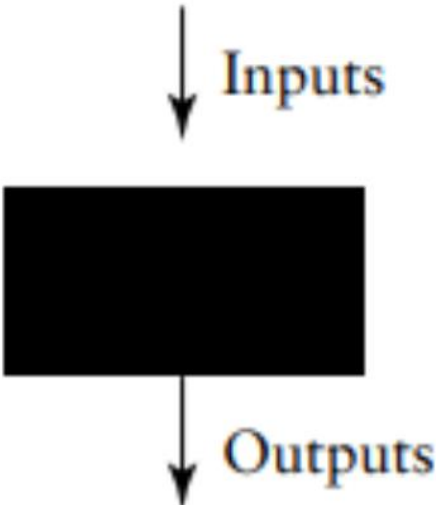



Black-Box Testing

Functional Testing

The two basic testing strategies.

| Test Strategy | Tester's View | Knowledge Sources | Methods |
|---------------|--|--|--|
| Black box |  | Requirements document Specifications Domain knowledge Defect analysis data | Equivalence class partitioning Boundary value analysis State transition testing Cause and effect graphing Error guessing |
| White box |  | High-level design Detailed design Control flow graphs Cyclomatic complexity | Statement testing Branch testing Path testing Data flow testing Mutation testing Loop testing |

Specification Based Techniques

Equivalence Partitioning

Boundary Value Analysis

Decision Tables

State Transition Diagrams

Orthogonal Arrays

Use Case Testing

Structure Based Techniques

Statement coverage

Decision Coverage

Path Coverage

Condition coverage

Experience Based Techniques

Exploratory Testing

Error Guessing

Real time Analogous Example

99% of the testing that QA team does comes under black box testing.

Infact, even if you are not a QA professional, you have done blackbox testing.

What is black box testing?

How to Black box testing?

Types

Techniques

- Buying Bulb , Sales man test and show the light
- Buying CAR/ Bike as a Consumer / Mechanic
- Patient Treatment for a Disease
- Performance of Student , verified by Parent

Equivalence Class Partitioning

For example, if you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousands of test cases for all 1000 valid input numbers plus other test cases for invalid data.

#1) One input data class with all valid inputs

#2) Input data class with all values below the lower limit

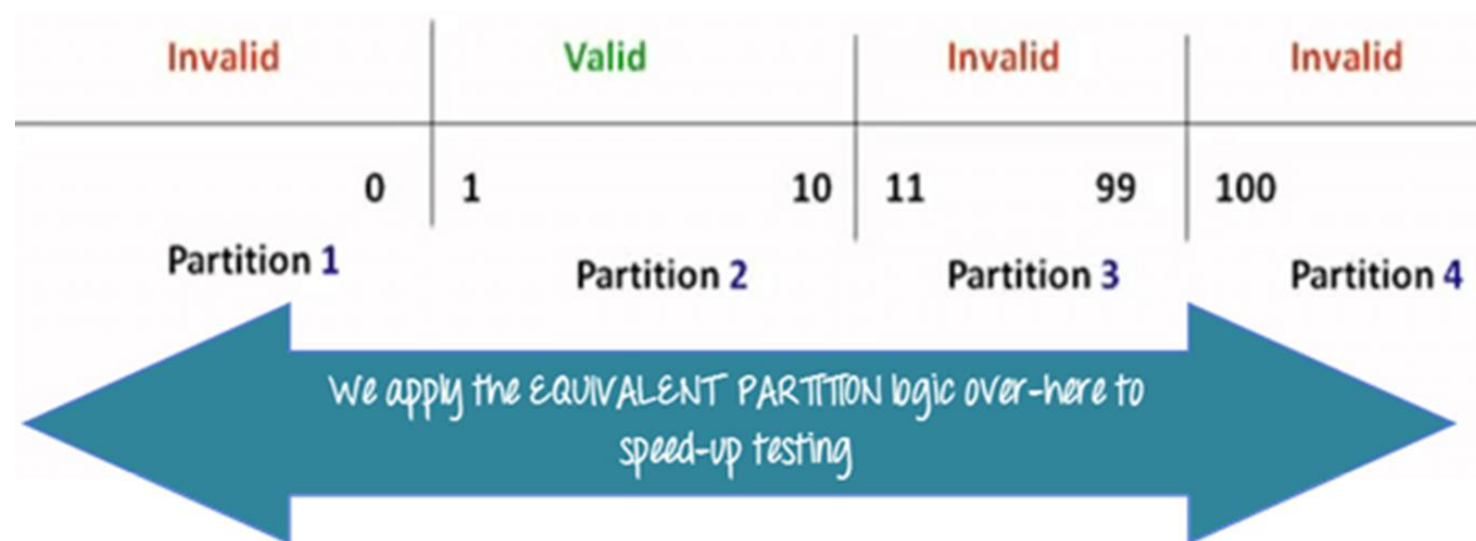
#3) Input data with any value greater than 1000 to represent the third invalid input class.

Equivalence Class Partitioning

Equivalence class partitioning results in a partitioning of the input domain of the software-under-test. The technique can also be used to partition the output domain, but this is not a common usage.

1. It eliminates the need for exhaustive testing, which is not feasible.
2. It guides a tester in selecting a subset of test inputs with a high probability of detecting a defect.
3. It allows a tester to cover a larger domain of inputs/outputs with a smaller subset selected from an equivalence class.

1. The tester must consider both valid and invalid equivalence classes. Invalid classes represent erroneous or unexpected inputs.
2. Equivalence classes may also be selected for output conditions.
3. The derivation of input or outputs equivalence classes is a heuristic process.
4. In some cases it is difficult for the tester to identify equivalence classes. The conditions/boundaries that help to define classes may be absent, or obscure, or there may seem to be a very large or very small number of equivalence classes for the problem domain.



Equivalence Partitioning

Equivalence partitioning is also known as “**Equivalence Class Partitioning**”. The whole range of input data for a given requirement are grouped into several sets or equivalence classes.

Equivalence classes are determined in such a way that all the input values in a input data range produce the same output when it is processed by the software.

| <u>Grade</u> | <u>Grade Points</u> | <u>Mark Range</u> |
|--------------|---------------------|-------------------|
| O | 10 | 91-100 |
| A+ | 9 | 81-90 |
| A | 8 | 71-80 |
| B+ | 7 | 61-70 |
| B | 6 | 50-60 |
| RA | 0 | 0 - 49 or <50 |

One more additional test case(For invalid data) is to test values which is greater than 100 and it should not be permitted.

Price

eCommerce Filters

☐

UNDER \$25

☐

\$25 TO \$50

☐

\$50 TO \$100☐☐

- a. Range of values
- b. Number of Values
- c. Ordered set

BLB—a value just below the lower bound

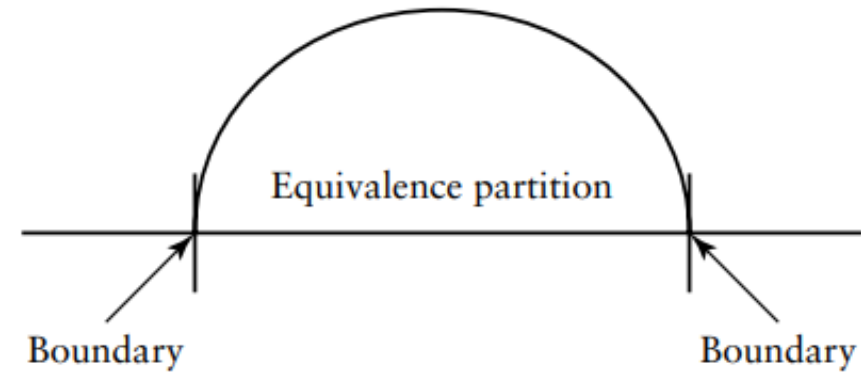
LB—the value on the lower boundary

ALB—a value just above the lower boundary

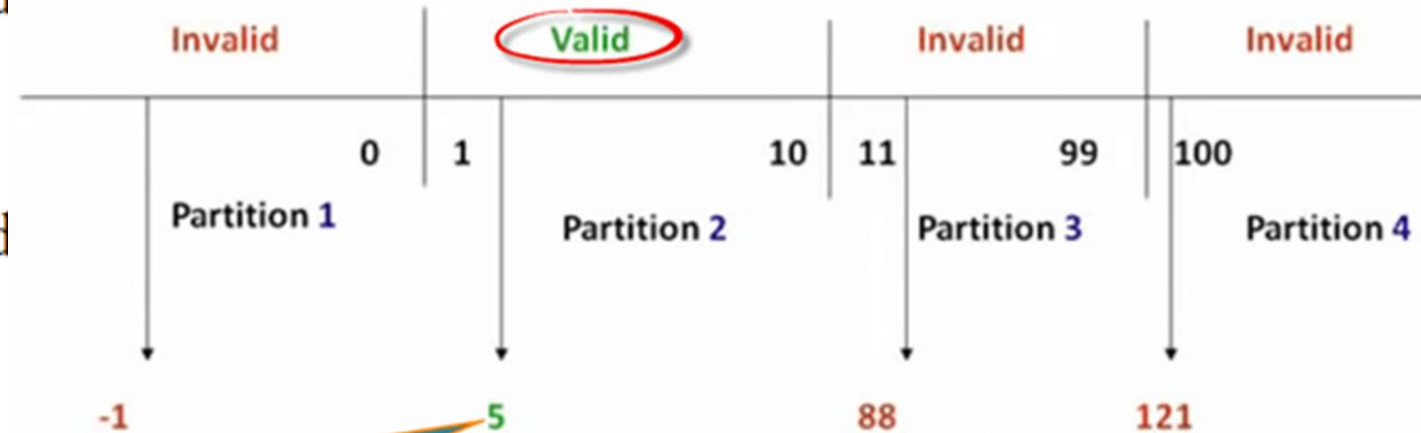
BUB—a value just below the upper bound

UB—the value on the upper bound

AUB—a value just above the upper bound



Boundaries of an equivalence partition.



If any one value from the set passes the test then the whole set of partition is considered pass or valid

```
Function square_root
  message (x:real)
    when  $x \geq 0.0$ 
      reply (y:real)
        where  $y \geq 0.0$  & approximately (y*y,x)
    otherwise reply exception imaginary_square_root
  end function
```

- EC1. The input variable x is real, valid.
- EC2. The input variable x is not real, invalid.
- EC3. The value of x is greater than 0.0, valid.
- EC4. The value of x is less than 0.0, invalid.

- EC1. Part name is alphanumeric, valid.
- EC2. Part name is not alphanumeric, invalid.

Then we treat condition 2, the range of allowed characters 3–15.

- EC3. The widget identifier has between 3 and 15 characters, valid.
- EC4. The widget identifier has less than 3 characters, invalid.
- EC5. The widget identifier has greater than 15 characters, invalid.

Finally we treat the “must be” case for the first two characters.

- EC6. The first 2 characters are letters, valid.
- EC7. The first 2 characters are not letters, invalid.

| Condition | Valid equivalence classes | Invalid equivalence classes |
|-----------|---------------------------|-----------------------------|
| 1 | EC1 | EC2 |
| 2 | EC3 | EC4, EC5 |
| 3 | EC6 | EC7 |

Module name: Insert_Widget
Module identifier: AP62-Mod4
Date: January 31, 2000
Tester: Michelle Jordan

| Test case identifier | Input values | Valid equivalence classes and bounds covered | Invalid equivalence classes and bounds covered |
|----------------------|------------------|--|--|
| 1 | abc1 | EC1, EC3(ALB) EC6 | |
| 2 | ab1 | EC1, EC3(LB), EC6 | |
| 3 | abcdef123456789 | EC1, EC3 (UB) EC6 | |
| 4 | abcde123456789 | EC1, EC3 (BUB) EC6 | |
| 5 | abc* | EC3(ALB), EC6 | EC2 |
| 6 | ab | EC1, EC6 | EC4(BLB) |
| 7 | abcdefg123456789 | EC1, EC6 | EC5(AUB) |
| 8 | a123 | EC1, EC3 (ALB) | EC7 |

Boundary Value Analysis

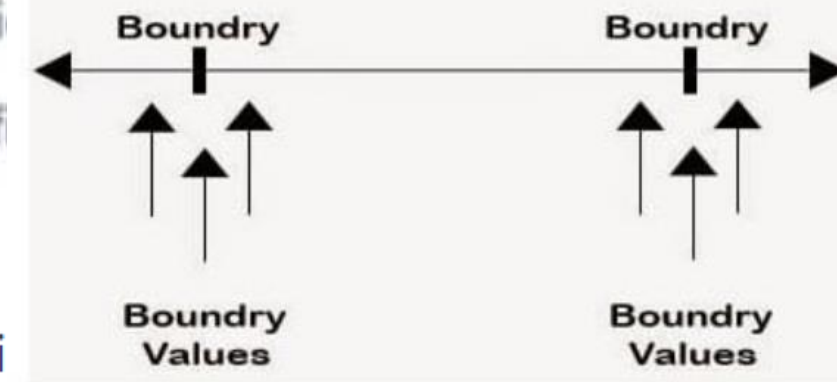
- In our earlier example instead of checking, one value for each partition you will check the values at the partitions like 0, 1, 10, 11 and so on.
- As you may observe, you test values at **both valid and invalid boundaries**.
- Boundary Value Analysis is also called **range checking**.
- Equivalence partitioning and boundary value analysis are closely related and can be used together at all levels of testing.

Boundary Value Analysis

This Boundary value analysis is used for developing test cases to test the boundaries which separates the continuous range of inputs. The boundaries are identified as part of this technique and then the test cases are written. For any scenario, we can write

- Lower boundary cases (using values just below the boundary specified)
- Upper Boundary cases (Using values just above the boundary specified)
- On boundary cases (Using the boundary values)

Boundary Value Analysis



#1) Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.

#2) Test data with values just below the extreme edges of input domains i.e. values 0 and 999.

#3) Test data with values just above the extreme edges of the input domain i.e. values 2 and 1001.

Input Box should accept the Number 1 to 10

Boundary Value Test Cases

| Test Scenario Description | Expected Outcome |
|---------------------------|--------------------------|
| Boundary Value = 0 | System should NOT accept |
| Boundary Value = 1 | System should accept |
| Boundary Value = 2 | System should accept |
| Boundary Value = 9 | System should accept |
| Boundary Value = 10 | System should accept |
| Boundary Value = 11 | System should NOT accept |

- Equivalence partitioning – break the input domain into different classes:
 - Class1: $n < 0$
 - Class2: $n > 0$ and $n!$ doesn't cause an overflow
 - Class3: $n > 0$ and $n!$ causes an overflow
- Boundary Value Analysis:
 - $n=0$ (between class1 and class2)

Enter Your Age (*only 18 to 60 Are allowed for this ride)

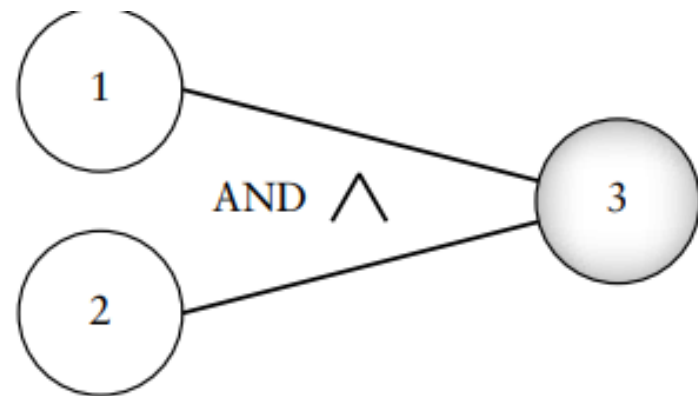
Boundary Value Analysis

| Invalid | Valid | Invalid |
|---------|------------------------|---------|
| (min-1) | min, min+1, max, max-1 | max+1 |
| 17 | 18, 19, 60, 59 | 61 |

Cause-and-Effect Graphing

Cause-and-effect

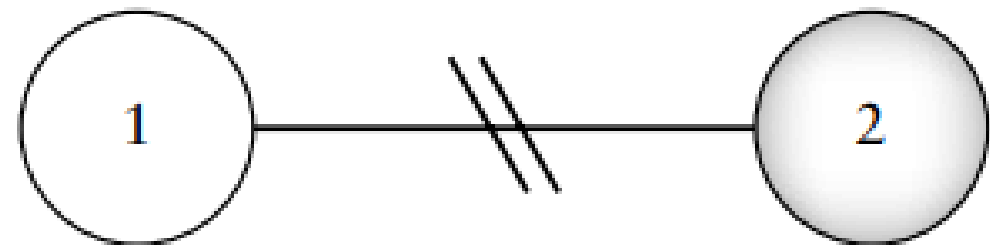
graphing is a technique that can be used to combine conditions and derive an effective set of test cases that may disclose inconsistencies in a specification.



Effect 3 occurs if both causes 1 and 2 are present.

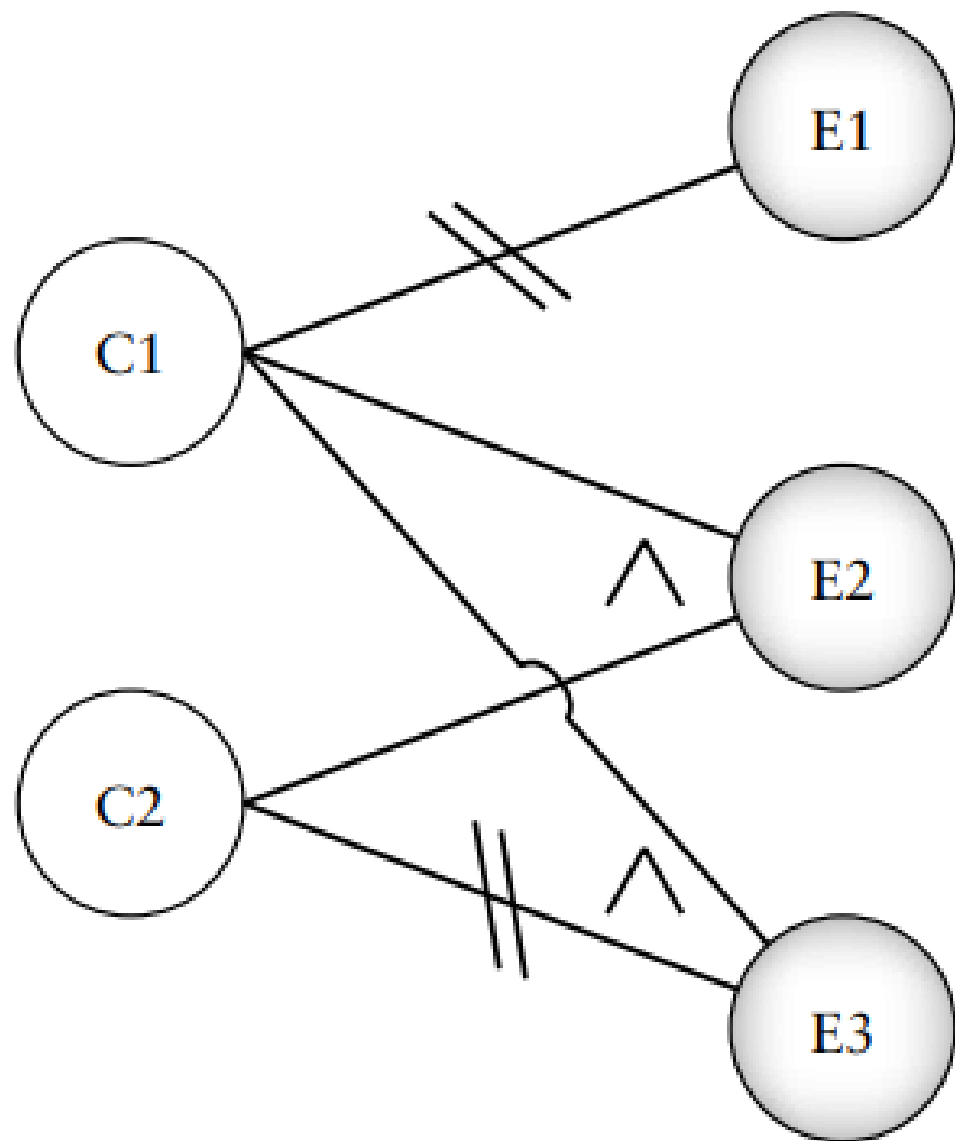


Effect 2 occurs if cause 1 occurs.



Effect 2 occurs if cause 1 does not occur.

Samples of cause-and-effect graph



Cause-and-effect graph for the character search example.

C1: Positive integer from 1 to 80

C2: Character to search for is in string

The output conditions, or effects are:

E1: Integer out of range

E2: Position of character in string

E3: Character not found

The rules or relationships can be described as follows:

If C1 and C2, then E2.

If C1 and not C2, then E3.

If not C1, then E1.

If the existing string is “abcde,” then possible tests are the following:

| Inputs | Length | Character to search for | Outputs |
|--------|--------|-------------------------|----------------------|
| T1 | 5 | c | 3 |
| T2 | 5 | w | Not found |
| T3 | 90 | | Integer out of range |

Decision table technique in Black box testing

Decision table technique is one of the widely used case design techniques for black box testing. This is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form.

| Email (condition1) | T | T | F | F |
|--------------------------|--------------|--------------------|-----------------|-----------------|
| Password (condition2) | T | F | T | F |
| Expected Result (Action) | Account Page | Incorrect password | Incorrect email | Incorrect email |

Number of possible conditions = $2^{\text{Number of Values of the second condition}}$

Number of possible conditions = $2^2 = 4$

Decision Table / Cause-Effect

| Decision Table | Rule 1 | Rule 2 | Rule 3 | Rule 4 |
|------------------------|--------|--------|--------|--------|
| Conditions | | | | |
| C1 - Male | F | F | T | T |
| C2 - Senior Citizen | F | T | F | T |
| Actions | | | | |
| A1 - Interest Rate 10% | | | | X |
| A2 - Interest Rate 9% | X | X | X | |

Decision table is one of the testing design techniques which will help us test the system's behavior with various set of inputs.

This technique can also be called as Cause and Effect table, as it captures the driving factor (cause, input data) and the captures the effect (output) in the tabular format.

Naukri. <https://www.naukri.com/>

- Maximum file size is 2MB
- List of supported file formats
 - Doc
 - PDF
 - docx
 - RTF

| Conditions | Case 1 | Case 2 | Case 3 | Case 4 | case 5 | case 6 | case 7 | case 8 |
|--------------|-----------|----------|----------|----------|-----------------------|-----------------------|-----------------------|----------|
| Input format | doc | docx | PDF | RTF | doc | docx | PDF | RTF |
| Input Size | size<2 MB | size<2MB | size<2MB | size<2MB | size=2 MB | size=2MB | size=2MB | size=2MB |
| Output | Valid | Valid | Valid | Valid | Valid | Valid | Valid | Valid |
| | | | | | | | | |
| | | | | | | | | |
| Conditions | Case 9 | Case 10 | Case 11 | Case 12 | case 13 | case 14 | case 15 | |
| Input format | docx | PDF | RTF | doc | any other file format | any other file format | any other file format | |
| Input Size | size>2MB | size>2MB | size>2MB | size>2MB | size<2MB | size>2MB | size=2MB | |
| Output | Invalid | Invalid | Invalid | Invalid | Invalid | Invalid | Invalid | |

1. The entire team can understand the nature of the application by simply looking at the table.
2. Provides a better test coverage for any given functionality.
3. Any type of complex requirements can easily be turned into decision tables and one thing to remember when the number of input conditions is higher the table will become large and will be difficult to manage. In that case we can rely on equivalence partitioning or Boundary value analysis.

| | T1 | T2 | T3 |
|----|----|----|----|
| C1 | 1 | 1 | 0 |
| C2 | 1 | 0 | — |
| E1 | 0 | 0 | 1 |
| E2 | 1 | 0 | 0 |
| E3 | 0 | 1 | 0 |

Decision table for character search

C1: Positive integer from 1 to 80

C2: Character to search for is in string

The output conditions, or effects are:

E1: Integer out of range

E2: Position of character in string

E3: Character not found

The rules or relationships can be described as follows:

- If C1 and C2, then E2.
- If C1 and not C2, then E3.
- If not C1, then E1.

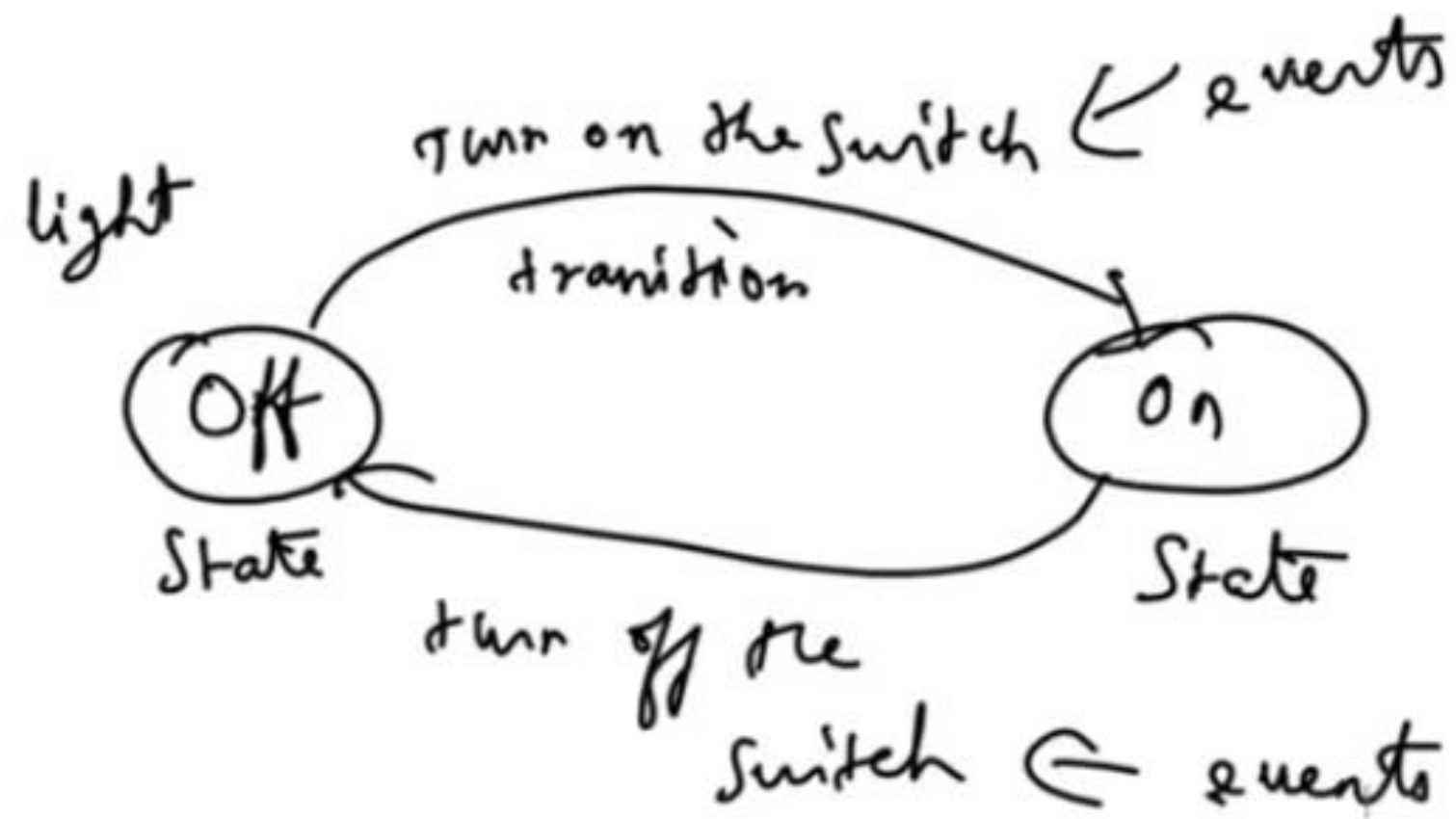
State Transition Testing

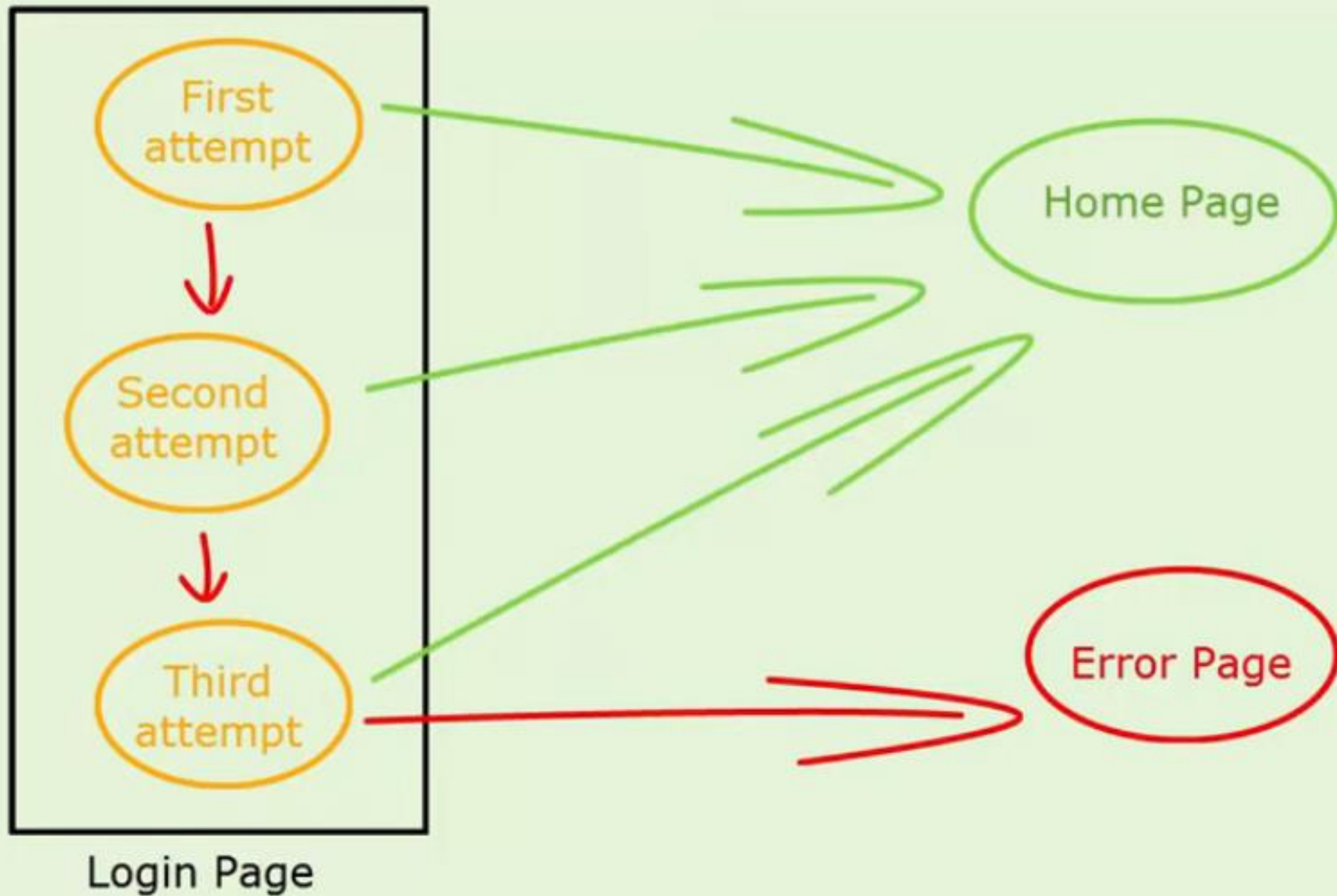
A state is an internal configuration of a system or component. It is defined in terms of the values assumed at a particular time for the variables that characterize the system or component.

A finite-state machine is an abstract machine that can be represented by a state graph having a finite number of states and a finite number of transitions between states.

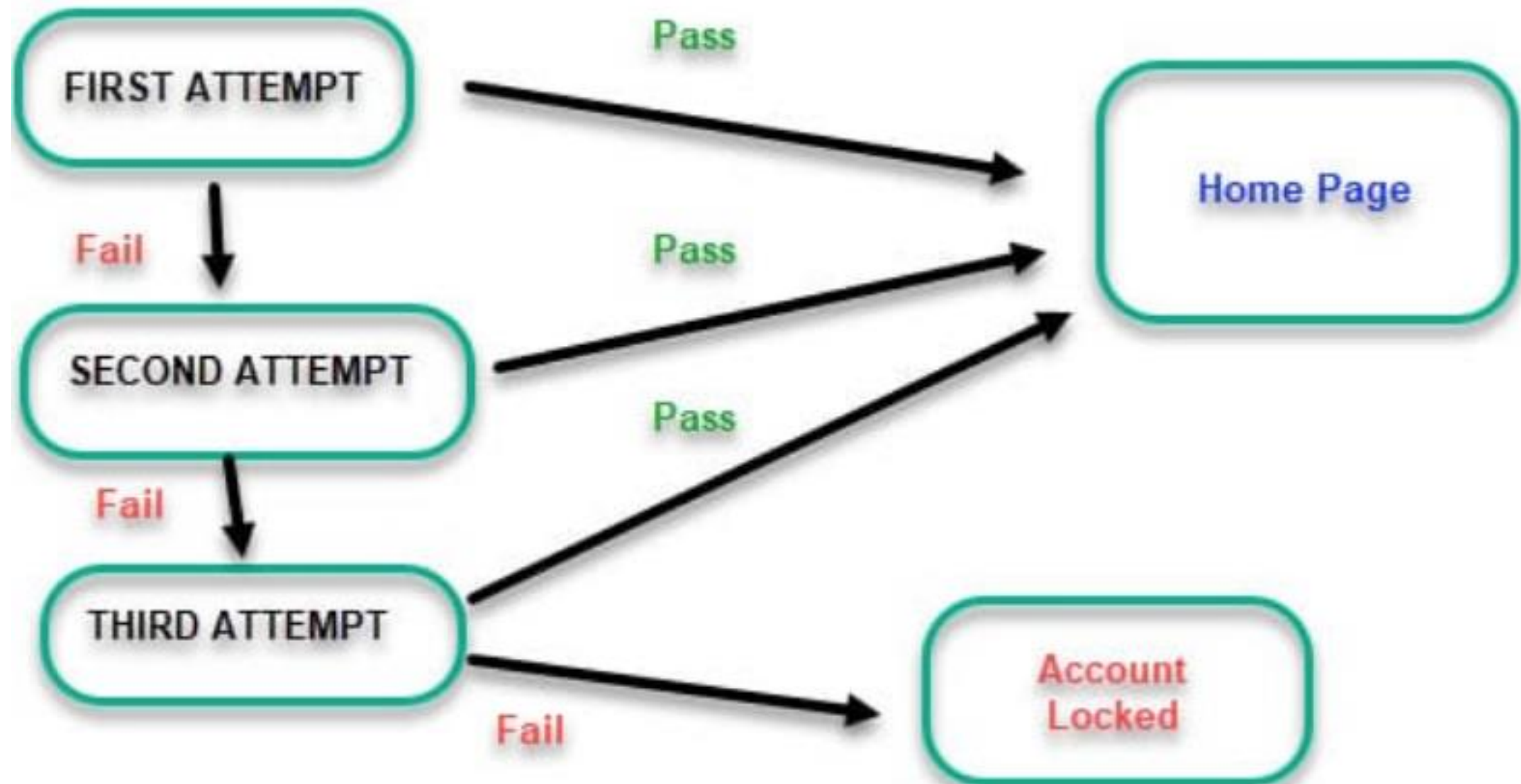
Test design is not just giving valid and invalid inputs and calculate the number of test cases to be written. Sometimes, the test structures are not simple to demonstrate them using input and output combination. In such cases, we will be interested in knowing the **state** of an application for any given input and how the state is getting **transitioned** from one to another.

- 1. State
- 2. Transitions
- 3. Events
- 4. Actions

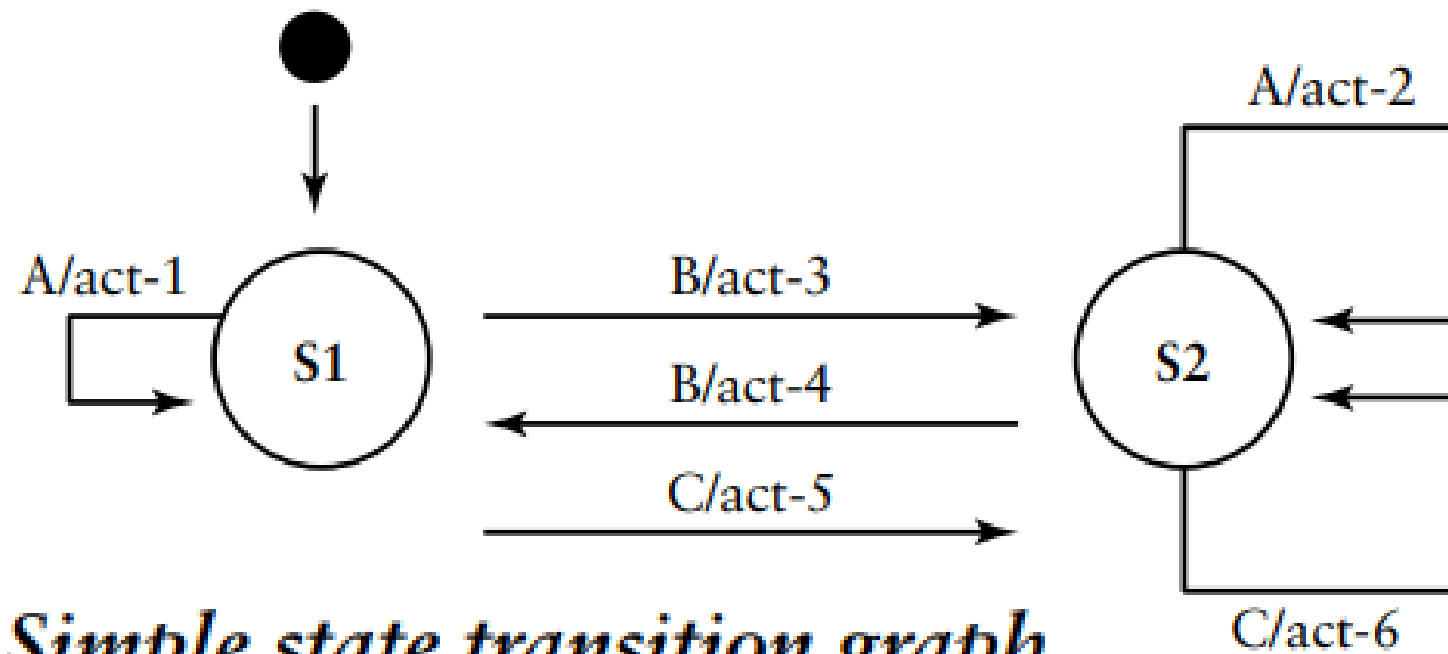




STATE TRANSITION TESTING



| STATE | LOGIN | VALID | INVALID |
|-------|----------------|-------|---------|
| S1 | First attempt | | S2 |
| S2 | Second attempt | | S3 |
| S3 | Third attempt | | S5 |
| S4 | Home page | | |
| S5 | Error message | | |



Simple state transition graph.

Input A in S1
 Input A in S2
 Input B in S1
 Input B in S2
 Input C in S1
 Input C in S2

| | S1 | S2 |
|---------|------------|------------|
| Inputs | | |
| Input A | S1 (act-1) | S2 (act-2) |
| Input B | S2 (act-3) | S1 (act-4) |
| Input C | S2 (act-5) | S2 (act-6) |

WHAT IS ORTHOGONAL ARRAY

- 2-Dimensional array of numbers
- Any two columns give pair wise combinations

| | Column1 | Column2 | Column3 |
|----------|---------|---------|---------|
| Expr. #1 | 1 | 1 | 1 |
| Expr. #2 | 1 | 2 | 2 |
| Expr. #3 | 2 | 1 | 2 |
| Expr. #4 | 2 | 2 | 1 |

| A | B | C |
|---|---|---|
|---|---|---|

| | | |
|---|---|---|
| 1 | 1 | 1 |
|---|---|---|

| | | |
|---|---|---|
| 2 | 2 | 2 |
|---|---|---|

$$2^3 = 8$$

| A | B | C |
|---|---|---|
|---|---|---|

| | | |
|---|---|---|
| 1 | 1 | 1 |
|---|---|---|

| | | |
|---|---|---|
| 1 | 1 | 2 |
|---|---|---|

| | | |
|---|---|---|
| 1 | 2 | 1 |
|---|---|---|

| | | |
|---|---|---|
| 2 | 1 | 1 |
|---|---|---|

| | | |
|---|---|---|
| 1 | 2 | 2 |
|---|---|---|

| | | |
|---|---|---|
| 2 | 2 | 1 |
|---|---|---|

| | | |
|---|---|---|
| 2 | 1 | 2 |
|---|---|---|

TAGUCHI ORTHOGONAL ARRAY SELECTION TABLE

| | | Number of parameters | | | | | | | | |
|------------------|---|----------------------|-----|-----|-----|-----|-----|-----|-----|-----|
| | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Number of levels | 2 | L4 | L4 | L8 | L8 | L8 | L8 | L12 | L12 | L12 |
| | 3 | L9 | L9 | L9 | L18 | L18 | L18 | L18 | L27 | L27 |
| | 4 | L16 | L16 | L16 | L16 | L32 | L32 | L32 | L32 | L32 |
| | 5 | L25 | L25 | L25 | L25 | L25 | L50 | L50 | L50 | L50 |

TAGUCHI'S L9 ORTHOGONAL ARRAY TABLE

| Experimental Run | Levels of the Input Process Parameters | | |
|------------------|--|---------|---------|
| | Input 1 | Input 2 | Input 3 |
| EX01 | 1 | 1 | 1 |
| EX02 | 1 | 2 | 2 |
| EX03 | 1 | 3 | 3 |
| EX04 | 2 | 1 | 2 |
| EX05 | 2 | 2 | 3 |
| EX06 | 2 | 3 | 1 |
| EX07 | 3 | 1 | 3 |
| EX08 | 3 | 2 | 1 |
| EX09 | 3 | 3 | 2 |

EXAMPLE

Considering an Optimization Study for the Turning Process Parameters,

| Input Variables | Levels | | |
|-------------------------|--------|--------|--------|
| | I | II | III |
| ✓ Spindle speed (m/min) | ✓ 500 | ✓ 1000 | ✓ 1500 |
| ✓ Feed rate (mm/rev) | ✓ 0.2 | ✓ 0.4 | 0.6 |
| ✓ Depth of cut (mm) | 0.5 | 1.0 | 1.5 |

| Experimental Run | Levels of the Input Process Parameters | | |
|------------------|--|--------------------|-------------------|
| | Spindle speed (m/min) | Feed rate (mm/rev) | Depth of cut (mm) |
| EX01 | 500 | 0.2 | 0.5 |
| EX02 | 500 | 0.4 | 1.0 |
| EX03 | 500 | 0.6 | 1.5 |
| EX04 | 1000 | 0.2 | 1.0 |
| EX05 | 1000 | 0.4 | 1.5 |
| EX06 | 1000 | 0.6 | 0.5 |
| EX07 | 1500 | 0.2 | 1.5 |
| EX08 | 1500 | 0.4 | 0.5 |
| EX09 | 1500 | 0.6 | 1.0 |

3 variables – A (2 values), B (3 values) and C (3 values)

No. of combinations = $2 * 3 * 3 = 18$

| | A | B | C | - |
|------|----|---|---|---|
| TS 1 | A1 | | | |
| TS 2 | A1 | | | |
| TS 3 | A1 | | | |
| TS 4 | A2 | | | |
| TS 5 | A2 | | | |
| TS 6 | A2 | | | |
| TS 7 | A1 | | | |
| TS 8 | A2 | | | |
| TS 9 | A1 | | | |

Orthogonal Array Testing

1. Orthogonal approach guarantees the pairwise coverage of all variables.

How OAT's is represented

L_{Runs}(Levels^{Factors})

- Runs (N) – Number of rows in the array, which translates into a number of test cases that will be generated.
- Factors (K) – Number of columns in the array, which translates into a maximum number of variables that can be handled.
- Levels (V) – Maximum number of values that can be taken on any single factor.

Example 1

A Web page has three distinct sections (Top, Middle, Bottom) that can be individually shown or hidden from a user

- No of Factors = 3 (Top, Middle, Bottom)
- No of Levels (Visibility) = 2 (Hidden or Shown)
- Array Type = L4(23)

(4 is the number of runs arrived after creating the OAT array)

If we go for Conventional testing technique, we need test cases like 2 X 3 = 6 Test Cases

| Test Cases | Scenarios | Values to be tested |
|------------|-----------|---------------------|
| Test #1 | HIDDEN | Top |
| Test #2 | SHOWN | Top |
| Test #3 | HIDDEN | Bottom |
| Test #4 | SHOWN | Bottom |
| Test #5 | HIDDEN | Middle |
| Test #6 | SHOWN | Middle |

If we go for OAT Testing we need 4 Test cases as shown below:

| Test Cases | TOP | Middle | Bottom |
|------------|---------|---------|---------|
| Test #1 | Hidden | Hidden | Hidden |
| Test #2 | Hidden | Visible | Visible |
| Test #3 | Visible | Hidden | Visible |
| Test #4 | Visible | Visible | Hidden |

A microprocessor's functionality has to be tested:

1. Temperature: 100C, 150C and 200C.
2. Pressure : 2 psi, 5psi and 8psi
3. Doping Amount : 4%, 6% and 8%
4. Deposition Rate : 0.1mg/s , 0.2 mg/s and 0.3mg/s

By using the Conventional method we need = 81 test cases to cover all the inputs. |

OATS method:

No. of factors = 4 (temperature, pressure, doping amount and Deposition rate)

Levels = 3 levels per factor (temperature has 3 levels-100C, 150C, and 200C and likewise other factors too have levels)

1. Columns with the No. of factors

| Test case # | Temperature | Pressure | Doping amount | Deposition rate |
|-------------|-------------|----------|---------------|-----------------|
| | | | | |

2. Enter the number of rows is equal to levels per factor. i.e temperature has 3 levels. Hence, insert 3 rows for each level for temperature,

| Test case # | Temperature | Pressure | Doping amount | Deposition rate |
|-------------|-------------|----------|---------------|-----------------|
| 1 | 100C | | | |
| 2 | 100C | | | |
| 3 | 100C | | | |
| 4 | 150C | | | |
| 5 | 150C | | | |
| 6 | 150C | | | |
| 7 | 200C | | | |
| 8 | 200C | | | |
| 9 | 200C | | | |

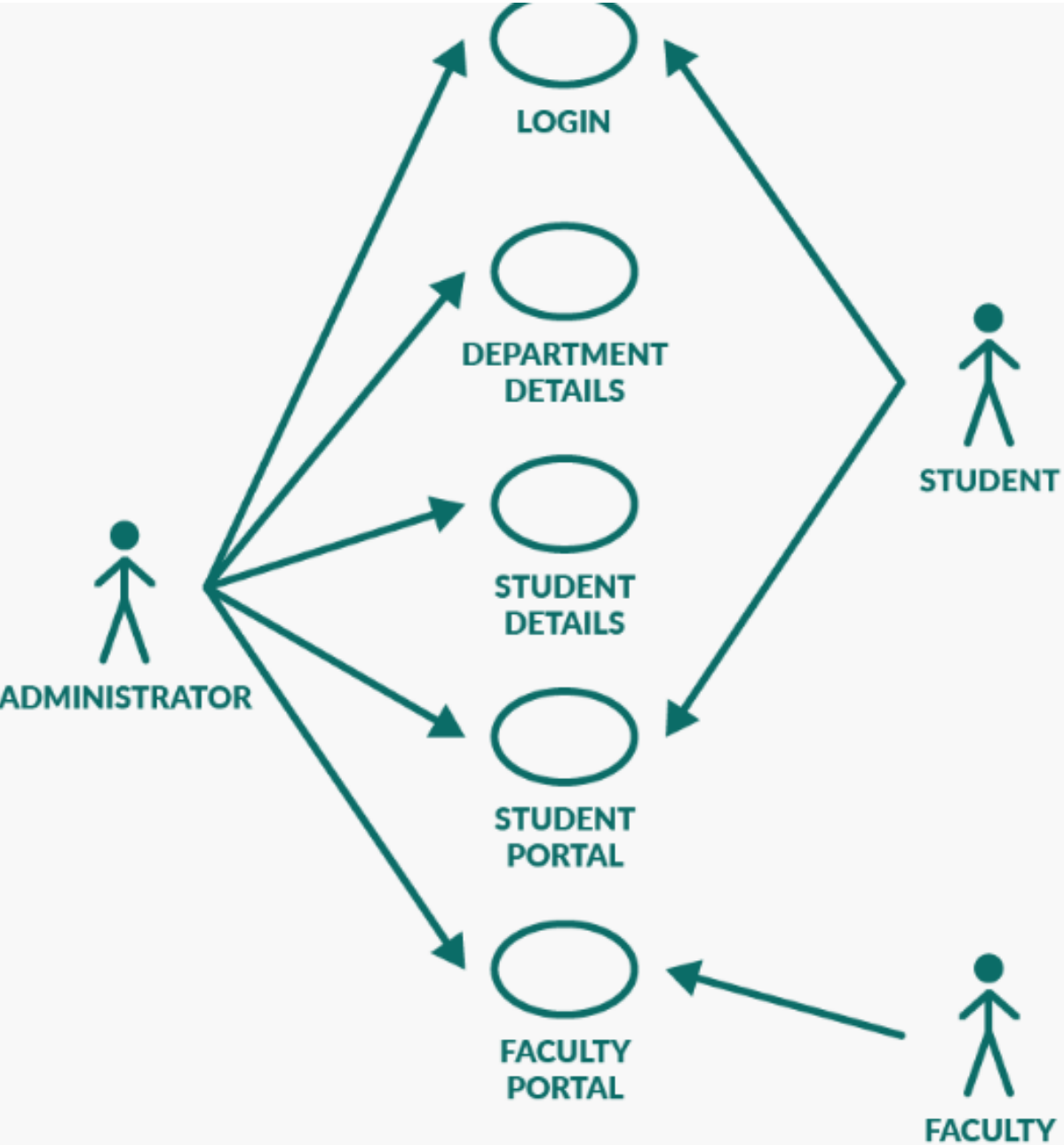
3. Now split up the pressure, doping amount and the deposition rates in the columns.

For eg: Enter 2 psi across temperatures 100C,150C and 200C likewise enter doping amount 4% for 100C,150C and 200C and so on.

| Test case # | Temperature | Pressure | Doping amount | Deposition rate |
|-------------|-------------|----------|---------------|-----------------|
| 1 | 100C | 2 psi | 4% | 0.1 mg/s |
| 2 | 100C | 5 psi | 6% | 0.2 mg/s |
| 3 | 100C | 8 psi | 8% | 0.3 mg/s |
| 4 | 150C | 2 psi | 4% | 0.1 mg/s |
| 5 | 150C | 5 psi | 6% | 0.2 mg/s |
| 6 | 150C | 8 psi | 8% | 0.3 mg/s |
| 7 | 200C | 2 psi | 4% | 0.1 mg/s |
| 8 | 200C | 5 psi | 6% | 0.2 mg/s |
| 9 | 200C | 8 psi | 8% | 0.3 mg/s |

Use case testing is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish.

| Main Success Scenario A: Actor S: System | Step | Description |
|---|-------------|--|
| | 1 | A: Inserts card |
| | 2 | S: Validates card and asks for PIN |
| | 3 | A: Enters PIN |
| | 4 | S: Validates PIN |
| | 5 | S: Allows access to account |
| Extensions | 2a | Card not valid S: Display message and reject card |
| | 4a | PIN not valid S: Display message and ask for re-try (twice) |
| | 4b | PIN invalid 3 times S: Eat card and exit |



| #Use Case | Actor(s) | Description |
|---------------------|---------------------------------|--|
| 1Login | Administrator, Faculty, Student | Login Change Password Forgot Password Add Department Edit Department Details Delete Department View Department Details Add New Student Edit Existing Student Delete Existing Student View Existing Student Add Student Details Add/Drop Courses View Progress/History |
| 2Department Details | Administrator | Generate Report Program-wise Generate Report Semester-wise |
| 3Student Details | Administrators | |
| 4Student Portal | Administrator, Student | |
| 5Faculty Portal | Administrator, Faculty | |

Best Black Box Testing Tools

1. [Squish by froglogic](#) — Best for graphical user interface (GUI) black box testing
2. [Gremlin](#) — Best for black box testing microservices and container-based applications
3. [AutoHotkey](#) — Best Windows macro-creation automation tool for keyboard and mouse functions
4. [IBM Rational Functional Tester \(RFT\)](#) — Best for creating black box test scripts with a test recorder
5. [Selendroid](#) — Best automated black box testing framework for native, web, and hybrid Android apps

1. [Selenium](#).

2. [Appium](#).

3. [Applitools](#).

4. [HP QTP](#).

5. [Microsoft Coded UI](#).

Q #1) One of the fields on a form contains a text box that accepts numeric values in the range of 18 to 25. Identify the invalid Equivalence class.

- a) 17
- b) 19
- c) 24
- d) 21

Class I: values < 18 \Rightarrow invalid class

Class II: 18 to 25 \Rightarrow valid class 17 falls under an invalid class. 19, 24 and 21 fall under valid

Class III: values > 25 \Rightarrow invalid class

Q #2) In an Examination, a candidate has to score a minimum of 24 marks in order to clear the exam. The maximum that he can score is 40 marks. Identify Valid Equivalence values if the student clears the exam.

- a) 22,23,26
- b) 21,39,40
- c) 29,30,31
- d) 0,15,22

Q #3) One of the fields on a form contains a text box that accepts alphanumeric values. Identify the Valid Equivalence class.

- a) BOOK
- b) Book
- c) Boo01k
- d) Book

Q #6) A program validates numeric fields as follows: values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following covers the MOST boundary values?

- a. 9,10,11,22
- b. 9,10,21,22
- c. 10,11,21,22
- d. 10,11,20,21

<https://www.softwaretestinghelp.com/istqb-exam-questions-equivalence-partitioning-boundary-value-analysis/>

In a system designed to work out the taxes to be paid:

An employee has £4000 of salary tax-free.

The next £1500 is taxed at 10%.

The next £28000 after that is taxed at 22%.

Any further amount is taxed at 40%.

To the nearest whole pound, which of these groups of numbers fall into three DIFFERENT equivalence classes?

a) £4000; £5000; £5500

b) £32001; £34000; £36500

c) £28000; £28001; £32001

d) £4000; £4200; £5600

The classes will be as follows:

Class I : 0 to £4000 => no tax

Class II : £4001 to £5500 => 10 % tax

Class III : £5501 to £33500 => 22 % tax

Class IV : £33501 and above => 40 % tax

In a system designed to work out the taxes to be paid:

An employee has £4000 of salary tax-free.

The next £1500 is taxed at 10%.

The next £28000 after that is taxed at 22%.

Any further amount is taxed at 40%.

To the nearest whole pound, which of these is a valid Boundary Value Analysis test case?

a) £28000

b) £33501

c) £32001

d) £1500

The classes will be as follows:

Class I : 0 to £4000 => no tax

Class II : £4001 to £5500 => 10 % tax

Class III : £5501 to £33500 => 22 % tax

Class IV : £33501 and above => 40 % tax

Advantages

- The tester does not need to have a technical background. It is important to test by being in the user's shoes and think from the user's point of view.
- Testing can start once the development of the project/application is done. Both the testers and developers work independently without interfering in each other's space.
- It is more effective for large and complex applications.
- Defects and inconsistencies can be identified in the early stages of testing.

Disadvantages

- Without any technical or programming knowledge, there are chances of ignoring possible conditions of the scenario to be tested.
- In a stipulated time there is a possibility of testing less and skipping all possible inputs and their output testing.
- Complete Test Coverage is not possible for large and complex projects.

1 Define the equivalence classes and develop a set of test cases to cover them for the following module description. The module accepts a color for a part. Color choices are {RED, BLUE, YELLOW, VIOLET}.

2 Define the equivalence classes and boundary values and develop a set of test cases to cover them for the following module description: The module is part of a public TV membership system. The module allows entry of a contribution from \$0.01 to \$99,999.99. It also enters a member status for the contributor that can be: regular, student/retiree, or studio club.

3 Develop black box test cases using equivalence class partitioning and boundary value analysis to test a module that is software component of an automated teller system. The module reads in the amount the user wishes to withdraw from his/her account. The amount must be a multiple of \$5.00 and be less than or equal to \$200.00. Be sure to list any assumptions you make and label equivalence classes and boundary values that you use.

Design a set of black box–based test cases using the coin problem specification as shown in Chapter 3. Use error guessing, random testing, equivalence class partitioning, and boundary value analysis to develop your test cases. List any assumptions you make. For each test case generated by equivalence class partitioning, specify the equivalence classes covered, input values, expected outputs, and test case identifiers. Show in tabular form that you have covered all the classes and boundaries. For the other black box testing methods show the test case identifiers, inputs, and expected outputs also in a tabular format. Implement the coin problem as shown in the specification in the language of your choice. Run the tests using your test cases, and record the actual outputs and the defects you found. Start with the original uncorrected version of the program for each black box technique you use. Save a copy of the original version for future use. Compare the methods with respect to their effectiveness in revealing defects. Were there any types of defects that were not detected by these methods?

Draw a state transition diagram for a simple stack machine. Assume the stack holds n data items where n is a small positive number. It has operations “push” and “pop” that cause the stack pointer to increment or decrement, respectively. The stack can enter states such as “full” if it contains n items and, “empty” if it contains no items. Popping an item from the empty stack, or pushing an item on the full stack cause a transition to an error state. Based on your state transition diagram, develop a set of black box test cases that cover the key state transitions. Be sure to describe the exact sequence of inputs, as well as the expected sequence of state changes and actions.

Suppose a program allowed a user to search for part name in a specific group of part records. The user inputs the record number that is believed to hold the part, and the part name to search for. The program will inform the user if the record number is within the legal range of allowed record numbers (1–1000). If it is not, then an error message will be issued—“record number is out of range.” If the record number is within the range, and the part is found, the program will return “part found,” else it will report “part not found.” Identify the input and output conditions, i.e., causes and effects. Draw a “cause-and-effect” graph and a decision table for this specification. From the table generate a set of test inputs and expected outputs.

Suppose you were developing a simple assembler whose syntax can be described as follows :

<statement> :: = <label field> <op code> <address>

<label field> :: = “none” | <identifier> :

<op code> :: = MOVE | JUMP

<address> :: = <identifier> | <unsigned integer>

A stream of tokens is input to the assembler. The possible states for such an assembler are:

S1, prelabel; S2, label; S3, valid op code; S4, valid address; S5, valid numeric address. Start, Error, and Done. A table that describes the inputs and actions for the assembler is as follows:

| Inputs | Actions |
|-----------------------|--|
| no more tokens | A1: Put the label in the symbol table. |
| identifer | A2: Look up the op code and store its binary value in op code field. |
| MOVE, JUMP | A3: Look up symbol in symbol table and store its value in address field. |
| colon | A4: Convert number to binary, and store that value in address field. |
| integer | A5: Place instruction in the object module, and print a line in the listing. A6: Print error message and put all zeroes in the instruction. |

Using this information and any assumptions you need to make, develop a state transition diagram for the assembler. From the state transition diagram develop a set of test cases that will cover all of the state transitions. Be sure to describe the exact sequence of inputs as well as the expected sequence of state changes and actions.

ATM - Automated Teller Machine

Withdrawal of money from ATM. 'User A' wants to withdraw 30,000 from the ATM. Imagine he could take 10,000 per transaction and total balance available in the account is 25,000.

In the first two attempts, he could withdraw money. Whereas in the third attempt, the ATM shows a message as "Insufficient balance, contact Bank". Same Action but due to change in the state, he couldn't withdraw the money in the third transaction.

Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on boundary-value analysis, execute the test cases and discuss the results.

| Case Id | Description | Input Data | | | Expected Output | Actual Output |
|---------|--|------------|---|---|---|---------------|
| | | a | b | c | | |
| 1 | Enter the min value for a , b and c | 1 | 1 | 1 | Should display the message Equilateral triangle | |
| 2 | Enter the min value for 2 items and min +1 for any one item1 | 1 | 1 | 2 | Message should be displayed can't form a Triangle | |
| 3 | Enter the min value for 2 items and min +1 for any one item1 | 1 | 2 | 1 | Message should be displayed can't form a triangle | |
| 4 | Enter the min value for 2 items and min +1 for any one item1 | 2 | 1 | 1 | Message should be displayed can't form a triangle | |
| 5 | Enter the normal value for 2 items and 1 item is min value | 5 | 5 | 1 | Should display the message Isosceles triangle | |
| 6 | Enter the normal value for 2 items and 1 item is min value | 5 | 1 | 5 | Should display the message Isosceles triangle | |

Example: The Triangle Problem ($1 \leq \text{side} \leq 100$)

O_1 : Equilateral Δ : 50, 50, 50

O_2 : Isosceles Δ : 99, 50, 50

O_3 : Scalene Δ : 99, 100, 50

O_4 : Not a Δ : 50, 50, 100

I_1 : $x < 1$

I_2 : $x > 100$

I_3 : $1 \leq x \leq 100$

I_4 : $y < 1$

I_5 : $y > 100$

I_6 : $1 \leq y \leq 100$

I_7 : $z < 1$

I_8 : $z > 100$

I_9 : $1 \leq z \leq 100$

I_{10} : $x = y = z$

I_{11} : $x = y, x \neq z$

I_{12} : $x = z, x \neq y$

I_{13} : $y = z, x \neq y$

I_{14} : $x \neq y, y \neq z, x \neq z$

I_{15} : $x = y + z$

I_{16} : $x > y + z$

I_{17} : $y = x + z$

I_{18} : $y > x + z$

I_{19} : $z = x + y$

I_{20} : $z > x + y$

| Test Case | x | y | z | Output |
|-----------|-----|-----|-----|----------------|
| 1 | 0 | 50 | 50 | y/p Invalid |
| 2 | 101 | 50 | 50 | y/p Invalid |
| 3 | 50 | 50 | 50 | Equilateral |
| 4 | 50 | 0 | 50 | Invalid y/p |
| 5 | 50 | 101 | 50 | Invalid y/p |
| 6 | 50 | 50 | 50 | Equilateral |
| 7 | 50 | 50 | 0 | Invalid y/p |
| 8 | 50 | 50 | 101 | Invalid y/p |
| 9 | 50 | 50 | 50 | Equilateral |
| 10 | 60 | 60 | 60 | Equilateral |
| 11 | 50 | 50 | 60 | Isosceles |
| 12 | 50 | 60 | 50 | Isosceles |
| 13 | 60 | 50 | 50 | Isosceles |
| 14 | 100 | 99 | 50 | Scalene |
| 15 | 100 | 50 | 50 | Not a Δ |
| 16 | 100 | 50 | 25 | Not a Δ |
| 17 | 50 | 100 | 50 | Not a Δ |
| 18 | 50 | 100 | 25 | Not a Δ |
| 19 | 50 | 50 | 100 | Not a Δ |
| 20 | 95 | 50 | 100 | Not a Δ |

