Functional Testing (Black Box Testing)

- ✓ In black box testing, program is treated as a **black box**.
- ✓ Implementation details do not matter.
- ✓ Takes a user point of view.
- ✓ Functional testing verifies that each **function** of the software application operates in conformance with the requirement specification.
- ✓ Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.

What do you test in Functional Testing?

The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates on -

- ✓ **Mainline functions**: Testing the main functions of an application
- ✓ **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- ✓ **Accessibility**: Checks the accessibility of the system for the user
- ✓ **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

Types of Functional Testing:

- ✓ Boundary Value Testing
- ✓ Equivalence Class Testing
- ✓ Decision Table based testing

Important Terms:

Single Fault Assumption: Specifies that exactly one component is malfunctioning and leads to the cause of the problem. We assume that fault in the program occurs due to value of just one variable.

Multiple Fault Assumption: More than one component leads to cause of the problem. We assume that fault in the program occurs due to value of more than one variable.

Boundary Value analysis

BVA is a testing technique that is useful for detecting faults that can happen at the boundaries of the input domain. Boundary value testing focuses on the boundary of the input space to identify the test cases. The *rationale* behind BV testing is that errors tend to occur near the extreme values of an input variable. Loop conditions for example, may test <= when they should test for <, and counters are often "off by 1".

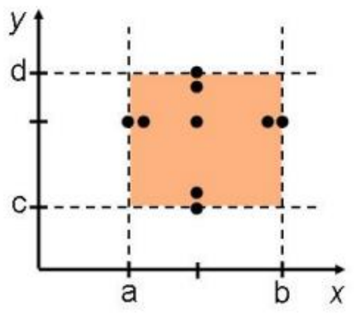
The basic idea of BV analysis is to use input variables values at their minimum, just above minimum, a nominal value, just below their maximum, and their maximum. BVA is based on a critical assumption known as the single fault assumption. It says that failures are only rarely the result of the simultaneous occurrence of two or more faults. Thus, the BVA test cases are obtained by holding the values of all but one variable at nominal values, and letting that variable assume its extreme values.

With boundary value analysis for each variable, we select 5 values:

Min	The minimum value
Min+	Slightly above minimum
Nom	Nominal
Max-	Slightly below the minimum
Max	Maximum

Ex: For a variable in the range 1-1000 the test data using BVA are 1, 2, 500, 999, 1000.

Generalizing boundary Value Analysis for 2 variables:



In case of 2 variables:

Boundary for x1: a, a+1, x1nom, b-1, b

Boundary for x2: c, c+1, x2nom, d-1, d

Therefore, the test data using BVA are:

<x1nom, c>	<a, x2nom>
<x1nom, c+1>	<a+1, x2nom>
<x1nom, x2nom>	<x1nom, x2nom>
<x1nom, d-1>	<b-1, x2nom>
<x1nom, d>	<b, x2nom>

Total unique test cases for function with 2 variables = 9.

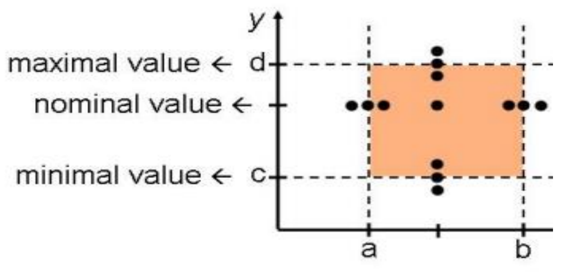
In general for ‘n’ input variables, we get **4n + 1** test cases.

Extensions of BVA

- ✓ Robustness testing
- ✓ Worst Case Testing
- ✓ Robust Worst Case Testing
- ✓ Special Value Testing
- ✓ Random Value Testing

Robustness Testing:

Robustness is defined as the degree to which a system operates correctly in the presence of invalid inputs. With robustness testing, we observe what happens when the boundary extremes are exceeded with a value slightly greater than maximum and value slightly less than minimum. Robustness testing mainly focuses on exception handling i.e. how well a system responds to invalid inputs.

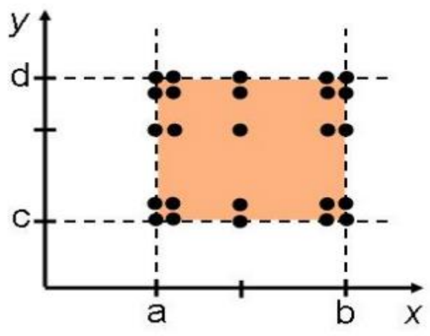


Ex: The robustness test cases for a function of 1 variable whose value ranges from 1 to 100 are: 0, 1, 2, 50, 99, 100, 101.

In general, for n input variables there are **6n + 1** test cases.

Worst Case Testing:

This testing tends to find what happens when more than one variable has extreme value simultaneously. That is, it rejects single fault assumption and tests all combinations of values. But, due to this we get large number of test cases. A worst case testing on a function of 'n' variables produces $\mathbf{5^n}$ test cases. Worst case test cases for a function of 2 variables are shown below:

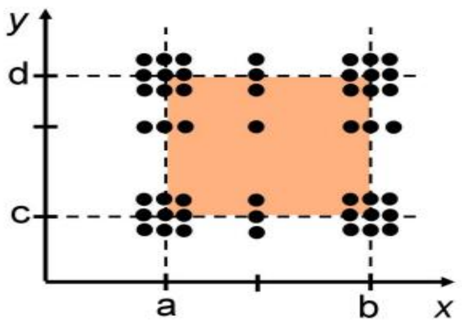


Ex: The number of test cases generated for a program with 3 variables is 5^3 =125.

Worst case testing is applicable where the physical variables have numerous interactions and where the failure of a function is extremely costly.

Robust Worst Case Testing

It's a combination of worst case testing with robust testing. We add the value min- and max+ to all the values of worst case testing. We give all combinations of 7 values for all variables. Thus, the total number of unique test cases with robust worst case testing is equals to $\mathbf{7^n}$.



Special Value Testing:

- ✓ In special value testing, the tester uses the domain knowledge and experience to test the software.
- ✓ It is an Ad-hoc testing and not based on any principle.
- ✓ It is the least systematic of all methods.
- ✓ It mainly depends on the ability of the testers.

For ex: In the nextDate function [a function where we input a date and the result is next date], a tester may use his knowledge to test for Feb 29 and leap year.

Random Testing:

The basic idea of random testing is that rather than always choose the min, min+, nom, max- and max values of a bounded variable; use a random number generator to pick the test case value. But, deciding how many random test cases are sufficient is a problem.

Limitations of BVA:

- ✓ Bounds may not be always obvious. (Deciding upper bound for some variable may be difficult)
- ✓ Bounds may not be appropriate (Ex: Boolean values)
- ✓ No bounds could be applied in case of logic variables.

Case Studies referred for Functional Testing:

Triangle Problem

Simple version: The triangle program accepts three integers, a, b, and c, as input. These are taken to be sides of a triangle. The output of the program is the type of triangle determined by the three sides: Equilateral, Isosceles, Scalene, or Not A Triangle.

Improved version: “Simple version” plus better definition of inputs:

The integers a, b, and c must satisfy the following conditions:

c1. $1 \leq a \leq 200$ c4. $a < b + c$

c2. $1 \leq b \leq 200$ c5. $b < a + c$

c3. $1 \leq c \leq 200$ c6. $c < a + b$

Final Version: "Improved version" plus better definition of outputs:

If an input value fails any of conditions c1, c2, or c3, the program notes this with an output message, for example, "Value of b is not in the range of permitted values. "If values of a, b, and c satisfy conditions c1, c2, and c3, one of four mutually exclusive outputs is given:

- ✓ If all three sides are equal, the program output is Equilateral.
- ✓ If exactly one pair of sides is equal, the program output is Isosceles.
- ✓ If no pair of sides is equal, the program output is Scalene.
- ✓ If any of conditions c4, c5, and c6 is not met, the program output is NotATriangle.

NextDate

NextDate is a function of three variables: month, date, and year. It returns the date of the day after the input date. The month, date, and year variables have integer values subject to these conditions:

c1. $1 \leq \text{month} \leq 12$

c2. $1 \leq \text{day} \leq 31$

c3. $1812 \leq \text{year} \leq 2012$

If any of conditions c1, c2, or c3 fails, NextDate produces an output indicating the corresponding variable has an out-of-range value —for example, "Value of month not in the range 1..12". Because numerous invalid day–month–year combinations exist, NextDate collapses these into one message: "Invalid Input Date."

Question 1: Considering the problem statement given for triangle problem, list the BVA test cases for the same.

Note: With BVA we follow single fault assumption principle. i.e. we use nominal values for any 2 variables (among a, b and c) and extremes for the third variable in each test case. Assume the nominal value to be 100.

Values for	min	Min+	nom	Max-	max
a	1	2	100	199	200
b	1	2	100	199	200
c	1	2	100	199	200

Total number of unique test cases with BVA are 4n+1

4x3+1=13

Case id	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

Test case with id 3, 8, 13 are same. Hence we can eliminate two and keep the first one. Total number of unique test cases are 13.

Question 2: List the BVA test cases for nextDate function.

The values of variables using BVA are as follows:

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

BVA of NextDate:

Case id	Month	day	year	Expected Output
1	6	15	1812	16-6-1812

2	6	15	1813	16-6-1813
3	6	15	1912	16-6-1912
4	6	15	2011	16-6-2011
5	6	15	2012	16-6-2012
6	6	1	1912	2-6-1912
7	6	2	1912	3-6-1912
8	6	15	1912	16-6-1912
9	6	30	1912	1-7-1912
10	6	31	1912	Error
11	1	15	1912	16-1-1912
12	2	15	1912	16-2-1912
13	6	15	1912	16-6-1912
14	11	15	1912	16-11-1912
15	12	15	1912	16-12-1912

Exercise:

- ✓ List Worst case Analysis test cases on Triangle and NextDate function.

Decision Table-Based Testing

Decision table approach for testing is ideal for describing situations where we have to deal with a combination of input. It is particularly useful in situations where different combinations of inputs results in different actions being taken. The features of decision table based testing are:

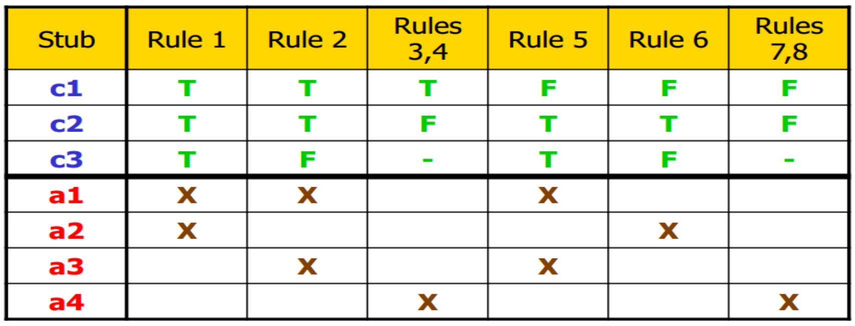
- ✓ It provides a compact way to model complicated logic
- ✓ Associate conditions with actions to perform
- ✓ Can associate many independent conditions with several actions in an elegant way

A decision table has four parts:

Conditions	Condition entries
Actions	Actions Entries

- ✓ Each condition corresponds to a decision or a predicate in the program whose possible values are listed among the condition entries portion of the table.
- ✓ Each action is a procedure or operation to perform, and the entries specify whether the action is to be performed for the set of condition alternatives the entry corresponds to.

Consider the following decision table- c1, c2, c3 are the decision variables and a1, a2, a3 and a4 are the different possible outputs of the given program.



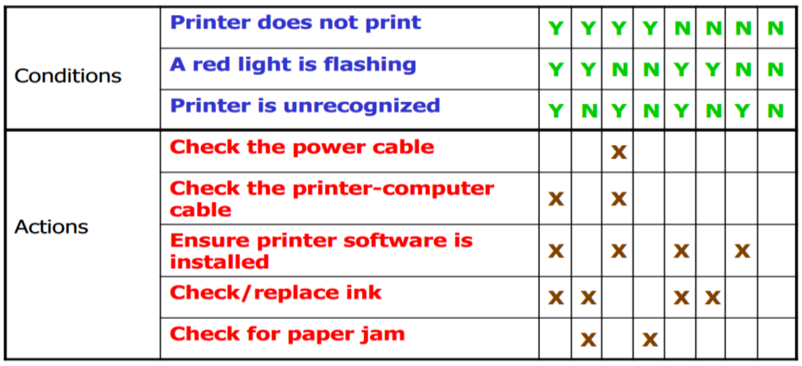
Stub	Rule 1	Rule 2	Rules 3,4	Rule 5	Rule 6	Rules 7,8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	-	T	F	-
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

In the above table, when c1, c2 and c3 are set to T it says that actions- a1 and a2 needs to be performed. The entry for c3 where c1 is true and c2 is false is called the 'don't care' entry i.e. when the value of c1 is true and c2 is false, the action being taken is always a4 regardless of what the value of c3 is; in other words for this combination of c1 and c2, the condition c3 is irrelevant or does not apply.

There are two types of decision table:

- ✓ **Limited entry table**: Condition entries restricted to binary
- ✓ **Extended entry table:** Condition entries have more than two.

Example for limited entry table: Printer Troubleshooting Decision Table



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	Check 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				

The steps in forming decision table are as follows:

1. Identify the decision variables.
2. Identify the possible values for each decision variable
3. Form a table, list all variables and actions and enumerate the allowed combinations of each of the variables.
4. Identify the cases when values assumed by a variable are immaterial for a given combination of other input variables. Represent such variables by don't care symbol.
5. For each combination of decision variables, list out the expected result or action.

Rule Count:

Rule Count is used to check if the decision table is redundant or not. Limited entry tables with N conditions have 2^N rules. Don't care entries reduce the number of explicit rules by implying the existence of non-explicitly stated rules. Each dont care entry in a rule doubles the count for the rule. For each rule determine the corresponding rule count. Total the rule counts.

Less rules than combination rule count

- ✓ Indicates missing rules

More rules than combination rule count

- ✓ Could indicate redundant rules.
- ✓ Could indicate inconsistent table

Question 1: Generate the decision table for triangle problem.

STEP 1: identify the decision variables

C1: a < b+c

C2: b < a+c

C3: c< a+b

C4: a=b?

C5: a=c?

C6: b=c?

Step 2: Identify the possible values for each decision variable

All the decision variables are conditional statements hence can take values T or F indicating true or false respectively.

Step 3: Form a table, list all variables and actions and enumerate the allowed combinations of each of the variables.

We have derived 6 decision variables and the variables can have 2 values each. Therefore the total number of rules that can be generated are 2^6=64 rules. As it is difficult to show all 64 rules we have generalized the table as shown below. Here 32 F indicates F is repeated 32 times.

	rules																															
C1: a<b+c?	32F																32T															
C2: b<a+c?	16F								16T								16F								16T							
C3: c<a+b?	8F				8T				8F				8T				8F				8T				8F				8T			
C4: a=b?	4T		4F		4T		4F		4T		4F		4T		4F		4T		4F		4T		4F		4T		4F		4T		4F	
C5: a=c?																																
C6: b=c?	…………….																															
A1:Equilateral																																
A2:Isosceles																																
A3: Scalene																																
A4: Not a Triangle																																

Step 4: Identify the cases when values assumed by a variable are immaterial for a given combination of other input variables. Represent such variables by don't care symbol. Whenever condition c1 or c2 or c3 is false the output is 'Not a triangle' irrespective of other values of other conditions. Therefore the first 32 columns in the table where c1 is false can be merged into single column with output marked 'not a triangle'. Similarly, the next 16 columns where c2 is false can be merged into single column. Later the next 8 columns where c3 is false can be merged into single column resulting in a reduced table as shown below.

	1-32	33-48	49-56	57	58	59	60	61	62	63	64
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	F	T	F	F	F
C5: a=c?	-	-	-	T	T	F	T	F	T	F	F
C6: b=c?	-	-	-	T	F	T	T	F	F	T	F
A1:Equilateral				X							
A2:Isosceles								X	X	X	
A3: Scalene											X
A4: Not a Triangle	X	X	X								
A5: impossible					X	X	X				
Rule Count											

An additional action is added to the table indicating combinations or rules that are impossible in practice.

Ex: For rule 58 it is impossible to come up with test data a, b, c such that a=b a=c but b!=c.

Based on the decision table constructed, we get the following test cases for triangle problem

Case ID	a	b	c	Expected Output
1	4	1	2	Not a Triangle
2	1	4	2	Not a Triangle
3	1	2	4	Not a Triangle
4	5	5	5	Equilateral
5	???	???	???	Impossible
6	???	???	???	Impossible
7	2	2	3	Isosceles
8	???	???	???	Impossible
9	2	3	2	Isosceles
10	3	2	2	Isosceles
11	3	4	5	Scalene

Question 2: Next Date Function:

Step 1: Identify the decision variables: 3 variables – month, day, year

C1: month in?
C2: day in?
C3: year in?

Possible actions are:
A1: Impossible date
A2: Increment day
A3: Reset day
A4: Increment Month
A5: Reset month
A6: Increment year

Step 2: Identify the possible values for each variables

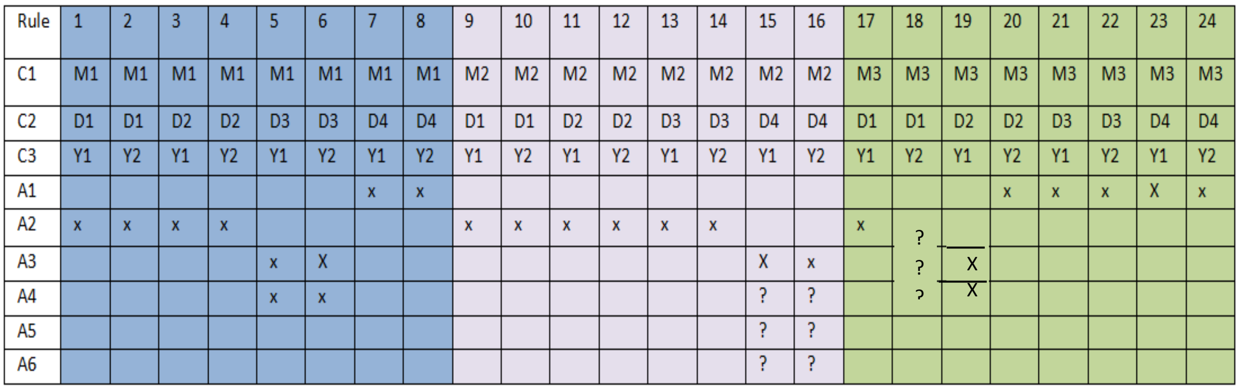
M1: {month has 30 days}
M2: {month has 31 days}
M3: {month is feb}

D1={ 1<=d<=28}
D2={d=29}
D3={d=30}
D4={d=31}

Y1={y is leap year}
Y2={y is common year}

Step 3: Form a table, list various decision variables, actions and enumerate all possible combinations of each variable.

We have derived 3 variables. Month can have 3 different values, day can have 4 different values and year can have two different values. Therefore total numbers of rules that can be generated are 3*4*2=24 rules.



Rule	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
C1	M1	M1	M1	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2	M2	M2	M2	M3	M3	M3	M3	M3	M3	M3	M3
C2	D1	D1	D2	D2	D3	D3	D4	D4	D1	D1	D2	D2	D3	D3	D4	D4	D1	D1	D2	D2	D3	D3	D4	D4
C3	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2	Y1	Y2
A1							x	x												x	x	x	X	x
A2	x	x	x	x					x	x	x	x	x	x			x	?						
A3					x	X									X	x		?	X					
A4					x	x									?	?		?	X					
A5															?	?								
A6															?	?								

Step 4: Identify the cases when the values assumed by a variable (or set of variables) are immaterial for a given combination of other input variables. Represent such variables using don’t care symbol.

C1:month in	M1	M1	M1	M1	M2	M2	M2	M2	M3	M3	M3	M3	M3	M3
C2: day in	D1	D2	D3	D4	D1	D2	D3	D4	D1	D1	D2	D2	D3	D4
C3: year in	-	-	-	-	-	-	-	-	Y1	Y2	Y1	Y2	-	-
A1:Impossible				x								x	x	x
A2:Inc day	x	x			x	x	x		X	?				
A3:Reset day			x					X		?	X			
A4: Increment month			x					?		?	x			
A5: Reset Month								?						
A6:Increment year								?						

The decision table produced above doesn’t help us in the case where month in M2 and day in D4. In such cases, we are not sure whether to increment month or to reset month or reset year. To clarify the decisions for the month of December and 28th Feb of leap year and non-leap year, we have the following changes.

M1: {month has 30 days}
M2: {month has 31 days}
M3: {month is February}
M4: {month is December}

D1={ 1<=d<=27}
D2={d=28}
D3={d=29}
D4={d=30}
D5={d=31}

Y1={y is leap year}
Y2={y is common year}

C1:month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2	M3	M3	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4
C2: day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5	D1	D2	D3	D4	D5
C3: year in	-	-	-	-	-	-	-	-	-	-	-	Y1	Y2	Y1	Y2	-	-	-	-	-	-	-
A1:Impossible					x										x	x	x					
A2:Inc day	x	x	X			x	x	x	X		x	X						x	x	x	x	
A3:Reset day				X						X			X	X								X
A4: Increment month				x						X			x	X								
A5: Reset Month																						X
A6:Increment year																						x

The following combinations of date and month have same action to be performed

M1 with D1, D2 and D3 results in increment day.
M2 with D1,D2,D3 and D4 results in increment day.
M3 with D4, D5 results in impossible date,
M4 with D1, D2, D3 and D4 results in increment day.

Hence, these dates could be combined into one rule so as to reduce redundancy in the table. The resulting table is as shown below:

C1:month in	M1	M1	M1	M2	M2	M3	M3	M3	M3	M3	M3	M4	M4
C2: day in	D1-D3	D4	D5	D1-D4	D5	D1	D2	D2	D3	D3	D4-D5	D1-D4	D5
C3: year in	-	-	-	-	-	-	Y1	Y2	Y1	Y2	-	-	-
A1:Impossible			X							X	X		
A2:Inc day	X			X		X	X					X	
A3:Reset day		X			X			X	X				X
A4: Increment month		X			X			X	X				
A5: Reset Month													X
A6:Increment year													X

Guidelines and Observations:

As with other testing techniques, decision table based testing works well for some applications such as NextDate problem. It is more suitable when the input variables are logically dependent on each other.

- The decision table technique is indicated for applications characterized by any of the following:
  - ✓ Prominent if-then logic.
  - ✓ Logical relationships among input variables
  - ✓ Calculations involving subsets of input variables.
  - ✓ Cause and effect relationships between input and output.
  - ✓ High cyclomatic complexity.
- Decision table do not scale up very well (a limited entry table with n condtions has 2^n rules).
- As with other techniques, iteration helps. The first set of conditions and actions you identify may be unsatisfactory. Use it as a stepping stone, and gradually improve it on until you are satisfied with a decision table.

Guidelines for selection of functional testing technique:

- ✓ If the variable refers to physical quantities, domain testing and equivalence class testing are indicated.
- ✓ If the variables are independent, domain testing and equivalence class testing are indicated.
- ✓ If the variables are logically dependent then decision table based testing is indicated.
- ✓ If the single fault assumption is warranted, boundary value analysis and robustness testing are indicated.
- ✓ If the multiple fault assumption is warranted, worst case testing, robust worst case testing and decision table based testing are indicated.
- ✓ If the program contains exception handling, robustness testing and decision table testing is indicated.
- ✓ If the variables refer to logical quantities, equivalence class and decision table based testing is applicable.