



SE-3002

SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Part II-Software Testing

Functional Testing

Lecture # 13, 14, 15

20, 21, 22 Sep

TODAY'S OUTLINE

- Functional testing
- Boundary value analysis
- Equivalence class testing
- Revision

LIMITATIONS OF TESTING

- Test everything before giving the software to the customers.
- 'EVERYTHING'
 - Execute every true and false condition
 - Execute every statement of the program
 - Execute every condition of a decision node
 - Execute every possible path
 - Execute the program with all valid inputs
 - Execute the program with all invalid inputs
- **impossible to achieve due to time and resource constraints**

FUNCTIONAL TESTING

- Main intent of software testing is to search of such test cases which may make the software fail.
- Functional testing techniques attempt to design those test cases which have a higher probability of making a software fail.
- These techniques also attempt to test every possible functionality of the software.
- Test cases are designed on the basis of functionality and the internal structure of the program is completely ignored.



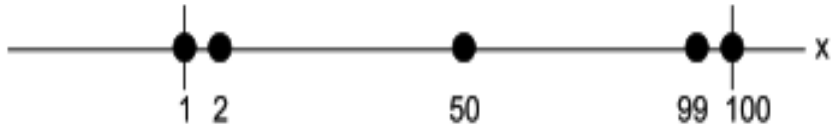
FUNCTIONAL TESTING

- Real life testing activities performed with only black box knowledge???
- In functional testing techniques, execution of a program is essential and hence these testing techniques come under the category of 'validation'.
- These techniques can be used at all levels of software testing like unit, integration, system and acceptance testing.

BOUNDARY VALUE ANALYSIS

- Popular functional testing technique that concentrate on input values and design test cases with input values that are on or close to boundary values.
- Write a program 'Square' which takes 'x' as an input and prints the square of 'x' as output. The range of 'x' is from 1 to 100.
- How to test this program???
- One possibility is to give all values from 1 to 100 one by one to the program and see the observed behavior.
- In boundary value analysis, we select values on or close to boundaries and all input values may have one of the following:
 - Just above minimum value
 - Minimum value
 - Maximum value
 - Just below maximum value
 - Nominal (Average) value

BOUNDARY VALUE ANALYSIS FOR SQUARE PROGRAM



Five values for input 'x' of 'Square' program

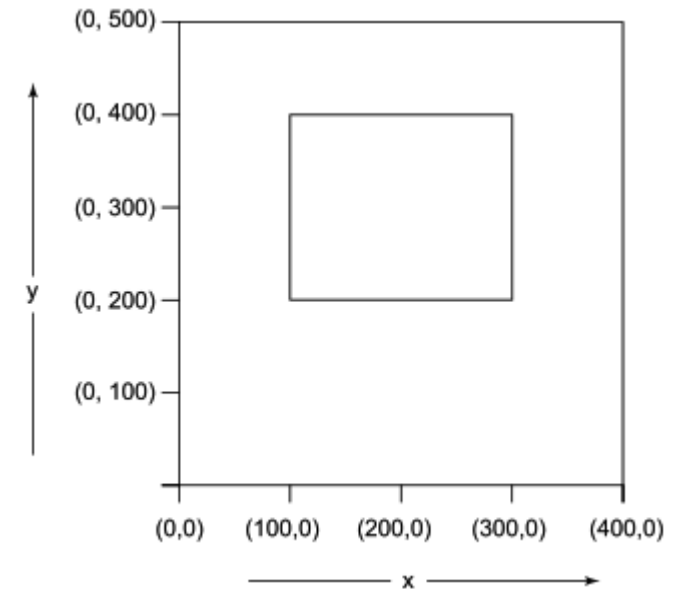
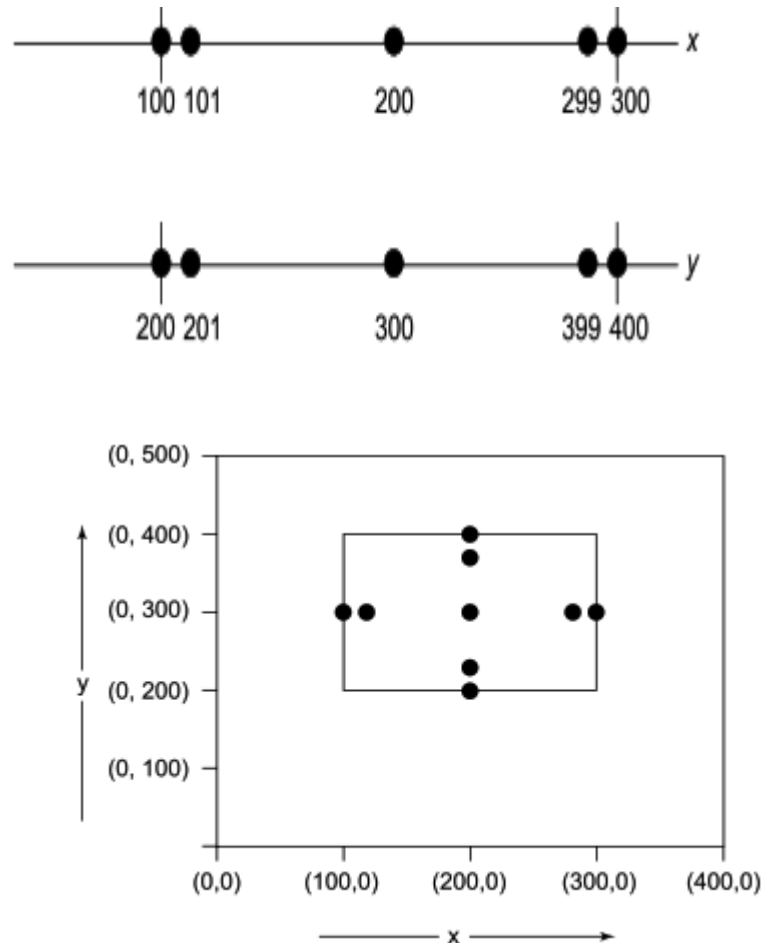
Test cases for the 'Square' program		
Test Case	Input x	Expected output
1.	1	1
2.	2	4
3.	50	2500
4.	99	9801
5.	100	10000

- The number of inputs selected by this technique is $4n + 1$ where 'n' is the number of inputs.

ANOTHER EXAMPLE

- Consider a program 'Addition' with two input values x and y and it gives the addition of x and y as an output. The range of both input values are given as:

- $100 \leq x \leq 300$
- $200 \leq y \leq 400$





Test cases for the program 'Addition'			
Test Case	x	y	Expected Output
1.	100	300	400
2.	101	300	401
3.	200	300	500
4.	299	300	599
5.	300	300	600
6.	200	200	400
7.	200	201	401
8.	200	300	500
9.	200	399	599
10.	200	400	600

APPLICABILITY

- This technique is suited to programs in which input values are within ranges or within sets and input values boundaries can be identified from the requirements.
- This is equally applicable at the unit, integration, system and acceptance test levels.
- This technique does not make sense for Boolean variables where input values are TRUE and FALSE only, and no choice is available for nominal values, just above boundary values, just below boundary values, etc.

EQUIVALENCE CLASS TESTING

- A large number of test cases are generated for any program. It is neither feasible nor desirable to execute all such test cases.
- We want to select a few test cases and still wish to achieve a reasonable level of coverage.
- We may divide input domain into various categories with some relationship and expect that every test case from a category exhibits the same behavior.
- If categories are well selected, we may assume that if one representative test case works correctly, others may also give the same results. This assumption allows us to select exactly one test case from each category and if there are four categories, four test cases may be selected.
- Each category is called an equivalence class and this type of testing is known as equivalence class testing.

CREATION OF EQUIVALENCE CLASSES

- The entire input domain can be divided into at least two equivalence classes: one containing all valid inputs and the other containing all invalid inputs.
- Each equivalence class can further be sub-divided into equivalence classes on which the program is required to behave differently.
- Square program

- (i) $I_1 = \{ 1 \leq x \leq 100 \}$ (Valid input range from 1 to 100)
- (ii) $I_2 = \{ x < 1 \}$ (Any invalid input where x is less than 1)
- (iii) $I_3 = \{ x > 100 \}$ (Any invalid input where x is greater than 100)

Test cases for program 'Square' based on input domain		
Test Case	Input x	Expected Output
I_1	0	Invalid Input
I_2	50	2500
I_3	101	Invalid Input

EQUIVALENCE CLASS TESTING

- A large number of test cases are generated for any program. It is neither feasible nor desirable to execute all such test cases.
- Select a few test cases and still wish to achieve a reasonable level of coverage.
- Divide input domain into various categories with some relationship and expect that every test case from a category exhibits the same behavior.
- If categories are well selected, we may assume that if one representative test case works correctly, others may also give the same results. This assumption allows us to select exactly one test case from each category and if there are four categories, four test cases may be selected.
- Each category is called an equivalence class and this type of testing is known as equivalence class testing.

CREATION OF EQUIVALENCE CLASSES

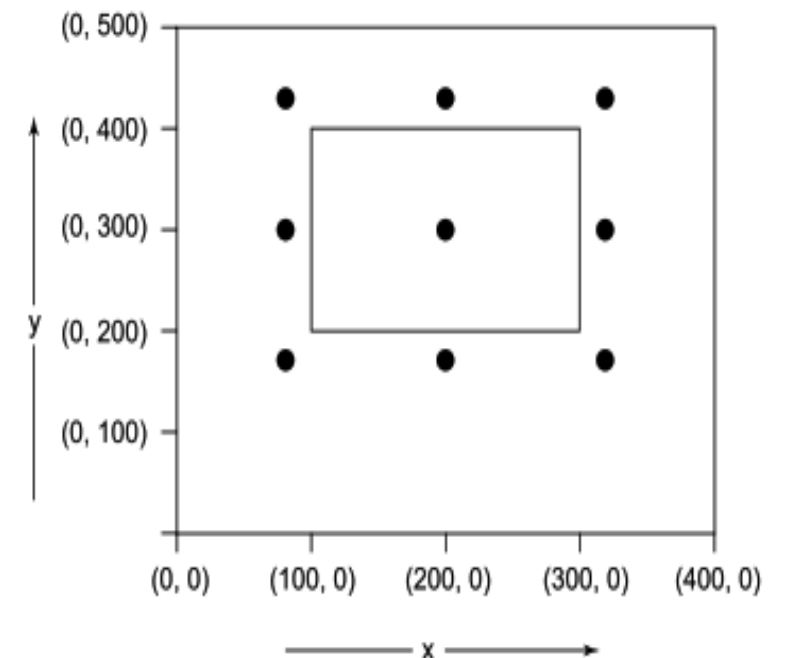
- The entire input domain can be divided into at least two equivalence classes: one containing all valid inputs and the other containing all invalid inputs.
- Each equivalence class can further be sub-divided into equivalence classes on which the program is required to behave differently.
- Square program

- (i) $I_1 = \{ 1 \leq x \leq 100 \}$ (Valid input range from 1 to 100)
- (ii) $I_2 = \{ x < 1 \}$ (Any invalid input where x is less than 1)
- (iii) $I_3 = \{ x > 100 \}$ (Any invalid input where x is greater than 100)

Test cases for program 'Square' based on input domain		
Test Case	Input x	Expected Output
I_1	0	Invalid Input
I_2	50	2500
I_3	101	Invalid Input

EQUIVALENCE CLASSES FOR ADDITION PROGRAM

- (i) $I_1 = \{ 100 \leq x \leq 300 \text{ and } 200 \leq y \leq 400 \}$ (Both x and y are valid values)
- (ii) $I_2 = \{ 100 \leq x \leq 300 \text{ and } y < 200 \}$ (x is valid and y is invalid)
- (iii) $I_3 = \{ 100 \leq x \leq 300 \text{ and } y > 400 \}$ (x is valid and y is invalid)
- (iv) $I_4 = \{ x < 100 \text{ and } 200 \leq y \leq 400 \}$ (x is invalid and y is valid)
- (v) $I_5 = \{ x > 300 \text{ and } 200 \leq y \leq 400 \}$ (x is invalid and y is valid)
- (vi) $I_6 = \{ x < 100 \text{ and } y < 200 \}$ (Both inputs are invalid)
- (vii) $I_7 = \{ x < 100 \text{ and } y > 400 \}$ (Both inputs are invalid)
- (viii) $I_8 = \{ x > 300 \text{ and } y < 200 \}$ (Both inputs are invalid)
- (ix) $I_9 = \{ x > 300 \text{ and } y > 400 \}$ (Both inputs are invalid)



TEST CASES FOR ADDITION

Test cases for the program 'Addition'			
Test Case	x	y	Expected Output
I ₁	200	300	500
I ₂	200	199	Invalid input
I ₃	200	401	Invalid input
I ₄	99	300	Invalid input
I ₅	301	300	Invalid input
I ₆	99	199	Invalid input
I ₇	99	401	Invalid input
I ₈	301	199	Invalid input
I ₉	301	401	Invalid input

APPLICABILITY

- Applicable at unit, integration, system and acceptance test levels.
- The basic requirement is that inputs or outputs must be partitioned based on the requirements and every partition will give a test case.
- If one test case catches a bug, the other probably will too. If one test case does not find a bug, the other test cases of the same equivalence class may also not find any bug.
- The design of equivalence classes is subjective and two testing persons may design two different sets of partitions of input and output domains.

EXAMPLE

- Consider a program for the determination of the largest amongst three numbers. Its input is a triple of positive integers (say x , y and z) and values are from interval $[1, 300]$.
- Design the boundary value test cases.
- Design the equivalence classes



That is all