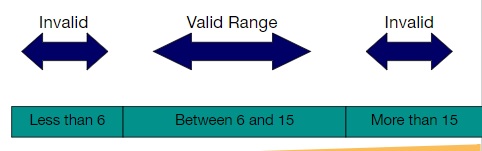
# **Types of Specification Based/ Black Box testing techniques**

1. Equivalence Partitioning
2. Boundary Value analysis
3. Decision Table Testing
4. State Transition Testing
5. Orthogonal Array Testing (OAT)
6. All Pairs

## **1. Equivalence Partitioning**

### **1.1 Approach**

1. It divides the input into classes of data for which the test cases are generated
2. Attempting to uncover classes of error: The data is divided into classes for Testing, the same partitioning will also help in uncovering any issues or errors
   * eg: for Equivalence class partitioning (ECP), there would be several valid and invalid inputs which will give rise to errors and correct results
3. Divide the input domain of a program into classes of data
4. Divide the test cases based on the partitions
5. An equivalence class is a set of valid and invalid inputs
6. The test design is based on the equivalence classes for an input domain



### **1.2 Examples**

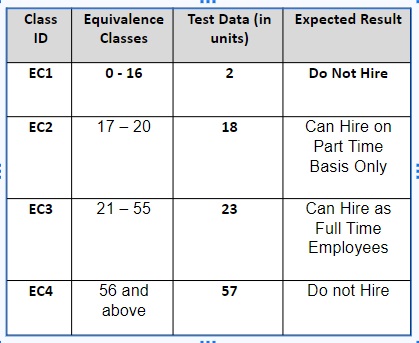
1. If the input condition is a continuous range, **there is one valid class and two invalid classes**
   * eg: the input variable is a mortgage application’s income
   * **The Valid range:** $1000/month to $75, 000/month
   * **Valid Class:** {1000 ≤ income ≤ 75000}
   * **Invalid Classes:** {income > 10000} | {income> 75000}
2. If the input condition is a discrete range of permissible values, **there is one valid class and two invalid classes**
   * eg: The input variable is the total number of houses being purchased, from 1 to 5
   * **Valid class:** {1 ≤ no\_of\_houses ≤ 5}
   * **Invalid class:** {no\_of\_houses <1} | {no\_of\_houses ≥5}
3. If the input specifies a set of values, **there is one valid class and one invalid class**
   * Example: Types of houses are Condo, townhouse and a single family
   * **Valid class:** {Condo, townhouse, single family}
   * **Invalid class:** {...anything else...}
4. if a “must be” condition is required, **there is one valid class and one invalid class**
   * Example: the mortgage applicant must be a person
   * **Valid class:** {person}
   * **Invalid class:** {....anything but a person...}

### **1.3 Test Cases**

1. Define the equivalence class
2. Write the first class to cover as much of the valid classes as possible
3. Continue writing test cases for all the valid classes until all the valid classes have been included
4. Write one test case for each invalid class

### **1.4 Real time example**

* A Human Resource system processes applicants based on age. The organization's rules are
  + 0 – 16: Do Not Hire
  + 17 – 20: Can Hire on Part Time Basis Only
  + 21 – 55: Can Hire as Full Time Employees
  + 56 and above: Do not Hire
* To Test this problem completely, we need to test for persons in the age group of 0, 1, 2,3 ,4,5 … 99, 100.



* Since this is not possible, divide the input range into equivalent classes.
* Select sample data from each partition (Eg 2, 18, 23, 57).
* Test the application with the selected sample data.

### **1.5 Guidelines**

1. Applicability
   * When the input data takes on values within ranges or set
   * Therefore, reducing the number of test cases created and executed
2. Advantages
   * Eliminates the need of exhaustive testing, which is not feasible
   * enables the tester to cover large domains of input or outputs
   * Because of the smaller subsets selected from an equivalence class
   * It enables the tester to select a subset of test inputs with probability of detecting a defect is high
3. Limitations
   * It makes assumptions that the data in the same equivalence class is processed in the same way by the system
   * ECP is not a stand alone method. It has to be supplemented by Boundary Value Analysis

## **2. Boundary Value Analysis**

### **2.1 What is Boundary Value Analysis?**

1. It is the act of dividing the given inputs based on a relation
2. The inputs are divided into groups
3. Which is treated the same way by the module or which should produce the same results
4. BVA focuses on the boundary simply because that is where many defects hide.
5. Experienced testers have encountered the situation many times
6. Inexperienced tester have the intuitive feel that a mistake happens more often at the boundary

### **2.2 Steps for using BVA**

1. Identify the Equivalence class partition (ECP) from the scenario or requirement
2. Identify the boundaries for the ECP
3. Create a test case for each of the boundary value by choosing
   * One value above the boundary
   * one value below the boundary
   * one value of the boundary
4. From the results, one member in the ECP extrapolate results for all the values in the partition
5. Example: If the boundary is 15 and the unit is integer, then the ‘below’ point is 14 and the ‘above’ point is 16.
6. A point just above one boundary may be in another equivalence class. There is no reason to duplicate the test.
7. The same may be true of the point just below the boundary.

### **2.3 Example of Boundary Value Analysis**

The Electricity bill computed by the service provider has a fixed component as well as a running component. All customers are charged at a rate of $40 flat as a fixed component. In addition to this, they would be charged a running component and/or a fine, depending upon their amount of consumption or usage. The rules for this are given below:

* If the number of units consumed by the consumer is less than 10 units, then the running cost is not charged for the consumer.
* If the number of units is between 11 and 20, then the running cost is charged at $1 per unit.
* If the number of units is between 21 and 40, then the running cost is charged at $2 per unit.
* If the number of units exceeds 40, then the running cost is charged at $5 per unit.
* Use Boundary Value Analysis to decide on the test cases to be designed.

| sr. no | Boundary Value Analysis | n+1 | n | n-1 | HLD |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 -10 | 2 & 11 | 1 &10 | 0 &9 | 0,1,2,9,10,11 |
| 2 | 11-20 | 12 &21 | 20 | 19 | 10,11,12,19,20,21 |
| 3 | 21 - 40 | 22 & 41 | 40 | 39 | 20,21,22,39,40,41 |
| 4 | 40 or more | 42 | 41 | - | 40,41,42 |

### **2.3 Applications, Advantages, Limitations**

**Applications:**

* Can be used in all scenarios where inputs that can be partitioned and boundaries that can be identified
* Typically be suited to systems in which much of the input data takes on ranges of values or within sets

**Advantages:**

* Boundaries and conditions are the two major sources of defect in software products.
* This technique aims to identify defects in these areas.

**Limitations**

* Does not work well for Boolean and logical variables.
* Not that useful for strongly-typed languages.

## **3. Decision Tree**

### **3.1 Introductions**

* Decision Tables can be used when the outcome or the logic involved in the program is based on a set of decisions and rules which need to be followed.
* A decision table lists the various decision variables, the conditions (or values) assumed by each of the decision variables and the actions taken in each combination or conditions.
* Variables that contribute to the decision table are listed as the columns of the table
* Last column of the table is the action to be taken for a combination of values of the decision variables.
* The Decision Table is a tool for ensuring excellent test coverage when the requirements can be represented as rules
* The tester still has to pick the exact data values which will make each rule “fire”.

### **3.2 Decision Table Representation**

* A table listing all possible “conditions” (inputs) and all possible “actions” (outputs)
* There is a “rule” for each possible combination of “conditions”
* For each “condition”, it is identified as a “yes” (present), a “no” (not present) or an “X” for immaterial (the result is the same for either yes or no)
* Good for highly critical software
* Considers all possible combinations

### **3.3 Approach**

1. Analyze the input requirements and write down all the possible conditions
2. The Decision table will have only two outcomes (True or False) depending how they word the outcome (yes or no, exempt or nonexempt)
3. Based on this, we can calculate the number of rules  
   * where 2 = Number of outcomes which is always equal to 2
5. Fill the columns of the decision table with all the possible columns
6. For each of the columns fill up the expected action and result
7. Create a test case for each of the rules

**How to fill the Columns:**

1. Say we have a table with 16 columns because (2^4 = 16)
2. Then first fill 16/2 = 8 with True and the last 8 with False
3. Then for the second row, fill the first 16/4 = 4 with True and the next four with false, keep alternating till you complete
4. Then for the next row we fill the first 16/8 = 2 with True and the next 2 with false
5. Finally T, F, T, F (alternate with true and false)

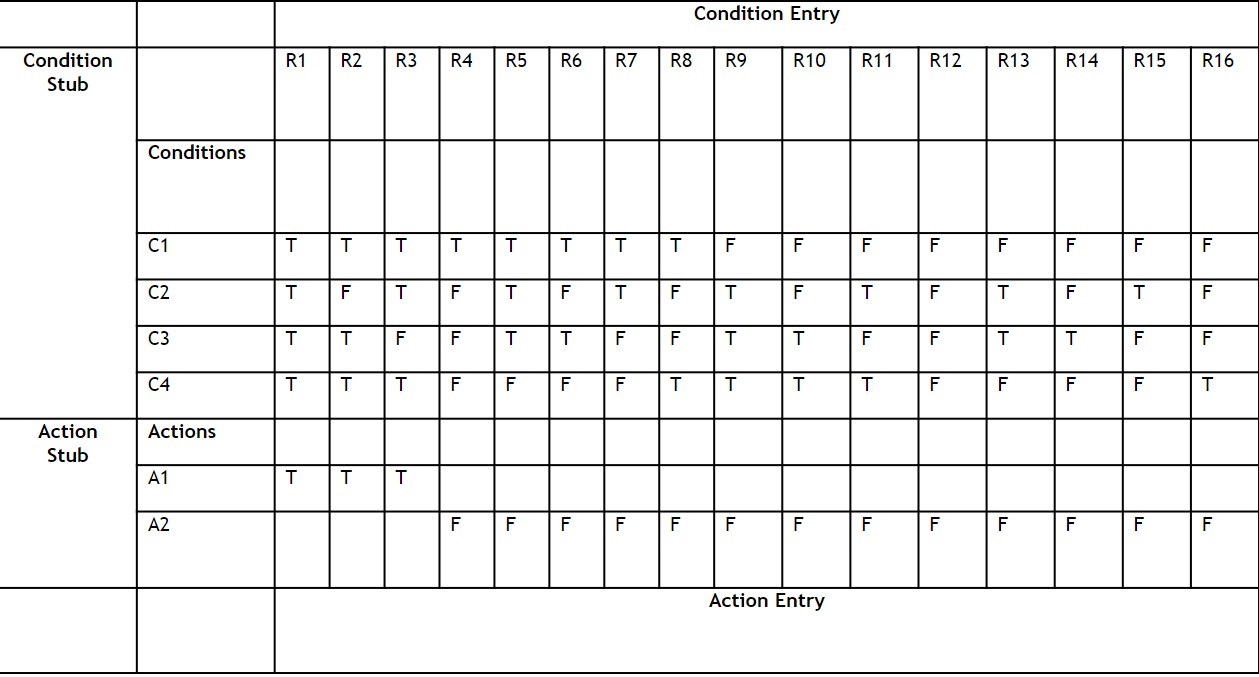
### **3.4 Example**

* Engineering Examination Result for a student is based on the following conditions:
* If the student has 80% attendance and has attended 3 internal tests with an average of 10 or more or has attended 2 internal tests with an average of 15 or more marks and has taken up the external examination and scored more than 35, then the student can be considered as pass in that subject.

1. formula = (number of outcomes)^(number of conditions)
2. number of outcomes = Pass or Fail
3. number of conditions :  
     
    C1: must have 80%  
     
    C2: must write 3 test with average of 10  
     
    C3: must write 2 test with average of 15  
     
    C4: appeared for an external exam and scored more than 35

**Therefore, the number of resultant = $2^{4} = 16$**

**The decision table is as follows**

****

### **3.5 Turning it to test cases**

→ Create the decision table by defining conditions and action combinations

→ Strike out any rules that are “impossible” (are they really? How can you guarantee it?)

→ Combine columns where the values are immaterial (we don’t care)

→ Select test data to make each rule “fire”

### **3.6 Applicability, Advantages and Guidelines**

**Applicability:**

* Decision Table Testing can be used whenever the system must implement complex business rules.
* They are efficient for describing situations where varying conditions produce different test actions.
* They are not suited for simple data-oriented applications that typically perform operations such as adding, deleting and modifying entries; a decision-table method is not appropriate.

**Advantages**

* Enables us to get a “complete” view, with no consideration of dependence
* Enables us to look at and consider “dependence,” “impossible,” and “not relevant” situations and eliminate some test cases.

**Limitations**

* When the number of conditions is large then the decision table becomes very huge and cumbersome.

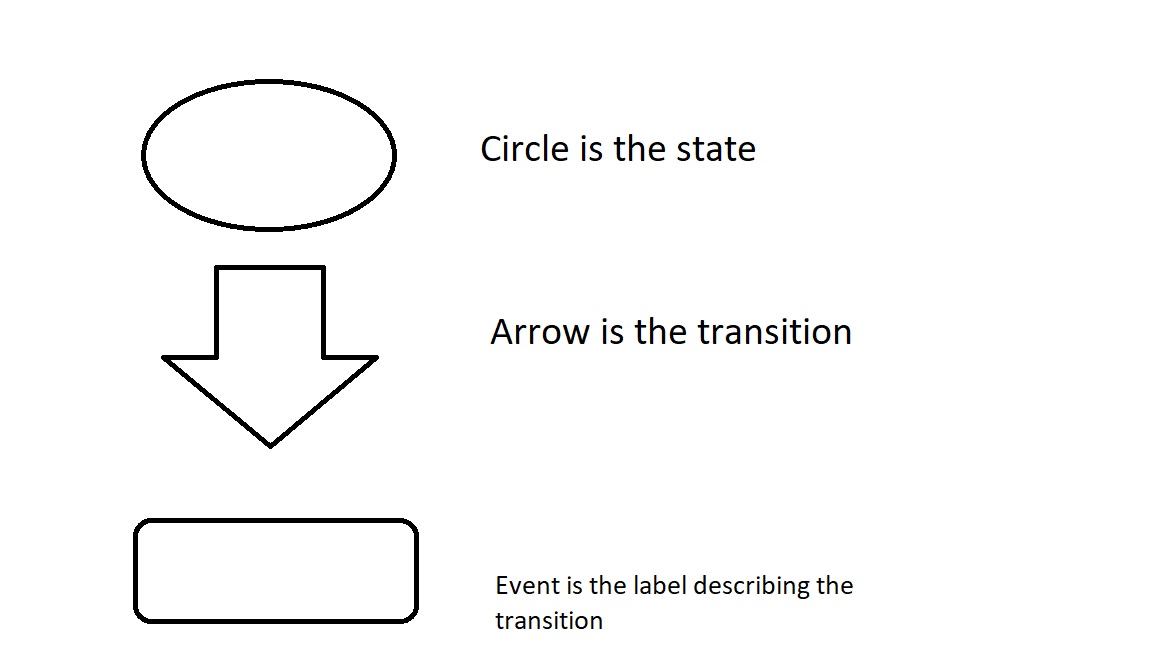
## **4. State Transition Based Testing**

### **4.1 Introduction**

* Excellent tool to capture certain types of system requirements and document internal system design.
* When a system must remember
  + what happened before or when valid and invalid orders of operation exist,
  + then state transition testing could be used
* It is useful in situations when Workflow modeling or data flow modeling has been done (i.e.; the system moves from one state to another)

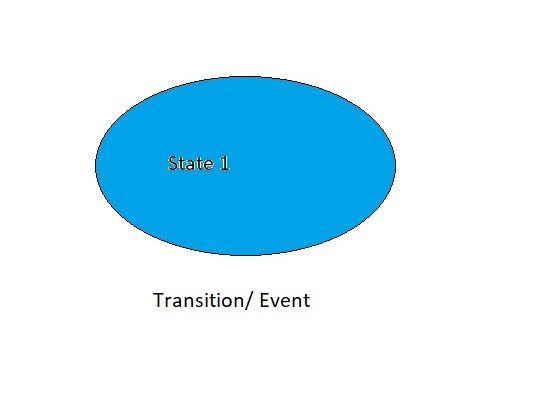
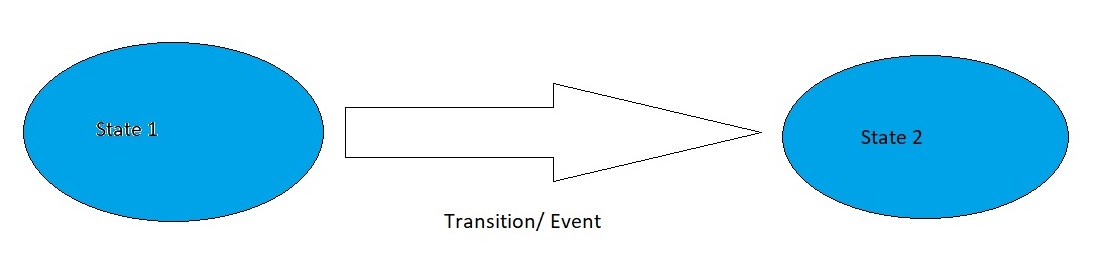
### **4.2 Representation**

1. State is represented by a circle
2. Transition is represented by an arrow
3. Event is represented by a label on the transition



1. Thus from the starting state to the end state the various transitions and routes are represented in the form of a transition diagram as mentioned.
2. Create test cases in such a way that all states are visited at least once, all events are triggered at least once and all paths are executed at least once
3. (i.e.; all transitions in the system are tested at least once)

### **4.3 Approach**

1. Understand the various states that the system, user, or object can be in, including the initial and final states.
2. Examples of states can be: ‘User raising a purchase order’ or ‘leave request is accepted’. These states will be represented as:  
     
    
3. Identify transitions, events, conditions, and actions that can - and can’t - apply in each state.  
     
    
4. Use a graph or table to model the system.
5. This graph or table also serves as an oracle to predict correct system behavior along with a requirements specification.
6. For each event and condition - that is, each transition - verify that the correct action and next state occurs.
7. Create test cases in such a way that all states are visited at least once, all events are triggered at least once and all paths are executed at least once
8. (i.e. all transitions in the system are tested at least once)

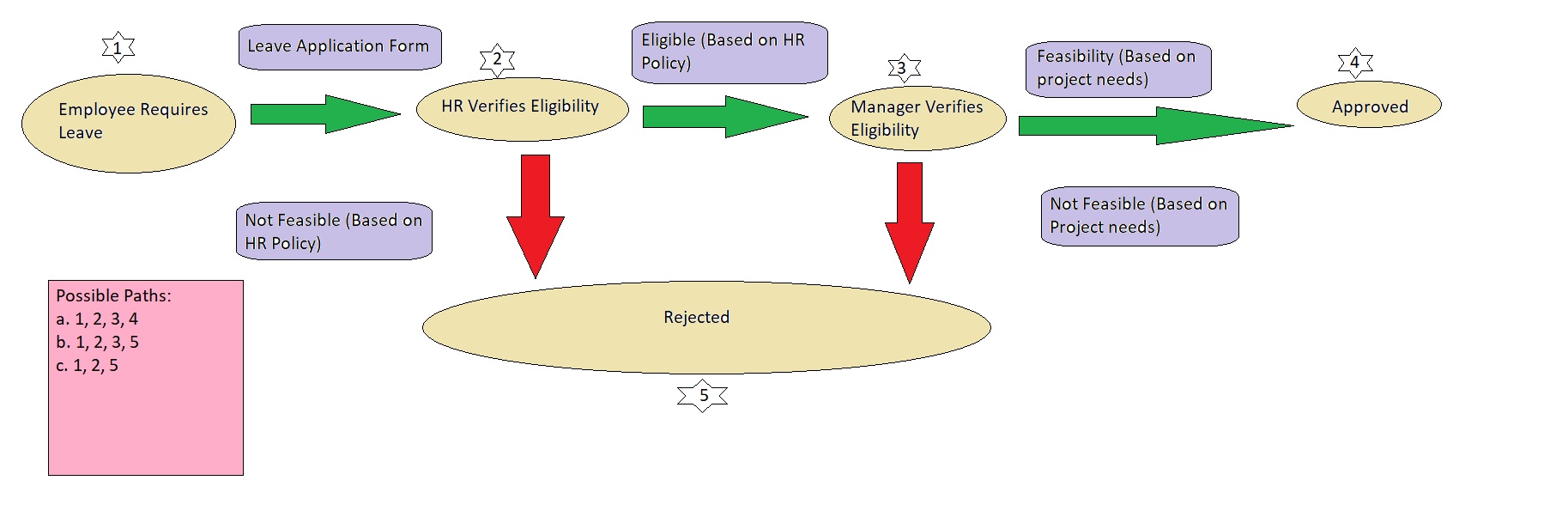
### **4.4 Example**

Consider a leave application system in an organization.

An employee can raise a request for a leave, and if he is eligible for a leave(based on the number of days he has already taken etc), the application is sent to the manager for approval.

The manager then validates and approves or rejects the leave based on the duration, reason for taking leave etc.

Now this problem can be represented in the form of a simple state transition graph



1. Since the possible paths are
   1. 1 - 2 - 3 - 4
   2. 1 - 2 - 3 - 5
   3. 1 - 2 - 5
2. Hence, three test cases are required to test the given scenario

### **4.5 Applicability, Advantages and Limitations**

**Applicability:**

Can be used when:

* The application can be characterized by a set of states
* Data values (click of a button, inputs etc) that cause the transition from one state to the other is well understood.
* Method of processing within each state is also understood clearly.
* State Transition testing cannot be used when the system does not have any state or does not respond to real time events.

**Advantages:**

* All possible states and transitions in a system would be covered (including valid and invalid).
* Critical when testing high risk systems like Avionics or medical devices where testing of all possible states and transitions is required (not just valid ones)

**Limitations:**

• It becomes very large and cumbersome when the number of states and events increase.

## **5. Orthogonal Arrays**

### **5.1 Introduction**

1. A device for selecting a “good” subset of all possible combinations
2. There will be too many combinations to consider since it is not impossible to have a large number of combinations to test. This is when we use orthogonal technique.
3. This Technique ensures that all pairs are tested at least once but not all combinations
4. The early detection of bugs can be found this way, and can reduce the realistic number of tests
5. It does require the tester to know all the legal combinations so as to eliminate the illegal combinations
6. It is risky to skip testing large parts of the functionality or combinations
7. So what is there as a compromise solution?
   * All PAIRS (each option with every other option ONCE, but not all combinations across all options): orthogonal array testing
   * Exercise multiple pairs simultaneously
   * Requires knowledge of all legitimate combinations

### **5.2 YouTube Explanation**

1. It is a type of Black Box testing
2. Useful to test large number of possible input combination
3. Type of pair wise testing
4. **Less number of test cases**

**What is Orthogonal Array?**

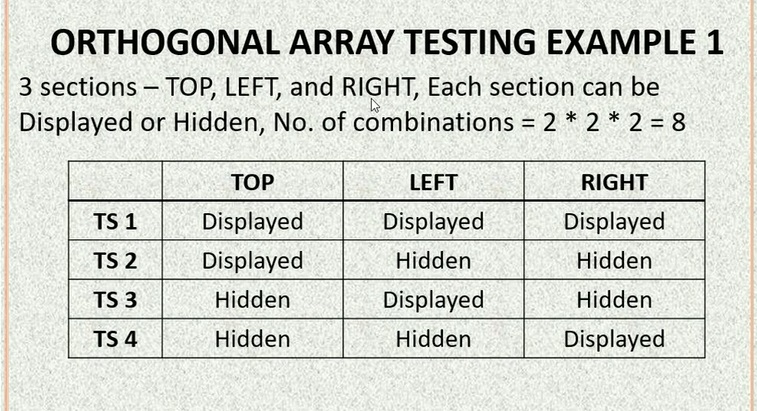
* It is a two dimensional array of numbers
* Any two columns give pairwise combinations
* The columns represent the independent variables
* Columns are aka factors
* Rows represent the combination of columns aka as experiments or runs
* The overall table is called as orthogonal array design of experiments

**Example:**

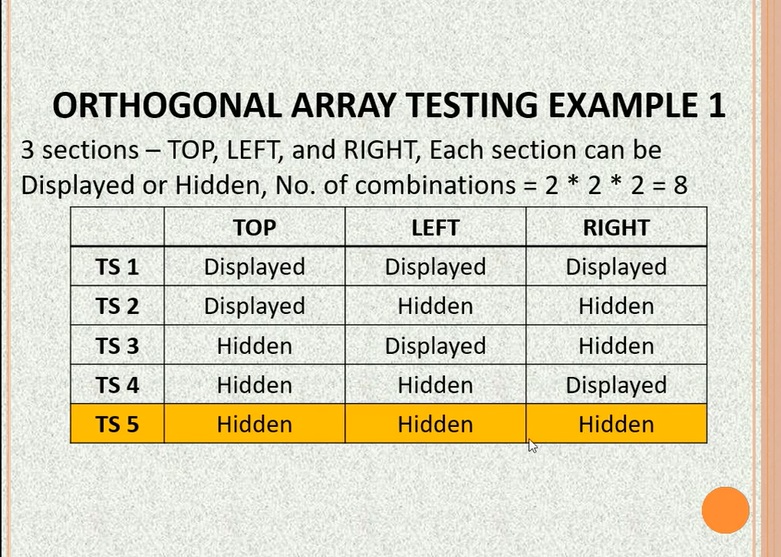
Say that a website is split into top, left and right

These sections can either be “Displayed” or “Hidden”

Then the orthogonal array are as follows



* But from the image, the number of experiments (rows) are only 4 and not 8 like calculated
* Since the orthogonal array performs pairwise test, we reduced the number of tests, so the number of test only including the top and left column (pairwise) = 2 \* 2 = 4



* We can display more rows if necessary

**Orthogonal Array Technique Approach:**

Follow the steps in order

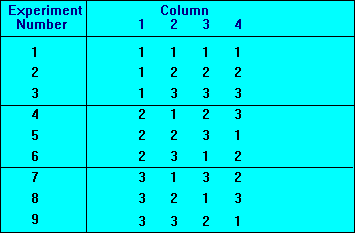
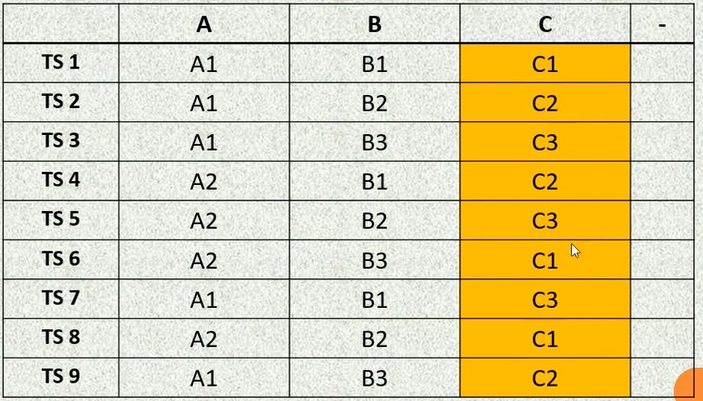
1. Identify the independent variables in the test, the independent variable will map to the column in the orthogonal array
2. For each variable, we need to identify the number of possible values (eg: the top section [independent variable] could be displayed or hidden)
3. Search for an OA with the smallest number of rows, we can do this using the [Genichi Taguchi](https://www.york.ac.uk/depts/maths/tables/taguchi_table.htm) Orthogonal arrays (Rows and columns and the cells filled are the number of experiments with the number of variables)
4. Searching for the smallest number of rows will give us the minimum number of tests we have to do
5. Place the variable names in the OA columns
6. Place the variable values in the OA cells
7. If there are empty OA cells, cycle values through them

**Example 2:**

3 Variables: A(2 values = A1 and A2), B(3 Values) and C(3 Values)

no. of combinations = 2 \*3 \*3 = 18

Using Taguchi’s orthogonal table we will result in [L9/3](https://www.york.ac.uk/depts/maths/tables/l9.gif)



### **5.3 Approach**

1. Analyze the given test inputs or requirements and list out the variables that need to be tested for interaction.
2. Determine the number of choices or values for each variable.
3. Locate an orthogonal array which has a column for each variable and values within the columns that correspond to the values for each variable.
4. Map the variables with their values onto the orthogonal array.
5. Each row in the table corresponds to a Test Condition or a unique Test Case.

### **5.4 Example from the slide 1**

* The Appearances tab of the Netscape Preferences dialog box has the following basic options:
* 3 toolbars (pictures, text, or both)
* 3 choices for launch on startup (browser, mail or news)
* 3 choices for startup page (blank home page, home page names a valid file, or home page name has a syntax error)
* 2 choices for underlining of links (don’t underline, or underline)
* 2 choices for expiration of links (never expire, expire after 30 days)
* 3 x 3 x 3 x 2 x 2 = 108 combinations!
* But from [Taguchi’s](https://www.york.ac.uk/depts/maths/tables/l16b.htm) orthogonal table, we only need 16 experiments

### **5.5 Example from the slide 2**

The borders and shading dialog box :

* 5 settings
* 5 styles (showing)
* 9 colors
* 9 widths
* 5 x 5 x 9 x 9 = 2025 combinations
* L81(94) array will suffice which is outside the Taguchi Table

### **5.5 Applicability, Advantages, Limitations**

**Applicability:**

* Orthogonal array technique is particularly useful for integration testing of software components (especially in OO systems where multiple subclasses can be substituted as the server for a client).
* It is also quite useful for testing combinations of configurable options (such as a web page that lets the user choose the font style, background color, and page layout).

**Advantages:**

* Provides uniformly distributed coverage of the test domain
* Concise test set with fewer test cases are created
* All pairwise combinations of test set are created
* Simpler to generate and less error prone than test sets created manually
* Reduces testing cycle time

**Limitations:**

* The limitation of this technique is that it does not guarantee the extensive coverage of the test domain.

### **5.6 Tools used**

**Unsecured tools**

<https://pairwise.teremokgames.com/>

**Secured tools (These are paid or proprietary)**

1. Hexawise:<https://hexawise.com/>
2. IBM CTD called Focus:<https://www-50.ibm.com/partnerworld/gsd/solutiondetails.do?solution=18011&expand=true&lc=en>

## **6. All Pairs**

### **6.1 Introduction:**

* Testing deals with validating the different values for all variables in the system.
* We generate test cases by pairing values of different variables.

### **6.2 Approach:**

1. List out the variables in the application to be tested and the various possible values each of the variables can hold
2. Combine or group the values where ever possible
3. Create the all pairs table by putting the variables in the top row and start by filling in the values for the variables in each column.
4. If a combination does not exist, then swap around with the values to see if the combination can be obtained.
5. Else add a new row
6. Each row in the table corresponds to a test case!