

Software Testing

Rajib Mall

CSE Department

IIT KHARAGPUR



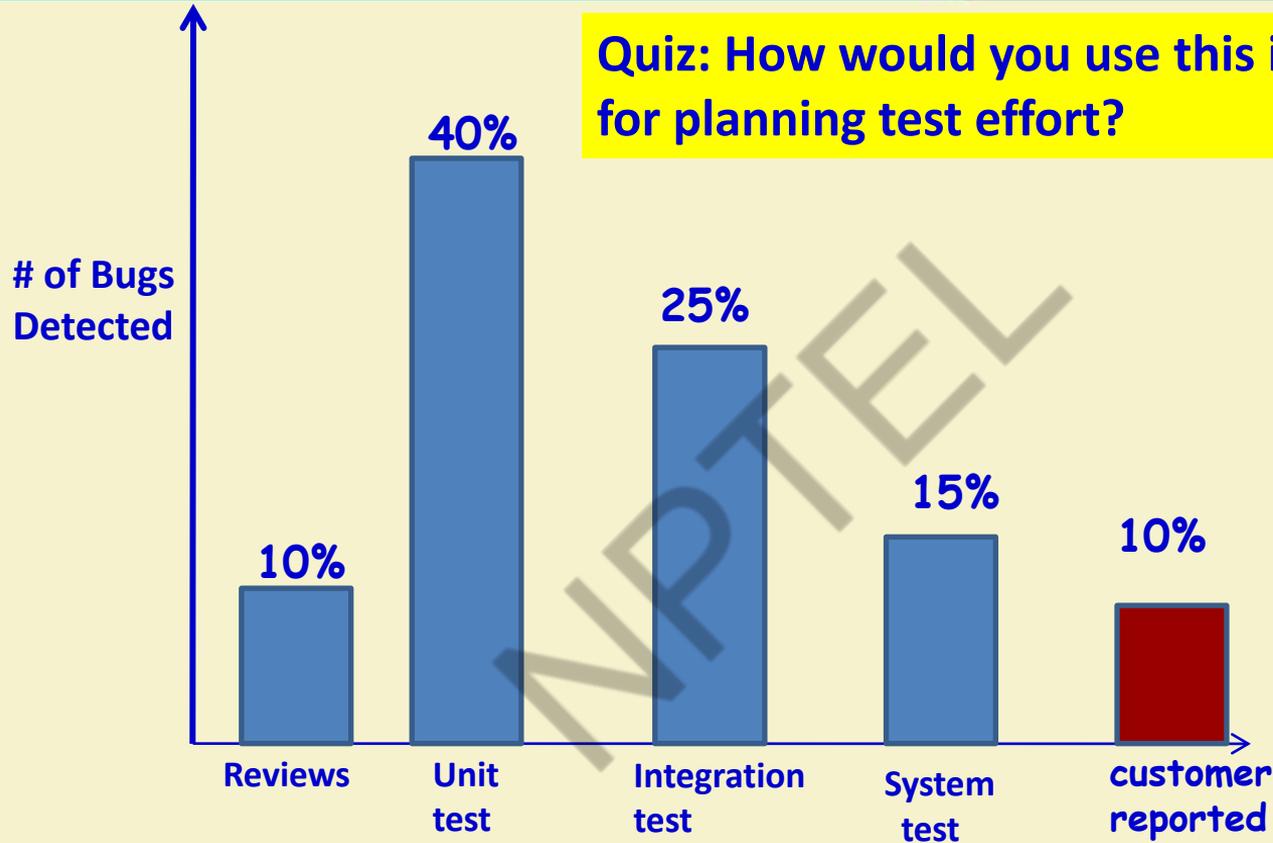
Design of Test Cases

- Systematic approaches are required to design an effective test suite:
 - Each test case in the suite should target different faults.

Testing Strategy

- Test Strategy primarily addresses:
 - **Which types of tests to deploy?**
 - **How much effort to devote to which type of testing?**
 - **Black-box: Usage-based testing** (based on customers' actual usage pattern)
 - **White-box testing** can be guided by black box testing results

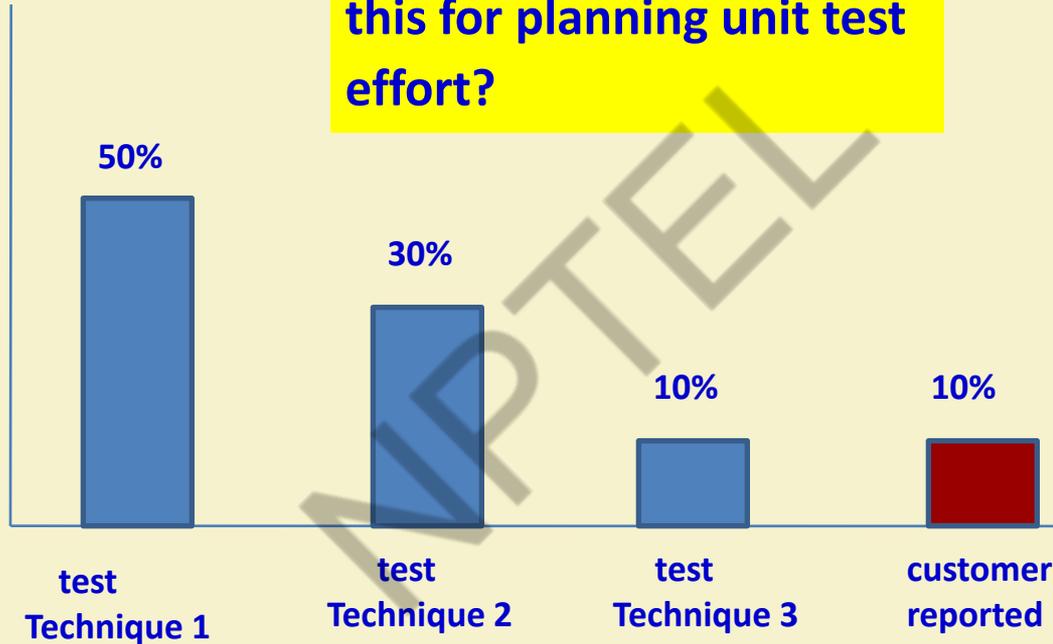
Quiz: How would you use this information for planning test effort?



Considering Past Bug Detection Data...

Quiz: How would you use this for planning unit test effort?

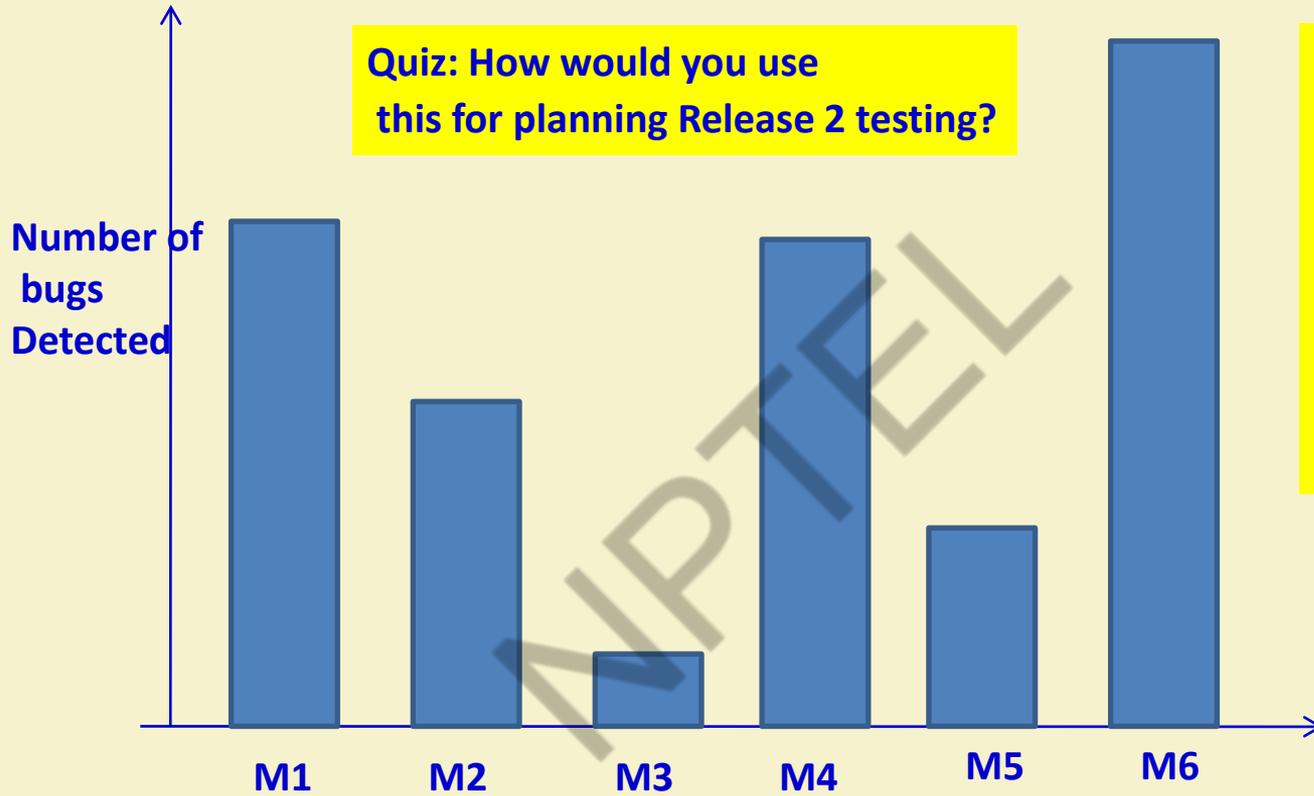
Problems Detected



Consider Past Bug Detection Data...

Quiz: How would you use this for planning Release 2 testing?

Distribution of Error Prone Modules customer reported bugs for Release 1



Unit Testing

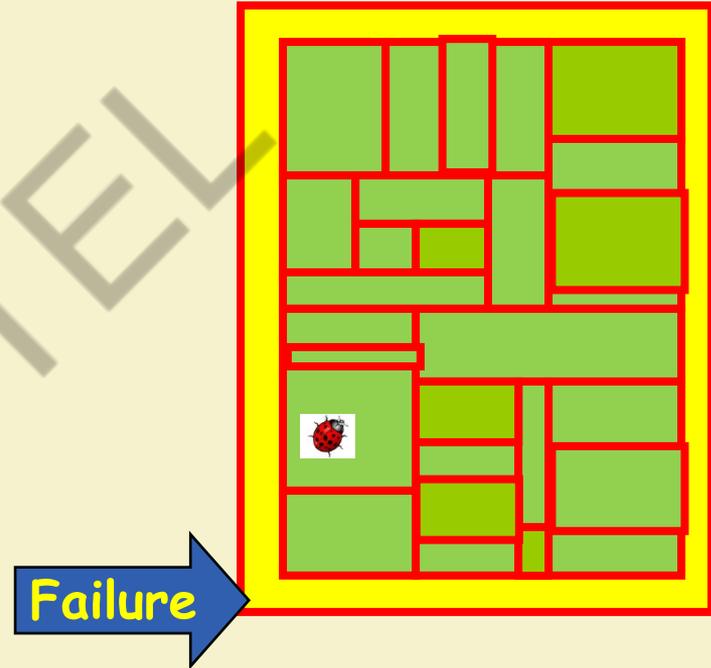


When and Why of Unit Testing?

- Unit testing carried out:
 - After coding of a unit is complete and it compiles successfully.
- Unit testing reduces debugging effort substantially.

Why unit test?

- Without unit test:
 - Errors become difficult to track down.
 - Debugging cost increases substantially...

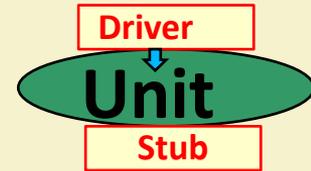


Testing of individual methods, modules, classes, or components in isolation:

Unit Testing

- Carried out before integrating with other parts of the software being developed.

Following support required for Unit testing:



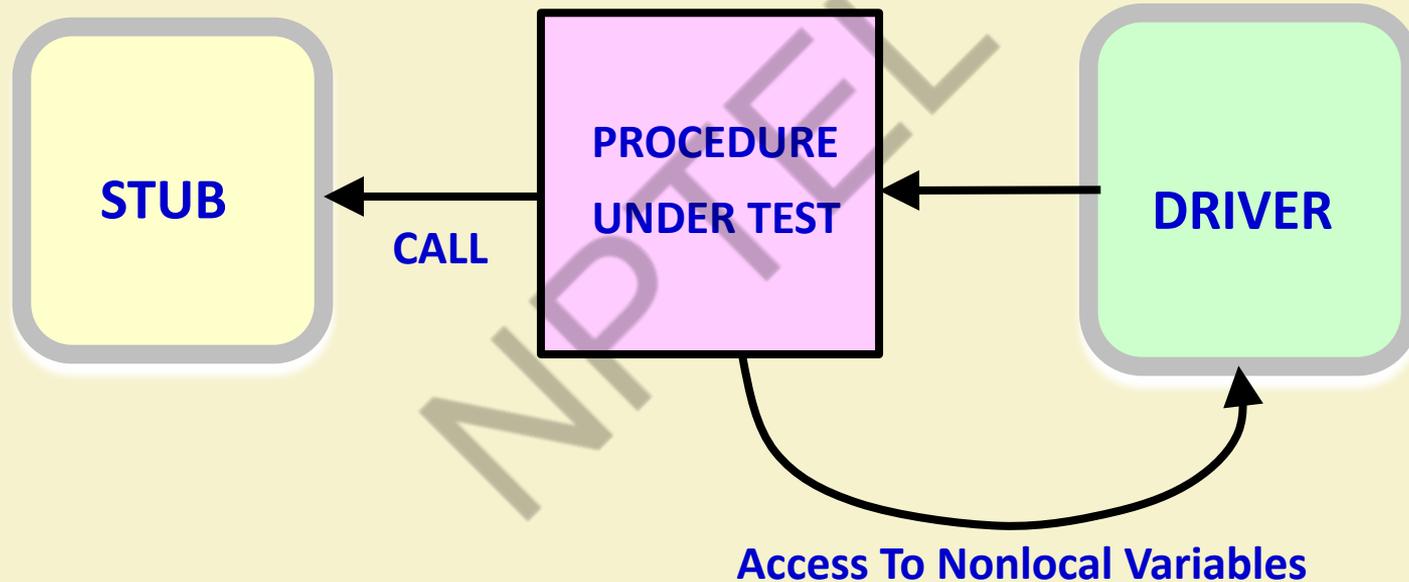
– Driver

- Simulates the behavior of a function that calls and supplies necessary data to the function being tested.

– Stub

- Simulates the behavior of a function that has not yet been written.

Unit Testing



Quiz

- Unit testing can be considered as which one of the following types of activities?
 - **Verification**
 - **Validation**

Design of Unit Test Cases

- There are essentially three main approaches to design test cases:
 - **Black-box approach**
 - **White-box (or glass-box) approach**
 - **Grey-box approach**

Black-Box Testing

- Test cases are designed using only **functional specification** of the software:

–Without any knowledge of the internal structure of the software.



- Black-box testing is also known as **functional testing**.

What is Hard about BB Testing

- Data domain is large
- A function may take multiple parameters:
 - We need to consider the combinations of the values of the different parameters.

What's So Hard About Testing?

- Consider `int check-equal(int x, int y)`
- Assuming a 64 bit computer
 - Input space = 2^{128}
- Assuming it takes 10secs to key-in an integer pair:
 - It would take about a billion years to enter all possible values!
 - Automatic testing has its own problems!

Solution

- Construct model of the data domain:
 - Called Domain based testing
 - Select data based on the domain model

White-box Testing

- To design test cases:
 - Knowledge of internal structure of software necessary.
 - White-box testing is also called structural testing.

Black Box Testing

- Software considered as a black box:
 - Test data derived from the specification
 - No knowledge of code necessary
- Also known as:
 - Data-driven or
 - Input/output driven testing
- The goal is to achieve the thoroughness of exhaustive input testing:
 - With much less effort!!!!

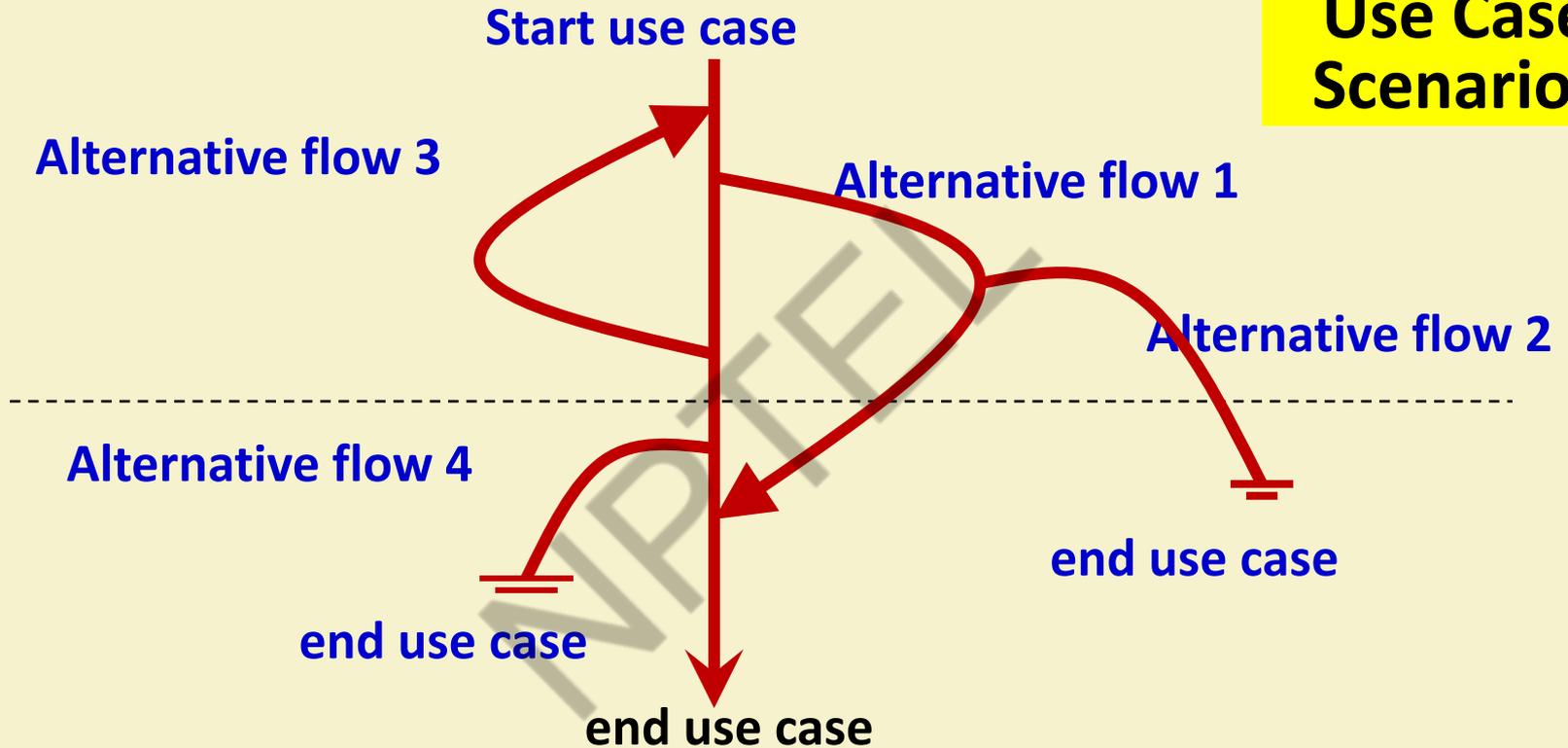


Black-Box Testing

- Scenario coverage
- Equivalence class partitioning
- Boundary value testing
- Cause-effect (Decision Table) testing
- Combinatorial testing
- Orthogonal array testing



Use Case Scenarios



Deriving test cases from use cases

1. Identify the use case scenarios
2. For each scenario, identify one or more test cases
3. For each test case, identify the conditions that will cause it to execute.
4. Complete the test case by adding data values

Scenario number	Originating flow	Alternative flow	Next alternative	Next alternative
1	Basic flow			
2	Basic flow	Alt. flow 1		
3	Basic flow	Alt. flow 1	Alt. flow 2	
4	Basic flow	Alt. flow 3		
5	Basic flow	Alt. flow 3	Alt. flow 1	
6	Basic flow	Alt. flow 3	Alt. flow 1	Alt. flow 2
7	Basic flow	Alt. flow 4		
8	Basic flow	Alt. flow 3	Alt. flow 4	

**Identify
use case
scenarios:
Example**

Identify the test cases

- Parameters of any test case:
 - Conditions
 - Input (data values)
 - Expected result
 - Actual result

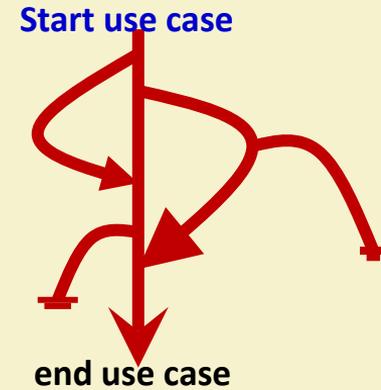
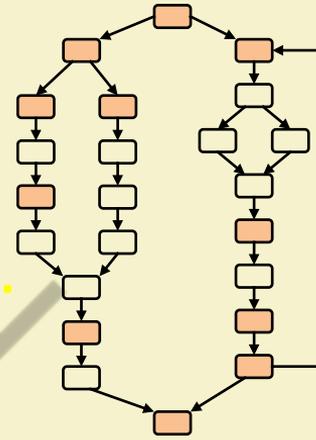
Test case ID	Scenario/condition	Data value 1	Data value 2	Data value N	Exp. results	Actual results
1	Scenario 1					
2	Scenario 2					
3	Scenario 3					

Equivalence Class Testing



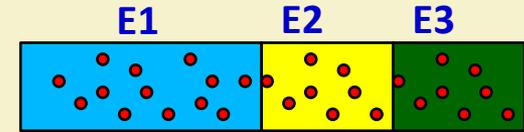
Equivalence Class Partitioning

- The input values to a program:
 - Partitioned into **equivalence classes**.
- Partitioning is done such that:



- **Program behaves in similar ways to every input value belonging to an equivalence class.**
- **At the least, there should be as many equivalence classes as scenarios.**

Why Define Equivalence Classes?



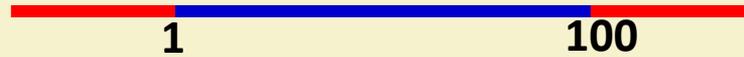
- Premise:
 - Testing code with any one representative value from a equivalence class:
 - As good as testing using any other values from the equivalence class.

Equivalence Class Partitioning

- How do you identify equivalence classes?
 - **Identify scenarios**
 - **Examine the input data.**
 - **Examine output**
- Few guidelines for determining the equivalence classes can be given...

- If an input is a range, one valid and two invalid equivalence classes are defined.

Example: 1 to 100



- If an input is a set, one valid and one invalid equivalence classes are defined. Example: {a,b,c}

- If an input is a Boolean value, one valid and one invalid class are defined.

Example:

- Area code:** input value defined between 10000 and 90000--- **range**
- Password:** string of six characters --- **set**

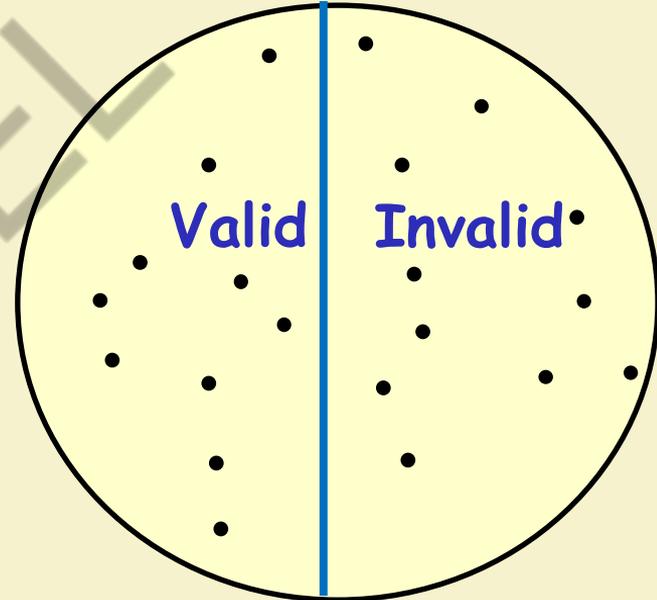
**Guidelines
to Identify
Equivalence
Classes**

Equivalent class partition: Example

- Given three sides, determine the type of the triangle:
 - Isosceles
 - Scalene
 - Equilateral, etc.
- Hint: scenarios correspond to output in this case.

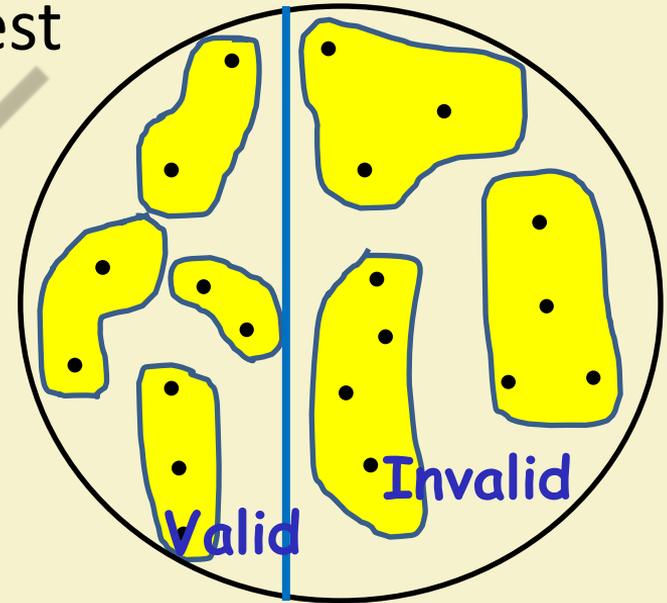
Equivalence Partitioning

- First-level partitioning:
 - Valid vs. Invalid test cases



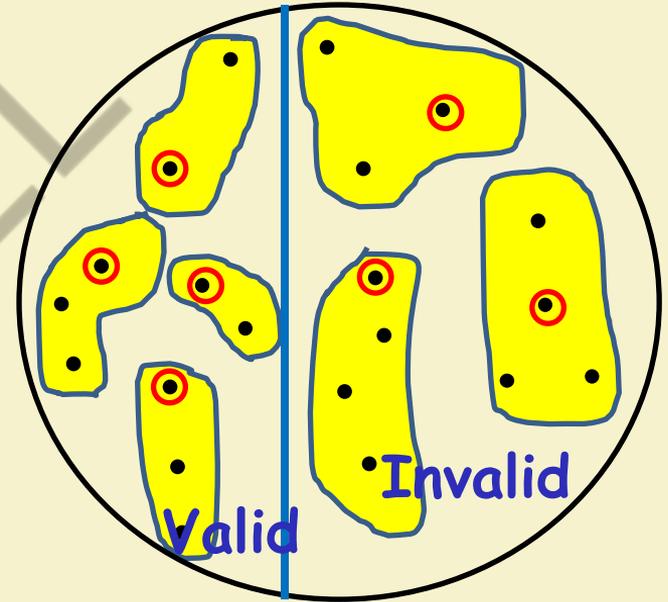
Equivalence Partitioning

- Further partition valid and invalid test cases into equivalence classes



Equivalence Partitioning

- Create a test case using at least one value from each equivalence class



Equivalence Class Partitioning

- If the input data to the program is specified by a **range of values**:
 - e.g. numbers between 1 to 5000.
 - One valid and two invalid equivalence classes are defined.

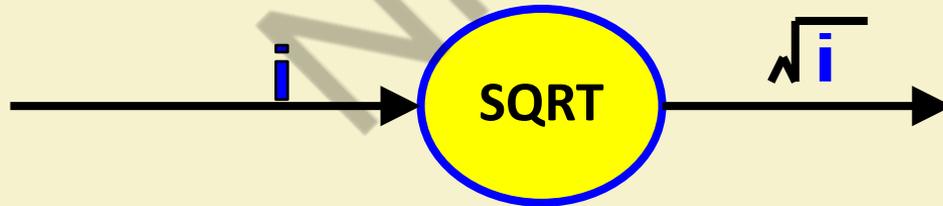


Equivalence Class Partitioning

- If input is an enumerated set of values, e.g. :
 - {a,b,c}
- Define:
 - One equivalence class for valid input values.
 - Another equivalence class for invalid input values..

Example

- A program reads an input value in the range of 1 and 5000:
 - Computes the square root of the input number



Example (cont.)

- Three equivalence classes:
 - The set of negative integers,
 - Set of integers in the range of 1 and 5000,
 - Integers larger than 5000.



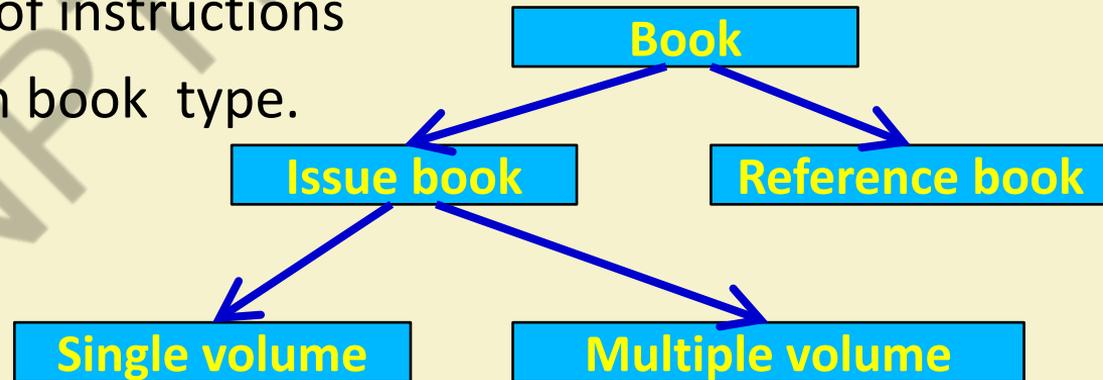
Example (cont.)

- The test suite must include:
 - Representatives from each of the three equivalence classes:
 - A possible test suite can be:
 $\{-5, 500, 6000\}$.
- 

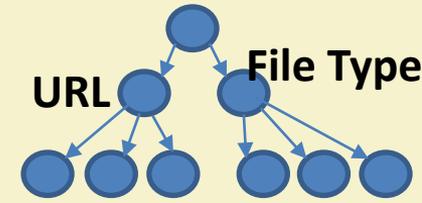
Equivalence Partitioning

- A set of input values constitute an equivalence class if the tester believes that these are processed identically:

- Example : `issue book(book id);`
- Different set or sequence of instructions may be executed based on book type.



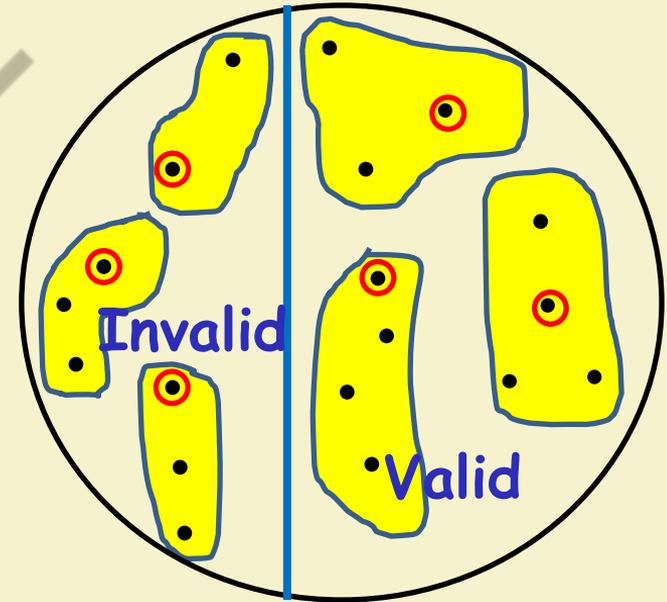
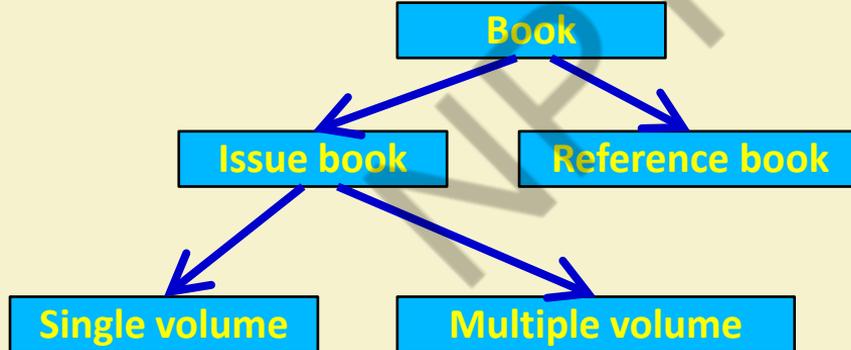
Equivalence Partitioning: Example 1



- Example: **Image Fetch-image(URL)**
 - **Equivalence Definition 1:** Partition based on URL protocol (“http”, “https”, “ftp”, “file”, etc.)
 - **Equivalence Definition 2:** Partition based on type of file being retrieved (HTML, GIF, JPEG, Plain Text, etc.)

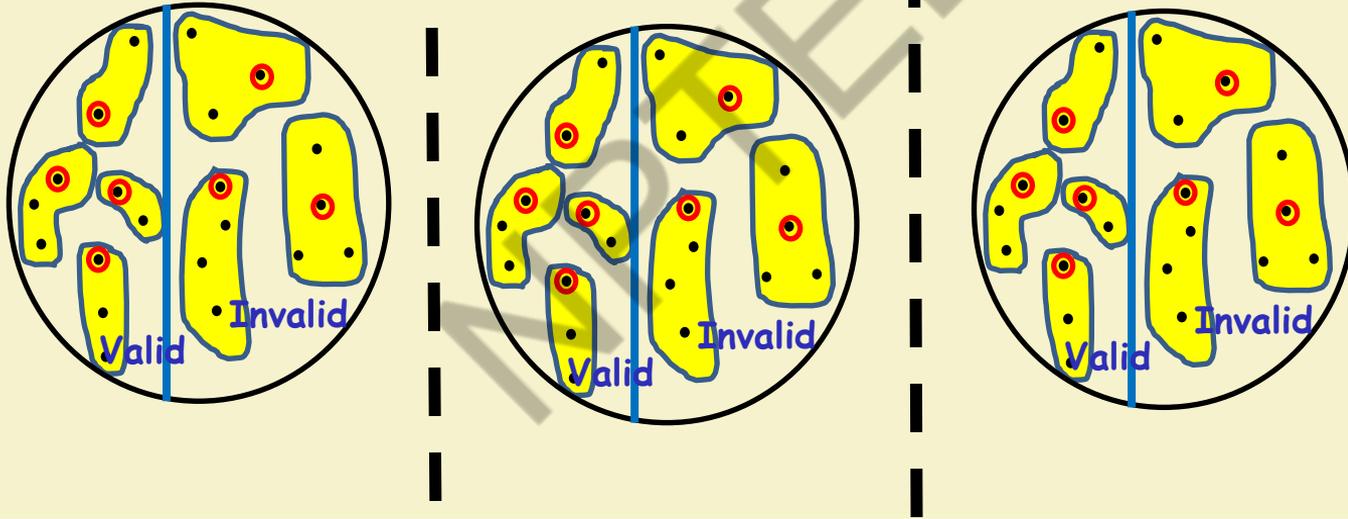
Equivalence Partitioning: Single Parameter Function

- `issue-book(int book-id)`
- Create a test case for at least one value from each equivalence class



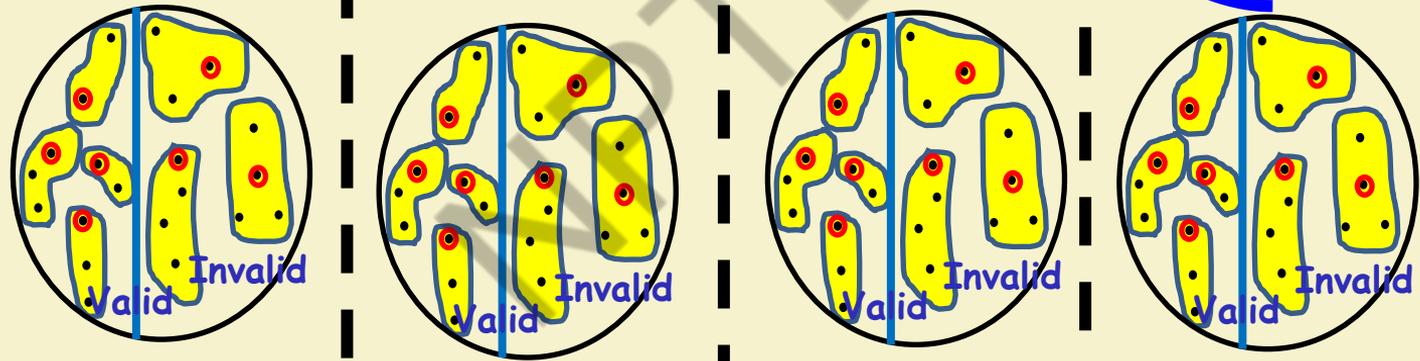
Multiparameter Functions

- `postGrade(Roll, CourseNo, Grade)`



```
int Normalization Factor;  
postGrade(Roll, CourseNo, Grade)  
{ Grade=Grade*NormalizationFactor  
-----}  
}
```

Multiparameter
Function Accessing
Global Variables



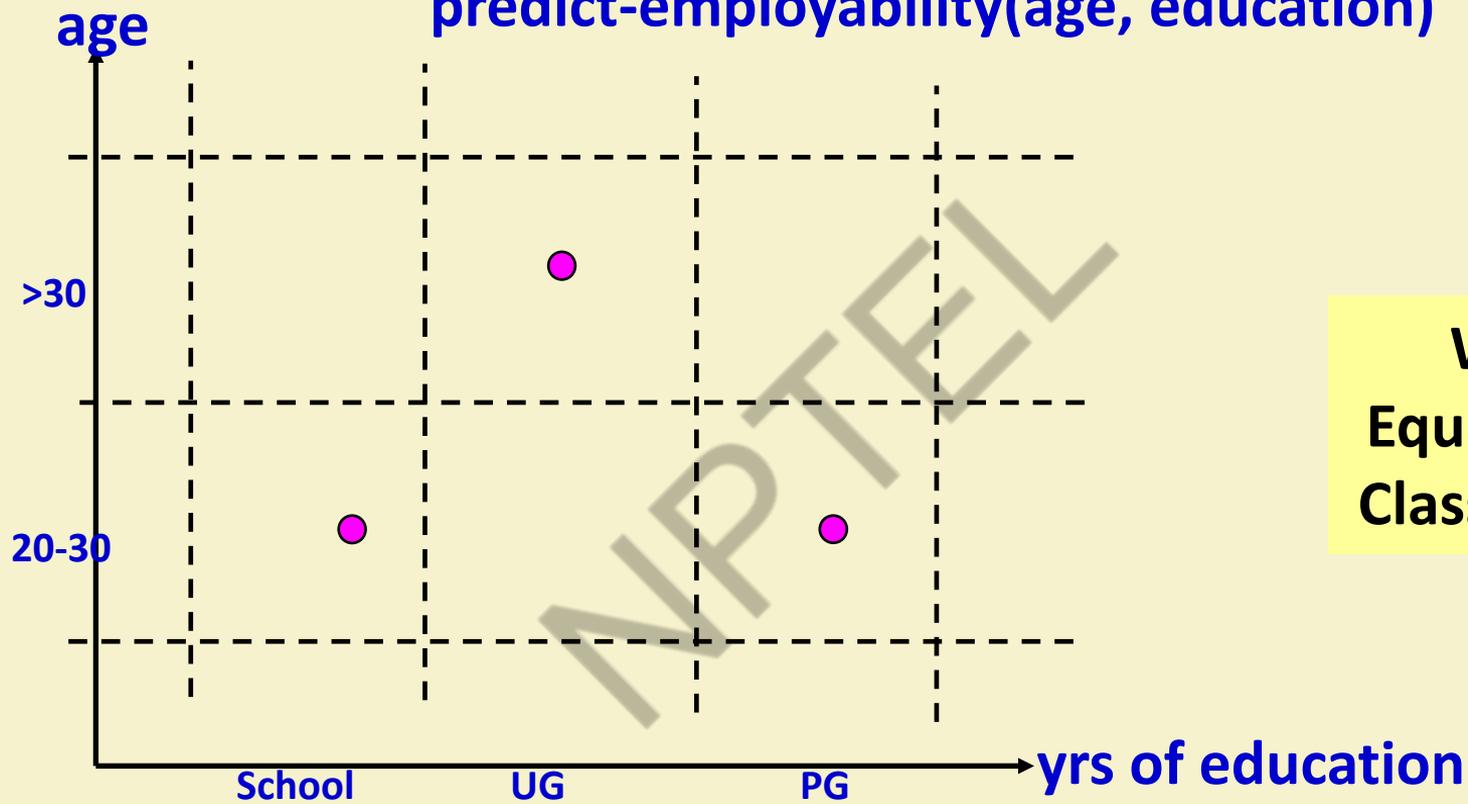
Multi Parameter Equivalence Partitioning: Example

Input Parameter	Valid Equivalence Classes	Invalid Equivalence Classes
An integer N such that: $-99 \leq N \leq 99$?	?
Phone Number Area code: [11,..., 999] Suffix: Any 6 digits	?	?

Equivalence Partitioning: Example

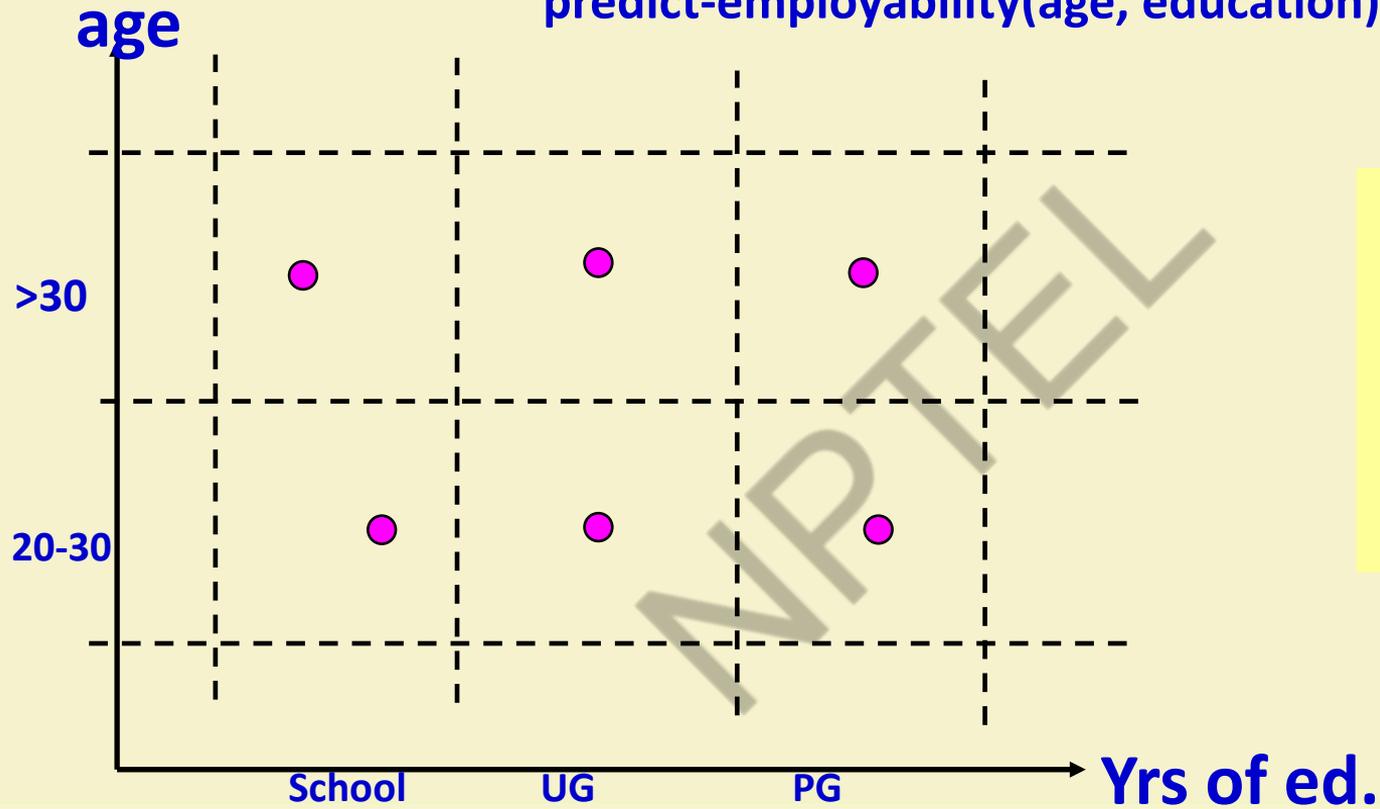
Input	Valid Equivalence Classes	Invalid Equivalence Classes
<p>A integer N such that:</p> <p>$-99 \leq N \leq 99$</p>	<p>$[-99, 99]$</p>	<p>< -99</p> <p>> 99</p> <p>Malformed numbers {12-, 1-2-3, ...}</p> <p>Non-numeric strings {junk, 1E2, \$13}</p> <p>Empty value</p>
<p>Phone Number</p> <p>Prefix: $[11, 999]$</p> <p>Suffix: Any 6 digits</p>	<p>$[11,999][000000,$ $999999]$</p>	<p>Invalid format 5555555,</p> <p>Area code < 11 or > 999</p> <p>Area code with non-numeric characters</p>

predict-employability(age, education)



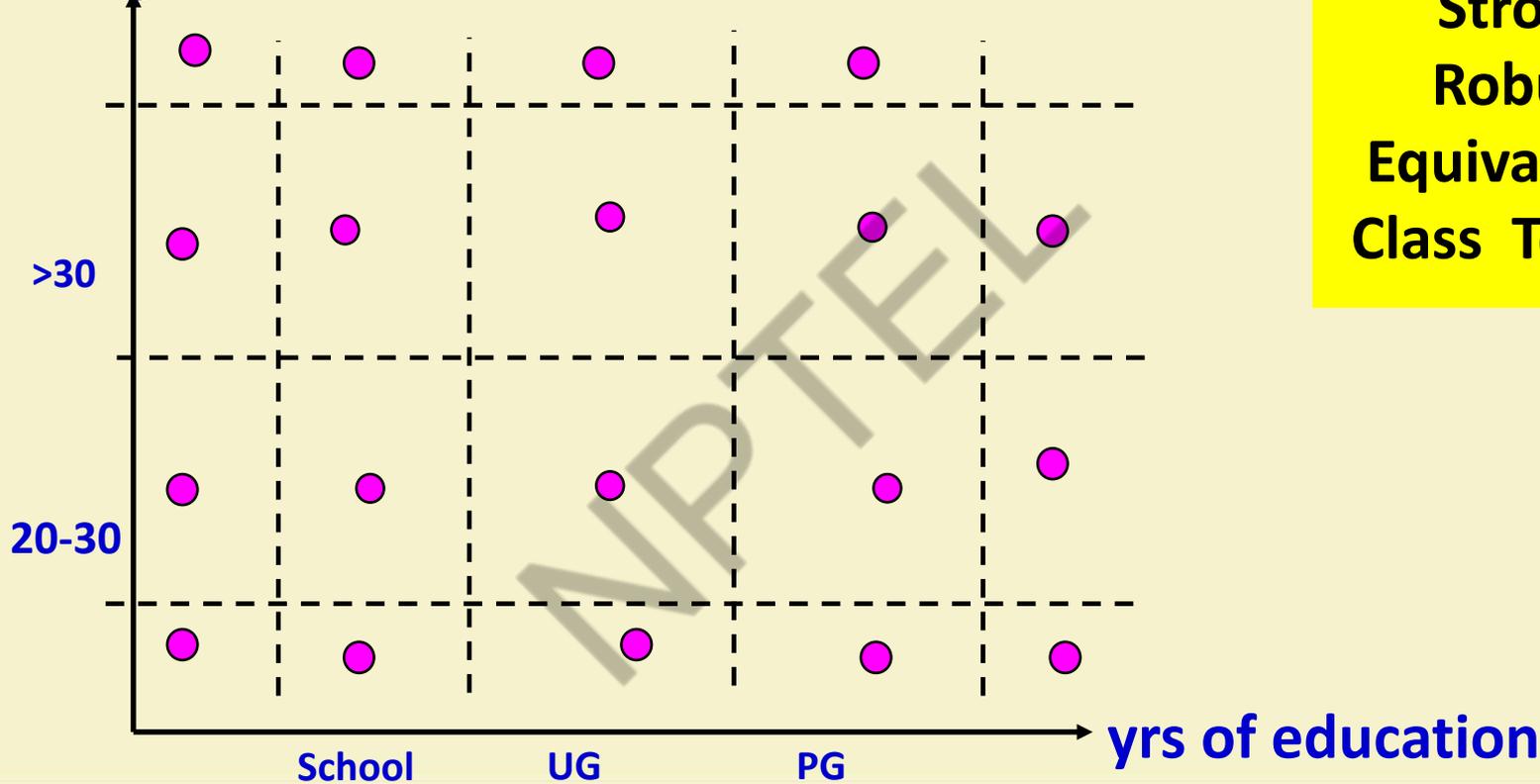
**Weak
Equivalence
Class Testing**

predict-employability(age, education)



**Strong
Equivalence
Class Testing**

age predict-employability(age, education)



**Strong
Robust
Equivalence
Class Testing**

- Design Equivalence class test cases: **compute-interest(days)**
- A bank pays different rates of interest on a deposit depending on the deposit period.
 - 3% for deposit up to 15 days
 - 4% for deposit over 15 days and up to 180 days
 - 6% for deposit over 180 days upto 1 year
 - 7% for deposit over 1 year but less than 3 years
 - 8% for deposit 3 years and above

Quiz 1

- Design Equivalence class test cases: **compute-interest(principal, days)**
- For deposits of less than or equal to Rs. 1 Lakh, rate of interest:
 - 6% for deposit upto 1 year
 - 7% for deposit over 1 year but less than 3 years
 - 8% for deposit 3 years and above
- For deposits of more than Rs. 1 Lakh, rate of interest:
 - 7% for deposit upto 1 year
 - 8% for deposit over 1 year but less than 3 years
 - 9% for deposit 3 years and above

Quiz 2

Quiz 3

- Design equivalence class test cases.
 - Consider a program that takes 2 strings of maximum length 20 and 5 characters respectively
 - Checks if the second is a substring of the first
 - **substr(s1 ,s2);**

Special Value Testing



Special Value Testing

- What are special values?
 - The tester has reasons to believe that execution with certain values may expose bugs:
 - **General risk:** Example-- Boundary value testing
 - **Special risk:** Example-- Leap year not considered

Boundary Value Analysis

- Some typical programming errors occur:

- At boundaries of equivalence classes

- Might be purely due to psychological factors.**



- Programmers often commit mistakes in the:

- Special processing at the boundaries of equivalence classes.**

Boundary Value Analysis

- Programmers may improperly use $<$ instead of $<=$
- Boundary value analysis: 
 - **Select test cases at the boundaries of different equivalence classes.**



Boundary Value Analysis: Guidelines

- If an input is a range, bounded by values a and b:
 - Test cases should be designed with value a and b, just above and below a and b.
- **Example 1:** Integer D with input range [-3, 10],
 - test values: -3, 10, 11, -2, 0
- **Example 2:** Input in the range: [3,102]
 - test values: 3, 102, -1, 200, 5

Boundary Value Testing Example

- Process employment applications based on a person's age.

0-16	Do not hire
16-18	May hire on part time basis
18-55	May hire full time
55-99	Do not hire

- Notice the problem at the boundaries.
 - Age "16" is included in two different equivalence classes (as are 18 and 55).

Boundary Value Testing: Code Example

- If (applicantAge >= 0 && applicantAge <=16) hireStatus="NO";
- If (applicantAge >= 16 && applicantAge <=18) hireStatus="PART";
- If (applicantAge >= 18 && applicantAge <=55) hireStatus="FULL";
- If (applicantAge >= 55 && applicantAge <=99) hireStatus="NO";

Boundary Value Testing Example (cont)

- Corrected boundaries:

0–15 Don't hire

16–17 Can hire on a part-time basis only

18–54 Can hire as full-time employees

55–99 Don't hire

- What about ages -3 and 101?
- The requirements do not specify how these values should be treated.

Boundary Value Testing Example (cont)

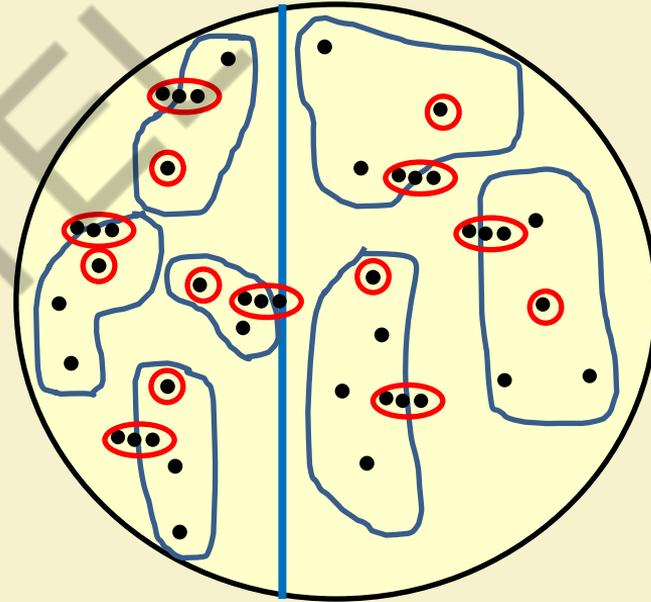
- The code to implement the corrected rules is:

```
If (applicantAge >= 0 && applicantAge <=15)           hireStatus="NO";  
If (applicantAge >= 16 && applicantAge <=17)         hireStatus="PART";  
If (applicantAge >= 18 && applicantAge <=54)         hireStatus="FULL";  
If (applicantAge >= 55 && applicantAge <=99)         hireStatus="NO";
```

- Special values on or near the boundaries in this example are {-1, 0, 1}, {14, 15, 16}, {17, 18, 19}, {54, 55, 56}, and {98, 99, 100}.

Boundary Value Analysis

- Create test cases to test boundaries of equivalence classes



Example 1

- For a function that computes the square root of an integer in the range of 1 and 5000:
 - Test cases must include the values:
{0,1,2,4999,5000,5001}.

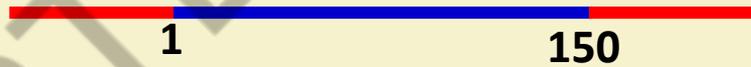


- Consider a program that reads the “age” of employees and computes the average age.

Example 2

input (ages) → **Program** → output: average age

Assume valid age is 1 to 150



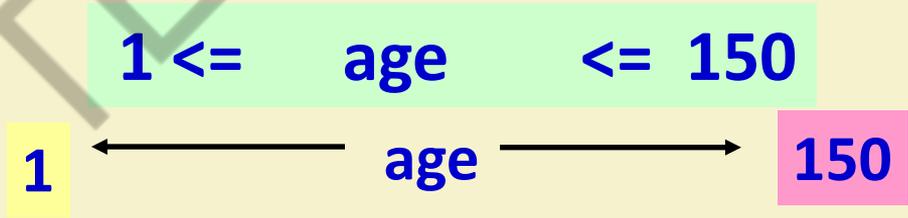
- How would you test this?
 - How many test cases would you generate?
 - What type of test data would you input to test this program?

Boundaries of the inputs

The “basic” boundary value testing would include 5 test cases:

1. - at minimum boundary
2. - immediately above minimum
3. - between minimum and maximum (nominal)
4. - immediately below maximum
5. - at maximum boundary

predict-longevity(age)



Test Cases for the Example

- How many test cases for the example ?

– answer : **5**

- Test input values? :

1 at the minimum

2 at one above minimum

45 at middle

149 at one below maximum

150 at maximum

predict-longevity(age)

Multiple Parameters: Independent distinct Data

- Suppose there are 2 “distinct” inputs that are assumed to be independent of each other.
 - Input field 1: years of education (say 1 to 23)
 - Input field 2: age (1 to 150)
- If they are independent of each other, then we can start with $5 + 5 = 10$ sets.

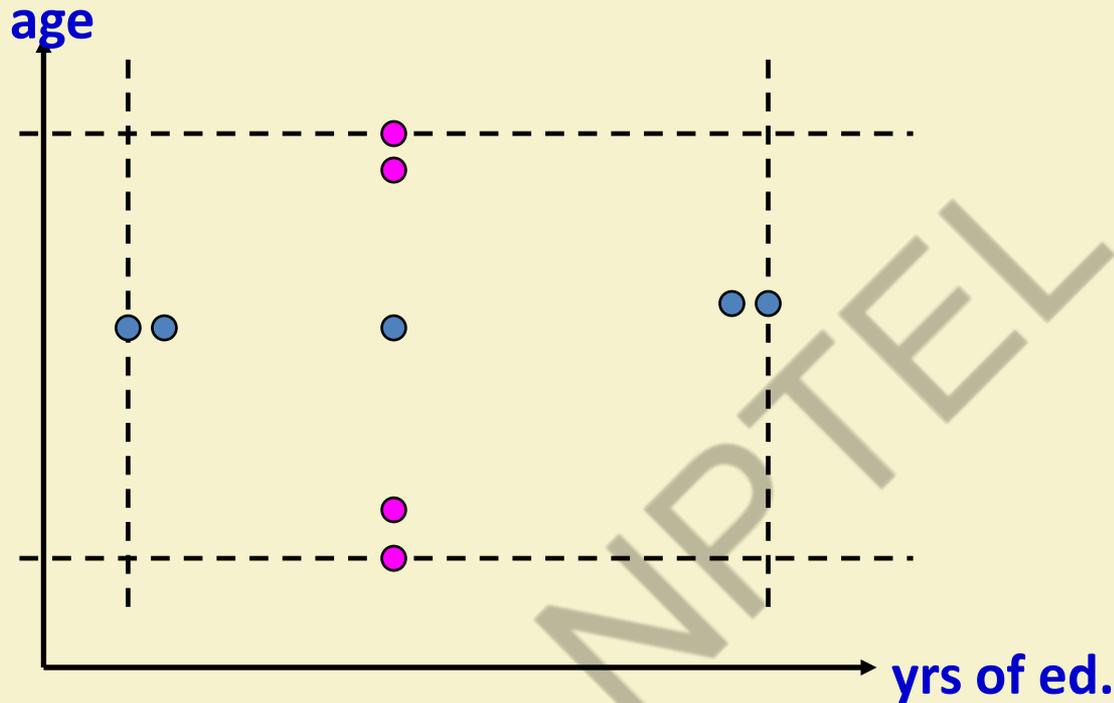
coverage of input data: yrs of ed

1. n = 1 ; age = whatever(37)
2. n = 2; age = whatever
3. n = 12; age = whatever
4. n = 22; age = whatever
5. n = 23; age = whatever



coverage of input data: age

1. n = 12; age = 1
2. n = 12; age = 2
3. n = 12; age = 37
4. n = 12; age = 149
5. n = 12; age = 150



2 –
Independent
inputs

- Note that there needs to be only 9 test cases for 2 independent inputs.
- In general, need $(4z + 1)$ test cases for z independent inputs.

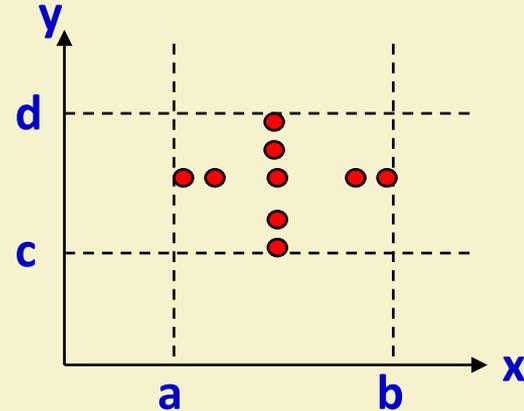
Boundary Value Test

Given $f(x, y)$ with constraints \longrightarrow

$$a \leq x \leq b$$
$$c \leq y \leq d$$

Boundary Value analysis focuses on the boundary of the input space to identify test cases.

Defined as input variable value at min, just above min, a nominal value, just above max, and at max.



Weak Testing: Single Fault Assumption

- **Premise:** “Failures rarely occur as the result of the simultaneous occurrence of two (or more) faults”
- Under this:
 - **Hold the values of all but one variable at their nominal values, and let that one variable assume its extreme values.**

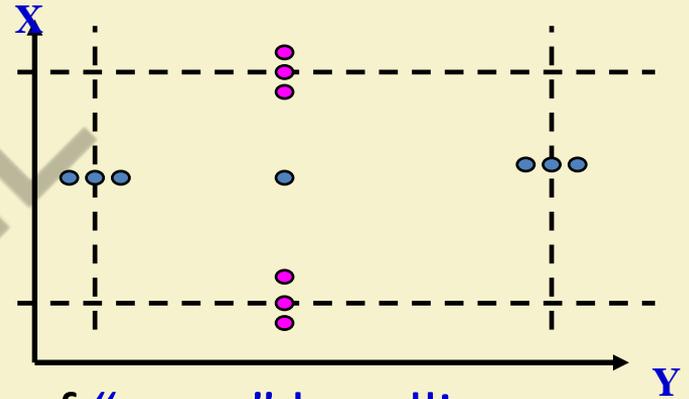
Boundary Value Analysis: Robustness

- Numeric values are often entered as strings :
 - Internally converted to numbers `[int x = atoi(str); val=x-48;]`
- This conversion requires the program to distinguish between digits and non-digits
- A boundary case to consider: Will the program accept / and : as digits?

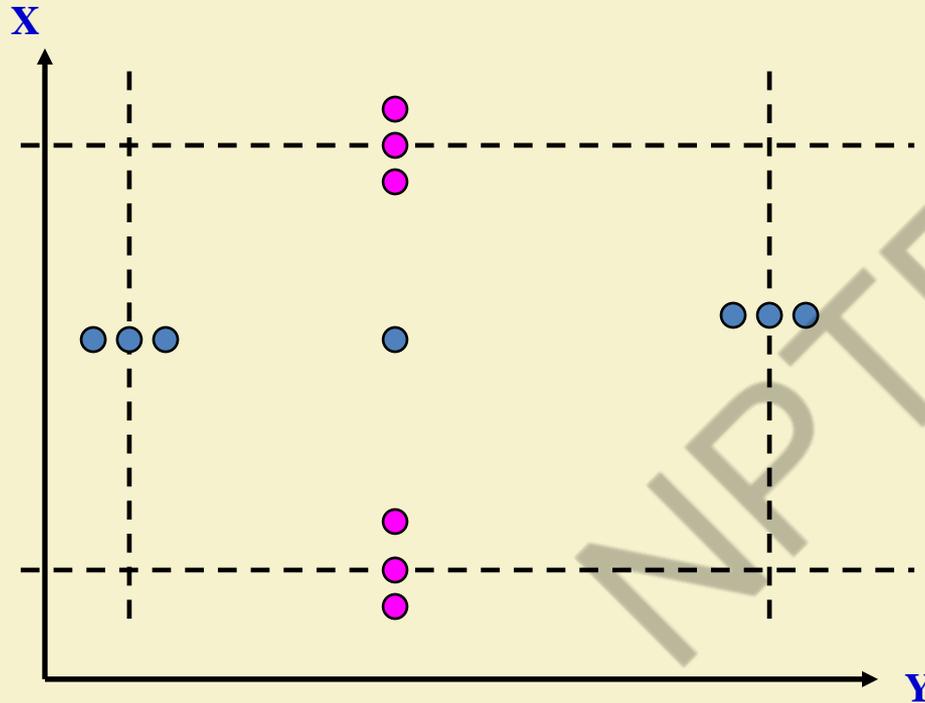
Char	/	0	1	2	3	4	5	6	7	8	9	:
ASCII	47	48	49	50	51	52	53	54	55	56	57	58

Robustness testing

- This is just an extension of the Boundary Values to include invalid values:
 - Less than minimum
 - Greater than maximum
- There are 7 test cases for each input
- The testing of robustness is really a test of “error” handling.
 1. *Did we anticipate the error situations?*
 2. *Do we issue informative error messages?*
 3. *Do we support some kind of recovery from the error?*



2 – independent inputs for robustness test

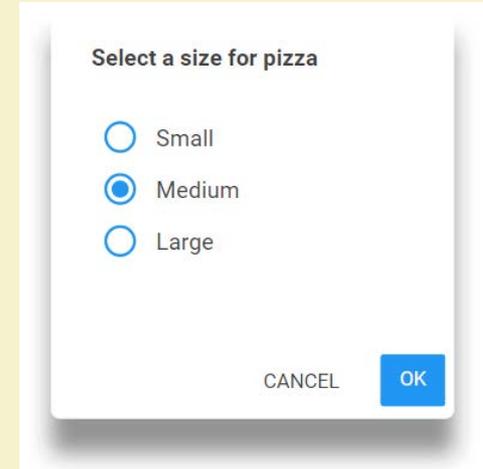


•Note that there needs to be only 13 test cases for 2 independent variables or inputs.

•In general, there will be $(6n+1)$ test cases for n independent inputs.

Some Limitations of Boundary Value Testing

- How to handle a set of values?
- How about set of Boolean variables?
 - True
 - False
- May be radio buttons
- What about a non-numerical variable whose values are text?



Quiz: BB Test Design

- Design black box test suite for a function that solves a quadratic equation of the form $ax^2+bx+c=0$.

- Equivalence classes
 - Invalid Equation
 - Valid Equation: Roots?

