SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Part II-Software Testing

Functional Testing

Lecture # 19, 20, 21

11, 12, 13 Oct

TODAY'S OUTLINE

- Mid-I paper review
- Pair-wise Testing
- State Transition Testing
- Domain Analysis Testing
- Use- case Testing

PAIR-WISE TESTING

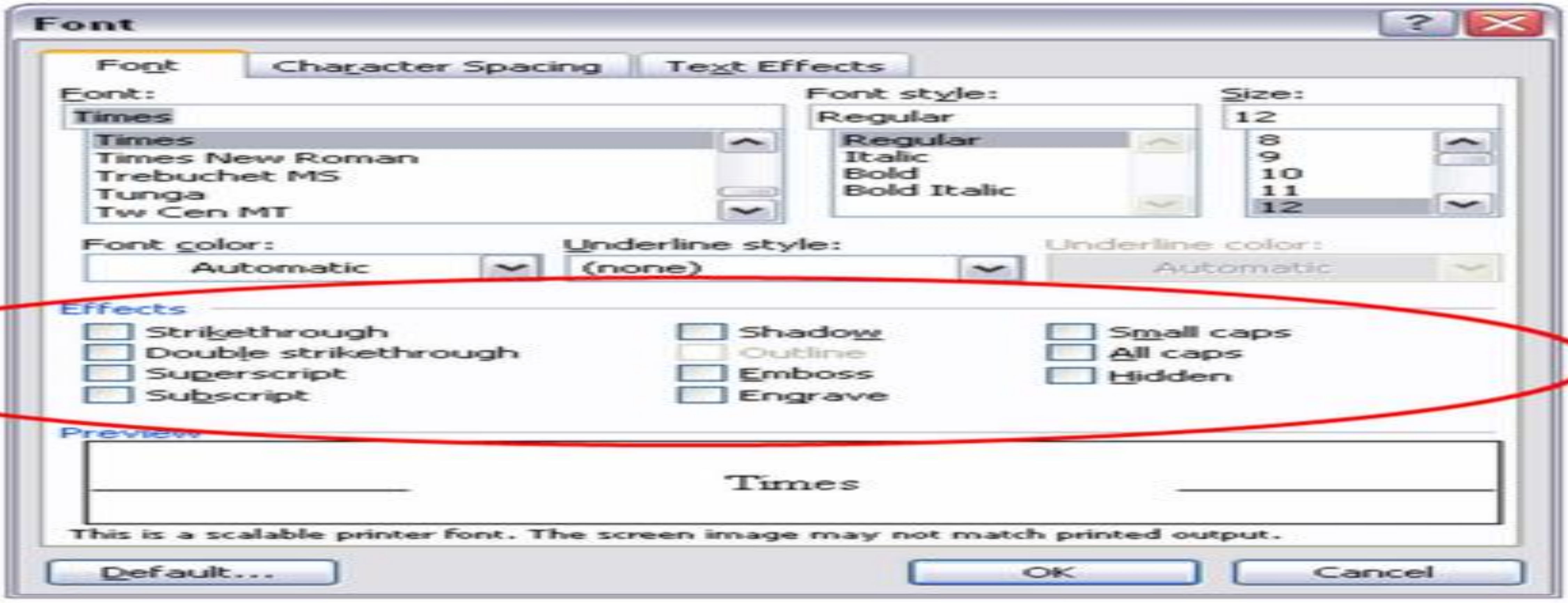
- **Pairwise Testing** is a test design technique that tries to delivers hundred percent test coverage.
- *A black-box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters.*
- Techniques like boundary value analysis and equivalence partitioning can be useful to identify the possible values for individual factors. But it is impractical to test all possible combinations of values for all those factors. So instead **a subset of combinations is generated** to satisfy all factors.
- All-Pairs technique is very helpful for designing tests for applications involving multiple parameters. Tests are designed such that for each pair of input parameters to a system, there are all possible discrete combinations of those parameters. The test suite covers all combinations; therefore it is not exhaustive yet very effective in finding bugs

EXAMPLE- COMBINATIONAL TESTING

- An application with simple list box with 10 elements (Let's say 0,1,2,3,4,5,6,7,8,9) along with a checkbox, radio button, Text Box and OK Button. The Constraint for the Text box is it can accept values only between 1 and 100. Below are the values that each one of the GUI objects can take :
- List Box - 0,1,2,3,4,5,6,7,8,9
- Check Box - Checked or Unchecked
- Radio Button - ON or OFF
- Text Box - Any Value between 1 and 100
- Exhaustive combination of the product B is calculated
  - List Box = 10
  - Check Box = 2
  - Radio Button = 2
  - Text Box = 100
- Total Number of Test Cases using Cartesian Method : 10*2*2*100 = 4000
- Total Number of Test Cases including Negative Cases will be > 4000

SOME EXAMPLES

- Suppose we have a system with on-off switches:



Font

Font | Character Spacing | Text Effects

Font: Times
- Times
- Times New Roman
- Trebuchet MS
- Tunga
- Tw Cen MT

Font style: Regular
- Regular
- Italic
- Bold
- Bold Italic

Size: 12
- 8
- 9
- 10
- 11
- 12

Font color: Automatic

Underline style: (none)

Underline color: Automatic

Effects
- ☐ Strikethrough
- ☐ Double strikethrough
- ☐ Superscript
- ☐ Subscript
- ☐ Shadow
- ☐ Outline
- ☐ Emboss
- ☐ Engrave
- ☐ Small caps
- ☐ All caps
- ☐ Hidden

Preview

Times

This is a scalable printer font. The screen image may not match printed output.

Default... | OK | Cancel

SOME MORE EXAMPLES

- **Car Ordering Application:**
- The car ordering application allows for Buying and Selling cars. It should support trading in Delhi and Mumbai.
- The application should have 5000 registration numbers that may be valid or invalid. It should allow the trade of following cars: BMW, Audi, and Mercedes.
- Two types of booking can be done: E-booking and In Store.
- Orders can be placed only during trading hours.

STEP #1: LET'S LIST DOWN THE VARIABLES INVOLVED.

- **1)** Order category
  a. Buy
  b. Sell
- **2)** Location
  a. Delhi
  b. Mumbai
- **3)** Car brand
  a. BMW
  b. Audi
  c. Mercedes
- **4)** Registration numbers
  a. Valid (5000)
  b. Invalid
- **5)** Order type
  a. E-Booking
  b. In-store
- **6)** Order time
  a. Working hours
  b. Non-working hours

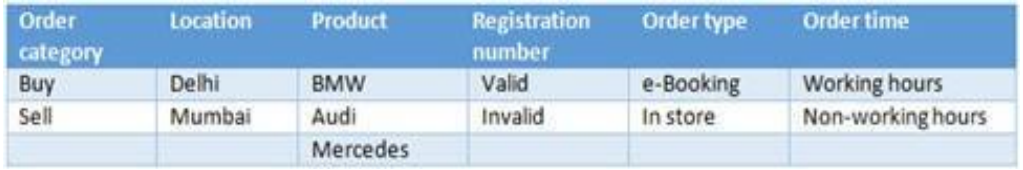
If we want to test all possible valid combinations:
= 2 X 2 X 3 X 5000 X 2 X 2
= 240000 Valid test cases combinations :(
There is also an infinite number of invalid combinations.

STEP #2: LET'S SIMPLIFY

- – Use a smart representative sample.
  – Use groups and boundaries, even when data is non-discrete.
  – Reduce Registration Number to Two
- Valid registration number
- Invalid registration number
- Now let's calculate the number of possible combinations
  = 2 X 2 X 3 X 2 X 2 X 2
  = 96

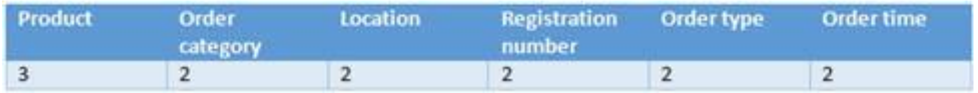
STEP #3: ARRANGING VARIABLES AND VALUES INVOLVED.

- When we arrange variables and values involved, it looks something like this.



Order category	Location	Product	Registration number	Order type	Order time
Buy	Delhi	BMW	Valid	e-Booking	Working hours
Sell	Mumbai	Audi	Invalid	In store	Non-working hours
		Mercedes			

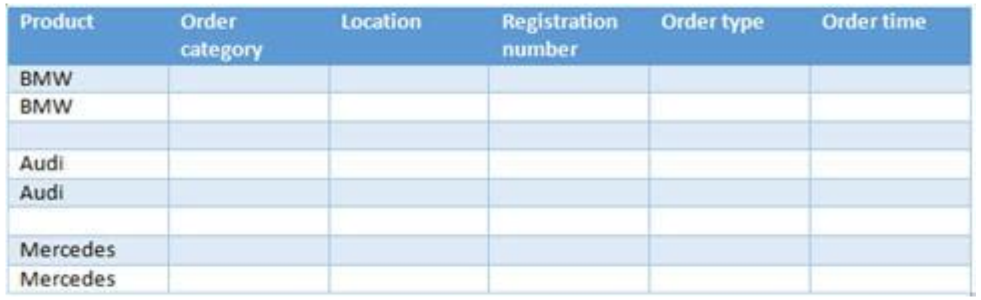
- Now order the variables so that the one with the most number of values is first and the least is last.



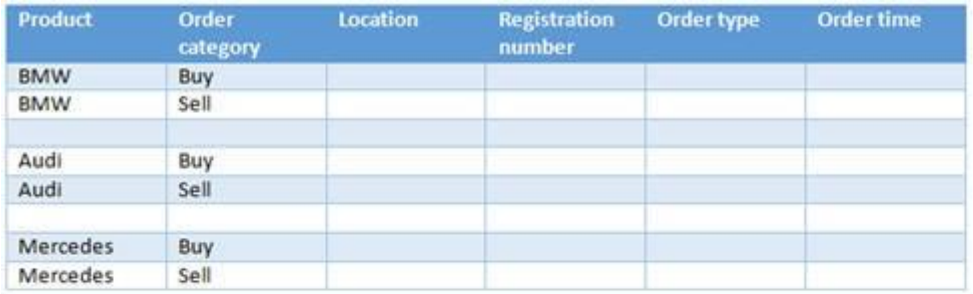
Product	Order category	Location	Registration number	Order type	Order time
3	2	2	2	2	2

STEP #4: ARRANGE VARIABLES TO CREATE A TEST SUITE

- Start filling in the table column by column. Initially, the table should look something like this. The three values of **Product** (variable having the highest number of values) should be written two times each (two is the number of values of next highest variable i.e. **Order category**).

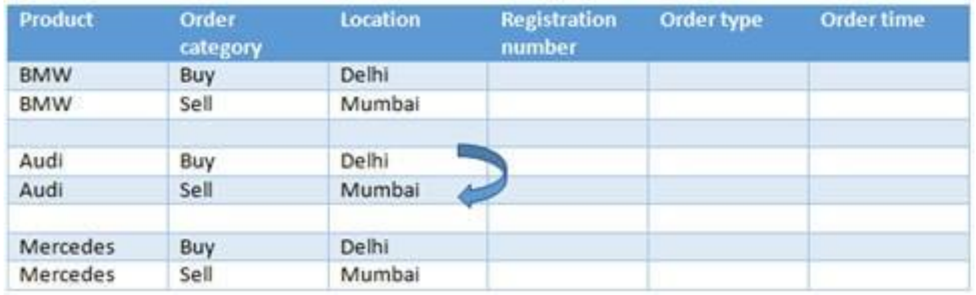


Product	Order category	Location	Registration number	Order type	Order time
BMW					
BMW					
Audi					
Audi					
Mercedes					
Mercedes					



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy				
BMW	Sell				
Audi	Buy				
Audi	Sell				
Mercedes	Buy				
Mercedes	Sell				

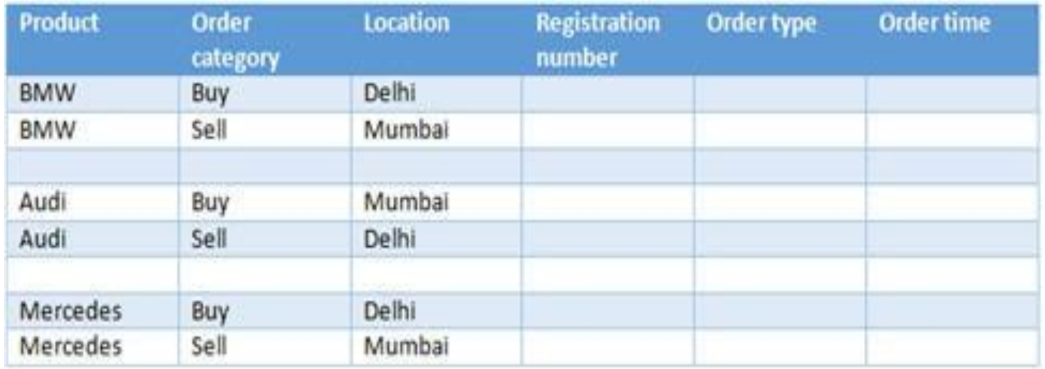
- For each set of values in column 1, we put both values of column 2. Repeat the same for column 3.



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi			
BMW	Sell	Mumbai			
Audi	Buy	Delhi			
Audi	Sell	Mumbai			
Mercedes	Buy	Delhi			
Mercedes	Sell	Mumbai			

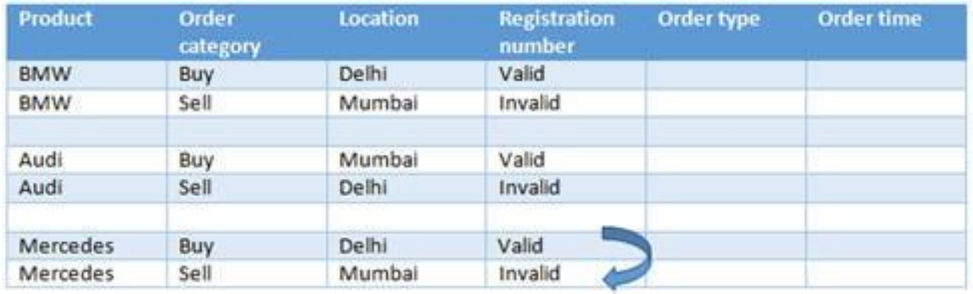
STEP #4: ARRANGE VARIABLES TO CREATE A TEST SUITE

- We have a Buy and Delhi, but wait – there's no Buy and Mumbai. We have a Sell and Mumbai, but there's no Sell and Delhi. Let's swap around the values in the second set in the third column.

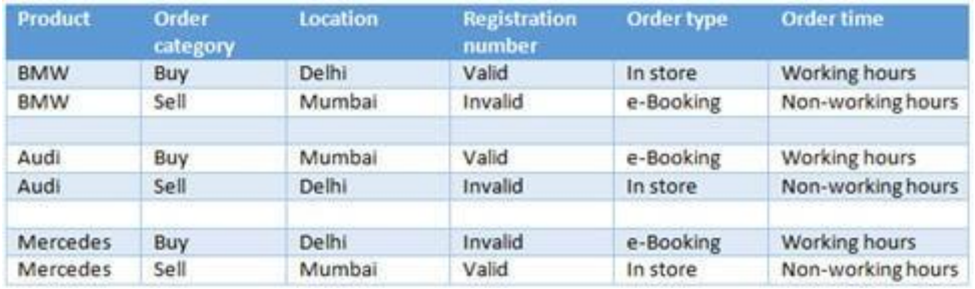


Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi			
BMW	Sell	Mumbai			
Audi	Buy	Mumbai			
Audi	Sell	Delhi			
Mercedes	Buy	Delhi			
Mercedes	Sell	Mumbai			

We will repeat the same steps for column 3 and 4.



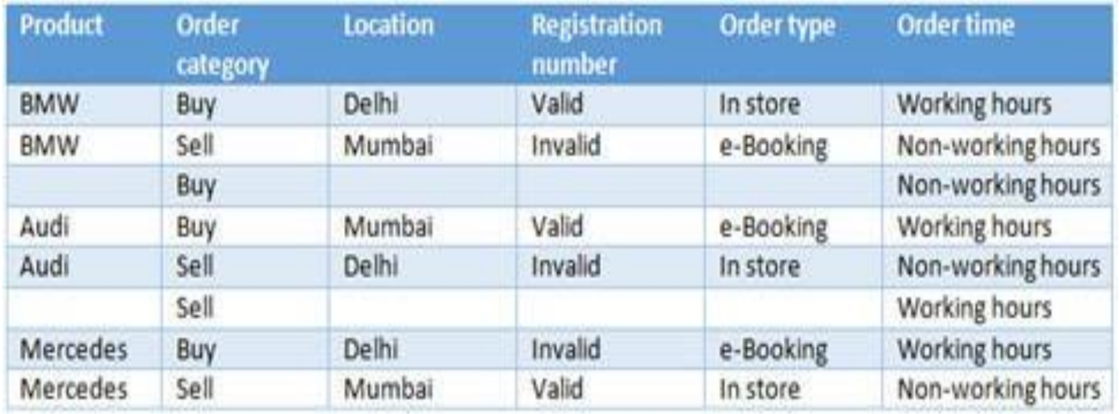
Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid		
BMW	Sell	Mumbai	Invalid		
Audi	Buy	Mumbai	Valid		
Audi	Sell	Delhi	Invalid		
Mercedes	Buy	Delhi	Valid		
Mercedes	Sell	Mumbai	Invalid		



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

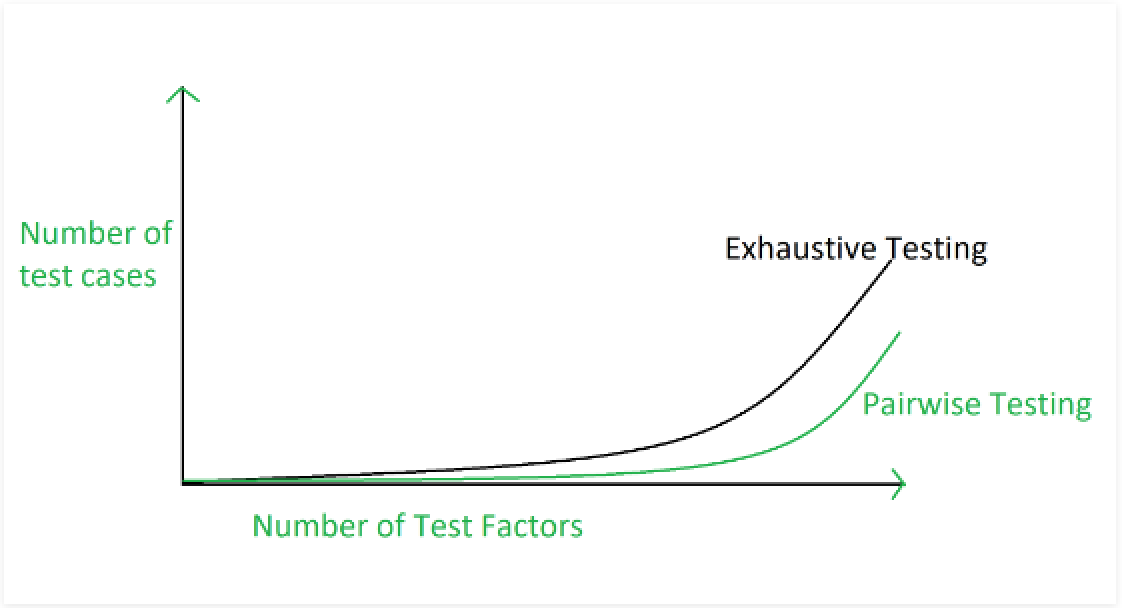
HURRAY! ALL PAIRS IN 8 CASES, INSTEAD OF ALL COMBINATIONS IN 96!

- Column 6 (Order time) is problematic. We are missing Buy/Non-working hours and Sell/Working hours. We can't fit our missing pairs by swapping around values as we already swapped all the rows if we swap now we may miss other possible pairs which are already sorted. So, we add two more test cases that contain these pairs. Hence, the blank rows!



Product	Order category	Location	Registration number	Order type	Order time
BMW	Buy	Delhi	Valid	In store	Working hours
BMW	Sell	Mumbai	Invalid	e-Booking	Non-working hours
	Buy				Non-working hours
Audi	Buy	Mumbai	Valid	e-Booking	Working hours
Audi	Sell	Delhi	Invalid	In store	Non-working hours
	Sell				Working hours
Mercedes	Buy	Delhi	Invalid	e-Booking	Working hours
Mercedes	Sell	Mumbai	Valid	In store	Non-working hours

GRAPHICAL REPRESENTATION OF PAIRWISE TESTING



ADVANTAGES & DISADVANTAGES OF PAIRWISE TESTING

- Pairwise testing reduces the number of execution of test cases.
- Pairwise testing increases the test coverage almost up to hundred percentage.
- Pairwise testing increases the defect detection ratio.
- Pairwise testing takes less time to complete the execution of the test suite.
- Pairwise testing reduces the overall testing budget for a project.

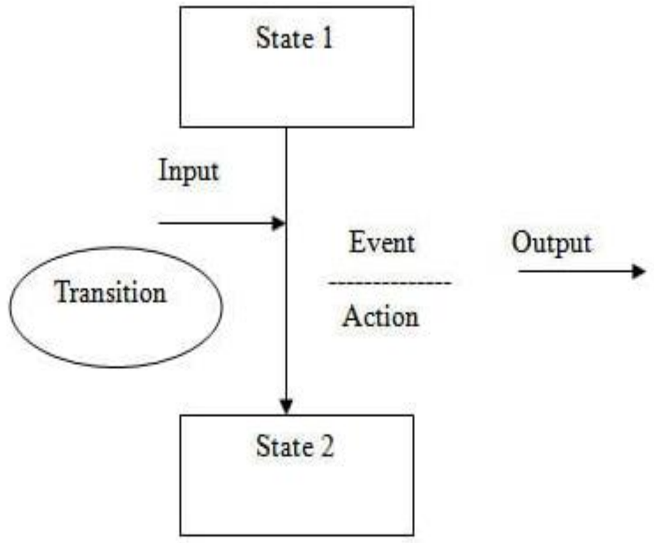
- Pairwise testing is not beneficial if the values of the variables are inappropriate.
- In pairwise testing it is possible to miss the highly probable combination while selecting the test data.
- In pairwise testing, defect yield ratio may be reduced if a combination is missed.
- Pairwise testing is not useful if combinations of variables are not understood correctly.

STATE TRANSITION TESTING

- Write test cases from given software models using state transition diagrams/tables.
- Many systems have a number of stable states and transitions from one stable state to another stable state. These machines are called "finite state machines".
- State transition testing is based on the fact that the same action will give different results depending on the initial state for this action.
- The actions that lead from one state to another are called transitions.

STATE TRANSITION TESTING

- This technique is based on the following steps:
- identify the different stable states of the system, including temporary states, including the initial and final states;
- for each state, identify the transactions, events, conditions, and actions that can be executed in this state;
- model the system as a graph (or drawing) or as a table, in order to use it;
- for each combination of state, event, and condition, verify the actions and resulting states.

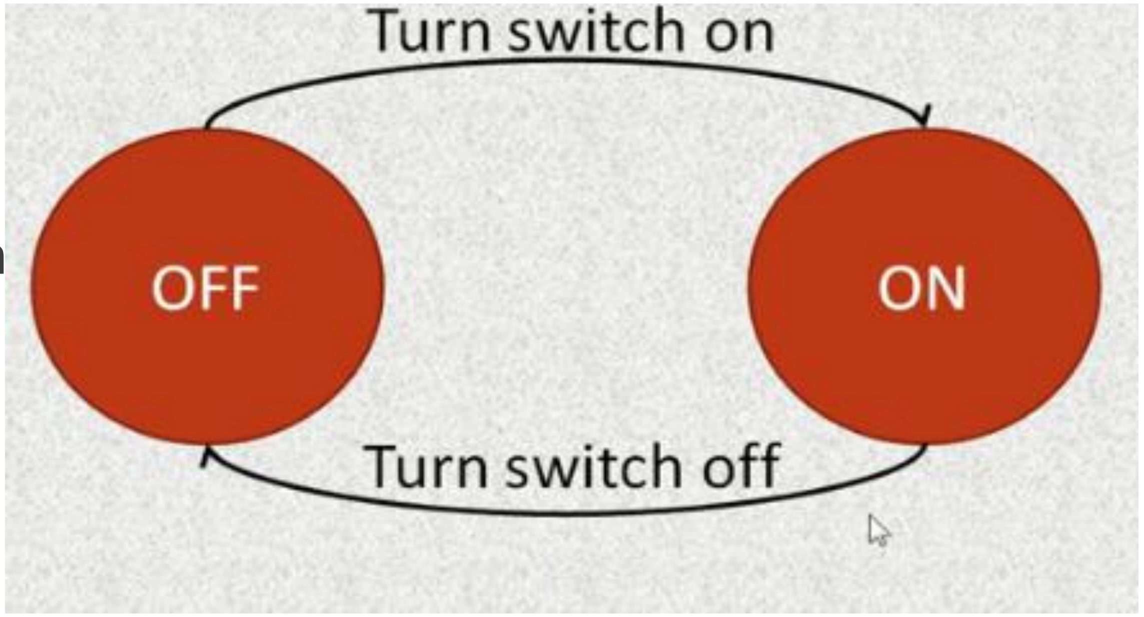


STATE TRANSITION TESTING

- A state is represented as a circle or oval
- A transition from one state to another is represented by an arrow going from the initial state to the resulting state (which can be the same as the initial state), with a description of the action executed,
- The initial state is identified by an arrow to that state that comes from a point outside the graph
- The representation of the final state is obtained by an arrow that leaves that state to reach a circle

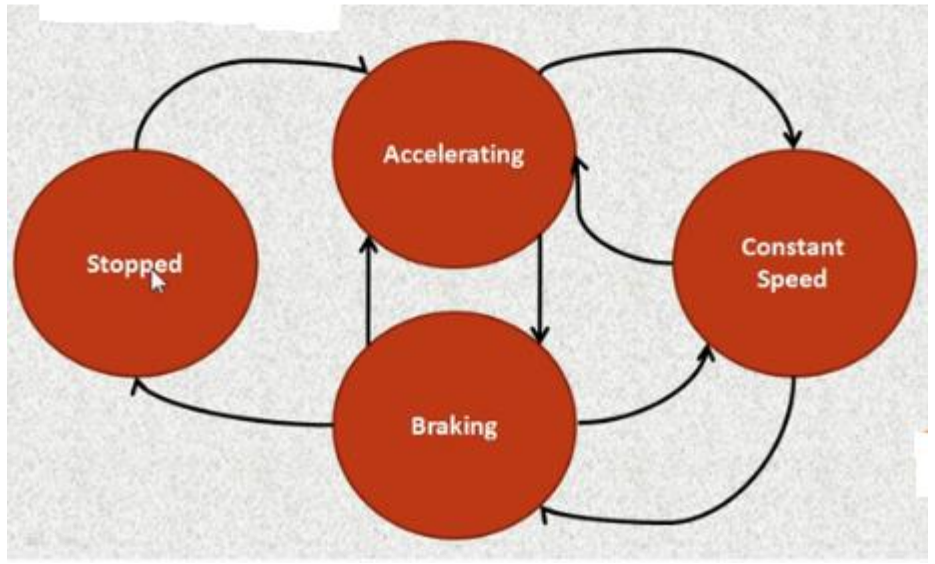
EXAMPLE: LIGHT SWITCH

- States are On> Off> On

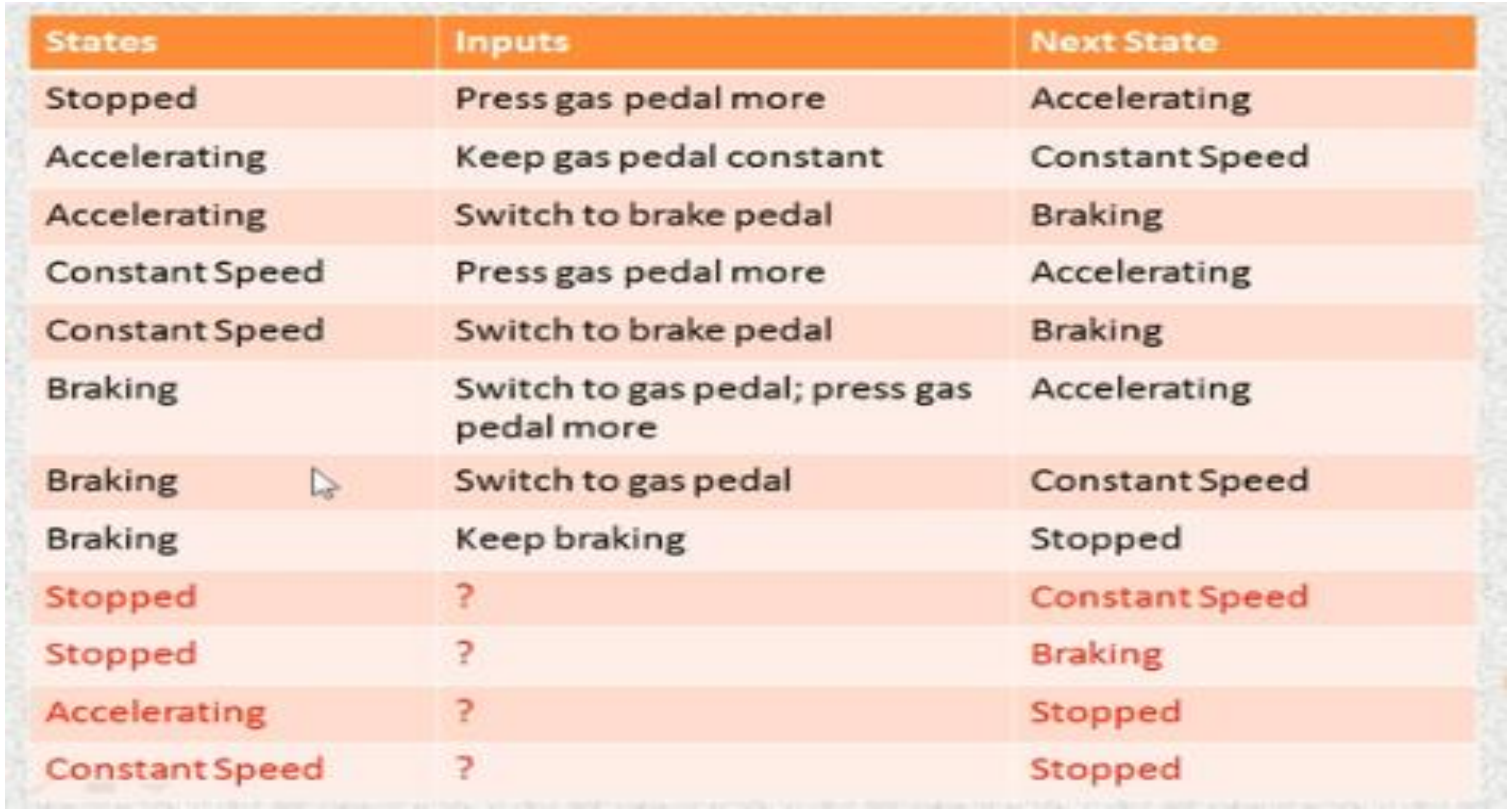


EXAMPLE 2: CAR STATES

- States are stopped, accelerating, constant speed and decelerating

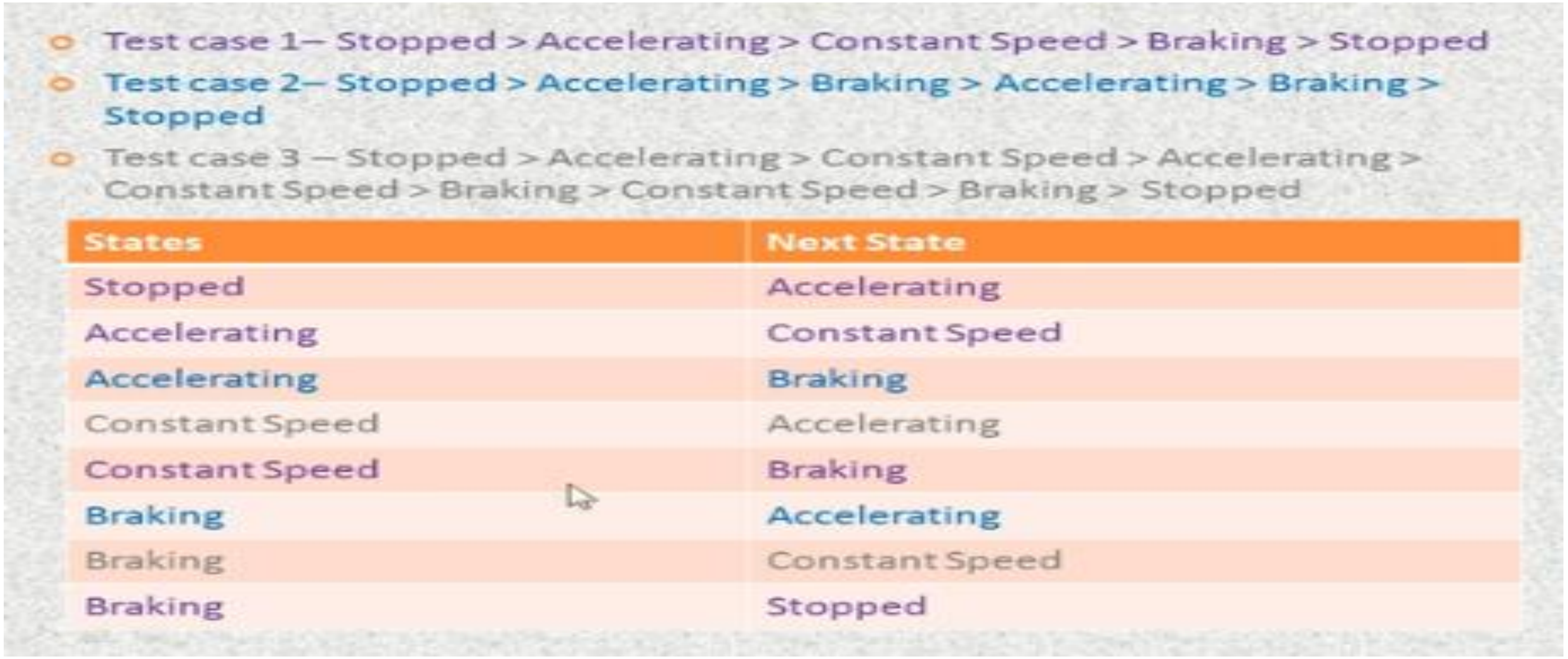


EXAMPLE: CAR STATE TABLE



States	Inputs	Next State
Stopped	Press gas pedal more	Accelerating
Accelerating	Keep gas pedal constant	Constant Speed
Accelerating	Switch to brake pedal	Braking
Constant Speed	Press gas pedal more	Accelerating
Constant Speed	Switch to brake pedal	Braking
Braking	Switch to gas pedal; press gas pedal more	Accelerating
Braking	Switch to gas pedal	Constant Speed
Braking	Keep braking	Stopped
Stopped	?	Constant Speed
Stopped	?	Braking
Accelerating	?	Stopped
Constant Speed	?	Stopped

EXAMPLE: CAR TEST CASES



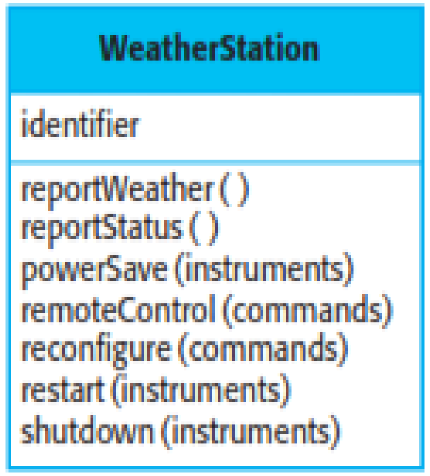
- Test case 1– Stopped > Accelerating > Constant Speed > Braking > Stopped
- Test case 2– Stopped > Accelerating > Braking > Accelerating > Braking > Stopped
- Test case 3 – Stopped > Accelerating > Constant Speed > Accelerating > Constant Speed > Braking > Constant Speed > Braking > Stopped

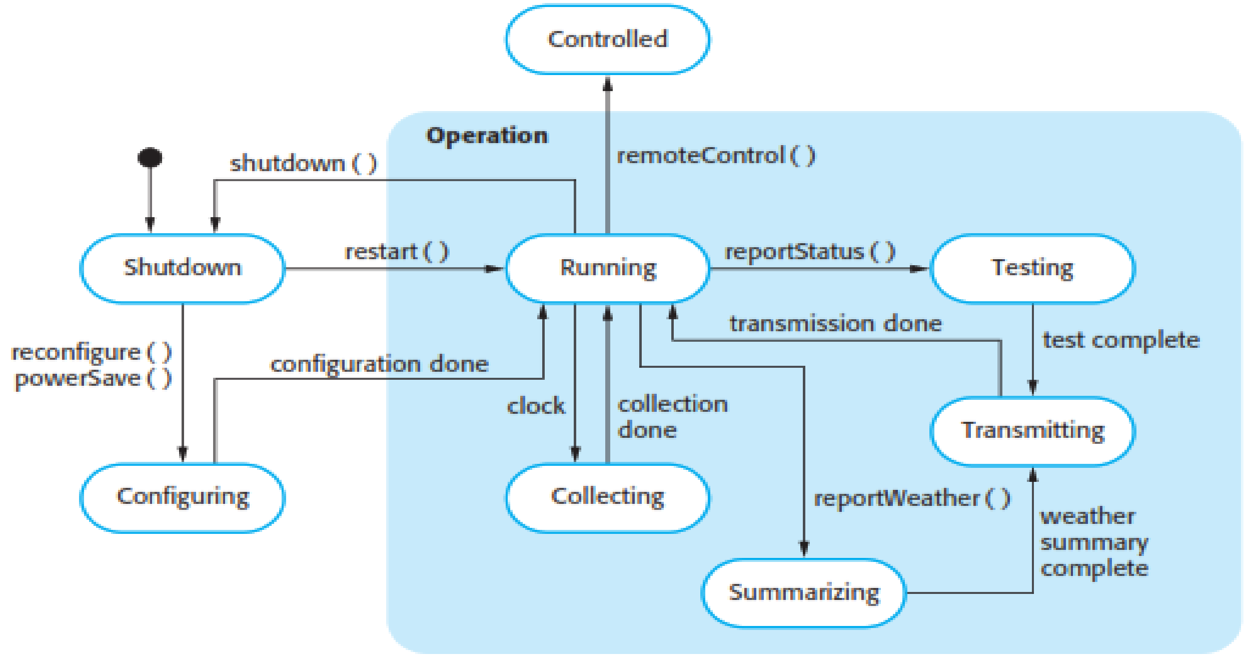
States	Next State
Stopped	Accelerating
Accelerating	Constant Speed
Accelerating	Braking
Constant Speed	Accelerating
Constant Speed	Braking
Braking	Accelerating
Braking	Constant Speed
Braking	Stopped

STATE TRANSITION TESTING

- A graphical representation of states and transitions quickly becomes cluttered and difficult to visualize if the graph has too many states or transitions. It is then necessary to group many different sub-states and their transitions in separate graphs showing the operation of the sub-system, with its inputs and outputs.
- To identify potentially forgotten actions, it is possible to explore the system by trying all possible actions at all the possible states.
- Identification of the valid states and transitions allows identification of invalid or non-authorized transitions from that state.
- It is thus possible to verify that the possible actions from that state do not allow execution of invalid transactions.
- Identification of invalid transitions and actions must be envisaged to ensure that such transitions and states are indeed impossible.
- Analysis of valid and invalid transitions, from all possible states of a system enables the design of test conditions, and sometimes identification of transitions that were not anticipated.

WEATHER STATION STATE DIAGRAM





Shutdown → Running → Shutdown

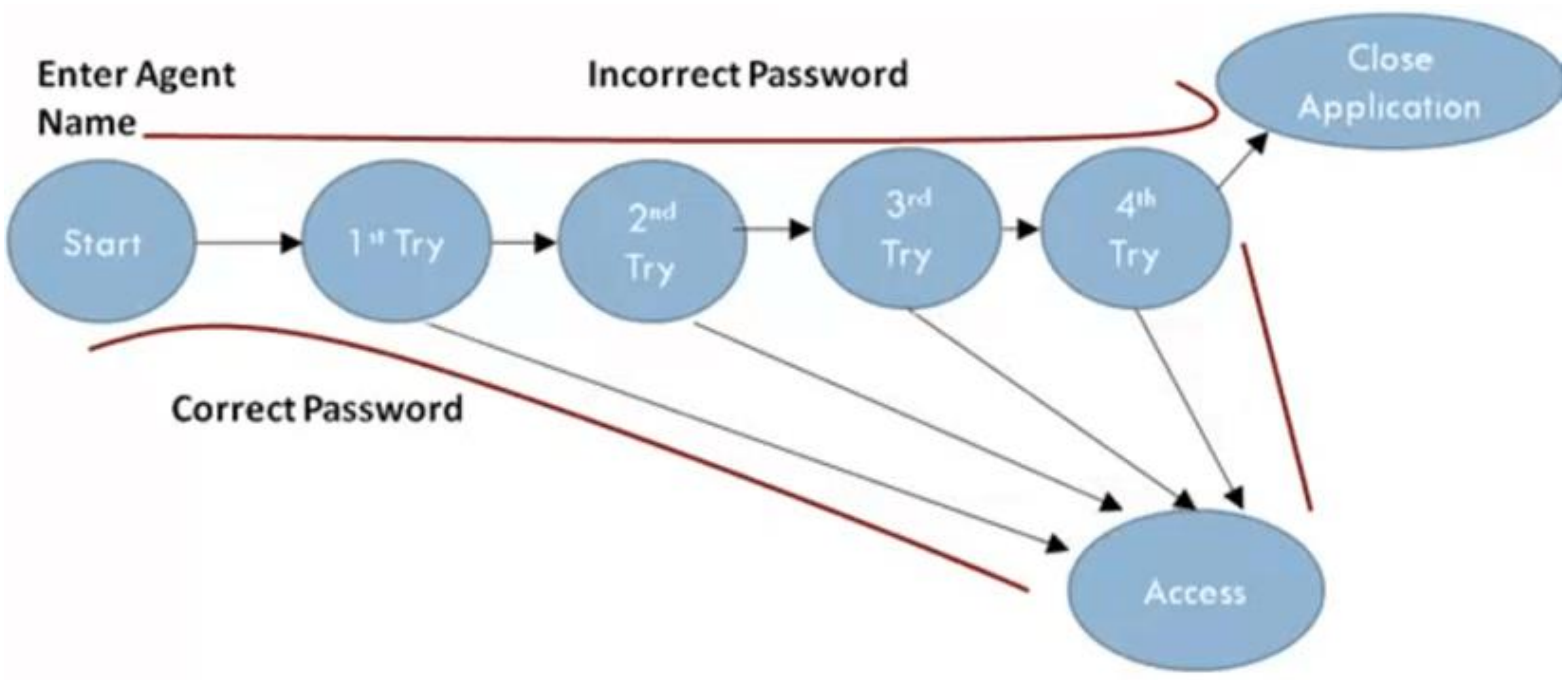
Configuring → Running → Testing → Transmitting → Running

Running → Collecting → Running → Summarizing → Transmitting → Running

LIMITATIONS AND ASSUMPTIONS

- The main assumptions with state transition testing are that we have identified all states and all transitions.
- Any possible "un-identified" state or transition can lead to potential defects being left for the customer to find.

EXAMPLE


Enter Agent Name
Incorrect Password
Close Application
Start
1st Try
2nd Try
3rd Try
4th Try
Correct Password
Access

DOMAIN ANALYSIS TESTING

- A software testing technique in which application is tested by providing input data and verification of relevant output.
- In domain testing, testing takes place with the minimum number of input data so that the application does not allow invalid and out of range data and evaluate the expected range of output.
- Domain testing ensures that an application does not have input data outside the mentioned valid range. This testing process is used to verify the application for its ability to react or operate for different variety of input values.

DOMAIN ANALYSIS TESTING

- In domain testing, we divide a domain into sub-domains (equivalence classes) and then test using values from each subdomain.
- For example, if a website (domain) has been given for testing, we will be dividing the website into small portions (subdomain) for the ease of testing.
- Domain might involve testing of any one input variable or combination of input variables. Practitioners often study the simplest cases of domain testing less than two other names, "boundary testing" and "equivalence class analysis."
- Boundary Testing– Boundary value analysis (BVA) is based on testing at the boundaries between partitions. We will be testing both the valid and invalid input values in the partition/classes.
- Equivalence Class testing– The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning.'

STRATEGY OF DOMAIN TESTING

1. **Domain Selection**
   - Input variables that need to be assigned, and the proper result has to be verified.
2. **Group the Input Data into Classes**
   - A similar type of input data is partitioned into subsets. There are two types of partitioning, Equivalence class partitioning & Boundary value analysis (BVA).
3. **Input Data of the Classes for Testing**
   - The boundary values should be considered as the data for testing. Boundaries represent the equivalence classes more likely to find an error than the other class members. A data in between the range is the best representative of an equivalence class.
4. **Verification of Output Data**
   - When input data is assigned to application concerning that output data verified. Output data should be invalid and specified range.

EXAMPLES OF DOMAIN TESTING

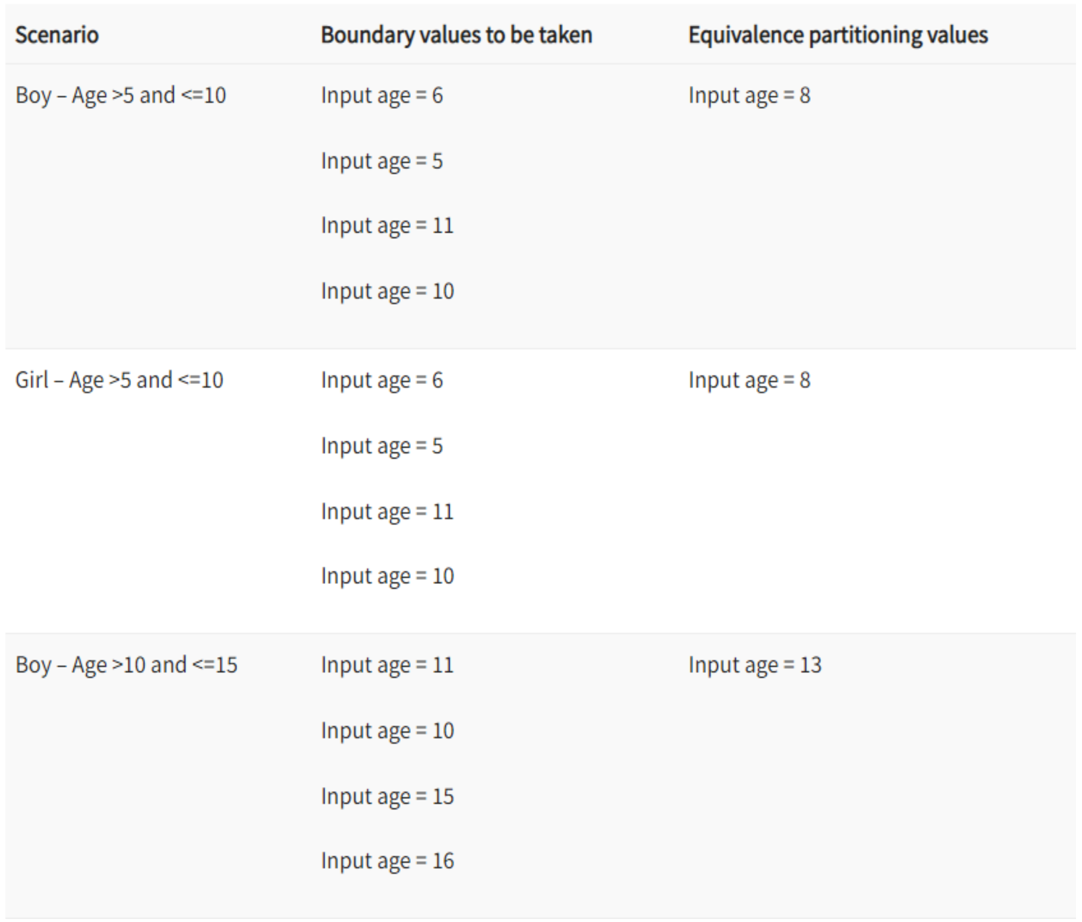
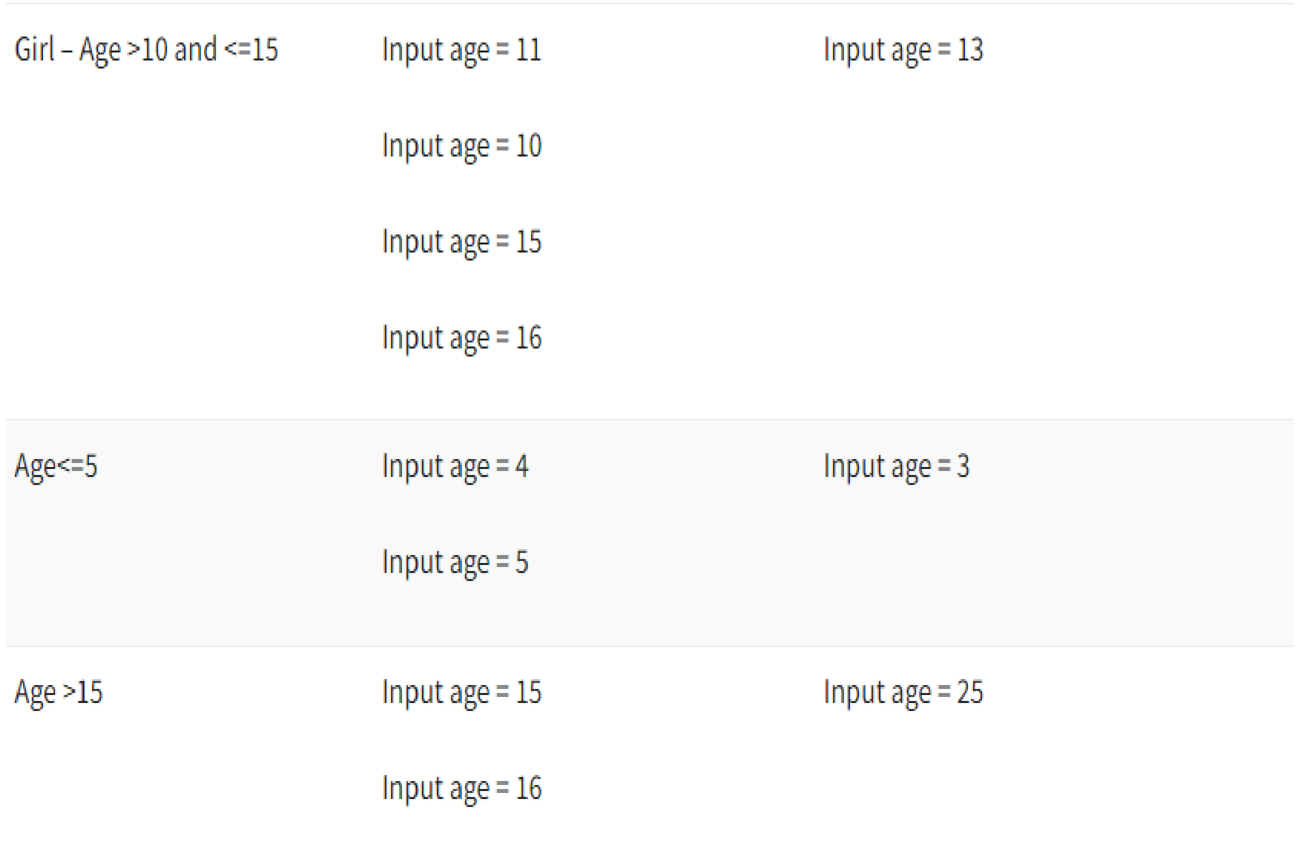
- Consider a games exhibition for Children, 6 competitions are laid out, and tickets have to be given according to the age and gender input. The ticketing is one of the modules to be tested in for the whole functionality of Games exhibition.
- According to the scenario, we got six scenarios based on the age and the competitions:
- Age >5 and <10, Boy should participate in Storytelling.
- Age >5 and <10 , girl should participate in Drawing Competition.
- Age >10 and <15, Boy should participate in Quiz.
- Age >10 and <15 , girl should participate in Essay writing.
- Age<5, both boys and girls should participate in Rhymes Competition.
- Age >15, both boys and girls should participate in Poetry competition.
- The input will be Age and Gender and hence the ticket for the competition will be issued.

EQUIVALENCE CLASSES

- For the above example, we are partitioning the values into a subset or the subset. We are partitioning the age into the below classes :
- **Class 1:** Children with age group 5 to 10
- **Class 2 :** Children with age group less than 5
- **Class 3:** Children with age group age 10 to 15
- **Class 4:** Children with age group greater than 15.

BVA FOR EQUIVALENCE CLASSES

- For example for the scenario #1:
- **Class 1:** Children with age group 5 to 10 (Age >5 and <=10)
- **Boundary values:**
- Values should be Equal to or lesser than 10. Hence, age 10 should be included in this class.
- Values should be greater than 5. Hence, age 5 should not be included in this class.
- Values should be Equal to or lesser than 10. Hence, age 11 should not be included in this class.
- Values should be greater than 5. Hence, age 6 should be included in this class.

Scenario	Boundary values to be taken	Equivalence partitioning values
Boy – Age >5 and <=10	Input age = 6 Input age = 5 Input age = 11 Input age = 10	Input age = 8
Girl – Age >5 and <=10	Input age = 6 Input age = 5 Input age = 11 Input age = 10	Input age = 8
Boy – Age >10 and <=15	Input age = 11 Input age = 10 Input age = 15 Input age = 16	Input age = 13
Girl – Age >10 and <=15	Input age = 11 Input age = 10 Input age = 15 Input age = 16	Input age = 13
Age<=5	Input age = 4 Input age = 5	Input age = 3
Age >15	Input age = 15 Input age = 16	Input age = 25

DOMAIN TESTING STRUCTURE

- Identify the potentially interesting variables.
- Create and identify boundary values and equivalence class values as above.
- Identify secondary dimensions and analyze each in a classical way. (In the above example, Gender is the secondary dimension).
- Identify and test variables that hold results (output variables).
- Evaluate how the program uses the value of this variable.
- Identify additional potentially-related variables for combination testing.
- Summarize your analysis with a risk/equivalence table.

That is all