Module 2

Functional Testing

Functional Testing

- Introduced by W.E Howden in late 1970s
- Also called Black box testing
- Involves testing a code / design without the knowledge of its internals (like the actual code structure, statements, design details etc.)
- Deals only with inputs and outputs applied to code / design / requirements
- Test Cases deal only with the inputs and outputs
- Program P - function that takes some inputs and produces outputs
  - Given inputs x_i, P computes outputs y_i

    such that $y_i = P(x_i)$
  - Eg: Sorting program
    - Input x_i - array of numbers
    - Output y_i - array in sorted order

Steps in Functional Testing

- Identify the domain of each i/p and each o/p variable
- Select values from domain of each variable having important properties
- Consider *combinations* of special values from different i/p domains to design testcases
- Consider input values such that the program under test produces special values from the domains of the output variables

Note: *Need to have minimal context information to obtain relevant values for inputs & outputs - knowing the context will help decide the right inputs to be given for the testcases*

Types of Functional Testing

- Boundary value analysis
- Equivalence class partitioning
- Decision tables
- Pair-wise testing
- Random testing

1^{st} TECHNIQUE - BOUNDARY VALUE TESTING

- Also called Input domain testing
- Based on the fact that input values near the boundary have higher chances of errors
- Boundary values:
  - Values lying on the boundary
  - Values just above the boundary
  - Values just below the boundary
- Example: 1 to 10 [range]
- Consider 2 variables $a \leq x \leq b$ and another set $c \leq y \leq d$
- X varying from 1 to 100

- Boundary value Analysis is a testing technique which aims to detect errors related to boundary values

    - Range related errors
    - Syntactically there may be no errors
    - Logical operators are prone to generate such errors
    - Identify lower limit value and upper limit value
    - Hence, check for UL-1, UL, UL+1, LL-1, LL, LL+1 (UL – Upper limit and LL – lower limit)

- Areas of applications such as string, banking transactions etc.
- Saves time in testing

BOUNDARY VALUE ANALYSIS

- Based on experience / heuristics:
  - Testing boundary conditions of equivalence classes is more effective, i.e. values directly on, above, and beneath edges of classes
  - If a system behaves correctly at boundary values, than it probably will work correctly at "middle" values
- Choose input boundary values as tests in input classes instead of, or additional to arbitrary values
- Choose also inputs that invoke output boundary values (values on the boundary of output classes)
- Example strategy as extension of equivalence class partitioning:
  - choose one (or more) arbitrary value(s) in each eq. class
  - choose values exactly on lower and upper boundaries of eq. class
  - choose values immediately below and above each boundary (if applicable)

- Thus, BVA includes for testing, the test inputs to be values lying on the boundary
- Value just above the boundary value (lower limit range)
- Value just below the boundary value (upper limit range)

BOUNDARY VALUE ANALYSIS

- Boundary conditions are those situations at the edge of the planned operational limits of the software
- Test valid data just inside the boundary, test the last possible data and test the invalid data just outside the boundary

- Boundary types: Numeric, Character, Position, quantity, Speed, Location, Size….
- Think of First / Last, Min / Max, Start / Finish, Over / Under, Empty / Full, Slowest / Fastest, Soonest / Latest

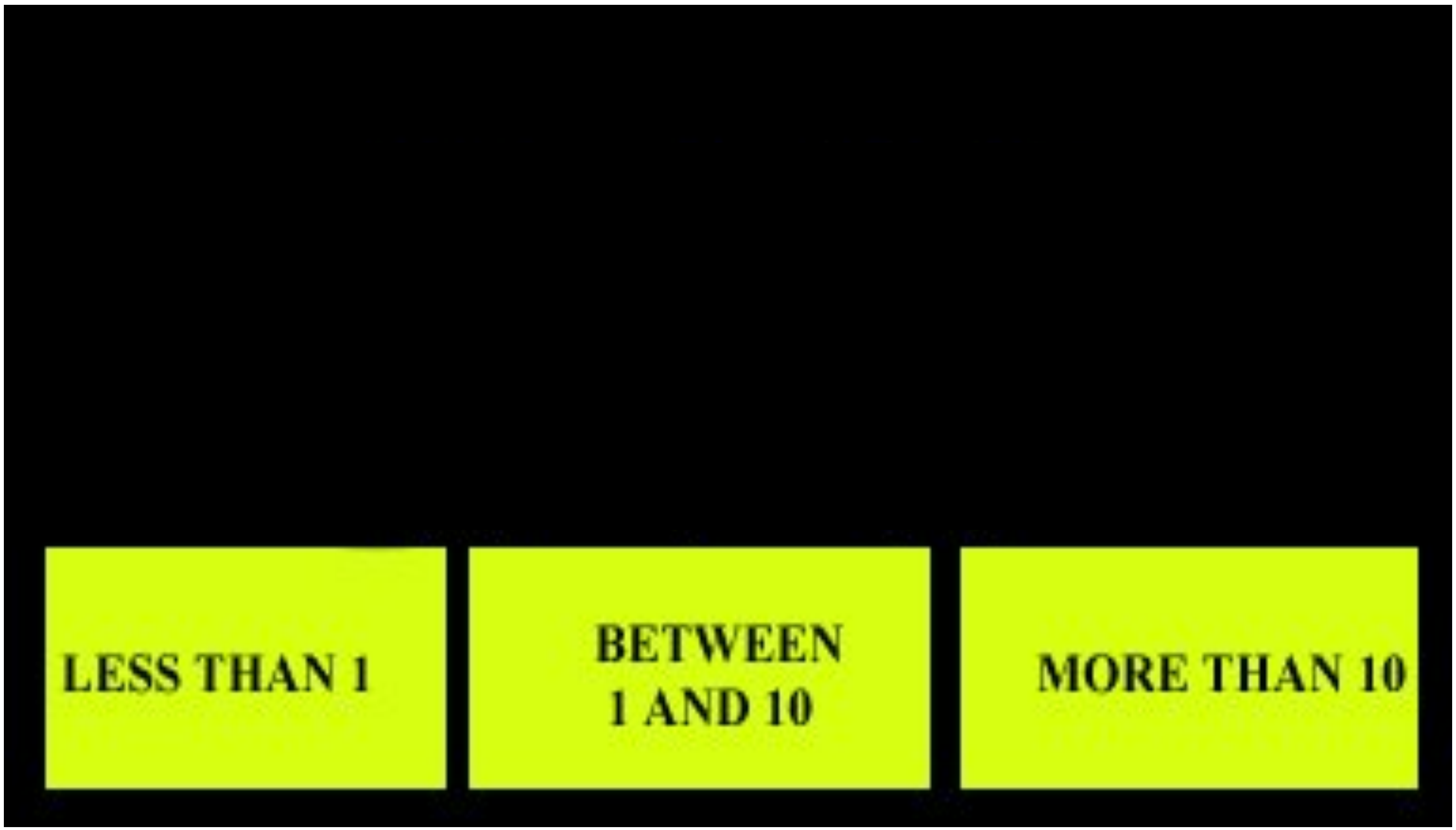
BOUNDARY VALUE BASIC EXAMPLE

- Let’s suppose that you wanted to test a program that only accepted integer values from 1 to 10
  - The possible test cases for such a program would be the range of all integers

- How can we narrow the number of test cases down from all integers to a few good test cases?

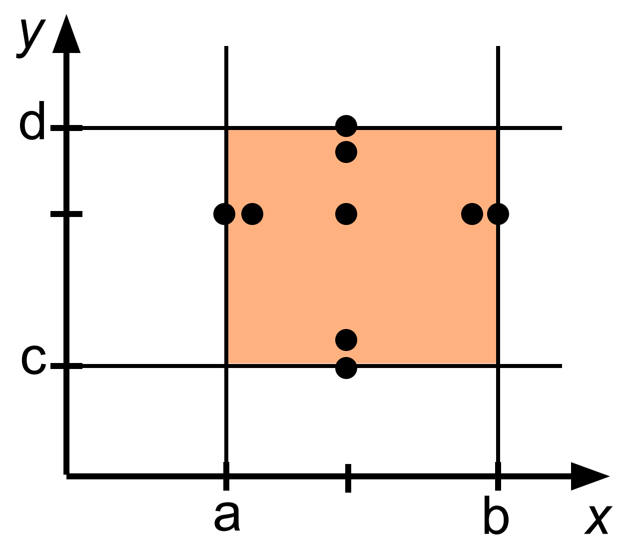
- What are boundary values?
  - Values lying on the boundary (upper and lower limit) -- 1 and 10)
  - Values just above the boundary (lower boundary -- 2, 2.5 1.5)
  - Values just below the boundary (Upper boundary , 9, 9.5)

SAMPLE BOUNDARY VALUE



BOUNDARY VALUE TESTING

- Detect errors to do with input domain bounds
- For integer input x with domain [a,b], test input values around a and b
- # tests = for *n* inputs, 4*n*+1 input combinations
  - Min, min+, nominal, max-, max values
- Assumes:
  - Independent quantity inputs
  - Single-fault assumption
  - At any point of time only one variable can be at fault and not all variables
  - Eg: if x is in boundary then y has to be a nominal value for 2 input variables



SINGLE VALUE ASSUMPTION

- Consider x values y values , (x, y) where x=100 and y =300,
- If x is in boundary state then y cannot be faulty hence x is boundary then y should be a nominal / middle value / assumed correct value
    - (100, 200)
    - (101, 200)
    - (50, 200)
    - (299, 200)
    - (300, 200)
  - For y, where y is in boundary then x should be a nominal value
    - (200, 100)
    - (200, 101)
    - (200, 50) (Repeated – Hence, written for in one case)
    - (200, 299)
    - (200, 300)
    - Hence test cases should have 4n+1 = 9 test cases

Example 2

- Consider a program for determining the previous (next date. Input: day, month, year, with valid ranges as
- $1 \leq$ month ≥ 12
- $1 \leq$ day ≥ 31
- $2000 \leq$ year ≥ 2020

- Solution: Let us assume 1, 12, 1, 31 and 1900, 2000 as boundary values
- Since, there are 3 variables, we should have 4(3) + 1 = 13 Test Cases
- 1 variable in each case, to be at boundary and other 3 variables to be nominal values
- **Apply (4n+1) formula to generate the total number of test cases**

- Possible outputs – Previous date or Invalid date

Test case	Month	Day	Year	Expected output	Actual Output
001			1900 (min)		
002			1901(min+1)		
003			1960 (nom)		
004			1999 (max-1)		
005			2000(max+1)		
	6	15		14th June 1900	
	6	15		14th June 1901	
	6	15		14th June 1960	
	6	15		14th June 1999	
	6	15		14th June 2000	
006	6	1	1960	31st May 1960	
007	6	2	1960	1st may 1960	
008	6	30	1960	29th June 1960	
009	6	31	1960	30th June 1960	
010	1	15	1960	14th Jan 1960	
011	2	15	1960	1st Jan 1960	
012	11	15	1960	14th Nov 1960	
013	12	15	1960	14th Dec 1960	

- Possible outputs – Next date or Invalid date

Test case	Month	Day	Year	Expected output	Actual Output
001			2000 (min)		
002			2001(min+1)		
003			2010 (nom)		
004			2019 (max-1)		
005			2020(max)		
	6	15	2000 (min)	16/06/2000	
	6	15	2001(min+1)	16/06/2001	
	6	15	2010 (nom)	16/06/2010	
	6	15	2019 (max-1)	16/06/2019	
	6	15	2020(max)	16/06/2020	
006	6	1	2010	02/06/2010	
007	6	2	2010	03/06/2010	
008	6	30	2010	01/07/2010	
009	6	31	2010	Invalid input	
010	1	15	2010	16/01/2010	
011	2	15	2010	16/02/2010	
012	11	15	2010	16/11/2010	
013	12	15	2010	16/12/2010	

Example 3

- Consider an example of triangle where the condition is sum of 2 sides cannot be greater than the third side, write test cases which will consider BVA
- Note: If sum of any 2 sides is greater than the third side, then indicate as Not a Triangle
- Also, consider Single Fault Assumption

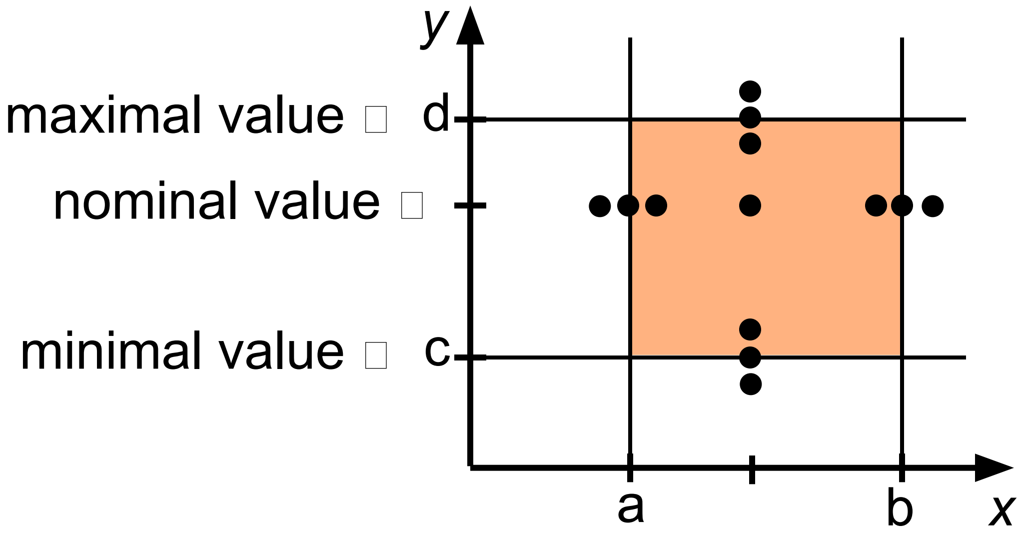
- Solution: Let us assume 1 and 100 as boundary values
- Since, there are 3 variables, we should have 4(3) + 1 = 13 Test Cases
- 1 variable in each case, to be at boundary and other 2 variables to be nominal values

- Sum of two sides should not be greater than third side

Test case	X	Y	Z	Expected output	Actual Output
001	50	50	1	Isosceles	
002	50	50	2	Isosceles	
003	50	50	50	Equilateral	
004	50	50	99	Isosceles	
005	50	50	100	Not a triangle	
006	50	1	50	Isosceles	
007	50	2	50	Isosceles	
008	50	99	50	Isosceles	
009	50	100	50	Not a triangle	
010	1	50	50	Isosceles	
011	2	50	50	Isosceles	
012	99	50	50	Isosceles	
013	100	50	50	Not a triangle	

ROBUSTNESS BOUNDARY VALUE

- Also test values outside the domain
- # tests =
- For *n* input variables, (**6n+1)** input combinations
- A test tuple: <*x*nom, *y*max+1, expected output>



6n +1 NUMBER OF TEST CASES

Test case	X	Y	Z	Expected output	Actual Output
001	50	50	0 (min-1)	Invalid	
002	50	50	1(min)	Isosceles	
003	50	50	2(min+1)	Isosceles	
004	50	50	50(Nom)	Equilateral	
005	50	50	99(max-1)	Isosceles	
006	50	50	100(max)	Not a triangle	
007	50	50	101(max+1)	Invalid	
008	50	0	50	Invalid	
009	50	1	50	Isosceles	
010	50	2	50	Isosceles	
011	50	99	50	Isosceles	
012	50	100	50	Not a triangle	
013	50	101	50	Invalid	
014	0	50	50	Invalid	
015	1	50	50	Isosceles	
016	2	50	50	Isosceles	
017	99	50	50	Isosceles	
018	100	50	50	Not a triangle	
019	101	50	50	Invalid	

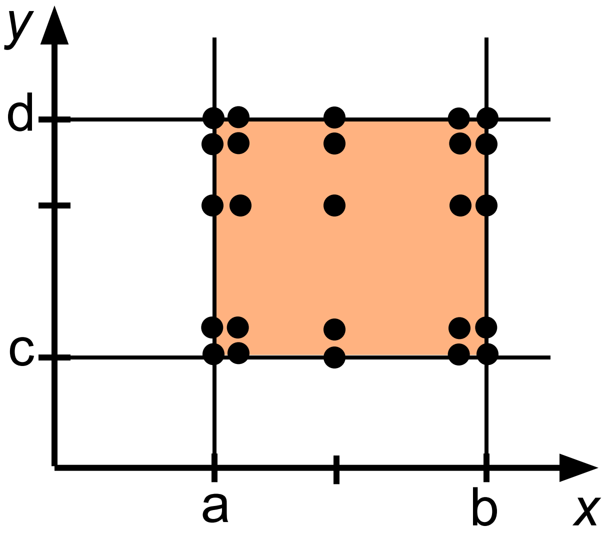
Next date Example – Single Fault Assumption (6n+1) number of Test cases

Test case	Month (1-12)	Day (1-31)	Year (2000-2020)	Expected output	Actual Output
001	6	15	1999 (min-1)	Invalid	
002	6	15	2000 (min)	14th June 2000	
003	6	15	2001(min+1)	14th June 2001	
004	6	15	2010 (nom)	14th June 2010	
005	6	15	2019 (max-1)	14th June 2019	
006	6	15	2020 (max)	14th June 2020	
007	6	15	2021(max+1)	Invalid	
008	6	0	2010	Invalid	
009	6	1	2010	May 31st 2010	
010	6	2	2010	1st May 2010	
011	6	30	2010	29th June 2010	
012	6	31	2010	30th June 2010	
013	6	32	2010	Invalid	
014	0	15	2010	Invalid	
015	1	15	2010	31st December 2009	
016	2	15	2010	1st Jan 2010	
017	11	15	2010	14th Nov 2010	

WORST-CASE BOUNDARY VALUE

- **Multiple-fault assumption**
- # tests =
-     for n input variables, 5^n input combinations
- Worse Case Testing is an extension of BVA without Single Fault Assumption. Hence, it is also treated as BVA to be a proper subset of worse case testing

- Hence, total no of test cases will be 5^n combinations where n is the number of input variables



2 Examples

- Generate test cases for Triangle problem and next date function

- Solution: Both cases, has 3 variables hence, number of test cases will be 125 test cases

- Hence, consider,
  - x, y constant and vary z
  - Similarly, keep x, z constant and vary y
  - Keep y, z constant and vary x

- Thus, it is Cartesian product

- For case when x=1, then y = (1,2,50, 99, 100),
- for x= 2, y = (1,2,50, 99, 100),
- for x= 50, y = (1,2,50, 99, 100),
- for x= 99, y = (1,2,50, 99, 100),
- for x= 100, y = (1,2,50, 99, 100),

- Similarly, consider x with z
- Then y will take values having x and z
- Lastly, z will take values having x and y constant

- Do the similar exercise for next date example

Example for Worst Case BVA for variable x and y ranging from 100 to 300 - Number of Test cases is 5^n

Test case	X	Y	Expected Output	Actual Output
001	100	100		
002	100	101		
003	100	200		
004	100	299		
005	100	300		
006	101	100		
007	101	101		
008	101	200		
009	101	299		
010	101	300		
011	200	100		
012	200	101		
013	200	200		
014	200	299		
015	200	300		
016	299	100		
017	299	101		
018	299	200		
019	299	299		
020	299	300		
021	300	100		
022	300	101		
023	300	200		

- It is also an extended version of BVA.
- It ALSO conducts tests for cases when extreme values are exceeded with values slightly greater than the maximum and a value slightly less than minimum
- Hence, it is a stronger version of BV testing.
- Hence, move beyond the valid input domain range
- Thus, robustness testing will contain (6n+1) test cases.

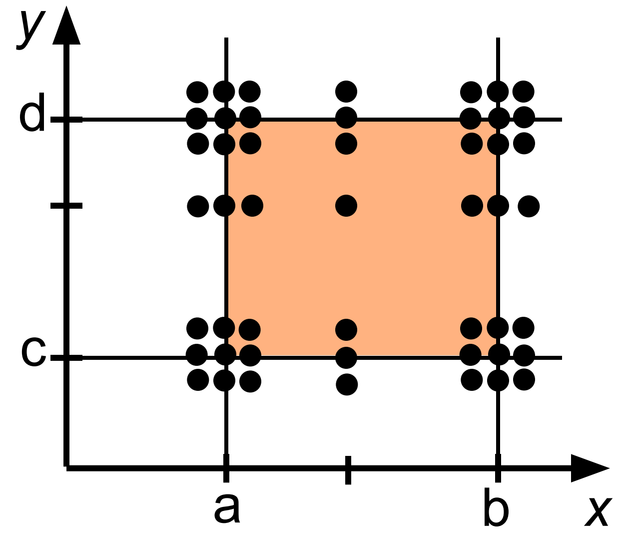
Examples

- Generate test cases with robustness testing for triangle problem and next date problem

- Solution: Both cases will have 6(3) + 1 test cases = 19 Test cases

WORST-CASE ROBUSTNESS BV

- Multiple-fault assumption, tests also outside the domain
- # tests =
- for n input variables, 7^n input combinations



SUBSUME RELATIONS

A □ B Technique A subsumes technique B if A tests at least what B tests (possibly more)

Which subsume relations for boundary value variants?

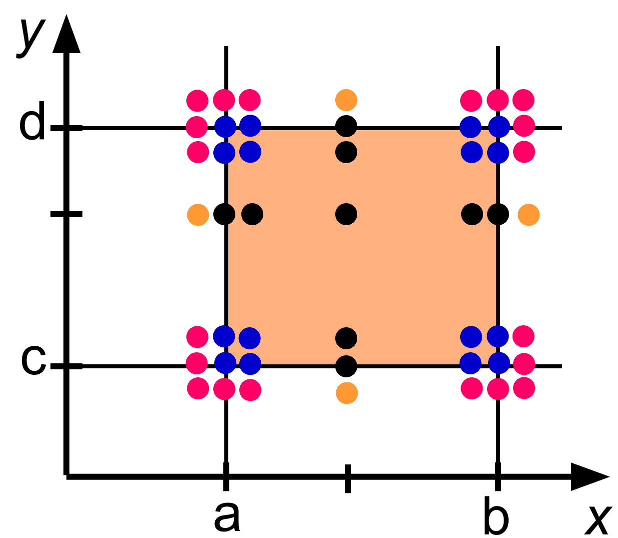
(Assume one fixed nominal value for each input)

boundary value

worst-case

robustness

worst-case robustness



BOUNDARY VALUE TESTING SUMMARY

- Coverage: not good
- #tests: moderate to very many
- Usage: straightforward, easy to implement

- When to use:
  - independent inputs
  - enumerable quantities, e.g. age
  - (obviously) when suspecting boundary errors

- See literature:
  - Patton (chapter 5, pages 70-74)
  - Jorgensen (chapter 5)
  - Zhu et al. (section 4.3)