Unit 2: Functional Testing

Boundary value Testing
Equivalence class Testing
Decision Table Based Testing

PRESENTED BY:
MR. C. R. BELAVI
DEPT. OF CSE, HSIT, NIDASOSHI

Subject Code: 10CS842	**I.A. Marks : 25**
Hours/Week : 04	Exam Hours: 03
Total Hours : 52	Exam Marks: 100

UNIT 2

Boundary Value Testing, Equivalence Class Testing, Decision Table-Based Testing: Boundary value analysis, Robustness testing, Worst-case testing, Special value testing, Examples, Random testing, Equivalence classes, Equivalence test cases for the triangle problem, NextDate function, and the commission problem, Guidelines and observations. Decision tables,
Test cases for the triangle problem, NextDate function, and the commission problem, Guidelines and observations.

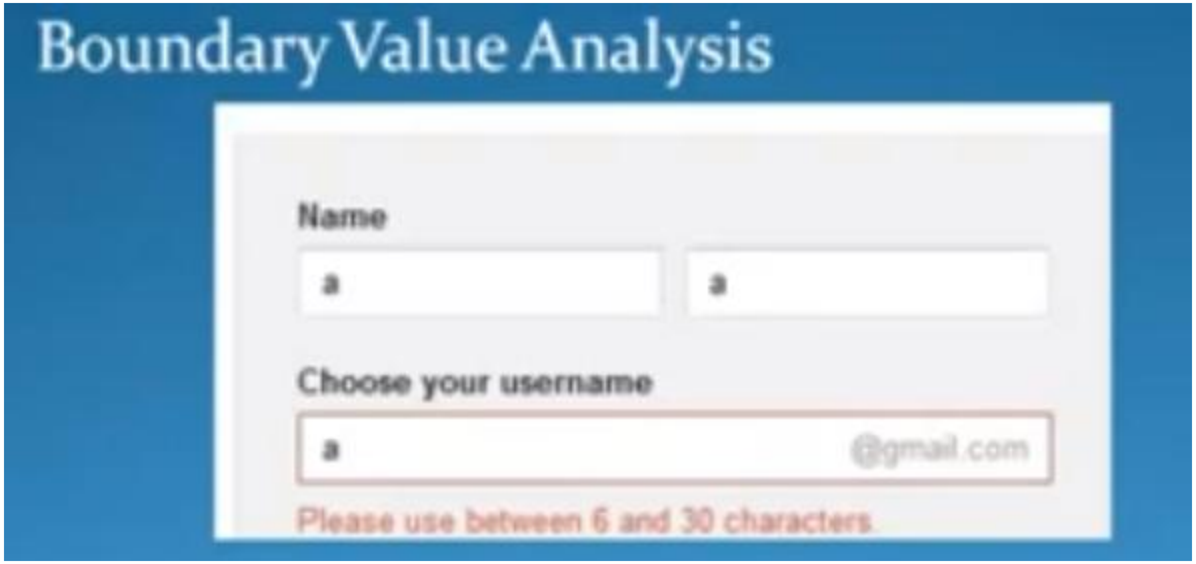
To Understand fundamental concepts in software testing, including software testing objectives, process, criteria, strategies, and methods.
To discuss various types of software testing and its techniques
To list out various tools which can be used for automating the testing process
To Understand various software quality standards for establishing quality environment
To Analyze planning , monitoring the process and Documentation

contents

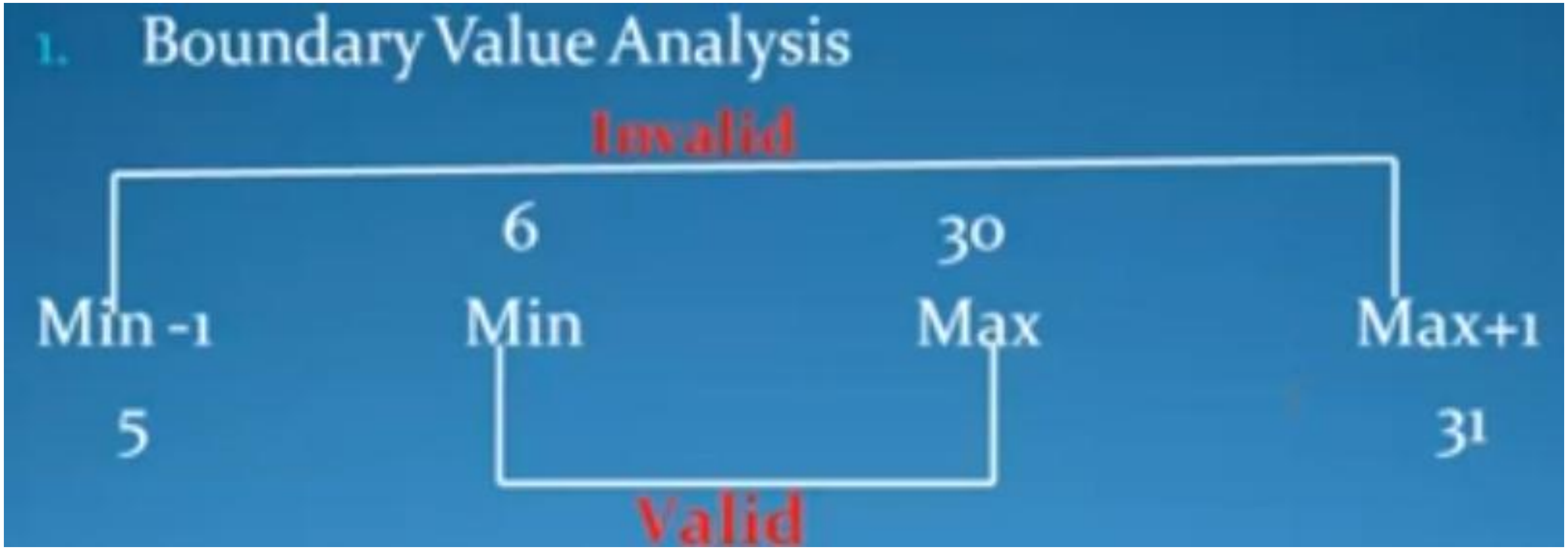
- Boundary Value Testing
  - Boundary Value Analysis
    - *Generalizing Boundary Value Analysis: variable 4n+1 and range*
    - *Limitations of Boundary Value Analysis: independent and physical quantity.*
  - Robustness Testing: Extrema value are exceeded
  - Worst Case Testing: more than one variable has extreme value
  - Special Value Testing: Tester uses his domain knowledge, experience.

Boundary Value Testing

- Any program can be considered to be a ***function*** in the sense that ***prog. I/p*** form its ***domain*** & prog. ***o/p*** form its ***range.***
- *I*nput domain testing is the best known functional testing technique.



For valid user name it should consist characters in the range from 6 to 30



Based on 5 elements values of BVA: min-(5) min(6), min+(7), nom(12), max-(29),max(30),max+(31)

Boundary Value Analysis

- When function F is implemented as a pogram, the input variables x1 & x2 will have some boundaries

  $F(x_1, x_2), a \leq x_1 \leq b, c \leq x_2 \leq d$

  [a,b] [c,d] are ranges of x1 & x2.

- Strongly typed languages (Ada, Pascal) permit such variable range.

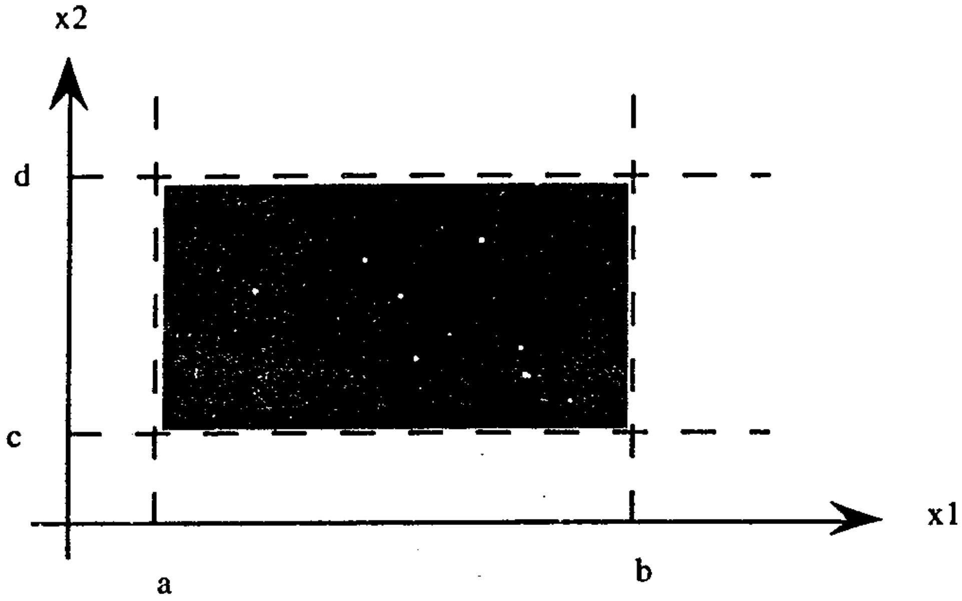


Figure 5.1 **Input domain of a function of two variables.**

- Input space(domain) of our function F is shown above.
- Any point within the shaded rectangle is a legitimate input to the function F.
- Boundary value analysis focuses on the boundary of the input space to identify test cases.

Cont.,

- Errors tend to occur near the extreme values of an input variable
  - e.g. loop conditions (< instead of ≤), counters
- Basic idea: use input variable values at their minimum **(min),** just above the minimum **(min+),** a nominal value **(nom),** just below their maximum **(max-),** and at their maximum **(max).**
- Testing tool (T) generates such Test Cases for Properly specified program. min, min+, max-, max.

Cont.,

- The boundary value analysis test cases are obtained by holding the values of all but one variable at their nominal values, and letting that variable assume its extreme values

$<x_{1nom}, x_{2min}>$
$<x_{1nom}, x_{2min+}>$
$<x_{1nom}, x_{2nom}>$
$<x_{1nom}, x_{2max-}>$
$<x_{1nom}, x_{2max}>$
$<x_{1min}, x_{2nom}>$
$<x_{1min+}, x_{2nom}>$
$<x_{1nom}, x_{2nom}>$
$<x_{1max-}, x_{2nom}>$
$<x_{1max}, x_{2nom}>$

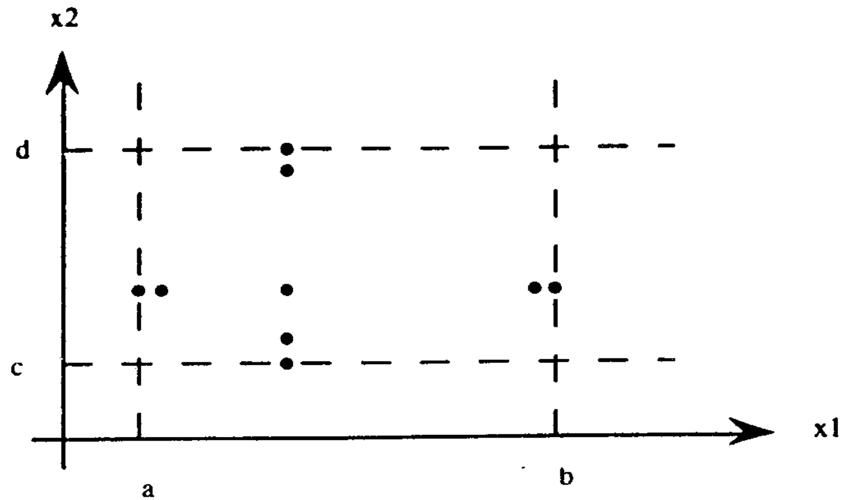


Figure 5.2 Boundary value analysis test cases for a function of two variables.

Generalizing Boundary value Analysis

- Generalized in 2 ways
  - No of variables.
  - Kinds of ranges.
- For a function of n variables, boundary value analysis yields 4n+1 unique test cases.

Conti.,

- By the kinds of ranges, depends on the type (nature) of the variables
  - Variables have discrete, bounded values
    - e.g. NextDate function, commission problem
  - Variables have no explicit bounds
    - Create "artificial" bounds
    - e.g. triangle problem
  - Boolean variables
    - Decision table-based testing
  - Logical variables (bound to a value or another logic variable)
    - e.g. PIN and transaction type in SATM System

Limitations of Boundary value Analysis

- Boundary value analysis works well when the program to be tested is a function of several *independent* variables that represent bounded *physical* quantities.
  - e.g. NextDate test cases are inadequate (little stress on February, dependencies among month, day, and year)
  - e.g. variables refer to physical quantities, such as temperature, air speed, load etc. {Sky Harbour International Airport 120 deg F eg.)

Robustness Testing

- Simple extension of boundary value analysis
- In addition to the five boundary value analysis values of a variable, see what happens when the extrema are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min-)
- Focuses on the expected outputs
  - e.g. exceeding load capacity of a public elevator
  - May 32 we expect error message.
- Forces attention on exception handling

Robustness Testing

Considering min- and max+ values along with 5 elements of BVA

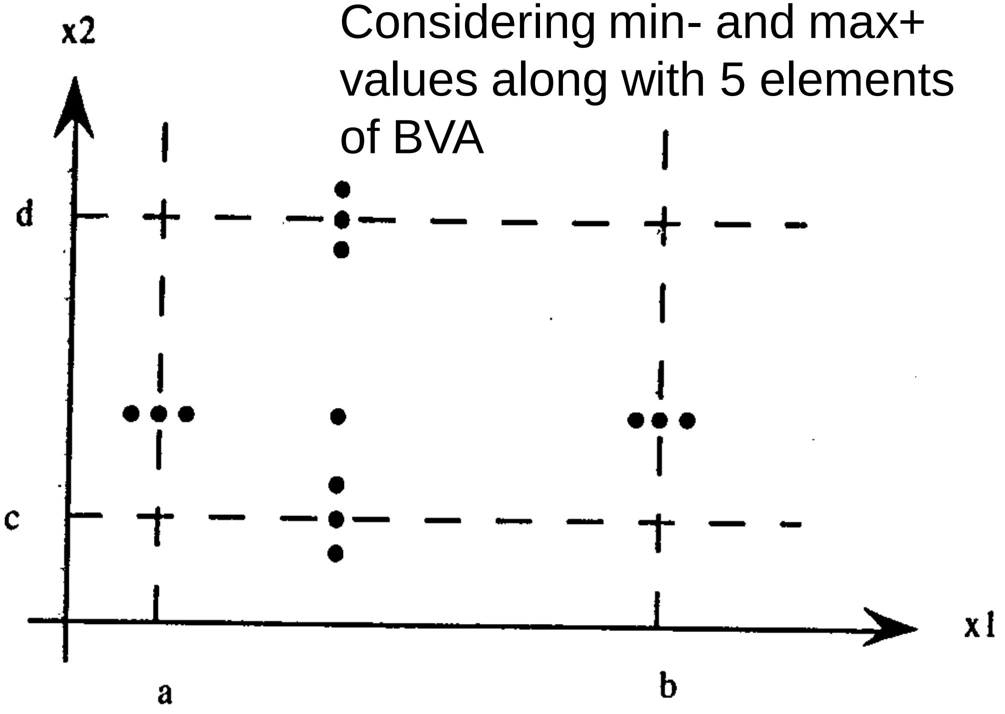


Figure 5.3 Robustness test cases for a function of two variables.

Worst-Case Testing

- Worst case analysis: more than one variable has an extreme value
- Procedure:
  - For each variable create the set <min, min+, nom, max-, max>
  - Take the Cartesian product of these sets to generate test cases
- More thorough than boundary value analysis
- Represents more effort
  - For n variables → 5^n test cases (as opposed to 4n+1 test cases for boundary value analysis)

Worst Case Testing

Taking Cartesian product of
X1:min, min+, nom, max-, max
X2:min,min+, nom, max-, max

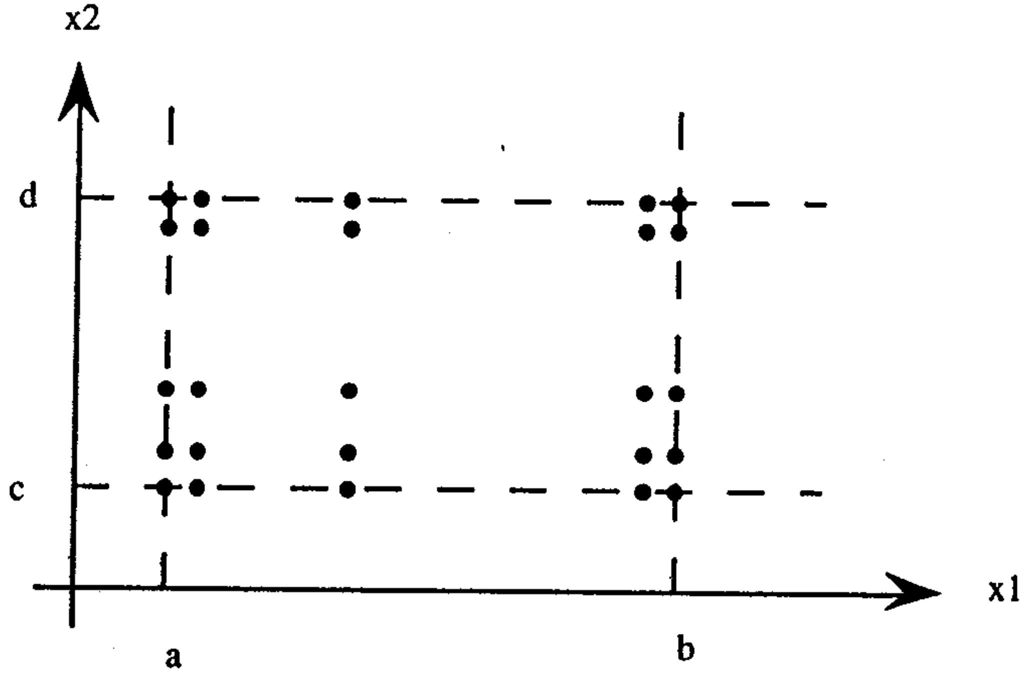


Figure 5.4 **Worst-case test cases for a function of two variables.** 5^n

Combination of Robust and worst case Testing. 7^n

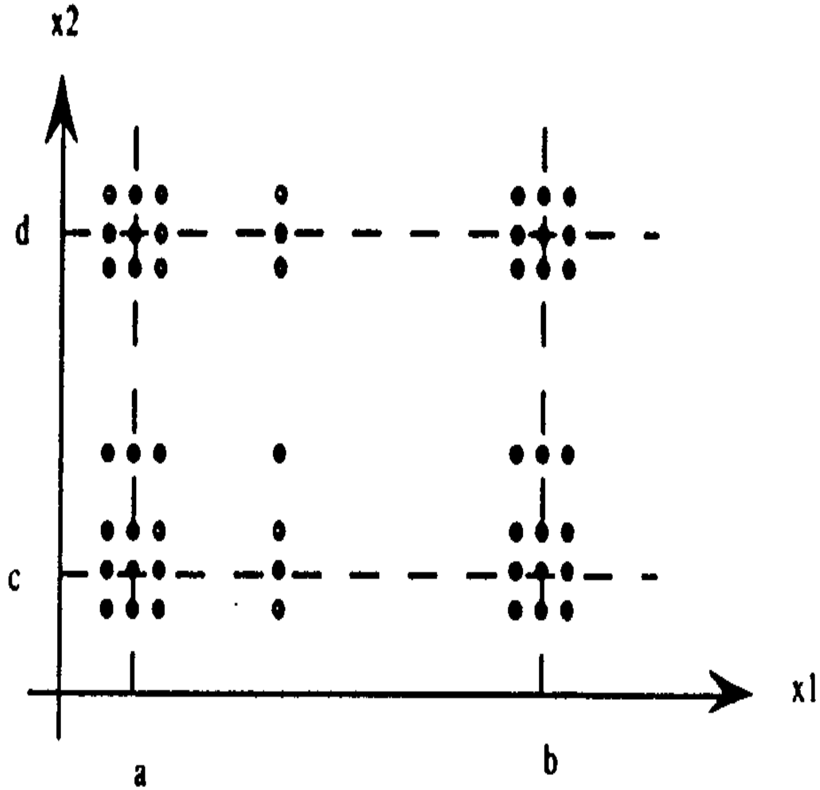


Figure 5.5 Robust worst-case test cases for a function of two variables.

Special value testing

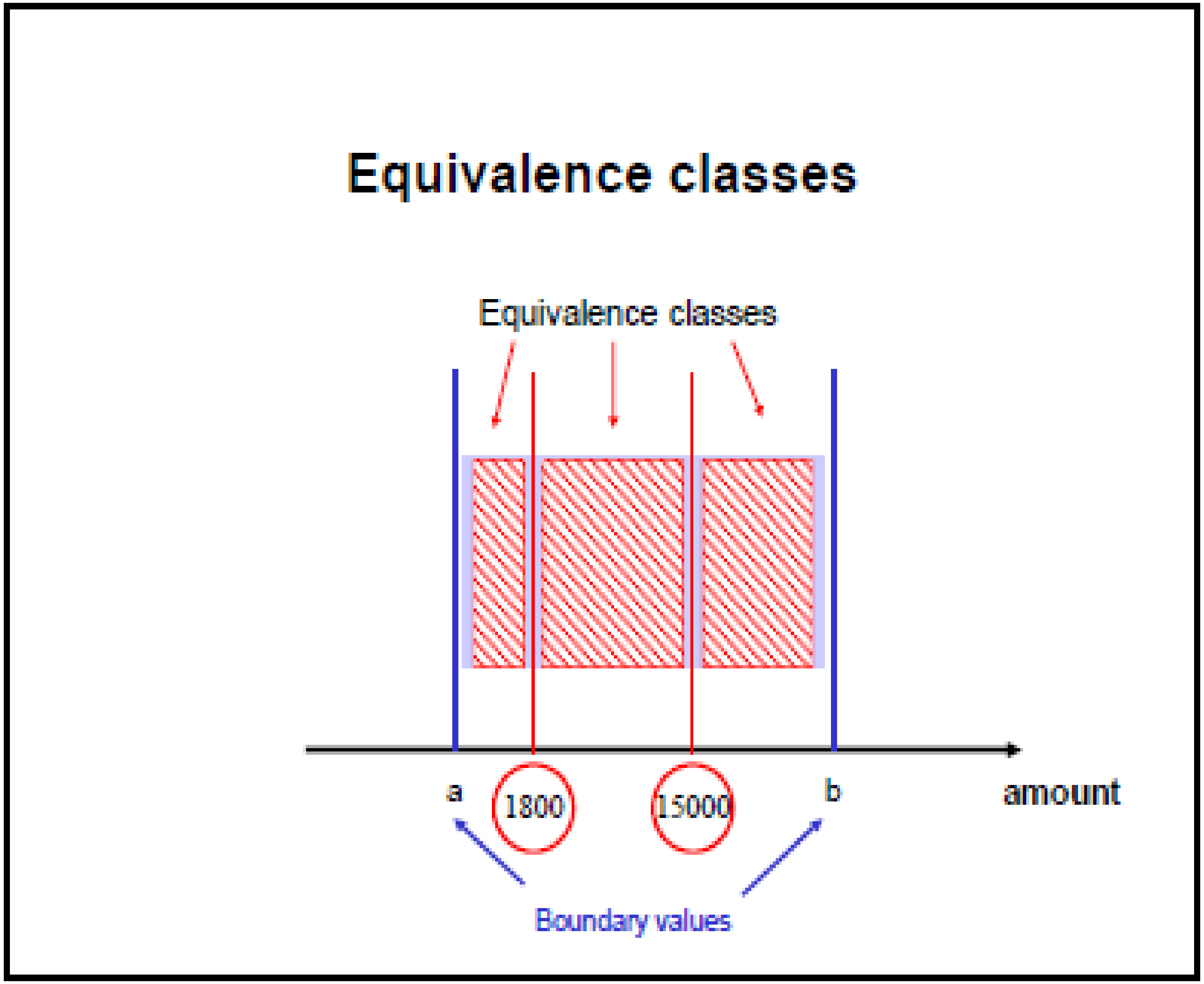
- The most widely practiced form of functional testing
- Most intuitive, least uniform, no guidelines
- The tester uses his/her domain knowledge, experience with similar programs, "ad hoc testing"
- It is dependent on the abilities of the tester
- Even though it is highly subjective, it often results in a set of test cases which is more effective in revealing faults than the test sets generated by the other methods

Contents

- Equivalence class.
  - Weak normal equivalence class testing.
  - Strong normal equivalence class testing.
  - Weak Robust equivalence class testing.
  - Strong Robust equivalence class testing.

Equivalence class Testing

– amount <= 1800
– 1800 < amount < 15000
– amount >= 15000

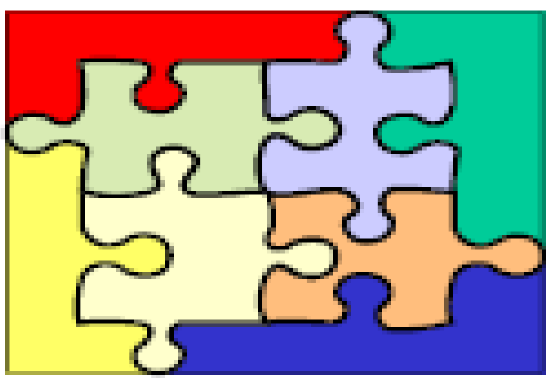


Equivalence class testing

- We need to test only one value from each equivalence class; testing more would be redundant

- Equivalence classes help us to design tests which ensure
  - Completeness
  - Non-redundancy

Domain set A



Completeness

$A = A1 \cup A2 \cup \ldots \cup A8$

Non-redundancy

$i \neq j \Rightarrow Ai \cap Aj = \varnothing$

Equivalence classes

- Motivations
  - Have a sense of complete testing
  - Avoid redundancy
- Equivalence classes form a partition of a set, where partition refers to a collection of mutually disjoint subsets whose union is the entire set (completeness, non-redundancy)
- The idea is to identify test cases by using one element from each equivalence class
- The key is the choice of the equivalence relation that determines the classes

Equivalence class Testing

- When Function F is implemented as a program, the input variables x1,x2 will have boundaries

$a \le x_1 \le d$, with intervals $[a, b)$, $[b, c)$, $[c, d]$

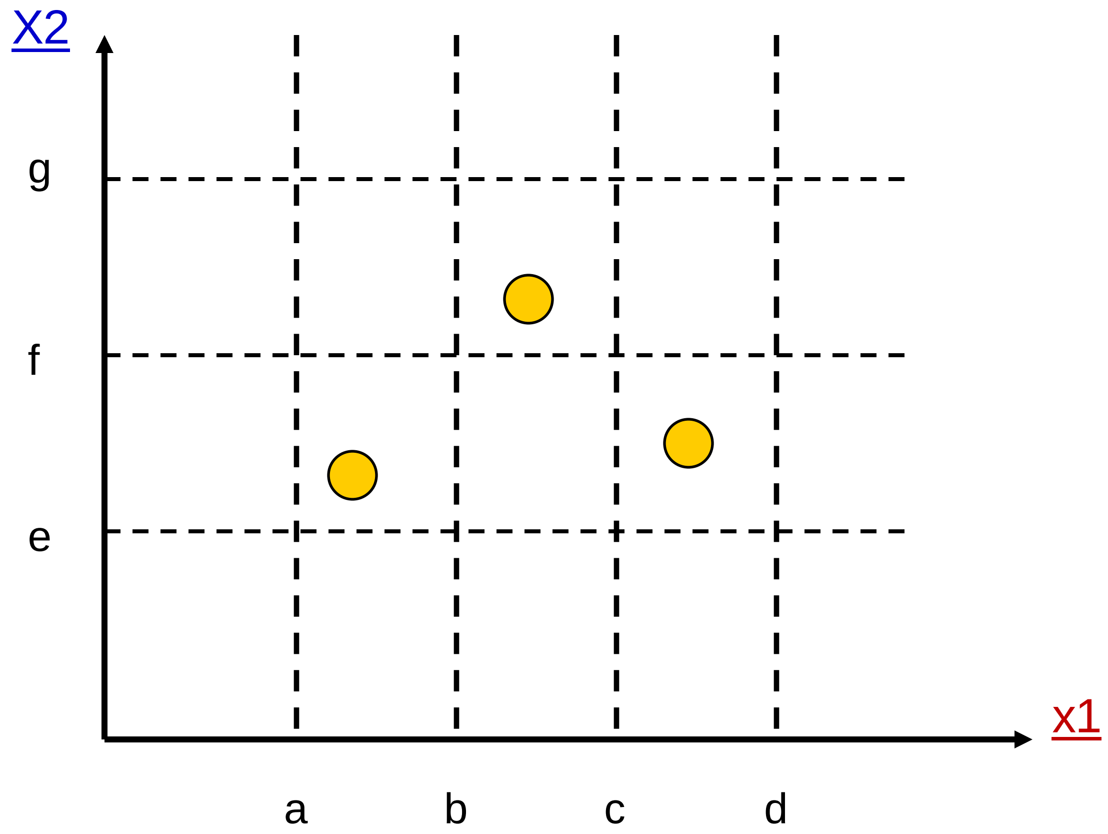
$e \le x_2 \le g$, with intervals $[e, f)$, $[f, g]$

Invalid values of x_1 and x_2 are: $x_1 < a$, $x_1 > d$, and $x_2 < e$, $x_2 > g$.

Weak Normal Equivalence class Testing

- 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.
- Partition the test cases of each input variable separately into one of the different equivalent classes.
- Choose the test case from each of the equivalence classes for each input variable independently of the other input variable
- Using 1 variable from each equivalence class(interval) in a test case.

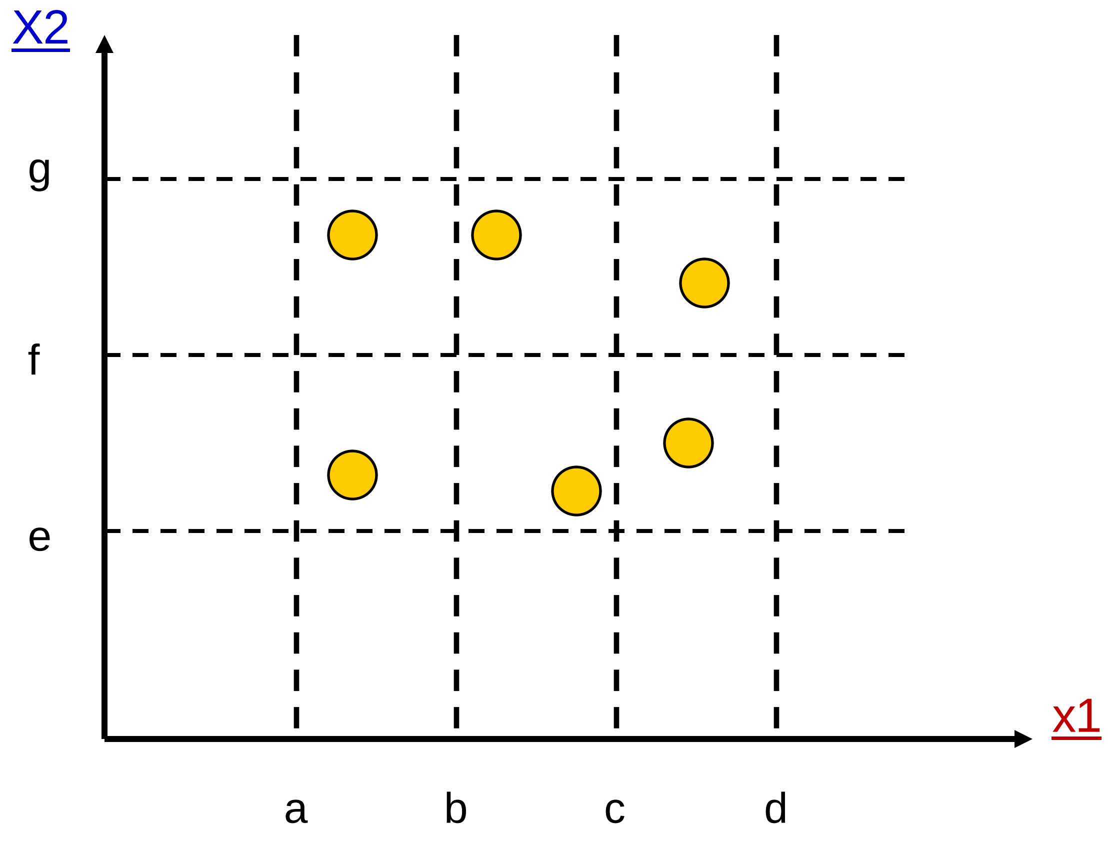
Weak Normal Equivalence class test cases



Strong Normal Equivalence testing

- **Multi Fault assumption.**
- We need Test cases from each element of the Cartesian product of the equivalence classes.
- The Cartesian product guarantees that we have a notion of completeness in two senses
  - We cover all the equivalence classes,
  - We have 1 of each possible combination of inputs.

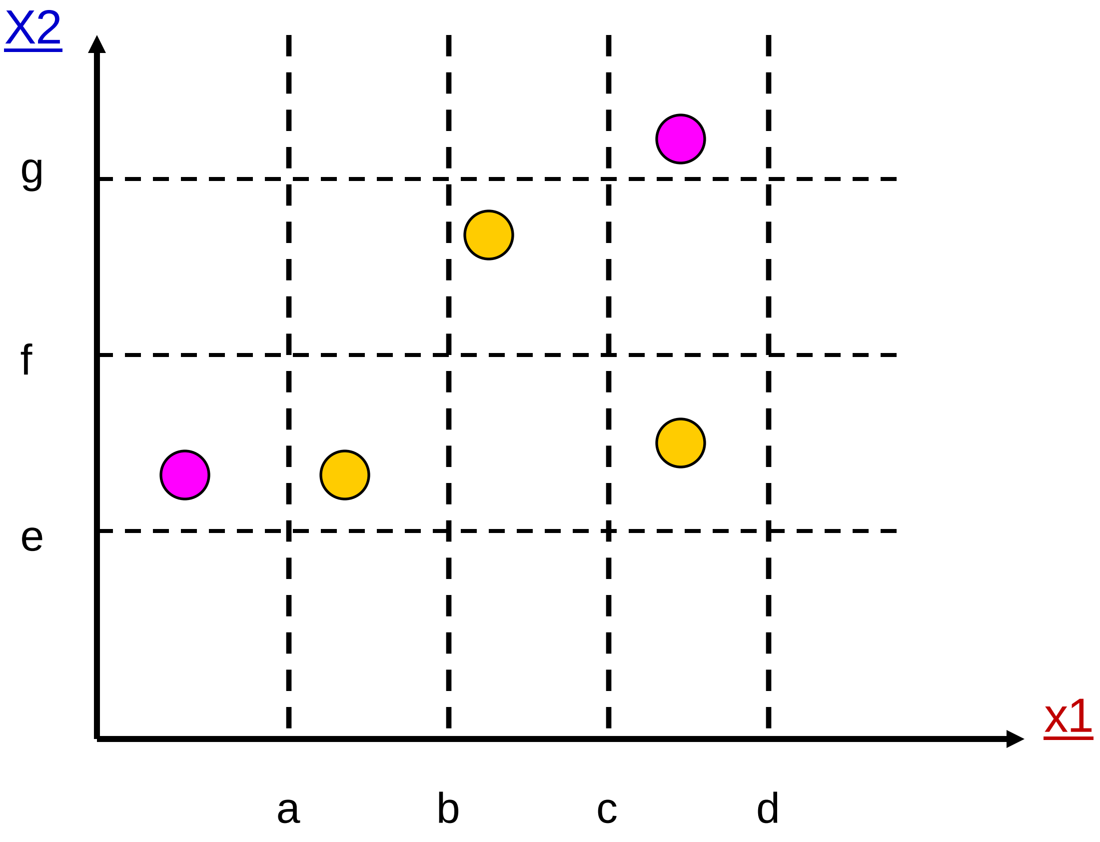
Strong Normal Equivalence class test cases



Weak Robust Equivalence class Testing

- Up to now we have only considered partitioning the *valid* input space.
- "Weak robust" is similar to "weak normal" equivalence test except that the *invalid* input variables are now considered.
- The robust part comes from consideration of invalid values, & the weak part refers to the single fault assumption.

weak robust Equivalence class test cases



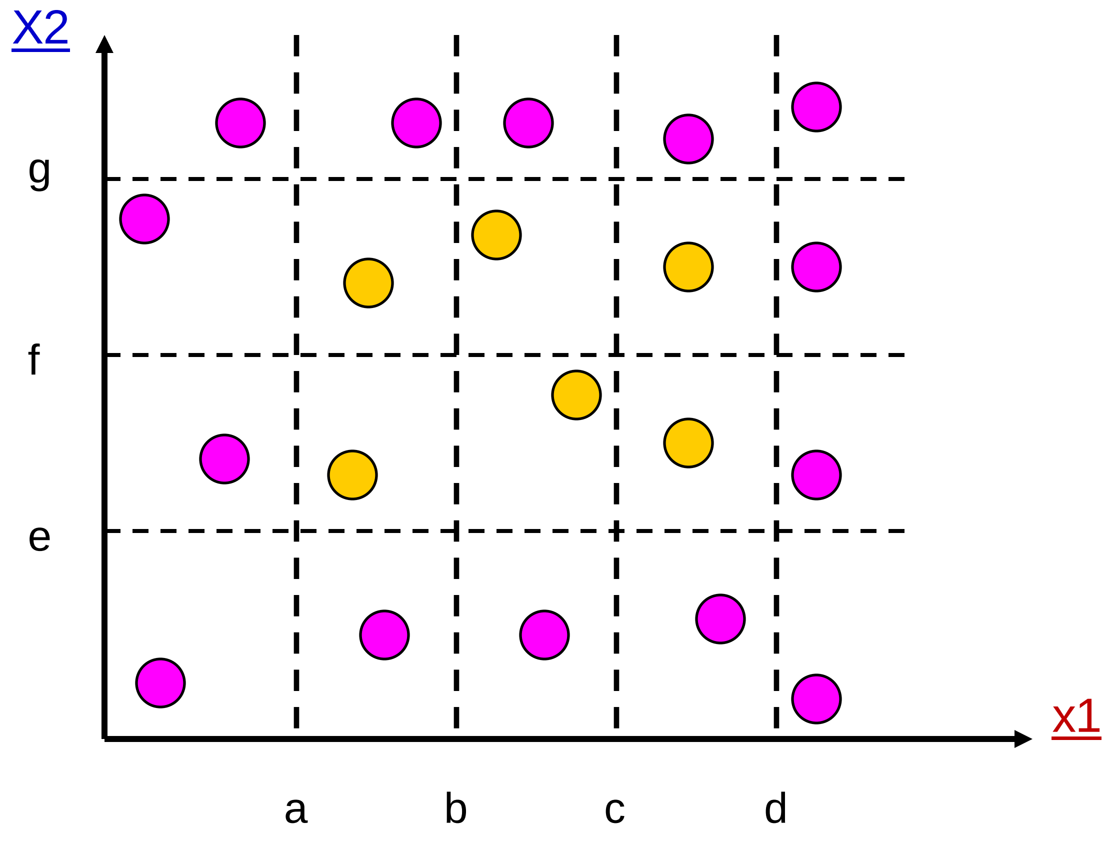
Cont.,

- 2 problems occur with robust equivalence testing.
  - Specification do not define what the expected output for an invalid input should be.
  - Strongly typed languages eliminate the need for the consideration of invalid inputs.

Strong Robust Equivalence Testing

- Robust part comes from consideration of invalid values,
- Strong part refers to the multiple fault assumption.
- We obtain test cases from each element of the Cartesian product of all the equivalence classes

Strong robust Equivalence class test cases



content

- Equivalence class test cases for
  - Triangle problem
  - NextDate Function
  - Commission problem

Equivalence class Test Cases for Triangle problem

R1 = {<a, b, c> : the triangle with sides a, b, and c is equilateral}

~~R2 = {<a, b, c> : the triangle with sides a, b, and c is isosceles}~~

R3 = {<a, b, c> : the triangle with sides a, b, and c is scalene}

R4 = {<a, b, c> : sides a, b, and c do not form a triangle}

The four weak normal equivalence class test cases are:

Test Case	*a*	*b*	*c*	*Expected Output*
WN1	5	5	5	Equilateral
WN2	2	2	3	Isosceles
WN3	3	4	5	Scalene
WN4	4	1	2	Not a Triangle

Considering the invalid values for a, b, and c yields the following additional weak robust equivalence class test cases:

Test Case	*a*	*b*	*c*	*Expected Output*
WR1	–1	5	5	Value of a is not in the range of permitted values
WR2	5	–1	5	Value of b is not in the range of permitted values
WR3	5	5	–1	Value of c is not in the range of permitted values

Test Case	*a*	*b*	*c*	*Expected Output*
WR4	201	5	5	Value of a is not in the range of permitted values
WR5	5	201	5	Value of b is not in the range of permitted values
WR6	5	5	201	Value of c is not in the range of permitted values

Here is one "corner" of the cube in 3-space of the additional strong robust equivalence class test cases:

Test Case	*a*	*b*	*c*	*Expected Output*
SR1	–1	5	5	Value of a is not in the range of permitted values
SR2	5	–1	5	Value of b is not in the range of permitted values
SR3	5	5	–1	Value of c is not in the range of permitted values
SR4	–1	–1	5	Values of a, b are not in the range of permitted values
SR5	5	–1	–1	Values of b, c are not in the range of permitted values
SR6	–1	5	–1	Values of a, c are not in the range of permitted values
SR7	–1	–1	–1	Values of a, b, c are not in the range of permitted values

Equivalence Class Test Cases for NextDate Function

$M1 = \{month : 1 \leq month \leq 12\}$
$D1 = \{day : 1 \leq day \leq 31\}$
$Y1 = \{year : 1812 \leq year \leq 2012\}$

The invalid equivalence classes are:

$M2 = \{month : month < 1\}$
$M3 = \{month : month > 12\}$
$D2 = \{day : day < 1\}$
$D3 = \{day : day > 31\}$
$Y2 = \{year : year < 1812\}$
$Y3 = \{year : year > 2012\}$

Because the number of valid classes equals the number of independent variables, only one weak normal equivalence class test case occurs, and it is identical to the strong normal equivalence class test case:

Case ID	*Month*	*Day*	*Year*	*Expected Output*
WN1, SN1	6	15	1912	6/16/1912

Here is the full set of weak robust test cases:

Case ID	*Month*	*Day*	*Year*	*Expected Output*
WR1	6	15	1912	6/16/1912
WR2	–1	15	1912	Value of month not in the range 1..12
WR3	13	15	1912	Value of month not in the range 1..12
WR4	6	–1	1912	Value of day not in the range 1..31
WR5	6	32	1912	Value of day not in the range 1..31
WR6	6	15	1811	Value of year not in the range 1812..2012
WR7	6	15	2013	Value of year not in the range 1812..2012

the additional strong robust equivalence class test cases:

Case ID	*Month*	*Day*	*Year*	*Expected Output*
SR1	–1	15	1912	Value of month not in the range 1..12

Case ID	*Month*	*Day*	*Year*	*Expected Output*
SR2	6	–1	1912	Value of day not in the range 1..31
SR3	6	15	1811	Value of year not in the range 1812..2012
SR4	–1	–1	1912	Value of month not in the range 1..12 Value of day not in the range 1..31
SR5	6	–1	1811	Value of day not in the range 1..31 Value of year not in the range 1812..2012
SR6	–1	15	1811	Value of month not in the range 1..12 Value of year not in the range 1812..2012
SR7	–1	–1	1811	Value of month not in the range 1..12 Value of day not in the range 1..31 Value of year not in the range 1812..2012

Equivalence Classes

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year = 2000}
Y2 = {year : year is a leap year}
Y3 = {year : year is a common year}

Equivalence Class Test Cases

Case ID	*Month*	*Day*	*Year*	*Expected Output*
WN1	6	14	2000	6/15/2000
WN2	7	29	1996	7/30/1996
WN3	2	30	2002	2/31/2002 (impossible date)
WN4	6	31	2000	7/1/2000 (impossible input date)

Strong Normal Equivalence test case

Case ID	*Month*	*Day*	*Year*	*Expected Output*
SN1	6	14	2000	6/15/2000
SN2	6	14	1996	6/15/1996
SN3	6	14	2002	6/15/2002
SN4	6	29	2000	6/30/2000
SN5	6	29	1996	6/30/1996
SN6	6	29	2002	6/30/2002
SN7	6	30	2000	6/31/2000 (impossible date)
SN8	6	30	1996	6/31/1996 (impossible date)
SN9	6	30	2002	6/31/2002 (impossible date)
SN10	6	31	2000	7/1/2000 (invalid input)
SN11	6	31	1996	7/1/1996 (invalid input)
SN12	6	31	2002	7/1/2002 (invalid input)
SN13	7	14	2000	7/15/2000
SN14	7	14	1996	7/15/1996
SN15	7	14	2002	7/15/2002
SN16	7	29	2000	7/30/2000
SN17	7	29	1996	7/30/1996

Equivalence Class Test case for commission problem

The valid classes of the input variables are:

$L1 = \{locks : 1 \leq locks \leq 70\}$
$L2 = \{locks = -1\}$
$S1 = \{stocks : 1 \leq stocks \leq 80\}$
$B1 = \{barrels : 1 \leq barrels \leq 90\}$

The corresponding invalid classes of the input variables are:

$L_3 = \{locks : locks = 0 \text{ OR } locks < -1\}$
$L_4 = \{locks : locks > 70\}$
$S2 = \{stocks : stocks < 1\}$
$S3 = \{stocks : stocks > 80\}$
$B2 = \{barrels : barrels < 1\}$
$B3 = \{barrels : barrels > 90\}$

Strong Robust equivalence Test cases

Case ID	*Locks*	*Stocks*	*Barrels*	*Expected Output*
SR1	–1	40	45	Value of Locks not in the range 1..70
SR2	35	–1	45	Value of Stocks not in the range 1..80
SR3	35	40	–1	Value of Barrels not in the range 1..90
SR4	–1	–1	45	Value of Locks not in the range 1..70 Value of Stocks not in the range 1..80
SR5	–1	40	–1	Value of Locks not in the range 1..70 Value of Barrels not in the range 1..90
SR6	35	–1	–1	Value of Stocks not in the range 1..80 Value of Barrels not in the range 1..90
SR7	–1	–1	–1	Value of Locks not in the range 1..70 Value of Stocks not in the range 1..80 Value of Barrels not in the range 1..90

Output range equivalence class test cases

$$sales = 45 \times locks + 30 \times stocks + 25 \times barrels$$

We could define equivalence classes of three variables by commission ranges:

S1 = {<locks, stocks, barrels> : sales ≤ 1000}

S2 = {<locks, stocks, barrels> : 1000 < sales ≤ 1800}

S3 = {<locks, stocks, barrels> : sales > 1800}

Output Range Equivalence Class Test Cases

Test Case	Locks	Stocks	Barrels	Sales	Commission
OR1	5	5	5	500	50
OR2	15	15	15	1500	175
OR3	25	25	25	2500	360

Guidelines & 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. (We can "reuse" the effort made in defining the equivalence classes.)

Content

- Decision tables
  - technique
- Test cases for the Triangle problem

Decision table based testing

- Used to represent & analyze complex logical relationships since the early 1960.
- Most rigorous because decision table enforces logical rigor.
- 2 types of methods
  - Cause effect graphing
  - Decision tableau method

Decision Tables - Structure

Conditions - *(Condition stub)*	Condition Alternatives – *(Condition Entry)*
Actions – *(Action Stub)*	Action Entries

- Each condition corresponds to a variable, relation or predicate
- Possible values for conditions are listed among the condition alternatives
  - *Boolean values (True / False) – Limited Entry Decision Tables*
  - *Several values – Extended Entry Decision Tables*
  - *Don't care value*
- Each action is a procedure or operation to perform
- The entries specify whether (or in what order) the action is to be performed

- To express the program logic we can use a limited-entry decision table consisting of 4 areas called the *condition stub*, *condition entry*, *action stub* and the *action entry*:

Condition entry

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No	No	Yes

Condition stub

Action stub

Action Entry

- We can specify *default rules* to indicate the action to be taken when none of the other rules apply.
- When using decision tables as a test tool, default rules and their associated predicates must be explicitly provided.

	Rule5	Rule6	Rule7	Rule8
Condition1	X	No	Yes	Yes
Condition2	X	Yes	X	No
Condition3	Yes	X	No	No
Condition4	No	No	Yes	X
Default action	Yes	Yes	Yes	Yes

Decision Table - Example

Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Heck the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

Printer Troubleshooting

Below table tells about the Condition and action to be taken

Table 7.1 Portions of a Decision Table

Stub	Rule 1	Rule 2	Rules 3, 4	Rule 5	Rule 6	Rules 7, 8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	—	T	F	—
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

Table 7.2 Decision Table for the Triangle Problem

c3: a, b, c form a triangle?	N	Y	Y	Y	Y	Y	Y	Y	Y
c2: a = b?	—	Y	Y	Y	Y	N	N	N	N
c3: a = c?	—	Y	Y	N	N	Y	Y	N	N
c4: b = c?	—	Y	N	Y	N	Y	N	Y	N
a1: Not a triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			

Table 7.3 Refined Decision Table for the Triangle Problem

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	—	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	—	—	F	T	T	T	T	T	T	T	T
c4: a = b?	—	—	—	T	T	T	T	F	F	F	F
c5: a = c?	—	—	—	T	T	F	F	T	T	F	F
c6: b = c?	—	—	—	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

In mutually exclusive only one condition can be performed at a time.

Table 7.4 Decision Table with Mutually Exclusive Conditions

Conditions	*R1*	*R2*	*R3*
c1: month in M1?	T	—	—
c2: month in M2?	—	T	—
c3: month in M3?	—	—	T
a1			
a2			
a3			

Table 7.5 Decision Table for Table 7.3 with Rule Counts

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	—	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	—	—	F	T	T	T	T	T	T	T	T
c4: a = b?	—	—	—	T	T	T	T	F	F	F	F
c5: a = c?	—	—	—	T	T	F	F	T	T	F	F
c6: b = c?	—	—	—	T	F	T	F	T	F	T	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Table 7.6 Rule Counts for a Decision Table with Mutually Exclusive Conditions

Conditions	*R1*	*R2*	*R3*
c1: month in M1	T	—	—
c2: month in M2	—	T	—
c3: month in M3	—	—	T
Rule Count	4	4	4
a1			

Table 7.7 Expanded Version of Table 7.6

Conditions	*1.1*	*1.2*	*1.3*	*1.4*	*2.1*	*2.2*	*2.3*	*2.4*	*3.1*	*3.2*	*3.3*	*3.4*
c1: mo. in M1	T	T	T	T	T	T	F	F	T	T	F	F
c2: mo. in M2	T	T	F	F	T	T	T	T	T	F	T	F
c3: mo. in M3	T	F	T	F	T	F	T	F	T	T	T	T
Rule Count	1	1	1	1	1	1	1	1	1	1	1	1
a1												

Table 7.8 Mutually Exclusive Conditions with Impossible Rules

	1.1	*1.2*	*1.3*	*1.4*	*2.3*	*2.4*	*3.4*	
c1: mo. in M1	T	T	T	T	F	F	F	F
c2: mo. in M2	T	T	F	F	T	T	F	F
c3: mo. in M3	T	F	T	F	T	F	T	F
Rule Count	1	1	1	1	1	1	1	1
a1: Impossible	X	X	X		X			X

Table 7.9 A Redundant Decision Table

Conditions	*1–4*	*5*	*6*	*7*	*8*	*9*
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	X
a2	—	X	X	X	—	—
a3	X	—	X	X	X	X

Table 7.10 An Inconsistent Decision Table

Conditions	*1–4*	*5*	*6*	*7*	*8*	*9*
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—

Table 7.11 Test Cases from Table 7.3

Case ID	*a*	*b*	*c*	*Expected Output*
DT1	4	1	2	Not a Triangle
DT2	1	4	2	Not a Triangle
DT3	1	2	4	Not a Triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

Test cases for NextDate Function

Equivalence classes

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year is a leap year}
Y2 = {year : year is not a leap year}

a1: Too many days in a month
a2: Cannot happen in a non-leap year
a3: Compute the next date

Why many rules were impossible

Test cases for NextDate Function

Table 7.12 First Try Decision Table with 256 Rules

Conditions											
c1: month in M1?	T										
c2: month in M2?		T									
c3: month in M3?			T								
c4: day in D1?											
c5: day in D2?											
c6: day in D3?											
c7: day in D4?											
c8: year in Y1?											
a1: impossible											
a2: next date											

Second Try

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year = 2000}
Y2 = {year : year is a leap year}
Y3 = {year : year is a common year}

Table 7.13 Second Try Decision Table with 36 Rules

	1	2	3	4	5	6	7	8
c1: month in	M1	M1	M1	M1	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D1	D2	D3	D4
c3: year in	—	—	—	—	—	—	—	—
Rule count	3	3	3	3	3	3	3	3
actions								
a1: impossible				X				
a2: increment day	X	X			X	X	X	
a3: reset day			X					X
a4: increment month			X					?
a5: reset month								?
a6: increment year								?

	9	10	11	12	13	14	15	16	17
c1: month in	M3	M3	M3	M3	M3	M3	M3	M3	M3
c2: day in	D1	D1	D1	D2	D2	D2	D3	D3	D4
c3: year in	Y1	Y2	Y3	Y1	Y2	Y3	—	—	—
Rule count	1	1	1	1	1	1	3	3	3
actions									
a1: impossible						X	X	X	X
a2: increment day		X							
a3: reset day	X		X	X	X				
a4: increment month	X		X	X	X				
a5: reset month									
a6: increment year									

Third try

M1 = {month : month has 30 days}
M2 = {month : month has 31 days except December}
M3 = {month : month is December}
M4 = {month : month is February}
D1 = {day : $1 \leq$ day ≤ 27}
D2 = {day : day = 28}
D3 = {day : day = 29}
D4 = {day : day = 30}
D5 = {day : day = 31}
Y1 = {year : year is a leap year}
Y2 = {year : year is a common year}

Table 7.14 Decision Table for the NextDate Function

	1	2	3	4	5	6	7	8	9	10
c1: month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5
c3: year in	—	—	—	—	—	—	—	—	—	—
actions										
a1: impossible					X					
a2: increment day	X	X	X			X	X	X	X	
a3: reset day				X						X
a4: increment month				X						X
a5: reset month										
a6: increment year										

	11	12	13	14	15	16	17	18	19	20	21	22
c1: month in	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2: day in	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: year in	—				—	—	Y1	Y2	Y1	Y2	—	—
actions												
a1: impossible										X	X	X
a2: increment day	X	X	X	X		X	X					
a3: reset day					X			X	X			
a4: increment month								X	X			
a5: reset month					X							
a6: increment year					X							

Table 7.15 Reduced Decision Table for the NextDate Function

	1–3	4	5	6–9	10
c1: month in	M1	M1	M1	M2	M2
c2: day in	D1, D2, D3	D4	D5	D1, D2, D3, D4	D5
c3: year in	—	—	—	—	—
actions					
a1: impossible			X		
a2: increment day	X			X	
a3: reset day		X			X
a4: increment month		X			X
a5: reset month					
a6: increment year					

	11–14	15	16	17	18	19	20	21, 22
c1: month in	M3	M3	M4	M4	M4	M4	M4	M4
c2: day in	D1, D2, D3, D4	D5	D1	D2	D2	D3	D3	D4, D5
c3: year in	—	—	—	Y1	Y2	Y1	Y2	—
actions								
a1: impossible							X	X
a2: increment day	X		X	X				
a3: reset day		X			X	X		
a4: increment month					X	X		
a5: reset month		X						
a6: increment year		X						

If the action sets of 2 rule in a limited entry decision table are identical, there must be 1 condition that allow 2 rules to be combined with a don't care entry

Table 7.16 Decision Table Test Cases for NextDate

Case ID	*Month*	*Day*	*Year*	*Expected Output*
1– 3	April	15	2001	April 16, 2001
4	April	30	2001	May 1, 2001
5	April	31	2001	Impossible
6–9	January	15	2001	January 16, 2001
10	January	31	2001	February 1, 2001
11–14	December	15	2001	December 16, 2001
15	December	31	2001	January 1, 2002
16	February	15	2001	February 16, 2001
17	February	28	2004	February 29, 2004
18	February	28	2001	March 1, 2001
19	February	29	2004	March 1, 2004
20	February	29	2001	Impossible
21, 22	February	30	2001	Impossible

Test cases for commission problem

- Commission problem is not well served by decision table analysis.
- Very little decision logic is used in the problem

Thank you all..?

Thank you all..?