Equivalence Class Testing

Equivalence Partitioning

Equivalence Partitioning or Equivalence Class Partitioning is type of black box testing technique which can be applied to all levels of software testing like unit, integration, system, etc. In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduces time required for testing because of small number of test cases.

- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.

Example 1: Equivalence and Boundary Value

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Equivalence partitioning

- Equivalence partitioning is a software testing technique that divides the input and/or output data of a software unit into partitions of data from which test cases can be derived.

- The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object.

- Test cases are designed to cover each partition at least once.

What can be found using equivalence partitioning?

- Equivalence partitioning technique uncovers classes of errors.

- Testing uncovers sets of inputs that causes errors or failures, not just individual inputs.

Equivalence Class Testing

- **Use the mathematical concept of partitioning into equivalence classes to generate test cases for Functional (Black-box) testing**

- **The key goals for equivalence class testing are similar to partitioning:**
  1. *completeness of test coverage*
  2. *lessen duplication of test coverage*

Equivalence Class Test Cases

- **Consider a numerical input variable, i, whose values may range from -200 through +200. Then a possible <u>partitioning</u> of testing input variable by 4 people may be:**
  - **-200 to -100**
  - **-101 to 0**
  - **1 to 100**
  - **101 to 200**

- **Define "same sign" as the <u>equivalence relation</u>, R, defined over the input variable's value set, i = {-200 - -,0, - -, +200}. Then one partitioning will be:**
  - **-200 to -1 (negative sign)**
  - **0   (no sign)**
  - **1 to 200   (positive sign)**

Example 2: Equivalence and Boundary Value

Following password field accepts minimum 6 characters and maximum 10 characters

That means results for values in partitions 0-5, 6-10, 11-14 should be <u>equivalent</u>

Enter Password: [   ] Submit

Test Scenario #	Test Scenario Description	Expected Outcome
1	Enter 0 to 5 characters in password field	System should not accept
2	Enter 6 to 10 characters in password field	System should accept
3	Enter 11 to 14 character in password field	System should not accept

Why Equivalence Class Testing

1. This testing is used to reduce a very large number of test cases to manageable chunks.
2. Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
3. Appropriate for calculation-intensive applications with a large number of variables/inputs

Comparison Summary

- Boundary Analysis testing is used when practically it is impossible to test a large pool of test cases individually
- Two techniques - Equivalence Partitioning & Boundary Value Analysis testing techniques are used
- In Equivalence Partitioning, first, you divide a set of test condition into a partition that can be considered.
- In Boundary Value Analysis you then test boundaries between equivalence partitions
- Appropriate for calculation-intensive applications with variables that represent physical quantities
- There is no reason why we could not define equivalence relations on the output

Weak Normal Equivalence testing

1. Assumes the 'single fault' or "independence of input variables."
   – e.g. If there are 2 input variables, these input variables are independent of each other.

2. Partition the test cases of each input variable separately into different equivalent classes.

3. Choose the test case from each of the equivalence classes for each input variable independently of the other input variable

WEAK NORMAL EQUIVALENCE CLASS TESTING

- **Weak normal** equivalence class testing is accomplished by using **one variable from each equivalence class (interval)** in a test case

- **One test case from one class and for example of total three classes** 'weak normal equivalence class' testing has 3 test cases shown with dots in Figure below.

- The test case in the lower left rectangle corresponds to a value of X1 in the class [a, b) and to a value of X2 in the class [e, f) and so on

Weak normal equivalence class test cases.
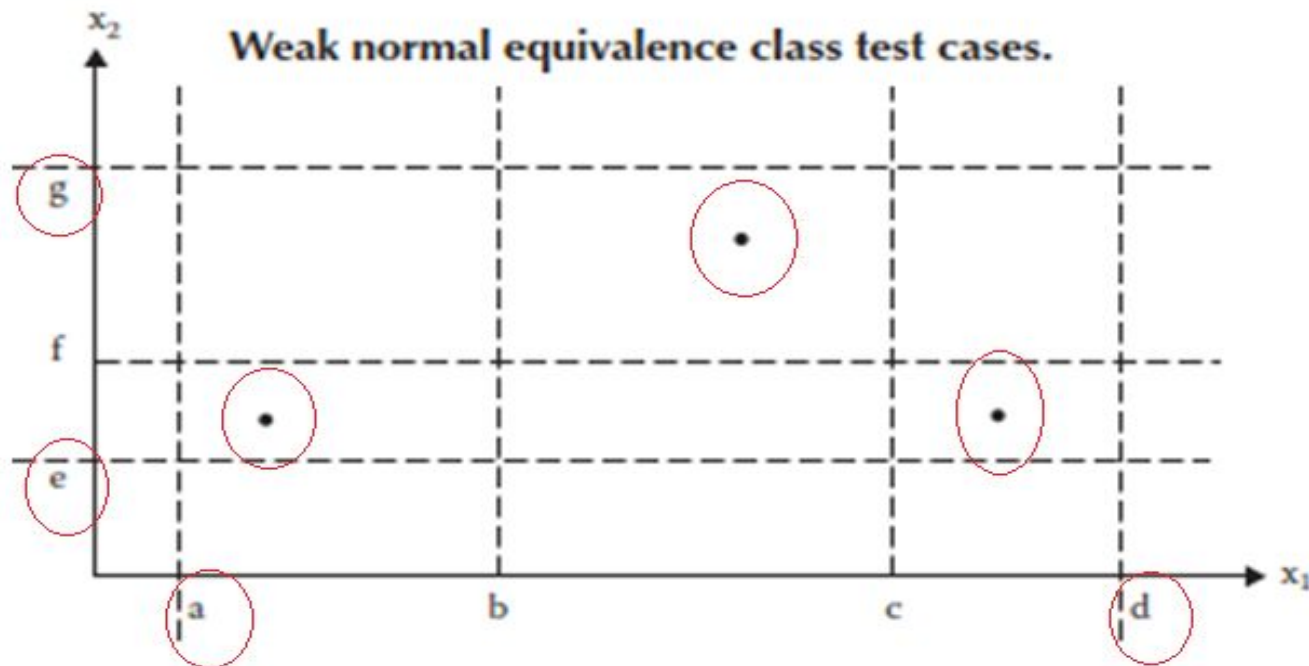
Example of : <u>Weak Normal</u> Equivalence testing

Assume the equivalence partitioning of input X is: 1 to 10; 11 to 20, 21 to 30
and the equivalence partitioning of input Y is: 1 to 5; 6 to 10; 11;15; and 16 to 20
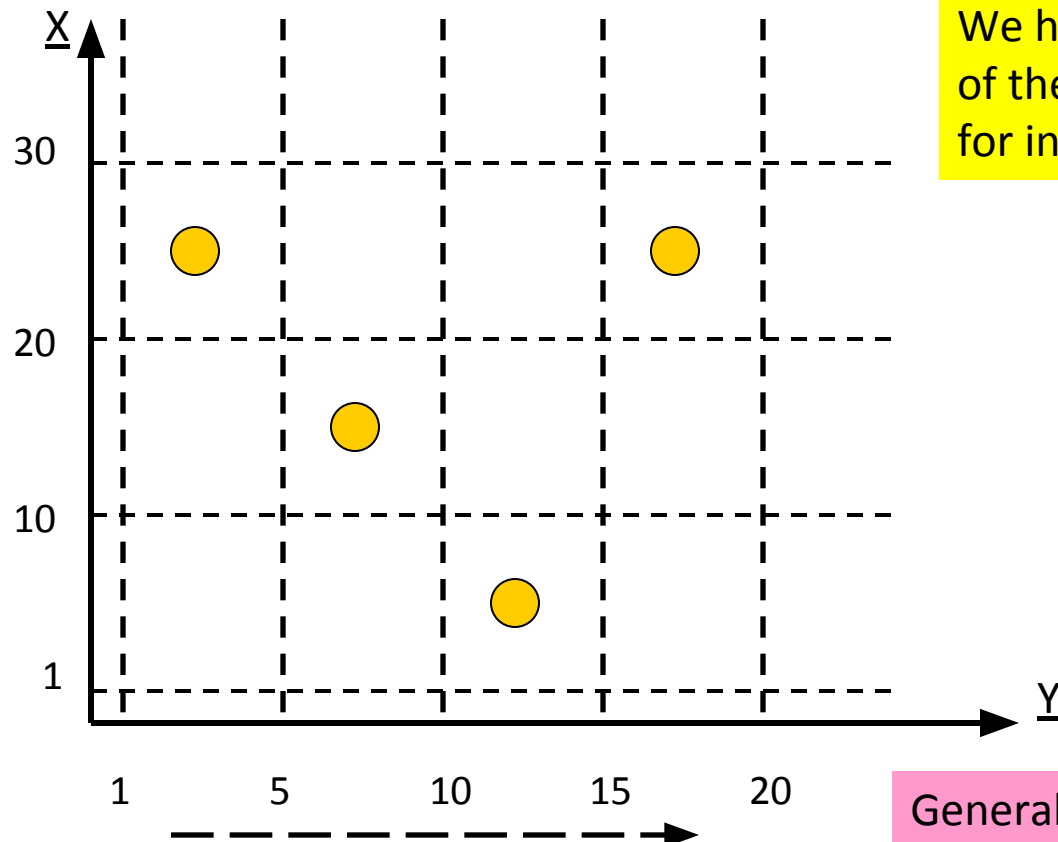
We have covered everyone of the <u>3</u> equivalence classes for input X.

For (x, y) we have:
(24, 2)
(15, 8)
(4, 13)
(23, 17)

General rule for # of test cases?
What do you think?
# of partitions of the largest set?

We have covered each of the <u>4</u> equivalence classes for input Y.

Strong Normal Equivalence testing

- This is the same as the weak normal equivalence testing except for

    "multiple fault assumption"

            or

    "dependence among the inputs"

- All the combinations of equivalence classes of the variables must be included.

STRONG NORMAL EQUIVALENCE CLASS TESTING

- **Strong** equivalence class testing is based on the **multiple fault assumption**

- We need test cases from each element of the Cartesian product of the equivalence classes, as shown in Figure.

- The Cartesian product guarantees that we have **a notion of "completeness"** in two senses: we cover all the equivalence classes, and **we have one of each possible combination of inputs.**

Strong normal equivalence class test cases.
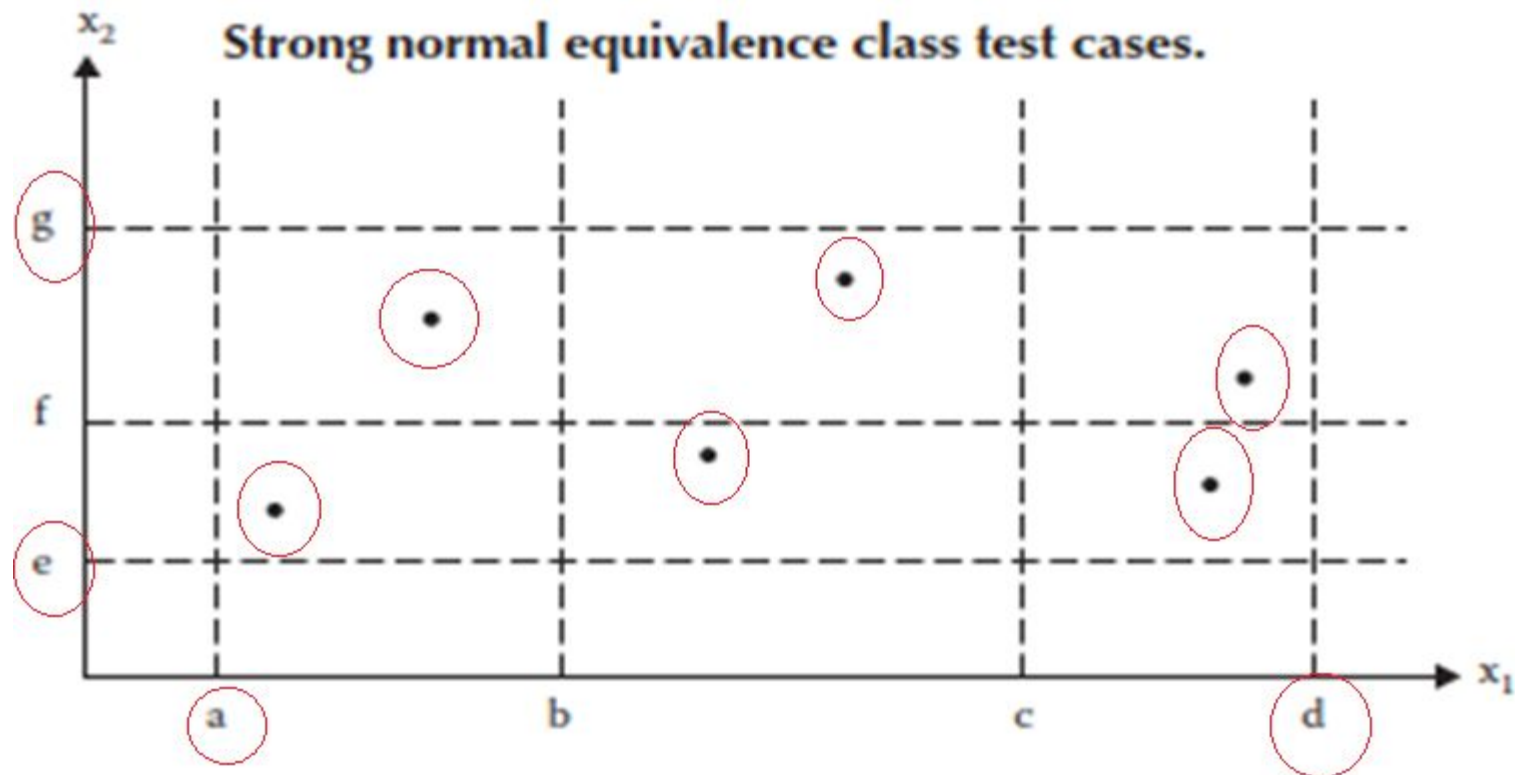
Example of : <u>Strong Normal</u> Equivalence testing

Assume the equivalence partitioning of input X is: 1 to 10; 11 to 20, 21 to 30
and the equivalence partitioning of input Y is: 1 to 5; 6 to 10; 11;15; and 16 to 20
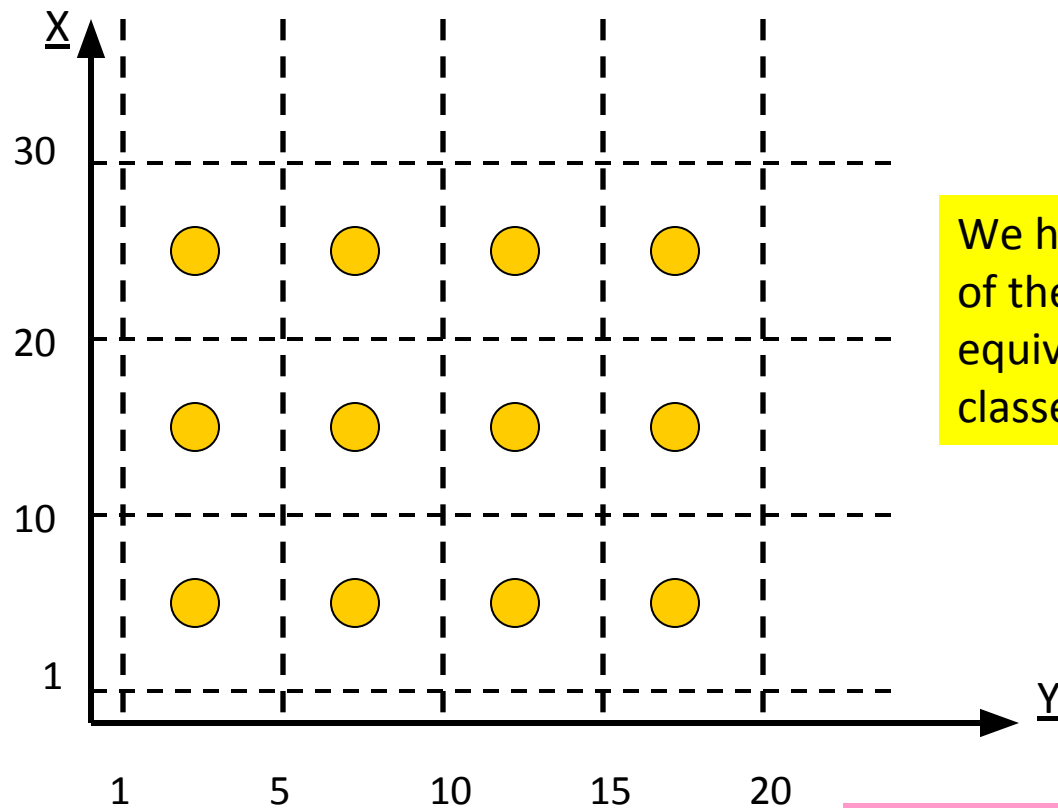
We have covered everyone
of the <u>3 x 4</u> Cartesian product of
equivalence
classes

General rule for # of test cases?
What do you think?

<u>Weak Robust</u> Equivalence testing

- **Up to now we have only considered partitioning the _<u>valid</u>_ input space.**

- **"Weak robust" is similar to "weak normal" equivalence test except that the _<u>invalid</u>_ input variables are now considered.**

<u>A note</u> about considering invalid input is that there may not be any definition "specified" for the various, different invalid inputs - - - making definition of the output for these invalid inputs a bit difficult at times. _(but fertile ground for testing_)
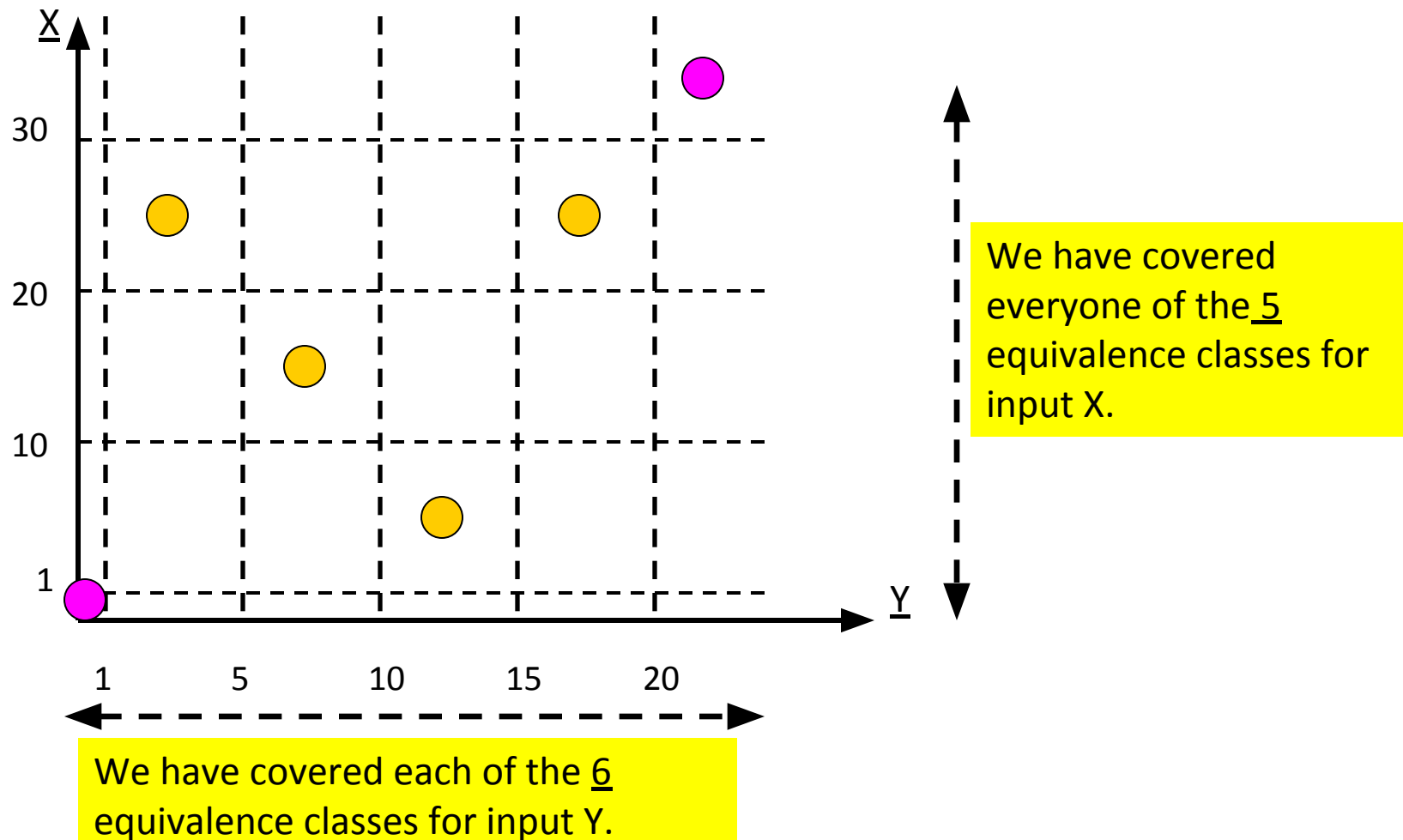
WEAK ROBUST EQUIVALENCE CLASS TESTING

- How something can be both **weak** and **robust**?

- The **robust part** comes from consideration of **invalid values**, and the **weak part** refers to the **single fault assumption**.

- The process of weak robust equivalence class testing is a simple extension of **weak normal equivalence class** testing + **classes that exceed the bounds**(lower, upper)

- Then **pick one test case from each class** , such that robust classes also represented as shown in Figure below.

Weak robust equivalence class test cases.
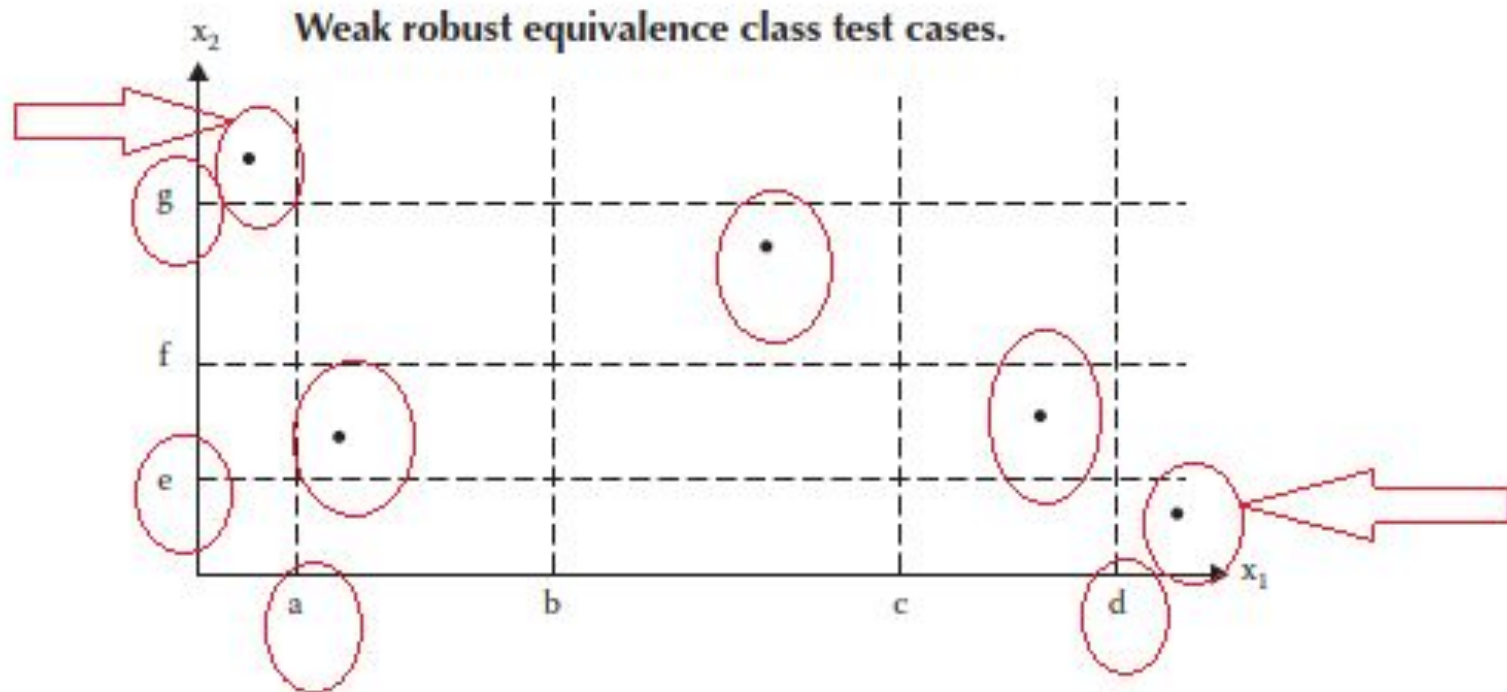
Example of : **Weak Robust** Equivalence testing

Assume the equivalence partitioning of input X is 1 to 10; 11 to 20, 21 to 30
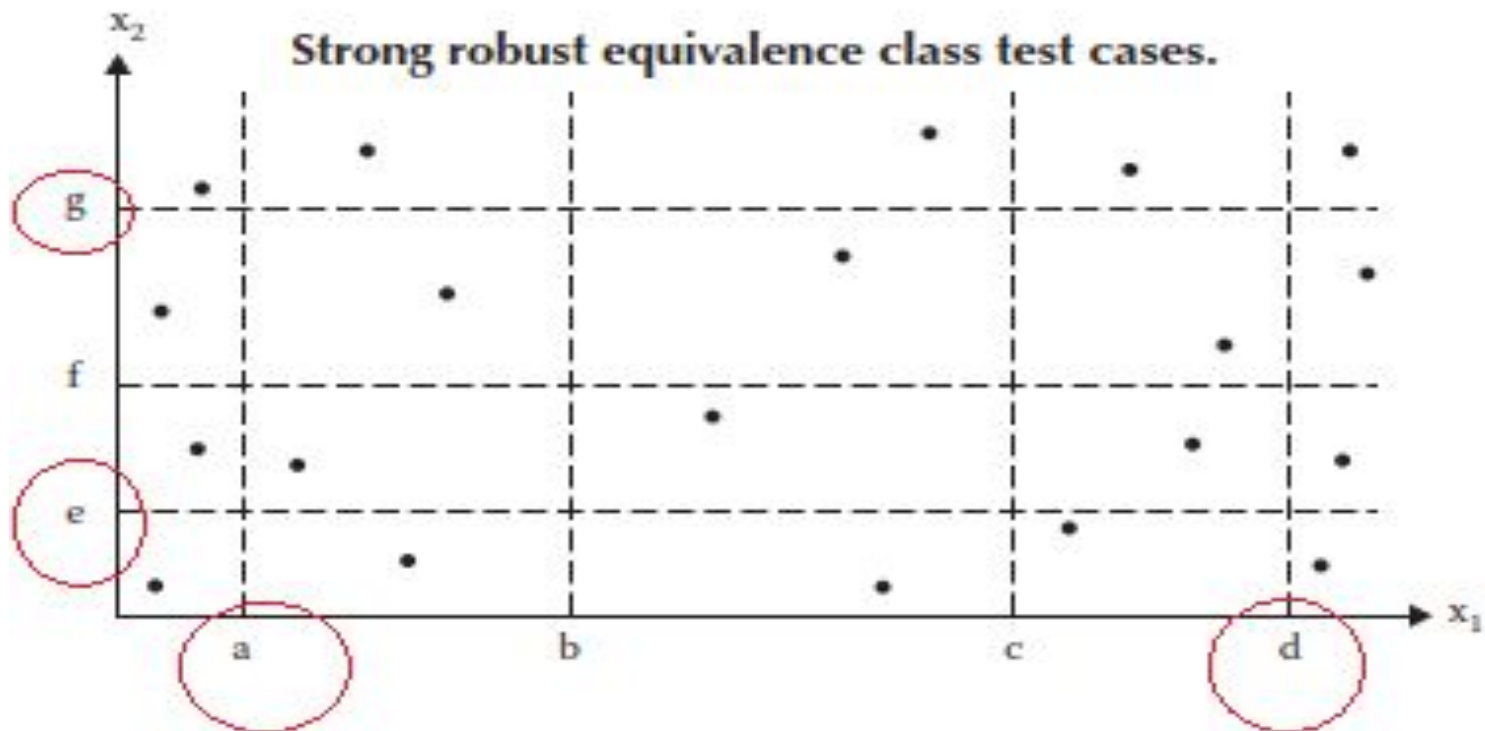and the equivalence partitioning of input Y is 1 to 5; 6 to 10; 11;15; and 16 to 20

We have covered everyone of the 5 equivalence classes for input X.

We have covered each of the 6 equivalence classes for input Y.

Strong Robust Equivalence testing

- **Does not assume "single fault" - - - assumes <u>dependency</u> of input variables**

- **"Strong robust" is similar to "strong normal" equivalence test except that the <u>invalid</u> input variables are now considered.**

STRONG ROBUST EQUIVALENCE CLASS TESTING

- As usual, the word **robust** part comes from consideration of **invalid values.**

- The **strong** part refers to the **multiple fault assumption**.

- We obtain test cases from **each element of the Cartesian product** in **all the equivalence classes**,

- Both valid and invalid, as shown in Figure below., which **consists** of **most redundant test cases**.

Strong robust equivalence class test cases.

Example of : <u>Strong Robust</u> Equivalence testing

Assume the equivalence partitioning of input X is: 1 to 10; 11 to 20, 21 to 30
and the equivalence partitioning of input Y is: 1 to 5; 6 to 10; 11;15; and 16 to 20
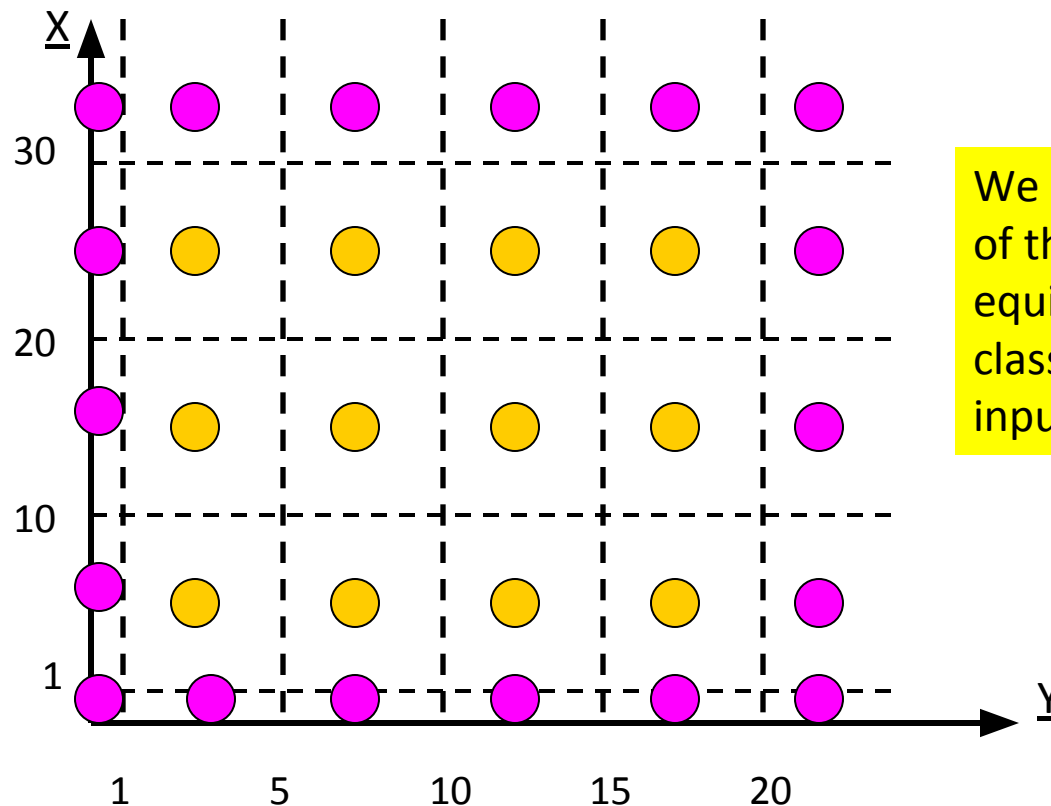
We have covered everyone
of the <u>5 x 6</u> Cartesian product of equivalence
classes (including invalid
inputs)

Equivalence class Definition

- **Note that the examples so far focused on defining input variables without considering the output variables.**
- **For the earlier "triangle problem," we are interested in 4 questions:**
  - **Is it a triangle?**
  - **Is it an isosceles?**
  - **Is it a scalene?**
  - **Is it an equilateral?**
- **We may define the input test data by defining the _equivalence class through "viewing" the 4 output groups_:**
  - *input sides <a, b, c> do not form a triangle*
  - *input sides <a, b ,c> form an isosceles triangle*
  - *input sides <a, b, c> form a scalene triangle*
  - *input sides <a, b, c> form an equilateral triangle*

Consider: <u>Weak Normal Equivalence Test Cases</u> for Triangle Problem

<u>"valid" inputs</u>:
1<= a <= 200
1<= b <= 200
1<= c <= 200
   and
for triangle:
 a < b + c
 b < a + c
 c < b + a

Valid Inputs to get

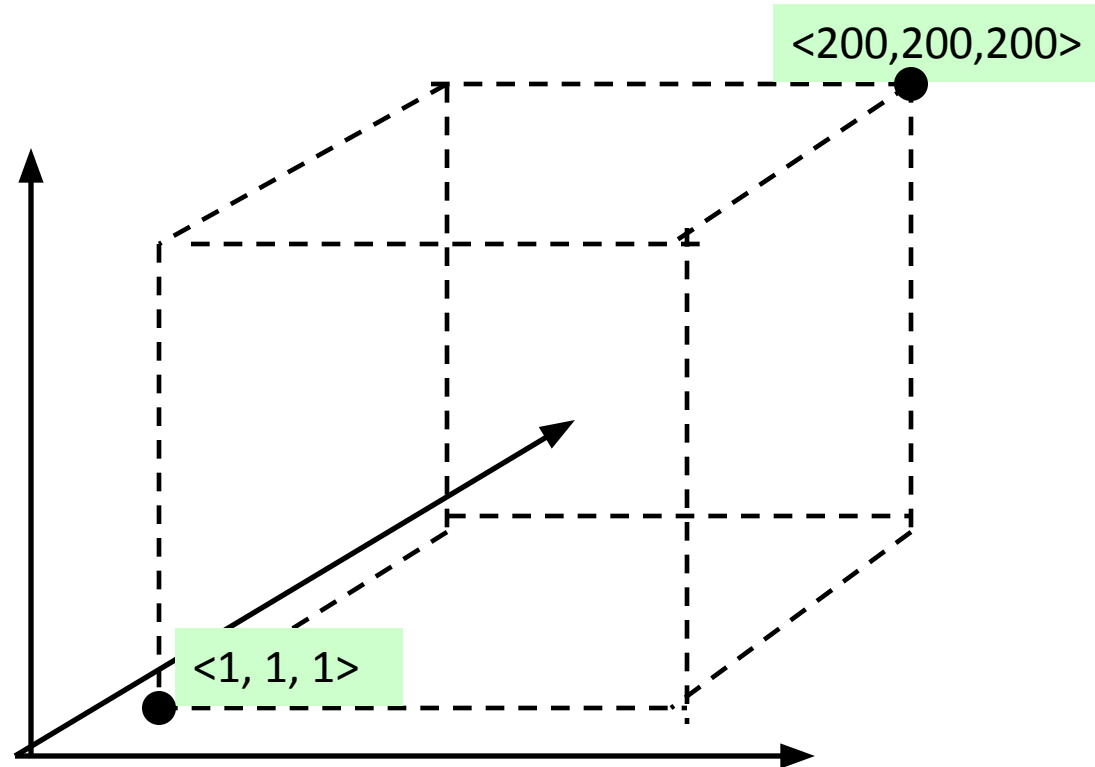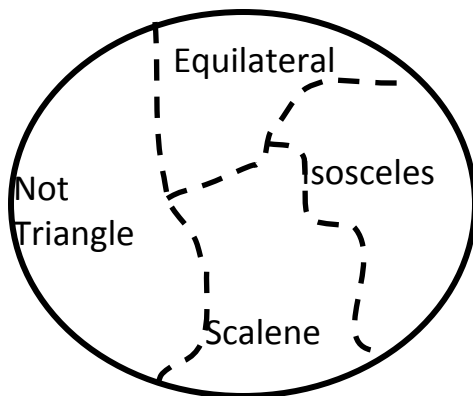|	inputs		
output	a	b	c
Not triangle	35	10	4
Equilateral	35	35	35
Isosceles	24	24	7
Scalene	35	18	24
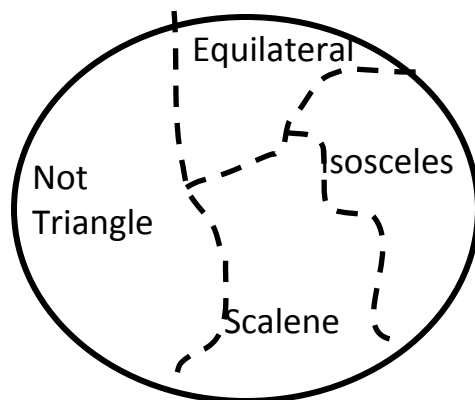
Strong Normal Equivalence Test Cases for Triangle Problem

- **Since there is no further sub-intervals inside the valid inputs for the 3 sides a, b, and c, are Strong Normal Equivalence is the same as the Weak Normal Equivalence**

Weak Robust Equivalence Test Cases for Triangle Problem

<200,200,200>

Now, on top of the earlier 4 normal test cases, include the "invalid" inputs

<1, 1, 1>

Valid Inputs

Equilateral

Isosceles

Not Triangle

Scalene

Include 6 invalid test case *in addition to Weak Normal*
above: below:

    <201, 45, 50 >   < -5, 76, 89 >
    <45, 204, 78 >   < 56, -20, 89 >
    <50, 78, 208 >   < 56, 89, 0 >

Strong Robust Equivalence Test Cases for Triangle Problem

- **Similar to Weak robust, but <u>all</u> combinations of "invalid" inputs must be _included to the Strong Normal._**
- **Look at the "cube" figure and consider the corners (two diagonal ones)**

a) Consider one of the corners <200,200,200> : there should be $(2^3 - 1) = 7$ cases of "invalids"

< 201, 201, 201 >	< 50 , 201, 50 >
< 201, 201, 50 >	< 50 , 201, 201 >
< 201, 50 , 201 >	< 50, 50 , 201 >
< 201, 50 , 50 >	

b) There will be 7 more "invalids" when we consider the other corner , <1,1,1 >:

< 0, 0, 0 >	<7, 0, 9 >
< 0, 0, 5 >	<8, 0, 0 >
< 0, 10, 0 >	<8, 9, 0 >
< 0, 8, 10>	

Guidelines and Observations

1. Obviously, the weak forms of equivalence class testing (normal or robust) are not as comprehensive as the corresponding strong forms.

2. If the implementation language is strongly typed (and invalid values cause run-time errors), it makes no sense to use the robust forms.

3. If error conditions are a high priority, the robust forms are appropriate.

4. Equivalence class testing is appropriate when input data is defined in terms of intervals and sets of discrete values. This is certainly the case when system malfunctions can occur for out-of-limit variable values

5. Equivalence class testing is strengthened by a hybrid approach with boundary value testing

Guidelines and Observations

6. Equivalence class testing is indicated when the program function is complex.

7. Strong equivalence class testing makes a presumption that the variables are independent, and the corresponding multiplication of test cases raises issues of redundancy.

8. The difference between the strong and weak forms of equivalence class testing is helpful in the distinction between progression and regression testing.