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

Lecture - 07
Special Value Testing

Welcome to this session. So far we had looked at some very basic concepts on testing, and then we started to look at black-box testing techniques. And we looked at a prominent black-box testing strategy which is equivalence class partitioning. And if you remember in equivalence class partitioning, we model the input data as ( ) classes and then the premise we said is that testing one value from each class is as good as testing all values in the class.
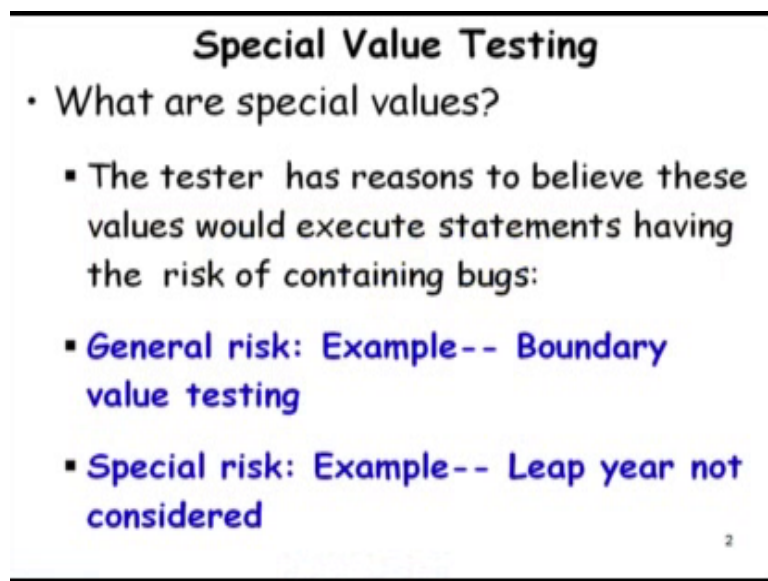
But then we said that identifying the equivalence classes given a problem description requires bit of experience and insight. We tried couple of problems and then we looked at when a unit test multiple parameters, and then we have to model multiple input domains for each of these parameters; and if these parameters are independent, then we have to consider all possible combinations of the equivalence classes for the two parameters.

And then we were looking at the concepts of weak equivalence testing, strong equivalence testing, and robust testing and so on. Today, we look at another black-box testing strategy called as special value testing. Now let us look at what is involved in special value testing.

(Refer Slide Time: 02:08)
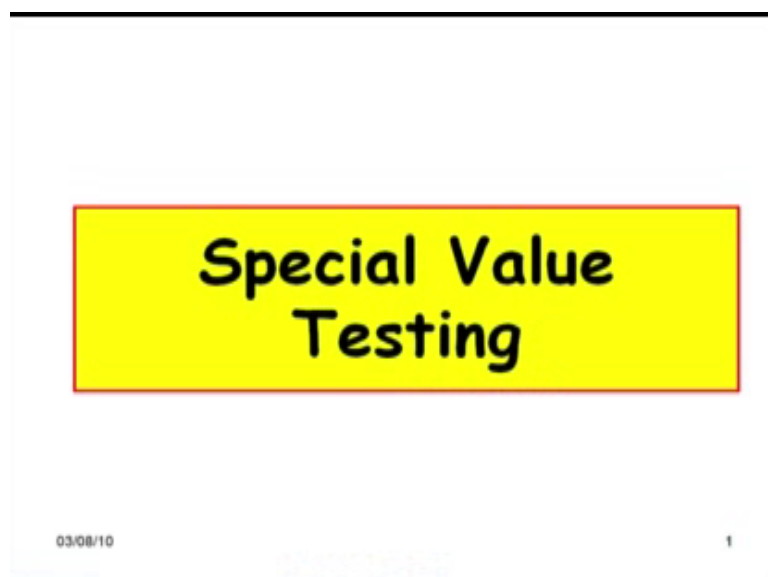
(Refer Slide Time: 02:11)

Special Value Testing

- What are special values?

  - The tester has reasons to believe these values would execute statements having the risk of containing bugs:

  - General risk: Example-- Boundary value testing

  - Special risk: Example-- Leap year not considered

We say that tester has some hunch given a software, he has hunch as to where the program might fail which input values may make the program fail and that is especially true for very experienced programmers. They somehow know some special values where the program is likely to fail. Let us look at what are the different special values where program can fail.
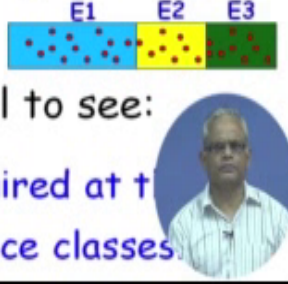
One is a general risk of failure applicable across all programs; the name of this special value is boundary value. The programmers for some reason while writing their program, commit errors at the boundary of the equivalence classes. We will just see why they commit mistakes; and then given any program, it is essential that we test the boundary values, because there is a large risk of bugs existing there.

The other is special risk, where depending on the kind of program, the problem description and so on. The programmer might miss certain things. For example, let's say we want to compute the day for a given date. So, the program computes given a date it would display what is the day. And here an experienced programmer would say that have leap years been taken into account, because the tester knows that while doing this kind of programming will have to convert the date into a number, and then by some algorithm we have to check what is the day for that. And here a likely error is that leap years have been not been taken into account. So, a possibility a special value is to give a test data which involves a leap year and check whether it is working.

Now, let us look at this very general special value testing called as boundary value, which is applicable to all types of software. So, here between the boundaries of equivalence classes, the programmer might commit error, because the programmer typically writes programs to distinguish between different equivalence classes by if statements or switch statements.

And there in the logical expression likely to be an error less than, less than equal to, greater than, greater than equal to there is a likely confusion. And therefore, we have to pick numbers at the boundary of the equivalence classes. So, the genesis or the importance of this testing is that there is a special processing or special consideration required from the programmer at the boundaries of equivalence classes, and we need to take whether he has taken care of that.

So, at the simplest the boundary value testing implies, we select test cases on the boundary of different equivalence classes.

I said ( ) the simplest because we have to also consider the other value that is namely the value that is just inside the boundary as well. So, given a range equivalence class specified as a range bounded by values a and b, we would have to include a and b, and also the one which is just above a and just below b. For example, if we have a range minus 3 to 10 integer values between minus 3

to 10 is range equivalence, it is equivalence class denoted by this range, then we would have to include minus 3, we would have to include 10 which are the boundaries of this equivalence class.

We would also have to include a number which is just higher than the lower bound that is minus 2 and just lower than the higher bound which is 9 and also a normal value which is 0. This because of the representative of the equivalence class we consider 0 as well. So, if we have a enumerated data 3, 5, 100, 102 etcetera, we would have to consider the lower bound 3, the higher bound 102, just above 3 which is 5, and the value which is just below 102 which is 100, it should written 100 and a value 1, sorry we should have written as 1.

(Refer Slide Time: 08:15)

Now, let us take some example. Let us say we have a HR application that we have written. And the policy for a HR is that if the applicant is between 0 to 12 years age, we do not hire; if the applicant is between 12 to 18 years of age we can only hire as an intern; and if it between 18 to 65, we can hire full time; and above 65, we do not hire.

So, the specification itself there is a problem; obviously, the code which as written based on this specification would have the similar problem. Just look at the boundary. So, 0 to 12, so is 12 included here, is 12 not hired, and is 12 hired is a intern, what about 18, can 18 be hired as an intern and also as a full time, what about 65. So, the specification itself has problem at the boundary.

Boundary Value Testing Example (cont)

- If (applicantAge >= 0 && applicantAge <=12)
  hireStatus="NO";

- If (applicantAge >= **12** && applicantAge <=18)
  hireStatus="Intern";

- If (applicantAge >= 18 && applicantAge <=55)
  hireStatus="FULL";

- If (applicantAge >= 65 && applicantAge <=99)
  hireStatus="NO";

If we translate it into code we would obviously, have the same problems at the boundary. So, just see here 12 would make this if statement true, and also the next if statement will also become true; similarly, the case for 18, 65 and so on.

Boundary Value Testing Example (cont)

- Corrected boundaries:

  0-11 Don't hire

  12-17   Can hire as intern

  18-64   Can hire as full-time employees

  65-99 Don't hire

- What about ages -3 and 101?

- The requirements do not specify how these values should be treated.

What we should have done in the specification is 0 to 11 - do not hire; 12 to 17 hire as intern; 18 to 64 - hire as full time employee and 65 to 99 - do not hire. So, this would have taken care of the consideration at the boundary and translation to code, a programmer coding this specification will

not have problem at the boundary, unless it does problem, commit mistake in the coding itself. And what we do not specify here is about the dates which exceed the boundaries here. What about minus 3, what about 101, so this also needs to be tested.

(Refer Slide Time: 10:49)

Boundary Value Testing Example (cont)
- The code to implement the corrected rules is:

```
If (applicantAge >= 0 && applicantAge <=11)
    hireStatus="NO";

If (applicantAge >= 12 && applicantAge <=17)
    hireStatus="Intern";

If (applicantAge >= 18 && applicantAge <=64)
    hireStatus="FULL";

If (applicantAge >= 65 && applicantAge <=99)
    hireStatus="NO";
```
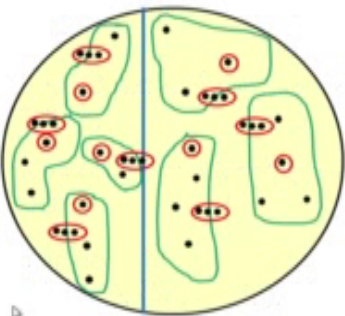
- Special values on or near the boundaries in this example are {-1, 0, 1}, {11, 12, 13}, {17, 18, 19}, {64, 65, 66}, and {98, 99, 100}.

So, this is the corrected code based on the corrected specification. And in any case, we will have to check the boundaries 11, 12, 0, minus 1, 17, 18, 64, 65, 99, 100 and of course, the nominal values we need to consider.

(Refer Slide Time: 11:25)

Boundary Value Analysis
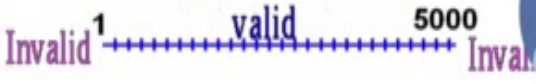- Create test cases to test boundaries of equivalence classes

So, to show it pictorially; at the boundary of different equivalence classes, the valid and invalid equivalence classes, we would have to take one value which is just on the boundary, one value just inside the boundary, one value just outside the boundary for each of the equivalence classes.

(Refer Slide Time: 11:53)

Now, let us take some examples. If we have a boundary 1 to 5000, we have equivalence class 1 to 5000, so what would be the boundary value test cases for this. We will have to have 5000 included, we will have one exceeding 5000, 5001, 4999, 1, 0 two and any other value which would be a representative of the equivalence class may be 1000.

Example 2
- Consider a program that reads the "age" of employees and computes the average age.

$$ages \rightarrow \boxed{Program} \rightarrow average\ age$$

Assume valid age is 1 to 100

- How would you test this?
  - How many test cases would you generate?
  - What would be test data?

12

Now, let us look at another example - Given a function which reads the age of an employee, age of various employees, and computes their average age. So, this is the function which reads the age of various employees stored in a file and then computes the average age and prints out. So, pictorially we can have represent it as a program a black-box which takes all the ages of various employees, and prints out the average age. And let us assume that the valid age for employees is 1 to 100 or may be 18, 1 to 100. Now what would be the boundary value test cases for this, and how many test cases will be needed?

Boundaries of the inputs

$$1 <= \quad age \quad <= 100$$
$$1 \longleftarrow age \longrightarrow 100$$

The "basic" boundary value testing would include 5 situations:
1. - at minimum boundary
2. - immediately above minimum
3. - between minimum and maximum (nominal)
4. - immediately below maximum
5. - at maximum boundary

13

So, to answer this question, we need actually 5 test cases. One at the minimum of the boundary, one at the maximum, 1 and 100, one inside say that is 2, one just below 100 and one representative. And if we consider the invalid ones, we would also need to consider 101 and 0, so that would make it 7.

So valid for this equivalence class, we need to select 5, but if we consider the one line outside a boundary, and as we said earlier that outside boundary is a negative test case. So, if you consider negative test cases and we will have to also consider 101 and 0. So, five if we do not consider negative test cases; and if we consider negative test cases as well, we will need seven test cases for boundary value testing for this problem.

(Refer Slide Time: 14:55)

So, this just gives some sample values for this.

Now let's say we have two variables; one is years of education and age. These are 2 parameters which a function takes; years of education which can be 1 to 23, and age which is 1 to 100. So, what will be the boundary values? So, we have 2 parameters the input there are 2 input data one is a years of education and the other is age. If we consider them independently then we need 5 for the years of education, and 5 for the age. And therefore, we would need 10 test cases and then we need one which is a representative of this. And we can select it such that it is applies to both of these and therefore, we would need 11 if we consider the representative of this equivalence class.
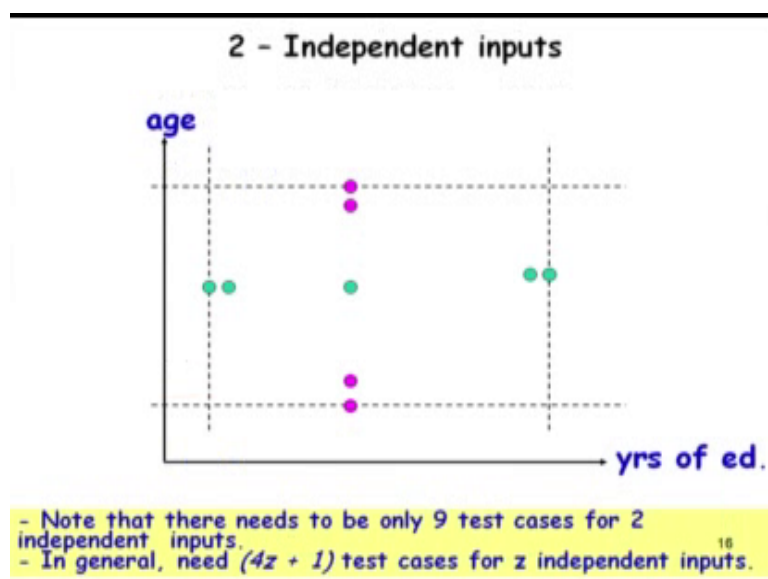
And if we represent it pictorially, so there are 2 boundaries for this variable; years of education and for age there are 2 boundaries. So, if we just consider the values within the boundary, which are the positive test cases we need 5 here and we need 5 here. Or we can consider that for each boundary if we have n parameters then we need 4*n+1 test cases, if we do not consider negative test cases.

(Refer Slide Time: 17:19)
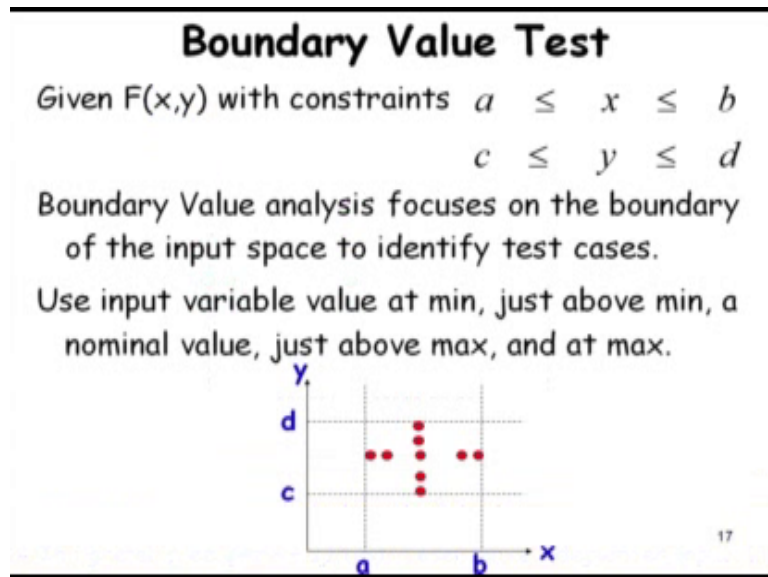
So, for n independent inputs, we need 4*n+1 test cases. So, we just have just represented that that there are two boundaries x and y; x has boundary between a and b and y has boundary between c and d, and therefore, we need 2*n+4 which is 9 test cases, so 4, 8 plus 1 – 9.

One assumption in what we said the way design the special value test cases for multiple parameters that at anytime one boundary can have a problem. But if it is possible that there is a situation where the error occurs when both boundaries have some value some specific combination of values, then we would have to consider all possible combinations between the different special values of the 2 parameters.

So the 4*n+1 result is true, if we consider that for a special value for one of the parameter the problem will be observed. But if this is the case that if only when both the parameters have some combination of values special values then only the problem occurs then we will have to consider all possible combinations, so that is 4 into 4, not 4 plus 4.
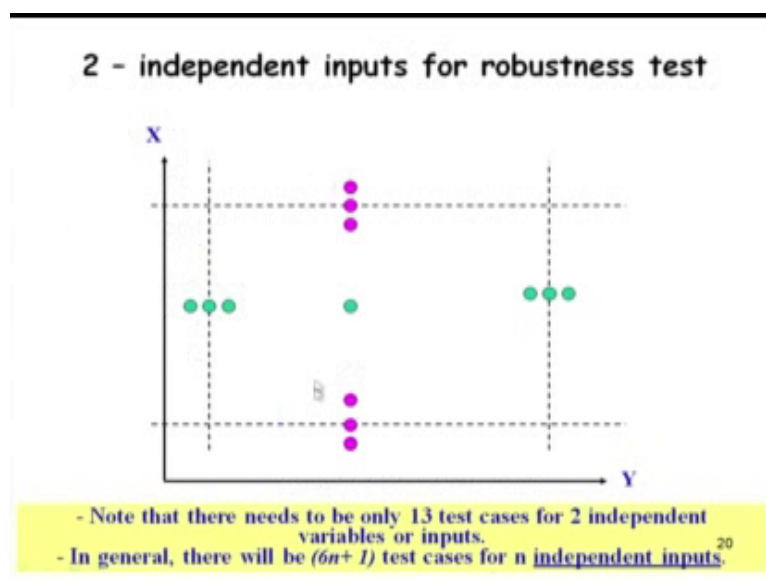
Robustness testing

- This is just an extension of the Boundary Values to include:
  - Less than minimum
  - Greater than maximum
- There are 7 cases or values to worry about for each independent variable input
- The testing of robustness is really a test of "error" handling.
  1. Did we anticipate the error situations?
  2. Did we issue informative error messages?
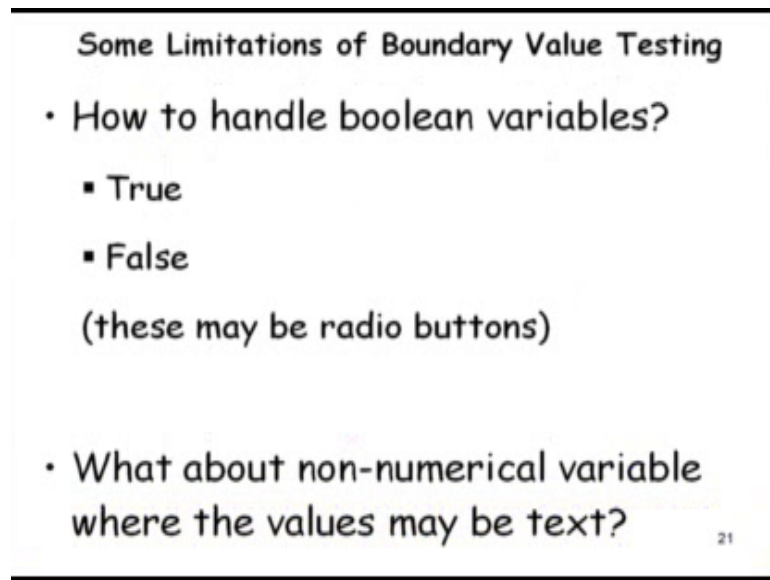 3. Did we allow some kind of recovery from the error? [19]

So, if we have to consider the negative test cases also, we call it as robustness testing. So, these are values outside the equivalence classes. And in this case, we need seven test cases for each variable. And in robustness testing, we check given a value which is not really a valid value, invalid value, does the programmer handle it, did he expect that such values can be given, did he issue an informative message that the value is not valid and please enter a valid value, and does it have some recovery or the programmer has to repeat all the actions. So just the last value is ignored or he would have to start fresh.

2 – independent inputs for robustness test

- Note that there needs to be only 13 test cases for 2 independent variables or inputs.
- In general, there will be $(6n+1)$ test cases for n independent inputs. [20]

So, for robustness test that is when we consider the negative test cases also, we need to gives a 6*n+1. So, for each variable three at both the boundaries, so that makes it 6; and for every boundary, we need 6. And therefore, if we have n equivalence classes, we need 6*n+1.

Some Limitations of Boundary Value Testing
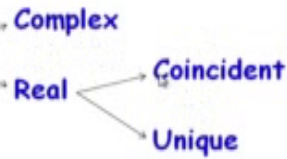- How to handle boolean variables?
  - True
  - False
  (these may be radio buttons)

- What about non-numerical variable where the values may be text?

But then what if we have Boolean variables, how do we handle it using this special value testing. Maybe these Boolean values are very common in user interfaces, where we have radio buttons either we mark it the radio button or unmark it. And what about a non-numerical variable, so we looked at only numerical boundaries what if it is a string. So, let us discuss these cases, how do we handle these special cases.

But before that let us have a small quiz which is about designing a black-box test suite for a function that solves a quadratic equation in the form ax^2+bx+c. And note that this is not asking to design the special value testing, we are asking to design a black-box test suite. So, in the black-box test suite, the first thing we have to consider is the equivalence class and as a hint if, if you look back at what we discussed. In equivalence class, we need to consider the scenarios and that gives us hint about what are the possible equivalence classes.

So, we have this function which takes three parameters a, b, c, which may be floating point numbers, and then displays the solution of this equation a x square plus b x plus c. It may take value let say 5.0, 7.0, 2.0, so that is the input to this function quad solver; so name of the function is quad solver, takes three parameters and displays the solution. So what would be the equivalence classes for this? If you think of it, we said that we need to consider the scenarios and what would be scenarios; the scenarios would be that the root is a coincident root.

The output may be the roots are coincident, and they are the same roots and they we might have distinct roots distinct and real roots, we might have imaginary roots. So, depending on what are the values of $b^2-4*a*c$, we might have different solutions different scenarios that might be occurring.

And what about the valid and invalid we said that first we have to consider the valid and invalid

classes. So, what about the valid - invalid input? So, what about the invalid input, the invalid input may be that both a, b, c are 0 so that is a invalid equivalence class. Or may be a, b, c are given as character strings, so we have to try out representative of those invalid equivalence classes as well.

And also we need to from the valid equivalence classes, we have coincident roots, we have distinct and real roots, and also imaginary roots. So, if we represent that we have the valid equations invalid equations so we have different equivalence classes, one is that a, b, c - 0 invalid equation or invalid input values; a b c are characters. And for valid equations, we might have we have real and complex and real the coincident and unique ( ). So, there are 3 equivalence classes here complex, coincident and unique.

So, we have to represent representative values of the input look considering b^2 - 4*a*c, when b^2 - 4*a*c you have to select values such that b^2 - 4*a*c is 0, positive and negative.

So far we looked at black-box testing techniques, equivalence testing and special value testing. One thing is that for simple problems, the concepts are can be applied in straight forward, but then we need lot of practice here. Please do problems for designing equivalence class tests and special value tests for problems that you can get for different situations. And we will stop here and we will continue with combinatorial testing.

Thank you.