OFTWARE TESTING & MAINTENANCE

Chapter-8

TESTING CONCEPTS

Failure: it is an visible incorrect behavior or state of a given system. In this case, the system displays a behavior that is different to its specifications/requirements.

Fault: (commonly named “bug/defect”) it is a defect in a system. A failure may be caused by the presence of one or more faults on a given system.

Error: it is the developer mistake that produce a fault. Often, it has been caused by human activities such as the typing errors.

CONT......

Test Case: input sequence and associated expected output

- **Test Suite:** a set of test cases for a system

Testing: testing is the process of executing a program with the intent of finding *errors*

- Testing cannot guarantee the absence of faults,
- Strategies for defining test suites,
- *Formal methods (e.g., model checking)* can be used to statically verify software properties, this is not testing.

Debugging: finding and fixing faults in the code

TESTING: TWO MAIN QUESTIONS …

At which level conducting the testing?

- System
- Unit
- Integration

How to choose inputs?

- Considering the program as a **black box**
  - **using the specifications/use cases/requirements**
- Considering the program as a **white box**
  - **using the structure**

A randomly selected set of inputs is often not sufficient …

<u>Unit testing</u> – this is testing of a single function, procedure, class. It is usually done by the developer, not by a separate testing team.

<u>Integration testing</u> – this checks that units tested in separation work properly when put together. It often requires drivers to simulate the missing components while integrating the system.

<u>System testing</u> – here the goal is to ensure that the whole system works properly.

-System & validation Testing:

-Verification & Validation:

Verification:- verify product meet to req. or not

-items :- Code ,Test case.

-Activities :- Reviews,inspections.

Validation:- (At the end of development)process to determine it satisfied business req.

- -Activities :- Testing.

- Unit Testing for Function oriented & Object Oriented:

In an object-oriented program, **a unit test often consists of a sequence of method calls that create and alter objects.**

System Testing:

- to ensure that the software does what the customer wants it to do.
- <u>Entry criteria for system testing</u>

  Complete software system should be developed

  Unit testing must be completed

  Integration testing must be completed

  Specifications for the product have been completed

  Test scripts and schedules are ready

TYPES OF SYSTEM TESTING

There are more than 50 types in system testing. The mainly using types are

Usability testing

Stress testing

Recovery testing

Deployment testing

Load testing

Performance testing &

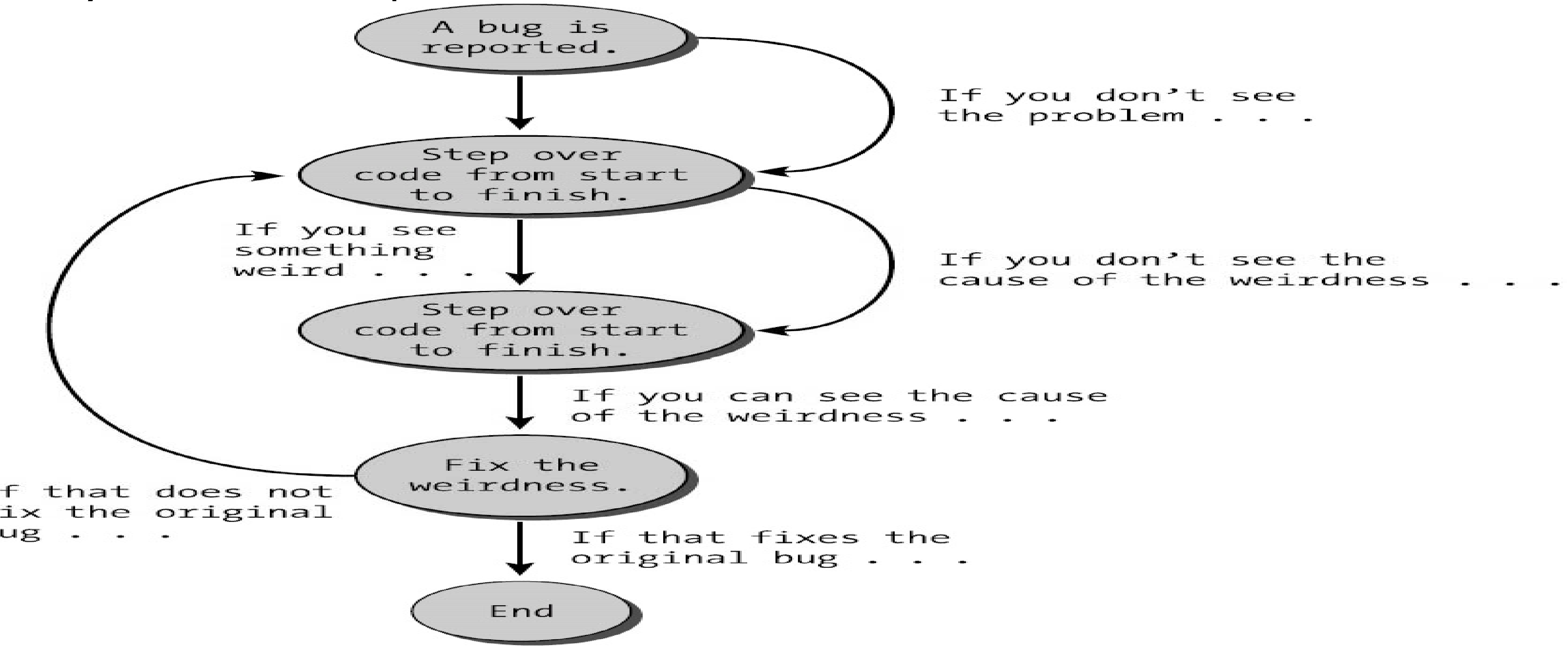
Security testing

DEBUGGING CONCEPTS & TECHNIQUE

All bugs stem from a one basic idea: **something thought to be right, was in fact wrong**. Due to this simple principle, truly weird bugs can challenge logic, making debugging software challenging. The typical behaviour of many inexperienced programmers is to shocked when unexpected problems arise.

BRUTE FORCE

Involves using the debugger to step across the code from start to finish until you notice something odd. "Wait, that variable looks wrong—how did that happen?" Then you start over and step through the code again, looking for the source of that oddity. When you reach the end and don't see the source of the problem, you start over and step through the code once more. Repeat over and over and over again until finally, "Oh, I see now—the problem is such-and-such. Now how do I fix that?"


A bug is reported.
If you don't see the problem . . .
Step over code from start to finish.
If you see something weird . . .
If you don't see the cause of the weirdness . . .
Step over code from start to finish.
If you can see the cause of the weirdness . . .
Fix the weirdness.
f that does not ix the original ug . . .
If that fixes the original bug . . .
End

Back Tracking:

Popular approach for small system.

Bug detect -> backward tracing in full source code

WHITE BOX TESTING

White – Box Testing is also called the **glass box testing**.

It is a test – case design philosophy that uses the control structure.

Using White – Box Testing, software engineer can derive test cases that.

- **G**uarantee that all independent paths within a module have been exercised at least once.
- **E**xercise all logical decisions on their true and false sides.
- **E**xecute all loops at their boundaries and within their operational bounds
- **E**xercise internal data structures to ensure their validity.

DIFFERENT METHODS OF WHITE – BOX TESTING

Basis Path Testing

Control Structure Testing

BASIS PATH TESTING

INTRODUCTION

Basis path testing is a white box testing technique first proposed by Tom McCabe.

The basis path method **enables the test case designer to derive a logical complexity** measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.

METHODS FOR BASIS PATH TESTING

Flow Graph Notations

Independent Program Paths

Graph Matrices

1. FLOW GRAPH NOTATION

Before the basis path method can be introduced, a simple notation for the representation of control flow, called a flow graph must be introduced.

The flow graph depicts logical control flow using the different notations

DIFFERENT FLOW GRAPH NOTATIONS

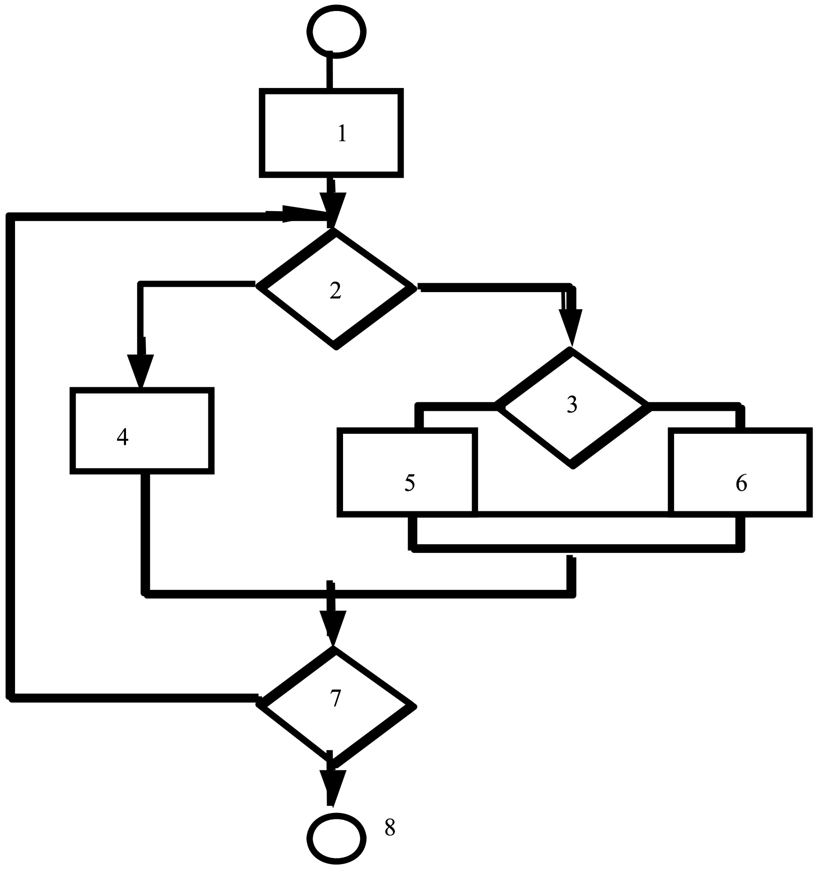
Sequence

Condition

Looping

Case Statement

2. INDEPENDENT PROGRAM PATHS



Next, we derive the independent paths:

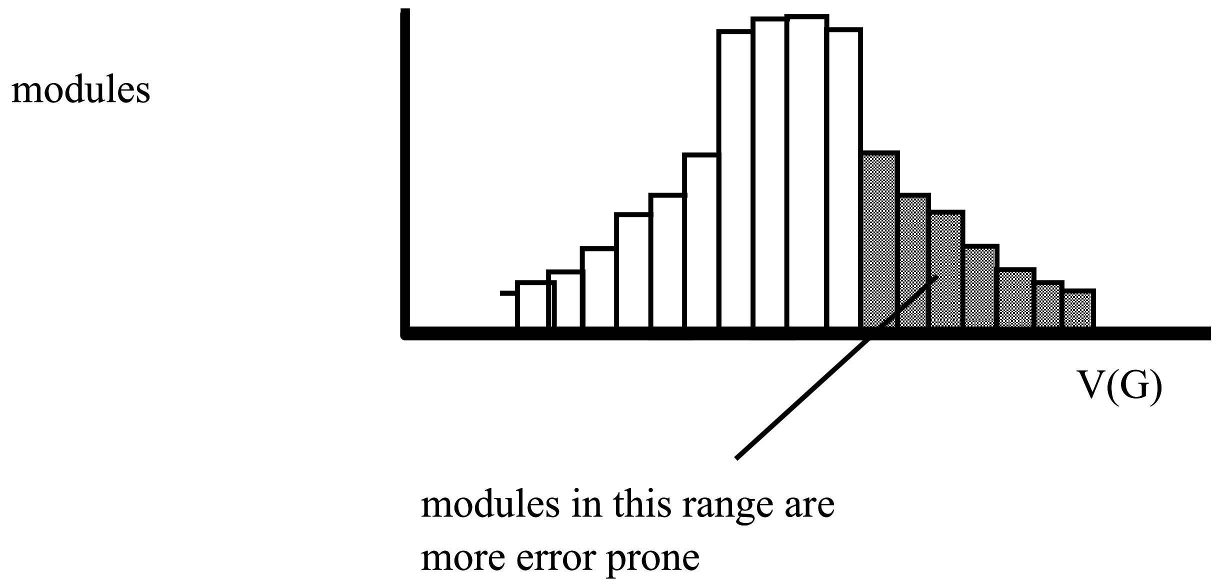
Since V(G) = 4, there are four paths

Path 1: 1,2,3,6,7,8
Path 2: 1,2,3,5,7,8
Path 3: 1,2,4,7,8
Path 4: 1,2,4,7,2,4,...7,8

Finally, we derive test cases to exercise these paths.

CYCLOMATIC COMPLEXITY

A number of industry studies have indicated that the higher V(G), the higher the probability of errors.

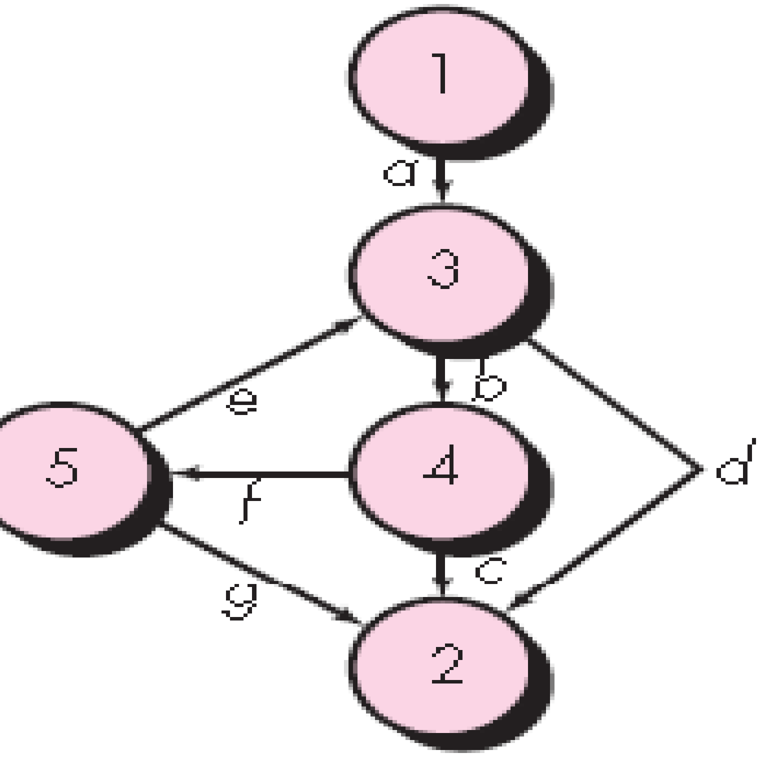


GRAPH MATRICES

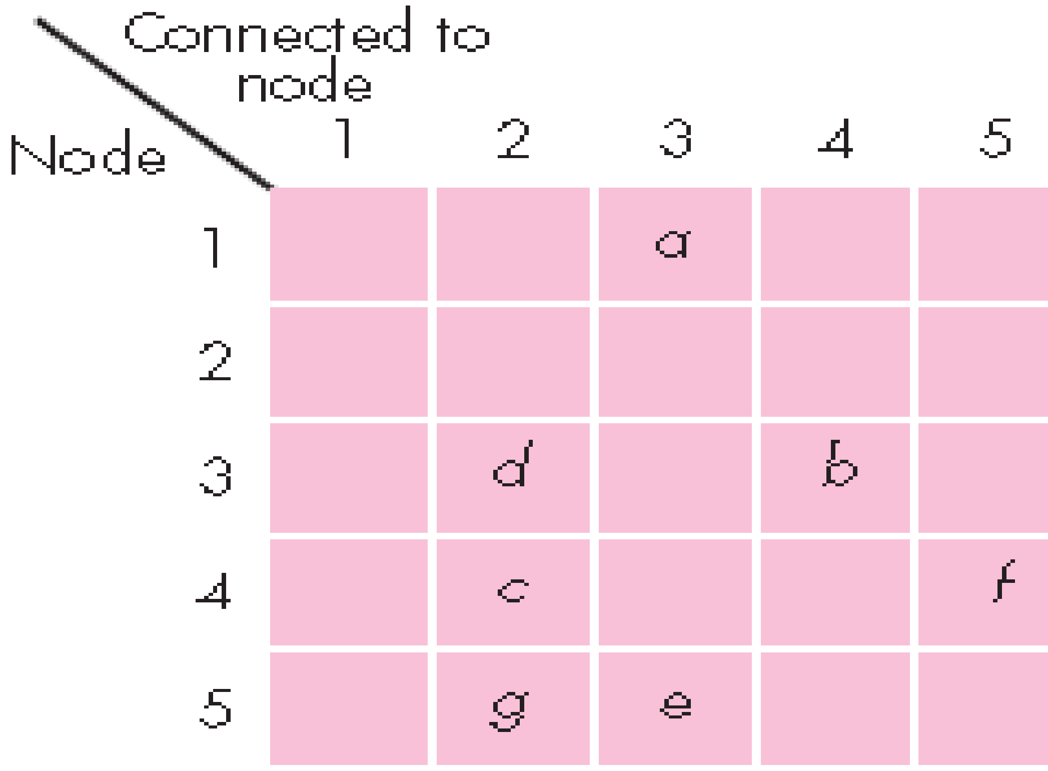
A Graph Matrix is a square matrix whose size is equal to the number of nodes on the flow graph.

Each row and column corresponds to an identified node, and matrix entries correspond to connections between nodes.

EXAMPLE



Flow graph



Node \ Connected to node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

Graph matrix

CONTROL STRUCTURE TESTING

METHODS OF CONTROL STRUCTURE TESTING

Condition Testing(Condition Testing is a test case design method that exercises the logical conditions contained in a program module)

Data Flow Testing(The Data Flow Testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.)

Loop Testing(Simple Loops,Nested Loops)

BLACK BOX TESTING

INTRODUCTION

Black – box testing, **also called a behavioural testing, focuses on the functional requirements of the software.**

That is black box testing enables the software engineer to **derive sets of conditions that will full-fill all functional requirements for a program.**

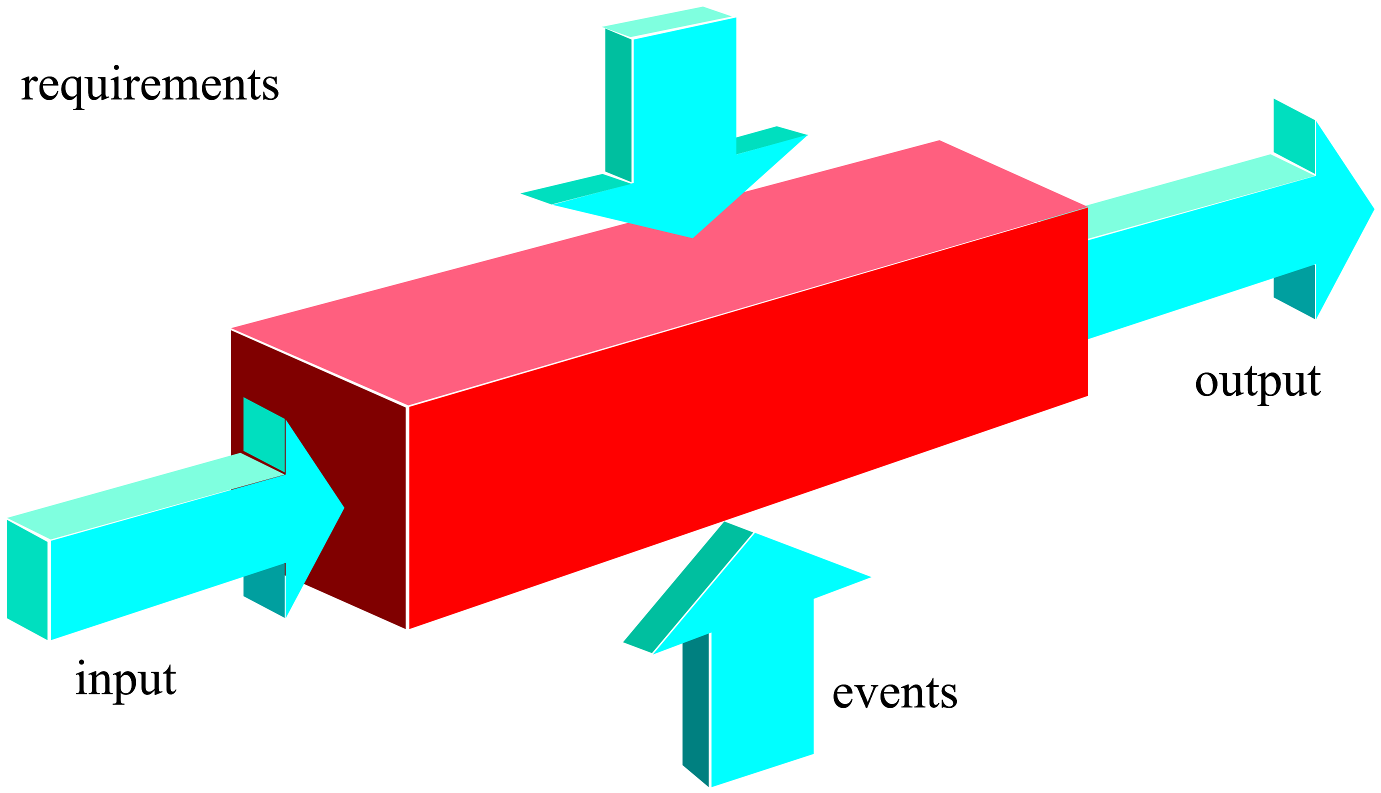
Black – box testing is not an alternative to white – box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white box methods.

CONTINUE...

Black – box testing attempts to find errors in the following categories:

1. Incorrect or missing functions.
2. Interface errors.
3. Errors in data structures or external database access.
4. Behaviour or performance errors
5. Initialization and termination errors.

ARCHITECTURE OF BLACK BOX TESTING



METHODS OF BLACK BOX TESTING

Graph Based Testing

Equivalence Partitioning

Boundary Value Analysis

Orthogonal Array Testing

1. GRAPH BASED TESTING

First step is to **understand the objects that are modelled in software and the relationship** that connects these objects.

Once this has been accomplished, the next step is to **define a series of tests that verify, "all objects have the expected relationship to one another".**

Software testing begins by creating a graph of important objects and their relationship and then develops a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.

2. EQUIVALENCE PARTITIONING

Equivalence partitioning is a black – box testing method that **divides the input domain of a program into classes.**

An ideal test case general error is observed.

Equivalence partitioning is to define test cases that uncover class of errors, thereby reducing the total number of test cases that must be developed.

3. BOUNDARY VALUE ANALYSIS

A greater number of errors occur at the boundaries of the input domain rather than in the "center“.

Boundary value analysis is a test case design method that complements equivalence partitioning

- It selects test cases at the edges of a class
- It derives test cases from both the input domain and output domain

4. ORTHOGONAL ARRAY TESTING

Orthogonal Array Testing can be applied to problems in which the input domain is relatively small.

The orthogonal array testing method is particularly useful in finding *region faults*—an error category associated with faulty logic within a software component.