Domain Testing

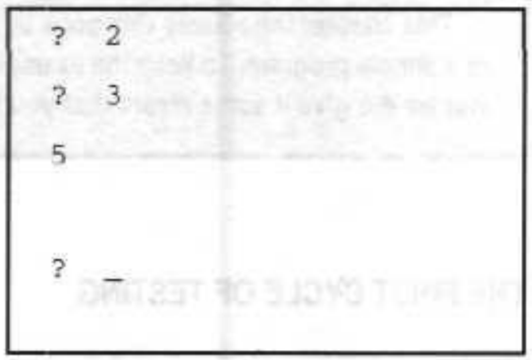
Tran Thi Bich Hanh

Faculty of Information Technology
University of Science, HCMC

Exercise 1

- The program's specification
  - This program is designed to add 2 numbers, which you will enter
  - Each number should be one or two digits

Figure 1.2 How the screen looks after the first test



The cursor (beside the question mark at the bottom of the screen) shows you where the next number will be displayed.

Possible Test Cases

- Valid Cases: 199 x 199 = 39,601
  - -99 → -1
  - 0 → 99
- Invalid Cases: INFINITE
  - <= -100
  - >= 100
  - Not a number

The problem

very large or infinite

number of test scenarios

+

finite amount of time

=

impossible to test everything

The solution

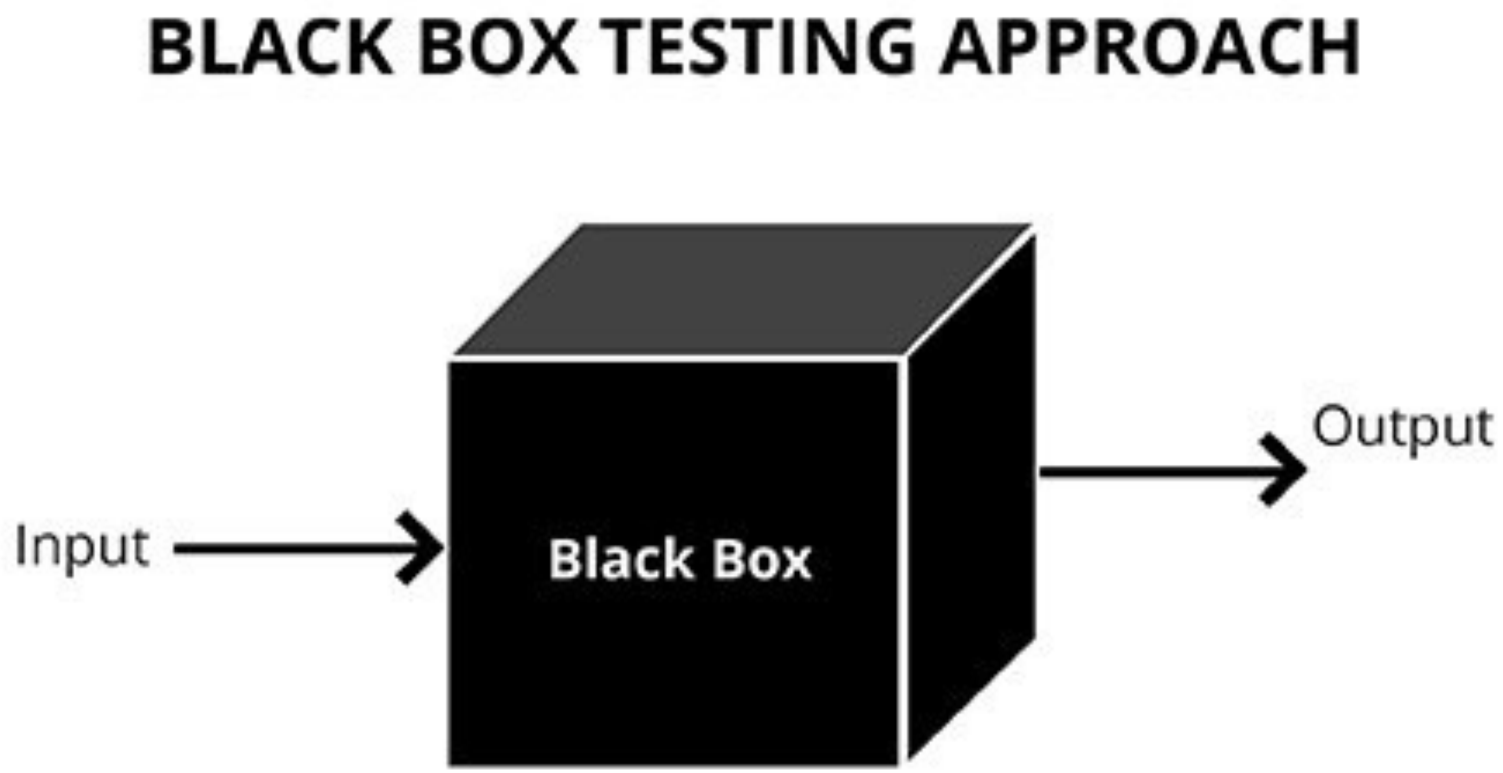
Software testing strategies and methods (techniques) exist to

reduce the number of tests to be run

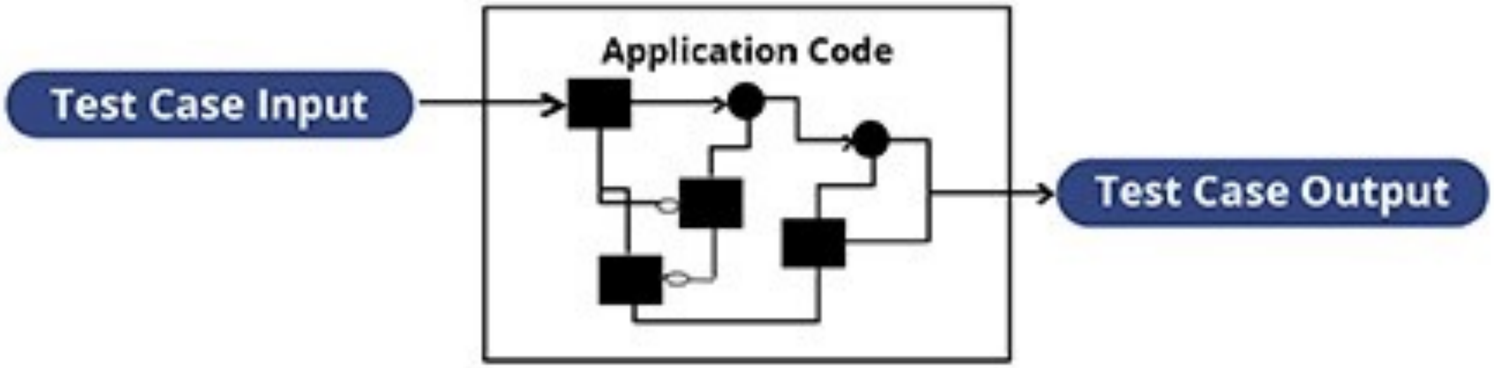
whilst still providing sufficient coverage

of the system under test

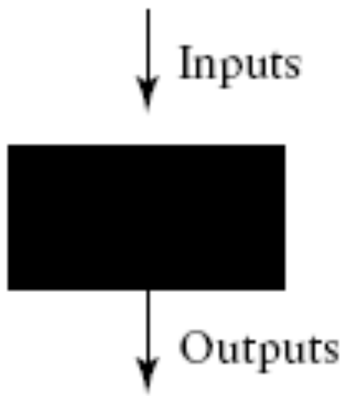
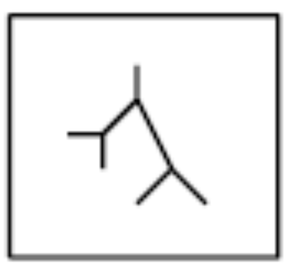
Testing Approaches


BLACK BOX TESTING APPROACH
Input
Black Box
Output

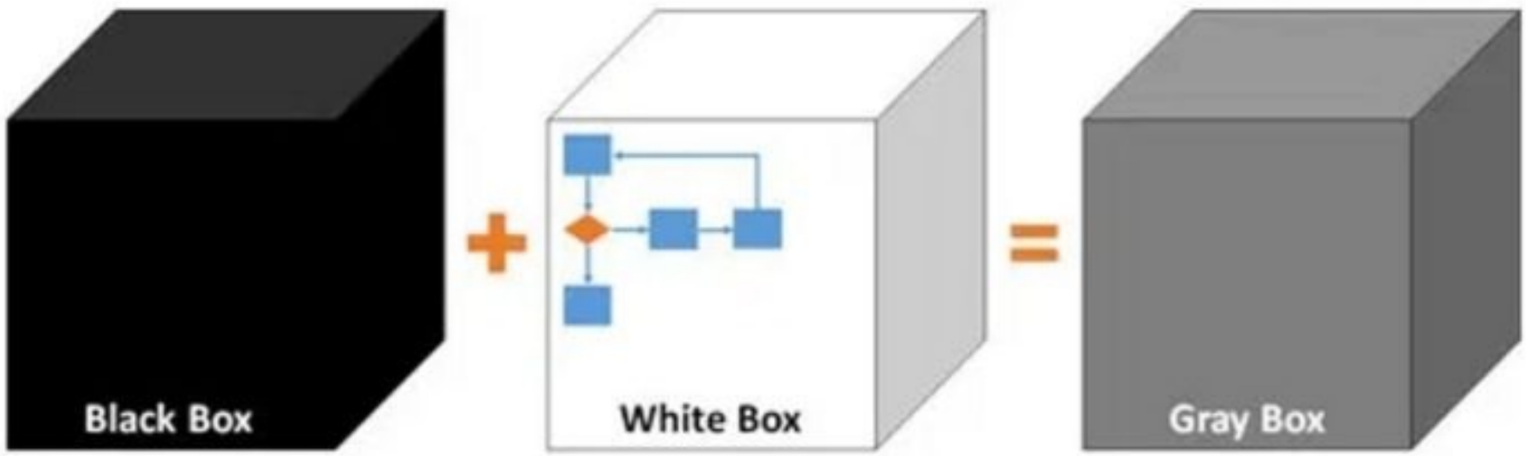
WHITE BOX TESTING APPROACH



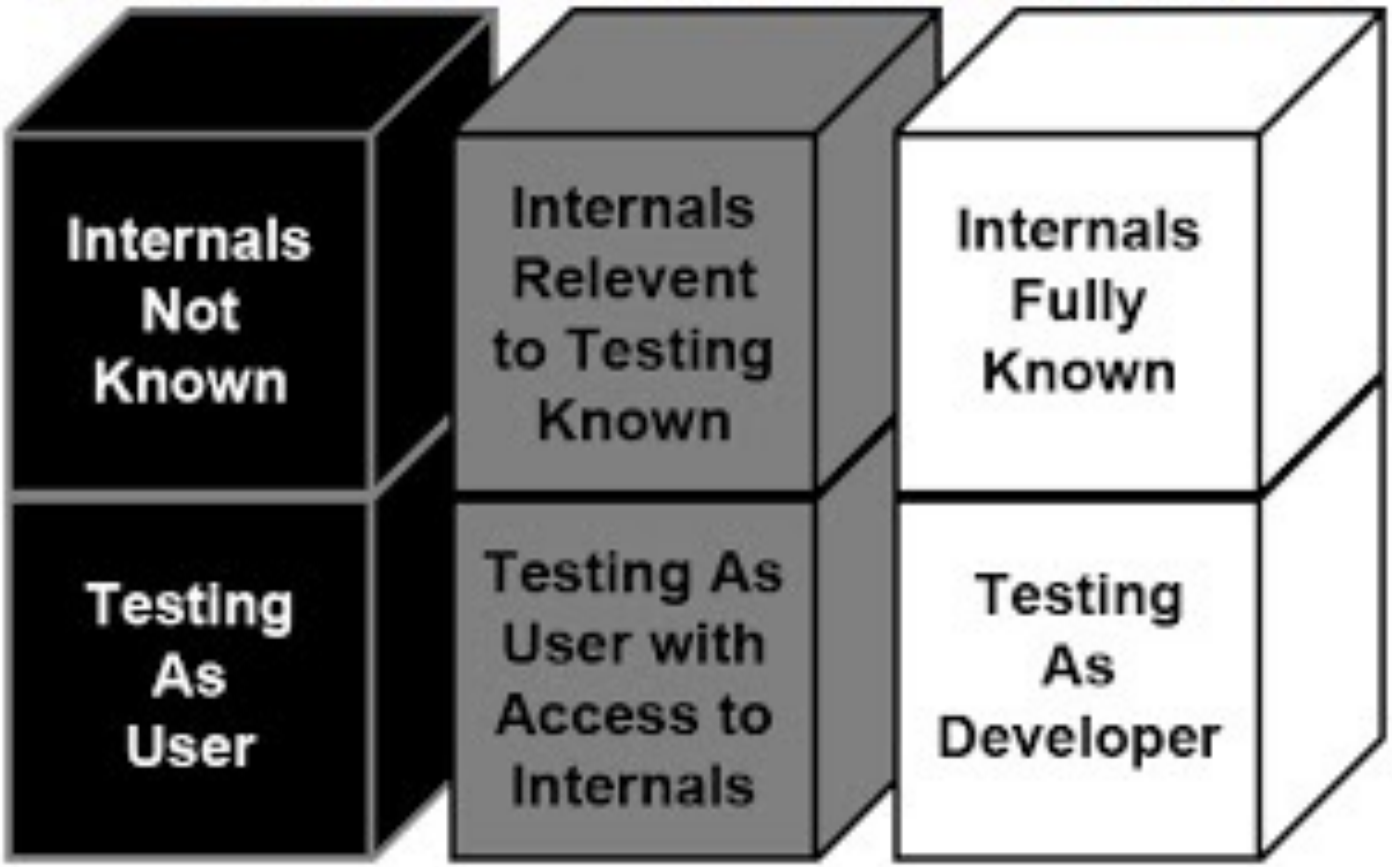
The two basic testing strategies

Test Strategy	Tester's View	Knowledge Sources	Methods
Black box		Requirements document Specifications Domain knowledge Defect analysis data	Equivalence class partitioning Boundary value analysis State transition testing Cause and effect graphing Error guessing
White box		High-level design Detailed design Control flow graphs Cyclomatic complexity	Statement testing Branch testing Path testing Data flow testing Mutation testing Loop testing

Grey Box Testing



Black – Grey – White Box Testing



Domain Testing

- EQUIVALENCE partitioning, equivalence analysis, boundary analysis
- Fundamental question or goal:
  - This confronts the problem that there are too many test cases for anyone to run. This is a stratified sampling strategy that provides a rationale for selecting a few test cases from a huge population.

In domain testing, we partition a domain into sub-domains (equivalence classes) and then test using values from each sub-domain.

General Approach

- 4 steps

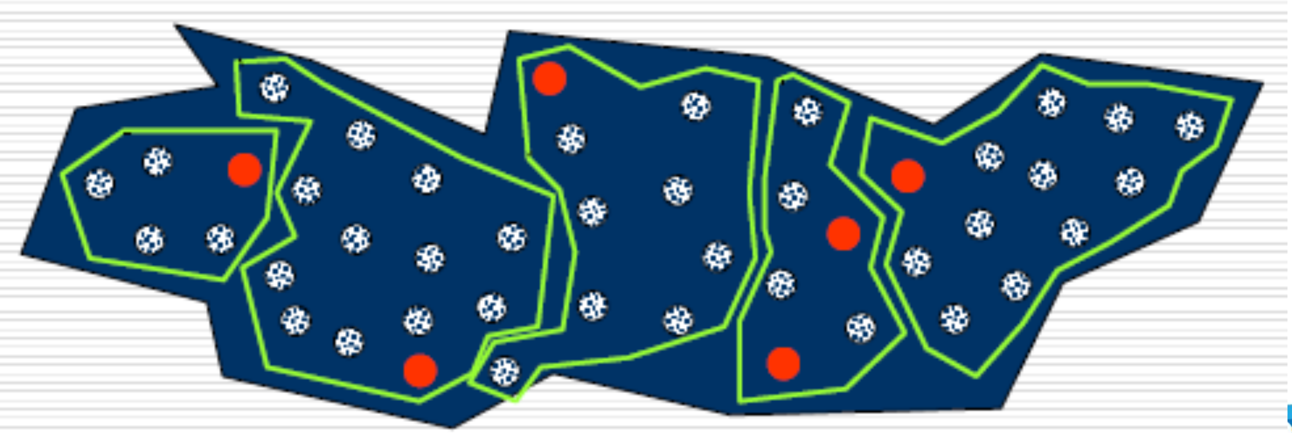
1. Identify Input & Output variables
2. Identify equivalence classes for each Input & Output
   1. Divide the set of possible values (domain) of the field into subsets (sub-domains – equivalence classes)
3. Find a “best representative” for each subset
4. Best representatives of ordered fields will typically be boundary values

Step 1. Identify Input & Output

- Based on program specification
- Ex: The program adds 2 numbers
  - Input: 2 numbers
    - A
    - B
  - Output:
    - SUM
    - Error message (Invalid Input)

Step 2. Identify Equivalence Classes

- Two tests belong to the same equivalence class if the expected result of each is the same.
- Executing multiple test cases of the same equivalence class is by definition, redundant testing.



Step 2. Identify Equivalence Classes

- Based on input/output conditions
- Equivalence classes
  - VALID EQUIVALENCE CLASSES are chosen to represent valid inputs.
  - INVALID EQUIVALENCE CLASSES are chosen to represent invalid inputs.

Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Has 1 or 2 digits	-99 <= Number <= 99	< -99 or > 99
Is a number	Is a number	Not a number
Output	SUM	Invalid Input

Complete Set of Partitions

STT	Input / Output	Equivalence Classes
EC1	A	-99 <= A <= 99
EC2		A < -99
EC3		A > 99
EC4		A is not an integer
EC5	B	-99 <= B <= 99
EC6		B < -99
EC7		B > 99
EC8		B is not an integer
EC9	SUM	= A+B
EC10		Error Message

Guidelines

- If an input condition specifies a range of value
  - E.g. “the item count can be from 1 to 999”
- Identify one valid equivalence class
  - 1 <= count <= 999
- and two invalid equivalence classes
  - Count < 1 and count > 999

Guidelines

- If an input condition specifies a set of input values and there is reason to believe that each is handle differently by the program
  - E.g. “type of vehicle must be BUS, TRUCK, TAXI-CAB, PASSENGER or MOTOCYCLE”
- Identify a valid equivalence class for each
- and one invalid equivalence classes
  - TRAILER

Guidelines

- If an input condition specifies a "must be" situation
  - E.g. "first character of the identifier must be a letter"
- Identify one valid equivalence class
  - It is a letter
- and one invalid equivalence classes
  - It is not a letter

Guidelines

- If there is any reason to believe that elements in an equivalence class are not handled in an identical manner by the program, split the equivalence class into two or more smaller equivalence classes

Example

- Enter a positive integer less than 100
  - C1: is an integer
    - EC1: is an integer, valid
    - EC2: not an integer, invalid
  - C2: [0, 100)
    - EC3: 0 <= x <100, valid
    - EC4: X < 0, invalid
    - EC5: x >= 100, invalid
  - Valid
    - Is an integer, 0 <= X < 100
  - Invalid
    - Is an integer, X < 0
    - Is an integer, X >= 100
    - Not an integer (0 <= x < 100, x < 0, x>=100)

Example (cont.)

- A string of 7 characters, the first charater must be upper-case
  - Valid
    - Length = 7, first charater is upper-case
  - Invalid
    - Length = 7, first charater is lower-case
    - Length < 7
    - Length > 7

Example (cont.)

- Coordinate point (X,Y):

  3 <= X <= 7, 5 <= Y <= 9

  - Valid
    - 3 <= X <= 7, 5 <= Y <= 9
  - Invalid
    - X < 3
    - X > 7
    - Y < 5
    - Y > 9

Example (cont.)

- Testing a module that allows a user to **enter** new widget **identifiers into** a widget **data base**.
- The input specification for the module states that a widget **identifier should consist of 3–15 alphanumeric characters of which the first two must be letters.**

Input Conditions

1. It must consist of alphanumeric characters
2. The range for the total number of characters is between 3 and 15
3. The first two characters must be letters.

Equivalence Classes

- Condition 1: the “must be” case for alphanumeric characters
  - EC1. The widget identifier is alphanumeric, valid.
  - EC2. The widget identifier is not alphanumeric, invalid.

Equivalence Classes

- Condition 2: the range of allowed characters 3–15.
  - EC3. The widget identifier has between 3 and 15 characters, valid.
  - EC4. The widget identifier has less than 3 characters, invalid.
  - EC5. The widget identifier has greater than 15 characters, invalid.

Equivalence Classes

- Condition 3: the "must be" case for the first two characters.
  - EC6. The first 2 characters are letters, valid.
  - EC7. The first 2 characters are not letters, invalid.

Equivalence Classes

- Valid
  - The widget identifier is alphanumeric, has between 3 and 15 characters, and the first 2 characters are letters
- Invalid
  - The widget identifier is not alphanumeric
  - The widget identifier has less than 3 characters
  - The widget identifier has greater than 15 characters
  - The first 2 characters are not letters

Step 3. Selecting test cases

- Choose at least one test case from each equivalence class
- For valid classes, choose test cases to cover as many equivalence classes as possible, until all valid classes have been covered
- For invalid classes, choose test cases so that each covers one and only one invalid class, until all classes are covered

Test Cases Providing Coverage of Partitions

#Partition Tested		Input 1 (A)	Input 2 (B)	Expected Output
EC1	-99 <= A <= 99	10	9	19
EC2	A < -99	-102	9	Invalid Input
EC3	A > 99	102	9	Invalid Input
EC4	A is not an integer	Abc	9	Invalid Input
EC5	-99 <= B <= 99	10	9	19
EC6	B < -99	10	-200	Invalid Input
EC7	B > 99	10	200	Invalid Input
EC8	B is not an integer	10	1.25	Invalid Input
EC9	SUM = A+B	10	9	19
EC10	Invalid Input	-102	9	Invalid Input

Minimum Set of Test Cases

#TC	Partitions Tested	Input 1 (A)	Input 2 (B)	Expected Output
TC1	EC1. -99 <= A <= 99 EC5. -99 <= B <= 99 EC9. SUM = A+B	10	9	19
TC2	EC2. A < -99 EC10. Invalid Input	-102	9	Invalid Input
TC3	EC3. A > 99	102	9	Invalid Input
TC4	EC4. A is not an integer	Abc	9	Invalid Input
TC5	EC6. B < -99	10	-200	Invalid Input
TC6	EC7. B > 99	10	200	Invalid Input
TC7	EC8. B is not an integer	10	1.25	Invalid Input

Step 4. Boundary Value Analysis

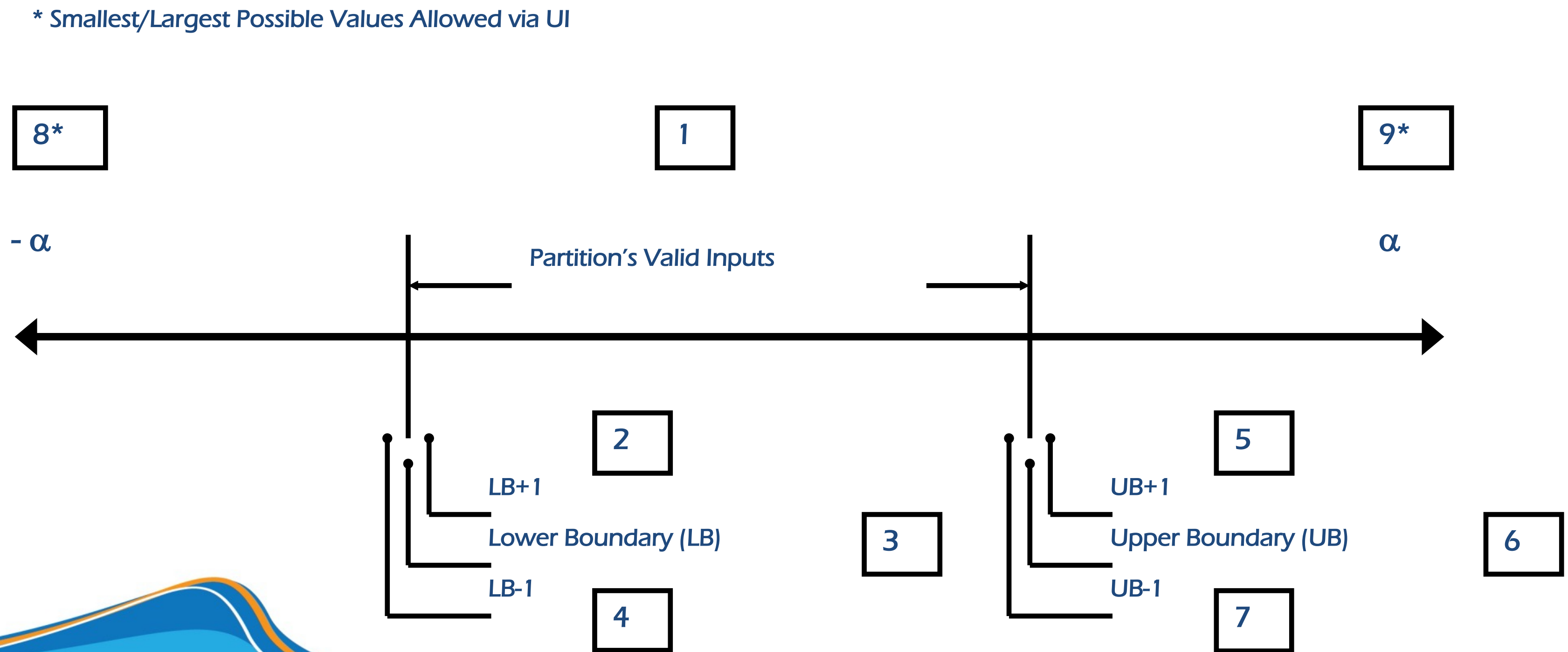
- The program is more likely to fail at a boundary?
  - ***Suppose program design***:
    - INPUT < 10 result: Error message
    - 10 <= INPUT < 25 result: Print "hello"
    - 25 >= INPUT result: Error message
  - ***Some error types***
    - Inequalities mis-specified (e.g. INPUT <= 25 instead of < 25)
      - Detect only at boundary
    - Boundary value mistyped (e.g. INPUT < 52, transposition error)
      - Detect at boundary and any other value that will be handled incorrectly

Boundary or Non-Boundary ?

- Boundary values (here, test at 25) catch ALL two errors
- Non-boundary values (consider 53) may catch NONE of the two errors

Boundary Value Analysis

- For each equivalence class partition, we'll have at most, 9 test cases to execute.
- It is essential to understand that each identified equivalence class represents a specific risk that it may pose.



Boundary Value Test Cases

#TC	Partition Tested	Input 1 (A)	Input 2 (B)	Expected Output
TC1	A < -99	-100	9	Invalid Input
TC2	-99 <= A <= 99	-99	9	90
TC3		-98	9	89
TC4		98	9	107
TC5		99	9	108
TC6	A > 99	100	9	Invalid Input
TC7	B < -99	-10	-100	Invalid Input
TC8	-99 <= B <= 99	10	-99	89
TC9		10	-98	88
TC10		10	98	108
TC11		10	99	109
TC12	B > 99	10	100	Invalid Input

Strengths Vs Weaknesses

- Strengths
  - Find highest probability errors with a relatively small set of tests.
  - Intuitively (trực giác) clear approach, easy to teach and understand
  - Extends well to multi-variable situations
- Blind spots or weaknesses
  - Errors that are not at boundaries or in obvious special cases
  - Also, the actual domains are often unknowable

Example – NextDate Problem

- Input variables:
  - **1 <= Day <= 31**
  - **1 <= Month <= 12**
  - **1900 <= Year**
- Output: the next date of the given date
- Note:
  - The year has been restricted so that test cases are not too large
  - All errors in the NextDate problem are denoted by “Invalid Input Date”. Ex: there is no 31/04/2012

Example: Triangle Problem

- Input:
  - 3 lengths of the sides of a triangle
- Output:
  - Scalene (three unequal sides)
  - Isosceles (two equal sides)
  - Equilaterial (three equal sides)
  - Right
  - Invalid

Example – NextDate Problem

month	day	year
min = 1	min = 1	min = 1812
min+ = 2	min+ = 2	min+ = 1813
nom = 6	nom = 15	nom = 1912
max- = 11	max- = 30	max- = 2011
max = 12	max = 31	max = 2012

Boundary Value Analysis Test Cases

Case	month	day	year	Expected Output
1	6	15	1812	June 16, 1812
2	6	15	1813	June 16, 1813
3	6	15	1912	June 16, 1912
4	6	15	2011	June 16, 2011
5	6	15	2012	June 16, 2012
6	6	1	1912	June 2, 1912
7	6	2	1912	June 3, 1912
8	6	30	1912	July 1, 1912
9	6	31	1912	error
10	1	15	1912	January 16, 1912
11	2	15	1912	February 16, 1912
12	11	15	1912	November 16, 1912
13	12	15	1912	December 16, 1912