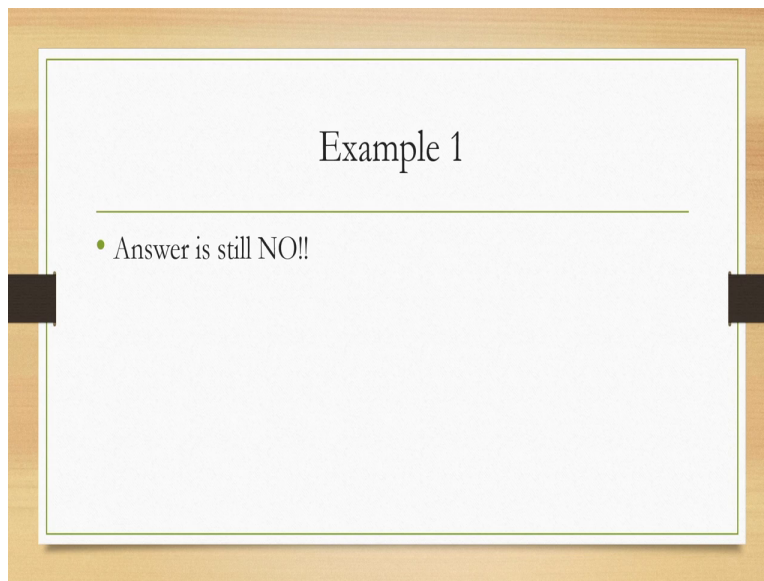
- To take care, let's us redefine our equivalence class definition – now, the class is defined as “the set of all integers, both positive and negative” – another obvious one!
- Will it serve our purpose?

To take care of this scenario let us redefine our equivalence class definition. Now the class is defined as the set of all integers both positive and negative because we talked of -9. So,

obviously it may come to our mind that ok then let us extend our definition let us incorporate all integers not only the integers within that specific range but all integers whether negative or positive. So, this is another obvious extension of the definition of equivalence classes.

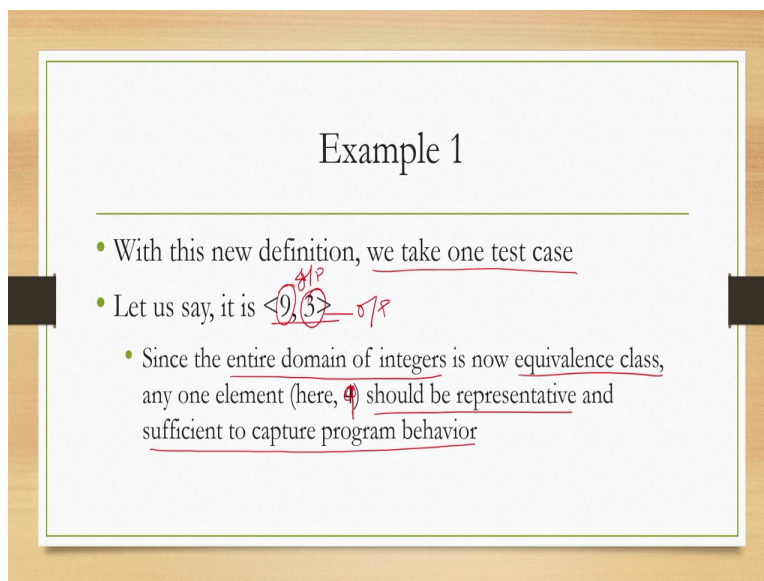
Will it serve our purpose with this extension of definition will we be able to test program behaviour for different scenarios.

(Refer Slide Time: 19:23)



The answer is still no we will not be able to test for different scenarios.

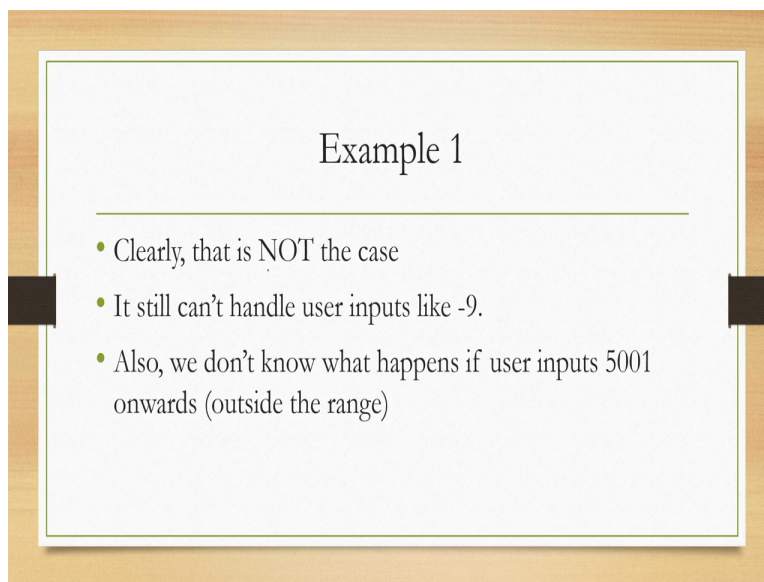
(Refer Slide Time: 19:27)



Why let us see, with this new definition again we take one test case let us say it is 9, 3; 9 is the input and 3 is the output that is square root of 9. Now since the entire domain of integers is. Now considered to be the equivalence class any one element here 9 it should be 9 rather than 4. Any one element should be representative and sufficient to capture program behavior the same thing that we have discussed earlier that is.

Now we have extended the definition of equivalence class it encompasses all integers however for any integer within this class the program should behave in the same way as per definition of equivalence class. So, if we choose any one integer from this class which in this case is 9 as input then the program behaviour should be same for all other integers from this class but is it really.

(Refer Slide Time: 20:39)



Example 1

- Clearly, that is NOT the case
- It still can't handle user inputs like -9.
- Also, we don't know what happens if user inputs 5001 onwards (outside the range)

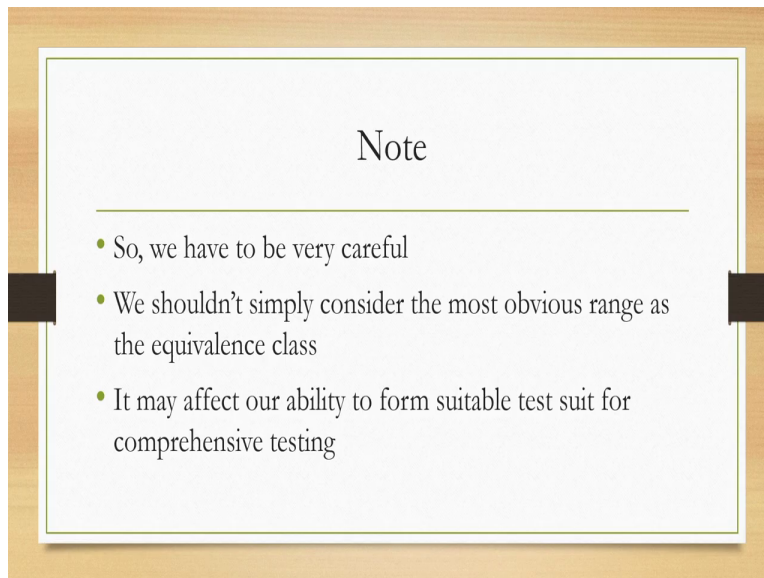
So, clearly that is not the case it still can't handle user inputs like -9. Remember that by definition we said that equivalence classes have elements that lead to same behavior for all inputs however this particular program that we are discussing are designed to deal with only inputs from the range between 0 and 5000. Now if we are considering minus 9 as an input then of course this program is not designed to take into account that input because it is outside the range.

On the other hand we have considered this as part of the equivalence class. So, clearly there is some issue with our definition of equivalence class considering the scope of the program. Also

we do not know what happens if user inputs 5001 onwards that is 5001, 5002 some integers that is greater than 5000 or outside the range that is defined for this program. So, with this definition of equivalence class where all integers are involved we can choose any one integer as representative case.

But that will not tell us anything about the program behaviour when erroneous inputs are provided such as negative integers or integers that is outside the range namely 5001 onwards.

(Refer Slide Time: 22:15)



So, you have to be very careful. We should not simply consider the most obvious range as the equivalence class. So, that is the biggest folly we generally make when we think of equivalence class as the most obvious range of values. It may affect our ability to form suitable test suit for comprehensive testing if we are unable to properly identify the equivalence classes.

(Refer Slide Time: 22:44)

Example 1

- Set 1 - Set of negative integers (INVALID input range)

Now let us go for a non-obvious solution for the same problem that is testing the code which produces square root for any integer within the range 0 to 5000. So, instead of 1 let us now consider 3 equivalence classes. First equivalence class: that is set 1 which is set of all negative integers which represents invalid input range. So, from the point of view of the design of the program any negative integer provided as input is an invalid input.

So, we consider a separate equivalence class for all invalid inputs where inputs are negative integers.

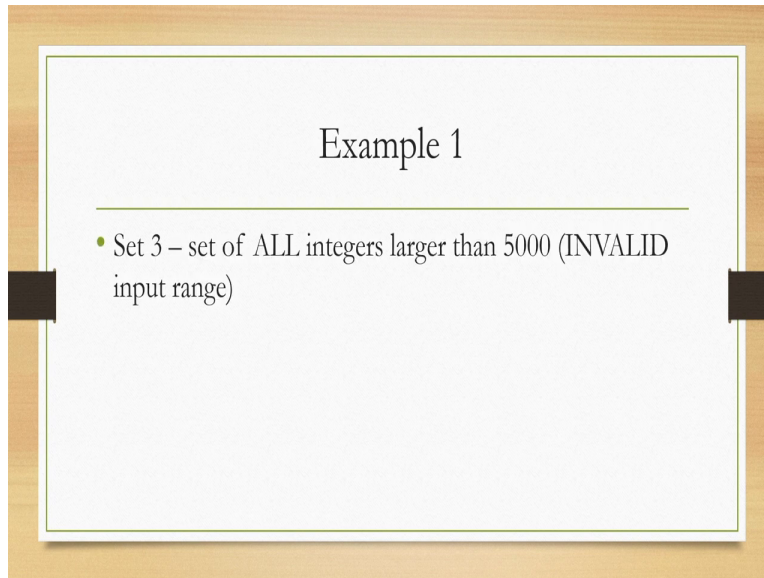
(Refer Slide Time: 23:40)

Example 1

- Set 2 - set of integers in the range of [0, 5000], both inclusive (VALID input range)

Then we have another set that is set to that is set of all integers in the range of 0 to 5000 both inclusive. Now this represents the valid input range.

(Refer Slide Time: 23:53)



and then we have another set that is set of all integers that are larger than 5000 which is another invalid input range. So, we have. Now three equivalence classes one represents all negative integers one represents all valid input that is integers within the range 0 to 5000 both inclusive.

And one represents all invalid positive integers that is integers which are greater than 5000. If we have these three sets as our equivalence classes then what will be the test suit.

(Refer Slide Time: 24:29)

Example 1

- Test suite must include representative input (and corresponding output) for each of the three equivalence classes

Test suite must include representative input from all equivalence classes that means in this case there will be three test cases comprising of the test suite one is for the three equivalence classes. Now in the test case of course along with the input we need to provide also the corresponding output to complete the doublet.

(Refer Slide Time: 24:57)

Example 1

- e.g., a test suite: $\{(-5, \text{output}), (500, \text{output}), (6000, \text{output})\}$
- Note
 - It is a set of THREE elements
 - Each element is a representative of the corresponding equivalence set (set 1, set 2 and set 3)
 - Each element of the form (input, output)

For example we can have a test suite like these three test cases comprising of these three test cases minus 5 with the corresponding output 500 with the corresponding output 6000 with the corresponding output. Now in minus 5 that is an invalid input in case of minus 5 which is an

invalid input the output can be a message a message that invalid input. Similarly when 6000 is provided as input output can be a message like invalid input.

Whereas in case of 500 output can be the square root of 500. So, with this will our problem be solved. Now before we go to that question let us just note these two things that is it is a set of three elements that means our test should consist of three test cases each element or each test case is a representative of the corresponding equivalence set that is set 1 set 2 and set 3. Each element is of the form input and output. So, these things we should keep in mind while we are going to design our test suit.

Now if this is the scenario where we are forming three equivalence classes for the input domain and we are going for non obvious classes then will it be able to handle all possible scenarios. Of course in this case the answer is yes we know what happens when the invalid inputs are provided and we know what happens when the valid inputs are provided. So, it gives a better partitioning than the earlier obvious partitioning schemes.

So, thus with a more granular classes we could take care of different situations and once problems are identified we can accordingly modify the code to take care of those let us give one example of how to identify the issues and take care of those. So, when the test should suppose minus 5 and output is given where output is coming as null. So, probably this is not a very informative output. So, we may like to provide to the user a message mentioning invalid output rather than simply printing null. So, then there is some issue.

So, maybe we need to rectify the corresponding code for dealing with invalid inputs and then there we need to print message. So, that is one way of taking care of issues while testing the code.

(Refer Slide Time: 28:14)

Example 2

- Let's consider another program for testing
- It computes intersection point of two straight lines and displays the result

Now let us go for another example to better understand the concepts let us assume that you are asked to write a program which computes intersection point of two straight lines and displays the result and you are asked to test it that means you are asked to design suitable test cases or the test suit to test this program. What you are going to do your job is to identify the equivalence classes and then correspondingly identify the test cases or the test suit.

(Refer Slide Time: 28:44)

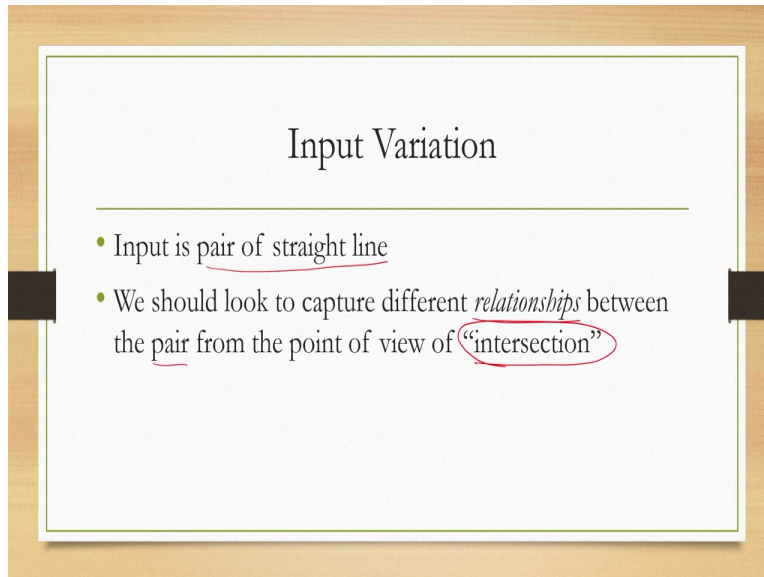
First (Obvious) Choice

- Set of ALL lines!
- Clearly, it will not work
- Why?

So, in this program we have as input two integer pairs $m_1 c_1$ and $m_2 c_2$ each pair defines a straight line assuming the form y equal to $mx + c$. So, m is the gradient and c is the intercept part. Now the output is the intersection point of these two lines if this is the problem that we are asked

to design suitable test cases for this testing of this code then what would be the equivalence classes. If we go by the obvious choice that is we can settle for set of all lines as representing the equivalence class will it work? It will not work. Why it will not work?

(Refer Slide Time: 29:34)



Input Variation

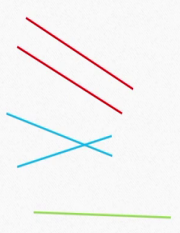
- Input is pair of straight line
- We should look to capture different *relationships* between the pair from the point of view of “intersection”

For the same reason we will not be able to create a test suit to capture variations in input that means we will not be able to capture different execution scenarios for different inputs. So, what are those different scenarios what are the variations that we are referring to here, let us quickly have a look at those variations. Now the input is pair of straight line when we talk of variation we should look to capture different relationships between the pair from the point of view of intersection. So, what can be the different relations between those input lines.

(Refer Slide Time: 30:18)

Relationship

- Parallel lines ($m_1=m_2$, $c_1 \neq c_2$)
- Intersecting lines ($m_1 \neq m_2$)
- Coincident lines ($m_1=m_2$, $c_1=c_2$)



There are three possibilities there can be parallel lines that is m_1 equal to m_2 and c_1 not equal to c_2 . We can have intersecting lines m_1 not equal to m_2 we can have coincident lines m_1 equal to m_2 c_1 equal to c_2 . So, these three relationships can be there between the input pairs between the input lines parallel intersecting or coincidental lines.

(Refer Slide Time: 30:55)

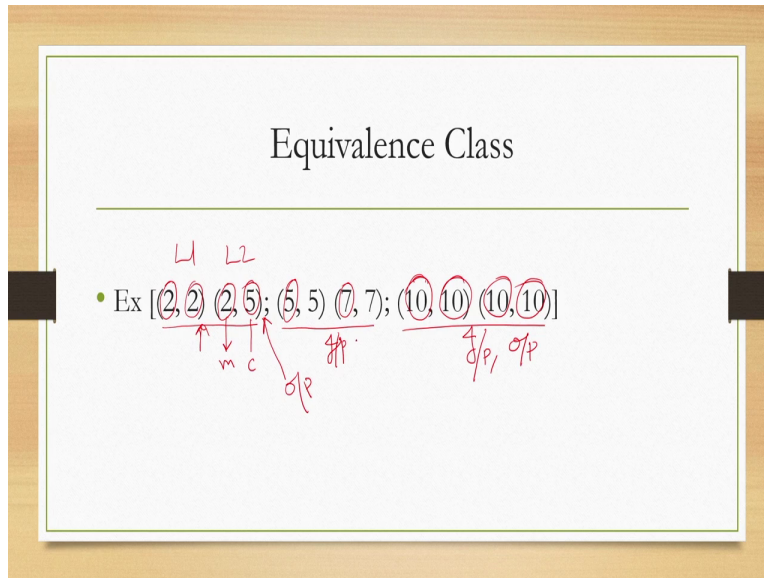
Equivalence Class

- THREE
 - Set of all parallel lines
 - Set of all intersecting lines
 - Set of all coincidental lines

So, if that is known that is apparent to us then what would be the equivalence classes. Clearly to take care of these three different relationships we need to have three equivalence classes, set of all parallel lines, set of all intersecting lines and set of all coincidental lines. So, we have these

three equivalence classes one is set of all parallel lines second one is set of all intersecting lines and third one is set of all coincidental lines.

(Refer Slide Time: 31:34)



An example test suit is shown here. So, we have three elements each corresponding to one of the equivalence classes in each element we have two pairs this is line one this is line two this is m this is c. So, for each element that is each line it is represented with these two values gradient and intercept. Now we can have these three elements corresponding to these three equivalence classes of set of all intersecting lines set of all coincidental lines and set of all parallel lines.

So, when we are talking of set of all parallel lines we have m1 equal to m2 but c1 not equal to c2 and this input represents a representative element from the equivalence class. Of course here we have not provided the output for simplicity in ideally here we should also provide the output. Then we have set of all intersecting lines m1 not equal to m2 and set of all parallel lines where set of all coincidental lines where we have m1 equal to m2 and c1 equal to c1.

So, this is the input along with that we have to specify the output in each case. So, this is one equivalence class which this is one test suit based on the definition of the equivalence classes for this particular example which ideally should take care of the different scenarios. So, we will be able to know the program behaviour for these three different scenarios.

(Refer Slide Time: 33:41)

Food for Thought

- What would be the equivalence classes if, instead of lines, we now consider line segments?
 - Input of the form: $(x_{11}, y_{11}), (x_{21}, y_{21}); (x_{12}, y_{12}), (x_{22}, y_{22})$
[each pair indicate one end point]
 - Output – intersection point or NULL

Now let us quickly try to figure out this other problem. So, what would be the equivalence classes if instead of lines we. Now consider line segments. So, input is of the form $x_{11} y_{11} x_{21} y_{21}$ and so on where each pair indicate one end point and the output is intersection point or null. So, instead of lines we are now providing line segments as input and output is intersection point or null. Now if that is the case what would be the equivalence class or classes for testing this program. This is given as a take home exercise you may try it whenever you get time.

(Refer Slide Time: 34:31)

Example 3

- Let's consider a slightly different example
- You have developed an user authentication system, as part of the a bigger system. The authentication system/function takes as input user ID and produces success/failure message (which can be used to proceed further).
- Let's try to design the test suite for the example

Let us now move to our third example. This is a slightly different example earlier once mostly straightforward examples but this one is more practical more realistic example. Suppose you

have developed an user authentication system which is part of a bigger system that you are working on. The authentication system or the function let us consider this whole authentication system as nothing but a function.

It takes as input user id and produces success or failure messages which can be used to proceed further for this given program that is the function which takes as input user id and produces some message if I want to test this program what would be the equivalence class or classes how do we design our test suit for this program.

(Refer Slide Time: 35:30)



Now here we know that input is user id or user identifier but that simple bit of information is not sufficient we need more clarity on exactly what is the user identifier before we go for designing our test suit.

(Refer Slide Time: 35:48)

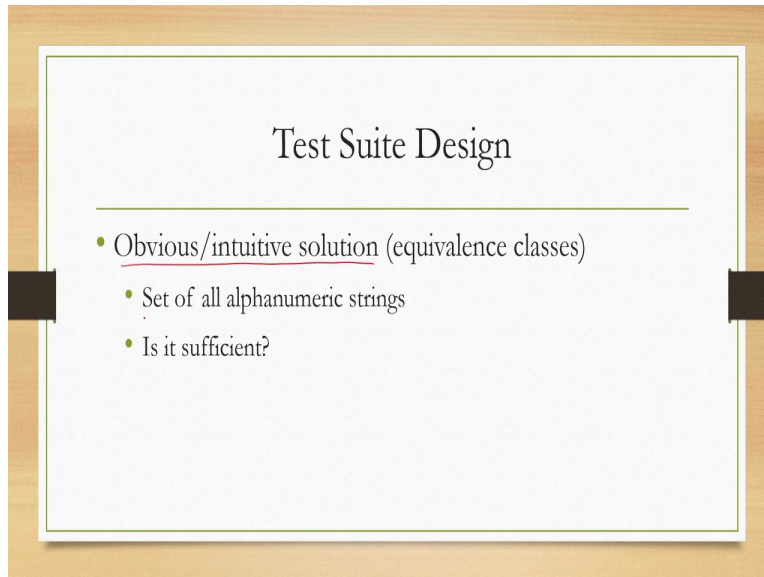
First Thing – Clearly Specified Input

- Clarified input – alphanumeric string
 - Still not sufficient
 - Need length information (string of any length acceptable)

So, first thing is you have to get more clarity on the input. Suppose better clarified input is provided which is the user identifier is nothing but an alphanumeric string is it a sufficient information? Still not sufficient. For further clarity we probably require length information also what is the length of the string. Is string of any length acceptable or the string must have a length restriction.

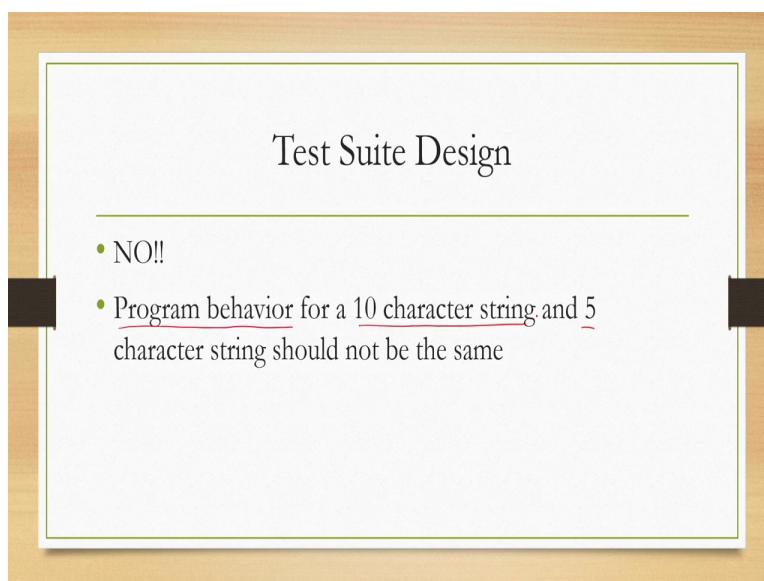
Suppose it is further specified that the length of the string is restricted to 5 and exactly 5 that is user id must be a five character string it cannot be a 4 character string it cannot be a 6 character string or any number other than 5. So, that is a very clear way of specifying the input that is the program takes as input a 5 character alphanumeric string as user identifier and produces a message of success or failure while authenticating the user.

(Refer Slide Time: 37:04)



Now with that knowledge let us try to go for design of the test suit. So, the obvious or intuitive solution for equivalence classes would be set of all alpha numeric strings but is it sufficient is it capable of identifying all issues with the program of course not.

(Refer Slide Time: 37:25)



Program behavior for a 10 character string and 5 character string should not be the same because we have specified that it should only accept 5 character string it should not accept 10 character string. So, definitely the output that means the behaviour of the program for these two different string lengths should be different.

(Refer Slide Time: 37:48)

A Better Solution

- Equivalence class 1 – set of all alphanumeric strings of length 5
- Equivalence class 2 – all other strings

- Will it work?

What can be a better solution let us have two equivalence classes set of all alpha numeric strings of length 5 and equivalence class 2 is all other strings, will it work? It looks like it should work but will it really work it may not work.

(Refer Slide Time: 38:10)

An Even Better Solution

- Class 1 – set of all 5 character alphabetic (only) strings
- Class 2 – set of all 5 character numeric (only) strings
- Class 3 – set of all 5 character alphanumeric strings
- Class 4 – set of all alphanumeric strings with length >5
- Class 5 – set of all alphanumeric strings with length < 5
- Class 6 – set of all strings with special characters

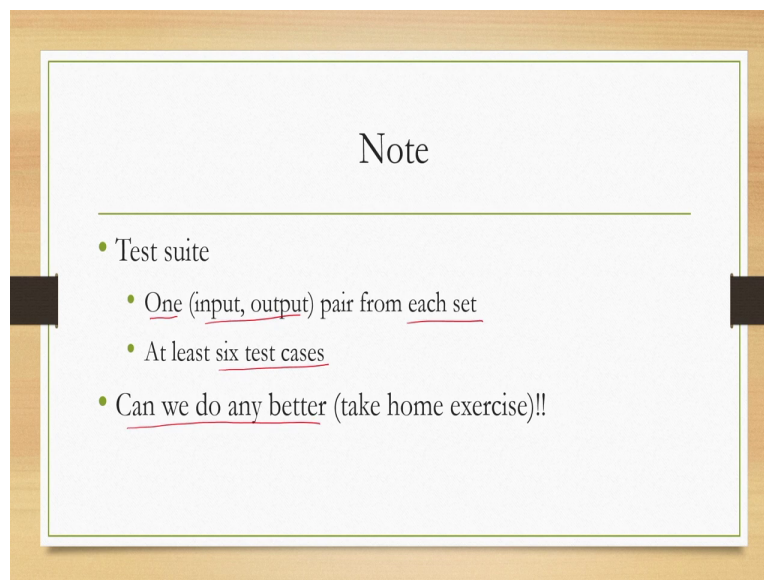
Let us have an even better solution. So, we have class one which is that is equivalence class one which is set of all five character alphabetic only strings equivalence, class 2o set of all five character numeric only strings, class 3 set of all five character alphanumeric strings, class 4 set of all alphanumeric strings with length greater than 5, class 5 set of all alpha numeric strings with

length less than 5 class 6 set of all strings with special characters that is characters other than alpha numeric characters.

So, if we have these six sets will it give us some better way of designing our test suits definitely yes if we have only two equivalence classes one is set of all five character alphanumeric strings and set of all strings that are not of five character length that will not give us many scenarios as listed here we will have our test suit comprising of only two cases which will not be sufficient to deal with these many scenarios that are listed in this improved solution.

That means what happens when we are inputting string with only numeric characters string with only alphabetic characters string with alphanumeric characters string with special characters although it may be a five character length strings of length more than five or less than five. So, all these different situations cannot be dealt with only two equivalence classes instead we have to go for more granular classes what is shown here.

(Refer Slide Time: 40:14)

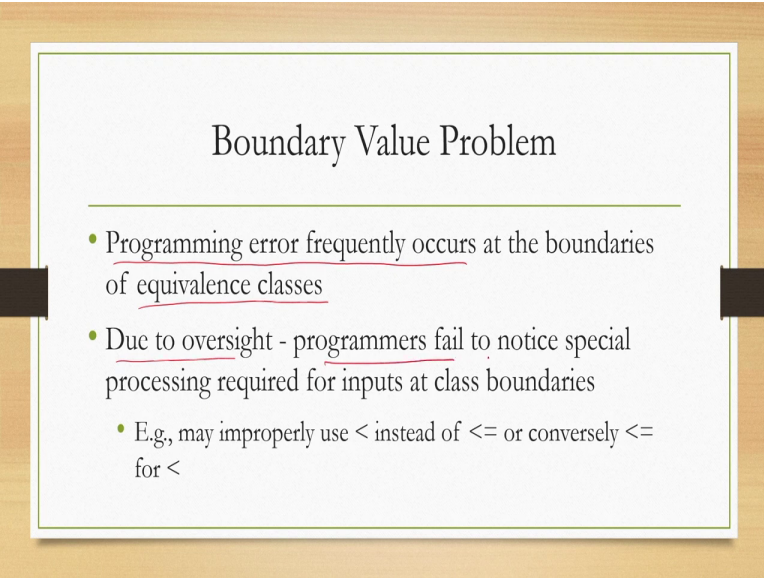


So, in this case what will be our test suit one input output pair from each set. So, that means at least six test cases we can have which will give us a better idea of the performance of our program in different input scenario in comparison with the test suit that we would have got had we been considering the two equivalence classes as mentioned earlier. Can we do any better than this six classes.

In fact these six classes are still limited there are other scenarios that are possible which will not be taken care of with these six classes. So, you can think of a better solution I am leaving it as a take home exercise like the earlier one where we are considering line segments instead of lines. So, that is all about understanding the concept of equivalence class partitioning I hope you got the idea that is when we are going for equivalence class partitioning we should be very careful what is not so, obvious can give us much better solution.

So, we should not always look for obvious solutions we should think hard think deeply to come up with suitable equivalence classes. The three examples are meant to give you an idea of how to think about suitably partitioning the input domain into appropriate equivalence classes. The other crucial component in designing our test suit while dealing with black box testing method is consideration of the boundary values or rather it is known as boundary value analysis.

(Refer Slide Time: 42:09)



Boundary Value Problem

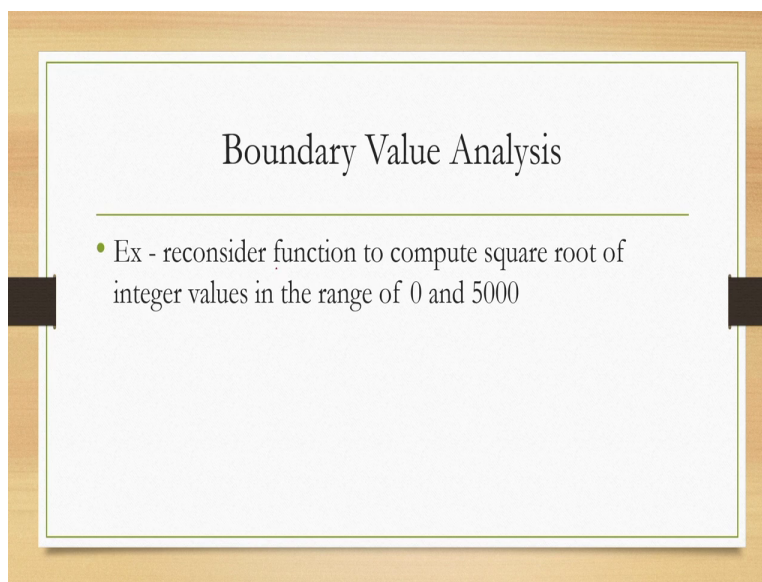
- Programming error frequently occurs at the boundaries of equivalence classes
- Due to oversight - programmers fail to notice special processing required for inputs at class boundaries
 - E.g., may improperly use < instead of <= or conversely <= for <

Programming errors frequently occurs at the boundaries of equivalence classes. So, equivalence classes are sets typically range of values at the boundaries of this range we often ignore those boundaries and at the boundaries of this range it is possible to encounter some error in the program. Now when we say that once an equivalence class is defined only one element from the class is good enough to test for all the values in that equivalence class.

We are ignoring this fact that at the boundaries program behaviour may change and we should also think carefully about our test cases when the boundaries are involved. So, due to oversight programmers often fail to notice special processing required for inputs at class boundaries for example while defining equivalence classes to define the set we may improperly use less than instead of less than equal to or conversely less than equal to instead of less than.

So, there may be oversight of defining these class boundaries and because of that oversight we often ignore some input values that may lead to different behaviour.

(Refer Slide Time: 43:38)



So, in order to address this issue this boundary value analysis is also important while designing the test suits for black box testing. It essentially refers to selection of test cases at the boundaries of the equivalence classes it is as simple as that. For example let us reconsider the earlier function to compute square root of integer values in the range of 0 to 5000 that is the first example that we have seen.

Now earlier we have seen that there are three equivalence classes that we can define to take care of different scenarios accordingly we can have three test cases comprising our test suit.

(Refer Slide Time: 44:24)

Boundary Value Analysis

- Earlier we considered test cases for 3 classes
 - Set of negative integers
 - Set of integers in the range of [0, 5000]
 - Integers larger than 5000

So, what are the three test cases one test case each for each of these equivalence classes that is set of negative integers set of integers in the range of zero to five thousand and integers larger than five thousand these are the three equivalence classes that we have defined and for each of these three classes we can choose one input output pair resulting in three test cases or a test suit comprising of three elements for testing our code.

(Refer Slide Time: 44:53)

Boundary Value Analysis

- Along with those, we should include test cases in our suit
 - $\langle -1, \text{output} \rangle$
 - $\langle 0, \text{output} \rangle$
 - $\langle 5000, \text{output} \rangle$
 - $\langle 5001, \text{output} \rangle$

Now along with those three test cases we also should take into account the boundaries of these equivalence classes. So, one boundary is the value minus one that is the boundary for negative integer for that boundary what should be the output we should take care of this one can be zero

that is one of the boundary for valid inputs one of the boundaries for valid input and the corresponding output should be there 5000 is the other boundary for valid inputs and the corresponding output should be there 5001 is another boundary for invalid input and the corresponding output should be there.


So, these are the boundary cases for the three equivalence classes which we should consider that means our test suit should have three plus this five this four boundary cases that means total seven test cases. So, only choosing test cases based on equivalence classes may lead to some problem if we ignore the boundaries because while defining the equivalence classes we may have one definition but that definition may be due to some oversight

And some improper usage of notations may lead to some confusion on interpreting the classes accordingly some values we may miss which may lead to different program behaviour. So, it is always preferable to be extra careful while defining the equivalence classes and include the boundary cases in the test suit. So, keep in mind to include boundary cases along with the equivalence classes while designing the test suit.

So, with that I hope you got some idea about the concepts that we are discussing namely equivalence classes and boundary values and why those are important few things we should note here.

(Refer Slide Time: 47:22)

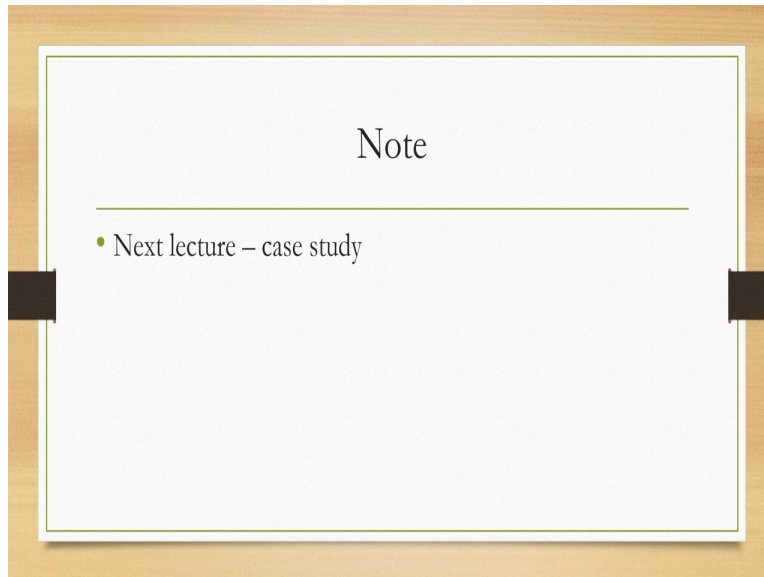
Note

- Equivalence classes are sets → should use set notations to represent
- Test cases should contain **both input and expected output** 
- Remember to include test case(s) from valid class(es), invalid class(es) and boundary cases in black-box test suites

One is equivalence classes are sets that means while trying to express those classes we should ideally use set notations. When you are defining test cases it should contain both input as well as the expected output. So, only one is not sufficient you must have this input output pair. And as a rule of thumb always keep in mind that you have to include test cases from valid classes invalid classes and boundary cases in the black box test suits.

Of course you should not consider this statement to be indicating that there will be only two equivalence classes valid and invalid. Valid classes can further be divided partisan as we have seen in our earlier examples similarly invalid classes can also be further divided into multiple equivalence classes. So, you should keep in mind that while designing equivalence classes you should consider valid cases or valid inputs invalid inputs and also while designing test suits you should consider boundary values for the equivalence classes.

(Refer Slide Time: 48:42)



In the next lecture we will go through one case study on how to create this testing document. So, as I said in this interactive system development life cycle at the end of each stage we produce some documentation earlier documents we have seen. After the code testing we also produce some documentation. So, after review based code testing as we have seen we produce a document after black box testing we can equally; we can similarly produce one document.

In the next lecture we will see how that document looks how it can be created in one case study. So, that is all for this lecture I hope you understood the concepts the last few things that we mentioned should be kept in mind that is equivalence classes are sets. So, you must use you must use set notations to represent them while creating the documents and there are three things you should keep in mind while going for equivalence classes.

That is think of valid inputs think of invalid inputs and think of boundary cases and while specifying the test suit you must include both input as well as output not only input which is a common mistake that we often make.

(Refer Slide Time: 50:22)

Reference

- Rajib Mall (2018). **Fundamentals of Software Engineering**, 5th ed, PHI Learning Pvt Ltd. **Chapter 10**
- Roger S Pressman (2020). **Software Engineering: A Practitioner's Approach**, 9th ed, McGraw-Hill Education, New York, **Chapter 19-21**

Whatever we are discussing can be found in these two books *Fundamentals Of Software Engineering* and *Software Engineering A Practitioners Approach* you can look at chapter 10 in the first book and chapters 19 to 21 in the second book to know more about software testing in more details. I hope you enjoyed the content and understood the concepts looking forward to meet you all in the next lecture, thank you and goodbye.