

A hand is shown holding a glowing blue sphere. The sphere is covered in a network of white lines connecting various blue circular icons. These icons include a globe, a clock, an envelope, a shopping cart, a Wi-Fi symbol, and a gear. In the center of the sphere, the words "SOFTWARE TESTING" are written in a bold, blue, sans-serif font. The background is a dark blue with vertical light streaks and a faint grid pattern.

# SOFTWARE TESTING

**Software Testing** is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.

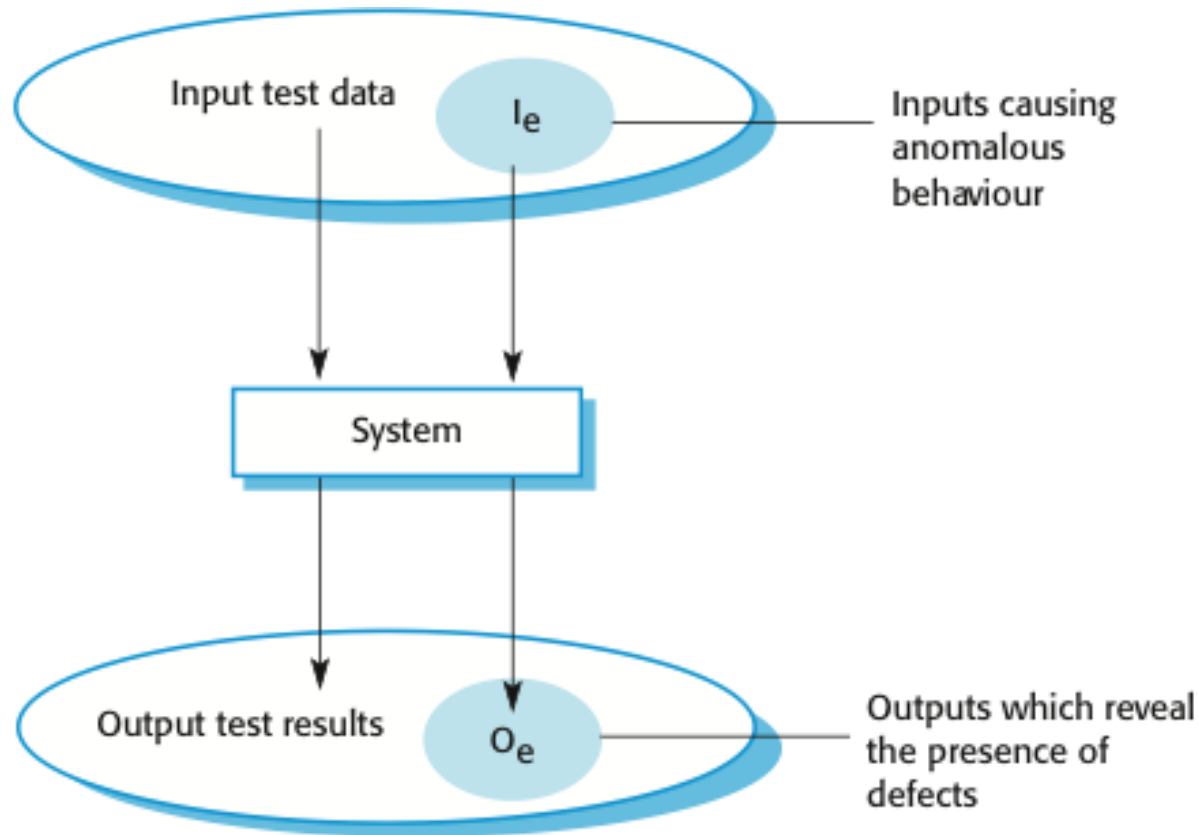


# How do you test a system?

- ▶ Input test data to the system.
- ▶ Observe the output:
  - ▶ Check if the system behaved as expected.



# How usually you test something?

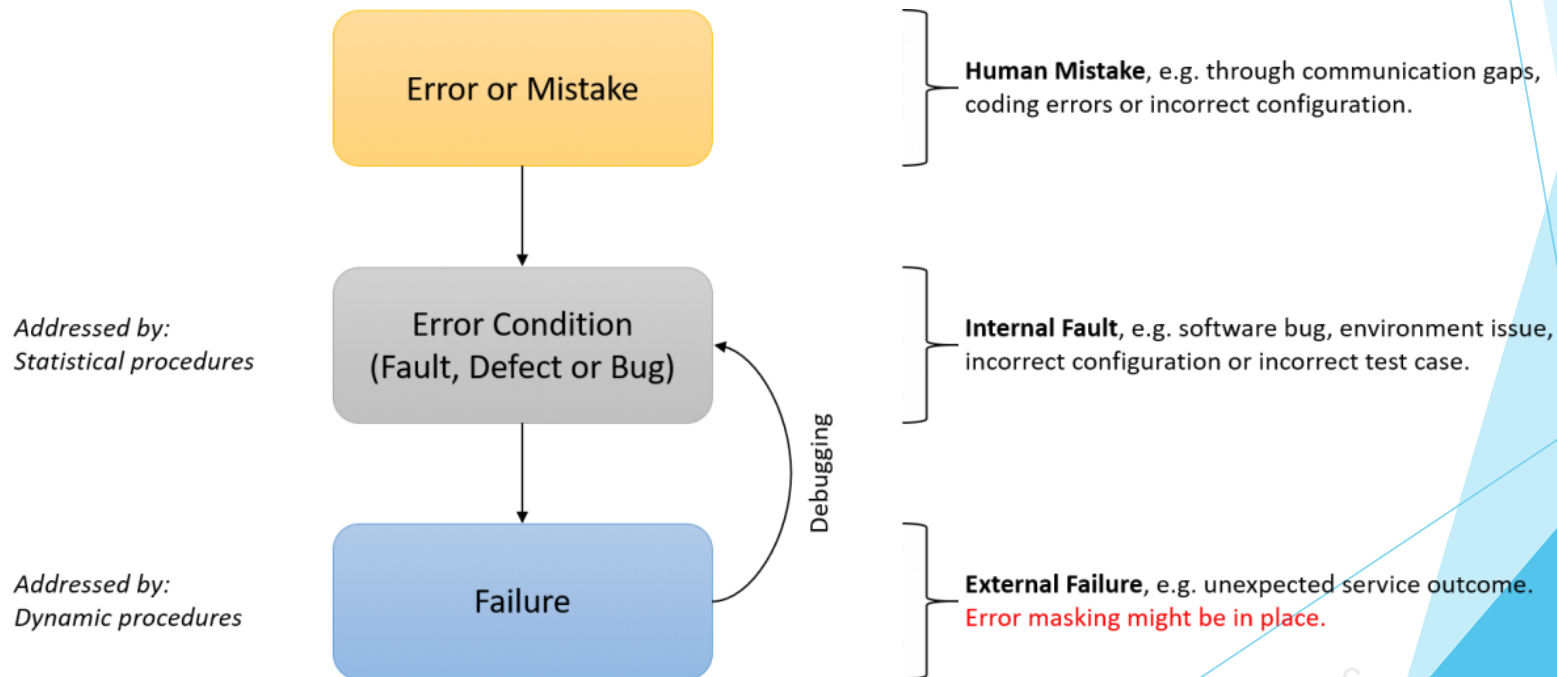


- ▶ If the program does not behave as expected:
  - ▶ note the conditions under which it failed.
  - ▶ later debug and correct.



# Errors and Failures

- ▶ A failure is a manifestation of an error.
  - ▶ presence of an error may not lead to a failure.



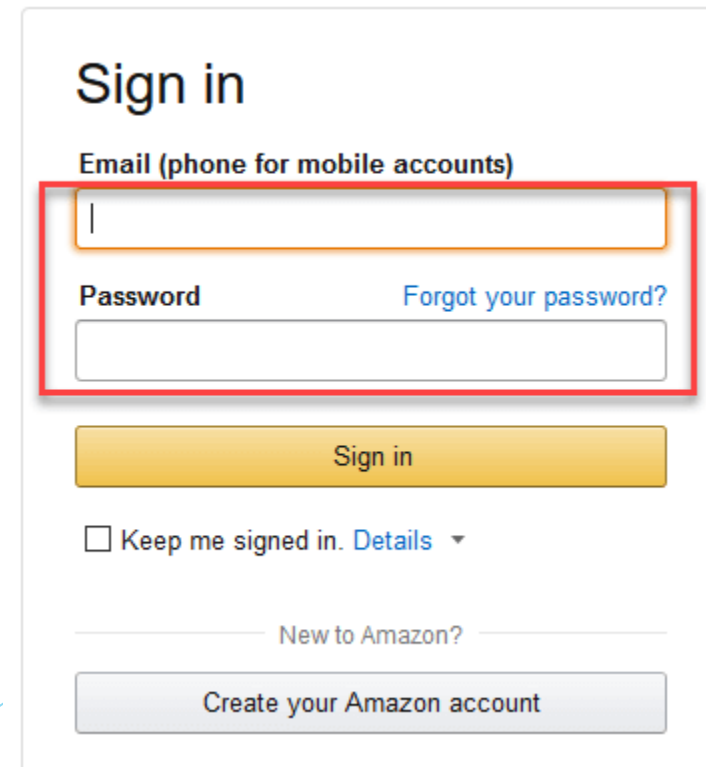


# Test cases and Test suite

A **TEST CASE** is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition

A **TEST SUITE** is a collection of test cases

1. Check system behavior when valid email id and password is entered.
  2. Check system behavior when invalid email id and valid password is entered.
  3. Check system behavior when valid email id and invalid password is entered.
  4. Check system behavior when invalid email id and invalid password is entered.
  5. Check system behavior when email id and password are left blank and Sign in entered.
  6. Check Forgot your password is working as expected
  7. Check system behavior when valid/invalid phone number and password is entered.
1. Check system behavior when "Keep me signed" is checked



The image shows a screenshot of the Amazon sign-in interface. The form is titled "Sign in" and contains fields for "Email (phone for mobile accounts)" and "Password". A red rectangular box highlights the email and password input fields. Below the password field is a "Sign in" button. Underneath the button is a checkbox labeled "Keep me signed in." followed by a link to "Details". At the bottom of the form, there is a link "New to Amazon?" and a button "Create your Amazon account".

## Sign in

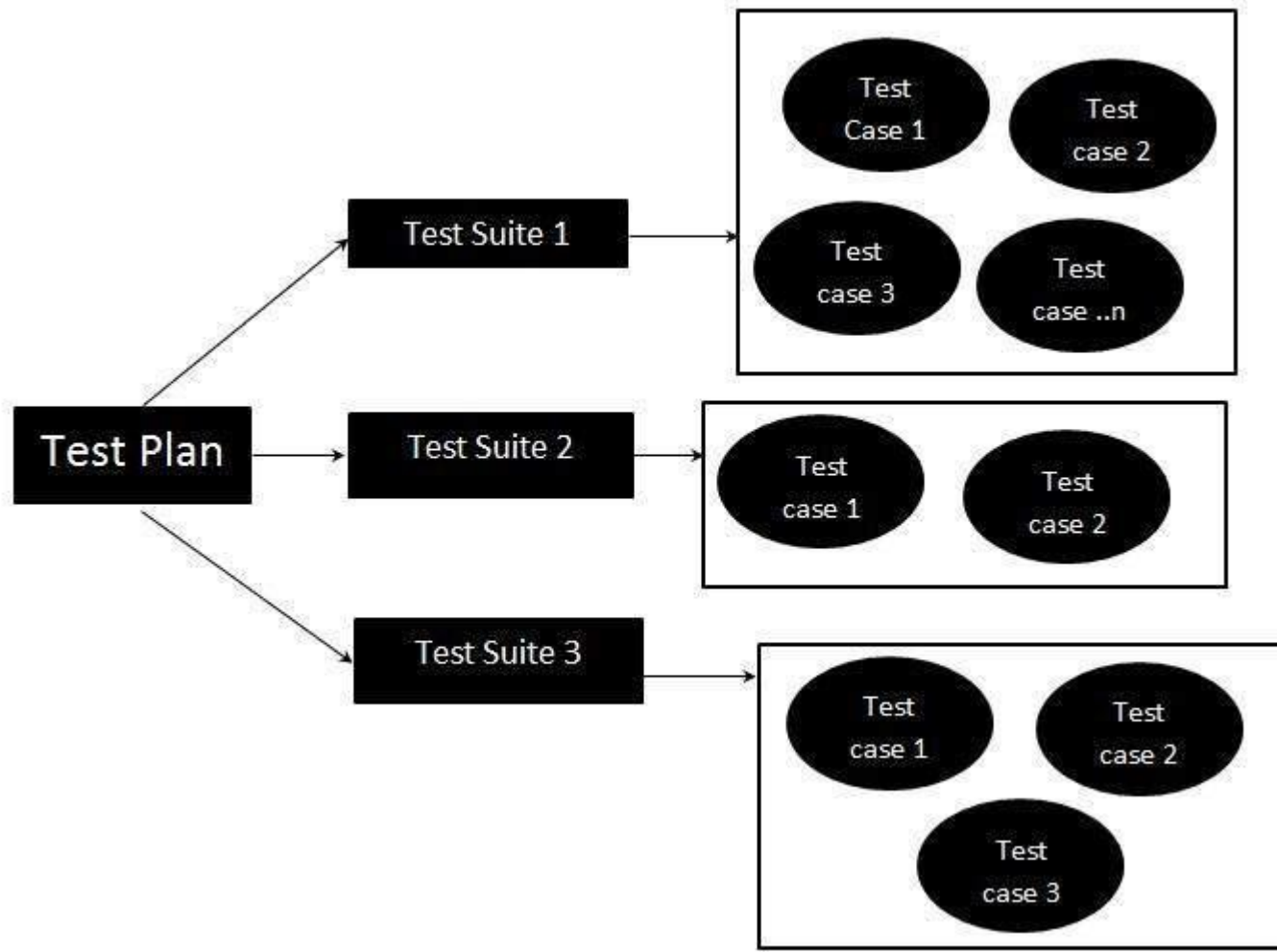
Email (phone for mobile accounts)

Password [Forgot your password?](#)

☐ Keep me signed in. [Details](#)

[New to Amazon?](#)





# Test cases and Test suite

- ▶ A test case is a triplet  $[I, S, O]$ :
  - ▶ **I** is the data to be input to the system,
  - ▶ **S** is the state of the system at which the data is input,
  - ▶ **O** is the expected output from the system.

# Test Case Document Template Examples


Your Company LOGO	Project Name:		Test Designed by:	
	Module Name:		Test Designed date:	
	Release Version:		Test Executed by:	
			Test Execution date:	
Pre-condition				
Dependencies:				
Test Priority				
Test Case#	Test Title	Test Summary	Test Steps	Test Data

Test Scenario ID	Login-1	Test Case ID	Login-1B				
Test Case Description	Login – Negative test case	Test Priority	High				
Pre-Requisite	NA	Post-Requisite	NA				
Test Execution Steps:							
S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	Test Comments
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter invalid Email & any Password and hit login button	Email id : invalid@xyz.com Password: *****	The email address or phone number that you've entered doesn't match any account. Sign up for an account.	The email address or phone number that you've entered doesn't match any account. Sign up for an account.	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Invalid login attempt stopped
3	Enter valid Email & incorrect Password and hit login button	Email id : valid@xyz.com Password: *****	The password that you've entered is incorrect. Forgotten password?	The password that you've entered is incorrect. Forgotten password ?	IE-11	Pass	[Priya 10/17/2017 11:46 AM]: Invalid login attempt stopped

### Sample Template for Test Case Log

<b>Test Name:</b>	Describe the Name of the Test
<b>Test Case Author:</b>	Describe the Test Case Author Name
<b>Tester Name:</b>	Describe the Tester Name
<b>Project ID / Name:</b>	Describe the Name of the Project
<b>Test Cycle ID:</b>	Describe the Test Cycle ID
<b>Date Tested:</b>	Describe the Date on which Test Case Was Completed

[illegible]

	Project Name	OrangeHRM Version 3.0 – My Info Module				
	Reference Document	Project Functional Requirement Specification , Version 1				
	Created by	© www.SoftwareTestingHelp.com Team				
	Date of creation	13-Feb-14				
	Date of review	20-Feb-14				
Test case ID	Test Objective	Precondition	Steps:	Test data	Expected result	Post-condition
TC_MI_01	Successful Employee login to OrangeHRM portal Check the screenshot to get an idea of what screen we are testing 	1. A valid ESS-User account to log	1. In the login Panel, enter the username	"A valid username" Enter the actual data in your real time situation	The user is logged in successfully. There is only one expected result for the entire test case. However, that does not have to be so. If it makes sense that for every step, you want to write the result of exactly what happens with it, please free to have an expected result for each test step	For first time users pers displayed. Note: This info is only a to the tester
			2. Enter the Password for the ESS-User account in the password field	"A valid Password"		
			3. Click "Login" button			
TC_MI_02	Error message on unsuccessful Employee login to OrangeHRM portal	1. A ESS-User name to login to be available 2. Orange HRM 3.0 site is launched on a compatible browser	1. In the login Panel, enter the username	"A valid username"	An Error message is displayed and the user is not logged in to the Orange HRM portal. "<Exact Error Message>" In the test case, it is not enough when we say, 'that an error is displayed'- in addition to that, we will have to mention the exact error message that is going to be encountered by the user- This information can generally be found in FRD(SRS). if not, look in the technical design document or Use cases.- Check the test next case, where we write Expected result, step wise.	As you can see, post con when there is nothing e
			2. Enter the Password for the ESS-User account in the password field	"A Invalid Password"		
			3. Click "Login" button			



Excel test case template						
Home Insert Draw Page Layout Formulas Data Review View						
<div> <div>Paste</div> <div> <div>Arial</div> <div>12</div> <div>A</div> <div>A</div> </div> <div> <div>B</div> <div>I</div> <div>U</div> </div> <div> <div>General</div> </div> <div> <div>Conditional Formatting</div> <div>Format as Table</div> <div>Cell Styles</div> </div> <div> <div>Insert</div> <div>Delete</div> <div>Format</div> </div> <div> <div>Σ</div> <div>Sort &amp; Filter</div> <div>Find &amp; Select</div> </div> <div> <div>Share</div> <div>Comments</div> </div> <div> <div>Icons</div> </div> </div>						
G19						
	A	B	C	D	E	F
1	Test case template					
2	Process	Test case	Step	Description	Status	Expected result
3	Logistics	Delivery	1	create delivery	passed	delivery created
4			2	print delivery form	failed	form printed successfully
5			3	print shipping label	in work	label printed successfully
6			4	notify freight forwarder	open	freight forwarder informed
7			5	freight forwarder picks up delivery	open	delivery picked up
8	Accounting	Billing process	1	Select customer delivery	failed	unable to find delivery
9			2	Print invoice	in work	
10			3	Send invoice to customer	open	
11			4	Create open item in AR	open	
12	Sales	Customer master data	1	Create new customer	passed	
13			2	Change customer data	passed	
14			3	Block customer	failed	missing authorization
15			4	Delete customer	in work	
16						
17						
18						
19						



# Verification versus Validation

- ▶ Verification is the process of determining:
  - ▶ whether output of one phase of development conforms to its previous phase.
- ▶ Validation is the process of determining
  - ▶ whether a fully developed system conforms to its SRS document.

# Verification versus Validation

- ▶ Aim of Verification:
  - ▶ phase containment of errors
- ▶ Aim of validation:
  - ▶ final product is error free.

# Verification versus Validation

- ▶ **Verification:**
  - ▶ are we doing right?
- ▶ **Validation:**
  - ▶ have we done right?

# Design of Test Cases

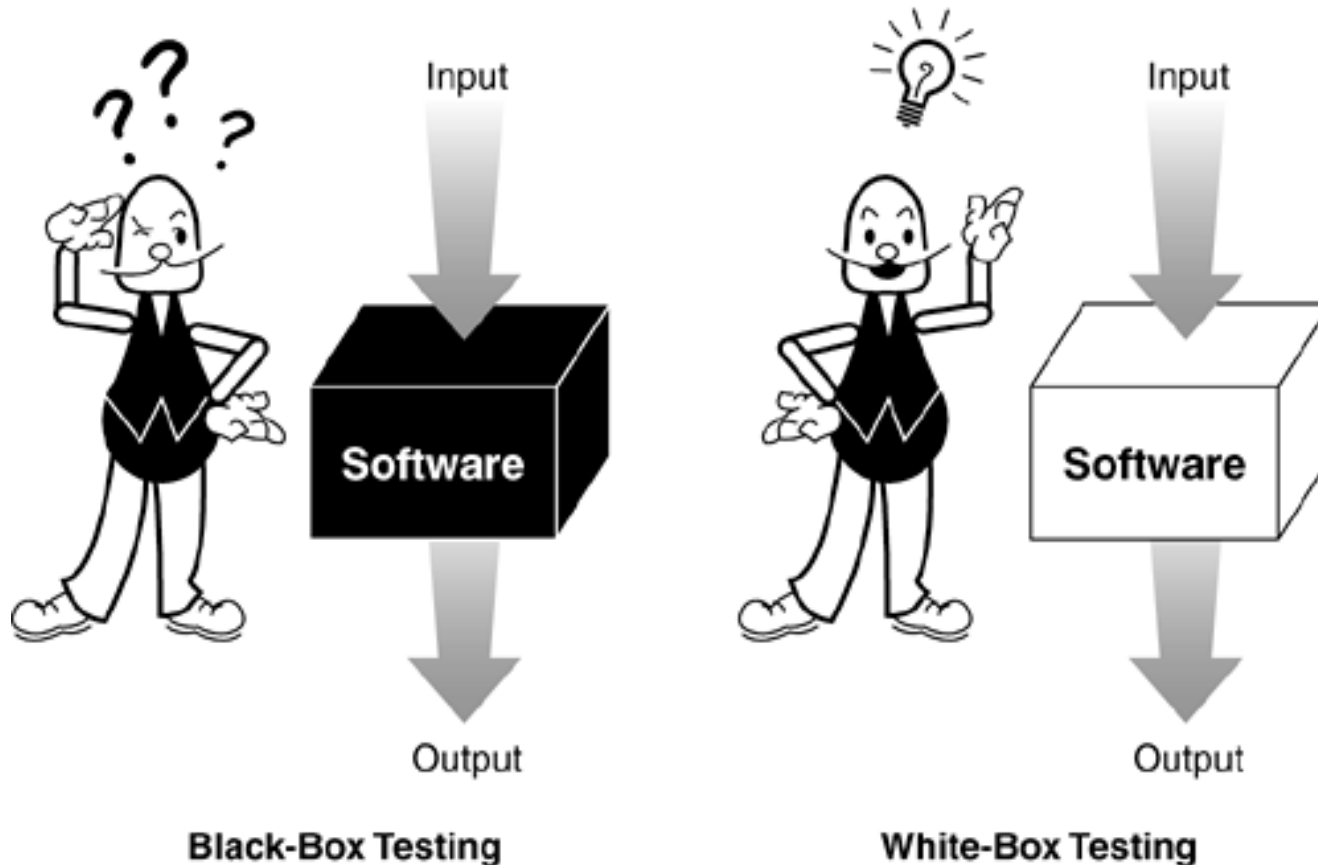
- ▶ Exhaustive testing of any system is impractical:
  - ▶ input data domain is extremely large.
- ▶ Design an **optimal test suite**:
  - ▶ of reasonable size
  - ▶ to uncover as many errors as possible.

# Design of Test Cases

- ▶ If test cases are selected randomly:
  - ▶ many test cases do not contribute to the **significance of the test suite**,
  - ▶ do not detect errors not already detected by other test cases in the suite.
- ▶ The number of test cases in a randomly selected test suite:
  - ▶ not an indication of the effectiveness of the testing.

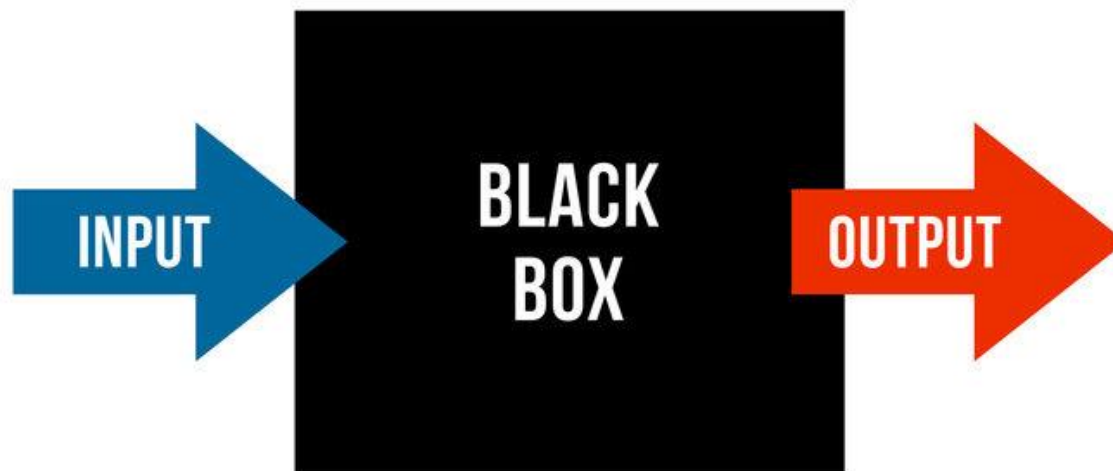
# TYPES/APPROACHES OF SOFTWARE TESTING

- ▶ Two main approaches to design test cases:
  - ▶ Black-box approach
  - ▶ White-box (or glass-box) approach





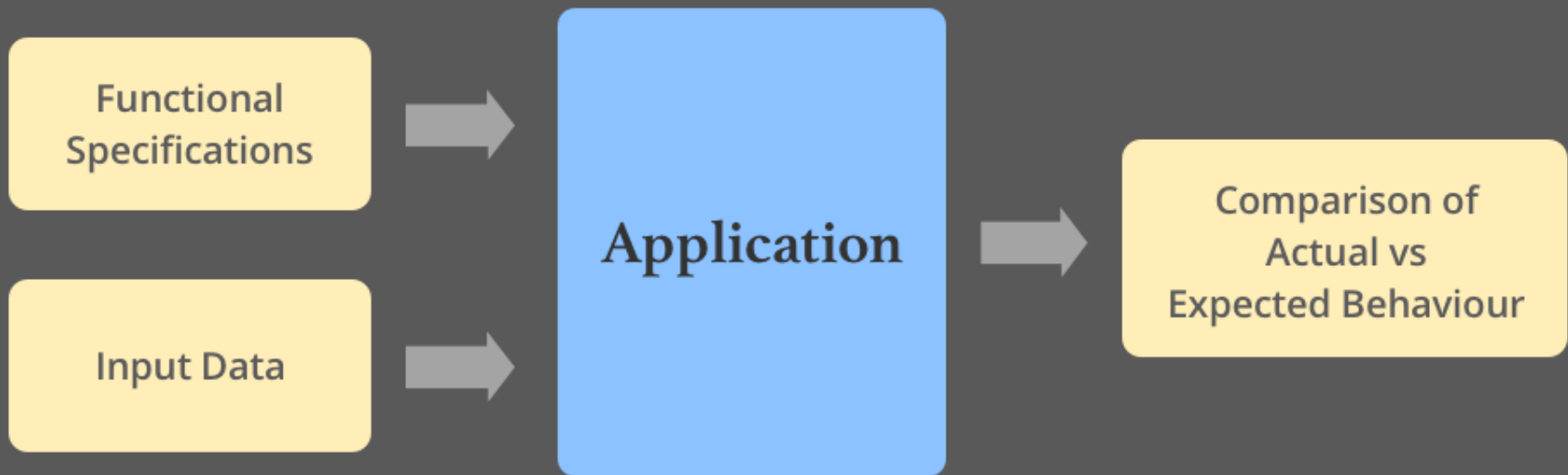
# Black Box Testing



# Black-box Testing

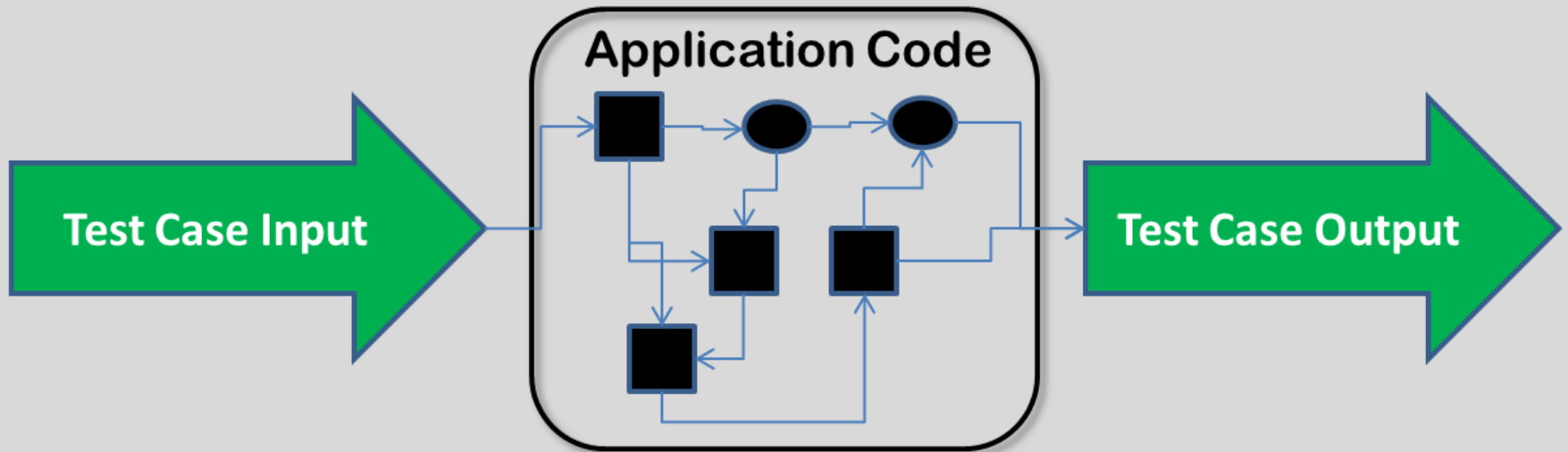
- ▶ Test cases are designed using only functional specification of the software:
  - ▶ **without any knowledge of the internal structure of the software.**
- ▶ For this reason, black-box testing is also known as functional testing.

# Functional Testing



# White-box Testing

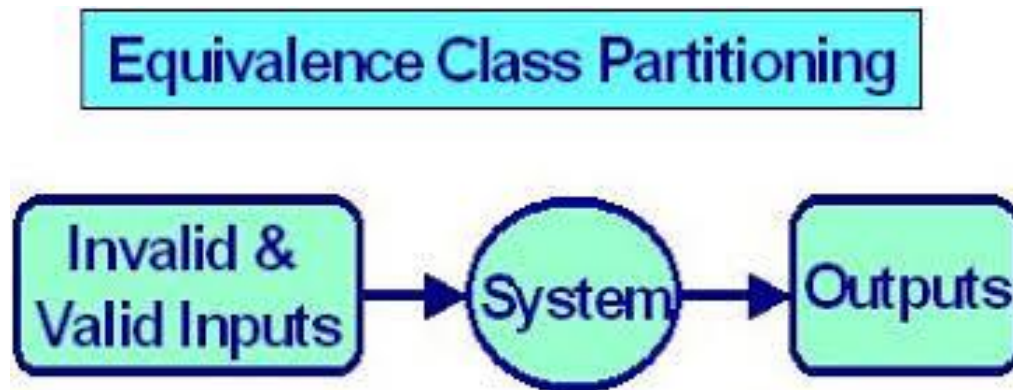
- ▶ Designing white-box test cases:
  - ▶ requires knowledge about the internal structure of software.
  - ▶ white-box testing is also called structural testing.



**WHITE BOX TESTING APPROACH**

# Black-box Testing

- ▶ There are essentially two main approaches to design black box test cases:
  - ▶ Equivalence class partitioning
  - ▶ Boundary value analysis



# Equivalence Partitioning Examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: $-99 \leq N \leq 99$	$[-99, -10]$ $[-9, -1]$ 0 $[1, 9]$ $[10, 99]$	$< -99$ $> 99$ Malformed numbers {12-, 1-2-3, ...} Non-numeric strings {junk, 1E2, \$13} Empty value
Phone Number  Area code: [200, 999]	555-5555 (555)555-5555 555-555-5555 $200 \leq \text{Area code} \leq 999$	Invalid format 5555555, (555)(555)5555, etc.  Area code $< 200$ or $> 999$ Area code with non-numeric characters



# Equivalence Class Partitioning

- ▶ Input values to a program are partitioned into **equivalence classes**
- 
- ▶ Partitioning is done such that:
  - ▶ **program behaves in similar ways to every input value belonging to an equivalence class.**

# Why define equivalence classes?

- ▶ Test the code with just one representative value from each equivalence class:
  - ▶ as good as testing using any other values from the equivalence classes.

# Equivalence Class Partitioning

- ▶ How do you determine the equivalence classes?
  - ▶ examine the input data.
  - ▶ few general guidelines for determining the equivalence classes can be given

# Equivalence Class Partitioning

- ▶ If the input data to the program is specified by a **range of values**:
  - ▶ e.g. numbers between 1 to 5000.
  - ▶ one valid and two invalid equivalence classes are defined.



# Equivalence Class Partitioning

- ▶ If input is an enumerated set of values:
  - ▶ e.g. {a,b,c}
  - ▶ one equivalence class for valid input values
  - ▶ another equivalence class for invalid input values should be defined.

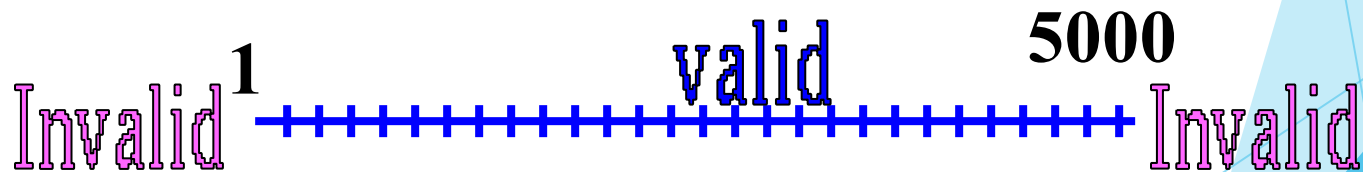
# Example (cont.)

- ▶ There are three equivalence classes:
  - ▶ the set of negative integers,
  - ▶ set of integers in the range of 1 and 5000,
  - ▶ integers larger than 5000.

Invalid ++++++ Invalid

# Example (cont.)

- ▶ The test suite must include:
  - ▶ representatives from each of the three equivalence classes:
  - ▶ a possible test suite can be:  
 $\{-5, 5000, 6000\}$ .





# Boundary Value Analysis

- ▶ Some typical programming errors occur:
  - ▶ at boundaries of equivalence classes
  - ▶ might be purely due to psychological factors.
- ▶ Programmers often fail to see:
  - ▶ special processing required at the boundaries of equivalence classes.

# Boundary Value Analysis

- ▶ Programmers may improperly use  $<$  instead of  $<=$
- ▶ Boundary value analysis:
  - ▶ select test cases at the boundaries of different equivalence classes.

**AGE**

Enter Age

\*Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
17	18, 19, 55, 56	57

# Example

- ▶ For a function that computes the square root of an integer in the range of 1 and 5000:
- ▶ test cases must include the values:  $\{0, 1, 5000, 5001\}$ .



# White-Box Testing

- ▶ There exist several popular white-box testing methodologies:
  - ▶ Statement coverage
  - ▶ branch coverage
  - ▶ path coverage
  - ▶ condition coverage
  - ▶ mutation testing
  - ▶ data flow-based testing

# Statement Coverage

- ▶ Statement coverage methodology:
  - ▶ design test cases so that
    - ▶ every statement in a program is executed at least once.

# Statement Coverage

- ▶ The principal idea:
  - ▶ unless a statement is executed,
  - ▶ we have no way of knowing if an error exists in that statement.

# Example

```
▶ int f1(int x, int y){  
▶ 1 while (x != y){  
▶ 2   if (x>y) then  
▶ 3     x=x-y;  
▶ 4   else y=y-x;  
▶ 5 }  
▶ 6 return x;    }
```

Euclid's GCD Algorithm

Test suit is: (3,3), (4,3) and (3,4) which will do 100% statement coverage

# Branch Coverage

- ▶ Test cases are designed such that:
  - ▶ different branch conditions
  - ▶ given true and false values in turn.



# Branch Coverage

- ▶ Branch testing guarantees statement coverage:
  - ▶ a stronger testing compared to the statement coverage-based testing.

# Example

```
▶ int f1(int x,int y){  
▶ 1 while (x != y){  
▶ 2   if (x>y) then  
▶ 3     x=x-y;  
▶ 4   else y=y-x;  
▶ 5 }  
▶ 6 return x;    }
```

# Example

- ▶ Test cases for branch coverage can be:
- ▶  $\{(x=3,y=3),(x=3,y=2), (x=4,y=3), (x=3,y=4)\}$

# Condition Coverage

- ▶ Test cases are designed such that:
  - ▶ each component of a composite conditional expression is executed at least once.
    - ▶ given both true and false values.

# Example

- ▶ Consider the conditional expression
  - ▶  $((c1.and.c2).or.c3)$ :
- ▶ Each of  $c1$ ,  $c2$ , and  $c3$  are exercised at least once,
  - ▶ i.e. given true and false values.

# Branch testing

- ▶ Condition testing
  - ▶ stronger testing than branch testing:
- ▶ Branch testing
  - ▶ stronger than statement coverage testing.

# Condition coverage

- ▶ Consider a boolean expression having  $n$  components:
  - ▶ for condition coverage we require  $2^n$  test cases. ( $n$  is no. of conditions)

# Path Coverage

- ▶ Design test cases such that:
  - ▶ all linearly independent paths in the program are executed at least once.



# Linearly independent paths

- ▶ Defined in terms of
  - ▶ control flow graph (CFG) of a program.

# Path coverage-based testing

- ▶ To understand the path coverage-based testing:
  - ▶ we need to learn how to draw control flow graph of a program.

# Control flow graph (CFG)

- ▶ A control flow graph (CFG) describes:
  - ▶ the sequence in which different instructions of a program get executed.
  - ▶ the way control flows through the program.

# How to draw Control flow graph?

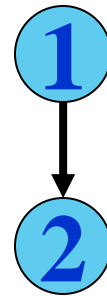
- ▶ Number all the statements of a program.
- ▶ Numbered statements:
  - ▶ represent nodes of the control flow graph.

# How to draw Control flow graph?

- ▶ Sequence:

- ▶ 1 `a=5;`

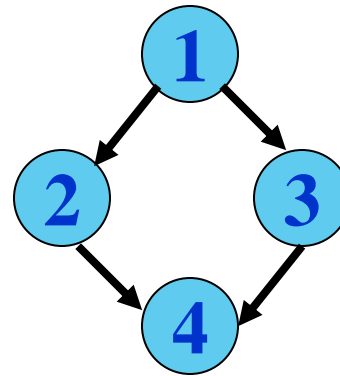
- ▶ 2 `b=a*b-1;`



# How to draw Control flow graph?

## ► Selection:

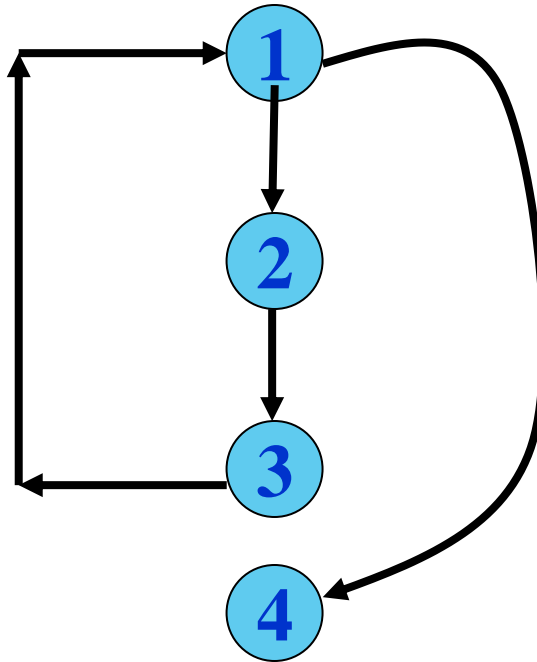
- 1 if(a>b) then
- 2       c=3;
- 3 else   c=5;
- 4 c=c\*c;



# How to draw Control flow graph?

► Iteration:

- 1 while(a>b){
- 2     b=b\*a;
- 3     b=b-1;}
- 4 c=b+d;

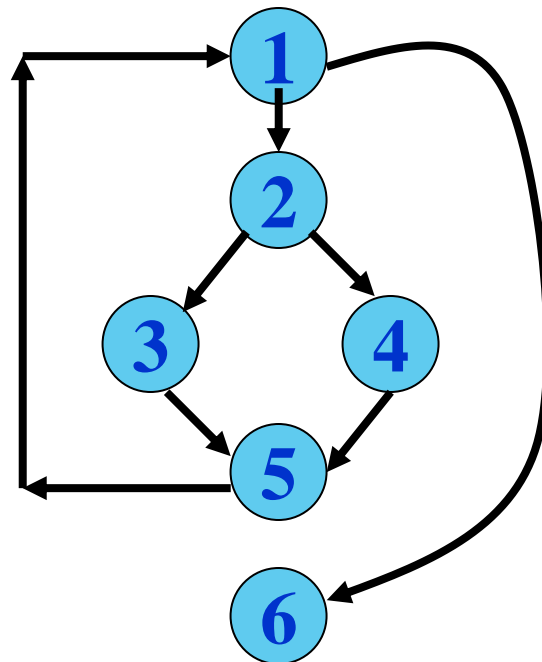


# Example

```
▶ int f1(int x,int y){  
▶ 1  while (x != y){  
▶ 2    if (x>y) then  
▶ 3      x=x-y;  
▶ 4    else y=y-x;  
▶ 5  }  
▶ 6  return x;      }
```



# Example Control Flow Graph



# Path

- ▶ A path through a program:
  - ▶ a node and edge sequence from the starting node to a terminal node of the control flow graph.
  - ▶ There may be several terminal nodes for program.

# Independent path

- ▶ Any path through the program:
  - ▶ introducing at least one new node:
    - ▶ that is not included in any other independent paths.

# Independent path

- ▶ It is straight forward:
  - ▶ to identify linearly independent paths of simple programs.
- ▶ For complicated programs:
  - ▶ it is not so easy to determine the number of independent paths.

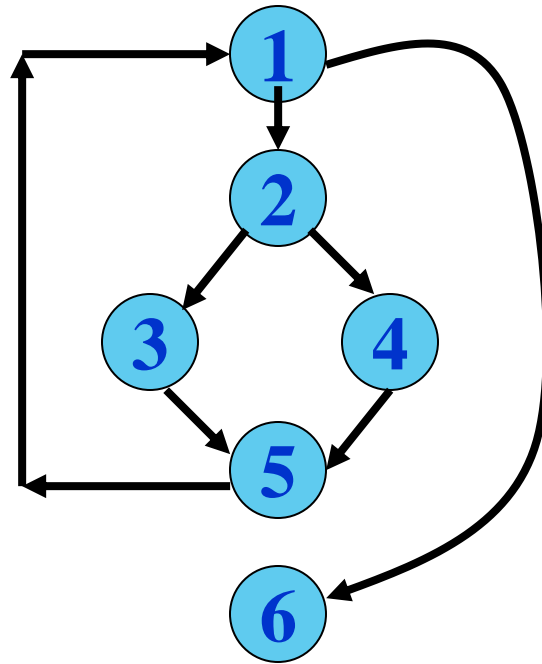
# McCabe's cyclomatic metric

- ▶ An upper bound:
  - ▶ for the number of linearly independent paths of a program
- ▶ Provides a practical way of determining:
  - ▶ the maximum number of linearly independent paths in a program.

# McCabe's cyclomatic metric

- ▶ Given a control flow graph  $G$ , cyclomatic complexity  $V(G)$ :
  - ▶  $V(G) = E - N + 2$ 
    - ▶  $N$  is the number of nodes in  $G$
    - ▶  $E$  is the number of edges in  $G$

# Example Control Flow Graph



# Example

► Cyclomatic complexity

=

$$7 - 6 + 2 = 3.$$



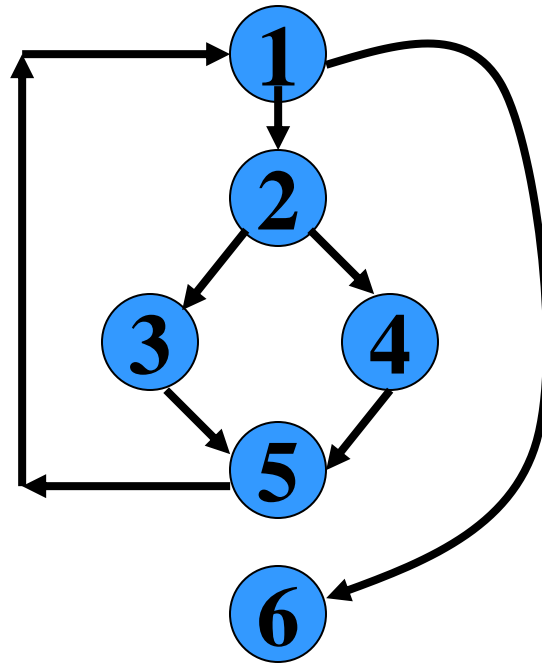
# Cyclomatic complexity

- ▶ Another way of computing cyclomatic complexity:
  - ▶ inspect control flow graph
  - ▶ determine number of bounded areas in the graph
- ▶  $V(G) = \text{Total number of bounded areas} + 1$

# Bounded area

- ▶ Any region enclosed by a nodes and edge sequence.

# Example Control Flow Graph



# Example

- ▶ From a visual examination of the CFG:
  - ▶ the number of bounded areas is 2.
  - ▶ cyclomatic complexity =  $2+1=3$ .

- ▶ IF A = 10 THEN
- ▶   IF B > C
- ▶     THEN A = B
- ▶   ELSE A = C
- ▶   ENDIF
- ▶ ENDIF
- ▶ Print A Print B Print C

# Cyclomatic complexity

- ▶ McCabe's metric provides:
  - ▶ a quantitative measure of testing difficulty and the ultimate reliability
- ▶ Intuitively,
  - ▶ number of bounded areas increases with the number of decision nodes and loops.

# Cyclomatic complexity

- ▶ The first method of computing  $V(G)$  is amenable to automation:
  - ▶ you can write a program which determines the number of nodes and edges of a graph
  - ▶ applies the formula to find  $V(G)$ .

# Cyclomatic complexity

- ▶ The cyclomatic complexity of a program provides:
  - ▶ a lower bound on the number of test cases to be designed
  - ▶ to guarantee coverage of all linearly independent paths.



# Cyclomatic complexity

- ▶ Defines the number of independent paths in a program.
- ▶ Provides a lower bound:
  - ▶ for the number of test cases for path coverage.

# Cyclomatic complexity

- ▶ Knowing the number of test cases required:
  - ▶ does not make it any easier to derive the test cases,
  - ▶ only gives an indication of the minimum number of test cases required.

Complexity No.	Corresponding Meaning of V(G)
1-10	1) Well-written code, 2) Testability is high, 3) Cost / effort to maintain is low
10-20	1) Moderately complex code, 2) Testability is medium, 3) Cost / effort to maintain is medium.
20-40	1) Very complex code, 2) Testability is low, 3) Cost / effort to maintain is high.
> 40	1) Not testable, 2) Any amount of money / effort to maintain may not be enough.

# Derivation of Test Cases

- ▶ Let us discuss the steps:
  - ▶ to derive path coverage-based test cases of a program.

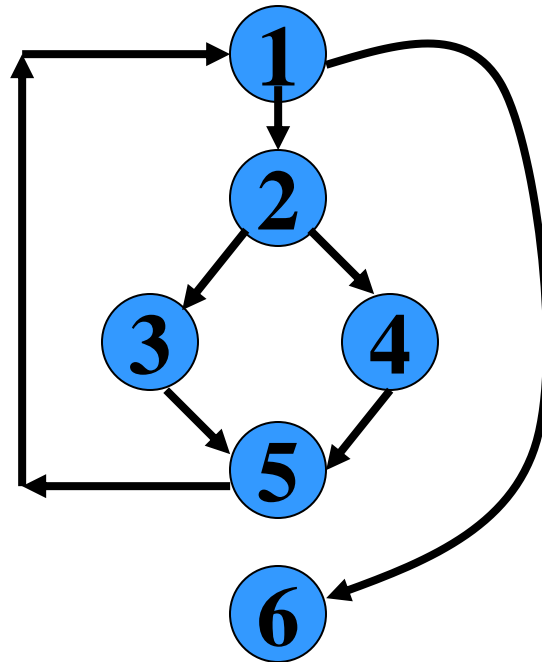
# Derivation of Test Cases

- ▶ Draw control flow graph.
- ▶ Determine  $V(G)$ .
- ▶ Determine the set of linearly independent paths.
- ▶ Prepare test cases:
  - ▶ to force execution along each path.

# Example

```
▶ int f1(int x,int y){  
▶ 1 while (x != y){  
▶ 2   if (x>y) then  
▶ 3     x=x-y;  
▶ 4   else y=y-x;  
▶ 5 }  
▶ 6 return x;      }
```

# Derivation of Test Cases



- ▶ Number of independent paths: 3
  - ▶ 1,6 test case (x=1, y=1)
  - ▶ 1,2,3,5,1,6 test case(x=1, y=2)
  - ▶ 1,2,4,5,1,6 test case(x=2, y=1)

## Exercise: 2

```
IF A = 354  
THEN IF B > C  
THEN A = B  
ELSE A = C  
ENDIF  
ENDIF  
Print A
```



## Exercise:3

```
i = 0;  
while (i < n - 1)  
do j = i + 1;  
  while (j < n)  
  do  
  if A[i] < A[j] then  
  swap (A[i], A[j]);  
  end do;  
  i = i + 1;  
  end do;
```

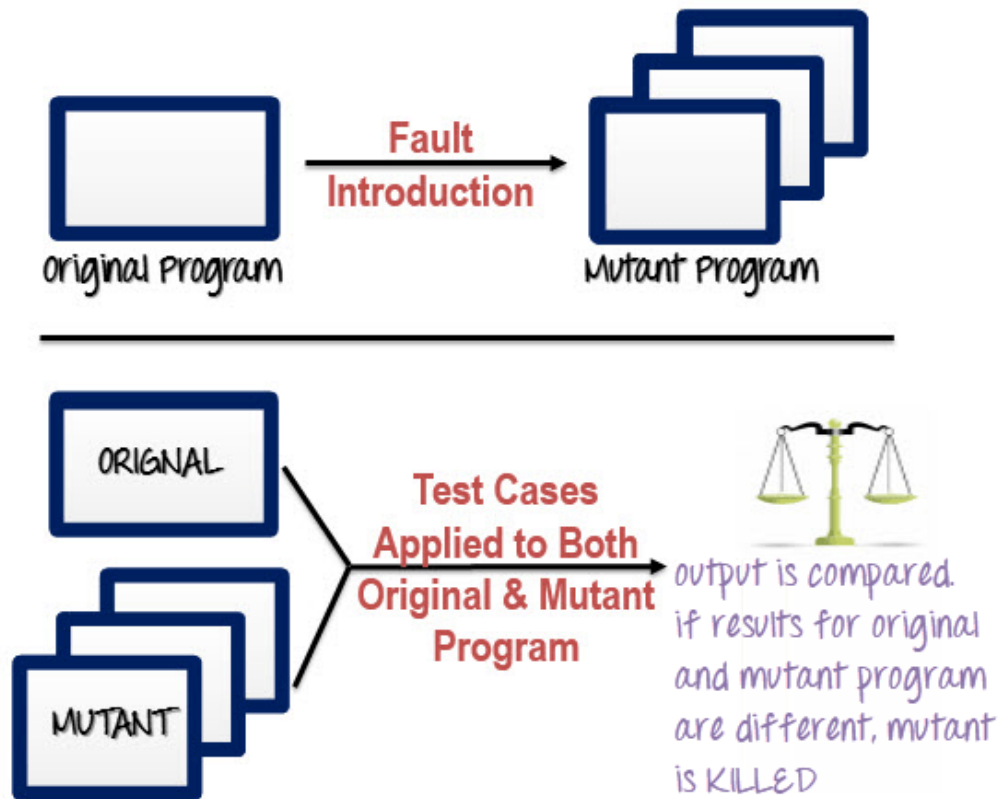
# Data Flow-Based Testing

- ▶ Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects.
- ▶ Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

# Advantages of Data Flow Testing:

- ▶ Data Flow testing helps us to pinpoint any of the following issues:
  1. A variable that is declared but never used within the program.
  2. A variable that is used but never declared.
  3. A variable that is defined multiple times before it is used.
  4. Deallocating a variable before it is used.

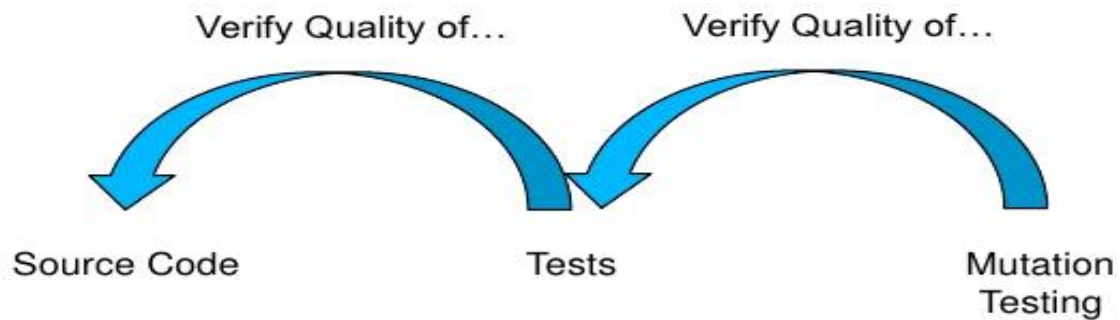
# Mutation Testing



# Definition

- ▶ **Mutation Testing** is a type of software **testing** in which certain statements of the source code are changed/**mutated** to check if the **test** cases are able to find errors in source code.
- ▶ The goal of **Mutation Testing** is ensuring the quality of **test** cases in terms of robustness that it should fail the **mutated** source code.

# What is Mutation Testing?



# Mutation Testing

- ▶ Each time the program is changed,
  - ▶ it is called a **mutated program**
  - ▶ the change is called a **mutant**.

# Mutation Testing

- ▶ The primitive changes can be:
  - ▶ altering an arithmetic operator,
  - ▶ changing the value of a constant,
  - ▶ changing a data type, etc.



# Mutation Testing

- ▶ A major disadvantage of mutation testing:
  - ▶ computationally very expensive,
  - ▶ a large number of possible mutants can be generated.