Module-2: Functional Testing

Boundary Value Testing

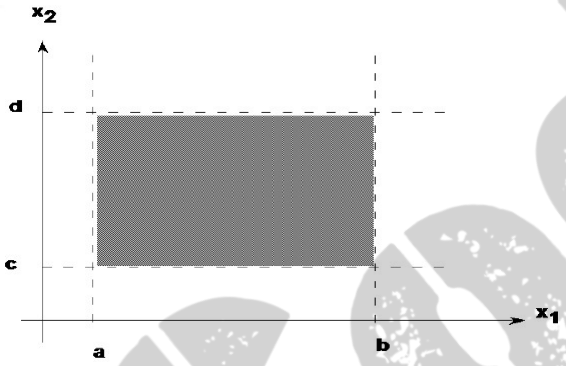
➢ Boundary Value Analysis

- When the **function F** is implemented as a program **of two input variables**, the input variables **x_1 and x_2** will have some boundaries:

$$a \leq x_1 \leq b$$
$$c \leq x_2 \leq d$$

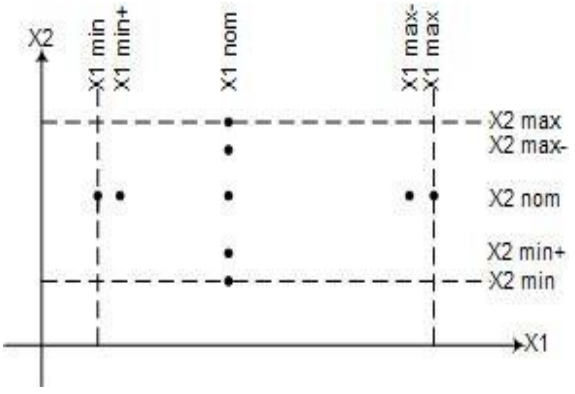
- The **intervals [a, b] and [c, d]** are referred to as the ranges of **x_1 and x_2**, (instead of domain) so we have overloaded the term.

Figure: Input domain of a function of two variables



- Boundary value analysis focuses on the **boundary of the input space** to identify test cases. The basis behind boundary value testing is that **errors tend to occur near the extreme values** of an input variable.
- The basic idea of boundary value analysis is to use input variable values at their **minimum, just above the minimum, a normal value, just below their maximum, and their maximum as shown in figures below.**
- A commercially available testing tool (originally named T) generates such tests cases for a properly specified program.

The values used to test the extremities are	
Min	Minimal
Min+	Just above Minimal
Nom	Average/Nominal/normal
Max-	Just below Maximum
Max	Maximum



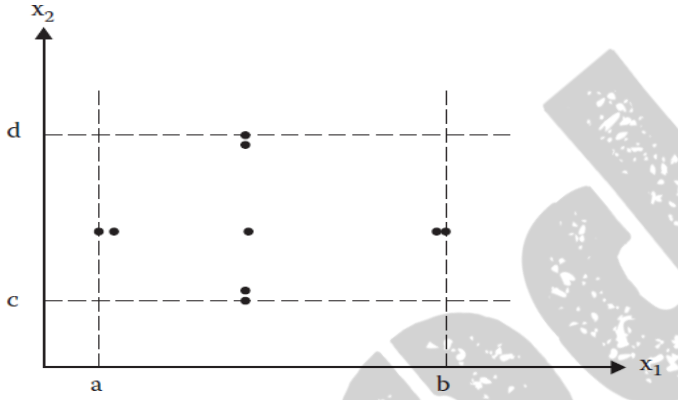
- The boundary value analysis test cases for **function F** of **two variables** are as shown in table and figure below.

Keep X_1 at normal value and vary X_2	Keep X_2 at normal value and vary X_1
$<X_{1nom},X_{2min}>, <X_{1nom}, X_{2min+}>, <X_{1nom}, X_{2nom}>, <X_{1nom},X_{2max-}>, <X_{1nom},X_{2max}>$	$<X_{1min},X_{2nom}>, <X_{1min+},X_{2nom}>, <X_{1max-}, X_{2nom}>, <X_{1max},X_{2nom}>$

✓ Generalizing boundary value analysis

- The basic boundary value analysis technique can be generalized in two ways:
- By the number of variables and the kinds of ranges.
- Generalizing the number of variables is easy. If we have a function of **n variables**, we **hold all but one at the normal values** and let the **remaining variable** assume the **Min, min+, nom, max-, and max values**, repeating this for each variable. Thus, for a function of n variables, boundary value analysis yields **4n+1 unique test case.**
- Boundary value analysis test cases for a function of two variables are shown below. There are **4*2+1=9 unique test cases**.

Figure: Boundary value analysis test cases for a function of two Variables



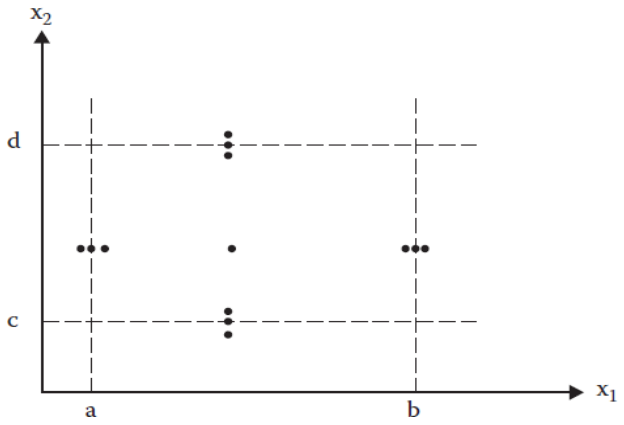
✓ Limitations of Boundary value analysis

- Boundary value analysis works well when the **program is a function of several independent variables** that represent bounded **physical quantities.**
- A quick look at the boundary value analysis test cases for next-date shows them to be inadequate.
  - For example, very little stress occurs on **February and on leap years**. The real problem here is that interesting **dependencies exist among the month, day, and year** variables.
- Boundary value analysis assumes the variables to be truly on **independent.** Even so boundary value analysis happens to catch **end-of-month and end-of-year faults**.
- Boundary value analysis test cases are derived from the extreme of bounded, independent variables that refer to physical quantities, with no consideration of the nature of the function, or of the semantic meaning of the variable.
- When a variable refers to a **physical quantity**, such as **temperature, pressure, air speed, angle of attack, load** and so forth, physical boundaries can be extremely important.

➢ Robustness testing

- Robustness testing is a **simple extension** of boundary value Analysis. Here we add **min-** and **max+** values as additions.
- The boundary value analysis applies directly to robustness testing, especially the generalizations and limitations. The most interesting part of the robustness testing is not with the inputs, but with the **expected outputs**.
- The main value of robustness testing is that it forces attention on **exception handling.**

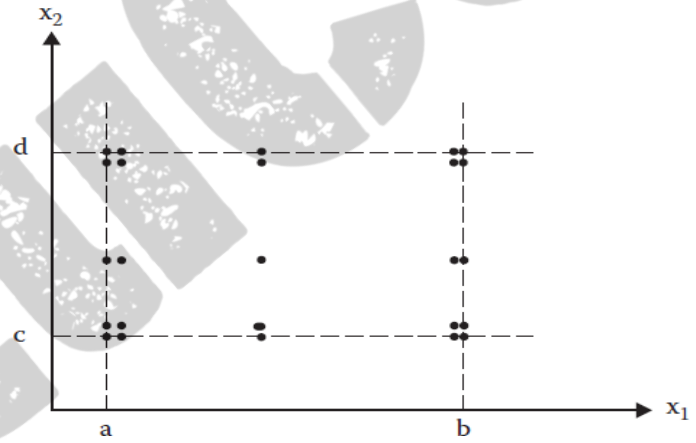
Figure: Robustness test cases for a function of two Variables



➢ Worst-Case testing

- Boundary value analysis makes a **single fault assumption** of reliability theory.
- Rejecting **single fault assumption,** we are interested in what happens when **more than one variable** has an **extreme value.** This is called **worst case analysis**.
- **For each variable**, we start with the **five elements set** that contain the **min, min+, nom, max-,** and **max values**.
- We then take the Cartesian product of the sets to generate test cases. **Worst-Case testing** for a **function** of **n variables generates 5^n test cases.** In this case **5^2= 5*5= 25 test cases.**

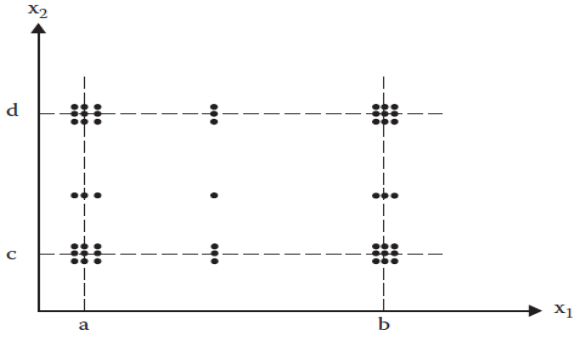
Figure: Worst-Case test cases for a function of two Variables



➢ Robust Worst-case Testing

- Worst-case testing follows the **generalization pattern** of boundary value analysis.
- It also has the same limitations, particularly those related to independence.
- The best application for worst-case testing is where physical variables have **numerous interactions and where failure of the function is extremely costly.**
- For really paranoid testing, we do robust worst- case testing. This involves **Cartesian product of the seven - elements sets** we used in **robustness testing** resulting **in 7^n test cases.** Here we have **7^2=7*7=49 test cases**

Figure: Robust Worst-Case test cases for a function of two Variables



➢ Special value testing

- Special value testing is the most widely practiced form of functional testing.
- It is most intuitive with similar programs, and information about "soft spots" to device test cases. We might also call this **adhoc testing** or **"seat-of-the- pants"** testing.
- No guidelines are used other than to use "best engineering judgment". As a result, special value testing is very dependent on the **abilities** of the tester.
- If an interested tester defines special value test cases for **NextDate**, we would see several test cases involving February 28, February 29 and leap years. Even though special value testing is **highly subjective**, it often results in a **set of test cases** that is more effective in revealing faults than the test cases generated by the **other methods** of software testing.

➢ Examples

✓ Test Cases for the Triangle Problem

- **Table below** contains Boundary Value Analysis Test Cases using 1-200 range. We have noticed that case 3, 8, 13 are identical, so two should be deleted.

Triangle Problem Boundary Value Analysis Test Cases

Case	a	b	c	Expected Output
1	**100**	100	**1**	Isosceles
2	**100**	100	**2**	Isosceles
3	**100**	100	**100**	Equilateral
4	**100**	100	**199**	Isosceles
5	**100**	100	**200**	Not a Triangle
6	**100**	1	**100**	Isosceles
7	**100**	2	**100**	Isosceles
8	**100**	100	**100**	Equilateral
9	**100**	199	**100**	Isosceles
10	**100**	200	**100**	Not a Triangle
11	**1**	100	**100**	Isosceles
12	**2**	100	**100**	Isosceles
13	**100**	100	**100**	Equilateral
14	**199**	100	**100**	Isosceles
15	**200**	100	**100**	Not a Triangle

- **Table below** contains **worst-case** test cases in just "one corner" of the input space cube.
- Remaining values could be generated by setting values of **'a' to 2, 5, 199, 200.**
- Each value of input **'a'** generates **25 test cases** in similar manner as given below and there by **5^3 =125** cases can be generated.

Triangle Problem Worst-Case-Test Cases

Case	a	b	c	Expected Output
1	**1**	1	**1**	Equilateral
2	**1**	1	**2**	Not a Triangle
3	**1**	1	**100**	Not a Triangle
4	**1**	1	**199**	Not a Triangle
5	**1**	1	**200**	Not a Triangle
6	**1**	2	**1**	Not a Triangle
7	**1**	2	**2**	Isosceles
8	**1**	2	**100**	Not a Triangle
9	**1**	2	**199**	Not a Triangle
10	**1**	2	**200**	Not a Triangle
11	**1**	100	**1**	Not a Triangle
12	**1**	100	**2**	Not a Triangle
13	**1**	100	**100**	Isosceles
14	**1**	100	**199**	Not a Triangle
15	**1**	100	**200**	Not a Triangle
16	**1**	199	**1**	Not a Triangle
17	**1**	199	**2**	Not a Triangle
18	**1**	199	**100**	Not a Triangle
19	**1**	199	**199**	Isosceles
20	**1**	199	**200**	Not a Triangle
21	**1**	200	**1**	Not a Triangle
22	**1**	200	**2**	Not a Triangle
23	**1**	200	**100**	Not a Triangle
24	**1**	200	**199**	Not a Triangle
25	**1**	200	**200**	Isosceles

✓ Test Cases for the NextDate Problem

- Table below contains **worst-case** test cases in just **"one corner"** of the input space cube.
- Remaining values could be generated by setting values of input **Month** to **2, 6, 11** and **12**.
- Each value of **input Month** generates 25 test cases in similar manner as given below and thereby **5^3 =125** cases can be generated.

NextDate Worst-Case Test Cases

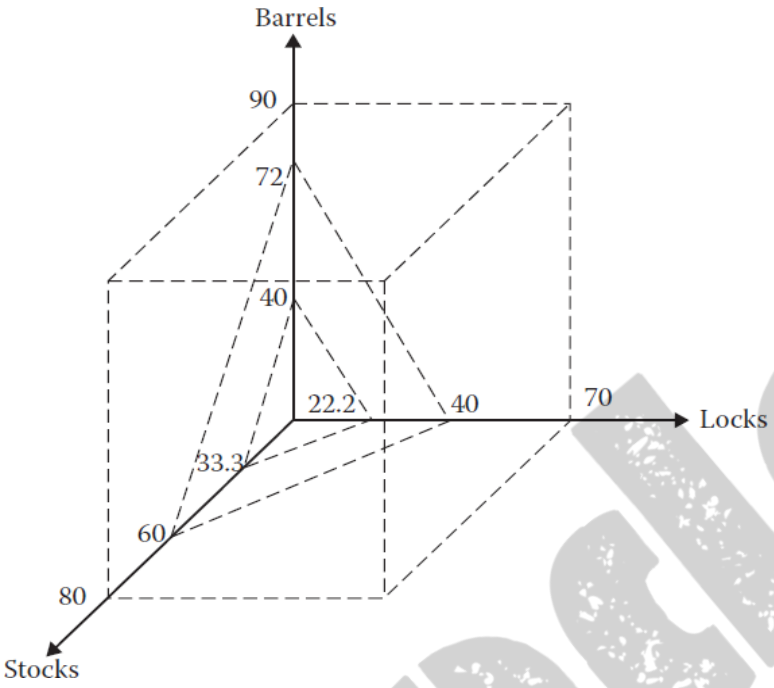
Case	Month	Day	Year	Expected Output
1	**1**	1	**1812**	January 2,1812
2	**1**	1	**1813**	January 2,1813
3	**1**	1	**1915**	January 2,1915
4	**1**	1	**2018**	January 2,2018
5	**1**	1	**2019**	January 2,2019
6	**1**	2	**1812**	January 3,1812
7	**1**	2	**1813**	January 3,1813
8	**1**	2	**1915**	January 3,1915
9	**1**	2	**2018**	January 3,2018
10	**1**	2	**2019**	January 3,2019
11	**1**	15	**1812**	January 16,1812
12	**1**	15	**1813**	January 16,1813
13	**1**	15	**1915**	January 16,1915
14	**1**	15	**2018**	January 16,2018
15	**1**	15	**2019**	January 16,2019
16	**1**	30	**1812**	January 31,1812
17	**1**	30	**1813**	January 31,1813
18	**1**	30	**1915**	January 31,1915
19	**1**	30	**2018**	January 31,2018
20	**1**	30	**2019**	January 31,2019
21	**1**	31	**1812**	February 1,1812
22	**1**	31	**1813**	February 1,1813
23	**1**	31	**1915**	February 1,1915
24	**1**	31	**2018**	February 1,2018
25	**1**	31	**2019**	February 1,2019

✓ Test Cases for the Commission Problem

- The volume in **below plane** corresponds to sales below $1000 threshold. The volume between two planes is the 15% commission range. Part of the reason for using the output range is to determine test cases is that test cases from the **input range** are almost all in the 20% zone.
- We want to find input variable combinations that stress the boundary values: $100, $1000, $1800, and $7800 ie., **min** border points and **max** and then **mid points**.

- Here is where it gets interesting; **case 9** and **case 17** are the **border points**. If we tweak the input variables, we get values **just below** and **just above** the **border**.

Figure: Input space of the commission problem



Commission Problem Output Boundary Value Analysis Test Cases

Case	Locks	Stocks	Barrels	Sales	Commission	Comment
1	**1**	1	**1**	100	**10**	Output minimum
2	**1**	1	**2**	125	**12.5**	Output minimum+
3	**1**	2	**1**	130	**13**	Output minimum+
4	**2**	1	**1**	145	**14.5**	Output minimum+
5	**5**	5	**5**	500	**50**	Midpoint
6	**10**	10	**9**	975	**97.5**	Border point-
7	**10**	9	**10**	970	**97**	Border point-
8	**9**	10	**10**	955	**95.5**	Border point-
9	**10**	10	**10**	1000	**100**	Border point
10	**10**	10	**11**	1025	**103.75**	Border point+
11	**10**	11	**10**	1030	**104.5**	Border point+
12	**11**	10	**10**	1045	**106.75**	Border point+
13	**14**	14	**14**	1400	**160**	Midpoint
14	**18**	18	**17**	1775	**216.25**	Border point-
15	**18**	17	**18**	1770	**215.5**	Border point-
16	**17**	18	**18**	1755	**213.25**	Border point-
17	**18**	18	**18**	1800	**220**	Border point

18	**18**	18	**19**	1825	**225**	Border point+
19	**18**	19	**18**	1830	**226**	Border point+
20	**19**	18	**18**	1845	**229**	Border point+
21	**48**	48	**48**	4800	**820**	Midpoint
22	**70**	80	**89**	7775	**1415**	Output maximum-
23	**70**	79	**90**	7770	**1414**	Output maximum-
24	**69**	80	**90**	7755	**1411**	Output maximum-
25	**70**	80	**90**	7800	**1420**	Output maximum

Output special Value Test Cases

Case	Locks	Stocks	Barrels	Sales	Commission	Comment
1	**10**	11	**9**	1005	**100.75**	Border Point+
2	**18**	17	**19**	1795	**219.25**	Border Point-
3	**18**	19	**17**	1805	**221**	Border Point+

➢ Random Testing

- Here, the idea is to generate random number to pick test cases rather than always choosing min, min+, nom, max-, max values. The tables below are derived from a visual basic application that picks values for a bounded variable **a ≤ x ≤ b** as follows:

  x=Int ((b-a+1) * Rnd+a)

  Int: returns integer, **Rnd:** generates random numbers in the interval [0, 1]
- The program keeps generating random test cases until at least **one of each output** occurs.

Random Test Cases for the Triangle Program

Test Cases	Nontriangles	Scalene	Isosceles	Equilateral
1289	**663**	593	**32**	1
15436	7696	7372	367	1
17091	8556	8164	367	1
2603	1284	1252	66	1
6475	3197	3122	155	1
5978	2998	2850	129	1
9008	4447	4353	207	1
Percentage	**49.83%**	**47.87%**	**2.29%**	**0.01%**

Random Test Cases for the Commission Program

Test Cases	10%	15%	20%
91	**1**	6	**84**
27	**1**	1	**25**
72	**1**	1	**70**
176	**1**	6	**169**
48	**1**	1	**46**
152	**1**	6	**145**
125	**1**	4	**120**
percentage	**1.01%**	**3.62%**	**95.37%**

Random Test Cases for the NextDate Program

Test Cases	Days 1-30 of 31-Day Months	Day 31 of 31-Day Months	Days 1-29 of 30-Day Months	Days 30 of 30-Day Months
913	542	17	274	10
1101	621	9	358	8
4201	2448	64	1242	46
1097	600	21	350	9
5853	3342	100	1804	82
3959	2195	73	1252	42
1436	786	22	456	13
Percentage	**56.76%**	**1.65%**	**30.91%**	**1.13%**
Probability	**56.45%**	**1.88%**	**31.18%**	**1.88%**

Days 1-27 of Feb.	Feb. 28 of a Leap Year	Feb. 28 of a Non-Leap Year	Feb. 29 of a Leap Year	Impossible Days
45	1	1	1	22
83	1	1	1	19
312	1	8	3	77
92	1	4	1	19
417	1	11	2	94
310	1	6	5	75
126	1	5	1	26
7.46%	**0.04%**	**0.19%**	**0.08%**	**1.79%**
7.26%	**0.07%**	**0.20%**	**0.07%**	**1.01%**

✓ **Advantages of BVA**

- Very good at exposing potential user interface/user input problems.
- Very clear guide lines on determining test cases.
- Very small set of test cases are generated.

✓ **Disadvantages of BVA**

- Does not test all possible inputs.
- Does not test dependencies between combinations of inputs.

Equivalence Class Testing

➢ Equivalence classes

- The important aspect of equivalence classes is that they form a **partition of a set,** where partition refers to collection of **mutually disjoint subsets**, the union of which is the entire set.
- This law has two important implications for testing:
  - The fact that the entire set is represented provides a form of **completeness**, and the **disjointedness** ensures a form of **non redundancy**.
- The elements of a subset have something in common. The idea of equivalence class testing is to **identify test cases** by using **one element** from **each equivalence class**.
- If the equivalences classes are chosen wisely, this greatly reduces the potential **redundancy** among test cases.
- The key point in equivalence class testing is to choose the **equivalence relation** that determines the classes.
- We need the function that we have used in boundary value testing for the sake of comprehensible drawings, which relates to a **function F** of **two variables x_1 and x_2.**
- When F is implemented as a program, the input variables **x_1 and x_2** will have the following boundaries, and intervals within the boundaries:

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

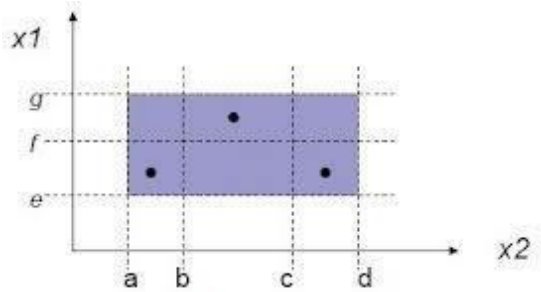
  $e \leq x_2 \leq g$, with intervals **[e, f), [f, g]**

- Where square brackets and parentheses denote respectively, closed and open interval endpoints. The intervals correspond to some distinction in the program being tested.

✓ Weak Normal Equivalence Class Testing

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

Figure: Weak normal equivalence class test cases.

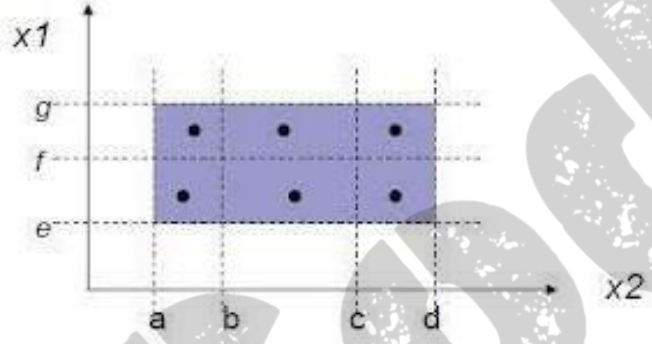


- These **three test cases** use one value from each equivalence class and identified in a systematic way. In fact, we will always have the same number of equivalence class test cases as classes in the partition with the largest number of subsets.

✓ Strong Normal Equivalence Class Testing

- Strong equivalence class testing is based on the **multiple fault** assumption, so we need test cases from each element of the **Cartesian product (3*2)** of the equivalence classes as shown in **figure**.
- **Cartesian product** guarantees that we have notion of completeness in two senses:
  i) We **cover all** the equivalence classes.
  ii) We have one of the **each** possible combination of inputs.
- The key to good equivalence class testing is the selection of the equivalence relation.

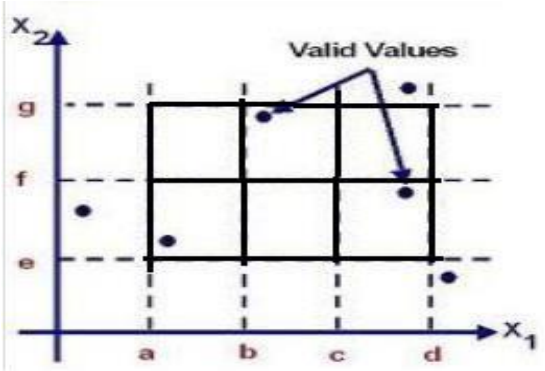
Figure: Strong normal equivalence class test cases.



✓ Weak Robust Equivalence Class Testing

- The name for this form is admittedly **counterintuitive** and **oxymoronic**.
  - How can something be both weak and robust?
- The **robust part** comes from consideration of **invalid values.**
- And **weak part** refers to the **single fault** assumption.
  - For **valid inputs**, use one value from each **valid class**.
  - For **invalid inputs**, a test case will have one **invalid value** and the remaining values will all be **valid**.
- Two problems occur with **robust Equivalence Class Testing**
  - The first is that, very often, the specification does not define what the **expected output** for an **invalid input** should be.
  - The second problem is that, **strongly typed** languages eliminate the need for the consideration of **invalid input**.

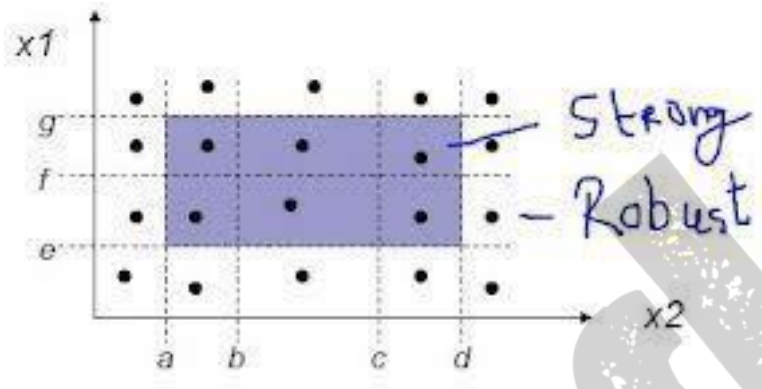
Figure: Weak robust equivalence class test cases.



✓ Strong Robust Equivalence Class Testing

- This form is neither **counterintuitive** nor **oxymoronic**, just redundant.
- The **robust part** comes from consideration of **invalid values**, and the **strong part** refers to the **multiple fault** assumption.
- We obtain test cases from **each element** of the **Cartesian product** of all the equivalence classes as shown in the **figure below.**

Figure: Strong robust equivalence class test cases.



➢ Equivalence Class Test Cases for the Triangle Problem

- According to problem statement, four possible outputs can occur: **Not a Triangle, Scalene, Isosceles** and **Equilateral.** We can use these to identify **output equivalence classes** as follows:

  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}

- Four **weak normal** equivalence class test cases, chosen arbitrarily from each class 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

- Because **no valid subintervals** of variables a, b, and c exist, the **strong normal** equivalence class test cases are **identical** to **weak normal** equivalence class test cases.
- 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
WR4	**11**	5	5	Value of a is not in the range of permitted values
WR5	5	**11**	5	Value of b is not in the range of permitted values
WR6	5	5	**11**	Value of c is not in the range of permitted values

- **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	Value of a, b is not in the range of permitted values
SR5	5	**-1**	**-1**	Value of b, c is not in the range of permitted values
SR6	**-1**	5	**-1**	Value of a, c is not in the range of permitted values
SR7	**-1**	**-1**	**-1**	Value of a, b, c is not in the range of permitted values

- We can notice that how thoroughly the **expected outputs** describe the **invalid inputs** values.
- Equivalence class testing is clearly sensitive to the equivalence relation used to define classes. If we base equivalence classes on the **input domain**, we obtain a **richer set** of test cases.
- What are some of the possibilities for the three integers, a, b and c? They can all be equal, exactly one pair can be or none can be equal:

  D1= {< a, b, c >: a = b = c}
  D2= {< a, b, c >: a = b, a ≠ c}
  D3= {< a, b, c >: a = c, a ≠ b}
  D4= {< a, b, c >: b = c, a ≠ b}
  D5= {< a, b, c >: a ≠ b, a ≠ c, b ≠ c}

- We can apply the triangle property to see if they even constitute a triangle.

  D6= {< a, b, c >: a ≥ b+c}
  D7= {< a, b, c >: b ≥ a+c}
  D8= {< a, b, c >: c ≥ a+b}

- If we want to be even more thorough, we could separate the "greater than or equal to" into two distinct cases; thus the set D6 would become:

  $D6'$ = {< a, b, c >: a=b+c}
  $D6''$ = {< a, b, c >: a>b+c} and similarly for D7 and D8.

➢ Equivalence Class Test Cases for the NextDate Function

- The NextDate function illustrates very well the craft of choosing the underlying **equivalence relation**. **NextDate** is a function of three variables **month, day** and **year** and these have intervals of **valid classes** defined as follows,

  M1 = {month: 1 ≤ month ≤ 12}
  D1 = {day: 1 ≤ day ≤ 31}
  Y1 = {year: 1812 ≤ year ≤ 2019}

- **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 > 2019}

- 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	1915	6/16/1915

- Here is the full set of **weak robust** test cases:

Case ID	Month	Day	Year	Expected Output
WR1	6	15	1915	6/16/1915
WR2	**-1**	15	1915	Value of month not in the range 1..12
WR3	**13**	15	1915	Value of month not in the range 1..12
WR4	6	**-1**	1915	Value of day not in the range 1..31
WR5	6	**32**	1915	Value of day not in the range 1..31
WR6	6	15	**1811**	Value of year not in the range 1812..2019
WR7	6	15	**2020**	Value of year not in the range 1812..2019

- **One corner of the cube in 3-space** of the additional **strong robust** equivalence class test cases:

Case ID	Month	Day	Year	Expected Output
SR1	**-1**	15	1915	Value of month not in the range 1..12
SR2	6	**-1**	1915	Value of day not in the range 1..31
SR3	6	15	**1811**	Value of year not in the range 1812..2019
SR4	**-1**	**-1**	1915	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..2019
SR6	**-1**	15	**1811**	Value of month not in the range 1..12 Value of year not in the range 1812..2019
SR7	**-1**	**-1**	**1811**	Value of month not in the range 1..12 Value of day is not in the range 1..31 Value of year not in the range 1812..2019

- If we more carefully choose the equivalence relation, the resulting equivalence classes will be more useful.
- If it is not the last day of a month what must be done to an input date?
  - The NextDate function will simply **increment the day** value.
  - At the **end of a month**, the next day is 1 and the **month is incremented**.
  - At the **end of a year**, both the **day** and **month** are **reset to 1**, and the **year is incremented**.
- Finally, the problem of leap year makes determining the last day of a month is interesting.

With all above in mind, we might postulate the following 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 **non-century leap year**}
Y3 = {year: year is a **common year**}

- By having separate classes for **30** and **31 day months**, we can **simplify** the question of the **last day** of the month.
- By taking **February** as a separate class, we can give **more attention to leap year** questions.
- We can give special attention to day values: **days in D1** are always incremented, while **days in D4** only have meaning for **months in M2**.
- Finally, we have **three classes** of **years**: the special case of the **year 2000**, **leap years**, and **non-leap years**.

Note: This is not a perfect set of equivalence classes, but its use will reveal many potential errors.

✓ Equivalence Class Test Cases

- The following are **weak equivalence class** test cases. The inputs are selected from the approximately middle value of the corresponding class.

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	Invalid Input Date
WN4	6	31	2000	Invalid Input Date

- The **strong normal equivalence class** test cases for the revised classes are,

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	7/1/2000
SN8	6	30	1996	7/1/1996
SN9	6	30	2002	7/1/2002
SN10	6	31	2000	Invalid Input Date
SN11	6	31	1996	Invalid Input Date
SN12	6	31	2002	Invalid Input Date
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
SN18	7	29	2002	7/30/2002
SN19	7	30	2000	7/31/2000
SN20	7	30	1996	7/31/1996
SN21	7	30	2002	7/31/2002
SN22	7	31	2000	8/1/2000
SN23	7	31	1996	8/1/1996
SN24	7	31	2002	8/1/2002
SN25	2	14	2000	2/15/2000
SN26	2	14	1996	2/15/1996
SN27	2	14	2002	2/15/2002
SN28	2	29	2000	3/1/2000
SN29	2	29	1996	3/1/1996
SN30	2	29	2002	Invalid Input Date
SN31	2	30	2000	Invalid Input Date
SN32	2	30	1996	Invalid Input Date
SN33	2	30	2002	Invalid Input Date
SN34	2	31	2000	Invalid Input Date
SN35	2	31	1996	Invalid Input Date
SN36	2	31	2002	Invalid Input Date

- When we move from **weak to strong normal** testing, issues of **redundancy** may arise.
- The move from **weak to strong**, whether with **normal or robust** classes, always makes the presumption of independence and this is reflected in the cross-product of the equivalence classes.
- **Three month classes** times **four day classes** times **three year classes (3*4*3=36)** results in **36 strong normal equivalence class test cases**.
- Adding two **invalid classes** for **each variable (3+2 * 4+2 * 3+2)** will result in **150 strong robust** equivalence class test cases ie., (5*6*5).

➢ Equivalence Class Test Cases for the Commission Problem

- The **input domain** of the commission problem is **partitioned** by the **limits** on **locks, stocks,** and **barrels**.
- These equivalence classes are exactly similar to be identified by traditional equivalence class testing.
- The **first class** is the **valid input** and **other two** are **invalid.**
- The **input domain** equivalence classes **lead to** very **unsatisfactory sets** of **test cases** and equivalence classes defined on the **output range** of the commission function will be an **improvement**.
- The **valid classes** of the input variables are:

  L1= {locks: **1 ≤ locks≤70**}
  L2= {locks= **-1**} (occurs if locks =-1 is used to control input iteration)
  S1= {stocks: **1 ≤ stocks ≤ 80**}
  B1= {barrels: **1 ≤ barrels ≤ 90**}

- The corresponding **invalid classes** of the input variables are:

  L3= {locks: locks=0 OR locks < -1}
  L4= {locks: locks > 70}
  S2= {stocks: stocks < 1}
  S3= {stocks: stocks > 80}
  B2= {barrels: barrels < 1}
  B3= {barrels: barrels > 90}

- When a value of **-1 is given** for **locks,** the **while loop terminates**, and the values of **totalLocks, totalStocks, and total Barrels** are used to compute **sales,** and then **commission.**
- Here, we will have exactly **one weak normal** equivalence class test case and it is identical to the **strong normal** equivalence class test case.

Case ID	Locks	Stocks	Barrels	Expected Output
WN1, SN1	35	40	45	$3900

- Note that the case for **locks = -1** just **terminates** the **iteration**. We will have eight weak **robust** test cases.

Case ID	Locks	Stocks	Barrels	Expected Output
WR1	10	10	10	$1000
WR2	**-1**	40	45	Input loop terminates
WR3	**-2**	40	45	Value of **Locks** not in the range 1..70
WR4	**71**	40	45	Value of **Locks** not in the range 1..70
WR5	35	**-1**	45	Value of **Stocks** not in the range 1..80
WR6	35	**81**	45	Value of **Stocks** not in the range 1..80
WR7	35	40	**-1**	Value of **Barrels** not in the range 1..90
WR8	35	40	**91**	Value of **Barrels** not in the range 1..90

- Finally, a corner of the cube will be in 3-space of the additional **strong robust** equivalence class test cases:

Case ID	Locks	Stocks	Barrels	Expected Output
SR1	-2	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	-2	-1	45	Value of **Locks** not in the range 1..70 Value of **Stocks** not in the range 1..80
SR5	-2	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	-2	-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

- We can get some help by considering equivalence classes defined on the **output range**. It is a function of the number of locks, stocks, and barrels sold.

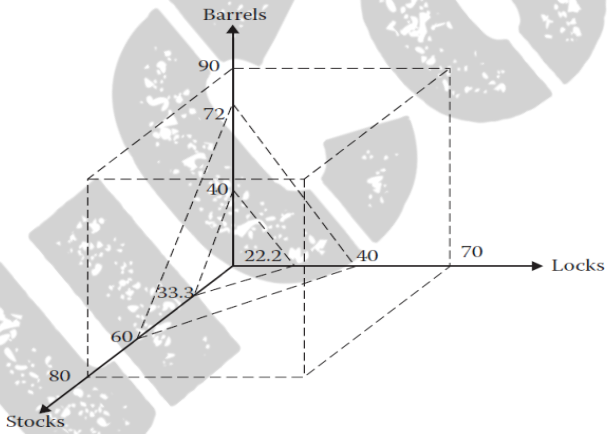
  Sales=45 * locks + 30 * stocks + 25 * barrels

- 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**}

- **Figure below** helps us get a better feel for the input space.
- Elements of **S1** are points with **integer coordinates** in the **pyramid** near the **origin**.
- Elements of **S2** are points in the **triangular slice** between the pyramid and the rest of the input space.
- Finally, elements of **S3** are all those points in the rectangular volume.
- All the error cases found by the **strong equivalence** classes of the input domain are not in S1 or S2.
- All error cases found by the **strong equivalence** classes of the input domain are **outside** of the rectangular space shown in figure below.

Figure: Input space of the commission problem



- Output test cases give us **some sense** that we are exercising **important parts** of the problem. We might want to add some boundary checking, just to make sure the transitions at **sales** of $1000 and $1800 are correct. This is not particularly easy because we can only choose values of locks, stocks, barrels.

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

✓ Advantages

- Equivalence partitions are designed so that **every possible input** belongs to **one** and **only one** equivalence partition.
- It is appropriate when input data is defined in **terms of intervals** and **sets of discrete** value.
- It is used when program function is complex.

- **Strong** and **weak** classes helps to distinguish between **progression** and **regression** testing.

✓ **Disadvantages**

- Doesn't test every input.
- No guide lines for choosing inputs.
- Several tries may be needed before the right equivalence relation is discovered.

➢ Guidelines and Observations

1. 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, it makes **no sense** to use the robust forms.
3. If **error conditions** are of 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.
6. Equivalence class testing is indicated when the program **function is complex**. In such cases, the complexity of the function can help **identify useful equivalence classes**, as in the NextDate function.
7. **Strong equivalence** class testing makes a **presumption** that the **variables are independent**, and the corresponding **multiplication of test cases** raises issues of redundancy. If any dependencies occur, they will often generate "error" test cases.
8. **Several tries** may be needed before the **"right"** equivalence relation is discovered. This is sometimes known as the **"competent programmer hypothesis".**
9. The difference between the strong and weak forms of equivalence class testing is **helpful** in the **distinction** between **progressio**n and **regression** testing.

Chapter 7

Decision Table – Based Testing

➢ Decision tables

- Functional testing based on **decision tables** are most **rigorous** because decision tables enforce **logical rigor**.
- Decision tables have been used to **represent** and **analyze complex logical relationships**. They are ideal for describing situations in which a **number of combinations of action** are taken under varying sets of conditions.
- A decision Table has **four portions**: the left- most columns is the **stub portion**; to the right is the **entry portion**. The condition portion is noticed by **c's,** action portion is noted as **a's.** They are referred as **condition stub, condition entries, action stub, and action entries.**
- When we have **binary conditions (true/false, yes/no, 0/1),** the condition portion of a decision table is **truth table** that has been rotated 90^0. This structure guarantees that we consider every possible **combination of condition values**.
- **Rules** indicate which **action**, if any, are taken for the **circumstances** (pre-conditions)

indicated in the condition portion of the rule.

- Decision tables in which all the conditions are **binary** are called **limited entry** decision tables. If conditions are allowed to have **several values**, the resulting tables are called **extended entry** decision tables.
- We use decision tables for **test case identification.** This **completeness** property of a decision table guarantees a form of complete testing.
- Decision tables are **declarative** that **conditions** and **actions** follow **no particular order**.
- The **Portion** of a **Decision table is shown below,**

Stub	Rule1	Rule2	Rule3,4	Rule5	Rule6	Rule7,8
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

✓ Technique

- To identify test cases with decision tables, we interpret **conditions as inputs** and **actions as outputs**. Sometimes conditions end up referring to equivalence classes of inputs, and actions refer to major functional processing portions of the item tested.
- The **rules** are then interpreted as **test cases**.
- The **Decision Table** for the **Triangle Problem is as shown below,**

	R1	R2	R3	R4	R5	R6	R7	R8	R9
C1:a,b,c form a triangle	F	T	T	T	T	T	T	T	T
C2: a=b?	-	T	T	T	T	F	F	F	F
C3: a=c?	-	T	T	F	F	T	T	F	F
C4: b=c?	-	T	F	T	F	T	F	T	F
A1: Not a Triangle	**X**								
A2:Scalene									**X**
A3: Isosceles					**X**		**X**	**X**	
A4: Equilateral		**X**							
A5:Impossible			**X**	**X**		**X**			

- **Refined Decision Table for the Triangle Problem (*) is as shown below,**

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
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**			

- **The above decision table illustrates another consideration related to technique:**
- The choice of conditions can greatly **expand** the size of a decision table.
  - We expanded the **old condition (c1: a, b, c form a triangle?)** to a more **detailed view** of the **three inequalities** of the triangle property.
  - If any one of these **fails**, the three integers do not constitute sides of a triangle.
- **Use of don't care entries**
  - Use of don't care entries has a slight effect on the way in which complete decision tables are recognized. For limited entry decision tables, if **n** conditions exist, there must be **2^n rul**es. When don't care entries really indicate that the condition is **irrelevant**, we can develop a **rule count** as follows:
    - Rules in which **don't care** entries **do not occur** count as **one rule** and
    - Each **don't care entry** in a rule **doubles** the count of that rule in **power of 2**.

- **Decision Table for above Table (previous page (*)) with Rule Counts**

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**
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			

- **Decision Table with Mutually Exclusive Conditions for NextDate**

Conditions	**R1**	**R2**	**R3**
C1: month in M1?	T	-	-
C2: month in M2?	-	T	-
C3: month in M3?	-	-	T
A1			
A2			
A3			

- If we apply this **simplistic algorithm** to the above decision table we get rule counts shown in **table below**. But the actual rule count is **2^3=8** but **12 rules** are generated. By expanding the

table and removing the redundant rules we can **reduce** to **8 rules**. If we have **redundant rules** and the **actions are different,** then those rules are called **inconsistent rules**. If decision table contains **inconsistent rules,** then the table is called **nondeterministic** decision table**.**

C1 : month in M1	T	-	-
C2 : month in M2	-	T	-
C3 : month in M3	-	-	T
Rule count	4	4	4
A1			

➢ Test Cases for the Triangle Problem

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

- **Using the above decision table (*), we obtain 11 functional test cases:**
  - 3 ways to fail the triangle property and
  - 1 way to get an equilateral triangle
  - 3 impossible cases
  - 3 ways to get an isosceles triangle
  - 1 way to get a scalene triangle

➢ Test case for the Next Date Function

- The Next Date function illustrates the problem of **dependencies** in the input domain.
- This makes it a perfect example for decision table based testing, because decision tables can highlight such dependencies.
- The decision table format lets us emphasize such dependencies using the notion of the **"impossible"** action to denote impossible combination of conditions which are actually impossible rules.

✓ First Try

Consider the following set of equivalence classes

M1 = {month: month has 30 days}

M2 = {month: month has 31 days}

M3 = {month: month is February}

D1 = {day: $1 \leq day \leq 28$}
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}

- To highlight **impossible combinations**, we could make a **limited entry decision table** with the following condition and actions as shown in **below table**. The **year** class is collapsed into one condition. This decision table will have **256 rules**, many of which will be impossible.

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			

- If we want to show why these rules are impossible we might **revise our actions** as follows,
  A1: Day invalid for this month
  A2: cannot happen in a non – leap year
  A3: compute the next date

✓ Second Try

- If we focus on **leap year** aspect of the NextDate function; we could use the set of equivalence classes. These classes have a Cartesian product that contains **36 entries**, with several that are impossible.
- Here, **Y2** is the set of years between **1812** and **2019,** excluding the year 2000.
  M1= {month: month has 30 days}
  M2= {month: month has 31 days}
  M3= {month: month is February}
  D1= {day: $1 \leq day \leq 28$}
  D2= {day: day=29}
  D3= {day: day=30}
  D4= {day: day=31}
  Y1= {year: year=2000}
  Y2= {year: year is a Non-century leap year}
  Y3= {year: year is a common year}
- To produce the next date of a given date, only **five** possible manipulations can be used; **incrementing** and **resetting the day** and **month** and **incrementing the year**.
- These conditions would result in a decision table with **36 rules (3*4*3)** that correspond to

the Cartesian product of the equivalence classes.

- Combining rules with **don't care** entries yield the decision table as shown below.
- We still have the **problem** with logically **impossible rules**, but this formulation helps us identify the expected outputs of a test case.

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
C1 : month in	M3	M3	M3	M3	M3	M3	M3	M3
C2 : day in	D1	D1	D1	D2	D2	D2	D3	D4
C3 : year in	Y1	Y2	Y3	Y1	Y2	Y3	-	-
Rule count	1	1	1	1	1	1	3	3
Actions								
A1 : Impossible						X	X	X
A2 : Increment day	X	X						
A3 : reset day			X	X	X			
A4 : Increment month			X	X	X			
A5 : reset month								
A6 : Increment year								

✓ Third try

- We can clear up the **end-of-year** consideration with a third set of equivalent classes. This time, we are very specific about days and month, and we revert to the simpler leap year or non-leap year condition of the first try, so the year 2000 gets no special attention.

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: $1 \leq day \leq 27$}
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}

- The Cartesian product of these contains 40 (4*5*2) elements. The result of combining rules with don't care entries is given and it has **22 rules**.
- We have a **22-rule** decision table that gives a clear picture of the NextDate function than the second try decision table.
- The first **five** rules **1 to 5** deal with **30-day month**. The next two sets of **rules 6 to 15** deal with 31-day months where rules **6 to 10** are **31 days months** and rules **11 to 15** are of **December** month.
- Rules **16 to 22** deals with **February** month.

Decision Table for the Next Date 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	-	-	-	-	-	-	-	-	-	-
Rule count										
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	-	-
Rule count												
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							

Reduced Decision Table for the Next Date 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	-	-	-	-	-
Rule count					
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	-
Rule count								
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						

Decision Table Test Cases for the Next Date

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	Invalid Input Date
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	Invalid Input Date
21,22	February	30	2001	Invalid Input Date

➢ Test case for commission problem

- The commission problem is **not well** served by a decision table analysis. This is not surprising because very little decisional logic is used in the problem.
- Because the variables in the equivalence classes are truly independent, no impossible rules will occur in a decision table in which conditions correspond to the equivalence classes.

Note: But could be done in the following manner (own solution)

The valid classes of the input variables are:

L1= {locks=**-1**} (occurs if locks =-1 is used to control input iteration)

L2= {locks: **1 ≤ locks ≤ 70**}

S1= {stocks: **1 ≤ stocks ≤ 80**}

B1= {barrels: **1 ≤ barrels ≤ 90**}

Input data decision Table										
RULES		**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**
Conditions	**C1: Locks = -1**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
	C2 : 1 ≤ Locks ≤ 70	**-**	**T**	**T**	**F**	**T**	**F**	**F**	**F**	**T**
	C3 : 1 ≤ Stocks ≤ 80	**-**	**T**	**F**	**T**	**F**	**T**	**F**	**F**	**T**
	C4 : 1 ≤ Barrels ≤ 90	**-**	**F**	**T**	**T**	**F**	**F**	**T**	**F**	**T**
Actions	**A1 : Terminate the input loop**	**X**								
	A2 : Invalid locks input				**X**		**X**	**X**	**X**	
	A3 : Invalid stocks input			**X**		**X**		**X**	**X**	
	A4 : Invalid barrels input		**X**			**X**	**X**		**X**	
	A5 : Calculate total locks, stocks and barrels		**X**	**X**	**X**	**X**	**X**	**X**		**X**
	A6 : Calculate Sales	**X**								
	A7: proceed to commission decision table	**X**								
Rule Count		**8**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Commission calculation Decision Table					
Precondition : lock = -1					
RULES		**R1**	**R2**	**R3**	**R4**
Condition	**C1 : Sales = 0**	**T**	**F**	**F**	**F**
	C2 : Sales > 0 AND Sales ≤ 1000		**T**	**F**	**F**
	C3 : Sales ≥ 1001 AND sales ≤ 1800			**T**	**F**
	C4 : sales ≥1801				**T**
Actions	**A1 : Terminate the program**	**X**			
	A2 : comm= 10%*sales		**X**		
	A3 : comm = 10%*1000 + (sales-1000)*15%			**X**	
	A4 : comm = 10%*1000 + 15% * 800 + (sales-1800)*20%				**X**

Commission Problem -Decision Table Test cases for input data

Precondition: Initial Value: Total Locks= 0, Total Stocks=0 and Total Barrels=0

Case Id	Description	Input Data			Expected Output
		Locks	Stocks	Barrels	
1	Enter the value of Locks= -1	-1			Terminate the input loop. Check for sales, if(sales=0) exit from program, else calculate commission.
2	Enter the valid input for locks and stocks and invalid for barrels	20	30	-5	Total of locks, stocks is updated if it is within a precondition limit and Should display value of barrels is not in the range 1..90.
3	Enter the valid input for locks and barrels and invalid for stocks	15	-2	45	Total of locks, barrels is updated if it is within a precondition limit and Should display value of stocks is not in the range 1..80.
4	Enter the valid input for stocks and barrels and invalid for locks	-4	15	16	Total of stocks , barrels is updated if it is within a precondition limit and Should display value of locks is not in the range 1..70
5	Enter the valid input for locks and invalid value for stocks and barrels	15	81	100	Total of locks is updated if it is within a precondition limit and **(i)** Should display value of stocks is not in the range 1..80. **(ii)** Should display value of barrels is not in the range 1..90.
6	Enter the valid input for stocks and invalid value for locks and barrels	88	20	99	Total of stocks is updated if it is within a precondition limit and **(i)** Should display value of locks is not in the range 1..70. **(ii)** Should display value of barrels is not in the range 1..90.
7	Enter the valid input for barrels and invalid value for locks and stocks	100	200	25	Total of barrels is updated if it is within a precondition limit and **(i)** Should display value of locks is not in the range 1..70 **(ii)** Should display value of stocks is not in the range 1..80
8	Enter the invalid input for locks, stocks and barrels	-5	400	-9	**(i)** Should display value of locks is not in the range 1..70. **(ii)** Should display value of stocks is not in the range 1..80 **(iii)** Should display value of barrels in not in the range 1..90.
9	Enter the valid input for locks, stocks and barrels	15	20	25	Total of locks, stocks and barrels is updated if it is within a precondition limit and calculates the sales and proceeds to commission.

Commission Problem -Decision Table Test cases for commission calculation
Precondition : Locks = -1

Description		Input Data	Expected Output	
		Sales	Commission	Value
1	Check the value of sales	0	Terminate the program where commission is zero	0
2	if sales value with in these range(Sales > 0 AND Sales ≤ 1000)	900	Then commission = 0.10*sales	90
3	if sales value with in these range(Sales > 1000 AND Sales ≤ 1800)	1400	Then commission = 0.10*1000 + 0.15*(sales - 1000)	160
4	if sales value with in these range(Sales > 1800	2500	Then commission = 0.10*1000 + 0.15*800 + 0.20 *(sales - 1800)	360

✓ **Applications of where Decision Tables are used**

- Where prominent if-then-else-logic.
- Logical relationships among input variables.
- Calculations involving subsets of the input variables.
- Cause-and-effect relationships between inputs and outputs.
- High cyclomatic complexity

✓ **Advantages of Decision Tables**

- Decision tables are precise and compact way to model complicated logic.
- Testing also works iteratively. The table that is drawn in the first iteration acts as a stepping stone to derive new decision table(s).
- These tables guarantee that we consider every possible combination of condition values. This is known as its **"completeness property".** This property promises a form of complete testing as compared to other techniques.
- Decision tables are declarative. There is no particular order for conditions and actions to occur.

✓ **Disadvantages of Decision Tables**

- Decision tables do not scale up well. We need to **"factor"** large tables into smaller ones to remove redundancy.

Module 2

1. What do you mean by Boundary value Analysis (BVA)? Explain the same with diagrams.
2. Explain how we introduce Robustness to BVA with diagram.
3. Discuss about Worst-case Testing in case Boundary Value Testing. How Robustness can be introduced here with diagram.
4. Give conditions and generate BVA test cases and discuss test cases for triangle problem.
5. Give conditions and generate BVA worst case test case and discuss on the same for triangle problem.
6. Generate robust test cases for triangle problem with BVA.
7. Give BVA test cases for Next Date problem with conditions.
8. Give worst case test cases for the Next Date problem along with conditions.
9. Give set of test cases for commission problem based on output boundary value analysis with conditions.
10. What are the Advantages and Disadvantages of BVA.
11. Explain equivalence class testing with relevant diagrams.
12. What do you mean by Weak normal equivalence class testing, explain with diagram.
13. What do you mean by strong normal equivalence class testing, explain with diagram.
14. What do you mean by weak robust equivalence class testing, explain with diagram.
15. What do you mean by strong robust equivalence class testing, explain with diagram.
16. Give test cases for the triangle problem with conditions on output/input for WNs, WRs, SRs with equivalence class testing.
17. Give WN, WR, SR test cases for Next Date problem.
18. What do you mean by Fault-Based testing? What are the assumptions made in Fault-Based testing? Explain.
19. What is Mutation Analysis? Explain Mutation Analysis with an appropriate example?
20. Explain Fault-Based Adequacy criteria.
21. What are different types of Mutation Analysis? Explain them.

Module 2 (VTU Question papers)

1. What are the limitations of boundary value analyses?
2. Differentiate between weak strong robust equivalence class testing and strong robust equivalence class testing with an example.
3. Explain about decision table. Construct decision table of the triangle problem. It accepts three integers a, b and c as 3 sides input and outputs 3 types of triangle: equilateral, Scalene, Isosceles or not a triangle and satisfy the following conditions $a < b+c$, $b < a+c$ and $c < a+b$.
4. Explain: i) boundary value testing ii) equivalence class testing iii) decision table based testing.
5. Explain in detail worst-case testing with an example.
6. Justify the usage of boundary value analysis with function of two variables and highlight the limitations of boundary value analysis.
7. Explain weak normal and strong robust equivalence class testing with next date problem as an example.
8. Discuss the usage of decision table method to device test cases with example of commission problem and triangle problem.

9. Develop a formula for the number of robust worst case test cases for a function of two variables.
10. Explain basic decision table terms.
11. Briefly explain weak normal and strong robust equivalence class testing with an example.
12. Write a short note on random testing.
13. Explain overview of assumptions in fault-based testing.
14. Explain in detail, mutation analysis and variations on mutation testing.
15. Define below terms with respect to fault based-testing:
    (a) Original program (c) program location.
    (b) Alternate expression (d) alternate program.
16. Explain mutation analysis software fault based testing.
17. List and explain the fault-based adequacy criteria.
18. Explain hardware fault-based testing.
19. Give BVA test cases for triangle problem.
20. Briefly explain the variants of equivalence class testing. Derive equivalence class test cases for NextDate problem.